MODULAR PERFORMANCE ANALYSIS FOR REAL - TIME EMBEDDED SYSTEMS

By

BAKARE, KAREEM AYENI

DEPARTMENT OF MATHEMATICS
FACULTY OF SCIENCE
AHMADU BELLO UNIVERSITY, ZARIA
NIGERIA

May, 2013

MODULAR PERFORMANCE ANALYSIS FOR REAL - TIME EMBEDDED SYSTEMS

BY

BAKARE, KAREEM AYENI B.Sc.(OAU, 1998) MSc.(ABU, 2008)

PhD/SCIE/00425/2008-09

A THESIS SUBMITTED TO THE POSTGRADUATE SCHOOL,

AHMADU BELLO UNIVERSITY, ZARIA NIGERIA

IN PARTIAL FULFILLMENT FOR THE AWARD OF

Ph.D. COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

May, 2013

DECLARATION

I declare that the work in the thesis entitled "Modular Performance Analysis For Real - Time Embedded Systems" has been performed by me in the Department of Mathematics under the supervision of Prof. S. B. Junaidu.

The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this thesis was previously presented for another degree or diploma at any university.

Bakare, Kareem Ayeni	**Date**
Name of student	

CERTIFICATION

This thesis entitled **"**MODULAR PERFORMANCE ANALYSIS FOR REAL - TIME EMBEDDED SYSTEMS" by BAKARE, Kareem Ayeni meets the regulations governing the award of the degree of PhD in Computer Science of Ahmadu Bello University, Zaria, and is approved for its contribution to knowledge and literary presentation.

_____	_____
Prof. S. B. Junaidu	**Date**
Major Supervisor	
_____	_____
Dr. C. Edeh	**Date**
Minor Supervisor	
_____	_____
Dr. S. E. Abdullahi	**Date**
Minor Supervisor	
_____	_____
Prof. S.E. Adewumi	**Date**
External Examiner	
_____	_____
Dr. B. Sanni	**Date**
Head of Department	
_____	_____
Prof. J. Adebayo	**Date**
Dean Postgraduate School	

ACKNOWLEDGEMENT

There is no way to acknowledge everybody that assisted me one way or the other during the course of my study. I express my gratitude to Almighty Allah for His care and protection over me throughout my course.

I am very grateful to my supervisor Prof. S. B. Junaidu, who guided me with enthusiasm and great humour and supported me to grow as a researcher. He devoted his time to help in guiding me and provide valuable information about the thesis, his constructive criticism and encouragement have contributed tremendously to the success of this work.

I am also grateful to Dr. Edeh Christopher, my second supervisor for his technical advice and contribution to this work. I am grateful also to Dr. Abdullahi my third supervisor and Dr. Paul Pettersson of Uppsala University for his advice on UPPAAL. I also acknowledge the effort of Controller F. K. Ojo and A. Buhari, Nigerian College of Aviation Technology, Zaria (NCAT). I wish to thank all my Lecturers and staff of Mathematics department, Ahmadu Bello University, Zaria (ABU), Zaria.

I am grateful to the entire staff of MicroScale Embedded Ltd., Kaduna, Nigeria for allowing us to use their laboratory to carry out the experiment.

I am deeply indebted to my family especially my mother, Alhaja S.S. Bakare and my wife, Barrister S.O. Bakare for their immeasurable support and understanding.

ABSTRACT

Modular performance analysis of real-time embedded systems plays an important role in the design process of complex embedded systems for analyzing essential performance characteristics of system design at an early phase. This thesis attempted to determine the most appropriate method for system-level performance analysis of real-time embedded systems, modelled and analysed a GPS-based Traffic Alert and Collision Avoidance System (TCAS) and incorporated reversal logic into this TCAS. Results from our analyses and simulation, using modern tools, show that our GPS-based TCAS preserves the functionalities of the traditional transponder-based TCAS. Additionally, our results show the GPS-based TCAS to be more accurate while supporting lateral and reversal resolution advisories. While we believe these ideas will carry over to other types of embedded systems, we note, however, that hardware-software co-simulations are often computationally complex with long running times and the effect of program's environment such as processor it runs on as potential limitations of our analyses.

TABLE OF CONTENTS

DECLARATION iii

CERTIFICATION iv

ACKNOWLEDGEMENT v

ABSTRACT vi

TABLE OF CONTENTS vii

LIST OF FIGURES xii

LIST OF TABLES xv

GLOSSARY xvi

CHAPTER ONE 1

INTRODUCTION 1

1.1 Background of the Study 1

1.2 Research Objectives 5

1.3 Motivation and Requirements for Modular Performance Analysis 5

1.4 Methodology 14

1.5 Contribution to Knowledge 17

1.6 Organisation of the rest of thesis 18

CHAPTER TWO 19

LITERATURE REVIEW 19

2.1 Introduction 19

2.2 Real-time Embedded Systems Modelling 22

2.2.1 Temporal Logic 24

2.3 Modelling Approach 25

2.3.1 General Purpose Approach for Modelling 26

2.3.2 Structured Analysis (SA) Approach 26

2.3.3 Structured Development for Real-Time Systems Approach 27

2.3.4 General Purpose Object Oriented Approach 28

2.4 Model checking/analysis 31

2.4.1 Model checking approach 33

2.4.2 Benefits of model checking 35

2.4.3 Model Checking Temporal Logic 36

2.4.4 Model Checking Branching Temporal Logic 38

2.5 Requirements for Embedded Systems Design 40

2.5.1 Behavioural Level 40

2.5.2 Structural Level 42

2.5.3 Physical Level 42

2.6 Additional Requirements for Embedded Systems Design 43

2.6.1 Multi-Paradigm Support 43

2.6.2 Refinement 43

2.6.3 Time 44

2.6.4 Configuration 44

2.7 Real-Time Embedded Systems Design Exploration 45

2.7.1 Hardware Selection....46
2.7.2 Hardware Design Consideration....49
2.7.3 Software Architecture Design Consideration....51
2.7.4 Embedded Systems Software....51
2.7.5 Programming Embedded Systems....53
2.7.6 Timers....55
2.7.7 Embedded Systems Operating Systems....55
2.8 Real-time Embedded Systems Design....60
2.8.1 Heterogeneous System Design Methodologies....61
2.8.1.1 COSYMA....64
2.8.1.2 Olympus....64
2.8.2 Ptolemy....64
2.8.3 Model-based Object Oriented Systems Engineering (MOOSE)....65
2.9 Performance Indexes....66
2.10 Review of Performance Analysis Methods....71
2.10.1 Simulation and Formal Analysis Methods....71
2.10.2 Real-Time Calculus....76
2.10.3 Holistic Scheduling Analysis....81
2.10.4 Compositional Approach....87
2.10.5 Timed Automata based performance analysis....90
2.10.6 Stochastic analysis method....104

2.11 Comparison of Performance Analysis Methods..... 105

2.12 Assessing Existing Performance Analysis Methods 108

CHAPTER THREE..... 110

SYSTEM MODELLING AND ANALYSIS 110

3.1 Introduction..... 110

3.2 Adopted Tools and Methods 111

3.3 Experimental/Laboratory Setup 112

3.3.1 Timing Analysis in Real-Time Embedded Systems..... 112

3.3.2 Features and Potential Application of PIC16F887A Microcontroller..... 114

3.3.2.1 Laboratory Setup 115

3.3.2.2 System Design..... 117

3.3.2.3 Experimental results of Simulating Microcontroller..... 120

3.3.2.4 Discussion on Simulating Microcontroller 120

3.4 Modelling and Performance Analysis of GPS based TCAS using 'UPPAAL' . 124

3.4.1 System Requirements 127

3.4.2 System Functionality 128

3.4.3 Formal Modelling of Proposed TCAS:..... 133

3.5 Validation and Verification of Results 142

CHAPTER FOUR..... 145

DESIGN OF TCAS REAL TIME EMBEDDED SYSTEM PROTOTYPE, RESULT AND DISCUSSION..... 145

4.1 Introduction 145

4.2 GPS based TCAS Design 145

4.3 Vertical Resolution Advisory 148

4.4 Lateral Resolution Advisory 151

4.5 Automatic Reversal Logic 153

4.6 Result and Discussion on Simulating our model 155

CHAPTER FIVE 162

CONCLUSION AND RECOMMENDATION 162

5.1 Conclusion 162

5.2 Recommendation 165

5.3 Future Research Work 167

REFERENCES 168

APPENDIX A 176

APPENDIX B 178

LIST OF FIGURES

Fig. 1.1: System life cycle and error introduction, detection and costs of repairment.....9

Fig. 1.2: Engine Management System.....13

Fig. 2.1: Elements of a Digital Controller 22

Fig. 2.2: Part of a structured development system model 28

Fig. 2.3: ROOM element identification and consolidation process.....31

Fig. 2.4: Verification methodology of model checking 32

Fig. 2.5: The VLSI Design Hierarchy 41

Fig. 2.6: Two Possible system configurations 45

Fig. 2.7: Production Cost.....49

Fig. 2.8: Microchip ICD2 Module 53

Fig. 2.9 : ICD2 Program and Debug System 54

Fig 2.10: Polled I/O Process 56

Fig. 2.11: Basic Interrupt Operation.....58

Fig. 2.12: Basic RTOS Operation 60

Fig. 2.13: The traditional hardware/software codesign methodology.....62

Fig. 2.14: A model based codesign methodology.....63

Fig. 2.15: A periodic arrival pattern. 70

Fig. 2.16: Periodic event stream with jitter.....70

Fig. 2.17: Periodic event stream with burst 71

Fig. 2.18: Interference of two data streams on a shared communication resource.....74

Fig. 2.19: Real Time Calculus processing component 78

Fig. 2.20: Modelling periodic event streams with jitter using arrival curves 78

Fig. 2.21: A scheduling network modelling the application-to-architecture mapping.... 83

Fig. 2.22: Computing upper bounds on the delay, D, and the backlog, B..... 85

Fig. 2.23: Timed automata model for a periodic event stream 91

Fig. 2.24: Timed automata model for a periodic event stream with jitter 91

Fig. 2.25: Timed automata model for a preemptive FP resource with two tasks 92

Fig. 2.26: Timed automata model to measure end- to-end delay..... 94

Fig.2.27: An overview of the model checker UPPAAL..... 97

Fig. 3.1: Model checking 113

Fig. 3.2: Simulation of PIC16F887A Microcontroller using Proteus 7 professional.... 118

Fig. 3.3: Simulation of GPS Based Trajectory Data Receiver..... 119

Fig. 3.4: A flow graph of existing TCAS 132

Fig. 3.5: A flow graph of Proposed TCAS 135

Fig. 3.6: The INTRUDER DETECTION Automaton..... 136

Fig. 3.7: The new SURVEILLANCE Automaton 137

Fig. 3.8: The TRACKING Automaton..... 138

Fig. 3.9: The new COORDINATION Automaton..... 139

Fig. 3.10: TCAS Corrective / Preventive Advisory Threshold..... 140

Fig. 3.11: The RESOLUTION Automaton..... 141

Fig. 3.12: The new REVERSAL Automaton 142

Fig. 4.1: GPS Based TCAS Prototype.....147

Fig. 4.2: TCAS A and TCAS B in the same altitude for vertical resolution 150

Fig. 4.3: TCAS simulation in different altitude for lateral resolution.....152

Fig. 4.4: TCAS in a Reverse Logic mode.....154

LIST OF TABLES

Table 1.1: Toyota recalls by place and problem 12

Table 2.1: Timer Functions..... 55

Table 2.2: Property Equivalences 104

Table 2.3: Comparison of performance analysis methods 107

Table 3.1: Experimental results of simulating PIC16F887A Microcontroller 123

Table 3.2: Performance data of existing TCAS 129

Table 4.1: Result of simulating TCAS A and TCAS B in the same altitude mode 156

Table 4.2: Result of simulating TCAS A and TCAS B on different altitude mode..... 158

Table 4.3: Result of simulating TCAS A and TCAS B in reverse logic mode 160

Table 4.4: Comparison of Tau (Seconds) for existing TCAS and simulated TCAS 161

GLOSSARY

ABS	Anti-lock Break System
AGC	Apollo Guidance Computer
ACSS	Aviation Communication & Surveillance Systems
ASIC	Application Specific Integrated Circuits
BCET	Best Case Execution Time
BNF	Bacchus Normal Form
CCS C	Custom Computer Services Compiler
CFIT	Controlled Flight Into Terrain
CM	Command Module
CMOS	Complementary Metal-Oxide Semiconductor
CPA	Closet Point of Approach
CSMA/CD	Carrier Sense Multiple Access/ Collision Detection
CTL	Computational Tree Logic
DES	Distributed Embedded System
DFD	Data Flow Diagrams
DMA	Direct Memory Access
DSP	Digital Signal Processor
DVD	Digital Versatile Disk
EAF	Event Adaption Functions
ECAM	Electronic Centralised Aircraft Monitor
ECUs	Electronic Control Units
EEPROM	Electrically Erasable Proframmable Read Only Memory
EDP	Electronic Data Processing
EMS	(Electronic) Engine Management System

EMIF	Event Model Interfaces
EPROM	Erasable Proframmable Read Only Memory
EUSART	Enhanced Universal Asynchronous Receiver Transmitter
ESC	Electronic Stability Control
FP	Fixed Priority
GNSS	Global Navigation Satellite System
GPWS	Ground Proximity Warning System
GPS	Global Positioning System
$GPGGA	Global Positioning System Fix Data
$GPGSV	GPS satellites in view
$GPGSA	GPS DOP (Dilution Of Precision) and Active Satellites
ICSP	In-Circuit Serial Programming
ICAO	International Civil Aviation Organization
I²C	Inter-Integrated Circuit
ISR	Interrupt Service Routine
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LM	Lunar Module
LSB	Least Significant Bit
LTL	Linear Temporal Logic
MAC	Mid-Air Collision
MCU	Microcomputer Unit
MTOM	Maximum Take-Off Mass
MSA	Minimum safety Altitude
MOOSE	Model-based Object Oriented Systems Engineering

MPA	Modular Performance Analysis
MP3	Motion Picture expert group layer 3
MRI	Magnetic Resonance Imaging
NHTSA	National Highway Traffic Safety Administration
OIDs	Object Interaction Diagrams
OSCI	Open SystemC Initiative
OTP NVM	One-Time Programmable Non-Volatile Memory
PCB	Printed Circuit Board
PET	Positron Emission Tomography
PEs	Processing Elements
PIC	Peripheral Interface Controller
PLTL	Propositional Linear Temporal Logic
PWM	Pulse Width Modulation
RA	Resolution Advisory
RTE	Real-Time Embedded
RRT	Real-Time Task
ROOM	Real-Time Object-Oriented Modelling
RTCC	Real-Time Counter Clock
RTOS	Real-Time Operating System
RTC	Real-Time Calculus
SNA	System Network Architecture
SoC	System on a Chip
SPI	Serial Peripheral Interface
SPECT	Single Photon Emission Computed Tomography

TA	Traffic Advisory
TAWS	Terrain Awareness Warning System
TCAS	Traffic Alert and Collision Avoidance System
TCS	Ttraction Control System
TDMA	Time Division Multiple Access
TLA	Temporal Logic of Actions
UML	Unified Modelling Language
UPPAAL	A performance analysis tool based on timed automata
VSM	Virtual System Modelling
WCED	Worst Case Execution Demand
WCET	Worst Case Execution Time

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer such as a personal computer can do many different tasks depending on programming. Embedded systems control many of the common devices in use today.

In the 1930-40s, computers were often dedicated to a single task, but were far too large and expensive for most kinds of tasks performed by the embedded computers of today. Over time, however, the concept of programmable controllers evolved from traditional electromechanical sequencers, via solid state devices, to the use of computer technology.

Real-time embedded systems are systems with precise timing requirements constraints. They react to stimuli within a time interval dictated by the environment (Robert, 1996). When a process is considered hard real-time, it must complete its operation by a specific time. Such a real-time constraint is called hard, if not meeting it could result in an impermissible failure of the system, and it is called soft otherwise. In hard real-time embedded system the deadlines must be kept without regard to any other factors, they are considered mission critical. Consider, for example, a car engine control system; such a system is considered hard real–time because a late process could cause the engine to fail.

One of the first recognizably modern embedded systems was the Apollo Guidance Computer, developed in 1960 by Charles Stark Draper at the MIT Instrumentation Laboratory (Michael, 2007). The Apollo Guidance Computer (AGC) was an on-board digital computer installed in each Apollo program spacecraft with both the Command Module (CM) and the Lunar Module (LM). It provided onboard computation to support spacecraft guidance, navigation and control. The AGC was used in real-time by astronaut pilots to collect and provide flight information and to automatically control all of the navigational functions of the Apollo spacecraft.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile released in 1961. It was used as missile guidance systems (Michael, 2007).

Embedded systems range from portable devices such as digital watches and MP3 players that are common nowadays, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units of internetwork individual autonomous microcontroller chips to form distributed embedded systems, with peripherals mounted inside a large chassis or enclosure.

Distributed Embedded System (DES) is a revolution in information technology (IT). It is driven by the increasing capabilities and ever-declining costs of computing and communications devices, resulting in networked systems of embedded computers whose functional components are nearly invisible to end users. Systems have the potential to alter radically the way in which people interact with their environment by

linking a range of devices and sensors that will allow information to be collected, shared, and processed in unprecedented ways. DES conjures up images of pervasive collaboration among connected devices, such as tiny, stand-alone, embedded microcontrollers, networking devices, embedded Personal Computers, robotics systems, computer peripherals, wireless data systems, sensors, and signal processors (Arvindra, 2008).

Embedded systems span all aspects of modern life and there are many examples of their uses. These include Telecommunications systems which employ numerous embedded systems from telephone switches for the network to mobile phones at the end-user. Computer networking uses dedicated routers and network bridges to route data.

Consumer electronics which include Personal Digital Assistants (PDAs), mp3 players, mobile phones, videogame consoles, digital cameras, DVD players, GPS receivers, and printers. Many household appliances, such as microwave ovens, washing machines and dishwashers include embedded systems to provide flexibility, efficiency and many features. Home automation uses wired and wireless-networking that can be used to control lights, climate, security, audio/visual, etc., all of which use embedded devices for sensing and controlling (Michael and Anthony, 2006).

Transportation systems from flight to automobiles increasingly use embedded systems. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. Various electric motors, brushless DC motors, induction motors and DC motors are using electric/electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles are increasingly using embedded systems to maximize efficiency and reduce pollution like in automotive safety systems such as Electronic Stability Control (ESC/ESP), Ttraction Control (TCS) and automatic four-wheel drive.

Medical equipment is continuing to advance with more embedded systems for vital signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging like Positron Emission Tomography **(**PET**),** Magnetic Resonance Imaging **(**MRI**),** Single Photon Emission Computed Tomography **(**SPECT) for non-invasive internal inspections.

The interface of embedded systems range from no user interface at all, which is dedicated only to one task and to complex graphical user interfaces that resemble modern computer desktop operating systems.

Simple embedded devices use buttons, LEDs, and small character or digit-only displays, often with a simple menu system to a more complex system which has a full graphical screen, with touch sensing or screen-edge buttons provides flexibility while minimizing space used, the meaning of the buttons can change with the screen, and selection involves the natural behaviour of pointing at what's desired. Handheld systems often have a screen with a "joystick button" for a pointing device.

The rise of the World Wide Web has given embedded designers another quite different option, providing a web page interface over a network connection. This avoids the cost of a sophisticated display, yet provides complex input and display capabilities when needed on another computer. This is successful for remote permanently installed equipment such as Pan-Tilt-Zoom cameras and network routers.

There are several different types of software architectures in common use. Some of them are Simple control loop, Interrupt controlled system, Cooperative multitasking, Preemptive multitasking or multi-threading, Microkernels and exokernels and Monolithic kernels.

Modular Performance analysis of real-time embedded system plays an important role in the design process of complex embedded systems for analyzing essential performance characteristics of system design at an early phase.

It gives the choice of important design decisions before much time and resources are invested in detailed implementation. Some of the approaches to performance analysis are simulation and formal analysis methods, real-time calculus, holistic scheduling analysis and compositional method (Nikolay and Lothar, 2007).

Many people have worked on this research area before but with some limitations; they could not model complicated programs and they could not model advanced micro-architectural features of the processor, such as cache memories and pipelines. This research work will identify and proffer solutions to some of these limitations.

1.2 Research Objectives

The objectives of this research are to:

i Determine the most appropriate methods for system-level performance analysis of real-time embedded systems

ii Model and analyse a GPS-based Traffic Alert and Collision Avoidance System

iii Investigate the possibility of incorporating reversal logic into a GPS- based Traffic Alert and Collision Avoidance System.

1.3 Motivation and Requirements for Modular Performance Analysis

The embedded system designers face the problem of evaluating many candidate hardware-software architectures with respect to various performance indexes in the early design cycle. These indexes may include system's throughput, response times, end-to-end delays, resource utilization and memory requirements. In most cases, building a prototype for each design alternative to directly measure these performance

characteristics is infeasible because of high implementation costs and stringent time-to-market constraints. On the other hand, due to the increasing complexity of modern embedded systems, back-of-the-envelope estimations cannot be used without taking the risk of being totally incorrect. As a result, the only option left for the designers is to carry out the performance analysis based on some kind of a performance model of the system.

Modular Performance analysis plays an important role in the design process of complex embedded systems for analyzing the most important performance characteristics of system design at an early design phase. The choices of important design decisions before much time and resources are invested in detailed implementation are made.

Embedded systems are typically reactive systems that are in continuous interaction with their physical environment to which they are connected through sensors and actuators. Consequently they must execute at a pace determined by their environment.

Generally, embedded systems must be highly efficient in terms of power consumption, size and cost, for example these are desirable in space exploration. In addition, they usually have to be highly dependable, as a malfunction or breakdown of the device they control is not acceptable. For many embedded systems not only the availability and the correctness of the computations is relevant, but also the timeliness of the computed results. Such systems with precise timing requirements are called real-time embedded systems. Their temporal behaviour in terms of computation times, latencies and end-to-end delays is a functional system requirement. This means that a right answer arriving too late (or even too early) is wrong. Thus, real-time does not simply mean fast, but is a synonym for timely and predictable.

This means that many embedded systems must meet real-time constraints, that is, they must react to stimuli within a time interval dictated by the environment. Such a real-

time constraint is called hard, if not meeting it could result in an impermissible failure of the system, and it is called soft otherwise.

A real-time system is expected to respond not just quickly, but also within a predictable period of time. A good example of a real-time computer system is a car's anti-lock break system (ABS). ABS is expected to release a vehicle's brakes, preventing dangerous wheel locking in a predictably short time frame. A non-real-time computer process on the other hand does not have a deadline. Such a process can be considered non-real-time, even if fast results are preferred.

Unfortunately, there are times when real-time systems fail to respond as desired. A real-time process fails when its task is not completed before its deadline. In computing, there is no grace period given because of other demands on a system. Real-time deadlines must be kept without regard to other factors, they are considered mission critical.

When a process is considered hard real-time, it must complete its operation by a specific time. If it fails to meet its deadline, its operation is without value and the system for which it is a component could face failure. When a system is considered soft real-time, however, there is some room for lateness. For example, in a soft real-time system, a delayed process may not cause the entire system to fail. Instead, it may lead to a decrease in the usual quality of the process or system.

Hard real-time systems are often used in embedded systems. Hard real-time systems are employed when it is crucial that a task or event is handled by a strict deadline. This is typically necessary when damage or the loss of life may occur as a result of a system failure.

Soft real-time systems are usually employed when there are multiple, connected systems that must be maintained despite shifting events and circumstances. These systems are also used when concurrent access requirements are present. For example,

the software used to maintain travel schedules for major transportation companies is often soft real-time. It is necessary for such software to update schedules with little delay. However, a delay of a few seconds is not likely to cause the kind of mayhem possible when a hard real-time system fails.

In practice, embedding into a larger product and the constraints imposed by a special application domain very often lead to distributed implementations. In particular, physical performance, safety and modularity constraints require distributed solutions. Distributed embedded systems consist of several hardware components that communicate via an interconnection network. Often the network is a shared resource that is contended for by several processing units. For instance, the network can be a data bus that handles various data streams, the individual computing nodes are not synchronized and communicate via message passing. They have a high degree of independency and make autonomous decisions concerning resource access and scheduling. As a result, it is particularly complex task to maintain global state information. In addition, in many cases each computing node has an individual functionality and a particular local environment. Thus, each node is provided with specific and adapted hardware resources, which leads to highly heterogeneous implementations. For instance in a car, the embedded control units responsible for engine control, power train control and stability control have a highly heterogeneous composition. In view of these points, it is obvious that heterogeneous and distributed embedded systems are inherently complex to design and analyze.

System level performance plays an important role in the design process of complex embedded systems. It allows analyzing essential performance characteristics of a system design at an early phase and consequently supports the choice of important design decisions before much time and resources are invested in detailed implementation.

Embedded systems are characterized by the presence of processors running application-specific software. Recent years have seen a large growth of such systems, and this trend is projected to continue with the growth of systems on a chip. Many of these systems have strict performance and cost requirements. To design these systems, sophisticated timing analysis tools are needed to accurately determine the extreme case (best case and worst case) performance of the software components (Yau-Tsun and Sharad, 1999).

There is a need for integration of performance analysis of system into design process. Errors occur at various stages of design process, interviews with several software engineering companies have resulted in some interesting figures for the costs of error repairment and statements about the stages in the design process where errors are introduced and detected. These investigations show once more that integration of performance analysis of system into the design process should take place at an early stage, particularly from an economical point of view (Fig.1.1).

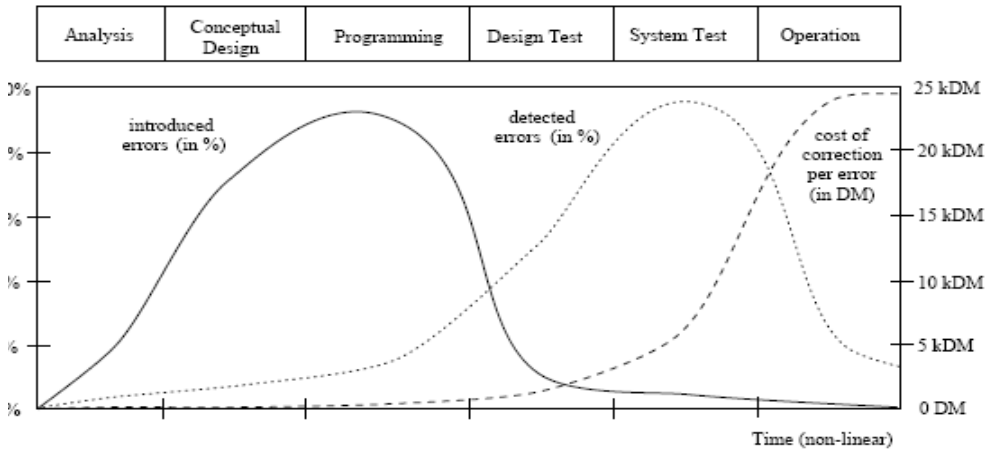


Fig. 1.1: System life cycle and error introduction, detection and costs of repairment

About 50% of all errors are introduced during the programming phase, i.e. the phase in which actual code is generated. The conceptual design phase also contributes

significantly to the introduction of errors. Whereas only about 15% of all errors have been detected before the test phase start, it is evident that most errors are found during testing, in particular during system integration tests, where software modules are glued together and the entire system is investigated.

Errors that are detected before the test phase can on average be repaired very cheaply. Investigation carried out in Germany showed about 500 German Marks (i.e. about 320 US dollar) per error (Liggesmeyer *et. al.*, 1998). In the design test, while testing the software modules on a stand-alone basis, this cost increases to 2,000 DM, reaching a maximum of 25,000 DM per error when an error is demonstrated during system operation (field tests). It is clear that it is of vital importance to detect errors as early as possible in the system design process, the costs to repair errors are substantially lower, and the impact of errors on the rest of the design is less substantial.

For example, three separate but related recalls of automobiles by Toyota Motor Corporation occurred at the end of 2009 and start of 2010. Toyota initiated the recalls, the first two with the assistance of the U.S. National Highway Traffic Safety Administration (NHTSA), after several vehicles experienced unintended acceleration. The first recall, on November 2, 2009, was to correct a possible incursion of an incorrect or out-of-place front driver's side floor mat into the foot pedal which can cause pedal entrapment. The second recall, on January 21, 2010, was begun after some crashes were shown not to have been caused by floor mat incursion. This latter defect was identified as a possible mechanical sticking of the accelerator pedal causing unintended acceleration, referred to as Sticking Accelerator Pedal by Toyota. The original action was initiated by Toyota in their Defect Information Report, dated October 5, 2009, amended January 27, 2010 (Toyota USA Newsroom). Following the

floor mat and accelerator pedal recalls, Toyota also issued a separate recall for hybrid anti-lock brake software in February 2010 (Takahashi and Yoshio).

As of January 28, 2010, Toyota had announced recalls of approximately 5.2 million vehicles for the pedal entrapment/floor mat problem, and an additional 2.3 million vehicles for the accelerator pedal problem. Approximately 1.7 million vehicles are subject to both. Certain related Lexus and Pontiac models were also affected. The next day, Toyota widened the recall to include 1.8 million vehicles in Europe and 75,000 in China. By then, the worldwide total number of cars recalled by Toyota stood at 9 million. Sales of multiple recalled models were suspended for several weeks as a result of the accelerator pedal recall, with the vehicles awaiting replacement parts. As of January 2010, 21 deaths were alleged due the pedal problem since 2000, but following the January 28 recall, additional NHTSA complaints brought the alleged total to 37 (http://pressroom.toyota.com, 2010).

The recall came at a difficult time for Toyota, as it was struggling to emerge from the recession and had already suffered from a resultant decrease in sales and the low exchange rate from yen to US dollar. On the day that the recall was announced in the US, it was announced that 750 jobs would be cut at Toyota's British plant at Burnaston, near Derby. It was estimated that each Toyota dealership in the US could lose between $1.75 million to $2 million a month in revenue, a total loss of $2.47 billion across the country from the entire incident. Additionally, Toyota Motors as a whole announced that it could face losses totalling as much as $2 billion from lost output and sales worldwide. Between 25 January and 29 January 2010 Toyota shares fell in value by 15%.(Guardian.co.uk, 2008). The summary of Toyota recall and problem is shown in Table 1.1.

Table 1.1: Toyota recalls by place and problem

Region/country	Hybrid braking 09/02/2010	Accelerator Pedals, 28/1/2010	Slipping floor mat 29/09/2009	Total recall problems
Canada	8,450	270,000	400,000	678,450
China		75,552		75,552
Other Europe	44,500	1,530,000		1,574,500
Japan	223,000			223,000
US	146,550	2,210,000	5,350,000	7,706,550
UK	8,500	180,000		188,500
Other	5,000	180,000		185,000
Total	436,000	4,445,552	5,750,000	10,631,552*

Source: (Guardian.co.uk, 2008)

*About 2.1 million vehicles are covered in both floormat and pedal recalls. Total vehicles approximate: 8.54m

Nowadays automotive industry generally employs what is called engine management systems with complex processors. The application embeds microprocessors that are fast enough to provide a real-time operation. This was firstly used in Formula 1 racing cars, but it didn't take long until regular cars used them as well. (www.vehicle-lab.net, 2010). This embedded systems, Electronic Control Units (ECUs) use a microprocessor which can process the inputs from the various sensors in real time (figure 1.2). ECUs contain both the hardware and software (firmware). The hardware consists of electronic components on a Printed Circuit Board (PCB), ceramic substrate or a thin laminate substrate. The main component on this circuit board is a microcontroller chip (CPU). The software is stored in the microcontroller or other chips on the PCB, typically use

EPROMs or flash memory so the CPU can be re-programmed by uploading updated code or replacing chips. This is referred to as an (electronic) Engine Management System (EMS).

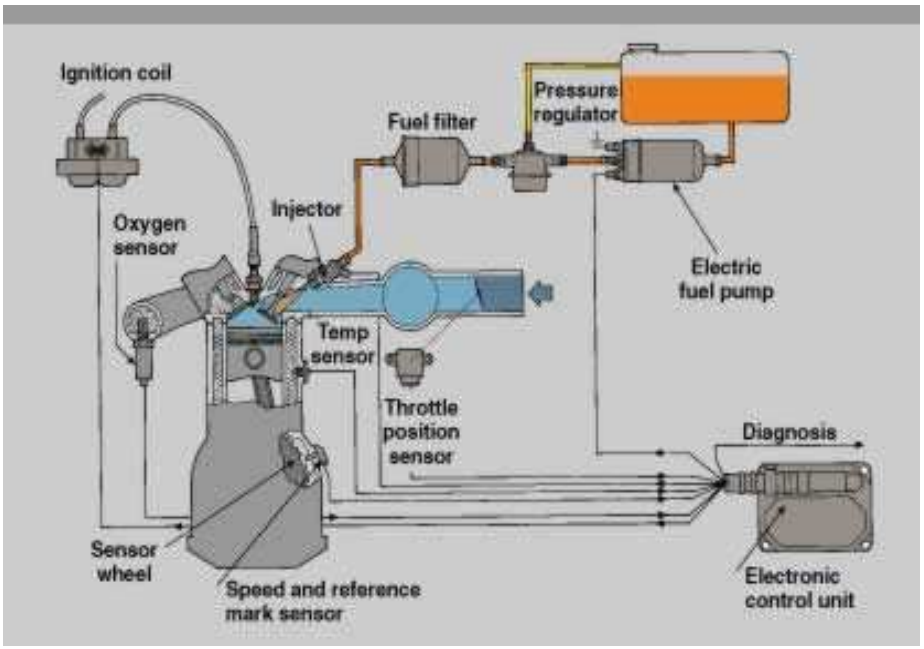


Fig. 1.2: Engine Management System

Source: (www.vehicle-lab.net, 2010)

Despite the use of EMS, product recalls happen all the time in the auto industry and elsewhere. They are almost always embarrassing because they reveal short-comings in research and development that have failed to be ironed out before the product has reached full-scale production (www.guardian.co.uk, 2010).

Errors can occur at various stages of design process, errors can occur at analysis, conceptual design, programming, design test, system test and operation. But the error may not be detected until at the final system test or in an actual operation as in the case of Toyota cars recalls.

Without overestimating, from an economical point of view, we can fairly conclude that there is a strong need for an integration of Performance Analysis of System into design process at an early stage.

1.4 Methodology

Systems such as vehicles, appliances, power plants and air space, are controlled by computer systems interacting in real-time with their environments. Since failure of many of these real-time computer systems may endanger human life or substantial economic values, there is a high demand for development of methods which minimize the risk of failure. A common cause of failure is timing violations.

Many embedded systems must meet real-time constraints, i.e. they must react to stimuli within time intervals dictated by the environment. It is apparent that heterogeneous and distributed embedded real-time systems are inherently difficult to design and to analyse because of the tight interaction between computation, communication and availability. Part of this difficulty is caused by the fact that the functional and extra- functional behaviour of the system is influenced by interferences on shared resources such as processors, memory or communication devices. As a result, resource sharing strategies influence the system behaviour to large extents. Moreover, the system environment which continuously interacts with the embedded system may vary and cause an additional degree of non-determinism.

Modular Performance Analysis (MPA) framework is a powerful framework for formal system level performance analysis of real-time embedded systems. Wandeler (2006) is one of the researchers that have worked in this area but with some limitations. Among his main contributions, he addressed the challenges of complex inputs, variable execution demands, and workload correlations in performance analysis of complex

embedded systems. He introduced components that enable the modelling and analysis of tasks with multiple inputs, where the task activation is determined as a Boolean function of the event availability on the various inputs. In the domain of embedded systems, Wandeler introduced greedy shaper into MPA that have two main applications. First, shapers is used in the system internally to re-shape internal traffic streams to reduce global buffer requirements and end-to-end delays and secondly, shapers may be added at the boundaries of a system, to ensure conformant input streams and to thereby prevent internal buffer overflows caused by malicious input. He used Real-Time Calculus (RTC) method to efficiently compute various curve operation on compact variability characterization curves. The theory of Real-Time Interfaces that connects the principles of Real-Time Calculus and interface-based design was introduced, together with a component system that enables interface-based embedded real-time system design. Novel models and methods were introduced that facilitate the modelling, analysis, and design of hierarchical scheduling policies in complex embedded systems. A new MATLAB Toolbox was introduced that supports efficient system level performance analysis and interface-based design of distributed embedded real-time systems. He presented a case study of a distributed in-car navigation system, where the MPA framework was used to answer design questions that typically arise in the early design phase of such a system. This case study demonstrates how performance analysis with the MPA framework can be seamlessly embedded into a Unified Modelling Language (UML) - based design process and it presents the first application of sensitivity analysis within the MPA framework. Wandeler used automata, scheduling and Real Time Calculus method. He used automata method to obtain the tightest analysis results for both the minimum required processor frequency, as well as to guarantee the remaining processing capacity. Scheduling method to ensure that

individual timing requirements of task sharing CPU resources are not violated. And Real-Time Calculus method to efficiently compute various curve operation on compact variability characterization curves in analysing complex distributed embedded real-time systems. Wandeler employed RTC to analyze complex distributed embedded real-time systems but with the following limitations:

He could not address the challenge of timing correlations in complex embedded systems. Problem of cyclic dependencies and fixed-point calculation within performance models left unanswered as well as the problem with dynamic systems with feedback and state-dependant behaviour.

In order to address these limitations, a review of approaches to performance analysis related literature was carried out. Notable among them are simulation based method, formal analysis, real-time calculus, holistic scheduling analysis, timed automata based performance analysis and compositional methods.

Codesign methodology was introduced which emphasized the heterogeneous implementation generation for the simultaneously verification of both software and hardware and in what extent it fits into the desired function.

Real-Time Object-Oriented Modelling (ROOM) was employed in the abstraction of the components since the existing techniques could not be easily retargeted for new hardware platforms.

Microcontroller used for our embedded real-time systems design was modelled using Proteus 7 professional. Embedded real-time systems design was modelled and analysed using timed automata based performance analysis method, “UPPAAL”.

The design process involved the construction of an essential model which described the required system behaviour, the production of an implementation model where the implementation technology that embodies the required behaviour was described and

system built where the implementation model was transformed into hardware and software.

Finally, the research work extended the framework to model the design and implementation of MPA in avionics. The Traffic Alert and Collision Avoidance System (TCAS) was modelled with UPPAAL, the prototype was designed and simulated using MATLAB to ascertain the correctness, integrity and safety of the system.

1.5 Contribution to Knowledge

In this research work, UPPAAL, a tool for performance analysis of Real-time embedded systems was used for modelling and analysis of our proposed GPS based TCAS which was further simulated using MATLAB to ascertain the correctness, integrity and safety of the system with the following contributions:

a. Introduction of horizontal resolution directives which are useful in a conflict between two aircrafts close to the ground where there may be little, if any, vertical manoeuvring space possible.
b. Introduction of traffic situation given within a “time-based" representation because the existing TCAS is range-based that displays the traffic situation within a configurable ranges of miles/feet, however, under certain condition there may be need for “time-based" representation.
c. Introduction of Reversal logic where an RA (Resolution Advisory) issued by TCAS is reversed when it has been shown that after few seconds one of the two aircrafts does not comply with the advisory given earlier by the TCAS.

These will prevent collision and consequently improve safety and reduce the stress of both the controller and pilot.

1.6 Organisation of the rest of thesis

The rest of the thesis is organised as follows: Chapter two focuses on the review of related literature, it covers introduction to Real-time embedded systems modelling, Temporal logic, modelling approach, requirements for embedded systems design, Real-time embedded systems design performance indexes, performance analysis methods and comparison of these methods. Chapter three presents modelling and analysis of Real-time embedded systems using "UPPAAL". Chapter four covers design of TCAS real time embedded systems prototype, result and discussion while chapter five concludes the work.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The major component of an embedded system is a microcontroller, which is also known as a microcomputer unit, (MCU) or μC. MCU is a small computer on a single integrated circuit consisting internally of a relatively simple CPU, clock, timers, I/O ports, and memory. Program memory in the form of NOR flash or One-Time Programmable Non-Volatile Memory (OTP NVM) is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for small or dedicated applications. Thus, in contrast to the microprocessors used in personal computers and other high-performance or general purpose applications, simplicity is emphasized. Some microcontrollers may use four-bit words and operate at clock rate frequencies as low as 4 kHz, as this is adequate for many typical applications, enabling low power consumption (milliwatts or microwatts). They will generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt. Power consumption while sleeping (CPU clock and most peripherals off) may be just nanowatts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a Digital Signal Processor (DSP), with higher clock speeds and power consumption. Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, and toys. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and

input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems.

The first single-chip microprocessor was the 4-bit Intel 4004 released in 1971, headed by Intels lead research scientist Hunter H. Hetfeld, the Intel 8008 and more capable microprocessors available over the next several years. These, however, all required external chip(s) to implement a working system, raising total system cost, and making it impossible to economically computerise appliances. The first computer system on a chip optimised for control applications microcontroller was the Intel 8048 released in 1975 with both RAM and ROM on the same chip. This chip would find its way into over one billion PC keyboards, and other numerous applications. At this time Intels President, Luke J. Valenter, stated that the Microcontroller was one of the most successful in the company's history, and expanded the division's budget over 25%. Most microcontrollers at this time had two variants. One had an erasable EEPROM program memory, which was significantly more expensive than the PROM variant which was only programmable once. In 1993, the introduction of EEPROM memory allowed microcontrollers beginning with the Microchip PIC16x84 to be electrically erased quickly without an expensive package as required for EPROM, allowing both rapid prototyping, and in System Programming (http://semico.com, 2010).

The same year, Atmel introduced the first microcontroller using Flash memory (Svendsli, 2003). Other companies rapidly followed suit, with both memory types. This development reduced the cost drastically.

Nowadays the microcontroller unit (MCU) can be big, or rather small in electronics. It is one of the most significant developments in the continuing miniaturization of electronic hardware. Now, even trivial products, such as a musical birthday card or

electronic price tag can include an MCU. They are an important factor in the digitization of analog systems, such as sound systems or television. In addition, they provide an essential component of larger systems, such as automobiles, robots, and industrial systems. There is no escape from microcontrollers, so it is pretty useful to know how they work (Martin, 2003).

The computer or digital controller has three main elements; input and output devices, which communicate with the outside world; a processor, to make calculations and handle data operations and memory, to store programs and data. Fig. 2.1 shows these in a little more detail. Unlike the conventional microprocessor system such as a Personal Computer (PC), which has separate chips on a printed circuit board, the microcontroller contains all these elements in one chip. The MCU is essentially a computer on a chip; however, it still needs input and output devices, such as a keypad and display, to form a working system. The microcontroller stores its program in ROM (Read Only Memory). In the past, UV (ultraviolet) Erasable Programmable ROM (EPROM) was used for prototyping or small batch production, and one-time programmable ROM for longer product runs.

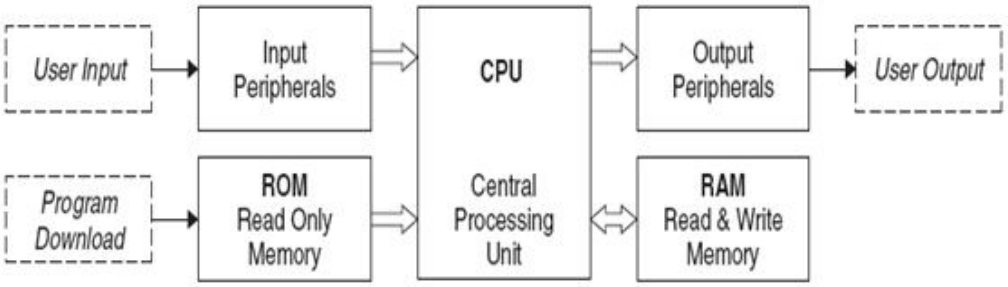


Fig. 2.1: Elements of a Digital Controller

Source: (Martin, 2003)

Programmable ROM chips are programmed in the final stages of manufacture, while EPROM could be programmed by the user. Flash ROM is now normally used for prototyping and low-volume production. This can be programmed in circuit by the user after the circuit has been built. The prototyping cycle is faster, and software variations are easier to accommodate. Nowadays flash ROM is used in USB memory sticks, digital camera memory, and so on, with Gb capacities (Svendsli, 2003).

2.2 Real-time Embedded Systems Modelling

Real-time Embedded Systems is a reactive systems, Lichtenstein *et. al.* (1985) coined the term reactive systems as been characterized by a continuous interaction with their environment. They typically continuously receive inputs from their environment and, usually within quite a short delay, react on these inputs. Typical examples of reactive systems are operating systems, distributed embedded systems which can be found in aircraft control systems, communication protocols, and process control software. For instance, a control program of a chemical process receives control signals regularly, like temperature and pressure, at several points in the process. Based on this information, the

program can decide to turn on the heating elements, to switch off a pump, and so forth. As soon as a dangerous situation is anticipated, for example, the pressure in the tank exceeds certain thresholds, the control software needs to take appropriate action. Usually reactive systems are rather complex; the nature of their interaction with the environment can be intricate and they typically have a distributed and concurrent nature. Correctness and well-functioning of reactive systems is crucial. To obtain correctly functioning and dependable reactive systems a coherent and well-defined methodology is needed in which different phases can be distinguished.

In the first phase, a thorough investigation of requirements is needed and as a result a requirement specification is obtained.

Secondly, a conceptual design phase results in an abstract design specification. This specification needs to be validated for internal consistency and it needs to be checked against the requirement specification. This validation process can be supported by formal verification simulation and model checking techniques, where a model (like a finite-state automaton) of the abstract design specification can be extensively checked.

Once a trustworthy specification has been obtained, the third phase consists of building a system that implements the abstract specification. Typically, testing is a useful technique to support the validation of the realization versus the original requirement specification.

A complementary technique to simulation and testing is to *prove* that a system operates correctly, in a similar way as one can prove that, for example, the square of an even number is always an even number. The term for this mathematical demonstration of the correctness of a system is formal verification. The basic idea is to construct a formal (that is, mathematical) *model* of the system under investigation which represents the possible behaviour of the system. In addition, the correctness requirements are written

in a formal *requirement specification* that represents the desirable behaviour of the system. Based on these two specifications one checks by formal proof whether the possible behaviour "agrees with" the desired behaviour. Since the verification is treated in a mathematical fashion, the notion of "agree with" can be made precise, and verification amounts to proving or disproving the correctness with respect to this formal correctness notion.

i A model of the system, typically consist of a set of states, incorporating information about values of variables, program counters and the like, and a transition relation, that describes how the system can change from one state to another.

ii A specification method for expressing requirements in a formal way.

iii A set of proof rules to determine whether the model satisfies the stated requirements.

2.2.1 Temporal Logic

From section 2.2 above, one can deduce that the correctness of a reactive system refers to the behaviour of the system over time, it does not necessarily refer only to the input-output relationship of a computation (as pre- and post conditions do), since usually computations of reactive systems do not terminate. Consider, for instance, a communication protocol between two entities, a sender *P* and a receiver, which are connected via some bidirectional communication means. A property like:

"if process *P* sends a message, then it will not send the next message until it receives an acknowledgement" cannot be formulated in a pre- and post condition way.

In other to facilitate the formal specification of these types of properties, propositional logic has been extended by some operators that refer to the behaviour of a system over

time. U (until) and G (globally) are examples of operators that refer to sequences of states (as executions). □U ψ means that property □ holds in all states until a state is reached in which ψ holds, and G □means that always, that is in all future states, □holds. Using U and G operators for instance, one can formalize the statement for the protocol,

$G[snd_p (m) \Longrightarrow \neg (snd_p (nxt(m)) \text{ U } rcv_p(ack))]$.

Stated in words, if a message *m* is sent by process *P,* then there will be no transmission by *P* of some next message (nxt(m)) until an acknowledgement has been received by *P*.

Logics extended by operators that allow the expression of properties about executions, in particular those that can express properties about the relative order between events, are called *temporal logics.* Such logics are rather well established and have their origins in other fields many decades ago. The introduction of these logics into computer science is due to Pnuelia (1977). Temporal logic is a well accepted and commonly used specification technique for expressing properties of computations of reactive systems at a rather high level of abstraction.

The Temporal Logic of Actions (TLA) of Lamport (1994) forms the basis of a good approach for reactive systems that would be applied in this research work. This approach allows one to specify requirements and system behaviour in the same notation in model checking for the modular performance analysis of real time embedded systems.

2.3 Modelling Approach

A model is defined as being an abstraction of the real system omitting uninteresting details. This enables large systems to be simulated where only the components of interest are modelled in detail.

A model also can be a representation of a process - a weather pattern, traffic flow, air flowing over a wing. Models are created from a mass of data, equations and computations that mimic the actions of things represented. Models usually include a graphical display that translates all this number crunching into an animation that you can see on a computer screen or by means of some other visual device.

In modelling it is only necessary to consider those aspects of the system that affect the problem under investigation. Nevertheless, the model should be sufficiently detailed to permit valid conclusions to be drawn about the real system. In the next section, some of the approaches for Real-time Embedded Systems modelling are discussed.

2.3.1 General Purpose Approach for Modelling

This is a traditional approach where Structured Analysis is often used to design traditional electronic data processing (EDP) applications which are to be implemented in traditional imperative programming languages. Structured development for real-time systems has a number of concepts in common but is extended to cover those aspects important for real time systems, namely concurrency and communication. Both methodologies are a combination of formal and informal models used to describe a system. Therefore, apart from simple consistency checks, no model verification can be carried out.

2.3.2 Structured Analysis (SA) Approach

Structured Analysis (SA) proposes a step wise decomposition of Data Flow Diagrams (DFD) so that each process in a *parent* DFD is broken down into a number of *child* DFDs (Yourdon and Constantine, 1979). SA proposes that a system should be modelled in two main steps. Firstly a *context diagram* is developed which shows how the system is connected to its environment. The user defines this context diagram in terms of

sources and sinks of the environment, processes, data flows and files. In the second step the user partitions and refines the system with each process of a DFD being described in increasing detail until atomic processes emerge. The user then describes the algorithmic structure in an informal mini spec. A data dictionary is created containing the data structure and also enforces consistent naming of processes, data flows and many more.

2.3.3 Structured Development for Real-Time Systems Approach

Structured development for real time systems is an answer to the inadequacies of both a purely data flow oriented approach to real time systems analysis and a purely program design oriented approach to real time systems design (Ward and Mellor, 1985). The design process is considered to consist of three main activities:

i. the construction of an *essential model* which describes the required system behaviour;

ii. the production of an *implementation model* where the implementation technology that embodies the required behaviour is described;

iii. *system build* where the implementation model is transformed into hardware and software.

The main tools for constructing models are data flow graphs extended with the addition of events (dashed arcs), double headed arcs that represent continuous data flow (that is, there is always a data value present) and control transformations (dashed circles) which represent finite state machines reacting to and producing events controlling transformations. Fig.2.2 shows part of a climate control system described using the extended data flow notation.

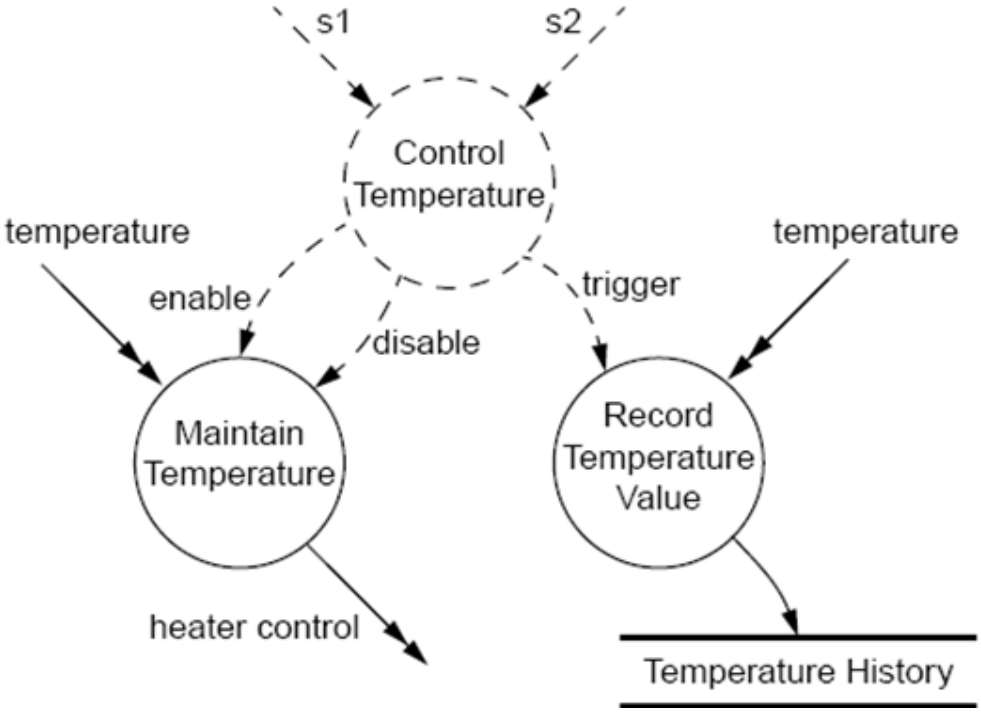


Fig. 2.2: Part of a structured development system model

Source: (Martin, 2003)

This approach is interesting as it combines event based and transformational modelling in a single representation. System models are semi formal, as executable models are not created and only basic consistency checks are possible. A proposal to formalise this approach with an executable model based on Petri nets is described in (Pulli *et. al.*, 1988).

2.3.4 General Purpose Object Oriented Approach

The popularity of the object oriented programming paradigm has resulted in a number of methodologies being developed with the aim of creating object based systems. The object oriented approach is a continuation of the work done in systems engineering and maintains proven concepts of information hiding, coupling and cohesion. Here, two

methodologies are presented; the Booch method, designed for the creation of standard object oriented applications and Real-Time Object-Oriented Modelling, for the design of object oriented real time systems.

(i) The Booch Approach

The Booch approach is a development method used to develop and communicate the analysis and design of a *software* system (White,1994). It is object oriented in that objects are used to model an abstraction of the real world (Booch, 1994). The Booch method is a set of guidelines and techniques for creating object oriented software systems. It is not formal, as only at the system design stage are executable models implementations produced. Nevertheless it is a valuable guide in producing any type of objects oriented system.

(ii) Real-Time Object-Oriented Modelling

Real-Time Object-Oriented Modelling (ROOM) is an approach for the design of real time systems. In ROOM, a system is described using three distinct types of objects: *actors* are used to describe concurrent behaviour and can have a hierarchical structure while *protocols* are objects used in a message passing environment and *data*. Actors exchange messages consisting of protocol objects and data modelled in the implementation language (Fig.2.3). The behaviour of actors is defined with ROOMCharts; a StateCharts like language. ROOMCharts are hierarchical finite state machines where messages replace the events of State-Charts and a state-transition is triggered by existing approaches and the arrival of a message from a predetermined port and the execution of a guard function. As a result of a state-transition occurring, messages can be sent and the extended state (not explicitly defined in the ROOMChart) of an actor modified through the execution of functions (Selic and Gullekson, 1994).

Associated with ROOM is other design methodology to be implemented in this research work as adapted from (Robert, 1996), which are discussed follow:

- Define the system and its boundary with the environment. This boundary may change during the course of developing a system if, for example, more intelligence is delegated to the input sensors and output actuators formulate scenarios (Rumbaugh *et. al.*, 1991), equivalent to *Use Case Analysis*, that the system must implement. Initial scenarios focus on system interaction leading to more detailed scenarios, including error scenarios. Scenarios are the main method of describing the required system functionality from the scenarios derive key protocols that cross the system boundary that capture an early view of the system structure. This structure can be derived by analysing scenarios and identifying *candidates* for concurrent execution between which messages flow, implementing scenarios define an initial set of actors implied by the concurrent physical components described in the scenarios. The defined associated protocol and data classes key actors are developed and prototyped, the system is validated at various stages during the development of the component class definitions.
- The consequences from a distributed implementation are analysed, resulting in further refinement iterations.

This approach is model based and consists of continual refinement of models representing the system and its environment. This refinement uses previous models, requirements and scenarios to produce new or refined models.

ROOM is well suited to designing software based, control dominated, real time systems. The asynchronous nature of communication between actors makes it well suited for modelling distributed systems.

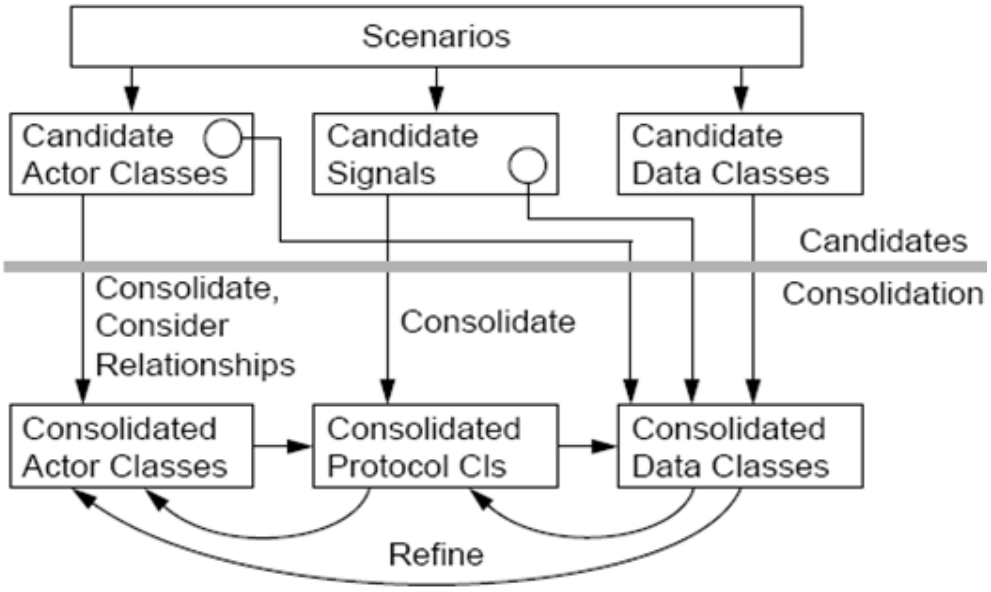


Fig. 2.3: ROOM element identification and consolidation process

Source: Selic and Gullekson, (1994).

2.4 Model checking/analysis

The basic idea of what is known as *model checking* is to use algorithms, executed by computer tools, to verify the correctness of systems. The user inputs a description of a model of the system (the possible behaviour) and a description of the requirements specification (the desirable behaviour) and leaves the verification up to the machine. If an error is recognized the tool provides a counter-example showing under which circumstances the error can be generated. The counterexample consists of a scenario in which the model behaves in an undesired way. Thus the counter-example provides evidence that the model is faulty and needs to be revised. In some cases the formal requirement specification might be wrong, in the sense that the verification engine checks something which the user did not intend to check. Here, the user probably made a mistake in the formalization of the requirements.

This allows the user to locate the error and to repair the model specification before continuing. If no errors are found, the user can refine its model description (for example by taking more design decisions into account, so that the model becomes more concrete or realistic) and can restart the verification process (Fig.2.4).

The algorithms for model checking are typically based on an *exhaustive state space search* of the model of the system for each state of the model it is checked whether it behaves "correctly", that is, whether the state satisfies the desired property. In its most simple form, this technique is known as *reachability analysis.*

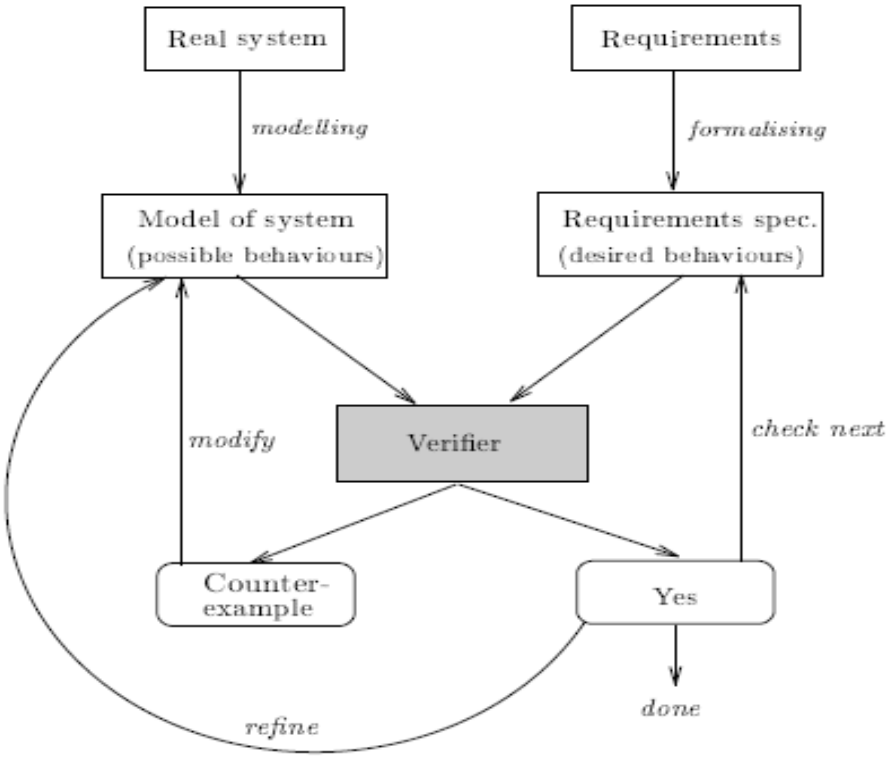


Fig. 2.4: Verification methodology of model checking

Source: Joost-Pieter (1999).

For example, in order to determine whether a system can reach a state in which no further progress is possible (a so-called deadlock), it suffices to determine all reachable

states and to determine whether there exists a reachable state from which no further progress can be made. Reachability analysis is only applicable to proving freedom from deadlock and proving invariant properties, properties that hold during an entire computation. This is, for example, insufficient for communication protocols where a paramount concern is, for instance, that once a message is sent it is eventually being received. These types of "progress" properties are not covered by a usual reachability check. There has been quite some work done on automated reachability analysis for communication protocols in the late seventies and early eighties (West, 1982). Here, protocols were modelled as a set of finite-state automata that communicate via bounded buffered, asynchronous message passing. Starting from the initial system state which is expressed in terms of the states of the interacting automata and message buffers, all system states are determined that can be reached by exchanging messages. Protocols like X.21, X.25, and IBM SNA protocols have been analyzed automatically using this technique. Model checking can in fact be considered as a successor of these early state-space exploration techniques for protocols. It allows a wider class of properties to be examined, and handles state spaces much more efficiently than these early techniques.

2.4.1 Model checking approach

There are basically two approaches in model checking that differ in the way the desired behaviour, that is the requirement specification, is described:

i Logic-based or heterogeneous approach:

In this approach, the desired system behaviour is captured by stating a set of properties in some appropriate logic, usually some temporal or modal logic. A system usually modelled as a finite-state automaton, where states represent the values of variables and control locations, and transitions indicate how a system can change from one state to

another, is considered to be correct with respect to these requirements if it satisfies these properties for a given set of initial states.

ii Behaviour-based or homogeneous approach:

In this approach, both the desired and the possible behaviour are given in the same notation (for example an automaton), and equivalence relations (or pre-orders) are used as a correctness criterion. The equivalence relations usually capture a notion like "behaves the same as", whereas the pre-order relation represents a notion like "behaves at least as". Since there are different perspectives and intuitions about what it means for two processes to "behave the same" (or "behave at least as"), various equivalence (and pre-order) notions have been defined.

One of the most well-known notions of equivalence is bisimulation. In a nutshell, two automata are bisimilar if one automaton can simulate every step of the other automaton, and vice versa. A frequently encountered notion of pre-order is (language) inclusion. An automaton *A* is included in automaton *B,* if all words accepted by *A* are accepted by *B*.

A system is considered to be correct if the desired and the possible behaviours are equivalent (or ordered) with respect to the equivalence (or pre-order) attribute under investigation.

Although these two techniques are conceptually different, connections between the two approaches can be established in the following way. In particular, logic induces an equivalence relation on systems as follows:

Two systems are equivalent if (and only if) they satisfy the same formulas. Using this concept, relationships between different logics and equivalence relations have been established. For instance, it is known that two models that are bisimilar satisfy the same formulas of the logic CTL (Computational Tree Logic), a logic that is commonly used

for model checking. The connection between the two different approaches is possible; if two models possess the same properties (checked using the logic approach), then they are behaviourally equivalent (as could be checked in the behavioural approach). The reverse direction is more interesting, since in general it is infeasible to check all properties of certain logic, whereas checking for equivalences like bisimulation can be done rather efficiently in time logarithmically in the number of states and linear in the number of transitions.

In this work we focus entirely on the logic-based approach. This approach originates from the independent work of two pairs: Quielle and Sifakis (1982) and Clarke and Emerson (1981). Since, in logical terms, one checks that the system description is a model of the temporal logic formulas, this logical approach has originally been referred to as model checking (this term has been coined by Clarke and Emerson).

Model checking is an automated technique that, given a finite-state model of a system and a logical property, systematically checks whether this property holds for a given initial state in that model. Roughly speaking, model checking is performed as an exhaustive state space search that is guaranteed to terminate since the model is finite.

2.4.2 Benefits of model checking

i General approach with applications to hardware verification, software engineering, multi-agent systems, communication protocols, embedded systems, and so forth.

ii Supports *partial* verification; a design can be verified against a partial specification, by considering only a subset of all requirements. This can result in improved efficiency, since one can restrict the validation to checking only the most

relevant requirements while ignoring the checking of less important, though possibly computationally expensive requirements.

iii Case studies have shown that the incorporation of model checking in the design process does not delay this process more than using simulation and testing. For several case studies, the use of model checking has led to shorter development times. In addition, due to several advanced techniques, model checkers are able to deal with rather large state spaces (an example with 10^{130} states has been reported in the literature).

iv Model checkers can potentially be *routinely used* by system designers with as much ease as they use, for instance, compilers; this is basically due to the fact that model checking does not require a high degree of interaction with the user.

v Rapidly increasing *interest of the industry:* jobs are offered where skills in applying model checkers are required; industry is building their own model checkers (like Siemens and Lucent Technologies), have initiated their own research groups on model checking (like Intel and Cadence) or is using existing tools (such as Fujitsu and Dutch Railways).

vi Sound and interesting mathematical foundations: modelling, semantics, concurrency theory, logic and automata theory, data structures, graph algorithms, and so on constitute the basis of model checking.

2.4.3 Model Checking Temporal Logic

As mentioned earlier, embedded system are reactive systems characterized by a continuous interaction with the environment. For instance, an operating system or a coffee machine interacts with its environment (that is, the user) and usually performs some actions, such as fetching a file or producing coffee. Thus a reactive system reacts

to a stimulus from the environment by a reaction. After an interaction and accompanying reaction, a reactive system is once more able to interact. The continuous character of interaction in reactive systems differs from the traditional view on sequential systems. These sequential systems are considered as functions from inputs to outputs. After receiving input, the sequential system generates output(s) and terminates eventually after a finite number of computation steps. Due to the "transformation" of inputs into outputs these types of systems are sometimes also referred to as transformational systems. Properties of such transformational systems typically relate outputs generated to inputs provided in terms of pre- and post conditions as discussed previously.

Typical reactive systems are non-terminating. Due to their non-terminating behaviour, properties of reactive systems usually refer to the relative order of events in the system; that is, the behaviour of the system while over time. Lamport argued that the requirements that designers wish to impose on reactive systems fall basically into two categories. These categories are:

i *Safety* properties which state that "something bad never happens". A system satisfies such a property if it does not engage in the proscribed activity. For instance, a typical safety property for a coffee-machine is that the machine will never provide tea if the user requests is coffee and

ii *Liveness* properties which state that "something good will eventually happen"; to satisfy such a property, the system must engage in some desired activity. An example of liveness property for the coffee-machine is that it will provide coffee eventually after a sufficient payment by the user. Stated differently, the user will always be provided coffee at some time after inserting the payment.

Although informal, this classification has proven to be rather useful; the two classes of properties are (almost) disjoint, and most properties can be described as a combination of safety and liveness properties.

Logics have been defined in order to describe precisely these types of properties. The most widely studied are *temporal logics,* which were introduced for specification and verification purposes in computer science by Pnuelia. (The origins of linear temporal logic go back to the field of philosophy where Pnuelia invented it in the 1960s.) Temporal logics support the formulation of properties of system behaviour over time. Different interpretations of temporal logics exist depending on how one considers the system to change with time.

There are various kinds of temporal logic: Linear Temporal Logic (LTL), Computational Tree Logic (CTL), CTL*, μ-calculus are types of *branching* temporal logic etc. (Raimondi, 2010).

Linear temporal logic allows the statement of properties of execution sequences of a system. *Branching* temporal logic allows the user to write formulas which include some sensitivity to the choices available to a system during its execution. It allows the statement of properties of about possible execution sequences that start in a state.

2.4.4 Model Checking Branching Temporal Logic

As mentioned above, Pnuelia introduced linear temporal logic to the computer science community for the specification and verification of reactive systems. In Linear Temporal Logic like Propositional Linear Temporal Logic (PLTL) the temporal logic is called linear, because there is only one possible successor state and thus only one possible future. Technically speaking, this follows from the fact that the interpretation of linear temporal logic formulas using the satisfaction relation |= is defined in terms of

a model in which a state *s* has precisely one successor state *R(s)*. Thus for each state *s* the model thus generates a unique infinite sequence of states *s, R(s), R(R(s)),….* A sequence of states represents a computation. Since the semantics of linear temporal logic is based on such "sequence-generating" models, the temporal operators X, U, F and G in fact describe the ordering of events along a *single* time path, i.e. a single computation of a system.

In the early eighties another type of temporal logic for specification and verification purposes which is not based on a linear notion of time, but on a branching notion of time was introduced (Clarke and Emerson, 1981). This logic is formally based on models where at each moment there may be several different possible futures. Due to this branching notion, this class of temporal logic is called *branching temporal logic.* Technically speaking, branching boils down to the fact that a state can have different possible successor states, thus *R(s)* is a (non-empty) set of states, rather than a single state as for PLTL. The underlying notion of the semantics of a branching temporal logic is thus a *tree* of states rather than a sequence. Each path in the tree is intended to represent a single possible computation. The tree itself thus represents all possible computations. More precisely, the tree rooted at state *s* represents all possible infinite computations that start in *s.* The temporal operators in branching temporal logic allow the expression of properties of (some or all) computations of a system. For instance, the property EF □ denotes that there exists a computation along which F □ holds. That is, it states that there is at least one possible computation in which a state is eventually reached that fulfils □. This does, however, not exclude the fact that there can also be computations for which this property does not hold, for instance, computations for which □ is never fulfilled. The property AF □, for instance, differs from this existential property over computations in that

it requires that all computations satisfy the property F □. Computational Tree Logic is one of the types of branching temporal logic. Each CTL operator is a pair of symbols:
The first one is either A ("for All paths"), or E ("there Exists a path").
The second one is one of X ("neXt state"), F ("in a Future state"), G ("Globally in the future") or U ("Until").

2.5 Requirements for Embedded Systems Design

In (Gajski *et. al.*, 1994) a design methodology for embedded systems is proposed to consist of three major levels: behavioural, structural and physical. These levels can also be seen as a grouping of hardware abstraction levels, shown in Fig.2.5, and are required high level abstractions for the support of codesign.
Codesign is the simultaneous design of both hardware and software to implement in a desired function. Successful codesign goes hand in hand with co-verification, which is the simultaneous verification of both software and hardware and in what extent it fits into the desired function.

2.5.1 Behavioural Level

At the behavioural level it is advantageous to specify a system's functionality with an executable model. This model can be considered to be a system prototype, testing whether it meets all design requirements. An executable model is very useful at all stages of the design process as it is unambiguous and can be used as a reference for the system's functionality. A model may be verified very early in the design to test whether certain constraints are feasible and it is also easier to evaluate the impact of proposed changes. Furthermore it can be used as a starting point for product upgrades and maintenance. The requirements for a behavioural system model include the ability to

model a system at high levels of abstraction without imposing an implementation regime. The model should be easily understood, even for non technical people, enabling the system functionality to be communicated. In addition it should be formal, executable and amenable to analysis.

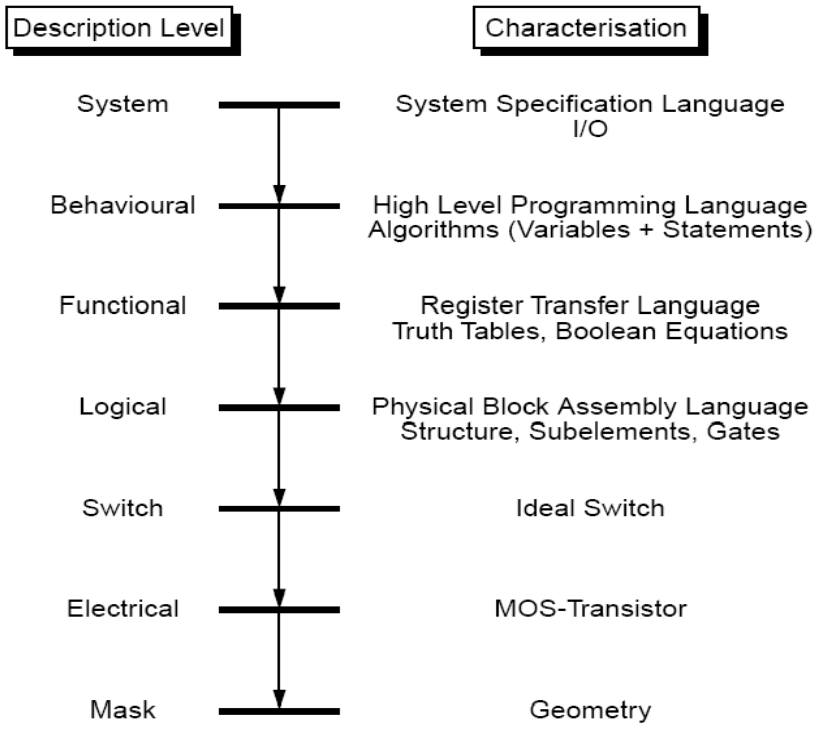


Fig. 2.5: The VLSI Design Hierarchy

2.5.2 Structural Level

At the structural level, components are defined that interact with one another to implement the required functionality. At this level, important decisions are made regarding the number, connectivity and function of system components in order to define the full functionality and constraints of the system as a set of communicating components. The design and test of algorithms may be considered as part of this level, which may also include the analysis of the effects of the use of restricted word lengths and general algorithm robustness.

2.5.3 Physical Level

The physical level is where the implementation decisions are made and implementation scenarios are evaluated. The models used are refined to include information details such as hardware or software implementation technology and number of physical implementation units. For example, a particular function may be implemented on a number of dedicated concurrently executing implementation elements or it may be implemented on a shared resource together with other functions. At this level scheduling is introduced to ensure the optimal use of resources while meeting constraints. This abstraction level relies on traditional methodologies and implementation techniques to provide information regarding the quality of a particular implementation option (Gajski *et. al.*, 1994). This may be in the form of a performance and area estimate for a particular function implemented in hardware or the size of code generated for software based component etc. This requires the use of many tools specific to particular implementation targets, such as compilers for microcontrollers, DSPs etc. and logic synthesis tools targeting programmable or custom hardware.

2.6 Additional Requirements for Embedded Systems Design

The various required level for a codesign methodology discussed above are very general. In this section a number of specific additional requirements are presented. These requirements are direct consequence of the heterogeneous nature of embedded real time systems where it is desirable to produce implementations that meet functional and non functional constraints.

2.6.1 Multi-Paradigm Support

The nature of many real time systems is such that no single formalism is suitable for modelling all aspects of them. An embedded system design methodology should allow designers the flexibility of choosing the most appropriate set of formalisms for the particular system to be modelled. Also many designers will prefer to apply familiar methods and notations from their working environment, application domain and experience. Allowing systems to be modelled with familiar formalisms reduces the risks, costs and difficulties of introducing new methods. Many existing formalisms also have well known implementation transformations and analysis algorithms and it is desirable that these continue to be applicable in a heterogeneous modelling environment.

2.6.2 Refinement

Refinement is the process of successively inserting more information into a model. A model as defined earlier is an abstraction of the real system omitting uninteresting details. The goal of a model is to provide the ability to test system characteristics that would be more difficult or even impossible on the real system. This is especially true if the real system does not even exist, in which case the model represents the system to be

designed and on which validation and verification can be carried out to determine whether constraints imposed by the environment can be met. In a typical design process models are initially abstract, requiring further refinement before they represent an implementation with the required level of detail. An important practical issue is the ability to simulate models consisting of components having different levels of abstraction. This enables large systems to be simulated where only the components of interest are modelled in detail. A model may be refined in a number of ways to evaluate design options. Eventually, as a result of the design exploration, a model will be found to better represents the system.

2.6.3 Time

Embedded systems are further characterised in terms of their interaction with their environment. An embedded system must not only be functionally correct but must also ensure that the appropriate responses are produced at the correct time. Real time requirements prescribe certain time relationships between particular inputs and outputs and models used for hardware / software codesign should support the formalisation of time constraints.

2.6.4 Configuration

Configuration is the process of generating and managing system variants. Fig.2.6 shows two variants of a system where the first is optimised for speed and the second for size and cost. System configurations handled within a single model are easier to maintain and keep consistent. The ability to easily configure systems must also take into account the heterogeneous nature of embedded systems where a function may be implemented on different targets.

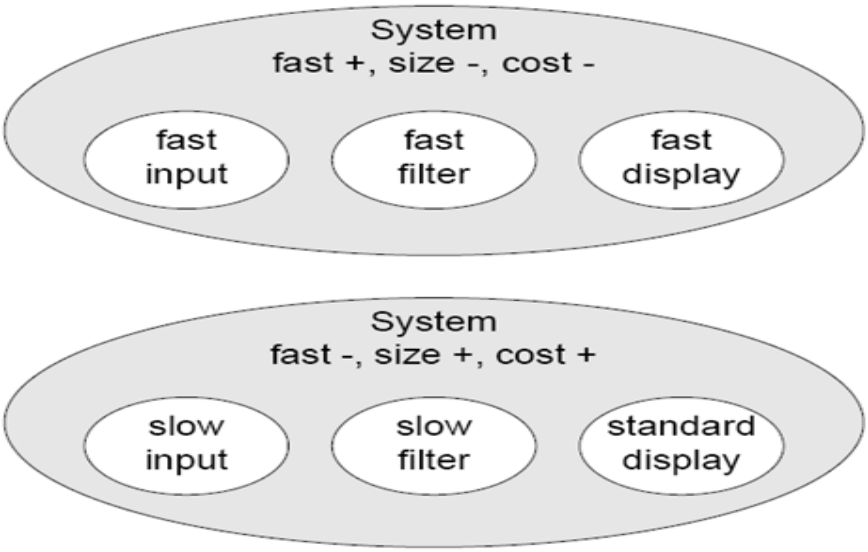


Fig. 2.6: Two Possible system configurations

Source: Martin, (2003).

2.7 Real-Time Embedded Systems Design Exploration

It is necessary to ensure that a system methodology allows an easy exploration of design alternatives once the system's functionality has been defined. It must be possible to allow designers to allocate architectural components and constraints and to cost the resulting system. The cost of a system not only consists of measurable criteria such as delay, power consumption etc. but also, for example, ease of maintenance and extensibility.

There is a need to look at some issues relating to more complex microcontroller-based systems. Numerous texts are available, written by experienced and knowledgeable engineers that discuss the finer points of real-time system design. In this section some relevant factors in the selection of the best combination of hardware, programming

language (software), and development tools for any given microcontroller product design is going to be reviewed.

2.7.1 Hardware Selection

There is a range of related devices around which embedded systems may be designed, these include the following:

a. Microcontroller Unit (MCU)

b. Microprocessor Unit (CPU)

c. System on a Chip (SoC)

The conventional microprocessor system embodies the traditional approach, where a central processing unit, memory, and peripherals can be put together to meet the requirements of a particular application as precisely as possible. Designing a custom-made CPU system is a relatively expensive option, and such an extensive range of other options are available that the conventional CPU-based system may be needed for only highly complex, specialist systems or where a low-cost, standard board such as the PC motherboard can be easily adapted.

The discrete microprocessor does, however, allow multiprocessor systems to be designed that typically use shared hardware resources, especially memory. Current standard processors typically incorporate features to support multiprocessor operation, and the dual core processor is currently becoming standard in PCs.

The SoC takes the concept of the microcontroller to the next level. It is, in effect, a configurable microcontroller, where the designer has control over the internal arrangement of the hardware elements. Using a dedicated design system, the processor core is selected and the required memory and peripherals added. These hardware elements are supported by corresponding standard drivers provided as part of the

package. With a complex interface, such as USB, for example, the provision of a standard protocol stack (software layers, not a hardware stack) is essential. The design can be fully tested in software, in the same way that a PIC (Peripheral Interface Controller) program can be tested in MPLAB (Microchip's Integrated Development Environment, IDE for Microcontroller). Only when finally verified is the design fabricated by the hardware supplier. If a design is to be created from scratch, then the most appropriate type of system may be selected from the three main options listed previously. However, this choice is unlikely to occur in isolation, factors such as the previous experience of the design team, existing company products, and so on are significant. Nevertheless, the designer should keep an open mind as far as possible and needs to keep up with a rapidly developing technology in the embedded systems field, to make the right choice may not be easy.

A designer who has a store of expertise using a particular microcontroller type and development system will need a good reason to look elsewhere for a solution. Gaining similar expertise in another system takes time and resources, and any change must also take into account the future strategy of the company or design group. The PIC family may be the first choice for the following reasons:

a. Low cost
b. Simplicity
c. Good documentation
d. An established market
e. A development system provided
f. Third party support

The PIC is well suited to the learning environment as it was originally pitched at the low-end (high-volume, low-complexity) market and is well supported by third party

products. The main alternatives are Atmel (AVR), Freescale (Motorola), STMicroelectronics, Hitachi, Philips, and National Semiconductor. We can approach hardware selection on the basis of the choice offered within the PIC range. Some of the main features to consider are:

a The number of I/O pins.

b The interface types.

c The program memory capacity.

d The RAM capacity.

e The operating speed.

f The power consumption.

We assume that adequate development system support and driver libraries are available. A logical approach to design is to select a chip that has spare capacity in relation to the draft specification. The application can be prototyped in simulation mode without penalty using an over specified device. When the I/O, memory, and peripheral requirements finally are established, a chip can be selected for hardware implementation that meets the specification at minimum cost. The anticipated scale of production is also a factor. The cost of each individual unit produced becomes more critical as the scale of production increases. On the other hand, the firmware can be reproduced at effectively no cost, unless variants are required. If we assume a fixed cost, *a*, for design development (hardware and software) and each board costs *b* to produce, the cost per unit is given by

$Y = a/x + b$

where *x* is the number of units produced. The fixed costs are divided by the number of boards produced (Martin, 2003). So, if the development costs are, say, 1000 units of currency ($a = 1000$) and the production cost 100 per board ($b = 100$), a curve showing

the cost per board as the volume of production is increased is obtained, as seen in Fig.2.7. We can see that the cost per board is initially high, falling away and levelling off as the production volume increases.

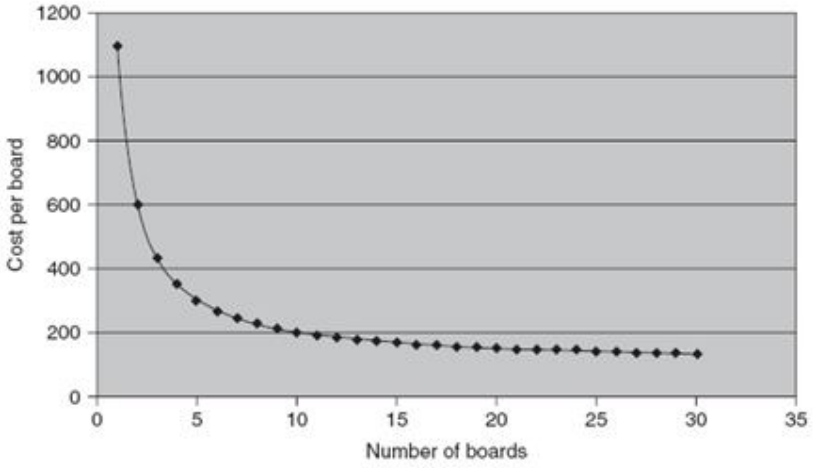


Fig. 2.7: Production Cost

Source: Martin, (2003).

2.7.2 Hardware Design Consideration

Taking the hardware design criteria in turn, we can consider some of the relevant factors in getting started with a design, assuming an agreement on initial specification. It is useful to know how much flexibility is allowed in meeting the specification, because a disproportionate cost might be involved. It may be acceptable to reduce the performance to reduce costs, for example, reducing the precision of analog measurements or the frequency range of a signal output. The cost of the microcontroller tends to increase with the number of I/O pins, so it is probably a good idea to look for ways to reduce the pin count. One example is to use a serial Liquid Crystal Display (LCD) instead of a parallel one. The serial type requires only one output, while the parallel LCD needs seven, or possibly eleven if 8-bit data are used. Certainly the serial interface should be

considered the default choice, and the parallel used only if high-speed access to the display is needed. The serial link can also be physically longer. Serial access sensors are becoming more common, where the data are sent to the MCU in serial form, rather than as an analog signal. Any pin can be used as an RS232 port, because CCS C (Custom Computer Services Compiler) provides a driver that generates the required interface purely in software. This means dedicated analog ports may not be necessary, giving greater flexibility in the choice of MCU. On the other hand, the sensor is likely to be more expensive.

Program memory capacity requirements are not easy to anticipate before the software has been finalized. C programs generally need more memory than assembler, so the choice of language is important. Also memory requirements expand rapidly with program complexity. As regards RAM requirements, the PIC is strictly limited, as the only onboard RAM consists of spare file registers. External data memory may well be necessary. An alternative type of MCU could even be necessary for data-intensive applications.

The PIC scores well on operating speed, however the 16 series devices can generally run at 20MHz, with the 18 and 24 series running at 40MHz. The clock speed does affect the power consumption, as the current consumed is proportional to the switching rate in CMOS **(**Complementary Metal-Oxide Semiconductor**)** devices. Low-power MCUs are an important ongoing development in microprocessor technology. Reduced operating voltages (for example, 3.3 V supply) are also increasingly used to reduce power dissipation. Power consumption is not one of the operating parameters normally predicted by simulation, so a real hardware prototype may be needed to finally specify the power supply. Obviously, power consumption is even more critical in battery-powered systems.

2.7.3 Software Architecture Design Consideration

There are two main options for creating the system firmware for low-complexity embedded systems, assembly language or C. A wider range of high-level languages and proprietary development systems are available to support more advanced processors. In general, assembly language is used for simple programs and those where direct access to control registers or speed is critical. Certainly, using assembler requires an intimate knowledge of the MCU architecture and it is an essential tool for the practicing embedded engineer. If necessary, assembly language blocks of code can be embedded within a C program. However, the main advantage is that C is by far the most widely used high-level language for embedded systems and can be applied by all embedded engineers, from beginner to expert. The overall structure of the embedded firmware is determined by the complexity and, to some extent, the hardware features of the host MCU. A simple program can use polled I/O in assembler program. If the chip has an interrupt structure that allows task priority and timing to be adequately managed, then interrupts can be used in assembler or C. The RTOS (Real-Time Operating System) approach may well be the best solution for more advanced applications as discussed in section 2.7.7. There is never a perfect solution to the embedded design challenge, but we can try for the best one that lies within our own limits of experience and enjoy the challenge it presents.

2.7.4 Embedded Systems Software

The microcontroller program comprises a list of machine code instructions, decoded and executed in sequence, resulting in data movement between registers, and arithmetic and logic operations. MCU reset starts execution at address zero, and the instructions are executed in address order until a program branch is decoded, at which point a new

target address is derived from the instruction. A decision is made to take the branch or continue in sequence based on the result of a bit condition test.

The program could be written in raw binary code, but this would require manual interpretation of the instruction set. Therefore, the machine code is generated from assembly code, where each instruction has a corresponding mnemonic form that is more easily recognizable, such as MOVF05,W (move the data at Port A to the working register). This low-level language is fine for relatively simple programs but becomes time consuming for more complex programs. In addition, assembly language is specific to a particular type of processor and, therefore, not "portable". Another level of abstraction is needed, requiring a high-level language C which has become the universal language for microcontrollers. It allows the MCU memory and peripherals to be controlled directly, while simplifying peripheral setup, calculations and other program functions. All computer languages need an agreed set of programming language rules. A processor-specific compiler converts the standard syntax into the machine code for a particular processor. The compiler package may also provide a set of function libraries, which implement the most commonly needed operations. There is variation between compilers in the library function syntax, but the general rules are the same. Usually, a choice of compilers is available for any given MCU family. Examples of compilers are Microchip C18 compiler, Hi-Tech PICC, and CCS C. The definitive C reference is The C Programming Language by Kernighan and Ritchie, second edition, incorporating ANSI C standards, published in 1983.

The primary function of the compiler is to take a source text file PROJNAME.C and convert it to machine code, PROJNAME.HEX. The hex file can then be downloaded to the PIC MCU. The source file must be written in the correct form, observing the conventions of both ANSI C and the specific compiler dialect.

Once the compiler has produced the hex file, it can be downloaded to the target application board.

2.7.5 Programming Embedded Systems

The purpose of an embedded program is to read in data or control inputs, process them and operate the outputs as required. Input from parallel, serial and analog ports are held in the file registers for temporary storage and processing and the results are output later on, as data or a signal. The program for processing the data usually contains repetitive loops and conditional branching, which depends on an input or calculated value.

The programmer is the kit used to download the program code to the target hardware, the one that is going to be used for the purpose of this work is Microchip PICkit2 programmer (Fig.2.8). This connects to the USB port of the host PC, with the programming module plugging direct into the target Printed Circuit Board (PCB). The six-way In-Circuit Serial Programming (ICSP) connector, between the programmer module and the target board, must be designed into the application circuit. An in-line row of pins provides the programmer connection to the target MCU, as shown in Fig. 2.9.

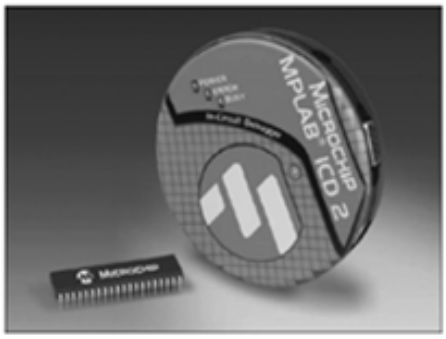


Fig. 2.8: Microchip ICD2 Module

Source: (Martin, 2003).

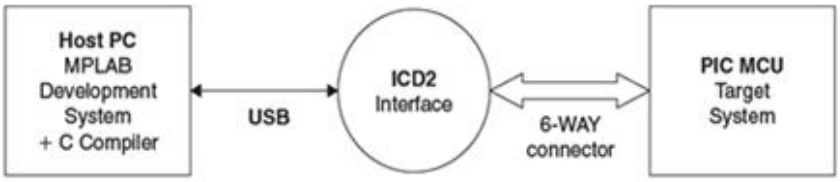


Fig. 2.9 : ICD2 Program and Debug System

Source: Martin, (2003).

The default programming language of any microprocessor or microcontroller is its own assembly language. The syntax used for any given processor is determined by its internal architecture and the machine code instructions that control it. Assembly language is the first-level abstraction from machine code, where each instruction is represented by a corresponding text mnemonic.

When compiled, a C program is converted into assembler, then to machine code. Since each C statement is independently translated into machine code, there is often unnecessary duplication of instructions. For example, each time a port is accessed, the required initialization is repeated. As a result, the assembler program derived from C source code is always considerably longer than an equivalent assembler program that performs these functions.

For this reason, many compilers contain optimization routines that try to minimize this problem by analyzing the compiler code and eliminating redundant operations. For example, when an I/O operation is converted, the compiler can check to see if the port is already correctly initialized, if so, repetition of the initialization can be eliminated from the code.

2.7.6 Timers

The PIC 16F877A has three hardware timers built in: Timer0 (originally called RTCC, the Real-Time Counter Clock), Timer1, and Timer2. The principal mode of operation of these registers is as counters for external events or timers using the internal clock. Additional registers are used to provide Capture, Compare, and Pulse Width Modulation (PWM) modes. The CCS timer function set is shown in Table 2.1.

Table 2.1: Timer Functions

Action	Description
TIMERX SETUP	Set up the timer mode
TIMERX READ	Read a timer register (8 or 16 bits)
TIMERX WRITE	Preload a timer register (8 or 16 bits)
TIMERX SETUP	Select PWM, capture, or compare mode
PWMX DUTY	Set PWM duty cycle

A counter/timer register consists of a set of bistable stages (flip-flops) connected in cascade (8, 16, or 32 bits). When used as a counter, a pulse train fed to its Least Significant Bit (LSB) causes the output of that stage to toggle at half the input frequency. This is fed to the next significant bit, which toggles at half that rate, and so on. An 8-bit counter thus counts up from 0x00 to 0x FF (255) before rolling over to 0 again (overflow). The binary count records the number of clock pulses input at the LSB.

2.7.7 Embedded Systems Operating Systems

As microcontroller operating programs become more complex, consideration must be given to the best method of organizing the program response to input, memory management, and output timing. Three main methods are used to handle input and

output events, which after all, is the primary requirement of a real-time system. In order of complexity, they are I/O polling, interrupts, and the Real-Time Operating System (RTOS).

(i) Polled I/O

This is the easiest, and may be considered the default, method of input and output, where operations are simply scheduled as part of the main loop. The basic principle is illustrated in Fig. 2.10.

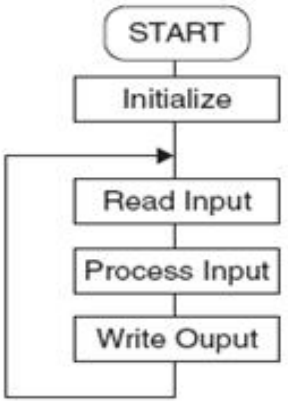


Fig 2.10: Polled I/O Process

This option is fine if the delay that occurs between input signal and output response is not critical to the correct overall operation of the system. The time taken to complete the input processing may vary significantly, depending on the input data or programmed options within the loop. For example, a test on the data may result in an optional sequence being executed, or not, depending on the value. In fact, this is pretty much inevitable in most real programs. However, it is often important for the input and output timing to be more predictable. Take the example of motor speed control, in small DC motors, this is usually implemented by pulse width modulation. The output is switched on and off over a regular cycle, the proportion of " on " to " off " time determining the

average motor current and hence the speed. To achieve accurate control, the shaft speed must be measured, usually by a pulse encoder. The input pulse interval must be measured and the PWM duty cycle adjusted accordingly. It is possible to achieve this using a polling process, but a more elegant solution can be implemented using interrupts.

(ii) Interrupts

Interrupts are internally or externally generated asynchronous hardware signals that force the processor to stop its current (background) task and carry out the Interrupt Service Routine (ISR), a higher-priority (foreground) task. The processor "context" (current register contents and status) must be saved and the current program address stored on the stack so that the background task can be resumed when the ISR has finished. For example, this can be applied in the motor controller, assuming we are using a 16F877A MCU (Fig.2.11). The input pulse period can be measured using one of the hardware timers. Since Timer2 is designed to provide PWM mode, Timer1 can be used to monitor the input, working in Capture mode. The counter/timer register is fed from the system clock to measure absolute time intervals, and the count is stored when the input changes. The pulse period can then be worked out and this result compared with a target value, which represents the required period (hence speed). If it is too long (speed too low), the motor speed is increased by increasing the PWM duty cycle in Timer2. If too short (speed too high), the duty cycle is reduced. An interrupt is generated by Timer1 when the count is captured; the ISR modifies the output duty cycle as required, and the controller then waits for the next interrupt to occur.

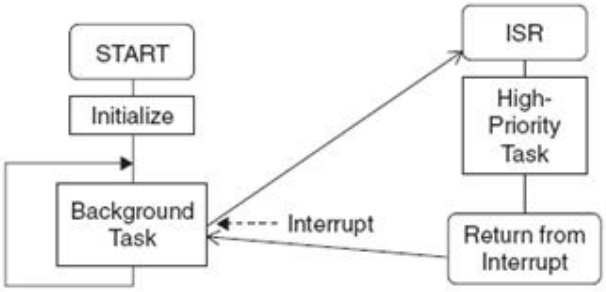


Fig. 2.11: Basic Interrupt Operation

If the program uses multiple interrupts, one ISR may be interrupted by another. The interrupts may need to be assigned an order of priority, so that a less important task does not interrupt a more important one. When the higher-priority ISR is being executed, the lower-priority interrupt can be disabled, or masked, until it is finished. In more complex programs, numerical levels of priority can be assigned, with higher priorities taking precedence. Further, the different interrupt sources have to be identified explicitly by a user routine. An operating system (OS) provides an alternative to interrupts as a means of providing a more predictable time response in the microcontroller system.

(iii) Real-Time Operating System

Nowadays, the most well known example of an operating system (OS) on most Personal Computer is Microsoft Windows. Why is this needed in PC-type computers? The answer is simply the complexity of the software compared with a microcontroller. The operating system provides a collection of the numerous program components required to run the computer. Each peripheral interface has its own driver (keyboard, screen, disks, mouse, network, etc.) plus modules for memory management and general system control. Therefore, the PC needs a more sophisticated task management system. A lengthy process, such as printing or disk access, cannot be allowed exclusive use of the system resources. If the processor ignores the keyboard completely while downloading

a large file from the internet, the user cannot access the system to do something more urgent. In addition, the OS has to be multitasking; that is, it must allow several operations to appear to be running simultaneously, such as allowing you to keep writing while printing. We also want to switch quickly between tasks by keeping more than one window open at a time, which means keeping multiple tasks loaded in memory. Multitasking is essentially achieved by time slicing. Each apparently concurrent task is allowed to run for a given time interval, say 100ms, then execution switches to another. Priority can be assigned, so that, for example, one Internet data packet is picked up and stored in memory before the next arrives and overwrites it in the network data buffer. Therefore, the OS is designed so that multiple tasks appear to run smoothly together and with the right priorities (www.pumpkininc.com, 2010).

The PC is essentially a batch processing system; that is, the timing of the major tasks is not critical. If a word-processing task is delayed for a few milliseconds, it is not apparent to the user and not significant in terms of overall system effectiveness.

On the other hand, the timing of events in so-called real-time systems must generally be highly predictable. When an input is received, it must be processed and the output generated within a known time frame. The point is obvious if one considers an example such as an aircraft flight control system or automobile engine controller. To manage complex control system software, we may need a real-time operating system. The principle of operation of a simple RTOS, as is going to be implemented in this work is shown in Fig.2.12. The program is divided into separate tasks, which are executed in turn. A timer interrupt causes the task switching, but interrupts are otherwise not used. When a task is suspended, its context (file register state) is saved and restored when it is restarted the next time around. In this way, multiple tasks are executed in rotation and can appear to execute simultaneously, and the I/O timing is more predictable. More

sophisticated systems incorporate task priority and implement more complex task management strategies.

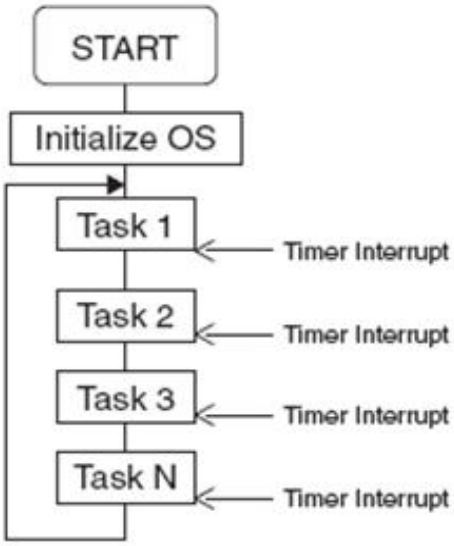


Fig. 2.12: Basic RTOS Operation

Tasks are executed in turn, with the frequency and duration specified for each. The CCS implementation is classified as a cooperative, multitasking RTOS. This means that the tasks return control to the scheduler voluntarily to allow the next to run. A set of functions are supplied that allow the tasks to work together for optimum effect. For example, rtos_enable (task1) and rtos_disable (task1) allow tasks to be selectively enabled and disabled. The function rtos_yield() allows the task to return control to the scheduler when finished. Some functions allow status information and messages to pass between tasks and the progress of the tasks to be monitored (CCS C, 1994).

2.8 Real-time Embedded Systems Design

Real-time Embedded Systems is heterogeneous in nature and the design is mostly application specific with emphasis on efficient implementation. Various design methodologies to accomplish these are discussed in the following section.

A design methodology defines the process of designing a system, typically from conception to implementation. It is usually derived from heuristics obtained over many designs and therefore provides a set of well tested design steps that, when followed, ensures that critical analysis and design aspects are not neglected. A design methodology may require the use of different models at various stages to highlight particular system characteristics that need to be addressed. A challenge to defining design methodologies is the requirement that it must be free from semantic discontinuities which force the designer to manually remodel the system in yet another formalism. Design methodologies are necessarily application specific, for example real time behaviour is critical for embedded systems design but may not even be addressed in methodologies for designing commercial software applications. However methodologies for different application areas often have many design steps in common.

Below are various system design methodologies. In the first part some heterogeneous system design methodologies are presented. This is followed by two representatives of traditional software oriented methodologies. Finally two object oriented software methodologies are presented, one of which is tailored for embedded systems design.

2.8.1 Heterogeneous System Design Methodologies

Design methodologies for heterogeneous system design are mostly based on methodologies for Application Specific Integrated Circuits (ASIC) design, with emphasis on efficient implementation. At the system level such methodologies are not well defined, often simply assuming that a formal system specification exists. This is illustrated in Fig.2.13 where a typical codesign methodology is presented which emphasizes heterogeneous implementation generation. Two examples implementing this design methodology are ’COSYMA’ and ’Olympus’.

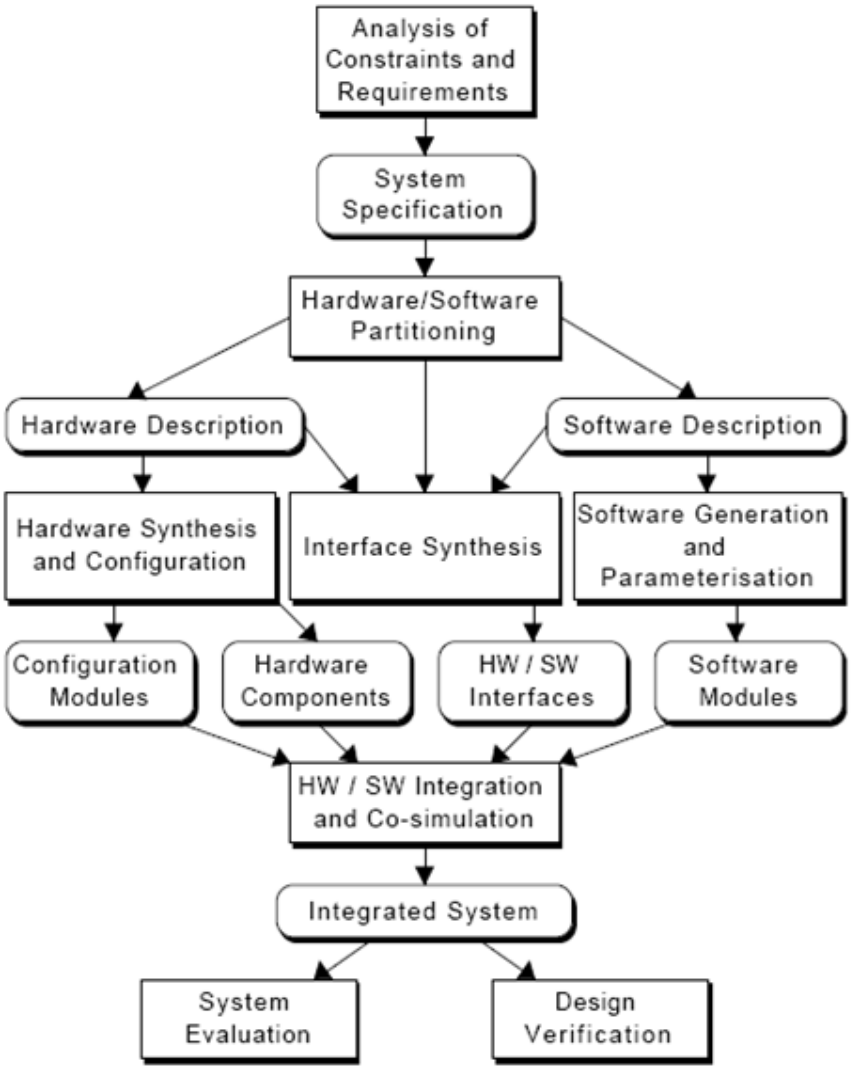


Fig. 2.13: The traditional hardware/software codesign methodology

Source: Buchenrieder and Rozenblit, (1995)

The 'Ptolemy' and 'MOOSE' environments represent a new approach to hardware / software codesign. In this approach greater emphasis is placed on system specification and modelling. This approach attempts to define the system in a formal way very early in the design process where it can be validated and perhaps even verified. Fig.2.14 taken

from (Buchenrieder and Rozenblit, 1995) shows a model based methodology where system modelling and refinement are emphasised.

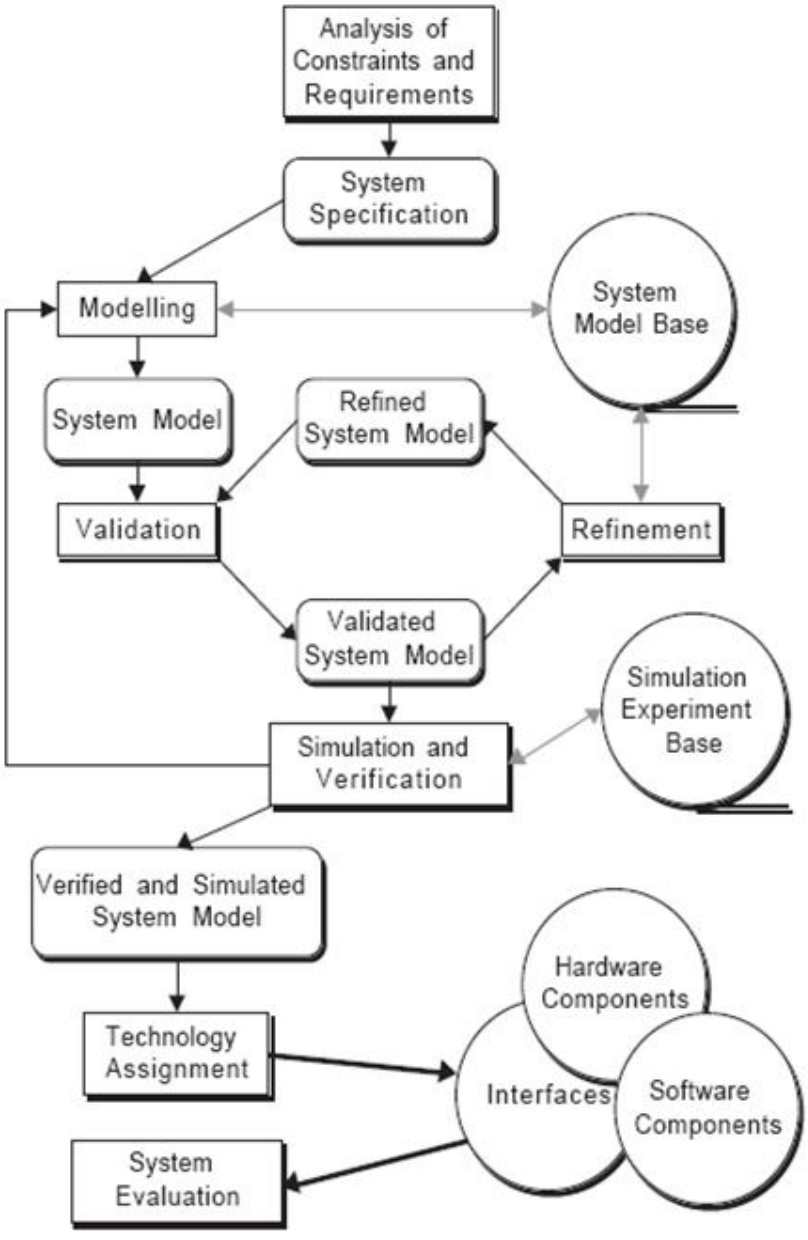


Fig. 2.14: A model based codesign methodology

Source: Buchenrieder and Rozenblit, (1995)

2.8.1.1 COSYMA

The approach taken in the COSYMA (Benner *et. al.*, 1995) project assumes that a specification exists written in the C^x language. A C^x description is translated into an internal extended syntax graph representation which can be simulated and profiled. The environment supports automatic partitioning based on a simulated annealing algorithm, where it is initially assumed that the complete system is implemented in software. The partitioning process migrates system blocks to hardware, attempting to find an implementation that meets the desired timing constraints. After partitioning, the hardware parts are translated into the Hardware-C language and implemented via the Olympus high level synthesis system.

2.8.1.2 Olympus

In the Olympus system, systems are specified with the Hardware-C language. The Hardware-C model is translated into an internal representation which can then be partitioned into hardware and software. In contrast to COSYMA where initially a software solution is investigated and functionality moved to hardware, in Olympus, practically the whole system is assumed to exist as hardware with only functionality characterised by non deterministic delays (data dependent loops and synchronisation primitives) residing in software. An iterative procedure moves processes between partitions with the goal of reducing communication while satisfying timing and other constraints (De Micheli *et. al.*, 1995).

2.8.2 Ptolemy

The Ptolemy design environment is a framework that allows hardware and software components to be integrated from specification through the synthesis, simulation and

evaluation phases (Kalavade, *et. al.*,1993). In Ptolemy objects described at different levels of abstraction and using different semantic models can communicate with each other using a hierarchy based mechanism where:

a an abstraction level with a specific semantic model is called a *domain*, such as data flow or discrete event;

b atomic objects called *stars* are primitives of the domain, i.e. data flow operators, logic gates etc. These objects are used in simulation mode, reacting to or producing events and in synthesis mode, producing target specific code

c *galaxies* are collections of stars and other galaxies. Instantiated galaxies may possibly belong to different domains than that of the encompassing domain. Each domain includes a scheduler which decides in which order the stars are to be executed. Whenever a galaxy instantiates another galaxy belonging to a different domain, a wormhole mechanism is created facilitating the communication between the two schedulers with appropriate data type conversion.

2.8.3 Model-based Object Oriented Systems Engineering (MOOSE)

MOOSE (Model-based Object Oriented Systems Engineering) is a graphical / textual method in which a system is described and investigated using abstract, executable and implementation neutral *(uncommitted)* models. Once the behaviour of the uncommitted model is satisfactory, its components are gradually *committed* to hardware or software on the basis of *systems engineering* decisions. These are based on non functional requirements and on the results of executing and analysing the uncommitted model (Green *et. al.*, 1994).

In MOOSE, systems are specified as asynchronous communicating processes. Synchronous parts of a model are described using class implementation diagrams which are similar to data flow diagrams that have operations as processes and state information as data stores. Communicating processes are specified as a hierarchy of Object Interaction Diagrams (OIDs). The top level OID is progressively decomposed to the primitive level, where any further decomposition would result in technology specific decisions (depending on the hardware and software implementation). MOOSE distinguishes between two kinds of communication, continuous data (non destructive read) and events (destructive read). MOOSE models are executable and object oriented facilitating reuse.

2.9 Performance Indexes

According to (Lothar and Ernesto, 2009), any methodology for performance analysis of distributed embedded systems must satisfy correctness, accuracy, short analysis time, embedding into the design process, reusability and modularity.

Correctness

A correct method is required for performance analysis of real time distributed embedded systems.

Accuracy

The calculated upper (lower) performance bounds should be close to the actual worst-case (best-case) performance of the system.

Short Analysis Time

The analysis of a distributed embedded system should be cheap in terms of analysis time and computational effort. This is particularly important if the performance analysis is used to drive a design space exploration. In this case the performance analysis is

executed repeatedly in order to evaluate several different system implementations and a short analysis time for a single evaluation is essential.

Embedding into the Design Process

A designer should be able to set up a system model for performance analysis with little effort. Also, an existing model should be easy to reconfigure.

Reusability

An ideal performance analysis method should be reusable across different abstraction levels. This means that it should permit designers to model and analyze a system with different levels of detail. In particular, it should be easy to refine an existing system model.

Modularity

Modular performance analysis methods are typically more scalable and easier to reconfigure than holistic approaches, because they permit designers to model and analyze a system by composing several smaller, ideally pre-built system modules. However, often the modularity is paid for by reduced analysis accuracy. Modular Performance Analysis (MPA) involves two basic concepts; the service requested by an application (task) and the service offered by the architecture to this application (task). Temporal interactions between the requested and the offered service determine performance characteristics of the system. The ultimate goal of any performance analysis method is therefore to properly capture these interactions. The method can be a simulator, a mathematical model or any other methods. The main desire is that, it should return sufficiently accurate estimates of the system performance. In order to allow for a fast design space exploration, the performance model should also be efficiently analyzable and easily constructible. The latter property is especially important for supporting automated design space exploration.

Designing embedded systems that must satisfy real-time constraints faces additional challenges associated with the need to verify timing correctness of their behaviour.

Input specification

A starting point for the system-level performance analysis is a specification which typically describes the following aspects of an embedded system:

Application task structure

The application task structure is typically modelled by a *task graph* (or a set of task graphs) that captures a partitioning of the target application into individual tasks, and models data and control dependencies between them. Interactions between the tasks in a task graph may be governed by a specific model of computation. For example, multimedia applications are often modelled using the formalism of *Kahn Process Networks* (Rutten *et. al.*, 2002), which assumes that the tasks communicate via FIFO channels.

Task assignment to processing elements

The application tasks performing data transformations are assigned for execution to *computational resources* such as CPUs, Digital Signal Processor (DSPs) and co-processors, while the tasks responsible for data transfers are assigned to *communication resources* such as buses, Direct Memory Access (DMA) controllers, bridges, etc. Both resource types are referred to as P*rocessing Elements* (PEs) because in principle, for the performance analysis, it is irrelevant whether an architectural resource executes computation or communication tasks.

Resource management policies

As a result of the task assignment, multiple tasks may be mapped on to one PE. In this case, a scheduling (or arbitration) policy is deployed to manage tasks access to this PE.

In general, several different scheduling and arbitration policies may be deployed within the architecture.

Storage resource allocation

Data arrays manipulated by the tasks, for example, the buffers implementing the FIFO communication channels, are assigned to the off- and on-chip memories.

Characteristics of processing elements

For the performance analysis we need to specify capabilities of processing elements. Therefore, such parameters as clock rates of processors and effective communication bandwidths of buses typically form a part of the input specification.

Task properties

These include a variety of relevant to the performance analysis task characteristics, for example, the number of processor cycles needed to complete a task on a given PE and the size of data items to be exchanged between the tasks.

Characteristics of the environment

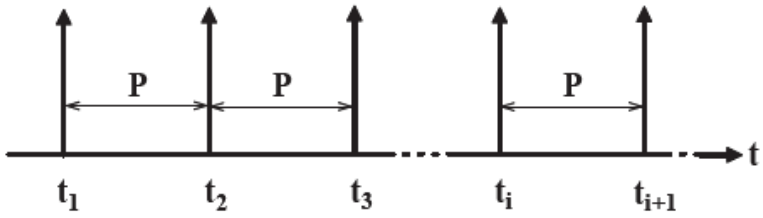
The input specification should also capture characteristics of the event streams to be processed by the embedded system. These characteristics may include timing properties of the event streams (e.g. their arrival rates) as well as their possible contents, for example, different event types that may appear in a given event flow.

Types of Event

The behaviour of the environment of a distributed embedded system is often modelled using common abstractions for event arrival patterns. These abstractions include periodic or sporadic event arrivals with potential jitters or bursts.

Periodic event stream

In a periodic event stream with period P the events arrive at intervals of exactly P time units. Fig. 2.15 depicts a periodic arrival pattern.



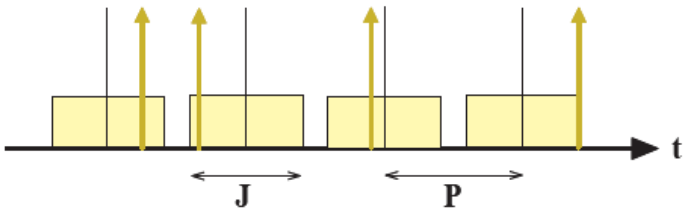
$$t_{i+1} - t_i = P$$

Fig. 2.15: A periodic arrival pattern.

Source: Simon, (2006).

Periodic event stream with jitter

In the periodic event stream model with jitter the events arrive at an average time interval of P time units, but may have a local deviation around the ideal periodic arrival. The deviation is bounded by an interval of length $J \leq P$. This is represented in Fig 2.16, where the intervals of admissible arrival times are represented as shaded rectangles.



$$J \leq P$$
$$t_i = i \cdot P + \varphi_i$$
$$0 \leq \varphi_i \leq J$$

Fig. 2.16: Periodic event stream with jitter.

Source: Simon, (2006).

In each jitter interval (shaded rectangle) exactly one event arrives.

Periodic event stream with burst

Events may arrive in bursts, if the deviation from the ideal periodic arrival time is larger than the period. In this case the admissible arrival time intervals of adjacent periods overlap, as depicted in Fig. 2.17. However, events cannot overtake each other: the

arrival time of an event is restricted by the arrival time of previous events. In particular an event may arrive only d time units (d = minimum event inter-arrival time, $d > 0$) after the arrival of the event belonging to the previous period.

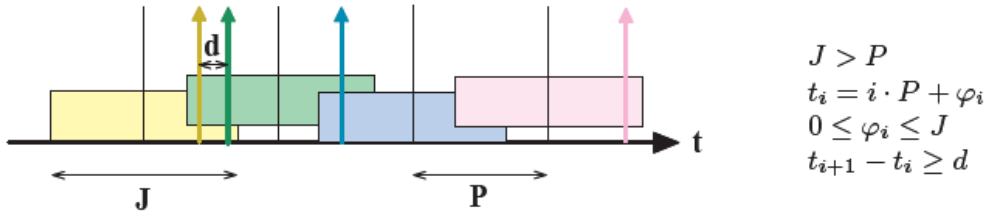


Fig. 2.17: Periodic event stream with burst

Source: Simon, (2006).

2.10 Review of Performance Analysis Methods

There are many approaches to performance analysis, notable among them are simulation and formal analysis methods, real-time calculus, holistic scheduling analysis and compositional method (Nikolay and Lothar, 2007). There are also Timed Automata based performance analysis and stochastic analysis method.

2.10.1 Simulation and Formal Analysis Methods

Simulation is defined as the imitation of some real thing, state of affairs, or process. The act of simulating something generally entails representing certain key characteristics or behaviours of a selected physical or abstract system. Simulation is used in many contexts, including the modelling of natural systems or human systems in order to gain insight into their functioning. Other contexts include simulation of technology for performance optimization, safety engineering, testing, training and education. Simulation can be used to show the eventual real effects of alternative conditions and courses of action. Simulation provides a tool to estimate the performance of a system

model, based on the simulation, monitoring of system properties as data throughput or resource usage can help to evaluate different designs (Alexander *et. al.,* 2009). The trend for simulation based performance analysis goes to full system simulation. Although simulation is accurate with completely specified models, it is not guaranteed to cover any corner case. Simulation based methods do not reveal worst-case bounds on essential system properties like end-to-end delay of events, throughput, and memory requirement. In addition, they often suffer from long running times which depend on the accuracy that is aimed for and from high set-up effort for each new architecture and mapping to be analyzed.

The use of SystemC, the Open SystemC Initiative (OSCI, 2010), a widespread platform for system level modelling and simulation, is very common. Many simulation methods are trace-based, that is, the system designer provides traces of input stimuli that drive the simulation of the modelled system. SystemC is a system description language that can be used to model the behaviour level of systems (Grotker *et. al.*, 2002). It consists of a set of library routines and macros implemented in C++, which makes it possible to simulate hardware building blocks and concurrent processes written in standard C++. SystemC supports the communication of C++ objects in a simulated real-time environment. SystemC is both a description language and a simulation kernel. The code written will compile together with the library's simulation kernel to give an executable that behaves like the described model when it is run. It supports the simulation of a system at various levels of abstraction from cycle-accurate up to the behavioural level. The main advantage of simulation is the large modelling scope. In contrast to formal analysis methods, basically every system can be modelled, as many dynamic and complex interactions can be taken into account for the simulation. In most of the cases the functioning and performance of a system can be verified with the same simulation

environment, simulation traces and benchmarks. Another advantage of simulation based methods is that they are usually reusable over different abstraction levels, as the simulation models can be refined. In other words, the level of abstraction for the simulation can be adapted to the required degree of accuracy. However, hardware-software co-simulations are often computationally complex and have long running times. Therefore, performance estimation quickly becomes a bottleneck in the design process, especially if it is used to drive a design space exploration.

For example, (Henia *et. al.*, 2005) shows that in order to find simulation patterns that lead to corner cases is an exiting challenge for apparently trivial distributed embedded systems. Fig 2.18 depicts the system architecture. In application A1, a task P1 of the CPU reads periodically data bursts from the sensor and stores the data in the memory. A second task P2 reads the data from the memory, processes it and transfers it to an output device via the shared bus. The task P2 has a Best Case Execution Time (BCET) and a Worst Case Execution Time (WCET). Suppose that the CPU implements static fixed priority scheduling and that P1 has higher priority than P2. In the second application A2, a task P4 running on the input interface periodically sends data packets to the Digital Signal Processor (DSP) over the shared bus. Task P5 on the DSP stores the data packets into the buffer. A second task P6 periodically removes data packets from the buffer, e.g. for playback. Suppose that the bus uses a first come first serve scheme for arbitration. As the two data streams of A1 and A2 interfere on the shared bus, there will be a jitter in the packet stream received by the DSP that may lead to an underflow or overflow of the buffer.

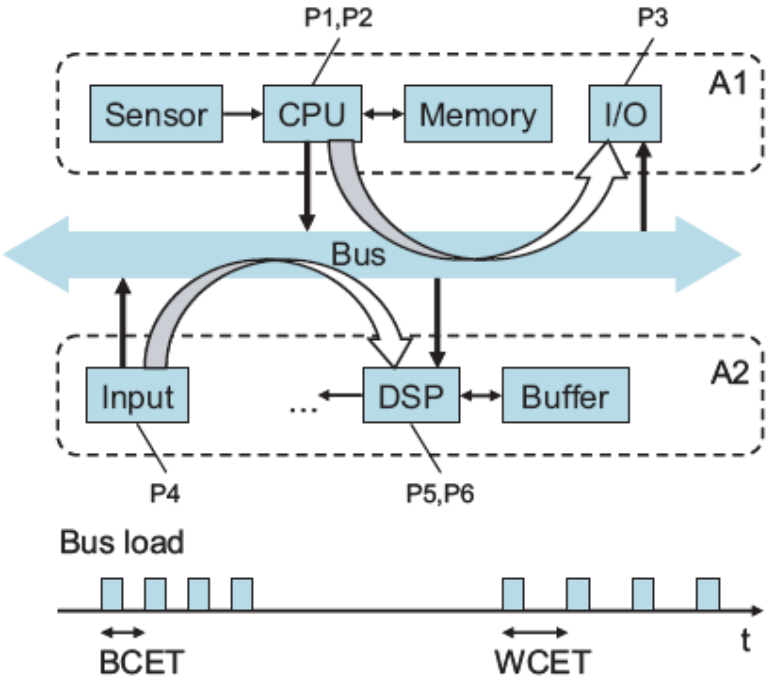


Fig. 2.18: Interference of two data streams on a shared communication resource

Source: Simon, (2006).

The interesting property of this system is that the DSP experiences the worst case input jitter when P2 executes continuously with its BCET. The reason is that in this case the distance between the packets of A1 on the bus is shortest and thus the transient bus load is highest. In other words, the worst case execution of A2 coincides with the best case execution of A1. The designer must perceive this system particularity in order to provide a simulation trace that reaches the corner case. In case of larger and more realistic systems, several computation and communication resources will be shared simultaneously, there may be different scheduling policies for the various resources and data/control dependencies will play a role.

It can be deduced that in Henia *et. al.*(2005) work, the corner cases will be extremely difficult to find, hence it cannot satisfied all (Lothar and Ernesto, 2009) criteria for

performance analysis. Therefore, simulation based methods are not suited to determining hard performance bounds of a general distributed embedded system. But nevertheless, simulative approaches can be useful to estimate the average system performance.

Formal methods for performance evaluation are emerging which enable the analysis of the whole systems using holistic and compositional approaches. In particular, the system can be analyzed using models of the individual components that can be later composed to capture the complete system. Formal analysis based methods lack the ability to incorporate complex interactions and state-dependent behaviour. Analysis results are therefore often pessimistic, but this pessimism does not threaten their correctness.

In a related work of (Agarwal 92), Analytical performance models for Digital Signal Processor (DSP) systems and embedded processors, the computation, communication, and memory resources of a processor are all described using simple algebraic equations that do not take into account the dynamics of the applications such as variations in resource loads and shared resources. These methods are therefore lacking in accuracy and the analysis results typically show large deviations from the properties of the final system implementation.

Actually, there is no sharp division into simulation-based approaches and formal methods for system-level analysis (Simon *et. al.*, 2009). In the area of embedded real-time systems design, one of the major differentiation criteria between these two classes of methods is the quality of results that are obtained with the respective methods.

To overcome the limitations of these two methods, (Lahiri *et. al*., 2001) combined simulation and analysis method by a hybrid trace-based simulation methodology which though shorten simulation run-times, but the problem of insufficient corner case

coverage still remains. To overcome this problem (Benini *et. al.*, 2003) proposed a method which combined SystemC simulation and formal analysis method based on Real-Time Calculus (RTC). Their work guarantee to deliver worst-case (and best-case) results for various system properties and also exhibit fast analysis run-times. Since an implementation of embedded real-time systems must meet a number of performance requirements related for example to end-to-end delays, buffer requirements, or throughput. When these quantities are measured on the final system implementation, the major variations is normally observed over time, as for example end-to-end delays may vary largely due to different input data or interference between concurrent system activities. However, there typically exists a worst-case and a best-case result for every quantity, such that we know for example that every observed end-to-end delay is larger or equal the best-case delay d_{BC} and smaller or equal the worst-case delay d_{WC} (Wandeler, 2006).

2.10.2 Real-Time Calculus

Real-Time Calculus (RTC), a system-level performance analysis method for embedded systems is a tool to characterize workload and processing capabilities respectively (Simon *et. al.*, 2009). Real-time calculus represents the resources and their processing or communication capabilities in a compatible manner and therefore, allows for a modular hierarchical scheduling and arbitration for distributed embedded systems (Nikolay and Lothar, 2007). In addition, the Real-Time Calculus allows computing various performance indexes of the system, such as upper bounds on the delay and backlog experienced by the events while being processed in the system (Wandeler, 2006).

The Real-Time Calculus provides powerful abstractions of the event and resource streams and uses these abstractions to mathematically model the behaviour of an elementary performance component. This basic model can then be used for a component-wise evaluation of a whole scheduling network.

Fig 2.19 depicts Real Time Calculus abstract processing component that models the processing of an event stream by an application process. In particular, an incoming event stream represented as a pair of arrival curves α^l and α^u, flows into a FIFO buffer in front of the processing component. The component is triggered by these events and will process them in a greedy manner while being restricted by the availability of resources, which are represented by a pair of service curves β^l and β^u. On its output, the component generates an outgoing stream of processed events, represented by a pair of arrival curves $\alpha^{l'}$ and α^u. Resources left over by the component are made available again on the resource output and are represented by a pair of service curves $\beta^{l'}$ and $\beta^{u'}$.

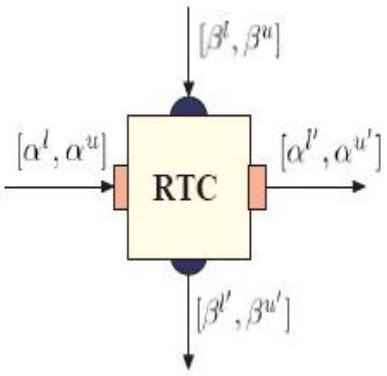


Fig. 2.19: Real Time Calculus processing component

Source: Wandeler, (2006).

The processing components can be freely combined to form performance models of distributed embedded systems. For instance in order to model the sequential processing of an event stream by two tasks, it is sufficient to connect two processing components in series so that the outgoing event stream of the first one is the ingoing event stream of the second one.

Characterization of Event and Resource streams

Timing properties of event and resource streams are captured using arrival and service curves.

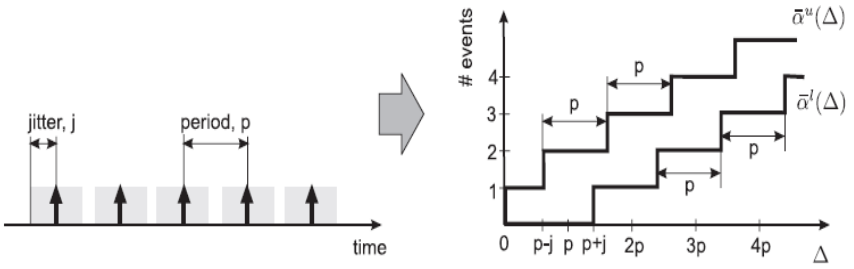


Fig. 2.20: Modelling periodic event streams with jitter using arrival curves

Source: Wandeler, (2006).

An event stream is abstracted by a pair of arrival curves, $\alpha^{-u}(\Delta)$ and $\alpha^{-l}(\Delta)$, which give respectively upper and lower bounds on the number of events seen in the event stream within any time interval of length Δ.

A resource stream is modelled by a pair of service curves, $\beta^{-u}(\Delta)$ and $\beta^{-l}(\Delta)$, which give respectively upper and lower bounds on the resource amount (e.g. number of processor cycles) offered within any time interval of length Δ.

The arrival and service curves can accurately describe streams with arbitrary complex timing behaviour. On the other hand, a single pair of upper and lower curves can capture an entire class of streams with similar timing properties. For example, many standard event models (e.g. sporadic or periodic, periodic with jitter, periodic with bursts as discussed above) can be represented by the arrival curves (Chakraborty *et. al.*, 2003). Fig.2.20 illustrates this fact by showing how the arrival curves model a class of periodic event streams with jitter.

Evaluation of Event and Resource streams

The evaluation can be accomplished component wise, by propagating the event and resource streams through the network. Doing this requires a model describing how the timing properties of the event and resource streams get changed as a result of passing through the performance components.

The model assumes that the events belonging to the same stream are processed in their arrival order and that they are stored in a FIFO buffer while waiting to be served. (Wandeler, 2006) introduced a compact representation for a special class of variability characterization curves, together with other methods to efficiently compute various Real-Time Calculus curve operations on these compact variability characterization curves, in order to efficiently conduct system level performance analysis and interface-based design within the Modular Performance Analysis (MPA) framework.

The Real-Time Calculus (RTC) Toolbox is now a toolbox for MATLAB that enables MPA framework based performance analysis and interface based design of embedded real-time systems within MATLAB.

In a related work, (Thiele *et. al.*, 2001) proposed framework for analysis of system properties which has a concept of lower and upper arrival as well as service curves that capture the best- and worst-case behaviour of the workload. In addition, they enhanced the analytical framework of (Naedele *et. al.*, 1999) with mechanisms to determine properties of the output event streams.

These developments pave the way to a modular approach to the performance analysis (Thiele *et. al.*, 2005). The limitation of Thiele *et. al.* is that it could model only tasks that consume and produce only one event per activation. (Maxiaguine *et. al.*, 2004) addressed this limitation, for computing the output event streams it becomes necessary to convert the arrival curves expressed in terms of event-based units into equivalents expressed in resource-based units and backwards. Since this conversion is performed by scaling the curves with a constant factor corresponding to W*orst-Case Execution Demand* (WCED) for processing of one event, the execution time variability is not accounted for, resulting in overly pessimistic analytic bounds for workloads with large variations in execution demand of tasks, therefore execution demand curves is used to model more than one event per activation.

(Thiele *et. al.*, 2002) relied on Real-Time Calculus to estimate various performance metrics, such as required buffer sizes and packet delays, resulting from implementing different scheduling policies on resource types PEs of a network processor. Their limitation is that their work could not account for the buffer constraints especially those related to the playout buffers and the variability of the task I/O rates.

(Chakraborty and Thiele, 2005) proposed a new task model for streaming applications combining the concept of arrival curves with the Recurring Real-time Task model (RRT), which may help to reduce the size of task graphs of the RRT model while modelling complex event streams.

(Wandeler, 2006) employed RTC in conjunction with other method to analyze complex distributed embedded real-time systems with a modular and extensible framework for system level performance analysis but his work could not address the challenge of timing correlations in complex embedded systems, problem of cyclic dependencies and fixed-point calculation within performance models and the problem with dynamic systems with feedback and state-dependant behaviour.

RTC has high level of abstraction and time-interval domain as limitations.

This shows that in MPA of real-time embedded systems, one particular method is not enough, a hybrid approach is the best.

2.10.3 Holistic Scheduling Analysis

In the real-time systems domain many results are available on schedulability analysis and worst-case response time analysis of individual tasks on single processor systems with various scheduling policies. Examples are analysis methods for fixed-priority, rate-monotonic, deadline monotonic, or earliest deadline first scheduling (Liu *et. al.*1973). Several proposals have been made to extend the concepts of classical scheduling theory to distributed systems. Such extensions must in particular consider the delays caused by the use of possibly shared communication resources that can typically not be neglected. The analytic integration of processor and communication infrastructure scheduling is often referred to as holistic scheduling analysis. But rather than denoting a specific performance analysis method, holistic scheduling analysis comprises a collection of

techniques for scheduling analysis of distributed embedded systems. In comparison to other performance analysis methods, the MPA framework models the service offered to an event stream explicitly, using the concept of resource streams. This approach has a number of advantages: First, it allows to model arbitrary complex resource availability patterns which may be experienced by individual event streams (or tasks) as a result of applying a certain scheduling or arbitration policy. Second, it supports the modularity of the performance analysis. Third, using the concept of resource streams it is easier to model hierarchical scheduling schemes and various resource reservation mechanisms. A variety of scheduling and arbitration policies can be modelled by a proper calculation (or definition) of the service curves within a scheduling network. For example, a fixed priority scheduling can be modelled by directly connecting the output resource stream (i.e. the remaining service) of a higher priority component to the resource input of the next (in terms of priority) component. For example, in the scheduling network in Fig 2.21, tasks T3 and T5 are scheduled on the CPU resource using the fixed priority scheme. T3 has the highest priority. To model proportional share schemes and their derivatives, we need to introduce into the scheduling network the corresponding scheduling modules that distribute the resource streams according to specified shares, and after that collect the remaining service. Fig. 2.21 depicts an example of such an arrangement for tasks T2 and T4.

The concept of scheduling network can be used to model the interactions between the requested and the offered service. In a scheduling network, the requested and the offered service are modelled by event and resource streams.

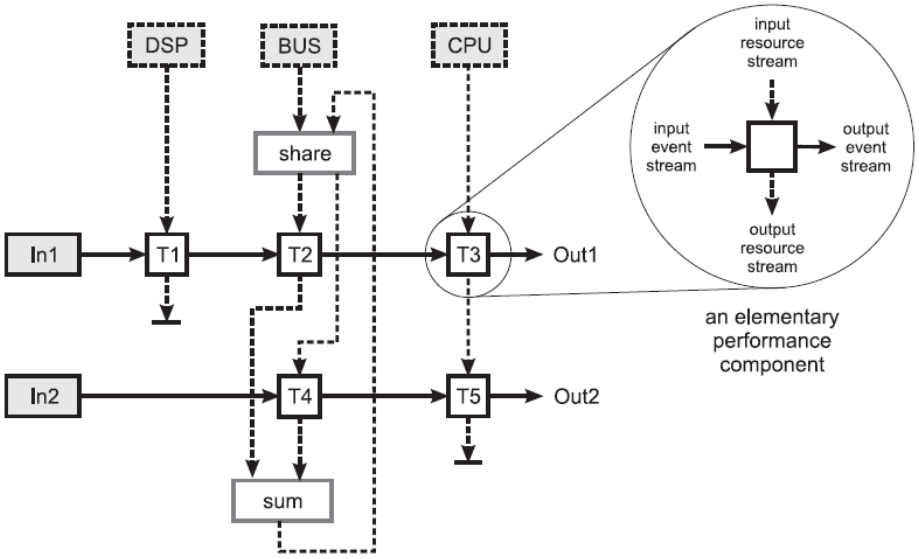


Fig. 2.21: A scheduling network modelling the application-to-architecture mapping.

Source: Liu et. al., (1973).

These streams flow through the network nodes called performance components that model the interactions between the streams. Fig. 2.21 shows an example of scheduling network corresponding to the application to architecture mapping. Solid and dashed arrows correspond to the event and resource streams, respectively.

An elementary performance component receives one event and one resource stream as its input (see Fig. 2.21). The input event stream abstracts arrivals of a certain request type, while the input resource stream models availability of a given resource for processing of this request type. Abstractly seen, the input event stream triggers the performance component, which in response proceeds by consuming resources provided by the input resource stream. This represents execution of a task on a PE.

An elementary performance component typically also produces one event and one resource stream as its output. An event within the output event stream signifies a completed processing of a corresponding input event. The output resource stream

represents the remaining service, that is, the service which has not been consumed by the performance component. This remaining service can then be used to process another event stream, i.e. it may serve as an input to another performance component. Likewise, the output event stream may represent requests for another resource, that is, it also may serve as an input to a different performance component. In this way, a scheduling network representing a performance model of the entire system (with a multitude of event streams and processing resources) can be constructed out of multiple independent performance components.

Besides the elementary performance components, a scheduling network may contain other types of nodes:

Resource modules model processing capabilities of PEs within the architecture. A resource module produces a stream corresponding to the unloaded resource that it models. In Fig. 2.21, resource modules are marked with dashed boxes. They represent the bus, DSP and CPU resources.

Input modules inject into the scheduling network event streams generated by the system's environment. In Fig.2.21, these are In1 and In2 modules.

Scheduling modules distribute resource streams between different performance components in accordance with a given resource management policy. A scheduling module receives and produces only resource streams (originated by the same resource). Using scheduling modules, we can model different scheduling and arbitration policies deployed on the PEs of the architecture. In Fig.2.21, for example, we have share and sum scheduling modules.

Hierarchical modules are complex performance components containing subnetworks of other components.

Calculating upper bounds on delay and backlog

In the performance analysis, in addition to the structural performance view of the system provided by the scheduling network, there is a need also to characterize the behaviour of the event and resource streams, and of the associated performance components. That is, we need to characterize timing properties of the streams and determine how these properties change when the streams pass through the performance components in the scheduling network as well as the backlog.

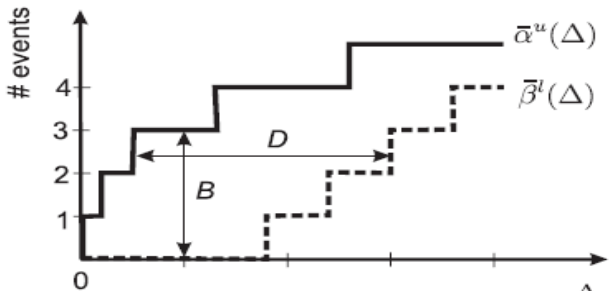


Fig. 2.22: Computing upper bounds on the delay, D, and the backlog, B.

Source: Liu et. al., (1973).

Given an upper arrival curve α^u and a lower service curve β^l as the input of an elementary performance component, (Fig. 2.22) we can compute upper bounds on the delay and on the backlog experienced by the event stream as a result of passing through this component (Le Boudec and Thiran, 2001).

The performance analysis of a distributed embedded system is done by combining the analysis of the single processing components of a performance model.

In a similar way, we can find upper bounds on the total delay and on the total backlog which an event stream may suffer as a result of passing through a chain of performance components. How this can be done is described in (Le Boudec and Thiran, 2001). This allows estimating such performance indexes of an embedded system as the worst-case

end-to-end delay and memory requirements. In the collection of holistic scheduling analysis techniques, every technique is tailored towards a particular combination of input event model, resource sharing policy and communication arbitration. While this permits detailed analysis of the temporal behaviour of a specific distributed system, it has the drawback that a new analysis method must be developed for every new input event model, communication protocol, resource sharing policy and combinations thereof. This circumstance not only restricts the applicability of holistic scheduling analysis, but the consequently large heterogeneous collection of different techniques also makes it difficult to use holistic scheduling analysis in practice (Wandeler, 2006).

In a related work of the area of classical real-time scheduling theory, (Liu *et. al.*, 1973) worked on scheduling but the result lead to poor processor utilization, and consequently to system designs with unreasonably high cost, or power consumption.

(Mok *et. al.*,1997) proposed the multiframe task model that extends the classical periodic task model of Liu *et. al.* (1973) by permitting periodic tasks whose WCETs may vary from one instance to another. In the multiframe task model, the WCETs of consecutive task instances are determined following a fixed cyclic pattern. The model was further extended in (Baruah *et. al.*,1999) which allowed not only to determine the WCET of a task instance, but also the time separation between two task instances following a cyclic pattern.

(Baruah,1998) presents a recurring Real-Time Task model (RRT) - a further generalization of the multiframe models. In the RRT model, a task is modelled by a set of subtasks arranged in a directed acyclic graph representing the conditional, non-deterministic behaviour of the task. Each subtask is characterized by its WCET, a relative deadline and a minimum triggering separation from its direct predecessors. The whole task graph is triggered sporadically with a specified minimum time separation

between the triggering of the last subtask in the graph and the triggering of the next task instance. Another workload model, also using conditional directed acyclic graphs to model tasks, is reported by (Pop *et. al.*, 2000). Instead of associating a deadline to each subtask in a task graph, the model associated a single deadline with the whole graph. Furthermore, it exposed the parallelism within a task for mapping on a multiprocessor architecture. In comparison to classical task models, the RRT model offers a great flexibility in modelling variability of the execution demand and irregular inter-arrival times. This flexibility is, however, limited to recurring patterns. The limitation of Baruah and Pop *et. al.* is that if workload bursts (characterized by periods with dense arrivals of tasks or increased execution demand or both) occur relatively seldom, then avoiding overly pessimistic results under the RRT model necessitates to consider very large task graphs, leading to inefficiency of the analysis. In other words, designers have to trade off the accuracy of the analysis for the analysis time, which for the RRT model increases exponentially with the problem size (Baruah, 2003).

Tindell and Clark (1994) addressed systems with fixed priority scheduling policy deployed on processor nodes communicating via a bus using a Time Division Multiple Access (TDMA) protocol. The methods can be very effective in modelling complex timing relations (e.g. phasing) between the tasks. However, they are often attributed to a lack of scalability and modularity (Jersak *et. al.*, 2004).

2.10.4 Compositional Approach

In compositional approach, every single processor or communication link of a distributed system is analyzed locally. To interconnect the various components, the method relies on a set of standard event arrival patterns. Based on the arrival patterns of the incoming event streams and on the scheduling policy of the component, the

appropriate classical analysis technique is chosen individually for every single processor or communication link to compute the worst-case and best-case response time of every event stream at the component as well as to compute the arrival patterns of the outgoing event streams that will trigger succeeding components. The local analysis results are then combined to obtain global end-to-end delays and buffer requirements. The approach is however only feasible if the arrival patterns of the incoming event streams at a component fit the basic models for which results on computing bounds on the response times are available (Wandeler, 2006). While using compositional methods, three main problems may arise. Firstly, the architecture of such systems which is highly heterogeneous, the different architectural components are designed assuming different input event models and use different arbitration and resource sharing strategies. This makes any kind of compositional performance analysis difficult. Secondly, applications very often rely on a high degree of concurrency. Therefore, there are multiple control threads, which additionally complicate timing analysis. And thirdly, we can not expect that an embedded system only needs to process periodic events where to each event a fixed number of bytes is associated. If for example the event stream represents a sampled voice signal, then after several coding, processing and communication steps, the amount of data per event as well as the timing may have changed substantially. In addition, stream based systems often also have to process other event streams that are sporadic or bursty, e.g. they have to react to external events or deal with best-effort traffic for coding, transcription or encryption. There are only a few approaches available that can handle such complex interactions (Lothar and Ernesto, 2009).

In a related work, (Henia *et. al.*, 2005) proposed a compositional performance analysis methodology with the main goal to directly exploit the successful results of classical scheduling theory, in particular for sharing a single processor or a single communication

link. In their work, they defined two types of interfaces that may be placed between components. The first one is Event Model Interfaces (EMIF) which perform a type conversion between certain arrival patterns, that is, they change the mathematical representation of event streams. The second is Event Adaption Functions (EAF) which must be used whenever they exist on EMIF. In this case, the hardware (HW) implementation of the analyzed system must be changed in order to make the system analyzable, for example by adding play-out buffers between components.

The work of Henia *et. al.* has the following limitation; the compositional approach is bound to a limited set of classical arrival patterns that is often not sufficient to represent event streams with complex timing behaviours. As a result, they must be represented in one of the supported arrival patterns, usually with loss in accuracy. Also the arrival patterns often need to be adapted between components, either again with loss in accuracy (EMIF), or even with enforcing a change in the system HW implementation (EAF). Finally the approach is not compositional in terms of the resources, as their service is not modelled explicitly.

To overcome these limitations, (Jersak *et. al.*, 2004) extended the compositional performance analysis framework presented by Henia *et. al.*, they introduced the concept of intra-stream contexts that specify a cyclic pattern of different events that arrive on an event stream. The timing properties of the event stream are thereby decoupled from the cyclic event pattern, and are specified using a set of classical arrival patterns. On such an event stream with intra-stream context, the WCET of every event, when triggering a computation resource, is then determined from its event type.

2.10.5 Timed Automata based performance analysis

Timed automata is a popular formalism for the specification of real- time systems. It can be used in combination with a logic language to verify system properties by model checking (Rajeev and David, 1994).

The work of (Christer *et. al.*, 1999) showed that the schedulability analysis of an event driven system can be represented as a reachability problem for timed automata and thus can be tackled with model checking. The timed automata based schedulability analysis is implemented in the TIMES tool (Tobias *et. al.*, 2002). TIMES permits users to analyze systems that are described as a set of tasks which are triggered either periodically or by external event streams modelled through appropriate timed automata. However, the TIMES tool is limited to the schedulability analysis of single processors. Thus, it is not suited for performance analysis of distributed systems.

In a related work, (Hendriks and Verhoef, 2006) presented an approach to performance analysis of distributed embedded systems based on the model checking of timed automata networks. They modelled the environment and the resources of a system as timed automata. The various components are then composed into a network of timed automata that models distributed embedded systems. The performance properties of the system are verified through exhaustive model checking. They used UPPAAL for the modelling and verification of timed automata networks.

The UPPAAL tool environment allows users to validate and verify real-time systems modelled as networks of timed automata (Gerd *et. al.*, 2001).

Their approach modelled the environment and the hardware resources. The timed automata models of the single system components are aggregated into a timed automata network that represents distributed embedded systems.

In modelling the environment, Fig.2.23 shows a timed automaton that models a periodic event stream with period *P*. After an undefined initial offset the automaton generates events at intervals of exactly *P* time units. The generation of an event is modelled by the increment of the global variable *req*. Fig.2.24 shows a timed automaton that models a periodic event stream with jitter $J \leq P$.

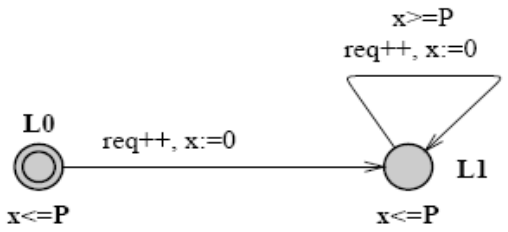


Fig. 2.23: Timed automata model for a periodic event stream

Source: Gerd *et. al.*, (2004).

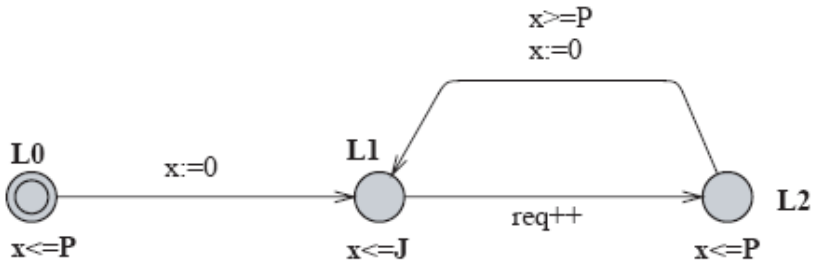


Fig. 2.24: Timed automata model for a periodic event stream with jitter

Source: Gerd *et. al.*, (2004).

From the above, new event stream models can be designed easily. Basically any deterministic event stream can be modelled.

In modelling the hardware resources, each processing component is modelled as a separate timed automaton. A processing component is either idle or busy computing some function. In the same way, each communication link is modelled as a timed

automaton. Each link is either idle or transporting some data. For shared resources the adopted scheduling policy determines the structure of the model. For example Fig.2.25 shows a timed automaton that models a hardware resource with two tasks implementing preemptive Fixed Priority (FP) scheduling. The resource can either be idle or process T1 or process T2. The location *pre T1* models the fact that T1 can pre-empt T2. The synchronization models urgent edge and makes sure that the corresponding edge is taken as soon as it is enabled; see (Gerd *et. al.*, 2004) for details.

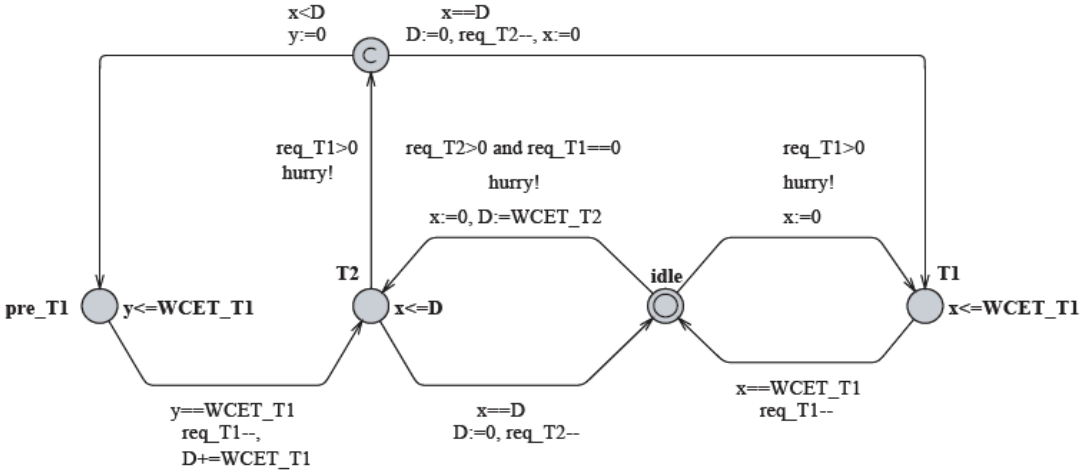


Fig. 2.25: Timed automata model for a preemptive FP resource with two tasks

Source: Gerd *et. al.*, (2004).

In performance analysis, timed automata models of the single system components are aggregated into a timed automata network that represents a distributed embedded system. The single components interact via global variables and channels. For example, suppose that the timed automaton of an input event generator increments a global variable *req* to model the request of a task activation on a certain resource. The timed automaton that models the corresponding resource is sensitive to increments of the variable *req* and immediately starts the execution of the corresponding task if no higher

priority task has to be executed. The completion of the task execution is modelled by the decrement of the variable *req*. Let's suppose that the corresponding output event triggers a second task. This can be modelled by incrementing a second global variable *req2* simultaneously with the decrement of *req*. Again, another automaton will be sensitive to the increments of *req2*, start the corresponding task and so on. In this way, the propagation of events through the distributed system can be easily modelled.

The performance attributes of distributed embedded systems are derived by verifying properties of the corresponding timed automata network. For instance, to ensure that the maximum backlog of a certain task does not exceed a given value *b*, it is sufficient to verify the following property by model checking: $AG\ (req \leq b)$ where 'AG' stands for 'always generally' (= invariantly) and *req* is the global variable that counts the activation requests of the corresponding task. Also it is possible to derive the *exact* maximum backlog by finding the smallest *b* that satisfies the above property which can be done by using a binary search strategy.

The verification of end-to-end delays is a little more involved as it requires adapting the timed automata models of the corresponding input event generators.

Fig.2.26 shows the variant of a periodic event stream generator that permits to verify end-to-end latencies.

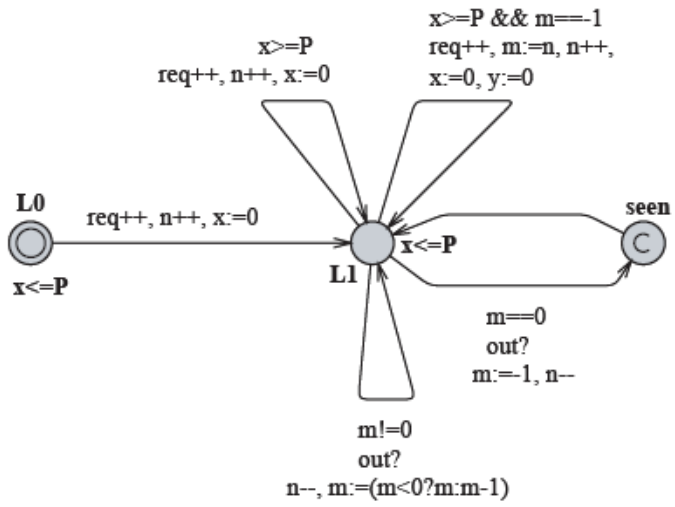


Fig. 2.26: Timed automata model to measure end- to-end delay

Source: Gerd *et. al.*, (2004).

The automaton is synchronized with the system output over the global channel *out* and can keep track of the amount of time that passes between the generation of an event and its output from the system. Basically, the automaton can generate input events in the same way as the automaton of Fig.2.23 (left upper transition), but it can also arbitrarily choose to measure the end- to-end delay of an event (right upper transition). The variable *n* (initially 0) keeps track of the number of events that have been fed into the system and for which no response (a synchronization over the channel *out*) has been received yet. The clock *y* measures the response time and *m* (initially -1) equals the number of responses that must be discarded before the one used for the measurement is seen. At most, one measurement can be in progress and $m = -1$ if no measurement is in progress.

Similarly measuring automaton variants are available also for other event streams. To ensure that the worst-case end-to-end delay of an event does not exceed a given value *d* it is sufficient to verify the following property by model checking: $AG\ (IG.seen \rightarrow IG.y$

$< d$) where we assume that 'IG' is the name of the measuring automaton. Again, the exact worst-case end-to-end delay can be determined by finding the smallest d that satisfies the property.

The timed automata method for performance analysis based on model checking permits to derive not only hard but also exact bounds for performance properties of a distributed system but with space problem. The modelling of a distributed embedded system as a network of timed automata can easily lead to a state space explosion which may prolong early termination and reachability analysis of the timed systems as potential limitation (Ericsson et. al, 1999).

UPPAAL is a tool suite for symbolic model checking of real-time (Larsen *et. al.*, 1997) jointly developed at the University of Uppsala (Sweden) and Aal-borg (Denmark). Besides model checking, it also supports simulation of timed automata and has some facilities to detect deadlocks. UPPAAL has been applied to several industrial case studies such as real-time protocols, multi-media synchronization protocols and also to communication protocols as well as multimedia applications (Howard *et. al.*, 1998).

UPPAAL is an integrated tool environment use for modelling, simulation and verification of real-time systems. It has been used to verify various benchmark examples and industrial applications, some of which are briefly discussed as follows:

Audio/Video control protocol which is highly dependent on real-time. The protocol was developed by Bang and Olufsen to transmit messages between audio/video components over a single bus, but it was known to be faulty and the error was not found using conventional testing methods. UPPAAL revealed the errors and suggested corrections (Klaus *et. al.*, 1997).

In the work of (Argenio *et. al.*, 1997), on Bounded Retransmission Protocol, UPPAAL showed that the correctness of the protocol is dependent on correctly chosen time-out values.

Magnus *et. al.*, (1998) applied UPPAAL in an industrial case-study to the design and analysis of an embedded systems gearbox controller prototype for vehicles by Mecel AB (a Swedish company developing control systems for vehicle industries). The correctness of the gear-box controller design was established by automatic verification of 46 properties derived from informal requirements specified by Mecel AB.

Also in the work of (Henrik and Paul 1997), TDMA (Time Division Multiple Access) Protocol Start-Up Mechanism, UPPAAL was used for formal verification of the start-up algorithm of a TDMA protocol. UPPAAL checked that an ensemble of three communicating stations becomes synchronized and operational within a bounded time.

Henrik *et. al.*, (1996) used UPPAAL to model Collision Avoidance Protocol implemented on top of an Ethernet-like medium such as the CSMA/CD (Carrier Sense Multiple Access/ Collision Detection) Protocol to ensure an upper bound on the communication delay between the network nodes.

The first version of UPPAAL was released in 1995. Since then it has been in constant development (Johan *et. al.*, 1998). Experiments and improvements include data structures, partial order reduction, symmetry reduction, a distributed version of UPPAAL, guided and minimal cost reachability, work on UML Statecharts, acceleration techniques and new data structures and memory reductions (Gerd *et. al.*1999). The current version of the tool is 4.0.11. It features a Java user interface and a verification engine written in C++ . An overview of the model checker UPPAAL is given in Fig.2.27.

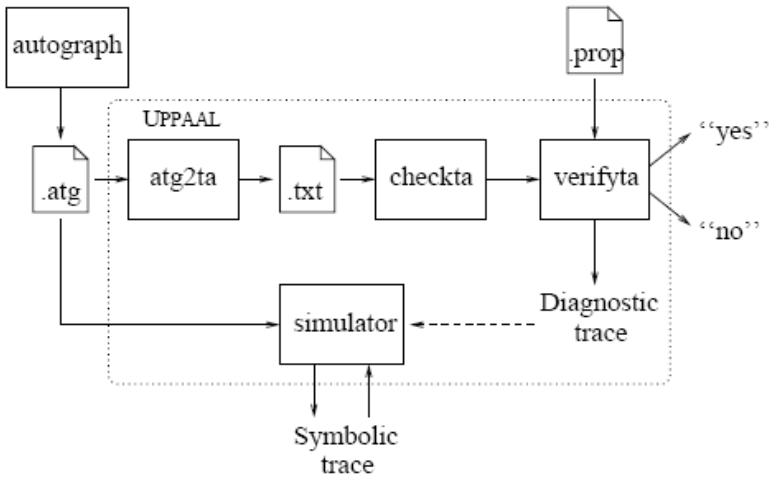


Fig.2.27: An overview of the model checker UPPAAL

Source: Larsen et. al., (1997)

For reasons of efficiency, the model checking algorithms that are implemented in UPPAAL are based on (sets of) clock constraints rather than on explicit (sets of) regions. The property that each region can be expressed by a clock constraint over the clocks involved is exploited. By dealing with (disjoint) sets of clock constraints, a coarser partitioning of the (infinite) state space is obtained (Yovine, 1998).

UPPAAL facilitates the graphical description of timed automata by using the tool AUTOGRAPH. The output of AUTOGRAPH is compiled into textual format (using component atg2ta), which is checked (by checkta) for syntactical correctness. This textual representation is one of the inputs to UPPAAL'S verifyta.

The verifier can be used to determine the satisfaction of a given real time property with respect to a timed automaton. If a property is not satisfied, a diagnostic trace can be generated that indicates how the property may be violated. UPPAAL also provides a simulator that allows a graphical visualization of possible dynamic behaviours of a system description (that is, a symbolic trace). The diagnostic trace, generated in case a

property is violated, can be fed back to the simulator so that it can be analyzed with the help of the graphical presentation.

The model-checker UPPAAL is based on the theory of timed automata (John *et. al.*, 2001) and its modelling language offers additional features such as bounded integer variables and urgency. The query language of UPPAAL, where properties to be checked is specified is a subset of CTL (Computation Tree Logic) as discussed section 2.4.1 (Thomas 1994). Subsequently we present the modelling and the query languages of UPPAAL and we give an intuitive explanation of time in timed automata.

Timed Automata in UPPAAL

The UPPAAL modelling language extends timed automata with the following additional features for performance analysis:

- Templates automata are defined with a set of parameters that can be of any type (e.g., **int, chan).** These parameters are substituted for a given argument in the process declaration.
- Constants are declared as **const name value.** Constants by definition cannot be modified and must have an integer value.
- Bounded integer variables are declared as **int** [min,max] name, where min and max are the lower and upper bound, respectively. Guards, invariants, and assignments may contain expressions ranging over bounded integer variables. The bounds are checked upon verification and violating a bound leads to an invalid state that is discarded (at run-time). If the bounds are omitted, the default range of-32768 to 32768 is used.
- Binary synchronisation channels are declared as **chan c.** An edge labelled with **c**! synchronises with another labelled c?. A synchronisation pair is chosen non-deterministically if several combinations are enabled.

- Broadcast channels are declared as **broadcast chan c.** In a broadcast synchronisation one sender **c**! can synchronise with an arbitrary number of receivers c?. Any receiver that can synchronise in the current state must do so. If there are no receivers, then the sender can still execute the **c**! action, i.e. broadcast sending is never blocking.
- Urgent synchronisation channels are declared by prefixing the channel declaration with the keyword **urgent**. Delays must not occur if a synchronisation transition on an urgent channel is enabled. Edges using urgent channels for synchronisation cannot have time constraints, i.e., no clock guards.
- Urgent locations are semantically equivalent to adding an extra clock x, that is reset on all incoming edges, and having an invariant x<=0 on the location. Hence, time is not allowed to pass when the system is in an urgent location.
- Committed locations are even more restrictive on the execution than urgent locations. A state is committed if any of the locations in the state is committed. A committed state cannot delay and the next transition must involve an outgoing edge of at least one of the committed locations.
- Arrays are allowed for clocks, channels, constants and integer variables. They are defined by appending a size to the variable name, e.g. **chan c [4] ; clock a[2]; int [3] u[7];.**
- Initialisers are used to initialise integer variables and arrays of integer variables. For instance, **int i := 2;** or **int i [3] := {1, 2, 3};.**

Expressions in UPPAAL

Expressions in UPPAAL range over clocks and integer variables. The Bacchus Normal Form (BNF) is given below. Expressions are used with the following labels:

- Guard: A guard is a particular expression satisfying the following conditions: it is side-effect free; it evaluates to a boolean; only clocks, integer variables, and constants are referenced (or arrays of these types); clocks and clock differences are only compared to integer expressions.
- Synchronisation: A synchronisation label is either on the form *Expression*! or *Expression?* or is an empty label. The expression must be side-effect free, evaluate to a channel, and only refer to integers, constants and channels.
- Assignment: An assignment label is a comma separated list of expressions with a side-effect; expressions must only refer to clocks, integer variables, constants and only assign integer values to clocks.
- Invariant: An invariant is an expression that satisfies the following conditions: it is side-effect free; only clock, integer variables, and constants are referenced.

Expression → ID | NAT

| Expression '[' Expression'] '

| '(' Expression ')'

| Expression ' ++' | '++' Expression

| Expression ' --' | '--' Expression

| Expression AssignOp Expression

| UnaryOp Expression

| Expression BinaryOp Expression

| Expression ' ? 'Expression ':' Expression

| Expression '.' ID

UnaryOp → '-' | ' !' | 'not'

BinaryOp→ '<' | '<=' |'==' | '!=' |'>=' |' >'

| '+' |'-' | '*' |'/' | '%' | '&'

| '|' | '^' | '<<' | '>>' | '&&' |'||'

| *'<?'* | *'>?'* | 'and' | 'or' | 'imply'

AssignOp → ':=' | '+=' | '- =' | '*=' | '/=' | '%='

| '|=' | '&=' | '^=' | '<<=' | '>>='

The Query Language in UPPAAL

The main purpose of a model checker is to verify the model with respect to a requirement specification. Like the model, the requirement specification must be expressed in a formally well-defined and machine readable language. Several such logics exist in the scientific literature and UPPAAL uses a simplified version of CTL discussed above. Like in CTL, the query language consists of path formulae and state formulae. State formulae describe individual states, whereas path formulae quantify over paths or traces

of the model. Path formulae can be classified into *reachability, safety* and *liveness.* Each type is described below.

State Formulae:

A state formula is an expression that can be evaluated for a state without looking at the behaviour of the model. For instance, this could be a simple expression, like **i** == 7, that is true in a state whenever *i* equals 7. The syntax of state formulae is a superset of that of guards, that is, a state formula is a side-effect free expression. It is also possible to test whether a particular process is in a given location using an expression on the form P.l, where P is a process and l is a location.

In UPPAAL, deadlock is expressed using a special state formula (although this is not strictly a state formula). The formula simply consists of the keyword **deadlock** and is satisfied for all deadlock states. A state is a deadlock state if there are no outgoing action transitions neither from the state itself or any of its delay successors. Due to current limitations in UPPAAL, the deadlock state formula can only be used with reachability and invariantly path formulae (see below).

Path formulae:

Reachability Properties:

Reachability properties are the simplest form of properties. They ask whether a given state formula, □, *possibly* can be satisfied by any reachable state. Another way of stating this is: Does there exist a path starting at the initial state, such that □ is eventually satisfied along that path. Reachability properties are often used while designing a model to perform sanity checks. For instance, when creating a model of a communication protocol involving a sender and a receiver, it makes sense to ask whether it is possible for the sender to send a message at all or whether a message can possibly be received. These

properties do not by themselves guarantee the correctness of the protocol (that is, that any message is eventually delivered), but they validate the basic behaviour of the model.

One can express that some state satisfying □ should be reachable using the path formula *E* <> □. In UPPAAL, this property is written using the syntax E <> □.

Safety Properties:

Safety properties are of the form: "something bad will never happen". For instance, in a model of a nuclear power plant, a safety property might be, that the operating temperature is always (invariantly) under a certain threshold, or that a meltdown never occurs. A variation of this property is that "something will possibly never happen". For instance when playing a game, a safe state is one in which we can still win the game, hence we will possibly not loose.

In UPPAAL these properties are formulated positively, for example, something good is invariantly true. Let □be a state formulae. One can express that □ should be true in all reachable states with the path formulae A [] □ whereas E [] □ says that there should exist a maximal path such that □ is always true. In UPPAAL, it is written A [ ] □ and E [ ] □, respectively.

Liveness Properties:

Liveness properties are of the form: something will eventually happen, for example, when pressing the *on* button of the remote control of the television, then eventually the television should turn on. Or in a model of a communication protocol, any message that has been sent should eventually be received.

In its simple form, liveness is expressed with the path formula A<> □, meaning □ is eventually satisfied. The more useful form is the *leads to* or *response* property, written □ ⇝ ψ which is read as whenever □ is satisfied, then eventually ψ will be satisfied, for

example whenever a message is sent, then eventually it will be received. In UPPAAL these properties are written as A<> □ and □⇢ *ψ,* respectively.

Property Equivalences

The UPPAAL requirement specification language supports five types of properties as illustrated below in Table 2.2.

Table 2.2: Property Equivalences

Name	Property
Possibly	E<> p
Invariantly	A[] p
Potentially always	E[] p
Eventually	A<> p
Leads to	p --> q

Legend:

A and E are called path quantifiers, and

[] and <> quantify over states of a selected path.

2.10.6 Stochastic analysis method

Stochastic analysis method is rarely used for performance analysis because of its tighter analytic bounds. (Lehoczky, 1996) uses stochastic characterization for inter-arrival times, execution demands and deadlines, and relies on queuing theoretic methods for performance evaluation. The stochastic workload models can result in tighter analytic bounds and hence in more economical designs, but at the expense of some (usually controlled) fraction of missed deadlines. Because of this, their application area is limited to soft real-time.

As discussed above each of these approaches has its short coming hence, in practice, a hybrid approach is adopted.

2.11 Comparison of Performance Analysis Methods

Most of the comparison and classification criteria for performance analysis methods are not directly quantifiable, but play an important role in the distinction of performance analysis approaches.

The classification criterion for performance analysis methods is given by the set of analyzable performance metrics. A performance analysis approach may support the analysis of system characteristics like timing aspects, memory requirement, resource utilization or power consumption. The analysis of timing aspects includes the determination of best-case and worst-case latencies as well as end-to-end delays. The analysis of the memory requirement is often related to the determination of worst-case buffer fill levels. The following performance metrics are considered to get the result shown in Table 2.3.

Modelling scope

A fundamental comparison criterion for performance analysis methods is the modelling scope. By the modelling scope of a certain approach we mean the set of distributed embedded systems that can be represented and analyzed using the modelling power of the method. For instance the capability to model several particular system characteristics, such as hierarchical scheduling, blocking times, multiple task activation etc., differentiates the modelling scopes of the various performance analysis methods.

Correctness and accuracy

A worst-case analysis is said to be correct if the result is a hard upper bound for the real worst-case performance of the considered system. In other words, there are no reachable system states which would allow the calculated bound to be violated.

The accuracy of a performance analysis is usually not quantifiable because the exact worst-case performance of the considered system is unknown. However, a performance analysis method is more accurate than another for a certain system if it provides a tighter upper (lower) bound for the worst-case (best-case) performance.

Modularity

Performance analysis methods can be classified into modular and holistic approaches. The modular approaches analyze the performance of single components of the system and propagate the results in order to determine the performance of the entire system. In contrast, the holistic approaches consider the system as a whole. Modular performance analysis methods are typically less complex and easier to reuse than holistic ones.

Modelling effort and tool support

An important criterion for the comparison of performance analysis methods is the effort that it costs the designer to create system models. The modelling effort can be largely alleviated by appropriate software tools.

Analysis effort

This criterion considers the computational effort that is necessary to obtain performance analysis results. For instance one could compare the running times of the tools that implement the different performance analysis approaches.

Scalability

A relevant comparison criterion for performance analysis approaches is scalability. This point is pertinent to several of the previous criteria: the modelling and analysis efforts as

well as the accuracy of the results may be greatly influenced by the dimension of the analyzed system.

End-user complexity and learning curve

Other aspects that can be considered for the comparison of performance analysis approaches are the complexity experienced by the end-user that applies a certain method or tool, as well as the progression of its learning curve. In particular, these points are largely influenced by the amount of background knowledge that a user must acquire about a certain performance analysis approach in order to be able to apply it. The multitude of heterogeneities among the performance analysis approaches and the different levels of abstraction in the modelling of particular system attributes make this task very complex (Simon, 2006). As a result, the comparison of the various modelling scopes is restricted to a number of key attributes and the result is shown in Table 2.3. The comparison is based on the modelling capabilities of concrete implementations of the various performance analysis approaches. Also, it is necessary to point out that a 'low/poor' does not mean that the modelling of the corresponding system characteristic is conceptually impossible for the corresponding performance analysis approach. Only that, it denotes that no significant research has so far been conducted to integrate this particular aspect.

Table 2.3: Comparison of performance analysis methods

	Simulation and Formal Analysis	Real-Time Calculus	Holistic Scheduling Analysis	Compositional Method	Timed Automata Based Performance

					Analysis
Resource Utilization	5	5	3	4	5
Memory (Buffer Spaces)	4	4	2	3	3
Throughput	3	4	4	3	4
End-to-end delays	4	2	2	2	4
Modularity	2	4	3	4	4
Response Time	3	4	3	3	4
Reusability	4	3	2	3	3

Interpretation

Very Good / Very High	-	5
Good / High	-	4
Average / Medium	-	3
Poor / Low	-	2
Very Poor / Very Low	-	1

2.12 Assessing Existing Performance Analysis Methods

According to (Yau-Tsun and Sharad, 1999), existing performance analysis techniques have one or more of the following limitations; they cannot model complicated programs, they cannot model advanced micro-architectural features of the processor, such as cache memories and pipelines and they cannot be easily retargeted for new hardware platforms, as a result, the problem is partitioned into two sub-problems; program path analysis and micro-architecture modelling. It is necessary to know the worst-case execution time (WCET) of a program when designing and verifying real-time systems. The WCET depends both on the program flow, such as loop iterations and function calls, and on hardware factors, such as caches and pipelines. WCET estimates should be both safe (no underestimation allowed) and tight (as little overestimation as possible) (Jakob *et. al.*, 2002). As a result, in practice, a hybrid approach of performance analysis methods is mostly employed with codesign

techniques which allow simultaneous verification of both software and hardware and in what extent it fits into the desired function.

CHAPTER THREE

SYSTEM MODELLING AND ANALYSIS

3.1 Introduction

The interest in verification of real-time embedded systems has increased significantly in the nineties and is still increasing. Introducing time into a temporal logic must be carried out in a careful way in order to keep the model-checking problem to be decidable. The basic idea of timed Computational Tree Logic (CTL) is to allow simple time constraints as parameters of the usual CTL temporal operators.

The main difficulty of model-checking models in which the time domain is continuous is that the model to be checked has infinitely many states. For each time value the system can be in a certain state, and there are infinitely many of those values. The essential idea to perform model checking on a continuous time-domain is to realize a discretization of this domain on-demand, that is depending on the property to be verified and the system model, because real-time embedded systems are heterogeneous in nature and the design is mostly application specific with emphasis on efficient implementation.

In this chapter, Proteus Virtual System Modelling (VSM) is used for the simulation of the major component, 'Microcontroller' used for our 'TCAS' (Traffic Alert and Collision Avoidance System) real time embedded system design to ascertain whether its performance meet the minimum requirement for the design. The performance indexes investigated are timing, memory usage, speed and power consumption.

'UPPAAL' a timed automata based performance analysis method among many performance analysis methods (see section 2.10), is used in this research work for modelling and performance analysis of TCAS.

Tool developers applied UPPAAL in various time-critical systems, ranging from communication protocols to safety-critical systems such as gearbox controller for vehicles by Mecel AB (Magnus *et. al.*, 1998).

The model checking algorithms implemented in UPPAAL are based on sets of clock constraints. UPPAAL facilitates the graphical description of timed automata by using the tool AUTOGRAPH. The output of AUTOGRAPH is compiled into textual format which is checked (by checkta) for syntactical correctness. This textual representation is one of the inputs to UPPAAL'S verifyta. (See section 2.10.5 for detailed information on UPPAAL). The verifier (verifyta) is used to determine the satisfaction of a given real time property with respect to a timed automaton. If a property is not satisfied, a diagnostic trace can be generated that indicates how the property may be violated.

3.2 Adopted Tools and Methods

This section outlines the software and hardware tools as well as methods adopted for the design of our own embedded system.

Model-based Object Oriented Systems Engineering (MOOSE) (see section 2.8.6) was adopted for modelling of Real- time embedded systems design.

Proteus Virtual System Modelling (VSM) is a tool that was used for the simulation and modelling of the major component, 'Microcontroller' was used for the real time embedded systems design.

Among several performance analysis methods discussed earlier (see section 2.10), 'UPPAAL' a Timed Automata based performance analysis method was used in this research work for modelling and performance analysis of 'TCAS'. A timed automata is an extension of finite-state automata with clocks that are used to measure time. Since their conception (Alur and Dill, 1996), timed automata have been used for the

specification of various types of time-critical systems, ranging from communication protocols to safety-critical systems. (See section 2.10.5 for details).

In the sequel, *x, y* or *z* will be used as clocks. A state of a timed automaton consists of the current location of the automaton plus the current values of all clock variables. Clocks can be initialized (to zero) when the system makes a transition. Once initialized, they start incrementing their value implicitly. All clocks proceed at the same rate. The value of a clock thus denotes the amount of time that has been elapsed since it has been initialized. Conditions on the values of clocks are used as enabling conditions of transitions, only if the clock constraint is fulfilled, the transition is enabled, and can be taken; otherwise, the transition is blocked. Invariants on clocks are used to limit the amount of time that may be spent in a location. Enabling conditions and invariants are constraints over clocks.

3.3 Experimental/Laboratory Setup

One of the objectives of this research work is to model and analyse a GPS-based Traffic Alert and Collision Avoidance System. We used Proteus to model and simulate a Microcontroller used for our TCAS design before modelling a complete TCAS system to investigate possible combination of parameters that will match closely to the design requirements.

3.3.1 Timing Analysis in Real-Time Embedded Systems

The common cause of failure in real-time embedded systems is timing violations but Temporal logics like Propositional Linear Temporal Logic (PLTL) and CTL use to analyse the occurrence of events with various states in embedded system facilitate the specification of properties that focus only on the temporal order of theses events as discussed in section 2.4.1 of chapter two. For example, given CTL model M, state $s \in S$,

and CTL formula , to find out whether *M, s* , that is, the interest is to establish whether the property is valid in state *s* of model *M,* here, *M* is assumed to be finite. Like the events of TCAS which are abstracted into logical model M, can be verified by the logical formula that represents the required property of TCAS system (Fig 3.1).

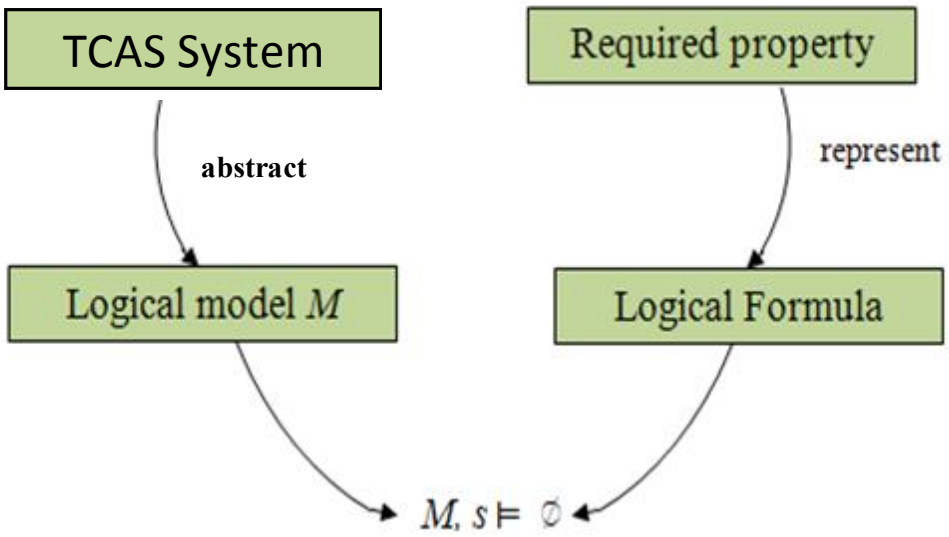


Fig. 3.1: Model checking

This temporal order of event is a qualitative notion but time is not considered quantitatively. For instance, let proposition *p* correspond to the occurrence of event A in TCAS system, and *q* correspond to the occurrence of another event B. Then the PLTL-formula G [*p* Fq]; where G means "Globally in the future" and F ("in a Future state"), states that event A is always eventually followed by event B, but it does not state anything about how long the period between the occurrences of A and B will be. This absent notion of quantitative time is essential for the specification of time-critical systems. That is, time-critical systems must satisfy not only functional correctness

requirements, but also timeliness requirements, such systems are typically characterized by quantitative timing properties relating occurrences of events. Quantitative timing information is also a necessary ingredient in order to analyze and specify the performance aspects of systems, for example, aircraft will arrive at Closet Point of Approach (CPA) which is considered dangerous to incoming aircraft within 30 seconds can only be stated if one can measure the amount of elapsed time.

In order to quantify timing, the branching-time temporal logic is extended with a quantitative notion of time. The most important change when switching to quantitative time is the ability to measure time. That is, how is the notion of time incorporated in the state-based model and how is the relationship between states and time? Also when incorporating time the following issues need to be addressed (Koymans, 1992):

a. what is the notion of time reference (absolute or relative)?
b. what is the semantical time domain (discrete or continuous)?
c. how is time measure presented (additive or metric)?

In order to answer the questions in a-c, and to investigate other properties of embedded system like memory usage, speed and power consumption, we used Proteus 7 professional to model and simulate 'Microcontroller', a major component used for our 'TCAS' real time embedded systems design.

3.3.2 Features and Potential Application of PIC16F887A Microcontroller

In any embedded system design, features of the MCU need to be matched as closely as possible to the application requirements (Martin, 2003).

PIC16F887A Microcontroller used for this experiment is a powerful easy-to-program 8-bit microcontroller with high-endurance 8K ROM memory in FLASH technology, 256

bytes EEPROM which allow data to be written more than 1,000,000 times and 368 bytes RAM memory chip that can be reprogrammed up to 100,000 times.

It has 35 input/output pins with a synchronous serial port that can be configured as either 3-wire Serial Peripheral Interface (SPI) or the 2-wire Inter-Integrated Circuit (I^2C) bus and an Enhanced Universal Asynchronous Receiver Transmitter (EUSART). It has In-Circuit Serial Programming option which enables the chip to be programmed even when embedded in the target device.

The power supply voltage is 2.0-5.5V with frequency of 0-60 MHz. PIC16F887A has three independent timers/counters; Capture, Compare and PWM **(**Pulse Width Modulation).

All of these features make PIC16F887A ideal for more advanced level Analogue/Digital (A/D) applications in automotive, industrial machine control devices, appliances or consumer applications.

3.3.2.1 Laboratory Setup

Quantitative timing information is not only essential for time-critical systems, but is also a necessary requirement in order to analyze and specify the performance aspects of systems. It is assumed that time is considered quantitatively because typical performance related statements like "job arrivals take place with an inter-arrival time of 10 seconds", can only be stated if it is possible to measure the amount of elapsed time. Results of the experiment are to a very good approximation of real-time embedded systems. To investigate timing in real-time embedded systems and some performance indexes of real-time embedded systems like memory usage, speed and power consumption, a laboratory at Micro-Scale Embedded Ltd., Kaduna, Nigeria, was

setup, where simulation and modelling of Real-time embedded systems design using Proteus was carried out.

Proteus VSM (Virtual System Modelling) combines a superb mixed mode circuit simulation based on the industry standard specification with animated components. It provides an architecture in which additional animated models may be created by end users and microprocessor models to facilitate co-simulation of complete microcontroller based designs. For the first time ever, it is possible to develop and test such designs before a physical prototype is constructed (www.labcenter.com, 2010).

Proteus is considered as one of the best among other circuit simulation software like PSpice (Simulation Program with Integrated Circuit Emphasis) which run on personal computer, Electronics Workbench, Multisim, Solve Elec, 5Spice and 555 Timer (www.physicsforums.com, 2010). Proteus allows professional users to run interactive simulations of real designs, and to reap the rewards of this approach to circuit simulation. This is possible because you can interact with the design using on screen indicators such as LED and LCD displays and actuators such as switches and buttons. The simulation takes place in real time , a 1GHz Pentium III can simulate a basic 8051 system clocking at over 12MHz. Proteus VSM also provides extensive debugging facilities including breakpoints, single stepping and variable display for both assembly code and high level language source (www.labcenter.com, 2010).

In this experiment, Proteus VSM runs on Pentium IV System with XP Service Park 2, 1G RAM and 100 GB of Hard Disk. The hardware components used for the experiment are Led-red, GPS receiver, Processor's Programmer (Microchip PICkit2) and LM016L Display Alphanumeric LCD output at Virtual Terminal and PIC16F887A while CCS C (Object-Oriented Program) is used as source code for the design.

3.3.2.2 System Design

Components were selected from devices' library and connected on the schematic layout to give desired design (Fig.3.2 and Fig.3.3). Properties of each component such as component value and Processor Clock Frequency which is equivalent to Crystal value are set. The source code for the design is linked. Proteus is run to simulate the circuit, the status bar with the time that the animation has been active for appeared. After that, detected errors are debugged in the program then the logic state of the pins could be seen on the schematic with display on LED and output on the Virtual Terminal. Component values are varied to obtain results in Table 3.1. In the table, we introduced different delay time and varied clock frequency values of PIC16F887A to see the effect on the CPU, memory usage and the output speed while each execution time was run for a constant time of 30s. We observed that various delay in the execution time of source program running on microcontroller has effect on performance characteristics.

To observe the information displays by GPS receiver when interfaced with the Microcontroller, GPS receiver, LCD and Microcontroller are connected; the source code for the design is downloaded to the Microcontroller by Processor's Programmer (Microchip PICkit2). The GPS receiver displays trajectory information such as latitude, longitude and time continuously. In Fig.3.3, various delays was introduced in the GPS based trajectory data receiver design simulation, delay of 60ms is used for transmitter buffer (TX_BUFF) while 10,100,1000ms for LCD_display. Also delay of 10ms was introduced before the receiver (RX).

Desired trajectory data are extracted from the strings of data received continuously every second from GPS and displayed on LCD.

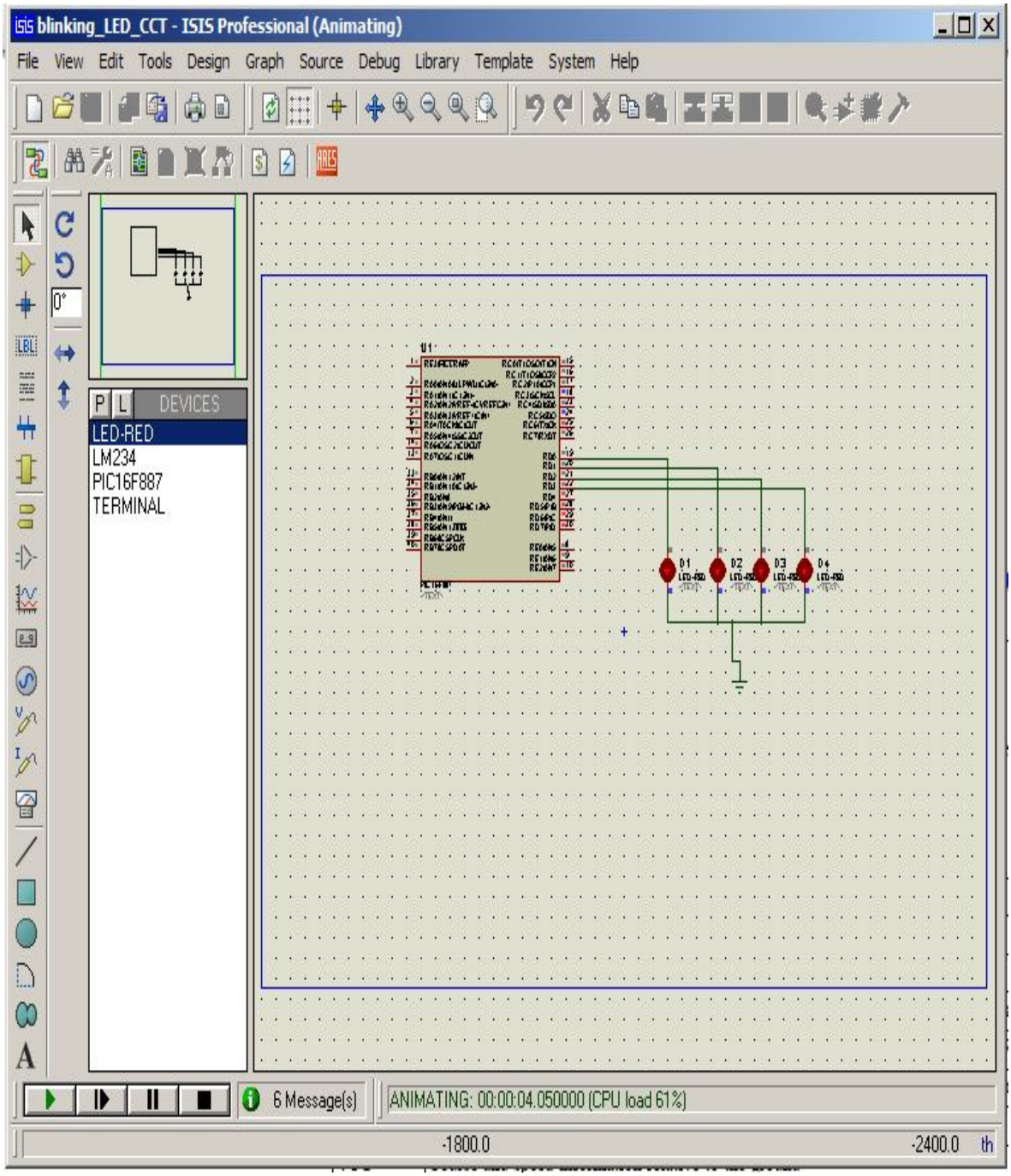


Fig. 3.2: Simulation of PIC16F887A Microcontroller using Proteus 7 professional

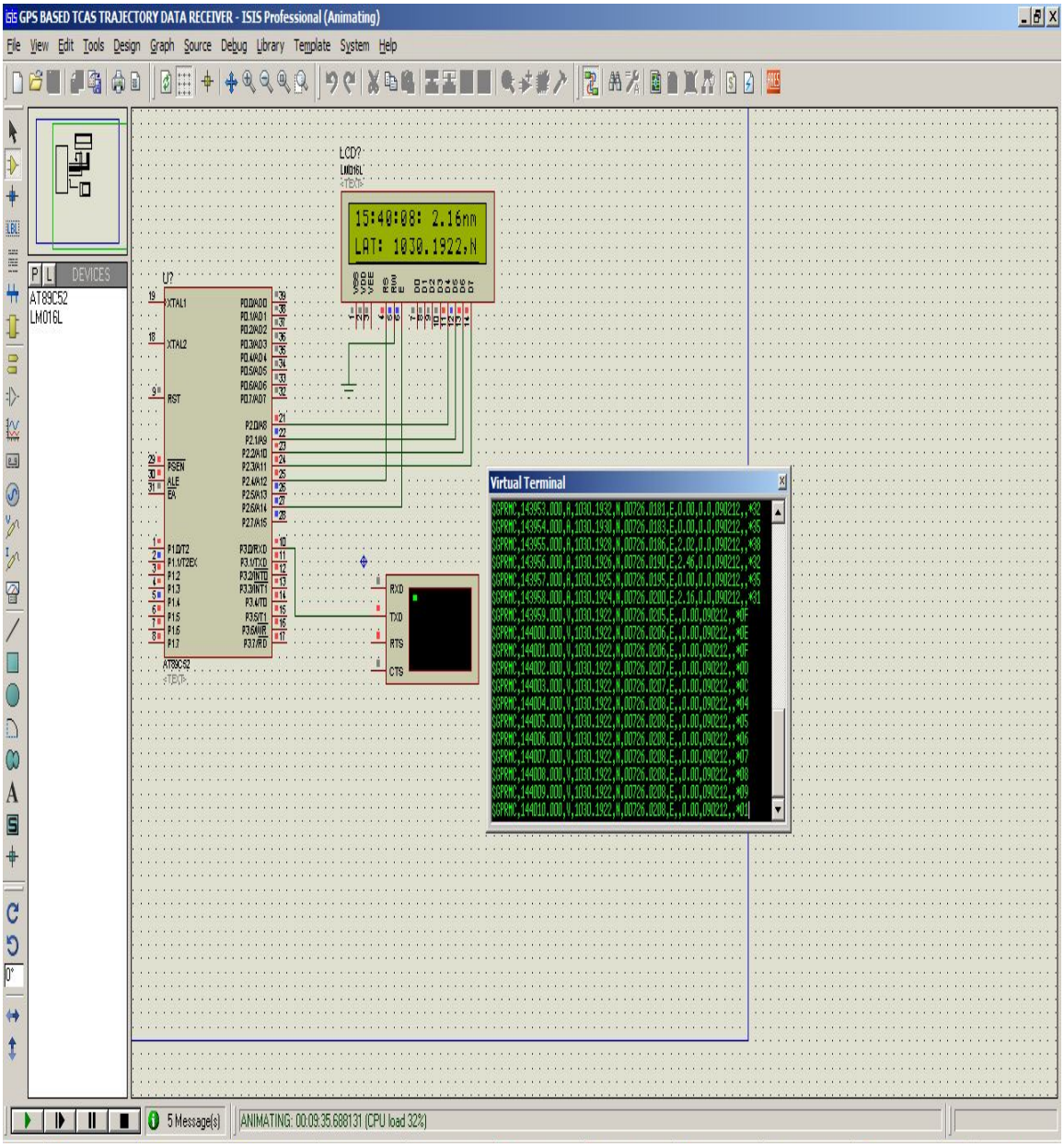


Fig. 3.3: Simulation of GPS Based Trajectory Data Receiver

3.3.2.3 Experimental results of Simulating Microcontroller

The experimental results of simulating our model using Proteus 7 professional is shown in Table 3.1. In the experiment, delay of different values was introduced and the clock frequency of PIC16F887A Microcontroller was varied to see the maximum usage of Central Processing Unit (CPU), memory and the speed while the execution time was constant for 30s except where Real Time was not feasible. In the first run for different value of the Clock frequency, the system could not operate in a Real Time because the maximum capacity of CPU was being used and the speed was extremely fast. For instance, when the Clock frequency was 5MHz, the CPU usage was 100% and the speed was very vey fast. But when the delay of 100ms was introduced, running for 30s, the CPU usage dropped to 13% and the speed was moderately fast. Also when the clock frequency was 45MHz, the CPU usage was 100% and the speed was very vey fast. But when the delay of 100ms was introduced, running for 30s, the CPU usage dropped to 93% and still, the speed was very vey fast. See section 3.3.2.4 for discussion on simulating our model.

3.3.2.4 Discussion on Simulating Microcontroller

The experiment on the simulation of Microcontroller was carried out and the findings are discussed in this section.

In the findings, a combination of 15MHz Clock frequency and 500ms delay gave 2% of memory consumption, 60% of CPU usage and moderate output speed.

In the design, clock frequency of 15MHz was used which is within the standard range of 0-60MHz together with delays of 10, 60, 100 and 1000ms between the Tx /Rx and display, these gave required output speed timely.

During the simulation, we were able to find out that there was a component called crystal that clocked the microcontroller. Crystal determines the speed of the microcontroller, the higher the value of the crystal, the faster the speed of the microcontroller but consequently with corresponding increase in the power consumption and environmental factors effects such as mechanical vibration and temperature. But increase in power consumption is not good for embedded systems design because most of them run on battery, if the power consumption is very high, battery will run down quickly.

In order to reduce the power consumption, codesign may be implemented during modelling by introducing delay into the program code of the microcontroller.

We found out that time delay would be minimal in modelling of a GPS based system because the information about current time, date, position and many more are displayed continuously every second from GPS (Global Positioning System) satellite. Also we found out that the time element is relative to GPS clock. The time is displayed continuously and additive as object changes locations or positions.

We received $GPRMC (Recommended minimum specific GPS/Transit data) form GPS in strings form because, depending on the satellite and desire purpose, data received form GPS are in strings form and could come in any of the data type format such as $GPGGA (Global Positioning System Fix Data), $GPGSV (GPS satellites in view), $GPGSA (GPS DOP (Dilution Of Precision) and Active Satellites).

From our experiment for instance, we received strings of information below which contained desired TCAS trajectory data continuously from GPS;

$GPRMC,220516,A,5133.82,N,00042.24,W,173.8,231.8,130694,004.2,W*70

$GPRMC,142227.000,A,1030.2167,N,00726.0835,E,0.00,0.0,090212,,*36

- $GPRMC Message ID

- 220516 Time Stamp
- A validity - A-ok, V-invalid
- 5133.82 current Latitude
- N North/South
- 00042.24 current Longitude
- W East/West
- 173.8 Speed in knots
- 231.8 True course
- 130611 Date Stamp
- 004.2 Variation
- W East/West
- *70 checksum

From the data received in $GPRMC format of GNSS (Global Navigation Satellite System), in our experiment we obtained the following trajectory data require for coordination of TCAS :

- current time from GPS clock
- speed which is the cruising speed of the object and;
- current location from latitude and longitude

Our findings also showed that delay and clock frequency have effect on the execution time of a system. We used the delay as guard and invariant clock constraints relative to the location or state in the modelling and performance analysis of our TCAS system design. The bounds on the constraints determine execution time of the system. The clock constraints are used as enabling conditions of transitions which must be satisfied before transition is enabled, (section 3.5).

Require trajectory data are received continuously from GPS Based Trajectory Data Receiver.

Based on our findings we are able to implement GPS in the modelling and performance analysis of our TCAS system design.

Table 3.1: Experimental results of simulating PIC16F887A Microcontroller

		Time	Memory	Max CPU Usage	Output Speed
Delay	**Clock Frequency**				
No delay	5MHz	Not Real Time	1%	100%	5
100ms	5MHz	30s	2%	13%	3
250ms	5MHz	30s	2%	15%	2
500ms	5MHz	30s	2%	18%	1
No delay	15MHz	Not Real Time	1%	100%	5
100ms	15MHz	30s	2%	53%	3
250ms	15MHz	30s	2%	55%	3
500ms	15MHz	30s	2%	60%	4
No delay	30MHz	Not Real Time	1%	85%	6
100ms	30MHz	30s	2%	85%	5
250ms	30MHz	30s	2%	81%	3
500ms	30MHz	30s	2%	80%	4
No delay	45MHz	Not Real Time	1%	100%	6
100ms	45MHz	30s	2%	93%	5
250ms	45MHz	30s	2%	99%	5
500ms	45MHz	30s	2%	99%	5

Legend:

Very Slow : 1
Slow : 2
Fast : 3
Very Fast : 4
Very very fast : 5
Extremely fast : 6

3.4 Modelling and Performance Analysis of GPS based TCAS using 'UPPAAL'

The Traffic alert and Collision Avoidance System (TCAS), a real-time embedded system, is an aircraft collision avoidance system designed to reduce the incidence of mid-air collisions between aircrafts. It monitors the airspace around an aircraft for other aircraft equipped with a corresponding active transponder, independent of air traffic control, and warns pilots of the presence of other transponder-equipped aircraft which may present a threat of Mid-Air Collision (MAC). Its implementation is mandated by International Civil Aviation Organization (ICAO) to be fitted to all aircraft with MTOM (Maximum Take-Off Mass) over 5,700kg or authorized to carry more than 19 passengers. TCAS detects and displays aircraft in the immediate vicinity. TCAS-equipped aircraft "interrogates" all other aircrafts in a determined range about their position (via the 1030 MHz radio frequency), and all other aircrafts reply to other interrogations (via 1090 MHz). This interrogation-and-response cycle may occur several times per second. Through this constant back-and-forth communication, the TCAS system builds a three dimensional map of aircraft in the airspace, incorporating their bearing, altitude and range. Then, by extrapolating current range and altitude difference to anticipate future values, it determines if a potential collision threat exists. Then it automatically negotiates mutual avoidance manoeuvre between the two or more conflicting aircrafts. Manoeuvres in current TCAS system are restricted to changes in altitude and modification of climb/sink rates, that is vertical plane only (Lufthansa, 2005). There may be instances where we can make use of lateral resolution manoeuvres which is addressed in this research work when we have complex traffic conflict scenarios between aircrafts taking off or landing.

The first generation of collision avoidance technology is TCAS I. It is cheaper but less capable than the modern TCAS II system. TCAS I could generate collision warnings in the form of a "Traffic Advisory" (TA). The TA warns the pilot that another aircraft is in near vicinity, announcing "traffic, traffic", but unlike in TCAS II it could not offer any suggested remedy; it is up to the pilot to decide what to do, usually with the assistance of Air Traffic Controller (http://adsb.tc.faa.gov, 2010). When a threat has passed, the system announces "clear of conflict".

As of 2006, the only implementation that meets the TCAS II standards set by ICAO is Version 7.0 of TCAS II (Alexandre and Claire, 2004), produced by three avionics manufacturers: Rockwell Collins, Honeywell, and ACSS (Aviation Communication & Surveillance Systems; an L-3 Communications and Thales Avionics company). TCAS II offers all the benefits of TCAS I, but also offers the pilot direct "Resolution Advisory" (RA) to avoid danger (Lufthansa, 2005). The suggestive action maybe "corrective", suggesting the pilot changes vertical speed by announcing, "descend, descend", "climb, climb" or "Adjust Vertical Speed Adjust" (meaning reduce or increase vertical speed) (www.eurocontrol, 2010).

One of the related works is TCAS III developed by Honeywell which is expected to be the "next generation" of collision avoidance technology.

TCAS III like our proposed TCAS is expected to incorporate technical upgrades to the TCAS II system, allow reverse logic and increase total separation between aircrafts both in horizontal and vertical aspects but the work was suspended due to limitations in the accuracy of the TCAS directional antennas. Also there are technological integration and cost effectiveness hurdles that preclude the introduction of this system (www.airliners, 2011).

In this work, 'UPPAAL', a tool for performance analysis of Real-time embedded systems is used for modelling and analysis of our proposed GPS based TCAS with the following contributions:

a. Introduction of horizontal resolution directives which are useful in a conflict between two aircrafts close to the ground where there may be little, if any, vertical manoeuvring space possible.
b. Introduction of traffic situation given within a "time-based" representation because the existing TCAS is range-based that displays the traffic situation within a configurable ranges of miles/feet, however under certain condition there may be need for "time-based" representation.
c. Introduction of Reversal logic where an RA issued by TCAS is reversed when it has been shown that after few seconds one of the two aircrafts does not comply with the advisory given earlier by the TCAS.

Performance analysis of real-time embedded system plays an important role in the design process of complex embedded systems for analyzing essential performance characteristics of system design at an early phase (section 1.3). Important design decisions are made earlier before much time and resources are invested in detailed design implementation and that is why we carried out some experiments earlier (see section 3.4), from where we found out that GPS can be implemented in our TCAS, RTE systems design.

In the past few years, a number of modelling and verification tools for real-time embedded systems have been developed based on the theory of timed automata. They have been successfully applied in various case studies. However, the tools have been mainly used in the academic community, namely by the tool developers. It has been a

great challenge to apply these tools to real situation case-studies that are useful for industrial development.

In this section, we present the use of UPPAAL in the design and analysis of a GPS based TCAS. This, to our knowledge, is the first attempt to use UPPAAL in the design of a TCAS.

We give a detailed description of the formal model of the TCAS with its surrounding environment, and its correctness formalised according to the informal requirements given by aircraft pilot.

3.4.1 System Requirements

In this section, the informal requirements of TCAS are identified. The TCAS is capable of tracking up to a combined total of 30 intruders. Tracking is performed by repetitive TCAS interrogations.

Measurement of Intruder Parameter:

Upon confirmed transponder reception, the TCAS starts to interrogate the intruder. Its altitude is transmitted directly in the reply and this information is used to determine the relative altitude of the two aircrafts, by calculating the barometric altitude difference.

Range measurement:

The range is calculated by measuring the elapsed time between transmission of the interrogation signal and the return of the reply transmitted by the intruder. Aircrafts are detected from the range of at least 14 NM.

Principle of Computation

In TCAS, target aircrafts are categorized depending on specific criteria varying in function of altitude. TCAS essentially uses two types of information to perform this classification: The relative altitude between two aircrafts, known by the difference of

their standard barometric altitudes and the distance or range separating them. Acquisition of these two parameters at regular intervals (tracking) enables their variations to be calculated, that is the altitude rate and range rate.

Assessment of the potential threat represented by an intruder depends on two criteria determined with respect to a point in the traffic area called Closet Point of Approach (CPA). This is the point of minimum distance between the two aircrafts, assuming that their trajectories do not deviate. The two criteria are vertical separation at CPA and time left before reaching CPA. The threat is evaluated by calculating the vertical separation between the two aircrafts at the Closet Point of Approach.

Time to intercept (TAU)

There is an area defined as the TAU (τ) within the surveillance arc, which represents the minimum time the flight crew needs to discern a collision threat and take evasive action. The TCAS does not need to locate the CPA in space, but rather it needs to know the time to intercept for two aircrafts. For example, if two aircrafts are approaching on the same axis on a collision course, this time is the ratio of distance between them to the sum of their speeds.

TCAS uses range and range rate measurement to compute this time (Lufthansa, 2005):

$$\text{TAU}\ (\tau) = \frac{RANGE}{RANGE\ RATE} \quad \text{TAU (s)} = \frac{R\ (NM)}{RR\ (Kts)}$$

With the risk of collision being in inverse proportion to this time, trajectory correction orders are initiated by crossing predetermined time thresholds whose values depend on the altitude layer in which the aircraft is located.

3.4.2 System Functionality

In this section, the functionality of the TCAS is formally described, abstract the behaviour of the aircraft equipped TCAS in the environment based on the desired

informal specification requirements identified in section 3.4.1 and finally model the system.

TCAS Principle of Operation

TCAS calculates time between own airplane and the target. TCAS then calculates how close the target will be to own airplane at the closet point of approach (CPA).

Evasive action is given against a collision threat by close monitoring of time it will take until CPA.

The first aircraft that detect a likely dangerous situation computes a deviation manoeuvre sense and communicates it to other aircraft within a certain time. This aircraft takes the information into account and in turn computes preventive or corrective advisories as the case may be.

Formal performance requirement:

Performance data of existing TCAS (FAA, 2000) is shown in Table 3.2.

Table 3.2: Performance data of existing TCAS

Altitude (Feet)	Tau (Seconds) for existing TCAS	
	TA	RA
<1000	20	N/A
1000 – 3000	25	15
5000 - 10000	40	25
10000 - 20000	45	30
20000 - 42000	48	35

The formal performance requirements of our model are specified as follow:

(a) The intruders should be detected within 60 seconds

(b) The range of intruders should be acquired within 50 seconds

(c) Traffic advisory should be given within 45 seconds if intruders range is within Closet Point of Approach (CPA)

(d) Intruder should be declared as a threat within 35 seconds if intruders range is considered dangerous at the Closet Point of Approach (CPA)

(e) Preventive resolution advisory should be given within 35 seconds if intruders range is considered dangerous at the Closet Point of Approach (CPA). Here the vertical separation is less than upper preventive advisory threshold but greater than upper corrective advisory threshold.

(f) Corrective resolution advisory should be given within 30 seconds if intruders range is considered dangerous at the Closet Point of Approach (CPA). In this case the vertical separation is lower than the upper corrective advisory threshold.

(g) TCAS should be able to indicate clear of traffic or traffic not clear within 5 seconds if conflict is resolved or otherwise.

(h) If conflict is not resolved within 5 seconds, TCAS should be able to perform reversal logic by carrying out the following steps:

(i) TCAS should be able to acquire the new range for the intruders within 5 seconds.

(j) TCAS should do step (f) and (g).

Predictability:

The predictability requirements are to ensure strict synchronisation and control between aircrafts equipped with TCAS.

(a) There should be no deadlocks in the system.

(b) TCAS of all the aircrafts during tracking should be functional

Error detection:

The TCAS detects and indicates error when:

(a) Intruder is detected but range and bearing could not be determined

(b) Appropriate advisory is not given timely

(c) Sensitivity level prevents intruder to comply with the advisory

(d) Collision threat is still detected after advisory if one of the intruders does not comply with TCAS.

Existing TCAS:

In the existing TCAS, see Fig.3.4, nodes in the flow graph of the model represent automata and edges represent synchronisation channels or shared variables. The components have two channels associated with each service, one for service request and one to respond when the service has been performed.

The existing TCAS is limited to supporting only vertical separation advisories but more complex traffic conflict scenarios may however be more easily and efficiently remedied by also making use of lateral resolution manoeuvres.

TCAS equipment today is often primarily range-based, as such it only displays the traffic situation within a configurable range of miles/feet, however under certain circumstances a "time-based" representation (that is, within the next certain minutes or seconds) might be more intuitive.

Today's TCAS displays do not provide information about resolution advisories issued to other (conflicting) aircraft. Our proposed TCAS proffers solution to some of these limitations.

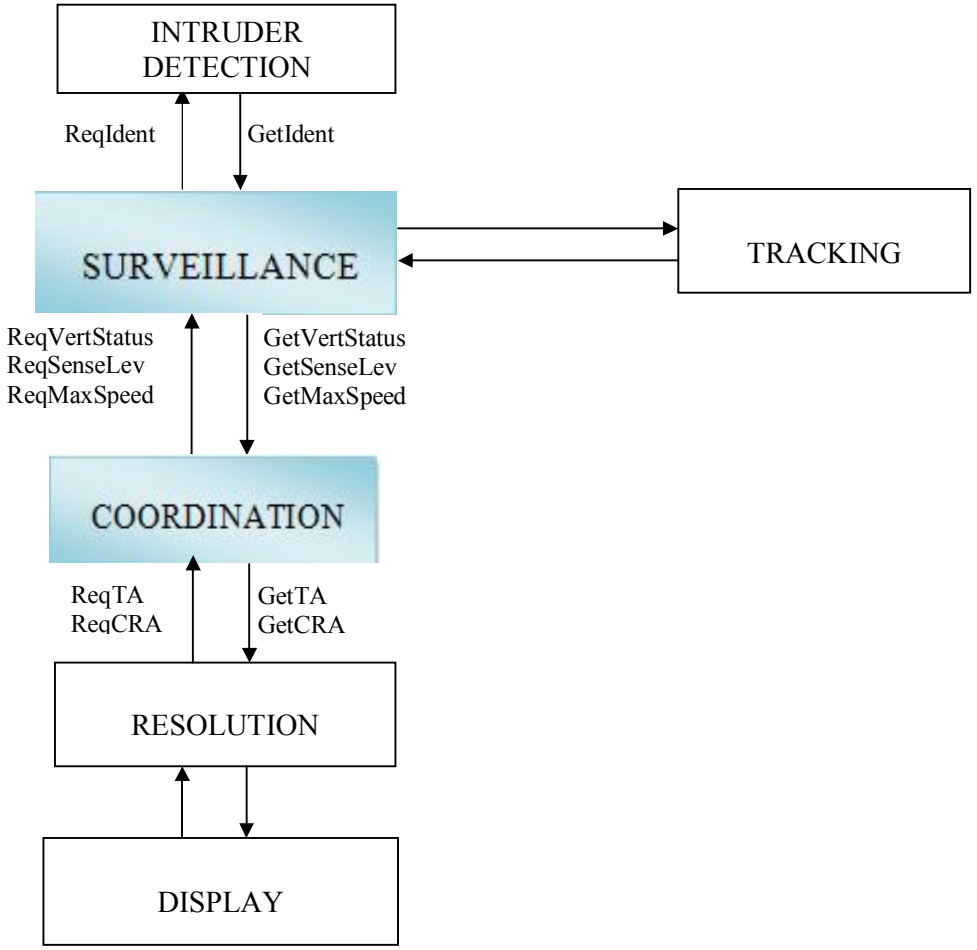


Fig. 3.4: A flow graph of existing TCAS

3.4.3 Formal Modelling of Proposed TCAS:

Generally takeoff of an aircraft is a phase of flight in which an aircraft goes through a transition from moving along the ground (taxing) to flying in the air, which normally starts from the runway. Thrust is the force required that make the aircraft speed up along the runway (www.fi.edu, 2010). Some aircrafts specifically designed for short takeoff and landing can take off at speeds below 40 knots (74 km/h), and can even become airborne from a standing start when pointed into a sufficiently strong wind (www.free-online-private-pilot-ground-school.com, 2010).

In the existing TCAS, if an aircraft is taking off and another one is approaching to land and is below the altitude of 1450ft on the same runway, although the TCAS on these two aircrafts can detect each other but vertical resolution cannot be issued for the two aircrafts, because naturally, the aircraft that is approaching to land has lost power during descent, the spoilers increase the drag force and slow down the aircraft. Issuing climb command of vertical resolution to the pilot at this moment could be hazardous to the aircraft because at that stage full power cannot be gained instantaneously, in the attempt the aircraft could stall.

For the aircraft that is taking off, if it is estimated to be within 400ft closer to the ground and is climbing at say 15^0 or 45^0, it will be difficult for such an aircraft to suddenly climb at 90^0 or higher or descend immediately to 270^0 in compliance with vertical resolution, the aircraft may overshoot the runway or crash land. As a result of this, lateral resolution is the only way out, where aircraft can be advised to yaw to the left or to the right while landing or taking off as the case may be.

In our proposed TCAS, we assumed that the TCAS is equipped with GPS and pilots of the two aircrafts will obey TCAS advisories. TCAS components are designed to be

functionally correct in the given environment. The existing TCAS in Fig.3.4 is extended. The automata Surveillance and Coordination of the existing TCAS are modified to include lateral resolution manoeuvres. Reversal automaton is also added to our proposed model to enabling reverse logic. See Fig.3.5 for the changes.

TCAS performance components are modelled using UPPAAL by the automata SURVEILLANCE, COORDINATION, RESOLUTION and DISPLAY. The environment is modelled by automata INTRUDER DETECTION, and TRACKING.

To enable formalisation and verification of these requirements and ensure modularity, each of the automata is verified separately before they are integrated. We achieved the integration by using broadcast synchronisation, where sender **chan**! synchronises with an arbitrary number of receivers **chan**?. In our model, the *Ident!* channel of automaton INTRUDER DETECTION synchronises with *Ident?* of SURVEILLANCE automaton. The *VertStatus!, SenseLev!, MaxSpeed!* and *MinTimeCPA!* of automaton TRACKING synchronise with *VertStatus?, SenseLev?, MaxSpeed?* of automaton SURVEILLANCE *and MinTimeCPA?* of automaton COORDINATION respectively. Channels *TA!* and *CRA!* of automaton COORDINATION synchronise with channels *TA!* and *CRA!* of automaton RESOLUTION.

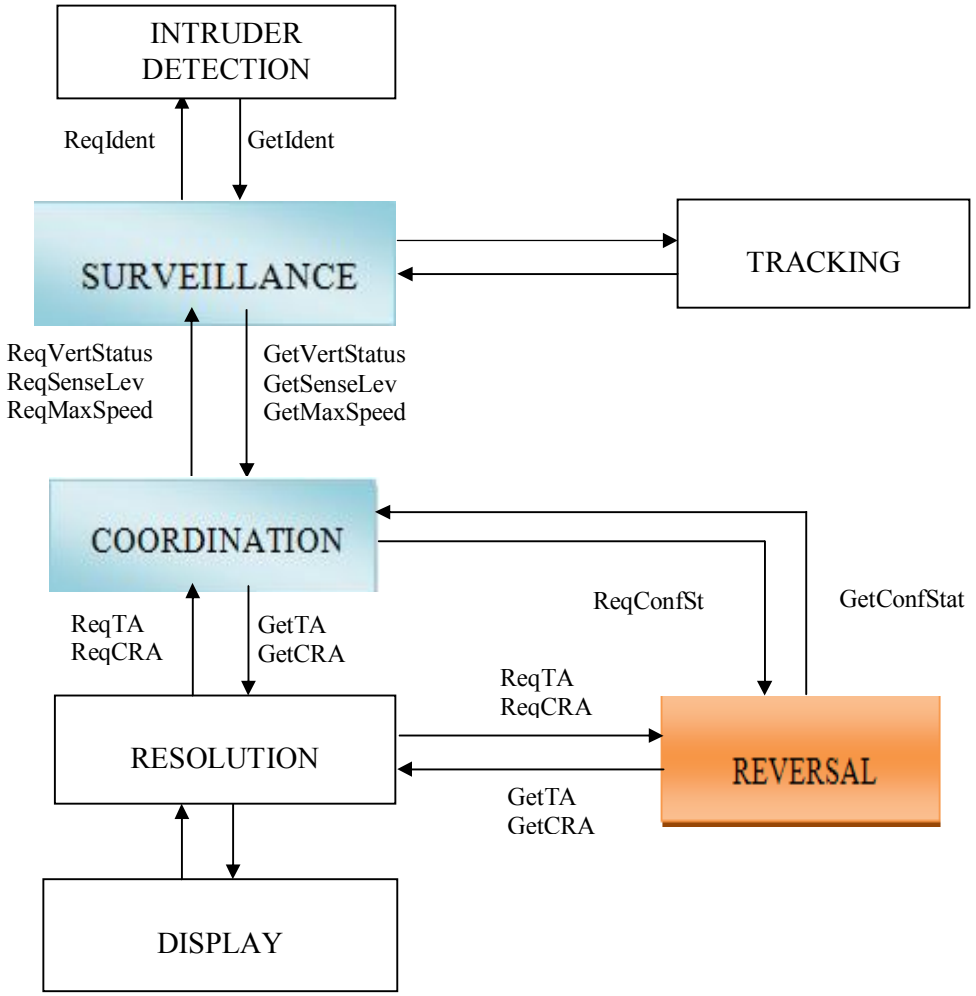


Fig. 3.5: A flow graph of Proposed TCAS

Intruder Detection:

In automaton INTRUDER DETECTION (Fig.3.6), aircraft equipped with TCAS receives messages transmitted by the intruder. Automaton continues broadcast at location activeBrodcast if x>=50 and aircraft is actually detected at location ACdetectd if x<=50. An urgent location Identf is entered if x<=45 and *Ident!* channel synchronises with *Ident?* of SURVEILLANCE automaton.

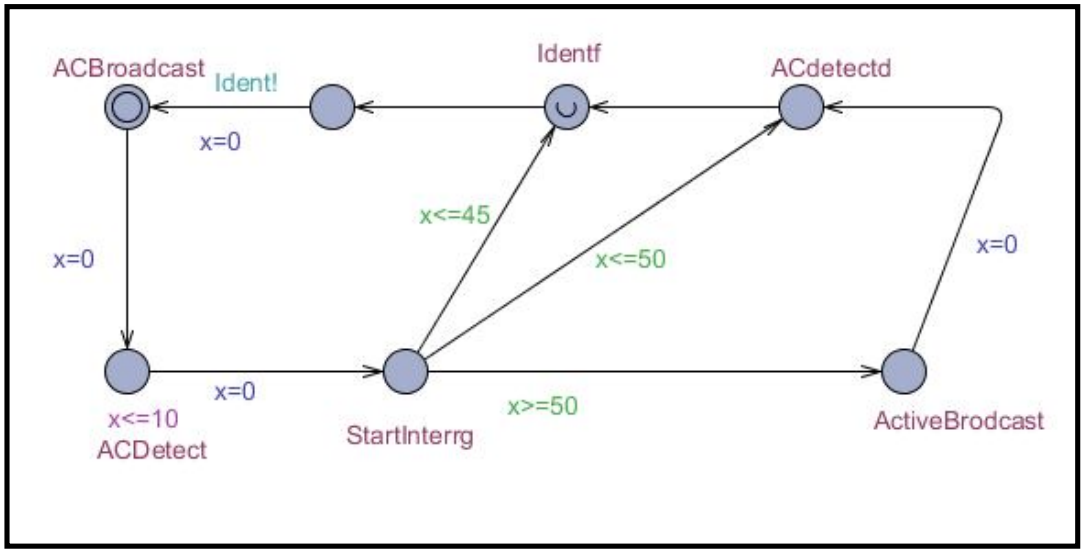


Fig. 3.6: The INTRUDER DETECTION Automaton

Surveillance:

The target aircraft identification is acquired in this automaton, the request services *VertStatus?, SenseLev?* and *MaxSpeed?* received during tracking as requested by COORDINATION automaton are responded to accordingly (Fig.3.7).

Transition is from start to initialise, IdentAcqd and status location after the corresponding guard or invariant constraints are satisfied. The status determines if the location is at ACground or ACairborne based on the clock constraints.

If the location is at *ACground* the transition is to *Speed* location instead of *start* location as modelled in the existing TCAS where there is no TCAS resolution between aircrafts on the ground and airborne. The transition then moves to *Aclevel*, *CurtRange* and finally to *start* of the automaton.

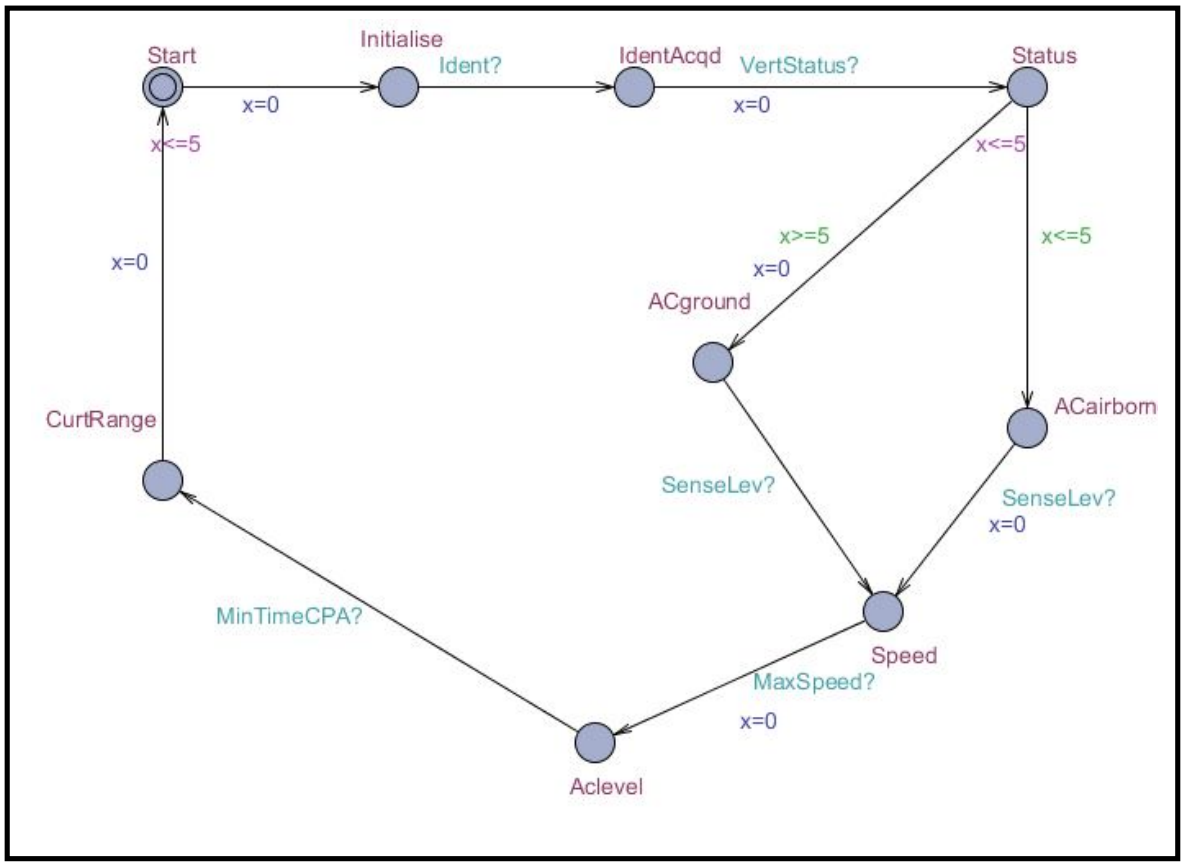


Fig. 3.7: The new SURVEILLANCE Automaton

Tracking:

Tracking is performed by repetitive TCAS interrogations, in the aircraft equipped with transponder; the TCAS is active and transmits pulses.

The aircraft under SURVEILLANCE automaton is tracked within 5 seconds and gives get services output, that is, *VertStatus!, SenseLev!* and *MaxSpeed!* (see Fig.3.8). After an aircraft is identified, TCAS establishes communication with the intruder and a delay

condition for a maximum of 10 clock constraints before *VertStatus!* channel is enabled and the transition enters location *Status*. Then transition moves from location Speed to TimesepReq location and back to StartTrakg location of automaton TRACKING.

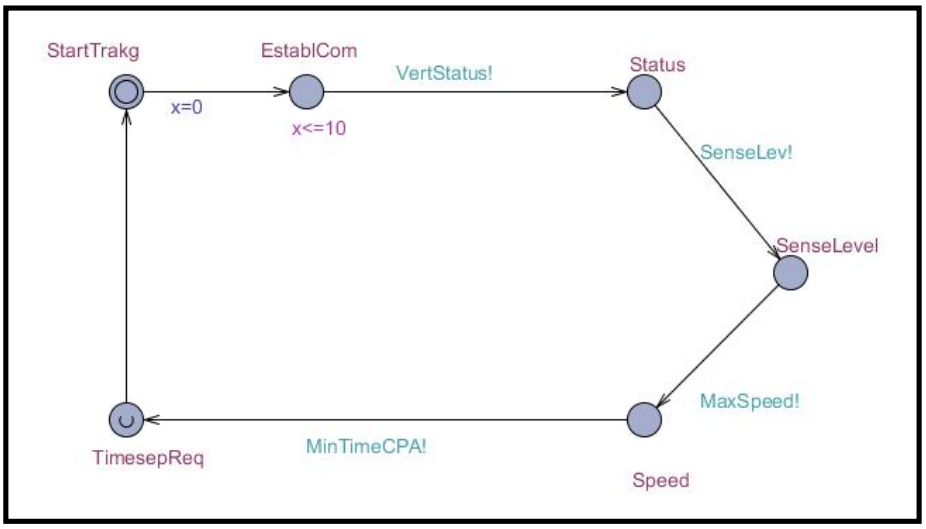


Fig. 3.8: The TRACKING Automaton

Coordination:

In the COORDINATION automaton, the transition is through the following locations; *AcBroadcast, ThreatDetc, UpperTAThrsd, TAdv, ThreatDetecd, ExcedUperTAThrsd, ThreatDeclad, ACAirbornST,* to *ACGroundST* or *ThreatDetAf* to *CPA, ConflctResd, DispTrafficClerd* and finally to *AcBroadcast* after satisfying the guard or invariant constraints on the edges or locations or to *Reverse, MinTime* and start the resolution all over again.

TCAS checks the vertical status of the aircraft, its sensitivity level and maximum speed of the two aircrafts before giving any advisory procedure. Preventive Traffic Advisory is given by TCAS if the aircraft is at the threshold in *UpperTAThrsd* location.

Like in the coordination automaton of the existing TCAS, a threat is detected within 32 seconds if traffic advisory given earlier could not resolve the sensed head-on collision. Transition moves to *ExcedUperTAThrsd* location, if threat is detected again, within a second transition enters location *ThreatDeclad*, resolution phase is entered where vertical Corrective Resolution Advisories (CRA) is given within 25 seconds if the aircraft that is air borne is very close to location CPA (see Fig 3.9).

Also, horizontal Corrective Resolution Advisories (CRA) is issued within 30 seconds if the ground aircraft is very close to *CPA*. Every 5 seconds, the status of the separated aircrafts is confirmed, if threat is still detected as a result of non compliance by the pilot of a particular aircraft, Reversal phase is enabled.

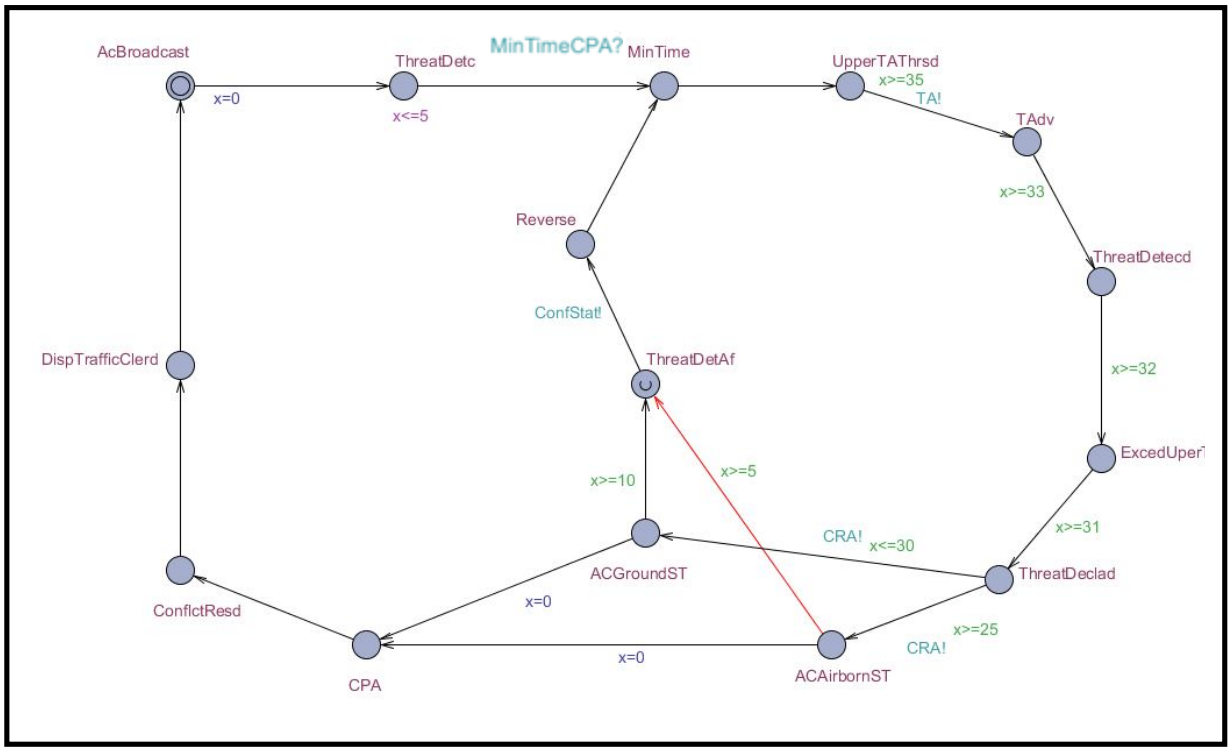


Fig. 3.9: The new COORDINATION Automaton

Resolution:

When the target intruder is relatively near but does not represent an immediate threat, in that case the target is between the upper traffic advisory threshold and upper preventive traffic advisory threshold, the TCAS issues a preventive traffic advisory, (see Fig.3.10) the display will indicate traffic**.** The range and relative bearing of the aircraft is known to the pilot as well as time TAU before CPA value. When the system is updated, now depending on the current trajectory, the target intruder may conserve this status and move away or it may become a collision threat, in that case the target is between the upper corrective advisory threshold and CPA. As a result of this, collision is possible, avoidance manoeuvres are issued immediately to the pilot via a corrective resolution advisory (Fig.3.11).

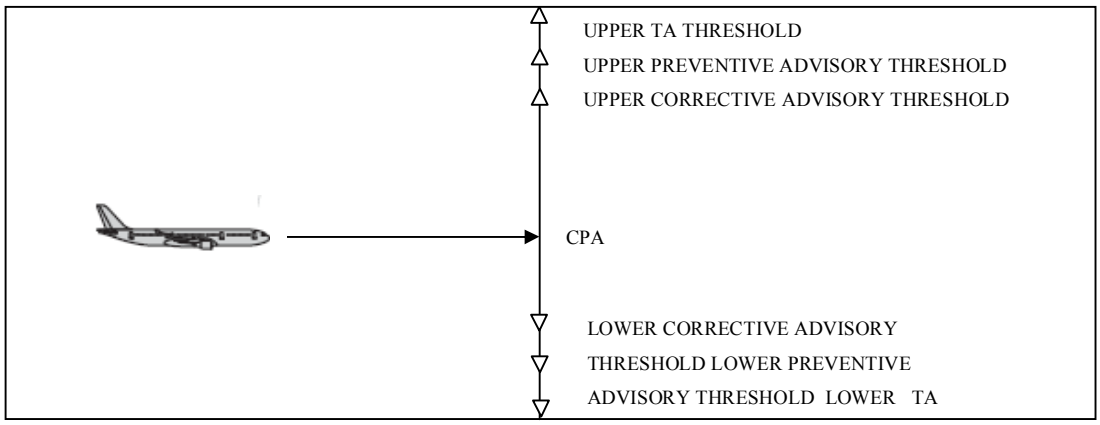


Fig. 3.10: TCAS Corrective / Preventive Advisory Threshold

The transition in this automaton is from *start* to *EstablCom* location, transition is enabled to location *TAAdv* or *CRAReq*, the edges *TA?* and *CRA?* of *TAAdv* and *CRAdv* synchronise with that of automaton COORDINATION. If corrective resolution is successful, the urgent location *DispTrafficClerd* is entered to display no traffic.

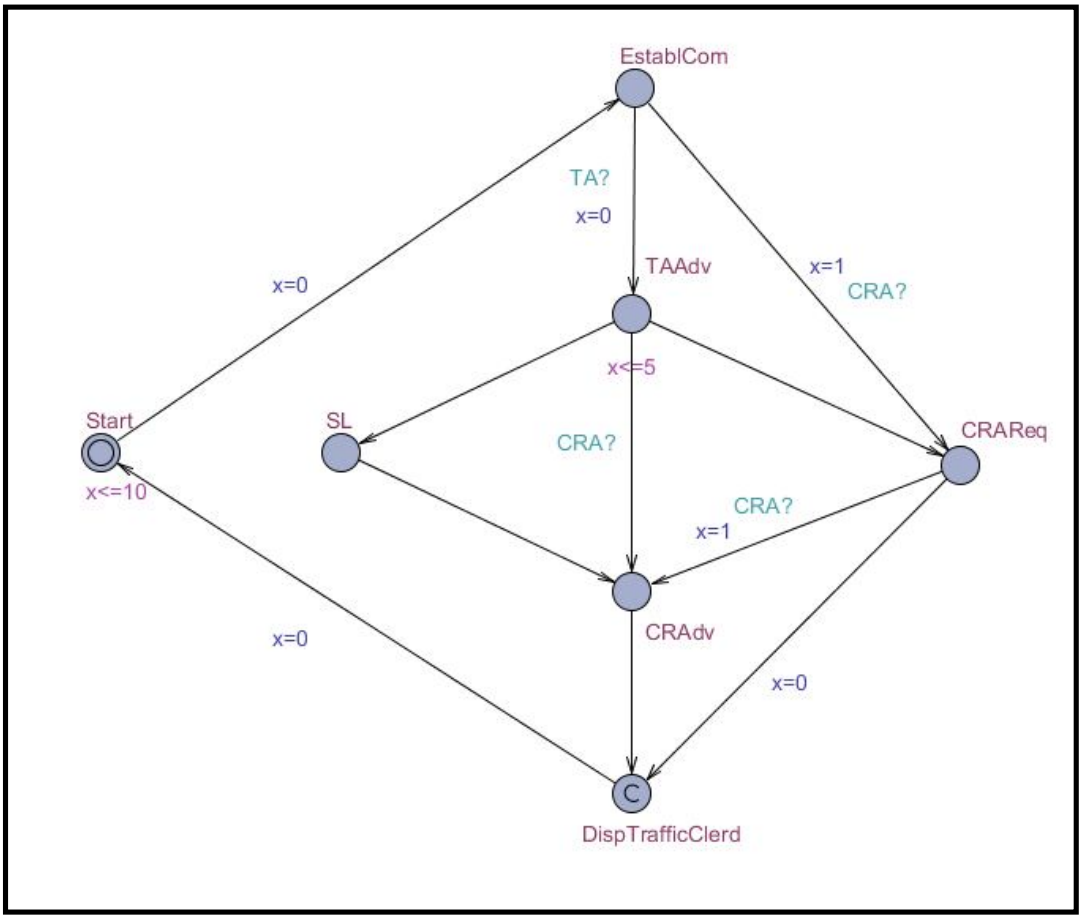


Fig. 3.11: The RESOLUTION Automaton

Reversal:

In REVERSAL automaton, the system checks to confirm Corrective Resolution Advisory (CRA) compliance by the targets every 5 seconds (Fig.3.12). When it is detected that an aircraft is not responding correctly to an RA, the process enters urgent location and reversal phase is triggered automatically.

The vertical separation at the closest point of approach based on current vertical speeds of the aircraft are probably going to be closer than a predefined upper threshold of TA, then the CRA is issued based on the current acquired range.

Reversal RAs are not triggered earlier than 5 seconds to leave time for the initial RAs to be efficient before reversing.

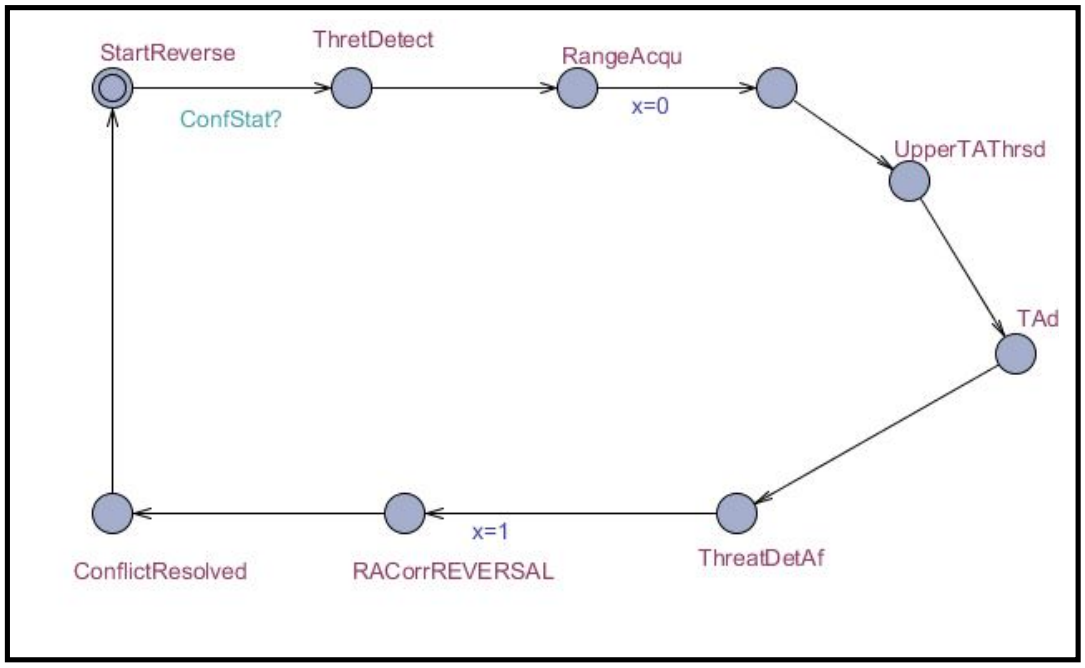


Fig. 3.12: The new REVERSAL Automaton

3.5 Validation and Verification of Results

In this section, we show the formal requirements given in section 3.4.2 and ascertain their correctness using the symbolic model-checker of UPPAAL to check the state and path formulae for their reachability, safety and liveness. In these properties, we check whether a given state formula, , possibly can be satisfied by any reachable state that is, given a path starting at the initial state, we find out whether such is eventually satisfied along that path and safely within a given time bound.

The first performance requirement is that the intruder must detect and identify itself within 60 seconds. We show that it requires the location *ACDetect* to be reached in automaton INTRUDER DETECTION if clock <=50 and urgent location *Identf is*

entered if the time is less than or equal 45 seconds. Also, error should occur if the intruder fails to identify itself.

It was verified that after the intruder has been identified, automaton SURVEILLANCE initialised interrogation to get required parameters. Transition is not delayed at location *status* to know whether the intruder is on the ground or airborne. We verified that if location *status* of automaton SURVEILLANCE is reached, invariably location *speed* will be reached. Property 21 requires that preventive traffic advisory should be given within 35 seconds if the detected intruder wants to exceed threshold. The corrective advisory both in vertical or horizontal resolution is given to the target aircraft within 25 and 30 seconds as soon as the process enters *ACAirbornST* and *ACGroundST* state in properties 22 and 23 respectively. (See appendix A for the list of some of the verified properties).

In properties 30 and 31 we confirm that *Conflict* Status of the resolution given to the intruder is checked every 5 seconds to ensure compliance. If target aircraft failed to implement recommended manoeuvre, the corrective resolution is reversed within 10 seconds. The time bound given for this operation is between 5 and 10 seconds. We verified in property (44) that invariantly there is no deadlock in the system. See appendix A for the list of some of the verified properties.

We have been able to verify over fourty various properties of the system using UPPAAL. It was verified that lateral resolution manoeuvres could be issued to TCAS equipped aircraft approaching to land and the other one taking off, because issuing climb command of vertical resolution to the pilot descending could be hazardous to the aircraft because at that stage full power cannot be gained instantaneously, in the attempt the aircraft could stall. So also the aircraft taking off could not be asked to

suddenly climb higher or descend immediately in compliance with vertical resolution, during the process, the aircraft may overshoot the runway or crash land.

Also we verified that our system is dynamic because with the feedback from *'ThreatDetAf'* state which depends on other states, Reversal logic is automatically enabled in a situation where it is obvious that one of the two aircrafts failed to comply with the resolution advisory given by the TCAS.

CHAPTER FOUR

DESIGN OF TCAS REAL TIME EMBEDDED SYSTEM PROTOTYPE, RESULT AND DISCUSSION

4.1 Introduction

The major components of Traffic alert and Collision Avoidance System (TCAS) are TCAS control panel, Microcontroller (TCAS Computer) and mode S (Select) transponder with two directional antennas, one on top and the other one below that monitor the airspace around an aircraft for other aircraft equipped with a corresponding active transponder. TCAS warns pilots of the presence of other transponder-equipped aircraft which may present a threat of Mid-Air Collision (MAC). TCAS-equipped aircraft interrogates all other aircrafts in a determined range about their position (via the 1030 MHz radio frequency), and all other aircrafts reply to other interrogations (via 1090 MHz).

In the prototype design, GPS is incorporated into the existing TCAS design to enable avoidance manoeuvres between the two or more conflicting aircrafts not restricted to changes in altitude and modification of climb/sink rates, that is, vertical plane only as obtained in the existing TCAS system, but to allow lateral resolution manoeuvres especially when there is a complex traffic conflict scenarios between aircrafts taking off or landing. In this chapter, the prototype of GPS based TCAS modelled is designed and simulated using MATLAB to ascertain the correctness, integrity and safety of the system.

4.2 GPS based TCAS Design

A thorough investigation of specification requirements needed for TCAS design were obtained in section 3.4.1 and this was followed by a conceptual design phase which

resulted in abstract design specifications. These were validated for internal consistency and checked against the requirement specifications. This was achieved by formal model and verification in section 3.4.3 and 3.5 respectively.

In this section, TCAS modelled in 3.4.3 is designed and simulated for two aircrafts equipped with GPS based TCAS, using MATLAB to ascertain the correctness, integrity and safety of the system.

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment (David, 2005). MATLAB is a modern programming language environment with sophisticated data structures, which contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for teaching and research.

MATLAB has many advantages compared to conventional computer languages like C or FORTRAN for solving technical problems. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide. It has powerful built-in routines that enable a very wide variety of computations. It also has easy to use graphics commands that make the visualization of results immediately available. Specific applications are collected in packages referred to as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

In our design, GPS receiver is incorporated into the existing TCAS that is made up of TCAS control panel, transponder (Transmitter/Receiver) with two antennas, Microcontroller, TA and RA display as well as aural annunciation (Fig 4.1).

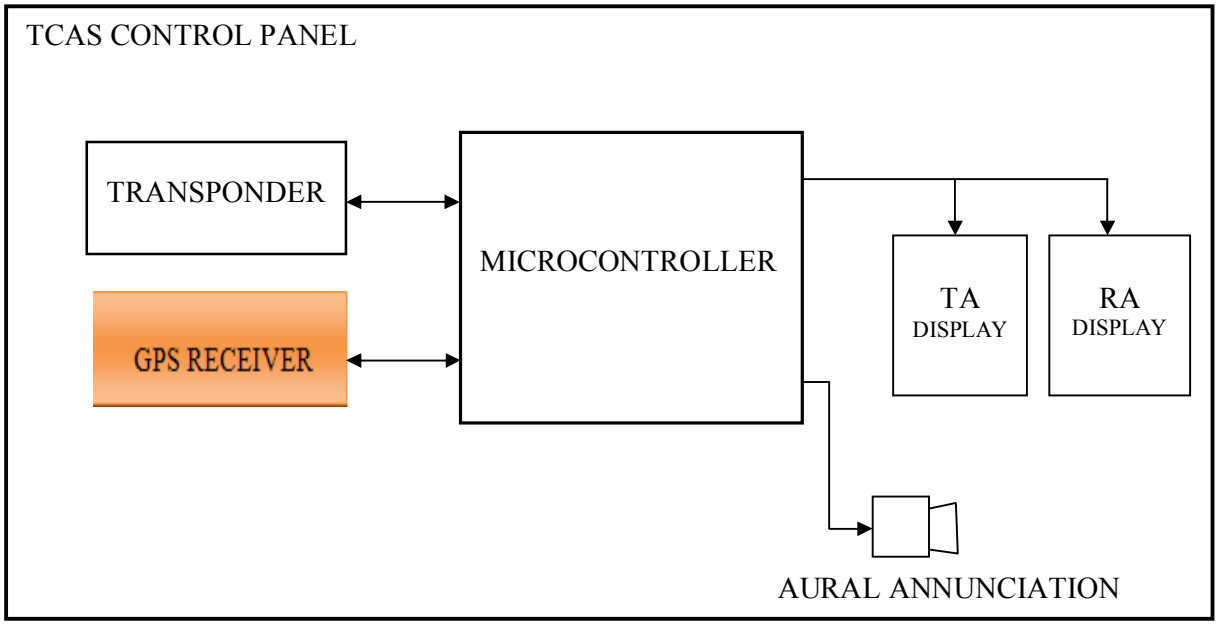


Fig. 4.1: GPS Based TCAS Prototype

Destinations and departure locations for the two aircrafts equipped with our GPS based TCAS, designated as; TCAS A and TCAS B are chosen by using Google Map to know their locations. Latitude and longitude for their destinations are obtained from the Google Map to take care of the function of the GPS receiver. TCAS A broadcasts and interrogates proximate TCAS B detected. TCAS B is under surveillance of TCAS A and is constantly tracked to receive trajectory data to determine a threat. TCAS A receives transmitted trajectory data of TCAS B continuously, such as identity, altitudes, bearing and other data required for TCAS coordination.

Microcontroller which is the main coordinator in TCAS, modelled in section 3.3.2.2 of chapter three will analyze the data and determine if the intruder airplane (that is TCAS B) is a threat to airplane (TCAS A). If there is a potential threat, while maintaining separation standard of 1000ft altitude, TCAS A will issue Traffic Advisory TA or

Resolution Advisories RA with aural annunciation that will instruct the pilot to avoid the conflict. If there is no threat, TCAS A will continue tracking the intruder airplane position. In our design, traffic situation is given within a time-based representation by monitoring and displaying approach time i.e Closet Point of Approach (CPA) continuously.

For the simulation purpose, the ranges of speed used are 250, 300, 350, 400km/hr. Distance apart is displayed continuously on TCAS server console. The distance between the two points is calculated using a MATLAB function, (MATLAB 2009):

dist = distance(*lat1,lon1,lat2,lon2*).

Where *lat1* and *lat2* represent latitude for point one and two respectively. And *lon1* and *lon2* represent longitude for point one and two respectively.

The simulated flight path is calibrated. Changes in the trajectory data is used to calculate the distance apart and consequently compute 'TAU', which is the time to the closet point of approach of the aircraft equipped TCAS using the formula stated in section 3.4.1. As the two aircrafts approach each other, the distance apart decreases and the time to CPA decreases as well. We simulated TCAS A and TCAS B in the same altitude mode for vertical resolution, in different altitude mode for lateral resolution and in reverse logic mode (Fig 4.2, Fig 4.3 and Fig 4.4).

4.3 Vertical Resolution Advisory

TCAS is simulated in the same altitude mode for vertical resolution. TCAS A departed a destination for example with the following trajectory data; longitude 7.433624, latitude 10.544521 and cruising speed of 250km/hr selected to a destination where another TCAS B was coming from, with 7.717896 longitude and 11.113727 latitude (Fig. 4.2). We assumed that TCAS B was equally going to where aircraft with TCAS A

departed and that the two TCAS equipped aircrafts were on the same altitude and in straight and level flight. The first performance requirement is that TCAS A must detect TCAS B when time to CPA is within 60 seconds. In our simulation, TCAS A was able to detect TCAS B in the average of 63 seconds before CPA. Traffic advisory was given within 43 seconds as against 50 seconds standard while corrective vertical resolution advisory was given within 34 seconds when the intruder range was considered dangerous at the Closet Point of Approach (CPA), as against 37.5 seconds standard (Table 4.1).

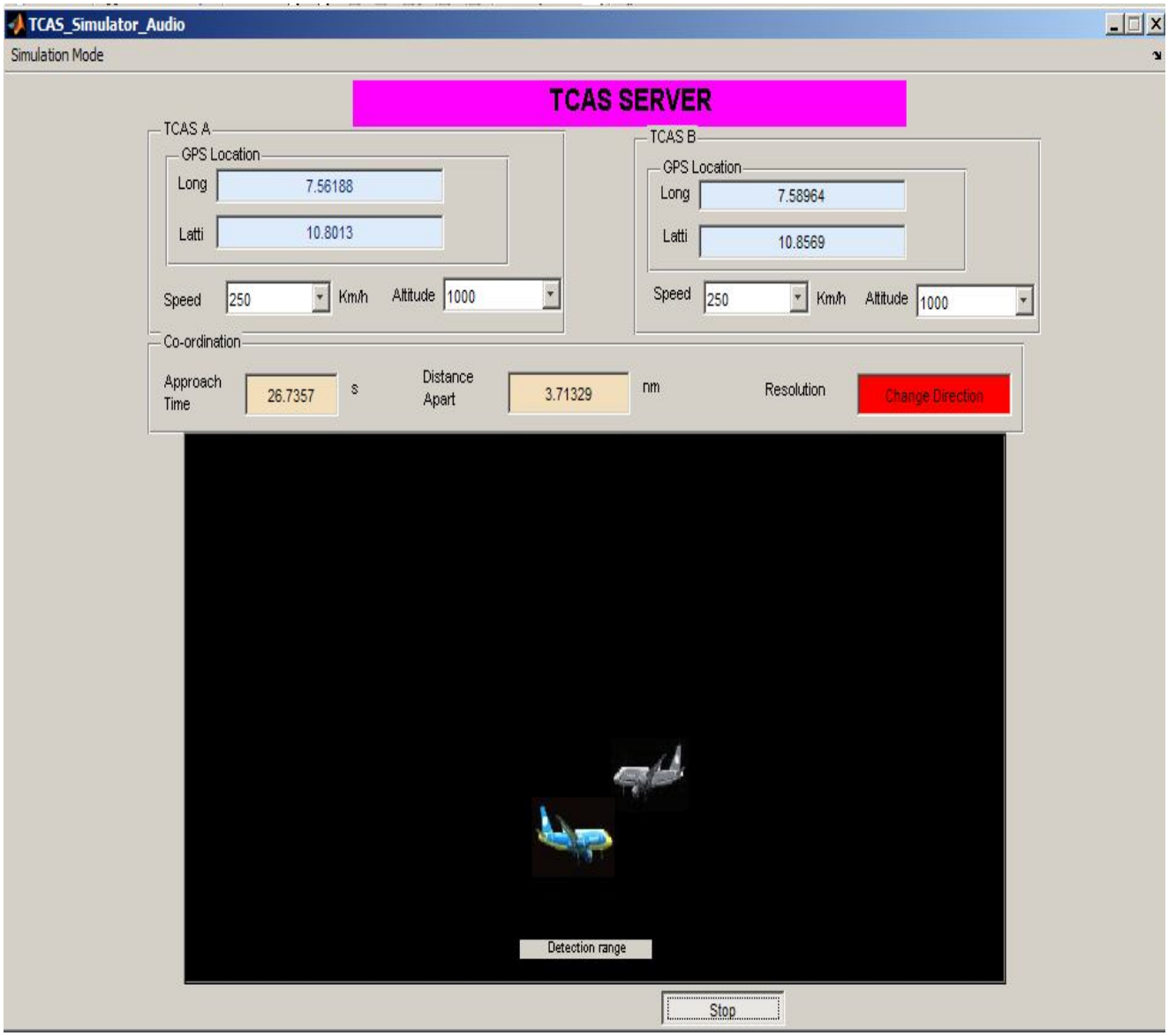


Fig. 4.2: TCAS A and TCAS B in the same altitude for vertical resolution

4.4 Lateral Resolution Advisory

TCAS is simulated in different altitude mode for lateral resolution advisory. Lateral resolution manoeuvres is issued to TCAS B equipped aircraft approaching to land and the TCAS A equipped aircraft taking off. TCAS A took off from a destination with longitude 7.545750, latitude 10.769100 and cruising speed of 250km/hr selected. While TCAS B in longitude 7.605770 and latitude 10.889200 was approaching to land (Fig. 4.3). We assumed that TCAS B was going to land where aircraft with TCAS A took off and that the two TCAS equipped aircrafts were on different altitude. Because of the continuous availability of current trajectory data for the two aircrafts received from GPS as shown in TCAS server console in Fig.4.3, corrective lateral resolution advisory could be issued for the two aircrafts. Corrective lateral RA was given within 34 seconds when the intruder range was considered dangerous at the Closet Point of Approach (CPA), as against standard 37.5 seconds (Table 4.2) because descending aircraft equipped with TCAS B could not gain full power instantaneously to climb and also ascending aircraft equipped with TCAS A, could not suddenly climb higher or descend immediately.

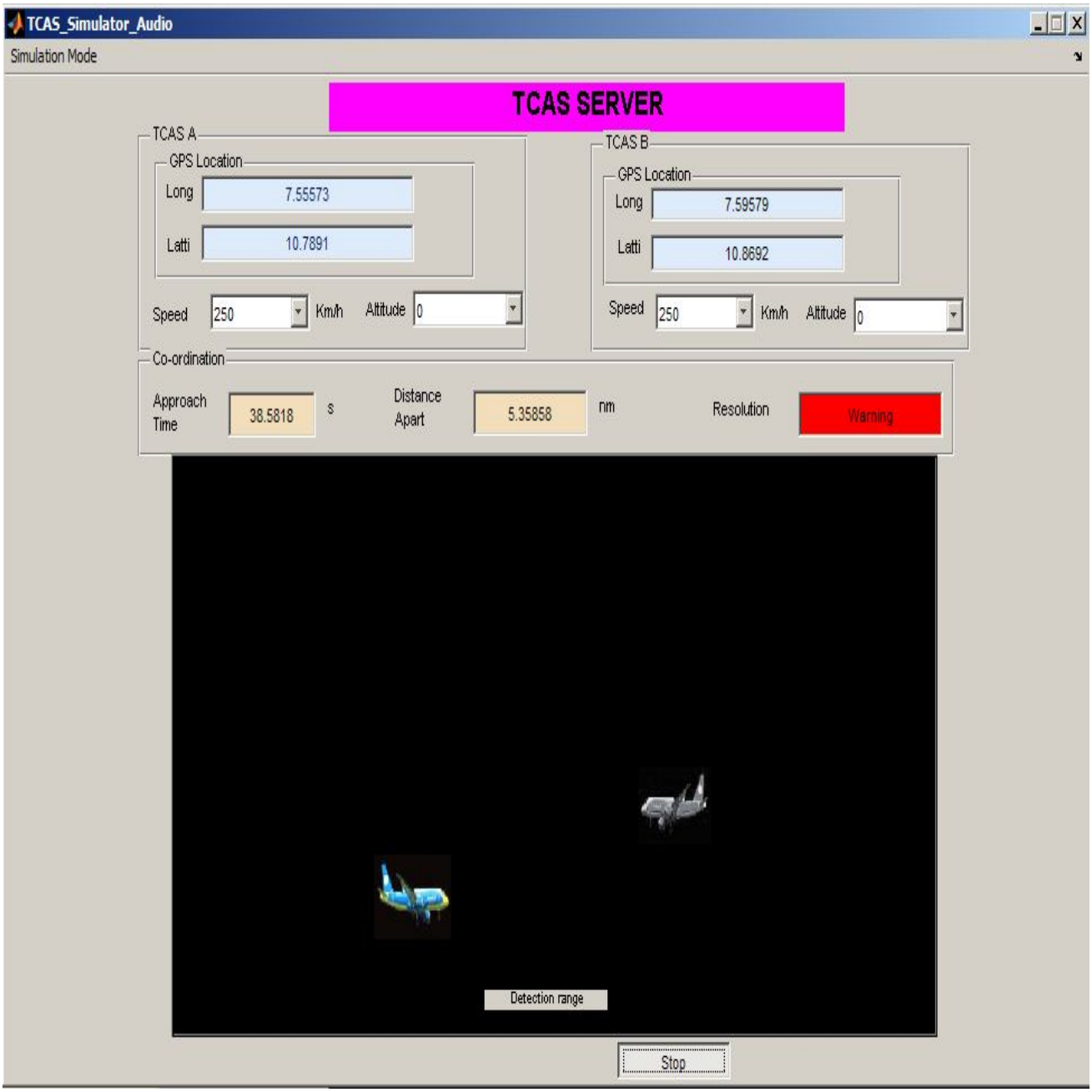


Fig. 4.3: TCAS simulation in different altitude for lateral resolution

4.5 Automatic Reversal Logic

TCAS is simulated in reverse logic mode. Reversal logic is automatically triggered when TCAS A that departed a destination with longitude 7.433624, latitude 10.544521 and with cruising speed of 250km/hr observed that TCAS B coming from a destination with 7.717896 longitude and 11.113727 latitude failed to obey TCAS resolution advisory that was given within 34 seconds when time to CPA and the intruder range was considered dangerous. Aircraft equipped with TCAS A reversed its earlier resolution to avert collision. When collision was firstly detected, TCAS A was earlier resolved to climb up, while TCAS B was to descend (Fig. 4.4). TCAS A obeyed the resolution while TCAS B did not comply. Instead of resolution to indicate clear of traffic, collision message persistently displayed, within 5 seconds reversal logic was automatically triggered and TCAS A reversed its earlier resolution and descended instead of climbing (Table 4.3).

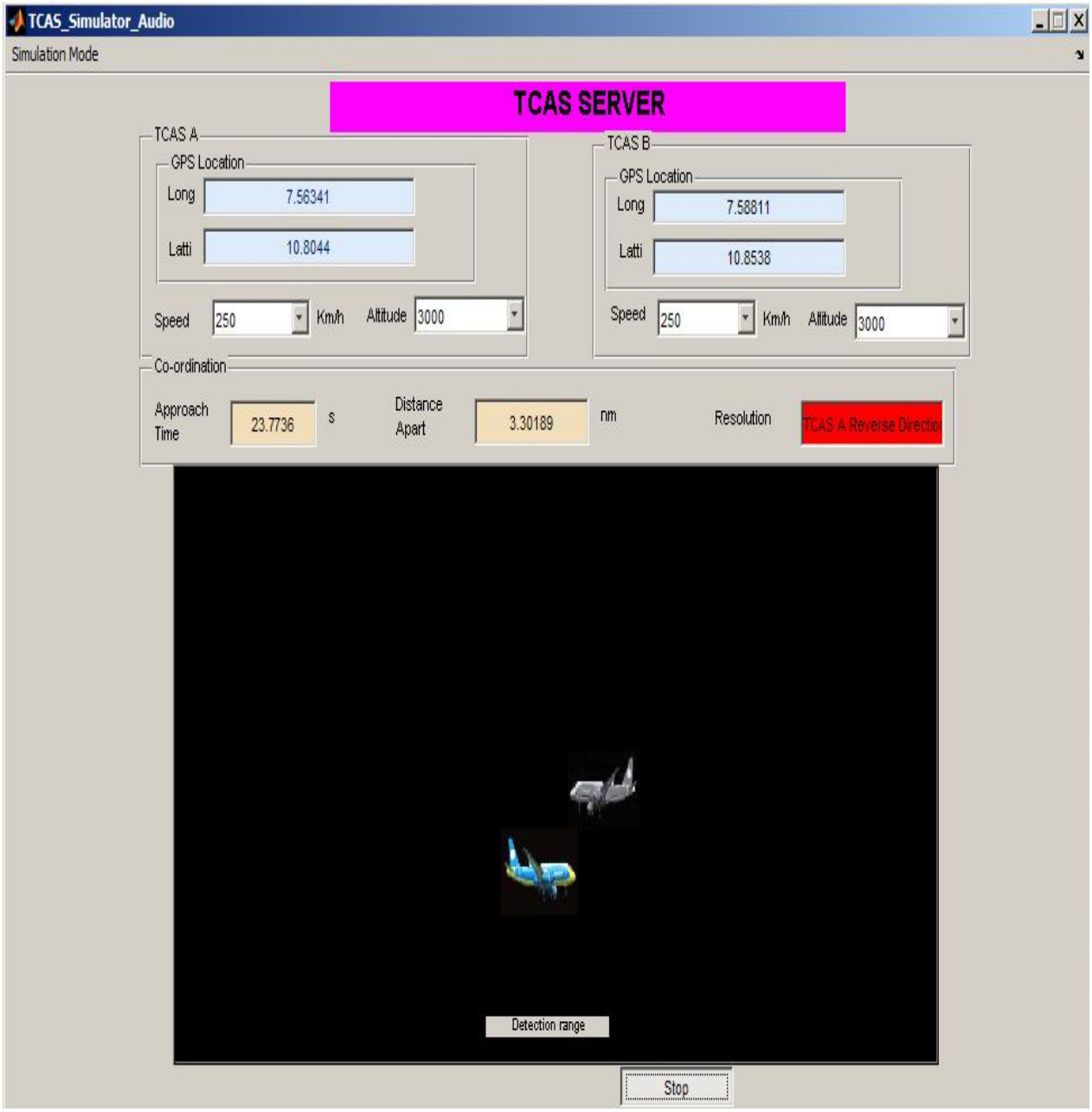


Fig. 4.4: TCAS in a Reverse Logic mode

4.6 Result and Discussion on Simulating our model

GPS based TCAS was simulated and our findings are discussed in this section.

Result of TCAS Simulation in the same altitude mode for vertical resolution:

TCAS A was able to detect TCAS B in the average of 63 seconds before CPA instead of 65 seconds. Traffic advisory was given within 46 seconds as against 50 seconds standard while corrective vertical resolution advisory was given within 36 seconds as against standard 37.5 seconds when the intruder range was considered dangerous at the Closet Point of Approach (Table 4.1).

Table 4.1: Result of simulating TCAS A and TCAS B in the same altitude mode

TCAS A		TCAS B		Distance Apart (nm)	TAU (Second)	Resolution	System Performance
Longitude (Deg)	Latitude (Deg)	Longitude (Deg)	Latitude (Deg)				
7.43	10.55	7.72	11.11	37.86	567.9		No Traffic
7.44	10.55	7.72	11.11	37.65	564.81		
7.44	10.56	7.71	11.1	36.21	543.21		
7.44	10.56	7.71	11.1	35.8	537.04		
7.55	10.78	7.6	10.88	6.8	101.98		
7.55	10.78	7.6	10.88	6.59	98.9		
7.55	10.78	7.6	10.88	6.39	95.81		
7.55	10.78	7.6	10.88	6.18	92.73		
7.56	10.79	7.6	10.87	5.36	80.39		
7.56	10.79	7.59	10.87	4.95	74.21		
7.56	10.79	7.59	10.86	4.74	71.13		
7.56	10.8	7.59	10.86	4.54	68.04		Detection
7.56	10.8	7.59	10.86	4.33	64.96		
7.56	10.8	7.59	10.86	4.12	61.87		
7.56	10.8	7.59	10.86	3.92	58.79		
7.56	10.8	7.59	10.86	3.71	55.7		
7.56	10.8	7.59	10.86	3.51	52.62		
7.56	10.8	7.59	10.85	3.3	49.53		
7.56	10.81	7.59	10.85	3.1	46.44	TA	Traffic
7.56	10.81	7.59	10.85	2.89	43.36	TA	Traffic
7.57	10.81	7.59	10.85	2.68	40.27	TA	Traffic
7.57	10.81	7.59	10.85	2.48	37.19		
7.57	10.81	7.58	10.85	2.27	34.1	RA	Collision
7.57	10.81	7.58	10.84	2.07	31.02	RA	Collision
7.57	10.82	7.58	10.84	1.86	27.93	RA	Collision
7.57	10.82	7.58	10.84	1.66	24.85		
7.57	10.82	7.58	10.84	1.45	21.76		
7.57	10.82	7.58	10.84	1.24	18.67		
7.57	10.82	7.58	10.84	1.04	15.59		
7.57	10.82	7.58	10.84	0.83	12.5		
7.57	10.82	7.58	10.83	0.63	9.42		
7.57	10.83	7.58	10.83	0.42	6.33		
7.57	10.83	7.58	10.83	0.22	3.25		
7.58	10.83	7.58	10.83	0.01	0.16		No Traffic
7.58	10.83	7.58	10.83	0.19	2.92		No Traffic

Result of TCAS Simulation in different altitude mode for lateral resolution:

TCAS A was able to detect TCAS B in the average of 63 seconds before CPA instead of 65 seconds and Traffic advisory was given within 46 seconds as against 50 seconds standard (Table 4.2).

Corrective lateral RA was equally given within 36 seconds because descending aircraft equipped with TCAS B could not gain full power instantaneously to climb and also ascending aircraft equipped with TCAS A, could not suddenly climb higher or descend immediately, to avoid accident, the only option left is to separate the aircraft laterally.

Table 4.2: Result of simulating TCAS A and TCAS B on different altitude mode

TCAS A		TCAS B		Distance Apart (nm)	TAU (Second)	Resolution	System Performance
Longitude (Deg)	Latitude (Deg)	Longitude (Deg)	Latitude (Deg)				
7.43	10.55	7.72	11.11	37.86	567.9		No Traffic
7.44	10.55	7.72	11.11	37.65	564.81		
7.44	10.56	7.71	11.1	36.01	540.13		
7.55	10.78	7.6	10.88	6.59	98.9		
7.55	10.78	7.6	10.88	6.39	95.81		
7.55	10.78	7.6	10.87	5.98	89.64		
7.55	10.79	7.6	10.87	5.56	83.47		
7.56	10.79	7.6	10.87	5.36	80.38		
7.56	10.79	7.6	10.87	5.15	77.3		
7.56	10.79	7.59	10.87	4.95	74.21		
7.56	10.79	7.59	10.86	4.74	71.13		
7.56	10.8	7.59	10.86	4.54	68.04		Detection
7.56	10.8	7.59	10.86	4.33	64.96		
7.56	10.8	7.59	10.86	4.12	61.87		
7.56	10.8	7.59	10.86	3.92	58.78		
7.56	10.8	7.59	10.86	3.71	55.7		
7.56	10.8	7.59	10.86	3.51	52.61		
7.56	10.8	7.59	10.85	3.3	49.53		
7.56	10.81	7.59	10.85	3.1	46.44	TA	Traffic
7.56	10.81	7.59	10.85	2.89	43.36	TA	Traffic
7.57	10.81	7.59	10.85	2.68	40.27	TA	Traffic
7.57	10.81	7.59	10.85	2.48	37.19		
7.57	10.81	7.58	10.85	2.27	34.1	RA	Collision
7.57	10.81	7.58	10.84	2.07	31.02	RA	Collision
7.57	10.82	7.58	10.84	1.86	27.93	RA	Reverse Direction
7.57	10.82	7.58	10.84	1.66	24.84		
7.57	10.82	7.58	10.84	1.45	21.76		
7.57	10.82	7.58	10.84	1.24	18.67		
7.57	10.82	7.58	10.84	1.04	15.59		
7.57	10.82	7.58	10.84	0.83	12.5		
7.57	10.82	7.58	10.83	0.63	9.42		
7.57	10.83	7.58	10.83	0.42	6.33		
7.57	10.83	7.58	10.83	0.22	3.25		No Traffic
7.58	10.83	7.58	10.83	0.01	0.16		No Traffic
7.58	10.83	7.58	10.83	0.2	2.93		No Traffic

Result of TCAS Simulation in Reverse Logic mode:

In addition to the fact that TCAS was able to detect each other, perform traffic advisory and corrective vertical/ lateral resolution in the time specified above; reversal logic was carried out because of continuous availability of current trajectory data as shown in TCAS server console. Feedback from resolution on the console displayed collision message persistently instead of clear of traffic, within 5 seconds reversal logic automatically triggered and instead of climbing TCAS A reversed its earlier resolution and descended.

Table 4.3: Result of simulating TCAS A and TCAS B in reverse logic mode

TCAS A		TCAS B		Distance Apart (nm)	TAU (Second)	Resolution	System Performance
Longitude (Deg)	Latitude (Deg)	Longitude (Deg)	Latitude (Deg)				
7.55	10.77	7.61	10.89	7.83	117.4		No Traffic
7.55	10.77	7.6	10.89	7.62	114.32		
7.55	10.78	7.6	10.88	6.8	101.98		
7.55	10.78	7.6	10.88	6.59	98.89		
7.55	10.78	7.6	10.88	6.18	92.72		
7.55	10.78	7.6	10.87	5.98	89.64		
7.56	10.79	7.6	10.87	5.36	80.38		
7.56	10.79	7.6	10.87	5.15	77.29		
7.56	10.79	7.59	10.86	4.74	71.12		
7.56	10.8	7.59	10.86	4.54	68.04		Detection
7.56	10.8	7.59	10.86	4.33	64.95		
7.56	10.8	7.59	10.86	4.12	61.87		
7.56	10.8	7.59	10.86	3.92	58.78		
7.56	10.8	7.59	10.86	3.71	55.69		
7.56	10.8	7.59	10.86	3.51	52.61		
7.56	10.8	7.59	10.85	3.3	49.52		
7.56	10.81	7.59	10.85	3.1	46.44	TA	Traffic
7.56	10.81	7.59	10.85	2.89	43.35	TA	Traffic
7.57	10.81	7.59	10.85	2.68	40.27	TA	Traffic
7.57	10.81	7.59	10.85	2.48	37.18		
7.57	10.81	7.58	10.85	2.27	34.1	RA	Collision
7.57	10.81	7.58	10.84	2.07	31.01	RA	Collision
7.57	10.82	7.58	10.84	1.86	27.92	RA	Collision
7.57	10.82	7.58	10.84	1.66	24.84		
7.57	10.82	7.58	10.84	1.45	21.75		
7.57	10.82	7.58	10.84	1.24	18.67		
7.57	10.82	7.58	10.84	1.04	15.58		
7.57	10.82	7.58	10.84	0.83	12.5		
7.57	10.82	7.58	10.83	0.63	9.41		
7.57	10.83	7.58	10.83	0.42	6.33		
7.57	10.83	7.58	10.83	0.22	3.24		
7.58	10.83	7.58	10.83	0.01	0.16		No Traffic
7.58	10.83	7.58	10.83	0.2	2.93		No Traffic

Results of our GPS based TCAS simulation showed that resolution advisory could be given to GPS based TCAS equipped aircraft compare to existing TCAS even if the altitude is below 1000ft because of continuous availability of current trajectory data, (Table 4.4).

Table 4.4: Comparison of Tau (Seconds) for existing TCAS and simulated TCAS

Altitude	**Tau (Seconds) for existing TCAS**		**Tau (Seconds) for simulated TCAS**	
	TA	**RA**	**TA**	**RA**
<1000	20	N/A	26.7	25.3
1000 – 3000	25	15	26.7	25.3
5000 - 10000	40	25	44.5	26
10000 - 20000	45	30	46	33.4
20000 - 42000	48	35	46	36

Continuous availability of current trajectory data for the two aircrafts equipped with GPS based TCAS received from satellite and displayed on the server console made it possible to issue lateral resolution manoeuvres whenever necessary to prevent collision. Also reversal logic could be automatically triggered when one of the two aircrafts refused to obey earlier resolution advisory given by the TCAS.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

Failure of many real-time embedded systems may endanger human life or may cause substantial loss in economic values. Performance analysis plays an important role in the design process of complex embedded systems for analyzing essential performance characteristics of system design at an early phase before actual implementation of the system. Embedded system designer will find it difficult to build a prototype for each design alternative to directly measure performance characteristics because of high implementation costs, stringent time-to-market constraints or the risk of being totally incorrect.

In our contribution on determination of most appropriate performance analysis methods to answer our first objective which is to determine the most appropriate methods for system-level performance analysis of real-time embedded systems, a thorough literature review was carried out and performance analysis requirements were analysed. We identified major performance analysis methods such as Simulation and formal analysis methods, Real-time calculus, Holistic scheduling analysis, Compositional method, Timed automata based performance analysis and Stochastic analysis method. We identified main limitations of the existing techniques and compared these techniques with respect to key performance indexes such as system's throughput, response times, memory requirements etc. We found the technique based on Timed automata to be relatively better than the others because it combines modelling and simulation with its

performance analysis. Challenges of modular performance analysis were also identified. Our work was based on the technique of Timed automata.

To answer our second goal which is to model and analyse a GPS-based Traffic Alert and Collision Avoidance System, we used Proteus which considered time quantitatively to model and simulate Microcontroller for TCAS design. We found out that quantitative time introduced as delay in the execution time of program running on microcontroller using Proteus has effect on output speed, CPU and memory usage. From the experiment, a combination of 500ms delay and 15MHz Clock frequency gave 60% of CPU usage, moderate memory consumption and output speed. Clock frequency of 15MHz was used in our design which is within the standard range of 0-60MHz with delays of 10, 60, 100 and 1000ms between the Tx /Rx and display which gave required output speed timely.

In an embedded system, result is expected to be timely, without too much delay and not too fast, introduction of delay with appropriate crystal value would give desire output speed. We found out that time element is relative to GPS clock, displayed continuously and additive as object changes locations or positions.

Quantitative timing information is not only essential for time critical systems, but also a necessary ingredient in order to analyze and specify performance aspects of systems.

GPS was included in our TCAS model to minimise delay and improve accuracy. We used UPPAAL to model and analyse a GPS based TCAS. We incorporated lateral resolution manoeuvres into GPS based TCAS which is not available in the existing TCAS. We found out that delay within range of 35-50s introduced in Traffic advisory state ensured reachability, safety and issuance of traffic advisory. Delay within the range of 25-40s introduced to aircraft that is in air-born and aircraft that is in Ground states ensured reachability, issuance of vertical and lateral resolution without deadlock.

Resolution could be issued on a time bases (that is, resolution is given within a certain period of time) than TCAS equipped only with Transponder because information about current time, date, position and many more are displayed continuously every second from GPS satellite.

Reversal resolution was incorporated to our GPS based TCAS model using UPPAAL to answer the third objective, which is to investigate the possibility of incorporating reversal logic into a GPS based Traffic Alert and Collision Avoidance System. We found out that delay within the range of 5-10s introduced in Reverse state ensured that reversal logic is automatically enabled as a result of feedback from 'Threat-Detect-after state which depends on other states.

This shows that our system is dynamic because reversal logic is automatically enabled in a situation where it is obvious that one of the two aircrafts failed to comply with the earlier resolution advisory given by TCAS.

The prototype of GPS based TCAS modelled with UPPAAL is designed and simulated using MATLAB to ascertain the correctness, integrity and safety of the system. Result of our simulation using MATLAB showed that aircrafts equipped with GPS based TCAS could detect one another within the average time of 63 seconds before CPA instead of 65 seconds.

Traffic advisory could be given within 46 seconds as against 50 seconds standard, while corrective vertical and lateral resolution advisory could be given within 36 seconds as against 37.5 seconds standard for vertical resolution.

Also, result showed that reversal logic could be automatically triggered within 5 seconds if one of the aircrafts failed to obey TCAS resolution advisory.

5.2 Recommendation

Due to the increasing complexity of modern embedded systems, the embedded system designers face the problem of evaluating many candidate hardware-software architectures with respect to various performance indexes in the early design cycle. Embedded system is heterogeneous in nature and mostly the design is application specific, it will be a great challenge for designers to build many prototypes to ascertain the performance characteristics of the system, hence there is a need to implement a method to evaluate the system early. Modular Performance analysis is crucial in the design process of complex embedded systems for analyzing essential performance characteristics of system design at an early phase. Its implementation is recommended because it gives the choice of important design decisions before much time and resources are invested in detailed implementation of the actual design.

Implementation of our GPS based TCAS is recommended because of its contributions to minimising aircraft accident and improve safety. In our design, performance analysis of the states and paths properties of the model were checked for their reachability, safety and liveness to ensuring increase in total separation between aircrafts both in horizontal and vertical and to allow reverse logic. Because manoeuvres in the existing TCAS system are restricted to changes in altitude and modification of climb/sink rates, that is vertical plane only. There may be instances where we can make use of lateral resolution manoeuvres in a situation where we have complex traffic conflict scenarios between aircrafts taking off or landing.

Implementation of horizontal resolution directives which are useful in a conflict between two aircrafts close to the ground where there may be little, if any, vertical manoeuvring space possible is recommended. Generally takeoff of an aircraft is a phase

of flight in which an aircraft goes through a transition from moving along the ground that is taxing, to flying in the air, which normally starts from the runway. In the existing TCAS, if an aircraft is taking off and another one is approaching to land and is below the altitude of 1450ft on the same runway, although the TCAS on these two aircrafts can detect each other but no resolution is issued because vertical resolution cannot be issued to the two aircrafts. At this point, naturally, the aircraft that is approaching to land has lost power during descent; the spoilers increase the drag force and slow down the aircraft. Issuing climb command of vertical resolution to the pilot at this moment could be hazardous to the aircraft because at that stage full power cannot be gained instantaneously, in the attempt the aircraft could stall. For the aircraft that is taking off, if it is estimated to be within 400ft closer to the ground and is climbing at say 15^0 or 45^0, it will be difficult for such an aircraft to suddenly climb at 90^0 or higher or descend immediately to 270^0 in compliance with vertical resolution, the aircraft may overshoot the runway or crash land. As a result of this, implementation of lateral resolution is recommended, where aircraft can be advised to yaw to the left or to the right while landing or taking off as the case may be.

Time-based representation is recommended for traffic situation because under certain condition there may be need for time-based representation unlike in the existing TCAS, where traffic situation is given within a range-based, that is, the traffic situation is given within a configurable ranges of miles/feet.

Finally, implementation of reversal logic is recommended where resolution advisory issued by TCAS is reversed when it has been shown that after few seconds, one of the two aircrafts does not comply with the advisory given earlier by the TCAS.

5.3 Future Research Work

Possible future research includes questions about design improvement. We note, however, that hardware-software co-simulations are often computationally complex with long running times and there is a need for future research to analyse the effect of program's environment such as processor it runs on, direct interference from operating system and concurrent running tasks like interrupts.

There is a need to improve the capability of timed automata notation to allow more than only one level of parallel composition as obtainable now, so that many numbers of states are covered in the modelling to enable early termination and reachability analysis of the timed systems.

There is a need to integrate our GPS based TCAS with other on-board avionics equipment in the aircraft such as Ground Proximity Warning System (GPWS) and Electronic Centralised Aircraft Monitor (ECAM).

And finally there is a need for further research in the connection between TCAS and terrain/ground and obstacle awareness such as connection to Terrain Awareness Warning System (TAWS), including Minimum safety Altitude (MSA) sector awareness, which might be critical for creating feasible (non-dangerous, in the context of terrain clearance) and useful resolution advisories that prevent extreme descent instructions if close to terrain, to ensure that TCAS RAs never facilitate Controlled Flight Into Terrain (CFIT) scenarios.

REFERENCES

Agarwal A.(1992) Performance Tradeoffs in Multithreaded Processors. *IEEE Transactions on Parallel and Distributed Systems*, 3(5), PP. 525–539.

Alexander Viehl, Timo Schonwald, Oliver B. and Wolfgang R.(2009). Formal Performance Analysis and Simulation of UML/SysML Models for ESL Design. (http://citeseerx.ist.psu.edu).

Alexandre M. Bayen and Claire J. T. (2004), Case Study - Air Traffic Management Systems, in Control Systems, Robotics, and Automation , [Ed. Heinz Unbehauen], in *Encyclopaedia of Life Support Systems (EOLSS), Developed under the Auspices of the UNESCO*, Eolss Publishers, Oxford ,UK, Retrieved October 25, 2010, from http://www.eolss.net.

Alur R.and Dill D. (1996). Automata-theoretic verification of real-time systems. In *Formal Methods for Real-Time Computing,* PP. 55-82.

Argenio P.R. D, Katoen J. P., Ruys T.C. and Tretmans J. (1997). The bounded retransmission protocol must be on time. In *Proc. of the 3rd Workshop on Tools and Algorithms for the Construction and Analysis of Systems, number 1217* in Lecture Notes in Computer Science, Springer–Verlag, PP. 416 – 431

Arvindra S.(2008). *Architecture Journal on Embedded System*. Retrieved February 29, 2009, from http://msdn.microsoft.com/en-us/library/dd129911.aspx.

Baruah S. K. (1998). A general model for recurring real-time tasks. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, PP. 114–122.

Baruah, S. K., Chen D. Gorinsky S. and Mok A. K. (1999). Generalized multiframe tasks. *Real-Time Systems*, 17(1): 5–22.

Baruah S. K. (2003). Dynamic- and static-priority scheduling of recurring real time tasks. *Real-Time Systems*, 24(1):93–128.

Benini L., D. Bertozzi, D. Bruni, N. Drago, Fummi F. and Poncino M. (2003). SystemC cosimulation and emulation of multiprocessor SoC designs. *IEEE Computer*, 36(4):53–59.

Booch G. (1994). *Object Oriented Analysis and Design With Applications.* The Benjamin/Cummings Publishing Company Inc. Redwood City, Calif. PP 35-57.

Benner Th., Ernst R., Könenkamp I., Schüler P. and Schaub H.-C. (1995). A Prototyping System for Verification and Evaluation in Hardware-Software Cosynthesis, Sixth *IEEE International Workshop on Rapid System Prototyping*, ISBN 0-8186-7100-9, PP. 124–132.

Buchenrieder K. and Rozenblit J.W. (1995). Codesign: An Overview, Codesign

Computer-Aided Software/Hardware Engineering. [Eds Rozenblit J.and Buchenrieder K.], *IEEE* Press ISBN 0-7803-1049-7.

Chakraborty S., K¨unzli S.and Thiele L. (2003). A general framework for analysing system properties in platform-based embedded system designs. In *Design, Automation and Test in Europe (DATE)*, Munich, Germany, Mar.. IEEE Press, PP. 190–195.

Chakraborty S. and Thiele L. (2005). A new task model for streaming applications and its schedulability analysis. In *Design, Automation and Test in Europe (DATE)*, PP. 486–491.

Christer N., Anders W. and Wang Y. (1999). Timed automata as task models for event- driven systems. In *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, PP.182, Washington, DC, USA, IEEE Computer Society.

Clarke E.M. and Emerson E.A. (1981). Synthesis of synchronisation skeletons for branching time logic. *Logic of Programs,* LNCS 131, PP. 52-71.

CCS C (Custom Computer Services C) Inco (1994). *CCS C Compiler Reference Manual*.

David H. (2005). Introduction to MATLAB for engineering students, Northwestern University, version 1.2, PP. 1-2.

De Micheli G., Ku D., Mailhot F. and Truong T. (1995). The Olympus Synthesis System, Stanford University, *IEEE Design and Test of Computers*, PP. 10-15.

Ericsson C., A. Wall, and Wang Y. (1999).Timed Automata as Task Models for Event-Driven Systems. In *RTCSA '99: Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, Washington, DC, USA, PP. 182.

FAA (Federal Aviation Administration), US department of transportation (2000), Introduction to TCAS II Version 7, PP. 22.

Gajski D. D., Vahid F., Narayan S. and Gong J. (1994). *Specification and Design of Embedded Systems*, PTR Prentice Hall Englewood Cliffs, PP 5-28.

Gerd B., Alexandre D., and Kim G. L.(2004). A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, Springer-Verlag, PP. 200-236.

Gerd B., Kim G. L., Justin P., Carsten W., and Wang Y. (1999). Efficient timed reachability analysis using clock difference diagrams. In *Proceedings of the 12th Int. Conf. on Computer Aided Verification*, volume 1633 of Lecture Notes in

Computer Science. Springer - Verlag.

Gerd B., Alexandre D., Kim G. L., Oliver M., Paul P. and Wang Y. (2001). UPPAAL - present and future. In *Proceeding of 40th IEEE Conference on Decision and Control*. IEEE Computer Society Press.

Green P., Rushton P.and Beggs R. (1994). An Example of Applying the Codesign MethodMoose, *Proceedings of the Third International Workshop on Hardware/Software Codesign*, Grenoble France.

Grotker T., Liao S., Martin G. and Swan S. (2002). *System Design with SystemC*. Kluwer Academic Publishers, Boston, MA, USA.

Guardian.co.uk. (2008). "Toyota shares fall for 6th day as recall woes deepen". London. Retrieved April 30, 2010, from http://www.guardian.co.uk /business/feedarticle/8921329.

Hendriks M. and Verhoef M.(2006). Timed automata based analysis of embedded System architectures. Technical Report ICIS-R06003, ICIS, Radboud University, Nijmegen, The Netherlands.

Henia R., Hamann A., Jersak M., Racu R., Richter K. and Ernst R. (2005). System level performance analysis – the symta/s approach. *IEE Proceedings – Computers and Digital Techniques*, 152(2):148–166.

Henrik L. and Paul P. (1997). Formal Verification of a TDMA Protocol Startup Mechanism. In *Proceeding of the Pacific Rim International. Symposium on Fault Tolerant Systems*, PP. 235-242.

Henrik E., Jensen K. Larsen G. and Arne S. (1996). Modelling and Analysis of a Collision Avoidance Protocol Using SPIN and UPPAAL. In *Proceeding of 2nd International Workshop on the SPIN Verification System*, PP. 1–20.

Howard B., Giorgio P. F., Joost-Pieter K., Diego L., and Mieke M.(1998). Automatic verification of a lip synchronisation algorithm using UPPAAL. In Bas Luttik Jan Friso Groote and Jos van Wamel, editors. In *Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems*. Amsterdam.

http://pressroom.toyota.com/safety-recall/ Retrieved April 4, 2010.

http://semico.com/studies/category.asp?id=9#1202. Retrieved 2010-04 -29

http://adsb.tc.faa.gov/TCAS.htm. Retrieved 2010-10-25.

Jakob E., Andreas E., Mikael S., Jan G. and Hans H. (2002). *International Journal on Software Tools for Technology Transfer (STTT)*. http://www.springerlink.com/content/101563

Jersak M., Henia R. and Ernst R. (2004). Context-aware performance analysis for

efficient embedded systems design. In *Proceeding 7th Design, Automation and Test in Europe (DATE)*.

Jersak M., Richter K. and Ernst R. (2004). Performance analysis for complex embedded applications. *International Journal of Embedded Systems, Special Issue on Codesign for SoC*.

Johan B., Kim G. L., Fredrik L., Paul P., Wang Y. and Carsten W.(1998). New generation of UPPAAL. In *International Workshop on Software Tools for Technology Transfer*.

John E. H. and Jeffrey D. U. (2001). *Introduction of Automata Theory, Languages, and Computation*. Addison Wesley, PP. 7-21.

Joost-Pieter K. (1999). Concepts, Algorithms and Tools for Model Checking, Lecture Notes of the Course "Mechanised Validation of Parallel Systems" (course number 10359) Semester.

Kalavade A., Lee E. A. and Berkeley U. (1993). A Hardware - Software Codesign Methodology for DSP Applications. *IEEE Design and Test of Computers*.

Klaus H., Arne S., Kim G. L. and Kristian L. (1997). Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL. In *Proceeding of the 18th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press.

Koymans R. (1992). Specifying Message Passing and Time-Critical Systems with Temporal Logic. LNCS 651., Springer-Verlag, PP. 99-142.

Lamport L. (1994). The temporal logic of actions. *ACM Transaction on Programming Languages and Systems,* 16(3): 872-923.

Larsen K.G., Pettersson P. and Wang Y. (1997). UPPAAL in a nutshell, *Software Tools for Technology Transfer,* 1: 134-153.

Le Boudec J. Y. and Thiran P. (2001). Network calculus: a theory of deterministic queuing systems for the International Springer-Verlag New York, Inc.

Lichtenstein O., Pnuelia A. and Zuck L. (1985) . The glory of the past *Logics of Programs,* LNCS 193, PP. 196-218.

Liggesmeyer P., Rothfelder M., Rettelbach M. and Ackermann T. (1998). Qualitatssicherung Software-basierter technischer Systeme - Problembereiche und osungsansatze. *Informatik Spektrum,* 21: 249-258.

Lothar T. and Ernesto W. (2009). Performance Analysis of Distributed Embedded Systems Networked. *Embedded Systems Handbook*, CRC Press/Taylor & Francis.

Lahiri K., A. Raghunathan, and Dey S. (2001). System level performance analysis for

designing on- chip communication architectures. *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems*, 20(6):768–783.

Liu L. C. and James W. L. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

Lehoczky J. P. (1996). Real-time queueing theory. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS)*, PP 186, Washington, DC, USA,. IEEE Computer Society.

Lufthansa Resource Technical Training Manual, 2005. Part 66: 328 - 370.

Magnus L., Paul P. and Wang Y. (1998), Formal Design and Analysis of a Gear-Box Controller. In *Proceeding of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, number 1384 in Lecture Notes in Computer Science, Springer–Verlag, 998 PP. 281–297.

Martin P. B. (2003). *Programming 8 Bit PIC Microcontroller in C with Interactive Hardware Simulation*, Retrieved February 29, 2009, from www.microchip.com PP. 11-205.

MATLAB (2009), Mapping Toolbox User's Guide, version 7.9.0. 529 (R2009B)

Mok A. K. and Chen D. (1997). A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645.

Maxiaguine A., K¨unzli S. and Thiele L. (2004). Workload characterization model for tasks with variable execution demand. In *Design, Automation and Test in Europe (DATE)*, PP. 1040–1045, Paris, France, IEEE Computer Society.

Michael B. (2007). Embedded Systems Glossary. Neutrino Technical Library. Retrieved February 29, 2009, from www.barrgroup.com/Embedded-Systems/Glossary.

Michael B. and Anthony M. (2006) Programming Embedded Systems: With C and GNU Development Tool. Published by O'reilly Media, Inc US PP. 1-11.

Naedele M., Thiele L. and Eisenring M. (1999). Characterising variable task releases and processor capacities. In *Proceedings of the 14th IFAC World Congress 1999*, Beijing.

Nikolay S. and Lothar T. (2007). Determining a Schedulability Region using Modular Performance Analysis and Real-Time Interfaces in *WSN, ETH* Zurich PP. 26-35.

OSCI (The Open SystemC Initiative) (2010). Retrieved April 29, 2010, from http://www.systemc.org.

Pnuelia A. (1977). The temporal logic of programs. *Proceedings 18th IEEE Symposium on Foundations of Computer Science,(FOCS)* PP. 46-57.

Pop P., Eles P. and Peng Z. (2000). Performance estimation for embedded systems with data and control dependencies. In *Proceedings of the 8th International Workshop on Hardware/Software Co-Design (CODES)*, PP. 62–66.

Pulli P., Dähler J., Gisiger H. P.and Kundig A. (1988). Execution of Ward's Transformation Schema on the Graphic Specification and Prototyping Tool *SPECS, CompEuro 88*, Brussels, 11.-14.

Quielle J. and Sifakis J. (1982). Specification and verification of concurrent systems in CESAR. *Proceedings 5th International Symposium on Programming,* LNCS 137, PP. 337-351.

Raimondi F. (2010). Computational Tree Logic and Model Checking A simple introduction http://www.cs.ucl.ac.uk/staff/f.raimondi/.

Rajeev A. and David L. D. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183-235.

Robert E.(1996). Object Oriented Petri Net Approach to Embedded System Design; a dissertation submitted to the Swiss Federal Institute of Technology Zurich for the degree of Doctor of Technical Sciences, PP 110-120.

Rumbaugh J., Blaha M., Premerlani W., Eddy F. and Lorensen W (1991). Object-Oriented Modeling and Design, Prentice Hall, PP 10-12.

Rutten M. J., Eijndhoven J. T. J. and Pol E. J. D. (2002). Robust media processing in a Flexible and cost-effective network of multi-tasking coprocessors. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS)*, page 223, Washington, DC, USA. IEEE Computer Society.

Selic and Gullekson, (1994). *Ward, Real-Time Object-Oriented Modelling*, Wiley ISBN 0- 471-59917, PP 150-160.

Simon K., Francesco P., Luca B., Lothar T.(2009). Combining Simulation and Formal Methods for System-Level Performance Analysis.

Simon P. (2006). Evaluation and Comparison of Performance Analysis Methods for Distributed Embedded Systems Master's thesis submitted to the Swiss Federal Institute of Technology Zurich.

Svendsli, O. J. (2003). Atmel's Self-Programming Flash Microcontrollers. Retrieved October 25, 2010, from www.atmel.com/Images/doc0313.pdf

Takahashi and Yoshio (2010). *Wall Street Journal.* Retrieved April 4, 2010, from http://online.wsj.com/article/BT-CO-20100209702754html?mod=WSJ_ World_MIDDLEHeadlinesAsia..

Thiele L., Chakraborty S., Gries M., Maxiaguine A. and Greutert J. (2001). Embedded

software in network processors - models and algorithms. In *Proceedings of the 1st International Workshop on Embedded Software (EMSOFT)*, PP. 416–434, London, UK,. Springer-Verlag.

Thiele L., Chakraborty S., Gries M. and K¨unzli S.(2002). A framework for evaluating design tradeoffs in packet processing architectures. In *Proceedings of the 39th Design Automation Conference (DAC)*, PP. 880–885, New Orleans LA, USA, ACM Press.

Thiele L., Wandeler E. and Chakraborty S. (2005). A stream-oriented component model for performance analysis of multiprocessor DSPs. *IEEE Signal Processing Magazine, special Issue on Hardware/Software Co-design for DSP*, 22(3):38-46.

Thomas A. Henzinger (1994). Symbolic model checking for real-time systems. Information and Computation, 111:193–244.

Tindell K. and Clark J. (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing & Microprogramming*, 40(2-3):117–134.

Tobias A., Elena F., Leonid M., Paul P. and Wang Y. (2002). Times - a tool for modelling and implementation of embedded systems. In *TACAS '02: Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 460-464, London, UK, Springer-Verlag.

Toyota USA Newsroom http://pressroom.toyota.com/pr/tms/toyota/toyota-consumer-safety- advisory-102572.aspx .Retrieved April 29, 2010.

Wandeler E. (2006). Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems. A dissertation submitted to the Swiss Federal Institute of Technology Zurich.Embedded.com – Under Hood:Robot Guitar embeds autotuning, PP 23-55.

Ward S.J. and Mellor P.T. (1985). *Structured Development for Real-Time Systems*, Yourdon Press Computing Series, Prentice-Hall, Englewood Cliffs.

West C.H. (1982). Applications and limitations of automated protocol validation. *Proceedings 2nd Symposium on Protocol Specification, Testing and Verification,* PP. 361-371.

White I.(1994). *Using the Booch Method: A Rational Approach*, The Benjamin/ Cummings Publishing Company Inc. Redwood City, Calif.ISBN 0-8053-06, 14-5.

www.guardian.co.uk/Guardian News and Media Limited 2010. Retrieved 2010-04 –30

www.labcenter.com/products/vsm_overview.cfm Retrieved 2010- 3-12

www.pumpkininc.com. Retrieved 2010-04 -29

www.vehicle-lab.net/ecu.html. Retrieved 2010-04 -29

www.fi.edu/flights/own2/forces.html. Retrieved 2010-12-25

www.free-online-private-pilot-ground-school.com/copyright.html Retrieved 2010-12-25

www.physicsforums.com/showthread.php. Retrieved 2010- 2-25

www.eurocontrol.int/TCAS II version 7.1. Retrieved 2010-10-25

www.airliners.net/tech_ops/. Retrieved 2011 - 08- 10

Yau-Tsun S. L. and Sharad M. (1999). Performance Analysis of Real-time Embedded Software. Published by Springer, ISBN 0792383826, 9780792383826.

Yourdon E. and Constantine L. L. (1979). Structured Design, Fundamentals of a Discipline of Computer Program and Systems Design, Prentice Hall Inc., Englewood Cliffs, PP 44-65.

Yovine S. (1998). Model checking timed automata. In *Embedded Systems,* LNCS 1494.

APPENDIX A

E< >IntrDetect.ACdetectd (1)

E<>IntrDetect.StartInterrg imply IntrDetect.Identf (2)

E[ ] (IntrDetect.Identf imply x<=60) (3)

E<> (IntrDetect.Identf imply x<=45 and x<=50) (4)

E<> IntrDetect.StartInterrg imply IntrDetect.Identf (5)

E<>Surveillance.IdentAcqd imply x<=5 (6)

E<>Surveillance.Status (7)

E<>(Surveillance.Status imply Surveillance.ACground) (8)

E<>(Surveillance.Status imply Surveillance.ACairborne) (9)

E<>(Surveillance.Status imply Surveillance.Speed) (10)

E<> Surveillance.Status imply Surveillance.Aclevel (11)

E<> Surveillance.Status imply Tracking.Status (12)

A[] Tracking.SenseLevel imply (x<=5) (13)

E<> Tracking.SenseLevel imply Surveillance.ACairborne (14)

E<> Tracking.SenseLevel imply Surveillance.ACground (15)

E<> Tracking.Speed imply Surveillance.Speed (16)

E<> Tracking.Speed imply x<=5 (17)

E<> Surveillance.Initialise imply Tracking.EstablCom (18)

A[] Coordination.ACAirbornST imply (x<=5 and x>=25) (19)

A[] Coordination.ACGroundST imply (x<=5 or x<=30) (20)

A<> Coordination. TAdv imply x>=35 (21)

A<> Coordination. ACAirbornST imply (x>=25) (22)

A<> Coordination.ACGroundST imply (x<=30) (23)

E<> Coordination. TAdv imply Coordination.ThreatDetecd (24)

E<>Coordination. ExcedUperTAThrsd imply Coordination. ThreatDeclad (25)

E<>Coordination.ACAirbornST imply Coordination.ACGroundST (26)

E<> Coordination.ThreatDetecd imply (Coordination.ACAirbornST and
Coordination.ACGroundST) (27)

E<>Coordination.ACAirbornST imply Coordination.ThreatDetAf (28)

E<>Coordination.ACGroundST imply Coordination.ThreatDetAf (29)

A[] Coordination.ThreatDetAf imply (x>=5) (30)

E<>Coordination.ThreatDetAf imply Coordination.Reverse (31)

E<>Coordination.ACAirbornST imply Resolution.Start (32)

E<>Coordination.MinTime imply (Coordination.ThreatDetc or
Coordination.Reverse) (33)

E<> Coordination.Reverse imply (x>=5 and x<=10) (34)

E<>Coordination.CPA imply (Coordination.ACAirbornST or
Coordination.ACGroundST) (35)

E<>Coordination.ConflctResd imply Coordination.CPA (36)

E<> Resolution.EstablCom imply Resolution.TAAdv (37)

E<>Resolution.TAAdv imply Resolution.CRAdv (38)

E<> Resolution.EstablCom imply Resolution.CRAReq (39)

E<>Resolution.CRAdv imply Resolution.DispTrafficClerd (40)

E<> Resolution.CRAReq imply Resolution.CRAdv (41)

A<> Resolution.TAAdv imply Resolution.CRAdv (42)

E<> Reversal.RACorrREVERSAL imply Reversal.ConflictResolved (43)

A[] not deadlock (44)

APPENDIX B

```
function varargout = TCAS_Simulator_Audio(varargin)
%  GPS based TCAS Prototype Simulation Codes
%TCAS_SIMULATOR_AUDIO M-file for TCAS_Simulator_Audio.fig
%       TCAS_SIMULATOR_AUDIO, by itself, creates a new TCAS_SIMULATOR_AUDIO or raises
the existing
%      singleton*.
%
%       H = TCAS_SIMULATOR_AUDIO returns the handle to a new TCAS_SIMULATOR_AUDIO or
the handle to
%      the existing singleton*.
%
%      TCAS_SIMULATOR_AUDIO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TCAS_SIMULATOR_AUDIO.M with the given input arguments.
%
%       TCAS_SIMULATOR_AUDIO('Property','Value',...) creates a new TCAS_SIMULATOR_AUDIO
or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before TCAS_Simulator_Audio_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to TCAS_Simulator_Audio_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help TCAS_Simulator_Audio
% Last Modified by GUIDE v2.5 13-Apr-2012 13:42:50

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @TCAS_Simulator_Audio_OpeningFcn, ...
                   'gui_OutputFcn',  @TCAS_Simulator_Audio_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before TCAS_Simulator_Audio is made visible.
function TCAS_Simulator_Audio_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to TCAS_Simulator_Audio (see VARARGIN)
```

```
% Choose default command line output for TCAS_Simulator_Audio
%% declearing global structure variables for figure TCAS_Simulator_Audio

global simulation_mode

simulation_mode = 1;
global my_server
global my_config
global bau
global sat
global spd
 sat = '1';
 spd = '1';
%bau = '4800';

%% setting the timer for updating, updates every two seconds
handles.update_timer = timer('StartDelay', 2, ...
                    'period', 2, ....
                    'ExecutionMode', 'fixedRate', ...
                    'TimerFcn',{@TimerFcn_Callback,hObject});

my_server = handles;

handles.output = hObject;
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes TCAS_Simulator_Audio wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = TCAS_Simulator_Audio_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%% function timer call to update server
function TimerFcn_Callback(timer_object, eventdata, hObject)
% provides access to the figure's handles structure
handles = guidata(hObject);

global s1
global a_long
global A_lat
global A_Speed
global A_Direction
global A_Dir
global b_long
global b_lat
global b_speed
global b_dir
```

```
global b_Direction
global temP_Blat
global sat

  try
     % get information for TCAS A from Server console
     fprintf(s1,'a'); %send string a to server console to get TCASA info
     com_tcasa = fscanf(s1,'%s'); % read the serial port for any in put
     size(com_tcasa);       %size of the receivced data from serial port
     length(com_tcasa)        %size of the receivced data from serial port
if isempty(com_tcasa)      % check if the serial port is empty
      disp('empty')           %if empty assign 0 to lat and longitude
      A_lat = 0 ;
      a_long = 0;

    deactivate_com_tcasa(handles)  %loss in communication of tcasa
else

   activate_com_tcasa(handles); % if TcasA data is received console server
 if (length(com_tcasa) <= 22)    % check lenght for complete info and if speed not received
     temp_A_lat = A_lat;
     A_lat = com_tcasa(1:9);       % extract latitude
     a_long = com_tcasa(11:20) ;    % extract longitude
     A_Speed = '0.00' ;           % assign zero to speed

 else                          % if the infor is complete
    temP_Alat = A_lat;% save previous value of latitude to a temp variable
    A_lat = com_tcasa(1:9);           % extract latitude
    a_long = com_tcasa(11:20);         % extract longitude
    A_Speed = com_tcasa(23:26);          % extract speed

 end
end % end of acquiring communication for TCAS A
% get information for TCAS B from Server console
     fprintf(s1,'b'); %send string b to server console to get tcas b info
     com_tCasb = fscanf(s1,'%s');
     lengthtcasb = length(com_tCasb);
  if isempty(com_tCasb)      % check if the serial port is empty
     disp('empty')
    deactivate_com_tcasb(handles);   %loss in communication of tcasb
     else
      activate_com_tcasb(handles);
  if (length(com_tCasb) <= 22)
      temP_Blat = b_lat;
      b_lat = com_tCasb(1:9);
      b_long = com_tCasb(11:20);
      b_speed = '0.00' ;
  else
     temP_Blat = b_lat ;
     b_lat = com_tCasb(1:9);
     b_long = com_tCasb(11:20);
     b_speed = com_tCasb(23:26);
   end
  end
     update_info(handles);       % update server with new info
catch err                % if no connection display error dialog box
    errmsg = err.message;
    errordlg('Error in update','error in update');
```

```
  end
guidata(hObject, handles);

% --- Executes on button press in server_console.
function server_console_Callback(hObject, eventdata, handles)
% hObject    handle to server_console (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of server_console
global s1
global a_long
global A_lat
 figure (handles.figure1);
 %global menu
a = get (handles.server_console,'Value'); % check to see if console is not clicked
if (a == 1)        % if yes connect and change color from red to green
   ser = instrhwinfo('serial');
   sed = ser.AvailableSerialPorts; % get available serial com ports
 if strcmp( 'COM6',sed)
    s1 = serial('COM6');
 end
     if strcmp( 'COM5',sed)
        s1 = serial('COM5');
     end
     if strcmp( 'COM4',sed)
        s1 = serial('COM4');
     end
     if strcmp( 'COM3',sed)
        s1 = serial('COM3');
     end
    if strcmp( 'COM2',sed)
        s1 = serial('COM2');
    end
     if strcmp( 'COM1',sed)
        s1 = serial('COM1');
     end
    set(s1,'BaudRate',4800);  %baud rate for serial communication
   set(s1,'Timeout',10);
   try
     fopen(s1);        %open serial port for connection
     if isequal (get(s1,'Status'),'open')
    a = 'Open' ; % check if there is serial communication on port
     end

   catch err                    % if no connection display error dialog box
      errmsg = err.message
      set (handles.server_console,'Value', 0);
         errordlg('Server not found','No Connection');%display error message
   end
   if isequal (get(s1,'Status'),'open')
    set(handles.server_console,'BackgroundColor',[0,0.502,0.251]); % change color to green
    set(handles.server_console,'String','Connected'); % display Connected
    set (handles.server_console,'Value', 1);
    start(handles.update_timer);       % start timer to update server information
   end
end
% Disconect server console
if (a ==0)                          % check to see if console is unclicked
```

```
   stop(handles.update_timer);        % stop the update timer

 if isequal (get(s1, 'Status'),'open')
   A_lat = 0 ;
   a_long = 0;

 fclose(s1);
  end
    %set the communication indicator to red and lost
    deactivate_com_tcasb(handles);
    deactivate_com_tcasa(handles);
    % set the sever console indicator to red and dsiconnected
    set(handles.server_console,'BackgroundColor',[1,0,0]); % change color to red
    set(handles.server_console,'String','Disconnected');   % display Disconnected
    set(handles.server_console,'Value', 0);

end

function A_locate_long_Callback(hObject, eventdata, handles)
% hObject    handle to A_locate_long (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of A_locate_long as text
%        str2double(get(hObject,'String')) returns contents of A_locate_long as a double
% --- Executes during object creation, after setting all properties.
function A_locate_long_CreateFcn(hObject, eventdata, handles)
% hObject    handle to A_locate_long (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%  See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
 function A_Speed_Callback(hObject, eventdata, handles)
% hObject    handle to A_Speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of A_Speed as text
% str2double(get(hObject,'String')) returns contents of A_Speed as a double

% --- Executes during object creation, after setting all properties.
function A_Speed_CreateFcn(hObject, eventdata, handles)
% hObject    handle to A_Speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%  See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function time_Callback(hObject, eventdata, handles)
% hObject    handle to time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 % Hints: get(hObject,'String') returns contents of time as text
```

```
%        str2double(get(hObject,'String')) returns contents of time as a double
 % --- Executes during object creation, after setting all properties.
function time_CreateFcn(hObject, eventdata, handles)
% hObject    handle to time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function A_Direction_Callback(hObject, eventdata, handles)
% hObject    handle to A_Direction (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of A_Direction as text
%        str2double(get(hObject,'String')) returns contents of A_Direction as a double
% --- Executes during object creation, after setting all properties.

function A_Direction_CreateFcn(hObject, eventdata, handles)
% hObject    handle to A_Direction (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
 % --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 % Hint: get(hObject,'Value') returns toggle state of radiobutton2
%global menu
 % --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton3
 function B_locate_long_Callback(hObject, eventdata, handles)
% hObject    handle to B_locate_long (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of B_locate_long as text
%        str2double(get(hObject,'String')) returns contents of B_locate_long as a double
% --- Executes during object creation, after setting all properties.
function B_locate_long_CreateFcn(hObject, eventdata, handles)
% hObject    handle to B_locate_long (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end
function B_Speed_Callback(hObject, eventdata, handles)
% hObject    handle to B_Speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of B_Speed as text
%        str2double(get(hObject,'String')) returns contents of B_Speed as a double
% --- Executes during object creation, after setting all properties.
function B_Speed_CreateFcn(hObject, eventdata, handles)
% hObject    handle to B_Speed (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit8_Callback(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of edit8 as text
%        str2double(get(hObject,'String')) returns contents of edit8 as a double
% --- Executes during object creation, after setting all properties.
function edit8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function B_Direction_Callback(hObject, eventdata, handles)
% hObject    handle to B_Direction (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of B_Direction as text
%        str2double(get(hObject,'String')) returns contents of B_Direction as a double
% --- Executes during object creation, after setting all properties.
function B_Direction_CreateFcn(hObject, eventdata, handles)
% hObject    handle to B_Direction (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Dist_calculate_Callback(hObject, eventdata, handles)
% hObject    handle to Dist_calculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: get(hObject,'String') returns contents of Dist_calculate as text
```

```
%        str2double(get(hObject,'String')) returns contents of Dist_calculate as a double
% --- Executes during object creation, after setting all properties.
function Dist_calculate_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Dist_calculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes during object creation, after setting all properties.
function resolution_CreateFcn(hObject, eventdata, handles)
% hObject    handle to resolution (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%% function to update all information on windows for TCAS A and TCAS B
function handles = update_info(handles)
figure (handles.figure1);
global sat
global a_long;
global A_lat;
global A_Speed;
global A_Direction;
global A_Dir;
%global conversion; % variable to hold conversion status
global b_long;
global b_lat;
global b_speed;
global b_dir;
global b_Direction;
global temP_Blat;
global spd;
time = 0.5;
clear(handles);

switch (sat)

    case ('1') % update for GPS raw data

    display_sat_Data(handles);  % display GPS data in raw format
    %tcasA update
    pro_long_a_raw = num2str(a_long);
    set(handles.A_locate_long,'String', pro_long_a_raw);
    pro_lat_a_raw = num2str(A_lat);

    set(handles.A_locate_latt,'String', pro_lat_a_raw);
    set(handles.A_Speed,'String', A_Speed);
    %tcasB update
```

```
    pro_long_b_raw = num2str(b_long);
    set(handles.B_locate_long,'String', pro_long_b_raw);
    pro_lat_b_raw = num2str(b_lat);
    set(handles.B_locate_latt,'String', pro_lat_b_raw);

     if (spd == '1'); % display speed in nautical miles
        display_spd_in_knot(handles);
        set(handles.B_Speed,'String', b_speed);
    else
        display_spd_in_km(handles);  % display speed in Kilometer
        b_speed2nm2km = nm2km(str2double(b_speed));
        set(handles.B_Speed,'String', b_speed2nm2km);

     end

LatDirect_tcasB(handles);  % process the direction of TCAS B using this function

    case ('2')  % Display latitude and longitude in Degree

        display_in_Degrees(handles) % display GPS data in degree

        pro_long_b = num2str(convlong(b_long));
        set(handles.B_locate_long,'String', pro_long_b);    % % display the longitude of tcasB

        pro_lat_ = convlat(b_lat);
        pro_lat_b = num2str(pro_lat_);
        set(handles.B_locate_latt,'String', pro_lat_b);     % display the latitude of tcasB

  if (spd == '1');
    display_spd_in_knot(handles);
    set(handles.B_Speed,'String', b_speed);
  else
     display_spd_in_km(handles);
     b_speed2nm2km = nm2km(str2double(b_speed));
     set(handles.B_Speed,'String', b_speed2nm2km);

  end

 LatDirect_tcasB(handles);  % display the direction of TCAS B

end
%------------function to calculate approach time--------------------

% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global s1;

 % check if serial port is connected and disconnect before closing down.

if isequal (get(s1, 'Status'),'open')

     fclose(s1);     % close serial port
    delete(s1);     %delete serial port object
```

```
 end
% stop(handles.update_timer);
  delete(handles.update_timer);
% Hint: delete(hObject) closes the figure

delete(hObject);

function B_locate_latt_Callback(hObject, eventdata, handles)
% hObject    handle to B_locate_latt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of B_locate_latt as text
%        str2double(get(hObject,'String')) returns contents of B_locate_latt as a double

% --- Executes during object creation, after setting all properties.
function B_locate_latt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to B_locate_latt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function A_locate_latt_Callback(hObject, eventdata, handles)
% hObject    handle to A_locate_latt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of A_locate_latt as text
%        str2double(get(hObject,'String')) returns contents of A_locate_latt as a double

% --- Executes during object creation, after setting all properties.
function A_locate_latt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to A_locate_latt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

 % --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on selection change in listbox3.
function listbox3_Callback(hObject, eventdata, handles)
% hObject    handle to listbox3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox3

% --- Executes during object creation, after setting all properties.
function listbox3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in listbox2.
function listbox2_Callback(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox2

% --- Executes during object creation, after setting all properties.
function listbox2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in listbox4.
function listbox4_Callback(hObject, eventdata, handles)
% hObject    handle to listbox4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox4 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox4

% --- Executes during object creation, after setting all properties.
```

```
function listbox4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --------------------------------------------------------------------
function serial_port_Callback(hObject, eventdata, handles)
% hObject    handle to serial_port (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%open('TCAS_Config.fig')

% --------------------------------------------------------------------
function Com_Callback(hObject, eventdata, handles)
% hObject    handle to Com (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%open('TCAS_Config.fig')
run TCAS_Config

% --------------------------------------------------------------------
function Conv_deg_Callback(hObject, eventdata, handles)
% hObject    handle to Conv_deg (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global sat
sat = '2';

function handles = display_in_Degrees(handles)
set(handles.disp_deg_Long_b,'String', 'deg');
set(handles.disp_deg_Lat_b,'String', 'deg');
set(handles.disp_deg_Lat_a,'String', 'deg');
set(handles.disp_deg_Long_a,'String', 'deg');

% --------------------------------------------------------------------
% displays the speed in nautical miles
function conv_nau_Callback(hObject, eventdata, handles)
% hObject    handle to conv_nau (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global spd % set the global variable spd to 1
spd = '1';

function handles = display_spd_in_knot(handles) % function to display speed in knot
set(handles.disp_km,'String', 'nau');
set(handles.disp_km_b,'String', 'nau');
% --------------------------------------------------------------------

% Displays the speed in Kilometers per second
function convspeed_km_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to convspeed_km (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global spd
spd = '2';


function handles = display_spd_in_km(handles)  % function to display speed in Km
set(handles.disp_km,'String', 'km/h');
set(handles.disp_km_b,'String', 'km/h');

% --- Executes during object creation, after setting all properties.
function disp_km_CreateFcn(hObject, eventdata, handles)
% hObject    handle to disp_km (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% --- Executes during object creation, after setting all properties.
function disp_km_b_CreateFcn(hObject, eventdata, handles)
% hObject    handle to disp_km_b (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% --------------------------------------------------------------------
function conv_rad_Callback(hObject, eventdata, handles)
% hObject    handle to conv_rad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.disp_deg_Long_b,'String', 'rad');
set(handles.disp_deg_Lat_b,'String', 'rad');
set(handles.disp_deg_Lat_a,'String', 'rad');
set(handles.disp_deg_Long_a,'String', 'rad');

% --------------------------------------------------------------------
function satelite_data_Callback(hObject, eventdata, handles)
% hObject    handle to satelite_data (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global sat;
sat = '1';

function handles = display_sat_Data(handles)

set(handles.disp_deg_Long_b,'String', '');
set(handles.disp_deg_Lat_b,'String', '');
set(handles.disp_deg_Lat_a,'String', '');
set(handles.disp_deg_Long_a,'String', '');

% --- Executes on button press in Start_simulator.
function Start_simulator_Callback(hObject, eventdata, handles)
% hObject    handle to Start_simulator (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
increase_long = 0;
```

```
long_ATT = get(handles.A_locate_long, 'String')

if isempty(long_ATT)

    hhh = 1
else
    while increase_long ~= 20
   total = str2double(long_ATT)+ 1
   increase_long + 1;
    end
end

% --- Executes on button press in toggle_start_simulator.
function toggle_start_simulator_Callback(hObject, eventdata, handles)          % toggle button to start
simulation
% hObject    handle to toggle_start_simulator (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of toggle_start_simulator

global current_data_TCAS_B   % variable to store TCAS B speed

global current_data_TCAS_A   % variable to store TCAS A speed

global move_A    %inital position of TCAS A on the x-axis
global move_B  %inital position of TCAS B on the x-axis
global move_B_Lat  % initial position of TCAS B y-axis (latitude)
global move_A_Lat  % initial position of TCAS A y-axis (latitude)

global simulation_mode

global curr_long_ATT;
global dist;
global curr_B_long_ATT;
global apptime;

increase_long = 0;

%AUTOMATION
for t = 1 : 1: 2
set(handles.A_locate_long, 'String',7.433624); %7.4336241   A_locate_long
set(handles.B_locate_long, 'String',7.717896); %B_locate_long
set(handles.A_locate_latt, 'String',10.544521);
set(handles.B_locate_latt, 'String', 11.113727);

set(handles.time, 'String','');
set(handles.Dist_calculate, 'String', '');

move_A = 4.400000000000005;
move_B = 124.00000000000227;
TCASA_Alt = get(handles.popupmenu4_TCASA_alt,'Value');
if(TCASA_Alt==2)
  move_A_Lat = 6.4615384615384315
elseif(TCASA_Alt==3)
   move_A_Lat = 6.5615384615384315
```

```
elseif(TCASA_Alt==4)
   move_A_Lat = 9.4615384615384315
elseif(TCASA_Alt==5)
   move_A_Lat = 12.4615384615384315
elseif(TCASA_Alt==6)
   move_A_Lat = 15.4615384615384315
elseif(TCASA_Alt==7)
   move_A_Lat = 18.4615384615384315
elseif(TCASA_Alt==8)
   move_A_Lat = 21.4615384615384315
end
%fprintf('%1.4f\n', move_A_Lat)

TCASB_Alt = get(handles.popupmenu5_TCASB_alt,'Value');
if(TCASB_Alt==2)
  move_B_Lat = 6.4615384615384315
elseif(TCASB_Alt==3)
   move_B_Lat = 6.5615384615384315
elseif(TCASB_Alt==4)
   move_B_Lat = 9.4615384615384315
elseif(TCASB_Alt==5)
   move_B_Lat = 12.4615384615384315
elseif(TCASB_Alt==6)
   move_B_Lat = 15.4615384615384315
elseif(TCASB_Alt==7)
   move_B_Lat = 18.4615384615384315
elseif(TCASB_Alt==8)
   move_B_Lat = 21.4615384615384315
end

%fprintf('%1.4f\n', move_A_Lat)

set(handles.img_tcasA,'Position', [move_A move_A_Lat  17.200000000000003 5.538461538461541]);
set(handles.img_tcasb,'Position', [move_B move_B_Lat  17.200000000000003 5.538461538461541]);
% disp 'move_A_Lat'
% disp 'move_B_Lat'

long_ATT = get(handles.A_locate_long, 'String');  % get  TCAS A longitude
lat_ATT = get (handles.A_locate_latt, 'String');  % get TCAS A latitude

B_long_ATT = get(handles.B_locate_long, 'String');  % get TCAS B longitude
B_lat_ATT = get(handles.B_locate_latt, 'String');   % get TCAS B latitude


speedA = current_data_TCAS_A; % get TCAS A  speed
%speedA = get(handles.A_Speed, 'String');
%speedB = get(handles.B_Speed, 'String');
speedB = current_data_TCAS_B; % get TCAS B

set(handles.toggle_start_simulatoo, 'Visible', 'off');
set(handles.toggle_start_simulatool, 'Visible', 'off');
if (simulation_mode == 1)

move_A = 4.400000000000005;
move_B = 124.00000000000227;
%move_B_Lat = 6.4615384615384315;
          %6.3846153846153895
```

```
% move_A_Lat = 6.4615384615384315;

button_state = get(hObject,'Value'); % get the value of state of start button

if button_state == get(hObject,'Max')
%switch (button_state)

    alg =  str2double(long_ATT);
    all = str2double(lat_ATT);
    blg = str2double(B_long_ATT);
    bll = str2double(B_lat_ATT);

   % asp = str2double(current_data_TCAS_A)
    asp = current_data_TCAS_A % current speed of TCAS A

    bsp = current_data_TCAS_B % current speed of TCAS B

    if isempty(asp) || isempty(bsp)
      errordlg('please select speed for TCAS A and TCAS B');

       else
          fprintf('long_A\tLat_A\tLong_B\tLat_B\tDist\tTAU\n')
          %for i = 1 : 1: 182 ,370

          for i = 1 : 1: 186
              button_state = get(hObject,'Value');
              if button_state == 0
                 %set(handles.toggle_start_simulator,'Value',0);
% %              errordlg('please enter data')
               continue
              end

               if(current_data_TCAS_A == 250)
                 pause(0.1)
              elseif(current_data_TCAS_A == 300)
                 pause(0.05)
              elseif(current_data_TCAS_A == 350)
                 pause(0.02)
              elseif(current_data_TCAS_A == 400)
             pause(0.01) % pause every 0.01 second to determine the speed
              end
               % move the two planes in opposite direction
move_A = move_A + 0.3; % increment and  move the TCAS A in the x direction step of 0.3
move_B = move_B - 0.3; % decrement and move the TCAS B in the  x direction step of -0.3

myPosition_A = [move_A move_A_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_B_Lat  17.200000000000003 5.538461538461541];
              set (handles.img_tcasA,'Position',myPosition_A);
              set (handles.img_tcasb,'Position',myPosition_B);

              %alg = alg - 0.151599;  % increment the value of longitude of TCAS A to crrespond to new
position of TCAS A
              alg = alg + 0.000768;
              blg = blg - 0.000768;
              all = all + 0.001538;
              bll = bll - 0.001538;
```

```
            set(handles.A_locate_long, 'String',alg);
            set(handles.B_locate_long, 'String',blg);
            set(handles.A_locate_latt, 'String',all);
            set(handles.B_locate_latt, 'String',bll);

            % calculate distance of the two TCAS

            curr_long_ATT = str2double(get(handles.A_locate_long,
                        'String'));  % get  TCAS A longitude
            curr_lat_ATT = str2double(get (handles.A_locate_latt,
                        'String'));  % get TCAS A latitude
            curr_B_long_ATT = str2double( get(handles.B_locate_long,
                  'String'));  % get TCAS B longitude
            curr_B_lat_ATT = str2double(get(handles.B_locate_latt,
                  'String'));   % get TCAS B latitude
                dist = distance(all,-alg,bll,-blg);
                Distance = deg2km(dist);
                %Dist_calculate = Distance/1.852;
                 Dist_calculate = (deg2km(dist)/1.852);
                %Dist_calculate_nm = Dist_calculate/1.852;
                %apptime = Dist_calculate/(asp + bsp);
                %approach_time = Dist_calculate/(asp + bsp)
                apptime = (Dist_calculate/(asp + bsp))*3600;
                %apptime = approach_time/3600
                %fprintf('%4.4f\t%4.4f\t%4.4f\t%4.4f\n', all,alg,bll,blg)
                set(handles.Dist_calculate,'String',Dist_calculate);
                set(handles.time,'String', apptime);
                Long_A(i) = curr_long_ATT;
                Long_B(i) = curr_B_long_ATT;
                Lat_A(i) =  curr_lat_ATT;
                Lat_B(i) = curr_B_lat_ATT;
                Distance(i) = Dist_calculate;
                TAU(i) = apptime;
                %fprintf('%d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t%4.6f\n',
i,Long_A(i),Lat_A(i),Long_B(i),Lat_B(i),Distance(i),TAU(i))

                fprintf('%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\n',
Long_A(i),Lat_A(i),Long_B(i),Lat_B(i),Distance(i),TAU(i))
                %dlmwrite('TCAS_TAU_REST.txt',TAU,',')
                fileID = fopen('TCAS_RESexmp.txt', 'w');
                fprintf(fileID,'TAU\n');
                fprintf(fileID,'%4.2f\n', TAU);
                fclose(fileID);

 if (move_B_Lat-move_A_Lat)<1
                %if (apptime <  60.0000)
                 if (apptime <  61.0000)
                 %set(handles.uipanel10, 'Visible', 'on');
                 %set(handles.text42_detect, 'String','60 s')

                end

                 if (apptime <52.000)

                  %set(handles.uipanel10, 'Visible', 'off');
                 end
                if (apptime <46.000)&& (apptime > 44.0)
```

```
                    %Traffic advisory is given within 45 seconds;

                    set(handles.resolution,'String','Traffic')
                    [Z,fs,nbits] =wavread ('warning2.wav');
                    soundsc(Z,fs);

                end
              if (apptime <38.000)&& (apptime > 37.00 )
%Intruder declared as a threat within 35 seconds
%
                    set(handles.resolution,'String','Collision')
%                     %set(handles.text42_detect, 'String','35 s')
%
                    [Z,fs,nbits] =wavread ('COLLISION2.wav'); %CHANGE RECORD TO 'COLLISION'

                    soundsc(Z,fs);
                end

            if (apptime < 36.9999 ) && (apptime > 30.0 )  % Corrective RA  is given within 30
               and 25 seconds for vertical and lateral respectively
                    set(handles.resolution,'String','Change Direction')
                    %set(handles.text42_detect, 'String','27 s')
                    [Z,fs,nbits] =wavread ('CHANGEDIRECT2.wav');
                    soundsc(Z,fs);
                    %disp 'Change Direction'

                   for j= 1 :1: 4

                                        move_A = move_A + 0.8; % increment and  move the TCAS A in the
                                        x direction step of 0.4
                                        move_B = move_B - 0.8; % decrement and move the TCAS B in the
                                        x direction step of -0.4
                                        move_B_Lat =  move_B_Lat + 0.3953846153846164;
                                        move_A_Lat =  move_A_Lat - 0.390769230769242;
                                        (handles.img_tcasA,'Position',myPosition_A);
                                        set(handles.img_tcasb,'Position', myPosition_B);
                        if(current_data_TCAS_A == 250)
                           pause(0.1)
                       elseif(current_data_TCAS_A == 300)
                           pause(0.05)
                       elseif(current_data_TCAS_A == 350)
                           pause(0.02)
                       elseif(current_data_TCAS_A == 400)
                  pause(0.01) % pause every 0.01 second to determine the speed
                           end
                            end
                 end
               if (apptime <5.000) % indicate clear
                  set(handles.resolution,'String','Clear')
                  %set(handles.text42_detect, 'String','5 s')
               end

               if (apptime <5.000)&&(apptime >1.000) % indicate clear
                  set(handles.resolution,'String','NO TRAFFIC')
                  %set(handles.text42_detect, 'String','')
                  %disp 'NO TRAFFIC'
                  [Z,fs,nbits] =wavread ('NOTRAFFIC2.wav');
```

```
                soundsc(Z,fs);
              end

            if (apptime <0.9999) % indicate clear
                set(handles.resolution,'String','NO TRAFFIC')
                %set(handles.text42_detect, 'String','')
%                   [Z,fs,nbits] =wavread ('NOTRAFFIC.wav');
%
%                    soundsc(Z,fs);
              break
            end
               if button_state == get(hObject,'Min')
                  set(handles.Dist_calculate,'String','');
                  set(handles.time,'String', '');
                  return
               end
  elseif (apptime <46.000)&& (apptime > 45.0) % No Traffic advisory is given
          within 45 seconds;
     set(handles.resolution,'String','NO TRAFFIC')
     [Z,fs,nbits] =wavread ('NOTRAFFIC2.wav');
     soundsc(Z,fs);
   end
          end

%            CONTINUE TRAFFIC
%
             %get(handles.text34,'Position');
             get(handles.img_tcasA,'Position');
             get(handles.img_tcasb,'Position');
             curr_long_ATT = str2double(get(handles.A_locate_long,
                      'String'));  % get  TCAS A longitude
             curr_lat_ATT = str2double(get (handles.A_locate_latt,
                      'String'));  % get TCAS A latitude
             curr_B_long_ATT = str2double( get(handles.B_locate_long,
                      'String'));  % get TCAS B longitude
             curr_B_lat_ATT = str2double(get(handles.B_locate_latt,
                      'String'));   % get TCAS B latitude

          for n = 1 : 1:185
%
             if(current_data_TCAS_A == 250)
                pause(0.1)
             elseif(current_data_TCAS_A == 300)
                pause(0.05)
             elseif(current_data_TCAS_A == 350)
                pause(0.02)
             elseif(current_data_TCAS_A == 400)
              pause(0.01)
                              % pause every 0.01 second to determine the speed
             end

myPosition_A = [move_A move_A_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_B_Lat  17.200000000000003 5.538461538461541];
                set(handles.img_tcasA,'Position', myPosition_A)
                %set(handles.text35,'Position', myPosition_B);
                 set(handles.img_tcasb,'Position', myPosition_B);
                 % adjust the location (Lat & Long)of the two TCAS A and B
                 % respectively
```

```
                set(handles.A_locate_long, 'String',alg);
                set(handles.B_locate_long, 'String',blg);
                set(handles.A_locate_latt, 'String',all);
                set(handles.B_locate_latt, 'String',bll);


                %alg = alg - 0.151599;  % increment the value of longitude of TCAS A to  crrespond to
                 new position of TCAS A
                alg = alg + 0.000768;
                blg = blg - 0.000768;
                all = all + 0.001538;
                bll = bll - 0.001538;

 %alg = alg - 0.151599;  % decrement the value of longitude of TCAS A to crrespond to new position of
TCAS A
%dist = distance(curr_long_ATT,-curr_lat_ATT,curr_B_lat_ATT,-curr_B_long_ATT);
            %NewD = (deg2km(distance(10.54,-7.43,11.11,-7.71))/1.852)
                dist = distance(all,-alg,bll,-blg);
                 Distance = deg2km(dist);
                 %Dist_calculate = Distance/1.852;
                 Dist_calculate = (deg2km(dist)/1.852);
                %Dist_calculate_nm = Dist_calculate/1.852;
                %apptime = Dist_calculate/(asp + bsp);
                %approach_time = Dist_calculate/(asp + bsp)
                apptime = (Dist_calculate/(asp + bsp))*3600;
                %apptime = approach_time/3600
                                set(handles.Dist_calculate,'String',Dist_calculate);
                set(handles.time,'String', apptime);
                Long_A(n) = alg;
                Long_B(n) = blg;
                Lat_A(n) =  all;
                Lat_B(n) =  bll;
                Distance(n) = Dist_calculate;
                TAU(n) = apptime;
                fprintf('%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\n',
Long_A(n),Lat_A(n),Long_B(n),Lat_B(n),Distance(n),TAU(n));
                if (apptime <21.000)&& (apptime > 19.0) % No Traffic
                                advisory is given within 45 seconds;
                   set(handles.resolution,'String','NO TRAFFIC')
                   [Z,fs,nbits] =wavread ('NOTRAFFIC2.wav');
                   soundsc(Z,fs);
                   %disp 'NOTRAFFIC'
                end
                if(apptime <65.000)&& (apptime >60.0)
                   for k= 1 :1: 2
move_A = move_A + 0.8; % increment and  move the TCAS A in the x direction step of 0.4
                     move_B = move_B - 0.8; % decrement and move
                                        the TCAS B in the  x direction step of -0.4
                     move_B_Lat =  move_B_Lat - 0.19534615384621;
                                        %0.3953846153846164;
                     move_A_Lat =  move_A_Lat + 0.195384615384640;
                                        %0.19999999999999;                  %0.099673076923105;
                                        %0.2441826923077625          %0.097673076923105          ;
                                        %0.19534615384621;    %0.390769230769242;
%                       disp 'move_A_Lat'
myPosition_A = [move_A move_A_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_B_Lat  17.200000000000003 5.538461538461541];
```

```
% set(handles.text34,'Position', myPosition_A);
                                        set (handles.img_tcasA,'Position',myPosition_A);
                    set(handles.img_tcasb,'Position', myPosition_B);
                         if(current_data_TCAS_A == 250)
                           pause(0.1)
                         elseif(current_data_TCAS_A == 300)
                           pause(0.05)
                         elseif(current_data_TCAS_A == 350)
                           pause(0.02)
                         elseif(current_data_TCAS_A == 400)
                          pause(0.01)
                                        % pause every 0.01 second to determine the speed
                         end


                          end
                    end
                    if button_state == get(hObject,'Min')
                       set(handles.Dist_calculate,'String','');
                       set(handles.time,'String', '');
                       %set(handles.toggle_start_simulator,'Value',0);
                       return
                    end
                    end
     end
   elseif button_state == get(hObject,'Min')
          %case 0      %  pressed to start/stop button to stop
          button_state == get(hObject,'Min')
          % Toggle button is not pressed, take appropriate action
          %reset the longitude
          set(handles.toggle_start_simulator, 'String', 'Start Simulator');
          set(handles.resolution, 'String', ' ');
          set(handles.A_locate_long, 'String',7.433624);
                      %7.4336241   A_locate_long
          set(handles.B_locate_long, 'String',7.717896); %B_locate_long
          set(handles.toggle_start_simulator,'Value',0);
          set(handles.img_tcasA,'Position', [move_A move_A_Lat
                17.200000000000003 5.538461538461541]);
          set(handles.img_tcasb,'Position', [move_B move_B_Lat
                      17.200000000000003 5.538461538461541]);
end
end
end
% --- Executes on selection change in popupmenu_TCAS_B.
function popupmenu_TCAS_B_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_TCAS_B (see GCBO)
global current_data_TCAS_B
% Determine the selected data set.
     str = get(hObject, 'String');
     val = get(hObject,'Value');
     % Set current data to the selected data set.
     switch str{val};
     case '250' % User selects Peaks.
       current_data_TCAS_B = 250;
     case '300' % User selects Membrane.
       current_data_TCAS_B = 300;
     case '350' % User selects Sinc.
       current_data_TCAS_B = 350;
     case '400' % User selects Sinc.
```

```
        current_data_TCAS_B = 400;
    end
% --- Executes during object creation, after setting all properties.
function popupmenu_TCAS_B_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu_TCAS_B (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
 % --- Executes on selection change in popupmenu_TCAS_A.
function popupmenu_TCAS_A_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu_TCAS_A (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu_TCAS_A contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu_TCAS_A
global current_data_TCAS_A
% Determine the selected data set.
     str = get(hObject, 'String');
    val = get(hObject,'Value');
    % Set current data to the selected data set.
    switch str{val};
    case '250' % User selects Peaks.
       current_data_TCAS_A = 250;
    case '300' % User selects Membrane.
       current_data_TCAS_A = 300;
    case '350' % User selects Sinc.
       current_data_TCAS_A = 350;
    case '400' % User selects Sinc.
       current_data_TCAS_A = 400;
    end
% --- Executes during object creation, after setting all properties.
function popupmenu_TCAS_A_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu_TCAS_A (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
% --- Executes on button press in togglebutton2.
function togglebutton2_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of togglebutton2
% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu3
% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --------------------------------------------------------------------
function same_altitude_Callback(hObject, eventdata, handles)
% hObject    handle to same_altitude (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set(handles.A_locate_long, 'String',7.433624); %7.4336241   A_locate_long
set(handles.B_locate_long, 'String',7.717896); %B_locate_long
set(handles.A_locate_latt, 'String',10.544521);
set(handles.B_locate_latt, 'String', 11.113727);
% set(handles.text34, 'Visible', 'on');
% set(handles.text35, 'Visible', 'on');
set(handles.toggle_start_simulator, 'Visible', 'on');
set(handles.toggle_start_simulatoo, 'Visible', 'off');
set(handles.toggle_start_simulatool, 'Visible', 'off');
% set(handles.text61, 'Visible', 'off');
% set(handles.text62, 'Visible', 'off');
% set(handles.text63, 'Visible', 'off');
% set(handles.text64, 'Visible', 'off');
set(handles.time, 'String',' ');
set(handles.Dist_calculate, 'String',' ');
global simulation_mode
simulation_mode = 1;

% --------------------------------------------------------------------
function different_altitude_Callback(hObject, eventdata, handles)
% hObject    handle to different_altitude (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

set(handles.toggle_start_simulatoo, 'Visible', 'on');
% set(handles.text61, 'Visible', 'on');
% set(handles.text62, 'Visible', 'on');
% %set(handles.text63, 'Visible', 'off');
% %set(handles.text64, 'Visible', 'off');
% set(handles.text34, 'Visible', 'off');
% set(handles.text35, 'Visible', 'off');
set(handles.toggle_start_simulator, 'Visible', 'off');
set(handles.toggle_start_simulatool, 'Visible', 'off');

global simulation_mode

set(handles.A_locate_long, 'String','7.545750');  % set  TCAS A longitude
set (handles.A_locate_latt, 'String','10.769100');  % set TCAS A latitude
set(handles.B_locate_long, 'String','7.605770');  % set TCAS B longitude
set(handles.B_locate_latt, 'String','10.889200');   % set TCAS B latitude
set(handles.time, 'String',' ');
set(handles.Dist_calculate, 'String',' ');
simulation_mode = 2;

%DIFFERNT ALTITUDE
```

```
% --- Executes on button press in toggle_start_simulatoo.
function toggle_start_simulatoo_Callback(hObject, eventdata, handles)
% hObject    handle to toggle_start_simulatoo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of toggle_start_simulatoo
global current_data_TCAS_B   % variable to store TCAS B speed
global current_data_TCAS_A   % variable to store TCAS A speed
global move_A    %inital position of TCAS A on the x-axis
global move_B  %inital position of TCAS B on the x-axis
global move_BB_Lat  % initial position of TCAS B y-axis (latitude)
global move_AA_Lat  % initial position of TCAS A y-axis (latitude)
global curr_long_ATT;
global dist;
global curr_B_long_ATT;
global apptime;
%increase_long = 0;
%set(handles.text61,'Position', myPosition_A);
 %AUTOMATION
for t = 1 : 1: 2
set(handles.A_locate_long, 'String',7.433624); %7.4336241   A_locate_long
set(handles.B_locate_long, 'String',7.717896); %B_locate_long
set(handles.A_locate_latt, 'String',10.544521);
set(handles.B_locate_latt, 'String', 11.113727);
set(handles.time, 'String','');
set(handles.Dist_calculate, 'String', '');
move_BB_Lat = 9.4758;
move_AA_Lat = 3.0308;
move_A = 22.8000;
move_B = 96.2000;

set(handles.img_tcasA,'Position', [move_A move_AA_Lat  17.200000000000003 5.538461538461541]);
set(handles.img_tcasb,'Position', [move_B move_BB_Lat  17.200000000000003 5.538461538461541]);
fprintf('%1.4f\n', move_AA_Lat)
fprintf('%1.4f\n', move_BB_Lat)

set(handles.A_locate_long, 'String','7.545750');  % set  TCAS A longitude
set (handles.A_locate_latt, 'String','10.769100');  % set TCAS A latitude
set(handles.B_locate_long, 'String','7.605770');  % set TCAS B longitude
set(handles.B_locate_latt, 'String','10.889200');   % set TCAS B latitude
long_ATT = get(handles.A_locate_long, 'String');  % get  TCAS A longitude
lat_ATT = get (handles.A_locate_latt, 'String');  % get TCAS A latitude
B_long_ATT = get(handles.B_locate_long, 'String');  % get TCAS B longitude
B_lat_ATT = get(handles.B_locate_latt, 'String');   % get TCAS B latitude

speedA = current_data_TCAS_A; % get TCAS A  speed
%speedA = get(handles.A_Speed, 'String');
%speedB = get(handles.B_Speed, 'String');
speedB = current_data_TCAS_B; % get TCAS B

global simulation_mode
if (simulation_mode == 2)
      button_state = get(hObject,'Value'); % get the value of state of start button
      if button_state == get(hObject,'Max')
            set(handles.toggle_start_simulatoo, 'String', 'Stop');
```

```
                %convert the string to numbers
                alg =  str2double(long_ATT);
                all = str2double(lat_ATT);
                blg = str2double(B_long_ATT);
                bll = str2double(B_lat_ATT);
               % asp = str2double(current_data_TCAS_A)
                asp = current_data_TCAS_A % current speed of TCAS A
                bsp = current_data_TCAS_B % current speed of TCAS B
                if isempty(asp) || isempty(bsp)
                errordlg('please select speed for TCAS A and TCAS B');
                else
                  %move_A = 9.8;
                  move_A = 22.8000;
                  %move_B = 123.2;
                  move_B = 96.2000;
                  %move_B_Lat = 11.230769230769232;
                  move_BB_Lat = 9.4758;
                  %move_A_Lat = 1.7307692307692;
                  move_AA_Lat = 3.0308;

myPosition_A = [move_A move_AA_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_BB_Lat  17.200000000000003 5.538461538461541];
                  set (handles.img_tcasA,'Position',myPosition_A);
                  set(handles.img_tcasb,'Position', myPosition_B);

                  fprintf('long_A\tLat_A\tLong_B\tLat_B\tDist\tTAU\n')
                   %for i = 1 : 1: 182 ,370

              for i = 1 : 1: 41
                 button_state = get(hObject,'Value');
                 if button_state == 0
                    continue

                 end

                if(current_data_TCAS_A == 250)
                   pause(0.1)
                elseif(current_data_TCAS_A == 300)
                   pause(0.05)
                elseif(current_data_TCAS_A == 350)
                   pause(0.02)
                elseif(current_data_TCAS_A == 400)
               pause(0.01) % pause every 0.01 second to determine the speed
                end

                 % move the two planes in opposite direction
                move_A = move_A + 1.0; % increment and  move the TCAS A in
                                  the x direction step of 0.3
                move_B = move_B - 1.0; % decrement and move the TCAS B in
                                  the  x direction step of -0.3
                move_AA_Lat = move_AA_Lat + 0.1;
                move_BB_Lat = move_BB_Lat - 0.065;
myPosition_A = [move_A move_AA_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_BB_Lat  17.200000000000003 5.538461538461541];
                  set (handles.img_tcasA,'Position',myPosition_A);
                  set(handles.img_tcasb,'Position', myPosition_B);
               % adjust the location (Lat & Long)of the two TCAS A and B
```

```
                % respectively
                %alg = alg - 0.151599;  % increment the value of longitude
                                of TCAS A to crrespond to new position of TCAS A
                alg = alg + 0.000768;
                blg = blg - 0.000768;
                all = all + 0.001538;
                bll = bll - 0.001538;
                %alg = alg - 0.151599;  % decrement the value of longitude
                                of TCAS A to crrespond to new position of TCAS A
                %blg = blg + 0.151599;
                set(handles.A_locate_long, 'String',alg);
                set(handles.B_locate_long, 'String',blg);
                set(handles.A_locate_latt, 'String',all);
                set(handles.B_locate_latt, 'String',bll);

                % calculate distance of the two TCAS
curr_long_ATT = str2double(get(handles.A_locate_long, 'String'));  % get  TCAS A longitude
curr_lat_ATT = str2double(get (handles.A_locate_latt, 'String'));  % get TCAS A latitude
curr_B_long_ATT = str2double( get(handles.B_locate_long, 'String'));  % get TCAS B longitude
curr_B_lat_ATT = str2double(get(handles.B_locate_latt, 'String'));   % get TCAS B latitude
                %dist = distance(curr_long_ATT,-curr_lat_ATT,curr_B_lat_ATT,-curr_B_long_ATT);
                %NewD = (deg2km(distance(10.54,-7.43,11.11,-7.71))/1.852)

                 dist = distance(all,-alg,bll,-blg);
                  Distance = deg2km(dist);
                  %Dist_calculate = Distance/1.852;
                  Dist_calculate = (deg2km(dist)/1.852);
                 %Dist_calculate_nm = Dist_calculate/1.852;
                 %apptime = Dist_calculate/(asp + bsp);
                 %approach_time = Dist_calculate/(asp + bsp)
                 apptime = (Dist_calculate/(asp + bsp))*3600;
                 %apptime = approach_time/3600
                 %fprintf('%4.4f\t%4.4f\t%4.4f\t%4.4f\n', all,alg,bll,blg)
                 set(handles.Dist_calculate,'String',Dist_calculate);
                 set(handles.time,'String', apptime);
                 Long_A(i) = curr_long_ATT;
                 Long_B(i) = curr_B_long_ATT;
                 Lat_A(i) =  curr_lat_ATT;
                 Lat_B(i) = curr_B_lat_ATT;

                 Distance(i) = Dist_calculate;
                 TAU(i) = apptime;
                 %fprintf('%d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t%4.6f\n',
i,Long_A(i),Lat_A(i),Long_B(i),Lat_B(i),Distance(i),TAU(i))
                 fprintf('%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\n',
Long_A(i),Lat_A(i),Long_B(i),Lat_B(i),Distance(i),TAU(i))
  %if (move_BB_Lat-move_AA_Lat)<1
                 %if (apptime <  60.0000)
                  if (apptime <  61.0000)
                 end
                 if (apptime <52.000)
                   %set(handles.uipanel10, 'Visible', 'off');
                  end
          %if (move_BB_Lat-move_AA_Lat)<1
                 if (apptime <46.000)&&(apptime > 44.0) %Traffic advisory
                        is given within 45 seconds;
                    set(handles.resolution,'String','Warning')
                    %set(handles.text42_detect, 'String','45 s')
```

```
                [Z,fs,nbits] =wavread ('warning2.wav');
                soundsc(Z,fs);
            end
            if (apptime <36.000)&& (apptime > 35.00) %Intruder
                declared as a threat within 35 seconds
                set(handles.resolution,'String','Collision')
                %set(handles.text42_detect, 'String','35 s')
                [Z,fs,nbits] =wavread ('COLLISION2.wav');
                %CHANGE RECORD TO 'COLLISION'
                soundsc(Z,fs);
            end
             if (apptime < 26.9999 ) && (apptime > 25.0 )
                            % Corrective RA  is given within 30 and 25 seconds for vertical and lateral
                            respectively
                set(handles.resolution,'String','Change Direction')
                %set(handles.text42_detect, 'String','27 s')
                [Z,fs,nbits] =wavread ('CHANGEDIRECT2.wav');
                soundsc(Z,fs);
                 if (apptime <  26.9999)
                for j= 1 :1: 3

move_A = move_A + 0.4; % increment and  move the TCAS A in the x direction step of 0.4
move_B = move_B - 0.4; % decrement and move the TCAS B in the  x direction step of -0.4
move_BB_Lat =  move_BB_Lat - 0.7153846153846154; %decrement and move the TCAS B in the  y
direction step of -0.7
                 move_AA_Lat =  move_AA_Lat + 0.530769230769232; % increment and  move the
TCAS A in the y direction step of 0.4
myPosition_A = [move_A move_AA_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_BB_Lat  17.200000000000003 5.538461538461541];
                set (handles.img_tcasA,'Position',myPosition_A);
                set(handles.img_tcasb,'Position', myPosition_B);
                %pause(0.2);
            if(current_data_TCAS_A == 250)
                pause(0.1)
              elseif(current_data_TCAS_A == 300)
                pause(0.05)
              elseif(current_data_TCAS_A == 350)
                pause(0.02)
              elseif(current_data_TCAS_A == 400)
               pause(0.01)
                            % pause every 0.01 second to determine the speed
            end
                end


                 end
              end

             if (apptime <5.000) % indicate clear

                set(handles.resolution,'String','Clear')
                %set(handles.text42_detect, 'String','5 s')
            end
            if (apptime <3.000) % indicate clear
                set(handles.resolution,'String','NO TRAFFIC')
                %set(handles.text42_detect, 'String','3 s')
            end
              if (apptime <0.9999) % indicate clear
                set(handles.resolution,'String','NO TRAFFIC')
```

```
            %set(handles.text42_detect, 'String','')
          break
           end
        end

%          CONTINUE TRAFFIC
            set (handles.img_tcasA,'Position',myPosition_A);
            set(handles.img_tcasb,'Position', myPosition_B);
          curr_long_ATT = str2double(get(handles.A_locate_long,
                          'String'));  % get  TCAS A longitude
          curr_lat_ATT = str2double(get (handles.A_locate_latt,
                          'String'));  % get TCAS A latitude
          curr_B_long_ATT = str2double( get(handles.B_locate_long,
                          'String'));  % get TCAS B longitude
          curr_B_lat_ATT = str2double(get(handles.B_locate_latt,
                    'String'));   % get TCAS B latitude
          for n = 1 : 1:35
          %pause(0.2) % pause every 0.1 second to determine the speed
        if(current_data_TCAS_A == 250)
            pause(0.1)
         elseif(current_data_TCAS_A == 300)
            pause(0.05)
         elseif(current_data_TCAS_A == 350)
            pause(0.02)
         elseif(current_data_TCAS_A == 400)
          pause(0.01)
         end
          % move the two planes in opposite direction
          move_A = move_A + 1.0; % increment and  move
                          the TCAS A in the x direction step of 0.3
          move_B = move_B - 1.0; % decrement and move the TCAS B in
                          the  x direction step of -0.3

          move_AA_Lat = move_AA_Lat + 0.1;
          move_BB_Lat = move_BB_Lat - 0.065;
         if(current_data_TCAS_A == 250)
            pause(0.1)
         elseif(current_data_TCAS_A == 300)
            pause(0.05)
         elseif(current_data_TCAS_A == 350)
            pause(0.02)
         elseif(current_data_TCAS_A == 400)
          pause(0.01)
          end

myPosition_A = [move_A move_AA_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_BB_Lat  17.200000000000003 5.538461538461541];
          set (handles.img_tcasA,'Position',myPosition_A);
          set(handles.img_tcasb,'Position', myPosition_B);

           % adjust the location (Lat & Long)of the two TCAS A and B
           % respectively
          set(handles.A_locate_long, 'String',alg);
          set(handles.B_locate_long, 'String',blg);
          set(handles.A_locate_latt, 'String',all);
          set(handles.B_locate_latt, 'String',bll);

          % increment the value of longitude of TCAS A
```

```
                                to crrespond to new position of TCAS A
                alg = alg + 0.000768;
                blg = blg - 0.000768;
                all = all + 0.001538;
                bll = bll - 0.001538;

                 dist = distance(all,-alg,bll,-blg);
                 Distance = deg2km(dist);
                 %Dist_calculate = Distance/1.852;
                 Dist_calculate = (deg2km(dist)/1.852);
                 %Dist_calculate_nm = Dist_calculate/1.852;
                 %apptime = Dist_calculate/(asp + bsp);
                 %approach_time = Dist_calculate/(asp + bsp)
                 apptime = (Dist_calculate/(asp + bsp))*3600;
                 %apptime = approach_time/3600
                 %fprintf('%4.4f\t%4.4f\t%4.4f\t%4.4f\n', all,alg,bll,blg)
                 set(handles.Dist_calculate,'String',Dist_calculate);
                 set(handles.time,'String', apptime);
                 Long_A(n) = alg;
                 Long_B(n) = blg;
                 Lat_A(n) =  all;
                 Lat_B(n) =  bll;
                 Distance(n) = Dist_calculate;
                 TAU(n) = apptime;
                 fprintf('%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\n',
Long_A(n),Lat_A(n),Long_B(n),Lat_B(n),Distance(n),TAU(n))
                 %fprintf('%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\n',
Long_A(i),Lat_A(i),Long_B(i),Lat_B(i),Distance(i),TAU(i))
                 if (apptime <21.000)&& (apptime > 19.0) % No Traffic advisory is given within 45 seconds;
                       set(handles.resolution,'String','NO TRAFFIC')
                       [Z,fs,nbits] =wavread ('NOTRAFFIC2.wav');
                       soundsc(Z,fs);
                       %disp 'NOTRAFFIC'
                    end
                 if button_state == get(hObject,'Min')
                    return
                 end
                 end

                 set(handles.A_locate_long, 'String',7.433624);
                 set(handles.B_locate_long, 'String',7.717896);
                 set(handles.toggle_start_simulatoo, 'String', 'Start
                                Simulator');
                 set(handles.resolution, 'String', ' ')
                 set(handles.img_tcasA,'Position', [9.8 1.7307692307692
                                17.200000000000003 5.538461538461541]);
                 set(handles.img_tcasb,'Position', [123.2
                               11.230769230769232  17.200000000000003 5.538461538461541]);
                 %set(handles.uipanel10, 'Visible', 'off');
                 %set(handles.text42_detect, 'String',' ')
                 set(handles.resolution,'String',' ')
                 set(handles.B_locate_latt,'String','11.113727');
                 set(handles.A_locate_latt,'String','10.544521');

                 %set(handles.text42_detect, 'String',' ')
                       end
elseif button_state == get(hObject,'Min')
                 % Toggle button is not pressed, take appropriate action
```

```
                %reset the longitude
                set(handles.A_locate_long, 'String',7.433624);
                set(handles.B_locate_long, 'String',7.717896);
                set(handles.toggle_start_simulatoo, 'String', 'Start
                                Simulator');
                set(handles.resolution, 'String', ' ')
                 set(handles.toggle_start_simulatoo,'Value',0);
                 set(handles.img_tcasA,'Position', [9.8 1.7307692307692
                                 17.200000000000003 5.538461538461541]);
                 set(handles.img_tcasb,'Position', [123.2
                                11.230769230769232  17.200000000000003 5.538461538461541]);
end
end
end
% --------------------------------------------------------------------
function reverse_logic_Callback(hObject, eventdata, handles)
% hObject    handle to reverse_logic (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% set(handles.text61, 'Visible', 'off');
% set(handles.text62, 'Visible', 'off');
% set(handles.text63, 'Visible', 'on');
% set(handles.text64, 'Visible', 'on');
% set(handles.text34, 'Visible', 'off');
% set(handles.text35, 'Visible', 'off');
set(handles.toggle_start_simulator, 'Visible', 'off');
set(handles.toggle_start_simulatoo, 'Visible', 'off');
set(handles.toggle_start_simulatool, 'Visible', 'on');
set(handles.time, 'String',' ');
set(handles.Dist_calculate, 'String',' ');
global simulation_mode
set(handles.A_locate_long, 'String',7.433624); %7.4336241   A_locate_long
set(handles.B_locate_long, 'String',7.717896); %B_locate_long
set(handles.A_locate_latt, 'String',10.544521);
set(handles.B_locate_latt, 'String', 11.113727);
simulation_mode = 3;
function toggle_start_simulatool_Callback(hObject, eventdata, handles)
% --- Executes on button press in toggle_start_simulatool.
% hObject    handle to toggle_start_simulatool (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of toggle_start_simulatool
global current_data_TCAS_B   % variable to store TCAS B speed
global current_data_TCAS_A   % variable to store TCAS A speed
global move_A    %inital position of TCAS A on the x-axis
global move_B  %inital position of TCAS B on the x-axis
global move_B_Lat  % initial position of TCAS B y-axis (latitude)
global move_A_Lat  % initial position of TCAS A y-axis (latitude)
global simulation_mode
global curr_long_ATT;
global dist;
global curr_B_long_ATT;
global apptime;
increase_long = 0;
%AUTOMATION
for t = 1 : 1: 2
set(handles.A_locate_long, 'String',7.433624); %7.4336241   A_locate_long
set(handles.B_locate_long, 'String',7.717896); %B_locate_long
set(handles.A_locate_latt, 'String',10.544521);
```

```
set(handles.B_locate_latt, 'String', 11.113727);

set(handles.time, 'String','');
set(handles.Dist_calculate, 'String', '');
move_A = 4.400000000000005;
move_B = 124.00000000000227;
TCASA_Alt = get(handles.popupmenu4_TCASA_alt,'Value');
if(TCASA_Alt==2)
   move_A_Lat = 6.4615384615384315
elseif(TCASA_Alt==3)
    move_A_Lat = 6.5615384615384315
elseif(TCASA_Alt==4)
    move_A_Lat = 9.4615384615384315
elseif(TCASA_Alt==5)
    move_A_Lat = 12.4615384615384315
elseif(TCASA_Alt==6)
    move_A_Lat = 15.4615384615384315
elseif(TCASA_Alt==7)
    move_A_Lat = 18.4615384615384315
elseif(TCASA_Alt==8)
    move_A_Lat = 21.4615384615384315
end
%fprintf('%1.4f\n', move_A_Lat)
TCASB_Alt = get(handles.popupmenu5_TCASB_alt,'Value');
if(TCASB_Alt==2)
   move_B_Lat = 6.4615384615384315
elseif(TCASB_Alt==3)
    move_B_Lat = 6.5615384615384315
elseif(TCASB_Alt==4)
    move_B_Lat = 9.4615384615384315
elseif(TCASB_Alt==5)
    move_B_Lat = 12.4615384615384315
elseif(TCASB_Alt==6)
    move_B_Lat = 15.4615384615384315
elseif(TCASB_Alt==7)
    move_B_Lat = 18.4615384615384315
elseif(TCASB_Alt==8)
    move_B_Lat = 21.4615384615384315
end
set(handles.img_tcasA,'Position', [move_A move_A_Lat  17.200000000000003 5.538461538461541]);
set(handles.img_tcasb,'Position', [move_B move_B_Lat  17.200000000000003 5.538461538461541]);
% disp 'move_A_Lat'
% disp 'move_B_Lat'
long_ATT = get(handles.A_locate_long, 'String');  % get  TCAS A longitude
lat_ATT = get (handles.A_locate_latt, 'String');  % get TCAS A latitude
B_long_ATT = get(handles.B_locate_long, 'String');  % get TCAS B longitude
B_lat_ATT = get(handles.B_locate_latt, 'String');   % get TCAS B latitude
speedA = current_data_TCAS_A; % get TCAS A  speed
%speedA = get(handles.A_Speed, 'String');
%speedB = get(handles.B_Speed, 'String');
speedB = current_data_TCAS_B; % get TCAS B
if (simulation_mode == 3)
move_A = 4.400000000000005;
move_B = 124.00000000000227;
%move_B_Lat = 6.4615384615384315;
          %6.3846153846153895

%move_A_Lat = 6.4615384615384315;
```

```
button_state = get(hObject,'Value'); % get the value of state of start button

if button_state == get(hObject,'Max')
            set(handles.toggle_start_simulatool, 'String', 'Stop');
            %convert the string to numbers
            alg =  str2double(long_ATT);
            all = str2double(lat_ATT);
            blg = str2double(B_long_ATT);
            bll = str2double(B_lat_ATT);
           % asp = str2double(current_data_TCAS_A)
            asp = current_data_TCAS_A % current speed of TCAS A
            bsp = current_data_TCAS_B % current speed of TCAS B
         if isempty(asp) || isempty(bsp)
              errordlg('please select speed for TCAS A and TCAS B');

            else
             fprintf('long_A\tLat_A\tLong_B\tLat_B\tDist\tTAU\n')
              %for i = 1 : 1: 182 ,370, 186
              for i = 1 : 1:200
               button_state = get(hObject,'Value');
                  if button_state == 0
                     %set(handles.toggle_start_simulator,'Value',0);
 %                 errordlg('please enter data')
                   continue
                  end
                 if(current_data_TCAS_A == 250)
                    pause(0.1)
                elseif(current_data_TCAS_A == 300)
                    pause(0.05)
                elseif(current_data_TCAS_A == 350)
                    pause(0.02)
                elseif(current_data_TCAS_A == 400)
                 pause(0.01) % pause every 0.01 second to determine the speed
                end

myPosition_A = [move_A move_A_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_B_Lat  17.200000000000003 5.538461538461541];
                set (handles.img_tcasA,'Position',myPosition_A);
                set(handles.img_tcasb,'Position', myPosition_B);
                 % adjust the location (Lat & Long)of the two TCAS A and B
                 % respectively
                 % increment the value of longitude of TCAS A to crrespond to new position of TCAS A
                alg = alg + 0.000768;
                blg = blg - 0.000768;
                all = all + 0.001538;
                bll = bll - 0.001538;
                %alg = alg - 0.151599;  % decrement the value of longitude of TCAS A to crrespond to new
                 position of TCAS A
                set(handles.A_locate_long, 'String',alg);
                set(handles.B_locate_long, 'String',blg);
                set(handles.A_locate_latt, 'String',all);
                set(handles.B_locate_latt, 'String',bll);

                % calculate distance of the two TCAS
              curr_long_ATT = str2double(get(handles.A_locate_long, 'String'));  % get  TCAS A longitude
                curr_lat_ATT = str2double(get (handles.A_locate_latt,
                                'String'));  % get TCAS A latitude
                curr_B_long_ATT = str2double( get(handles.B_locate_long,
```

```
                    'String'));  % get TCAS B longitude
                curr_B_lat_ATT = str2double(get(handles.B_locate_latt,
                    'String'));   % get TCAS B latitude
                dist = distance(all,-alg,bll,-blg);
                 Distance = deg2km(dist);
                 %Dist_calculate = Distance/1.852;
                 Dist_calculate = (deg2km(dist)/1.852);
                 %Dist_calculate_nm = Dist_calculate/1.852;
                 %apptime = Dist_calculate/(asp + bsp);
                 %approach_time = Dist_calculate/(asp + bsp)
                 apptime = (Dist_calculate/(asp + bsp))*3600;
                 %apptime = approach_time/3600
                 %fprintf('%4.4f\t%4.4f\t%4.4f\t%4.4f\n', all,alg,bll,blg)
                 set(handles.Dist_calculate,'String',Dist_calculate);
                 set(handles.time,'String', apptime);

                 Long_A(i) = curr_long_ATT;
                 Long_B(i) = curr_B_long_ATT;
                 Lat_A(i) =  curr_lat_ATT;
                 Lat_B(i) = curr_B_lat_ATT;

                 Distance(i) = Dist_calculate;
                 TAU(i) = apptime;

                 fprintf('%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\n',
Long_A(i),Lat_A(i),Long_B(i),Lat_B(i),Distance(i),TAU(i))
                 %fprintf('%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\n',
Long_A(i),Lat_A(i),Long_B(i),Lat_B(i),Distance(i),TAU(i))

 if (move_B_Lat-move_A_Lat)<1
                 %if (apptime <  60.0000)
                  if (apptime <  61.0000)
                  %set(handles.uipanel10, 'Visible', 'on');
                  %set(handles.text42_detect, 'String','60 s')
                 end
                  if (apptime <52.000)
                   %set(handles.uipanel10, 'Visible', 'off');
                  end
                 if (apptime <46.000)&&(apptime > 45.0) %Traffic advisory
                                          is given within 45 seconds;
                    set(handles.resolution,'String','Warning')
                    %set(handles.text42_detect, 'String','45 s')
                    [Z,fs,nbits] =wavread ('warning2.wav');
                    soundsc(Z,fs);
                 end
                 if (apptime <36.000)&& (apptime > 35.00) %Intruder
                                         declared as a threat within 35 seconds
                    set(handles.resolution,'String','Collision')
                    %set(handles.text42_detect, 'String','35 s')
                    [Z,fs,nbits] =wavread ('COLLISION2.wav'); %CHANGE RECORD TO 'COLLISION'
                    soundsc(Z,fs);
                 end
                  if (apptime < 27.9999 ) && (apptime > 26.0 )
                                         % Corrective RA  is given within 30 and 25 seconds
                                         for vertical and lateral respectively
                    set(handles.resolution,'String','Change Direction')
                    %set(handles.text42_detect, 'String','27 s')
                    [Z,fs,nbits] =wavread ('CHANGEDIRECT2.wav');
```

```
                soundsc(Z,fs);
                if (apptime <  27.9999)&& (apptime > 25.0 )

for j= 1 :1: 3
                    move_A = move_A + 0.4; % increment and  move the TCAS A in the x
                    direction step of 0.4
                    move_B = move_B - 0.4; % decrement and move the TCAS B in the  x
                    direction step of -0.4
                    move_B_Lat =  move_B_Lat + 0.2553846153846164;
                    %move_A_Lat =  move_A_Lat - 0.330769230769242;
                    move_A_Lat =  move_A_Lat + 0.250769230769242;
                   Rever_A = move_A_Lat;
                   Rever_B = move_B_Lat;
                   %Rever_LA = move_A_Lat(j+1)
myPosition_A     =     [move_A     move_A_Lat          17.200000000000003     5.538461538461541];
myPosition_B = [move_B move_B_Lat  17.200000000000003 5.538461538461541];
             set (handles.img_tcasA,'Position',myPosition_A);
             set(handles.img_tcasb,'Position', myPosition_B);
             %pause(0.2)
             if(current_data_TCAS_A == 250)
               pause(0.1)
            elseif(current_data_TCAS_A == 300)
               pause(0.05)
            elseif(current_data_TCAS_A == 350)
               pause(0.02)
            elseif(current_data_TCAS_A == 400)
            end
                end
                 end
               end
          if (apptime <  24.9999)&&(apptime >  21.9999)
            if(move_B_Lat - move_A_Lat)<=1 % Reverse Logic Altitude
                                    between TCASA  and TCASB less than 1
               set(handles.resolution,'String','TCAS A Reverse
                                    Direction')
               for k = 1 :1: 5
move_A = move_A + 0.4; % increment and  move the TCAS A in the x direction step of 0.4
move_B = move_B - 0.4; % decrement and move the TCAS B in the  x direction step of -0.4
move_A_Lat =   move_A_Lat - 0.7553846153846164;                                    %move_B_Lat =
move_B_Lat + 0.2553846153846164;
myPosition_A = [move_A move_A_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_B_Lat  17.200000000000003 5.538461538461541];
               set (handles.img_tcasA,'Position',myPosition_A);
               set(handles.img_tcasb,'Position', myPosition_B);
             %pause(0.2);
             if(current_data_TCAS_A == 250)
               pause(0.1)
            elseif(current_data_TCAS_A == 300)
               pause(0.05)
            elseif(current_data_TCAS_A == 350)
               pause(0.02)
            elseif(current_data_TCAS_A == 400)
           pause(0.01) % pause every 0.01 second to determine the speed
            end
                end
             end
           end
            if (apptime <5.000) % indicate clear
               set(handles.resolution,'String','Clear')
```

```
                %set(handles.text42_detect, 'String','5 s')
            end

            if (apptime <3.000) % indicate clear
                set(handles.resolution,'String','NO TRAFFIC')
                %set(handles.text42_detect, 'String','3 s')
            end

            if (apptime <0.9999) % indicate clear
                set(handles.resolution,'String','NO TRAFFIC')
                %set(handles.text42_detect, 'String','')
            break
            end
 elseif (apptime <46.000)&& (apptime > 45.0) % No Traffic advisory is given within 45 seconds;
     set(handles.resolution,'String','NO TRAFFIC')
     [Z,fs,nbits] =wavread ('NOTRAFFIC2.wav');
     soundsc(Z,fs);
   end
        end

%          CONTINUE TRAFFIC MOVEMENT
%
        set (handles.img_tcasA,'Position',myPosition_A);
        set(handles.img_tcasb,'Position', myPosition_B);
        curr_long_ATT = str2double(get(handles.A_locate_long,
                'String'));  % get  TCAS A longitude
        curr_lat_ATT = str2double(get (handles.A_locate_latt,
                'String'));  % get TCAS A latitude
        curr_B_long_ATT = str2double( get(handles.B_locate_long,
                'String'));  % get TCAS B longitude
        curr_B_lat_ATT = str2double(get(handles.B_locate_latt,
                'String'));   % get TCAS B latitude
          for n = 1 : 1:185
         if(current_data_TCAS_A == 250)
            pause(0.1)
         elseif(current_data_TCAS_A == 300)
            pause(0.05)
         elseif(current_data_TCAS_A == 350)
            pause(0.02)
         elseif(current_data_TCAS_A == 400)
          pause(0.01)
                        % pause every 0.01 second to determine the speed
        end
move_A = move_A + 0.3; % increment and  move the TCAS A in the x direction step of 0.3
move_B = move_B - 0.3; % decrement and move the TCAS B in the  x direction step of -0.3
myPosition_A = [move_A move_A_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_B_Lat  17.200000000000003 5.538461538461541];
            set (handles.img_tcasA,'Position',myPosition_A);
            set(handles.img_tcasb,'Position', myPosition_B);
          % adjust the location (Lat & Long)of the two TCAS A and B
          % respectively
            set(handles.A_locate_long, 'String',alg);
            set(handles.B_locate_long, 'String',blg);
            set(handles.A_locate_latt, 'String',all);
            set(handles.B_locate_latt, 'String',bll);

%alg = alg - 0.151599;  % increment the value of longitude of TCAS A to crrespond to new position of
TCAS A
```

```
                    alg = alg + 0.000768;
                    blg = blg - 0.000768;
                    all = all + 0.001538;
                    bll = bll - 0.001538;
                    dist = distance(all,-alg,bll,-blg);
                     Distance = deg2km(dist);
                     %Dist_calculate = Distance/1.852;
                     Dist_calculate = (deg2km(dist)/1.852);
                     %Dist_calculate_nm = Dist_calculate/1.852;
                     %apptime = Dist_calculate/(asp + bsp);
                     %approach_time = Dist_calculate/(asp + bsp)
                     apptime = (Dist_calculate/(asp + bsp))*3600;
                     %apptime = approach_time/3600
                     set(handles.Dist_calculate,'String',Dist_calculate);
                     set(handles.time,'String', apptime);
                     Long_A(n) = alg;
                     Long_B(n) = blg;
                     Lat_A(n) =  all;
                     Lat_B(n) =  bll;
                     Distance(n) = Dist_calculate;
                     TAU(n) = apptime;
                  fprintf('%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\t%4.2f\n',
Long_A(n),Lat_A(n),Long_B(n),Lat_B(n),Distance(n),TAU(n))
                  if (apptime <21.000)&& (apptime > 19.0) % No Traffic
                                  advisory is given within 45 seconds;
                      set(handles.resolution,'String','NO TRAFFIC')
                      [Z,fs,nbits] =wavread ('NOTRAFFIC2.wav');
                      soundsc(Z,fs);
                      %disp 'NOTRAFFIC'
                  end
                   if(apptime <65.000)&& (apptime >60.0)
                  for k = 1 :1: 3
                       move_A = move_A + 0.4; % increment and  move the
                                          TCAS A in the x direction step of 0.4
                       move_B = move_B - 0.4; % decrement and move the
                                          TCAS B in the  x direction step of -0.4
move_A_Lat =  move_A_Lat + 0.3776923076923082; % 0.7553846153846164;
myPosition_A = [move_A move_A_Lat  17.200000000000003 5.538461538461541];
myPosition_B = [move_B move_B_Lat  17.200000000000003 5.538461538461541];
                       set (handles.img_tcasA,'Position',myPosition_A);
                       set(handles.img_tcasb,'Position', myPosition_B);
                 %pause(0.2);
                 if(current_data_TCAS_A == 250)
                    pause(0.1)
                elseif(current_data_TCAS_A == 300)
                    pause(0.05)
                elseif(current_data_TCAS_A == 350)
                    pause(0.02)
                elseif(current_data_TCAS_A == 400)
                 pause(0.01) % pause every 0.01 second to
                                  determine the speed
                end
                  end
                   end
                  if button_state == get(hObject,'Min')
                      set(handles.Dist_calculate,'String','');
                      set(handles.time,'String',  '');
                      return
                  end
```

```
                end
                set(handles.A_locate_long, 'String',7.433624);
                set(handles.B_locate_long, 'String',7.717896);
                %set(handles.A_locate_long, 'String',060.440343);
                %set(handles.B_locate_long, 'String',005.440343);
                set(handles.toggle_start_simulatool, 'String', 'Reverse
                                        Logic');
                set(handles.resolution, 'String', ' ')
                set(handles.img_tcasA,'Position', [5.2 6.384615384615394
                                17.200000000000003 5.538461538461541]);
                set(handles.img_tcasb,'Position', [118.0
                                6.538461538461543  17.200000000000003 5.538461538461541]);
                %set(handles.uipanel10, 'Visible', 'off');
                %set(handles.text42_detect, 'String',' ')
                set(handles.resolution,'String',' ')
                set(handles.B_locate_latt,'String','11.113727');
                set(handles.A_locate_latt,'String','10.544521');
        end

   %case 0       %  pressed to start/stop button to stop
elseif button_state == get(hObject,'Min')
       button_state == get(hObject,'Min')
       set(handles.Dist_calculate,'String','');
       set(handles.time,'String', '');
    % Toggle button is not pressed, take appropriate action
    %reset the longitude
     set(handles.A_locate_long, 'String',7.433624);
     set(handles.B_locate_long, 'String',7.717896);
    set(handles.toggle_start_simulatool, 'String', 'Reverse Logic');
    set(handles.resolution, 'String', ' ')
    set(handles.toggle_start_simulatool,'Value',0);
    set(handles.img_tcasA,'Position',  [4.400000000000005  6.4615384615384315    17.200000000000003
5.538461538461541]);
    set(handles.img_tcasb,'Position',  [124.00000000000227  6.4615384615384315    17.200000000000003
5.538461538461541]);
end
end
end

% --- Executes during object creation, after setting all properties.
% --- Executes during object creation, after setting all properties.
function img_tcasA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to img_tcasA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate img_tcasA
img = imread('air_flip.jpg', 'jpg');
imagesc(img)
axis off;
% --- Executes during object creation, after setting all properties.
function img_tcasb_CreateFcn(hObject, eventdata, handles)
% hObject    handle to img_tcasb (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: place code in OpeningFcn to populate img_tcasb
%imgtcasb = imread('air2_flip.jpg', 'jpg');
imgtcasb = imread('air2_flipMOD.jpg', 'jpg');
imagesc(imgtcasb)
```

```
axis off;
% --- Executes on selection change in popupmenu5_TCASB_alt.
function popupmenu5_TCASB_alt_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu5_TCASB_alt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu5_TCASB_alt contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu5_TCASB_alt
 global move_B_Lat;
   str = get(hObject, 'String');
    val = get(hObject,'Value');
    % Set current data to the selected data set.
  switch str{val};
    case '900' % User selects Peaks.
       %current_alt_TCAS_B = 1000;
        %move_B_Lat = 6.4615384615384315;
        move_B_Lat = 6.4615384615384315;
    case '1000' % User selects Membrane.
      %current_alt_TCAS_B = 3000;
       %move_B_Lat = 9.4615384615384315;
       move_B_Lat = 6.5615384615384315;
    case '3000' % User selects Sinc.
      %current_alt_TCAS_B = 5000;
       %move_B_Lat = 12.4615384615384315;
       move_B_Lat = 9.4615384615384315;
    case '5000' % User selects Sinc.
      %current_alt_TCAS_B = 9000;
      %move_B_Lat = 15.4615384615384315;
      move_B_Lat = 12.4615384615384315;
      case '10000' % User selects Sinc.
      %current_alt_TCAS_B = 9000;
      %move_B_Lat = 15.4615384615384315;
      move_B_Lat = 15.4615384615384315;
      case '20000' % User selects Sinc.
      %current_alt_TCAS_B = 9000;
      %move_B_Lat = 15.4615384615384315;
      move_B_Lat = 18.4615384615384315;
      case '42000' % User selects Sinc.
      %current_alt_TCAS_B = 9000;
      %move_B_Lat = 15.4615384615384315;
      move_B_Lat = 21.4615384615384315;
 end

% --- Executes during object creation, after setting all properties.
function popupmenu5_TCASB_alt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu5_TCASB_alt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in popupmenu4_TCASA_alt.
function popupmenu4_TCASA_alt_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu4_TCASA_alt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu4_TCASA_alt contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu4_TCASA_alt
global move_A_Lat;
    str = get(hObject, 'String');
    val = get(hObject,'Value');
    % Set current data to the selected data set.
    switch str{val};
    case '900' % User selects Peaks.
       %current_alt_TCAS_B = 1000;
        %move_B_Lat = 6.4615384615384315;
        move_A_Lat = 6.4615384615384315;
    case '1000' % User selects Membrane.
      %current_alt_TCAS_B = 3000;
       %move_B_Lat = 9.4615384615384315;
       move_A_Lat = 6.5615384615384315;
    case '3000' % User selects Sinc.
      %current_alt_TCAS_B = 5000;
       %move_B_Lat = 12.4615384615384315;
       move_A_Lat = 9.4615384615384315;
    case '5000' % User selects Sinc.
      %current_alt_TCAS_B = 9000;
      %move_B_Lat = 15.4615384615384315;
      move_A_Lat = 12.4615384615384315;
      case '10000' % User selects Sinc.
      %current_alt_TCAS_B = 9000;
      %move_B_Lat = 15.4615384615384315;
      move_A_Lat = 15.4615384615384315;
      case '20000' % User selects Sinc.
      %current_alt_TCAS_B = 9000;
      %move_B_Lat = 15.4615384615384315;
      move_A_Lat = 18.4615384615384315;
      case '42000' % User selects Sinc.
      %current_alt_TCAS_B = 9000;
      %move_B_Lat = 15.4615384615384315;
      move_A_Lat = 21.4615384615384315;
    end

% --- Executes during object creation, after setting all properties.
function popupmenu4_TCASA_alt_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu4_TCASA_alt (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
   set(hObject,'BackgroundColor','white');
end
```