

IMPLEMENTATION AND SIMULATION OF MC68HC11  
MICROCONTROLLER UNIT USING SYSTEMC  
FOR CO-DESIGN STUDIES

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES  
OF  
MIDDLE EAST TECHNICAL UNIVERSITY

BY

CUMHUR ERKAN TUNCALI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR  
THE DEGREE OF MASTER OF SCIENCE  
IN  
THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2007

Approval of the Thesis

**IMPLEMENTATION AND SIMULATION OF MC68HC11  
MICROCONTROLLER UNIT USING SYSTEMC  
FOR CO-DESIGN STUDIES**

Submitted by **CUMHUR ERKAN TUNCALI** in partial fulfillment of the requirements  
for the degree of **Master of Science in Electrical and Electronics Engineering** by,

Prof. Dr. Canan Özgen

Dean, Graduate School of **Natural and Applied Sciences** \_\_\_\_\_

Prof. Dr. İsmet Erkmn

Head of Department, **Electrical and Electronics Eng., METU** \_\_\_\_\_

Prof. Dr. Murat Aşkar

Supervisor, **Electrical and Electronics Engineering, METU** \_\_\_\_\_

**Examining Committee Members:**

Prof. Dr. Hasan Güran (\*)

Electrical and Electronics Engineering, METU \_\_\_\_\_

Prof. Dr. Murat Aşkar (\*\*)

Electrical and Electronics Engineering, METU \_\_\_\_\_

Assist. Prof. Dr. Cüneyt Bazlamaçcı

Electrical and Electronics Engineering, METU \_\_\_\_\_

Assoc. Prof. Dr. Gözde Bozdağı Akar

Electrical and Electronics Engineering, METU \_\_\_\_\_

M.Sc. Lokman KESEN

ASELSAN \_\_\_\_\_

**Date:** \_\_\_\_\_

(\*) Head of Examining Committee

(\*\*) Supervisor

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

Name, Last name : Cumhur Erkan Tuncali

Signature :

## **ABSTRACT**

### **IMPLEMENTATION AND SIMULATION OF MC68HC11 MICROCONTROLLER UNIT USING SYSTEMC FOR CO-DESIGN STUDIES**

Tuncalı, Cumhuri Erkan

M.Sc., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Murat Aşkar

December 2007, 127 pages

In this thesis, co-design and co-verification of a microcontroller hardware and software using SystemC is studied. For this purpose, an MC68HC11 microcontroller unit, a test bench that contains input and output modules for the verification of microcontroller unit are implemented using SystemC programming language and a visual simulation program is developed using C# programming language in Microsoft .NET platform.

SystemC is a C++ class library that is used for co-designing hardware and software of a system. One of the advantages of using SystemC in system design

is the ability to design each module of the system in different abstraction levels. In this thesis, test bench modules are designed in a high abstraction level and microcontroller hardware modules are designed in a lower abstraction level.

At the end, a simulation platform that is used for co-simulation and co-verification of hardware and software modules of overall system is developed by combining microcontroller implementation, test bench modules, test software and visual simulation program. Simulations at different levels are performed on the system in the developed simulation platform. Simulation results helped observing errors in designed modules easily and making corrections until all results verified designed hardware modules. This situation showed that co-designing and co-verifying hardware and software of a system helps finding errors and making corrections in early stages of system design cycle and so reducing design time of the system.

Keywords: SystemC, MC68HC11, Microcontroller Simulator, Hardware and Software Co-design, SystemC Visual Simulation Tool.

# ÖZ

## BÜTÜNLEŞİK TASARIM ÇALIŞMALARI İÇİN MC68HC11 MİKRODENETLEYİCİSİNİN SYSTEMC KULLANILARAK GERÇEKLEŞTİRİLMESİ VE SİMÜLASYONUNUN YAPILMASI

Tuncalı, Cumhur Erkan

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Murat Aşkar

Aralık 2007, 127 sayfa

Bu tezde, bir mikrodenetleyicinin donanım ve yazılımının SystemC kullanılarak bütünleşik tasarım ve bütünleşik doğrulaması incelenmiştir. Bu amaçla, SystemC programlama dili kullanılarak, bir MC68HC11 mikrodenetleyici ünitesi, mikrodenetleyici ünitesini doğrulamak için giriş ve çıkış modülleri içeren bir test ünitesi ve Microsoft .NET platformunda C# programlama dili kullanılarak bir görsel simülasyon programı geliştirilmiştir.

SystemC, bir sistemin donanım ve yazılımının bütünleşik tasarımının yapılması için kullanılan bir C++ sınıf kütüphanesidir. Sistem tasarımlarında SystemC

kullanımının avantajlarından birisi, sistemin her modülünü farklı soyutlama seviyelerinde tasarlama imkanıdır. Bu tezde, test modülleri yüksek bir soyutlama seviyesinde, mikrodenetleyici donanımı modülleri ise daha düşük bir soyutlama seviyesinde tasarlanmıştır.

Neticede, gerçekleştirilmiş mikrodenetleyici, test modülleri, test yazılımı ve görsel simülasyon program birleştirilerek, bütün sistemin donanım ve yazılım modüllerinin bütünleşik simülasyon ve bütünleşik doğrulamasını yapmak için kullanılan bir simülasyon platformu oluşturulmuştur. Geliştirilen simülasyon platformunda, sistem üzerinde farklı seviyelerde simülasyonlar uygulanmıştır. Simülasyon sonuçları, tüm sonuçlar tasarlanan donanım modüllerini doğrulayana dek tasarlanan modüllerdeki hataları kolaylıkla görmeye ve düzeltmelerin yapılmasına yardımcı olmuştur. Bu durum, bir sistemin donanım ve yazılımının bütünleşik tasarım ve bütünleşik doğrulamasının, sistem tasarım sürecinin erken safhalarında hataları bulmaya, düzeltmelerin yapılmasına ve böylece sistem tasarım süresinin düşmesine yardımcı olduğunu göstermiştir.

Anahtar Kelimeler: SystemC, MC68HC11, Mikrodenetleyici Simülatörü, Donanım ve Yazılım Bütünleşik Tasarımı, SystemC Görsel Simülasyon Aracı

To My Family

## **ACKNOWLEDGEMENTS**

The author would like to express his deepest gratitude to Prof. Dr. Murat Aşkar for his guidance, encouragement and unlimited patience throughout this thesis work.

The author would also like to thank his colleagues in BOTT for their encouragement and support.

Finally, the author would like to express his special thanks to his family for their great support during this thesis work.

# TABLE OF CONTENTS

|   |      |
|---|------|
| ABSTRACT .....  | iv   |
| ÖZ .....  | vi   |
| ACKNOWLEDGEMENTS.....   | ix   |
| TABLE OF CONTENTS .....   | x    |
| LIST OF FIGURES.....  | xii  |
| LIST OF TABLES .....  | xvi  |
| LIST OF ABBREVIATIONS .....   | xvii |
| CHAPTER   |      |
| 1. INTRODUCTION.....  | 1    |
| 2. USING SYSTEMC FOR HARDWARE / SOFTWARE_CO-DESIGN AND<br>CO-VERIFICATION ..... | 7    |
| 2.1 Need for Hardware / Software Co-design of Systems .....                     | 7    |
| 2.2 Using SystemC for Co-design.....  | 8    |
| 2.3 Development Environment for SystemC.....                                    | 11   |
| 3. M68HC11 FAMILY OF MICROCONTROLLER UNITS .....                                | 16   |
| 3.1 General Description .....   | 16   |
| 3.2 Operation Modes of MC68HC11 .....   | 19   |
| 3.3 On-Chip Memory Systems .....  | 20   |
| 3.4 Central Processing Unit (CPU).....  | 22   |
| 3.5 Addressing Modes .....  | 25   |
| 3.6 Parallel Input / Output (I/O) .....   | 26   |
| 3.7 Synchronous Serial Peripheral Interface (SPI) .....                         | 31   |
| 3.8 Asynchronous Serial Communications Interface (SCI) .....                    | 33   |
| 3.9 Main Timer and Real Time Interrupt.....                                     | 35   |
| 3.10 Pulse Accumulator .....  | 36   |

|   |     |
|---|-----|
| 4. DESIGN OF MC68HC11 MICROCONTROLLER MODEL               |     |
| USING SYSTEMC .....                                       | 37  |
| 4.1 MC68HC11 SystemC Model Internal Structure .....       | 38  |
| 4.2 CPU Controller Unit .....                             | 40  |
| 4.3 Clock Divider.....                                    | 45  |
| 4.4 Arithmetic and Logic Unit (ALU) .....                 | 47  |
| 4.5 Register File.....                                    | 56  |
| 4.6 Address Bus Controller .....                          | 60  |
| 4.7 Handshake I/O Module .....                            | 63  |
| 4.8 Timer System.....                                     | 64  |
| 4.9 Serial Communications Module.....                     | 67  |
| 4.10 Read Only Memory (ROM).....                          | 69  |
| 4.11 Random Access Memory (RAM).....                      | 71  |
| 4.12 Electrically Erasable Programmable ROM (EEPROM)..... | 73  |
| 4.13 VLSI Implementation of SystemC Modules .....         | 75  |
| 5. VISUAL SIMULATION PLATFORM.....                        | 77  |
| 5.1 Structure of Visual Simulation Platform .....         | 77  |
| 5.2 Test Bench.....                                       | 78  |
| 5.3 Features of Visual Simulation Software.....           | 79  |
| 6. CONCLUSIONS .....                                      | 94  |
| REFERENCES.....   | 98  |
| APPENDIX  |     |
| A. M68HC11 INSTRUCTION SET .....                          | 100 |
| B. MICROCONTROLLER TEST CODE.....                         | 109 |
| B.1 Instructions and Addressing Modes Test Program .....  | 109 |
| B.2 Execution of Test Program on Original MC68HC11 .....  | 115 |
| B.2. Serial Port Test Program.....                        | 119 |
| C. VISUAL SIMULATION TOOL USER GUIDE.....                 | 120 |

## LIST OF FIGURES

|  |    |
|--|----|
| Figure 2.1: A Typical Example of SoC Systems. ....                   | 7  |
| Figure 2.2: Design Flow Comparison.....                              | 10 |
| Figure 2.3: “Open Workspace” Menu Item in Microsoft Visual C++ ..... | 12 |
| Figure 2.4: Building SystemC Library .....                           | 12 |
| Figure 2.5: Enabling Run-Time Type Information .....                 | 13 |
| Figure 2.6: Including SystemC Library to Library List .....          | 14 |
| Figure 2.7: Inserting Signals in GTKWave.....                        | 15 |
| Figure 3.1: MC68HC11E9 Pin Assignments.....                          | 17 |
| Figure 3.2: Block Diagram of MC68HC11E9 .....                        | 18 |
| Figure 3.3: Memory Map of MC68HC11E9.....                            | 20 |
| Figure 3.4: CPHA Equals Zero SPI Transfer Format .....               | 32 |
| Figure 3.5: CPHA Equals One SPI Transfer Format.....                 | 33 |
| Figure 3.6: Start Bit Reception .....                                | 34 |
| Figure 4.1: Internal Structure of MC68HC11 SystemC Model.....        | 39 |
| Figure 4.2: Datapath of Microprocessor .....                         | 40 |
| Figure 4.3: Block Diagram of CPU Controller .....                    | 41 |
| Figure 4.4: Internal Clock Cycles of a Bus Cycle .....               | 41 |
| Figure 4.5: Clock Divider Symbol .....                               | 45 |
| Figure 4.6: Internal Clock Signals of MC68HC11 .....                 | 46 |

|              |  |    |
|--------------|--|----|
| Figure 4.7:  | Resulting Waveforms of Clock Divider Simulation .....    | 46 |
| Figure 4.8:  | Arithmetic and Logic Unit Symbol.....                    | 47 |
| Figure 4.9:  | ALU Block Diagram .....                                  | 48 |
| Figure 4.10: | Arithmetic and Logic Unit Test Results .....             | 51 |
| Figure 4.11: | ALU Addition Test Waveforms.....                         | 52 |
| Figure 4.12: | ALU Increment Test Waveforms.....                        | 52 |
| Figure 4.13: | ALU Subtract Test Waveforms.....                         | 52 |
| Figure 4.14: | ALU Decrement Test Waveforms .....                       | 53 |
| Figure 4.15: | ALU AND Operation Test Waveforms.....                    | 53 |
| Figure 4.16: | ALU OR Operation Test Waveforms.....                     | 53 |
| Figure 4.17: | ALU XOR Operation Test Waveforms.....                    | 54 |
| Figure 4.18: | ALU Complement Test Waveforms.....                       | 54 |
| Figure 4.19: | ALU Negate Test Waveforms .....                          | 54 |
| Figure 4.20: | ALU Arithmetic Shift Right Test Waveforms.....           | 55 |
| Figure 4.21: | ALU Arithmetic / Logical Shift Left Test Waveforms ..... | 55 |
| Figure 4.22: | ALU Logical Shift Right Test Waveforms .....             | 55 |
| Figure 4.23: | ALU Rotate Left Test Waveforms .....                     | 56 |
| Figure 4.24: | ALU Rotate Right Test Waveforms .....                    | 56 |
| Figure 4.25: | Register File Symbol.....                                | 57 |
| Figure 4.26: | Register File Test Waveforms.....                        | 58 |
| Figure 4.27: | Register File Test Console Outputs .....                 | 59 |
| Figure 4.28: | Address Bus Controller Symbol .....                      | 60 |

|              |   |    |
|--------------|---|----|
| Figure 4.29: | Address Bus Controller Test Results .....           | 61 |
| Figure 4.30: | Console Outputs of Address Bus Controller Test..... | 62 |
| Figure 4.31: | Handshake I/O Module Symbol .....                   | 63 |
| Figure 4.32: | Handshake I/O Module Block Diagram .....            | 64 |
| Figure 4.33: | Timer System Symbol.....                            | 65 |
| Figure 4.34: | Timer System Block Diagram.....                     | 66 |
| Figure 4.35: | Serial Communications Module Symbol.....            | 67 |
| Figure 4.36: | Serial Communications Module Block Diagram.....     | 68 |
| Figure 4.37: | ROM Symbol .....                                    | 69 |
| Figure 4.38: | Console Output of ROM Test.....                     | 70 |
| Figure 4.39: | Resulting Waveforms of ROM Test.....                | 70 |
| Figure 4.40: | RAM Symbol.....                                     | 71 |
| Figure 4.41: | Waveforms of RAM Test Results .....                 | 72 |
| Figure 4.42: | RAM Test Results.....                               | 72 |
| Figure 4.43: | EEPROM Symbol .....                                 | 73 |
| Figure 4.44: | EEPROM Write / Read Test Results.....               | 74 |
| Figure 4.45: | EEPROM Clear / Read Test Results.....               | 75 |
| Figure 5.1:  | Visual Simulation Platform.....                     | 78 |
| Figure 5.2:  | Main Window of Visual Simulation Software.....      | 81 |
| Figure 5.3:  | 68HC11 Assembly Code Editor Window .....            | 82 |
| Figure 5.4:  | Machine Code Generator Window.....                  | 83 |
| Figure 5.5:  | Test Environment Region of Simulation Program ..... | 85 |

|              |   |     |
|--------------|---|-----|
| Figure 5.6:  | 8-bit Binary Switch Configuration.....                      | 86  |
| Figure 5.7:  | Push Button Pulse Generator Configuration.....              | 87  |
| Figure 5.8:  | Serial Monitor Configuration.....                           | 88  |
| Figure 5.9:  | Test Bench Configuration Window.....                        | 89  |
| Figure 5.10: | Running Simulation.....                                     | 90  |
| Figure 5.11: | Instruction and Special Function Registers Information..... | 91  |
| Figure 5.12: | Internal Registers Information.....                         | 92  |
| Figure 5.13: | RAM Content Window.....                                     | 92  |
| Figure 5.14: | ROM Content Window.....                                     | 93  |
| Figure 5.15: | EEPROM Content Window.....                                  | 93  |
| Figure C.1:  | Main Window of Visual Simulation Software.....              | 120 |
| Figure C.2:  | Code Editor Example.....                                    | 121 |
| Figure C.3:  | Code Generator Example.....                                 | 122 |
| Figure C.4:  | Test Environment Example.....                               | 123 |
| Figure C.5:  | Test Bench Port Configuration Example.....                  | 124 |
| Figure C.6:  | 8-bit Binary Switch Configuration.....                      | 125 |
| Figure C.7:  | Push Button Pulse Generator Configuration.....              | 125 |
| Figure C.8:  | Serial Monitor Configuration.....                           | 126 |
| Figure C.9:  | Running Simulation.....                                     | 126 |
| Figure C.10: | Simulation Results of Serial Monitor Module.....            | 127 |

## LIST OF TABLES

|            |   |     |
|------------|---|-----|
| Table 3.1: | Internal Registers of M68HC11 CPU ..... | 23  |
| Table 3.2: | Condition Codes Register .....          | 24  |
| Table 3.3: | Summary of Port A Pins .....            | 27  |
| Table 3.4: | Summary of Port B Pins.....             | 28  |
| Table 3.5: | Summary of Port C Pins .....            | 29  |
| Table 3.6: | Summary of Port D Pins .....            | 30  |
| Table 4.1: | CPU States.....                         | 43  |
| Table 4.2: | ALU Commands and Meanings .....         | 49  |
| Table 4.3: | Synthesis Results of ALU Module.....    | 76  |
| Table A.1: | Information on Operands .....           | 100 |
| Table A.2: | Information on Condition Codes.....     | 101 |
| Table A.3: | M68HC11 Instruction Set .....           | 102 |

## LIST OF ABBREVIATIONS

|           |   |
|-----------|---|
| ALU       | Arithmetic and Logic Unit                             |
| ASM       | Assembly Language                                     |
| ASIC      | Application Specific Integrated Circuit               |
| CCR       | Condition Codes Register                              |
| CISC      | Complex Instruction Set Computer                      |
| Co-design | Compound Design                                       |
| CPU       | Central Processing Unit                               |
| EDA       | Electronic Design Automation                          |
| EDIF      | Electronic Design Interchange Format                  |
| EEPROM    | Electrically Erasable Programmable Read Only Memory   |
| FPGA      | Field Programmable Gate Array                         |
| HDL       | Hardware Description Language                         |
| HW        | Hardware  |
| IC        | Integrated Circuit                                    |
| IEEE      | The Institute of Electrical and Electronics Engineers |
| I/O       | Input / Output  |
| IP        | Intellectual Property                                 |
| MCU       | Microcontroller Unit                                  |
| METU      | Middle East Technical University                      |
| Opcode    | Operation Code  |

|       |                                     |
|-------|-------------------------------------|
| OSCI  | Open SystemC Initiative             |
| PC    | Program Counter                     |
| SP    | Stack Pointer                       |
| SW    | Software                            |
| RAM   | Random Access Memory                |
| RISC  | Reduced Instruction Set Computer    |
| ROM   | Read Only Memory                    |
| RTL   | Register Transfer Level             |
| SCI   | Serial Communications Interface     |
| SPI   | Serial Peripheral Interface         |
| SoC   | System on a Chip                    |
| SRAM  | Static RAM                          |
| VCD   | Value Change Dump                   |
| VHDL  | VHSIC Hardware Description Language |
| VHSIC | Very High Speed Integrated Circuit  |

# CHAPTER 1

## INTRODUCTION

Today's applications require complex electronic systems that contain multiple modules. These modules can be sub-systems, application specific integrated circuits (ASIC), microcontrollers and memory devices around central processing unit (CPU). Collecting all elements of a system on a circuit board is getting harder and more expensive as the system complexities increase. Developments in VLSI technologies allow combination of all hardware and software of systems on a single chip. Such a system is called "System On a Chip" (SoC). SoC designs allow significant reduction in size, design time and cost of a system. The need for SoC approach in system design by considering increased complexities and importance of time-to-market is presented by Kahn A. in [1].

SoC designs have both hardware and software parts. In a traditional design approach, hardware and software of a system are designed by different teams by using different tools. After hardware and software designs are completed, a prototype for system is produced where hardware and software are first brought together. Overall system tests are performed after this stage in the design flow. If design errors are discovered in these tests, hardware and software teams refine their designs and new prototypes of the system are re-manufactured. This cycle repeats itself until the system verification is successful. Prototype manufacturing is generally time consuming and expensive. This situation results in significantly increased design time and cost of the system. Compound design (co-design) approach targets to solve this problem. In this approach, both hardware and software of system are designed and verified together. Preliminary system tests

can be performed before prototypes are manufactured. Co-designing and co-verifying hardware and software of a system reduces the number of errors that are discovered after prototyping and the time-to-market period of a system.

Based on the requirements and the experience, system designers can use different languages and approaches for co-design purposes. Schulz S. *et al* proposed a model based specification approach for co-design [2]. SystemC, SystemVerilog, Verilog, VHDL and OpenVera are some examples of design languages. Each language has its own advantages and disadvantages over the others. For example, it is generally difficult and time consuming task to use Verilog as a high level system design language when compared to SystemC; however low abstraction level Verilog designs may yield more area and performance efficient systems. As a co-design approach, more than one language can also be used in a design flow; however interfacing subsystems that are described in different languages is difficult. Multi-language co-design approaches and interfacing problems of subsystems in these approaches are discussed by Benmohammed M. and Merniz S. in [3].

Among system design languages, ability of SystemC to model hardware and software at different abstraction levels from transaction level to register transfer level (RTL) makes it the best candidate for co-designing and co-verifying hardware and software of a system using single language. IEEE Standard SystemC language has been proposed as an ANSI standard C++ class library for system and hardware design for use by designers and architects who need to address complex systems [4]. SystemC is actually a class library which extends C++ language with new constructs for modeling hardware components. These constructs can be listed as modules, ports, processes, events, interfaces and channels. It has an event-driven simulation kernel that allows concurrency of signals. Detailed information on SystemC language, its constructs, data types and abstraction levels can be found in [4] and [5]. SystemC designs can be synthesized to RTL by using electronic design automation (EDA) tools; however not all SystemC constructs are synthesizable. A subset of SystemC language that

is suitable for synthesis is given in [6]. Celoxica Agility Compiler [7] provides a SystemC system level synthesis tool. It outputs to RTL for synthesis tools like Design Compiler and optimized Electronic Design Interchange Format (EDIF) for Altera and Xilinx FPGAs. SystemCrafter SC [8] is another software tool that synthesizes SystemC into RTL VHDL or Verilog for Xilinx FPGAs. Synthesis of SystemC transaction level and register transfer level designs are presented in [9] by Calazans N. *et al* and area comparisons are done between RTL SystemC description synthesis and RTL VHDL descriptions. Results showed that, area of hardware synthesized from SystemC is comparable with area of equivalent hardware that is synthesized from VHDL. A 4-bit microprocessor is implemented in VHDL and SystemC RTL as a case study and these implementations are compared in terms of easiness of the design, functional simulations, logic optimizations and timing analysis [10]. Their results showed that SystemC is more efficient than VHDL in terms of verification, because SystemC simulations run faster and development of test vectors are easier.

Intellectual property (IP) cores are hardware blocks that are designed using languages like VHDL, Verilog and SystemC. Implementations and verification of these cores are completed before they are delivered to customers. System designers can reduce design and verification effort by using these pre-designed and pre-verified IP cores in their system designs. IP providers deliver cores in soft or hard forms. Soft IP cores are descriptions of hardware that are ready to synthesis. Hard IP cores are already synthesized and ready to manufacture. SystemC IP cores are available and being developed all over the world. Synopsis Inc. provides SystemC models for the PowerPC processors and a broad range of peripherals as a part of its System Studio product [11]. CoWare Model Library [12] from CoWare Inc. provides a collection of SystemC IP models including ARM and MIPS processors. Open source SystemC IP cores of USB1.1 function from Usselmann R., an area improved DES coprocessor and MD5 hash algorithm from Villar J. C. are available in [13]. Jonsson B. suggests a JPEG encoder SystemC implementation in [14].

Several SystemC IP cores have been developed in Middle East Technical University (METU). Implementations of industry standard 80C51 compatible 8-bit microcontroller unit by Kesen L. [15], an 16-bit RISC based MSP430 microcontroller unit by Zengin S. [16], optimized reconfigurable Viterbi decoder by Sözen S. [17], direct digital synthesis based function generator by Kazancioglu U. [18], analog and mixed signal modeling for PIC 16F871 microcontroller unit by Mert Y. M. [19] are theses on SystemC IP models previously completed in the Electrical and Electronics Engineering Department of METU.

Companies and designers that have previously developed know-how and software for a specific microprocessor want to use this microcontroller in their new SoC designs. This situation have arisen the need for IP cores of industry standard microprocessors and microcontrollers. Using IP cores of a microcontroller give system designers the ability to use their past know-how on that microcontroller in their new SoC designs and the flexibility to remove unnecessary peripherals of microcontroller at hand and optimize its peripherals for target system or replacing old peripherals with the ones that use new standards. For example an old serial communications interface can be replaced with a modern interface such as USB 2.0 or Bluetooth. Developments in IC fabrication techniques may also allow implementation of the new microcontroller core to operate at higher frequencies than the original one.

The objective of this thesis is to make a synthesizable SystemC implementation of a microcontroller that is instruction set and timing compatible with industry standard MC68HC11 [20] [21] microcontroller with its peripheral devices and to provide a powerful simulation platform for the implemented microcontroller with a test bench and visual interface, using co-design capabilities of SystemC. With today's technology, hardware of the designed microcontroller unit can be manufactured after synthesizing the developed SystemC model to HDLs using electronic design automation tools. MC68HC11 is selected because it is widely used in several applications; it has commonly required peripherals and its well designed architecture makes it a good candidate for microprocessor architecture

for educational purposes. In this study, it is aimed to develop a user friendly simulator interface for configuring and running SystemC simulations of implemented system that consists of microcontroller hardware, software and test hardware. The developed simulator is also responsible for presenting simulation results in an easily understandable format. This architecture can be considered as a complete design and verification environment for MC68HC11 where microprocessor hardware and software can be tested before hardware prototypes including peripheral components are manufactured. Main difference of this simulation platform from microcontroller simulators is its usability as a synthesizable hardware model of a microcontroller.

MC68HC11 microcontroller core and its peripherals are implemented in SystemC platform using Microsoft Visual C++ 6.0. Developed core and peripheral devices are designed to mostly comply with the original ones. When compiled with appropriate flags for simulation purposes, implemented microcontroller takes simulation options and provides some information on its internal workings to outside world by using input and output files. It reads ROM and EEPROM contents from input files and provides internal register values, memory contents, some of its internal signals and operation states of microcontroller CPU in each clock cycle. Analog to digital converter module of MC68HC11 is kept out of this thesis concept because standard SystemC language does not support modeling of analog hardware.

A configurable test bench which employs different input and output device modules such as serial monitor, TTL oscillators, switches and seven segment displays, is designed in a higher abstraction level than microcontroller core. This test bench communicates with a visual simulation program, which is designed as a part of overall simulation environment, using input and output files. It reads test hardware configurations and port connections from input files and writes cycle accurate simulation results to output files. Visual simulation program is developed using C# language in Microsoft .NET platform and is responsible for user

interaction in order to configure simulation environment and present simulation results.

In Chapter 2, using SystemC for co-design and co-verification purposes is studied. SystemC language and its capabilities are explained without much detail and how to set up a development environment for SystemC is described step by step.

Information on original MC68HC11 family of microcontroller units is given in Chapter 3. Operation and addressing modes of its central processing unit, its memory devices and peripherals are described briefly.

Chapter 4 explains the SystemC implementation of MC68HC11 microcontroller unit and its peripherals. Each SystemC module is explained separately with its structure. This chapter also presents test results of designed modules and overall microcontroller unit.

Developed simulation platform and SystemC implementation of developed test bench is explained in Chapter 5. This chapter also explains the link between SystemC implementations and visual user interface part of the developed simulation platform.

Finally in Chapter 6, conclusions of the work are presented and directions for future work are suggested. References are presented for further reading.

## CHAPTER 2

### USING SYSTEMC FOR HARDWARE / SOFTWARE CO-DESIGN AND CO-VERIFICATION

#### 2.1 Need for Hardware / Software Co-design of Systems

Today's SoC systems may be complex structures with multiple processors, ICs and software. They may also contain other sub-systems. Different bus interfaces connect on-chip devices and sub-systems. Figure 2.1 is presented as a typical example of SoC systems.

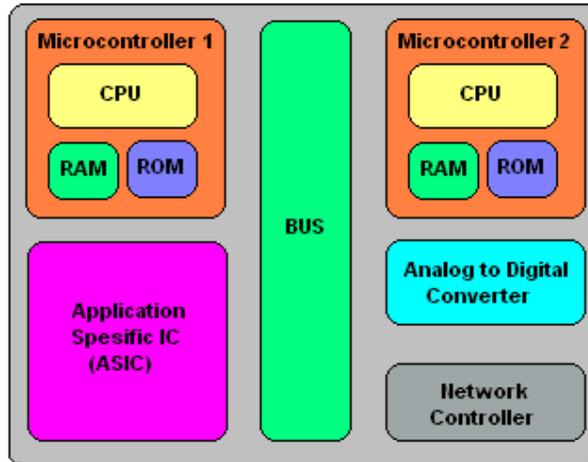


Figure 2.1: A Typical Example of SoC Systems.

In a traditional design flow, hardware and software parts of the systems are designed by different teams without much interaction between these parts. Design teams targets to achieve given specifications. Each part has its own design flow. After each team completes designs and a prototype of hardware is manufactured, hardware and software are brought together and overall system tests are done. Discovering design errors at this stage of overall system design flow causes large amounts of money time to companies because design flow of problematic parts should be repeated until overall system is verified. If design errors are discovered in hardware, prototyping of hardware should be done again.

Competition in the market makes a pressure on companies to reduce time-to-market periods and costs of the systems. In order to achieve shorter design periods without any decrease in reliability of complex systems, modeling of overall architecture and integration of hardware and software parts should be done in early stages of design flow. Tests of embedded systems should be done before manufacturing hardware prototypes. Co-design technique offers designing hardware and software parts together starting from very early stages of overall system design flow. These parts are in interaction with each other during almost whole design flow and they are verified together before prototyping of hardware is done. This minimizes number of errors that are discovered after prototyping.

## **2.2 Using SystemC for Co-design**

SystemC is an open source system design language that is based on C++ language. It is actually an ANSI standard C++ class library which is developed for hardware and system design. It is developed by Open SystemC Initiative (OSCI). C++ language is inadequate for describing concurrent behavior of hardware and lacks notion of time. SystemC extends C++ library for describing hardware by providing data types for describing hardware and structure hierarchy. SystemC has a simulation kernel which has a scheduler that synchronizes execution of functions in accordance with time notion and event driven architecture of functions.

Verification of a system generally takes too much time, in most cases verification period may be longer than design period. This is a handicap for reducing time to market, so simulations should be done easily and in shorter times; test benches should be developed in less time. In order to achieve these in complex systems, design tool should allow high abstraction levels as it gets harder and more time consuming to design everything in a system in register transfer level. SystemC has a great feature that it allows designing systems in high abstraction levels. This is called system level design. Using SystemC, a designer can start modeling overall architecture at a very high level of abstraction and refine model by lowering abstraction level of described parts. Using very high level abstractions and then making refinements is more difficult and sometimes impossible in other hardware description languages (HDLs). It is also possible to keep test benches at very high level without refining them in order to reduce design time without affecting reliability of tests.

SystemC is a great platform for making hardware / software co-design of systems. Because it is a class library in C++, it inherits properties of C++ language. System designer can design both hardware and software of system using same language and making refinements on any part of the system does not affect rest of the system. Co-verification of hardware and software can also be done using SystemC during design time. Figure 2.2 shows comparison of traditional design flow with SystemC co-design flow.

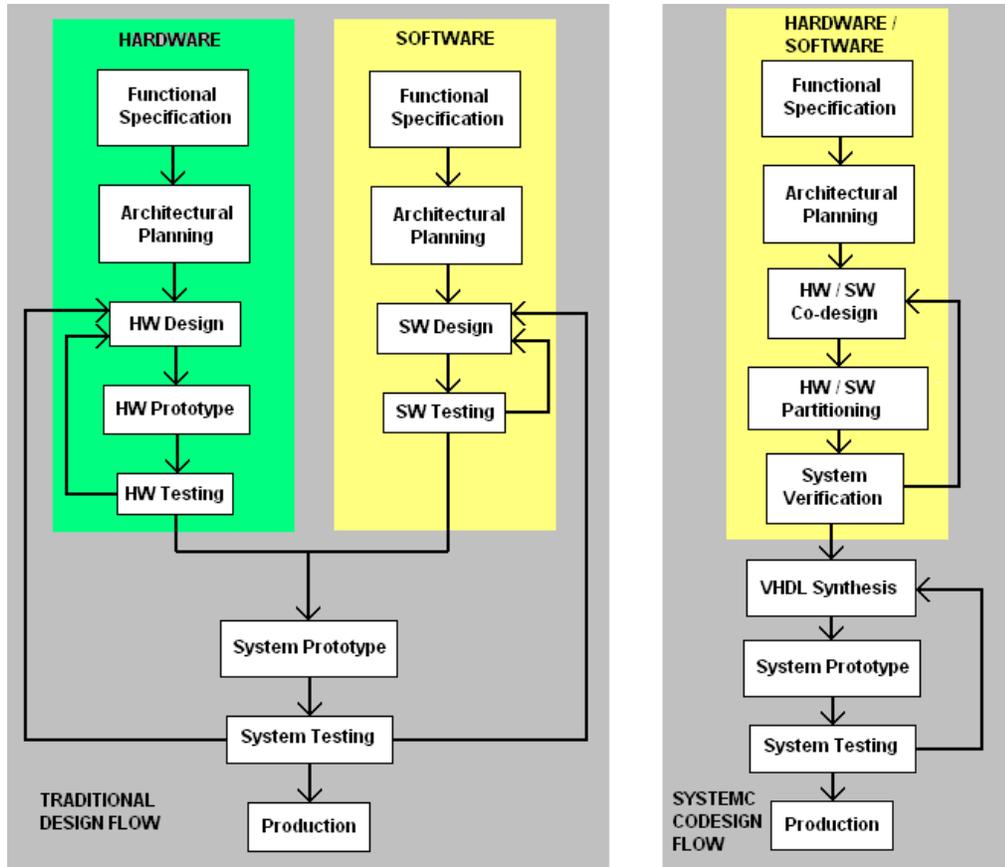


Figure 2.2: Design Flow Comparison

### 2.3 Development Environment for SystemC

In order to start making designs using SystemC and viewing simulation result waveforms, some tools are needed. First of all SystemC library source should be downloaded. Downloading documentation is also recommended. SystemC is open source and freely available at Open SystemC Initiative web site "[www.systemc.org](http://www.systemc.org)". During this thesis work, SystemC library version 2.1.v.1 is used. After downloading SystemC library source, it should be compiled to generate a library file for design environment. Any C++ compiler can be used to generate library file. Microsoft XP is used as operating system and Microsoft Visual C++ 6.0 is used for C++ compilation in this thesis work. Steps for compiling source of SystemC library in Microsoft Visual C++ 6.0 are presented below. SystemC documentation can be used as a guideline for library creation process for different development environments.

SystemC library compilation steps:

1. Downloaded source files come in an archive file. This archive should be extracted to any folder using a suitable archive manager program.
2. In Microsoft Visual C++, "Open Workspace..." menu item under "File" menu should be clicked and "systemc.dsw" file should be selected by browsing into "msvc60\SystemC" directory which is under the directory where library source files are extracted.

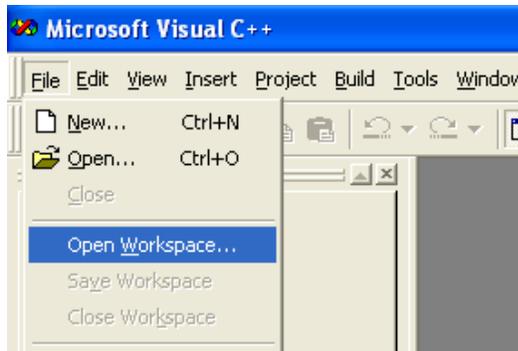


Figure 2.3: "Open Workspace" Menu Item in Microsoft Visual C++

3. Workspace file is adjusted for compilation of SystemC library in Microsoft Visual C++. Selecting "Build systemc.lib" under "Build" menu is enough for compilation of the library.

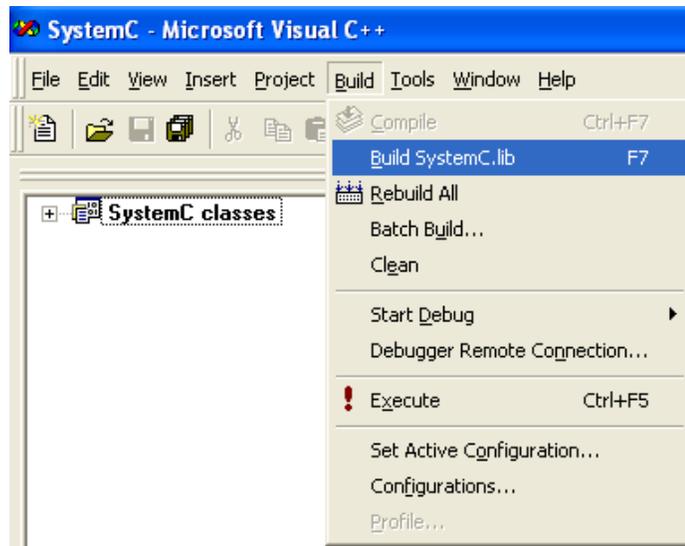


Figure 2.4: Building SystemC Library

For creating a SystemC project, a new empty Win32 console application should be created first. After creating a new project, some adjustments should be done on this project options for SystemC compilation. Below are steps of creating a SystemC design in Microsoft Visual C++. Compiler documentation can be referred for better understanding.

SystemC project creation steps in Microsoft Visual C++:

1. A new, empty C++ project is created by using “New” menu item under “File” menu and selecting “Win32 Console Application” option.
2. “Enable Run-Time Type Information (RTTI)” checkbox should be checked by selecting “C++ Language” category in “C/C++” tab under “Settings” menu item of “Project” menu. This is shown in Figure 2.5.

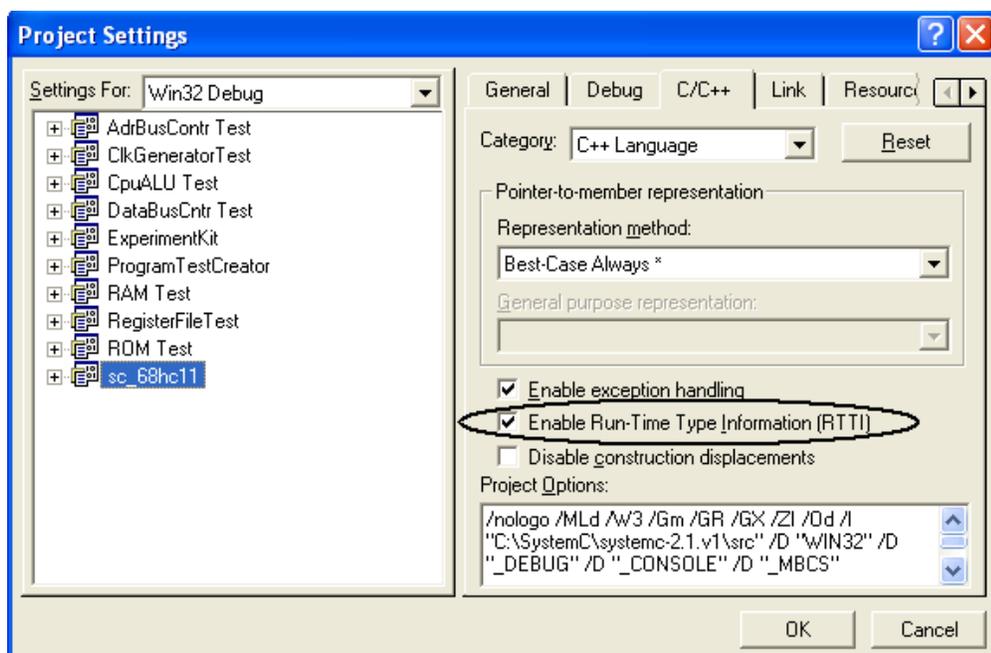


Figure 2.5: Enabling Run-Time Type Information

3. For using SystemC data types and functions, SystemC library should be used in link step. Under “Projects” menu and “Settings” menu item, “Link” tab should be selected and “systemc.lib” should be added to the “Object / library modules” list. This is presented in Figure 2.6 below.

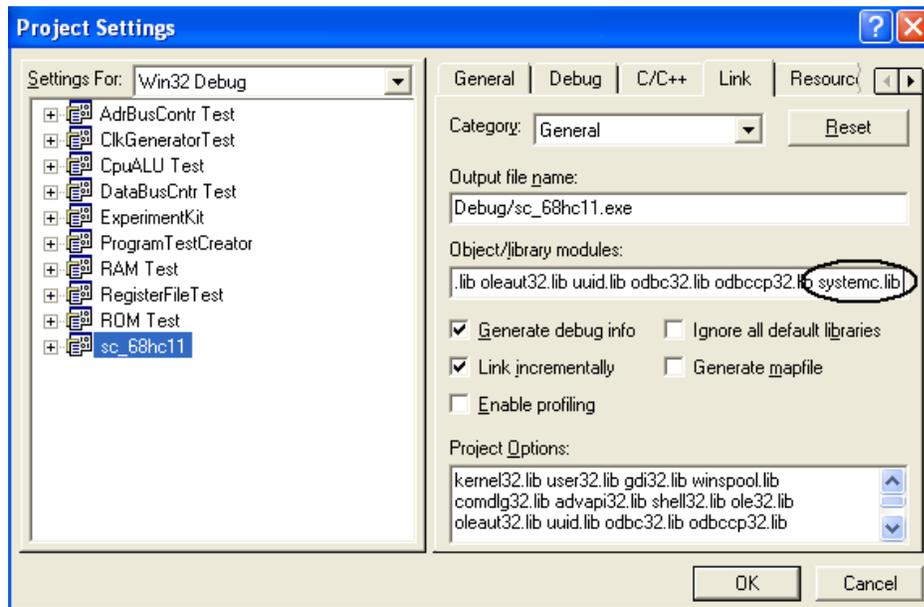


Figure 2.6: Including SystemC Library to Library List

4. Include file and library directory search paths should be added to project. For doing this, “Settings” menu item is selected under “Project” menu. Under “C/C++” tab, “Preprocessor” category is selected and path of “src” directory which is in extracted SystemC library directory is entered in text field into area labeled “Additional include directories”. Path to SystemC library which is under “msvc60\systemc\debug” directory in extracted SystemC library directory is entered in “Additional library path” text box under “Input” category in “Link” tab.

5. Project settings are completed for SystemC compilation. After adding source files to created project and developing SystemC model, “Build” menu can be used for generating executable program file of developed model.

If SystemC code is written appropriately for value change dump (.vcd) file generation, which is achieved by using “sc\_trace” method of SystemC library, executable file will generate a “.vcd” file (also called trace file) that contains simulation waveforms of selected signals. There are different programs that can be used to view these trace files. In this thesis “GTKWave” application is used for this purpose. In order to see waveforms in GTKWave, “Search -> Signal Search Tree” menu should be accessed and signals that are wanted to be shown on screen should be selected and inserted. Figure 2.7 shows an example screenshot from GTKWave for inserting signals.

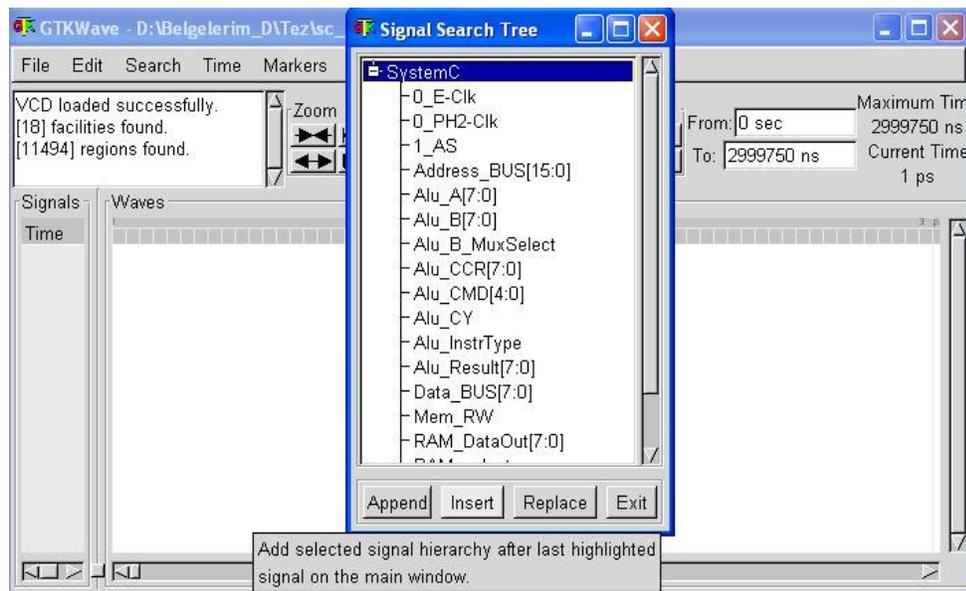


Figure 2.7: Inserting Signals in GTKWave.

## CHAPTER 3

### M68HC11 FAMILY OF MICROCONTROLLER UNITS

MC68HC11 microcontroller is briefly explained in this chapter with its core and peripheral functions. Reader can refer to M68HC11 E Series datasheet [20] and M68HC11 reference manual [21] for more detail.

#### 3.1 General Description

M68HC11 is a family of 8-bit general purpose microcontroller units. Members of this family differ from each other with small differences in their components. In this thesis a SystemC design best matches to this microcontroller unit family members is done. MC68HC11E9 microcontroller unit which is a M68HC11 family member is chosen as a model. MC68HC11E9 is chosen because it has most of the peripherals available in the family, it is used in Motorola Semiconductors Evaluation Board (EVBU) which is used widely for educational purposes and vast amount of information on this model is available.

There are different packaging options for different members of M68HC11 family of microcontroller unit. For MC68HC11E9 52-pin plastic leaded chip carrier (PLCC), 52-pin windowed ceramic-leaded chip carrier (CLCC), 64-pin quad flat pack (QFP), 52-pin thin quad flat pack (TQFP) and 56 pin shrink dual in-line package (SDIP) options are available. Most of the pins serve at least two different functions. Pin assignments for 52-pin PLCC package option of MC68HC11E9 are presented in Figure 3.1.

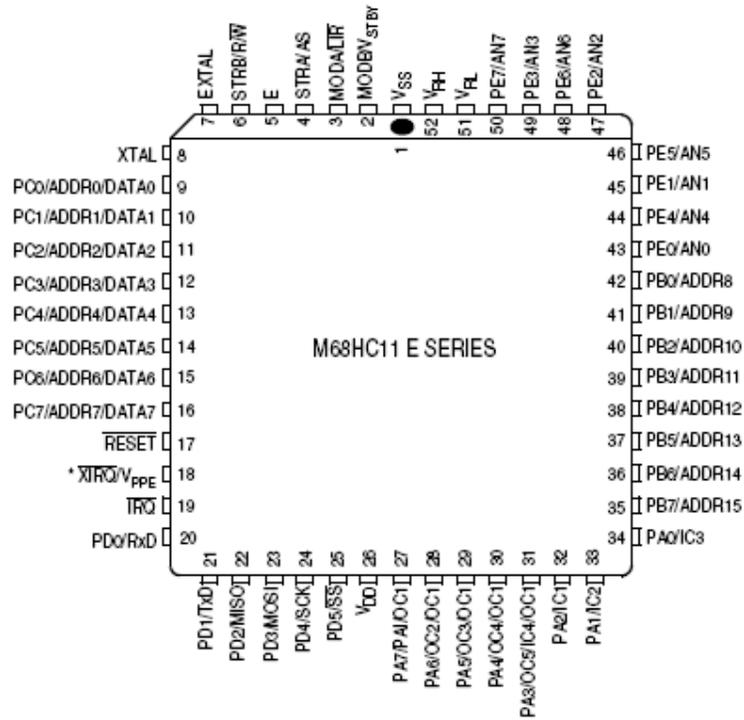


Figure 3.1: MC68HC11E9 Pin Assignments

MC68HC11E9 can operate at external clock frequencies up to 8 MHz which generates up to 2 MHz of internal bus clock. It has peripheral functions including an 8-channel A/D (analog-to-digital) converter which has 8-bits of resolution, an asynchronous serial communications interface (SCI), a synchronous serial peripheral interface (SPI), a 16-bit, free running main timer system with three input-capture lines, five output compare lines and a real-time interrupt function. An 8-bit pulse accumulator subsystem that can count external events or measure externally applied signal periods is also included in MC68HC11E9. On-chip memory systems of MC68HC11E9 include 8 Kbytes of read-only memory (ROM), 512 bytes of electrically erasable programmable ROM (EEPROM) and 256 bytes of random access memory (RAM). These peripherals and on-chip memory can be seen on block diagram of MC68HC11E9 in Figure 3.2

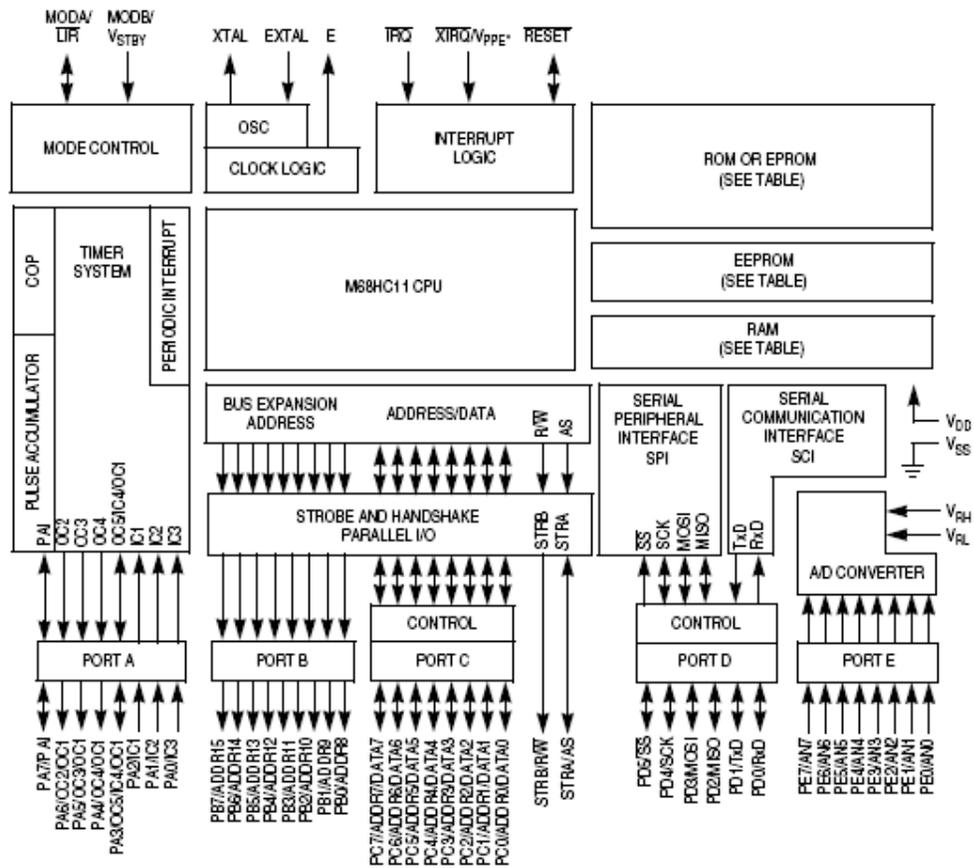


Figure 3.2: Block Diagram of MC68HC11E9

### 3.2 Operation Modes of MC68HC11

MC68HC11 microcontroller unit has two main operation modes. These are:

- Single-chip operation mode
- Expanded operation mode

Each of these main operation modes also has two variations. These variations are called normal and special variations. All operation modes are listed below:

- Normal single-chip operation mode
- Special bootstrap mode (special variation of single-chip mode)
- Normal expanded operation mode
- Special test mode (special variation of expanded operation mode)

In normal single-chip mode of operation everything that will be accessed using address and data buses is assumed to be contained in microcontroller chip. There are no external memory elements or peripherals in this mode of operation.

Normal variation of expanded operation mode is used for accessing external memory and/or peripherals. In this mode of operation, address/data bus is multiplexed and available on port B and port C pins. Normal expanded mode of operation has two additional control pins.

Special bootstrap mode is used for downloading programs into on-chip RAM at startup using asynchronous serial communications interface (SCI). Special test mode generally used for factory testing of microcontroller.

### 3.3 On-Chip Memory Systems

MC68HC11 microcontroller unit includes 512 bytes of random access memory (RAM), 12 Kbytes of program (user) read-only memory (ROM), 192 bytes of bootloader ROM and 512 bytes of electrically erasable programmable ROM (EEPROM). Other members of M68HC11 family may have different sizes of RAM, ROM and EEPROM memories. ROM or EEPROM memories are not included or disabled in some variations of microcontroller units in M68HC11 family. Memory map of MC68HC11E9 is presented in Figure 3.3.

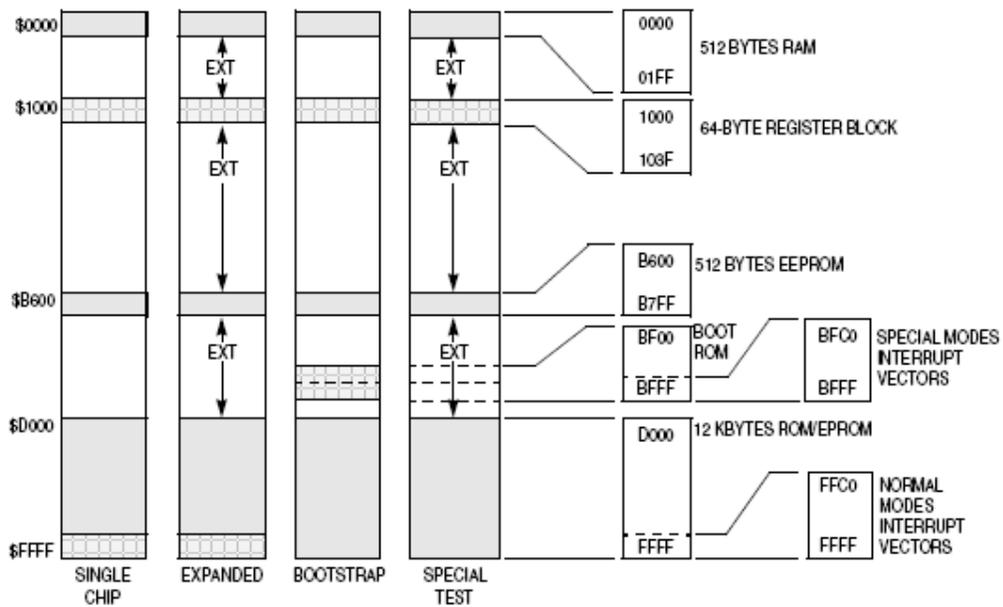


Figure 3.3: Memory Map of MC68HC11E9

### **3.3.1 Read-Only Memory (ROM)**

There are two read only memories (ROMs) on MC68HC11 microcontrollers. One of them is called program ROM or user ROM and the other one is bootloader ROM.

As its name implies, program ROM contains instructions of user's program. Program ROM is not writable or changeable by user after fabrication, so instructions of program are stored into this memory when the microcontroller unit is manufactured. Program ROM occupies 12 Kbytes in 64 Kbytes memory space of microcontroller unit. User may disable on-chip ROM if not needed. If ROM is disabled it does not occupy area in memory space anymore.

Other read-only memory included in MC68HC11 microcontrollers is 192 byte bootloader ROM. This memory unit is used for loading bootloader program in special bootstrap mode. In normal operation modes, bootloader ROM is disabled and does not occupy any area in memory space of microcontroller.

### **3.3.2 Random-Access Memory (RAM)**

Random access memory can be thought as a temporary storage space during run-time. User's program accesses to this memory during execution and uses this memory space for storing and reading variables for making operations on them.

RAM occupies first 512 bytes of memory space normally, but it can be mapped to beginning of other blocks of memory space in first 64 CPU cycles of microcontroller operation.

### **3.3.3 Electrically Erasable Programmable ROM (EEPROM)**

EEPROM allows user to store and change programs when needed after manufacturing of microcontroller unit is completed. With the help of this memory unit, user programs may be updated at any time.

MC68HC11E9 has 512 bytes of on-chip EEPROM. In MC68HC11 microcontrollers in addition to this EEPROM unit, there is another EEPROM byte which is used for controlling some basic features. This EEPROM byte is named as “CONFIG” register.

EEPROM in the MC68HC11E9 is fixed at locations \$B600-\$B7FF. Reads from EEPROM memory can be done by a read operation from address of location to be read. Writes to EEPROM is controlled by EEPROM programming register (PPROG). For writing to a location, first EEPROM programming voltage should be enabled using EEPGM bit in PPROG register, then write operation should be performed to the location and finally EEPROM programming voltage should be disabled again. There are different ways to erase EEPROM locations. These are; “byte erase”, in which EEPROM locations are erased one by one; “row erase”, in which EEPROM locations are erased in rows and finally “bulk erase” in which all bytes of EEPROM are erased at once. These methods are not applicable to erasure of CONFIG register.

### **3.4 Central Processing Unit (CPU)**

MC68HC11 microcontrollers utilize M68HC11 central processing unit (CPU). CPU is responsible for executing software instructions in their programmed sequence. There are 235 different operation codes (opcodes) in M68HC11 instruction set, using page-select prebytes before opcodes, some new instructions are specified and a total number of 310 instructions are reached.

The M68HC11 CPU accesses all input/output, peripheral and memory locations as any location in memory space. This technique of access is called memory-mapped I/O.

M68HC11 CPU contains two accumulators named accumulator A and B. Accumulators A and B form double accumulator D together. There are two index registers (IX and IY) in CPU which are generally used for calculating indexed addresses. Stack pointer (SP), which is a CPU register, always points to next free location of stack area. Program Counter register (PC), as its name implies, holds address of next program instruction. Condition code register (CCR) holds status indicator flags that indicate status of CPU after last instruction is executed. Table 3.1 shows internal registers of M68HC11 central processing unit.

Table 3.1: Internal Registers of M68HC11 CPU

| <b>Register</b>               | <b>Explanation</b>  |
|-------------------------------|---|
| Accumulator A                 | 8 bit accumulator   |
| Accumulator B                 | 8 bit accumulator   |
| Double Accumulator D          | Concatenation of accumulators A and B (A:B)   |
| Index Register IX             | 16 bit index register   |
| Index Register IY             | 16 bit index register   |
| Stack Pointer (SP)            | 16 bit stack location pointer   |
| Program Counter (PC)          | 16 bit program instruction pointer  |
| Condition Code Register (CCR) | 8 bit register with status indicators, interrupt masking bits and STOP disable bit. |

Five status indicators in condition code register give some information on execution and results of instructions. Two interrupt masking bits are used for masking global interrupts and interrupts generated from  $\overline{XIRQ}$  pin. STOP disable bit is used for avoiding STOP instruction to stop microcontroller operation. Table 3.2 summarizes meanings of flags in condition code register.

Table 3.2: Condition Codes Register

| CCR Bit              | Bit Location | Meaning   |
|----------------------|--------------|---|
| Carry (C)            | 0            | A carry out or borrow has occurred as a result of operation.              |
| Overflow (V)         | 1            | Indicates a two's complement overflow condition as a result of operation. |
| Zero (Z)             | 2            | Informs whether the result of operation is zero or not.                   |
| Negative (N)         | 3            | Indicates that result of the operation is negative.                       |
| I Interrupt Mask (I) | 4            | Disables all maskable interrupts.   |
| Half Carry (H)       | 5            | Set if carry from bit 3 has occurred.                                     |
| X Interrupt Mask (X) | 6            | Disables $\overline{XIRQ}$ pin interrupt.                                 |
| STOP Disable (S)     | 7            | Disables STOP instruction.  |

### **3.5 Addressing Modes**

M68HC11 uses six different ways for accessing memory. These are called “addressing modes”. Addressing modes are techniques for calculating address information for memory access. Different modes of addressing in M68HC11 are immediate, extended, direct, indexed, inherent and relative addressing modes. These are studied in detail in following subsections.

#### **3.5.1 Immediate Addressing Mode**

In immediate addressing mode there is no need to calculate an effective address to access data. Data needed is contained in bytes following opcode. Number of data bytes is specific to opcode.

#### **3.5.2 Extended Addressing Mode**

In the extended addressing mode, two bytes following the opcode are effective address of the data needed. No other calculations are needed; reading these two bytes is enough for knowing effective address.

#### **3.5.3 Direct Addressing Mode**

Direct addressing mode which is also called “zero page addressing mode” is used for accessing only to first 256 locations of memory space. High order byte of effective address is zero and low order byte of the effective address is contained in byte following the opcode.

### **3.5.4 Indexed Addressing Mode**

In the indexed addressing mode, an offset value is contained in the byte following the opcode. Effective address is calculated by adding this offset value to one of the index registers. Information on which index register will be used for address calculation is specific to opcode.

### **3.5.5 Inherent Addressing Mode**

In the inherent addressing mode no address information is needed because actually no addressing is done. Information on operands is available in opcode.

### **3.5.6 Relative Addressing Mode**

Relative addressing mode is used for accessing a location within a range of  $\pm 128$  relative to program counter. Offset to program counter is available in the byte following the opcode. This offset value is actually a signed byte. Relative addressing mode is only used for branching program execution purposes.

## **3.6 Parallel Input / Output (I/O)**

The MC68HC11E9 has five input / output (I/O) ports and 40 I/O pins that are shared between these ports. All I/O pins also have alternative functions. These alternative functions are used by peripheral systems of microcontroller unit. Input / output ports are named port A to port E. Number of pins on these ports may not be equal to each other. Some of these pins are fixed-direction input or fixed-direction output pins and some of them are bidirectional pins.

### 3.6.1 Data Ports

Port A is an 8-bit port with three fixed-direction input pins, four fixed-direction output pins and one bidirectional pin. Port A pin 7 can be configured as input or output port using DDRA7 bit in PACTL register. Port A pins can be used as general purpose input/output pins and they also have alternative functions. Table 3.3 summarizes alternative functions of port A pins.

Table 3.3: Summary of Port A Pins

| Pin | GPIO feature           | Alternative function(s)                  |
|-----|------------------------|--|
| PA0 | Fixed-direction input  | Input capture                            |
| PA1 | Fixed-direction input  | Input capture                            |
| PA2 | Fixed-direction input  | Input capture                            |
| PA3 | Fixed-direction output | Output compare                           |
| PA4 | Fixed-direction output | Output compare                           |
| PA5 | Fixed-direction output | Output compare                           |
| PA6 | Fixed-direction output | Output compare                           |
| PA7 | Bidirectional          | Pulse accumulator input / Output compare |

Port B and port C actually function together with STRA and STRB pins of microcontroller unit. These ports and pins are all together form handshake I/O

subsystem in single-chip mode and multiplexed address/data bus in expanded mode. Information on handshake I/O subsystem can be found in section 3.6.2.

Port B is an 8-bit port which has all of its pins as fixed-direction outputs. As these pins can be used for general purpose output, they also have alternative functions in expanded operation mode. Port B is a part of handshake I/O subsystem. Handshake I/O properties of port B is summarized in section 3.6.2. In expanded operation mode, port B serves as high order byte of address information. Table 3.4 presents functions of port B pins.

Table 3.4: Summary of Port B Pins

| <b>Pin</b> | <b>GPIO feature</b>    | <b>Alternative function(s)</b> |
|------------|------------------------|--------------------------------|
| PB0        | Fixed-direction output | Address bus bit 8 (A8)         |
| PB1        | Fixed-direction output | Address bus bit 9 (A9)         |
| PB2        | Fixed-direction output | Address bus bit 10 (A10)       |
| PB3        | Fixed-direction output | Address bus bit 11 (A11)       |
| PB4        | Fixed-direction output | Address bus bit 12 (A12)       |
| PB5        | Fixed-direction output | Address bus bit 13 (A13)       |
| PB6        | Fixed-direction output | Address bus bit 14 (A14)       |
| PB7        | Fixed-direction output | Address bus bit 15 (A15)       |

Port C is an 8-bit port. All pins of port C can be used as bidirectional general purpose input/output pins. Port C is a part of handshake I/O subsystem. In expanded mode of microcontroller operation, this port is used for multiplexed address / data bus. Table 3.5 lists functions of port C pins.

Table 3.5: Summary of Port C Pins

| <b>Pin</b> | <b>GPIO feature</b> | <b>Alternative function(s)</b> |
|------------|---------------------|--------------------------------|
| PC0        | Bidirectional       | Address / data bus bit 0 (AD0) |
| PC1        | Bidirectional       | Address / data bus bit 1 (AD1) |
| PC2        | Bidirectional       | Address / data bus bit 2 (AD2) |
| PC3        | Bidirectional       | Address / data bus bit 3 (AD3) |
| PC4        | Bidirectional       | Address / data bus bit 4 (AD4) |
| PC5        | Bidirectional       | Address / data bus bit 5 (AD5) |
| PC6        | Bidirectional       | Address / data bus bit 6 (AD6) |
| PC7        | Bidirectional       | Address / data bus bit 7 (AD7) |

Port D is a 6-bit bidirectional parallel data port. Two of port D pins alternatively function as part of asynchronous communications interface (SCI) subsystem. Other four pins alternatively function as a part of synchronous serial peripheral interface (SPI) subsystem. All six pins of port D can also be used for general purpose input/output functions. Summary on port D functions can be found in Table 3.6.

Table 3.6: Summary of Port D Pins

| Pin | GPIO feature  | Alternative function(s)              |
|-----|---------------|--------------------------------------|
| PD0 | Bidirectional | SCI Receive data (RxD)               |
| PD1 | Bidirectional | SCI Transmit data (TxD)              |
| PD2 | Bidirectional | SPI Master in / slave out (MISO)     |
| PD3 | Bidirectional | SPI Master out / slave in (MOSI)     |
| PD4 | Bidirectional | SPI Master clock out (SCK)           |
| PD5 | Bidirectional | SPI Slave select ( $\overline{SS}$ ) |

Port E is an 8-bit fixed-direction input port. Pins of port E alternatively function as analog-to-digital (A/D) converter channel inputs.

### 3.6.2 Handshake I/O Subsystem

Handshake I/O subsystem is used for sending and receiving data to external devices in a more guaranteed way than normal parallel I/O. Ports B and C, STRA input pin, STRB output pin are parts of this subsystem. Each device or only one device that takes place in data transfer informs other side when it reads and/or writes data to port. Operation mode of handshake I/O subsystem determines handshaking rules. Operation modes are called simple strobe mode, full-input handshake mode and full-output handshake mode.

In simple strobe mode, port B is used as a simple output port which works together with STRB strobe output and port C is used as a simple latching input together

with STRA strobe input. Input and output work independent of each other. In simple strobe mode of handshake operation, a strobe signal is generated at STRB pin whenever a data is written to port B. Data is read from port C and latched when an active edge is encountered at STRA input.

Only port C, STRA and STRB pins are used in full-input handshake mode. In this mode, a selected edge at STRA input causes data on port C to be latched into a register and negation of STRB output. When the latched data is read in microcontroller, STRB output is asserted again to indicate that data reception is complete. In this mode, external system knows when to write data on port C using STRB information. As a summary, data sending system informs receiving system when it writes data to port and data receiving system informs data sending system when it reads data from port.

In full-output handshake mode, external system is informed via STRB pin when a data is written on port C and ready signal is read from STRA pin which indicates that external system has read data from port. Full-output and full-input handshake modes can be thought as two different ends of data receiving and sending systems. Full-output handshake mode has a variation named three-state full-output handshake mode. In this mode, port C pins becomes driven outputs when STRA goes to its active level.

### **3.7 Synchronous Serial Peripheral Interface (SPI)**

Synchronous serial peripheral interface (SPI) can be used for communicating other microcontroller units or peripheral devices. SPI system can be configured as master or slave. If it is configured as master, communication speed can be as high as 1 Mbps and if it is configured as slave communication speed can be as high as 2 Mbps.

In an SPI transfer, a clock, which is generated by master device, synchronizes shifting and sampling jobs. Shifting out of data and sampling of incoming bit occurs

at opposite edges of clock line, so reception and transmission occurs simultaneously. Any slave device which is not selected via slave select input does not read or write data from / to SPI bus.

In an SPI system only one master bus should be in communication network at a moment. When SPI system is configured as a master and another device becomes bus master, this situation is detected and SPI output drivers are disabled to avoid harms. Error detection system which detects this condition is named “multiple-master fault detector”. There is another error detector in SPI which is called “write-collision detector”. This system detects and avoids a write attempt to serial shift register while a transfer is in progress.

There are two different transfer formats of SPI system and these transfer formats have two different clock polarity variations. Figures 3.4 and 3.5 which are taken from “MC68HC11 Reference Manual” presents timing diagrams of CPHA equals zero and CPHA equals one SPI transfer formats respectively.

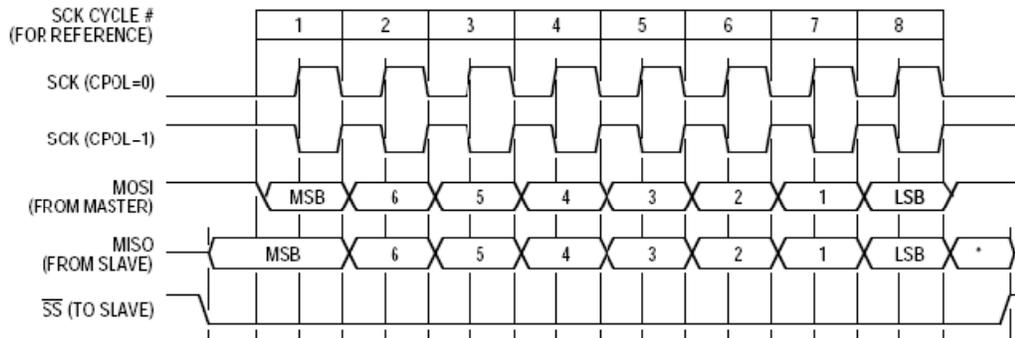


Figure 3.4: CPHA Equals Zero SPI Transfer Format

As seen in Figure 3.4, in CPHA equals zero transfer format, transfer is started when  $\overline{SS}$  line goes to low. When CPHA equals one transfer format is selected, transfer is started with first selected clock edge.  $\overline{SS}$  line should be low for both transfer formats.

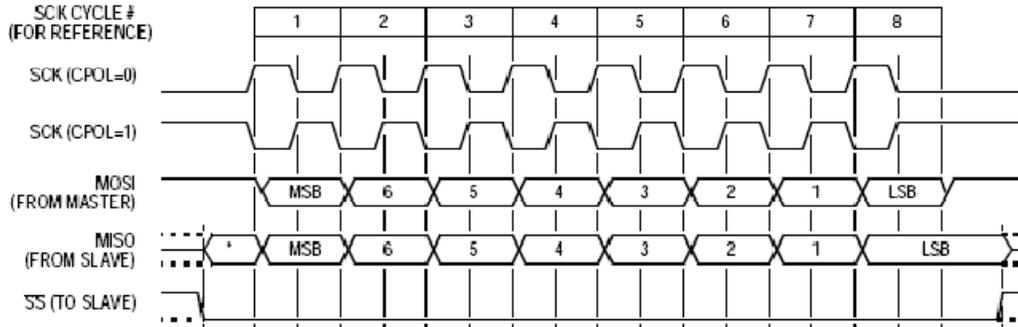


Figure 3.5: CPHA Equals One SPI Transfer Format

### 3.8 Asynchronous Serial Communications Interface (SCI)

Asynchronous serial communications interface (SCI) used in MC68HC11 microcontrollers is full-duplex and uses one start bit, eight or nine data bits, one stop bit none return to zero (NRZ) transfer format. SCI system has a transmitter and a receiver part. Although they use same transfer formats and same baud rates, transmitter and receiver parts of SCI system operate independent from each other. Baud rates are generated using PH2-clock of microcontroller unit.

Transmitter of SCI is double-buffered so a character can be written to transmit buffer while transmission of a character is in progress. Break and idle characters can be queued for transmission. Transmitter can generate “transmit complete”

interrupt when it finishes sending all data in its queue or “transmitter data register empty” interrupt when transmit data register is available for new character.

Receiver of SCI system is also double-buffered, so software has some time to read received character before next character is received. Receiver can go to sleep mode and wake-up when selected event occurs on line. Wake-up can be initiated by an idle-line or address-mark detection in received data. SCI receiver of MC68HC11 has an advanced noise detection and correction technique. This technique is called “data sampling technique”. Receiver samples line data with a clock frequency of 16 times the baud rate and uses these samples to decide logic level of received bit. A start bit is recognized if a zero is sampled after three ones and at least two of third fifth and seventh samples are zeroes. Reception is synchronized to start bit in this way. Other bits are recognized by using eighth ninth and tenth samples taken. If at least one of these samples does not agree with others, noise flag is set. Received bit is decided by using majority of samples. Generally detection would be correct even there is noise on line.

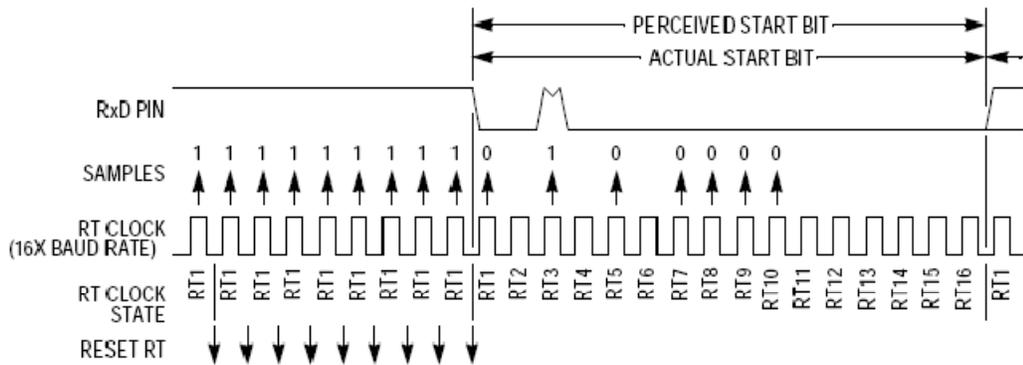


Figure 3.6: Start Bit Reception

Figure 3.6 shows an example of start bit detection with some noise on line. Although in this example start bit is received correctly, there may be some shifts in locations of perceived and actual start bits due to amount of noise on line. If these shifts are in an acceptable range, data sampling is expected to correctly receive rest of incoming character.

### **3.9 Main Timer and Real Time Interrupt**

Main timer system has a free running 16-bit counter. Counting frequency of this counter can be programmed by user. Main time system includes three input capture and four output compare subsystems, real time interrupt logic and computer operating properly (COP) watchdog timer.

Input capture function records value of free running counter when a selected edge is detected on corresponding input line. Input capture functions can generate interrupt requests. This feature of input capture functions can be used for measuring period or length of a signal on input pin.

Output compare functions generate an output when selected time has been reached. In order to do this, an output compare register is loaded with 16-bit value and when free running counter reaches to this value, a signal is outputted to indicate the condition. Output compare functions can also generate interrupts when selected count is reached by free running counter.

Computer operation properly (COP) watchdog timer function is used for resetting microcontroller if a software error occurs. COP timer counts up with user selected frequency and when it overflows microcontroller reset signal is generated automatically. Software which is running properly should touch to COP timer and clear it if COP is enabled. COP overflow means that software is not running properly so it could not clear COP timer, so microcontroller should be reset to correct this situation with a fresh startup.

Real time interrupt logic generates hardware interrupts with a user defined period. RTI can be used for sharing microcontroller time between different tasks for multi-tasking.

### **3.10 Pulse Accumulator**

Pulse accumulator is actually an 8-bit counter that is used for counting selected events or gated time accumulation. It can generate interrupt requests at every event detected or when overflow occurs in 8-bit counter.

In event counting mode, pulse accumulator counts number of edges signal on PAI input. It can be configured to count positive or negative edges.

In gated time accumulation mode, 8 bit counter is incremented at every 64 E-clock cycles of microcontroller. The name “gated” comes from the feature that, counting operation is gated to PAI input. That means counting stops if PAI input is not its selected level. This feature can be used to measure durations of events.

## **CHAPTER 4**

### **DESIGN OF MC68HC11 MICROCONTROLLER MODEL USING SYSTEMC**

The object oriented nature of SystemC allows parts of a system to be designed as separate modules. These modules can also be divided into sub modules. This ability of SystemC allows unnecessary modules to be extracted from the whole design, adding new modules to the design and using different abstraction levels within different modules. Opportunity to use different abstraction levels for different modules makes it possible to refine modules of a system independently without affecting rest of the design.

In this thesis, the MC68HC11 microcontroller unit is divided into sub modules and each sub module is implemented in SystemC as a part of overall microcontroller system. These sub modules are, address bus controller, clock generator, arithmetic and logic unit, controller unit, handshake I/O sub-system, timer system, RAM, ROM, register file and serial communications modules. SystemC modules have been shaped according to design complexity, similarities and common requirements of peripherals in microcontroller unit. Modules that constitute microcontroller core are tested and verified before they are brought together. Although original MC68HC11E9 model has an analog to digital converter subsystem; SystemC design made in this thesis does not contain analog to digital converter module because standard SystemC library does not have data types for modeling analog hardware.

SystemC implementation of MC68HC11 microcontroller unit is explained in this chapter. Internal structure of designed microcontroller is given. Modules that construct microcontroller are explained and their test results are presented.

#### **4.1 MC68HC11 SystemC Model Internal Structure**

Inside the MC68HC11 microcontroller model, all modules are interconnected to each other by internal signals. Correct timing of these internal signals is very important for correct operation of overall microcontroller model. Internal signals may be divided into four categories mainly. These are internal clock signals that synchronize modules with each other, control bus signals that are used for maintaining correct operation of modules, data bus signals that carry data and address bus signals that carry address information for memory modules. Internal structure of MC68HC11 SystemC model and interconnections between modules are presented in Figure 4.1 without much detail.

The developed SystemC model of MC68HC11 is verified using the test code that is given in Appendix B. This test code contains all addressing modes and all types of instructions. The test program has been run on original MC68HC11 EVBU board in microprocessor design laboratory of METU electrical and electronics engineering department. Test results that are presented in Appendix B verified that implemented microcontroller is working same as the original microcontroller.

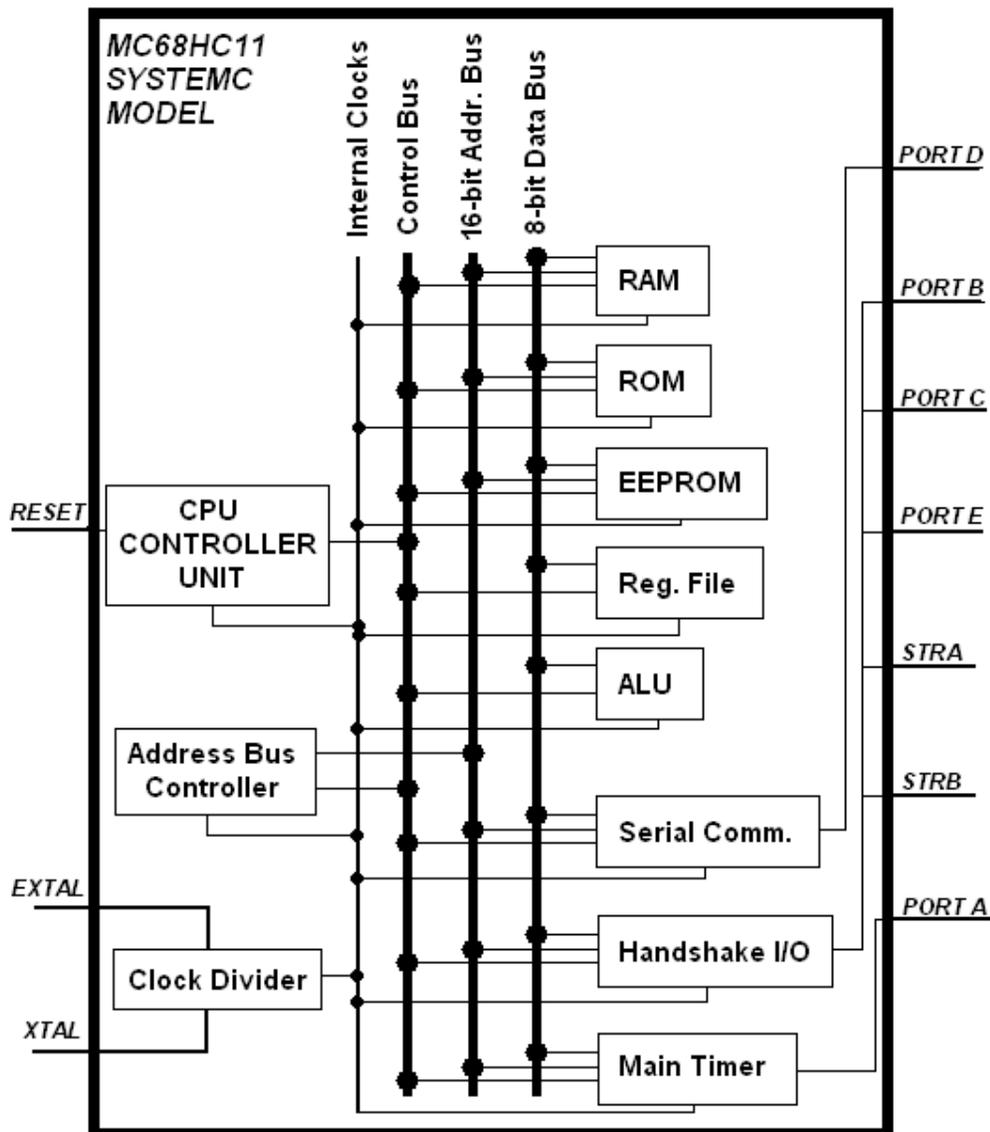


Figure 4.1: Internal Structure of MC68HC11 SystemC Model

Figure 4.2 shows datapath of the microprocessor. Source of the data that will be processed can be register file or data bus.

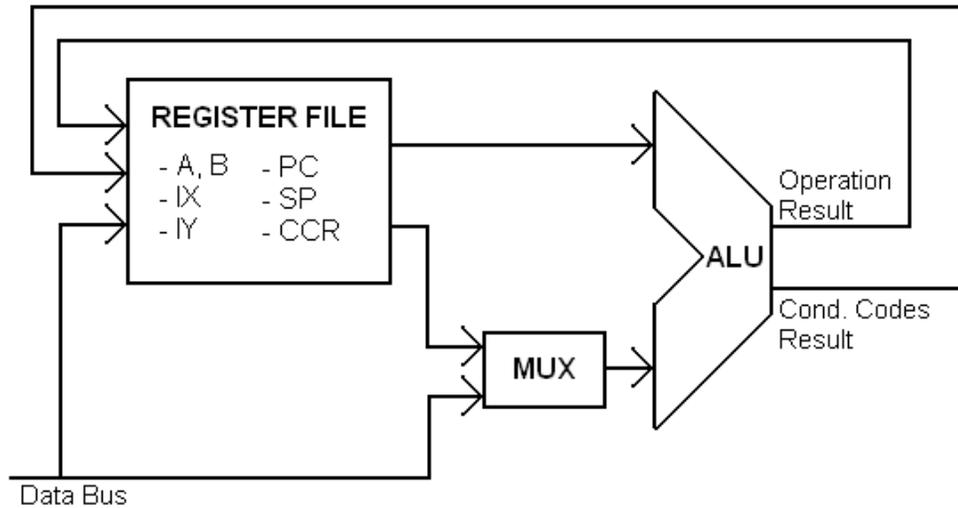


Figure 4.2: Datapath of Microprocessor

## 4.2 CPU Controller Unit

CPU controller unit module is central controlling element of MC68HC11 model. This module is responsible for generating appropriate control signals for other modules and to maintain correct operation of overall system. CPU controller unit chooses correct signals with correct timing for other modules, according to state of microcontroller, events occurring and instruction that is being executed. This module actually consists of a state machine that is performing different tasks at different clock edges. These tasks include opcode fetching, instruction decoding, selecting arithmetic logic unit parameters and deciding owners of data and address buses. A simple block diagram of CPU controller is shown in Figure 4.3.

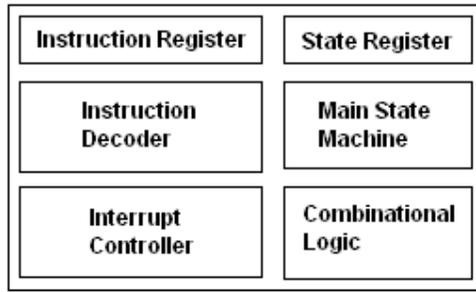


Figure 4.3: Block Diagram of CPU Controller

Synchronization between different modules of MC68HC11 model is done using four internal clock cycles which constitute a bus cycle (E-Clock cycle). Modules perform different jobs in each of these four internal clock cycles. Main processes of CPU controller unit are sensitive to all of these four clock cycles and generate different signals and perform different tasks at each clock cycle. Figure 4.4 shows these four internal clock cycles.

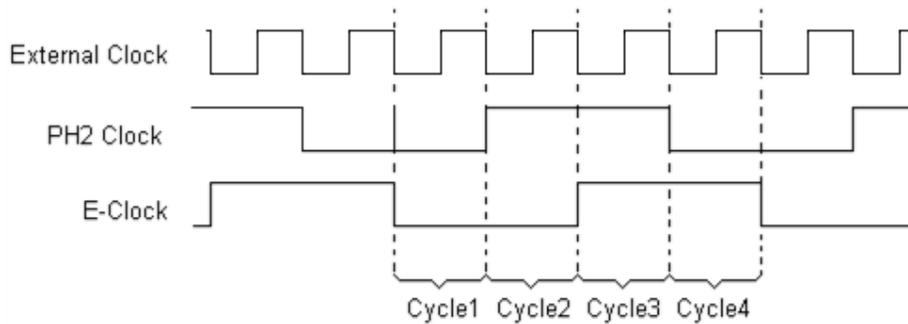


Figure 4.4: Internal Clock Cycles of a Bus Cycle

As seen in Figure 4.4, internal clock cycle 1 starts with negative edge of E-clock, cycle 2 starts with positive edge of PH2 clock, cycle 3 starts with positive edge of E-clock and cycle 4 starts with negative edge of PH2 clock.

At negative edge of E-clock which is called first edge, controller unit sends commands to register file module for incrementing, decrementing or not changing program counter and stack pointer. At positive edge of PH2 clock which is called second edge, chip select and read/write signals for RAM and ROM, data bus owner select signal, register file output select signal, ALU command, instruction type and B operand select signals, CCR mask information for instruction that is being executed are sent to interested modules. At positive edge of E-clock which is called third edge, condition code register is read, carry information is sent to ALU and address bus owner is selected. At fourth cycle, instruction is fetched and decoded if controller is in fetch state. Input source and destination for register file is selected.

An instruction execution consists of at least 2 bus cycles and at most 41 bus cycles. Instruction is fetched and decoded in first bus cycle and executed in the remaining cycles.

Other modules of MC68HC11 SystemC model are synchronous to these four internal clock cycles. Generally CPU controller unit puts a signal one or two clock cycles before that signal is read by target module. This guarantees a setup time for signals before they are being read. CPU controller unit decides signal values according to states of CPU. Different states of CPU are presented in Table 4.1 with explanations.

Table 4.1: CPU States

| <b><i>CPU State</i></b>      | <b><i>Explanation</i></b>  |
|------------------------------|--|
| START                        | Starting state of CPU. Internal register are loaded with their initial values. Devices are set to their initial states of operation. |
| Opcode Fetch                 | Opcode is fetched from memory.   |
| Opcode Fetch 2               | If first fetched opcode was page information, actual opcode is fetched from memory again.  |
| Read Extended Addr. High     | High order byte of effective address value in extended addressing mode is read from memory.  |
| Read Extended Addr. Low      | Low order byte of effective address value in extended addressing mode is read from memory.   |
| Calculate Ind. Addr. Low     | Low order byte of effective address value in indexed addressing mode is calculated.  |
| Calculate Ind. Addr. High    | High order byte of effective address value in indexed addressing mode is calculated.   |
| Calculate Rel. Addr. Low     | Low order byte of effective address value in relative addressing mode is calculated.   |
| Calculate Rel. Addr. High    | High order byte of effective address value in relative addressing mode is calculated.  |
| Read Direct Address          | Low order byte of effective address value in direct addressing mode is read from memory.   |
| 8 Bit Execution              | An 8 bit operation is executed.  |
| Arith. 16 Bit Execution Low  | Low order byte of 16 bit arithmetic operation is executed.   |
| Arith. 16 Bit Execution High | High order byte of 16 bit arithmetic operation is executed.  |
| Logic 16 Bit Execution Low   | Low order byte of 16 bit logical operation is executed.  |
| Logic 16 Bit Execution High  | High order byte of 16 bit logical operation is executed.   |

|                        |  |
|------------------------|--|
| Read Operand           | Operand is read into temporary register for use in following cycles of operation.                            |
| Read Execution Operand | An operand is read into temporary register that will be used for execution in following cycles of operation. |
| Write Memory 1         | Result of operation is being prepared to write to memory.  |
| Write Memory 2         | Result of operation that is prepared in Write Memory 1 state is stored into memory.                          |
| Multiplication         | CPU is performing a multiplication operation.  |
| Integer Division       | CPU is performing an integer division operation.   |
| Fractional Division    | CPU is performing a fractional division operation.   |
| Stack Operation        | An operation on stack space is being executed.   |
| Stack Operation INCSP  | An operation on stack space is being executed with incrementing stack pointer at the same time.              |
| Push Data              | Data is pushed onto stack.   |
| Pull Data              | Data is pulled from stack.   |
| Set Interrupt Mask     | Interrupt mask is set.   |
| Load Interrupt Vector  | Interrupt vector is loaded to program counter after an interrupt occurred.                                   |
| TEST                   | TEST instruction has put CPU in TEST state. CPU stays in this state until a reset.                           |
| ERROR                  | CPU should never be in this state. If CPU enters into this state that means an error has occurred.           |

### 4.3 Clock Divider

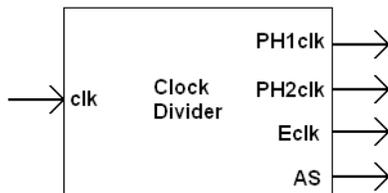


Figure 4.5: Clock Divider Symbol

Clock divider module takes external clock source as an input for generating internal clock signals and address strobe (AS) signal. Figure 4.5 shows input and output ports of clock divider. Internal clock signals of MC68HC11 MCU model are phase-1 (PH1) clock, phase-2 (PH2) clock and E-clock. Frequencies of PH1, PH2 and E-clock signals are equal to each other and  $\frac{1}{4}$  of the frequency of external clock input. PH1 and PH2 clocks are 180 degree shifted versions of each other. E-clock lags 90 degrees behind PH2 clock. AS signal low-to-high transition (positive edge) lags behind E-clock negative edge with a 45 degrees phase difference and AS signal remains high for a time equal to one external clock period.

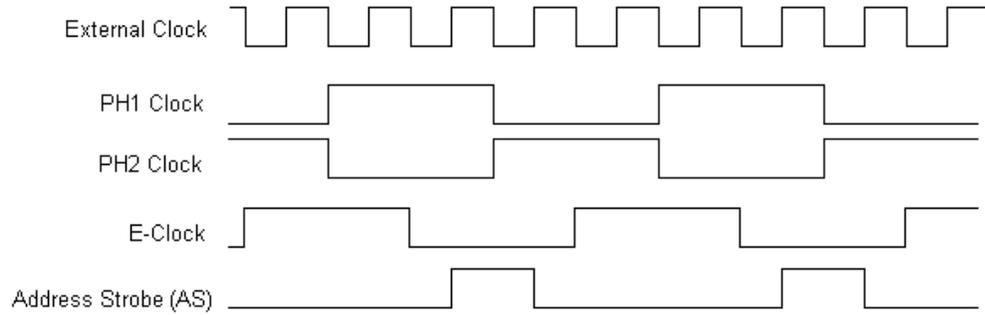


Figure 4.6: Internal Clock Signals of MC68HC11

For verification of clock divider module, external clock signal (XTAL\_Clock) with a frequency of 2 MHz is applied to the module input and outputs of the module are observed. Internal clock signals of original MC68HC11 MCU are presented in Figure 4.6. Resulting input and output signals waveforms of clock divider module simulation are shown in Figure 4.7.

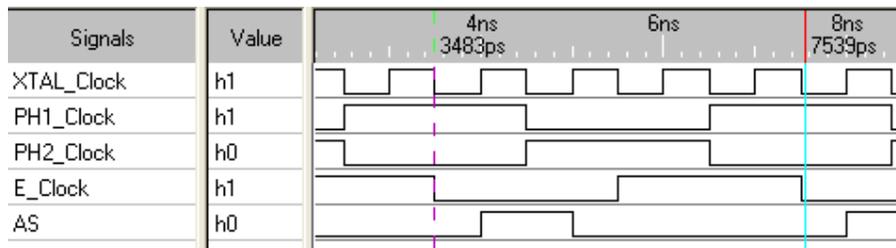


Figure 4.7: Resulting Waveforms of Clock Divider Simulation

#### 4.4 Arithmetic and Logic Unit (ALU)

ALU performs arithmetic and logic operations. It takes operands, command code, carry and instruction type as inputs, executes operation that is specified by command code and outputs result and condition codes. Input and output ports of ALU are shown in Figure 4.8. ALU uses an “instruction type” input (`alu_instr_type`) for deciding if the operation is 8-bit operation or high order part of a 16-bit operation. Since ALU can perform only 8-bit operations, for making 16-bit operations `alu_instr_type` is used in order to specify whether current operands are low order or high order operands.

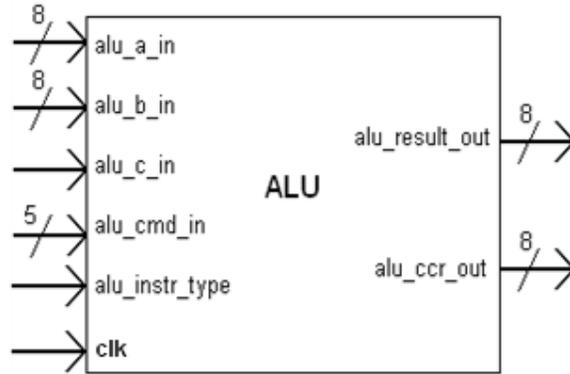


Figure 4.8: Arithmetic and Logic Unit Symbol

MC68HC11 model actually contains another module related to ALU, which is named ALU input multiplexer. This is a small module that takes two 8-bit inputs from register file and data bus and outputs one of them to arithmetic logic unit according to select signal that is asserted by controller unit.

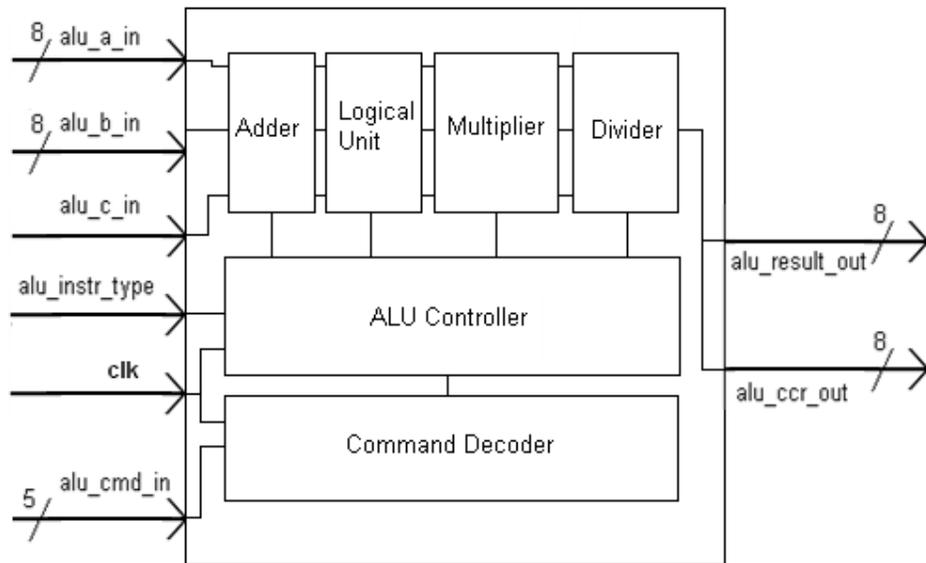


Figure 4.9: ALU Block Diagram

Block diagram of Arithmetic and logic unit is presented in Figure 4.9. ALU is synchronous and main process of this module is sensitive to negative edge of PH2-clock input. Different commands and corresponding operations are listed in Table 4.2.

Table 4.2: ALU Commands and Meanings.

| <b>ALU Commands</b> |             |  |
|---------------------|-------------|--|
| <b>Name</b>         | <b>Code</b> | <b>Operation</b>                                 |
| nop_cmd             | 0x00        | No operation                                     |
| add_cmd             | 0x01        | Addition of operand A to B without carry         |
| inc_cmd             | 0x02        | Increment operand A by one                       |
| sub_cmd             | 0x03        | Subtraction of operand B from A without borrow   |
| dec_cmd             | 0x04        | Decrement operand A by one.                      |
| and_cmd             | 0x05        | Logical AND operation on operand A and operand B |
| or_cmd              | 0x06        | Logical OR operation on operand A and operand B  |
| xor_cmd             | 0x07        | Logical XOR operation on operand A and operand B |
| com_cmd             | 0x10        | Take one's complement of operand A               |
| neg_cmd             | 0x11        | Take two's complement of operand A               |
| lsl_cmd             | 0x0B        | Logical shift left                               |
| asr_cmd             | 0x0C        | Arithmetic shift right                           |
| lsr_cmd             | 0x0D        | Logical shift right                              |
| rol_cmd             | 0x0E        | Rotate left                                      |
| ror_cmd             | 0x0F        | Rotate right                                     |
| daa_cmd             | 0x12        | Decimal adjust accumulator                       |
| addwc_cmd           | 0x08        | Add operands with carry                          |
| subwc_cmd           | 0x09        | Subtract operand B from operand A with borrow    |
| clr_cmd             | 0x0A        | Clear all bits                                   |

|               |      |  |
|---------------|------|--|
| addsigned_cmd | 0x13 | Make signed addition between operands  |
| andinv_cmd    | 0x14 | AND inverted version of operand A with operand B   |
| andinv2_cmd   | 0x15 | AND inverted version of operand B with operand A   |
| tst_cmd       | 0x19 | Subtract operand B from operand A. Status flags are different from standard subtraction. |
| strmul_cmd    | 0x16 | Start multiplication.  |
| mul_cmd       | 0x17 | Do multiplication.   |
| end_mul       | 0x18 | End multiplication, give result.   |
| ldn_cmd       | 0x1A | Load numerator for integer division.   |
| lddivn_cmd    | 0x1F | Load numerator for fractional division.  |
| div_cmd       | 0x1B | Do division.   |
| divresQ_cmd   | 0x1C | End division, give quotient.   |
| divresR_cmd   | 0x1D | Give remainder of division operation.  |
| fdivsub_cmd   | 0x1E | Do subtraction step of fractional division.  |

In order to verify ALU, a test bench that outputs all ALU commands and operands chosen for boundary condition testing is designed. Using this test bench, arithmetic and logic unit is verified. Figure 4.10 shows some part of the test results.

```

Operand1 = F5, Operand2 = C8, Carry = 0, Command = Addition without carry
Result = BD, CCR = 9

Operand1 = FF, Operand2 = 00, Carry = 0, Command = Increment
Result = 1, CCR = 0

Operand1 = C0, Operand2 = F0, Carry = 0, Command = Subtract
Result = D0, CCR = 9

Operand1 = 00, Operand2 = FF, Carry = 0, Command = Decrement
Result = FE, CCR = 8

Operand1 = AA, Operand2 = F0, Carry = 0, Command = AND
Result = A0, CCR = 8

Operand1 = 0F, Operand2 = 55, Carry = 0, Command = OR
Result = 5F, CCR = 0

Operand1 = 55, Operand2 = 0F, Carry = 0, Command = XOR
Result = 5A, CCR = 0

Operand1 = 00, Operand2 = AA, Carry = 0, Command = One's Complement
Result = 55, CCR = 1

Operand1 = 00, Operand2 = AA, Carry = 0, Command = Two's Complement
Result = 56, CCR = 1

Operand1 = 82, Operand2 = AA, Carry = 0, Command = Logical Shift Left
Result = 54, CCR = 3

Operand1 = 88, Operand2 = 81, Carry = 0, Command = Arithmetic Shift Right
Result = C0, CCR = 9

Operand1 = 55, Operand2 = A1, Carry = 0, Command = Logical Shift Right
Result = 50, CCR = 3

Operand1 = 52, Operand2 = 43, Carry = 1, Command = Rotate Left
Result = 87, CCR = A

Operand1 = 52, Operand2 = 43, Carry = 1, Command = Rotate Right
Result = A1, CCR = B

Operand1 = 7C, Operand2 = 00, Carry = 0, Command = Decimal Adjust Accumulator
Result = 82, CCR = A

```

Figure 4.10: Arithmetic and Logic Unit Test Results

Waveforms of test results for different instructions are given in Figures 4.11 to 4.24. ALU Operands and operations results can be seen in these figures. Results of the tests verify that ALU is operating correctly.

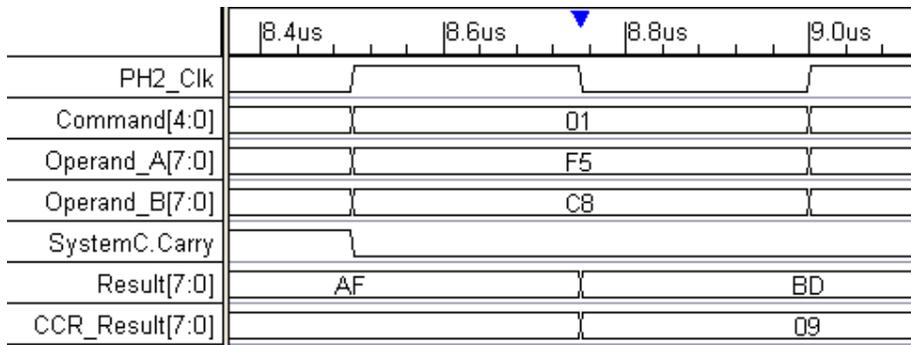


Figure 4.11: ALU Addition Test Waveforms

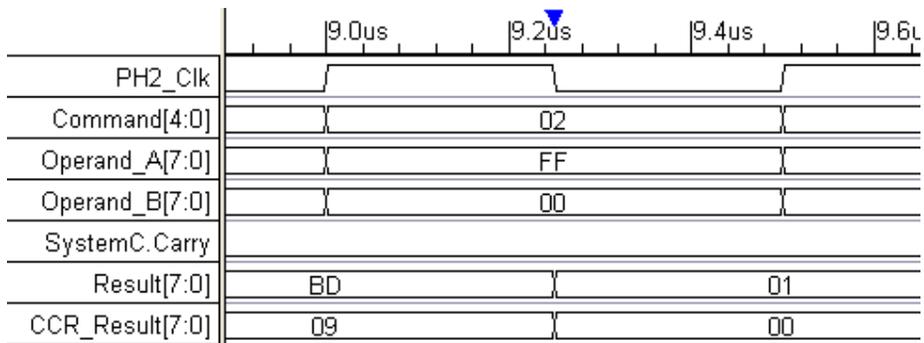


Figure 4.12: ALU Increment Test Waveforms

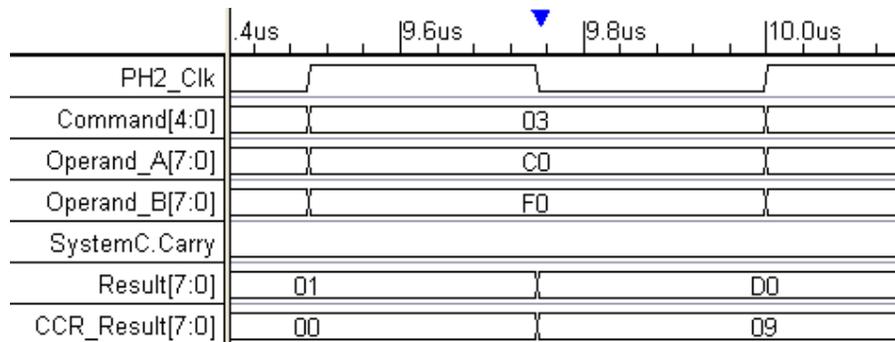


Figure 4.13: ALU Subtract Test Waveforms

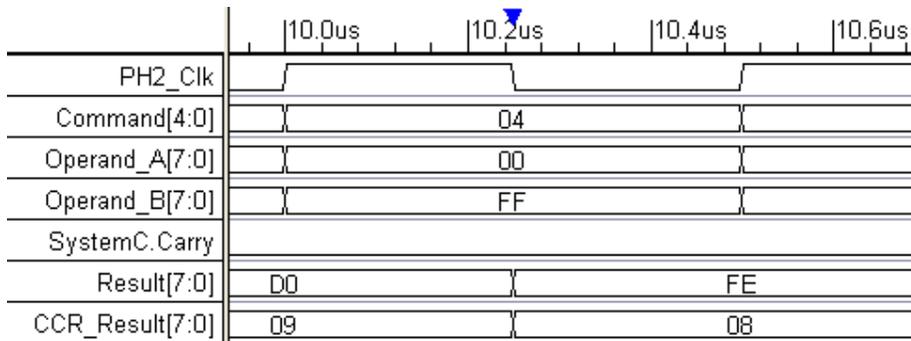


Figure 4.14: ALU Decrement Test Waveforms

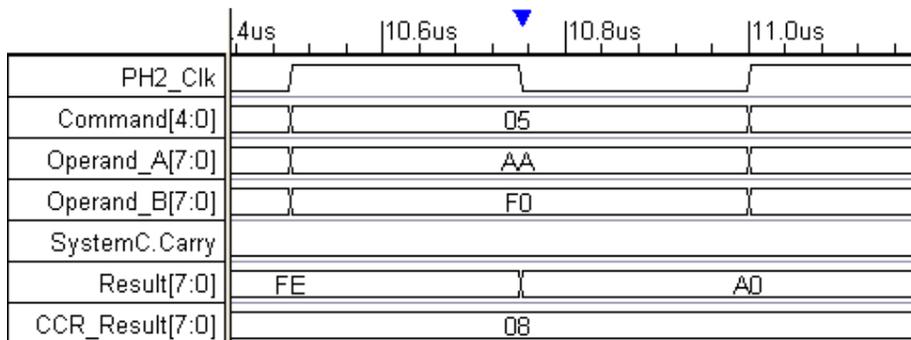


Figure 4.15: ALU AND Operation Test Waveforms

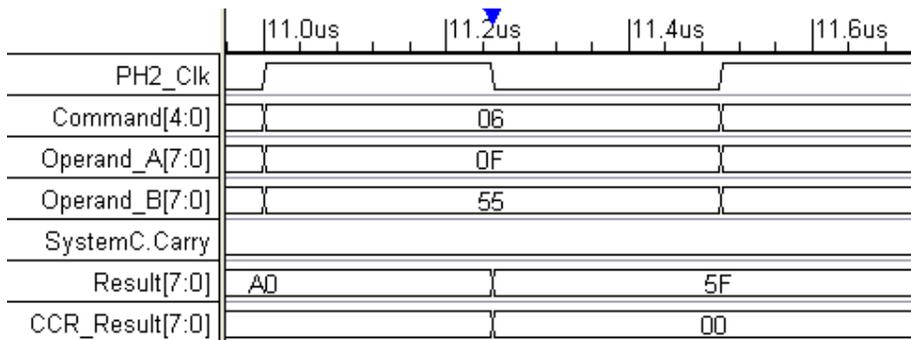


Figure 4.16: ALU OR Operation Test Waveforms

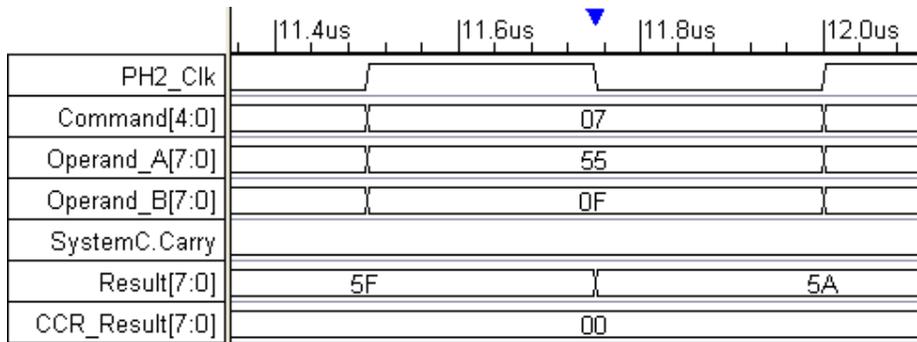


Figure 4.17: ALU XOR Operation Test Waveforms

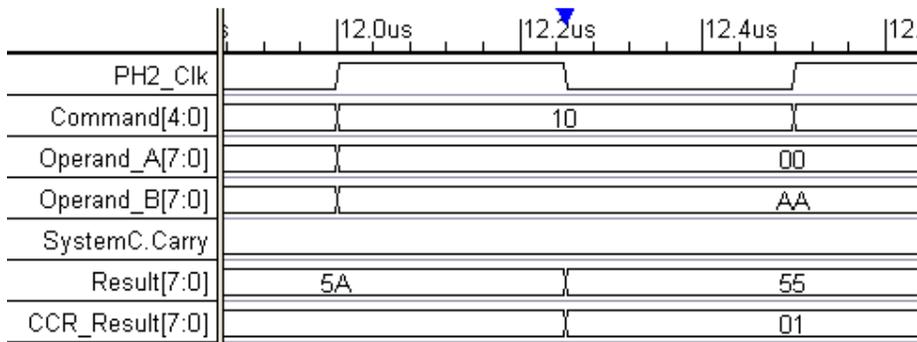


Figure 4.18: ALU Complement Test Waveforms

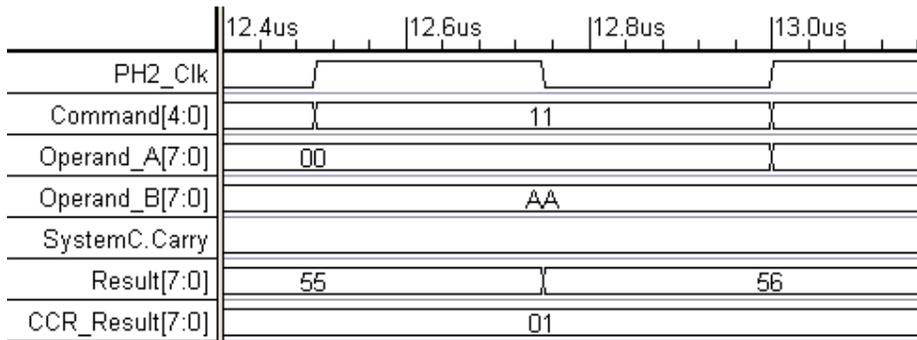


Figure 4.19: ALU Negate Test Waveforms

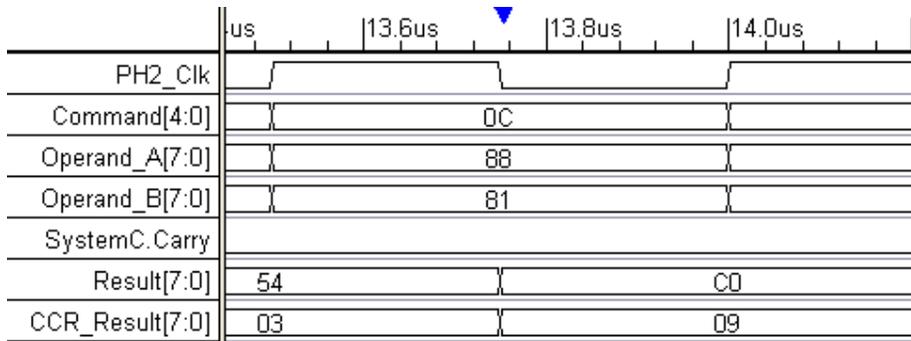


Figure 4.20: ALU Arithmetic Shift Right Test Waveforms

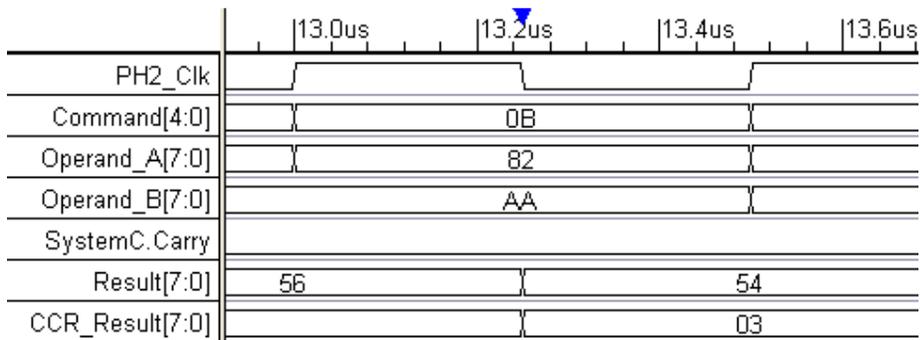


Figure 4.21: ALU Arithmetic / Logical Shift Left Test Waveforms

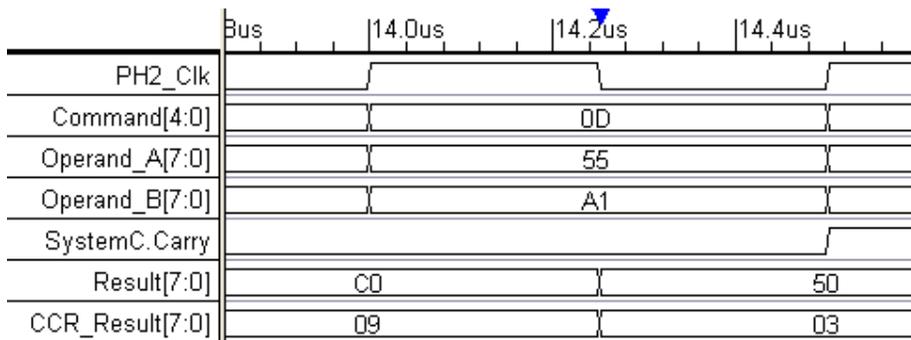


Figure 4.22: ALU Logical Shift Right Test Waveforms

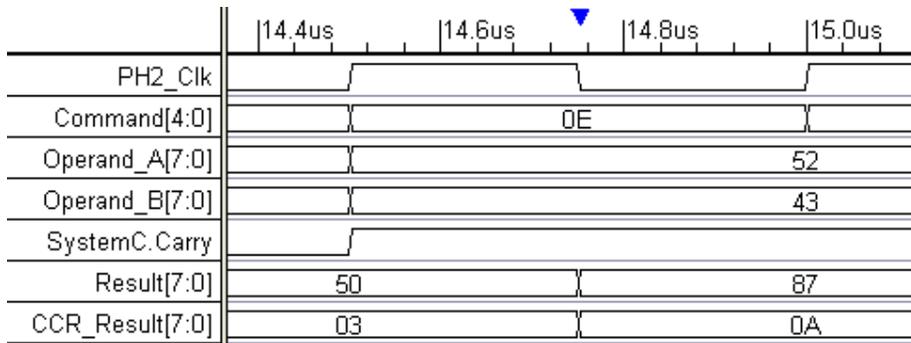


Figure 4.23: ALU Rotate Left Test Waveforms

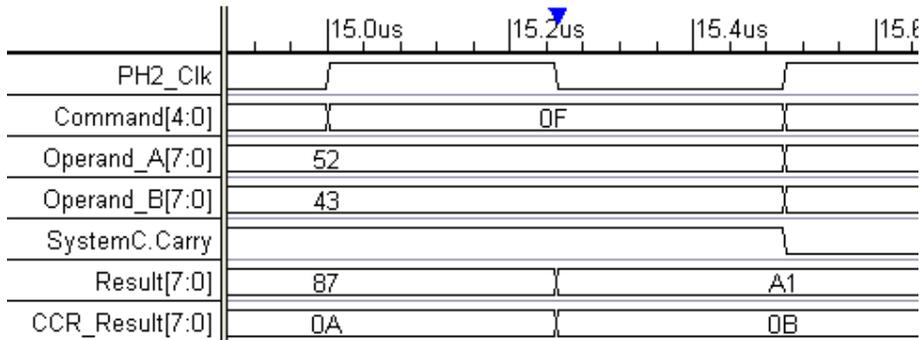


Figure 4.24: ALU Rotate Right Test Waveforms

## 4.5 Register File

Register file contains all CPU related registers. Accumulators A and B, index registers X and Y, program counter (PC), stack pointer (SP), condition codes register (CCR) are contained in this module. Register file also contains some temporary registers that are used for different purposes like holding address values or ALU operands temporarily. Inputs and outputs of register file are controlled by CPU controller unit. This module is synchronous and performs

additional tasks like incrementing program counter, incrementing or decrementing stack pointer or exchanging registers. Input and output ports of register file module are given in Figure 4.25.

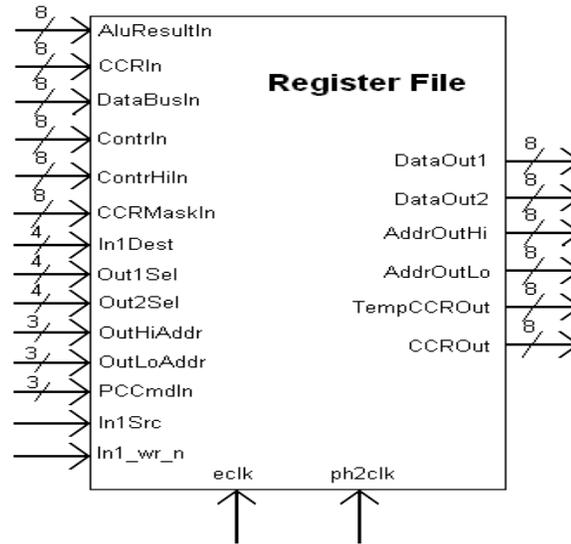


Figure 4.25: Register File Symbol.

User available registers contained in register file module are listed below:

- ACCD : 16-bit accumulator D register that consists of accumulator A (ACCA) and accumulator B (ACCB).
- IX : 16-bit index register X
- IY : 16-bit index register Y.
- SP : 16-bit stack pointer register.
- PC : 16-bit program counter register.
- CCR : 8-bit condition codes register.

A register file test bench is developed for verification of register file. This test bench performs tests on register file by writing values to its registers and then using these values in registers for address outputs or data outputs. Outputs of register file are compared with previously written values. Figure 4.26 shows resulting waveforms and Figure 4.27 shows console output of simulation.

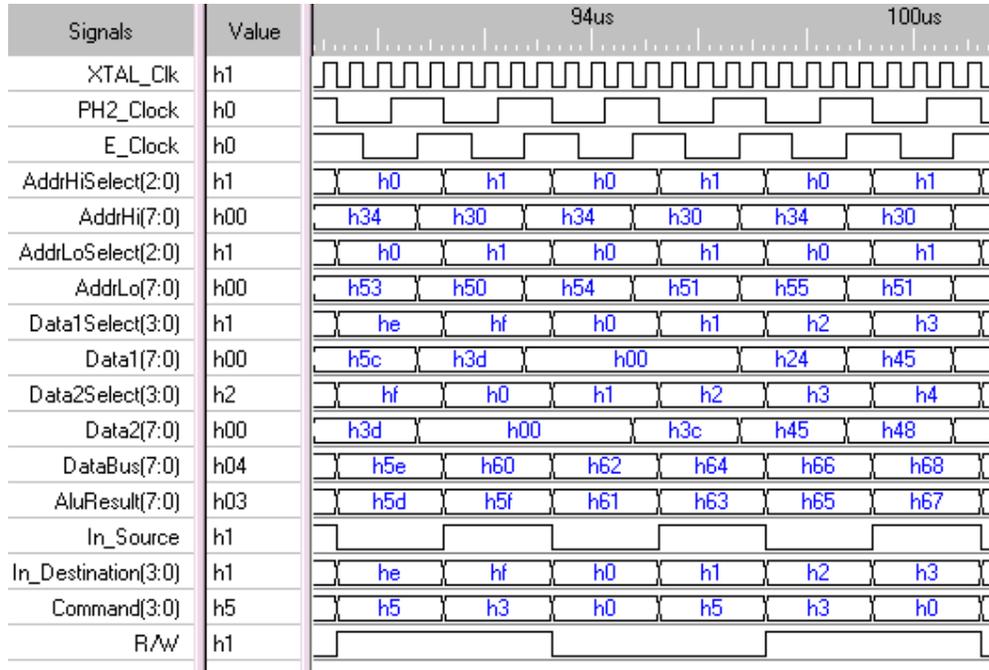


Figure 4.26: Register File Test Waveforms

```

Received : IncrementSP
AddrOutHI : PCH : 70 - AddrOutLO : PCL : 71
CCR0ut = 0x49, TempCCR0ut = 0x41
RegFile Write New Data :0x84, Destination: CCR
IX: 0x4845 - IY: 0x6C69
PC: 0x7071 - SP: 0x745A
  A: 0x7C - B: 0x79 - CCR: 0x84

Received : No Change in PC
AddrOutHI : SPH : 74 - AddrOutLO : SPL : 5A
CCR0ut = 0x02, TempCCR0ut = 0x42
RegFile Write New Data :0x85, Destination: IX Low
IX: 0x4885 - IY: 0x6C69
PC: 0x7071 - SP: 0x745A
  A: 0x7C - B: 0x79 - CCR: 0x02

Received : IncrementPC
AddrOutHI : PCH : 70 - AddrOutLO : PCL : 72
CCR0ut = 0x42, TempCCR0ut = 0x43
RegFile Write New Data :0x88, Destination: IX High
IX: 0x8885 - IY: 0x6C69
PC: 0x7072 - SP: 0x745A
  A: 0x7C - B: 0x79 - CCR: 0x42

Received : IncrementSP
AddrOutHI : SPH : 74 - AddrOutLO : SPL : 5B
CCR0ut = 0x46, TempCCR0ut = 0x44
RegFile Write New Data :0x89, Destination: IY Low
IX: 0x8885 - IY: 0x6C89
PC: 0x7072 - SP: 0x745B
  A: 0x7C - B: 0x79 - CCR: 0x46

Received : No Change in PC
AddrOutHI : PCH : 70 - AddrOutLO : PCL : 72
CCR0ut = 0x44, TempCCR0ut = 0x45
IX: 0x8885 - IY: 0x6C89
PC: 0x7072 - SP: 0x745B
  A: 0x7C - B: 0x79 - CCR: 0x44

Received : IncrementPC
AddrOutHI : SPH : 74 - AddrOutLO : SPL : 5B
CCR0ut = 0x46, TempCCR0ut = 0x46
IX: 0x8885 - IY: 0x6C89
PC: 0x7073 - SP: 0x745B
  A: 0x7C - B: 0x79 - CCR: 0x46

Received : IncrementSP
AddrOutHI : PCH : 70 - AddrOutLO : PCL : 73
CCR0ut = 0x47, TempCCR0ut = 0x47
IX: 0x8885 - IY: 0x6C89
PC: 0x7073 - SP: 0x745C
  A: 0x7C - B: 0x79 - CCR: 0x47

```

Figure 4.27: Register File Test Console Outputs

## 4.6 Address Bus Controller

Address bus controller module is designed to put correct address on address bus. This module takes select inputs from CPU controller unit and using these inputs, determines source of address information and asserts chip select signals to memory units. Source of address information can be outputs of register file, ALU result or data bus. Address bus controller also includes INIT and PPROG registers. “*InitWriteTimeout*” input is tied to CPU controller unit and informs this module that 64 cycles has passed after start of microcontroller operation, so INIT register is not writable anymore. This module is synchronous and sensitive to PH2 and E-clock signals. Address bus controller takes two inputs from register file, one input from arithmetic and logic unit and one input from data bus. Input and output ports of this module are given in Figure 4.28.

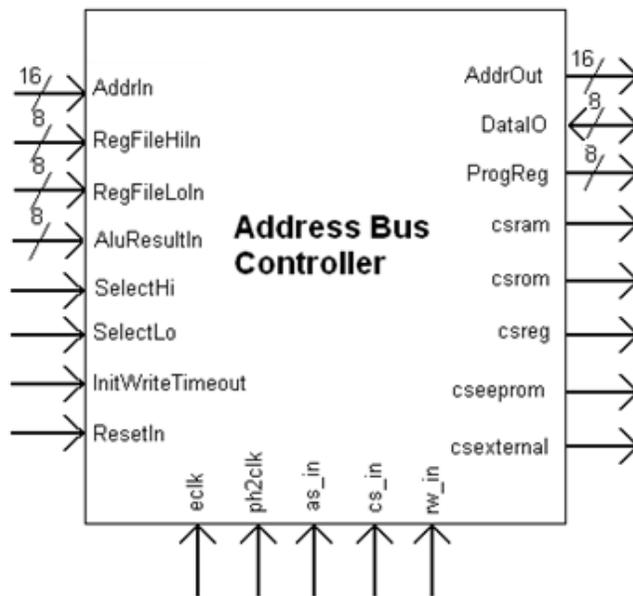


Figure 4.28: Address Bus Controller Symbol

For verification, an address bus test bench is designed that applies random signals to address bus controller unit. Address bus controller inputs are tied to “RegFile\_HI\_0”, “RegFile\_LO\_0”, “Alu\_Result\_1”, “Data\_Bus\_1” signals for verification. Two select inputs of module are tied to “SelectHI” and “SelectLO” signals. “Address\_Bus” signal is tied to output of the address bus controller module. When “SelectHI” signal is low, “RegFile\_HI\_0” signal; when “SelectHI” signal is high, “Alu\_Result\_1” signal is selected for address bus high order byte. Module selects “RegFile\_LO\_0” signal for low order byte of address bus when “SelectLO” signal is low and “Data\_Bus\_1” signal when “SelectLO” signal is high. Figure 4.29 and Figure 4.30 show test results which verify the module.

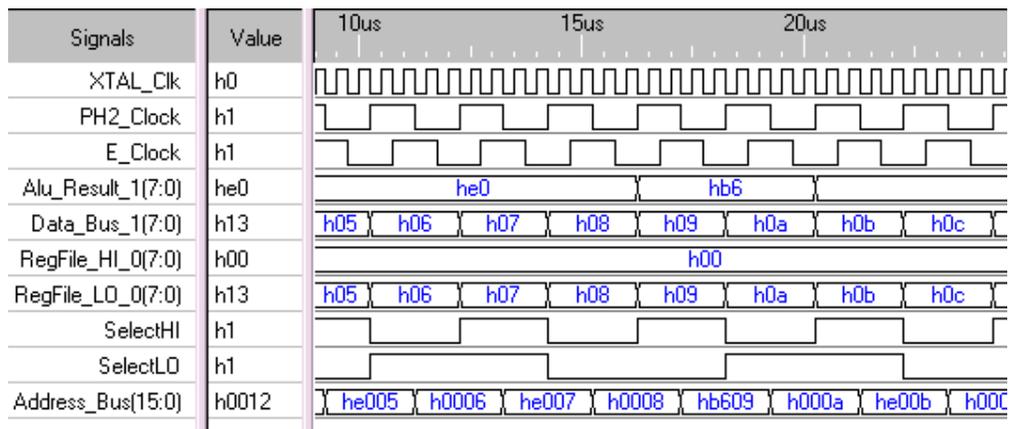


Figure 4.29: Address Bus Controller Test Results

```
Address Bus Controller Addr Out : 0xB653
Selected: EEPROM

Address Bus Controller Addr Out : 0x0054
Selected: RAM

Address Bus Controller Addr Out : 0xE055
Selected: ROM

Address Bus Controller Addr Out : 0x0056
Selected: RAM

Address Bus Controller Addr Out : 0xB657
Selected: EEPROM

Address Bus Controller Addr Out : 0x0058
Selected: RAM

Address Bus Controller Addr Out : 0xB659
Selected: EEPROM

Address Bus Controller Addr Out : 0x005A
Selected: RAM

Address Bus Controller Addr Out : 0xE05B
Selected: ROM

Address Bus Controller Addr Out : 0x005C
Selected: RAM

Address Bus Controller Addr Out : 0xB65D
Selected: EEPROM

Address Bus Controller Addr Out : 0x005E
Selected: RAM

Address Bus Controller Addr Out : 0xB65F
Selected: EEPROM

Address Bus Controller Addr Out : 0x0060
Selected: RAM

Address Bus Controller Addr Out : 0xB661
Selected: EEPROM

Address Bus Controller Addr Out : 0x0062
Selected: RAM

Address Bus Controller Addr Out : 0xE063
Selected: ROM
AddressBusController Test Ended
```

Figure 4.30: Console Outputs of Address Bus Controller Test

## 4.7 Handshake I/O Module

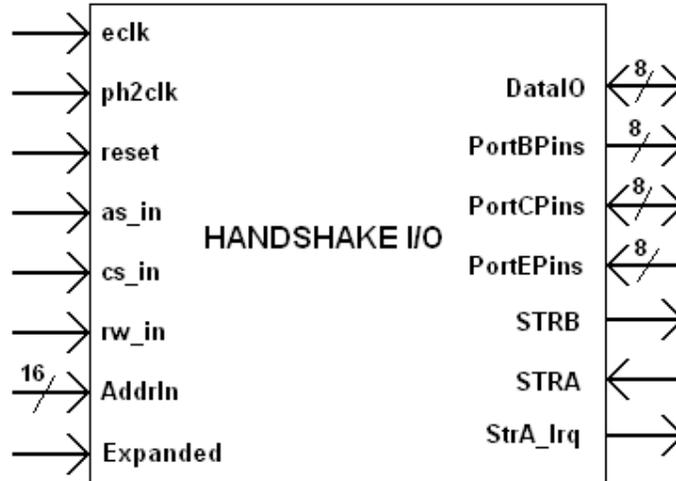


Figure 4.31: Handshake I/O Module Symbol

Handshake I/O module includes port B, port C, strobe A, strobe B and handshake I/O subsystem. Input and output ports of this module are presented in Figure 4.31. Expanded mode operations of ports B and C are also implemented in this module. Data on port C pins is latched into PORTCL register when a selected edge is detected at strobe A (STRA) input and strobe B signal is negated at upcoming PH2 clock positive edge if full-input handshake mode is selected. Strobe A flag (STAF) and strobe B signals are synchronized with internal PH2 clock positive edge as it is in original MC68HC11. All modes of handshake I/O subsystems are implemented in this module. These modes are simple strobe handshake mode, full-input handshake mode, normal and three-state variations of full-output

handshake mode. Verification of this module is done by using simulation platform that is developed as a goal of this thesis.

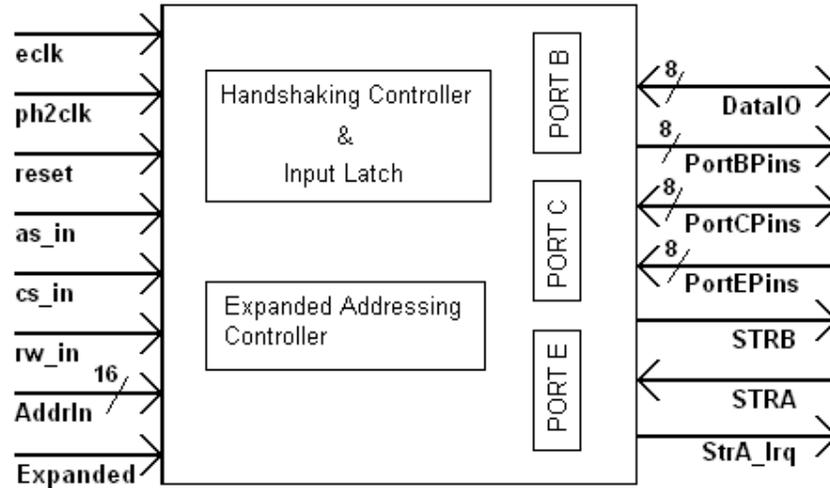


Figure 4.32: Handshake I/O Module Block Diagram

#### 4.8 Timer System

Timer system module of MC68HC11 MCU model includes main timer, pulse accumulator and Port A. These three peripherals are all contained in one module because of their interactions with each other. These peripherals share some internal registers; main timer and pulse accumulator systems share port A pins. Real-time interrupt, computer operating properly (COP), input capture and output compare functions of main timer system are all implemented in the timer system module of MC68HC11 SystemC model. This module contains largest number of internal registers among the other modules. Input and output ports of timer system module are presented in Figure 4.33.

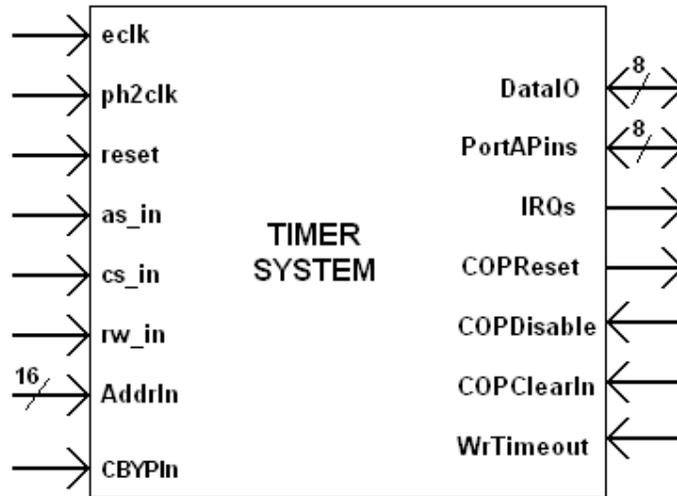


Figure 4.33: Timer System Symbol

Main timer clock is generated using E-clock signal. Free running counter is sensitive to this main timer clock. Also pulse accumulator clock, real time interrupt function clock and COP function clock are generated using main timer clock. Output compare functions compares free running counter with values loaded into interested registers and sets corresponding port A pin is these two values become equal to each other. Input capture functions save value of free running counter into corresponding registers when selected edges are detected at corresponding pins of port A.

There are eleven interrupt sources in this module. Ten of them are main timer system related interrupt sources. There are five interrupt sources from output-compare function, three interrupt sources from input-compare functions, one interrupt source is free running timer overflow and one is real time interrupt function. Pulse accumulator system requests interrupt when its internal counter overflows.

Pulse accumulator system operates in two different modes. These modes are, time accumulation and event-counting modes. Time accumulation function of this module counts selected edges of pulse accumulator clock and sets overflow flag and requests interrupt, if enabled, when internal pulse accumulator counter overflows. Event counting function operates similar to time accumulation function with a difference that it counts selected edges on PAI pin of port A. Port A functions are also included in this module including sharing pin-7, forcing of outputs in three state variation of full-output handshake mode. Verification of this module is done in developed simulation platform. Block diagram of timer system is presented in Figure 4.34. Timer system is tested and verified using examples from M68HC11 Referece Manual.

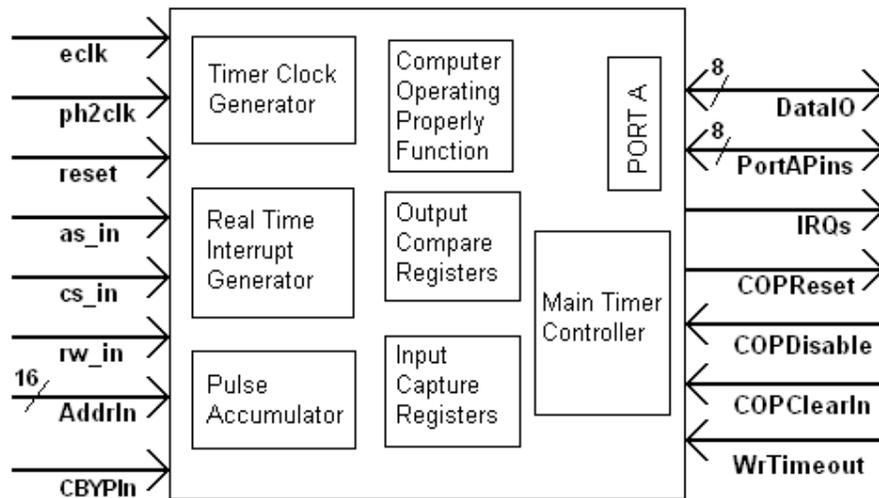


Figure 4.34: Timer System Block Diagram

#### 4.9 Serial Communications Module

Serial communications module contains asynchronous serial communications interface (SCI), synchronous serial peripheral interface (SPI) and port D. These three peripherals are brought together in this module because they share some registers and pins with each other. Port D is a 6-bit bidirectional general purpose I/O port. Four pins of port D are shared between general purpose I/O functions and SPI system and other two pins are shared between general purpose I/O functions and SCI system. Input and output ports of this module are given in Figure 4.35.

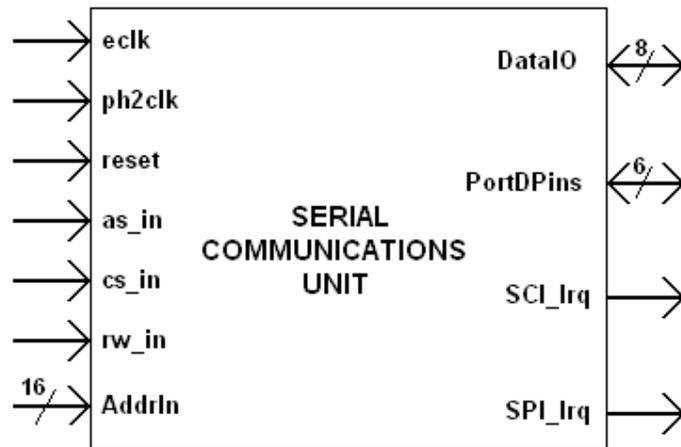


Figure 4.35: Serial Communications Module Symbol

There exists a clock divider process in the module which divides ph2 clock and generates baud rate and SPI clock signals. Receiver and transmitter processes for SCI are also included in serial communications module. SCI receiver is implemented compatible with 16x data sampling technique to reduce reception errors. Master and slave mode operations of SPI system are implemented in two

separate processes because they are sensitive to different signals. Block diagram of serial communications module is presented in Figure 4.36. Serial transmitter and receiver are tested and verified using the serial port test program given in Appendix B.

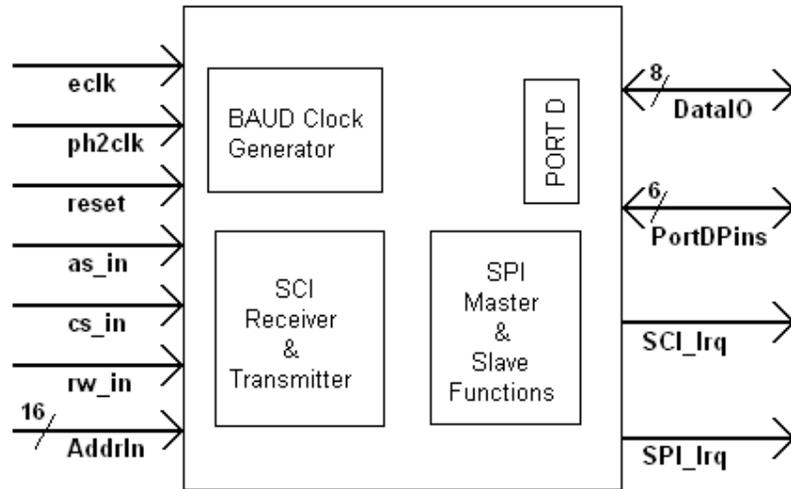


Figure 4.36: Serial Communications Module Block Diagram

#### 4.10 Read Only Memory (ROM)

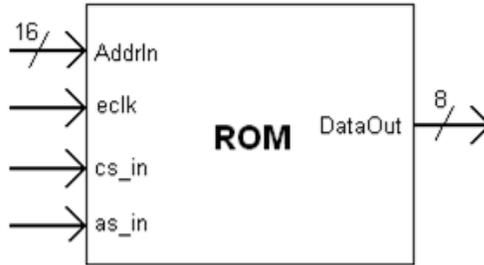


Figure 4.37: ROM Symbol

This module is designed as read only memory (ROM) of microcontroller unit model. Size of ROM is 12 Kbytes. ROM is synchronous, it latches address information using address strobe input (as\_in) and outputs addressed data at positive edge of E-clock if chip select input is at its active level. This module contains a process to read ROM contents from a data file for debug and simulation purposes. Symbol of ROM module is presented in Figure 4.37.

Verification of ROM module is done by first loading data into ROM memory using a file and then reading all locations of ROM and comparing values read with original values. Figure 4.38 shows console output of test process.

```

ROM: Address:0x694 Data: 0xc0
Original value : 0xc0
ROM: Address:0x695 Data: 0xc5
Original value : 0xc5
ROM: Address:0x696 Data: 0xec
Original value : 0xec
ROM: Address:0x697 Data: 0x2
Original value : 0x2
ROM: Address:0x698 Data: 0x94
Original value : 0x94
ROM: Address:0x699 Data: 0x12
Original value : 0x12
ROM: Address:0x69a Data: 0xec
Original value : 0xec
ROM: Address:0x69b Data: 0x94
Original value : 0x94
ROM: Address:0x69c Data: 0x69
Original value : 0x69
ROM: Address:0x69d Data: 0x30
Original value : 0x30
ROM: Address:0x69e Data: 0xf6
Original value : 0xf6
ROM: Address:0x69f Data: 0x83
Original value : 0x83
ROM: Address:0x6a0 Data: 0xaf
Original value : 0xaf
Read Count : 100000 Error Count: 0
ROM Test Ended

```

Figure 4.38: Console Output of ROM Test

Figure 4.38 shows values read from address locations of ROM and original values that were loaded to ROM. Consistency between these values verifies ROM module. Some part of resulting waveforms of test is presented in Figure 4.39.

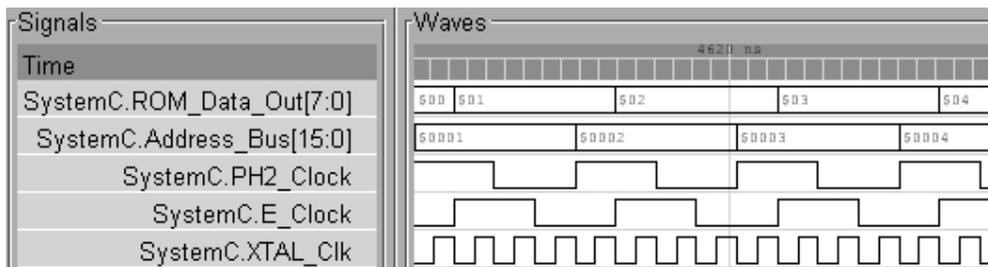


Figure 4.39: Resulting Waveforms of ROM Test

#### 4.11 Random Access Memory (RAM)

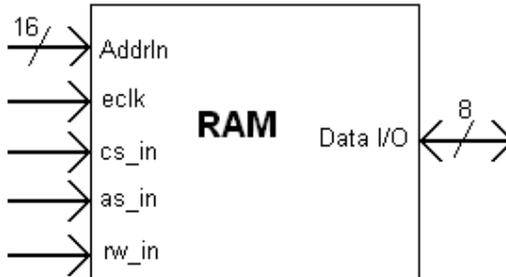


Figure 4.40: RAM Symbol

RAM module is an implementation of 512 bytes random access memory. This module is synchronous to E-clock. Address information is latched using address strobe input (as\_in). Data from data I/O port is written into addressed register at negative edge of E-clock if read / write (R/W) input (wr\_in) is at its “write” level and chip select input (cs\_in) is at its active level. Addressed data is put onto data I/O port if chip select input is at its active level and R/W input is at its “read” level. Data I/O port is at high impedance if this module is not selected.

Verification of RAM module is done by writing random data on random address locations and reading back the written data. Data read from RAM is compared to data written to RAM at previous cycle. During the test, one million random write and read operations are performed. According to test results, all data written to RAM was read correctly.

A sample part of waveforms of signals during the verification process of RAM module are presented in Figure 4.41. Figure 4.42 shows last part of the test results. Data written to RAM and read from RAM during the last portion of test and total count of read/write operations and total count of errors are also given in Figure 4.42. Test results verify RAM module.

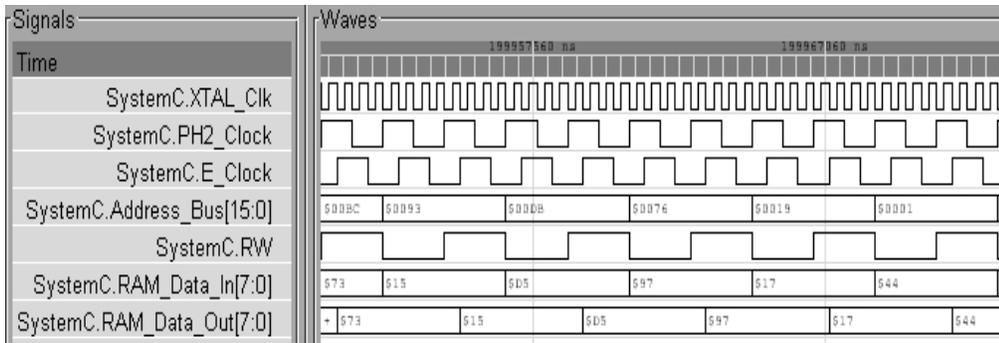


Figure 4.41: Waveforms of RAM Test Results

```

RAM: WRITE Address: 0xde Data: 0x9f
RAM: READ Address: 0xde, Data: 0x9f
RAM: WRITE Address: 0xbc Data: 0x73
RAM: READ Address: 0xbc, Data: 0x73
RAM: WRITE Address: 0x93 Data: 0x15
RAM: READ Address: 0x93, Data: 0x15
RAM: WRITE Address: 0xdb Data: 0xd5
RAM: READ Address: 0xdb, Data: 0xd5
RAM: WRITE Address: 0x76 Data: 0x97
RAM: READ Address: 0x76, Data: 0x97
RAM: WRITE Address: 0x19 Data: 0x17
RAM: READ Address: 0x19, Data: 0x17
RAM: WRITE Address: 0x1 Data: 0x44
RAM: READ Address: 0x1, Data: 0x44
RAM: WRITE Address: 0x63 Data: 0x80
RAM: READ Address: 0x63, Data: 0x80
RAM: WRITE Address: 0x2e Data: 0xc1
RAM: READ Address: 0x2e, Data: 0xc1
RAM: WRITE Address: 0x20 Data: 0xed
RAM: READ Address: 0x20, Data: 0xed
RAM: WRITE Address: 0x3a Data: 0xe7
RAM: READ Address: 0x3a, Data: 0xe7
RAM: WRITE Address: 0x14 Data: 0x21
RAM: READ Address: 0x14, Data: 0x21
RAM: WRITE Address: 0xa9 Data: 0xb8
RAM: READ Address: 0xa9, Data: 0xb8
RAM: WRITE Address: 0x85 Data: 0x5d
RAM: READ Address: 0x85, Data: 0x5d
Read/Write Count : 100000 Error Count: 0
RAM Test Ended

```

Figure 4.42: RAM Test Results

#### 4.12 Electrically Erasable Programmable ROM (EEPROM)

EEPROM module is an implementation of 512 bytes of EEPROM. Reading from an EEPROM location is same as reading from ROM or RAM, but writing to an EEPROM location or erasing a location is different. In order to write data to an address in EEPROM, EEPGM bit of PPROG register should be set first. This enables EEPROM programming voltage. Write operation should be done following this action and finally EEPROM programming voltage should be disabled again by clearing EEPGM bit. Three different erase operations are available for EEPROM locations. These are row erase, byte erase and bulk erase operations which erase a row consisting of two bytes, a single byte and whole device respectively. EEPROM module of MC68HC11 SystemC module uses a file for holding its contents. Figure 4.43 presents port information of EEPROM module.

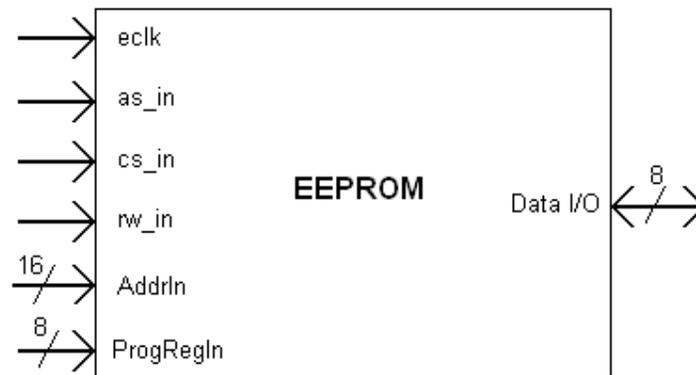


Figure 4.43: EEPROM Symbol

EEPROM module is verified by a two-step test. First step was filling EEPROM locations with random values, than reading these values and comparing with originally written values. Second step was clearing EEPROM locations and than reading these locations and checking if they were cleared correctly. Test results successfully verified EEPROM module. Figures 4.44 and 4.45 shows test results of first and second step of test respectively.

```
EEPROM: Address:0x1f4 Data: 0xf0
Original value : 0xf0 Read value: 0xf0

EEPROM: Address:0x1f5 Data: 0x44
Original value : 0x44 Read value: 0x44

EEPROM: Address:0x1f6 Data: 0x3a
Original value : 0x3a Read value: 0x3a

EEPROM: Address:0x1f7 Data: 0xb4
Original value : 0xb4 Read value: 0xb4

EEPROM: Address:0x1f8 Data: 0xa6
Original value : 0xa6 Read value: 0xa6

EEPROM: Address:0x1f9 Data: 0x66
Original value : 0x66 Read value: 0x66

EEPROM: Address:0x1fa Data: 0x53
Original value : 0x53 Read value: 0x53

EEPROM: Address:0x1fb Data: 0x33
Original value : 0x33 Read value: 0x33

EEPROM: Address:0x1fc Data: 0xb
Original value : 0xb Read value: 0xb

EEPROM: Address:0x1fd Data: 0xcb
Original value : 0xcb Read value: 0xcb

EEPROM: Address:0x1fe Data: 0xa1
Original value : 0xa1 Read value: 0xa1

EEPROM: Address:0x1ff Data: 0x10
Original value : 0x10 Read value: 0x10

Error Count: 0

***NEW TEST STATE = WILL NOW CLEAR EEPROM !
```

Figure 4.44: EEPROM Write / Read Test Results.

```
EEPROM: Address:0x1f9 Data: 0
Original value : 0 Read value: 0

EEPROM: Address:0x1fa Data: 0
Original value : 0 Read value: 0

EEPROM: Address:0x1fb Data: 0
Original value : 0 Read value: 0

EEPROM: Address:0x1fc Data: 0
Original value : 0 Read value: 0

EEPROM: Address:0x1fd Data: 0
Original value : 0 Read value: 0

EEPROM: Address:0x1fe Data: 0
Original value : 0 Read value: 0

EEPROM: Address:0x1ff Data: 0
Original value : 0 Read value: 0

Error Count: 0

***EEPROM TEST FINISHED
```

Figure 4.45: EEPROM Clear / Read Test Results.

### 4.13 VLSI Implementation of SystemC Modules

VLSI Implementation of the developed SystemC modules is possible using SystemC to HDL conversion tools. These conversion tools are commercial applications and trial versions of these tools have limitations. Only ALU module of the microcontroller SystemC module could be converted into VHDL using the SystemCrafter tool due to the compiler limitations of the tool. VLSI implementation of developed overall SystemC model is left as a future work. After VHDL synthesis of SystemC descriptions using SystemCrafter, VHDL outputs of SystemCrafter is synthesized for Spartan 3 XC3S50 FPGA using Xilinx ISE 9.2i tool. The results of the synthesis process are given in Table 4.3.

Table 4.3: Synthesis Results of ALU Module

| <b>Device Utilization Summary</b>                    |             |                  |                    |
|--|-------------|------------------|--------------------|
| <b><i>Logic Utilization</i></b>                      | <b>Used</b> | <b>Available</b> | <b>Utilization</b> |
| Number of Slice Flip Flops                           | 71          | 1536             | 4 %                |
| Number of 4 input LUTs                               | 907         | 1536             | 59 %               |
| <b><i>Logic Distribution</i></b>                     |             |                  |                    |
| Number of occupied Slices                            | 496         | 768              | 64 %               |
| Number of Slices containing only related logic       | 496         | 496              | 100 %              |
| Number of Slices containing unrelated logic          | 0           | 496              | 0 %                |
| <b><i>Total Number of 4 input LUTs</i></b>           | 909         | 1536             | 59 %               |
| Number used as logic                                 | 907         |                  |                    |
| Number used as route-thru                            | 2           |                  |                    |
| Number of bonded IOBs                                | 40          | 97               | 41 %               |
| IOB Flip Flops                                       | 28          |                  |                    |
| Number of GCLKs                                      | 1           | 8                | 12 %               |
| <b><i>Total equivalent gate count for design</i></b> | 6747        |                  |                    |
| Additional JTAG gate count for IOBs                  | 1920        |                  |                    |

## CHAPTER 5

### VISUAL SIMULATION PLATFORM

In this thesis study, a visual simulation platform is developed using co-design capabilities and object oriented nature of SystemC. Visual simulation platform consists of SystemC model of MC68HC11 microcontroller unit, its peripheral devices, test bench modules and user friendly visual simulation software that is designed for controlling simulations by configuring test bench, starting simulations and observing simulation results. Structures of the developed visual simulation platform, test bench modules and finally visual simulation software are explained in this chapter.

#### 5.1 Structure of Visual Simulation Platform

Visual simulation platform consists of three main parts. These are MC68HC11 SystemC model with its peripherals, test bench and visual simulation software which are presented in Figure 5.1. Test bench contains SystemC models of test hardware. Visual simulation software outputs test bench configuration files that are used to configure which test module is connected to which port and whether it is enabled or not and to configure operations of test hardware models in test bench. Visual simulation software also outputs microcontroller program file that is stored into ROM of microcontroller unit. Test bench is connected to MC68HC11 via input and output ports and applies signals to and reads signals from MC68HC11 according to its configuration. After simulation ends, test bench module outputs a simulation results file that contains information on ports of microcontroller unit and

internals of test modules for each cycle of simulation process. MC68HC11 model outputs simulation result files that contain information on internal working of microcontroller unit such as state of microcontroller, address and data bus signals, register values and RAM content at each cycle.

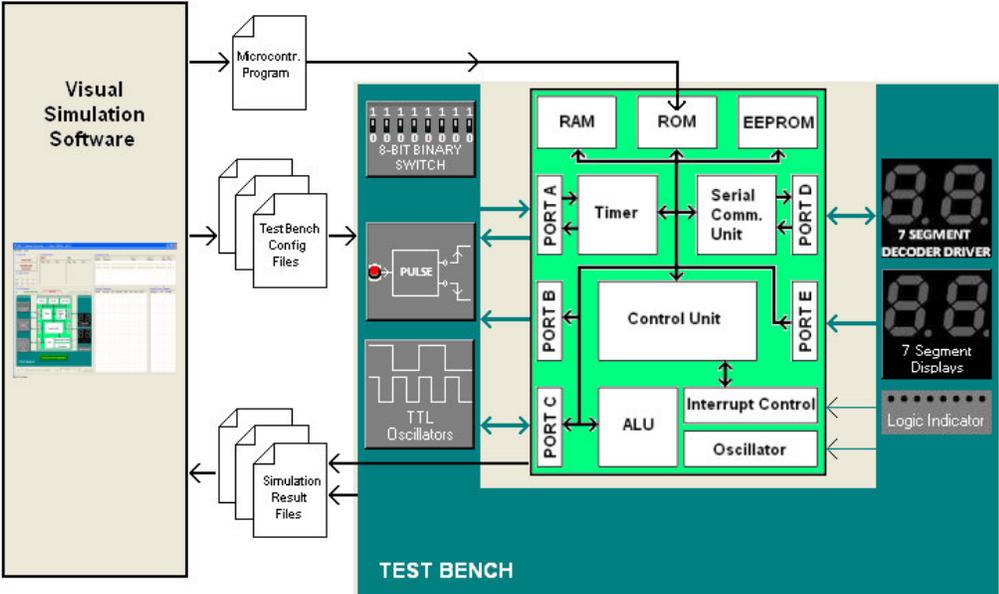


Figure 5.1: Visual Simulation Platform

**5.2 Test Bench**

Test bench consists of SystemC models of test hardware that are available in most electronic experiment kits. These hardware modules are designed at a much higher abstraction level than microcontroller unit. This is a powerful ability of SystemC, because designing test hardware in a higher abstraction level than tested system saves great time of system designers. Test bench is configurable by

a configuration file. According to the configuration file, some hardware modules are enabled or disabled and they are connected to different I/O ports. Test hardware contained in test bench are an 8-bit binary switch, a push button pulse generator, some TTL oscillators, seven segment BCD decoder / driver, seven segment common anode / common cathode displays, logic indicator and serial monitor.

8-bit binary switch is an output module that takes a configuration file and outputs specified binary data value at specified times by configuration file. Push button pulse generator outputs a pulse at specified time using push button state and timing information on a configuration file. It generates both negative edge and a positive edge pulse at the same time. TTL Oscillators module contains four oscillators. It has an output for each oscillator. These TTL oscillators have oscillation frequencies of 100 KHz, 10 KHz, 1 KHz and 100 Hz. Seven segment BCD decoder / driver module is an input module that decodes BCD value on its inputs and converts it to a form applicable to a seven segment display. Seven segment display module reads its inputs and outputs states of seven segment displays which can be configured as common anode or common cathode displays. Logic indicator module is a simple module that just saves binary data on its inputs and time information of value changes on this data. Serial monitor has serial transmitter and receiver and it outputs serial data that is read from a file and shows the data that is received serially.

### **5.3 Features of Visual Simulation Software**

Visual simulation software is the end-user face of the simulation platform. The user interface is designed in a way that it guides user from first to last step of a simulation process. First step of simulation process is to write assembly codes for microcontroller unit in code editor window. Then, the written code is compiled to Motorola s-record (.s19) file and this s-record file is converted to machine code (.hex file) that will be placed in ROM. After machine code file is generated, this file

is downloaded to microcontroller ROM. At this point program waits for user to finish configuration of test bench. After test bench configuration is finished, user can select simulation duration and start a simulation. When the simulation ends, simulation results are presented on main screen of program. User can examine simulation results that consist of test hardware information and information on microcontroller internal workings. Details on program and using graphical user interface are explained in following subsections.

### **5.3.1 Main Window of Simulation Software**

Main window of visual simulation software is presented in Figure 5.2. Regions in main window are labeled with capital letters in the figure. Region which is labeled with 'A' is program file region. User can write assembly code and convert written code to machine code using "Code Editor" and "Machine Code Generator" buttons respectively. 'B' region contains condition codes register information. Condition code register information is update at every clock cycle. Region labeled with 'C' shows contents of registers in register file before and after the selected cycle or instruction. Region 'D' is test environment region. In this region user can download machine code into microcontrollers ROM, configure test bench and test hardware, select simulation duration, run simulation and view simulation results using buttons. After a simulation ends, executed instructions are presented as a list in region 'E'. Region 'F' shows information on internal cycles of instruction execution. This region is filled when user selects an executed instruction. Region 'G' shows values of special function registers for select clock cycle.

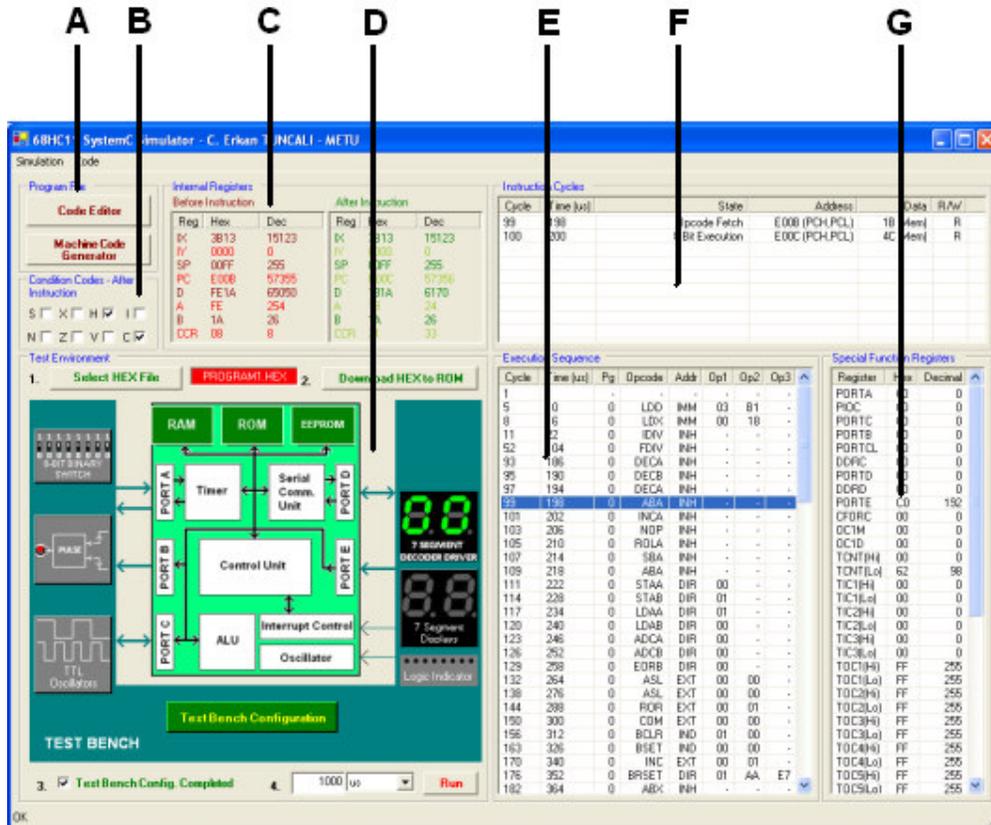


Figure 5.2: Main Window of Visual Simulation Software

Main window of simulation software is arranged in a way that it guides user through simulation process. Buttons are located in an order and numbered for simulation steps and they are not enabled until next step of simulation process is using that button.

### 5.3.2 Code Editor

Visual simulation tool has a built-in code editor that works like a simple text editor program. User can write or modify 68HC11 assembly programs using this editor and save them in “.asm” format. A screenshot from code editor window is shown in Figure 5.3.

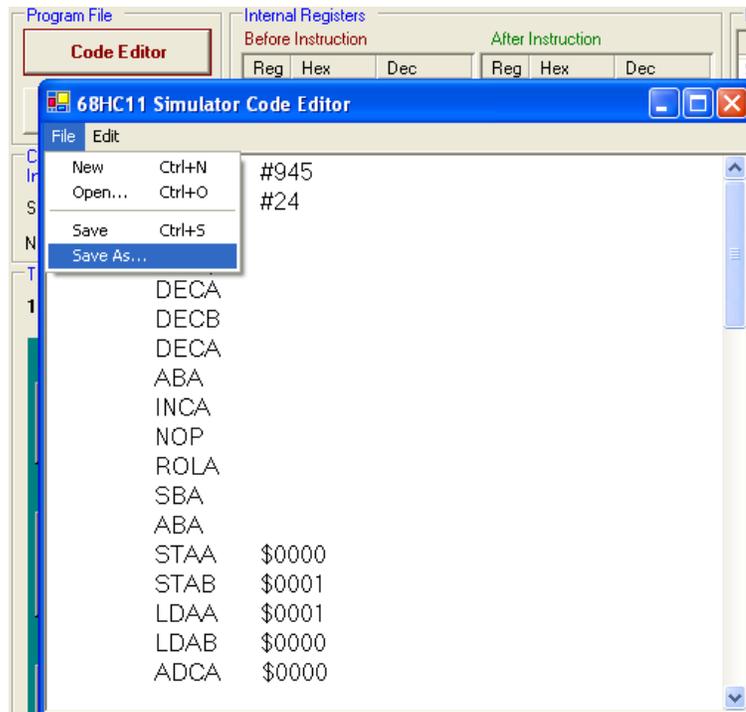


Figure 5.3: 68HC11 Assembly Code Editor Window

### 5.3.3 Machine Code Generator

User can compile programs to generate MC68HC11 machine codes. For machine code generation, visual simulation program uses Motorola's assembler program "68HC11AS11.exe" and "hex to bin converter v.2.00" from "Tech Edge Pty. Ltd." which are available for free download on the internet. These conversion steps are done manually by user for better understanding of machine code generation steps starting from an assembly code file (".asm" file). Figure 5.4 shows a screenshot from machine code generator window.

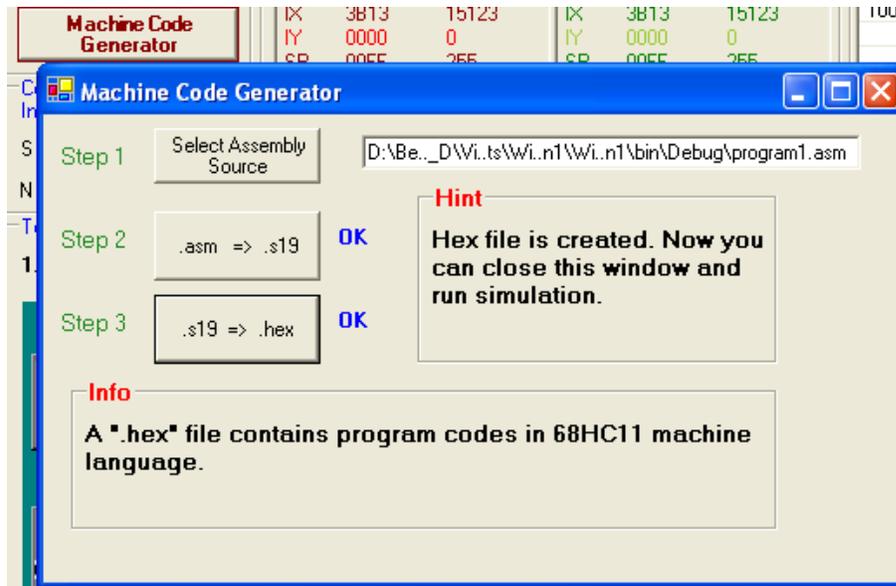


Figure 5.4: Machine Code Generator Window

At first step, user selects the “.asm” file which was written in code editor or in any other editor. Then user converts this file to Motorola s-record file (“.s19” file) format using a button labeled “.asm => .s19”. At this moment, simulator runs Motorola’s 68HC11AS11ASM compiler at background. After completing “.asm” to “.s19” conversion, conversion from “.s19” to machine code file (“.hex” file) format is done using “.s19 => .asm” button. Simulator runs “hex to bin converter v.2.00” from “Tech Edge Pty. Ltd.” for this process. This step gives machine code file that can be downloaded to microcontroller’s ROM for simulation purposes. In the code generator window, “Hint” space gives guides user for steps of machine code generation and “Info” space gives simple information on the file formats that are generated on each step.

#### **5.3.4 Preparing Simulation Environment and Running Simulation**

In the main window of visual simulation software, test environment region is used for downloading machine code file into microcontroller’s ROM, test bench configuration and running simulation. Figure 5.5 shows a screenshot from this region. Everything needed by user for preparing and running a simulation using a machine code file is available in this region.

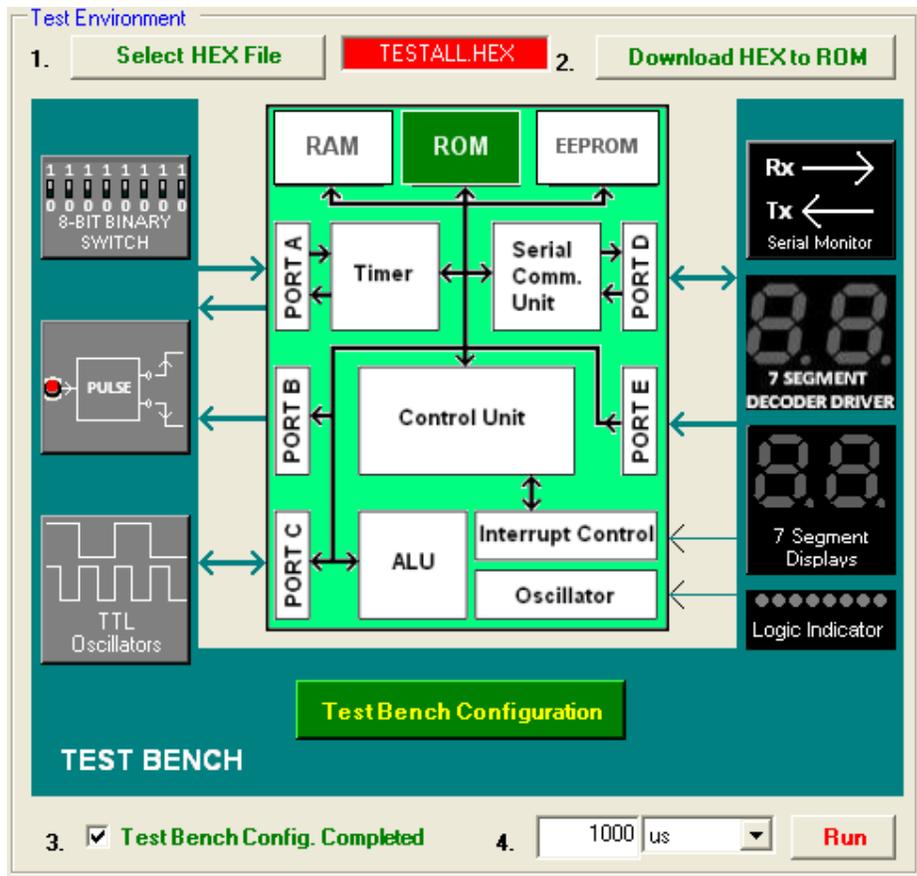


Figure 5.5: Test Environment Region of Simulation Program

When machine code file is ready, it can be downloaded to microcontroller ROM using “Select .HEX File” and “Download .HEX to ROM” buttons that are shown in Figure 5.5. When a machine code file is selected, file name is updated on main screen and ROM button is enabled when the selected machine code file is put into microcontroller’s ROM. Viewing ROM locations is explained in following subsections.

After user program is put into ROM, test bench configuration should be done by user. User should configure hardware modules of test bench modules that are needed in simulation.

Configuration of 8-bit binary switch operation can be done by using its button on test bench which can be seen in Figure 5.5. When user clicks on this button, a new window opens as shown in Figure 5.6. After “Start Configuration” button is pressed, user can generate a sequence of binary data that will be applied at different times. Switch conditions are selected, timing of the switch configuration is selected and it is added by using “Add” button. When configuration ends, it should be saved to take effect. Any row can be removed using “Remove” button.

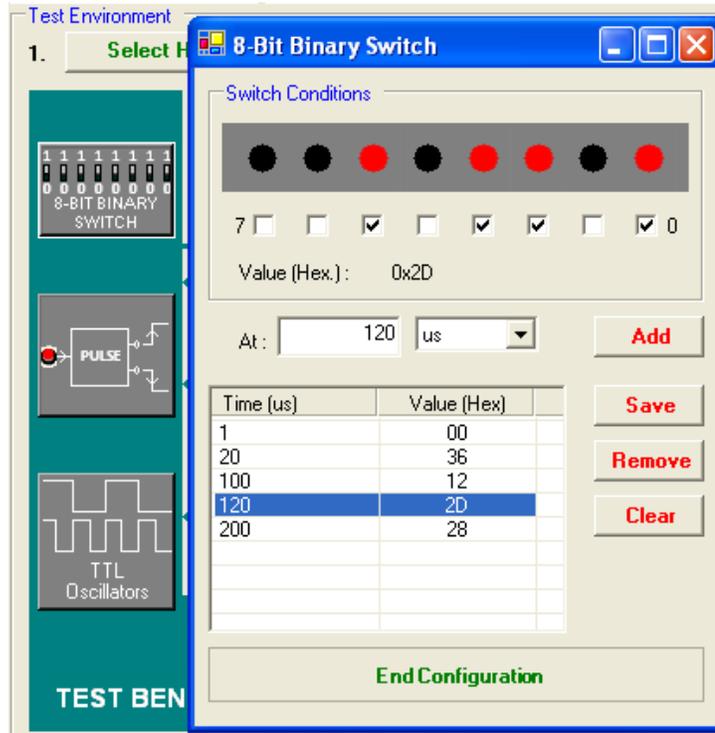


Figure 5.6: 8-bit Binary Switch Configuration

Configurations of push button pulse generator and TTL oscillators are very similar to configuration of 8-bit binary switch. Only difference is push button has only one bit released or pressed states and TTL oscillators have one bit power on and power off states instead of 8-bit information. Window used for push button pulse generator configuration is shown in Figure 5.7.

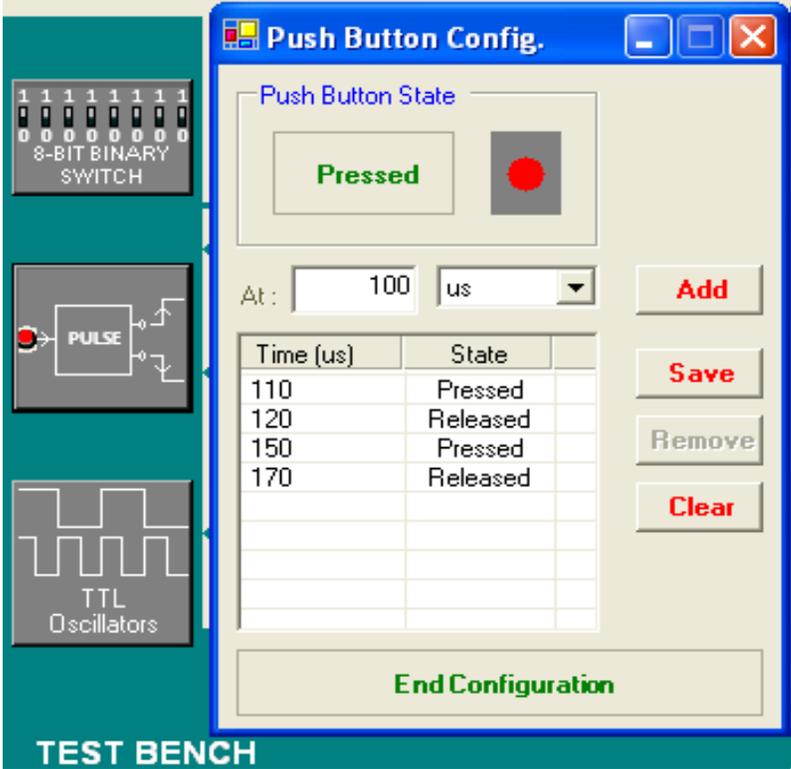


Figure 5.7: Push Button Pulse Generator Configuration

Configuration of serial monitor can be done by using “Serial Monitor” button. Figure 5.8 shows a screenshot from serial monitor configuration window. User can add any text that will be sent at any desired time.

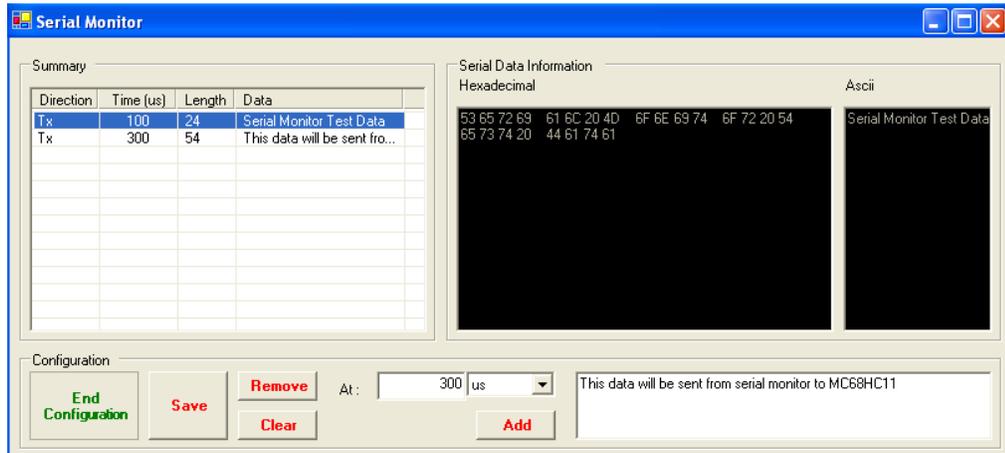


Figure 5.8: Serial Monitor Configuration

Test bench port connections can be configured using “Test Bench Configuration” button in test environment. This button opens a new window for directing test hardware input / output pins to desired port pins of microcontroller. User can also enable or disable test hardware modules. Figure 5.9 shows test bench port configuration window. User should first press “Start Configuration” button, make necessary changes and save configuration before ending simulation for configuration to take effect.

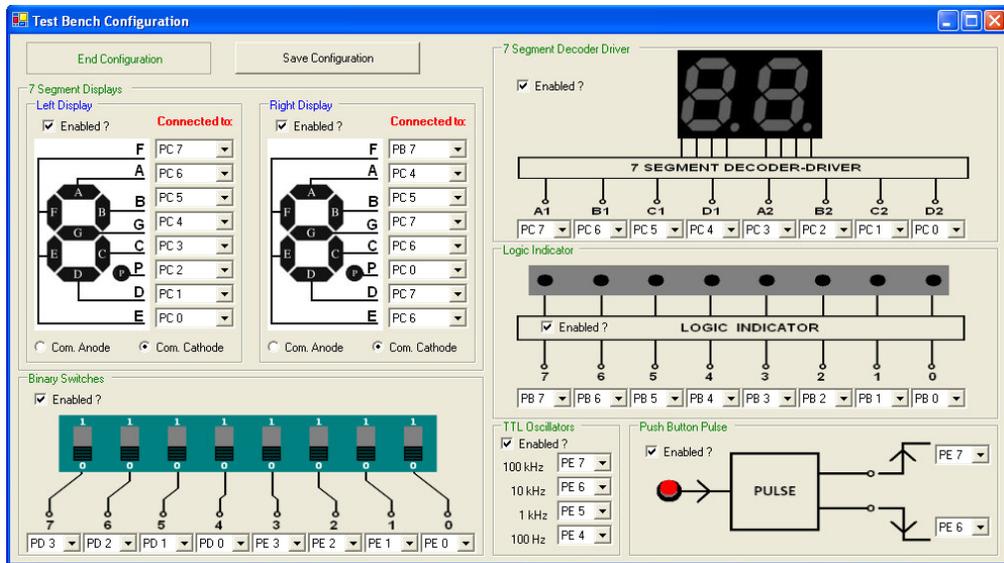


Figure 5.9: Test Bench Configuration Window

When configuration of test bench is finished, user should check “Test Bench Config. Complete” box, select simulation duration and press “Run” button. Program waits confirmation before running simulation. Figure 5.10 shows these steps. Visual simulation program runs executable file of MC68HC11 SystemC model. This is a flexibility of using SystemC for co-design. SystemC simulations can be performed by only executing a program file.

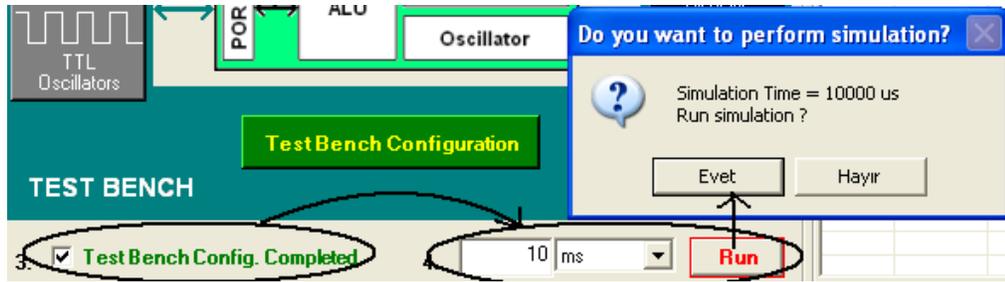


Figure 5.10: Running Simulation

### 5.3.5 Viewing Simulation Results

When the simulation ends, program automatically fills execution sequence table. This table gives information on instructions in a sequence. Execution time of instruction, opcode, operands, addressing mode and opcode page of instruction can be seen on this table. Op1, Op2 and Op3 columns present operands. If number of operands is smaller than three a dash (“-”) is put into empty cells. A screenshot of execution sequence table is given in Figure 5.11. When an instruction is selected, internal cycles execution information of that instruction is given in instruction cycles table as shown in Figure 5.11. This information consists of time, microcontroller state, address bus, data bus and read/write signal.

Because, this visual simulator actually runs a SystemC simulation, it can access available information on internal workings of microcontroller unit. This is a very important feature of this simulator which makes it different from standard simulators that are available in the market or in the internet for free download. This feature allows better understanding of internal workings of microcontroller. Values of special function registers are presented for selected cycle when an instruction or instruction cycle information is selected. Values of internal register values and condition code register information are presented as in Figure 5.12.

| Instruction Cycles |           |                           |                   |          |     |  |
|--------------------|-----------|---------------------------|-------------------|----------|-----|--|
| Cycle              | Time (us) | State                     | Address           | Data     | R/W |  |
| 176                | 352       | Opcode Fetch              | E034 (PCH,PCL)    | 12 (Mem) | R   |  |
| 177                | 354       | Read Direct Address       | E035 (PCH,PCL)    | 01 (Mem) | R   |  |
| 178                | 356       | Read Execution Operand    | 0001 ( -_DataBus) | 0E (Mem) | R   |  |
| 179                | 358       | 8 Bit Execution           | E036 (PCH,PCL)    | AA (Mem) | R   |  |
| 180                | 360       | Calculate Rel. Addr. Low  | E037 (PCH,PCL)    | E7 (Mem) | R   |  |
| 181                | 362       | Calculate Rel. Addr. High | FFFF (FF,FF)      | 00 (Mem) | R   |  |

| Execution Sequence |           |    |        |      |     |     |     |
|--------------------|-----------|----|--------|------|-----|-----|-----|
| Cycle              | Time (us) | Pg | Opcode | Addr | Op1 | Op2 | Op3 |
| 1                  | -         | -  | -      | -    | -   | -   | -   |
| 5                  | 10        | 0  | LDD    | IMM  | 03  | B1  | -   |
| 8                  | 16        | 0  | LDX    | IMM  | 00  | 18  | -   |
| 11                 | 22        | 0  | IDIV   | INH  | -   | -   | -   |
| 52                 | 104       | 0  | FDIV   | INH  | -   | -   | -   |
| 93                 | 186       | 0  | DECA   | INH  | -   | -   | -   |
| 95                 | 190       | 0  | DECB   | INH  | -   | -   | -   |
| 97                 | 194       | 0  | DECA   | INH  | -   | -   | -   |
| 99                 | 198       | 0  | ABA    | INH  | -   | -   | -   |
| 101                | 202       | 0  | INCA   | INH  | -   | -   | -   |
| 103                | 206       | 0  | NOP    | INH  | -   | -   | -   |
| 105                | 210       | 0  | ROLA   | INH  | -   | -   | -   |
| 107                | 214       | 0  | SBA    | INH  | -   | -   | -   |
| 109                | 218       | 0  | ABA    | INH  | -   | -   | -   |
| 111                | 222       | 0  | STAA   | DIR  | 00  | -   | -   |
| 114                | 228       | 0  | STAB   | DIR  | 01  | -   | -   |
| 117                | 234       | 0  | LDAA   | DIR  | 01  | -   | -   |
| 120                | 240       | 0  | LDAB   | DIR  | 00  | -   | -   |
| 123                | 246       | 0  | ADCA   | DIR  | 00  | -   | -   |
| 126                | 252       | 0  | ADCB   | DIR  | 00  | -   | -   |
| 129                | 258       | 0  | EORB   | DIR  | 00  | -   | -   |
| 132                | 264       | 0  | ASL    | EXT  | 00  | 00  | -   |
| 138                | 276       | 0  | ASL    | EXT  | 00  | 00  | -   |
| 144                | 288       | 0  | ROR    | EXT  | 00  | 01  | -   |
| 150                | 300       | 0  | COM    | EXT  | 00  | 00  | -   |
| 156                | 312       | 0  | BCLR   | IND  | 01  | 00  | -   |
| 163                | 326       | 0  | BSET   | IND  | 00  | 00  | -   |
| 170                | 340       | 0  | INC    | EXT  | 00  | 01  | -   |
| 176                | 352       | 0  | BRSET  | DIR  | 01  | AA  | E7  |
| 182                | 364       | 0  | ABX    | INH  | -   | -   | -   |

| Special Function Registers |     |         |
|----------------------------|-----|---------|
| Register                   | Hex | Decimal |
| PORTA                      | 00  | 0       |
| PIOC                       | 00  | 0       |
| PORTC                      | 00  | 0       |
| PORTB                      | 00  | 0       |
| PORTCL                     | 00  | 0       |
| DDRC                       | 00  | 0       |
| PORTD                      | 00  | 0       |
| DDRD                       | 00  | 0       |
| PORTE                      | C0  | 192     |
| CFORC                      | 00  | 0       |
| OC1M                       | 00  | 0       |
| OC1D                       | 00  | 0       |
| TCNT(Hi)                   | 00  | 0       |
| TCNT(Lo)                   | B3  | 179     |
| TIC1(Hi)                   | 00  | 0       |
| TIC1(Lo)                   | 00  | 0       |
| TIC2(Hi)                   | 00  | 0       |
| TIC2(Lo)                   | 00  | 0       |
| TIC3(Hi)                   | 00  | 0       |
| TIC3(Lo)                   | 00  | 0       |
| TOC1(Hi)                   | FF  | 255     |
| TOC1(Lo)                   | FF  | 255     |
| TOC2(Hi)                   | FF  | 255     |
| TOC2(Lo)                   | FF  | 255     |
| TOC3(Hi)                   | FF  | 255     |
| TOC3(Lo)                   | FF  | 255     |
| TOC4(Hi)                   | FF  | 255     |
| TOC4(Lo)                   | FF  | 255     |
| TOC5(Hi)                   | FF  | 255     |
| TOC5(Lo)                   | FF  | 255     |

Figure 5.11: Instruction and Special Function Registers Information



Figure 5.12: Internal Registers Information

User can view RAM, ROM and EEPROM contents for each cycle or instruction selected. ROM information is cycle independent because it is read-only and does not change in run time. Figures 5.13, 5.14 and 5.15 present screenshots of RAM, ROM and EEPROM content windows respectively.

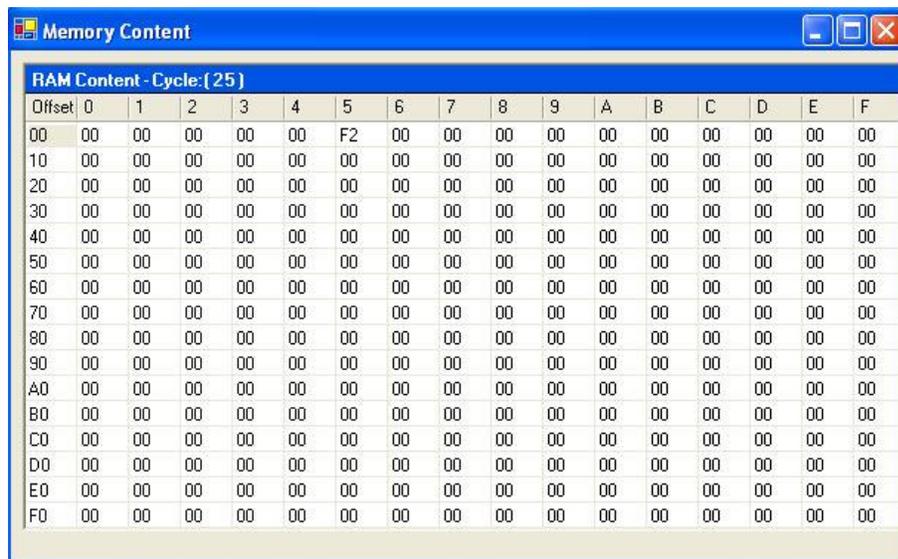


Figure 5.13: RAM Content Window

| Offset | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | A  | B  | C  | D  | E  | F  |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0000   | 86 | 80 | 16 | C6 | 08 | 4F | 5F | CC | 33 | 99 | C6 | F2 | D7 | 05 | 96 | 05 |
| 0010   | CE | 00 | 02 | EB | 03 | 1B | 8B | 80 | 9B | 05 | 0D | 0C | 86 | 80 | 16 | C6 |
| 0020   | 08 | 4F | 5F | CC | 33 | 99 | C6 | F2 | D7 | 05 | 96 | 05 | CE | 00 | 02 | EB |
| 0030   | 03 | 1B | 8B | 80 | 9B | 05 | 0D | 0C | 09 | 18 | 08 | 47 | 7E | E0 | 00 | 3F |
| 0040   | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
| 0050   | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
| 0060   | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
| 0070   | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F |
| 0080   | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F |
| 0090   | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F |
| 00A0   | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF |
| 00B0   | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF |
| 00C0   | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF |
| 00D0   | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF |
| 00E0   | E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | EA | EB | EC | ED | EE | EF |
| 00F0   | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | FA | FB | FC | FD | FE | FF |

Figure 5.14: ROM Content Window

| Offset | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | A  | B  | C  | D  | E  | F  |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 000    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 010    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 020    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 030    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 040    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 050    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 060    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 070    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 080    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 090    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0A0    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0B0    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0C0    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0D0    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0E0    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0F0    | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

Figure 5.15: EEPROM Content Window

## CHAPTER 6

### CONCLUSIONS

Today's complex systems contain both hardware and software parts. Hardware may contain multiple processors, application specific integrated circuits and subsystems. Competition in market pushes companies to produce area and cost effective systems in a short time-to-market period. As a consequence, designers tend to bring all hardware and software of their system together on a single chip with the help of developments in VLSI techniques. Such a system is called system on a chip (SoC). IP cores are descriptions of hardware modules that are ready to be used in SoC designs. In order to use previous know-how on a specific microprocessor, a designer use IP core of that microprocessor in a system design.

Compound design of hardware and software parts of a system allows these parts to be brought together in earlier stages of design flow and allows design of whole system to be verified before hardware prototypes are manufactured. SystemC, as a system design language, is a good co-design and co-verification platform for system-on-a-chip designs. As SystemC allows different abstraction level designs for different modules of a system, designers can use high abstraction levels for test bench devices while they are using low abstraction levels for system hardware modules. This approach reduces effort spent on test bench design and shortens testing periods. If properly written, SystemC codes are synthesizable to hardware description languages with the help of automatic conversion tools and no manual conversion from SystemC to HDLs is needed.

Goal of this thesis has been to develop SystemC implementation of MC68HC11 microcontroller and to provide a user friendly simulation platform with test bench

hardware models. For this purpose, original microcontroller architecture and operation with its peripherals is studied, hardware architecture of microcontroller is implemented in SystemC with the ability of giving information on its internal workings for simulation purposes. Microcontroller core modules are initially designed and tested separately and these modules are brought together with microcontroller software to form initial microcontroller system. Refinements on initially designed core and design of peripheral devices are done using co-design capabilities of SystemC. Microcontroller hardware and software are co-verified using SystemC. System modules are RAM, ROM, ALU, register file and CPU controller unit that constitute CPU of MC68HC11. Serial communications unit, main timer unit and handshake I/O unit modules are peripheral devices of the microcontroller. Microcontroller implementation with its peripheral devices is almost identical to original MC68HC11E9. SystemC implementation of the microcontroller does not contain analog to digital converter peripheral because data structures for analog hardware modeling are not included in standard SystemC library.

For making simulations on microcontroller implementation, a reconfigurable test bench is designed in SystemC using high abstraction levels. Test bench contains test hardware modules such as TTL oscillator, binary switch, seven segment decoder/driver and serial monitor. Any device in test bench can be disabled or enabled and connected to any port of microcontroller. User can configure test bench modules for generating desired signals with desired timings.

For a user friendly interface, a visual application is developed in Microsoft .NET platform in order to allow user to write microcontroller programs, reconfigure test bench, perform simulations of microcontroller system and view simulation results. This visual application is a layer between SystemC implementations and user. When user runs a simulation in visual application, SystemC simulations of system are performed and simulation results are dumped to files. The visual simulation application communicates with SystemC implementation of microcontroller and test bench using input and output files. It passes configuration parameters given

by user to implemented modules and reads simulation results and presents to user in a clearly understandable format.

Using a visual simulation platform for co-simulation of microcontroller programs and SystemC implementation of suggested microcontroller hardware made verification process easy to perform and simulation results clearly understandable. Different simulations are performed on designed microcontroller and these simulation results verified that the designed microcontroller comply with original microcontroller.

The designed simulation platform architecture is expandable and is easily applicable to different systems. It can be used by design teams for performing simulations on their system designs and visualizing simulation results. Using or developing IP cores that give internal information for simulation purposes and developing a visual simulation platform may reduce effort on test and verification and help test personnel to easily discover errors in system.

The developed simulator actually runs cycle accurate hardware and software simulations of microcontroller unit and test bench. So it is not working as fast as non-timed microcontroller simulators that are widely available. Simulation duration is a linear function that is directly proportional with microcontroller simulation time and inversely proportional with microcontroller clock frequency. On a modern computer with 1.5 GHz CPU, simulation speed is about 7000 external clock cycles / second. This means about 150 times slower simulations compared to real time. If average execution time of MC68HC11 instructions is assumed to be 5 bus cycles, about 350 instructions can be simulated per second. Simulation run time is a disadvantage of suggested simulation platform but it has an advantage of presenting internal workings of microcontroller for every micro cycle and allows better understanding of microcontroller operation. It is impractical to use simulation platform for long lasting simulations if only concern in doing simulation is observing microcontroller program performance. This simulation platform is useful if the user

is concerned with cycle accurate information on internal workings of microcontroller hardware when it is running a given program.

For the future works, SystemC implementations can be converted into HDL by using commercially available tools for observing area and speed efficiency and a visual simulation platform that supports both SystemC and HDL simulations can be suggested. Simulations on HDL model or layout should be carried and signal values on specific points and I/O ports of microcontroller should be compared with simulation platform results before hardware implementation is done. Designed microcontroller core can be employed in a larger system and simulator can be improved for that system.

## REFERENCES

- [1] Khan A., "Recent Developments in High-Performance System on Chip", IEEE International Conference on Integrated Circuit Design and Technology, 2004
- [2] Schulz S., Rozenblit J. W., Mrva M., Buchenrieder K., "Model-Based Codesign", IEEE Computer Society Press, August 1998
- [3] Benmohammed M., Merniz S., "Multi-language Co-design Environment for Controller System Design", Journal of Computer Science 1 pp. 337-340, Science Publications, 2005
- [4] IEEE Computer Society, "IEEE Standard SystemC Language Reference Manual", IEEE, New York, USA, 31 March 2006
- [5] Grötter T., Liao S., Martin G., Swan S., "System Design with SystemC", Kluwer Academic Publishers, 2002
- [6] Synthesis Working Group of OSCI, "SystemC Synthesizable Subset", 23 December 2004
- [7] Celoxica, "Agility Compiler", [www.celoxica.com](http://www.celoxica.com), Last accessed: December 2007
- [8] SystemCrafter Ltd., "SystemCrafter", [www.systemcrafter.com](http://www.systemcrafter.com), Last accessed: December 2007
- [9] Calazans N., Moreno E., Hessel F., Rosa V., Moraes F., Carara E., "From VHDL Register Transfer Level to SystemC Transaction Level Modeling: a Comparative Case Study", Proceedings of the 16th Symposium on Integrated Circuits and Systems Design, IEEE, 2003
- [10] Zabawa C. M., Wunnava V. S., "Efficient Digital System Design Methodology with SystemC Register Transfer Level Modeling", IEEE SoutheastCon Proceedings, 2004
- [11] Synopsis Inc., [www.synopsis.com](http://www.synopsis.com), Last accessed: December 2007

- [12] CoWare Inc. [www.coware.com](http://www.coware.com), Last accessed: December 2007
- [13] Organization of OpenCores, "OpenCores.org", [www.opencores.com](http://www.opencores.com), Last accessed: December 2007
- [14] Jonsson B., "A JPEG Encoder in SystemC", A thesis submitted to Lulea University of Technology, Tokyo, 2005
- [15] Kesen L., "Implementation of an 8-bit Microcontroller with SystemC", A thesis submitted to The Graduate School of Natural and Applied Sciences of Middle east Technical University in The Department of Electrical and Electronics Engineering, Ankara, Turkiye, November 2004
- [16] Zengin S., "SystemC Implementation of a RISC-Based Microcontroller Architecture", A thesis submitted to The Graduate School of Natural and Applied Sciences of Middle east Technical University in The Department of Electrical and Electronics Engineering, Ankara, Turkiye, December 2006
- [17] Sözen S., "A Viterbi Decoder Using SystemC for Area Efficient VLSI Implementation", A thesis submitted to The Graduate School of Natural and Applied Sciences of Middle east Technical University in The Department of Electrical and Electronics Engineering, Ankara, Turkiye, September 2006
- [18] Kazancıoğlu U., "The Implementation of a Direct Digital Synthesis Based Function Generator Using SystemC and VHDL", A thesis submitted to The Graduate School of Natural and Applied Sciences of Middle east Technical University in The Department of Electrical and Electronics Engineering, Ankara, Turkiye, February 2007
- [19] Mert Y. M., "SystemC Implementation with Analog and Mixed Signal Modeling for a Microcontroller", A thesis submitted to The Graduate School of Natural and Applied Sciences of Middle east Technical University in The Department of Electrical and Electronics Engineering, Ankara, Turkiye, May 2007
- [20] Freescale Semiconductor, Inc., "M68HC11 Microcontrollers Reference Manual", M68HC11RM/D Rev.6, April 2002.
- [21] Freescale Semiconductor, Inc., "M68HC11E Family Data Sheet", M68HC11E Rev.5.1, July 2005

## APPENDIX A

### M68HC11 INSTRUCTION SET

Table A.1: Information on Operands

| Operands |  |
|----------|--|
| dd       | 8-bit direct address. High byte = \$00. Low byte = (\$00 - \$FF)             |
| ff       | 8-bit positive offset which will be added to index register. (\$00 - \$FF)   |
| hh       | High-order byte of 16-bit extended address                                   |
| ll       | Low-order byte of 16-bit extended address                                    |
| ii       | 8-bit immediate data   |
| jj       | High-order byte of 16-bit immediate data                                     |
| kk       | Low-order byte of 16-bit immediate data                                      |
| mm       | 8-bit mask data  |
| rr       | Signed relative offset which will be added to program counter. (\$80 - \$7F) |

Table A.2: Information on Condition Codes

| Condition Codes |                                       |
|-----------------|---------------------------------------|
| -               | Not affected                          |
| 0               | Cleared                               |
| 1               | Set                                   |
| $\Delta$        | Set or cleared depending on operation |
| $\downarrow$    | Can be cleared but not set            |

Table A.3: M68HC11 Instruction Set (1 / 7)

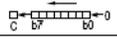
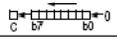
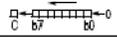
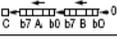
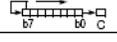
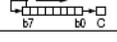
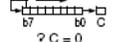
| Mnemonic         | Operation                | Description   | Addressing Mode                               | Instruction                |                                  |                       | Condition Codes |   |   |   |   |   |   |   |
|------------------|--------------------------|---|---|----------------------------|----------------------------------|-----------------------|-----------------|---|---|---|---|---|---|---|
|                  |                          |   |   | Opcode                     | Operand                          | Cycles                | S               | X | H | I | N | Z | V | C |
| ABA              | Add Accumulators         | $A + B \Rightarrow A$   | INH   | 1B                         | —                                | 2                     | —               | — | Δ | — | Δ | Δ | Δ | Δ |
| ABX              | Add B to X               | $IX + (00 : B) \Rightarrow IX$  | INH   | 3A                         | —                                | 3                     | —               | — | — | — | — | — | — | — |
| ABY              | Add B to Y               | $IY + (00 : B) \Rightarrow IY$  | INH   | 18 3A                      | —                                | 4                     | —               | — | — | — | — | — | — | — |
| ADCA (opr)       | Add with Carry to A      | $A + M + C \Rightarrow A$   | A IMM<br>A DIR<br>A EXT<br>A IND,X<br>A IND,Y | 99<br>99<br>B9<br>A9<br>A9 | ii<br>dd<br>hh 11<br>ff<br>ff    | 2<br>3<br>4<br>4<br>5 | —               | — | Δ | — | Δ | Δ | Δ | Δ |
| ADCB (opr)       | Add with Carry to B      | $B + M + C \Rightarrow B$   | B IMM<br>B DIR<br>B EXT<br>B IND,X<br>B IND,Y | C9<br>D9<br>F9<br>E9<br>E9 | ii<br>dd<br>hh 11<br>ff<br>ff    | 2<br>3<br>4<br>4<br>5 | —               | — | Δ | — | Δ | Δ | Δ | Δ |
| ADDA (opr)       | Add Memory to A          | $A + M \Rightarrow A$   | A IMM<br>A DIR<br>A EXT<br>A IND,X<br>A IND,Y | 8B<br>9B<br>9B<br>AB<br>AB | ii<br>dd<br>hh 11<br>ff<br>ff    | 2<br>3<br>4<br>4<br>5 | —               | — | Δ | — | Δ | Δ | Δ | Δ |
| ADDB (opr)       | Add Memory to B          | $B + M \Rightarrow B$   | B IMM<br>B DIR<br>B EXT<br>B IND,X<br>B IND,Y | CB<br>DB<br>FB<br>EB<br>EB | ii<br>dd<br>hh 11<br>ff<br>ff    | 2<br>3<br>4<br>4<br>5 | —               | — | Δ | — | Δ | Δ | Δ | Δ |
| ADDD (opr)       | Add 16-Bit to D          | $D + (M : M + 1) \Rightarrow D$   | IMM<br>DIR<br>EXT<br>IND,X<br>IND,Y           | C3<br>D3<br>F3<br>E3<br>E3 | jj kk<br>dd<br>hh 11<br>ff<br>ff | 4<br>5<br>6<br>6<br>7 | —               | — | — | — | Δ | Δ | Δ | Δ |
| ANDA (opr)       | AND A with Memory        | $A \cdot M \Rightarrow A$   | A IMM<br>A DIR<br>A EXT<br>A IND,X<br>A IND,Y | 84<br>94<br>B4<br>A4<br>A4 | ii<br>dd<br>hh 11<br>ff<br>ff    | 2<br>3<br>4<br>4<br>5 | —               | — | — | — | Δ | Δ | 0 | — |
| ANDB (opr)       | AND B with Memory        | $B \cdot M \Rightarrow B$   | B IMM<br>B DIR<br>B EXT<br>B IND,X<br>B IND,Y | C4<br>D4<br>F4<br>E4<br>E4 | ii<br>dd<br>hh 11<br>ff<br>ff    | 2<br>3<br>4<br>4<br>5 | —               | — | — | — | Δ | Δ | 0 | — |
| ASL (opr)        | Arithmetic Shift Left    |   | EXT<br>IND,X<br>IND,Y                         | 78<br>08<br>68             | hh 11<br>ff<br>ff                | 6<br>6<br>7           | —               | — | — | — | Δ | Δ | Δ | Δ |
| ASLA             | Arithmetic Shift Left A  |  | A INH   | 48                         | —                                | 2                     | —               | — | — | — | Δ | Δ | Δ | Δ |
| ASLB             | Arithmetic Shift Left B  |  | B INH   | 58                         | —                                | 2                     | —               | — | — | — | Δ | Δ | Δ | Δ |
| ASLD             | Arithmetic Shift Left D  |  | INH   | 05                         | —                                | 3                     | —               | — | — | — | Δ | Δ | Δ | Δ |
| ASR              | Arithmetic Shift Right   |  | EXT<br>IND,X<br>IND,Y                         | 77<br>67<br>67             | hh 11<br>ff<br>ff                | 6<br>6<br>7           | —               | — | — | — | Δ | Δ | Δ | Δ |
| ASRA             | Arithmetic Shift Right A |  | A INH   | 47                         | —                                | 2                     | —               | — | — | — | Δ | Δ | Δ | Δ |
| ASRB             | Arithmetic Shift Right B |  | B INH   | 57                         | —                                | 2                     | —               | — | — | — | Δ | Δ | Δ | Δ |
| BCC (rel)        | Branch if Carry Clear    | ? C = 0   | REL   | 24                         | rr                               | 3                     | —               | — | — | — | — | — | — | — |
| BCLR (opr) (msk) | Clear Bit(s)             | $M \cdot (mm) \Rightarrow M$  | DIR<br>IND,X<br>IND,Y                         | 15<br>1D<br>1D             | dd mm<br>ff mm<br>ff mm          | 6<br>7<br>8           | —               | — | — | — | Δ | Δ | 0 | — |
| BCS (rel)        | Branch if Carry Set      | ? C = 1   | REL   | 25                         | rr                               | 3                     | —               | — | — | — | — | — | — | — |
| BEQ (rel)        | Branch if = Zero         | ? Z = 1   | REL   | 27                         | rr                               | 3                     | —               | — | — | — | — | — | — | — |
| BGE (rel)        | Branch if Δ Zero         | ? N ⊕ V = 0   | REL   | 2C                         | rr                               | 3                     | —               | — | — | — | — | — | — | — |

Table A.3: M68HC11 Instruction Set (2 / 7)

| Mnemonic               | Operation                 | Description       | Addressing Mode | Instruction |         |        | Condition Codes |   |   |   |   |   |   |   |
|------------------------|---------------------------|-------------------|-----------------|-------------|---------|--------|-----------------|---|---|---|---|---|---|---|
|                        |                           |                   |                 | Opcode      | Operand | Cycles | S               | X | H | I | N | Z | V | C |
| BGT (rel)              | Branch if > Zero          | ? Z + (N ⊕ V) = 0 | REL             | 2E          | rr      | 3      | —               | — | — | — | — | — | — | — |
| BHI (rel)              | Branch if Higher          | ? C + Z = 0       | REL             | 22          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BHS (rel)              | Branch if Higher or Same  | ? C = 0           | REL             | 24          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BITA (opr)             | Bit(s) Test A with Memory | A • M             | A IMM           | 85          | ii      | 2      | —               | — | — | — | Δ | Δ | 0 | — |
|                        |                           |                   | A DIR           | 95          | dd      | 3      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | A EXT           | B5          | hh 11   | 4      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | A IND,X         | A5          | ff      | 4      | —               | — | — | — | — | — | — | — |
| A IND,Y                | A5                        | ff                | 5               | —           | —       | —      | —               | — | — | — | — | — |   |   |
| BITB (opr)             | Bit(s) Test B with Memory | B • M             | B IMM           | C5          | ii      | 2      | —               | — | — | — | Δ | Δ | 0 | — |
|                        |                           |                   | B DIR           | D5          | dd      | 3      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | B EXT           | F5          | hh 11   | 4      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | B IND,X         | E5          | ff      | 4      | —               | — | — | — | — | — | — | — |
| B IND,Y                | E5                        | ff                | 5               | —           | —       | —      | —               | — | — | — | — | — |   |   |
| BLE (rel)              | Branch if Δ Zero          | ? Z + (N ⊕ V) = 1 | REL             | 2F          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BLO (rel)              | Branch if Lower           | ? C = 1           | REL             | 25          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BLS (rel)              | Branch if Lower or Same   | ? C + Z = 1       | REL             | 23          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BLT (rel)              | Branch if < Zero          | ? N ⊕ V = 1       | REL             | 2D          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BMI (rel)              | Branch if Minus           | ? N = 1           | REL             | 2B          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BNE (rel)              | Branch if not = Zero      | ? Z = 0           | REL             | 26          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BPL (rel)              | Branch if Plus            | ? N = 0           | REL             | 2A          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BRA (rel)              | Branch Always             | ? 1 = 1           | REL             | 20          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BRCLR(opr) (msk) (rel) | Branch if Bit(s) Clear    | ? M • mm = 0      | DIR             | 13          | dd mm   | 6      | —               | — | — | — | — | — | — |   |
|                        |                           |                   | IND,X           | 1F          | rr      | 7      | —               | — | — | — | — | — | — |   |
|                        |                           |                   | IND,Y           | 1F          | ff mm   | 8      | —               | — | — | — | — | — | — |   |
|                        |                           |                   |                 |             | rr      |        | —               | — | — | — | — | — | — |   |
|                        |                           |                   |                 | ff mm       |         | —      | —               | — | — | — | — | — |   |   |
|                        |                           |                   |                 | rr          |         | —      | —               | — | — | — | — | — |   |   |
| BRN (rel)              | Branch Never              | ? 1 = 0           | REL             | 21          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BRSET(opr) (msk) (rel) | Branch if Bit(s) Set      | ? (M) • mm = 0    | DIR             | 12          | dd mm   | 6      | —               | — | — | — | — | — | — |   |
|                        |                           |                   | IND,X           | 1E          | rr      | 7      | —               | — | — | — | — | — | — |   |
|                        |                           |                   | IND,Y           | 1E          | ff mm   | 8      | —               | — | — | — | — | — | — |   |
|                        |                           |                   |                 |             | rr      |        | —               | — | — | — | — | — | — |   |
|                        |                           |                   |                 | ff mm       |         | —      | —               | — | — | — | — | — |   |   |
|                        |                           |                   |                 | rr          |         | —      | —               | — | — | — | — | — |   |   |
| BSET (opr) (msk)       | Set Bit(s)                | M + mm ⇒ M        | DIR             | 14          | dd mm   | 6      | —               | — | — | — | Δ | Δ | 0 | — |
|                        |                           |                   | IND,X           | 1C          | ff mm   | 7      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | IND,Y           | 1C          | ff mm   | 8      | —               | — | — | — | — | — | — | — |
| BSR (rel)              | Branch to Subroutine      | See Figure 3-2    | REL             | 8D          | rr      | 6      | —               | — | — | — | — | — | — | — |
| BVC (rel)              | Branch if Overflow Clear  | ? V = 0           | REL             | 28          | rr      | 3      | —               | — | — | — | — | — | — |   |
| BVS (rel)              | Branch if Overflow Set    | ? V = 1           | REL             | 29          | rr      | 3      | —               | — | — | — | — | — | — |   |
| CBA                    | Compare A to B            | A – B             | INH             | 11          | —       | 2      | —               | — | — | — | Δ | Δ | Δ | Δ |
| CLC                    | Clear Carry Bit           | 0 ⇒ C             | INH             | 0C          | —       | 2      | —               | — | — | — | — | — | — | 0 |
| CLI                    | Clear Interrupt Mask      | 0 ⇒ I             | INH             | 0E          | —       | 2      | —               | — | — | 0 | — | — | — | — |
| CLR (opr)              | Clear Memory Byte         | 0 ⇒ M             | EXT             | 7F          | hh 11   | 6      | —               | — | — | — | 0 | 1 | 0 | 0 |
|                        |                           |                   | IND,X           | 6F          | ff      | 6      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | IND,Y           | 6F          | ff      | 7      | —               | — | — | — | — | — | — | — |
| CLRA                   | Clear Accumulator A       | 0 ⇒ A             | A INH           | 4F          | —       | 2      | —               | — | — | — | 0 | 1 | 0 | 0 |
| CLRB                   | Clear Accumulator B       | 0 ⇒ B             | B INH           | 5F          | —       | 2      | —               | — | — | — | 0 | 1 | 0 | 0 |
| CLV                    | Clear Overflow Flag       | 0 ⇒ V             | INH             | 0A          | —       | 2      | —               | — | — | — | — | — | 0 | — |
| CMPA (opr)             | Compare A to Memory       | A – M             | A IMM           | 81          | ii      | 2      | —               | — | — | — | Δ | Δ | Δ | Δ |
|                        |                           |                   | A DIR           | 91          | dd      | 3      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | A EXT           | B1          | hh 11   | 4      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | A IND,X         | A1          | ff      | 4      | —               | — | — | — | — | — | — | — |
|                        |                           |                   | A IND,Y         | A1          | ff      | 5      | —               | — | — | — | — | — | — | — |

Table A.3: M68HC11 Instruction Set (3 / 7)

| Mnemonic   | Operation                   | Description        | Addressing Mode | Instruction |         |        | Condition Codes |   |   |   |   |   |   |   |
|------------|-----------------------------|--------------------|-----------------|-------------|---------|--------|-----------------|---|---|---|---|---|---|---|
|            |                             |                    |                 | Opcode      | Operand | Cycles | S               | X | H | I | N | Z | V | C |
| CMPB (opr) | Compare B to Memory         | B - M              | B IMM           | C1          | ii      | 2      | -               | - | - | - | Δ | Δ | Δ | Δ |
|            |                             |                    | B DIR           | D1          | dd      | 3      |                 |   |   |   |   |   |   |   |
|            |                             |                    | B EXT           | F1          | hh 11   | 4      |                 |   |   |   |   |   |   |   |
|            |                             |                    | B IND,X         | E1          | ff      | 4      |                 |   |   |   |   |   |   |   |
|            |                             |                    | B IND,Y         | E1          | ff      | 5      |                 |   |   |   |   |   |   |   |
| COM (opr)  | Ones Complement Memory Byte | \$FF - M ⇒ M       | EXT             | 73          | hh 11   | 6      | -               | - | - | - | Δ | Δ | 0 | 1 |
|            |                             |                    | IND,X           | 63          | ff      | 6      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,Y           | 63          | ff      | 7      |                 |   |   |   |   |   |   |   |
| COMA       | Ones Complement A           | \$FF - A ⇒ A       | A INH           | 43          | -       | 2      | -               | - | - | - | Δ | Δ | 0 | 1 |
| COMB       | Ones Complement B           | \$FF - B ⇒ B       | B INH           | 53          | -       | 2      | -               | - | - | - | Δ | Δ | 0 | 1 |
| CPD (opr)  | Compare D to Memory 16-Bit  | D - M : M + 1      | IMM             | 1A 83       | jj kk   | 5      | -               | - | - | - | Δ | Δ | Δ | Δ |
|            |                             |                    | DIR             | 1A 93       | dd      | 6      |                 |   |   |   |   |   |   |   |
|            |                             |                    | EXT             | 1A B3       | hh 11   | 7      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,X           | 1A A3       | ff      | 7      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,Y           | CD A3       | ff      | 7      |                 |   |   |   |   |   |   |   |
| CPX (opr)  | Compare X to Memory 16-Bit  | IX - M : M + 1     | IMM             | 8C          | jj kk   | 4      | -               | - | - | - | Δ | Δ | Δ | Δ |
|            |                             |                    | DIR             | 9C          | dd      | 5      |                 |   |   |   |   |   |   |   |
|            |                             |                    | EXT             | BC          | hh 11   | 6      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,X           | AC          | ff      | 6      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,Y           | CD AC       | ff      | 7      |                 |   |   |   |   |   |   |   |
| CPY (opr)  | Compare Y to Memory 16-Bit  | IY - M : M + 1     | IMM             | 18 8C       | jj kk   | 5      | -               | - | - | - | Δ | Δ | Δ | Δ |
|            |                             |                    | DIR             | 18 9C       | dd      | 6      |                 |   |   |   |   |   |   |   |
|            |                             |                    | EXT             | 18 BC       | hh 11   | 7      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,X           | 1A AC       | ff      | 7      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,Y           | 18 AC       | ff      | 7      |                 |   |   |   |   |   |   |   |
| DAA        | Decimal Adjust A            | Adjust Sum to BCD  | INH             | 19          | -       | 2      | -               | - | - | - | Δ | Δ | Δ | Δ |
| DEC (opr)  | Decrement Memory Byte       | M - 1 ⇒ M          | EXT             | 7A          | hh 11   | 6      | -               | - | - | - | Δ | Δ | Δ | - |
|            |                             |                    | IND,X           | 6A          | ff      | 6      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,Y           | 18 6A       | ff      | 7      |                 |   |   |   |   |   |   |   |
| DECA       | Decrement Accumulator A     | A - 1 ⇒ A          | A INH           | 4A          | -       | 2      | -               | - | - | - | Δ | Δ | Δ | - |
| DECB       | Decrement Accumulator B     | B - 1 ⇒ B          | B INH           | 5A          | -       | 2      | -               | - | - | - | Δ | Δ | Δ | - |
| DES        | Decrement Stack Pointer     | SP - 1 ⇒ SP        | INH             | 34          | -       | 3      | -               | - | - | - | - | - | - | - |
| DEX        | Decrement Index Register X  | IX - 1 ⇒ IX        | INH             | 09          | -       | 3      | -               | - | - | - | - | Δ | - | - |
| DEY        | Decrement Index Register Y  | IY - 1 ⇒ IY        | INH             | 18 09       | -       | 4      | -               | - | - | - | - | Δ | - | - |
| EORA (opr) | Exclusive OR A with Memory  | A ⊕ M ⇒ A          | A IMM           | 88          | ii      | 2      | -               | - | - | - | Δ | Δ | 0 | - |
|            |                             |                    | A DIR           | 98          | dd      | 3      |                 |   |   |   |   |   |   |   |
|            |                             |                    | A EXT           | B8          | hh 11   | 4      |                 |   |   |   |   |   |   |   |
|            |                             |                    | A IND,X         | A8          | ff      | 4      |                 |   |   |   |   |   |   |   |
|            |                             |                    | A IND,Y         | 18 A8       | ff      | 5      |                 |   |   |   |   |   |   |   |
| EORB (opr) | Exclusive OR B with Memory  | B ⊕ M ⇒ B          | B IMM           | C8          | ii      | 2      | -               | - | - | - | Δ | Δ | 0 | - |
|            |                             |                    | B DIR           | D8          | dd      | 3      |                 |   |   |   |   |   |   |   |
|            |                             |                    | B EXT           | F8          | hh 11   | 4      |                 |   |   |   |   |   |   |   |
|            |                             |                    | B IND,X         | E8          | ff      | 4      |                 |   |   |   |   |   |   |   |
|            |                             |                    | B IND,Y         | 18 E8       | ff      | 5      |                 |   |   |   |   |   |   |   |
| FDIV       | Fractional Divide 16 by 16  | D / IX ⇒ IX; r ⇒ D | INH             | 03          | -       | 41     | -               | - | - | - | - | Δ | Δ | Δ |
| IDIV       | Integer Divide 16 by 16     | D / IX ⇒ IX; r ⇒ D | INH             | 02          | -       | 41     | -               | - | - | - | - | Δ | 0 | Δ |
| INC (opr)  | Increment Memory Byte       | M + 1 ⇒ M          | EXT             | 7C          | hh 11   | 6      | -               | - | - | - | Δ | Δ | Δ | - |
|            |                             |                    | IND,X           | 6C          | ff      | 6      |                 |   |   |   |   |   |   |   |
|            |                             |                    | IND,Y           | 18 6C       | ff      | 7      |                 |   |   |   |   |   |   |   |
| INCA       | Increment Accumulator A     | A + 1 ⇒ A          | A INH           | 4C          | -       | 2      | -               | - | - | - | Δ | Δ | Δ | - |



Table A.3: M68HC11 Instruction Set (5 / 7)

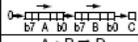
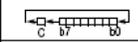
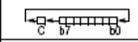
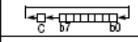
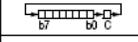
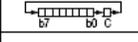
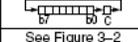
| Mnemonic   | Operation                    | Description  | Addressing Mode                               | Instruction                |                               |                       | Condition Codes |              |          |          |          |          |          |          |
|------------|------------------------------|--|---|----------------------------|-------------------------------|-----------------------|-----------------|--------------|----------|----------|----------|----------|----------|----------|
|            |                              |  |   | Opcode                     | Operand                       | Cycles                | S               | X            | H        | I        | N        | Z        | V        | C        |
| LSRD       | Logical Shift Right Double   | <br>$A \Rightarrow B \Rightarrow D$ | INH   | 04                         | —                             | 3                     | —               | —            | —        | —        | 0        | $\Delta$ | $\Delta$ | $\Delta$ |
| MUL        | Multiply 8 by 8              | $A \times B \Rightarrow D$   | INH   | 3D                         | —                             | 10                    | —               | —            | —        | —        | —        | —        | —        | $\Delta$ |
| NEG (opr)  | Two's Complement Memory Byte | $0 - M \Rightarrow M$  | EXT<br>IND,X<br>IND,Y                         | 70<br>60<br>60             | hh 11<br>ff<br>ff             | 6<br>6<br>7           | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| NEGA       | Two's Complement A           | $0 - A \Rightarrow A$  | A INH   | 40                         | —                             | 2                     | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| NEGB       | Two's Complement B           | $0 - B \Rightarrow B$  | B INH   | 50                         | —                             | 2                     | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| NOP        | No operation                 | No Operation   | INH   | 01                         | —                             | 2                     | —               | —            | —        | —        | —        | —        | —        | —        |
| ORAA (opr) | OR Accumulator A (Inclusive) | $A + M \Rightarrow A$  | A IMM<br>A DIR<br>A EXT<br>A IND,X<br>A IND,Y | 8A<br>9A<br>BA<br>AA<br>AA | ii<br>dd<br>hh 11<br>ff<br>ff | 2<br>3<br>4<br>4<br>5 | —               | —            | —        | —        | $\Delta$ | $\Delta$ | 0        | —        |
| ORAB (opr) | OR Accumulator B (Inclusive) | $B + M \Rightarrow B$  | B IMM<br>B DIR<br>B EXT<br>B IND,X<br>B IND,Y | CA<br>DA<br>FA<br>EA<br>EA | ii<br>dd<br>hh 11<br>ff<br>ff | 2<br>3<br>4<br>4<br>5 | —               | —            | —        | —        | $\Delta$ | $\Delta$ | 0        | —        |
| PSHA       | Push A onto Stack            | $A \Rightarrow \text{Stk}, SP = SP - 1$  | A INH   | 36                         | —                             | 3                     | —               | —            | —        | —        | —        | —        | —        | —        |
| PSHB       | Push B onto Stack            | $B \Rightarrow \text{Stk}, SP = SP - 1$  | B INH   | 37                         | —                             | 3                     | —               | —            | —        | —        | —        | —        | —        | —        |
| PSHX       | Push X onto Stack (Lo First) | $IX \Rightarrow \text{Stk}, SP = SP - 2$   | INH   | 3C                         | —                             | 4                     | —               | —            | —        | —        | —        | —        | —        | —        |
| PSHY       | Push Y onto Stack (Lo First) | $IY \Rightarrow \text{Stk}, SP = SP - 2$   | INH   | 18 3C                      | —                             | 5                     | —               | —            | —        | —        | —        | —        | —        | —        |
| PULA       | Pull A from Stack            | $SP = SP + 1, A \Leftarrow \text{Stk}$   | A INH   | 32                         | —                             | 4                     | —               | —            | —        | —        | —        | —        | —        | —        |
| PULB       | Pull B from Stack            | $SP = SP + 1, B \Leftarrow \text{Stk}$   | B INH   | 33                         | —                             | 4                     | —               | —            | —        | —        | —        | —        | —        | —        |
| PULX       | Pull X From Stack (Hi First) | $SP = SP + 2, IX \Leftarrow \text{Stk}$  | INH   | 38                         | —                             | 5                     | —               | —            | —        | —        | —        | —        | —        | —        |
| PULY       | Pull Y from Stack (Hi First) | $SP = SP + 2, IY \Leftarrow \text{Stk}$  | INH   | 18 38                      | —                             | 6                     | —               | —            | —        | —        | —        | —        | —        | —        |
| ROL (opr)  | Rotate Left                  | <br>$C \quad b7 \quad b0$         | EXT<br>IND,X<br>IND,Y                         | 79<br>69<br>69             | hh 11<br>ff<br>ff             | 6<br>6<br>7           | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| ROLA       | Rotate Left A                | <br>$C \quad b7 \quad b0$         | A INH   | 49                         | —                             | 2                     | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| ROLB       | Rotate Left B                | <br>$C \quad b7 \quad b0$         | B INH   | 59                         | —                             | 2                     | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| ROR (opr)  | Rotate Right                 | <br>$b7 \quad b0 \quad C$         | EXT<br>IND,X<br>IND,Y                         | 76<br>66<br>66             | hh 11<br>ff<br>ff             | 6<br>6<br>7           | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| RORA       | Rotate Right A               | <br>$b7 \quad b0 \quad C$         | A INH   | 46                         | —                             | 2                     | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| RORB       | Rotate Right B               | <br>$b7 \quad b0 \quad C$         | B INH   | 56                         | —                             | 2                     | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| RTI        | Return from Interrupt        | See Figure 3-2   | INH   | 3B                         | —                             | 12                    | $\Delta$        | $\downarrow$ | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |
| RTS        | Return from Subroutine       | See Figure 3-2   | INH   | 39                         | —                             | 5                     | —               | —            | —        | —        | —        | —        | —        | —        |
| SBA        | Subtract B from A            | $A - B \Rightarrow A$  | INH   | 10                         | —                             | 2                     | —               | —            | —        | —        | $\Delta$ | $\Delta$ | $\Delta$ | $\Delta$ |

Table A.3: M68HC11 Instruction Set (6 / 7)

| Mnemonic   | Operation                  | Description                            | Addressing Mode | Instruction |         |        | Condition Codes |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|------------|----------------------------|--|-----------------|-------------|---------|--------|-----------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|----|
|            |                            |  |                 | Opcode      | Operand | Cycles | S               | X | H | I | N | Z | V | C |   |   |   |   |   |   |   |    |
| SBCA (opr) | Subtract with Carry from A | $A - M - C \Rightarrow A$              | A IMM           | 92          | ii      | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — |   |    |
|            |                            |  | A DIR           | 92          | dd      | 3      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | A EXT           | B2          | hh 11   | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | A IND,X         | A2          | ff      | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| A IND,Y    | A2                         | ff                                     | 5               | 18          |         |        |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| SBCB (opr) | Subtract with Carry from B | $B - M - C \Rightarrow B$              | B IMM           | C2          | ii      | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — |   |    |
|            |                            |  | B DIR           | D2          | dd      | 3      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | B EXT           | F2          | hh 11   | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | B IND,X         | E2          | ff      | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| B IND,Y    | E2                         | ff                                     | 5               | 18          |         |        |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| SEC        | Set Carry                  | $1 \Rightarrow C$                      | INH             | 0D          | —       | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | 1 |    |
| SEI        | Set Interrupt Mask         | $1 \Rightarrow I$                      | INH             | 0F          | —       | 2      | —               | — | — | — | 1 | — | — | — | — | — | — | — | — | — | — |    |
| SEV        | Set Overflow Flag          | $1 \Rightarrow V$                      | INH             | 0B          | —       | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | 1 |    |
| STAA (opr) | Store Accumulator A        | $A \Rightarrow M$                      | A DIR           | 97          | dd      | 3      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | A EXT           | B7          | hh 11   | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | A IND,X         | A7          | ff      | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | A IND,Y         | A7          | ff      | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 18 |
| STAB (opr) | Store Accumulator B        | $B \Rightarrow M$                      | B DIR           | D7          | dd      | 3      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | B EXT           | F7          | hh 11   | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | B IND,X         | E7          | ff      | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | B IND,Y         | E7          | ff      | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 18 |
| STD (opr)  | Store Accumulator D        | $A \Rightarrow M, B \Rightarrow M + 1$ | DIR             | DD          | dd      | 4      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | EXT             | FD          | hh 11   | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | IND,X           | ED          | ff      | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | IND,Y           | ED          | ff      | 6      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 18 |
| STOP       | Stop Internal Clocks       | —                                      | INH             | CF          | —       | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
| STS (opr)  | Store Stack Pointer        | $SP \Rightarrow M : M + 1$             | DIR             | 9F          | dd      | 4      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | EXT             | BF          | hh 11   | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | IND,X           | AF          | ff      | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | IND,Y           | AF          | ff      | 6      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 18 |
| STX (opr)  | Store Index Register X     | $IX \Rightarrow M : M + 1$             | DIR             | DF          | dd      | 4      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | EXT             | FF          | hh 11   | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | IND,X           | EF          | ff      | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | IND,Y           | EF          | ff      | 6      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 18 |
| STY (opr)  | Store Index Register Y     | $IY \Rightarrow M : M + 1$             | DIR             | 18          | DD      | 5      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | EXT             | 18          | FF      | hh 11  |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 6  |
|            |                            |  | IND,X           | 1A          | EF      | ff     |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 6  |
|            |                            |  | IND,Y           | 18          | EF      | ff     |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 6  |
| SUBA (opr) | Subtract Memory from A     | $A - M \Rightarrow A$                  | A IMM           | 80          | ii      | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | A DIR           | 90          | dd      | 3      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | A EXT           | B0          | hh 11   | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | A IND,X         | A0          | ff      | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| A IND,Y    | A0                         | ff                                     | 5               | 18          |         |        |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| SUBB (opr) | Subtract Memory from B     | $B - M \Rightarrow B$                  | A IMM           | C0          | ii      | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | A DIR           | D0          | dd      | 3      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | A EXT           | F0          | hh 11   | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | A IND,X         | E0          | ff      | 4      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| A IND,Y    | E0                         | ff                                     | 5               | 18          |         |        |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| SUBD (opr) | Subtract Memory from D     | $D - M : M + 1 \Rightarrow D$          | IMM             | 83          | jj kk   | 4      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | DIR             | 93          | dd      | 5      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | EXT             | B3          | hh 11   | 6      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | IND,X           | A3          | ff      | 6      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| IND,Y      | A3                         | ff                                     | 7               | 18          |         |        |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
| SWI        | Software Interrupt         | See Figure 3-2                         | INH             | 3F          | —       | 14     | —               | — | — | — | 1 | — | — | — | — | — | — | — | — | — | — |    |
| TAB        | Transfer A to B            | $A \Rightarrow B$                      | INH             | 16          | —       | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
| TAP        | Transfer A to CC Register  | $A \Rightarrow CCR$                    | INH             | 06          | —       | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
| TBA        | Transfer B to A            | $B \Rightarrow A$                      | INH             | 17          | —       | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
| TEST       | TEST (Only in Test Modes)  | Address Bus Counts                     | INH             | 00          | —       | *      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
| TPA        | Transfer CC Register to A  | $CCR \Rightarrow A$                    | INH             | 07          | —       | 2      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
| TST (opr)  | Test for Zero or Minus     | $M - 0$                                | EXT             | 7D          | hh 11   | 6      | —               | — | — | — | — | — | — | — | — | — | — | — | — | — | — |    |
|            |                            |  | IND,X           | 6D          | ff      | 6      |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |    |
|            |                            |  | IND,Y           | 18          | 6D      | ff     |                 |   |   |   |   |   |   |   |   |   |   |   |   |   |   | 7  |

Table A.3: M68HC11 Instruction Set (7 / 7)

| Mnemonic | Operation                   | Description       | Addressing Mode | Instruction |         |        | Condition Codes |   |   |   |   |   |   |   |
|----------|-----------------------------|-------------------|-----------------|-------------|---------|--------|-----------------|---|---|---|---|---|---|---|
|          |                             |                   |                 | Opcode      | Operand | Cycles | S               | X | H | I | N | Z | V | C |
| TSTA     | Test A for Zero or Minus    | A = 0             | A INH           | 4D          | —       | 2      | —               | — | — | — | Δ | Δ | 0 | 0 |
| TSTB     | Test B for Zero or Minus    | B = 0             | B INH           | 5D          | —       | 2      | —               | — | — | — | Δ | Δ | 0 | 0 |
| TSX      | Transfer Stack Pointer to X | SP + 1 ⇒ IX       | INH             | 30          | —       | 3      | —               | — | — | — | — | — | — | — |
| TSY      | Transfer Stack Pointer to Y | SP + 1 ⇒ IY       | INH             | 18 30       | —       | 4      | —               | — | — | — | — | — | — | — |
| TXS      | Transfer X to Stack Pointer | IX - 1 ⇒ SP       | INH             | 35          | —       | 3      | —               | — | — | — | — | — | — | — |
| TYS      | Transfer Y to Stack Pointer | IY - 1 ⇒ SP       | INH             | 18 35       | —       | 4      | —               | — | — | — | — | — | — | — |
| WAI      | Wait for Interrupt          | Stack Regs & WAIT | INH             | 9E          | —       | **     | —               | — | — | — | — | — | — | — |
| XGDX     | Exchange D with X           | IX ⇒ D, D ⇒ IX    | INH             | 8F          | —       | 3      | —               | — | — | — | — | — | — | — |
| XGDY     | Exchange D with Y           | IY ⇒ D, D ⇒ IY    | INH             | 18 8F       | —       | 4      | —               | — | — | — | — | — | — | — |

## APPENDIX B

### MICROCONTROLLER TEST CODE

#### B.1 Instructions and Addressing Modes Test Program

```

                                ORG  $D000
                                LDAA  #$20 ;***ARITHMETIC OPERATIONS***
                                LDAB  #$30
                                STAB  $0001
                                ADDA  $0001 ;DIRECT ADDITION
                                LDAB  #$50
                                STAB  $0002
                                SUBA  $0002 ;DIRECT SUBTRACTION
                                BEQ   OK01
                                LDAA  #$01
                                JMP   FAILED
OK01:
                                LDAA  #$50
                                ADDA  #$40 ;IMMEDIATE ADDITION
                                SUBA  #$90 ;IMMEDIATE SUBTRACTION
                                BEQ   OK02
                                LDAA  #$02
                                JMP   FAILED
OK02:
                                LDAA  #$90
                                LDAB  #$10
                                STAB  $0100
                                ADDA  $0100 ;EXTENDED ADDITION
                                LDAB  #$A0
                                STAB  $0101
                                SUBA  $0101 ;EXTENDED SUBTRACTION
                                BEQ   OK03
                                LDAA  #$03
                                JMP   FAILED
OK03:
                                LDAA  #$A0
                                LDAB  #$05
                                STAB  $0102
```

```

LDX  #$0100
ADDA $02,X ;INDEXED ADDITION
LDAB #$A5
STAB $0103
SUBA $03,X ;INDEXED SUBTRACTION
BEQ  OK04
LDAA #$04
JMP  FAILED

OK04:
LDAA #$A5
INCA                ;INHERENT INCREMENT
CMPA #$A6
BEQ  OK05
LDAA #$05
JMP  FAILED

OK05:
DECA                ;INHERENT DECREMENT
CMPA #$A5
BEQ  OK06
LDAA #$06
JMP  FAILED

OK06:
INC  $0101 ;EXTENDED INCREMENT
LDAB #$A1
CMPB $0101
BEQ  OK07
LDAA #$07
JMP  FAILED

OK07:
DEC  $0101 ;EXTENDED DECREMENT
LDAB #$A0
CMPB $0101
BEQ  OK08
LDAA #$08
JMP  FAILED

OK08:
INC  $02,X ;INDEXED INCREMENT
LDAB #$06
CMPB $0102
BEQ  OK09
LDAA #$09
JMP  FAILED

OK09:
DEC  $02,X ;INDEXED DECREMENT
LDAB #$05
CMPB $0102
BEQ  OK0A
LDAA #$0A
JMP  FAILED

OK0A:
LDAA #$23 ;***MULTIPLICATION & DIVISION***

```

```

LDAB #$17
MUL      ;MULTIPLICATION
SUBD #$0325
BEQ OK0B
LDAA #$0B
JMP FAILED

OK0B:
LDD #$0325
LDX #$0013
IDIV     ;DIVISION
SUBD #$0007 ;CHECK REMAINDER
BEQ OK0C
LDAA #$0C
JMP FAILED

OK0C:
XGDX
SUBD #$002A ;CHECK QUOTIENT
BEQ OK0D
LDAA #$0D
JMP FAILED

OK0D:
LDAA #$25 ;***LOGICAL OPERATIONS***
ANDA #$F0 ;IMMEDIATE AND
LDAB #$20
SBA
BEQ OK0E
LDAA #$0E
JMP FAILED

OK0E:
ORAA #$97 ;IMMEDIATE OR
LDAB #$0F
STAB $0004
ANDA $0004 ;DIRECT AND
CMPA #$07
BEQ OK0F
LDAA #$0F
JMP FAILED

OK0F:
LDAB #$A0
STAB $0005
ORAA $0005 ;DIRECT OR
ASLA     ;INHERENT ARITHMETIC SHIFT LEFT
CMPA #$4E
BEQ OK10
LDAA #$10
JMP FAILED

OK10:
SEC
RORA     ;INHERENT ROTATE RIGHT
CMPA #$A7
BEQ OK11

```

```

LDAA #$11
JMP FAILED

OK11:
SEC
LDAA #$25
STAA $0105
ROL $0105 ;EXTENDED ROTATE LEFT
LDAB #$4B
CMPB $0105
BEQ OK12
LDAA #$12
JMP FAILED

OK12:
LDX #$0100
NEG $05,X ;INDEXED NEGATE
LDAA #$B5
CMPA $05,X
BEQ OK13
LDAA #$13
JMP FAILED

OK13:
ASR $0105 ;EXTENDED ARITHMETIC SHIFT RIGHT
LDAA $05,X
CMPA #$DA
BEQ OK14
LDAA #$14
JMP FAILED

OK14:
LSR $05,X ;INDEXED LOGICAL SHIFT RIGHT
LDAA #$6D
CMPA $0105
BEQ OK15
LDAA #$15
JMP FAILED

OK15:
COMA ;INHERENT COMPLEMENT
CMPA #$92
BEQ OK16
LDAA #$16
JMP FAILED

OK16:
COM $0105 ;EXTENDED COMPLEMENT
LDAB #$92
CMPB $05,X
BEQ OK17
LDAA #$17
JMP FAILED

OK17:
EORA #$F0 ;IMMEDIATE EXCLUSIVE OR
CMPA #$62

```

```

      BEQ  OK18
      LDAA #$18
      JMP  FAILED
OK18:
      CLR  $05,X ;INDEXED CLEAR
      CLRA ;INHERENT CLEAR
      CMPA $0105
      BEQ  OK19
      LDAA #$19
      JMP  FAILED
OK19:
      BSET $05,X #$$AA ;INDEXED SET BITS
      LDAA #$$AA
      CMPA $0105
      BEQ  OK1A
      LDAA #$$1A
      JMP  FAILED
OK1A:
      BCLR $05,X #$$0F ;INDEXED CLEAR BITS
      LDAA #$$A0
      CMPA $05,X
      BEQ  OK1B
      LDAA #$$1B
      JMP  FAILED
OK1B:
      LDAA #$$45
      LDAB #$$CD
      PSHA ;PUSH
      PSHB
      CLRA
      CLRB
      PULB ;PULL
      PULA
      CMPA #$$45
      BEQ  OK1C
      LDAA #$$1C
      JMP  FAILED
OK1C:
      CMPB #$$CD
      BEQ  OK1D
      LDAA #$$1D
      JMP  FAILED
OK1D:
      LDAB #$$15
      STAB $0010
      BRCLR $0010 #$$EA OK1E ;BRANCH IF BITS CLEAR
      LDAA #$$1E
      JMP  FAILED
OK1E:
      BRSET $0010 #$$05 OK1F ;BRANCH IF BITS SET
      LDAA #$$1F

```

```
OK1F:      JMP   FAILED
           LDAA  #$00
           LDAB  #$00
           STOP
FAILED:    LDAB  #$EE
           STOP
```

## B.2 Execution of Test Program on Original MC68HC11

|             |        |        |        |      |      |      |        |
|-------------|--------|--------|--------|------|------|------|--------|
| LDAA #\$20  | P-8002 | Y-FFFF | X-FFFF | A-20 | B-FF | C-90 | S-0041 |
| LDAB #\$30  | P-8004 | Y-FFFF | X-FFFF | A-20 | B-30 | C-90 | S-0041 |
| STAB \$01   | P-8006 | Y-FFFF | X-FFFF | A-20 | B-30 | C-90 | S-0041 |
| ADDA \$01   | P-8008 | Y-FFFF | X-FFFF | A-50 | B-30 | C-90 | S-0041 |
| LDAB #\$50  | P-800A | Y-FFFF | X-FFFF | A-50 | B-50 | C-90 | S-0041 |
| STAB \$02   | P-800C | Y-FFFF | X-FFFF | A-50 | B-50 | C-90 | S-0041 |
| SUBA \$02   | P-800E | Y-FFFF | X-FFFF | A-00 | B-50 | C-94 | S-0041 |
| BEQ \$8015  | P-8015 | Y-FFFF | X-FFFF | A-00 | B-50 | C-94 | S-0041 |
| LDAA #\$50  | P-8017 | Y-FFFF | X-FFFF | A-50 | B-50 | C-90 | S-0041 |
| ADDA \$040  | P-8019 | Y-FFFF | X-FFFF | A-90 | B-50 | C-9A | S-0041 |
| SUBA #\$90  | P-801B | Y-FFFF | X-FFFF | A-00 | B-50 | C-94 | S-0041 |
| BEQ \$8022  | P-8022 | Y-FFFF | X-FFFF | A-00 | B-50 | C-94 | S-0041 |
| LDAA #\$90  | P-8024 | Y-FFFF | X-FFFF | A-90 | B-50 | C-98 | S-0041 |
| LDAB #\$10  | P-8026 | Y-FFFF | X-FFFF | A-90 | B-10 | C-90 | S-0041 |
| STAB \$0100 | P-8029 | Y-FFFF | X-FFFF | A-90 | B-10 | C-90 | S-0041 |
| ADDA \$0100 | P-802C | Y-FFFF | X-FFFF | A-A0 | B-10 | C-98 | S-0041 |
| LDAB #\$A0  | P-802E | Y-FFFF | X-FFFF | A-A0 | B-A0 | C-98 | S-0041 |
| STAB \$0101 | P-8031 | Y-FFFF | X-FFFF | A-A0 | B-A0 | C-98 | S-0041 |
| SUBA \$0101 | P-8034 | Y-FFFF | X-FFFF | A-00 | B-A0 | C-94 | S-0041 |
| BEQ \$803B  | P-803B | Y-FFFF | X-FFFF | A-00 | B-A0 | C-94 | S-0041 |
| LDAA #\$A0  | P-803D | Y-FFFF | X-FFFF | A-A0 | B-A0 | C-98 | S-0041 |
| LDAB #\$05  | P-803F | Y-FFFF | X-FFFF | A-A0 | B-05 | C-90 | S-0041 |
| STAB \$0102 | P-8042 | Y-FFFF | X-FFFF | A-A0 | B-05 | C-90 | S-0041 |
| LDX #\$0100 | P-8045 | Y-FFFF | X-0100 | A-A0 | B-05 | C-90 | S-0041 |
| ADDA \$02,X | P-8047 | Y-FFFF | X-0100 | A-A5 | B-05 | C-98 | S-0041 |
| LDAB #\$A5  | P-8049 | Y-FFFF | X-0100 | A-A5 | B-A5 | C-98 | S-0041 |
| STAB \$0103 | P-804C | Y-FFFF | X-0100 | A-A5 | B-A5 | C-98 | S-0041 |
| SUBA \$03,X | P-804E | Y-FFFF | X-0100 | A-00 | B-A5 | C-94 | S-0041 |
| BEQ \$8055  | P-8055 | Y-FFFF | X-0100 | A-00 | B-A5 | C-94 | S-0041 |
| LDAA #\$A5  | P-8057 | Y-FFFF | X-0100 | A-A5 | B-A5 | C-98 | S-0041 |
| INCA        | P-8058 | Y-FFFF | X-0100 | A-A6 | B-A5 | C-98 | S-0041 |
| CMPA #\$A6  | P-805A | Y-FFFF | X-0100 | A-A6 | B-A5 | C-94 | S-0041 |
| BEQ \$8061  | P-8061 | Y-FFFF | X-0100 | A-A6 | B-A5 | C-94 | S-0041 |
| DECA        | P-8062 | Y-FFFF | X-0100 | A-A5 | B-A5 | C-98 | S-0041 |
| CMPA #\$A5  | P-8064 | Y-FFFF | X-0100 | A-A5 | B-A5 | C-94 | S-0041 |
| BEQ \$806B  | P-806B | Y-FFFF | X-0100 | A-A5 | B-A5 | C-94 | S-0041 |
| INC \$0101  | P-806E | Y-FFFF | X-0100 | A-A5 | B-A5 | C-98 | S-0041 |
| LDAB #\$A1  | P-8070 | Y-FFFF | X-0100 | A-A5 | B-A1 | C-98 | S-0041 |
| CMPB \$0101 | P-8073 | Y-FFFF | X-0100 | A-A5 | B-A1 | C-94 | S-0041 |
| BEQ \$807A  | P-807A | Y-FFFF | X-0100 | A-A5 | B-A1 | C-94 | S-0041 |
| DEC \$0101  | P-807D | Y-FFFF | X-0100 | A-A5 | B-A1 | C-98 | S-0041 |
| LDAB #\$A0  | P-807F | Y-FFFF | X-0100 | A-A5 | B-A0 | C-98 | S-0041 |
| CMPB \$0101 | P-8082 | Y-FFFF | X-0100 | A-A5 | B-A0 | C-94 | S-0041 |
| BEQ \$8089  | P-8089 | Y-FFFF | X-0100 | A-A5 | B-A0 | C-94 | S-0041 |
| INC \$02,X  | P-808B | Y-FFFF | X-0100 | A-A5 | B-A0 | C-90 | S-0041 |
| LDAB #\$06  | P-808D | Y-FFFF | X-0100 | A-A5 | B-06 | C-90 | S-0041 |
| CMPB \$0102 | P-8090 | Y-FFFF | X-0100 | A-A5 | B-06 | C-94 | S-0041 |
| BEQ \$8097  | P-8097 | Y-FFFF | X-0100 | A-A5 | B-06 | C-94 | S-0041 |

|              |  |
|--------------|--|
| DEC \$02,X   | P-8099 Y-FFFF X-0100 A-A5 B-06 C-90 S-0041 |
| LDAB #\$05   | P-809B Y-FFFF X-0100 A-A5 B-05 C-90 S-0041 |
| CMPB \$0102  | P-809E Y-FFFF X-0100 A-A5 B-05 C-94 S-0041 |
| BEQ \$80A5   | P-80A5 Y-FFFF X-0100 A-A5 B-05 C-94 S-0041 |
| LDAA #\$23   | P-80A7 Y-FFFF X-0100 A-23 B-05 C-90 S-0041 |
| LDAB #\$17   | P-80A9 Y-FFFF X-0100 A-23 B-17 C-90 S-0041 |
| MUL          | P-80AA Y-FFFF X-0100 A-03 B-25 C-90 S-0041 |
| SUBD #\$0325 | P-80AD Y-FFFF X-0100 A-00 B-00 C-94 S-0041 |
| BEQ \$80B4   | P-80B4 Y-FFFF X-0100 A-00 B-00 C-94 S-0041 |
| LDD #\$0325  | P-80B7 Y-FFFF X-0100 A-03 B-25 C-90 S-0041 |
| LDX #\$0013  | P-80BA Y-FFFF X-0013 A-03 B-25 C-90 S-0041 |
| IDIV         | P-80BB Y-FFFF X-002A A-00 B-07 C-90 S-0041 |
| SUBD #\$0007 | P-80BE Y-FFFF X-002A A-00 B-00 C-94 S-0041 |
| BEQ \$80C5   | P-80C5 Y-FFFF X-002A A-00 B-00 C-94 S-0041 |
| XGDX         | P-80C6 Y-FFFF X-0000 A-00 B-2A C-94 S-0041 |
| SUBD #\$002A | P-80C9 Y-FFFF X-0000 A-00 B-00 C-94 S-0041 |
| BEQ \$80D0   | P-80D0 Y-FFFF X-0000 A-00 B-00 C-94 S-0041 |
| LDAA #\$25   | P-80D2 Y-FFFF X-0000 A-25 B-00 C-90 S-0041 |
| ANDA #\$F0   | P-80D4 Y-FFFF X-0000 A-20 B-00 C-90 S-0041 |
| LDAB #\$20   | P-80D6 Y-FFFF X-0000 A-20 B-20 C-90 S-0041 |
| SBA          | P-80D7 Y-FFFF X-0000 A-00 B-20 C-94 S-0041 |
| BEQ \$80DE   | P-80DE Y-FFFF X-0000 A-00 B-20 C-94 S-0041 |
| ORAA #\$97   | P-80E0 Y-FFFF X-0000 A-97 B-20 C-98 S-0041 |
| LDAB #\$0F   | P-80E2 Y-FFFF X-0000 A-97 B-0F C-90 S-0041 |
| STAB \$04    | P-80E4 Y-FFFF X-0000 A-97 B-0F C-90 S-0041 |
| ANDA \$04    | P-80E6 Y-FFFF X-0000 A-07 B-0F C-90 S-0041 |
| CMPA #\$07   | P-80E8 Y-FFFF X-0000 A-07 B-0F C-94 S-0041 |
| BEQ \$80EF   | P-80EF Y-FFFF X-0000 A-07 B-0F C-94 S-0041 |
| LDAB #\$A0   | P-80F1 Y-FFFF X-0000 A-07 B-A0 C-98 S-0041 |
| STAB \$05    | P-80F3 Y-FFFF X-0000 A-07 B-A0 C-98 S-0041 |
| ORAA \$05    | P-80F5 Y-FFFF X-0000 A-A7 B-A0 C-98 S-0041 |
| ASLA         | P-80F6 Y-FFFF X-0000 A-4E B-A0 C-93 S-0041 |
| CMPA #\$4E   | P-80F8 Y-FFFF X-0000 A-4E B-A0 C-94 S-0041 |
| BEQ \$80FF   | P-80FF Y-FFFF X-0000 A-4E B-A0 C-94 S-0041 |
| SEC          | P-8100 Y-FFFF X-0000 A-4E B-A0 C-95 S-0041 |
| RORA         | P-8101 Y-FFFF X-0000 A-A7 B-A0 C-9A S-0041 |
| CMPA #\$A7   | P-8103 Y-FFFF X-0000 A-A7 B-A0 C-94 S-0041 |
| BEQ \$810A   | P-810A Y-FFFF X-0000 A-A7 B-A0 C-94 S-0041 |
| SEC          | P-810B Y-FFFF X-0000 A-A7 B-A0 C-95 S-0041 |
| LDAA #\$25   | P-810D Y-FFFF X-0000 A-25 B-A0 C-91 S-0041 |
| STAA \$0105  | P-8110 Y-FFFF X-0000 A-25 B-A0 C-91 S-0041 |
| ROL \$0105   | P-8113 Y-FFFF X-0000 A-25 B-A0 C-90 S-0041 |
| LDAB #\$4B   | P-8115 Y-FFFF X-0000 A-25 B-4B C-90 S-0041 |
| CMPB \$0105  | P-8118 Y-FFFF X-0000 A-25 B-4B C-94 S-0041 |
| BEQ \$811F   | P-811F Y-FFFF X-0000 A-25 B-4B C-94 S-0041 |
| LDX #\$0100  | P-8122 Y-FFFF X-0100 A-25 B-4B C-90 S-0041 |
| NEG \$05,X   | P-8124 Y-FFFF X-0100 A-25 B-4B C-99 S-0041 |
| LDAA #\$B5   | P-8126 Y-FFFF X-0100 A-B5 B-4B C-99 S-0041 |
| CMPA \$05,X  | P-8128 Y-FFFF X-0100 A-B5 B-4B C-94 S-0041 |
| BEQ \$812F   | P-812F Y-FFFF X-0100 A-B5 B-4B C-94 S-0041 |
| ASR \$0105   | P-8132 Y-FFFF X-0100 A-B5 B-4B C-99 S-0041 |
| LDAA \$05,X  | P-8134 Y-FFFF X-0100 A-DA B-4B C-99 S-0041 |

|                       |  |
|-----------------------|--|
| CMPA # \$DA           | P-8136 Y-FFFF X-0100 A-DA B-4B C-94 S-0041 |
| BEQ \$813D            | P-813D Y-FFFF X-0100 A-DA B-4B C-94 S-0041 |
| LSR \$05,X            | P-813F Y-FFFF X-0100 A-DA B-4B C-90 S-0041 |
| LDAA # \$6D           | P-8141 Y-FFFF X-0100 A-6D B-4B C-90 S-0041 |
| CMPA \$0105           | P-8144 Y-FFFF X-0100 A-6D B-4B C-94 S-0041 |
| BEQ \$814B            | P-814B Y-FFFF X-0100 A-6D B-4B C-94 S-0041 |
| COMA                  | P-814C Y-FFFF X-0100 A-92 B-4B C-99 S-0041 |
| CMPA # \$92           | P-814E Y-FFFF X-0100 A-92 B-4B C-94 S-0041 |
| BEQ \$8155            | P-8155 Y-FFFF X-0100 A-92 B-4B C-94 S-0041 |
| COM \$0105            | P-8158 Y-FFFF X-0100 A-92 B-4B C-99 S-0041 |
| LDAB # \$92           | P-815A Y-FFFF X-0100 A-92 B-92 C-99 S-0041 |
| CMPB \$05,X           | P-815C Y-FFFF X-0100 A-92 B-92 C-94 S-0041 |
| BEQ \$8163            | P-8163 Y-FFFF X-0100 A-92 B-92 C-94 S-0041 |
| EORA # \$F0           | P-8165 Y-FFFF X-0100 A-62 B-92 C-90 S-0041 |
| CMPA # \$62           | P-8167 Y-FFFF X-0100 A-62 B-92 C-94 S-0041 |
| BEQ \$816E            | P-816E Y-FFFF X-0100 A-62 B-92 C-94 S-0041 |
| CLR \$05,X            | P-8170 Y-FFFF X-0100 A-62 B-92 C-94 S-0041 |
| CLRA                  | P-8171 Y-FFFF X-0100 A-00 B-92 C-94 S-0041 |
| CMPA \$0105           | P-8174 Y-FFFF X-0100 A-00 B-92 C-94 S-0041 |
| BEQ \$817B            | P-817B Y-FFFF X-0100 A-00 B-92 C-94 S-0041 |
| BSET \$05,X \$AA      | P-817E Y-FFFF X-0100 A-00 B-92 C-98 S-0041 |
| LDAA # \$AA           | P-8180 Y-FFFF X-0100 A-AA B-92 C-98 S-0041 |
| CMPA \$0105           | P-8183 Y-FFFF X-0100 A-AA B-92 C-94 S-0041 |
| BEQ \$818A            | P-818A Y-FFFF X-0100 A-AA B-92 C-94 S-0041 |
| BCLR \$05,X \$0F      | P-818D Y-FFFF X-0100 A-AA B-92 C-98 S-0041 |
| LDAA # \$A0           | P-818F Y-FFFF X-0100 A-A0 B-92 C-98 S-0041 |
| CMPA \$05,X           | P-8191 Y-FFFF X-0100 A-A0 B-92 C-94 S-0041 |
| BEQ \$8198            | P-8198 Y-FFFF X-0100 A-A0 B-92 C-94 S-0041 |
| LDAA # \$45           | P-819A Y-FFFF X-0100 A-45 B-92 C-90 S-0041 |
| LDAB # \$CD           | P-819C Y-FFFF X-0100 A-45 B-CD C-98 S-0041 |
| PSHA                  | P-819D Y-FFFF X-0100 A-45 B-CD C-98 S-0040 |
| PSHB                  | P-819E Y-FFFF X-0100 A-45 B-CD C-98 S-003F |
| CLRA                  | P-819F Y-FFFF X-0100 A-00 B-CD C-94 S-003F |
| CLRB                  | P-81A0 Y-FFFF X-0100 A-00 B-00 C-94 S-003F |
| PULB                  | P-81A1 Y-FFFF X-0100 A-00 B-CD C-94 S-0040 |
| PULA                  | P-81A2 Y-FFFF X-0100 A-45 B-CD C-94 S-0041 |
| CMPA # \$45           | P-81A4 Y-FFFF X-0100 A-45 B-CD C-94 S-0041 |
| BEQ \$81AB            | P-81AB Y-FFFF X-0100 A-45 B-CD C-94 S-0041 |
| CMPB # \$CD           | P-81AD Y-FFFF X-0100 A-45 B-CD C-94 S-0041 |
| BEQ \$81B4            | P-81B4 Y-FFFF X-0100 A-45 B-CD C-94 S-0041 |
| LDAB # \$15           | P-81B6 Y-FFFF X-0100 A-45 B-15 C-90 S-0041 |
| STAB \$10             | P-81B8 Y-FFFF X-0100 A-45 B-15 C-90 S-0041 |
| BRCL \$10 \$EA \$81C1 | P-81C1 Y-FFFF X-0100 A-45 B-15 C-90 S-0041 |
| BRSE \$10 \$05 \$81CA | P-81CA Y-FFFF X-0100 A-45 B-15 C-90 S-0041 |
| LDAA # \$00           | P-81CC Y-FFFF X-0100 A-00 B-15 C-94 S-0041 |
| LDAB # \$00           | P-81CE Y-FFFF X-0100 A-00 B-00 C-94 S-0041 |
| STOP                  | P-81CF Y-FFFF X-0100 A-00 B-00 C-94 S-0041 |

**RAM Locations after program run:**

```
0000 FF 30 50 FF 0F A0 FF FF FF FF FF FF FF FF FF
0010 15 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0030 10 EE 50 01 00 FF FF E3 74 D0 06 A5 01 00 FF FF
0040 81 D5 FF E4 E4 6D E3 D4 00 E4 6D E3 E4 E4 6D E3

0100 10 A0 05 A5 FF A0 FF FF FF FF FF FF FF FF FF
0110 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0120 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0130 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0140 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0150 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0160 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0170 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0180 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0190 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
01A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
01B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
01C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
01D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
01E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
01F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

## B.2. Serial Port Test Program

```
BASE      EQU    $1000
SCSR      EQU    $102E      ; SERIAL COMMS STATUS REG
SCDR      EQU    $102F      ; SERIAL COMMS DATA REG
SCCR2     EQU    $102D
BAUD      EQU    $102B
DDRD      EQU    $1009

          ORG    $D000
          LDAA  #$FE
          LDAA  #$02
          STAA  DDRD
          LDAA  #$00
          STAA  BAUD
          LDAA  #$0C
          STAA  SCCR2
          LDX   #BASE      ; POINTER TO REGISTER BASE
LOOP:     LDAA  #$21      ; START WITH ASCII 0X21 TO TRANSMIT
XMIT:     STAA  SCDR      ; TRANSMIT REG A
HERE:     BRCLR $2E,X #40 HERE ; LOOP HERE UNTIL BYTE TRANSMITTED
          INCA          ; INC CHAR TO BE TRANSMITTED
          CMPA  #$5B      ; SEE IF IT'S BEYOND ASCII 0X5A
          BNE  XMIT      ; NO, SEND THE NEXT ONE
          BRA   LOOP      ; RESET CHAR TO 0X21
```

# APPENDIX C

## VISUAL SIMULATION TOOL USER GUIDE

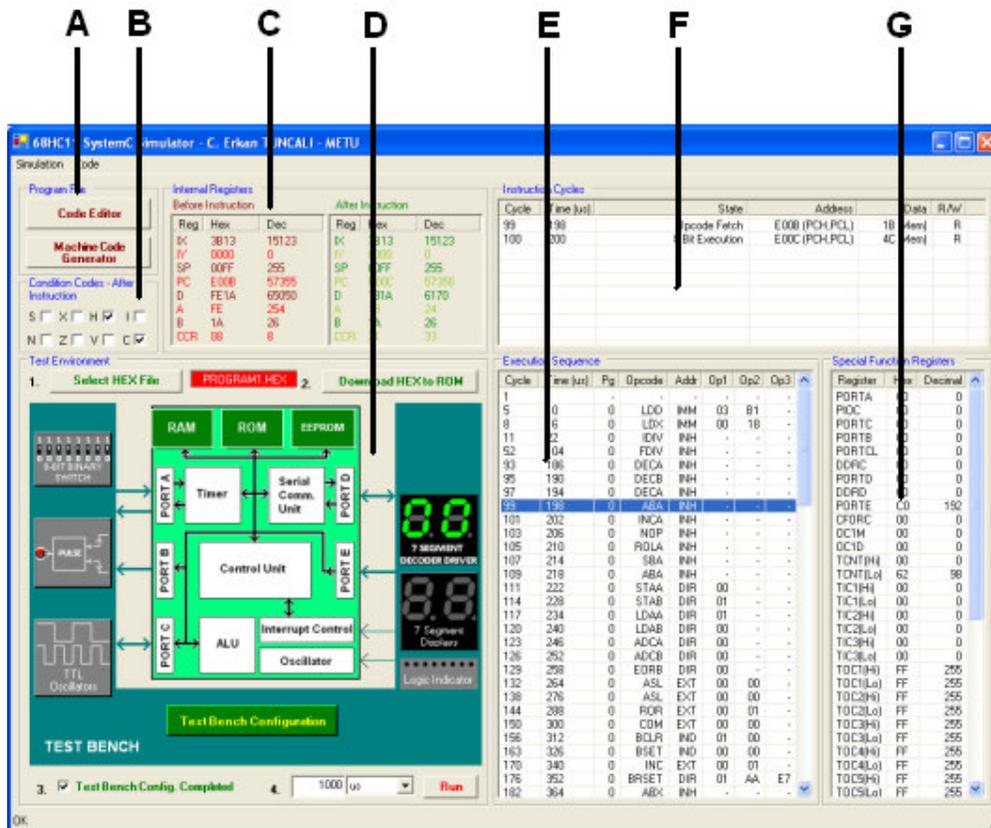


Figure C.1: Main Window of Visual Simulation Software

For proper operation of simulation tool, steps mentioned below should be followed for making simulations.

1. Simulation program can be run by executing sc68hc11.exe file. For executing sc68hc11.exe, .NET framework should have been installed on the system. When simulation program is executed, main window that is shown in Figure C.1 is loaded.
2. Program code can be written in M68HC11 assembly using code editor tool (can be accessed using “Code Editor” button which is in group box labeled as “A” Figure C.1.) or any other text editor. Implemented microcontroller has its ROM located between address locations \$D000 - \$FFFF. Directives in program source code should be given appropriately. An example is shown in Figure C.2 with file name “TESTALL.asm”.

```
TESTALL.ASM - 68HC11 Simulator Code Editor
File Edit
New Ctrl+N
Open... Ctrl+O
Save Ctrl+S
Save As...
ORG $D000
LDAA #$20 ;***ARITHMETIC OPERATIONS***
LDAB #$30
STAB $0001
ADDA $0001 ;DIRECT ADDITION
LDAB #$50
STAB $0002
SUBA $0002 ;DIRECT SUBTRACTION
BEQ OK01
LDAA #$01
JMP FAILED
OK01:
```

Figure C.2: Code Editor Example

3. Assembly source file should be saved in the simulator program directory.
4. Saved assembly source file should be converted to .hex format using machine code generator tool which is located in group box A that is shown in Figure C.1. At first, assembly file should be selected, then it should be compiled into .s19 file using “.asm -> .s19” button. This file can be used for downloading the program into EVBU boards (memory related directives should be compatible with target EVBU board). For simulation, .s19 file should be converted to .hex file using “.s19 -> .hex” button.

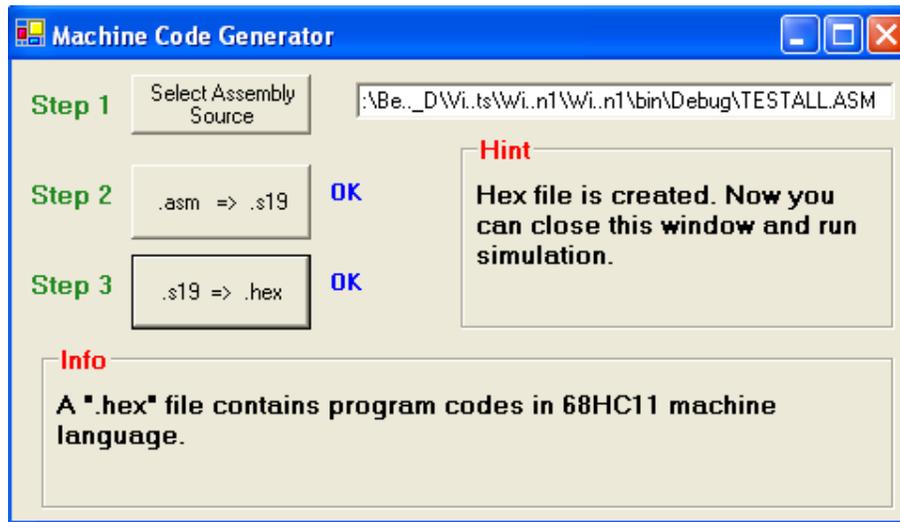


Figure C.3: Code Generator Example

5. After converting source file to .hex format, this program file should be downloaded into microcontrollers ROM using “Select HEX File” and

“Download HEX to ROM” buttons that are labeled as “1” and “2” in Figure C.4. Selected .hex file should be in simulator directory.

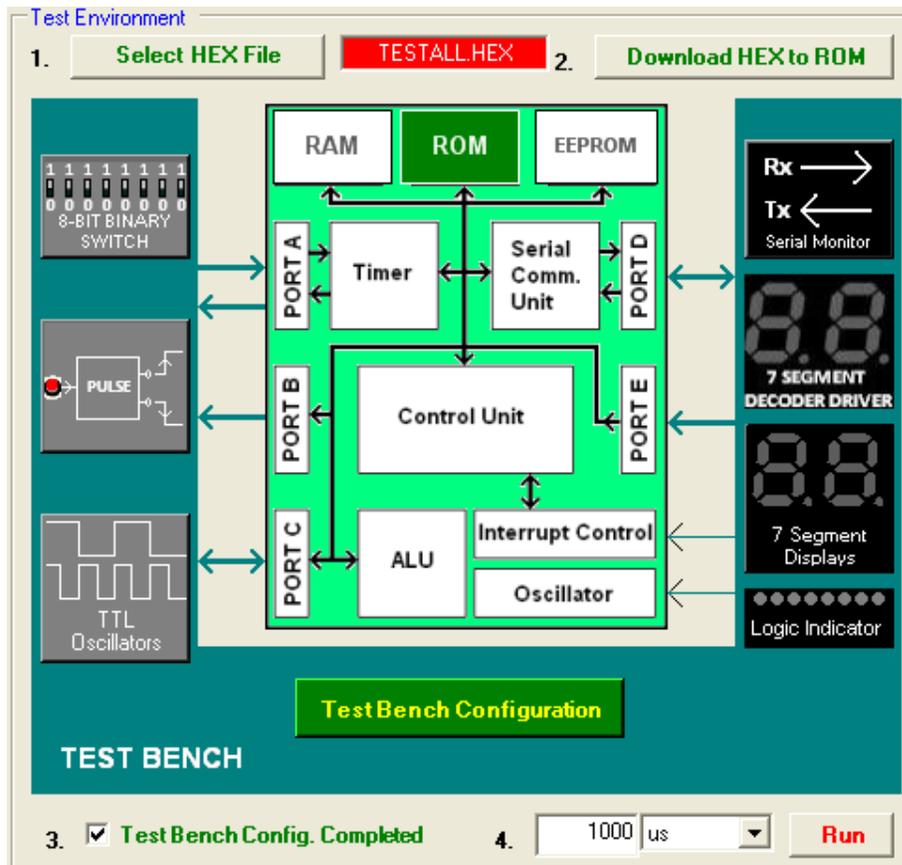


Figure C.4: Test Environment Example

6. After downloading program file into ROM, user should prepare test environment for simulation requirements. Behaviour of test bench devices can be changed by clicking on symbols of the devices. Configuration windows of

binary switch, pulse generator and serial monitor are shown in Figures C.6, C.7 and C.8 respectively. Test bench devices can be enabled / disabled and their connections to microcontroller ports can be changed using “Test Bench Configuration” button that is shown in Figure C.4. An example test bench port configuration window can be found in Figure C.5.

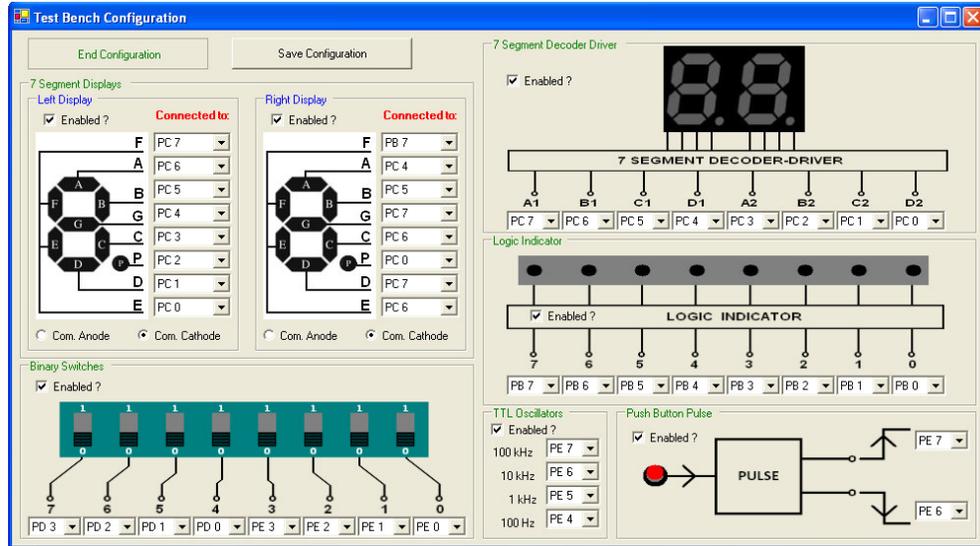


Figure C.5: Test Bench Port Configuration Example

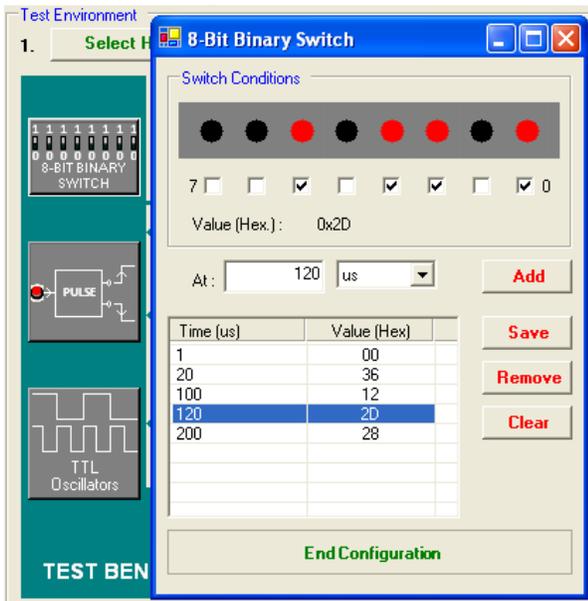


Figure C.6: 8-bit Binary Switch Configuration

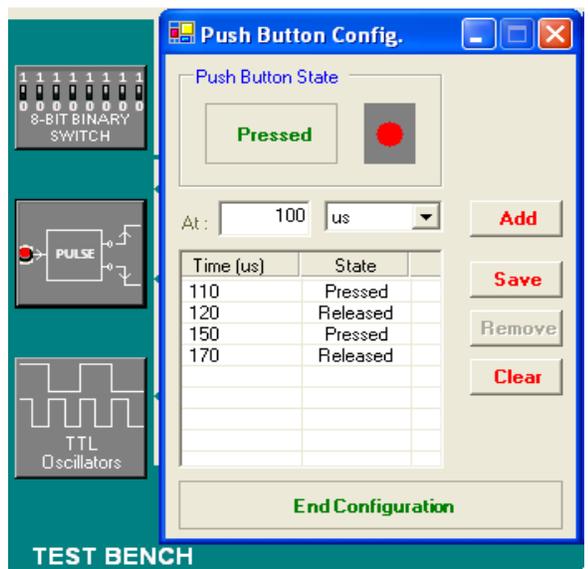


Figure C.7: Push Button Pulse Generator Configuration

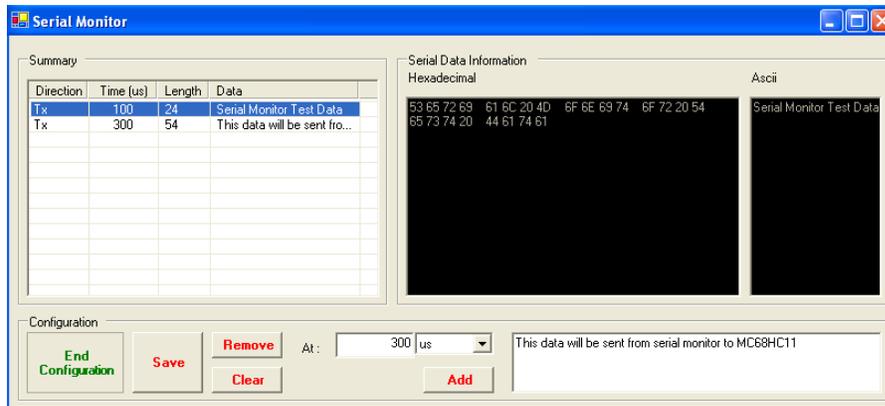


Figure C.8: Serial Monitor Configuration

7. When test bench configuration is completed, “Test Bench Config. Completed” checkbox should be checked (Figure C.4).
8. For running simulation, simulation duration should be selected and “Run” button should be clicked (Figure C.9). User should click on “OK” button on the window that will be opened for informing about simulation duration. This simulator is not a traditional microcontroller simulator but a simulator that runs SystemC implementation of microcontroller and presents internal workings of microcontroller, so simulations with long durations may last very long. This situation should be considered when simulation duration is being selected. Simulation execution speed is about 350 instructions / second on a computer with 1.5 GHz CPU.

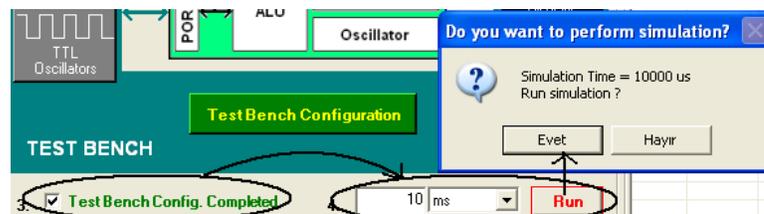


Figure C.9: Running Simulation

9. After running simulation, program brings simulation results. User can get information on executed instruction using listviews “Execution Sequence” and “Instruction Cycles” that are labeled as “E” and “F” respectively in Figure C.1. Information on internal registers are also shown on main window regions that are labeled as “B”, “C” and “G”.
  
10. For viewing RAM or EEPROM locations at a time, a line with desired timing information should be selected from listviews “E” or “F” and “RAM” or “EEPROM” buttons that are shown in Figure C.4. should be clicked.
  
11. Information on test bench modules can be obtained by clicking on module symbols at any time. As an example, simulation information of “serial monitor” module can be accessed using “Read Simulation Results” button that is shown in Figure C.10.

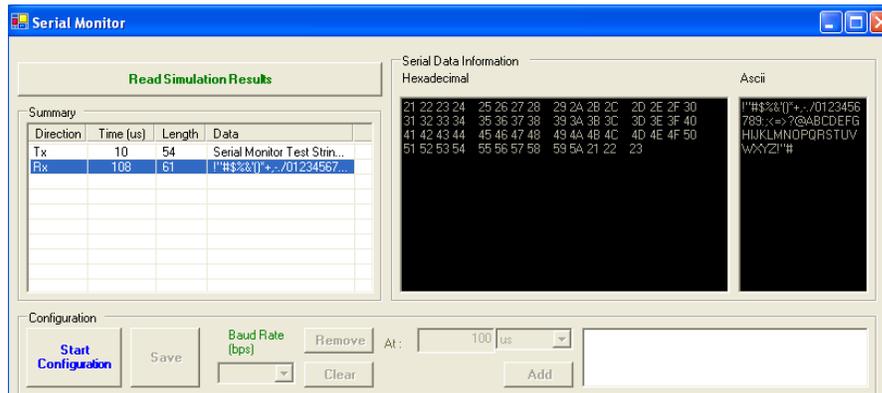


Figure C.10: Simulation Results of Serial Monitor Module