

**Simple-As-Possible Computer System Development on
Field Programmable Gate Array**

by

Lee Siang Yeek

4141

Dissertation submitted in partial fulfilment of the
requirements for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

JUNE 2007

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Simple-As-Possible Computer System Development on Field Programmable Gate Array

by

Lee Siang Yeek
4141

A dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
BACHELOR OF ENGINEERING (Hons)
(ELECTRICAL & ELECTRONICS ENGINEERING)

Approved by,



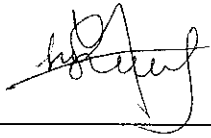
(MR. LO HAI HIUNG)

Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
JUNE 2007

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

A handwritten signature in black ink, appearing to read 'Lee Siang Yee', is written over a horizontal line.

(LEE SIANG YEEK)

ABSTRACT

This report presents the project work and results of the Simple-As-Possible (SAP) computer system development on Field Programmable Gate Array (FPGA) project. This project undertaken as fulfilment of the two semesters EEB5034 & EEB5044 Final Year Project course is aimed to develop the first generation of SAP computer (SAP-1) introduced by Albert Paul Malvino on FPGAs for educational purpose. This includes system level synthesis of SAP-1 computer on a single FPGA chip, as well as modular synthesis of SAP-1 with each SAP-1 functional block on a Complex Programmable Logic Device (CPLD) or FPGA chip.

The objective of this project is to develop SAP-1 computer model for better structured lab practices of the Computer System Architecture course. Implementation of SAP-1 computer is initially suggested by Malvino to be based on TTL logic circuits. FPGA and CPLD are selected instead in this project due to their improved robustness and ease of debugging. The project also serves as introductory practice for understanding of fundamental computer architecture and Verilog Hardware Description Language (HDL) simulation and synthesis of digital systems to the developer.

ACKNOWLEDGEMENT

I would like to express my highest appreciation to a few people who has contributed greatly towards making this project a successful and valuable one. Firstly, I would want to thank the project supervisor, Mr. Lo Hai Hiung for his guidance throughout the development of this project. Thanks Mr. Lo for his kind advice in determining the project development flow and methodology, consultations on learning process of Verilog HDL, HDL simulator, and development software, and valuable suggestions in debugging of the prototype. His effort in reviewing the reports written throughout this project is also highly appreciated.

I would want to thank the technician of the Digital System Design Lab, Mr. Badrulnizam for his coordination in leasing of Altera's University Program 2 (UP2) development platforms for project prototype development purpose. I would want to thank Printed Circuit Board (PCB) Lab technician, Mr. Isnani too for his assistance in PCB development for organized interconnectivity of UP2 platforms used in modular synthesis of SAP-1 computer.

Last but not least, I would also want to thank all lecturers whose name is not mentioned here, yet have contributed their opinion or suggestion valuable to development of this project. Thanks to all technicians for their kind cooperation especially in leasing of components and equipment for project development purpose.

TABLE OF CONTENT

CERTIFICATIONS.....	ii
ABSTRACT.....	iv
ACKNOWLEDGEMENT.....	v
CHAPTER 1 INTRODUCTION.....	1
1.1 Project Background.....	1
1.2 Problem Statement.....	2
1.3 Objective & Scope of Study.....	3
CHAPTER 2 LITERATURE REVIEW.....	4
2.1 SAP-1 Architecture.....	4
2.1.1 <i>W Bus</i>	4
2.1.2 <i>Program Counter (Control Unit)</i>	4
2.1.3 <i>Memory Address Register & Address Input</i> <i>(Memory Unit / Input Unit)</i>	4
2.1.4 <i>Random Access Memory & Instruction/Data Input</i> <i>(Memory Unit / Input Unit)</i>	5
2.1.5 <i>Instruction Register (Control Unit)</i>	5
2.1.6 <i>Controller/Sequencer (Control Unit)</i>	5
2.1.7 <i>Mode-Select Switches, De-bouncers, and Clock Buffer</i> <i>(Input Unit / Control Unit)</i>	5
2.1.8 <i>Accumulator (Arithmetic Logic Unit)</i>	5
2.1.9 <i>Adder/Subtractor (Arithmetic Logic Unit)</i>	5
2.1.10 <i>B Register (Arithmetic Logic Unit)</i>	6
2.1.11 <i>Output Register & Binary Display (Output Unit)</i>	6
2.2 SAP-1 Instruction Set.....	6
2.3 SAP-1 Programming.....	6
2.4 SAP-1 Machine Cycle & Instruction Cycle.....	7

CHAPTER 3	METHODOLOGY & PROJECT WORK.....	8
3.1	Literature Study of Digital Electronics Fundamentals and Applications.....	9
3.2	Literature Study of SAP-1 Computer System.....	9
3.3	Literature Study of Verilog HDL for Synthesis of Digital Systems.....	9
3.4	Simulation of SAP-1 Computer System.....	10
3.5	System Synthesis of SAP-1 Computer System.....	11
3.6	Modular Synthesis of SAP-1 Computer System.....	16
CHAPTER 4	RESULTS & DISCUSSIONS.....	19
4.1	Simulation of SAP-1 Computer System.....	19
4.1.1	<i>Program Counter</i>	19
4.1.2	<i>MAR</i>	20
4.1.3	<i>2 to 1 Multiplexer</i>	21
4.1.4	<i>16 × 8 RAM</i>	21
4.1.5	<i>Instruction Register</i>	22
4.1.6	<i>Accumulator</i>	22
4.1.7	<i>Adder/Subtractor</i>	23
4.1.8	<i>B Register</i>	24
4.1.9	<i>Output Register</i>	25
4.1.10	<i>Controller/Sequencer (Instruction Decoder, Ring Counter & Control Matrix)</i>	25
4.1.11	<i>Mode-Select Switches, De-bouncers & Clock Buffer</i>	27
4.1.12	<i>System Level Simulation of SAP-1 Computer</i>	29
4.1.13	<i>Overall Findings</i>	35
4.2	System Synthesis of SAP-1 Computer System.....	35
4.3	Modular Synthesis of SAP-1 Computer System.....	40
CHAPTER 5	CONCLUSION & RECOMMENDATION.....	46
5.1	Conclusion.....	46
5.2	Recommendation.....	47
REFERENCES.....		48

APPENDIX A	MODIFIED BLOCK DIAGRAM OF SAP-1.....	50
APPENDIX B	TRUTH TABLE OF BINARY TO HEXADECIMAL	
	7-SEGMENT DISPLAY DECODER MODULE.....	51
APPENDIX C	ALTERA'S UNIVERSITY PROGRAM 2 DEVELOPMENT	
	PLATFROM COMPONENT LAYOUT.....	52
APPENDIX D	SAP-1 SYSTEM SYNTHESIS SOURCE CODE.....	53
D-1	Program Counter.....	53
D-2	MAR & 2 to 1 Multiplexer.....	54
D-3	16 × 8 RAM.....	55
D-4	Instruction Register.....	56
D-5	Accumulator.....	57
D-6	Adder/Subtractor.....	58
D-7	B Register.....	59
D-8	Output Register.....	60
D-9	Controller/Sequencer (Instruction Decoder, Ring Counter & Control Matrix.....	61
D-10	Mode-Select Switches, De-bouncers & Clock Buffer.....	63
D-11	SAP-1.....	65
D-12	Additional Hexadecimal Display of MAR & 2 to 1 MUX Output on MAX 7000S Device.....	66
APPENDIX E	SAP-1 MODULAR SYNTHESIS SOURCE CODE.....	68
E-1	Program Counter.....	68
E-2	MAR & 2 to 1 Multiplexer.....	69
E-3	16 × 8 RAM.....	71
E-4	Instruction Register.....	73
E-5	Accumulator.....	75
E-6	Adder/Subtractor.....	76
E-7	B Register.....	78
E-8	Output Register.....	79

E-9	Controller/Sequencer (Instruction Decoder, Ring Counter & Control Matrix.....	80
E-10	Mode-Select Switches, De-bouncers & Clock Buffer.....	83

APPENDIX F PIN ASSIGNMENTS, PIN INTERCONNECTIONS, AND INPUT & OUTPUT DEVICE UTILIZATION OF THE MODULAR SAP-1 PROTOTYPE.....86

F-1	Program Counter.....	86
F-2	MAR & 2 to 1 Multiplexer.....	87
F-3	16 × 8 RAM.....	88
F-4	Instruction Register.....	89
F-5	Accumulator.....	90
F-6	Adder/Subtractor.....	91
F-7	B Register.....	92
F-8	Output Register.....	93
F-9	Controller/Sequencer (Instruction Decoder, Ring Counter & Control Matrix.....	94
F-10	Mode-Select Switches, De-bouncers & Clock Buffer.....	95

APPENDIX G PHOTOS OF MODULAR SAP-1 PROTOTYPE..... 96

LIST OF FIGURES

FIGURE 1	Project development flowchart.....	8
FIGURE 2	Simulation results of the Program Counter module.....	20
FIGURE 3	Simulation results of the MAR module.....	20
FIGURE 4	Simulation results of the 2 to 1 Multiplexer module.....	21
FIGURE 5	Simulation results of the 16×8 RAM module.....	22
FIGURE 6	Simulation results of the Adder/Subtractor module for addition operation.....	24
FIGURE 7	Simulation results of the Adder/Subtractor module for subtraction operation.....	24
FIGURE 8	Simulation results of the B Register module.....	24
FIGURE 9	Simulation results of the Output Register module.....	25
FIGURE 10	Simulation results of the Controller/Sequencer module.....	26
FIGURE 11	Input and output waveforms of simulation of the Controller/ Sequencer module at the earliest 2 instruction cycles.....	27
FIGURE 12	Simulation results of the Mode-Select Switches, De-bouncers & Clock Buffer module.....	28
FIGURE 13	Input and output waveforms of SAP-1 system level simulation in RAM-programming mode.....	30
FIGURE 14	Input and output waveforms of SAP-1 system level simulation for execution of the ADD and SUB routine.....	31
FIGURE 15	Final results of SAP-1 system level simulation.....	34
FIGURE 16	Schematic diagram of the <i>LOW_HIGH</i> input switches implementation.....	38
FIGURE 17	Modified block diagram of SAP-1 redrawn from chapter 10 of [1] <i>Digital Computer Electronics</i> by Malvino (1983).....	50
FIGURE 18	Component layout of Altera's University Program 2 Development Platform from page 3 of [13] " <i>University Program UP2 Education Kit User Guide v3.1</i> ", Altera Corporation, www.altera.com/ literature/univ/upds.pdf	52
FIGURE 19	Modular SAP-1 prototype, picture 1.....	96
FIGURE 20	Modular SAP-1 prototype, picture 2.....	96

LIST OF TABLES

TABLE 1	SAP-1 Instruction Set Summary from chapter 10 of [1] <i>Digital Computer Electronics</i> by Malvino (1983).....	6
TABLE 2	UP2 platform input & output devices and driven or driving signals in SAP-1 system synthesis.....	14
TABLE 3	Active control signals for each routine at every T state obtained from simulation results of the Controller/Sequencer module.....	26
TABLE 4	Program used for verification of SAP-1 system level simulation.....	29
TABLE 5	Summary of internal operations of the ADD routine in test Program of SAP-1 system simulation.....	32
TABLE 6	Detailed observations of test program execution on SAP-1 system synthesis prototype in manual-clocked mode.....	39
TABLE 7	Detailed observations of test program execution on modular SAP-1 prototype in manual-clocked mode.....	44
TABLE 8	Truth table of binary to hexadecimal 7-segment display decoder module for common anode, active low 7 segment display on Altera's UP 2 Development Platform.....	51

CHAPTER 1

INTRODUCTION

1.1 Background of Study

Simple-As-Possible (SAP) computer system is an educational computer system introduced by Albert Paul Malvino in his book, *Digital Computer Electronics* published in 1983. This computer system introduces only the most crucial ideas behind computer operation. Though, these ideas make up valuable fundamental for understanding of many modern and more complex computer architectures. SAP comes in three generations with increasing number of functional blocks and operational complexity, namely SAP-1, SAP-2, and SAP-3. SAP computer system has been implemented in universities using TTL logic circuits as proposed by Malvino's original design.

Field Programmable Gate Arrays (FPGA) are programmable logic chips utilizing large scale integration technology [2]. It is made up of basic logic components that can be programmed for certain logical behavior. These programmed blocks can then be linked to generate more complex logic system [3]. This technology has replaced the usage of Application Specific Integrated Circuit (ASIC) for design and prototyping of Integrated Circuits (IC) [4]. Utilization of FGPA has tremendously increased efficiency of IC design and prototyping in terms of cost and time.

Complex Programmable Logic Devices (CPLD) which can be traced back as historical root of invention of FPGA [5], is another type of programmable IC with different architecture. It is made up of fully programmable AND and OR gates array for logical functions and a macrocells bank performing combinatorial or sequential logic [6]. CPLD typically has less logical elements than FPGA [7]. Another

noticeable difference between FPGA and CPLD is the presence of on-chip non-volatile memory in the CPLD [8].

Traditional circuit design uses schematic to describe circuits for simulation. As new circuits increase greatly in complexity, this approach has become impractical. Development of Hardware Description Language (HDL) has evolved to solve this problem. HDL uses textual description representation of electronic circuits and systems [9]. Verilog is one of the most widely used HDL, with another being VHDL. Having its syntax being similar to 'C' language, Verilog is preferred by most commercial designers [10]. Verilog is first designed for circuit simulation [9]. Latter efforts have made Verilog capable for synthesis of circuit on chips too. Despite the limitations on circuit synthesis with Verilog, as in not all statements implemented in simulation are synthesizable, Verilog is still a powerful HDL for synthesis of digital systems. Verilog allows various specification levels and styles.

1.2 Problem Statement

This project is aimed to develop first generation of SAP computers (SAP-1) on FPGA for educational purpose. Verilog HDL will be used to capture each functional block of SAP system for simulation and synthesis purpose.

The developed prototype will be utilized in lab sessions of EEB5253 Computer System Architecture course. Previously, lab practices would require the students to construct their circuit for experimental purpose based on TTL logic circuits. Maintaining a working Verilog program, the circuit can be reprogrammed or duplicated whenever circuit failure occurs. This is far more efficient than debugging complicated TTL circuits as error is encountered. Therefore, students of the course can concentrate better on learning of computer system architecture rather than spending time debugging TTL circuits during the lab session.

Meanwhile, FPGA also features higher robustness compared to TTL circuits. TTL circuits implementation of SAP-1 computer suggested by Malvino requires more than 50 TTL chips with messy wirings. Failure of a single TTL chip or connection of

a single wire may lead to malfunction of the entire system. This justifies the decision to implement SAP computer system on FPGA rather than TTL circuits. Development of this project also provides an alternative of learning computer system architecture through Verilog HDL and FPGA besides construction of TTL logic circuits.

1.3 Objective & Scope of Study

This project is aimed to develop hardware required for better structured lab experiments of the Computer System Architecture course. This includes system level synthesis of SAP-1 computer on a single University Program 2 (UP2) development platform by Altera, as well as modular synthesis of SAP-1 computer utilizing a UP2 platform for each functional block. Presenting only the basic components of vital importance, while eliminating unnecessary details at the same time, SAP computers promote easy understanding of computer operation through the lab session [1].

From the project developer's perspective, this project enables me to gain understanding on fundamentals of computer system. Although SAP computers present a simple design for very basic computer operation, it forms valuable fundamental for understanding of modern and more complex computer systems to be encountered in the future. Apart from that, the project also provides me good practice for picking up knowledge on Verilog HDL. Implementation of the computer system also exposes me to FPGA technology. In short, this project is a value-added activity for the participant.

Schematics and detailed description of all SAP generations are available in Albert Paul Malvino's book, [1] *Digital Computer Electronics* (1983). Meanwhile, reference source of Verilog HDL is widely available. The project supervisor, Mr. Lo Hai Hiung is a very knowledgeable person on Verilog for valuable consultancy. The university also provides good facility for digital system design on FPGA. Hence with proper project scheduling, this project has high feasibility.

CHAPTER 2

LITERATURE REVIEW

This chapter provides a high level architectural overview of SAP-1. As SAP computer is introduced by Albert Paul Malvino, information in this chapter is cited from his publication, [1] *Digital Computer Electronics* (1983). Kindly refer to the book for further details.

2.1 SAP-1 Architecture

Brief description of each functional block of SAP-1 is provided below. Kindly refer to Appendix A for modified block diagram of SAP-1 computer system.

2.1.1 *W Bus*

The W Bus is a three-state 8-bit wide bus interconnecting various SAP-1 modules, allowing orderly transfer of data [1].

2.1.2 *Program Counter (Control Unit)*

The program counter is essentially a ripple counter. It is incremented for every instruction cycle to point to the memory location of the next instruction to be fetched and executed [1].

2.1.3 *Memory Address Register & Address Input (Memory Unit / Input Unit)*

The Memory Address Register (MAR) stores memory address coming from the Program Counter and Instruction Register for access of instruction or data stored in the Random Access Memory. Switches are utilized for memory location selection during programming stage [1].

2.1.4 Random Access Memory & Instruction/Data Input (Memory Unit / Input Unit)

The 16 8-bit-words Random Access Memory (RAM) stores program and data of SAP-1. 8-bit switches are attached to this RAM for instruction/data inputting purpose at the programming stage [1].

2.1.5 Instruction Register (Control Unit)

The Instruction Register is used to separate SAP-1 instructions into the instruction field (upper nibble) and data location field (lower nibble) and subsequently outputs to the Controller/Sequencer and MAR respectively [1].

2.1.6 Controller/Sequencer (Control Unit)

This block decodes the instruction fetched by the Instruction Register and outputs a 12-bit control word that coordinates operation of each functional block in every T state [1].

2.1.7 Mode-Select Switches, De-bouncers, and Clock Buffer (Input Unit / Control Unit)

Mode-Select Switches is made up of the START'/CLEAR switch for running or stopping of program execution and, LOW'/HIGH and MANUAL'/AUTO switches facilitating the clocking mode of SAP-1 computer. CLK and CLR signals generated in this block are fed into the other SAP-1 modules [1].

2.1.8 Accumulator (Arithmetic Logic Unit)

This 8-bit register stores intermediate value and final result of arithmetic operations. The value stored in the Accumulator is sent to the Output Register and Binary Display when the "OUT" routine is executed [1].

2.1.9 Adder/Subtractor (Arithmetic Logic Unit)

A 2's complement Adder/Subtractor is used in SAP-1. For subtraction operation, a high S_U signal is sent to the Adder/Subtractor to convert one of the operand (stored in B Register) into 2's complement form [1].

2.1.10 B Register (Arithmetic Logic Unit)

The B register is another buffer register used in arithmetic operation. It holds the number to be added to or subtracted from the number stored in the Accumulator [1].

2.1.11 Output Register & Binary Display (Output Unit)

When the “OUT” instruction is executed, number stored in the Accumulator is sent to the Output Register to trigger 8 Light-Emitting Diodes (LEDs) Binary Display [1].

2.2 SAP-1 Instruction Set

SAP-1’s instruction set consists of only 5 instructions. These instructions and the corresponding operation are summarized in TABLE 1.

TABLE 1: SAP-1 instruction set summary from chapter 10 of [1] *Digital Computer Electronics* by Malvino (1983)

Mnemonic	Op Code	Operation
LDA	0000	Load RAM data into Accumulator
ADD	0001	Load RAM data to B Register, and add B Register data to Accumulator
SUB	0010	Load RAM data to B Register, and 2’s complement subtract B Register data from Accumulator
OUT	1110	Load Accumulator data into Output Register
HLT	1111	Stop processing

2.3 SAP-1 Programming

SAP-1 program is stored in lower RAM location (from 0x0) whereas the data are stored in the higher locations. The 8-bit instruction consists of the upper nibble operation type and the lower nibble operand [1]. This is demonstrated in the following example presented in both assembly language (left) and machine language (right):-

Address	Instruction/Data	Address	Instruction/Data
0x0	LDA 0x9	0000	0000 1001
0x9	0x11	1001	0001 0001

The example shows that instruction which essentially loads the accumulator with the content of location 0x9 of the RAM is stored at memory location 0x0 (lower location). The data involved in this operation is stored in a higher RAM location (0x9). When presented in machine language, the op-code and data are given in binary representation. We see that op-code of the “LDA” operation (0000 binary) and memory location storing the operand occupies the upper and lower nibble of the instruction respectively [1].

2.4 SAP-1 Machine Cycle & Instruction Cycle

Each T state of SAP-1 is characterized by each clock cycle, starting and ending with a falling clock edge. As SAP-1 is positive-edge-triggered, this selection makes all clocked operation to occur midway through each T state. This gives an allowance of half a cycle time for setup time, hold time, and steady state setup time for the signal being present at the W Bus [1].

The first three T states are named the Fetch Cycle. It generally involves accessing of memory location of instruction pointed to by Program Counter, incrementing the Program Counter, and fetching the instruction in the RAM to the Instruction Register [1]. The latter three T states make up the Execution Cycle. Operation taking place in the system during Fetch Cycle is generally common for all instructions but differs among instructions in the Execution Cycle [1].

6 T states (T_1 through T_6) makes up a machine cycle for SAP-1. The number of T states needed to fetch and execute an instruction defines an instruction cycle. SAP-1 has fixed instruction cycle of 6 T states which equals the machine cycle [1].

CHAPTER 3

METHODOLOGY & PROJECT WORK

The project work carried throughout this Final Year Project course is presented in this section. Shown below is the development flowchart of this project:-

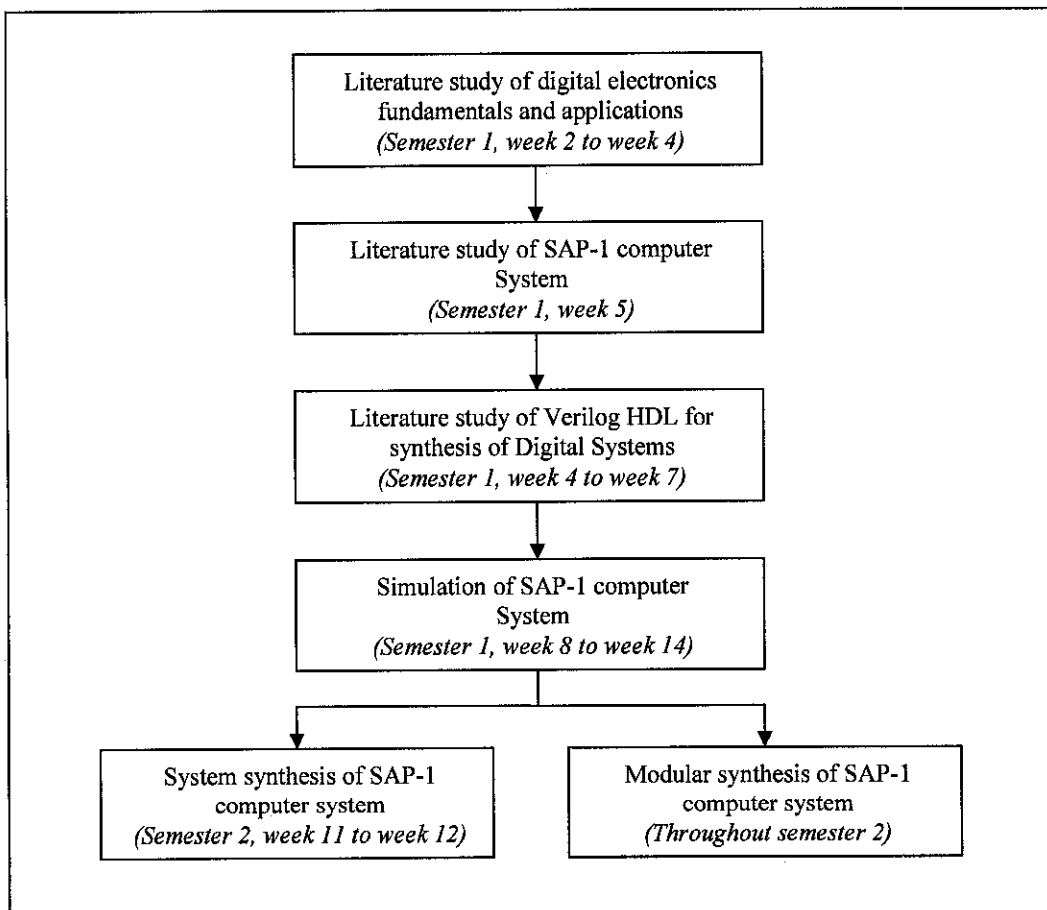


FIGURE 1: Project development flowchart

Details of each project activity will be discussed throughout this chapter. Results obtained from these activities will be discussed in the next chapter of this report.

3.1 Literature Study of Digital Electronics Fundamental and Applications

This is the first activity of the project carried out from Week 2 to Week 4 of Semester 1. The study is carried out based on the first nine chapters of [1] *Digital Computer Electronics* by Malvino (1983). The early chapters cover the digital electronics fundamentals which are recap of the Digital Electronics I course. Subsequently, various applications of the fundamental logic components such as the adder-subtractor, flip-flops, registers, counters, and memory are demonstrated. Along the descriptions in these latter chapters, sub-modules of SAP-1 have been introduced as appropriate. This ensures good understanding during literature study of SAP-1.

3.2 Literature Study of SAP-1 Computer System

Literature study of SAP-1 is done based on chapter 10 of [1] *Digital Computer Electronics* by Malvino (1983) in Week 5 of the Semester 1. In this chapter, how small digital circuits introduced previously are combined for a simple computer system has been seen. Details of this activity have been discussed in Chapter 2 of this report. It is of vital importance that thorough understanding is gained at this stage for correct Verilog HDL capturing of the computer system during latter simulation and synthesis of the system on FPGA.

3.3 Literature Study of Verilog HDL for Synthesis of Digital Systems

This activity has been performed from Week 4 to Week 7 of the Semester 1. [10] *Verilog Styles for Synthesis of Digital Systems* by Smith & Franzon (2000) and [11] *Verilog Coding for Logic Synthesis* by W. F. Lee is the main reference materials used. Meanwhile, lecture notes of the Digital System Design course have also been used as additional reference. Topics gone through are as follows:-

- **Basic Language Constructs** on preliminaries, data types, and modules of Verilog HDL.
- **Structural vs. Behavioural Specification** that details in the writing styles of both specification types.

- **Procedural Specification** that introduces the ‘always’ block, ‘if’ statement, ‘case’ statement, and various looping statements having higher expressive power for behavioural specification style.
- **Design Approaches for Single Modules** on recommended design steps and strategies for single modules in Verilog HDL.
- **Validation of Single Modules** that discusses the element of good Verilog HDL modules testing such as good testbench coding, proper test coverage, and multiple test vector sources.

It is worth noting that Verilog HDL is initially designed for simulation thus not all statements are synthesizable. Due to that, special attention has been paid to identify statements that can only be used for simulation. These statements will be avoided in simulation of SAP-1 system so that code verified through simulation can be applied straight for synthesis purpose.

3.4 Simulation of SAP-1 Computer System

Simulation of SAP-1 has been started during Week 8 of Semester 1. ModelSim Xilinx Edition (XE) III / Starter 6.0d has been used as the simulator. Early attempt of this activity involves familiarization of the software by going through tutorials of the software. Simulation is then started by capturing SAP-1 modules in Verilog HDL and writing testbench for each module. Verilog coding used is ensured to be synthesizable so that validated modules can be used for synthesis without any need for modification. Testbench is written so that the simulation reflects actual operation of the module being tested in SAP-1 system. For example, the C_P control signal that triggers an increment in the Program Counter goes high at every T_2 state [1]. The testbench is programmed to have this behaviour as well.

As Malvino’s design uses TTL chips, behaviour of the control, clock, and clear signal are designed to comply with the TTL chips’ requirements (whether they are active high/low, positive/negative-edge-triggered). However, this is not necessary as SAP is to be implemented on FPGA. Appropriate simplifications as listed below

have been made during simulation while maintaining the general behaviour of the modules in the system:-

- All negative-edge-triggered flip-flops that accept inverted clock signal can be simplified to positive-edge-triggered ones which accept non-inverted clock signal to eliminate inverted clock from the system.
- Clear inputs of all blocks are active high.
- Inputs in each module that accept control signals from the Controller-Sequencer block are active high. In other words, control signals go high when it is active.

These simplifications may reduce mistakes in the Verilog code. However, there might be other concerns (e.g. power efficiency) behind Malvino's original design. Hence any modification made unto the original design is documented well. This will serve as possible parameters that require correction in case of error when SAP-1 system is synthesized on FPGA.

Apart from the simplifications mentioned above, the Verilog program design is kept similar to that of the original SAP-1 computer during the simulation stage. In making sure that this fundamental program works, other simplifications or extra features can easily be added to the system as necessary at the synthesis stage with ease of debugging. Any error encountered that time can easily be traced to the modifications made. We shall discuss the enhancements introduced to SAP-1 computer in detail in the next part of this chapter.

A system level simulation has also been done combining Verilog programs of all modules. Most signals that are essentially "wire" have been declared as "output" in the Verilog program of SAP-1. This is done to enable observation of waveform of these signals during simulation. System level simulation of SAP-1 has been completed towards the end of Semester 1.

3.5 System Synthesis of SAP-1 Computer System

SAP-1 system synthesis is aimed to implement all SAP-1 modules on a single Altera's University Program 2 (UP2) development platform. To understand the features available on the UP2 platform, the document [13] *University Program UP2*

Education Kit User Guide v3.1 has been downloaded from Altera's website. This document provides sufficient information for usage of the development platform in prototype development.

The Quartus II 6.1 Web Edition software has also been acquired from Altera's website. To obtain an overview on the features available in this software, the document [12] *Introduction to Quartus II - Version 6.1* available from Altera's website has been gone through. The Quartus II is a powerful all-in-one tool for project development using Altera's FPGA or CPLD products. It supports HDL & Schematic Design Entry, Analysis & Synthesis, Place & Route, Timing Analysis, Simulation, and Configuration & Programming of the selected target device [12].

The interactive tutorial of the Quartus II attached to the software is also valuable in getting familiarized with the user interface of the software. Subsequently, the project development tools relevant to this project are identified, including HDL Design Entry, Analysis & Synthesis, Pin Assignment & Fitting, and Configuration & Programming. Simulation is not carried out as the Verilog programs used in synthesis have been verified in the simulation stage earlier.

After getting familiarized with the hardware and software to be utilized in the synthesis stage, SAP-1 system synthesis has been started by setting up project targeted for the FLEX 10K device on the UP2 platform in the Quartus II software. Some simplifications and additional features have been introduced as appropriate to the Verilog programs developed in the simulation stage. Each "if" statement in the Verilog programs is paired with an "else" statement to avoid inferred latch that may introduce timing analysis issues during synthesis.

Another simplification has been implemented on the De-bouncers block. Referring to the design of the Mode-Select Switches and De-bouncers in page 159 of [1] *Digital Computer Electronics* by Malvino (1983), single pole double throw (SPDT) switches are used at the latches' inputs. In order to utilize the dual in-line package (DIP) switches available on the UP2 platform, this design can be modified so that the upper input of each latch is driven by an ON/OFF switch. The lower input of the latches will be tied to their respective ON/OFF switch through an inverter, always

maintaining inputs of opposite logic level at the latches. This modification also reduces the number of input signals required in the module by 3 inputs.

Meanwhile, the 7-segment displays available on the UP2 platform can be utilized as hexadecimal display for output of SAP-1 computer. This improves the readability of the computational result originally driving eight Light Emitting Diodes (LEDs) as binary display. Thus an additional binary number to hexadecimal 7-segment display decoder module has been written and being instantiated by the Verilog program of the Output Register module. The truth table of this module is provided in Appendix B. Note that this is a 4-bit to 8-bit (including decimal point signal) decoder module written for common-anode active-low display unit. As content of the Output Register consists of 2 hexadecimal digits, 2 instantiations of the binary to hexadecimal 7-segment display module is made in the Output Register module. The result is then concatenated to form a 16-bit decoded signal, *OUT_REG_HEX*. Note that signals in Verilog programs of SAP-1 are presented in *Italic* throughout this report (e.g. *OUT_REG_HEX*).

Upon proper consideration and design, the LEDs available on the UP2 platform can be utilized as additional indicators so that functionality of the prototype will be easily understandable. Table 2 summarizes the input and display devices on the UP2 platform used in SAP-1 system synthesis and their corresponding driven or driving signals. Kindly refer to Appendix C for board layout and components naming of the UP2 development platform. Note the difference between naming format of individual switch of DIP switches and the numbering used for referencing throughout this report. Individual switch of DIP switches are represented following the naming convention of arrays in Verilog, e.g. *FLEX_SWITCH[1]* and *MAX_SW[5]*. Numbering of references is presented in boldface to make the difference clear.

As LEDs available on UP2 platform (D1 through D16) are active low [13], additional complemented signals of *T*, *LDA*, *ADD*, *SUB*, *OUT*, *HLT*, *CLK*, and *CLR* are generated for display purpose. *MAX_SW1*, *MAX_SW2*, and the LEDs are connected to the input/output (IO) pins of the FLEX 10K device using wires through wire wraps soldered at the expansion holes of those pins. External wiring is not

TABLE 2: UP2 platform input & output devices and driven or driving signals in SAP-1 system synthesis

Signal Type	Component	Signal	Description
Input	FLEX_SWITCH[1] FLEX_SWITCH[2] FLEX_SWITCH[3] FLEX_SWITCH[4] FLEX_SWITCH[5] FLEX_SWITCH[6] FLEX_SWITCH[7] FLEX_SWITCH[8]	<i>DATA_IN[7]</i> <i>DATA_IN[6]</i> <i>DATA_IN[5]</i> <i>DATA_IN[4]</i> <i>DATA_IN[3]</i> <i>DATA_IN[2]</i> <i>DATA_IN[1]</i> <i>DATA_IN[0]</i>	Instruction/Data input during RAM programming stage. Active when both <i>RUN_PROG</i> and <i>READ_WRITE</i> inputs are at LOW logic
Input	MAX_SW1[5] MAX_SW1[6] MAX_SW1[7] MAX_SW1[8]	<i>ADDR_IN[3]</i> <i>ADDR_IN[2]</i> <i>ADDR_IN[1]</i> <i>ADDR_IN[0]</i>	Address input for programming of RAM. Active when <i>RUN_PROG</i> = 0
Input	MAX_SW2[1] MAX_SW2[2] MAX_SW2[3]	<i>START_CLEAR</i> <i>MANUAL_AUTO</i> <i>LOW_HIGH</i>	SAP-1 mode-select inputs. The first parameter is active low, e.g., SAP-1 is running in manual clock mode with low clock signal when all 3 inputs = 0.
Input	MAX_SW2[8] FLEX_PB1	<i>RUN_PROG</i> <i>READ_WRITE</i>	16 × 8 RAM mode-select inputs The first parameter is active high, e.g., the RAM is in running and reading mode when both inputs = 1.
Output	D1 D2 D3 D5 D6 D7	<i>Tnot[1]</i> <i>Tnot[2]</i> <i>Tnot[3]</i> <i>Tnot[4]</i> <i>Tnot[5]</i> <i>Tnot[6]</i>	Complemented T state signals for active low LEDs
Output	D9 D10 D11 D12 D13	<i>LDAnot</i> <i>ADDnot</i> <i>SUBnot</i> <i>OUTnot</i> <i>HLTnot</i>	Complemented decoded instruction signals for active low LEDs
Output	D15 D16	<i>CLKnot</i> <i>CLRnot</i>	Complemented <i>CLK</i> and <i>CLR</i> signals for active low LEDs
Output	FLEX_DIGIT	<i>DIGIT_DISPLAY</i>	Decoded signal of Output Register content (<i>OUT_REG</i>) in RUN mode and RAM content (<i>RAM_DISPLAY</i>) in PROG mode. *
Output	MAX_DIGIT	<i>MUX_OUT_HEX</i>	Decoded signal of 2 to 1 MUX's output (<i>MUX_OUT</i>) for hexadecimal display *

* Please read further for more detailed explanation.

necessary for the remaining components used as they are routed permanently to fixed IO pins of the FLEX 10K device in the printed circuit board (PCB).

In order to take full advantages of the resources available on the UP2 development platform, the MAX_DIGIT device is designed for display of output of the Memory Address Register (MAR) & 2 to 1 Multiplexer (MUX) module. This enables the user of the prototype to observe the location of RAM being accessed during SAP-1 program execution. As the MAX_DIGIT display is routed to the fixed IO pins of the MAX 7000S device, which are not accessible through the expansion holes, it has been decided that decoding for the hexadecimal display is to be implemented on the MAX 7000S device. This reduces number of external wirings required as the 4-bit output signal of the MAR and 2 to 1 MUX module (*MUX_OUT*) is passed from the FLEX 10K device to the MAX 7000S device instead of the 8-bit decoded hexadecimal display signal. This connection is made using insulation displacement connectors (IDC) with ribbon cable through pin headers soldered at expansion holes of both FLEX 10K device and MAX 7000S device side.

For better clarity in the RAM-programming mode, an enhancement has been introduced on FLEX_DIGIT device initially used only for display of Output Register's content. FLEX_DIGIT is made to display the content of memory location pointed to by the RAM address input switches (*ADDR_IN* at MAX_SW1[5] through MAX_SW1[5]) when the *RUN_PROG* input is low. With this, the user is able to verify that correct instruction and data have been entered at correct memory locations before executing the program. To achieve this, 2 instantiations of the 4-bit binary to hexadecimal display decoder module is made in the 16×8 RAM module too, yielding decoded signal *RAM_DISPLAY_HEX*. Selection of decoded signal being passed to the FLEX_DIGIT display is done according to the logic state of the *RUN_PROG* input. *RAM_DISPLAY_HEX* is passed instead of *MUX_OUT_HEX* when *RUN_PROG* is low and vice versa.

Applying all changes discussed earlier in this section to the Verilog program developed during the simulation stage, the Quartus II project created for SAP-1 system synthesis is compiled for the target FLEX 10K device using the Analysis and Synthesis function. Erroneous source code is traced and corrected in case errors are

reported. Manual pin assignments for the input and output signals are then done, aiming to avoid messiness of wirings on the prototyping board. Pins attached to expansion holes at FLEX_EXPAN_A located nearer to the switches and LEDs are selected. The Fitter is then run to place and route the design to logic cells of the FLEX 10K device, whereas Assembler is executed to generate the programming file (SRAM Object File with extension of .sof) of the project [12].

The same process is applied to the additional binary to hexadecimal display decoder module on the MAX 7000S device, done in a different project. The Programmer Object File (.pof) generated is added into the Chain Description File (.cdf) of the SAP-1 system project [12]. This enables programming of both FLEX 10K and MAX 7000S devices in a chain. Applying appropriate jumper settings, the chips are programmed using the Programmer available in the Quartus II software, with the board being connected to the computer through ByteBlaster II cable. The programmed platform is then ready for testing.

System synthesis of SAP-1 computer has been completed. Test results and findings of hardware implementation of SAP-1 system will be presented in Chapter 4 of this report. Verilog programs created for this system synthesis is available in Appendix D.

3.6 Modular Synthesis of SAP-1 Computer System

Modular synthesis is aimed to implement each functional block of SAP-1 computer on an Altera's UP2 development platform. SAP-1 computer will be implemented as 10 separate modules as illustrated in modified clock diagram of SAP-1 computer available in Appendix A. All 10 boards will be interconnected to form a complete SAP-1 computer.

This implementation has advantage over system synthesis of SAP-1 which the entire system is implemented on a single board, as it is able to show every single detail of SAP-1 computer down to the microinstruction level. It promotes good understanding of the system by examining operations within the prototype alone.

Separate Quartus II projects are set up for each SAP-1 module to be implemented on different UP2 platform. Binary to hexadecimal 7-segment display decoder module as described in Section 3.5 and Appendix C is incorporated into all SAP-1 modules except the Controller / Sequencer and Mode-Select Switches, De-bouncers & Clock Buffer modules. This enables display of output of these modules on the 7-segment displays available on UP2 boards, enhancing readability of the outputs. LEDs on the boards are also properly utilized to indicate logic state of outputs of the modules to make the prototype easily understandable. Verilog source code of modular synthesis of SAP-1 is available in Appendix E. Detailed pin assignments, pin interconnections across boards, and input and output device utilization of all UP2 platforms used in SAP-1 modular synthesis are presented in Appendix F of this report.

As the FLEX 10K device on the UP2 development platform is based on Static RAM technology [13], it is volatile. Hence, it is impractical to implement modular synthesis of SAP-1 on the FLEX 10K device as programming of 10 boards prior to any usage of the prototype is tedious. This justifies our selection of the non-volatile MAX 7000S CPLD device for this activity [13]. However, this device consisting of only 128 macrocells could not accommodate the 16×8 RAM module requiring 192 macrocells [13]. We have two choices dealing with this issue. The 16×8 RAM module can be implemented on the FLEX 10K device. This reduces the number of UP2 boards required by one as the module can be implemented at any one of the nine boards which the FLEX 10K device initially remains unused. Nevertheless, a major drawback exists that the prototype must be programmed with this 16×8 RAM module each time it is powered up. For the second option, the 16×8 RAM module can be reduced to an 8×8 RAM module, enabling implementation on the MAX 7000S device. This option is adopted as it provides better convenience to the prototype users. Although the RAM is shrunk in size, it is still sufficiently big to store instructions and data of the test program to be used throughout discussions in Chapter 4, which consists of all five instructions of SAP-1 computer.

Interconnections of the *WBus*, *CLK*, and *CLR* signals are implemented on extra PCB. This is done to reduce the messiness of wirings on the prototype, thus decreasing the probability of mistakes when setting up the circuits. Typically,

interconnection of multiple-bit signals such as the *WBus*, *ACCU_OUT*, *B_REG_OUT*, *MUX_OUT*, and *IR_OUT_INS* are done using insulation displacement connectors (IDC) with ribbon cable through pin headers soldered at the UP2 platforms or the PCB. Other and typically single-bit signals are interconnected using wires through wire-wraps soldered at the UP2 boards or the PCB.

The same project development flow in Quartus II software (HDL Design Entry, Analysis & Synthesis, Pin Assignment & Fitting, and Configuration & Programming) as discussed in Section 3.5 is applied here for projects set up for each SAP-1 module. The prototyping boards are ready for testing and verification as they are programmed using the Programmer in Quartus II software with ByteBlaster II cable connecting the computer and UP2 platform.

CHAPTER 4

RESULTS & DISCUSSIONS

This chapter presents the test results and findings of all SAP-1 computer simulation, SAP-1 system synthesis, and SAP-1 modular synthesis. Kindly refer to chapter 10 of [1] *Digital Computer Electronics* by Malvino (1983) as appropriate to aid your understanding in this chapter.

4.1 Simulation of SAP-1 Computer System

Results of modular simulation and system simulation of SAP-1 computer will be discussed in this section referring to the signal waveforms obtained. Some brief description on the Verilog code and testbench of each module will also be given. Note that signals in the Verilog programs are presented in *Italic*. As mentioned earlier, the Verilog programs are designed for similar behaviour as SAP-1 computer in its original TTL circuits implementation, besides simplifications stated in Section 3.4 of this report. The tesbenches are also written to reflect actual behaviour of the modules in SAP-1 computer system.

4.1.1 Program Counter

A positive-edge-triggered JK flip-flop module with active high clear is first written. The Program Counter which is essentially a ripple counter is then described by instantiating the JK flip-flop module. Inverted output of the least significant bit (LSB) JK flip-flop is fed as the clock input of the next JK flip-flop. Output of Program Counter (signal *PC*) will only be made available at *WBus* when the enable signal *Ep* goes high. Else this connection remains in high-impedance state (z state).

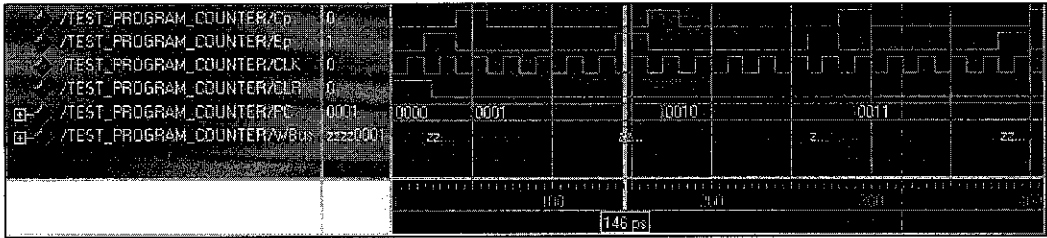


FIGURE 2: Simulation results of the Program Counter module

Clock cycle of 20ps is used in all simulations. The Program Counter is reset to 0000 when *CLR* is high. The first machine cycle starts at the falling clock edge at 20ps. *EP* signal that puts value of *PC* on the *WBus* goes high for every clock cycle starting from 20ps + $n*6*20ps$, or technically during the T_1 state. Meanwhile, the count enable signal, *CP* goes high at every T_2 state (clock cycle starting from 40ps + $n*6*20ps$). *PC* is incremented at each positive clock edge appearing at the middle of each T state [1].

4.1.2 MAR

The MAR is built based on positive-edge-triggered D flip-flops with active high enable input for data loading. Signal appearing at the input of the D flip-flop will only be accepted at the positive clock edge if the enable signal is high. The MAR module shows 4 instantiations of the D flip-flop module. Each D flip-flop acts as a register for storing one address bit. These D flip-flops are fed with the input coming from the *WBus* at a positive clock edge when the control signal *Lm* is high. Else, output of each D flip-flop is fed back into its input.

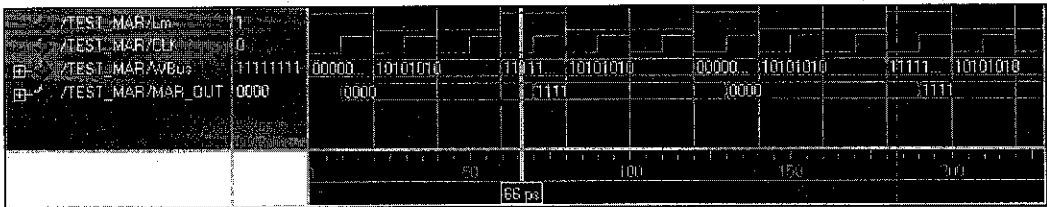


FIGURE 3: Simulation results of the MAR module.

First machine cycle starts at time 0ps for this simulation. Timing of the *Lm* signal applies for all LDA, ADD, and SUB routines for this testbench coding. It goes high at every T_1 and T_4 state and remains low at any other instance of the machine

cycle [1]. Assuming that the instruction being executed is stored at location 0x0 of the RAM, whereas the data of the operand is stored at location 0xf, the behaviour of the module is shown in Figure 3. These memory addresses start appearing at *MAR_OUT* at positive clock edge midway through *T*₁ and *T*₄ respectively and is made available to the 2 to 1 Multiplexer.

4.1.3 2 to 1 Multiplexer

Source code of the 2 to 1 Multiplexer is made up of a single module. It is simply a multiplexer outputting either *MAR_OUT* or *ADDR_IN* 4-bit address depending on the *RUN_PROG* select signal. The *MAR_OUT* address is selected if the *RUN_PROG* switch is in RUN position (*RUN_PROG* == 1) and vice versa [1]. The output at this multiplexer will be used as pointer to access the corresponding memory location of the RAM.

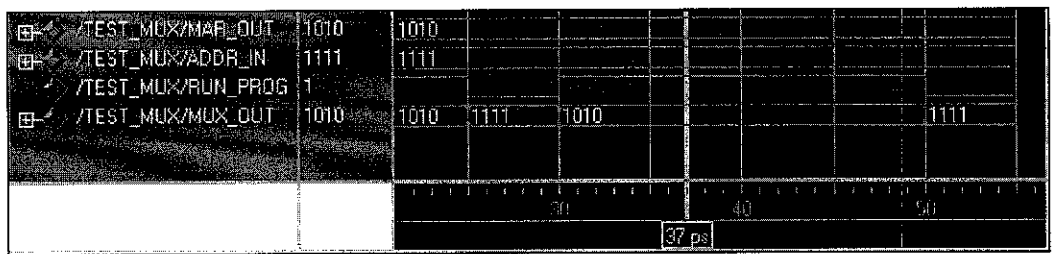


FIGURE 4: Simulation results of the 2 to 1 Multiplexer module

As this module is not clocked, the testbench coding is straightforward. Assuming that the input coming from *MAR_OUT* is 1010 binary whereas the *ADDR_IN* switches send signals of 1111 binary, 1010 is selected if *RUN_PROG* == 1. Output of the module (*MUX_OUT*) is 1111 if *RUN_PROG* == 0.

4.1.4 16 × 8 RAM

Low logic of *RUN_PROG* and *READ_WRITE* signals represent RAM programming and writing mode respectively, and vice versa. When both signals are low, data from the input switches (*DATA_IN*) is written to the memory location given by output of

the 2 to 1 Multiplexer (*MUX_OUT*). For read operation, the high *RUN_PROG* input is required. Data at address indicated by *MUX_OUT* will be retrieved and made available to the *WBus* when the Chip Enable signal (*CE*) goes high [1].

Testbench of the 16×8 RAM module is written to simulate two write operations to memory location 0x0 and 0x1 followed by read operations from these locations. Data being retrieved from the RAM is only made available to the *WBus* when *CE* goes high during the read operation. The *WBus* remains in high-impedance state at any other instances. Note that logical value of *READ_WRITE* switch does not impose any effect to the module when it is operating in *RUN* mode.

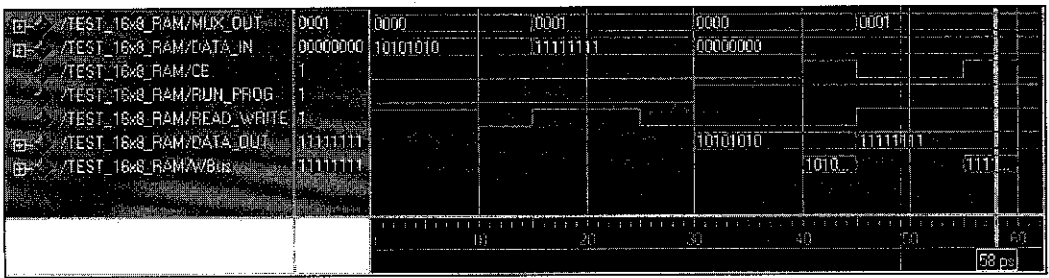


FIGURE 5: Simulation results of the 16×8 RAM module

4.1.5 Instruction Register

The Instruction Register generally functions to separate the instruction fetched to it into the upper nibble instruction op code and lower nibble data address. It utilizes D flip-flops similar to that of MAR module as registers to store the input values. It accepts input (instruction in machine code being fetched from RAM) at the *WBus* at T_3 and outputs the data address at *WBus* through its tri-state output at T_4 [1]. The op code output going into the Controller/Sequencer however is not clocked.

4.1.6 Accumulator

The Verilog program of the Accumulator module is written by instantiations of D flip-flop module with enable input for data loading (*La*). A D flip-flop is used to hold 1 bit of data thus a total of 8 D flip-flops have been used. Data is loaded into the

Accumulator at T_5 (for LDA routine) or T_6 (for ADD & SUB routines) when La control signal goes high [1]. Data stored in the Accumulator is made available at the $WBus$ at T_4 when Ea signal goes high as OUT routine is executed. The connection between the Accumulator output and the $WBus$ remains at high-impedance state whenever Ea has low logic [1].

4.1.7 Adder/Subtractor

The following formula is used in description of the Adder/Subtractor module:-

$$ADD_SUB_OUT = ACCU_OUT + B + Su$$

where ADD_SUB_OUT is the output of the Adder/Subtractor, $ACCU_OUT$ is the input to the Adder/Subtractor from the Accumulator. B is obtained from the output of B Register, B_REG_OUT depending on the logic level of the subtraction enable signal, Su . When Su is low, indicating an addition operation, B is made equal to B_REG_OUT . Hence, addition of the Accumulator and B Register's content is achieved through the formula above.

For a subtraction operation, Su is high. B is equals to complement of B_REG_OUT in this case. 1 (Su) is added into B (complemented B_REG_OUT) and forms 2's complement conversion of B_REG_OUT . The same formula now executes addition of an unsigned number and a 2's complement negative number that yields an unsigned number that equals result of a subtraction operation.

"*bufif1*" primitives are instantiated in this module to establish a three-state connection between ADD_SUB_OUT and the $WBus$. These tri-state buffers are activated by the Eu control signal [1].

FIGURE 6 and 7 show the simulation waveforms of the written testbench for addition and subtraction operation respectively. Note that as the Adder/Subtractor module is asynchronous in nature (not clocked), result of the arithmetic operation is available at the module's output as soon as it is fed with the inputs ($ACCU_OUT$, B_REG_OUT , and Su). This result is made available at the $WBus$ at T_6 when Eu goes high as the ADD or SUB instruction is executed [1].

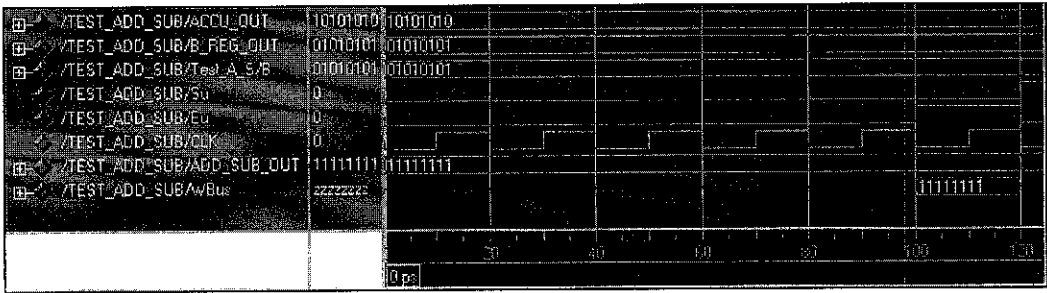


FIGURE 6: Simulation results of the Adder/Subtractor module for addition operation

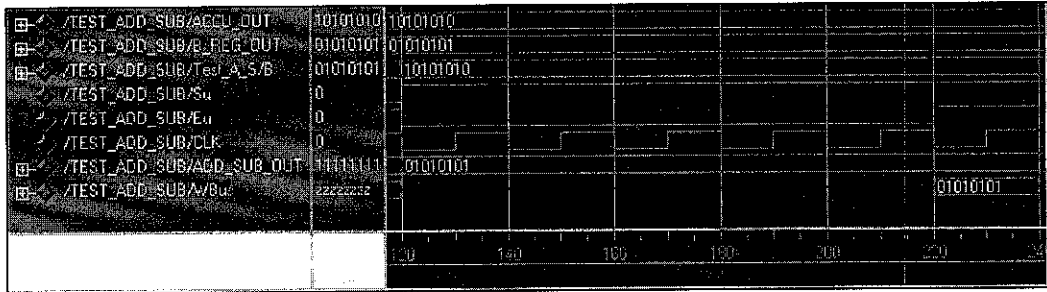


FIGURE 7: Simulation results of the Adder/Subtractor module for subtraction operation

4.1.8 B Register

The B Register accepts input from the *WBus* when the load signal *Lb* goes high at T_5 for execution of ADD or SUB instruction [1]. Data loading operation occurs at the positive clock edge midway through T_5 state as positive-edge-triggered D flip-flops are used to hold the input data [1]. This can be observed in the simulation waveforms shown below. Meanwhile, these D flip-flops also continuously drive output signal, *B_REG_OUT* of the module as soon as data is loaded.

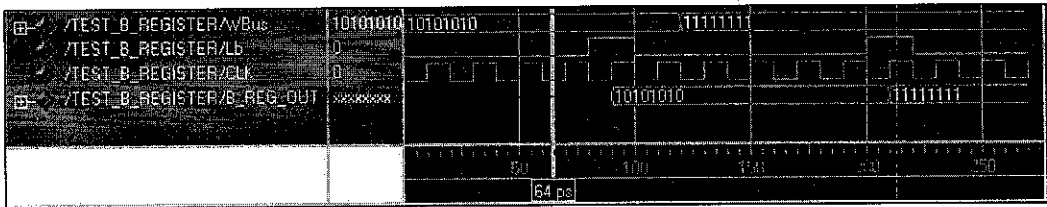


FIGURE 8: Simulation results of the B Register module

4.1.9 Output Register

Operation of the Output Register is essentially the same as the B Register except that it is driven by data loading signal *Lo*. *Lo* only goes high at T₄ when the OUT routine is executed [1].

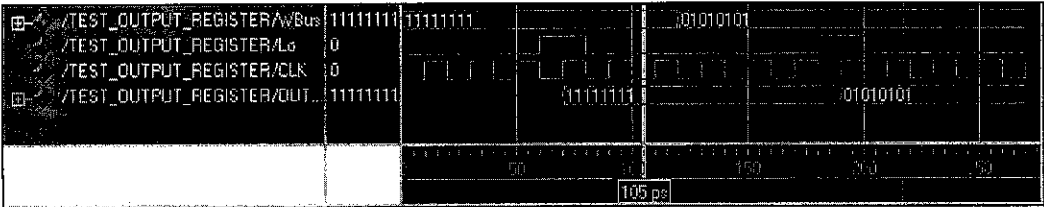


FIGURE 9: Simulation results of the Output Register module

4.1.10 Controller/Sequencer (Instruction Decoder, Ring Counter & Control Matrix)

First of all, the Ring Counter in the Controller/Sequencer module is constructed using 6 negative-edge-triggered JK flip-flops with active high clear. *Qnot* and *Q* outputs of the least-significant-bit (LSB) flip-flop drive the *J* and *K* inputs of the next flip-flop so that count of 000001 will be obtained at *Qnot* output when high *CLR* signal is applied. On the other hand, *Q* and *Qnot* outputs of the second and higher flip-flops drive the *J* and *K* input of the next higher flip-flop. Thus the Ring Counter shifts left at each clock cycle starting with a negative clock edge. This creates 6 T states with a positive clock edge midway through each state [1].

Meanwhile, the Instruction Decoder and Control Matrix modules are described using logical and conditional statements. FIGURE 10 shows the waveforms obtained from testbench written to verify this Controller/Sequencer module. Note that as simplifications mentioned in Section 3.4 of this report apply, all control signals goes high as they are active. The testbench simulates operations of the Controller/Sequencer module for all LDA, ADD, SUB, OUT, and HLT routines in the order as listed, each occupying an instruction cycle (6 T states). Observe that the decoded instruction signals (*LDA*, *ADD*, *SUB*, *OUT*, and *HLT*) only goes active one at a time after the corresponding op code is fetched to the instruction decoder. Table 3

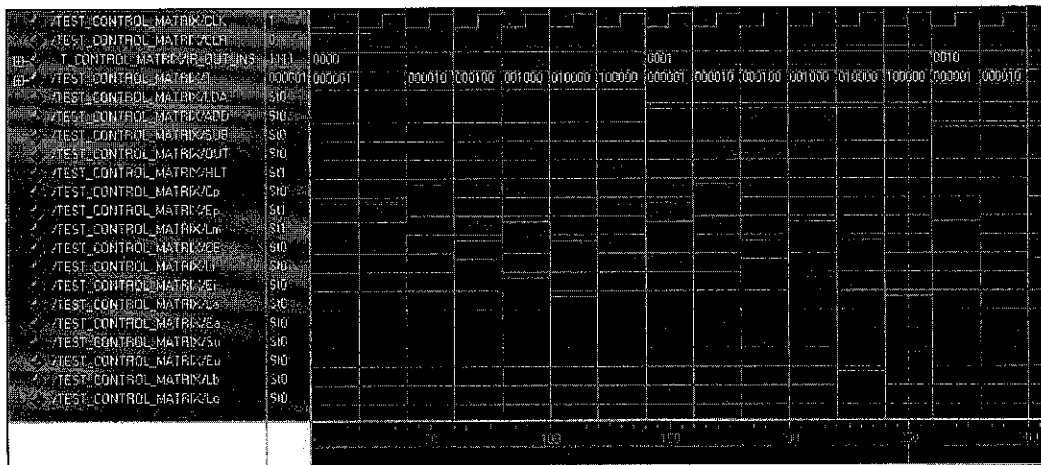


FIGURE 11: Input and output waveforms of simulation of the Controller/Sequencer module at the earliest 2 instruction cycles

Figure 11 is similar to Figure 10 except that it is zoomed to the first 2 instruction cycles which the LDA and ADD instructions are being fetched and decoded. This figure is included for clearer inspection of the Ring Counter's output, *T*. Note that this signal shifts left at each falling clock edge when the *CLR* signal is at logic low.

4.1.11 Mode-Select Switches, De-bouncers & Clock Buffer

The Clear-Start De-bouncer, Single-Step De-bouncer, Manual-Auto De-bouncer, and Clock Buffer modules are written as separate Verilog modules using SR latch, JK flip-flop, and logic gates. These modules are then instantiated in a single module named DEBOUNCERS. Due to the simplification introduced to this project, all modules are made to accept non-inverted clock signal, Hence, the inverted clock signal present in the original design in [1] *Digital Computer Electronics* by Malvino (1983) will not be used and the block that generates it can be eliminated.

FIGURE 12 shows the simulation results of testbench written to verify the DEBOUNCERS module:-

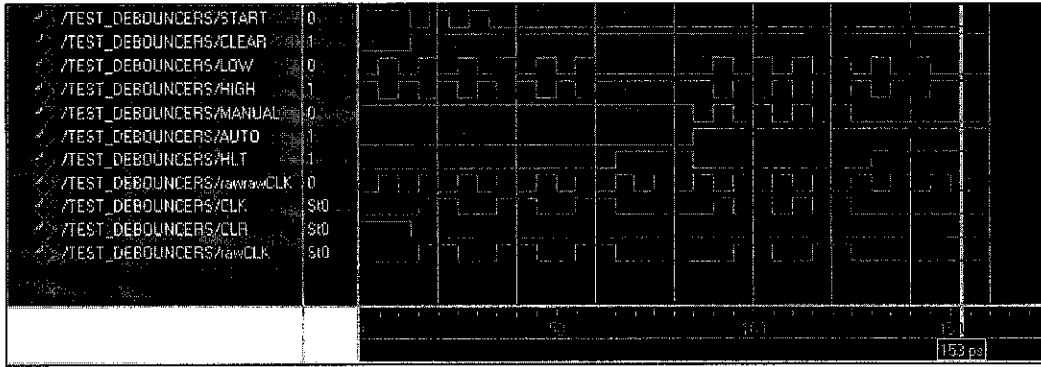


FIGURE 12: Simulation results of the Mode-Select Switches, De-bouncers & Clock Buffer module

Note that the *START* & *CLEAR*, *LOW* & *HIGH*, and *MANUAL* & *AUTO* switches are active when they are at low logic level. When the *CLEAR* input is low, the *CLR* signal goes high, resetting the *CLK* signal. Auto clock mode is simulated after the initial resetting of the module (*START* and *AUTO* go low, *CLEAR* and *MANUAL* go high). In this mode, the output *CLK* signal follows the *rawCLK* signal, which is essentially the *rawrawCLK* signal being scaled down to half its frequency. The bouncing phenomenon of the *CLEAR* & *START* switches pair is also simulated through this testbench. Operation of the module is not affected by this phenomenon as SR latch used serves as a switch de-bouncer. It can also be observed through the figure that logic level of the *LOW/HIGH* switch does not have any effect in auto clock mode. The output *CLK* signal stops changing state as the *HLT* signal goes active.

Step-through or manual-clocking mode is simulated starting from time 85ps. The *MANUAL* signal goes low now whereas the *AUTO* signal changes state to high. Bouncing phenomenon of the *MANUAL* switch is also simulated. The output *CLK* signal now follows the logic level of the *LOW* & *HIGH* switches pair. The *HLT* signal is activated at 130ps. The output *CLK* stops changing state regardless of the logic level of the *LOW* & *HIGH* switches pair from then on.

4.1.12 System Level Simulation of SAP-1 Computer

System level simulation of SAP-1 computer is done to verify the functionality of the entire system when all independently written and tested SAP-1 modules are combined. A module named *SAP1* is written to instantiate all top level sub-modules within each functional blocks of SAP-1 computer. Testbench of this module has been properly designed to test its functionality thoroughly. Shown below is the SAP-1 program used in the testbench, in both assembly language and machine code:-

TABLE 4: Program used for verification of SAP-1 system level simulation

Assembly Code		Machine Code	
Address	Instruction/Data	Address	Instruction/Data
0x0	LDA 0xa	0000	0000 1010
0x1	ADD 0xb	0001	0001 1011
0x2	SUB 0xc	0010	0010 1100
0x3	OUT	0011	1110 xxxx
0x4	HLT	0100	1111 xxxx
0xa	0xaa	1010	1010 1010
0xb	0x55	1011	0101 0101
0xc	0x0f	1100	0000 1111

Referring to descriptions given in Section 2.3 of this report, SAP-1 instructions are programmed at lower RAM locations starting at location 0x0. Meanwhile, the data used as operand of the instructions reside in higher location such as location 0xa as shown in the program above. The instruction format is made up of the upper nibble instruction op code, and the lower nibble operand RAM location. Thus execution of the first instruction in the test program loads data stored at location 0xa (value 0xaa) into the Accumulator. In short, the test program above does arithmetic computation of 0xaa plus 0x55 minus 0x0f. A result of 0xf0 is expected to be present at the Output Register upon completion of execution of this program.

Outcome of the system simulation will be discussed next. As detailed explanation of all operations in the entire SAP-1 system during execution of the test program is lengthy, only details during RAM programming stage and execution stage of the ADD and HLT routines will be examined closely here. Figure 13 shows the waveforms obtained in SAP-1 system level simulation showing steps involved in programming of the RAM.

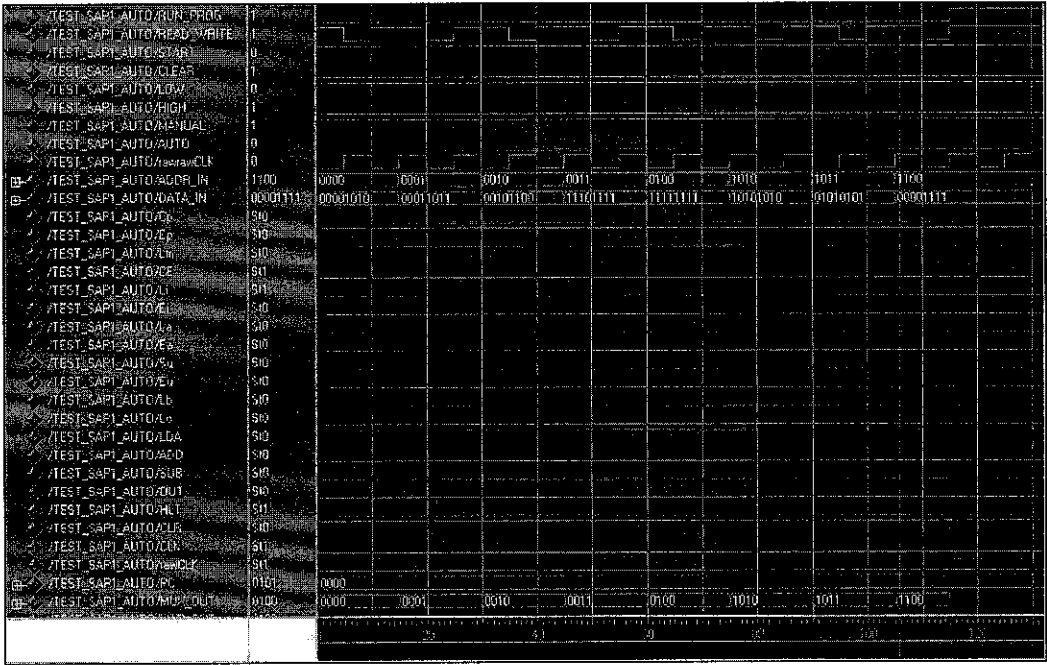


FIGURE 13: Input and output waveforms of SAP-1 system level simulation in RAM-programming mode

Inputs of the *START* and *CLEAR* switches pair are held at logic high and logic low respectively to activate the global *CLR* signal. The activated *CLR* signal resets the output of both Program Counter (*PC*) and upper nibble of Instruction Register (*IR_OUT_INS*) to value of 0000, and output of the Ring Counter (*T*) to 000001. Hence, it can be observed that the decoded instruction line of *LDA* and the corresponding control signals in T_1 state (*Ep*, *Lm*) are active at beginning of the testbench execution. Note that the *CLK* signal stops when this mode is selected.

Logic level of 0 is applied to the *RUN_PROG* signal at the beginning of the testbench for RAM-programming mode. Inputs corresponding to value of the Address and Instruction/Data field of test program shown in Table 4 are applied sequentially to the *ADDR_IN* and *DATA_IN* signal respectively, row by row. Note that the output of the 2 to 1 MUX (*MUX_OUT*) always follows logic level of the signal present at *ADDR_IN* in this mode. Allowing some time for stabilization of *ADDR_IN* and *DATA_IN* inputs, logic state of the *READ_WRITE* input signal is switched to low and held constant for 5ps before returning it to logic high for memory write operation. Repeating this for all appropriate combinations of *ADDR_IN* and *DATA_IN* inputs,

TABLE 5: Summary of internal operations of the ADD routine in test program of
SAP-1 system simulation

T State	Active Decoded Instruction Line Signal	Active Control Signal	Description
T₁	<i>LDA</i>	<i>Ep, Lm</i>	<ul style="list-style-type: none"> - Activated <i>Ep</i> signal makes <i>PC</i> signal (0001) available at the <i>WBus</i> (zzzz0001) throughout the T state. - The MAR loads the lower nibble value of the <i>WBus</i> as <i>Lm</i> goes active at positive clock edge halfway through the T state and outputs it through <i>MUX_OUT</i> (0001).
T₂	<i>LDA</i>	<i>Cp</i>	<ul style="list-style-type: none"> - <i>PC</i> is incremented by one from 0001 to 0010.
T₃	<i>LDA</i> (1 st half of T ₃) <i>ADD</i> (2 nd half of T ₃)	<i>CE, Li</i>	<ul style="list-style-type: none"> - The RAM is enabled by the <i>CE</i> signal so that instruction stored at location pointed to by <i>MUX_OUT</i> is retrieved from the RAM and made available at the <i>WBus</i> (00011011) throughout the T state. - The Instruction Register loads the data of the <i>WBus</i> as <i>Li</i> goes active at positive clock edge halfway through the T state. The data is separated into the upper nibble - <i>IR_OUT_INS</i> (0001) and lower nibble <i>IR_OUT_ADDR</i> (1011). <i>IR_OUT_INS</i> is decoded in the Controller/Sequencer block, causing a change of active decoded-instruction line signal from <i>LDA</i> to <i>ADD</i> halfway through the T state.
T₄	<i>ADD</i>	<i>Lm, Ei</i>	<ul style="list-style-type: none"> - <i>Ei</i> enables the tri-state buffer at the lower nibble output of the Instruction Register, making data carried by the - <i>IR_OUT_ADDR</i> (1011) signal available at the <i>WBus</i> (zzzz1011) throughout the T state. - The MAR loads the lower nibble of data of the <i>WBus</i> as <i>Lm</i> goes active at positive clock edge halfway through the T state. Loaded data is output through <i>MUX_OUT</i> (1011).
T₅	<i>ADD</i>	<i>CE, Lb</i>	<ul style="list-style-type: none"> - The RAM is enabled by the <i>CE</i> signal so that data stored at location pointed to by <i>MUX_OUT</i> is retrieved from the RAM and made available at the <i>WBus</i> (01010101) throughout the T state. - The <i>Lb</i> signal enables data loading of the B Register from the <i>WBus</i> at the positive clock edge halfway through the T state. B Register's content is output through <i>B_REG_OUT</i> (01010101) to the asynchronous Adder/Subtractor, causing a change at its output, <i>ADD_SUB_OUT</i> (11111111) immediately.
T₆	<i>ADD</i>	<i>La, Eu</i>	<ul style="list-style-type: none"> - Value carried by the <i>ADD_SUB_OUT</i> signal is made available at the <i>WBus</i> (11111111) throughout the T state as <i>Eu</i> goes active. - <i>La</i> enables data loading of the Accumulator from the <i>WBus</i> at the positive edge halfway through the T state. Loaded data is output through signal <i>ACCU_OUT</i> (11111111).

The behaviour of the *ADD_SUB_OUT* and *WBus* signals may seem abnormal at the second half of T_6 state (time 365ps to 375ps). The observation will be justified here. Since the Adder/Subtractor module is asynchronous in nature, result of arithmetic computation will be available at *ADD_SUB_OUT* as soon as data is present at both *ACCU_OUT* and *B_REG_OUT*. As the result of addition operation in the ADD routine is loaded into the Accumulator through the *WBus* at the positive edge halfway through T_6 , a new data is available at *ACCU_OUT*. This yields a new value of *ADD_SUB_OUT* resulting from addition operation of the new *ACCU_OUT* value and unchanged *B_REG_OUT* value almost immediately. This new value of *ADD_SUB_OUT* is also passed to the *WBus* as the *Eu* signal driving the tri-state buffer between *ADD_SUB_OUT* and *WBus* remains active for the entire T_6 state. However, this invalid new *ADD_SUB_OUT* value is not passed to *ACCU_OUT* as data loading of the Accumulator is positive-clock-edge-triggered. Hence, correct functionality of the SAP-1 system is not affected.

Figure 15 shows the final outcome of SAP-1 system simulation's testbench. The HLT routine is executed from time 612ps onwards as shown in the figure. As mentioned in Section 2.4 of this report, operations taking place are common for all instructions in the Fetch Cycle occupying the first 3 T states. Internal operations during T_1 to T_3 of the HLT routine is very similar to those that has been discussed in Table 5, thus they are not repeated here to keep the discussion concise. Instead, we will examine closely effect imposed on the system as the decoded instruction line signal *HLT* is activated. As op code of the HLT instruction is decoded halfway through T_3 state, the activated *HLT* signal forces the *rawCLK* signal to stop changing state. This cause the Program Counter and Ring Counter to stop counting, hence halted all operations within the SAP-1 system.

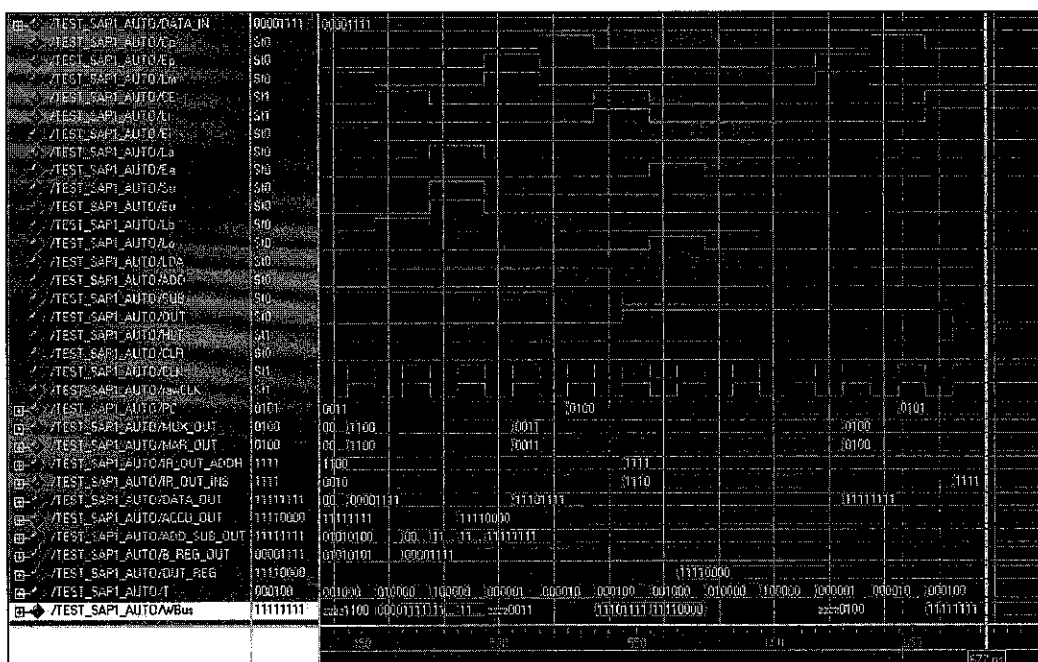


FIGURE 15: Final results of SAP-1 system level simulation

Examining the value of *OUT_REG* driven by content of the Output Register, result of 0xf0 is obtained for computations in the tests program. This value matches our prediction made earlier. Therefore, we can conclude that functionality of the Verilog programs of SAP-1 system has been verified.

Note that the testbench that yields input and output waveforms shown in this sectioned is tailored for auto-clocked execution mode. Another testbench that simulates operations of SAP-1 in manual-clocked mode is also written. It is very similar to the testbench for auto-clocked mode except that the *MANUAL* & *AUTO* switches pairs are tied to logic 0 & 1 respectively, while logic state of the *LOW* & *HIGH* switches pair are inverted repeatedly. This testbench also yields similar input and output waveforms; except that the *CLK* signal follows the logic state of the *LOW* & *HIGH* switches pair instead of the *rawCLK* signal. Similar final outcome of 0xf0 at *OUT_REG* is also obtained running this testbench.

4.1.13 Overall Findings

As mentioned earlier, testbenches of all modules are written so that they best reflect real operations in the SAP-1 system. Test coverage has been made wide in modular verification of the Verilog program (e.g. behaviour of all LDA, ADD, SUB, OUT, HLT routines in the Controller/Sequencer module have been tested). Only synthesizable Verilog statements are used in coding of SAP-1 modules. As discussed throughout Section 4.1 of this chapter, results of all modular simulations and system simulation show correct behaviour. No irresolvable bug has been identified in this simulation stage. Hence, the Verilog programs written in simulation stage of SAP-1 computer will serve as fundamental programs for both system synthesis and modular synthesis of SAP-1 computer system.

4.2 System Synthesis of SAP-1 Computer System

Verification of the prototype is started as soon as the chips are programmed and all necessary wirings are done. The same test program as used in system simulation of SAP-1 computer shown in Table 4 is being used here. Please refer to Table 2 in Section 3.5 for naming of UP2 platform input and output devices and the driven or driving signals in prototype of SAP-1 system synthesis. Note also that signals in the Verilog programs of SAP-1 computer are presented in *Italic*.

MAX_SW2[1] (*START_CLEAR*) and MAX_SW2[2] (*MANUAL_AUTO*) are initially set at OFF state (logic high) for activation of the *CLR* signal (indicated by LED D16) and the auto-clocked mode. MAX_SW2[3] (*LOW_HIGH*) is held at ON state. To switch to RAM-programming mode, MAX_SW2[8] (*RUN_PROG*) is set at ON state to provide a logic low input. MAX_DIGIT now displays input signal present at MAX_SW1[5] through MAX_SW1[8] in hexadecimal representation, which is essentially the memory location selected for programming. FLEX_DIGIT shows the content of memory location being pointed to by address input at MAX_SW1[5] through MAX_SW1[8]. Referring to Table 4 for the first test instruction, inputs of 0000 and 00001010 are applied at MAX_SW1[5] - MAX_SW1[8] and FLEX_SW[1] - FLEX_SW[8] for address input and data input respectively. Pressing the

FLEX_PB1 to give a logic low input to the *READ_WRITE* signal, the data input is programmed into the selected RAM location, and the new content of the RAM is being updated immediately on FLEX_DIGIT. Repeating these steps for the rest of instructions in test program shown in Table 4, programming of the SAP-1 RAM is completed thus program execution can be commenced.

A program test run in auto-clocked mode is first being done. With MAX_SW2[2] being held at OFF state (*MANUAL_AUTO* = 1, auto-clocked mode), MAX_DIGIT and FLEX_DIGIT display character '4' and 'F0' as soon as MAX_SW2[1] is turned to ON state (*START_CLEAR* = 0, start program execution). Character being present at MAX_DIGIT represents the final RAM location being accessed in execution of the test program, which is the location where the HLT instruction is stored. FLEX_DIGIT indicates the content of the Output Register, storing result of arithmetic computations carried out in the test program (0xaa plus 0x55 minus 0x0f).

The oscillator attached to the UP2 platform provides a CLK signal running at 25.175MHz. Being scaled down by 2 at the Clock Buffer of SAP-1, the prototype practically runs at clock frequency of 12.5875MHz. Execution of the test program takes 1 instruction cycle (6 T states) for LDA, ADD, SUB, OUT routines. The HLT routine takes 3 T states before the instruction is decoded, enabling the decoded instruction line signal, *HLT* which stops the *CLK* signal. This sums up to 27 T states or clock cycles for execution of the entire test program, which essentially takes only $26 * (1 / 12.5875\text{MHz}) = 2.066\mu\text{s}$. Hence, it is impossible for the user to examine internal operation of the computer at each T state at this clock frequency in auto-clocked execution mode.

Ripple counter can be programmed at the Clock Buffer to scale down the clock signal supplied by UP2 board's crystal oscillator. Verilog program of the Clock Buffer is modified to include a 26 bits ripple counter constructed using JK flip-flop modules. Taking output of the MSB JK flip-flop as the clock signal, it is scaled down to approximately $25.175\text{MHz} / 2^{26} = 0.3751\text{Hz}$. Hence each clock state (half a clock cycle) now takes approximately 1.3328s, making observation of operations within the system at each clock state possible. The test program mentioned above being

executed in auto-clocked mode at this clock frequency also yields the same final result of 0xf0. Details of the operations at each clock state in auto-clocked mode are similar to that of manual-clocked mode, which will be presented and discussed next.

Initial program test run attempts in manual-clocked mode have not been successful. Bouncing phenomenon has been encountered on MAX_SW2[3] used to generate manual clock signal *LOW_HIGH* of SAP-1 computer. This causes generation of multiple *CLK* cycles in one switching action, prohibiting the prototype user from observing changes taking place in each T state clearly. This problem is however understandable examining the simplification introduced in Section 3.5 on the De-bouncers block. Referring to page 159 of [1] *Digital Computer Electronics* by Malvino (1983), the active-low SR latches used as the mode-select switch de-bouncers functions provided that contact of the switch is lost as the switch bounces in a switching action, resulting in high logic being applied at both inputs. For your information, high logic at both inputs of an active-low SR latch results in unchanged outputs.

Simplifications introduced in Section 3.5 causes the inputs to the latch to always remain at opposite logic level, resulting in changes of the outputs state as the switch bounces in a switching action. Examination of the bouncing phenomenon of single pole double throw (SPDT) switch on oscilloscope has found that its output tend to fall into an unconnected condition as it bounces. Understanding these, solution to this problem is straight forward. Figure 15 shows the schematic diagram of *LOW_HIGH* switch implementation applied to eliminate the bouncing problem. This implementation is essentially the original design presented by Malvino. Both inputs to the active-low SR latch are pulled up to V_{cc} through 10k Ω resistors. The switch contact made drains the current thus providing low logic at one of the inputs. This maintains high logic at both inputs of the latch as contact of the switch is lost when it bounces. However, this implementation requires additional circuitry as dual inline package (DIP) ON/OFF switch (initially implemented on MAX_SW2[3] as stated in Table 2) available on UP2 platform cannot be utilized. This additional circuit has been constructed on veroboard. Slight modifications have been made to the Verilog program of the Debouncers module written earlier to follow Malvino's original design with 2 input signals *LOW* and *HIGH* driven by an SPDT switch.

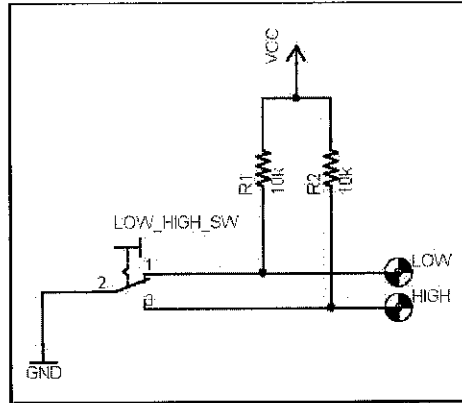


FIGURE 16: Schematic diagram of the *LOW_HIGH* input switches implementation

Bouncing phenomenon of input switches used for all other input signals is not critical in ensuring correct functionality of the system. Hence modification is not necessary. This eliminates the need of extra expansions of the prototyping board as DIP switches on the UP2 platform can be used.

The same test program is executed in manual-clocked mode. The same method of RAM programming as discussed earlier in this chapter is used. Upon completion of instruction/data entry, both MAX_SW2[8] and FLEX_PB1 are held at OFF state to drive signals *RUN_PROG* and *READ_WRITE* to logic low, enabling running-mode of the RAM. Logic low is applied at both *START_CLEAR* and *MANUAL_AUTO* inputs for program execution in manual-clocked mode. The SPDT switch is now used to generate manual *CLK* signal of the SAP-1 computer. Table 6 shows observations on the prototyping board for execution of the test program at each clock state resulting from switching of the SPDT toggle switch. '1' in the table indicates that LED is on whereas '0' represents the opposite. Note that the active low LEDs on the UP2 platform are driven by inverted signals of *T*, *LDA*, *ADD*, *SUB*, *OUT*, *HLT*, *CLK*, and *CLR* (*Tnot*, *LDAnot*, *ADDnot*, *SUBnot*, *OUTnot*, *HLTnot*, *CLKnot*, and *CLRnot*). For better clarity, logic levels of the non-inverted signals are presented in the table.

TABLE 6: Detailed observations of test program execution of SAP-1 system
synthesis in manual-clocked mode

Switching Action	Routine	T State	LEDs													7-Segment Display	
			D15 CLK	T						Decoded Instruction Line Signal					D16 CLR	MAX DIGIT MUX OUT	FLEX DIGIT DIGIT DISPLAY
				D1 T[1]	D2 T[2]	D3 T[3]	D5 T[4]	D6 T[5]	D7 T[6]	D9 LDA	D10 ADD	D11 SUB	D12 OUT	D13 HLT			
-1	-	-	0	1	0	0	0	0	0	1	0	0	0	0	1	-	-
0	LDA	T ₁	0	1	0	0	0	0	0	1	0	0	0	0	0	-	-
1			1	1	0	0	0	0	0	1	0	0	0	0	0	-	-
2		T ₂	0	0	1	0	0	0	0	1	0	0	0	0	0	-	-
3			1	0	1	0	0	0	0	1	0	0	0	0	0	-	-
4		T ₃	0	0	0	1	0	0	0	1	0	0	0	0	0	-	-
5			1	0	0	1	0	0	0	1	0	0	0	0	0	-	-
6		T ₄	0	0	0	0	1	0	0	1	0	0	0	0	0	-	-
7			1	0	0	0	1	0	0	1	0	0	0	0	0	-	-
8		T ₅	0	0	0	0	0	1	0	1	0	0	0	0	0	-	-
9			1	0	0	0	0	1	0	1	0	0	0	0	0	-	-
10		T ₆	0	0	0	0	0	0	1	1	0	0	0	0	0	-	-
11			1	0	0	0	0	0	1	1	0	0	0	0	0	-	-
12	ADD	T ₁	0	1	0	0	0	0	0	1	0	0	0	0	0	-	-
13			1	1	0	0	0	0	0	1	0	0	0	0	0	-	-
14		T ₂	0	0	1	0	0	0	0	1	0	0	0	0	0	-	-
15			1	0	1	0	0	0	0	1	0	0	0	0	0	-	-
16		T ₃	0	0	0	1	0	0	0	1	0	0	0	0	0	-	-
17			1	0	0	1	0	0	0	1	0	0	0	0	0	-	-
18		T ₄	0	0	0	0	1	0	0	0	1	0	0	0	0	-	-
19			1	0	0	0	1	0	0	0	1	0	0	0	0	-	-
20		T ₅	0	0	0	0	0	1	0	0	1	0	0	0	0	-	-
21			1	0	0	0	0	1	0	0	1	0	0	0	0	-	-
22		T ₆	0	0	0	0	0	0	1	0	1	0	0	0	0	-	-
23			1	0	0	0	0	0	1	0	1	0	0	0	0	-	-
24	SUB	T ₁	0	1	0	0	0	0	0	0	1	0	0	0	0	-	-
25			1	1	0	0	0	0	0	0	1	0	0	0	0	-	-
26		T ₂	0	0	1	0	0	0	0	0	1	0	0	0	0	-	-
27			1	0	1	0	0	0	0	0	1	0	0	0	0	-	-
28		T ₃	0	0	0	1	0	0	0	0	1	0	0	0	0	-	-
29			1	0	0	1	0	0	0	0	0	1	0	0	0	-	-
30		T ₄	0	0	0	0	1	0	0	0	0	1	0	0	0	-	-
31			1	0	0	0	1	0	0	0	0	1	0	0	0	-	-
32		T ₅	0	0	0	0	0	1	0	0	0	1	0	0	0	-	-
33			1	0	0	0	0	1	0	0	0	1	0	0	0	-	-
34		T ₆	0	0	0	0	0	0	1	0	0	1	0	0	0	-	-
35			1	0	0	0	0	0	1	0	0	1	0	0	0	-	-
36	OUT	T ₁	0	1	0	0	0	0	0	0	0	1	0	0	0	-	-
37			1	1	0	0	0	0	0	0	0	1	0	0	0	-	-
38		T ₂	0	0	1	0	0	0	0	0	0	1	0	0	0	-	-
39			1	0	1	0	0	0	0	0	0	1	0	0	0	-	-
40		T ₃	0	0	0	1	0	0	0	0	0	1	0	0	0	-	-
41			1	0	0	1	0	0	0	0	0	1	0	0	0	-	-
42		T ₄	0	0	0	0	1	0	0	0	0	0	1	0	0	-	-
43			1	0	0	0	1	0	0	0	0	0	1	0	0	-	-
44		T ₅	0	0	0	0	0	1	0	0	0	0	1	0	0	-	-
45			1	0	0	0	0	1	0	0	0	0	1	0	0	-	-
46		T ₆	0	0	0	0	0	0	1	0	0	0	1	0	0	-	-
47			1	0	0	0	0	0	1	0	0	0	1	0	0	-	-
48	HLT	T ₁	0	1	0	0	0	0	0	0	0	0	1	0	0	-	-
49			1	1	0	0	0	0	0	0	0	0	1	0	0	-	-
50		T ₂	0	0	1	0	0	0	0	0	0	0	1	0	0	-	-
51			1	0	1	0	0	0	0	0	0	0	1	0	0	-	-
52		T ₃	0	0	0	1	0	0	0	0	0	0	1	0	0	-	-
53			1	0	0	1	0	0	0	0	0	0	1	0	0	-	-
54		T ₄	0	0	0	0	1	0	0	0	0	0	0	1	0	-	-
55			0	0	0	0	1	0	0	0	0	0	0	1	0	-	-
Switching Action	Routine	T State	D15 CLK	D1 T[1]	D2 T[2]	D3 T[3]	D5 T[4]	D6 T[5]	D7 T[6]	D9 LDA	D10 ADD	D11 SUB	D12 OUT	D13 HLT	D16 CLR	MAX DIGIT MUX OUT	FLEX DIGIT DIGIT DISPLAY

From observations recorded in the table, the prototype shows clearly when the instruction op code is decoded. This happens at the positive clock edge occurring half-way through each T_3 state. The prototype also displays the address of memory location being accessed throughout execution of the program. Memory location at *MUX_OUT* for access of instruction stored in RAM (locations 0, 1, 2, 3, and 4) is updated at positive clock edge mid-way through every T_1 state. For *LDA*, *ADD*, and *SUB* routines involving data, memory location for data retrieval from RAM is performed at positive clock cycle of each T_4 state.

The -1st switching is included in Table 6 to show behaviour of the system when the *CLR* signal is active (*START_CLEAR* == 1). As described in Section 4.1.12, activated *CLR* signal resets *PC* and *IR_OUT_INS* to 0000, and *T* to 000001. This verifies our observation as *T[1]* and *LDA* are found to have logic high.

Note that the *CLK* signal remains unchanged at logic low even with further switching after decoding of op code of the *HLT* routine (*HLT* == 1). Halting of operations within the system is achieved by gating the *CLK* signal.

Number being displayed at *FLEX_DIGIT* (signal *DIGIT_DISPLAY*) remains invalid until result of the arithmetic operations is fetched from the Accumulator to the Output Register at positive clock edge of T_4 during the *OUT* routine (after the 43rd switching). These 7-segment displays output the non-initialized content of the Output Register (*OUT_REG*) taking any random number prior to this. Table 6 shows that execution of the test program yields 0xf0 as the final result. This verifies the functionality of the prototype. The entire system is also found to function correctly examining each state of Table 6 carefully. Kindly refer to Chapter 10 of [1] *Digital Computer Electronics* by Malvino (1983) to verify test results of this section in detail.

4.3 Modular Synthesis of SAP-1 Computer System

Each SAP-1 module on a separate UP2 platform is first tested independently. Inputs are supplied to each module using switches available on the board and the resultant output is verified. This is similar to functionality of testbenches used to verify Verilog

program of SAP-1 modules created during SAP-1 simulation stage. Detailed discussion on the test results of hardware of each individual SAP-1 module is not presented in this report considering the length that it might take. All boards implementing an SAP-1 module each have been found to be working correctly independently.

All 10 UP2 platforms each implementing functionality of a SAP-1 computer module is then interconnected to form a complete SAP-1 computer. Various problems have been encountered interconnecting the SAP-1 module each implemented on a UP2 platform for a complete SAP-1 system. The encountered problems and their solution will be discussed in this section. Experience on overcoming bouncing problem of the *HIGH_LOW* switch used for generation of manual clock signal gained during SAP-1 system synthesis is applied here. Hence, an SPDT switch as illustrated in Figure 16 is used to drive the Single-Step De-bouncer's input.

Signals in the system tend to be unstable when the UP2 boards being supplied by independent power adaptors are interconnected. This problem arises as different ground level is present at each UP2 board being grounded to the earth through respective power adaptors. To overcome this problem, all UP2 boards are powered using a single external power supply unit while Vcc and Gnd pin of all boards are being interconnected.

Physically connected but unused pins between the interconnected boards also create problem in the modular SAP-1 prototype. These pins tend to drain considerable amount of current. As a result, signals that is supposed to have high logic falls within an indeterminate logic level. A convenient solution to this problem is to place always disabled tri-state buffer at these pins. This implementation does not require any extra hardware.

The modular SAP-1 prototype has not been functioning properly when it is being clocked at 12.5875MHz as calculated Section 4.2. The cause of this problem can be traced to degradation of signal integrity of the *CLK* signal being connected from the Clock Buffer module to other modules through long wires at high frequency as mentioned. This problem can be resolved by adopting the 26 bits ripple counter

design at the Clock Buffer as implemented in prototype of SAP-1 system synthesis to scale the clock speed down to approximately 0.3751Hz. Furthermore, this implementation also enables the prototype users to examine operations within each clock state of program execution clearly in auto-clocked mode.

Another problem observed during program execution of the modular SAP-1 prototype is due to impulse noise of the CLR signal. CLR input of all Program Counter modules, Instruction Register module, and Controller/Sequencer module is driven by a switch in the Mode-Select Switches module through the Start-Clear De-bouncer. When this is implemented, content of the Program Counter and upper nibble of the Instruction Register (holding signal *IR_OUT_INS*, the instruction op code field of SAP-1 instruction) tend to reset itself. Trials have been thrown by rearranging position of the UP2 platforms so that the CLR signals can be sent through shorter wires. Verilog program of the De-bouncers module has also been modified so that the Start-Clear De-bouncer is driven by an SPDT switch instead of DIP switch, similar to that of the Single-Step De-bouncer module. The problem does not work out however applying any or both of these measures. Hence, the board configuration is altered to place an independent CLR signal on the Program Counter module and the Instruction Register module driven by a DIP switch on the respective UP2 board each. CLR input of the Controller/Sequencer module remains being driven by CLR output of the Mode-Select Switches & De-bouncers module.

With all enhancements introduced to the modular SAP-1 prototype, features available on the prototype can be summarized as follows:-

- Supports both auto- and manual-clocked modes.
- Runs at clock frequency of approximately 0.3751Hz in auto-clocked mode.
- 8 x 8 RAM.
- DIP switches for address input during programming of RAM.
- DIP switches for data input during programming of RAM.
- RAM-Mode-Select switches:-
 - o RUN/PROG' (DIP switch)
 - o READ/WRITE' (Push Button)
- Execution-Mode-Select switches:-
 - o START'/CLEAR (DIP switch)

- LOW' – HIGH' (SPDT switch)
- MANUAL'/AUTO (DIP switch)
- 7-segment displays showing register content of the Program Counter, MAR & 2 to 1 MUX, 8×8 RAM, Instruction Register, Accumulator, Adder/Subtractor, B Register, and Output Register modules in hexadecimal representation.
- LEDs displaying logic level of the control signals at the respective modules (*Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo*).
- LEDs displaying logic level of the *CLK* and *CLR* signals.
- LEDs displaying logic level of the T states (*T[1]* through *T[6]*) within machine cycle of SAP-1 computer.
- LEDs displaying logic level of the decoded instruction signal (*LDA, ADD, SUB, OUT, HLT*).

Test program presented in Table 4 is modified slightly so that it fit into the shrunk 8×8 RAM. Instead of storing the data at locations 0xa, 0xb, and 0xc, they are stored at locations 0x5, 0x6, and 0x7. The operand field of the LDA, ADD, and SUB instructions are also changed accordingly. This program is first loaded into the RAM applying similar procedures as discussed in Section 4.2. Applying appropriate logic level to the *MANUAL_AUTO* input signal (low for manual-clocked mode, high for auto-clocked mode), program test run has been carried out for both clocking mode. The SPDT switch connected to the inputs of the Single-Step De-bouncer is used to generate manual *CLK* signal in manual-clocked mode. Kindly refer to Appendix F for details on pin assignments, pin interconnections across boards, and input/output device utilization of all UP2 platforms used in modular SAP-1 prototype.

Table 7 summarizes the observations at each clock state resulting from switching of the Single-Step switch for test program execution in manual-clocked mode. Similar to representations in Table 6, '1' in the Table 6 indicates that LED is on whereas '0' represents the opposite. Note that the active low LEDs on the UP2 platform are driven by inverted signals of *T, LDA, ADD, SUB, OUT, HLT, CLK*, and *CLR* (*Tnot, LDAnot, ADDnot, SUBnot, OUTnot, HLTnot, CLKnot*, and *CLRnot*). For better clarity, logic levels of the non-inverted signals are presented in the table.

**TABLE 7: Detailed observations of test program execution on modular SAP-1
prototype in manual-clocked mode**

Switching Action	CLK	T State	Active Decoded Instruction Signal	Active Control Signals	Register Transfer / Operation	Register Value
0	0	$T[1]$	LDA	Ep, Lm	$R[MAR] \leftarrow R[PC]$	0x0
1	1	$T[2]$		Cp	$R[PC] \leftarrow R[PC+1]$	0x1
2	0					
3	1	$T[3]$		CE, Li	$R[IR] \leftarrow R[RAM]$	0x05
4	0					
5	1	$T[4]$		Ei, Lm	$R[MAR] \leftarrow R[IR]$	0x5
6	0					
7	1	$T[5]$		CE, La	$R[ACC] \leftarrow R[RAM]$	0xaa
8	0					
9	1	$T[6]$		-	-	-
10	0					
11	1	$T[1]$		Ep, Lm	$R[MAR] \leftarrow R[PC]$	0x1
12	0					
13	1	$T[2]$		Cp	$R[PC] \leftarrow R[PC+1]$	0x2
14	0					
15	1	$T[3]$	CE, Li	$R[IR] \leftarrow R[RAM]$	0x16	
16	0					
17	1	$T[4]$	Ei, Lm	$R[MAR] \leftarrow R[IR]$	0x6	
18	0					
19	1	$T[5]$	CE, Lb	$R[BR] \leftarrow R[RAM]$	0x55	
20	0					
21	1	$T[6]$	Eu, La	$R[ACC] \leftarrow R[A/S]$	0xff	
22	0					
23	1	$T[1]$	Ep, Lm	$R[MAR] \leftarrow R[PC]$	0x2	
24	0					
25	1	$T[2]$	Cp	$R[PC] \leftarrow R[PC+1]$	0x3	
26	0					
27	1	$T[3]$	CE, Li	$R[IR] \leftarrow R[RAM]$	0x27	
28	0					
29	1	$T[4]$	Ei, Lm	$R[MAR] \leftarrow R[IR]$	0x7	
30	0					
31	1	$T[5]$	CE, Lb	$R[BR] \leftarrow R[RAM]$	0x0f	
32	0					
33	1	$T[6]$	Su, Eu, La	$R[ACC] \leftarrow R[A/S]$	0xf0	
34	0					
35	1	$T[1]$	Ep, Lm	$R[MAR] \leftarrow R[PC]$	0x3	
36	0					
37	1	$T[2]$	Cp	$R[PC] \leftarrow R[PC+1]$	0x4	
38	0					
39	1	$T[3]$	CE, Li	$R[IR] \leftarrow R[RAM]$	0xe7	
40	0					
41	1	$T[4]$	Ea, Lo	$R[OR] \leftarrow R[ACC]$	0xf0	
42	0					
43	1	$T[5]$	-	-	-	
44	0					
45	1	$T[6]$	-	-	-	
46	0					
47	1	$T[1]$	Ep, Lm	$R[MAR] \leftarrow R[PC]$	0x4	
48	0					
49	1	$T[2]$	Cp	$R[PC] \leftarrow R[PC+1]$	0x5	
50	0					
51	1	$T[3]$	CE, Li	$R[IR] \leftarrow R[RAM]$	0xf7	
52	0					
53	1	$T[4]$	HLT	-	-	-
54	0					
55	0					
56	0					
57	0					

Abbreviations

PC	-	Program Counter	ACC	-	Accumulator
MAR	-	MAR & 2 to 1MUX	A/S	-	Adder/Subtractor
RAM	-	8 × 8 RAM	BR	-	B Register
IR	-	Instruction Register	OR	-	Output Register

From Table 7, it is clear that the modular SAP-1 prototype is able to show logic level of the *CLK* signal, *T[1]* through *T[6]* signals, decoded instruction signals, control signals, as well as all register transfers in the system. All register transfers take place during the positive *CLK* edge of each T state. They are recorded in the table using the following format:-

$$R[\text{destination}] \leftarrow R[\text{source}]$$

The column labelled “Register Value” is used to record value of the register content loaded from the source register to the destination register. For example, content of the Program Counter (0x0) is loaded into the MAR & 2 to 1 MUX at the positive *CLK* edge in between the 1st and 2nd switching action. All register transfers take place through the WBus, except for incremental of the Program Counter count occurring at each T₂ state.

The active control signals remain high throughout the entire T state as observed. Similar to observations in the prototype developed for SAP-1 system synthesis, all instructions are decoded at the positive *CLK* edge halfway through T₃, as soon as the instruction is loaded from the 8 × 8 RAM into the Instruction Register. At the positive *CLK* edge between the 42nd and 43rd switching action, it has been observed that final computational result of 0xf0 is loaded from the Accumulator to the Output Register. This verifies functionality of the prototype as the test program essentially performs computation of 0xaa plus 0x55 minus 0x0f. Behaviour of all other signals has also been found to be correct examining the prototype carefully. Kindly refer to Chapter 10 of [1] *Digital Computer Electronics* by Malvino (1983) to verify test results of this section in detail.

CHAPTER 5

CONCLUSION & RECOMMENDATION

5.1 Conclusion

SAP-1 computer architecture is a good model for introductory computer system architecture understanding. It eliminates advanced functional blocks that are difficult to understand, yet retaining basic components sufficient to introduce all essential concepts in computer operation [1]. Thus necessity of SAP-1 computer models in lab experiments of the university's course can be seen.

Developing the computer system on FPGA, the project provides not only exposure to knowledge on computer system architecture, but also creates learning and practice opportunity of digital system design using HDL programming such as Verilog. Development on FPGA has also higher robustness and ease of debugging compared to TTL circuits. Implementation on FPGA also provides an alternative for learning of computer system through HDL and FPGA.

Project work in Semester 1 generally involves literature studies such as digital electronics fundamentals, SAP-1 architecture, and Verilog HDL, as well as modular and system simulation of SAP-1 computer on ModelSim XE III software. All Verilog programs written for SAP-1 computer have been verified successfully.

System synthesis and modular synthesis of SAP-1 computer on Altera's UP2 platforms have been carried out in Semester 2. Working prototypes have been produced for SAP-1 system on a single UP2 platform, and SAP-1 modular synthesis utilizing 10 UP2 platforms. With proper design, the modular SAP-1 prototype is able to demonstrate operations of SAP-1 computer down to microinstruction level for good understanding of SAP-1 computer system.

5.2 Recommendation

It is recommended that design of SAP-1 system synthesis prototype can be expanded to show more details similar to that of modular SAP-1 prototype. This may be useful if resource of the UP2 development platform is limited.

Apart from that, projects on development of SAP-2 and SAP-3 computer system on FPGA can be offered to students undertaking this Final Year Project course in the future. These computer models features more instructions than SAP-1 computer, useful for enhancing understanding of computer systems with more complex architecture.

REFERENCES

- [1] A. P. Malvino, "*Digital Computer Electronics: An Introduction to Microcomputers*", New York: McGraw-Hill Publishing, 1983.
- [2] S. Thibault, D. Pellerin, "*The FPGA as a Computing Platform - Sample Chapter*", Prentice Hall, 6 May 2005,
<http://www.informit.com/articles/article.asp?p=382614&rl=1>.
- [3] Bluewater Systems Ltd., "*FPGA Design - Introduction*", 2004,
<http://www.bluewaternz.com/consulting/doc/fpga/>.
- [4] Andraka Consulting Group, Inc., "*What is an FPGA?*", 25 Jan 2003,
<http://www.andraka.com/whatisan.htm>.
- [5] Wikipedia Foundation, Inc., "*Field-Programmable Gate Array*", 2007,
<http://en.wikipedia.org/wiki/FPGA>.
- [6] Xilinx, Inc., "*Getting Started with CPLDs*", 2007, http://www.xilinx.com/products/silicon_solutions/cplds/cpld_users_guide/getting_started_with_cplds.htm.
- [7] GlobalSpec, Inc., "*About Complex Programmable Logic Devices (CPLD)*", 2007, http://semiconductors.globalspec.com/LearnMore/Semiconductors/Programmable_Logic_Devices/CPLD.
- [8] Wikipedia Foundation, Inc., "*Complex Programmable Logic Device*", 2007,
<http://en.wikipedia.org/wiki/CPLD>.
- [9] Doulos Ltd, "*What is Verilog? & A Brief History of Verilog*", 2006,
http://www.doulos.com/knowhow/verilog_designers_guide/.
- [10] D. R. Smith, P. D. Franzon, "*Verilog Styles for Synthesis of Digital Systems*", New Jersey: Prentice Hall, 2000.

- [11] W. F. Lee, *"Verilog Coding for Logic Synthesis"*, New Jersey: Wiley-Interscience, 2003.
- [12] Altera Corporation, *"Introduction to Quartus II - Version 6.1"*, 2006, www.altera.com/literature/manual/intro_to_quartus2.pdf.
- [13] Altera Corporation, *"University Program UP2 Education Kit User Guide v3.1"*, Dec 2004, www.altera.com/literature/univ/upds.pdf.

APPENDIX A

MODIFIED BLOCK DIAGRAM OF SAP-1

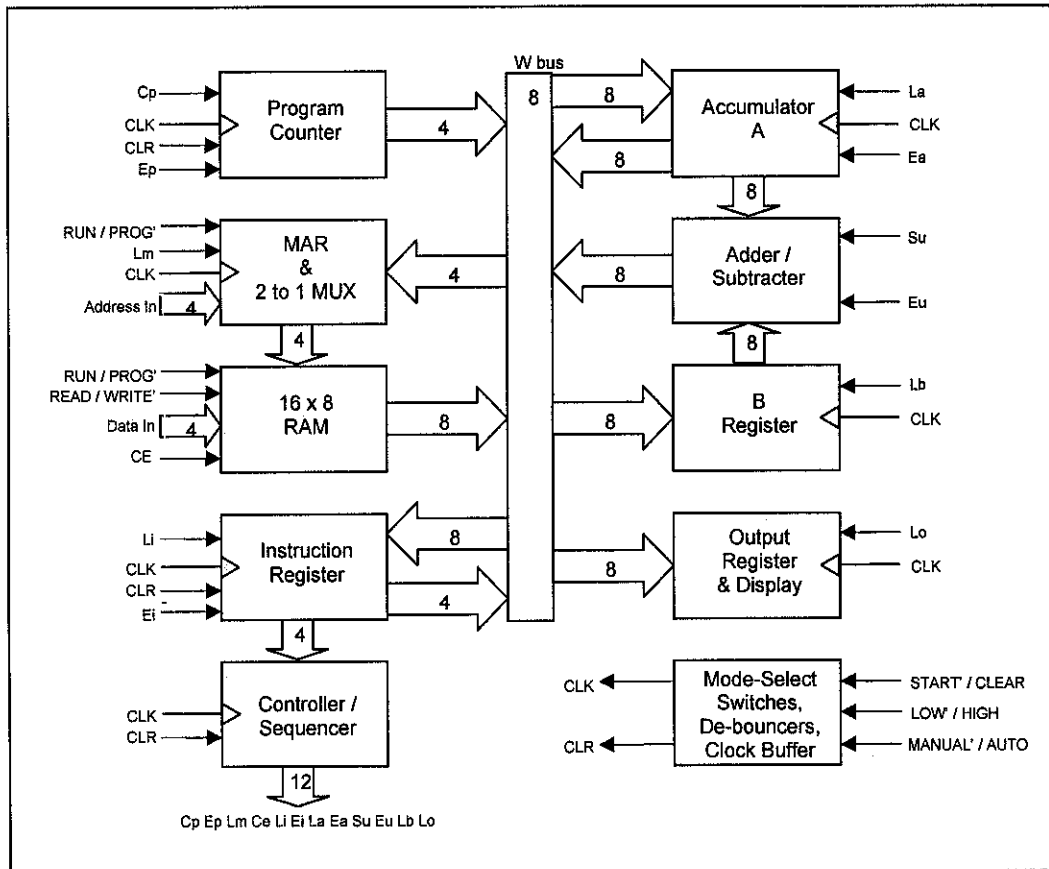


FIGURE 17: Modified block diagram of SAP-1 redrawn from chapter 10 of [1]
Digital Computer Electronics by Malvino (1983)

APPENDIX B **TRUTH TABLE OF BINARY TO HEXADECIMAL** **7-SEGMENT DISPLAY DECODER MODULE**

TABLE 8: Truth table of binary to hexadecimal 7-segment display decoder module
for common anode, active low 7 segment display on
Altera's UP 2 Development Platform

Binary Input	Output (a = MSB, h = LSB = decimal point)							
	a	b	c	d	e	f	g	h
0000	0	0	0	0	0	0	1	1
0001	1	0	0	1	1	1	1	1
0010	0	0	1	0	0	1	0	1
0011	0	0	0	0	1	1	0	1
0100	1	0	0	1	1	0	0	1
0101	0	1	0	0	1	0	0	1
0110	0	1	0	0	0	0	0	1
0111	0	0	0	1	1	1	1	1
1000	0	0	0	0	0	0	0	1
1001	0	0	0	0	1	0	0	1
1010	0	0	0	1	0	0	0	1
1011	1	1	0	0	0	0	0	1
1100	0	1	1	0	0	0	1	1
1101	1	0	0	0	0	1	0	1
1110	0	1	1	0	0	0	0	1
1111	0	1	1	1	0	0	0	1

APPENDIX C

ALTERA'S UNIVERSITY PROGRAM 2

DEVELOPMENT PLATFORM COMPONENT LAYOUT

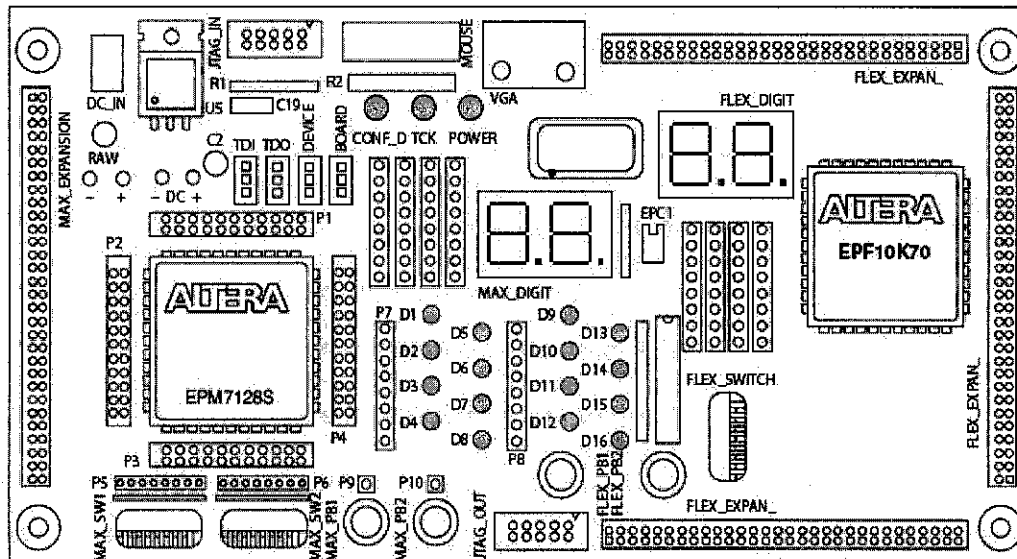


FIGURE 18: Component layout of Altera's University Program 2 Development Platform from page 3 of [13] *"University Program UP2 Education Kit User Guide v3.1"*, Altera Corporation, www.altera.com/literature/univ/upds.pdf

APPENDIX D

SAP-1 SYSTEM SYNTHESIS SOURCE CODE

As compilation of SAP-1 Verilog programs during system synthesis is done in a single project without partitioning them according to modules, duplication of similar modules is not allowed. For example, only D flip-flop module (*D_FF()* in source file *d_flipflop.v*) from the MAR & 2 to 1 MUX module is included in the project. All instantiations of this module within the system will take the same source file. However, for better clarity, the source code is arranged here with duplications as if compilation of independent modules is done.

D-1 Program Counter

D-1.1 Program Counter with Tri-state Output

```
module PROGRAM_COUNTER(PC, WBus, Cp, Ep, CLK, CLR);

// Cp -> high to increment PC count
// Ep -> high to put PC count at WBus

    output [3:0] PC;
    output [7:0] WBus;
    tri [7:0] WBus;
    input Cp, Ep, CLK, CLR;
    wire none;

    JKFF_Q_POSCLK_POSCLR PC0(PC[0], Cp, Cp, CLK, CLR);
    JKFF_Q_POSCLK_POSCLR PC1(PC[1], Cp, Cp, ~PC[0], CLR);
    JKFF_Q_POSCLK_POSCLR PC2(PC[2], Cp, Cp, ~PC[1], CLR);
    JKFF_Q_POSCLK_POSCLR PC3(PC[3], Cp, Cp, ~PC[2], CLR);

    assign none = 0;

    bufifl(WBus[0], PC[0], Ep);
    bufifl(WBus[1], PC[1], Ep);
    bufifl(WBus[2], PC[2], Ep);
    bufifl(WBus[3], PC[3], Ep);
    bufifl(WBus[4], none, none);
    bufifl(WBus[5], none, none);
    bufifl(WBus[6], none, none);
    bufifl(WBus[7], none, none);

endmodule
```

D-1.2 JK Flip-flop

```
module JKFF_Q_POSCLK_POSCLR(Q, J, K, CLK, CLR);

    input J, K, CLK, CLR;
    output Q;
    reg Q;
```



```

always @(posedge CLK or posedge CLR)
begin
    if (CLR) Q = 1'b0;
    else if ((J,K) == 2'b00) Q = Q;
    else if ((J,K) == 2'b01) Q = 1'b0;
    else if ((J,K) == 2'b10) Q = 1'b1;
    else if ((J,K) == 2'b11) Q = ~Q;
end
endmodule

```

D-2 MAR & 2 to 1 Multiplexer

D-2.1 MAR + 2 to 1 Multiplexer

```

module MAR_MUX(MUX_OUT, ADDR_IN, WBus, Lm, CLK, RUN_PROG);

    input Lm, CLK, RUN_PROG;
    input [3:0] ADDR_IN;
    input [3:0] WBus;
    wire [3:0] MAR_OUT;
    output [3:0] MUX_OUT;

    MAR MAR_MUX_1(MAR_OUT, WBus, Lm, CLK);

    MUX MAR_MUX_2(MUX_OUT, MAR_OUT, ADDR_IN, RUN_PROG);

endmodule

```

D-2.2 MAR

```

module MAR(MAR_OUT, WBus, Lm, CLK);

// Lm -> high to load PC count from WBus

    input [3:0] WBus;
    input Lm, CLK;
    output [3:0] MAR_OUT;

    D_FF MAR0(MAR_OUT[0], WBus[0], Lm, CLK);
    D_FF MAR1(MAR_OUT[1], WBus[1], Lm, CLK);
    D_FF MAR2(MAR_OUT[2], WBus[2], Lm, CLK);
    D_FF MAR3(MAR_OUT[3], WBus[3], Lm, CLK);

endmodule

```

D-2.3 2 to 1 Multiplexer

```

module MUX(MUX_OUT, MAR_OUT, ADDR_IN, RUN_PROG);

// RUN_PROG == 0 -> program
//                -> address from switches taken
// RUN_PROG == 1 -> run
//                -> address from MAR output taken

    input [3:0] MAR_OUT, ADDR_IN;
    input RUN_PROG;
    output [3:0] MUX_OUT;

    assign MUX_OUT = RUN_PROG? MAR_OUT : ADDR_IN;

endmodule

```

```
endmodule
```

D-2.4 D Flip-flop with Enable Input for Data Loading

```
module D_FF(Q,D,EN,CLK);

    input D,EN,CLK;
    output Q;
    reg Q;

    always @(posedge CLK)
        if (EN == 1)
            Q = D;
        else
            Q = Q;

endmodule
```

D-3 16 × 8 RAM

D-3.1 16 × 8 RAM

```
module _16x8_RAM(WBus, DATA_OUT, RAM_DISPLAY_HEX,
                MUX_OUT, DATA_IN, CE, RUN_PROG, READ_WRITE);

    // RUN_PROG == 0 --> PROG;      RUN_PROG == 1 --> RUN
    // READ_WRITE == 0 --> WRITE;   READ_WRITE == 1 --> READ

    input [3:0] MUX_OUT;
    input [7:0] DATA_IN;
    input CE, RUN_PROG, READ_WRITE;
    output [7:0] WBus, DATA_OUT;
    output [15:0] RAM_DISPLAY_HEX;
    wire [7:0] RAM_DISPLAY;

    tri [7:0] WBus;
    reg [7:0] DATA_OUT;
    reg [7:0] MEM [15:0];

    always @ (MUX_OUT or DATA_IN or RUN_PROG or READ_WRITE)
    begin

        if (!RUN_PROG)
        begin
            if (!READ_WRITE)
                MEM [MUX_OUT] = DATA_IN;
            end

        else if (RUN_PROG)
            DATA_OUT = MEM [MUX_OUT];

        else;

    end

    assign RAM_DISPLAY = MEM [MUX_OUT];

    HEX_DISPLAY RAM_HEX_1
        (RAM_DISPLAY_HEX[15:8],RAM_DISPLAY[7:4]);
    HEX_DISPLAY RAM_HEX_2
        (RAM_DISPLAY_HEX[7:0],RAM_DISPLAY[3:0]);

    bufif1(WBus[0],DATA_OUT[0],CE);
    bufif1(WBus[1],DATA_OUT[1],CE);
    bufif1(WBus[2],DATA_OUT[2],CE);
    bufif1(WBus[3],DATA_OUT[3],CE);
```

```

        bufifl(WBus[4],DATA_OUT[4],CE);
        bufifl(WBus[5],DATA_OUT[5],CE);
        bufifl(WBus[6],DATA_OUT[6],CE);
        bufifl(WBus[7],DATA_OUT[7],CE);

endmodule

```

D-3.2 Hexadecimal Display

```

module HEX_DISPLAY(HEX_OUT,HEX_IN);

// HEX_OUT -> signals a-h of active low hex 7-segment display
//          MSB = a, LSB = h (decimal point)

    output [7:0] HEX_OUT;
    input [3:0] HEX_IN;
    reg [7:0] HEX_OUT;

    always @(HEX_IN)
        case(HEX_IN)
            4'b0001 : HEX_OUT = 8'b10011111;
            4'b0010 : HEX_OUT = 8'b00100101;
            4'b0011 : HEX_OUT = 8'b00001101;
            4'b0100 : HEX_OUT = 8'b10011001;
            4'b0101 : HEX_OUT = 8'b01001001;
            4'b0110 : HEX_OUT = 8'b01000001;
            4'b0111 : HEX_OUT = 8'b00011111;
            4'b1000 : HEX_OUT = 8'b00000001;
            4'b1001 : HEX_OUT = 8'b00001001;
            4'b1010 : HEX_OUT = 8'b00010001;
            4'b1011 : HEX_OUT = 8'b11000001;
            4'b1100 : HEX_OUT = 8'b01100011;
            4'b1101 : HEX_OUT = 8'b10000101;
            4'b1110 : HEX_OUT = 8'b01100001;
            4'b1111 : HEX_OUT = 8'b01110001;
            default : HEX_OUT = 8'b00000011;
        endcase

endmodule

```

D-4 Instruction Register

D-4.1 Instruction Register with Tri-state Address Output

```

module INSTRUCTION_REGISTER(IR_OUT_ADDR, IR_OUT_INS, WBus, Li, Ei, CLK, CLR);

// Li -> high to load instruction from WBus
// Ei -> high to put lower nibble of instruction
//      (data address field) at WBus
//
// IR_OUT_ADDR -> data address field of instruction
// IR_OUT_INS -> instruction opcode field (upper nibble)

    input Li, Ei, CLK, CLR;
    inout [7:0] WBus;
    output [3:0] IR_OUT_ADDR, IR_OUT_INS;
    tri [7:0] WBus;

    D_FF IR0(IR_OUT_ADDR[0],WBus[0],Li,CLK);
    D_FF IR1(IR_OUT_ADDR[1],WBus[1],Li,CLK);
    D_FF IR2(IR_OUT_ADDR[2],WBus[2],Li,CLK);
    D_FF IR3(IR_OUT_ADDR[3],WBus[3],Li,CLK);
    DFF_POSCLR IR4(IR_OUT_INS[0],WBus[4],Li,CLK,CLR);
    DFF_POSCLR IR5(IR_OUT_INS[1],WBus[5],Li,CLK,CLR);
    DFF_POSCLR IR6(IR_OUT_INS[2],WBus[6],Li,CLK,CLR);

```

```

DFF_POSCLR IR7(IR_OUT_INS[3],WBus[7],Li,CLK,CLR);

bufifl(WBus[0],IR_OUT_ADDR[0],Ei);
bufifl(WBus[1],IR_OUT_ADDR[1],Ei);
bufifl(WBus[2],IR_OUT_ADDR[2],Ei);
bufifl(WBus[3],IR_OUT_ADDR[3],Ei);

endmodule

```

D-4.2 D Flip-flop with Enable Input for Data Loading

```

module D_FF(Q,D,EN,CLK);

input D,EN,CLK;
output Q;
reg Q;

always @(posedge CLK)
if (EN == 1)
Q = D;
else
Q = Q;

endmodule

```

D-4.3 D Flip-flop with Data Loading Enable & Clear Input

```

module DFF_POSCLR(Q,D,EN,CLK,CLR);

input D,EN,CLK,CLR;
output Q;
reg Q;

always @(posedge CLK or posedge CLR)
if (CLR == 1)
Q = 0;
else if (EN == 1)
Q = D;
else
Q = Q;

endmodule

```

D-5 Accumulator

D-5.1 Accumulator

```

module ACCUMULATOR(ACCU_OUT, WBus, La, Ea, CLK);

// La -> high to load data from WBus
// Ea -> high to put accumulator content ot WBus

input La, Ea, CLK;
inout [7:0] WBus;
output [7:0] ACCU_OUT;
tri [7:0] WBus;

D_FF ACCU0(ACCU_OUT[0],WBus[0],La,CLK);
D_FF ACCU1(ACCU_OUT[1],WBus[1],La,CLK);
D_FF ACCU2(ACCU_OUT[2],WBus[2],La,CLK);
D_FF ACCU3(ACCU_OUT[3],WBus[3],La,CLK);
D_FF ACCU4(ACCU_OUT[4],WBus[4],La,CLK);

```

```

D_FF ACCU5 (ACCU_OUT[5], WBus[5], La, CLK);
D_FF ACCU6 (ACCU_OUT[6], WBus[6], La, CLK);
D_FF ACCU7 (ACCU_OUT[7], WBus[7], La, CLK);

bufifl1(WBus[0], ACCU_OUT[0], Ea);
bufifl1(WBus[1], ACCU_OUT[1], Ea);
bufifl1(WBus[2], ACCU_OUT[2], Ea);
bufifl1(WBus[3], ACCU_OUT[3], Ea);
bufifl1(WBus[4], ACCU_OUT[4], Ea);
bufifl1(WBus[5], ACCU_OUT[5], Ea);
bufifl1(WBus[6], ACCU_OUT[6], Ea);
bufifl1(WBus[7], ACCU_OUT[7], Ea);

endmodule

```

D-5.2 D Flip-flop with Enable Input for Data Loading

```

module D_FF(Q,D,EN,CLK);

    input D,EN,CLK;
    output Q;
    reg Q;

    always @(posedge CLK)
        if (EN == 1)
            Q = D;
        else
            Q = Q;

endmodule

```

D-6 Adder/Subtractor

D-6.1 Adder/Subtractor

```

module ADD_SUB(ADD_SUB_OUT, WBus, ACCU_OUT, B_REG_OUT, Su, Eu);

    // Su -> subtraction operation enable bit, high to convert
    //      B Register output to 2's complement form
    //
    // Eu -> high to put arithmetic operation result on WBus
    //
    // ADD_SUB_OUT -> Adder/Subtractor's output, arithmetic
    //               operation result
    //
    // ACCU_OUT -> Accumulator's output, operand of arithmetic
    //             operation
    //
    // B_REG_OUT -> B Register's output
    //
    // B -> temporary register to hold operand of arithmetic
    //      operation
    // == B_REG_OUT for addition operation
    // == ~B_REG_OUT for subtraction operation
    // (added with 1 (Su) for 2's complement of B_REG_OUT)

    output [7:0] ADD_SUB_OUT, WBus;
    input [7:0] ACCU_OUT, B_REG_OUT;
    input Su, Eu;
    reg [7:0] ADD_SUB_OUT,B;
    tri [7:0] WBus;

    always @ (ACCU_OUT or B_REG_OUT or Su)
        begin
            if (Su)

```

```

        begin B = ~(B_REG_OUT); end
    else
        begin B = B_REG_OUT;      end
    ADD_SUB_OUT = ACCU_OUT + B + Su;
end

bufifl(WBus[0],ADD_SUB_OUT[0],Eu);
bufifl(WBus[1],ADD_SUB_OUT[1],Eu);
bufifl(WBus[2],ADD_SUB_OUT[2],Eu);
bufifl(WBus[3],ADD_SUB_OUT[3],Eu);
bufifl(WBus[4],ADD_SUB_OUT[4],Eu);
bufifl(WBus[5],ADD_SUB_OUT[5],Eu);
bufifl(WBus[6],ADD_SUB_OUT[6],Eu);
bufifl(WBus[7],ADD_SUB_OUT[7],Eu);

endmodule

```

D-7 B Register

D-7.1 B Register

```

module B_REGISTER(B_REG_OUT, WBus, Lb, CLK);

// Lb -> high to load data from WBus

    output [7:0] B_REG_OUT;
    input [7:0] WBus;
    input Lb, CLK;

    D_FF B_REG_0(B_REG_OUT[0],WBus[0],Lb,CLK);
    D_FF B_REG_1(B_REG_OUT[1],WBus[1],Lb,CLK);
    D_FF B_REG_2(B_REG_OUT[2],WBus[2],Lb,CLK);
    D_FF B_REG_3(B_REG_OUT[3],WBus[3],Lb,CLK);
    D_FF B_REG_4(B_REG_OUT[4],WBus[4],Lb,CLK);
    D_FF B_REG_5(B_REG_OUT[5],WBus[5],Lb,CLK);
    D_FF B_REG_6(B_REG_OUT[6],WBus[6],Lb,CLK);
    D_FF B_REG_7(B_REG_OUT[7],WBus[7],Lb,CLK);

endmodule

```

D-7.2 D Flip-flop with Enable Input for Data Loading

```

module D_FF(Q,D,EN,CLK);

    input D,EN,CLK;
    output Q;
    reg Q;

    always @(posedge CLK)
        if (EN == 1)
            Q = D;
        else
            Q = Q;

endmodule

```

D-8 Output Register

D-8.1 Output Register

```
module OUTPUT_REGISTER(OUT_REG, OUT_REG_HEX, WBus, Lo, CLK);

// Lo -> high to load data from WBus

    output [7:0] OUT_REG;
    output [15:0] OUT_REG_HEX;
    input [7:0] WBus;
    input Lo, CLK;

    D_FF OUT_REG_0(OUT_REG[0],WBus[0],Lo,CLK);
    D_FF OUT_REG_1(OUT_REG[1],WBus[1],Lo,CLK);
    D_FF OUT_REG_2(OUT_REG[2],WBus[2],Lo,CLK);
    D_FF OUT_REG_3(OUT_REG[3],WBus[3],Lo,CLK);
    D_FF OUT_REG_4(OUT_REG[4],WBus[4],Lo,CLK);
    D_FF OUT_REG_5(OUT_REG[5],WBus[5],Lo,CLK);
    D_FF OUT_REG_6(OUT_REG[6],WBus[6],Lo,CLK);
    D_FF OUT_REG_7(OUT_REG[7],WBus[7],Lo,CLK);

    HEX_DISPLAY OUT_HEX_1
        (OUT_REG_HEX[15:8],OUT_REG[7:4]);
    HEX_DISPLAY OUT_HEX_2
        (OUT_REG_HEX[7:0],OUT_REG[3:0]);

endmodule
```

D-8.2 D Flip-flop with Enable Input for Data Loading

```
module D_FF(Q,D,EN,CLK);

    input D,EN,CLK;
    output Q;
    reg Q;

    always @(posedge CLK)
        if (EN == 1)
            Q = D;
        else
            Q = Q;

endmodule
```

D-8.3 Hexadecimal Display

```
module HEX_DISPLAY(HEX_OUT,HEX_IN);

// HEX_OUT -> signals a-g of hex 7-segment display
//          MSB = a, LSB = g

    output [6:0] HEX_OUT;
    input [3:0] HEX_IN;
    reg [6:0] HEX_OUT;

    always @(HEX_IN)
        case(HEX_IN)
            4'b0001 : HEX_OUT = 7'b0110000;
            4'b0010 : HEX_OUT = 7'b1101101;
            4'b0011 : HEX_OUT = 7'b1111001;
            4'b0100 : HEX_OUT = 7'b0110011;
            4'b0101 : HEX_OUT = 7'b1011011;
            4'b0110 : HEX_OUT = 7'b1011111;
            4'b0111 : HEX_OUT = 7'b1110000;
            4'b1000 : HEX_OUT = 7'b1111111;
            4'b1001 : HEX_OUT = 7'b1111011;
```

```

        4'b1010 : HEX_OUT = 7'b1110111;
        4'b1011 : HEX_OUT = 7'b0011111;
        4'b1100 : HEX_OUT = 7'b1001110;
        4'b1101 : HEX_OUT = 7'b0111101;
        4'b1110 : HEX_OUT = 7'b1001111;
        4'b1111 : HEX_OUT = 7'b1000111;
        default : HEX_OUT = 7'b1111110;
    endcase

endmodule

```

D-9 Controller/Sequencer (Instruction Decoder, Ring Counter & Control Matrix)

D-9.1 Controller/Sequencer

```

module CONTROLLER_SEQUENCER
    (Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo,
     LDAnot, ADDnot, SUBnot, OUTnot, HLTnot,
     HLT, IR_OUT_INS, T, CLK, CLR);

    output Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo,
           LDAnot, ADDnot, SUBnot, OUTnot, HLTnot, HLT;
    output [6:1] T;
    input CLK, CLR;
    input [3:0] IR_OUT_INS;
    wire LDA, ADD, SUB, OUT;
    wire [6:1] Tnot;

    RING_COUNTER RC(T, Tnot, CLK, CLR);
    INSTRUCTION_DECODER ID(LDA, ADD, SUB, OUT, HLT, IR_OUT_INS);
    CONTROL_MATRIX CM(Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo,
                     LDA, ADD, SUB, OUT, HLT, T, CLK, CLR);

    assign LDAnot = ~LDA,
           ADDnot = ~ADD,
           SUBnot = ~SUB,
           OUTnot = ~OUT,
           HLTnot = ~HLT;

endmodule

```

D-9.2 Instruction Decoder

```

module INSTRUCTION_DECODER(LDA, ADD, SUB, OUT, HLT, IR_OUT_INS);

    // associate control line of each routine with their
    // corresponding opcode

    output LDA, ADD, SUB, OUT, HLT;
    input [3:0] IR_OUT_INS;

    assign LDA = (IR_OUT_INS == 4'b0000)? 1'b1 : 1'b0,
           ADD = (IR_OUT_INS == 4'b0001)? 1'b1 : 1'b0,
           SUB = (IR_OUT_INS == 4'b0010)? 1'b1 : 1'b0,
           OUT = (IR_OUT_INS == 4'b1110)? 1'b1 : 1'b0,
           HLT = (IR_OUT_INS == 4'b1111)? 1'b1 : 1'b0;

endmodule

```


D-9.3 Ring Counter

```
module RING_COUNTER(T, Tnot, CLK, CLR);

// CLR high -> Ring Counter resets to 000001
//
// Ring Counter shifts left at each negative
// edge of CLK

output [6:1] T, Tnot;
input CLK, CLR;

JKFF_Qnot_NEGCLK_POSCLR RC1
(Tnot[1], T[1], Tnot[6], T[6], CLK, CLR);
JKFF_Qnot_NEGCLK_POSCLR RC2
(T[2], Tnot[2], T[1], Tnot[1], CLK, CLR);
JKFF_Qnot_NEGCLK_POSCLR RC3
(T[3], Tnot[3], T[2], Tnot[2], CLK, CLR);
JKFF_Qnot_NEGCLK_POSCLR RC4
(T[4], Tnot[4], T[3], Tnot[3], CLK, CLR);
JKFF_Qnot_NEGCLK_POSCLR RC5
(T[5], Tnot[5], T[4], Tnot[4], CLK, CLR);
JKFF_Qnot_NEGCLK_POSCLR RC6
(T[6], Tnot[6], T[5], Tnot[5], CLK, CLR);

endmodule
```

D-9.4 Control Matrix

```
module CONTROL_MATRIX(Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo,
                      LDA, ADD, SUB, OUT, HLT, T, CLK, CLR);

output Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo;
input LDA, ADD, SUB, OUT, HLT, CLK, CLR;
input [6:1] T;

assign Cp = (T[2])? 1'b1 : 1'b0,
       Ep = (T[1])? 1'b1 : 1'b0,
       Lm = (T[1] || (LDA && T[4]) || (ADD && T[4]) || (SUB && T[4]))?
           1'b1 : 1'b0,
       CE = (T[3] || (LDA && T[5]) || (ADD && T[5]) || (SUB && T[5]))?
           1'b1 : 1'b0,
       Li = (T[3])? 1'b1 : 1'b0,
       Ei = ((LDA && T[4]) || (ADD && T[4]) || (SUB && T[4]))?
           1'b1 : 1'b0,
       La = ((LDA && T[5]) || (ADD && T[6]) || (SUB && T[6]))?
           1'b1 : 1'b0,
       Ea = (OUT && T[4])? 1'b1 : 1'b0,
       Su = (SUB && T[6])? 1'b1 : 1'b0,
       Eu = ((ADD && T[6]) || (SUB && T[6]))? 1'b1 : 1'b0,
       Lb = ((ADD && T[5]) || (SUB && T[5]))? 1'b1 : 1'b0,
       Lo = (OUT && T[4])? 1'b1 : 1'b0;

endmodule
```

D-9.5 Positive-edge-triggered JK Flip-flop with Active High Clear

```
module JKFF(Q,Qnot,J,K,CLK,CLR);

input J,K,CLK,CLR;
output Q,Qnot;
reg Q,Qnot;

always @(negedge CLK or posedge CLR)
begin
    if (CLR)
    begin
        Q = 1'b0;
        Qnot = ~Q;
    end
    else if ({J,K} == 2'b00)
    begin
        Q = Q;
        Qnot = ~Q;
    end
    else if ({J,K} == 2'b01)
    begin
        Q = 1'b0;
        Qnot = ~Q;
    end
    else if ({J,K} == 2'b10)
    begin
        Q = 1'b1;
        Qnot = ~Q;
    end
    else if ({J,K} == 2'b11)
    begin
        Q = ~Q;
        Qnot = ~Q;
    end
end
endmodule
```

D-10 Mode-Select Switches, De-bouncers & Clock Buffer

D-10.1 De-bouncers

```
// START_CLEAR switch -> START active low
//                               CLEAR active high
// MANUAL_AUTO switch -> MANUAL active low
//                               AUTO active high
// LOW -> LOW manual CLK signal for logic 0 input
// HIGH -> HIGH manual CLK signal for logic 0 input
// **All switches gives logic low when pressed**]
//
// DB_TEMP1 -> clock signal in MANUAL mode
// DB_TEMP2 -> clock signal in AUTO mode

module DEBOUNCERS(CLK, CLR, rawCLK, START_CLEAR, LOW, HIGH,
                 MANUAL_AUTO, HLT, rawrawCLK);

output CLK;
input START_CLEAR, LOW_HIGH, LOW_HIGH_not, MANUAL_AUTO, HLT,
      rawrawCLK;
inout CLR, rawCLK;
wire CLRnot, HI_LOnot, HI_LO, MANUAL_OUT, AUTO_OUT,
      DB_TEMP1, DB_TEMP2;

CLEAR_START    DB1(CLR, START_CLEAR);
SINGLE_STEP     DB2(HI_LO, LOW, HIGH);
AUTO_MANUAL    DB3(MANUAL_OUT, AUTO_OUT, MANUAL_AUTO);
```

```

        CLOCK          DB4(rawCLK, HLT, rawrawCLK, CLR);

    and    DB5(DB_TEMP1, ~HLT, HI_LO, MANUAL_OUT),
          DB6(DB_TEMP2, AUTO_OUT, rawCLK);

    or     DB7(CLK, DB_TEMP1, DB_TEMP2);

endmodule

```

D-10.2 Clear-Start De-bouncer

```

module CLEAR_START(CLR, START_CLEAR);

    output CLR;
    input START_CLEAR;
    wire CLRnot, START_CLEAR_NOT;

    not g1(START_CLEAR_NOT, START_CLEAR);

    SR_LATCH CS(CLRnot, CLR, START_CLEAR, START_CLEAR_NOT);

endmodule

```

D-10.3 Single-Step De-bouncer

```

module SINGLE_STEP(HI_LO, LOW, HIGH);

    output HI_LO;
    input LOW, HIGH;
    wire HI_LOnot;

    SR_LATCH SS(HI_LOnot, HI_LO, LOW, HIGH);

endmodule

```

D-10.4 Manual-Auto De-bouncer

```

module AUTO_MANUAL(MANUAL_OUT, AUTO_OUT, MANUAL_AUTO);

    output MANUAL_OUT, AUTO_OUT;
    input MANUAL_AUTO;
    wire MANUAL_AUTO_NOT;

    not g1(MANUAL_AUTO_NOT, MANUAL_AUTO);

    SR_LATCH
        AM(MANUAL_OUT, AUTO_OUT, MANUAL_AUTO, MANUAL_AUTO_NOT);

endmodule

```

D-10.5 Clock Buffer

```

module CLOCK(rawCLK, HLT, rawrawCLK, CLR);

    output rawCLK;
    input HLT, rawrawCLK, CLR;

    JKFF_Q_POSCLK_POSCLR Clock1
        (rawCLK, ~HLT, ~HLT, rawrawCLK, CLR);

endmodule

```

D-10.6 Active Low SR Latch

```
module SR_LATCH(Q,Qnot,Snot,Rnot);

    output Q, Qnot;
    input Snot, Rnot;

    or g1(Q,~Snot,~Qnot),
        g2(Qnot,~Rnot,~Q);

endmodule
```

D-10.7 Positive-edge-triggered JK Flip-flop with Active High Clear

```
module JKFF_Q_POSCLK_POSCLR(Q,J,K,CLK,CLR);

    input J,K,CLK,CLR;
    output Q;
    reg Q;

    always @(posedge CLK or posedge CLR)
        begin
            if (CLR) Q = 1'b0;
            else if ({J,K} == 2'b00) Q = Q;
            else if ({J,K} == 2'b01) Q = 1'b0;
            else if ({J,K} == 2'b10) Q = 1'b1;
            else if ({J,K} == 2'b11) Q = ~Q;
        end

endmodule
```

D-11 SAP-1

D-11.1 SAP-1

```
module SAP1(RUN_PROG, READ_WRITE, START_CLEAR, LOW, HIGH,
    MANUAL_AUTO, rawrawCLK, ADDR_IN, DATA_IN,
    CLK, CLR, Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo,
    HLT, WBus,
    LDAnot, ADDnot, SUBnot, OUTnot, HLTnot, CLKnot, CLRnot,
    MUX_OUT, Tnot, OUT_REG, DIGIT_DISPLAY);

    input RUN_PROG, READ_WRITE, START_CLEAR, LOW, HIGH,
        MANUAL_AUTO, rawrawCLK;
    input [3:0] ADDR_IN;
    input [7:0] DATA_IN;

    inout CLK, CLR, Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo, HLT;
    inout [7:0] WBus;

    output LDAnot, ADDnot, SUBnot, OUTnot, HLTnot, CLKnot, CLRnot;
    output [3:0] MUX_OUT;
    output [6:1] Tnot;
    output [7:0] OUT_REG;
    output [15:0] DIGIT_DISPLAY;

    wire rawCLK;
    wire [3:0] PC, IR_OUT_ADDR, IR_OUT_INS;
    wire [6:1] T;
    wire [7:0] DATA_OUT, ACCU_OUT, ADD_SUB_OUT, B_REG_OUT;
    wire [15:0] OUT_REG_HEX, RAM_DISPLAY_HEX;

    tri [7:0] WBus;
```

```

PROGRAM_COUNTER
    SAP1_01(PC, WBus, Cp, Ep, CLK, CLR);

MAR_MUX
    SAP1_02(MUX_OUT, ADDR_IN, WBus, Lm, CLK, RUN_PROG);

_16x8_RAM
    SAP1_03(WBus, DATA_OUT, RAM_DISPLAY_HEX,
            MUX_OUT, DATA_IN, CE, RUN_PROG, READ_WRITE);

INSTRUCTION_REGISTER
    SAP1_04(IR_OUT_ADDR, IR_OUT_INS, WBus, Li, Ei, CLK, CLR);

ACCUMULATOR
    SAP1_05(ACCU_OUT, WBus, La, Ea, CLK);

ADD_SUB
    SAP1_06(ADD_SUB_OUT, WBus, ACCU_OUT, B_REG_OUT, Su, Eu);

B_REGISTER
    SAP1_07(B_REG_OUT, WBus, Lb, CLK);

OUTPUT_REGISTER
    SAP1_08(OUT_REG, OUT_REG_HEX, WBus, Lo, CLK);

CONTROLLER_SEQUENCER
    SAP1_09(Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo,
            LDAnot, ADDnot, SUBnot, OUTnot, HLTnot,
            HLT, IR_OUT_INS, T, CLK, CLR);

DEBOUNCERS
    SAP1_10(CLK, CLR, rawCLK, START_CLEAR, LOW, HIGH,
            MANUAL_AUTO, HLT, rawrawCLK);

assign DIGIT_DISPLAY = RUN_PROG? OUT_REG_HEX : RAM_DISPLAY_HEX,
        Tnot = ~T,
        CLKnot = ~CLK,
        CLRnot = ~CLR;

endmodule

```

D-12 Additional Hexadecimal Display of MAR & 2 to 1 MUX Output on MAX 7000S Device

D-12.1 Additional Hexadecimal Display of SAP-1 on MAX7000S Device

```

module SAP1_MAX(MUX_OUT_HEX, BLANK_DISPLAY, MUX_OUT);

    output [7:0] MUX_OUT_HEX, BLANK_DISPLAY;
    input [3:0] MUX_OUT;

    HEX_DISPLAY SAP1_MAX_DISPLAY(MUX_OUT_HEX, MUX_OUT);

    assign BLANK_DISPLAY = 8'hff;

endmodule

```

D-12.1 Hexadecimal Display

```

module HEX_DISPLAY(HEX_OUT, HEX_IN);

    // HEX_OUT -> signals a-g of hex 7-segment display
    //          MSB = a, LSB = g

```

```

output [6:0] HEX_OUT;
input [3:0] HEX_IN;
reg [6:0] HEX_OUT;

always @ (HEX_IN)
  case (HEX_IN)
    4'b0001 : HEX_OUT = 7'b0110000;
    4'b0010 : HEX_OUT = 7'b1101101;
    4'b0011 : HEX_OUT = 7'b1111001;
    4'b0100 : HEX_OUT = 7'b0110011;
    4'b0101 : HEX_OUT = 7'b1011011;
    4'b0110 : HEX_OUT = 7'b1011111;
    4'b0111 : HEX_OUT = 7'b1110000;
    4'b1000 : HEX_OUT = 7'b1111111;
    4'b1001 : HEX_OUT = 7'b1111011;
    4'b1010 : HEX_OUT = 7'b1110111;
    4'b1011 : HEX_OUT = 7'b0011111;
    4'b1100 : HEX_OUT = 7'b1001110;
    4'b1101 : HEX_OUT = 7'b0111101;
    4'b1110 : HEX_OUT = 7'b1001111;
    4'b1111 : HEX_OUT = 7'b1000111;
    default : HEX_OUT = 7'b1111110;
  endcase
endmodule

```

APPENDIX E

SAP-1 MODULAR SYNTHESIS SOURCE CODE

E-1 Program Counter

E-1.1 Program Counter with Tri-state Output

```

module PROGRAM_COUNTER(PC,PC_HEX,BLANK_DISPLAY,WBus,Cp,Ep,CLK,CLR,Cpnot,Epnnot);

// Cp -> high to increment PC count
// Ep -> high to put PC count at WBus

    output [3:0] PC;
    output [7:0] WBus,PC_HEX,BLANK_DISPLAY;
    output Cpnot,Epnnot;
    tri [7:0] WBus;
    input Cp,Ep,CLK,CLR;
    wire none;

    JKFF_Q_POSCLK_POSCLR PC0(PC[0],Cp,Cp,CLK,CLR);
    JKFF_Q_POSCLK_POSCLR PC1(PC[1],Cp,Cp,~PC[0],CLR);
    JKFF_Q_POSCLK_POSCLR PC2(PC[2],Cp,Cp,~PC[1],CLR);
    JKFF_Q_POSCLK_POSCLR PC3(PC[3],Cp,Cp,~PC[2],CLR);

    assign none = 0;

    bufif1(WBus[0],PC[0],Ep);
    bufif1(WBus[1],PC[1],Ep);
    bufif1(WBus[2],PC[2],Ep);
    bufif1(WBus[3],PC[3],Ep);
    bufif1(WBus[4],none,none);
    bufif1(WBus[5],none,none);
    bufif1(WBus[6],none,none);
    bufif1(WBus[7],none,none);

    HEX_DISPLAY PCHEx(PC_HEX,PC);

    assign BLANK_DISPLAY = 8'b11111111,
           Cpnot = ~Cp,
           Epnot = ~Ep;

endmodule

```

E-1.2 JK Flip-flop

```

module JKFF_Q_POSCLK_POSCLR(Q,J,K,CLK,CLR);

    input J,K,CLK,CLR;
    output Q;
    reg Q;

    always @(posedge CLK or posedge CLR)
    begin
        if (CLR) Q = 1'b0;
        else if ({J,K} == 2'b00) Q = Q;
        else if ({J,K} == 2'b01) Q = 1'b0;
        else if ({J,K} == 2'b10) Q = 1'b1;
        else if ({J,K} == 2'b11) Q = ~Q;
    end

endmodule

```

E-1.3 Hexadecimal Display

```
module HEX_DISPLAY(HEX_OUT,HEX_IN);

// HEX_OUT -> signals a-h of active low hex 7-segment display
//          MSB = a, LSB = h (decimal point)

output [7:0] HEX_OUT;
input [3:0] HEX_IN;
reg [7:0] HEX_OUT;

always @(HEX_IN)
begin
    case(HEX_IN)
        4'b0001 : HEX_OUT = 8'b10011111;
        4'b0010 : HEX_OUT = 8'b00100101;
        4'b0011 : HEX_OUT = 8'b00001101;
        4'b0100 : HEX_OUT = 8'b10011001;
        4'b0101 : HEX_OUT = 8'b01001001;
        4'b0110 : HEX_OUT = 8'b01000001;
        4'b0111 : HEX_OUT = 8'b00011111;
        4'b1000 : HEX_OUT = 8'b00000001;
        4'b1001 : HEX_OUT = 8'b00001001;
        4'b1010 : HEX_OUT = 8'b00010001;
        4'b1011 : HEX_OUT = 8'b11000001;
        4'b1100 : HEX_OUT = 8'b01100011;
        4'b1101 : HEX_OUT = 8'b10000101;
        4'b1110 : HEX_OUT = 8'b01100001;
        4'b1111 : HEX_OUT = 8'b01110001;
        default : HEX_OUT = 8'b00000011;
    endcase
end
endmodule
```

E-2 MAR & 2 to 1 Multiplexer

E-2.1 MAR + 2 to 1 Multiplexer

```
module MAR_MUX(MUX_OUT, MUX_OUT_HEX, BLANK_DISPLAY, HighZ,
               ADDR_IN, WBus, Lm, CLK, RUN_PROG, Lmnot);

input Lm, CLK, RUN_PROG;
input [2:0] ADDR_IN;
inout [7:0] WBus;
wire none;
wire [2:0] MAR_OUT;
output Lmnot;
output [3:0] MUX_OUT;
output [5:0] HighZ;
output [7:0] MUX_OUT_HEX, BLANK_DISPLAY;
tri [5:0] HighZ;

MAR MAR_MUX_1(MAR_OUT, WBus, Lm, CLK);

MUX MAR_MUX_2
    (MUX_OUT, MUX_OUT_HEX, BLANK_DISPLAY, MAR_OUT, ADDR_IN,
     RUN_PROG);

assign Lmnot = ~Lm,
        none = 0;

bufif1(HighZ[0],none,none);
bufif1(HighZ[1],none,none);
bufif1(HighZ[2],none,none);
bufif1(HighZ[3],none,none);
bufif1(HighZ[4],none,none);
```



```

        bufifl(HighZ[5],none,none);

endmodule

```

E-2.2 MAR

```

module MAR(MAR_OUT,WBus,Lm,CLK);

// Lm -> high to load PC count from WBus

    inout [7:0] WBus;
    input Lm,CLK;
    output [2:0] MAR_OUT;
    wire none;

    D_FF MAR0(MAR_OUT[0],WBus[0],Lm,CLK);
    D_FF MAR1(MAR_OUT[1],WBus[1],Lm,CLK);
    D_FF MAR2(MAR_OUT[2],WBus[2],Lm,CLK);

    assign none = 0;

    bufifl(WBus[0],none,none);
    bufifl(WBus[1],none,none);
    bufifl(WBus[2],none,none);
    bufifl(WBus[3],none,none);
    bufifl(WBus[4],none,none);
    bufifl(WBus[5],none,none);
    bufifl(WBus[6],none,none);
    bufifl(WBus[7],none,none);

endmodule

```

E-2.3 2 to 1 Multiplexer

```

module MUX(MUX_OUT, MUX_OUT_HEX, BLANK_DISPLAY, MAR_OUT, ADDR_IN, RUN_PROG);

// RUN_PROG == 0 -> program
//                -> address from switches taken
// RUN_PROG == 1 -> run
//                -> address from MAR output taken

    input [2:0] MAR_OUT, ADDR_IN;
    input RUN_PROG;
    output [3:0] MUX_OUT;
    output [7:0] MUX_OUT_HEX, BLANK_DISPLAY;

    assign MUX_OUT[2:0] = RUN_PROG? MAR_OUT : ADDR_IN,
           MUX_OUT[3] = 1'b0;

    HEX_DISPLAY MUX_HEX(MUX_OUT_HEX, MUX_OUT);

    assign BLANK_DISPLAY = 8'b11111111;

endmodule

```

E-2.4 D Flip-flop with Enable Input for Data Loading

```

module D_FF(Q,D,EN,CLK);

    input D,EN,CLK;
    output Q;
    reg Q;

    always @(posedge CLK)
        if (EN == 1)
            Q = D;
        else

```

```

        Q = Q;

endmodule

```

E-2.5 Hexadecimal Display

```

module HEX_DISPLAY(HEX_OUT,HEX_IN);

// HEX_OUT -> signals a-h of active low hex 7-segment display
//          MSB = a, LSB = h (decimal point)

    output [7:0] HEX_OUT;
    input [3:0] HEX_IN;
    reg [7:0] HEX_OUT;

    always @ (HEX_IN)
        case(HEX_IN)
            4'b0001 : HEX_OUT = 8'b10011111;
            4'b0010 : HEX_OUT = 8'b00100101;
            4'b0011 : HEX_OUT = 8'b00001101;
            4'b0100 : HEX_OUT = 8'b10011001;
            4'b0101 : HEX_OUT = 8'b01001001;
            4'b0110 : HEX_OUT = 8'b01000001;
            4'b0111 : HEX_OUT = 8'b00011111;
            4'b1000 : HEX_OUT = 8'b00000001;
            4'b1001 : HEX_OUT = 8'b00001001;
            4'b1010 : HEX_OUT = 8'b00010001;
            4'b1011 : HEX_OUT = 8'b11000001;
            4'b1100 : HEX_OUT = 8'b01100011;
            4'b1101 : HEX_OUT = 8'b10000101;
            4'b1110 : HEX_OUT = 8'b01100001;
            4'b1111 : HEX_OUT = 8'b01110001;
            default : HEX_OUT = 8'b00000011;
        endcase

endmodule

```

E-3 8 × 8 RAM

E-3.1 8 × 8 RAM

```

module _8x8_RAM_MAX(WBus,DATA_OUT,RAM_DISPLAY_HEX,HighZ,
                    MUX_OUT,DATA_IN,CE,CENot,RUN_PROG,READ_WRITE);

// RUN_PROG == 0 --> PROG;      RUN_PROG == 1 --> RUN
// READ_WRITE == 0 --> WRITE;   READ_WRITE == 1 --> READ

    input [2:0] MUX_OUT;
    input [7:0] DATA_IN;
    input CE, RUN_PROG, READ_WRITE;
    output CENot;
    output [6:0] HighZ;
    output [7:0] WBus, DATA_OUT;
    output [15:0] RAM_DISPLAY_HEX;
    wire none;
    wire [7:0] RAM_DISPLAY;
    tri [6:0] HighZ;

    tri [7:0] WBus;
    reg [7:0] DATA_OUT;
    reg [7:0] MEM [15:0];

    always @ (MUX_OUT or DATA_IN or RUN_PROG or READ_WRITE)
        begin

```

```

        if (!RUN_PROG)
            begin
                if (!READ_WRITE)
                    MEM [MUX_OUT] = DATA_IN;
            end

        else if (RUN_PROG)
            DATA_OUT = MEM [MUX_OUT];

        else;

    end

    bufifl1(WBus[0],DATA_OUT[0],CE);
    bufifl1(WBus[1],DATA_OUT[1],CE);
    bufifl1(WBus[2],DATA_OUT[2],CE);
    bufifl1(WBus[3],DATA_OUT[3],CE);
    bufifl1(WBus[4],DATA_OUT[4],CE);
    bufifl1(WBus[5],DATA_OUT[5],CE);
    bufifl1(WBus[6],DATA_OUT[6],CE);
    bufifl1(WBus[7],DATA_OUT[7],CE);

    assign RAM_DISPLAY = MEM [MUX_OUT],
           CEnot = ~CE,
           none = 0;

    HEX_DISPLAY RAM_HEX_1
        (RAM_DISPLAY_HEX[15:8],RAM_DISPLAY[7:4]);
    HEX_DISPLAY RAM_HEX_2
        (RAM_DISPLAY_HEX[7:0],RAM_DISPLAY[3:0]);

    bufifl1(HighZ[6],none,none);
    bufifl1(HighZ[5],none,none);
    bufifl1(HighZ[4],none,none);
    bufifl1(HighZ[3],none,none);
    bufifl1(HighZ[2],none,none);
    bufifl1(HighZ[1],none,none);
    bufifl1(HighZ[0],none,none);

endmodule

```

E-3.2 Hexadecimal Display

```

module HEX_DISPLAY(HEX_OUT,HEX_IN);

// HEX_OUT -> signals a-h of active low hex 7-segment display
//           MSB = a, LSB = h (decimal point)

    output [7:0] HEX_OUT;
    input [3:0] HEX_IN;
    reg [7:0] HEX_OUT;

    always @(HEX_IN)
        case(HEX_IN)
            4'b0001 : HEX_OUT = 8'b10011111;
            4'b0010 : HEX_OUT = 8'b00100101;
            4'b0011 : HEX_OUT = 8'b00001101;
            4'b0100 : HEX_OUT = 8'b10011001;
            4'b0101 : HEX_OUT = 8'b01001001;
            4'b0110 : HEX_OUT = 8'b01000001;
            4'b0111 : HEX_OUT = 8'b00011111;
            4'b1000 : HEX_OUT = 8'b00000001;
            4'b1001 : HEX_OUT = 8'b00001001;
            4'b1010 : HEX_OUT = 8'b00010001;
            4'b1011 : HEX_OUT = 8'b11000001;
            4'b1100 : HEX_OUT = 8'b01100011;
            4'b1101 : HEX_OUT = 8'b10000101;
            4'b1110 : HEX_OUT = 8'b01100001;
            4'b1111 : HEX_OUT = 8'b01110001;
            default : HEX_OUT = 8'b00000011;
        endcase

endmodule

```

E-4 Instruction Register

E-4.1 Instruction Register with Tri-state Address Output

```
module INSTRUCTION_REGISTER(IR_OUT_ADDR,IR_OUT_INS,HighZ,
                           IR_OUT_ADDR_HEX,IR_OUT_INS_HEX,
                           WBus,Li,Ei,CLK,CLR,Linot,Einot);

// Li -> high to load instruction from WBus
// Ei -> high to put lower nibble of instruction
//      (data address field) at WBus
//
// IR_OUT_ADDR -> data address field of instruction
// IR_OUT_INS -> instruction opcode field (upper nibble)

input Li,Ei,CLK,CLR;
inout [7:0] WBus;
output Linot,Einot;
output [3:0] IR_OUT_ADDR,IR_OUT_INS;
output [5:0] HighZ;
output [7:0] IR_OUT_ADDR_HEX,IR_OUT_INS_HEX;
wire none;
tri [5:0] HighZ;
tri [7:0] WBus;

D_FF IR0(IR_OUT_ADDR[0],WBus[0],Li,CLK);
D_FF IR1(IR_OUT_ADDR[1],WBus[1],Li,CLK);
D_FF IR2(IR_OUT_ADDR[2],WBus[2],Li,CLK);
D_FF IR3(IR_OUT_ADDR[3],WBus[3],Li,CLK);
DFF_POSCLR IR4(IR_OUT_INS[0],WBus[4],Li,CLK,CLR);
DFF_POSCLR IR5(IR_OUT_INS[1],WBus[5],Li,CLK,CLR);
DFF_POSCLR IR6(IR_OUT_INS[2],WBus[6],Li,CLK,CLR);
DFF_POSCLR IR7(IR_OUT_INS[3],WBus[7],Li,CLK,CLR);

assign none = 0;

bufif1(WBus[0],IR_OUT_ADDR[0],Ei);
bufif1(WBus[1],IR_OUT_ADDR[1],Ei);
bufif1(WBus[2],IR_OUT_ADDR[2],Ei);
bufif1(WBus[3],IR_OUT_ADDR[3],Ei);
bufif1(WBus[4],none,none);
bufif1(WBus[5],none,none);
bufif1(WBus[6],none,none);
bufif1(WBus[7],none,none);

HEX_DISPLAY IR_ADDR_HEX(IR_OUT_ADDR_HEX,IR_OUT_ADDR);
HEX_DISPLAY IR_INS_HEX(IR_OUT_INS_HEX,IR_OUT_INS);

assign Linot = ~Li,
       Einot = ~Ei;

bufif1(HighZ[5],none,none);
bufif1(HighZ[4],none,none);
bufif1(HighZ[3],none,none);
bufif1(HighZ[2],none,none);
bufif1(HighZ[1],none,none);
bufif1(HighZ[0],none,none);

endmodule
```

E-4.2 D Flip-flop with Enable Input for Data Loading

```
module D_FF(Q,D,EN,CLK);

input D,EN,CLK;
output Q;
reg Q;

always @(posedge CLK)
    if (EN == 1)
```

```

        Q = D;
    else
        Q = Q;
    endmodule

```

E-4.3 D Flip-flop with Data Loading Enable & Clear Input

```

module DFF_POSCLR(Q,D,EN,CLK,CLR);

    input D,EN,CLK,CLR;
    output Q;
    reg Q;

    always @(posedge CLK or posedge CLR)
        if (CLR == 1)
            Q = 0;
        else if (EN == 1)
            Q = D;
        else
            Q = Q;

endmodule

```

E-4.4 Hexadecimal Display

```

module HEX_DISPLAY(HEX_OUT,HEX_IN);

    // HEX_OUT -> signals a-h of active low hex 7-segment display
    //           MSB = a, LSB = h (decimal point)

    output [7:0] HEX_OUT;
    input [3:0] HEX_IN;
    reg [7:0] HEX_OUT;

    always @(HEX_IN)
        case(HEX_IN)
            4'b0001 : HEX_OUT = 8'b10011111;
            4'b0010 : HEX_OUT = 8'b00100101;
            4'b0011 : HEX_OUT = 8'b00001101;
            4'b0100 : HEX_OUT = 8'b10011001;
            4'b0101 : HEX_OUT = 8'b01001001;
            4'b0110 : HEX_OUT = 8'b01000001;
            4'b0111 : HEX_OUT = 8'b00011111;
            4'b1000 : HEX_OUT = 8'b00000001;
            4'b1001 : HEX_OUT = 8'b00001001;
            4'b1010 : HEX_OUT = 8'b00010001;
            4'b1011 : HEX_OUT = 8'b11000001;
            4'b1100 : HEX_OUT = 8'b01100011;
            4'b1101 : HEX_OUT = 8'b10000101;
            4'b1110 : HEX_OUT = 8'b01100001;
            4'b1111 : HEX_OUT = 8'b01110001;
            default : HEX_OUT = 8'b00000011;
        endcase

endmodule

```

E-5 Accumulator

E-5.1 Accumulator

```
module ACCUMULATOR(ACCU_OUT,ACCU_OUT_HEX,WBus,La,Ea,CLK,Lanot,Eanot,HighZ);

// La -> high to load data from WBus
// Ea -> high to put accumulator content ot WBus

    input La,Ea,CLK;
    inout [7:0] WBus;
        output Lanot,Eanot;
        output [1:0] HighZ;
    output [7:0] ACCU_OUT;
    output [15:0] ACCU_OUT_HEX;
    wire none;
        tri [1:0] HighZ;
        tri [7:0] WBus;

    D_FF ACCU0(ACCU_OUT[0],WBus[0],La,CLK);
    D_FF ACCU1(ACCU_OUT[1],WBus[1],La,CLK);
    D_FF ACCU2(ACCU_OUT[2],WBus[2],La,CLK);
    D_FF ACCU3(ACCU_OUT[3],WBus[3],La,CLK);
    D_FF ACCU4(ACCU_OUT[4],WBus[4],La,CLK);
    D_FF ACCU5(ACCU_OUT[5],WBus[5],La,CLK);
    D_FF ACCU6(ACCU_OUT[6],WBus[6],La,CLK);
    D_FF ACCU7(ACCU_OUT[7],WBus[7],La,CLK);

    bufif1(WBus[0],ACCU_OUT[0],Ea);
    bufif1(WBus[1],ACCU_OUT[1],Ea);
    bufif1(WBus[2],ACCU_OUT[2],Ea);
    bufif1(WBus[3],ACCU_OUT[3],Ea);
    bufif1(WBus[4],ACCU_OUT[4],Ea);
    bufif1(WBus[5],ACCU_OUT[5],Ea);
    bufif1(WBus[6],ACCU_OUT[6],Ea);
    bufif1(WBus[7],ACCU_OUT[7],Ea);

    HEX_DISPLAY ACCU_HEX 1
        (ACCU_OUT_HEX[15:8],ACCU_OUT[7:4]);
    HEX_DISPLAY ACCU_HEX 2
        (ACCU_OUT_HEX[7:0],ACCU_OUT[3:0]);

    assign Lanot = ~La,
           Eanot = ~Ea,
           none = 0;

    bufif1(HighZ[1],none,none);
    bufif1(HighZ[0],none,none);

endmodule
```

E-5.2 D Flip-flop with Enable Input for Data Loading

```
module D_FF(Q,D,EN,CLK);

    input D,EN,CLK;
    output Q;
    reg Q;

    always @(posedge CLK)
        if (EN == 1)
            Q = D;
        else
            Q = Q;

endmodule
```

E-5.3 Hexadecimal Display

```
module HEX_DISPLAY(HEX_OUT,HEX_IN);

// HEX_OUT -> signals a-h of active low hex 7-segment display
//           MSB = a, LSB = h (decimal point)

output [7:0] HEX_OUT;
input [3:0] HEX_IN;
reg [7:0] HEX_OUT;

always @ (HEX_IN)
    case(HEX_IN)
        4'b0001 : HEX_OUT = 8'b10011111;
        4'b0010 : HEX_OUT = 8'b00100101;
        4'b0011 : HEX_OUT = 8'b00001101;
        4'b0100 : HEX_OUT = 8'b10011001;
        4'b0101 : HEX_OUT = 8'b01001001;
        4'b0110 : HEX_OUT = 8'b01000001;
        4'b0111 : HEX_OUT = 8'b00011111;
        4'b1000 : HEX_OUT = 8'b00000001;
        4'b1001 : HEX_OUT = 8'b00001001;
        4'b1010 : HEX_OUT = 8'b00010001;
        4'b1011 : HEX_OUT = 8'b11000001;
        4'b1100 : HEX_OUT = 8'b01100011;
        4'b1101 : HEX_OUT = 8'b10000101;
        4'b1110 : HEX_OUT = 8'b01100001;
        4'b1111 : HEX_OUT = 8'b01110001;
        default : HEX_OUT = 8'b00000011;
    endcase

endmodule
```

E-6 Adder/Subtractor

E-6.1 Adder/Subtractor

```
module ADD_SUB(ADD_SUB_OUT,ADD_SUB_OUT_HEX,WBus,HighZ,
               ACCU_OUT,B_REG_OUT,Su,Eu,Sunot,Eunot);

// Su -> subtraction operation enable bit, high to convert
//       B Register output to 2's complement form
//
// Eu -> high to put arithmetic operation result on WBus
//
// ADD_SUB_OUT -> Adder/Subtractor's output, arithmetic
//               operation result
//
// ACCU_OUT -> Accumulator's output, operand of arithmetic
//             operation
//
// B_REG_OUT -> B Register's output
//
// B -> temporary register to hold operand of arithmetic
//      operation
// == B_REG_OUT for addition operation
// == ~B_REG_OUT for subtraction operation
//      (added with 1 (Su) for 2's complement of B_REG_OUT)

output [3:0] HighZ;
output [7:0] ADD_SUB_OUT, WBus;
output [15:0] ADD_SUB_OUT_HEX;
output Sunot, Eunot;
input [7:0] ACCU_OUT, B_REG_OUT;
input Su, Eu;
reg [7:0] ADD_SUB_OUT,B;
```

```

wire none;
tri [3:0] HighZ;
tri [7:0] WBus;

always @ (ACCU_OUT or B_REG_OUT or Su)
begin
    if (Su)
        begin B = ~(B_REG_OUT); end
    else
        begin B = B_REG_OUT; end
    ADD_SUB_OUT = ACCU_OUT + B + Su;
end

bufif1(WBus[0], ADD_SUB_OUT[0], Eu);
bufif1(WBus[1], ADD_SUB_OUT[1], Eu);
bufif1(WBus[2], ADD_SUB_OUT[2], Eu);
bufif1(WBus[3], ADD_SUB_OUT[3], Eu);
bufif1(WBus[4], ADD_SUB_OUT[4], Eu);
bufif1(WBus[5], ADD_SUB_OUT[5], Eu);
bufif1(WBus[6], ADD_SUB_OUT[6], Eu);
bufif1(WBus[7], ADD_SUB_OUT[7], Eu);

HEX_DISPLAY_ADD_SUB_HEX_1
(ADD_SUB_OUT_HEX[15:8], ADD_SUB_OUT[7:4]);
HEX_DISPLAY_ADD_SUB_HEX_2
(ADD_SUB_OUT_HEX[7:0], ADD_SUB_OUT[3:0]);

assign Sunot = ~Su,
    Eunot = ~Eu,
    none = 0;

bufif1(HighZ[3], none, none);
bufif1(HighZ[2], none, none);
bufif1(HighZ[1], none, none);
bufif1(HighZ[0], none, none);

endmodule

```

E-6.2 Hexadecimal Display

```

module HEX_DISPLAY(HEX_OUT, HEX_IN);

// HEX_OUT -> signals a-h of active low hex 7-segment display
//          MSB = a, LSB = h (decimal point)

output [7:0] HEX_OUT;
input [3:0] HEX_IN;
reg [7:0] HEX_OUT;

always @ (HEX_IN)
begin
    case(HEX_IN)
        4'b0001 : HEX_OUT = 8'b10011111;
        4'b0010 : HEX_OUT = 8'b00100101;
        4'b0011 : HEX_OUT = 8'b00001101;
        4'b0100 : HEX_OUT = 8'b10011001;
        4'b0101 : HEX_OUT = 8'b01001001;
        4'b0110 : HEX_OUT = 8'b01000001;
        4'b0111 : HEX_OUT = 8'b00011111;
        4'b1000 : HEX_OUT = 8'b00000001;
        4'b1001 : HEX_OUT = 8'b00001001;
        4'b1010 : HEX_OUT = 8'b00010001;
        4'b1011 : HEX_OUT = 8'b11000001;
        4'b1100 : HEX_OUT = 8'b01100011;
        4'b1101 : HEX_OUT = 8'b10000101;
        4'b1110 : HEX_OUT = 8'b01100001;
        4'b1111 : HEX_OUT = 8'b01110001;
        default : HEX_OUT = 8'b00000011;
    endcase
end

endmodule

```


E-7 B Register

E-7.1 B Register

```
module B_REGISTER(B_REG_OUT,B_REG_OUT_HEX,WBus,Lb,CLK,Lbnot,HighZ);

// Lb -> high to load data from WBus

    output [1:0] HighZ;
    output [7:0] B_REG_OUT;
    output [15:0] B_REG_OUT_HEX;
    output Lbnot;
    input [7:0] WBus;
    input Lb, CLK;
    wire none;
    tri [1:0] HighZ;

    D_FF B_REG_0(B_REG_OUT[0],WBus[0],Lb,CLK);
    D_FF B_REG_1(B_REG_OUT[1],WBus[1],Lb,CLK);
    D_FF B_REG_2(B_REG_OUT[2],WBus[2],Lb,CLK);
    D_FF B_REG_3(B_REG_OUT[3],WBus[3],Lb,CLK);
    D_FF B_REG_4(B_REG_OUT[4],WBus[4],Lb,CLK);
    D_FF B_REG_5(B_REG_OUT[5],WBus[5],Lb,CLK);
    D_FF B_REG_6(B_REG_OUT[6],WBus[6],Lb,CLK);
    D_FF B_REG_7(B_REG_OUT[7],WBus[7],Lb,CLK);

    HEX_DISPLAY B_REG_HEX_1
        (B_REG_OUT_HEX[15:8],B_REG_OUT[7:4]);
    HEX_DISPLAY B_REG_HEX_2
        (B_REG_OUT_HEX[7:0],B_REG_OUT[3:0]);

    assign Lbnot = ~Lb,
        none = 0;

    bufif1(HighZ[1],none,none);
    bufif1(HighZ[0],none,none);

endmodule
```

E-7.2 D Flip-flop with Enable Input for Data Loading

```
module D_FF(Q,D,EN,CLK);

    input D,EN,CLK;
    output Q;
    reg Q;

    always @(posedge CLK)
        if (EN == 1)
            Q = D;
        else
            Q = Q;

endmodule
```

E-7.3 Hexadecimal Display

```
module HEX_DISPLAY(HEX_OUT,HEX_IN);

// HEX_OUT -> signals a-h of active low hex 7-segment display
//          MSB = a, LSB = h (decimal point)

    output [7:0] HEX_OUT;
    input [3:0] HEX_IN;
    reg [7:0] HEX_OUT;
```

```

always @(HEX_IN)
  case(HEX_IN)
    4'b0001 : HEX_OUT = 8'b10011111;
    4'b0010 : HEX_OUT = 8'b00100101;
    4'b0011 : HEX_OUT = 8'b00001101;
    4'b0100 : HEX_OUT = 8'b10011001;
    4'b0101 : HEX_OUT = 8'b01001001;
    4'b0110 : HEX_OUT = 8'b01000001;
    4'b0111 : HEX_OUT = 8'b00011111;
    4'b1000 : HEX_OUT = 8'b00000001;
    4'b1001 : HEX_OUT = 8'b00001001;
    4'b1010 : HEX_OUT = 8'b00010001;
    4'b1011 : HEX_OUT = 8'b11000001;
    4'b1100 : HEX_OUT = 8'b01100011;
    4'b1101 : HEX_OUT = 8'b10000101;
    4'b1110 : HEX_OUT = 8'b01100001;
    4'b1111 : HEX_OUT = 8'b01110001;
    default : HEX_OUT = 8'b00000011;
  endcase
endmodule

```

E-8 Output Register

E-8.1 Output Register

```

module OUTPUT_REGISTER(OUT_REG,OUT_REG_HEX,WBus,Lo,CLK,Lonot);

// Lo -> high to load data from WBus

output [7:0] OUT_REG;
output [15:0] OUT_REG_HEX;
output Lonot;
input [7:0] WBus;
input Lo, CLK;

D_FF OUT_REG_0(OUT_REG[0],WBus[0],Lo,CLK);
D_FF OUT_REG_1(OUT_REG[1],WBus[1],Lo,CLK);
D_FF OUT_REG_2(OUT_REG[2],WBus[2],Lo,CLK);
D_FF OUT_REG_3(OUT_REG[3],WBus[3],Lo,CLK);
D_FF OUT_REG_4(OUT_REG[4],WBus[4],Lo,CLK);
D_FF OUT_REG_5(OUT_REG[5],WBus[5],Lo,CLK);
D_FF OUT_REG_6(OUT_REG[6],WBus[6],Lo,CLK);
D_FF OUT_REG_7(OUT_REG[7],WBus[7],Lo,CLK);

HEX_DISPLAY OUT_HEX_1
  (OUT_REG_HEX[15:8],OUT_REG[7:4]);
HEX_DISPLAY OUT_HEX_2
  (OUT_REG_HEX[7:0],OUT_REG[3:0]);

  assign Lonot = ~Lo;

endmodule

```

E-8.2 D Flip-flop with Enable Input for Data Loading

```

module D_FF(Q,D,EN,CLK);

input D,EN,CLK;
output Q;
reg Q;

always @(posedge CLK)
  if (EN == 1)

```

```

        Q = D;
    else
        Q = Q;
    endmodule

```

E-8.3 Hexadecimal Display

```

module HEX_DISPLAY(HEX_OUT,HEX_IN);

// HEX_OUT -> signals a-g of hex 7-segment display
//          MSB = a, LSB = g

    output [6:0] HEX_OUT;
    input [3:0] HEX_IN;
    reg [6:0] HEX_OUT;

    always @(HEX_IN)
        case(HEX_IN)
            4'b0001 : HEX_OUT = 7'b0110000;
            4'b0010 : HEX_OUT = 7'b1101101;
            4'b0011 : HEX_OUT = 7'b1111001;
            4'b0100 : HEX_OUT = 7'b0110011;
            4'b0101 : HEX_OUT = 7'b1011011;
            4'b0110 : HEX_OUT = 7'b1011111;
            4'b0111 : HEX_OUT = 7'b1110000;
            4'b1000 : HEX_OUT = 7'b1111111;
            4'b1001 : HEX_OUT = 7'b1111011;
            4'b1010 : HEX_OUT = 7'b1110111;
            4'b1011 : HEX_OUT = 7'b0011111;
            4'b1100 : HEX_OUT = 7'b1001110;
            4'b1101 : HEX_OUT = 7'b0111101;
            4'b1110 : HEX_OUT = 7'b1001111;
            4'b1111 : HEX_OUT = 7'b1000111;
            default : HEX_OUT = 7'b1111110;
        endcase

endmodule

```

E-9 Controller/Sequencer (Instruction Decoder, Ring Counter & Control Matrix)

E-9.1 Controller/Sequencer

```

module CONTROLLER_SEQUENCER
    (Cp,Ep,Lm,CE,Li,Ei,La,Ea,Su,Eu,Lb,Lo,LDAnot,ADDnot,SUBnot,OUTnot,HLTnot,
     HLT,HighZ,BLANK_DISPLAY,IR_OUT_INS,Tnot,CLK,CLR);

    output Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo,
           LDAnot, ADDnot, SUBnot, OUTnot, HLTnot, HLT;
    output [6:1] Tnot;
    output [5:0] HighZ;
    output [15:0] BLANK_DISPLAY;
    input CLK, CLR;
    input [3:0] IR_OUT_INS;
    wire LDA, ADD, SUB, OUT, none;
    wire [6:1] T;
    tri [5:0] HighZ;

    RING_COUNTER RC(T,Tnot,CLK,CLR);
    INSTRUCTION_DECODER ID(LDA,ADD,SUB,OUT,HLT,IR_OUT_INS);
    CONTROL_MATRIX CM(Cp,Ep,Lm,CE,Li,Ei,La,Ea,Su,Eu,Lb,Lo,
                     LDA,ADD,SUB,OUT,HLT,T,CLK,CLR);
endmodule

```

```

assign BLANK_DISPLAY = 16'hffff,
       LDAnot = ~LDA,
       ADDnot = ~ADD,
       SUBnot = ~SUB,
       OUTnot = ~OUT,
       HLTnot = ~HLT,
       none = 0;

bufif1(HighZ[5],none,none);
bufif1(HighZ[4],none,none);
bufif1(HighZ[3],none,none);
bufif1(HighZ[2],none,none);
bufif1(HighZ[1],none,none);
bufif1(HighZ[0],none,none);

endmodule

```

E-9.2 Instruction Decoder

```

module INSTRUCTION_DECODER(LDA, ADD, SUB, OUT, HLT, IR_OUT_INS);

// associate control line of each routine with their
// corresponding opcode

output LDA, ADD, SUB, OUT, HLT;
input [3:0] IR_OUT_INS;

assign  LDA = (IR_OUT_INS == 4'b0000)? 1'b1 : 1'b0,
        ADD = (IR_OUT_INS == 4'b0001)? 1'b1 : 1'b0,
        SUB = (IR_OUT_INS == 4'b0010)? 1'b1 : 1'b0,
        OUT = (IR_OUT_INS == 4'b1110)? 1'b1 : 1'b0,
        HLT = (IR_OUT_INS == 4'b1111)? 1'b1 : 1'b0;

endmodule

```

E-9.3 Ring Counter

```

module RING_COUNTER(T, Tnot, CLK, CLR);

// CLR high -> Ring Counter resets to 000001
//
// Ring Counter shifts left at each negative
// edge of CLK

output [6:1] T, Tnot;
input CLK, CLR;

JKFF_Qnot_NEGCLK_POSCLR RC1
(Tnot[1],T[1],Tnot[6],T[6],CLK,CLR);
JKFF_Qnot_NEGCLK_POSCLR RC2
(T[2],Tnot[2],T[1],Tnot[1],CLK,CLR);
JKFF_Qnot_NEGCLK_POSCLR RC3
(T[3],Tnot[3],T[2],Tnot[2],CLK,CLR);
JKFF_Qnot_NEGCLK_POSCLR RC4
(T[4],Tnot[4],T[3],Tnot[3],CLK,CLR);
JKFF_Qnot_NEGCLK_POSCLR RC5
(T[5],Tnot[5],T[4],Tnot[4],CLK,CLR);
JKFF_Qnot_NEGCLK_POSCLR RC6
(T[6],Tnot[6],T[5],Tnot[5],CLK,CLR);

endmodule

```

E-9.4 Control Matrix

```

module CONTROL_MATRIX(Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo,
                      LDA, ADD, SUB, OUT, HLT, T, CLK, CLR);

output Cp, Ep, Lm, CE, Li, Ei, La, Ea, Su, Eu, Lb, Lo;

```

```

input LDA, ADD, SUB, OUT, HLT, CLK, CLR;
input [6:1] T;

assign  Cp = (T[2])? 1'b1 : 1'b0,
        Ep = (T[1])? 1'b1 : 1'b0,
        Lm = (T[1] || (LDA && T[4]) || (ADD && T[4]) || (SUB && T[4]))?
            1'b1 : 1'b0,
        CE = (T[3] || (LDA && T[5]) || (ADD && T[5]) || (SUB && T[5]))?
            1'b1 : 1'b0,
        Li = (T[3])? 1'b1 : 1'b0,
        Ei = ((LDA && T[4]) || (ADD && T[4]) || (SUB && T[4]))?
            1'b1 : 1'b0,
        La = ((LDA && T[5]) || (ADD && T[6]) || (SUB && T[6]))?
            1'b1 : 1'b0,
        Ea = (OUT && T[4])? 1'b1 : 1'b0,
        Su = (SUB && T[6])? 1'b1 : 1'b0,
        Eu = ((ADD && T[6]) || (SUB && T[6]))? 1'b1 : 1'b0,
        Lb = ((ADD && T[5]) || (SUB && T[5]))? 1'b1 : 1'b0,
        Lo = (OUT && T[4])? 1'b1 : 1'b0;

endmodule

```

E-9.5 Positive-edge-triggered JK Flip-flop with Active High Clear

```

module JKFF(Q,Qnot,J,K,CLK,CLR);

input J,K,CLK,CLR;
output Q,Qnot;
reg Q,Qnot;

always @(negedge CLK or posedge CLR)
begin
    if (CLR)
        begin
            Q = 1'b0;
            Qnot = ~Q;
        end
    else if ({J,K} == 2'b00)
        begin
            Q = Q;
            Qnot = ~Q;
        end
    else if ({J,K} == 2'b01)
        begin
            Q = 1'b0;
            Qnot = ~Q;
        end
    else if ({J,K} == 2'b10)
        begin
            Q = 1'b1;
            Qnot = ~Q;
        end
    else if ({J,K} == 2'b11)
        begin
            Q = ~Q;
            Qnot = ~Q;
        end
    end
end

endmodule

```

E-10 Mode-Select Switches, De-bouncers & Clock Buffer

E-10.1 De-bouncers

```
// START_CLEAR DIP switch -> START active low
//                               CLEAR active high
// MANUAL_AUTO DIP switch -> MANUAL active low
//                               AUTO active high
// **DIP switch gives logic low when pressed**
//
// LOW & HIGH SPDT switch -> LOW active low
//                               HIGH active low
//
// DB_TEMP1 -> clock signal in MANUAL mode
// DB_TEMP2 -> clock signal in AUTO mode

module DEBOUNCERS(CLK,CLR,CLKnot,CLRnot,BLANK_DISPLAY,
                 START_CLEAR,LOW,HIGH,MANUAL_AUTO,HLT,rawrawCLK);

    output CLK, CLKnot, CLRnot;
    output [15:0] BLANK_DISPLAY;
    input START_CLEAR, LOW, HIGH, MANUAL_AUTO, HLT,
          rawrawCLK;
    inout CLR;
    wire [25:0] rawCLK;
    wire CLRnot, HI_LOnot, HI_LO, MANUAL_OUT,
          AUTO_OUT, DB_TEMP1, DB_TEMP2;

    CLEAR_START    DB1(CLR,START_CLEAR);
    SINGLE_STEP    DB2(HI_LO,LOW,HIGH);
    AUTO_MANUAL    DB3(MANUAL_OUT,AUTO_OUT,MANUAL_AUTO);
    CLOCK          DB4(rawCLK,HLT,rawrawCLK,CLR);

    and    DB5(DB_TEMP1,~HLT,HI_LO,MANUAL_OUT),
          DB6(DB_TEMP2,AUTO_OUT,rawCLK[25]);

    or     DB7(CLK,DB_TEMP1,DB_TEMP2);

    assign BLANK_DISPLAY = 16'hffff,
           CLKnot = ~CLK,
           CLRnot = ~CLR;

endmodule
```

E-10.2 Clear-Start De-bouncer

```
module CLEAR_START(CLR, START_CLEAR);

    output CLR;
    input START_CLEAR;
    wire CLRnot, START_CLEAR_NOT;

    not g1(START_CLEAR_NOT, START_CLEAR);

    SR_LATCH CS(CLRnot, CLR, START_CLEAR, START_CLEAR_NOT);

endmodule
```

E-10.3 Single-Step De-bouncer

```
module SINGLE_STEP(HI_LO, LOW, HIGH);

    output HI_LO;
    input LOW, HIGH;
    wire HI_LOnot;

    SR_LATCH SS(HI_LOnot, HI_LO, LOW, HIGH);

endmodule
```

```
endmodule
```

E-10.4 Manual-Auto De-bouncer

```
module AUTO_MANUAL(MANUAL_OUT, AUTO_OUT, MANUAL_AUTO);

    output MANUAL_OUT, AUTO_OUT;
    input MANUAL_AUTO;
    wire MANUAL_AUTO_NOT;

    not g1(MANUAL_AUTO_NOT, MANUAL_AUTO);

    SR_LATCH
    AM(MANUAL_OUT, AUTO_OUT, MANUAL_AUTO, MANUAL_AUTO_NOT);

endmodule
```

D-10.5 Clock Buffer

```
module CLOCK(rawCLK,HLT,rawrawCLK,CLR);

    output[25:0] rawCLK;
    input HLT, rawrawCLK, CLR;

    JKFF_Q_POSCLK_POSCLR Clock00 (rawCLK[0],~HLT,~HLT,rawrawCLK,CLR);
    JKFF_Q_POSCLK_POSCLR Clock01 (rawCLK[1],~HLT,~HLT,rawCLK[0],CLR);
    JKFF_Q_POSCLK_POSCLR Clock02 (rawCLK[2],~HLT,~HLT,rawCLK[1],CLR);
    JKFF_Q_POSCLK_POSCLR Clock03 (rawCLK[3],~HLT,~HLT,rawCLK[2],CLR);
    JKFF_Q_POSCLK_POSCLR Clock04 (rawCLK[4],~HLT,~HLT,rawCLK[3],CLR);
    JKFF_Q_POSCLK_POSCLR Clock05 (rawCLK[5],~HLT,~HLT,rawCLK[4],CLR);
    JKFF_Q_POSCLK_POSCLR Clock06 (rawCLK[6],~HLT,~HLT,rawCLK[5],CLR);
    JKFF_Q_POSCLK_POSCLR Clock07 (rawCLK[7],~HLT,~HLT,rawCLK[6],CLR);
    JKFF_Q_POSCLK_POSCLR Clock08 (rawCLK[8],~HLT,~HLT,rawCLK[7],CLR);
    JKFF_Q_POSCLK_POSCLR Clock09 (rawCLK[9],~HLT,~HLT,rawCLK[8],CLR);
    JKFF_Q_POSCLK_POSCLR Clock10 (rawCLK[10],~HLT,~HLT,rawCLK[9],CLR);
    JKFF_Q_POSCLK_POSCLR Clock11 (rawCLK[11],~HLT,~HLT,rawCLK[10],CLR);
    JKFF_Q_POSCLK_POSCLR Clock12 (rawCLK[12],~HLT,~HLT,rawCLK[11],CLR);
    JKFF_Q_POSCLK_POSCLR Clock13 (rawCLK[13],~HLT,~HLT,rawCLK[12],CLR);
    JKFF_Q_POSCLK_POSCLR Clock14 (rawCLK[14],~HLT,~HLT,rawCLK[13],CLR);
    JKFF_Q_POSCLK_POSCLR Clock15 (rawCLK[15],~HLT,~HLT,rawCLK[14],CLR);
    JKFF_Q_POSCLK_POSCLR Clock16 (rawCLK[16],~HLT,~HLT,rawCLK[15],CLR);
    JKFF_Q_POSCLK_POSCLR Clock17 (rawCLK[17],~HLT,~HLT,rawCLK[16],CLR);
    JKFF_Q_POSCLK_POSCLR Clock18 (rawCLK[18],~HLT,~HLT,rawCLK[17],CLR);
    JKFF_Q_POSCLK_POSCLR Clock19 (rawCLK[19],~HLT,~HLT,rawCLK[18],CLR);
    JKFF_Q_POSCLK_POSCLR Clock20 (rawCLK[20],~HLT,~HLT,rawCLK[19],CLR);
    JKFF_Q_POSCLK_POSCLR Clock21 (rawCLK[21],~HLT,~HLT,rawCLK[20],CLR);
    JKFF_Q_POSCLK_POSCLR Clock22 (rawCLK[22],~HLT,~HLT,rawCLK[21],CLR);
    JKFF_Q_POSCLK_POSCLR Clock23 (rawCLK[23],~HLT,~HLT,rawCLK[22],CLR);
    JKFF_Q_POSCLK_POSCLR Clock24 (rawCLK[24],~HLT,~HLT,rawCLK[23],CLR);
    JKFF_Q_POSCLK_POSCLR Clock25 (rawCLK[25],~HLT,~HLT,rawCLK[24],CLR);
    JKFF_Q_POSCLK_POSCLR Clock26 (rawCLK[25],~HLT,~HLT,rawCLK[24],CLR);

endmodule
```

E-10.6 Active Low SR Latch

```
module SR_LATCH(Q,Qnot,Snot,Rnot);

    output Q, Qnot;
    input Snot, Rnot;

    or g1(Q,~Snot,~Qnot),
    g2(Qnot,~Rnot,~Q);

endmodule
```

E-10.7 Positive-edge-triggered JK Flip-flop with Active High Clear

```
module JKFF_Q_POSCLK_POSCLR(Q,J,K,CLK,CLR);  
    input J,K,CLK,CLR;  
    output Q;  
    reg Q;  
  
    always @(posedge CLK or posedge CLR)  
    begin  
        if (CLR) Q = 1'b0;  
        else if ({J,K} == 2'b00) Q = Q;  
        else if ({J,K} == 2'b01) Q = 1'b0;  
        else if ({J,K} == 2'b10) Q = 1'b1;  
        else if ({J,K} == 2'b11) Q = ~Q;  
    end  
  
endmodule
```


APPENDIX F **PIN ASSIGNMENTS, PIN INTERCONNECTIONS, AND INPUT & OUTPUT DEVICE UTILIZATION OF THE MODULAR SAP-1 PROTOTYPE**

Abbreviations

- WW - Wire through wire wrap
- IDC - IDC socket & ribbon cable through pin header

F-1 Program Counter

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	CLR	1	12	MAX_SW2[0]	WW
Input	CLK	2	14	From CLK of Clock Buffer	WW through extra PCB
Input	Cp	4	15	From Cp of Cont/Seq	WW
Input	Ep	84	13	From Ep of Cont/Seq	WW
Output	WBus[7]	49	49	WBus PCB expansion	IDC through extra PCB
	WBus[6]	50	50		
	WBus[5]	51	51		
	WBus[4]	52	52		
	WBus[3]	54	53		
	WBus[2]	55	54		
	WBus[1]	56	55		
	WBus[0]	57	56		
Output	BLANK_DISPLAY[7]	58	-	MAX_DIGIT (MSD)	- (Routed on PCB)
	BLANK_DISPLAY[6]	60	-		
	BLANK_DISPLAY[5]	61	-		
	BLANK_DISPLAY[4]	63	-		
	BLANK_DISPLAY[3]	64	-		
	BLANK_DISPLAY[2]	65	-		
	BLANK_DISPLAY[1]	67	-		
	BLANK_DISPLAY[0]	68	-		
Output	PC_HEX[7]	69	-	MAX_DIGIT (LSD)	- (Routed on PCB)
	PC_HEX[6]	70	-		
	PC_HEX[5]	73	-		
	PC_HEX[4]	74	-		
	PC_HEX[3]	76	-		
	PC_HEX[2]	75	-		
	PC_HEX[1]	77	-		
	PC_HEX[0]	79	-		
Output	Cpnot	81	-	LED D1	WW
Output	Epnot	80	-	LED D9	WW

F-2 MAR & 2 to 1 Multiplexer

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	<i>CLK</i>	2	14	From <i>CLK</i> of Clock Buffer	WW through extra PCB
Input	<i>Lm</i>	4	15	From <i>Lm</i> of Cont/Seq	WW
Input	<i>RUN_PROG</i>	6	17	From <i>RUN_PROG</i> of RAM	WW
Input	<i>ADDR_IN[2]</i>	18	26	MAX_SW1[7]	WW
	<i>ADDR_IN[1]</i>	20	27	MAX_SW1[6]	
	<i>ADDR_IN[0]</i>	21	28	MAX_SW1[5]	
Input	<i>WBus[7]</i>	49	49	<i>WBus</i> PCB expansion	IDC through extra PCB
	<i>WBus[6]</i>	50	50		
	<i>WBus[5]</i>	51	51		
	<i>WBus[4]</i>	52	52		
	<i>WBus[3]</i>	54	53		
	<i>WBus[2]</i>	55	54		
	<i>WBus[1]</i>	56	55		
Output	<i>MUX_OUT[3]</i>	35	39	To <i>MUX_OUT</i> of RAM	IDC
	<i>MUX_OUT[2]</i>	36	40		
	<i>MUX_OUT[1]</i>	37	41		
	<i>MUX_OUT[0]</i>	39	42		
Output	<i>BLANK_DISPLAY[7]</i>	58	-	MAX_DIGIT (MSD)	(Routed on PCB)
	<i>BLANK_DISPLAY[6]</i>	60	-		
	<i>BLANK_DISPLAY[5]</i>	61	-		
	<i>BLANK_DISPLAY[4]</i>	63	-		
	<i>BLANK_DISPLAY[3]</i>	64	-		
	<i>BLANK_DISPLAY[2]</i>	65	-		
	<i>BLANK_DISPLAY[1]</i>	67	-		
Output	<i>BLANK_DISPLAY[0]</i>	68	-	MAX_DIGIT (LSD)	(Routed on PCB)
	<i>MUX_OUT_HEX[7]</i>	69	-		
	<i>MUX_OUT_HEX[6]</i>	70	-		
	<i>MUX_OUT_HEX[5]</i>	73	-		
	<i>MUX_OUT_HEX[4]</i>	74	-		
	<i>MUX_OUT_HEX[3]</i>	76	-		
	<i>MUX_OUT_HEX[2]</i>	75	-		
Output	<i>MUX_OUT_HEX[1]</i>	77	-	LED D1	WW
	<i>MUX_OUT_HEX[0]</i>	79	-		
	<i>Lmnot</i>	81	-		

F-3 8 × 8 RAM

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	READ_WRITE	4	15	MAX_PB1	WW
Input	RUN_PROG	6	17	MAX_SW2[7]	WW
Input	CE	84	13	From CE of Cont/Seq	WW
Input	DATA_IN[7]	11	21	MAX_SW[7]	WW
	DATA_IN[6]	12	22	MAX_SW[6]	
	DATA_IN[5]	15	23	MAX_SW[5]	
	DATA_IN[4]	16	24	MAX_SW[4]	
	DATA_IN[3]	17	25	MAX_SW[3]	
	DATA_IN[2]	18	26	MAX_SW[2]	
	DATA_IN[1]	20	27	MAX_SW[1]	
	DATA_IN[0]	21	28	MAX_SW[0]	
Input	MUX_OUT[2]	36	40	From MUX_OUT of MAR & MUX	IDC
	MUX_OUT[1]	37	41		
	MUX_OUT[0]	39	42		
Output	WBus[7]	49	49	WBus PCB expansion	IDC through extra PCB
	WBus[6]	50	50		
	WBus[5]	51	51		
	WBus[4]	52	52		
	WBus[3]	54	53		
	WBus[2]	55	54		
	WBus[1]	56	55		
	WBus[0]	57	56		
Output	RAM_DISPLAY_HEX[15]	58	-	MAX_DIGIT (MSD)	(Routed on PCB)
	RAM_DISPLAY_HEX[14]	60	-		
	RAM_DISPLAY_HEX[13]	61	-		
	RAM_DISPLAY_HEX[12]	63	-		
	RAM_DISPLAY_HEX[11]	64	-		
	RAM_DISPLAY_HEX[10]	65	-		
	RAM_DISPLAY_HEX[9]	67	-		
	RAM_DISPLAY_HEX[8]	68	-		
Output	RAM_DISPLAY_HEX[7]	69	-	MAX_DIGIT (LSD)	(Routed on PCB)
	RAM_DISPLAY_HEX[6]	70	-		
	RAM_DISPLAY_HEX[5]	73	-		
	RAM_DISPLAY_HEX[4]	74	-		
	RAM_DISPLAY_HEX[3]	76	-		
	RAM_DISPLAY_HEX[2]	75	-		
	RAM_DISPLAY_HEX[1]	77	-		
	RAM_DISPLAY_HEX[0]	79	-		
Output	CEnot	80	-	LED_D9	WW

F-4 Instruction Register

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	<i>CLR</i>	1	12	MAX_SW2[0]	WW
Input	<i>CLK</i>	2	14	From <i>CLK</i> of Clock Buffer	WW through extra PCB
Input	<i>Li</i>	4	15	From <i>Li</i> of Cont/Seq	WW
Input	<i>Ei</i>	84	13	From <i>Ei</i> of Cont/Seq	WW
Output	<i>IR OUT INS</i> [3]	30	35	To <i>IR OUT INS</i> of Cont/Seq	IDC
	<i>IR OUT INS</i> [2]	31	36		
	<i>IR OUT INS</i> [1]	33	37		
	<i>IR OUT INS</i> [0]	34	38		
Inout	<i>WBus</i> [7]	49	49	<i>WBus</i> PCB expansion	IDC through extra PCB
	<i>WBus</i> [6]	50	50		
	<i>WBus</i> [5]	51	51		
	<i>WBus</i> [4]	52	52		
	<i>WBus</i> [3]	54	53		
	<i>WBus</i> [2]	55	54		
	<i>WBus</i> [1]	56	55		
Output	<i>WBus</i> [0]	57	56	MAX_DIGIT (MSD)	(Routed on PCB)
	<i>IR OUT INS HEX</i> [7]	58	-		
	<i>IR OUT INS HEX</i> [6]	60	-		
	<i>IR OUT INS HEX</i> [5]	61	-		
	<i>IR OUT INS HEX</i> [4]	63	-		
	<i>IR OUT INS HEX</i> [3]	64	-		
	<i>IR OUT INS HEX</i> [2]	65	-		
	<i>IR OUT INS HEX</i> [1]	67	-		
Output	<i>IR OUT INS HEX</i> [0]	68	-	MAX_DIGIT (LSD)	(Routed on PCB)
	<i>IR OUT ADDR HEX</i> [7]	69	-		
	<i>IR OUT ADDR HEX</i> [6]	70	-		
	<i>IR OUT ADDR HEX</i> [5]	73	-		
	<i>IR OUT ADDR HEX</i> [4]	74	-		
	<i>IR OUT ADDR HEX</i> [3]	76	-		
	<i>IR OUT ADDR HEX</i> [2]	75	-		
	<i>IR OUT ADDR HEX</i> [1]	77	-		
Output	<i>IR OUT ADDR HEX</i> [0]	79	-	LED D1	WW
	<i>Einot</i>	81	-		
Output	<i>Einot</i>	80	-	LED D9	WW

F-5 Accumulator

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	<i>CLK</i>	2	14	From <i>CLK</i> of Clock Buffer	WW through extra PCB
Input	<i>La</i>	4	15	From <i>La</i> of Cont/Seq	WW
Input	<i>Ea</i>	84	13	From <i>Ea</i> of Cont/Seq	WW
Inout	<i>WBus[7]</i>	49	49	<i>WBus</i> PCB expansion	IDC through extra PCB
	<i>WBus[6]</i>	50	50		
	<i>WBus[5]</i>	51	51		
	<i>WBus[4]</i>	52	52		
	<i>WBus[3]</i>	54	53		
	<i>WBus[2]</i>	55	54		
	<i>WBus[1]</i>	56	55		
	<i>WBus[0]</i>	57	56		
Output	<i>ACCU_OUT[7]</i>	30	35	To <i>ACCU_OUT</i> of Adder/Subtractor	IDC
	<i>ACCU_OUT[6]</i>	31	36		
	<i>ACCU_OUT[5]</i>	33	37		
	<i>ACCU_OUT[4]</i>	34	38		
	<i>ACCU_OUT[3]</i>	35	39		
	<i>ACCU_OUT[2]</i>	36	40		
	<i>ACCU_OUT[1]</i>	37	41		
	<i>ACCU_OUT[0]</i>	39	42		
Output	<i>ACCU_OUT_HEX[15]</i>	58	-	MAX_DIGIT (MSD)	- (Routed on PCB)
	<i>ACCU_OUT_HEX[14]</i>	60	-		
	<i>ACCU_OUT_HEX[13]</i>	61	-		
	<i>ACCU_OUT_HEX[12]</i>	63	-		
	<i>ACCU_OUT_HEX[11]</i>	64	-		
	<i>ACCU_OUT_HEX[10]</i>	65	-		
	<i>ACCU_OUT_HEX[9]</i>	67	-		
	<i>ACCU_OUT_HEX[8]</i>	68	-		
Output	<i>ACCU_OUT_HEX[7]</i>	69	-	MAX_DIGIT (LSD)	- (Routed on PCB)
	<i>ACCU_OUT_HEX[6]</i>	70	-		
	<i>ACCU_OUT_HEX[5]</i>	73	-		
	<i>ACCU_OUT_HEX[4]</i>	74	-		
	<i>ACCU_OUT_HEX[3]</i>	76	-		
	<i>ACCU_OUT_HEX[2]</i>	75	-		
	<i>ACCU_OUT_HEX[1]</i>	77	-		
	<i>ACCU_OUT_HEX[0]</i>	79	-		
Output	<i>Lanot</i>	81	-	LED D1	WW
Output	<i>Eanot</i>	80	-	LED D9	WW

F-7 B Register

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	<i>CLK</i>	2	14	From <i>CLK</i> of Clock Buffer	WW through extra PCB
Input	<i>Lb</i>	4	15	From <i>Lb</i> of Cont/Seq	WW
Input	<i>WBus[7]</i>	49	49	<i>WBus</i> PCB expansion	IDC through extra PCB
	<i>WBus[6]</i>	50	50		
	<i>WBus[5]</i>	51	51		
	<i>WBus[4]</i>	52	52		
	<i>WBus[3]</i>	54	53		
	<i>WBus[2]</i>	55	54		
	<i>WBus[1]</i>	56	55		
Output	<i>WBus[0]</i>	57	56		
	<i>B REG OUT[7]</i>	11	21	To <i>B_REG_OUT</i> of Adder/Subtractor	IDC
	<i>B REG OUT[6]</i>	12	22		
	<i>B REG OUT[5]</i>	15	23		
	<i>B REG OUT[4]</i>	16	24		
	<i>B REG OUT[3]</i>	17	25		
	<i>B REG OUT[2]</i>	18	26		
	<i>B REG OUT[1]</i>	20	27		
Output	<i>B REG OUT[0]</i>	21	28		
	<i>B REG OUT HEX[15]</i>	58	-	MAX_DIGIT (MSD)	- (Routed on PCB)
	<i>B REG OUT HEX[14]</i>	60	-		
	<i>B REG OUT HEX[13]</i>	61	-		
	<i>B REG OUT HEX[12]</i>	63	-		
	<i>B REG OUT HEX[11]</i>	64	-		
	<i>B REG OUT HEX[10]</i>	65	-		
	<i>B REG OUT HEX[9]</i>	67	-		
Output	<i>B REG OUT HEX[8]</i>	68	-	MAX_DIGIT (LSD)	- (Routed on PCB)
	<i>B REG OUT HEX[7]</i>	69	-		
	<i>B REG OUT HEX[6]</i>	70	-		
	<i>B REG OUT HEX[5]</i>	73	-		
	<i>B REG OUT HEX[4]</i>	74	-		
	<i>B REG OUT HEX[3]</i>	76	-		
	<i>B REG OUT HEX[2]</i>	75	-		
Output	<i>B REG OUT HEX[1]</i>	77	-		
	<i>B REG OUT HEX[0]</i>	79	-		
Output	<i>Lbnot</i>	81	-	LED D1	WW

F-8 Output Register

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	CLK	2	14	From CLK of Clock Buffer	WW through extra PCB
Input	Lo	4	15	From Lo of Cont/Seq	WW
Input	WBus[7]	49	49	WBus PCB expansion	IDC through extra PCB
	WBus[6]	50	50		
	WBus[5]	51	51		
	WBus[4]	52	52		
	WBus[3]	54	53		
	WBus[2]	55	54		
	WBus[1]	56	55		
Output	OUT REG HEX[15]	57	56	MAX_DIGIT (MSD)	- (Routed on PCB)
	OUT REG HEX[14]	58	-		
	OUT REG HEX[13]	60	-		
	OUT REG HEX[12]	61	-		
	OUT REG HEX[11]	63	-		
	OUT REG HEX[10]	64	-		
	OUT REG HEX[9]	65	-		
Output	OUT REG HEX[8]	67	-	MAX_DIGIT (LSD)	- (Routed on PCB)
	OUT REG HEX[7]	68	-		
	OUT REG HEX[6]	69	-		
	OUT REG HEX[5]	70	-		
	OUT REG HEX[4]	73	-		
	OUT REG HEX[3]	74	-		
	OUT REG HEX[2]	76	-		
Output	OUT REG HEX[1]	75	-	LED DI	WW
	OUT REG HEX[0]	77	-		
Output	Lonot	81	-		

F-9 Controller/Sequencer (Instruction Decoder, Ring Counter & Control Matrix)

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	<i>CLR</i>	1	12	From <i>CLR</i> of De-bouncers	WW
Input	<i>CLK</i>	2	14	From <i>CLK</i> of Clock Buffer	WW through extra PCB
Input	<i>IR_OUT_INS[3]</i>	30	35	From <i>IR_OUT_INS</i> of Instruction Register	IDC
	<i>IR_OUT_INS[2]</i>	31	36		
	<i>IR_OUT_INS[1]</i>	33	37		
	<i>IR_OUT_INS[0]</i>	34	38		
Output	<i>HLT</i>	51	51	To <i>HLT</i> of Mode-Select Switches, De-bouncers, and Clock Buffer	WW
Output	<i>Cp</i>	9	19	To <i>Cp</i> of Program Counter	WW
	<i>Ep</i>	10	20	To <i>Ep</i> of Program Counter	
	<i>Lm</i>	11	21	To <i>Lm</i> of MAR & MUX	
	<i>CE</i>	12	22	To <i>CE</i> of RAM	
	<i>Li</i>	15	23	To <i>Li</i> of Instruction Register	
	<i>Ei</i>	16	24	To <i>Ei</i> of Instruction Register	
	<i>La</i>	17	25	To <i>La</i> of Accumulator	
	<i>Ea</i>	18	26	To <i>Ea</i> of Accumulator	
	<i>Su</i>	20	27	To <i>Su</i> of Adder/Subtractor	
	<i>Eu</i>	21	28	To <i>Eu</i> of Adder/Subtractor	
	<i>Lb</i>	22	29	To <i>Lb</i> of B Register	
	<i>Lo</i>	24	30	To <i>Lo</i> of Output Register	
Output	<i>Tnot[6]</i>	44	45	LED D7	WW
	<i>Tnot[5]</i>	45	46	LED D6	
	<i>Tnot[4]</i>	46	47	LED D5	
	<i>Tnot[3]</i>	48	48	LED D3	
	<i>Tnot[2]</i>	49	49	LED D2	
	<i>Tnot[1]</i>	50	50	LED D1	
Output	<i>HLTnot</i>	52	52	LED D13	WW
	<i>OUTnot</i>	54	53	LED D12	
	<i>SUBnot</i>	55	54	LED D11	
	<i>ADDnot</i>	56	55	LED D10	
	<i>LDAnot</i>	57	56	LED D9	
Output	<i>BLANK_DISPLAY[15]</i>	58	-	MAX_DIGIT (MSD)	(Routed on PCB)
	<i>BLANK_DISPLAY[14]</i>	60	-		
	<i>BLANK_DISPLAY[13]</i>	61	-		
	<i>BLANK_DISPLAY[12]</i>	63	-		
	<i>BLANK_DISPLAY[11]</i>	64	-		
	<i>BLANK_DISPLAY[10]</i>	65	-		
	<i>BLANK_DISPLAY[9]</i>	67	-		
	<i>BLANK_DISPLAY[8]</i>	68	-		
Output	<i>BLANK_DISPLAY[7]</i>	69	-	MAX_DIGIT (LSD)	(Routed on PCB)
	<i>BLANK_DISPLAY[6]</i>	70	-		
	<i>BLANK_DISPLAY[5]</i>	73	-		
	<i>BLANK_DISPLAY[4]</i>	74	-		
	<i>BLANK_DISPLAY[3]</i>	76	-		
	<i>BLANK_DISPLAY[2]</i>	75	-		
	<i>BLANK_DISPLAY[1]</i>	77	-		
	<i>BLANK_DISPLAY[0]</i>	79	-		

F-10 Mode-Select Switches, De-bouncers & Clock Buffer

Signal Type	Signal	Pin	Expan. Hole	Input/Output Device / Interconnection	Connector
Input	<i>START CLEAR</i>	33	37	MAX_SW[0]	WW
Input	<i>LOW</i>	35	39	SPDT switch on extra veroboard	WW
	<i>HIGH</i>	36	40		
Input	<i>MANUAL AUTO</i>	37	41	MAX_SW[7]	WW
Input	<i>HLT</i>	39	42	From <i>HLT</i> of Cont/Seq	WW
Input	<i>rawrawCLK</i>	83	-	On-board oscillator	- (Routed on PCB)
Output	<i>CLR</i>	31	36	To <i>CLR</i> of Cont/Seq	WW
Output	<i>CLK</i>	30	35	To <i>CLK</i> PCB expansion	WW through extra PCB
Output	<i>BLANK_DISPLAY[15]</i>	58	-	MAX_DIGIT (MSD)	- (Routed on PCB)
	<i>BLANK_DISPLAY[14]</i>	60	-		
	<i>BLANK_DISPLAY[13]</i>	61	-		
	<i>BLANK_DISPLAY[12]</i>	63	-		
	<i>BLANK_DISPLAY[11]</i>	64	-		
	<i>BLANK_DISPLAY[10]</i>	65	-		
	<i>BLANK_DISPLAY[9]</i>	67	-		
	<i>BLANK_DISPLAY[8]</i>	68	-		
Output	<i>BLANK_DISPLAY[7]</i>	69	-	MAX_DIGIT (LSD)	- (Routed on PCB)
	<i>BLANK_DISPLAY[6]</i>	70	-		
	<i>BLANK_DISPLAY[5]</i>	73	-		
	<i>BLANK_DISPLAY[4]</i>	74	-		
	<i>BLANK_DISPLAY[3]</i>	76	-		
	<i>BLANK_DISPLAY[2]</i>	75	-		
	<i>BLANK_DISPLAY[1]</i>	77	-		
	<i>BLANK_DISPLAY[0]</i>	79	-		
Output	<i>CLRnot</i>	52	52	LED D9	WW
Output	<i>CLKnot</i>	51	51	LED D1	WW

APPENDIX G
PHOTOS OF MODULAR SAP-1 PROTOTYPE

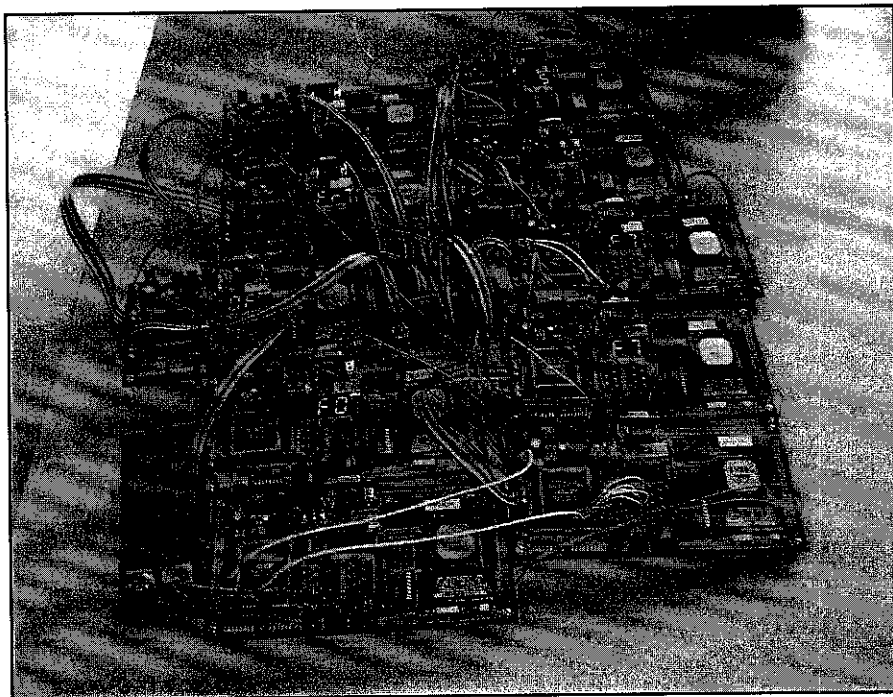


FIGURE 19: Modular SAP-1 prototype, picture 1

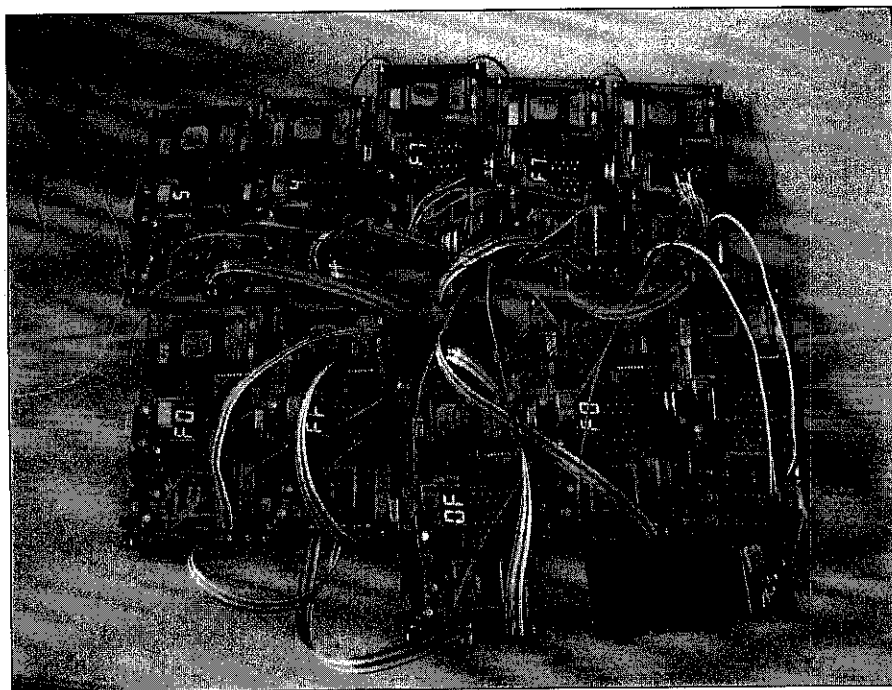


FIGURE 20: Modular SAP-1 prototype, picture 2