Design Methodology for Implementing a Micro-controller in an FPGA.

Outline

- Introduction
- Design Process - From goals to implementation
- Results

Inroduction

- Background Information
- Microcontrollers VS. Microprocessors
- Intel 8031 defined
- Goals of the Design Process

Background Information

- modeling a microcontroller, the 8031
- implementing the design in an FPGA

MicroControllers versus MicroProcessors

- MicroP's are a general purpose machine
- MicroC's are a true computer on a chip
- MicroP's need additional components to make a complete system
- MicroC's have all necessary features including, ROM,RAM, parallel I/O, etc.

Project focal point, Intel 8031

- 8-bit CPU
- Extensive Boolean processing
- 64K Data & Memory Space
- 128 bytes of on-chip Data Ram
- 32 bidirectional/individually addressable I/O lines
- 2 16-bit timer/counters
- Full Duplex UART
- 6-source/5-vector interrupt structure

Intel 8031 Architecture Overview

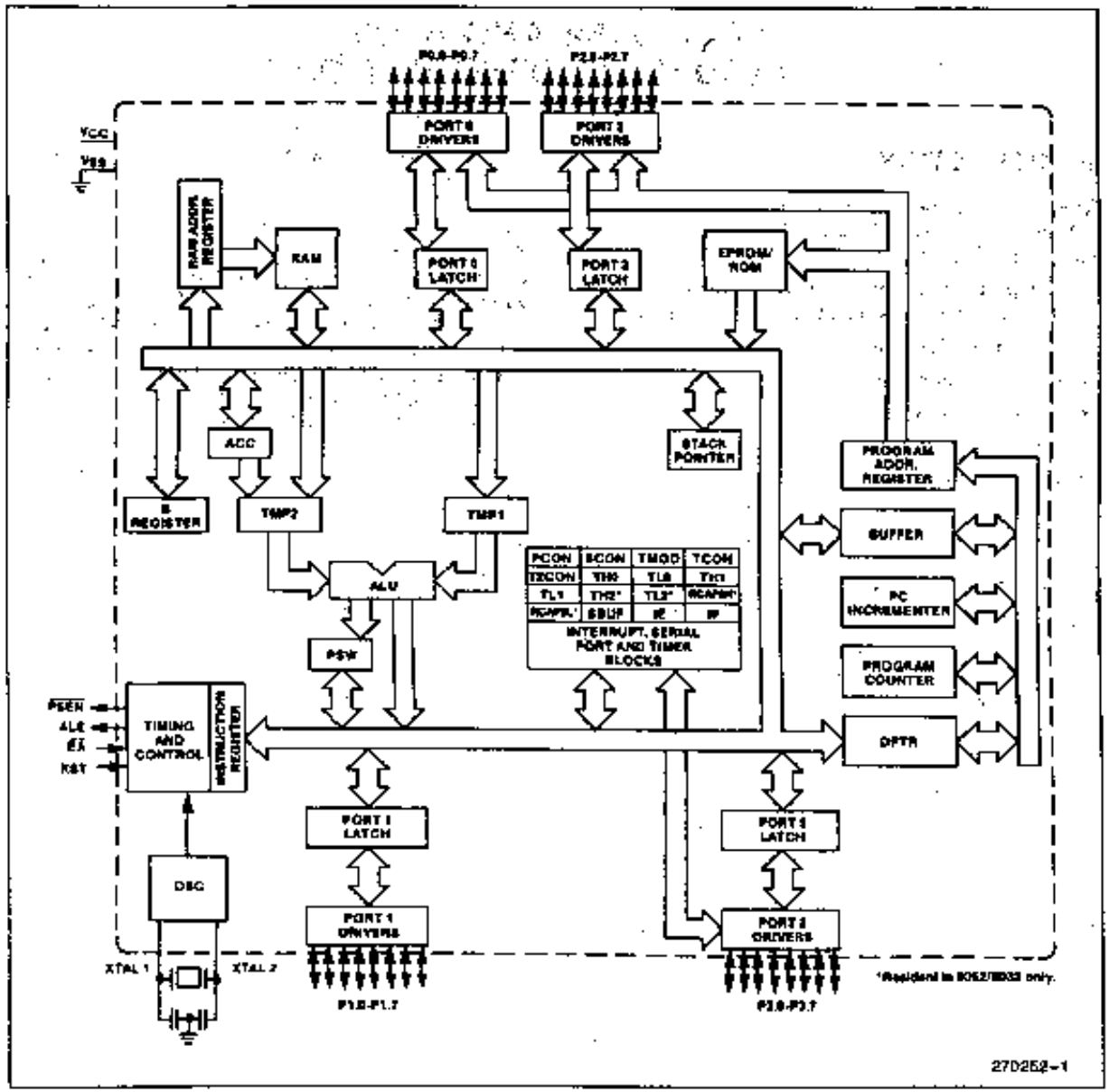


Figure 1. MCS-51 Architectural Block Diagram

Goals of the Design Process

- To develop an accurate behavioral model of 8031 in VHDL
- To develop an accurate *RTL VHDL model* of the 8031
- Synthesize the RTL model
- Successfully **implement** the synthesized model in a **Xilinx FPGA**

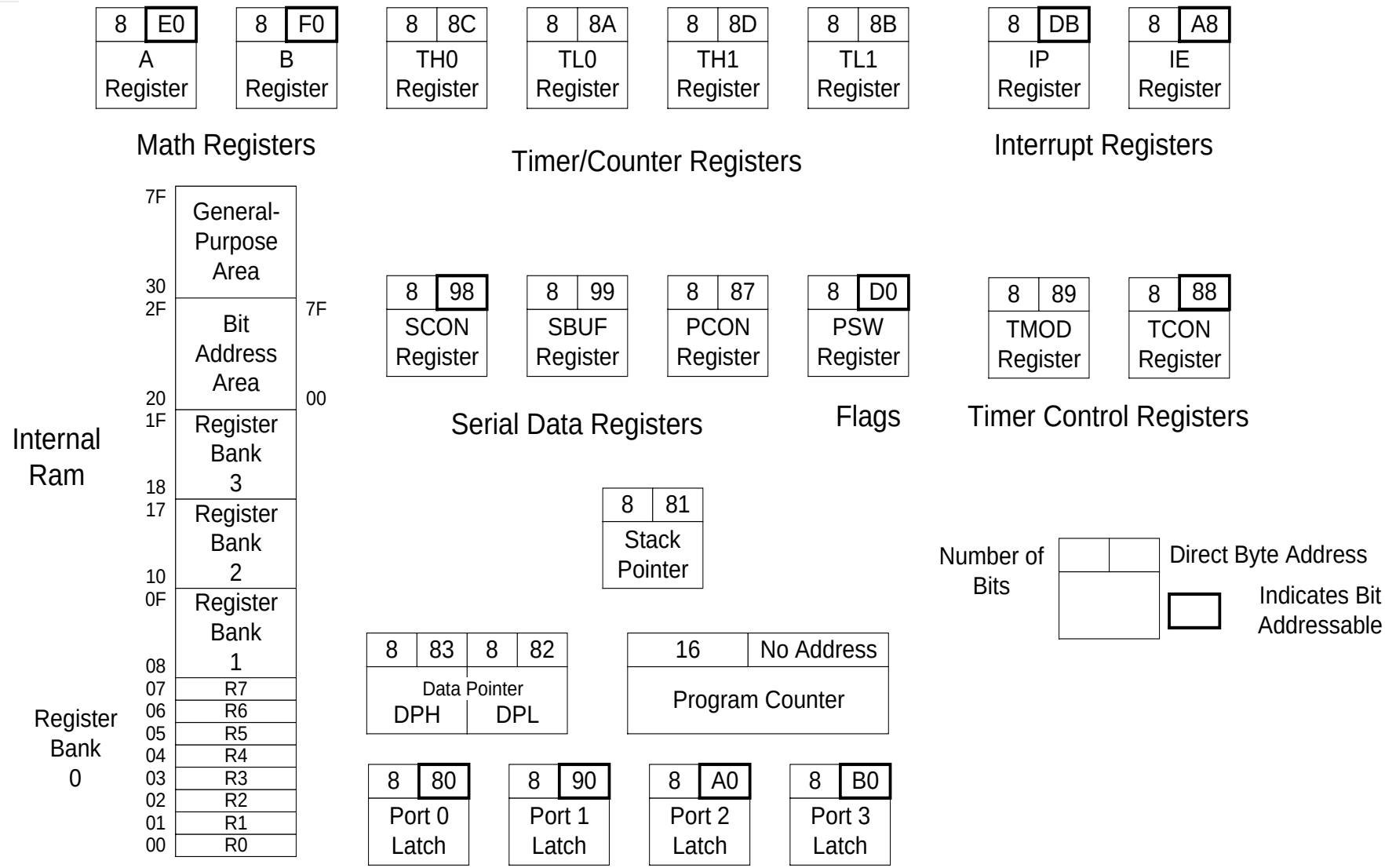
The Design Process, Part 1

- **Define** the following:
  - register structure,
  - instruction set,
  - addressing modes.
- Construct a table showing register transfers and State Machine graph
- Design the control state machine
- Write ***behavioral VHDL code*** based on the above completed tasks
- **Simulate** execution to verify accurate modeling

The Design Process, Part 2

- Develop block diagram of major units and *determine control signals*
- Rewrite VHDL based on previous step
- Again, simulate execution to verify model
- Make needed changes in code for Synthesis
- **Synthesize the controller** from the VHDL code
- Download bit stream file to FPGA for ***hardware verification***

Step 1, Define Register Structure, Instruction Set, & Addressing Modes



Step 1- Instruction Set

Arithmetic Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (uS)
		Dir	Ind	Reg	Imm	
ADD A,<byte>	A = A + <byte>	X	X	X	X	1
ADDC A,<byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A, <byte>	A = A - <byte> - C	X	X	X	X	1
INC A	A = A + 1	Accumulator only				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A = A - 1	Accumulator only				1
DEC <byte>	<byte> = <byte> - 1	X	X	X		1
MUL AB	B:A = BxA	ACC and B only				4
DIV AB	A = Int [A/B] B = Mod [A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

Step 1- Instruction Set

Logical Instructions

Mnemonic	Operation	Addressing Modes				Execution Time (uS)
		Dir	Ind	Reg	Imm	
ANL A, <byte>	A = A .AND. <byte>	X	X	X	X	1
ANL <byte>,A	<byte> = <byte> .AND. A	X				1
ANL <byte>, #data	<byte> = <byte> .AND. # data	X				2
ORL A, <byte>	A = A . OR. <byte>	X	X	X	X	1
ORL <byte>,A	<byte> = <byte> .OR. A	X				1
ORL <byte>, #data	<byte> = <byte> .OR. # data	X				2
XRL A, <byte>	A = A . XRL. <byte>	X	X	X	X	1
XRL <byte>,A	<byte> = <byte> .XRL. A	X				1
XRL <byte>, #data	<byte> = <byte> .XRL. # data	X				2
CRL A	A = 00H	Accumulator only				1
CPL A	A = .NOT. A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

Step 1- Instruction Set

Internal Data Memory Data Transfer

Mnemonic	Operation	Addressing Modes				Execution Time (uS)
		Dir	Ind	Reg	Imm	
MOV A, <src>	A = <src>	X	X	X	X	1
MOV <dest>,A	<dest> = A	X	X	X		1
MOV <dest>, <src>	<dest> = <src>	X	X	X	X	2
MOV DPTR,#data16	DPTR = 16-bit imm constant				X	1
PUSH <src>	INC SP: MOV “@SP”: DEC SP	X				1
POP <dest>	MOV <dest>, “@SP”: DEC SP	X				2
XCH A,<byte>	ACC and <byte> exchange byte	X	X	X		1
XCHD A,@Ri	ACC and @Ri exchange low nibbles		X			1

Step 1- Instruction Set

External Data Memory Data Transfer

Address Width	Mnemonic	Operation	Execution Time (uS)
8 bits	MOVX A, @Ri	Read external Ram@Ri	2
8 bits	MOVX @RI, A	Write external RAM@Ri	2
16 bits	MOVX A,@DPTR	Read external RAM@DPTR	2
16 bits	MOVX @DPTR,A	Read external RAM@DPTR	2

Step 1- Instruction Set

Lookup Table Read Instructions

Mnemonic	Operation	Execution Time (uS)
MOVC A,@A + DPTR	Read Pgm Memory at(A + DPTR)	2
MOVC A,@A + PC	Read Pgm Memory at(A + PC)	2

Step 1- Instruction Set

Boolean Instructions

Mnemonic	Operation	Execution Time (uS)
ANL C,bit	C = C.AND. bit	2
ANL C,/bit	C = C.AND. .NOT. bit	2
ORL C,bit	C = C .OR. bit	2
ORL C,/bit	C = C .OR. .NOT. bit	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = .NOT.C	1
CPL bit	bit = .NOT. bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if bit = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump is bit = 0	2
JBC bit,rel	Jump if bit = 1; CLR bit	2

Step 1- Instruction Set

Unconditional Jumps

Mnemonic	Operation	Execution Time (uS)
JMP addr	Jump to addr	2
JMP @A + DPTR	Jump to A + DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

Step 1- Instruction Set

Conditional Jumps

Mnemonic	Operation	Addressing Modes				Execution Time (uS)
		Dir	Ind	Reg	Imm	
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump is A ≠ 0	Accumulator only				2
DJNZ <byte>,rel	Decrement and jump is not zero	X		X		
CJNE A,<byte>,rel	Jump if A ≠ <byte>	X			X	
CJNE <byte>, #data,rel	Jump if <byte> ≠ # data		X	X		2

Step 1 - Addressing Modes

- Direct Addressing
  - Only internal Data Ram and external Ram and SFR's can be directly addressed
- Indirect Addressing
  - Both internal and external Ram can be indirectly addressed
  - The address register for 8-bit addresses can be R0 or R1 of the current register bank, or the Stack Pointer
  - The address register for 16-bit addresses can be only be the 16-bit "data pointer" register, DPTR

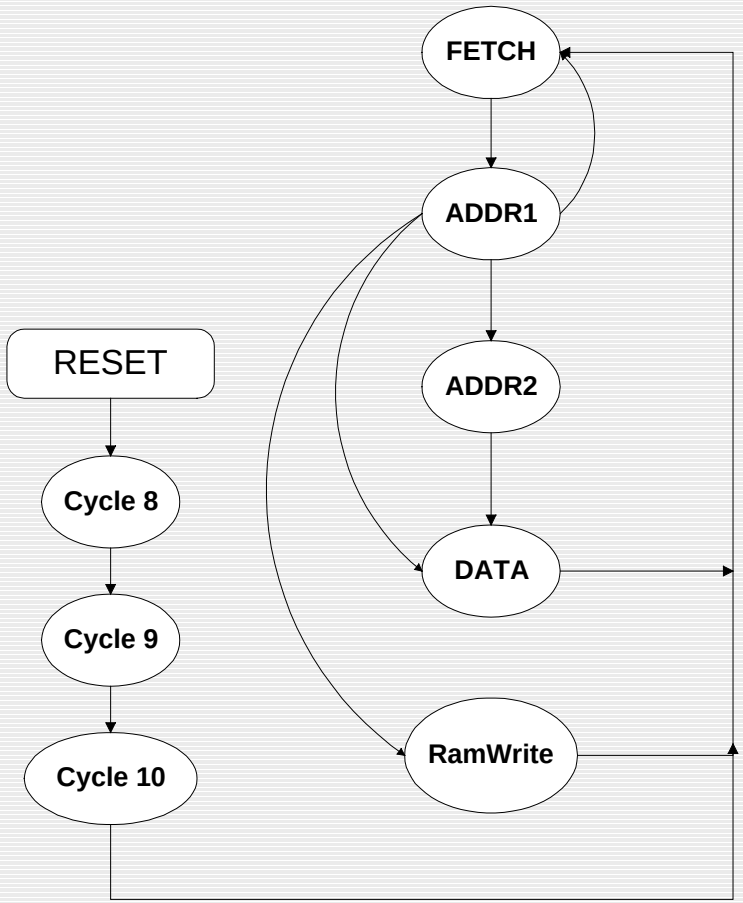
Step 1 - Addressing Modes

- Register Addressing
  - Opcodes that use register addressing use a single byte for identifying the instruction and the register
  - One of four banks is selected at execution time by the two bank select bits in the PSW
- Immediate Addressing
  - The value of a constant can follow the opcode in Program Memory

Step 2 - Register Transfer Table

		1st Cycle	2nd Cycle	3 rd Cycle	4th Cycle
Addressing Mode					
Immediate	**Add A, #data**	**{fetch}**	**{addr1}** **Tmp1 ← mem(PC)** **PC ← PC + 1**	**(A ← A + Tmp1)**	
Direct	**Add A, Direct**	**{fetch}**	**{addr1}** **Rar ← mem(PC)** **PC ← PC + 1**	**{data}** **Tmp1 ← Ram(Rar)**	**(A ← A + Tmp1)**
Direct	**MOV Direct, A**	**{fetch}**	**{addr1}** **Rar ← mem(PC)** **PC ← PC + 1**	**{RamWrite}** **Ram(Rar) ← A**	
Register	**Add A, Rn**	**{fetch}**	**{addr1}** **Rar ← mem(PC)** **PC ← PC + 1**	**{data}** **Tmp1 ← Ram(Rar)**	**(A ← A + Tmp1)**
Indirect	**Add A, @Ri**	**{fetch}**	**{addr1}** **Rar ← mem(PC)** **PC ← PC + 1**	**{data}** **Tmp1 ← Ram(Rar)**	**(A ← A + Tmp1)**
	LJMP	**{fetch}**	**{addr1}** **MarH ← mem(PC)** **PC ← PC + 1**	**{addr2}** **PCL ← mem(PC)** **PCH ← MarH**	

Step 2 - State Machine Graph



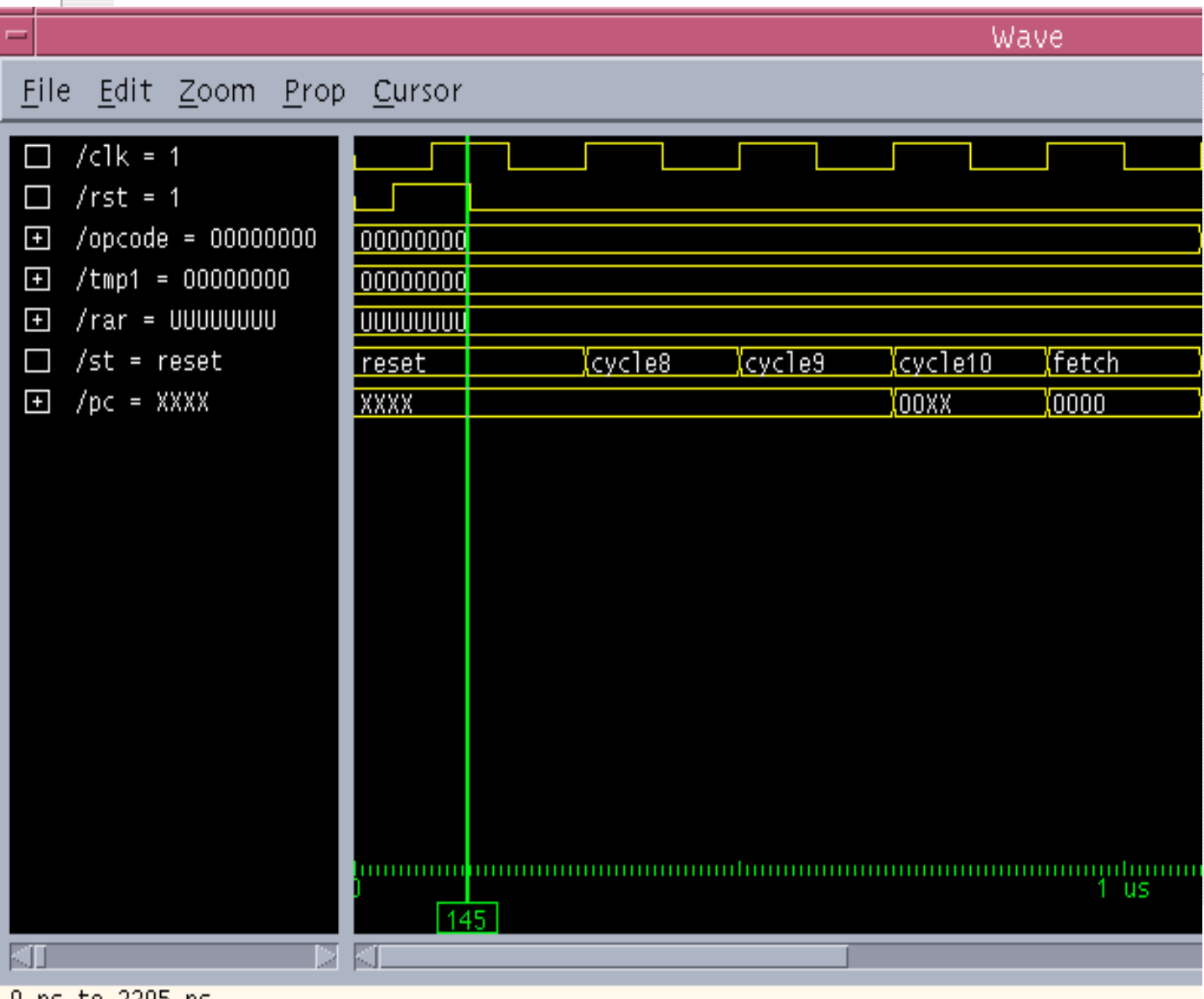
Step 3, 4 Design the control state machine and Write behavioral VHDL

- VHDL code was written based on State Machine Flow Graph and Register Transfer Table

Step 5 , Simulate model to verify accurate modeling

- Simulation was performed using Mentor Graphics Quick VHDL
- A short program was used to verify execution
- Program performs simple addition and data transfers

Step 5, Behavioral Simulation Results 1



Program

LJMP 0AH

MOV A, #08h

MOV PSW, A

MOV A, #04h

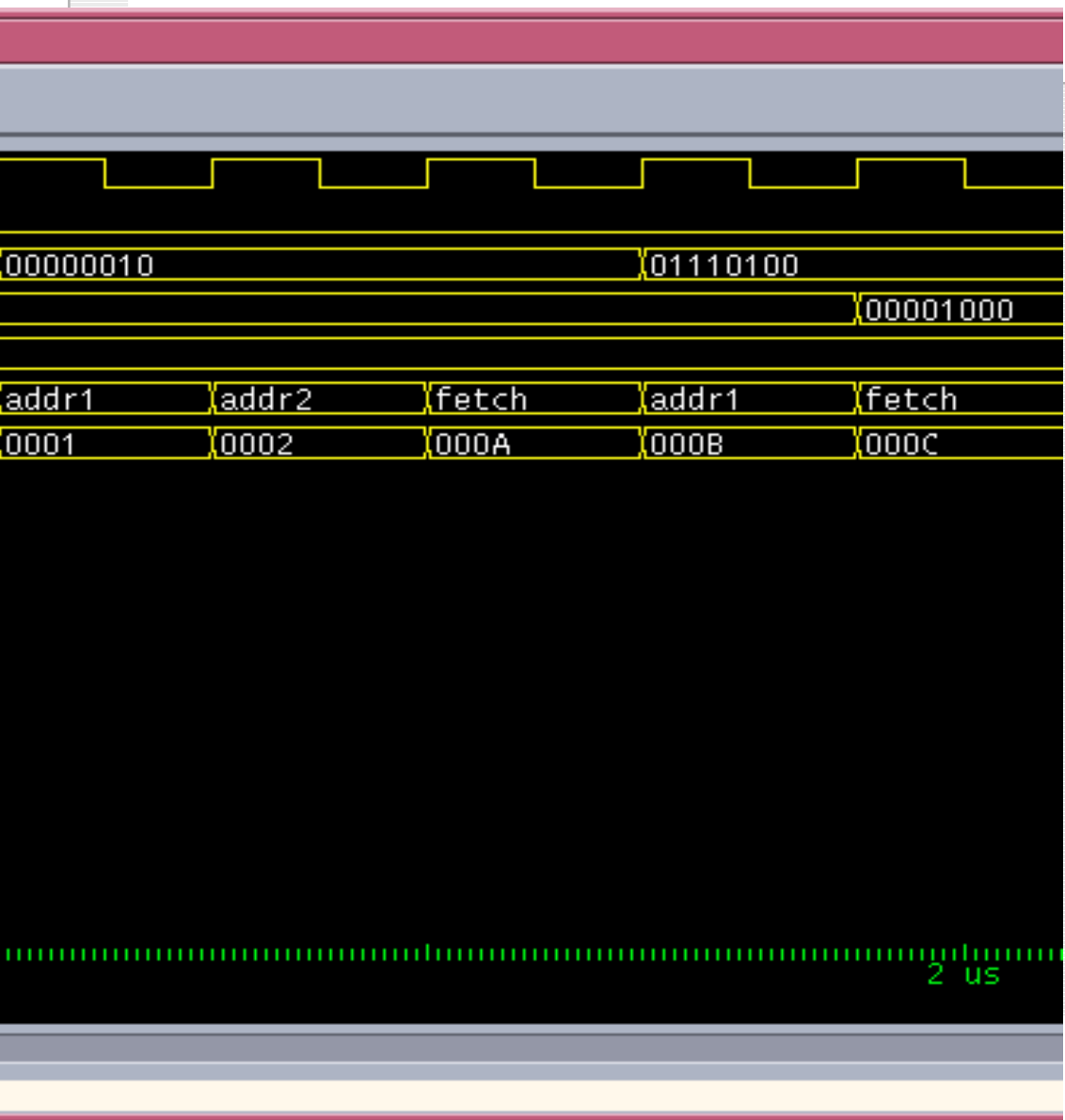
MOV 08h, A

MOV 09h, A

ADD A, R1

ADD A, R0

Step 5, Behavioral Simulation Results 2



Program

LJMP 0AH

MOV A, #08h

MOV PSW, A

MOV A, #04h

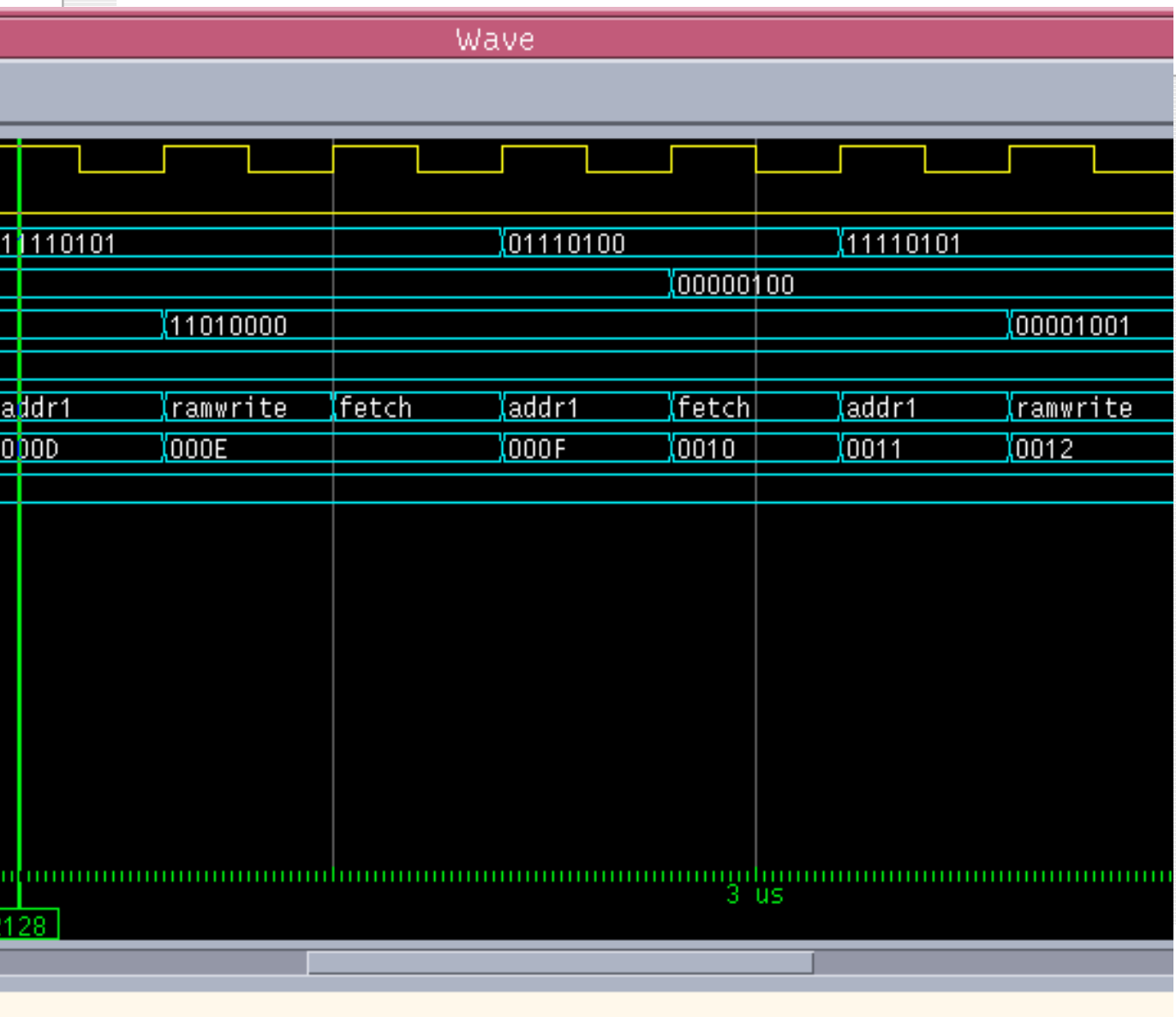
MOV 08h, A

MOV 09h, A

ADD A, R1

ADD A, R0

Step 5, Behavioral Simulation Results 3



Program

LJMP 0AH

MOV A, #08h

MOV PSW, A

MOV A, #04h

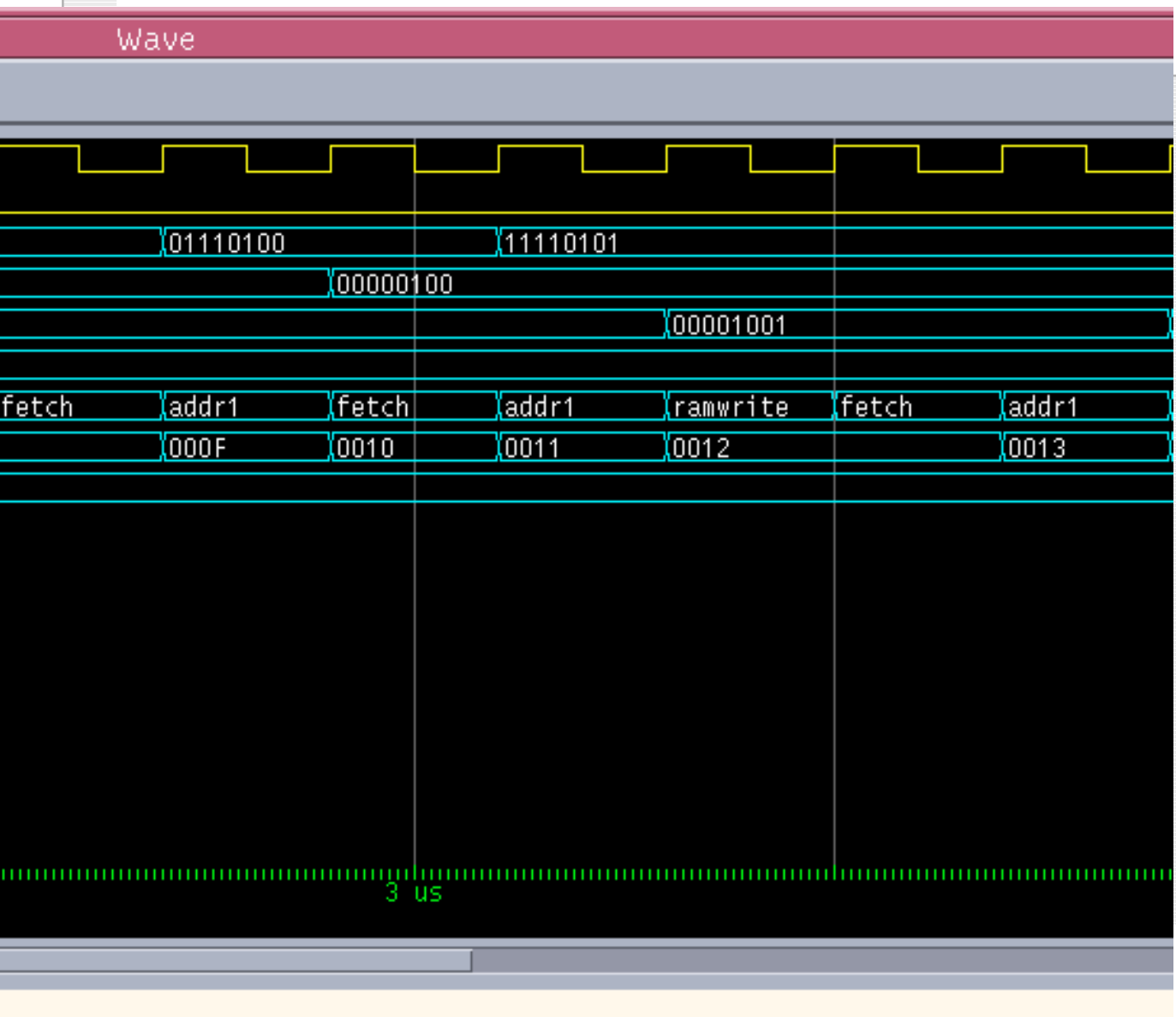
MOV 08h, A

MOV 09h, A

ADD A, R1

ADD A, R0

Step 5, Behavioral Simulation Results 4



Program

LJMP 0AH

MOV A, #08h

MOV PSW, A

MOV A, #04h

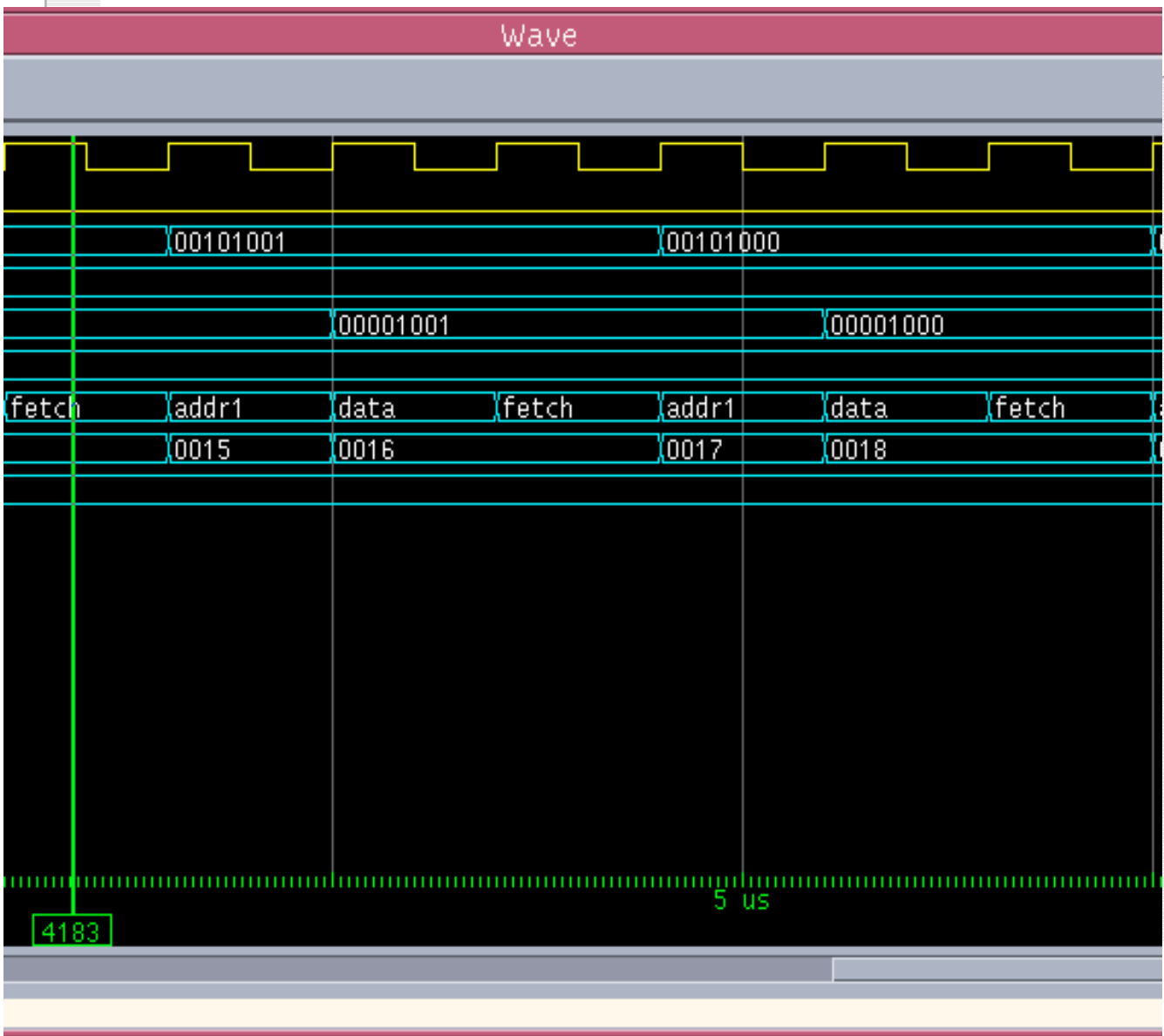
MOV 08h, A

MOV 09h, A

ADD A, R1

ADD A, R0

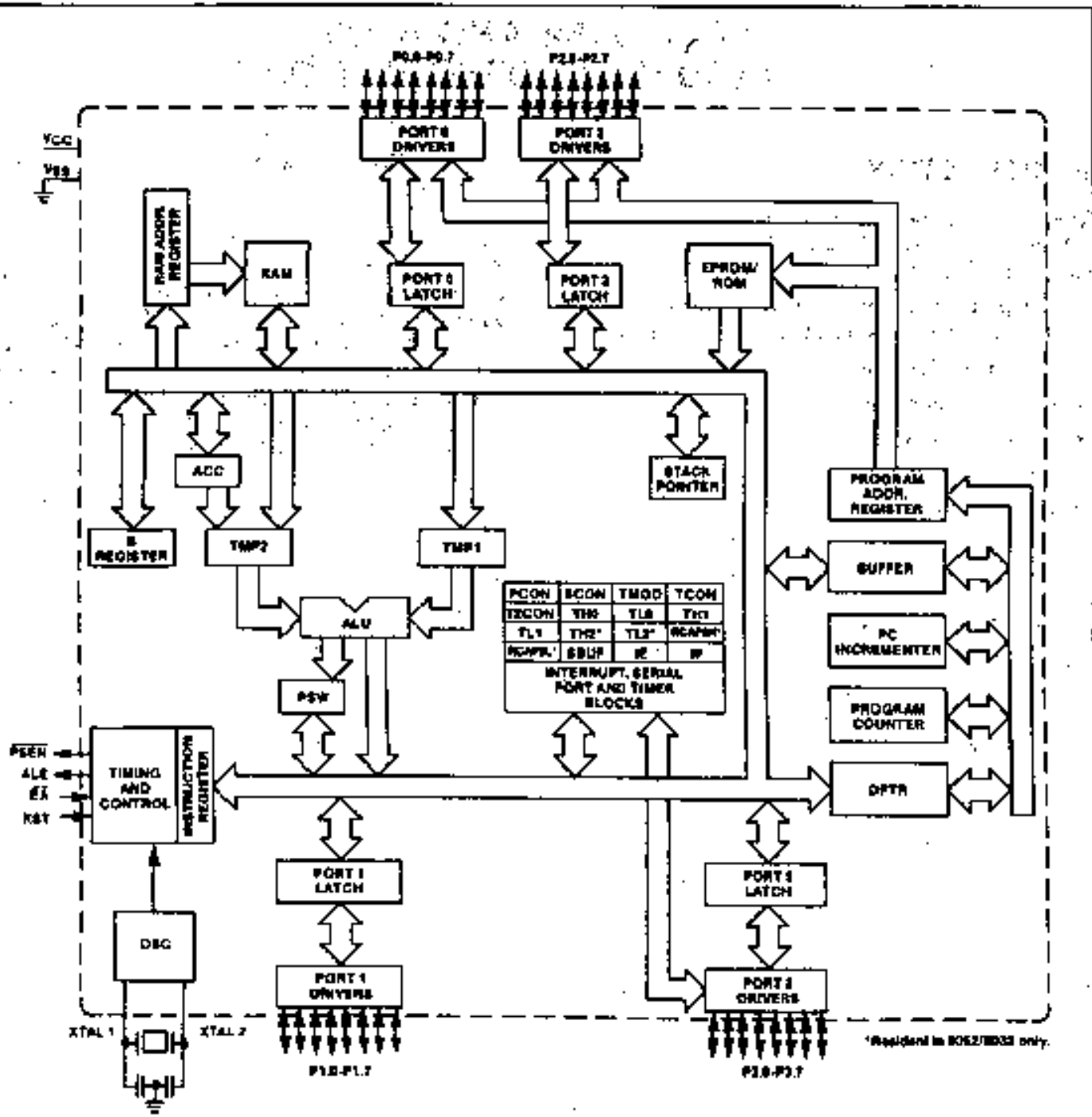
Step 5, Behavioral Simulation Results 5



- Program
- LJMP 0AH
- MOV A, #08h
- MOV PSW, A
- MOV A, #04h
- MOV 08h, A
- MOV 09h, A
- ADD A, R1
- ADD A, R0

The Design Process, Part 2

- Develop block diagram of major units and determine control signals
- Rewrite VHDL based on previous step
- Again, simulate execution to verify model
- Make needed changes in code for Synthesis
- Synthesize the controller from the VHDL code
- Download bit stream file to FPGA for hardware verification


P0.0-P0.7
P2.0-P2.7
VCC
VSS
PORT 0 DRIVERS
PORT 2 DRIVERS
RAM ADDR. REGISTER
RAM
PORT 0 LATCH
PORT 2 LATCH
EPROM/ ROM
ACC
STACK POINTER
PROGRAM ADDR. REGISTER
B REGISTER
TMP2
TMP1
BUFFER
ALU
PCON
SCON
TMOD
TCON
TCON
TH0
TL0
TH1
TL1
TH2*
TL2*
RCAP2H*
RCAP2L*
SBUF
IE
IP
INTERRUPT, SERIAL PORT AND TIMER BLOCKS
PC INCREMENTER
PROGRAM COUNTER
PSW
PSEN
ALE
EA
RST
TIMING AND CONTROL
INSTRUCTION REGISTER
DPTR
PORT 1 LATCH
PORT 3 LATCH
OSC
PORT 1 DRIVERS
PORT 3 DRIVERS
XTAL 1
XTAL 2
P1.0-P1.7
P3.0-P3.7
*Applies to 8052/8032 only.

Results

- The first half of the design process was demonstrated by using a subset of instructions from the 8031
- The behavioral model is accurate for instructions implemented

Sources

Sarnoff Corporation

Charles H. Roth, Jr., Digital Systems Design Using VHDL

Phillip Southard

Ohio University / 5 March, 1998

EE 690 Reconfigurable Design