AN INSTRUCTIONAL PROCESSOR DESIGN USING VHDL AND AN FPGA

Ronald J. Hayne
Department of Electrical and Computer Engineering
The Citadel

Abstract

Most modern processors are too complex to be used as an introductory design example. Many digital design courses and texts use hardware description language models of processors, but they are often ad hoc. What is needed is a basic processor with sufficient complexity, that can be modified, programmed, and tested.

An instructional processor has been developed for use as a design example in an Advanced Digital Systems course. The architecture is separated into teachable subsets. The data path contains the registers and interconnecting busses, while the controller implements the fetch, decode, and execute sequences. The VHDL model of the system can be simulated to demonstrate operation of the processor.

The instructional processor is now in its second iteration with an updated controller design and a new microcontroller extension. Results from student homework assignments indicate that they are able to successfully design modifications to the processor and demonstrate their function via simulation. The project continues to achieve its goal as a valuable instructional tool.

Introduction

Teaching Advanced Digital Systems involves use of many design examples including counters, registers, arithmetic logic units, and memory. The design of a computer processor combines these components into an integrated digital system. Most modern commercial microprocessors are too complex to be used as an introductory example of processor design. Hardware description language models of these processors exist, but are often ad hoc and don't divide the architecture into teachable subsets [1,2]. Other microprocessor designs are part of a larger or follow-on course in computer architecture [3,4]. What is needed is a basic processor with sufficient complexity to illustrate major design elements, that can be modified, programmed, and tested.

An instructional processor has been developed for use as an integrated design example in an Advanced Digital Systems course at The Citadel. The architecture is separated into the data path and a sequential controller. The data path contains the memory, registers, arithmetic logic unit (ALU), and interconnecting busses, based on models developed throughout the course. The controller implements the fetch, decode, and execute sequences, using basic state machine design techniques. The entire system is modeled in VHDL and can be simulated to demonstrate operation of the processor. A field programmable gate array (FPGA) implementation also provides a functional hardware version of the processor.

Instruction Set Architecture

The instruction set architecture of the example processor has been designed to illustrate multiple operations and basic addressing modes. It is based on a three bus organization of a 16-bit data path with a four word register file (REGS). Key registers include: program counter (PC), instruction register (IR), memory data register (MDR), and memory address register (MAR). The basic processor has a 256 word by 16-bit memory (MEM) for storing both programs and data. The complete data path is shown in Figure 1 [5].

The initial implementation of the instructional processor includes opcodes for move (MOVE), add (ADD), and conditional branch (BGTZ), with the capability to modify or supplement these instructions. Access to operands from registers and memory includes provisions for direct (absolute), indirect, and immediate addressing modes. The resulting instruction format contains fields for the opcode (OP), operand source (SRC),
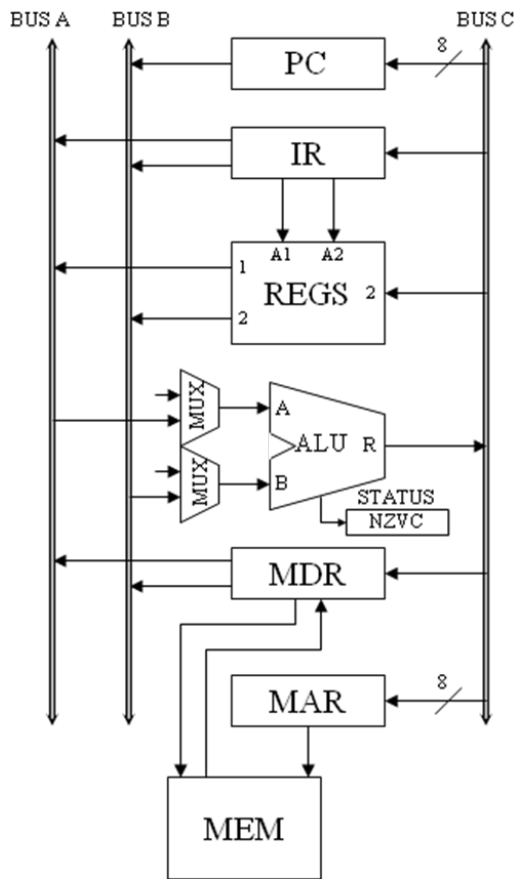
Figure 1: Data Path for Instructional Processor.

and operand destination (DST), as shown in Figure 2 [5].

The instruction set and addressing modes have been chosen to provide a basis to illustrate fundamental programming concepts. These include data transfer, counting, indexing, and looping. A simple assembly language program, shown in Figure 3, calculates the sum of an array of numbers to demonstrate these concepts. The example programs are translated into machine code and loaded into memory for testing.

Design of the instructional processor is taught in sections covering the instruction set architecture, followed by implementation of the data path, and finally the fetch, decode and execute sequences for the control unit. Each component is modeled in VHDL and functionally verified using ModelSim [6]. Student homework assignments then involve modification of the VHDL model to implement additional instructions. The upgraded processor is verified by the students via execution of their own test programs.

15 14 13	12 11 10 9	8 7 6	5 4 3 2 1 0	
OP	SRC	DST	VALUE	IR
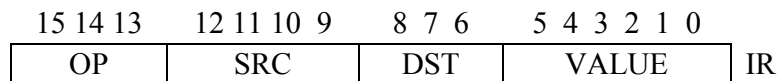
	Mode	REG #	Name	Syntax	Effective Address
SRC	00	00-11	Register Direct	Rn	EA = Rn
	01	00-11	Register Indirect	(Rn)	EA = [Rn]
	10	vv	Immediate	#Value	Operand = Value
	11	vv	Absolute	Value	EA = Value

	Mode	REG #	Name	Syntax	Effective Address
DST	0	00-11	Register Direct	Rn	EA = Rn
	1	vv	Absolute	Value	EA = Value

vv = Upper 2 bits of Value

OP	Fn	Assembly Language	Register Transfer Notation
000	MOVE	MOVE SRC,DST	DST ← SRC
001	ADD	ADD SRC,DST	DST ← SRC + DST
...			
110	BGTZ	BGTZ #Value	PC ← PC + Value (> 0)
111	HALT	HALT	Stop ← 1

Figure 2: Processor Instruction Format.

```
        MOVE  N,R1
        MOVE  #NUM1,R2
        MOVE  #0,R0
LOOP ADD   (R2),R0
        ADD   #1,R2
        ADD   #-1,R1
        BGTZ  LOOP
        MOVE  R0,SUM
        HALT
```

Figure 3: Assembly Language Program.

Control Unit Design

The control unit for the instructional processor is a hardwired controller which generates control signals using a control step counter and the various fields from the instruction register. The organization of the control unit is shown in Figure 4 [5]. The unit is designed as a finite state machine which implements the fetch, decode, and execute sequences for the instruction set architecture.

The first phase of the controller design includes the instruction fetch from memory into the instruction register. The sequence of steps is determined by the control step counter, T0 through T2. The required operations are written in register transfer notation (RTN) and translated into the appropriate control signals, as shown in Figure 5. The instruction is then decoded and an execute sequence is determined for each combination of opcode and addressing mode. A sample instruction, MOVE Rs,Rd, uses register direct addressing for the source (S0), and register direct addressing for the destination (D0). The RTN and control signals for this instruction are shown in Figure 6.

VHDL Model

The VHDL model for the instructional processor is developed in phases, with new capabilities added in each phase. Phase 1 includes the components of the data path, which have been developed throughout the course. These include
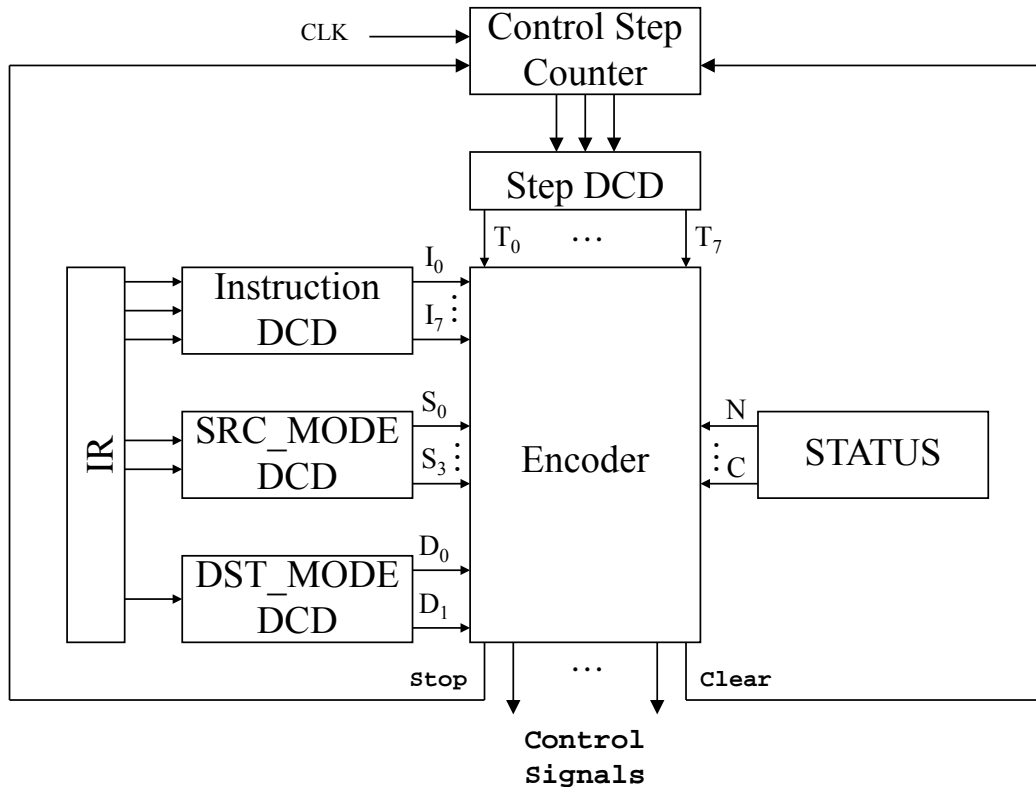

Figure 4: Control Unit Organization.

Step	RTN	Control Signals

T0	MAR ← PC, PC ← PC + 1	`BUS_B <= PC`
		`ALU_OP <= Pass_B`
		`Load_MAR <= '1'`
		`Inc_PC <= '1'`

T1	MDR ← MEM(MAR)	`MEM_Read <= '1'` `Load_MDR <= '1'`

T2	IR ← MDR	`BUS_B <= MDR`
		`ALU_OP <= Pass_B`
		`Load_IR <= '1'`

Figure 5: Instruction Fetch Sequence.

Step	RTN	Control Signals

T3	R(D) ← R(S)	`REGS_Read1 <= '1'`
		`ALU_OP <= Pass_A`
		`Load_STATUS <= '1'`
		`REGS_Write <= '1'`
		`Clear <= '1'`

Figure 6. Instruction Execute for `MOVE Rs,Rd`.

```
-- Data Path
REGS: REG4 port map (CLK, REGS_Read1, REGS_Read2, REGS_Write,
                     SRC_REG, DST_REG, BUS_C, BUS_A, BUS_B);
ALU: ALU16 port map (ALU_OP, ALU_A, BUS_B, BUS_C, N, Z, V, C);
MEM: MEM256 port map (CLK, MEM_Read, MEM_Write, MAR, MDR, MEM_Out);
```

Figure 7. VHDL Model for Data path.

the registers, ALU, and memory shown in Figure 1. A portion of the VHDL data path is shown in Figure 7.

Implementation of the controller follows the state machine design techniques used in many previous examples in the course. The VHDL model follows directly from the control signals derived in the previous section. A portion of the control unit implementing the instruction fetch sequence from Figure 5 is shown in Figure 8. The execute sequence for the `MOVE` instruction in Figure 6 is also shown in Figure 9.

Functional verification of the VHDL model is accomplished via simulation using sample assembly language programs translated into machine code. The program is loaded into memory as part of the VHDL model as shown in Figure 10. Simulation with ModelSim produces the waveform shown in Figure 11. This waveform demonstrates the correct fetch and execute sequence for the Phase 1 data transfer instructions.

Phase 2 of the design process includes development of execution sequences for the remaining instructions and addressing modes necessary to implement the sample assembly language program from Figure 3. This example program is sufficient to demonstrate fundamental programming concepts such as counting, indexing, and looping. The resulting processor is again verified via functional simulation using ModelSim. While the instructional processor is now capable of executing the sample program, a number of instructions and addressing modes are still incomplete and can serve as student homework exercises.

```
Control : process(STEP, IR,
                  STATUS)
begin
 case STEP is -- Fetch
  when T0 =>
   BUS_B <= X"00" & PC;
   ALU_OP <= Pass_B;
   Load_MAR <= '1';
   Inc_PC <= '1';
  when T1 =>
   MEM_Read <= '1';
   Load_MDR <= '1';
  when T2 =>
   BUS_B <= MDR;
   ALU_OP <= Pass_B;
   Load_IR <= '1';
```

Figure 8: VHDL Model for Control Unit Fetch.

```
   case OP is  -- Execute
    when MOVE =>
     case SRC_MODE is
      when S0 =>
       case DST_MODE is
        when D0 =>  -- MOVE Rs,Rd
         case STEP is
          when T3 =>
           REGS_Read1 <= '1';
           ALU_OP <= Pass_A;
           Load_STATUS <= '1';
           REGS_Write <= '1';
           Clear <= '1';
```

Figure 9: VHDL Model for Control Unit Execute.

```
architecture Behave of MEM256 is
   type RAM256 is array (0 to 255) of unsigned(15 downto 0);
   signal MEM256: RAM256 := (X"1043",  -- 00        MOVE #3,R1
                             X"0280",  -- 01        MOVE R1,R2
                             X"E000",  -- 02        HALT
                             others => X"0000");  -- 03-FF;
```
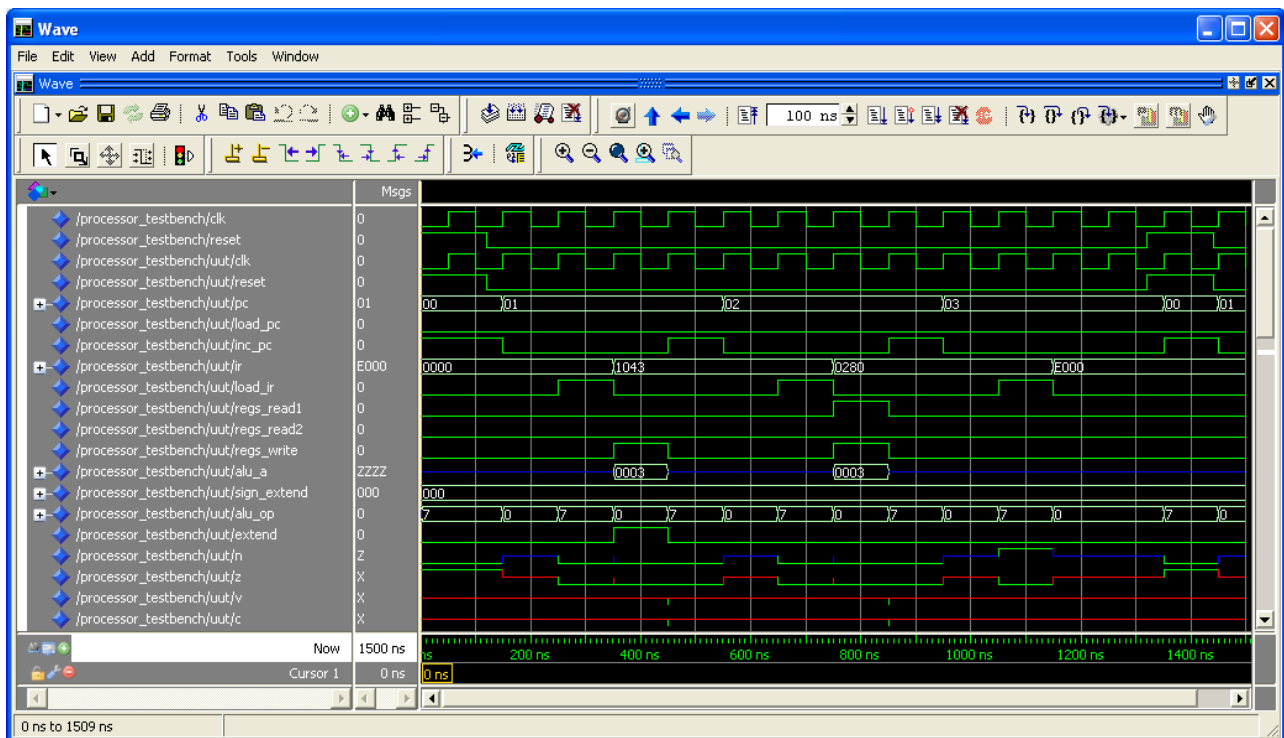
Figure 10: VHDL Model for Memory.

Figure 11: VHDL Simulation Waveform.

Student Homework Assignments

Student homework assignments involve expansion of the instructional processor by adding new addressing mode and opcode combinations. For example the assembly language instruction `MOVE (Rs),Rd` uses register indirect addressing for the source (S1) and register direct addressing for the destination (D0). Students must first determine the correct RTN sequence necessary to execute the instruction. Then, they translate the RTN sequence into the appropriate control signals for the data path.

The VHDL model for the execute sequence for the added instructions follows directly from the RTN and control signals developed by the students. By adding their VHDL sequence to the instructional processor model provided by the professor, the students then have a functional model of the processor which can be tested via simulation. Students encode their assembly language instructions into machine code and load them into the VHDL memory model. The entire model is then simulated with ModelSim to verify correct functioning of the students new instructions.

Students can also add new branch instructions to the processor. The `HALT` instruction from the Phase 1 model can be readily replaced by a branch always (`BRA #Value`) command. This provides an alternate method for stopping a program by creating an infinite loop. Unlike the data manipulation instructions, these relative branch instructions modify the program counter to alter the execution sequence of the program. Development of the RTN and control signals leads directly to a new execute sequence for the VHDL model.

Microcontroller Extension

Phase 3 of the processor design involves creation of a basic microcontroller by adding simple memory-mapped input and output to the processor. An 8-bit parallel input port (PORTA) and an 8-bit parallel output port (PORTB) are mapped to fixed locations in the memory address space. The VHDL model for the memory is easily modified to accommodate the new ports. Users can then interface with external devices using the existing data transfer instructions.

The microcontroller extension also includes addition of several new instructions: logic and (`AND`), logic invert (`INV`), and rotate left (`ROTL`). The expanded instruction set is now sufficient to create timing loops with the ability to produce various flashing and shifting patterns on output LEDs. User inputs via switches provide the ability to interact with the processor and select specific output sequences and timing.

The VHDL model of the microcontroller was synthesized using the Xilinx ISE design tools [7]. The resulting hardware model is mapped to a Xilinx Spartan 3e FPGA on a Digilent Basys board [8]. PORTA of the microcontroller is mapped to eight input switches on the board, while PORTB is mapped to eight LEDs. The final hardware implementation, shown in Figure 12, utilizes less than 25% of the FPGA resources, leaving plenty of room for further expansion.

Results and Conclusions

Implementation of the instructional processor is now in its second iteration with an updated controller design and the new microcontroller extension. The processor is used as a design example to replace the existing MIPS micro-processor in the current course text [1]. Because the new design uses examples from throughout the semester, it integrates directly into the flow of the course. Development of the model in phases allows separate coverage of the data path and the sequential controller. Design of the control unit is further subdivided into the fetch sequence and execute sequences for instructions as they are added to the processor.

Realization of only a subset of the processor instructions provides sufficient capabilities to demonstrate fundamental programming concepts such as data transfer, counting, indexing, and looping. Additional instructions are implemented as student homework assignments allowing direct application of the design techniques taught in class. Student feedback is very positive that the processor illustrates basic design concepts without unnecessary complexity. Results from homework
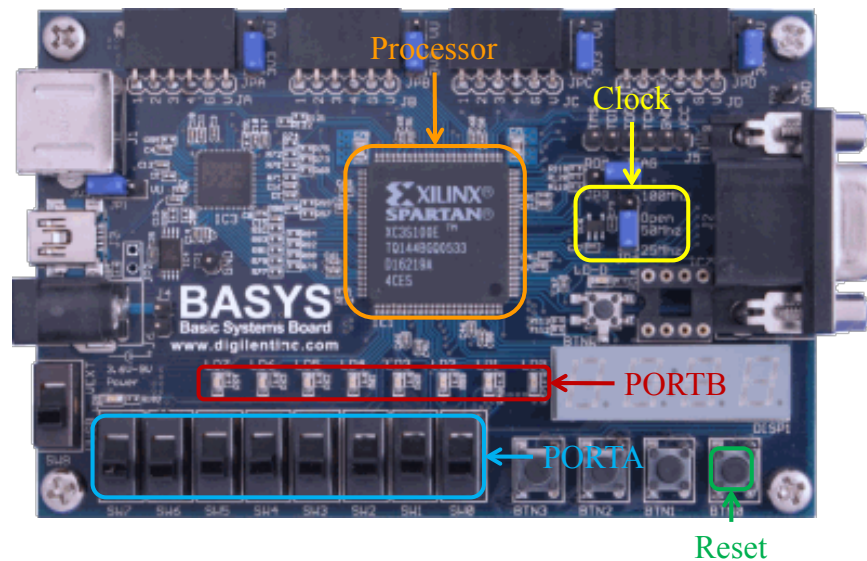
Figure 12: FPGA Implementation of Microcontroller.

submissions indicate that the students are able to successfully design modifications to the processor and demonstrate their functionality via simulation.

Addition of the new microcontroller extension provides the final link between the processor model and actual hardware. Simulation exercises are sufficient for functional demonstrations and homework assignments, but the FPGA implementation puts processor hardware in the hands of the students. Inclusion of memory-mapped input and output allows integration with external devices like switches and LEDs providing an interactive presentation of the microcontroller system. The complete project, instruction set architecture, VHDL model, software and hardware, continues to achieve its goal as a valuable instructional tool.

Bibliography

1. C. H. Roth, and L. K. John, *Digital Systems Design Using VHDL,* 2nd ed., Thompson, Toronto, Canada, 2008.

2. S. Lee, *Advanced Digital Logic Design: Using VHDL, State Machines, and Synthesis for FPGAs*, Thompson, Toronto, Canada, 2007.

3. D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture,* Morgan Kaufmann, San Francisco, CA, 2007.

4. E. O. Hwang, *Digital Logic and Microprocessor Design with VHDL*, Thompson, Toronto, Canada, 2006.

5. R. J. Hayne, "VHDL Projects to Reinforce Computer Architecture Classroom Instruction," *Computers in Education Journal*, Vol. XVIII No. 2, April - June 2008.

6. ModelSim PE Student Edition, Mentor Graphics Corp., 2010.

7. Xilinx ISE 10.1i Software Manuals, Xilinx, Inc., 2008.

8. Digilent Basys Board Reference Manual, Digilent Inc., 2007.

Biographical Information

Ronald J. Hayne is an Assistant Professor in the Department of Electrical and Computer Engineering at The Citadel. He received his B.S. in Computer Science from the United States Military Academy in 1980, his M.S. in Electrical Engineering from the University of Arizona in 1987, and his Ph.D. in Electrical Engineering from the University of Virginia in 1999. Dr. Hayne's professional areas of interest include digital systems design and hardware description languages. He is a retired Army Colonel with experience in academics and Defense Laboratories.