TEACHING DATA PATH AND CONTROL LOGIC USING MICROCODED ARCHITECTURE

Jonathan Hill
Electrical and Computer Engineering Department
University of Hartford

Abstract

I regularly teach a course in VHDL and typically find that students struggle with the data path and controller (DPC) paradigm. The notion of the paradigm is that many systems, including microprocessors and peripheral devices, in theory or in practice are constructed of a data path that essentially performs the "work" of the system, along with a state machine that controls the behavior of the data path. This paradigm is essentially the register-transfer system level, which helps students to better understand many types of devices. The notion of the DPC paradigm is relevant not only to VHDL but to other hardware description languages including Verilog as well. Students generally find that coming to terms with such an advanced topic takes time and is at least as significant as a step forward as grasping the concept of state machines in logic circuits.

I have found that a data path constructed with simple components combined with a microcoded state machine aids students in overcoming the hurdle of mastering the DPC paradigm. The discussion of microcoding here is limited to that needed to complete the project. It is not my intention to develop an interpreter to construct a conventional microprocessor. Rather, as a programming language in its own right, microcoding provides a means to present the DPC paradigm in a convincing way.

An amazing thing about the paradigm is that a simple data path constructed with an arithmetic logic unit (ALU), multiplexers, and a few registers, along with a controller, can provide such varied and complex behavior. Furthermore, with such flexibility in design it is relatively easy to make changes for the project to be different each semester. The first version produced the Fibonacci sequence, and a more recent version incorporates external memory for the Sieve of Eratosthenes, producing prime numbers. Another version is the traditional Hello World program. Such a system also provides a context in which memory systems and peripheral devices can be presented and discussed.

This project has proven to be educationally valuable and is regularly assigned in my VHDL course. This paper outlines the DPC paradigm, introduces the microcoded data path (MDP) project, and outlines the how the MDP can be used as a context for advanced topics such as memory systems and peripheral devices, touches on some exercises, and presents student feedback.

Introduction

Every fall semester I teach an introductory course in VHDL to seniors and graduate students. The seniors often make use of their new skills in performing their senior projects during the following spring semester. Our graduate students generally take VHDL early in their course of study and can apply their skills in research. The course typically involves four projects that build upon one another. Students work as individuals and have two weeks to implement a given project that is demonstrated in class and one week to write the corresponding project report.

Our students use Xilinx ISE software[1] to simulate and implement their projects along with an inexpensive Field Programmable Gate Array (FPGA) development board, such as the Spartan-3 Starter Board[2] outlined in Figure 1. Of the features, we use the buttons, switches, LEDs, seven-segment displays, and the RS232-

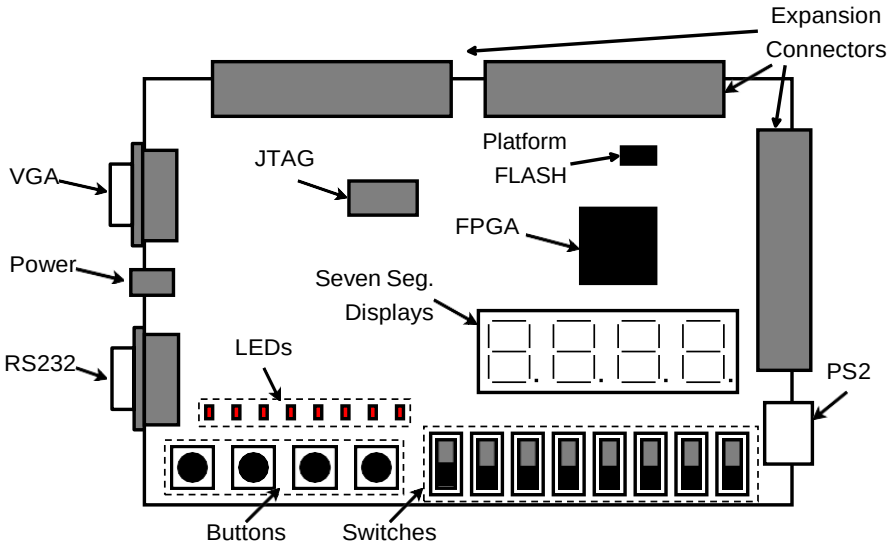

Figure 1: Spartan-3 Starter Board.

style serial communications port. There is a 50MHz clock oscillator and two 512 kByte memory devices, which are not shown here. Two different JTAG connectors are provided to configure the board. The platform FLASH is used to make a design permanent. The board also has a VGA video port, a PS2 keyboard and mouse port, as well as power and expansion connectors.

Students in the course are introduced to the general structure of an FPGA, in which a two-dimensional array of regular logic blocks along with an interconnect resource forms what is called the *FPGA fabric*. Each logic block can implement combinational or registered logic or a RAM or ROM type device. The Xilinx ISE software itself determines how each such logic block in the FPGA fabric is to be actually used. For the projects considered, other than the clock and the actual input and output devices, the designs are implemented entirely in the FPGA.

The third project traditionally involves the data path and controller (DPC) paradigm and typically is the most challenging and, as such, is a significant milestone. Other projects involving the DPC paradigm that are not necessarily microcoded include an asynchronous communications device or UART, a multiplier for signed integers, and a stopwatch. The microcoded data path (MDP) project is another means to introduce the DPC paradigm. In a nutshell, the MDP uses simple components such as registers, multiplexers, and an arithmetic logic unit (ALU) along with a state machine to implement a rudimentary processor. Tanenbaum[3] as well Mano and Kime[4] each provide a more general outline of MDP structures.

I use class examples as well as homework and other projects to introduce students to the building blocks used to construct an MDP. Despite this practical experience, in selecting the MDP as the third project, students generally find it to be challenging. To make the experience even more challenging, the project can include an external memory system and a peripheral device such as a serial communications transmitter. Example algorithms using the MDP include those that

- Produce the Fibonacci sequence
- Produce prime numbers with the Sieve of Eratosthenes
- Perform multiplication by repeated addition
- Perform division by repeated subtraction
- Behave as a repeating counter
- Play a simple game involving input and output

- Produce the serial data message “Hello World!”

The MDP is presented to students using a somewhat deductive approach, which is generally helpful when first introducing such an idea. Students are given a block diagram that uses components they already know. In implementing the given structure, students must investigate, debug, and demonstrate their own work. Once the system is complete, the MPD provides students an inductive approach to further develop their knowledge. Students are asked design and analysis type questions to expand their knowledge and also write their own microcode programs. Eventually, students propose their own MDP and DPC structures.

The Data Path and Controller Paradigm

Figure 2 is a general overview of the DPC paradigm. The data path is essentially where the work of the system is performed. The controller is a state machine; it receives status information and provides control information, directing the data path to provide the desired behavior. The external memory system is optional. It is the interplay between the controller and data path that makes such a system so interesting. Based solely on the control information provided, the behavior of the data path can be made to be like that of a very different system.

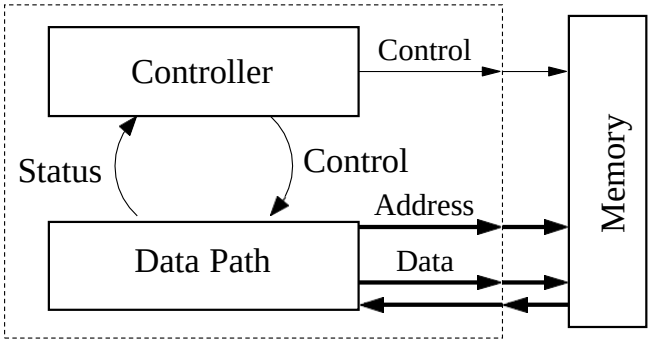

Figure 2: DPC Paradigm Overview.

In the MDP project we regard the external memory system as optional and agree that the controller itself contains a program expressed in microcode. The discussion of microcoding is limited to that needed to complete the project. It is not my intention to construct a conventional microprocessor or machine code interpreter. Rather, my concern is presenting the DPC paradigm in a convincing way. As a programming language in its own right, microcoding provides a clear means to make the necessary points.

The MDP Project Data Path

In preparing the project, the choice of data path components is based on the algorithm selected for the semester. Figure 3 is the data path used to implement the prime numbers sieve as well as the Hello World program. When used with external components, this data path is regarded as more challenging. Prior to assigning the project, all the components are introduced to students in examples, homework, and other projects. The MDP project calls on students to make use of the structural expressiveness in VHDL in which each component is instantiated and interconnected with other components using signals.

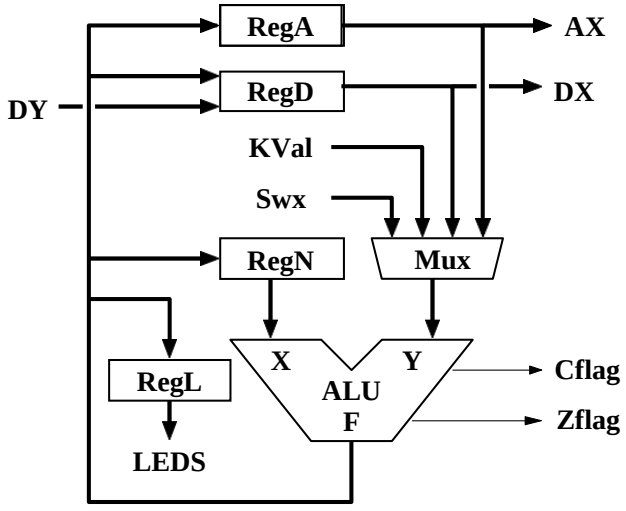

Figure 3: Example Data Path.

The data path in Figure 3 is constructed with registers (Reg), multiplexers (Mux), and a simple arithmetic logic unit (ALU). Registers A, D, and L are all identical synchronous registers; when enabled, each performs a parallel load at a rising clock edge. Register D is similar in behavior; however, it selects from one of two inputs. The N register produces the X input to the ALU. This feature is actually a significant constraint that we discuss. The multiplexer provides the second ALU input. The Swx signal is input from switches, and the L register provides output to LEDs. The signal KVal is a value provided by the controller.

The following tables summarize the behavior of register D, the multiplexer, and the ALU. In performing addition or subtraction, the Cflag signal indicates either a carry out or a borrow condition. The Zflag signal indicates when then ALU output is zero. The "and" operation is bitwise, between the corresponding bits in X and Y.

A data path like that in Figure 3 provides students an opportunity to explore the difference between registered signals and combinational logic signals. The carry/borrow flag (CFlag) and the zero-result flag (Zflag) are produced by combinational logic and thus correspond to the given moment. Signals such as AX and DX, however, are produced by registers and can only change following an active clock edge and hence appear to be delayed. In forming the difference between two equal values, the Zflag signal will immediately be asserted; however, the actual zero value in F will not be seen in a register until after the following active clock edge.

Microcode-Based Controller

Figure 4 shows the microcode structure controlling the data path in Figure 3. The microstore is a read-only memory device that is addressed by the parallel loadable binary up-counter. Based on the signal Test, a multiplexer selects one of four signals for the counter Load signal, providing the various branch conditions listed in Table 4. The signal name "NATT" is the acronym for "*next address if the test result is true*" and serves as the target of a branch action, so that a branch is performed by loading the NATT value into the address counter.

The signals EnA, EnN, and EnL enable a parallel load for the A, N, and L registers, respectively. The signal EnD enables the D register according toTable 1. Likewise, SelMux and SelALU control the multiplexer and ALU according to Table 2 and Table 3, respectively. The KVal signal provided to the data path is constant for one clock cycle. The Wr signal indicates a memory write action.

Table 1: Register D

EnD	Action
00	Store
01	Load F
10	Load DY
11	Load F

Table 2: Multiplexer Y

SelMux	Selection
00	Y = RegA
01	Y = RegD
10	Y = KVal
11	Y = Swx

Table 3: ALU Behavior

SelALU	Operation	Cflag	Zflag
00	F = X + Y	Carry	F = 0?
01	F = X – Y	Borrow	F = 0?
10	F = Y	Low	F = 0?
11	Z = X and Y	Low	F = 0?

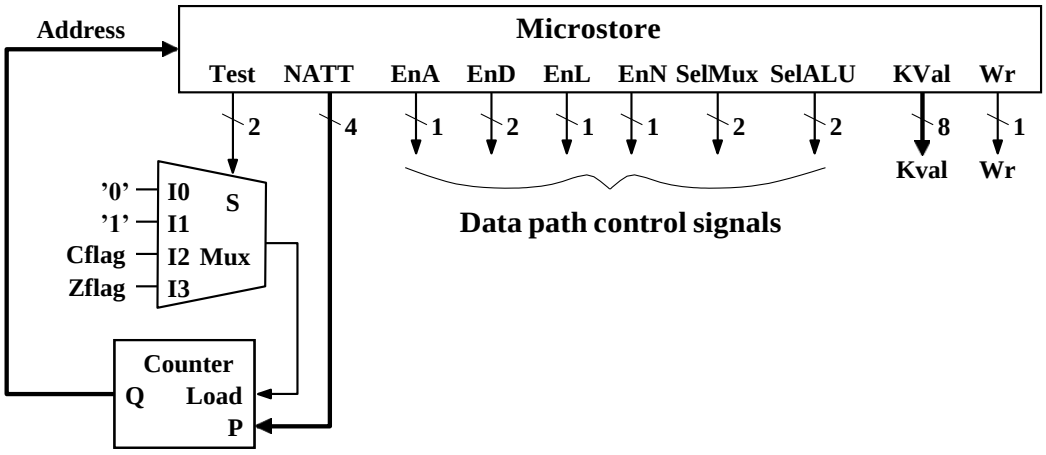

Figure 4: Microcode-Based Controller.

Table 4: Microcode Branch Conditions

Test	Branch Condition
00	Do not branch
01	Branch always
10	Branch if Cflag is high
11	Branch is Zflag is high

To see how this controller is a state machine, consider that the address counter value is the actual state value. The corresponding Test and NATT field values and the flag values determine the next state value. The remaining microstore field values are the controller output. Given that the output is a function of only the current state, the controller is a Moore type[5] state machine.

Symbolic Microcode

The contents of the microstore are dense and inconvenient to read. For this reason, we generally use a language called symbolic microcode to discuss microcode programs. I currently do not have tools for automatically converting between symbolic and actual microcode. However, the act of converting between these forms is a good exercise for students, requiring an understanding of the interplay between the controller and data path. We use the Xilinx simulator with the MDP project as a first step to verify a microcode program.

Symbolic microcode is organized into lines or statements, where each statement describes all the actions that will be performed in one clock cycle by the data path. Compiling or assembling microcode involves choosing microstore field values that produce the desired actions. Given that some actions can be performed several different ways, it is possible for a given symbolic microcode statement to be expressed several ways in microcode. Conversely, disassembling microcode involves describing the actions produced by actual microcode.

A symbolic statement includes an optional data action part and an optional branching part. The symbol "<−" implies the loading of a register and the validity of each such statement depends on what the actual hardware is capable of performing. With respect to Figure 3 the data actions can involve the A, D, N, and L registers though the L register cannot be used on the right side of an assignment and in operations involving two registers, N must be one of the registers. Consider the following, where "$" indicates hexadecimal notation:

```
N,L <- $01
```

Figure 5 is highlighted to show how the given action can be performed, in which case the following conditions must be true.

- The value of KVal from the controller is `$01`
- The multiplexer passes KVal to the ALU
- The ALU passes its Y input to output F
- The N and L registers perform loads

To complete the microcode statement, fill in microstore fields as necessary so that the given behavior will be produced. We find that

- EnL = 1, EnN = 1, and that
- SelMux = 10, SelAlu = 10

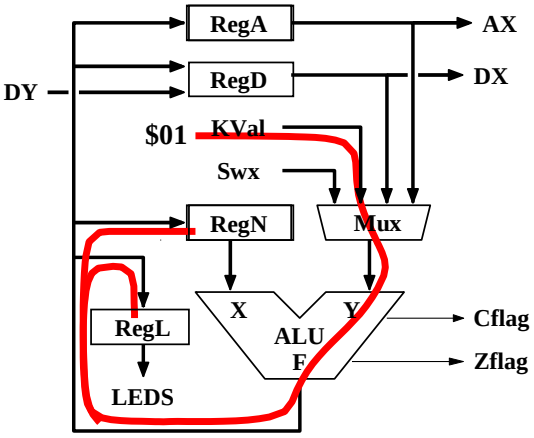

Figure 5: Data Path Activity Highlighted.

Since the ALU can pass the Y input value, passing registers A or D, and the KVal signal is simple. To pass the N register value, however, the ALU can be used to add zero to the N register value so that the following are equivalent statements:

- `A <- N`
- `A <- N + $00`

Sometimes it is useful to specify an ALU operation without assigning the result to any register. The following performs a branch if the N register contains the value four.

- `N - $04, if Zflag goto somewhere`

In the example program in Figure 6, the semicolon indicates the start of a comment, and the keyword ORG sets the address in microcode. This program implements a counter that repeatedly counts from one to four. The leftmost value is the address of each microcode statement. The language format is similar to assembly language in that each label is a name for an actual address value.

In compiling the symbolic code, the microstore contents in Table 5 is produced. Values without a prefix symbol are assumed to be in binary. Note that a value must be assigned to each and every field. In cases where a value does not matter, the arbitrary value zero is used here.

```
   ; A repeating counter, one to four
          ORG $0
$0 Start:  N,L <- $01                    ; initial val
$1 Loop:   N,L <- N + $01                ; increment
$2         N - $04, if Zflag goto Start  ; start over?
$3         goto Loop                     ; repeat loop
```

Figure 6: Repeating Counter Example Program.

Table 5: Mircrostore Contents for Repeating Counter Program.

Address	Test	NATT	EnA	EnD	EnL	EnN	SelMux	SelAlu	KVal	Wr
$0	00	$0	0	00	1	1	10	10	$01	0
$1	00	$0	0	00	1	1	10	00	$01	0
$2	11	$0	0	00	0	0	10	01	$04	0
$3	01	$1	0	00	0	0	00	00	$00	0

The Event Table

An event table is used to list the cycle-by-cycle behavior of a given program. From a pedagogical view, producing or interpreting such listings involves the close examination of each system component. The following notes apply:

- "T" is the clock period
- "+" is a brief moment after a rising clock edge allowing the system to settle
- A blank entry in the event table means no change from the previous clock cycle
- A value loaded into a register is entered into the table, even if the value is the same

The following event table summarizes the behavior of the repeating counter program given above. The reset is asynchronous and clears the address counter and all the registers. Note that with the blank entries a pattern appears. Also note that, unlike registers, the Zflag signal shown here is produced by combinational logic and represent conditions currently in the ALU. At time 8T+ the address refers to the conditional branch instruction, and the Zflag is immediately affected with no delay due to the clock. The program restarts at 9T+ but with N containing $04.

Table 6: Event Table for Repeating Counter Program

Time	Reset	Address	N	F	Zflag
0T	1	$0	$00	$01	0
0.5T	0				
1T+		$1	$01	$02	
2T+		$2	$02	$FE	
3T+		$3		$02	
4T+		$1		$03	
5T+		$2	$03	$FF	
6T+		$3		$03	
7T+		$1		$04	
8T+		$2	$04	$00	1
9T+		$0		$01	0
10T+		$1	$01	$02	
11T+	Program continues the pattern above				

```
; Fibonacci sequence
        ORG $0
Start:  A,L <- $00                          ; older val in A
        D,N <- $01                          ; newer val in N
Loop:   L,N <- D                            ; display newer
        D <- A+N, if CFlag goto Start       ; restart?
        A <- N, goto Loop                   ; repeat
```

Figure 7: Fibonacci Sequence Program.

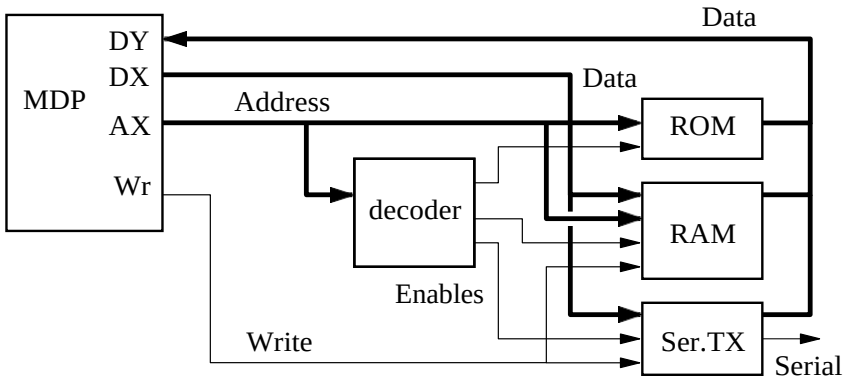

Figure 8: System with Memory and One Device.

The Fibonacci Sequence

The MDP project was first inspired by the well known algorithm used to produce the Fibonacci sequence. By this definition the first two numbers in the sequence are zero and one, and each of the following numbers is the sum of the previous two numbers. Hence, the sequence starts with the values 0, 1, 1, 2, 3, 5, 8, and continues. The program in Figure 7 displays in order all the numbers in the Fibonacci sequence that are less than 255. Producing the microstore contents and execution history are left as exercises for students, on the project web page[6].

Using Memory and Peripheral Devices

Memory systems and peripheral devices are normally used with conventional microprocessors. Unfortunately for such a VHDL course, presenting a microprocessor along with memory systems and peripherals is too topically broad and is worthy of a course in itself. The microcoded data path, however, provides a useful context in which memory systems and peripheral devices can be presented and discussed in this course.

The following outlines how the memory system in Figure 8 is used with the data path and controller in Figure 3 and Figure 4, respectively. Two unidirectional data buses are used for memory accesses. In the data path, register A conveys an address through the AX bus. Register D conveys data sent to and returned from the memory system, using the DX and DY buses, respectively. The microstore Wr field provides the write enable for the memory system.

In the memory system a decoder uses the address provided to enable a given device, causing the device to appear in a particular

region of the memory map. The devices here include a ROM, a synchronous-write, asynchronous-read (SWAR) RAM, and a serial communications data transmitter (Ser.TX).

In symbolic microcode, in performing a read or write memory action, the symbol `M(A)` refers to the memory system. The symbol implies that the A register provides the address and that memory is like an array structure. To read from memory, the D register performs a load from the DY bus and is expressed as:

```
D <- M(A)
```

Memory actions do not involve the rest of the data path so that concurrent actions involving the ALU may also be performed in the same clock cycle. To perform a write to memory, the memory write enable is asserted. The corresponding statement is as follows:

```
M(A) <- D
```

The following example program demonstrates how to use the RAM memory device. The program writes the input switch value to memory, reads the value back, and then writes the value to the output LEDs.

```
; Demonstrate how to use the RAM device
       ORG $0
Start: A <- $C0     ; An address
       D <- Swx     ; Some data
       M(A) <- D    ; Perform write
       D <- $00     ; Proof of no tricks
       D <- M(A)    ; Perform read
```

Figure 9: Example Demonstrating the Use of RAM

To implement the Hello World program, the ROM in Figure 8 contains the ASCII code values representing the message that is transmitted. The RAM is used as a scratch pad. An input switch is moved from low to high to signal when to send the message, and the LEDs display the number of times the message is transmitted. The message is transmitted at 9600 Baud with eight data bits, no parity, and one stop bit. Further details of the Hello World project will be posted on the project web page[6].

Related Exercises

In using an MDP in a course, it is suggested that the project topics be included in other assignments including homework. It is important that students are already familiar with multiplexers, registers, and an ALU. It is also important that students are comfortable with the state machine concept. The MDP project uses all these ideas and familiar components in an exciting new way.

To become familiar with the data path, students can make copies of the block diagram and for each microcode statement, as in Figure 5, shade the path that data follows from source to destination. Based on their understanding of the data path, students can next be asked if a given microcode statement is valid, that is, whether or not the data path is capable of performing the given action.

In addition to hand assembling microcode, it is worthwhile to disassemble microstore contents to the corresponding symbolic microcode. To become familiar with how the controller is actually a state machine, students can draw the state diagram for a given program. Producing an event table is worthwhile. Students can be asked to explain a particular entry in an event table or to identify common errors made in producing an event table.

With some mastery of the MDP project, students can be asked to write small programs in symbolic microcode. Most microcode programs are short. With four address bits, the largest program can have at most 16 statements. Students can then be asked to assess the strengths and weaknesses of a given MDP structure and to propose their own structure.

The recent inclusion of a memory system and a peripheral device with the MDP project opens many new opportunities. It is difficult to

present an example memory system or peripheral device without first introducing the concept of what a bus is and how data is communicated through a bus to attached devices. The MDP can be useful in such a VHDL course in teaching topics involving busses, memory systems, and peripheral devices.

Student Feedback

A survey was recently distributed and a modest number of students replied. The following are the questions and a summary of the survey results. In a nutshell, students mostly feel that the MDP project is a satisfying challenge, that they learned much, that the project is worthwhile, it should be assigned as project 3 and that project 4 can go further to provide a design opportunity. These results agree with the discussions that I have had with students.

1. In what year did you take the VHDL course?

Except for 2002, at least one student replied for each year. The responses for 2009 are the entire course enrollment for that year. The largest enrollment ever for the course was 22 students.

2. Which course project was the MDP assigned as?
3. Which project do you think the MDP should be assigned as?

I was not able to correlate these two questions; however, a 57% majority reported the MDP as project 3, and a larger 71% majority feel that the MDP should be assigned as the third project.

4. How challenging was the MDP project to perform?
5. How much did you learn in performing the MDP project?
6. How worthwhile was the experience in performing the MDP project?

Of the responses, 73.3% felt that the project was challenging or difficult, 80% felt that they learned quite a bit or very much, and 86.6% felt that the project was mostly or very worthwhile.

Table 7: Responses for the Year VHDL was Taken

2001	2002	2003	2004	2005	2006	2007	2008	2009
1	0	1	2	1	1	1	3	5

Table 8: (Q4)Perceived Challenge of MDP Project

Easy	Not simple	Moderate	Challenging	Difficult
6.7% (1)	6.7% (1)	13.2% (2)	53.3% (8)	20% (3)

Table 9: (Q5)Perceived Learning with MDP Project

Very Little	Not so much	Typical	Quite a bit	Very much
6.7% (1)	6.7% (1)	6.7% (1)	53.3% (8)	26.7% (4)

Table 10: (Q6) Perceived Worth of MDP Project

Not worthwhile	A little	Partly	Mostly	Very worthwhile
0% (0)	13.2% (2)	0% (0)	33.3% (5)	53.3% (8)

7. When you performed the MDP project, did it include an external memory system?

This question was to gauge the actual difficulty of the project assigned. It is well known that a memory system was used only in 2008 and 2009; unfortunately, the responses did not correlate with those from question 1.

8. Please provide comments about how you think the MDP project can be improved.
9. Please make any other comments about the MDP project.

There was a range of replies, but the general consensus is that the MDP is good as project 3 and that project 4 should provide a needed design element for the course. One student also remarked that memory systems can be expanded as a topic in the course.

- "…there was not enough [actual design involved] in this project."
- "…make the MDP project #3 and make #4 a project that builds on top of the structure in #3"
- "…a thorough review of memory systems might be in order. That was my major hang-up."

In summary, the feedback agrees with my experience in talking with students, that the project is a satisfying challenge. Students mostly feel that they learned much and that the project is worthwhile. They also feel that the MDP should be assigned as project 3 but that project 4 should take the ideas further.

Summary and Conclusion

The microcoded data path (MDP) project has proven to be of educational value in the way it introduces students to the concept of the data path and controller (DPC) paradigm. The project uses simple building blocks to construct a rudimentary processor programmed in microcode that is capable of providing a range of behaviors. The project is also motivational for students, and students have asked for additional material to build upon the MDP project.

References

1. Xilinx ISE software download, http://www.xilinx.com/
2. Spartan-3 Starter Board, http://www digilentinc.com/
3. Tanenbaum, Structured Computer Organization, fifth edition, 2006, Pearson Prentice Hall
4. M. Mano and C. Kime, Logic and Computer Design Fundamentals, fourth edition, 2007, Pearson Prentice Hall
5. M. Mano and M. Ciletti, Digital Design, fourth edition, 2007, Pearson Prentice Hall
6. Jonathan Hill, project page, http://uhaweb.hartford.edu/jmhill/suppnotes/MicroCodeDP/index.htm

Biographical Information

Jonathan Hill is an Associate Professor in Electrical and Computer Engineering at the University of Hartford in Connecticut with Ph.D. and M.S.E.E. from Worcester Polytechnic Institute of Worcester, MA. He was previously a project engineer at Digital Corp. He instructs graduate and undergraduate computer engineering computer courses, directs graduate research, and performs research involving embedded microprocessor based systems. His specific projects involve digital communications, signal processing, and intelligent instrumentation