AC 2011-1016: ADVANCE FEATURES OF HARDWARE DESCRIPTION LANGUAGE (VHDL) FOR UNDERGRADUATE STUDENTS

Alireza Kavianpour, DeVry University, Pomona

Dr. Alireza Kavianpour received his PH.D. Degree from University of Southern California (USC). He is currently Senior Professor at DeVry University, Pomona, CA. Dr. Kavianpour is the author and co-author of over forty technical papers all published in IEEE Journals or referred conferences. Before joining DeVry University he was a researcher at the University of California, Irvine and consultant at Qualcom Inc. His main interests are in the areas of embedded systems and computer architecture.

Advance Features of Hardware Description Language (VHDL) for Undergraduate Students

This paper describes the use of Very High Speed Integrated Circuit Hardware Description Language (VHDL) in a computer architecture course. VHDL is a programming language that allows an individual to define how a piece of hardware behaves. This language was developed first by US military and became IEEE standard in 1987. It was updated in 1993 and is known today as "IEEE standard 1076 1993". The complexity of ASIC and FPGA designs has caused an increase in the use of hardware description languages such as VHDL. As a result, students must learn the advance features of this language. Companies like Xilinx, Altera, and Cadence have designed proper hardware interface for the use of this language. Recently, most chip manufacturers start on VHDL (or Verilog) before the company goes to actual production of a design. There are two aspects to modeling hardware that any hardware description language facilitates: true abstract behavior and hardware structure. VHDL has three parts: Library, Entity, and Architecture.

Despite reviewing many undergraduate digital books, there is no reference on how to teach different processes and cycles in a computer architecture/organization course by VHDL. In this paper, the author explains how VHDL could be used to teach different topics such as: fetch cycle, decode cycle, execution cycle, control unit, arithmetic logic unit, read/write memory, and input/output operations. Most of the embedded applications we are familiar with use a MIPS processor. Therefore, as a teaching tool in a computer architecture course, MIPS processor and VHDL could be used for teaching different topics.

1- Introduction

There are now two industry standard for hardware description languages: VHDL and Verilog. The complexity of ASIC and FPGA designs has caused an increase in the number of specialist design consultants with specific tools and an increase in libraries of macro and mega cells written in either VHDL or Verilog. VHDL became IEEE standard 1076 in 1987. It was updated in 1993 and is known today as "IEEE standard 1076 1993". VHDL is a concise and verbose language; its roots are based on Ada. Although an existing programmer of both C and Ada may find the mix of constructs somewhat confusing at first, the VHDL model follows the same principle as defined for the C model. When reading integer values from a file, those numerical values must be read and assigned to a variable; they can neither be read nor assigned to a signal.

2- VHDL Description

A VHDL[1,4,5] program has three parts: Library, Entity, and Architecture. Library part defines components. Entity part declares the inputs and outputs. Architecture part defines the relation between inputs and outputs. VHDL library is where the VHDL compiler stores information about a design. Example of library declaration is:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE STD.TEXTIO.ALL;
```

Example of ENTITY part for a three inputs OR gate with inputs A,B, C and output D is:

```
ENTITY OR  IS
PORT(A, B, C :IN STD_LOGIC;
        D          :OUT STD_LOGIC);
END OR;
```

Example of Architecture part for a three inputs OR gate with inputs A, B, C, and output D is:

```
ARCHITECTURE OR OF OR IS
BEGIN
D <= A OR B OR C;
END OR;
```

Following is an example of 32KB SRAM (Static Random Access Memory) with active low read, write, and chip select inputs. Memory has 15 address lines and 8 data lines.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY sram is
  PORT (
     nCS: in STD_LOGIC;
     nOE: in STD_LOGIC;
     nWE: in STD_LOGIC;
     addr: in STD_LOGIC_VECTOR (14 downto 0);
     data: inout STD_LOGIC_VECTOR (7 downto 0)  );
END sram;
ARCHITECTURE  sram of sram is
FUNCTION addr2index (addr: in STD_LOGIC_VECTOR (14 downto 0)) return INTEGER is
VARIABLE index: INTEGER;
VARIABLE I: INTEGER;
BEGIN
          index := 0;
                    for I in 14 downto 0 loop
                    index := index * 2;
                    if (addr(I) = '1') then
                              index := index + 1;
                    END IF;
          END loop;
                    RETURN index;
END addr2index;
type MEMORY is array (0 to 32767) of STD_LOGIC_VECTOR (7 downto 0);
```

```
SIGNAL mem: MEMORY;
BEGIN
        PROCESS (nCS, nOE, nWE, addr)
        VARIABLE I: INTEGER;
        BEGIN
                IF(nCS = '1') then
                        data <= "ZZZZZZZZ" after 100 ns;
                ELSIF (nOE = '0') then
                        I := addr2index(addr);
                        data <= mem(I) after 100 ns;
                        ELSIF (nWE'event and (nWE = '1')) then
                        I := addr2index(addr);
                        mem(I) <= data;
                END IF;
        END PROCESS;
END sram;
```

3- MIPS Architecture

MIPS [1,2,6] is an acronym for a microprocessor architecture called Microprocessor without Interlocked Pipeline Stages. The MIPS processor is a RISC (Reduced Instruction Set Computer) processor designed in 1981by a company called MIPS Technology. Most of the embedded applications we are familiar with use a MIPS processor. The MIPS processor is similar to (and therefore a good example of) many other available RISC processors in the market. Figure 1 display MIPS architecture. MIPS processor operates with the following types of instructions: Arithmetic, logical, data transfer (or shift), and branch (both conditional and unconditional).

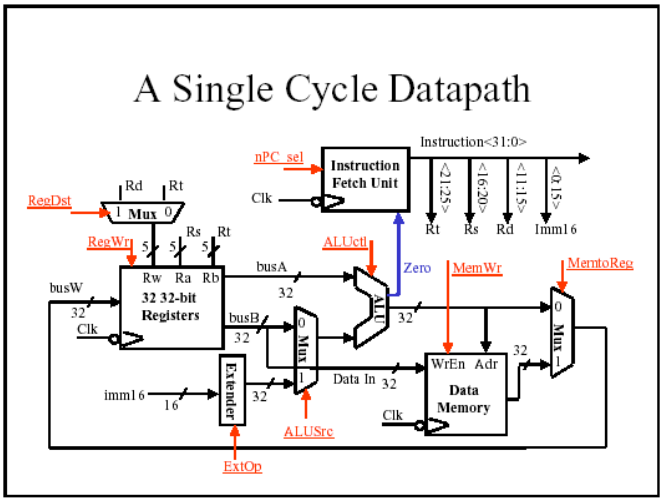

Figure 1: MIPS architecture.

Following are sample formats for these instructions.

Example of Arithmetic Instruction Format

Instruction	Meaning	Comment
Add $1,$2,$3	$1 = $2 +$3	3 operand
sub $1,$2,$3	$1 = $2 -$3	3 operand
Addi $1,$2,10	$1 = $2 +10	3 operand
addu $1,$2,$3	$1 = $2 +$3	3 operand
Addiu $1,$2,10	$1 = $2 +10+cons.	3 operand
subu $1,$2,$3	$1 = $2 -$3	3 operand

Instruction	Example	Meaning	Comments
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
Mult. unsigned	multu$2,$3	Hi, Lo = $2 x $3	64-bit unsigned product
divide	div $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Lo = quotient, Hi = rem.
divide unsigned	divu $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Unsigned quo. & rem.
Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
Move from Lo	mflo $1	$1 = Lo	Used to get copy of Lo

Example of Logical Instruction Format

Instruction	Meaning	Comment
and 1,$2,$3	$1 = $2 & $3	3 reg. operands
or $1,$2,$3	$1 = $2 | $3	3 reg. perands
xor $1,$2,$3	$1 = $2 $3	3 reg. operands
nor $1,$2,$3	$1 = ~($2 |$3)	3 reg. Operands
andi $1,$2,10	$1 = $2 & 10	reg, constant
ori $1,$2,10	$1 = $2 | 10	reg, constant
xori $1, $2,10	$1 = ~$2 &~10	reg, constant

Example of Data Transfer (Shift) Instruction Format

Instruction	Meaning	Comment
sll $1,$2,10	$1 = $2 << 10	Shift left by constant
srl $1,$2,10	$1 = $2 >> 10	Shift right by constant
sra $1,$2,10	$1 = $2 >> 10	Shift right (sign extend)
sllv $1,$2,$3	$1 = $2 << $3	Shift left by variable
srlv $1,$2, $3	$1 = $2 >> $3	Shift right by variable
srav $1,$2, $3	$1 = $2 >> $3	Shift R arith. by variable

Example of Conditional Branch Instruction Format

Instruction	Meaning
Branch on =	beq $1,$2,100 if ($1 == $2) go to PC+4+100
	Equal test; PC relative branch
Branch on	bne $1,$2,100 if ($1!= $2) go to PC+4+100
	Not equal test; PC relative SSS

ssss set on less than slt $1,$2,$3 if ($2 < $3) $1=1; else $1=0
Compare less than; 2's comp.
set < imm. slti $1,$2,100 if ($2 < 100) $1=1; else $1=0
Compare < constant; 2's comp.

set < uns. sltu $1,$2,$3 if ($2 < $3) $1=1; else $1=0
Compare less than; natural numbers
set < imm. uns. sltiu $1,$2,100 if ($2 < 100) $1=1; else $1=0
Compare < constant; natural numbers

Example of Unconditional Branch Instruction Format

Instruction	Meaning
jump	j 10000 go to 10000
	Jump to target address
jump register	jr $31 go to $31
	For switch, procedure return
jump and link	jal 10000 $31 = PC + 4; go to 10000
	For procedure call

These instructions depend on the type of operands used; there are three distinct types of

instructions. The instruction is *I-type* if the operand is an *immediate* instruction, *R-type* if the operand is a *register* instruction, or *J-type* if the operand is related to *jump* instructions. Figure 2 displays the differences in these three types. In a MIPS processor each instruction is performed as a sequence of steps. The steps corresponding to one instruction are referred as an *instruction cycle*. Figure 2 represents this concept:
Instruction Cycle = Fetch instruction + Decode + Fetch operand + Execute instruction +Store + Next instruction.
The *Fetch instruction* reads instructions from the memory.
The *Decode instruction* finds types of instructions and operands.
The *Fetch operand* reads operands from the memory.
The *execution cycle* executes instruction.

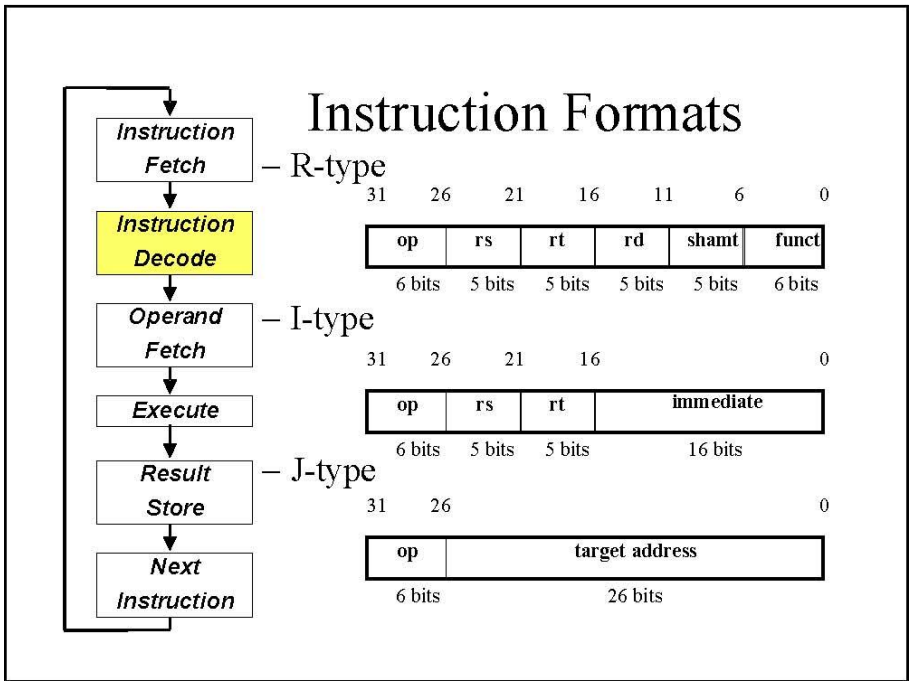

Figure 2: A view of instruction cycle and instruction format

4. VHDL model for MIPS processor

Advance features of VHDL could be used to describe a processor and its operations. Some of the features are listed here.
Example of memory declaration is as follows:

type MEMORY is array (0 to 32767) of STD_LOGIC_VECTOR (7 downto 0);

Example of input/output declaration is as follows:

-example of reading infile

File infile: text is in “ram.txt”;
VARIABLE buf: line;
VARIABLE b: std_vector(7 downto 0);
If not ENDFILE (infile) then
READLINE (infile,buf);
HREAD (buf, b);
Else…..

Sometimes designer prefers to assign a name to a number. The following program illustrates the use of **GENERIC** declaration for an 8KB memory. In this program *width* replaces number 8 and *address* replaces 15.

GENERIC (width: positive:=8);
GENERIC (address : positive:=15);
type MEMORY is array (0 to 2**address-1) of STD_LOGIC_VECTOR (width-1 downto 0);
VHDL model of a processor consists of several files.

1-Fetch.vhd
2-Decode.vhd
3-Control.vhd
4-Execute.vhd
5-Memory.vhd
6-Top.vhd

Fetch.vhd file explains instruction reading from memory. Decode.vhd detects type of instruction. Control.vhd generates all proper signals for a specific opcode (operation code). Execute.vhd executes instruction. Memory.vhd explains memory reading and writing. Top.vhd contains all the files. Following is a homework assigned to the students.

Homework: Describe in VHDL, MIPS instructions such as: LW (load) with opcode = 100011, SW (store) with opcode = 101011, and BEQ (branch equal) with opcode = 000100.

Solution: Following is a portion of the solution to the homework problem above. A complete solution is in the Appendix 1.

```
architecture behavioral of control is
    signal  Rformat, Lw, Sw, Beq : std_logic;
begin          -- behavior of SPIM control
                    Rformat <= ((NOT Op(5)) AND (NOT Op(4)) AND (NOT Op(3)) AND
                                    (NOT Op(2)) AND (NOT Op(1)) AND (NOT Op(0)));
                    Lw      <= (    Op(5)) AND (NOT Op(4)) AND (NOT Op(3)) AND
                                    (NOT Op(2)) AND (    Op(1)) AND (    Op(0));
                    Sw      <= (    Op(5)) AND (NOT Op(4)) AND (    Op(3)) AND
                                (NOT Op(2)) AND (    Op(1)) AND (    Op(0));
                    Beq     <= (NOT Op(5)) AND (NOT Op(4)) AND (NOT Op(3)) AND
                                    (    Op(2)) AND (NOT Op(1)) AND (NOT Op(0));
```

```
        RegDst <=  Rformat;
        ALUSrc <=  Lw or Sw;
        MemtoReg <=  Lw;
        RegWrite <=  Rformat or Lw;
        MemRead <=  Lw;
        MemWrite <=  Sw;
        Branch <=  Beq;
        ALUOp1 <=  Rformat;
        ALUOp0 <=  Beq;
end behavioral;
```

5- Conclusion

Combining VHDL and MIPS processor will result a useful tool for teaching hardware courses. This paper discusses the use of VHDL in a computer architecture course. As a demonstration, MIPS processor is selected to explain different steps in an instruction cycle. The MIPS processor is similar to (and therefore a good example of) many other available RISC processors in the market today. With the help of VHDL notation and MULTISIM software, instructors may assign numerous homework and projects (See Appendix A) for implementing different processes such as fetch, decode, and execute cycles in a processor.

6- References

1- Sudhakar Yalamanchili, VHDL Starter's Guide, Prentice Hall, ISBN 0-13-519802-X, 1998

2-William Stallings, Computer Organization and Architecture, Fifth Edition, Prentice Hall, ISBN 0-13-081294-3, 2001

3 - D. Patterson and J. Hennessy, Computer Organization and Design:
The Hardware/Software Interface, Second Edition, Morgan Kaufman Publishers, 1998

4- J. W. Stewart and C.Y. Wang, Digital Electronics Laboratory Experiments using the Xilinix XC95108 CPLD, Prentice Hall, 2005

5- Foundation series software, XILINIX Student Edition 4.2i, 2006

6- B. Parhami, Computer Architecture From Microprocessors to Supercomputers, Oxford University Press, 2005

7- Appendix

This section includes solution to the homework.

A- Control.vhd

```
-- control module (simulates SPIM control module)
 library Synopsys, IEEE;
```

```
use Synopsys.attributes.all;
use  IEEE.STD_LOGIC_1164.all;
entity control is
  port( signal Op : in std_logic_vector(5 downto 0);
            signal RegDst : out std_logic;
            signal ALUSrc : out std_logic;
            signal MemtoReg : out std_logic;
            signal RegWrite : out std_logic;
            signal MemRead : out std_logic;
            signal MemWrite : out std_logic;
            signal Branch : out std_logic;
            signal ALUop0 : out std_logic;
            signal ALUop1 : out std_logic;
            signal phi1,phi2: in std_logic);

end control;
-- SPIM control architecture
architecture behavioral of control is

  signal  Rformat, Lw, Sw, Beq : std_logic;


begin         -- behavior of SPIM control

                  Rformat <= ((NOT Op(5)) AND (NOT Op(4)) AND (NOT Op(3)) AND
                                    (NOT Op(2)) AND (NOT Op(1)) AND (NOT Op(0)));

                  Lw     <= (    Op(5)) AND (NOT Op(4)) AND (NOT Op(3)) AND
                                    (NOT Op(2)) AND (    Op(1)) AND (    Op(0));

                  Sw     <= (    Op(5)) AND (NOT Op(4)) AND (    Op(3)) AND
                                    (NOT Op(2)) AND (    Op(1)) AND (    Op(0));

                  Beq    <= (NOT Op(5)) AND (NOT Op(4)) AND (NOT Op(3)) AND
                                    (    Op(2)) AND (NOT Op(1)) AND (NOT Op(0));

            RegDst <=  Rformat;
            ALUSrc <=  Lw or Sw;
            MemtoReg <=  Lw;
            RegWrite <=  Rformat or Lw;
            MemRead <=  Lw;
            MemWrite <=  Sw;
            Branch <=  Beq;
            ALUOp1 <=  Rformat;
            ALUOp0 <=  Beq;
```

```
end behavioral;
```

B- Memory.vhd

```
-- DMEMORY module (provides the data memory for the SPIM computer)
library Synopsys, IEEE;
use Synopsys.attributes.all;
use IEEE.STD_LOGIC_1164.all;

entity dmemory is
  port(rd_bus : out std_logic_vector(7 downto 0);
       ra_bus : in std_logic_vector(7 downto 0);
       wd_bus : in std_logic_vector(7 downto 0);
       wadd_bus : in std_logic_vector(7 downto 0);
       MemRead, Memwrite, MemtoReg : in std_logic;
            phi1,phi2,reset: in std_logic);

end dmemory;

-- DMEMORY architecture

architecture behavioral of dmemory is

        signal mem0,mem1,mem2,mem3,mem4,mem5,mem6,mem7 : std_logic_vector(7 downto
0);
        signal mux : std_logic_vector(7 downto 0);
        signal mem0write, mem1write, mem2write, mem3write : std_logic;
        signal mem4write, mem5write, mem6write, mem7write : std_logic;
begin
-- Read Data Memory
with ra_bus(2 downto 0) select
mux &lt;= mem0 WHEN "000",
                mem1 WHEN "001",
                mem2 WHEN "010",
                mem3 WHEN "011",
                mem4 WHEN "100",
                mem5 WHEN "101",
                mem6 WHEN "110",
         mem7 WHEN "111",
         To_stdlogicvector(X"FF") WHEN others;
-- Mux to skip data memory for Rformat instructions
rd_bus &lt;= ra_bus(7 downto 0) WHEN (MemtoReg='0') ELSE mux WHEN (MemRead='1')
            ELSE To_Stdlogicvector(B"11111111");
-- Write to data memory?
-- The following code sets an initial value and replaces the next line
--dff_v(wd_bus,phi2 AND Memwrite AND (wadd_bus(2 downto 0)="000"),mem0);
```

```
mem0write &lt;= '1' When ((Memwrite='1') AND(wadd_bus(2 downto 0)="000"))
          ELSE '0';
mem1write &lt;= '1' When ((Memwrite='1') AND (wadd_bus(2 downto 0)="001"))
          ELSE '0';
mem2write &lt;= '1' When ((Memwrite='1') AND (wadd_bus(2 downto 0)="010"))
          ELSE '0';
mem3write &lt;= '1' When ((Memwrite='1') AND (wadd_bus(2 downto 0)="011"))
          ELSE '0';
mem4write &lt;= '1' When ((Memwrite='1') AND (wadd_bus(2 downto 0)="100"))
          ELSE '0';
mem5write &lt;= '1' When ((Memwrite='1') AND (wadd_bus(2 downto 0)="101"))
          ELSE '0';
mem6write &lt;= '1' When ((Memwrite='1') AND (wadd_bus(2 downto 0)="110"))
          ELSE '0';
mem7write &lt;= '1' When ((Memwrite='1') AND (wadd_bus(2 downto 0)="111"))
          ELSE '0';
process
begin
wait until phi2'event and phi2='1';
  if (reset = '1') then
      mem0 &lt;= To_stdlogicvector(X"55");
      mem1 &lt;= To_Stdlogicvector(X"AA");
  else
  if mem0write= '1' then mem0 &lt;= wd_bus; else mem0 &lt;= mem0; end if;
  if mem1write= '1' then mem1 &lt;= wd_bus; else mem1 &lt;= mem1; end if;
  if mem2write= '1' then mem2 &lt;= wd_bus; else mem2 &lt;= mem2; end if;
  if mem3write= '1' then mem3 &lt;= wd_bus; else mem3 &lt;= mem3; end if;
  if mem4write= '1' then mem4 &lt;= wd_bus; else mem4 &lt;= mem4; end if;
  if mem5write= '1' then mem5 &lt;= wd_bus; else mem5 &lt;= mem5; end if;
  if mem6write= '1' then mem6 &lt;= wd_bus; else mem6 &lt;= mem6; end if;
  if mem7write= '1' then mem7 &lt;= wd_bus; else mem7 &lt;= mem7; end if;
  end if;
end process;
end behavioral;
```