HL-HDL (High-Level Hardware Description Language)

COMS W4115 PLT

Project Proposal

Woon G. Lee (wgl2107)

March 25, 2014

1. Introduction

There are two major HDLs (Hardware Description Languages), VHDL and Verilog, mostly used to describe the hardware behaviors. They can define signal types, components, functions, and procedures using their own syntax. Then, the program is compiled and synthesized by HDL compilers supported by multiple companies such as Altera, Xilinx, Mentor Graphics and so on. HDL designers should describe all the details using HDLs which are sometimes tedious tasks. When the designer plans hierarchical design, it often starts with top-level block diagram, then each block is filled up by writing HDL codes and that is when the designer can verify the whole design. In other words, the designer has to implement the design with HDL only from top to all the way down to bottom layer. HL-HDL is designed to support the VHDL implementation of the design engineer's concepts easily without direct VHDL coding. It can support basic building blocks such as buffer, latch, multiplexer, divider, state machine using pre-defined keywords. By doing so, the hardware design engineer does not need to set up the VHDL coding structure but simply implement the design block using HL-HDL, then it will be implemented into VHDL source code by HL-HDL compiler. As stated, the output of the HL-HDL compiler would be the VHDL source files which can be applied to the existing VHDL compiler. In this project, the output will be compiled and simulated using Quartus II (Altera's VDHL compilation software) and verified.

2. Syntax

2.1 Data Types

Constant: binary or hex number that will be used to define reset value of the registers, bus width.

Signal: assign single-bit signal names

Bus: assign multiple-bits names

2.2 Operators

= : signal assignment

AND : logical AND for single bit or bus signals

OR : logical OR for single bit or bus signals

INV : invert single bit or bus signals

== : comparator (equal to)

!= : comparator (not equal to)

2.3 Functions

BUF(signal or bus A): Adds a buffer to signal A

LAT(signal CLK, signal or bus A): Adds a latch to signal A at the rising edge of clock CLK

MUX(signal or bus CTRL1, signal or bus A, B, …, N): Outputs a multiplexed signal based on CTRL1

REG(bus BASE_ADDR, bus A, signal RW): Generates a register for signal A with READ/WRITE property (R = Read only, W = Write only, RW = read/write) at base address BASE_ADDR and it is saved on a separate VDHL file called "Constants_Registers.vhd"

RDATA(signal CS, signal OE, bus REG_ADDR, bus DATA): Reads data from a register at address REG_ADDR when CS and OE are logic value 1 and returns it to DATA.

WDATA(signal CS, signal WR, bus REG_ADDR, bus DATA): Writes DATA onto a register at address REG_ADDR when CS and OE are logic value 1.

User-defined function can be used with "FUNC" keyword then define the rest like C. It will be used like components in VHDL.

2.4 Control Statement

If .. then .. else : If statement is similar to C-language that can be used to implement State Machines

2.5 Line terminator

;(semi-colon) is used as line terminator (C-like convention)

2.6 Comments

/* .. */ pair is used for comments (C-like convention)

3. Sample programs

The following sample program shows that CPU tried to read data from the pre-defined registers by asserting control signals and register address. Overall structure is pretty similar to C-language. First, FUNC read_reg is used for a component declaration which reads data from a register at the address given as a parameter. It is called in main body and implemented as a process in VHDL output file. Register write function can be coded in similar way.

```
FUNC read_reg(signal in_clk, reset, CSN, OEN, bus reg_addr(5:0), read_data(7:0))
/* functional block declaration that read back register value at register address reg_addr(5:0) */
{
  Signal CS, OE;

  CS = INV(CSN); /* invert input signal CSN in order to use active-high input to RDATA */
  OE = INV(OEN); /* invert input signal OEN in order to use active-high input to RDATA */
```

```
   if (reset == 1) then read_data = 0x00;   /* load the data at the rising edge of the clock */
   else if (in_clk == 1) then RDATA(CS, OE, reg_addr(5:0), read_data(7:0));
   end;
}

/* main body */
main(
      signal CLK, CPU_RESET, CPU_RDN, CPU_WRN, CPU_CSN;
      bus CPU_ADDR(5:0), CPU_DATA(7:0)
   )
{
   constant VERSION_REG(7:0) = 0x10;
   bus GENERAL1(7:0), GENERAL2(7:0);
   signal clk_buf;
   bus addr_reg(5:0), data_reg(7:0);

   clk_buf = BUF(CPU_CLK);   /* buffered input clock */

   /* Define registers */
   REG(0x00, VERSION_REG, R);
   REG(0x01, GENERAL1, RW);
   REG(0x02, GENERAL2, W);

   addr_reg = LAT(clk_buf, CPU_ADDR(5:0)); /* registered CPU address at rising edge of the clock */
   read_reg(clk_buf, CPU_CSN, CPU_RDN, addr_reg(5:0), data_reg(7:0))'
}
```