HL-HDL (High-Level Hardware Description Language)

Project Report

COMS W4115 PLT

Spring 2014

Woon G. Lee (wgl2107)

June 9, 2014
# Table of Contents

1. Introduction................................................................................................................................. 4

2. Tutorial........................................................................................................................................ 4  
   2.1 Input file .................................................................................................................................. 4  
   2.2 Execution ................................................................................................................................. 5  
   2.3 Output files .............................................................................................................................. 6  
   2.4 Example .................................................................................................................................. 6

3. Reference Manual .......................................................................................................................... 11  
   3.1 Grammar Notation ................................................................................................................... 11  
   3.2 Lexical Conventions ............................................................................................................... 11  
   3.3 Data Types .............................................................................................................................. 13  
   3.4 Expressions ............................................................................................................................. 14  
   3.5 Statements ............................................................................................................................... 15  
   3.6 Functions ................................................................................................................................. 15

4. Project Plan .................................................................................................................................... 18  
   4.1 Project execution steps ............................................................................................................. 18  
   4.2 Project timeline ....................................................................................................................... 19  
   4.3 Tools used for the project ....................................................................................................... 19

5. Architectural Design ....................................................................................................................... 20  
   5.1 Block Diagram ......................................................................................................................... 20  
   5.2 Interface between components in the block diagram ............................................................. 21

6. Test Plan......................................................................................................................................... 23  
   6.1 Test Environment .................................................................................................................... 23  
   6.2 Representative source programs and the generated targets ....... ......................................... 23  
   6.3 Test Suites ............................................................................................................................... 46
7. Lessons Learned .................................................................................................................47
   7.1 Importance of specification.........................................................................................47
   7.2 Understanding of the Implementation Language .....................................................47
   7.3 Time Management ....................................................................................................47
8. References .........................................................................................................................48
9. Appendix ............................................................................................................................49
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 Modelsim-Altera and verified.

2. Tutorial

HL-HDL takes an input file which is composed of global signals/constants, user-defined functions, and main function, then generates two output files in VHDL format. One has top-level VHDL code from main function and the other contains package declaration from global signals/constants and user-defined functions.

2.1 Input file

HL-HDL takes one input file which has one or more functions and optional global variables. Among functions in the input file, one main function with the keyword “main” should be included and other functions are indicated as user-defined ones by the keyword “func”. Global variables are declared as constant or signal with optional initial value assigned. The input file layout shows as follows.

```vhdl
/* comments can be added in the same fashion as C language */
/* Global constants and signals declared */
```
```vhdl
constant const1[7:0] := 0x5A;
signal global_sig1;

func user1( /* formals declaration */ )
{
    /* local variables declaration */
    /* Statements */
}

func user2( /* formals declaration */ )
{
    /* local variables declaration */
    /* Statements */
}

/* main function */
main foo(( /* formals declaration */ )
{
    /* local variables declaration */
    /* Statements */
}
```

### 2.2 Execution

Once the input file is complete, it is translated into multiple VHDL files by running the following command. It is not required but recommended using “.hhl” extension for the input filename to clearly show it is input file for HL-HDL.

```bash
/* -c option for compilation of the source file */
> /HL-HDL root directory/hl-hdl -c <input filename>

/* -t option for test bench VHDL code generation */
```
2.3 Output files

There are two VHDL files generated by compilation, `<input filename>_top.vhd` and `<input filename>_defs.vhd`.

<table>
<thead>
<tr>
<th>filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;input filename&gt;_top.vhd</code></td>
<td>Top-level entity generated from main function in the input file.</td>
</tr>
<tr>
<td><code>&lt;input filename&gt;_defs.vhd</code></td>
<td>VHDL package file which includes constants and signal declaration from global variables in the input file, component declaration and its entity with architecture from user-defined functions in the input file.</td>
</tr>
</tbody>
</table>

As an option, if `-t` command is executed, test bench VHDL file is generated. The test bench file has basic template as a helper file in case the user wants to create a test bench for simulation.

<table>
<thead>
<tr>
<th>filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;input filename&gt;_top_tbframe.vhd</code></td>
<td>Test bench VHDL file for the Top-level entity. It has library use clause and top-level component declaration with its instantiation.</td>
</tr>
</tbody>
</table>

2.4 Example

The following example program shows address decoder which takes two-bit address and generates 4-bit read strobe and 4-bit write strobe based on the control inputs such as chip select, out enable, and write enable. It is constructed as a user-defined function and called from the main function. The following shows input file called “test5.hhl”.

```vhdl
/* two-bit addr is decoded into four strobes using chip select, out enable, write enable */
func addr_decode(insig rst, insig cs, insig oe, insig we, insig addr[1:0],
                 outsig rd_strobe[3:0], outsig wr_strobe[3:0])
{
    signal rd_str;
    signal wr_str;
    signal strobe[3:0];

    rd_str = cs and oe;
    wr_str = cs and we;
    rst_cond: if (rst == 0b1) then
```
{  
  rd_strobe = 0x0;
  wr_strobe = 0x0;
}  
else  
{  
if (addr == 0b00) then  
  {  
  rd_strobe[0] = rd_str;
  rd_strobe[3:1] = 0b000;
  wr_strobe[0] = wr_str;
  wr_strobe[3:1] = 0b000;  
  }  
else if (addr == 0b01) then  
  {  
  rd_strobe[0] = 0b0;
  rd_strobe[1] = rd_str;
  rd_strobe[3:2] = 0b00;
  wr_strobe[0] = 0b0;
  wr_strobe[1] = wr_str;
  wr_strobe[3:2] = 0b00;  
  }  
else if (addr == 0b10) then  
  {  
  rd_strobe[1:0] = 0b00;
  rd_strobe[2] = rd_str;
  rd_strobe[3] = 0b0;
  wr_strobe[1:0] = 0b00;
  wr_strobe[2] = wr_str;
  wr_strobe[3] = 0b0;  
  }  
else  
  {  
  rd_strobe[2:0] = 0b000;
  rd_strobe[3] = rd_str;
  wr_strobe[2:0] = 0b000;
  wr_strobe[3] = wr_str;  
  } 
} 

main test5(insig rst, insig csn, insig oen, insig wrn, insig addr[7:0],
outsig rd_str[3:0], outsig wr_str[3:0])
{  
  signal chip_sel;
  signal rd_en;
  signal wr_en;
  /* invert active-low input signals to active-high ones */
  chip_sel = inv(csn);
  rd_en = inv(oen);
  wr_en = inv(wrn);
  /* call address decoder with partial address addr[1:0] */
  inst_addr_decode: addr_decode(rst, chip_sel, rd_en, wr_en, addr[1:0], rd_str, wr_str);
} 

The first two output files with –c compilation option are shown as below. Test5_top.vhd is the main entity and test5_defs.vhd is the package code.

<table>
<thead>
<tr>
<th>Filename:</th>
<th>test5_top.vhd</th>
</tr>
</thead>
<tbody>
<tr>
<td>library IEEE;</td>
<td></td>
</tr>
<tr>
<td>use IEEE.STD_LOGIC_1164.all;</td>
<td></td>
</tr>
<tr>
<td>use IEEE.STD_LOGIC_ARITH.ALL;</td>
<td></td>
</tr>
<tr>
<td>use IEEE.STD_LOGIC_UNSIGNED.ALL;</td>
<td></td>
</tr>
<tr>
<td>use work.globals.all;</td>
<td></td>
</tr>
</tbody>
</table>
```vhdl
use work.lib_comp.all;

ENTITY test5_top is
port (
  rst : in  std_logic;
  csn : in  std_logic;
  oen : in  std_logic;
  wrn : in  std_logic;
  addr : in std_logic_vector(7 downto 0);
  rd_str : out std_logic_vector(3 downto 0);
  wr_str : out std_logic_vector(3 downto 0);
); end test5_top;

architecture RTL of test5_top is
  signal chip_sel :  std_logic;
  signal rd_en :  std_logic;
  signal wr_en :  std_logic;
begin
  chip_sel <= NOT csn;
  rd_en <= NOT oen;
  wr_en <= NOT wrn;
  inst_addr_decode_addr_decode : addr_decode port map (
    rst => rst,
    cs => chip_sel,
    oe => rd_en,
    we => wr_en,
    addr => addr(1 downto 0),
    rd_strobe => rd_str,
    wr_strobe => wr_str
  );
end RTL;

Filename:  test5_defs.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

package globals is
  -- Constants and global signals declaration

  -- Components declaration
  component addr_decode is
    port (
      rst : in  std_logic;
      cs : in  std_logic;
      oe : in  std_logic;
      we : in  std_logic;
      addr : in std_logic_vector(1 downto 0);
      rd_strobe : out std_logic_vector(3 downto 0);
      wr_strobe : out std_logic_vector(3 downto 0);
    );
  end component addr_decode;

end globals;
```

-- Entities and architectures of the components
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY addr_decode is
  port (  
    rst : in  std_logic;  
    cs : in  std_logic;  
    oe : in  std_logic;  
    we : in  std_logic;  
    addr : in std_logic_vector(1 downto 0);  
    rd_strobe : out std_logic_vector(3 downto 0);  
    wr_strobe : out std_logic_vector(3 downto 0)
  );
end addr_decode;

architecture RTL of addr_decode is
  signal rd_str :  std_logic;  
  signal wr_str :  std_logic;  
  signal strobe : std_logic_vector(3 downto 0);
begin  
  rd_str <= cs AND oe;  
  wr_str <= cs AND we;
  proc_rst_cond : process (all)
  begin
    if rst = '1' then
      rd_strobe <= x"0000";
      wr_strobe <= x"0000";
    else
      if addr = "00" then
        rd_strobe(0) <= rd_str;
        rd_strobe(3 downto 1) <= "0000";
        wr_strobe(0) <= wr_str;
        wr_strobe(3 downto 1) <= "0000";
      else
        if addr = "01" then
          rd_strobe(1 downto 0) <= "00";
          rd_strobe(2) <= rd_str;
          rd_strobe(3) <= '0';
          wr_strobe(1 downto 0) <= "00";
          wr_strobe(2) <= wr_str;
          wr_strobe(3) <= '0';
        else
          rd_strobe(2 downto 0) <= "0000";
          rd_strobe(3) <= rd_str;
          wr_strobe(2 downto 0) <= "0000";
          wr_strobe(3) <= wr_str;
        end if;
      end if;
    end if;
  end if;
end if;
end if;
end if;
end if;
end if;
end process;

end RTL;

configuration CFG_addr_decode of addr_decode is
    for RTL
        end for;
    end CFG_addr_decode;

The following file shows optional test bench template by the compilation with –t option. User needs to fill up the rest of the code from this template since it is kind of user helper output.

```
Filename:  test5_top_tbframe.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY test5_top_tb is
end ENTITY test5_top_tb;

ARCHITECTURE behavioral of test5_top_tb is

COMPONENT test5_top is
    port (  
        rst : in  std_logic;  
        csn : in  std_logic;  
        oen : in  std_logic;  
        wrn : in  std_logic;  
        addr : in std_logic_vector(7 downto 0);  
        rd_str : out std_logic_vector(3 downto 0);  
        wr_str : out std_logic_vector(3 downto 0)
    );
end COMPONENT;

begin

inst_test5_top : test5_top
    port map(  
        rst => rst,  
        csn => csn,  
        oen => oen,  
        wrn => wrn,  
        addr => addr,  
        rd_str => rd_str,  
        wr_str => wr_str
    );
```

3.1 Grammar Notation

Regular expression notation is used in this document. ‘s*’ denotes that it has zero or more s’s, ‘s+’ that it has at least one s, (s|r) that s or r could be, (s & r) that s and r are concatenated, where parentheses are used to group the symbols.

3.2 Lexical Conventions

A program is contained in a single file, which has a sequence of tokens to be processed.

3.2.1 Line Terminator

Semi-colon character (;) is used as line terminator.

3.2.2 Comments

C-like comment is supported. Comment begins with characters “/*” and ends with “*/”. Any sequence of characters except for “*/” can be used between these two character combinations.

3.2.3 Tokens

There are five kinds of tokens: Identifiers, keywords, constants, expression operators, and separators. White spaces such as spaces, tabs, newlines, and carriage returns are used to delineate tokens.

3.2.4 Identifiers

An identifier is a sequence of alphanumeric characters which must begin with letter and/or can be followed by numbers. Underscore (‘_’) is considered as a letter and bus identifier must have parenthesized number range at the end. Identifier is not case-sensitive whose length is limited to 16.

letter = [‘a’-‘z’ ‘A’-‘Z’]
digit = [‘0’-‘9’]
identifier = letter (letter | digit | ‘_’)*
3.2.5 Keywords

The following keywords are reserved for use as keywords and may not be used otherwise.

<table>
<thead>
<tr>
<th>insig</th>
<th>outsig</th>
<th>iosig</th>
<th>constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>signal</td>
<td>and</td>
<td>or</td>
<td>inv</td>
</tr>
<tr>
<td>buf</td>
<td>lat</td>
<td>mux</td>
<td>func</td>
</tr>
<tr>
<td>if</td>
<td>then</td>
<td>else</td>
<td>main</td>
</tr>
</tbody>
</table>

3.2.6 Constants

There are three kinds of constants as follows:

3.2.6.1 Integer constants

An integer constant is a sequence of digits.

Integer constant: (digit)+

3.2.6.2 Binary constants

A binary constant is a sequence of binary digits ‘0’ or ‘1’, which could be single or multiple digits. A prefix ‘0b’ is required.

Binary constant: “0b” (‘0’|’1’)+

For example, ‘0’ or ‘1’ is a single digit binary constant for a signal type and “0011”, “1010” are 4-digit binary constants for a bus type.

3.2.6.3 Hexadecimal constants

A hexadecimal constant is a sequence of hexadecimal digits, ‘0’ ~ ‘9’, ‘A’ ~ ‘F’. A prefix ‘0x’ is required.

Hexadecimal constant: “0x” hexdigit +

For example, 0x1FC is a valid hexadecimal constant.

3.2.7 Expression Operators

| = : | signal or bus assignment |
| AND : | logical AND for signals or buses |
| OR : | logical OR for signals or buses |
| INV : | invert signal or bus |
| == : | logical comparator (equal to) for signals or buses |
!= : logical comparator (not equal to) for signals or buses

3.2.8 Separators

The following ASCII characters are used as separators:

; : line terminator

[ : ] : bus range specifier

{ } : main or user function definition

, : argument separator

3.3 Data Types

3.3.1 Primitive data types

The following primitive data types are supported in HL-HDL.

3.3.1.1 Integer

An integer type is used to define integral value to the identifier.

3.3.1.2 Signal

A signal type is used to define binary value to the single-bit identifier or multi-digit binary / hexadecimal value to the multiple-bit identifier.

e.g. foo = 0b1; /* single-bit binary number */
foo[7:0] = 0b11010011; /* 8-bit binary number */
foo[7:0] = 0x5A; /* 8-bit hexadecimal number */

3.3.2 Literals

3.3.2.1 Integer Literals

Integer literal is a sequence of digits and defined as follows:

integer = (digit)+

3.3.2.2 Binary Literals

Single-bit signal literal is a single binary digit or a sequence of binary digits and defined as follows:

Binary = “0b”(‘0’|’1’)+

3.3.2.3 Hexadecimal Literals
Hexadecimal literal is a sequence of hexadecimal digits defined as follows:

\[
\text{hex} = \{\text{'0'}-\text{'9'}, \text{'a'}-\text{'f'}, \text{'A'}-\text{'F'}\}
\]

Hexadecimal = “0” (hex)+

3.4 Expressions

Expressions are evaluated in the order as follows. Also, left- or right-associativity is specified in each subsection.

3.4.1 Primary expressions

Primary expressions have left-to-right associativity.

3.4.1.1 Identifier

An identifier is a primary expression whose type must be properly declared.

3.4.1.2 Constant

An integer, binary, or hexadecimal constant is a primary expression. Its type is integer, signal, and bus, respectively.

3.4.1.3 (expression)

A parenthesized expression is a primary expression and its type and value are identical to those of “expression”.

3.4.2 expression₁ ( expression₂ )

Expression followed by an expression is a part of whole bus signals, where expression₁ is bus identifier and expression₂ is in the range of the bus space.

3.4.3 Logical Expressions

The following are the logical expressions and the first two have left-to-right and the last does right-to-left associativity.

- \( \text{expression}_1 \text{ and expression}_2 \) : logical AND
- \( \text{expression}_1 \text{ or expression}_2 \) : logical OR
- inv (expression) : logical NOT

3.4.4 Numerical Expressions

The following numerical expressions have left-to-right associativity.

- \( \text{expression}_1 == \text{expression}_2 \) : equal to
expression₁ \neq expression₂ :\ not\ equal\ to

3.4.5 Assignment Expressions

An assignment expression has right-to-left associativity.

expression₁ = expression₂

3.4.6 Operator precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator type</th>
<th>Operators</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>primary</td>
<td>( )</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>2</td>
<td>unary</td>
<td>NOT</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>3</td>
<td>binary</td>
<td>AND, OR</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>4</td>
<td>numerical</td>
<td>==, !=</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>5</td>
<td>assignment</td>
<td>=</td>
<td>Right-to-left</td>
</tr>
</tbody>
</table>

3.5 Statements

Statements are executed in sequence.

3.5.1 Expression statement

Most expression statements have the form as follows:

expression ;

3.5.2 IF statement

If statement is a conditional statement which has two forms as follows:

optional identifier : if ( expression ) then statement ;

optional identifier : if ( expression ) then statement else statement ;

In both cases, expression is evaluated first and if it is true, the first statement is executed. However, in the second case, if it is not true, the second statement is executed.

An identifier is required for outer-most if statement for easier translation to VHDL format.

3.6 Functions

There are a couple of generic functions provided in HL-HDL, which are the most generic functional block used in VHDL. Otherwise, user-defined function can be declared and called. User-defined function can be declared
anywhere in the program. In regard to mapping to VHDL implementation, it is mapped to “component”
declaration in the VHDL package file.

3.6.1 Generic functions (built-in functions)

The following functions are provided in HL-HDL.

Function_identifier ( parameter list )

3.6.1.1 BUF function

(buf expression): expression is either single or multiple-bit signal and a buffer is added, whose output is the 
same type as input expression.

e.g. A = buf(B);

3.6.1.2 INV function

(inv expression): expression is either single or multiple-bit signal and an inverter is added, whose output is the 
same type as input expression.

e.g. A = inv(B);

3.6.1.3 LAT function

(lat reset_identifier, clock_identifier, list of expression ): “reset_identifier” is reset input source to reset the 
output of the latch, “clock_identifier” is a clock source for the latch and the list of expressions has input 
expression to the latch and the output out of the latch. Function instantiation identifier is required at the 
beginning of the call.

e.g. latch_inst1: lat(rst, clk, LAT_in, LAT_out);

3.6.1.4 4-to-1 MUX function

(mux output_identifier, mux_enable_identifier, SELECT_expression, list of expressions): “output_identifier” is 
the output assigned out of four inputs. “mux_enable_identifier” is the output enable input which enables the 
mux output. When the mux is disabled, its output is tri-stated. “SELECT_expression” is two-bit input to 
select one out of four inputs to the mux output. The list of expressions is the 4 inputs to the mux..

e.g. mux_inst1 : mux (mux_output, mux_enable, mux_out_select, mux_input0, , mux_input1, , 
mux_input2, mux_input3);

3.6.2 User-defined function

User-defined function can be declared with a keyword func followed by list of formal parameters in parentheses
and statements in { }.

```vhdl
func identifier (expressions, ..., expressions)
{
  Statements
}
```

This corresponds to a COMPONENT declaration in VHDL. It can be instantiated by calling with function instantiation identifier in main function.

3.6.3 main function

Main function can be declared with a keyword main followed by a list of formal parameters in parentheses and statements in { }.

```vhdl
main identifier (expressions, ..., expressions)
{
  Statements
}
```
4. Project Plan

The project plan for the HL-HDL compiler design was to set up the steps and execute them.

4.1 Project execution steps

At project planning stage, the following steps were set up. Even after one stage was complete, it did not mean that the corresponding task, mostly coding task, did not freeze, but had to be revisited whenever a bug was found or new feature had to be added.

- **Project Specification**: After the research for what is appropriate for this project and based on my experience as a hardware engineer, a language to generate VHDL file were chosen. Main purpose of the language was to help a hardware designer generate VHDL code from the plain c-style input code. Due to the time limit on this project, a minimal set of expressions and statements were picked, which was the major input for the project proposal and the initial draft of the language reference manual.

- **Scanner design**: Tokens were defined based on the project specification and coded into the scanner. The initial scanner was tested using simple print function similar to wordcount.ml shown in class homework 1.

- **AST and Parser design**: All the data types, expressions, and statement structures were coded into ast.ml and parser.mly as specified. A pretty printer were implemented in ast.ml as helper functions for debug purpose, which is also added to an option for the compiler (“-a” option).

- **Translator design**: Global variables and a list of functions were used as the inputs to the translator. Each part was decomposed to signals/components in VHDL package and main entity in top-level VHDL code, respectively. Basic VHDL code structure such as library use clause was added to the output files. For verification purpose, a sort of helper function was added to generate test bench template for the top-level entity, so that the user can add their own test bench procedure easily.

- **Test program design**: Test codes for typical usage of the language, a couple of simple code with
more complicated multi-function code were generated. In order to verify the behavior of the

generated VHDL code, the Modelsim-Altera functional VDHL simulator was chosen as major
simulation tool, since it is easy to get, good for small size simulation, and also free software.

**Final Project Report**

### 4.2 Project timeline

<table>
<thead>
<tr>
<th>Period</th>
<th>Description for the tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1 ~ Week 4</td>
<td>Research for the project and proposal</td>
</tr>
<tr>
<td>Week 5 ~ Week 8</td>
<td>Initial draft of the scanner (scanner.mll) was coded and LRM (Language Reference Manual) submitted.</td>
</tr>
<tr>
<td>Week 9 ~ Week 12</td>
<td>AST (ast.ml) and Parser (parser.mly) coded and tested with “pretty printer” included in ast.ml as helper function. Initial main program (hlhdl.ml) was also coded.</td>
</tr>
<tr>
<td>Week 13 ~ Week 15</td>
<td>Translator (translate.ml) designed and the generated VDHL files from the source file was compiled and simulated with Modelsim-Altera VHDL simulator.</td>
</tr>
<tr>
<td>Week 16</td>
<td>Final project report.</td>
</tr>
</tbody>
</table>

### 4.3 Tools used for the project

The following tools were used for the development and testing.

- **Notepad++**: main editing tool
- **OCAML**: compiler language
- **OcamWin**: supplementary ocaml command test
- **Cygwin 64-bit terminal**: terminal for compiling ocaml source code and execution
- **Modelsim-Altera**: main VHDL simulation tool
5. Architectural Design

5.1 Block Diagram

HL-HDL is composed of the following components shown in the picture below. HHL input file is taken as an input, then outputs three VHDL files and one text file based on the compilation options. AST (ast.ml) provides helper functions for pretty printer in translator as well as syntax tree itself. Interface between components is described in more detail in the following section.

![Block Diagram of HL-HDL](image-url)
5.2 Interface between components in the block diagram

5.2.1 Scanner
Scanner takes HHL input file in text format as an input and generates tokens based on the rules set up in scanner.mll, which is passed to Parser.

5.2.2 Parser
Parser takes tokens from the Scanner and builds an AST based on the parsing rules set up in parser.mly. The output is not semantically checked AST, which is covered in Translator.

5.2.3 Translator
Translator takes AST from the Parser and process the input list such as global variable and function list in the following way, and generates output string for the main HLHDL. HLHDL uses one of the following strings to generate the output file.

(i) Global variables and User-defined functions
Global variables declared in hhl input file are the first part of package called “globals” in the <filename>_defs.vhd. A header string for VHDL library and use clause is added to the beginning of the file, too.

User defined function is used to create a string that contains component declaration and the entity with its architecture body. Typical VHDL libraries are also added to each entity to the beginning of the entity. This portion of string is also the second part of <filename>_defs.vhd.

(ii) Main function
A top-level entity and its architecture is generated from the main function with the same VHDL libraries used for package file above. This string is used for top-level vhdl file, <filename>_top.vhd

(iii) Test bench template
VHDL simulation is the most popular way to verify the behavior of the entity generated and requires a test bench. A header, top-level component declaration, and its instantiation are easily collected from the string above, which forms a string for the test bench for top-level entity.

(iv) Pretty printer
Pretty printer simply prints out a string that is close to the input file format just for debug purpose.
Usually, this string can be used at the early stage for the verification of the scanner or when new tokens are added.

5.2.4 HLHDL

HLHDL calls one of the translate functions to let it create a string shown in 5.2.3 and write it to the appropriate filename. It creates a file, `<filename>_top.vhd from the main function string, `<filename>_defs.vhd from the global variables and user-defined functions, `<filename>_tbframe.vhd from part of main function string, and `<filename>.ast from AST.
6. Test Plan

In order to verify the generated target program works as designed, a VHDL functional simulation tool, Modelsim-Altera, was chosen. There are a couple of test programs generated to test the language features and the output was verified using the simulation tool.

6.1 Test Environment

Two output files, top-level entity and the package file, are fed into the Modelsim-Altera as input VHDL source files with a test bench source file modified from the test bench template. In addition, a library file “lib_comps.vhd” is provided for simulation. The library file contains two built-in functions such as latch and 4-to-1 mux. One weak point with this tool is, automated testing is not supported due to the tool usage limit.

![Modelsim-Altera Simulation Diagram]

6.2 Representative source programs and the generated targets

6.2.1 Global constant and main function with built-in function

In this test source program, a couple of global constants were declared which is used in main function. Eight 8-bit data (data_in1[7:0] ~ data_in8[7:0]) are declared as global constants, and one of first 4 data is selected by the first 4-to-1 mux and the other from the rest by the second 4-to-1 mux based on two-bit address bus in the main function.

```
Filename: test4.hhl
COMS W4115 PLT - 2014 Spring Term Project
HL-HDL compiler: translate.ml
Woon Lee (wgl2107)

test4.hhl: test case 4
- global constant declaration test based on test case 3
```
- constants declared are used for 8-to-1 mux input

constant data_in1[7:0] := 0x80;
current data_in2[7:0] := 0x85;
current data_in3[7:0] := 0x8A;
current data_in4[7:0] := 0x8F;
current data_in5[7:0] := 0xC0;
current data_in6[7:0] := 0xC5;
current data_in7[7:0] := 0xCA;
current data_in8[7:0] := 0xCF;

main test4(insig clk, insig rst, insig strobe1, insig strobe2, insig addr[2:0], outsig data_out[7:0])
{
    signal mux1_en;
signal mux2_en;

    mux_ctrl: if (rst == 0b1) then
    {
        mux1_en = 0b0;
mux2_en = 0b0;
    }
else
    {
        if ((strobe1 == 0b1) and (strobe2 == 0b1)) then
            {mux1_en = inv(addr[2]);
mux2_en = addr[2];
        }
else
        {
mux1_en = 0b0;
mux2_en = 0b0;
    }
    }

    /* first 8-bit data 4-to-1 mux */
    four_bit_mux1: mux(data_out, mux1_en, addr[1:0], data_in1, data_in2, data_in3, data_in4);
    /* second 8-bit data 4-to-1 mux */
    four_bit_mux2: mux(data_out, mux2_en, addr[1:0], data_in5, data_in6, data_in7, data_in8);
}

The following files, test4_top.vhd and test4_defs.vhd, are generated by using “-c” compile option, and test
 bench template by using “-t” option.

<table>
<thead>
<tr>
<th>Filename: test4_top.vhd</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- COMS W4115 PLT - 2014 Spring Term Project</td>
</tr>
<tr>
<td>-- HL-HDL compiler</td>
</tr>
<tr>
<td>-- Woon Lee (wg2107)</td>
</tr>
</tbody>
</table>

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY test4_top is
port (
    clk : in  std_logic;
    rst : in  std_logic;
    strobe1 : in  std_logic;
    strobe2 : in  std_logic;
    addr : in std_logic_vector(2 downto 0);
    data_out : out std_logic_vector(7 downto 0)
); 
end test4_top;

architecture RTL of test4_top is
begin
    proc_mux_ctrl : process (all)
    begin
        if rst = '1' then
            mux1_en <= '0';
            mux2_en <= '0';
        else
            if (strobe1 = '1') AND (strobe2 = '1') then
                mux1_en <= NOT addr(2);
                mux2_en <= addr(2);
            else
                mux1_en <= '0';
                mux2_en <= '0';
            end if;
        end if;
    end process;

    four_bit_mux1 : MUX_n_bit
    generic map(num_bits => 8)
    port map ( 
        MUX_OUT => data_out,
        EN => mux1_en,
        OUT_SEL => addr(1 downto 0),
        MUX_IN0 => data_in1,
        MUX_IN1 => data_in2,
        MUX_IN2 => data_in3,
        MUX_IN3 => data_in4 
    ); 

    four_bit_mux2 : MUX_n_bit
    generic map(num_bits => 8)
    port map ( 
        MUX_OUT => data_out,
        EN => mux2_en,
        OUT_SEL => addr(1 downto 0),
        MUX_IN0 => data_in5,
        MUX_IN1 => data_in6,
        MUX_IN2 => data_in7,
        MUX_IN3 => data_in8 
    ); 
end RTL;
Since the source program has no user-defined functions, there is no corresponding component declaration in this test case.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

package globals is

    -- Constants and global signals declaration
    constant data_in1 : std_logic_vector(7 downto 0) := x"80";
    constant data_in2 : std_logic_vector(7 downto 0) := x"85";
    constant data_in3 : std_logic_vector(7 downto 0) := x"8A";
    constant data_in4 : std_logic_vector(7 downto 0) := x"8F";
    constant data_in5 : std_logic_vector(7 downto 0) := x"C0";
    constant data_in6 : std_logic_vector(7 downto 0) := x"C5";
    constant data_in7 : std_logic_vector(7 downto 0) := x"CA";
    constant data_in8 : std_logic_vector(7 downto 0) := x"CF";

    -- Components declaration
end globals;

-- Entities and architectures of the components
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY test4_top_tb is
end ENTITY test4_top_tb;

ARCHITECTURE behavioral of test4_top_tb is

COMPONENT test4_top is
port(
    clk : in  std_logic;
    rst : in  std_logic;

```
The test bench template above is modified and saved to test bench program for simulation as follows.
Different address bits for mux output selection every 75 ns are used for stimulus which resulted in mux output accordingly.

Filename: test4_top_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY test4_top_tb is
end ENTITY test4_top_tb;

ARCHITECTURE behavioral of test4_top_tb is

COMPONENT test4_top is
port (  
  strobe1 : in  std_logic;
  strobe2 : in  std_logic;
  addr : in std_logic_vector(2 downto 0);
  data_out : out std_logic_vector(7 downto 0)
);
end COMPONENT;

signal clk :  std_logic;
signal rst :  std_logic;
signal strobe1 :  std_logic;
signal strobe2 :  std_logic;
signal addr : std_logic_vector(2 downto 0);
signal data_out : std_logic_vector(7 downto 0);

begin
inst_test4_top : test4_top
  port map(
    clk => clk,
    rst => rst,
    strobe1 => strobe1,
    strobe2 => strobe2,
    addr => addr,
    data_out => data_out
  );

end;
```
clk : in  std_logic;
    rst : in  std_logic;
    strobe1 : in  std_logic;
    strobe2 : in  std_logic;
    addr : in std_logic_vector(2 downto 0);
    data_out : out std_logic_vector(7 downto 0)
end COMPONENT;

    signal clk :  std_logic;
    signal rst :  std_logic;
    signal strobe1 :  std_logic;
    signal strobe2 :  std_logic;
    signal addr : std_logic_vector(2 downto 0);
    signal data_out : std_logic_vector(7 downto 0);
    signal sim_done : boolean := false;

begin

inst_test4_top : test4_top
    port map(
        clk => clk,
        rst => rst,
        strobe1 => strobe1,
        strobe2 => strobe2,
        addr => addr,
        data_out => data_out
    );

clk_proc : process
    begin
        if (not sim_done) then
            clk <= '0';
            wait for 20 ns;
            clk <= '1';
            wait for 20 ns;
        else
            report "sim_done";
            wait; --wait forever
        end if;
    end process clk_proc;

input_proc : process
    begin
        strobe1 <= '0';
        strobe2 <= '0';
        wait for 30 ns;
        strobe1 <= '1';
        addr <= "000";
        wait for 10 ns;
        strobe2 <= '1';
        wait for 30 ns;
        strobe1 <= '0';
        wait for 5 ns;
        strobe2 <= '0';
        wait for 30 ns;
        strobe1 <= '1';
        addr <= "001";
        wait for 10 ns;
strobe2 <= '1';
wait for 30 ns;
strobe1 <= '0';
wait for 5 ns;
strobe2 <= '0';

wait for 30 ns;
strobe1 <= '1';
addr <= "010";
wait for 10 ns;
strobe2 <= '1';
wait for 30 ns;
strobe1 <= '0';
wait for 5 ns;
strobe2 <= '0';

wait for 30 ns;
strobe1 <= '1';
addr <= "011";
wait for 10 ns;
strobe2 <= '1';
wait for 30 ns;
strobe1 <= '0';
wait for 5 ns;
strobe2 <= '0';

wait for 30 ns;
strobe1 <= '1';
addr <= "100";
wait for 10 ns;
strobe2 <= '1';
wait for 30 ns;
strobe1 <= '0';
wait for 5 ns;
strobe2 <= '0';

wait for 30 ns;
strobe1 <= '1';
addr <= "101";
wait for 10 ns;
strobe2 <= '1';
wait for 30 ns;
strobe1 <= '0';
wait for 5 ns;
strobe2 <= '0';

wait for 30 ns;
strobe1 <= '1';
addr <= "110";
wait for 10 ns;
strobe2 <= '1';
wait for 30 ns;
strobe1 <= '0';
wait for 5 ns;
strobe2 <= '0';

wait for 30 ns;
strobe1 <= '1';
addr <= "111";
wait for 10 ns;
strobe2 <= '1';
wait for 30 ns;
strobe1 <= '0';
wait for 5 ns;
strobe2 <= '0';
wait;
end process;
sim_time : process
begin
rst <= '1';
wait for 10 ns;
rst <= '0';
wait for 600 ns;
sim_done <= true;
wait;
end process;
end;

From the waveform picture below, test result was verified.

6.2.2 Complex program test

Based on simple test cases, this test program covers almost every aspect of the language by implementing global variables/constants and multiple user-defined functions. Data are written to and read back from 16 registers represented by 16 global variables. Two user-defined functions, 2-to-4 address decoder and register reader, are created and controlled from the main function. In the main function, all the built-in functions are instantiated and single-bit/multiple-bit signals are declared and assigned in the statements.

The source program is shown as follows.

```hhl
Filename:  test6.hhl

COMS W4115 PLT - 2014 Spring Term Project
HL-HDL compiler: translate.ml
Woon Lee (wgl2107)

Filename: test6.hhl
```
- Complete test suite: covers almost every aspect of HL-HDL
  - multiple functions: user-defined and built-in functions
  - constants and signals as global variable (set of registers in this test case)
  - implement read/write function for 16 registers

/****************************************
***********************************************/

/* Global constants and variables */
constant VERSION[7:0] := 0x10;
constant REVISION[7:0] := 0x20;
signal REG0[7:0];
signal REG1[7:0];
signal REG2[7:0];
signal REG3[7:0];
signal REG4[7:0];
signal REG5[7:0];
signal REG6[7:0];
signal REG7[7:0];
signal REG8[7:0];
signal REG9[7:0];
signal REG10[7:0];
signal REG11[7:0];
signal REG12[7:0];
signal REG13[7:0];
signal REG14[7:0];
signal REG15[7:0];

/* two-bit input is decoded into four strobes */
func decode_2_to_4(insig rst, insig strb_in, insig in_2bit[1:0], outsig strb_out[3:0])
{
  rst_cond:
  if (rst == 0b1) then
  {
    strb_out = 0x0;
  }
  else
  {
    if (in_2bit == 0b00) then
    {
      strb_out[0] = strb_in;
      strb_out[3:1] = 0b000;
    }
    else if (in_2bit == 0b01) then
    {
      strb_out[0] = 0b0;
      strb_out[1] = strb_in;
      strb_out[3:2] = 0b00;
    }
    else if (in_2bit == 0b10) then
    {
      strb_out[1:0] = 0b00;
      strb_out[2] = strb_in;
      strb_out[3] = 0b0;
    }
    else
    {
      strb_out[2:0] = 0b000;
      strb_out[3] = strb_in;
    }
  }
}

/* Register READ: 16-to-1 8-bit-wide data mux using 4 mux functions */
func reg_read(insig rst, insig strb_in, insig addr_in[3:0], outsig data_out[7:0])
{
  signal bank_en[3:0];
  /* 4 enable strobes for 4-to-1 MUX generation using upper 2 addr_in bits */
  bank_sel: decode_2_to_4(rst, strb_in, addr_in[3:2], bank_en);
first_bank: `mux(data_out, bank_en[0], addr_in[1:0], REG0, REG1, REG2, REG3);
second_bank: `mux(data_out, bank_en[1], addr_in[1:0], REG4, REG5, REG6, REG7);
third_bank: `mux(data_out, bank_en[2], addr_in[1:0], REG8, REG9, REG10, REG11);
fourth_bank: `mux(data_out, bank_en[3], addr_in[1:0], REG12, REG13, REG14, REG15);

main test6(insig clk, insig rst, insig csn, insig oen, insig wrn, insig addr[3:0], iosig data[7:0])
{
  signal chip_sel;
  signal rd_en;
  signal rd_strb;
  signal rd_strb_latched;
  signal wr_en;
  signal wr_strb;
  signal wr_strb_latched;
  signal addr_latched[3:0];
  signal datain[7:0];
  signal dataout[7:0];
  signal reg_sel_strb[15:0];
  signal reg_bank_sel[3:0];

  /* Initialize the first two registers */
  REG0 = VERSION;
  REG1 = REVISION;

  /* invert active-low input signals to active-high ones */
  chip_sel = inv(csn);
  rd_en = inv(oen);
  rd_strb = chip_sel and rd_en;
  wr_en = inv(wrn);
  wr_strb = chip_sel and wr_en;

  /* make the strobes synchronous to the clk */
  latch_rd: lat(rst, 0b1, clk, rd_strb, rd_strb_latched);
  latch_wr: lat(rst, 0b1, clk, wr_strb, wr_strb_latched);
  latch_addr: lat(rst, 0b1, clk, addr, addr_latched);
  latch_datain: lat(rst, wr_strb, clk, data, datain);

  /* Register read */
  inst_cpu_read: reg_read(rst, rd_strb_latched, addr_latched, dataout);
  data = dataout;

  /* Register write */
  four_bank_sel: decode_2_to_4(rst, wr_strb_latched, addr_latched[3:2], reg_bank_sel);
  first_bank_strobes: decode_2_to_4(rst, reg_bank_sel[0], addr_latched[1:0], reg_sel_strb[3:0]);
  second_bank_strobes: decode_2_to_4(rst, reg_bank_sel[1], addr_latched[1:0], reg_sel_strb[7:4]);
  third_bank_strobes: decode_2_to_4(rst, reg_bank_sel[2], addr_latched[1:0], reg_sel_strb[11:8]);
  fourth_bank_strobes: decode_2_to_4(rst, reg_bank_sel[3], addr_latched[1:0], reg_sel_strb[15:12]);

  /* write the data to the selected Register on reg_sel_strb */
  /* The first two registers are read-only */
  /* data_write_reg0: lat(rst, reg_sel_strb[0], clk, datain, REG0);
  data_write_reg1: lat(rst, reg_sel_strb[1], clk, datain, REG1); */
  data_write_reg2: lat(rst, reg_sel_strb[2], clk, datain, REG2);
  data_write_reg3: lat(rst, reg_sel_strb[3], clk, datain, REG3);
  data_write_reg4: lat(rst, reg_sel_strb[4], clk, datain, REG4);
  data_write_reg5: lat(rst, reg_sel_strb[5], clk, datain, REG5);
  data_write_reg6: lat(rst, reg_sel_strb[6], clk, datain, REG6);
  data_write_reg7: lat(rst, reg_sel_strb[7], clk, datain, REG7);
  data_write_reg8: lat(rst, reg_sel_strb[8], clk, datain, REG8);
The following file shows top-level entity generated from the main function.

```vhdl
ENTITY test6_top is
  port ( 
    clk : in  std_logic;
    rst : in  std_logic;
    csn : in  std_logic;
    oen : in  std_logic;
    wrn : in  std_logic;
    addr : in std_logic_vector(3 downto 0);
    data : inout std_logic_vector(7 downto 0)
  );
end test6_top;

architecture RTL of test6_top is
begin
  REG0 <= VERSION;
  REG1 <= REVISION;
  chip_sel <= NOT csn;
  rd_en <= NOT oen;
```

rd_strb <= chip_sel AND rd_en;
wr_en <= NOT wrn;
wr_strb <= chip_sel AND wr_en;

latch_rd :Latch port map (  
  RST => rst,
  EN => '1',
  CLK => clk,
  LAT_IN => rd_strb,
  LAT_OUT => rd_strb_latched
);

latch_wr :Latch port map (  
  RST => rst,
  EN => '1',
  CLK => clk,
  LAT_IN => wr_strb,
  LAT_OUT => wr_strb_latched
);

latch_addr :Latch n_bit  
generic map(num_bits => 4)  
port map (  
  RST => rst,
  EN => '1',
  CLK => clk,
  LAT_IN => addr,
  LAT_OUT => addr_latched
);

latch_datain :Latch n_bit  
generic map(num_bits => 8)  
port map (  
  RST => rst,
  EN => wr_strb,
  CLK => clk,
  LAT_IN => data,
  LAT_OUT => datain
);

inst_cpu_read_reg_read : reg_read port map (  
  rst => rst,
  strb_in => rd_strb_latched,
  addr_in => addr_latched,
  data_out => dataout
);

data <= dataout;

four_bank_sel_decode_2_to_4 : decode_2_to_4 port map (  
  rst => rst,
  strb_in => wr_strb_latched,
  in_2bit => addr_latched(3 downto 2),
  strb_out => reg_bank_sel
);

first_bank_strobes_decode_2_to_4 : decode_2_to_4 port map (  
  rst => rst,
  strb_in => reg_bank_sel(0),
  in_2bit => addr_latched(1 downto 0),
  strb_out => reg_sel_strb(3 downto 0)
);

second_bank_strobes_decode_2_to_4 : decode_2_to_4 port map (  
  rst => rst,
  strb_in => reg_bank_sel(1),
in_2bit => addr_latched(1 downto 0),
strb_out => reg_sel_strb(7 downto 4)
);
third_bank_strobes_decode_2_to_4 : decode_2_to_4 port map (  
rst => rst,
strb_in => reg_bank_sel(2),
in_2bit => addr_latched(1 downto 0),
strb_out => reg_sel_strb(11 downto 8)
);
fourth_bank_strobes_decode_2_to_4 : decode_2_to_4 port map (  
rst => rst,
strb_in => reg_bank_sel(3),
in_2bit => addr_latched(1 downto 0),
strb_out => reg_sel_strb(15 downto 12)
);
data_write_reg2 : LATCH_n_bit
generic map(num_bits => 8)
port map (  
RST => rst,
EN => reg_sel_strb(2),
CLK => clk,
LAT_IN => datain,
LAT_OUT => REG2
);
data_write_reg3 : LATCH_n_bit
generic map(num_bits => 8)
port map (  
RST => rst,
EN => reg_sel_strb(3),
CLK => clk,
LAT_IN => datain,
LAT_OUT => REG3
);
data_write_reg4 : LATCH_n_bit
generic map(num_bits => 8)
port map (  
RST => rst,
EN => reg_sel_strb(4),
CLK => clk,
LAT_IN => datain,
LAT_OUT => REG4
);
data_write_reg5 : LATCH_n_bit
generic map(num_bits => 8)
port map (  
RST => rst,
EN => reg_sel_strb(5),
CLK => clk,
LAT_IN => datain,
LAT_OUT => REG5
);
data_write_reg6 : LATCH_n_bit
generic map(num_bits => 8)
port map (  
RST => rst,
EN => reg_sel_strb(6),
CLK => clk,
LAT_IN => datain,
LAT_OUT => REG6
);

data_write_reg7 : LATCH_n_bit
generic map(num_bits => 8)
port map ( 
  RST => rst,
  EN => reg_sel_strb(7),
  CLK => clk,
  LAT_IN => datain,
  LAT_OUT => REG7
);

data_write_reg8 : LATCH_n_bit
generic map(num_bits => 8)
port map ( 
  RST => rst,
  EN => reg_sel_strb(8),
  CLK => clk,
  LAT_IN => datain,
  LAT_OUT => REG8
);

data_write_reg9 : LATCH_n_bit
generic map(num_bits => 8)
port map ( 
  RST => rst,
  EN => reg_sel_strb(9),
  CLK => clk,
  LAT_IN => datain,
  LAT_OUT => REG9
);

data_write_reg10 : LATCH_n_bit
generic map(num_bits => 8)
port map ( 
  RST => rst,
  EN => reg_sel_strb(10),
  CLK => clk,
  LAT_IN => datain,
  LAT_OUT => REG10
);

data_write_reg11 : LATCH_n_bit
generic map(num_bits => 8)
port map ( 
  RST => rst,
  EN => reg_sel_strb(11),
  CLK => clk,
  LAT_IN => datain,
  LAT_OUT => REG11
);

data_write_reg12 : LATCH_n_bit
generic map(num_bits => 8)
port map ( 
  RST => rst,
  EN => reg_sel_strb(12),
  CLK => clk,
LAT_IN => datain,
LAT_OUT => REG12
);

data_write_reg13 : LATCH_n_bit
  generic map(num_bits => 8)
  port map (  
    RST => rst,
    EN => reg_sel_strb(13),
    CLK => clk,
    LAT_IN => datain,
    LAT_OUT => REG13
  );

data_write_reg14 : LATCH_n_bit
  generic map(num_bits => 8)
  port map (  
    RST => rst,
    EN => reg_sel_strb(14),
    CLK => clk,
    LAT_IN => datain,
    LAT_OUT => REG14
  );

data_write_reg15 : LATCH_n_bit
  generic map(num_bits => 8)
  port map (  
    RST => rst,
    EN => reg_sel_strb(15),
    CLK => clk,
    LAT_IN => datain,
    LAT_OUT => REG15
  );
end RTL;

From the global variables and user-defined functions, package file is generated as below.

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
package globals is
  -- Constants and global signals declaration
  constant VERSION : std_logic_vector(7 downto 0) := x"10";
  constant REVISION : std_logic_vector(7 downto 0) := x"20";
  signal REG0 : std_logic_vector(7 downto 0);
  signal REG1 : std_logic_vector(7 downto 0);
  signal REG2 : std_logic_vector(7 downto 0);
  signal REG3 : std_logic_vector(7 downto 0);
-- Components declaration
component decode_2_to_4 is
port (    rst : in  std_logic;
        strb_in : in  std_logic;
        in_2bit : in std_logic_vector(1 downto 0);
        strb_out : out std_logic_vector(3 downto 0)
    );
end component decode_2_to_4;

component reg_read is
port (    rst : in  std_logic;
         strb_in : in  std_logic;
         addr_in : in std_logic_vector(3 downto 0);
         data_out : out std_logic_vector(7 downto 0)
    );
end component reg_read;

end globals;

-- Entities and architectures of the components
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY decode_2_to_4 is
port (    rst : in  std_logic;
         strb_in : in  std_logic;
         in_2bit : in std_logic_vector(1 downto 0);
         strb_out : out std_logic_vector(3 downto 0)
    );
end decode_2_to_4;

architecture RTL of decode_2_to_4 is
begin
    proc_rst_cond : process (all)
      begin
        if rst = '1' then
            strb_out <= x"0"
        else
            if in_2bit = "00" then

Page 38 of 67
strb_out(0) <= strb_in;
strb_out(3 downto 1) <= "000";
else
  if in_2bit = "01" then
    strb_out(0) <= '0';
    strb_out(1) <= strb_in;
    strb_out(3 downto 2) <= "00";
  else
    if in_2bit = "10" then
      strb_out(1 downto 0) <= "00";
      strb_out(2) <= strb_in;
      strb_out(3) <= '0';
    else
      strb_out(2 downto 0) <= "000";
      strb_out(3) <= strb_in;
    end if;
  end if;
end if;
end if;
end process;

end RTL;

configuration CFG_decode_2_to_4 of decode_2_to_4 is
  for RTL
  end for;
end CFG_decode_2_to_4;

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY reg_read
  is
  port(
    rst : in  std_logic;
    strb_in : in  std_logic;
    addr_in : in std_logic_vector(3 downto 0);
    data_out : out std_logic_vector(7 downto 0)
  );
end reg_read;

architecture RTL of reg_read is
begin
  signal bank_en : std_logic_vector(3 downto 0);
  bank_sel_decode_2_to_4 : decode_2_to_4 port map (
    rst => rst,
    strb_in => strb_in,
    in_2bit => addr_in(3 downto 2),
    strb_out => bank_en
  );

  first_bank : MUX_n_bit
  generic map(num_bits => 8)
  port map(
    MUX_OUT => data_out,
    EN => bank_en(0),
    OUT_SEL => addr_in(1 downto 0),
    MUX_IN0 => REG0,
MUX_IN1 => REG1,
MUX_IN2 => REG2,
MUX_IN3 => REG3
);

second_bank : MUX_n_bit
generic map(num_bits => 8)
port map (  
MUX_OUT => data_out,
EN => bank_en(1),
OUT_SEL => addr_in(1 downto 0),
MUX_IN0 => REG4,
MUX_IN1 => REG5,
MUX_IN2 => REG6,
MUX_IN3 => REG7
);

third_bank : MUX_n_bit
generic map(num_bits => 8)
port map (  
MUX_OUT => data_out,
EN => bank_en(2),
OUT_SEL => addr_in(1 downto 0),
MUX_IN0 => REG8,
MUX_IN1 => REG9,
MUX_IN2 => REG10,
MUX_IN3 => REG11
);

fourth_bank : MUX_n_bit
generic map(num_bits => 8)
port map (  
MUX_OUT => data_out,
EN => bank_en(3),
OUT_SEL => addr_in(1 downto 0),
MUX_IN0 => REG12,
MUX_IN1 => REG13,
MUX_IN2 => REG14,
MUX_IN3 => REG15
);

end RTL;

configuration CFG_reg_read of reg_read is
  for RTL
end for;
end CFG_reg_read;

This file is the test bench template which is followed by the edited test bench program.
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY test6_top_tb is
end ENTITY test6_top_tb;

ARCHITECTURE behavioral of test6_top_tb is

COMPONENT test6_top is
port (  
  clk : in  std_logic;
  rst : in  std_logic;
  csn : in  std_logic;
  oen : in  std_logic;
  wrn : in  std_logic;
  addr : in std_logic_vector(3 downto 0);
  data : inout std_logic_vector(7 downto 0));
end COMPONENT;

signal clk :  std_logic;
signal rst :  std_logic;
signal csn :  std_logic;
signal oen :  std_logic;
signal wrn :  std_logic;
signal addr : std_logic_vector(3 downto 0);
signal data : std_logic_vector(7 downto 0);

begin
inst_test6_top : test6_top
  port map(
    clk => clk,
    rst => rst,
    csn => csn,
    oen => oen,
    wrn => wrn,
    addr => addr,
    data => data
  );
end;

Filename:  test6_top_tb.vhd
-------------
-- COMS W4115 PLT - 2014 Spring Term Project
--  HL-HDL compiler
--  Woon Lee (wgl2107)
-------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.globals.all;
use work.lib_comp.all;

ENTITY test6_top_tb is
end ENTITY test6_top_tb;

ARCHITECTURE behavioral of test6_top_tb is

procedure RegRd ( 
    signal cs_n : out std_logic;
    signal read_n : out std_logic;
    signal write_n : out std_logic;
    signal reg_addr : in std_logic_vector(3 downto 0);
    signal addr_out : out std_logic_vector(3 downto 0)
) is
begin
    cs_n <= '1';
    read_n <= '1';
    write_n <= '1';
    wait for 30 ns;
    addr_out <= reg_addr;
    wait for 5 ns;
    cs_n <= '0';
    wait for 5 ns;
    read_n <= '0';
    -- read access
    wait for 60 ns;
    read_n <= '1';
    wait for 5 ns;
    cs_n <= '1';
    wait for 20 ns;
end procedure;

procedure RegWr ( 
    signal cs_n : out std_logic;
    signal read_n : out std_logic;
    signal write_n : out std_logic;
    signal reg_addr : in std_logic_vector(3 downto 0);
    signal addr_out : out std_logic_vector(3 downto 0);
    signal data_in : in std_logic_vector(7 downto 0);
    signal data_out : out std_logic_vector(7 downto 0)
) is
begin
    cs_n <= '1';
    read_n <= '1';
    write_n <= '1';
    wait for 30 ns;
    addr_out <= reg_addr;
    wait for 5 ns;
    cs_n <= '0';
    data_out <= data_in;
    wait for 5 ns;
    write_n <= '0';
    -- read access
wait for 60 ns;
write_n <= '1';
wait for 5 ns;
cs_n <= '1';
data_out <= (others => 'Z');
wait for 20 ns;
end procedure;

COMPONENT test6_top is
port (
    clk : in  std_logic;
    rst : in  std_logic;
    csn : in  std_logic;
    oen : in  std_logic;
    wrn : in  std_logic;
    addr : in std_logic_vector(3 downto 0);
    data : inout std_logic_vector(7 downto 0)
);)
end COMPONENT;

signal clk :  std_logic;
signal rst :  std_logic;
signal csn :  std_logic;
signal oen :  std_logic;
signal wrn :  std_logic;
signal addr : std_logic_vector(3 downto 0);
signal data : std_logic_vector(7 downto 0);
signal register_addr : std_logic_vector(3 downto 0);
signal wr_value : std_logic_vector(7 downto 0);
signal sim_done : boolean := false;

begin
inst_test6_top : test6_top
port map(
    clk => clk,
    rst => rst,
    csn => csn,
    oen => oen,
    wrn => wrn,
    addr => addr,
    data => data
);

clk_proc : process
begin
    if (not sim_done) then
        clk <= '0';
        wait for 10 ns;
        clk <= '1';
        wait for 10 ns;
    else
        report "sim_done";
        wait; --wait forever
    end if;
end process clk_proc;

reg_rd_wr_seq : process
begin
    data <= (others => 'Z');
    wait for 30 ns;
end process reg_rd_wr_seq;
-- check version and revision
register_addr <= x"0"; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"1"; RegRd(csn, oen, wrn, register_addr, addr);
-- write all 14 registers with unique values, respectively
register_addr <= x"2"; wr_value <= x"22"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"3"; wr_value <= x"33"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"4"; wr_value <= x"44"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"5"; wr_value <= x"55"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"6"; wr_value <= x"66"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"7"; wr_value <= x"77"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"8"; wr_value <= x"88"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"9"; wr_value <= x"99"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"A"; wr_value <= x"AA"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"B"; wr_value <= x"BB"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"C"; wr_value <= x"CC"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"D"; wr_value <= x"DD"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"E"; wr_value <= x"EE"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"F"; wr_value <= x"FF"; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
-- read back register values and verify
register_addr <= x"0"; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"1" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"2" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"3" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"4" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"5" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"6" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"7" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"8" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"9" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"A" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"B" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"C" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"D" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"E" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"F" ; RegRd(csn, oen, wrn, register_addr, addr);
-- write different values to selected registers
register_addr <= x"2" ; wr_value <= x"E8" ; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"7" ; wr_value <= x"5A" ; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
register_addr <= x"C" ; wr_value <= x"A5" ; RegWr(csn, oen, wrn, register_addr, addr, wr_value, data);
-- read back and verify
register_addr <= x"2" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"7" ; RegRd(csn, oen, wrn, register_addr, addr);
register_addr <= x"C" ; RegRd(csn, oen, wrn, register_addr, addr);
wait;
end process;

sim_time : process
begin
  rst <= '1';
  wait for 10 ns;
  rst <= '0';
  wait for 5000 ns;
  sim_done <= true;
  wait;
end process;

Finally, the waveform was reported by the simulator.
### 6.3 Test Suites

The following table shows the test suites with description of the test target.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Input file</th>
<th>Output files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>test1.hhl</td>
<td>test1_top.vhd  test1_defs.vhd  test1_top_tbframe.vhd</td>
<td>Simple built-in function and bit operator test: - buf, inv, and, or</td>
</tr>
<tr>
<td>2</td>
<td>test2.hhl</td>
<td>test2_top.vhd  test2_defs.vhd  test2_top_tbframe.vhd</td>
<td>Two built-in function test: - lat, mux for single-bit and multiple-bit signals</td>
</tr>
<tr>
<td>3</td>
<td>test3.hhl</td>
<td>test3_top.vhd  test3_defs.vhd  test3_top_tbframe.vhd</td>
<td>More complex main function: - implementing 8-to-1 mux using built-in 4-to-1 mux - if statement with comparison operator tested.</td>
</tr>
<tr>
<td>4</td>
<td>test4.hhl</td>
<td>test4_top.vhd  test4_defs.vhd  test4_top_tbframe.vhd</td>
<td>Global constant test based on test case 3. See section 6.2.1 for more detail.</td>
</tr>
<tr>
<td>5</td>
<td>test5.hhl</td>
<td>test5_top.vhd  test5_defs.vhd  test5_top_tbframe.vhd</td>
<td>User-defined function implementation test: - One user function declared and called up in the main function - The corresponding component declaration and entity are added to test5_defs.vhd</td>
</tr>
<tr>
<td>6</td>
<td>test6.hhl</td>
<td>test6_top.vhd  test6_defs.vhd  test6_top_tbframe.vhd</td>
<td>This covers almost every aspect of the language by implementing global variables/constants and multiple user-defined functions. See section 6.2.2 for more detail.</td>
</tr>
</tbody>
</table>
7. Lessons Learned

7.1 Importance of specification
As an experienced VHDL designer, it was thought to be easier to design a language that translates a source program into VHDL format. It led me to not pay attention to what can be or cannot be implemented, which made me modify the language specification back and forth as the compiler design phase passed. Modification of the specification is not always bad, but if it requires structural changes, it cost much more than expected. It was a good lesson that clear definition and specification at early design phase is most important task in design process.

7.2 Understanding of the Implementation Language
As a novice on OCAML, I was amazed to know how great this language is in terms of data type handling, concise grammar even though it was hard to understand at the beginning, and overall language structure. As aforementioned, I had to spend much time on getting the OCAML concept and its unique way of realization through the example codes. Once I got used to it, it was a lot easier to use the language features in implementation. If I had more understanding on it, it would have taken less time for this language design and allowed me to add more complicated features.

7.3 Time Management
This would the most expressed cliché, “start earlier as possible and spend more time on verification”, that hit me once again. Struggling with implementation, debugging took most of the project time and therefore there were less time for tests. I could have worked on wider test suites and more automated design process.
8. References


9. Appendix

< scanner.mll >

```mll
(- --------------------------------------------------------------
  COMPS W4115 PLT - 2014 Spring Term Project
  ML-MDL compiler: scanner.mll
  Name: Woon Lee
  UNI: wg12107
  ---------------------------------------------------------------)

  ( open Parser )

  let letter = ['a'..'z' 'A'..'Z']
  let digit = ['0'..'9']
  let bidigit = ['0'..'9']
  let hexdigit = ['0'..'9' 'a'..'f' 'A'..'F']

  rule token =
    parse ['/ ' '
    | ' 
    | 't' ''
    | EOF } { token lexbuf }
    | "insig" { INSIG } | "outsig" { OUTSIG } |
    | "iosig" { IOSIG } | "constant" { CONST } |
    | "signal" { SIGNAL } | "and" { AND } |
    | "or" { OR } | "inv" { INV } |
    | "buf" { BUF } | "lat" { LAT } |
    | "mux" { MUX } |
    | "func" { FUNC } | "if" { IF } |
    | "then" { THEN } | "else" { ELSE } |
    | "main" { MAIN } | \
    | | ' ' { COMMA } |
    | '(' { LPAREN } | ')' { RPAREN } |
    | '{' { LBRACE } | '}' { RBRACE } |
    | ':' { SHMI } | \
    | | ' ' { COLON } |
    | letter ( letter | digit | "+") as id { ID(id) } |
    | digit+ as lit { NUM(int_of_string lit) } |
    | "0b" bidigit+ as bdigit { BNUM(bdigit) } |
    | "0x" hexdigit+ as hdigit { HNUM(hdigit) } |
    | "--" { EQ } | "!-" { NRQ } |
    | "=" { ASSIGN } | ":=" { IAASSIGN } |
    | "/\" { comment lexbuf } |
    | "(" { raise Failure("Unrecognized token: " ^ Lexing.lexeme lexbuf)) } |
  and comment =
    parse "*/" { token lexbuf }
    | _ { comment lexbuf }
```

---

Page 49 of 67
```ml
/* **************************************************************/
COMS W4115 PLT - 2014 Spring Term Project
ML-HDL compiler: parser.mly
   Name: Woon Lee
   UNI: wgl2107
**************************************************************/

% open Ast %

%token EOF INSIG OUTSIG IOSIG CONST SIGNAL AND OR INV BUF LAT MUX LBRACKET RBRACKET
%token FUNC IF THEN ELSE MAIN COMMA LPAREN RPAREN LBRACE RBRACE SEMI COLON ASSIGN IASSIGN
%token <string> ID RNUM HNUM
%token <int> INUM
%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left AND OR EQ NEQ

%start program
%type <Ast.program> program

%%

program: /* nothing */ { [], [] }
    | program vdecl { ($2 :: fst $1), snd $1 }
    | program fdecl { fst $1, ($2 :: snd $1) }

fdecl:
  fn_type ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE
  { (func_type  = $1;
     func_name  = $2;
     formals   = $4;
     locals     = list.rev $7;
     body       = list.rev $8) }

fn_type:
  FUNC { User }
  | MAIN { Main }

formals_opt:
  /* nothing */ { [] }
  | formal_list { list.rev $1 }

formal_list:
  formal_decl { [$1] }
  | formal_list COMMA formal_decl { $3 :: $1 }

formal_decl:
  formaltype ID bus_opt { [formal_type = $1; formal_name = $2; formal_range = $3] }

```

Page 50 of 67
formalyte:
  INSIG   | Insig |
  | OUTSIG  | Outsig |
  | Rcsig   | Rcsig |

bus_opt:
  /* nothing */    { (-1, -1) }
  | LBRACKET INUM COLON INUM RBRACKET { ($2, $4) }

vdecl_list:
  /* nothing */    { [] }
  | vdecl_list vdecl   { $2 :: $1 }

vdecl:
  var_decl SEMI    { $1 }

var_decl:
  vartype ID bus_opt init_value    { { var_type = $1; var_name = $2; var_range = $3; var_value = $4 } }

vartype:
  | CONST   | Const |
  | SIGNAL   | Signal |

init_value:
  /* nothing */    { "none" }
  | IASSIGN INUM   | string_of_int $2 }
  | IASSIGN BNUM   | $2 }
  | IASSIGN HNUM   | $2 }

stmt_list:
  /* nothing */    { [] }
  | stmt_list stmt   { $2 :: $1 }

stmt:
  expr SEMI    { Expr($1) }
  | LBRACE stmt_list RBRACE    { Block(List.rev $2) }
  | id_opt IF LPAREN expr RPAREN THEN stmt %prec NOELSE   { If ($1, $4, $7, block([[]])) }
  | id_opt IF LPAREN expr RPAREN THEN stmt ELSE stmt    { If ($1, $4, $7, $9) }
  | expr    { $error ($1) }

id_opt:
  /* nothing */    { "none" }
  | ID COLON    | $1 }

expr:
  | ID   | Id($1) }
  | INUM   | Lit_int($1) }
  | HNUM   | Lit_hex($1) }
  | BNUM   | Lit_bin($1) }
  | ID LBRACKET INUM COLON RBRACKET   { Single($1, $3) }
  | ID LBRACKET INUM COLON RBRACKET   { Range($1, $3, $5) }

-2-
```
105 | expr AND expr   { Binop($1, And, $3) }
106 | expr OR expr    { Binop($1, Or, $3) }
107 | expr EQ expr    { Binop($1, Equal, $3) }
108 | expr NEQ expr   { Binop($1, Nequal, $3) }
109 | assign_opt ASSIGN expr { Assign($1, $3) }
110 | ID COLON ID LPAREN actuals_opt RPAREN { Call($1, $3, $5) }
111 | INV LPAREN actuals_opt RPAREN { Bcall("", Inv, $3) }
112 | BUF LPAREN actuals_opt RPAREN { Bcall("", Buf, $3) }
113 | id_opt LAT LPAREN actuals_opt RPAREN { Bcall($1, Lat, $4) }
114 | id_opt MUX LPAREN actuals_opt RPAREN { Bcall($1, Mux, $4) }
115 | LPAREN expr RPAREN { Braces($2) }

116 assign_opt:
117    | ID           { Id($1) }
118    | ID LBRACKET INUM RBRACKET { Single($1, $3) }
119    | ID LBRACKET INUM COLON INUM RBRACKET { Range($1, $3, $5) }

121 actuals_opt:
122    /* nothing */ { [] }
123    | actuals_list { List.rev $1 }

126 actuals_list:
127    | expr         { [$1] }
128    | actuals_list COMMA expr { $3 :: $1 }
```
(***)

(* expressions *)

<table>
<thead>
<tr>
<th>Lit_int of int</th>
<th>Lit_hex of string</th>
<th>Lit_bin of string</th>
<th>Id of string</th>
<th>Binop of expr op expr</th>
<th>Single of string int</th>
<th>Range of string int int</th>
<th>Braces of expr</th>
<th>Assign of expr expr</th>
<th>Call of string string expr list</th>
<th>Bcall of string btin_f expr list</th>
</tr>
</thead>
<tbody>
<tr>
<td>(* 10 *)</td>
<td>(* x1010 *)</td>
<td>(* b1010 *)</td>
<td>(* foo *)</td>
<td>(* a AND b *)</td>
<td>(* a[2] *)</td>
<td>(* a[3:5] *)</td>
<td>(* (a OR b) *)</td>
<td>(* foo - x10ac *)</td>
<td>(* foo(a, b) *)</td>
<td>(* inv(a) *)</td>
</tr>
</tbody>
</table>

(* statements *)

<table>
<thead>
<tr>
<th>Block of stmt list</th>
<th>Expr of expr</th>
<th>If of string expr stmt stmt</th>
<th>StError of expr</th>
</tr>
</thead>
<tbody>
<tr>
<td>(* {...} *)</td>
<td>(* c = a AND b *)</td>
<td>(* if (a == b) then () else [] *)</td>
<td>(* missing :)</td>
</tr>
</tbody>
</table>

(* variable declaration *)

<table>
<thead>
<tr>
<th>var_type : v_type;</th>
<th>var_name : string;</th>
<th>var_range : int int;</th>
<th>var_value : string</th>
</tr>
</thead>
<tbody>
<tr>
<td>(* variable type *)</td>
<td>(* variable name *)</td>
<td>(* variable range for bus *)</td>
<td>(* variable init value *)</td>
</tr>
</tbody>
</table>

(* function declaration *)

<table>
<thead>
<tr>
<th>func_type : fun_type;</th>
<th>func_name : string;</th>
<th>formals : formal list;</th>
</tr>
</thead>
<tbody>
<tr>
<td>(* user-defined or main *)</td>
<td>(* name of the function *)</td>
<td>(* formal arguments *)</td>
</tr>
</tbody>
</table>
```ml
let rec string_of_expr = function
  | Lit_int(literal) -> string_of_int literal
  | Lit_hex(hex_lit) -> hex_lit
  | Lit_bin(bin_lit) -> bin_lit
  | Id(id) -> id
  | Single(id, index) -> id ^ "(" ^ (string_of_int index) ^ ")"
  | Range(id, first, last) -> id ^ "(" ^ (string_of_int first) ^ ") ; (" ^ (string_of_int last) ^ ")")
  | Braces(e) -> "(" ^ (string_of_expr e) ^ ")"
  | Binop(e1, oper, e2) -> (string_of_expr e1) ^ " " ^ (match oper with
    | And -> "AND"
    | Or -> "OK"
    | Equal -> "=="
    | Nequal -> "!=") ^ (string_of_expr e2)
  | Assign(s1, e2) -> (string_of_expr e1) ^ " = " ^ (string_of_expr e2)
  | Call(inst_id, f_id, e_list) -> inst_id ^ ": " ^ f_id ^ "(" ^ String.concat ", " (List.map string_of_expr e_list) ^ ")"
  | Bcall(inst_id, btif_id, e_list) -> (match btif_id with
    | Inv -> "INV"
    | Buf -> "BUF"
    | Lat -> inst_id ^ ": " ^ "LAT"
    | Mux -> inst_id ^ ": " ^ "MUX"
  ) ^ "(" ^ String.concat ", " (List.map string_of_expr e_list) ^ ")"

let rec string_of_stmt = function
  | Block(s) -> "\{\\n\}
  | Expr(s) -> "\t" ^ string_of_expr s ^ "\\n"
  | If(pid, e, s, Block([])) -> "\\n\{\\n```
  ```string_of_expr e ^ "\\n\}\\n```
  ```string_of_stmt s ^ "\\n```
  ```string_of_stmt s ^ "\\n```
  ```\n\}\\n```
  ```string_of_stmt s1
  | StmtError (o) -> "\t" ^ string_of_expr o ^ "\\n"

let string_of_v_type vtype = match vtype with Const -> "constant" | Signal -> "signal"
  | Insg -> "in" | Outsg -> "out" | Iosg -> "iout"

let string_of_btif btif = match btif with Inv -> "inv" | Buf -> "buf" | Lat -> "lat" | Mux -> "mux"
```
let string_of_f_type ftype = match ftype with
  | Insig  -> "in" | Outsigt  -> "out" |
  | IOsig   -> "inout" | *

let string_of_functype functype = match functype with
  | User    -> "User" | Main     -> "Main" |

let string_of_formal fn_param = string_of_v_type fn_param.formal_type ^ " " ^ fn_param.formal_name ^
  match fn_param.formal_range with
  | _      -> ""
  | _      -> "(" ^ string_of_int (fst fn_param.formal_range) ^ ":" ^ string_of_int (snd fn_param.formal_range) ^ ")")

let string_of_var_decl variable =
  (string_of_v_type variable.var_type) ^ " " ^ variable.var_name ^
  match variable.var_range with
  | _      -> ""
  | _      -> "(" ^ string_of_int (fst variable.var_range) ^ ":" ^ string_of_int (snd variable.var_range) ^ ")") ^
    match variable.var_value with
    | none   -> ""
    | _      -> " = " ^ variable.var_value ^ ";\n"

let string_of_func_decl func =
  (string_of_functype (func.func_type) ^ " " ^ func.func_name ^ "(" ^ String.concat ", " (List.map string_of_formal_func.formals) ^ ")" ^ String.concat "\n" ^ String.concat "\n" ^ List.map string_of_var_decl func.locals) ^
  "\n" ^ String.concat "\n" ^ String.concat (List.map string_of_stat func.body) ^ "\n"

let string_of_program (global_variables, func_declarations) =
  String.concat "" (List.map string_of_var_decl (List.rev global_variables)) ^ "\n" ^
  String.concat "\n" (List.map string_of_func_decl decls)
translate.ml

(* Symbol table: Information about all the names in scope *)
type env = {
  function_decl : string StringMap.t; (* function signature *)
  global_var : v_type StringMap.t; (* global variables name and types *)
  local_var : v_type StringMap.t; (* local & function arg variable names and types *)
}

let vhd_header_common = "library IEEE; \n" ^ 
  "use IEEE.STD_LOGIC_1164.all; \n" ^ 
  "use IEEE.STD_LOGIC_ARITH.ALL; \n" ^ 
  "use IEEE.STD_LOGIC_UNSIGNED.ALL; \n"

let vhd_header_opt = "use work.globals.all; \n" ^ 
  "use work.lib_comp.all; \n"

(* Extract formal info from function declaration *)
let string_of_formal formal = 
  "^ formal.formal_name ~ " ^ 
  "^ Ast.string_of_v_type formal.formal_type ~ " ^ 
  match (fst formal.formal_range) with
    | _ -> " std_logic" 
    | _ -> let first = fst formal.formal_range and last = snd formal.formal_range in 
      if (first >= last) then "std_logic_vector(" ^ string_of_int first ^ " downto " ^ string_of_int last ^ 
        " std_logic")" 
      else "std_logic_vector(" ^ string_of_int last ^ 
        " downto " ^ string_of_int first ^ 
        " std_logic")" 

(* Extract formal info from function declaration - for test bench frame generation *)
let string_of_formal_tb formal = 
  "^ formal.formal_name ~ " ^ 
  match (fst formal.formal_range) with
    | _ -> " std_logic" 
    | _ -> let first = fst formal.formal_range and last = snd formal.formal_range in 

```ml
if (first >= last) then "std_logic_vector(" ^ string_of_int first ^ 
  downto " ^ string_of_int last ^ ")"
else "std_logic_vector(" ^ string_of_int last ^ 
  downto " ^ string_of_int first ^ ")"

(* extract the name and range of variable *)
let string_of_variable_variable = "\" ^ Ast.string_of_v_type variable.var_type ^ " ^ variable.var_name ^ " : " ^
  (match (fst variable.var_range) with
    _  -> " std_logic"
  | _  -> let first = fst variable.var_range and last = snd variable.var_range in
    if (first >= last) then "std_logic_vector(" ^ string_of_int first ^ 
      downto " ^ string_of_int last ^ ")"
    else "std_logic_vector(" ^ string_of_int last ^ 
      downto " ^ string_of_int first ^ ")")
  )

(match variable.var_value with
  "none"       -> "(\n" |
  _            -> let value_type = String.sub variable.var_value 1 1 in
    match value_type with
      "k"       -> " := x\" ^ String.sub variable.var_value 2 ((String.length 
                      variable.var_value)-1) ^ "\n" |
      "h"       -> " := \" ^ String.sub variable.var_value 2 ((String.length 
                      variable.var_value)-1) ^ "\n" |
      _         -> " := " ^ variable.var_value ^ "\n" )

(* Extract variable type and name *)
let rec enum_vdecl = function
  []       -> []
  | hd::tl  -> (hd.var_type, hd.var_name)::enum_vdecl tl

(* Extract formal type and name *)
let rec enum_fdecl = function
  []       -> []
  | hd::tl  -> (hd.formal_type, hd.formal_name)::enum_fdecl tl

(* Extract function type and name *)
let rec enum_func = function
  []       -> []
  | hd::tl  -> (Ast.string_of_functype hd.func_type, hd.func_name)::enum_func tl

(* Extract function name with formals *)
let rec enum_f_formal = function
  []       -> []
  | hd::tl  -> (hd.formals, hd.func_name)::enum_f_formal tl

(* Extract variable name and width *)
let rec var_width = function
  []       -> []
  | hd::tl  -> let width = ((fun x -> if x < 0 then x else x) ((fst hd.var_range)-( 
                      snd hd.var_range)))+1 in
    ((string_of_int width, hd.var_name))::var_width tl

(* Extract formal name and width *)
```

let rec formal_width = function
[| [] -> []
    | hd::tl -> let width = ((fun x -> if x < 0 then -x else x) ((fst hd.formal_range)
    )-(snd hd.formal_range))::formal_width tl
]

(* val string_map_pairs StringMap 'a -> (int * 'a) list -> StringMap 'a *)
let string_map_pairs map pairs =
List.fold_left (fun m (i, n) -> StringMap.add n i m) map pairs

(* Translate the AST to vhdl code *)
let trans_to_vhdl (globals, functions) out_opt =

(* Allocate "addresses" for each global variable *)
(* (variable type, variable name) pair for global variables *)
let global_variables = string_map_pairs StringMap.empty (enum_vdecl globals) in
(* (variable type, variable name) pair for global variables *)
let global_var_widths = string_map_pairs StringMap.empty (var_width globals) in

(* (function type, function name) pairs *)
let function_decls = string_map_pairs StringMap.empty (enum_func functions) in
(* (function type, function name) pairs *)
let function_formals = string_map_pairs StringMap.empty (enum_f_formal functions) in

(* Translate a function in AST to a list of bytecode statements *)
let translate env func_decl =
(* (variable type, variable name) pair for local variables *)
let local_variables = enum_vdecl func_decl.locals
(* (variable type, variable name) pair for local variables *)
let local_var_widths = var_width func_decl.locals
(* (variable type, variable name) pair for formals *)
let formal_variables = enum_f decl.formals
(* (variable type, variable name) pair for formals *)
let formal_var_widths = formal_width func_decl.formals in

let env = { env with
    local_var = string_map_pairs StringMap.empty (local_variables @
        formal_variables) }
and variable_widths = string_map_pairs StringMap.empty (local_var_widths @
        formal_var_widths) in

(* Extract variable type for type checking *)
let exps_type =
match o with
| Id(id) -> if (StringMap.mem id global_variables) then
    string_of_v_type (StringMap.find id global_variables)
  else if (StringMap.mem id env.local_var) then
    string_of_v_type (StringMap.find id env.local_var)
  else ""
| Single (id, index) -> if (StringMap.mem id global_variables) then
    string_of_v_type (StringMap.find id global_variables)
  else if (StringMap.mem id env.local_var) then
    string_of_v_type (StringMap.find id env.

-3-
| 143 | local_var | else "" |
| 144 | | if (StringMap.mem id global_variables) then |
| 145 | | string_of_v_type (StringMap.find id global_variables) |
| 146 | | else if (StringMap.mem id env.local_var) then |
| 147 | | string_of_v_type (StringMap.find id env. local_var) |
| 148 | | else "" |
| 149 | _ | ""
| 150 | in |
| 151 | | evaluate an expression for type checking *) |
| 152 | let ree eval_expr e = |
| 153 | | match e with |
| 154 | | Lit_int(l) 0 |
| 155 | | Lit_hex(l) (String.length l 2) |
| 156 | | Lit_bin(l) (String.length l 2) |
| 157 | | Id(id) |
| 158 | | if (StringMap.mem id global_var_widths) |
| 159 | | then int_of_string (StringMap.find id global_var_widths) |
| 160 | | else if (StringMap.mem id variable_widths) |
| 161 | | then int_of_string (StringMap.find id variable_widths) |
| 162 | | else |
| 163 | | if (e1_length e2_length) |
| 164 | | then e1_length |
| 165 | | else |
| 166 | | if (e1_length 0) e2_length |
| 167 | | else if (e2_length 0) e1_length |
| 168 | | else raise (Failure ("type mismatch " Ast. string_of_expr e1 ":" Ast.string_of_expr e2)) |
| 169 | | |
| 170 | | Single (id, index) |
| 171 | | Call (inst_id, f_id, actuals) |
| 172 | | Bcall (inst_id, btif_id, actuals) |
| 173 | | Let (e1, e2) |
| 174 | | if (e2_length 0) then e1 else |
| 175 | | (if (e1_length e2_length) then e1 else |
| 176 | | |
| 177 | | Call (inst_id, f_id, actuals) 0 |
| 178 | | Bcall (inst_id, btif_id, actuals) |
| 179 | | Let (btif_name = Ast.string_of_btif btif_id) |
| 180 | | match btif_name with |
| 181 | | "Inv" |
| 182 | | "Buf" |
| 183 | | "Lat" |
| 184 | | |
| 185 | | |

Page 59 of 67
(* Translate an expression *)

let rec expr e =
  match e with
  | Lit_int(l)  -> string_of_int l
  | Lit_hex(l)  -> let bh_val = String.sub l 2 (String.length l - 2) in
    "\"" ^ bh_val ^ "\""
  | Lit_bin(l)  -> let bh_val = String.sub l 2 ((String.length l) - 2) in
    if (String.length bh_val == 1) then "\"" ^ bh_val ^ "\""
    else "\"" ^ bh_val ^ "\""
  | Id(id)     -> if ((StringMap.mem id env.local_var) || (StringMap.mem id env.global_var))
    then id else raise (Failure ("undeclared variable: " ^ id))
  | Binop (el, oper, e2) ->
    expr el ^ (match oper with
    And -> " AND "
    | Or -> " OR "
    | Equal -> " == "
    | Nequal-> " /= " ^ (expr e2)
  | Single (id, index) -> id ^ "(" ^ string_of_int index ^ "")"
  | Range (id, first, last) -> id ^ (if (first >= last) then "(" ^
    string_of_int first ^ " downto " ^ string_of_int last ^ ")" ^
    else "(" ^ string_of_int last ^ " downto " ^
    string_of_int first ^ ")")
  | Braces (e)    -> "(" ^ expr e ^ ")"
  | Assign (el, e2) -> (match expr_type el with
    | "constant" -> raise (Failure ("illegal assignment to a constant: " ^
      Ast.string_of_expr el ^ ") \\
      Ast.string_of_expr e2))
    | _ -> expr el ^ " <= " ^ expr e2
  | Call (inst_id, f_id, actuals) -> if not (StringMap.mem f_id env.
    function_decl) then
    raise (Failure ("undefined function: " ^
      f_id))
    else
      let f_formal = (StringMap.find f_id
        function_formals) in
      if ((List.length f_formal) != (List.length actuals))
        then raise (Failure ("number of function call argument mismatch: " ^
          f_id))
      else (match inst_id with
        | "none" -> raise (Failure ("Function instantiation id required: " ^
            f_id))
        | _ -> inst_id ^ " \\
          f_id -> " ^
            (String.concat ",\n             " ^
            List.map (fun x -> x ^ ")
            List.map (fun x -> x."


formal_name" => ") f_formal) {
  list.map expr actuals)))"n(t)"
}

| Bcall (inst_id, btif_id, actuals) ->
  let btif_name = Ast.string_of_btif btif_id in
  match btif_name with
  | "Inv" -> if ((List.length actuals) != 1) then raise (Failure ("Function takes ONE argument: " ^ btif_name))
  | "NOT" -> if ((List.length actuals) != 1) then raise (Failure ("Function takes ONE argument: " ^ btif_name))
  | "Buf" -> if ((List.length actuals) != 1) then raise (Failure ("Function takes ONE argument: " ^ btif_name))
  | "Lat" -> if ((List.length actuals) != 5) then raise (Failure ("Function takes FIVE arguments: " ^ btif_name))
  else (let data_width = eval_expr (List.hd (List.rev actuals)) in
    let fcall_name = (if (data_width > 1) then
      "LATCH\tbit\n\tgeneric map(num_bits => " ^ string_of_int data_width ^ 
      "int\tport map (\n\tt")
    else ("LATCH\tport map (\n\tt")
    match inst_id with
    | "none" -> raise (Failure ("Function instantiation id required: Lat"))
    | _ -> ("L\t\"inst_id"fcall_name ^
      String.concat (\n\tt" (List.map2 (fun x y -> x^y) lat_port (List.map expr actuals)))"n\t")
  )
  |
  | "Mux" -> if ((List.length actuals) != 7) then raise (Failure ("Function takes SEVEN arguments: " ^ btif_name))
  else (let data_width = eval_expr (List.hd (List.rev actuals)) in
    let fcall_name = (if (data_width > 1) then
      "MUX\n\tgeneric map(num_bits => " ^ string_of_int data_width ^ 
      "int\tport map (\n\tt")
    else ("MUX\tport map (\n\tt")
    match inst_id with
    | "none" -> raise (Failure ("Function instantiation id required: Lat"))
    | _ -> ("\t\"inst_id"fcall_name ^
      String.concat (\n\tt" (List.map2 (fun x y -> x^y) mux_port (List.map expr actuals))^"n\t")
  )
  )
let rec stmt = function
  | Block(s) -> String.concat "" (List.map stmt s)
  | Expr(e) -> if (eval_expr e < 0) then
    raise (Failure ("Type mismatch in expression: " ^ Ast.string_of_expr e))
    else "\t" ^ expr ^ " ;\n"
  | If (pid, c, s, Block([])) -> (match pid with
      "none" -> "\tif " ^ expr ^ " then\n" ^ stmt s ^ "\nend if;\n"
    | _ -> "\n:\nproc_" ^ pid ^ " : process (all);\n" ^ "\nbegin\n\tif " ^ expr ^ " then\n" ^ stmt s ^ "\nend if;\n\ntend process;\n\n"
  )
  | If (pid, c, s1, s2) -> (match pid with
      "none" -> "\tif " ^ expr ^ " then\n" ^ stmt s1 ^ "\nend if;\n"
    | _ -> "\n:\nproc_" ^ pid ^ " : process (all);\n" ^ "\nbegin\n\tif " ^ expr ^ " then\n" ^ stmt s1 ^ "\ntelse\n" ^ stmt s2 ^ "\nend if;\n\ntend process;\n\n"
  )
  | StError(a) -> raise (Failure ("Syntax Error: semi-colon(;) is expected : " ^ Ast.string_of_expr e))

match out_opt with
  (* translate user-defined function declarations to component declarations in package file *)
  "pkg_decl" -> (match (Ast.string_of_functype func_decl.functype) with
    "User" -> "component " ^ func_decl.functype ^ " is\n" ^ "\nport \n" ^ String.concat "" (List.map string_of_formal func_decl.formals) ^ "\n\nend component " ^ func_decl.functype ^ " ;\n"
    | _ -> ""
  )
  (* translate user-defined functions into package body - ENTITY and ARCHITECTURE *)
  "pkg_body" -> (match (Ast.string_of_functype func_decl.functype) with
    "User" ->

translate.ml

297     vhdl_header_common ^ vhdl_header_opt ^ "\n\n" ^
298     "ENTITY " ^ func_decl.func_name ^ " is\n" ^ "port (\n" ^ String.
299     concat "\n"
300     (List.map string_of_formal func_decl.formals) ^ "(\n)\nend " ^
301     func_decl.func_name ^
302     "\n" ^ "architecture RTL of " ^ func_decl.func_name ^ " is\n" ^
303     String.concat "\n"
304     (List.map string_of_variable func_decl.locals) ^ "begin\n" ^
305     String.concat "(list.map stmt func_decl.body) ^ "end RTL;\n"
306     "\n"
307     "\n
(* translate main functions into main VHDL *)
308     "main" -> {
309     match (Ast.string_of_functype func_decl.func_type) with
310     "Main " ->
311     "ENTITY " ^ func_decl.func_name ^ "_top is\n" ^
312     String.concat "\n"
313     (List.map string_of_formal func_decl.formals) ^ "(\n)\nend " ^
314     func_decl.func_name ^
315     "_top is\n" ^ "architecture RTL of " ^ func_decl.func_name ^
316     "_top is\n" ^ String.concat "\n"
317     (List.map string_of_variable func_decl.locals) ^ "begin\n" ^
318     String.concat "(list.map stmt func_decl.body) ^ "end RTL;\n"
319     "\n"
320     "\n
(* translate main functions into main VHDL *)
321     "tbench" -> {
322     match (Ast.string_of_functype func_decl.func_type) with
323     "Main " ->
324     "ENTITY " ^ func_decl.func_name ^ "_top_tb is\n" ^ "end
325     "ARCHITECTURE behavioral of " ^ func_decl.func_name ^ ",_top_tb
326     is\n" ^
327     "COMPONENT " ^ func_decl.func_name ^ ",_top is\n" ^ "port (\n" ^
328     String.concat "\n"
329     (List.map string_of_formal func_decl.formals) ^ "(\n)\nend
330     "\n"
331     "\n
(* translate main functions into main VHDL *)
332     "\n
in let env = {
333     "\n
-{


```
function_decls = function_DECLS;

local_var = StringMap.empty in

(* Code executed to start the program *)

let _ = try
    (StringMap.mem "main" function_decls)

with Not_found -> raise (Failure ("no \"main\" function"))

(* Compile the functions *)

in

  (match out_opt with
   (* translate global variables to variable declarations in package file *)
   "const" -> file_header ^vhdl_header_common^ "\npackage globals is
"^ "\n-- Constants and global signals declaration
" ^
    (String.concat "" (List.map string_of_variable (List.rev
        globals))) ^ "\n"

| "pkg_decl" -> "-- Components declaration\n" ^ (String.concat "\n" (List.
    map (translate env) (List.rev functions)))

| "pkg_body" -> "\n\n-- Entities and architectures of the components\n" ^
    (String.concat "\n" (List.map (translate env) (List.rev
        functions))) ^ "\n"

| "main" -> file_header ^vhdl_header_common^ vhdl_header_opt ^ "\n\n" ^
    (String.concat "\n" (List.map (translate env) (List.rev
        functions))) ^ "\n"

| "tbench" -> file_header ^vhdl_header_common^ vhdl_header_opt ^ "\n\n" ^
    (String.concat "\n" (List.map (translate env) (List.rev
        functions))) ^ "\n"

| _ -> raise (Failure ("Input file is empty"))

```

Page 64 of 67
< hhlml.ml >

(* *****************************************************
 ** COMS W4115 PLT - 2014 Spring Term Project         
 ** HL-MDL compiler: hhlml.ml                        
 ** Name: Woon Lee                                  
 ** UNI: wq12107                                   
 *****************************************************)

open Printf

type action = Ast | Compile | TestBench | Error

let out_name = ref "str"

let get_filename arg = if (String.contains arg ".") then String.sub arg 0 (String.index arg ".")
                        else arg

let fwrite name strings =
  let out = open out name in
  fprintf out "\n\n" strings;
  close out

let usage_string = "Usage: HHLML COMPILER\n\n  \"\t AST pretty printer: hhlml -a <filename>\n\n  \"\t Compile to VHDL: hhlml -c <filename>\n\n  \"\t Test Bench framework: hhlml -t <filename>\n"

let _ =
  let action = if (Array.length Sys.argv > 2) then
    try
      List.assoc Sys.argv.(2) [ ("-a", Ast);
                              ("-c", Compile);
                              ("-t", TestBench) ]
    with
      _ -> Error
  else Error in
  let lexbuf = ignore(out_name := get_filename Sys.argv.(3));
  Lexing.from_channel (open in Sys.argv.(3)) in
  let program = Parser.program Scanner.token lexbuf in
  match action with
    Ast -> let listing = Ast.string_of_program program in
            fwrite ("./" ^ !out_name ^ ".ast") listing
    | Compile ->
      let const_listing = (Translate.trans_to_vhdl program) "const"
      and pkg_decls_listing = (Translate.trans_to_vhdl program) "pkg_decls"
      and pkg_body_listing = (Translate.trans_to_vhdl program) "pkg_body"
      and main_listing = (Translate.trans_to_vhdl program) "main"
      in
      fwrite ("./" ^ !out_name ^ ".defs.vhd") (const_listing ^
                                             pkg_decls_listing ^ pkg_body_listing);

-
fwrite ("./" ^ !out_name ^ ",_top.vhd") main_listing

| TestBench ->
| let tb_frame = (Translate.trans_to_vhdl program) "tbench" in
| fwrite ("./" ^ !out_name ^ ",_top_tbframe.vhd") tb_frame

| Error -> print_string usage_string
< Makefile >

```makefile
OBJS = scanner.cmo ast.cmo parser.cmo translate.cmo hihdl.cmo

TARFILES = Makefile scanner.ml ast.ml parser.mly translate.ml hihdl.ml

hihdl : $(OBJS)
    ocamlc -o hihdl $(OBJS)

parser.ml parser.mli: parser.mly
    ocamlyacc -v parser.mly

scanner.ml : scanner.ml
    ocamllex scanner.ml

%.cmo : %.ml
    ocamlc -c $<

%.cmi : %.mli
    ocamlc -c $<

.PHONY : clean

clean :
    rm -f scanner.ml parser.ml parser.mli *.cmo *.cml *.out *.diff

# Generated by ocamldep *.ml *.mli
ast.cmo:
    ast.cmx:
parser.cmo: ast.cmo parser.cmi
parser.cmx: ast.cmx parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmo
translate.cmo: ast.cmo
translate.cmx: ast.cmx
hihdl.cmo: scanner.cmo parser.cmi ast.cmo translate.cmo
hihdl.cmx: scanner.cmx parser.cmx ast.cmx translate.cmx

```

---

Page 67 of 67