Introductory Digital Logic Design and Bluespec

XingYing Cheng[1], Daren Wilcox[2]

Abstract

Most digital logic courses in engineering curriculums introduce hardware design using VHDL. The three primary levels of abstraction in VHDL taught in the introductory course follow closely to the traditional steps of Boolean logic, namely structure, data flow, and behavior. However, presenting the introductory material at a higher level of abstraction might just be as beneficial. Recently, a new electronic design automation tool, Bluespec, has emerged that promises a high level of abstraction for digital hardware design. In this paper, Bluespec will be introduced in relation to current VHDL design concepts with the intent of determining its applicability to introductory digital logic instruction.

Keywords: Bluespec, VHDL, digital design, Register Transfer Logic

Introduction

The instruction of digital logic in engineering curriculums has progressed from discrete logic gates in the 7400 series TTL to simple programmable logic devices (SPLD) to complex programmable logic devices (CPLD) to field programmable gate arrays (FPGA). Along with that progression has been advancement from manual programming of PLDs to computer programming using CUPL, then ABEL, then Verilog, and now VHDL. As the power of the hardware description language (HDL) advanced, so did the level of abstraction. At first, the code resembled the structure of the older discrete hardware systems. Then Boolen equations could be rendered in logic. Stepping higher, the state diagram or state sequence could be the entry method for finite state machines (FSM). As the state machines became more advanced and share control of the same resources, arbitration rules or arbiter circuits with grants and requests became necessary. Finite state machines yield sequential circuits in parallel hardware. The more advanced state machines with shared resources attempt to create parallel circuits. Bluespec from the company Bluespec, Inc., is a new hardware description language the promises to incorporate easily arbitration rules to design true parallel processing systems. Bluespec offers a higher level of abstraction for digital hardware design than Verilog and VHDL. In this paper, an overview of Bluespec is presented, its history and relation to VHDL, along with a simple example of code relating VHDL to Bluespec, followed by the operation of Bluespec in Linux.

Bluespec History

The Bluespec language was the development of Prof. Arvind who founded the semiconductor tool design company, Bluespec Inc., in 2003. Bluespec is a high-level functional hardware

---

1 Southern Polytechnic State University, 1100 S. Marietta Parkway, Marietta, GA 30060-2896, xcheng@spsu.edu

2 Southern Polytechnic State University, 1100 S. Marietta Parkway, Marietta, GA 30066-2896, dwilcox@spsu.edu

description programming language. The patented Bluespec technology is based on over eight years of research at MIT, starting in 1997[1]. In 2000, Prof. Arvind developed Bluespec version 1 which used the Haskellish syntax. The version 1 is much like TRAC which is written in the python Programming language. In 2001, the new version which is also the current version of Bluespec was launched. This version is based on Haskell syntax. Haskell syntax is derived from O'Haskell which is an object-oriented, concurrent extension of the functional programming language. It was developed at Oregon Graduate Institute and Chalmers University of Technology. The new version Bluespec contains full Haskell functionality at compile time, monads for handling state[1]. Besides, Bluespec is the only ESL synthesis solution for control logic, complex data paths and algorithms. It has delivered high-level ESL synthesis abstraction to system-C. The Bluespec toolset allows ASIC and FPGA designers to dramatically reduce design time, bugs, verification resources and re-spins that contribute to product delays and escalating costs. Bluespec improve the quality of verification both with a faster schedule and fewer resources. Blue-spec designers can decrease the time closure by changing the micro-architectural instead of trying to close timing sub-optimally by avoiding risky, late stage RTL changes [2].

Bluespec and VHDL

VHDL stands for VHSIC (Very High Speed Integrated Circuits) Hardware Description Language. It is now one of industry's standard languages used to describe digital systems. The other widely used hardware description language is Verilog. Both are powerful languages that allow you to describe and simulate complex digital systems. To be the newest hardware description language as both VHDL and Verilog, Bluespec has more benefits : Accelerate time to verified design by 50%; Reduce both bugs and verification costs by 50%; Retain flexibility to make architectural changes late in the; design cycle; Enable rapid timing closure; Experience unparalleled reuse, including faster derivatives; Leverage a unified environment for transaction level modeling through to hardware generation; Deliver safe, low-impact ECOs that do not disrupt designs [2].

When Blue-spec is using the exactly same hardware semantic and toolset as VHDL, it creates a new way to express concurrency and inter-module communications[2].

- High-level description of behavior (Rules and Interface Methods) resulting in designs that are more succinct, more correct-by construction, and easier to verify.
- Very powerful interface semantics, which enhance correctness when an IP block is plugged into varying environments and automatically manage resource sharing.
- Very high degree of parameterization, which greatly improves reuse.
- Strong support for embedded assertions, multiple clock domains and gated clocks [2].

Designers have the option of writing their designs at different levels, from transaction-level to implementation-targeted hardware. When starting intentionally high, designers can perform, at their control, a series of successive refinement steps on the design[3].

Bluespec provides a significantly higher level of abstraction than Verilog, SystemVerilog, VHDL and SystemC in the following dimensions[3]:

Behavioral descriptions: Bluespec uses rules and interface methods for behavioral description, adding a powerful way to express complex concurrency & control[3]:

1. Across multiple shared resources.
2. Across module boundaries.

Structural descriptions: Bluespec has significantly higher ways to describe and perform[3]:

1. High-level abstract types
2. Powerful static checking
3. Powerful parameterization
4. Powerful static elaboration
5. Advanced clock management

Bluespec Abstraction Level

Bluespec includes five basic concepts: date type, interface, methods, modules and rules. As to type, Bluespec is Hindley-Milner polymorphism + overloading type, whereas C is Non-polymorphic; VHDL is somewhat polymorphic. In Blue-spec data type using in the programs, Each variable and expression can be set to a type. The type checking is occurring before program elaboration or execution which can ensure that object types are compatible. As to interface, it is the connection between one module to the other. The difference of Bluespec is that you don't need describe a interface inside one module, it can be shared by many modules. As to methods, Methods are the specific functions which may be invoked by the caller. These functions take zero or more arguments, can return values or cause actions to occur.   When translated into RTL, each method becomes a bundle of wires. The method definition is part of the module definition. In Blue-spec, all descriptions of hardware is done in a monad-Module. Using a monad we can easily keep track of the state. The Module monad is built in to the compiler and its internals are not accessible to the programmer. The Module monad is extensible. A module consists of three things: state, rules that operate on that state, and the module's interface to the outside world (surrounding hierarchy). A module definition specifies a scheme that can be instantiated multiple times. Rules are used in Bluespec SystemVerilog to describe how data is moved from one state to another [4].

Register Transfer Level

Register transfer level (RTL) description is a way of describing the operation of a synchronous digital circuit. In RTL Design, a circuit behavior is defined in terms of the flow of signal between hardware design, and the logical operation performed on those signals. Register transfer level abstraction is used in hardware description language (HDLS) like Verilog and VHDL to create high level representation of a circuit, from which low-level representation and ultimately actual wiring can be derived [5]. RTL is based on the same core technology: state-centric, with low-level, explicitly described control logic and event-driven simulation RTL and system C are simulation-centric, but BSV is not simulation-centric, it is closer to traditional hardware view [6].

Classic Traffic Light Example

It is a 4-way intersection, with main road and side road. The Main route is a “major” route whereas the Side routes are “minor” routes. The normal cycle of the lights is: Main road red with side road green, Main road red with side road amber, Main road green with side road red, Main road amber with side road red.

Structure:

In this section, we use the block diagram to achieve the structure approach. The function of structure approach is to describe a logic function in VHDL. This block diagram specifies the relationships of the block, how they are connected with the principal parts. In this particular traffic light example, we have the state decoder to divide the state into individual light statement. We have the clock to accomplish the different timers function. The sensor of the each road will also take part into the sequential logic circuit.

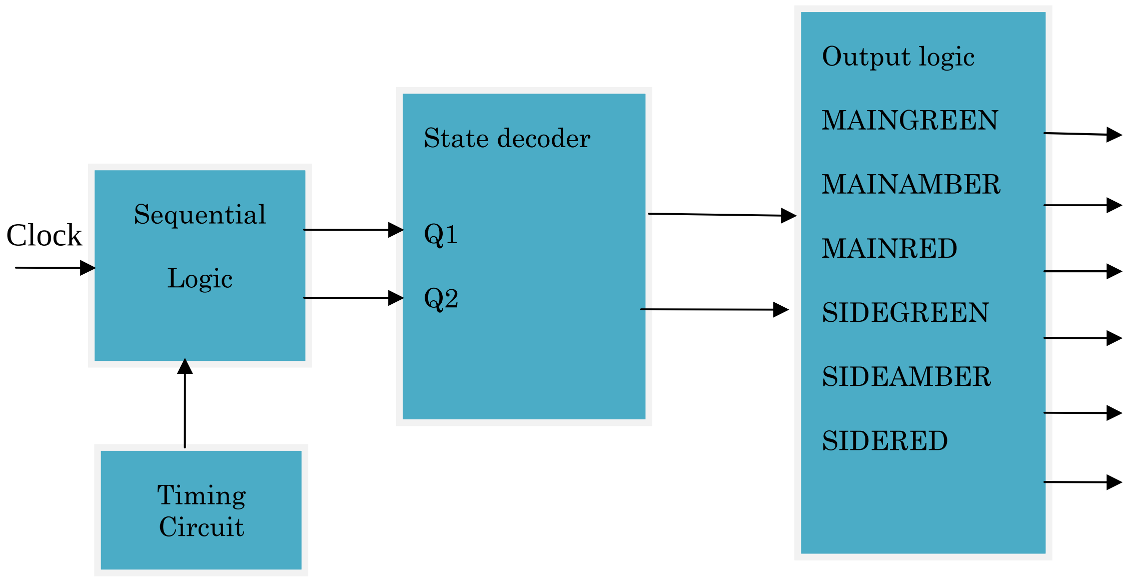

Figure 1: Block Diagram of Traffic light

Data Flow:

In this part, we will apply the Boolean expression to achieve the data flow approach. In the traffic light example, we have nine states for the road lights. We will express how the data flow and what logical relationships are.

MAINGREEN= Q1Q2

MAINAMBER= Q1Q2

MAINRED = (-Q1-Q2) + (-Q1Q2)

SIDEGREEN = -Q1-Q2

SIDEAMBER = -Q1Q2

SIDERED = Q1Q2 + (Q1-Q2)

Behavior:

In this section, we will use the state diagram to accomplish the behavior approach. It is the most abstract ways to describe a logic function. State diagram is usually consisting of several numbers of states or cases. It is also a common way to describe the behavior of a logic function or system. Each diagram indicates the object of a single state.

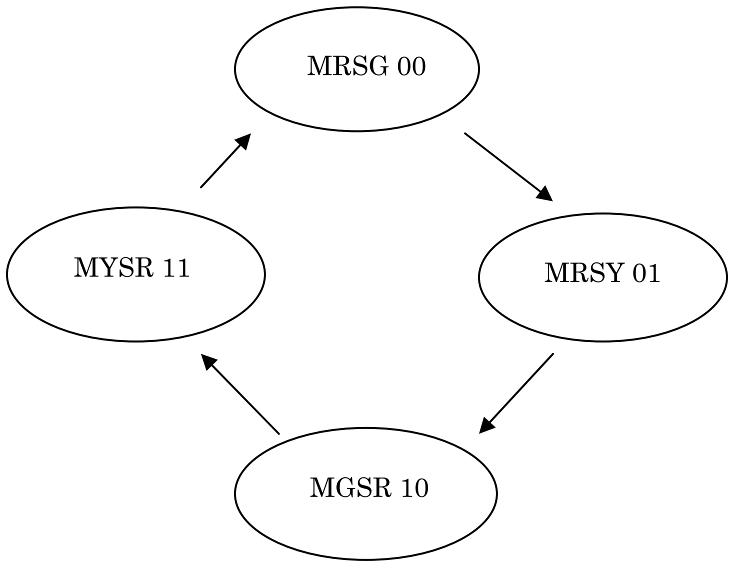

Figure 2: State Diagram of the traffic light

Traffic light small example in VHDL:

```
library ieee;
use ieee.std_logic_1164.all;
entity trafficlight is
  port( State: out std_logic_vector(5 downto 0);
        clk0: in std_logic);
attribute LOC: string;
attribute LOC of clk0:          signal is "P11";
attribute LOC of State:     signal is "P02,P03,P04,P06,P07,P08";
end entity ;
architecture  Behavior of trafficlight is
signal Q: std_logic_vector (1 downto 0);
begin
process (clk0,Q)
begin
        if (clk0 = '1' and clk0' event) then
                case Q is
                when "00" => Q <= "01";
                when "01" => Q <= "10";
                when "10" => Q <= "11";
                when "11" => Q <= "00";
                when others => Q <= "00";
                end case;
        end if;
end process;
process (Q)
        begin
                case Q is
        when "00" => State <="011110";
        when "01" => State <="011101";
        when "10" => State <="110011";
        when "11" => State <="101011";
        when others => State <="011110";
        end case;
        end process;
end architecture Behavior;
```

Traffic light small example in Bluespec:

```
package TL0;

interface TL;
   method Bool lampRedM();
   method Bool lampAmberM();
   method Bool lampGreenM();
   method Bool lampRedS();
   method Bool lampAmberS();
   method Bool lampGreenS();
endinterface: TL

typedef enum {
   GreenM, AmberM, RedM,
   GreenS, AmberS, RedS} TLstates
deriving (Eq, Bits);
(* synthesize *)

module sysTL(TL);

Reg#(TLstates) state <- mkReg(RedM);
Reg#(TLstates) states <- mkReg(GreenS);

rule fromRedM (state == RedM && states == GreenS );
   state <= RedM;
   states <= AmberS;
endrule: fromRedM

rule fromAmberS (state == RedM && states == AmberS );
   state <= GreenM;
   states <= RedS;
endrule: fromAmberS

rule fromGreenM (state == GreenM && states == RedS) ;
   state <= AmberM;
   states <= RedS;
endrule: fromGreenM

rule fromAmberM (state == AmberM && states == RedS);
   state <= RedM;
   states <= GreenS;
endrule: fromAmberM

method lampRedM() = (state == AmberM ||
state == GreenM);
method lampAmberM() = (!(state ==
AmberM));
method lampGreenM() = (!(state ==
GreenM) );
method lampRedS() = (states == AmberS ||
states == GreenS );
method lampAmberS() = (!(states ==
AmberS) );
method lampGreenS() = (!(states ==
GreenS) );

endmodule: sysTL

endpackage: TL0
```

LINUX

Ubuntu

As a Linux operating system based on Debian , unlike the MEPIS-, Xandros, Linspire, Propeny and Libranet , Ubuntu is more closed to the Dobian theory, because it uses free open software sources instead of closed sourced by the others[7].

- Goal: provide an up-to-date, stable operating system for average user, focus on usability and ease of installation.
- Title: the most popular linux distribution for the desktop, claiming approximately 30% of desktop Linux installations in 2007.
- Benefit: users can run, copy, distribute, study, change and improve the software freely because it is composed of free and open source software distributed under various licenses, especially the GNU general public license.
- Sponsor and owner: sponsored by UK based company canonical Ltd; owned by South African entrepreneur Mark Shuttleworth.

Gvim to Gedit

Gvim is the text editor for the Redhat Linux Operating system. Gedit is the text editor of the GNOME desktop environment. Because we are using Ubuntu Linux operating system which applies the GNOME desktop environment, when we create a project, we have to select others and type gedit into the “other command” under the project option.

Bluespec Operation in Linux

After we installed both the Bluespec environment and the Ubuntu Linux operating system, we can launch the Bluepec development workstation by typing the command ‘bluespec’ in the Linux environment.

In the workstation of Bluespec, we can execute all the behaviors like creating project, type checking, compile, linking and simulating. The first step is creating a project. In this step, the project option setting is very important, because we have to choose the top file and module for this particular project and setting the file location for all the output files, we have to decide which way to compile the file and which simulator to simulate the project. There are two ways to compile in the Bluespec environment: compile via bsc, compile via make file and compile via custom command. There are also two different way for simulating. One is Bluesim which means we compile the project to Bluesim simulator. The other one is Verilog which contains the following simulator options: iverlog, modelsim, ncverlog, vcsr, cver, and veriwell.

After we finish setting all the project option, we can compile, type check, link and simulate the project following the tool bar.

We get the Verilog file from the Bluespec output location and apply it to the lattice verilog simulator. We can get the generated file (Gedec file) and output it into CPLD board chip from where we get out small traffic light display as the VHDL example.

We cannot generate VHDL file from Bluespec workstation because the bluespec environment can only generate file with .bsv extend.

Conclusion

Comparing Bluespec to VHDL, Bluespec is a higher level language which will make the project more flexible and achievable. Unfortunately, to a freshman or a sophomore student in an introductory digital logic course the higher abstraction requires too much knowledge of the lower levels for a seamless transition into HDL design. At this point, the student would be required to have an intermediate level of experience with Linux to navigate through the Bluespec command line environment. To implement Bluespec at this level, considerable packaging would be required by the instructor.

Acknowledgements

Special thanks to Rishiyur S. Nikhil, Ph.D., Co Founder of Bluespec, Inc. for support of this honor undergraduate research. The authors would like to thank to Jeff Carter, SPSU's network engineer, for his assistance.

References

1. Xilinx, Inc, ©Copyright 2009
   Xilinx ,http://www.xilinx.com/products/design_tools/logic_design/advanced/esl/bluespec.htm.
2. Bluespec, Inc. • 200 West Street • Waltham, MA 02451 • ©2005 Bluespec, Inc. All Rights Reserved
3. Xilinx, Inc, ©Copyright 2009
   Xilinx ,http://www.xilinx.com/products/design_tools/logic_design/advanced/esl/bluespec.htm.
4. Bluespec, Inc. • 200 West Street • Waltham, MA 02451 • ©2005 Bluespec, Inc. All Rights Reserved
5. http://en.wikipedia.org/wiki/Register_transfer_level
6. WHY BLUESPEC Vs.RTL, http://www.bluespec.com/why-bluespec/vs-rtl.htm, ©Copyright 2009 Bluespec, Inc. All Rights Reserved.
7. http://en.wikipedia.org/wiki/Ubuntu