

An Efficient Implementation of High Speed complex Statemachine in FPGA Architecture using VHDL Synopsys Environment

Shreyas Shah and Bala Sreekandath

Abstract

Designing an efficient high speed complex state machine has always been a challenging task in the field of high speed digital logic design. In the last decade, there has been notable developments in implementing and prototyping the digital logic design in FPGA (Field Programmable Gate Array) before making an ASIC (Application Specific Integrated Circuit). This has not only reduced the design time but also reduced the cost of the development cycle. The FPGA architecture provides various levels of flexibilities to implement high speed complex designs. Hardware Description Languages (HDLs) have emerged to aid the designers to describe a complex circuit behavior at higher abstraction level. This has led to the popularity of VHDL (Very High speed Integrated Circuit Hardware Description Language). VHDL has become a defacto standard in describing a behavioral circuit model. The extraction and usage of all the architectural features of FPGA using VHDL is a key issue in FPGA implementation of high speed design. In this paper, we have addressed the different issues involved in designing high speed state machines in FPGA architecture using VHDL under Synopsys¹ Environment. We have analyzed two styles of writing VHDL code for such implementations. The authors

¹Synopsys is a Trademark of Synopsys, Inc

recommend an optimal method to implement the design in Xilinx XC4000 FPGA² family. The method, however, is also applicable to any other FPGA architecture.³

1 Introduction

In this paper, we have made an attempt to describe the VHDL coding style for a high speed complex statemachine implemented in FPGA. The statemachine is developed using VHDL under Synopsys environment. The State Machine can be of type mealy or moore or hybrid depending upon dependency of outputs over inputs. The complex statemachines are either mealy or hybrid statemachines. In the following section, a general FPGA architecture is explained in brief.

2 FPGA Architecture: Brief

Recently, FPGAs are widely used for prototyping the circuit. FPGA implementation, reduces the design cycle time and cost of the

²Xilinx is a Trademark of Xilinx, Inc

³This paper is the gist of authors' experience in using VHDL-Synopsys-Xilinx extensively for implementing very high speed, complex statemachines in Xilinx XC4000 FPGA family architecture. Efforts have been made to make this article independent of the vendor specific implementations.

development cycle. The major component in FPGA is a Logic Block. The Logic Block contains one or more combinatorial logic generators. The number of inputs to combinatorial logic generator depend upon the architecture of an FPGA. In Logic Block, alongwith the combinatorial logic generators, one or more flipflops are provided. To extract the maximum performance out of an FPGA architecture, pipeline design is recommended. The FPGA architecture has abundant flipflops, to support the pipelined design.

There are various parameters that decide the performance of the implemented circuit in FPGA.

- Clock to Setup
Clock to setup time can be defined as : Clock to output time of a statevariable flipflop + path delay (from Q out to another combinatorial logic) + combinatorial logic delay + setup time of next flipflop. This parameter decides the frequency of operation of the statemachine. This should be equal or less than the desired clock cycle time.
- Pad to Setup
Pad to Setup Time can be defined as : Pad Delay + path delay + combinatorial logic delay + setup time of flipflop. This parameter is defined for input signals. This, in addition to clock to output delay of driving device, decides whether it is possible for the statemachine to sense the signal on immediate clock after the signal generation. If not, the input signal is latched before sensing. This introduces one level of pipeline. Once the input signal is latched, the pad to setup time is being merged into clock to setup parameter.
- Clock to output
Clock to output is defined as : clock to output of a statemachine flipflop

+ combinatorial logic delay + pad delay. This parameter in addition to setup time of next device should be less than one clock period. If not, the output signal is latched in a flipflop before taking it to I/O pad. This introduces one level of pipeline.

- Pad to Pad
This is a pure combinatorial logic delay. It depends upon the design , hence it is not discussed in this paper.

3 FSM implementation using VHDL

The VHDL code for statemachine can be written in two different ways.

1. Asynchronous Coding Style
In this coding style, there are two processes. One is asynchronous process and another is synchronizing process. State machine is written in asynchronous process. In this, nextstate variable is assigned the value depending upon the inputs and previous state condition. In synchronizing process, this value is assigned to the statevariable on the clock edge. In this coding style, all the outputs are combinatorial, but are registered selectively, in a synchronizing process.

In this particular implementation style of a statemachine, the flipflop used as a state variable will be without clock enable. Xilinx 4000 family of devices have flipflop with clock enable input, they are used as normal flipflop with clock always enabled.

```
out1 ← booltostd(statevar = busy1);
syncro : process(clk)
begin
wait until clk'event and clk = '1';
```

```

statevar ← nextstate;
out1reg ← out1;
end process;
stateproc : process(in1,in2)
begin
case statevar is
when idle ⇒
if(in1 = '1') then
nextstate ← busy1;
else
nextstate ← idle;
end if;
when busy1 ⇒
if(in2 = '1') then
nextstate ← idle;
else
nextstate ← busy1;
end if;
when others ⇒
nextstate ← idle;
end case;
end process;

```

```

when idle ⇒
if(in1 = '1') then
statevar ← busy1;
out1 ← '1';
end if;
when busy ⇒
if(in2 = '1') then
statevar ← idle;
out1 ← '0';
end if;
when others ⇒
statevar ← idle;
out1 ← '0';
end case;
end process;

```

2. Synchronous Coding Style

In this coding style, the statemachine is written as a synchronous process. In this, all the outputs are registered. The clock enable input of a flipflop is used for all the outputs and statevariable in this style. FPGA compiler of Synopsys uses flipflops with clock enable when the target device is one of the Xilinx 4000 family. The disadvantage of writing this kind of statemachine is that all the outputs are registered and when one tries to reduce clock to output parameter by introducing one level of pipeline, this outputs introduces one more level of pipeline and there will be two level of pipeline, that may not be acceptable in many of the statemachines. statepro :

```

process(clk,in1,in2)
begin
case statevar is

```

4 VHDL Coding style for FPGA Architecture

In this section we will go through the VHDL coding style to meet various parameters mentioned in Section No. 2.

- Clock to Setup

Authors have found that using asynchronous process for FSM implementation, as mentioned in Section No. 3, has following advantages.

1. Outputs can be selectively registered. The outputs, which are required to be registered can be selected and put in synchronizing process. In FPGA architecture, abundant flipflops are provided to implement pipelined design to meet the timing requirements.
2. Authors have found that Synopsys FPGA compiler generates less logic for the state machine. The outputs are generated using state variable equations. Thus, the asynchronous process helps in reducing clock to

setup time and eventually, increases the speed.

- Pad to setup

```
Here let us see the following VHDL code
if(w= 1 or x=0 or y= 0 or z =1 ) then
Out1 ← "0001";
elsif(w= 1 or x=0 or y= 0 or z =0 ) then
Out1 ← "0010";
else
Out1 ← "1111";
end if;
```

same code can be written as following,

```
signal abcd : std_logic_vector(1 downto
0);
abcd(0) ← w and not x and not y and z
;
abcd(1) ← w and not x and not y and
not z;
case abcd is
when "00" ⇒
Out1 ← "1111";
when "01" ⇒
Out1 ← "0001";
when "10" ⇒
Out1 ← "0010";
when "11" ⇒
Out1 ← "1111";
end case;
```

In the first code, the else part consists of 14 different conditions. In those 14 conditions, the Out1 is assigned 1111. In the second code, for only two conditions, Out1 is assigned 1111. Therefore, the combinatorial logic to generate Out1 reduces in the second style and it helps in meeting the clock to output pad timing. In this, if w , x , y , z are inputs from pads, then generation of output Out1, the combinatorial logic will be in multiple levels and makes it difficult to meet

the pad to setup time. This is experienced while designing very complex (approx. 33 states) statemachine and targeting to a Xilinx XC4005 device.

- Clock to output timing

Assume that the number of states in a statemachine is 35. There is one active low signal, which is asserted in 5 states and not asserted in 30 states. This means that the output is '1' in 30 states and '0' in 5 states. Synopsys Compiler will connect an OR gate of 30 inputs (when statemachine is one hot encoded) and output is inverted.

Our suggestion is to consider the output as an active high. This means that the output will be '1' in 5 states and '0' in 30 states. Hence Synopsys Compiler will connect a 5 inputs OR gate instead of 30 inputs OR gate. This has more significance in FPGA architecture, because it reduces the number of Logic Block levels that helps in reducing clock to output parameter.

5 Conclusion

We have reached the following conclusions to efficiently write a high speed complex state machine in VHDL under Synopsys environment and targeting to FPGAs. We claim that this is an optimal method when targeting the design on Xilinx 4000 Family of devices under Synopsys Environment. However, these can be applied to the design targeted towards any other FPGA architecture, as well.

1. Write the statemachine as a asynchronous process. This helps in optimizing the clock to setup time.
2. Do not generate any output inside a statemachine.

3. Write the equations for the output generation as shown in the first statemachine. This helps in optimizing clock to output time.
4. Selectively register the outputs which are required, in a separate synchronizing process.
5. Follow the rule of using case statement instead of if-else. This helps in optimizing pad to setup and clock to setup time.
6. All the active low outputs should be considered as active high, while writing equations and then invert it to get actual output.

Contact Address:

Shreyas Shah, Bala Sreekandath
Comit Systems, Inc
1250 Oakmead Parkway, Suite 210
Sunnyvale CA 94088.

Phone : (408) 988 2988

Fax : (408) 988 2133

E-mail : shreyas@comit.com
bala@comit.com

Acknowledgement

The authors would like to thank Mr B L Priyadarshan of Comit Systems, Inc for his useful criticisms and suggestions. His detailed comments have greatly improved the quality of the paper. The authors also thank Mr Vijay Raghavan, President Comit Systems, Inc for his useful suggestions and constant encouragement. Authors are thankful to Comit Systems, Inc for extending the Lab facilities for this work.

References

1. Synopsys VHDL Compiler Reference Manual.
2. Synopsys Design Compiler Reference Manual.
3. Xilinx Synopsys Interface User's Manual.
4. XSI FPGA User Guide.