

HDL Coding Style for LUT Based FPGAs

Samir Samhouri
AT&T Microelectronics
555 Union Blvd.
Allentown, PA 18103

Abstract

In an ideal world, synthesis tools would be able to understand all of the FPGA architectures available and would be able to take advantage of the special features that these devices provide without the interference of the designers. In the real world, however, this is not the case, for applications that are speed and area intensive, designers have to be aware of the consequences of the coding style that they choose to follow. In addition, an understanding of the FPGA's architecture, the synthesis tool and the back end software is also a requirement. In this paper a description of certain VHDL constructs as well as their synthesis results is going to be discussed illustrating some of the shortcomings in the High Level Design Flow for LUT based FPGAs.

Introduction

Most FPGAs are not fine grain but instead they are made up of programmable functional units (PFU) that can implement combinational logic in the look up tables (LUT) as well as a certain number of flipflops or latches. The following are some of the FPGA features that the synthesis tools might have difficulties in implementing:

- The flipflops inside the PFUs share some of the control signals like the clock, clock enable and reset/set and hence in order to get 4 flipflops (in an ORCA architecture) to fit inside a single PFU, they have to have the same mentioned signals. Most synthesis tools do not understand this, yet, and, therefore, if a design was coded without keeping this fact in mind the tools might utilize some of the flipflops inefficiently resulting in the inflation of the size of the chip.
- Memory elements inside some FPGAs can be implemented in the LUT portion of the PFU. This method of constructing RAM or ROM inside FPGAs saves a large number of gates and drastically improves the speeds of the devices. Unfortunately, there is no one way to implement memory in HDL and hence the synthesis tools cannot detect their presence to utilize the FPGAs LUT feature.
- Counters and state machines, there are so many different kinds of these circuits and a reason for using one over the other is mostly dependent on the application, but a knowledge of the FPGAs architecture also helps in deciding which method is most efficient.
- Design hierarchy and floorplanning.
- Global Set Reset signal (GSR) is an internally routed reset signal that does not consume any of the chips routing resources. As of now, there is no way to implement this feature in VHDL and hence synthesis tools cannot utilize this feature unless the GSR component gets instantiated in the HDL code.

There are three techniques to writing a VHDL code. Starting with the least efficient method they are:

1. A generic code that has not been targeted to an architecture.
2. A generic code targeted towards a device architecture.
3. An HDL code with macro instantiation.

In the next few sections a comparison is going to be made between these three methods, incorporating coding styles that would be targeted to reduce the synthesis inefficiencies shown on the previous page. The ORCA FPGA from AT&T has been targeted for the applications in this paper.

Implementation of Synchronous Logic

1 Implementation of Synchronous Reset/Set and Clock Enable

FFs and Latches in most LUT based FPGAs can be configured in synchronous set/reset mode using the Local Set Reset (LSR) assigned by the designers. In order for a Latch or FF to be implemented correctly the proper library macro has to be instantiated by the synthesis tool, but this will not happen unless the HDL code has the correct description. A basic understanding of the FPGA architecture that is going to be used is a must.

Be aware of the following:

- Designers have to keep in mind the kinds of FFs and latches that are available in the vendors macro library. If the code implements a register functionality that is not represented by a corresponding macro in the library, the extra functionality will be added to circuit using additional logic, most of the time on the data path of the registers, increasing the area as well as delay.
- Each Programmable Functional Unit (PFU) can implement up to a certain number of Latch/FFs that share some of the inputs to the PFU. So in order to get the highest area utilization out of the device, Latch/FFs are recommended to be grouped in multiples of the PFU's register capacity.
- If the synchronous functionality of the flipflops is required then the Global Set Reset(GSR) signal cannot be used to implement the set/reset signal since the GSR has an asynchronous functionality. However, it can be used in addition to the LSR signal.
- If the code implies a gated CE signal, the synthesis tools tend to duplicate the enable logic for every register in the design. To avoid this, it is recommended to keep the gated signals in a separate process and to pass their output to the CE input of the main module.

In order to use the correct flipflop, the HDL code has to describe the correct functionality. For instance to implement a two bit register with a +ve level synchronous reset and a +ve level enable signal the code shown below is used.

```

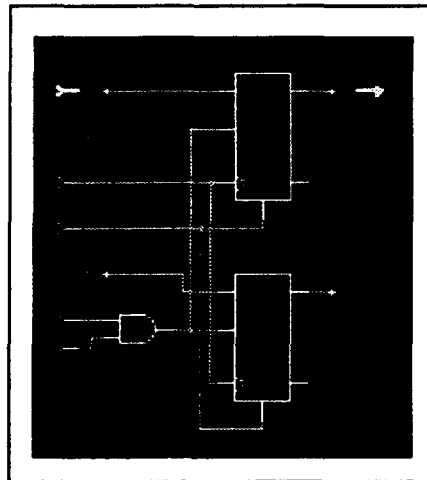
DO <= D1 AND D2;
SYNC_RST : Process (CLK,RST)
begin
  if (CLK'event and CLK='1') then
    if (RST = '1') then
      DATA_OUT <= (others => '0');
    elsif (DO = '1') then
      DATA_OUT <= DATA_IN ;
    end if;
  end if;
end if;

```

```
end process SYNC_RST;
```

Notice from the code that in order to implement a synchronous reset correctly the “*if (RST = '1') then*” statement has to be entered after the *CLK*'event inside the process, and for “DO” to be connected to the CE input of the FF the “*elsif (DO = '1') then*” statement has to go after the “*if (RST = '1') then*”. The circuit for the above code is shown in figure 1.

Figure 1 Two Bit Register with +ve Level Synchronous Reset and Clock Enable



Caveats:

- Some synthesis tools have known limitations in implementing synchronous reset/set and can produce some unpredictable results that, although are functionally correct, would effect the area and speeds of the circuits.
- Signal will be connected to the CE of the flipflops only if the code was implemented as shown in the previous HDL example.
- Some signals that were not meant to get connected to the CE port will be if designers do not know what kind of coding algorithm will result in a CE connection. For instance:

```
SYNC_RST : Process (CLK,RST)
begin
  if (CLK'event and CLK='1') then
    if (D1 = '0' and DATA_IN = "10") then
      DATA_OUT(0) <= DATA_IN(0) ;
    elsif (D2 = '0' and DATA_IN = "01") then
      DATA_OUT(1) <= DATA_IN(1);
    end if;
  end if;
end process SYNC_RST;
```

The above code will generate two FFs with two different CE signals because of two reasons:

4. There are some undefined states in the process (i.e. state when (D1='1' and DATA_IN="10"))
5. For the defined states, not all of the outputs were defined under every “if” statement.

Both of these issues will force the synthesis tool to use the CE port of the FFs in order to retain their previous values and as a result this circuit will consume two PLCs instead of one. To avoid these kinds of inefficiencies try to do the following when writing an HDL code:

1. Always try to group multiple of four FFs under every "if" statement.
2. Try to define all the states of the control signals and the status of the register outputs for every state.

```

if (CLK'event and CLK='1') then
  if (D1 = '0') then
    DATA_OUT <= "01" ;
  elsif (D2 = '0') then
    DATA_OUT <= "10";
  else DATA_OUT <= DATA_IN;
end if;

```

The code above is an example of code that will not try and utilize the CE inputs of the FFs.

Synchronous Memory Modules

The most efficient way to implement memory in SRAM FPGA is by using the internal Look Up Tables (LUT) inside of the Programmable Functional Units (PFU). In an AT&T FPGA each PFU can implement either of the following RAM or ROM arrays:

- A single 16x4 element.
- Two 16x2 memory blocks .

Multiple PFUs can then be used to implement other arrays sizes (i.e/ 16x8, 32x4, 64x8 etc..). In this section, three methods are going to be used for implementing a 16x8 memory block:

- **Method1: A generic VHDL code:**

```

entity fig25a is
  Port ( clk_w, reset, wr: in std_logic;
        add_in : integer range 0 to 15;
        data_in : in std_logic_vector(7 downto 0);
        data_out : out std_logic_vector(7 downto 0));
end fig25a;
architecture synth of fig25a is
  constant depth : integer := 16;
  type data_array is array ( integer range <> ) of std_logic_vector (7 downto 0);
  signal data : data_array (0 to depth - 1);
begin
  process (clk_w)
  begin
    if (clk_w='1' and clk_w'event) then
      if wr = '0' then
        data (add_in) <= data_in;
      end if;
    end if;
  end process;
end architecture;

```

```

        end process;
        data_out <= data (add_in);
    end synth;

```

Utilization in a 2C04:

Number of FFs used: 128
 Number of PFUs in design: 76 out of 100
 Number of TBUFs in design: 0 out of 800

Timing Report:

38MHz is the maximum frequency for this circuit after map,place & route.

- **Method 2: A generic VHDL code targeted towards FPGAs:**

```

entity fig25b is Port ( clk_w, wr: in std_logic;
  add_in : integer range 0 to 15;
  data_in : in std_logic_vector(7 downto 0);
  data_out : out std_logic_vector(7 downto 0));
end fig25b;
architecture synth of fig25b is
  constant depth : integer := 16;
  constant idepth : integer := (depth - 1);
  type data_array is array ( integer range <> ) of std_logic_vector (7 downto 0);
  signal data : data_array (0 to depth - 1);
begin
  GEN_LABEL : for I in idepth downto 0 generate
    data_out <= DATA(I) when (wr='1' and I=add_in) else (others=>'Z');
    Process begin
      wait until (clk_w'event and clk_w = '1');
      if (I = add_in and wr = '0') then
        data(I) <= data_in;
      end if;
    end process;
  end generate GEN_LABEL;
end synth;

```

Utilization in a 2C04:

Number of FFs used: 128
 Number of PFUs in design: 41 out of 100
 Number of TBUFs in design: 128 out of 800

Timing Report:

40MHz is the maximum frequency for this circuit after map,place & route.

- **Method 3: Instantiatoin of RAM**

```

architecture ram16x8z of ram16x8z is
  component ram16x8z

```

```

Port ( clk_w      : in std_logic;
      reset: in std_logic;
      add_in  : in std_logic_vector(3 downto 0);
      data_in : in std_logic_vector(7 downto 0);
      wr     : in std_logic;
      data_out : out std_logic_vector(7 downto 0));
end component;
begin
  u1: ram16x8z port map (clk_w, reset, add_in, data_in, wr, data_out);
end ram16x8z;

```

Utilization in a 2C04:

Number of FFs used: 20
 Number of PFUs in design: 6 out of 100
 Number of TBUFs in design: 8 out of 800

Timing Report:

52MHz is the maximum frequency for this circuit after map,place & route.

A Comparison:

Method 1 :

Advantages:

1. Maintains a generic VHDL code that can be targeted to any technology.
2. No knowledge of the FPGA architecture is required.

Disadvantages:

1. No utilization of the FPGA's architectural features.
2. Poor area and timing results.

Method 2 :

Advantages:

1. Maintains a generic VHDL code that can be targeted to any technology.
2. An improvement of almost 50% in terms of area over method 1.
3. An improvement of almost 200% of Clock to Out delays over method 1.

Disadvantages:

1. The use of the FPGA's Tri-state buffers. In bigger designs this might make routing difficult.
2. No utilization of the FPGA's architectural features.

Method 3 :

Advantages:

1. An improvement of almost 25% in the overall timing performance of the design.
2. A reduction of almost 35 PFUs from method 2.

Disadvantages:

1. VHDL code is locked to a specific technology.

Counters and Statemachines

There are many types of counters and statemachines that can be implemented through VHDL each of which is application specific. In this part of the paper, efficiencies and inefficiencies are going to be discussed for some of these circuits.

- **Binary Counters**

These kinds of circuits are the easiest to implement in HDL, fortunately, they also fit very efficiently in some of the LUT based FPGAs. In an ORCA architecture, for instance, each LUT can be configured in a ripple mode after which the PFU can implement up to 4-bit arithmetic functions. Moreover, most common synthesis tools understand this FPGA feature and can take advantage of it while keeping the HDL code generic. The code below shows a very simple HDL implementation of a synchronous eight bit up counter with an enable line.

```
SYNC_CNT : Process (CLK,RST)
begin
  if (RST = '1') then
    CNT <= (others => '0');
  elsif (CLK'event and CLK='1') then
    if (ENBL = '1') then
      CNT <= CNT + '1';
    end if;
  end if;
end process SYNC_RST;
DATA_OUT <= CNT;
```

Utilization in a 2C04:

Number of FFs used: 8
Number of PFUs in design: 2 out of 100

Timing Report:

91.542MHz is the maximum frequency for this circuit after map, place & route.

Advantages:

1. Very simple to implement.
2. Fits very efficiently in most LUT based FPGAs.

Disadvantages:

1. If the counters output was required to get decoded for applications like a statemachine controller or a generic memory block, the decode logic will add a considerable amount of gates to the circuit and most probably will degrade the performance.

- **One-Hot Key or Shift Register Counters**

If performance is the desired goal, this method would be more suitable than the previous one. This method however has a serious draw back, it consumes a large number of gates. The code below shows the HDL code for a 4-bit counter implemented in One-Hot Key.

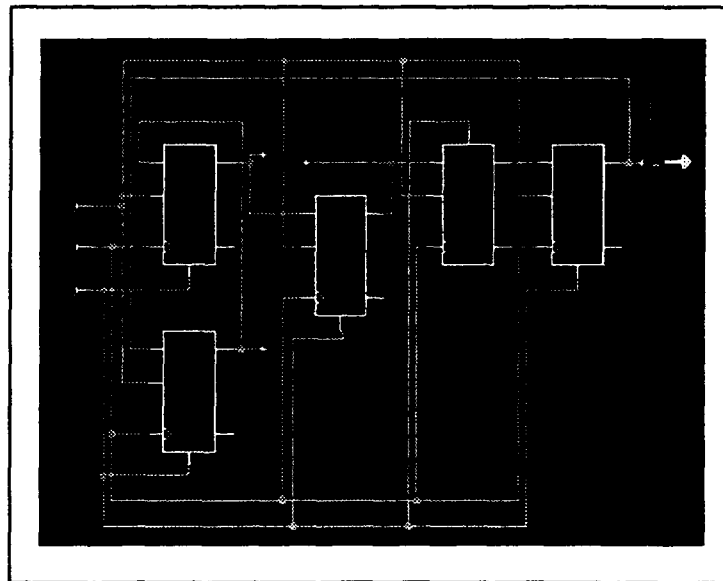
```

SYNC_CNTR : Process (CLK,RST)
begin
  if (RST = '1') then
    CNT(3 downto 0) <= "0000";
    CNT(4) <= '1';
  elsif (CLK'event and CLK='1') then
    if (ENBL = '1') then
      CNT <= CNT(3 downto 0) & CNT(4);
    end if;
  end if;
end process SYNC_RST;
DATA_OUT( 3 downto 0) <= CNT(3 downto 0);

```

The circuit resulting from the above code is shown below.

Figure 2: A 4-bit One-Hot Key Counter Circuit



Utilization in a 2C04:

Number of FFs used: 5

Number of PFUs in design: 2 out of 100

Timing Report:

109MHz is the maximum frequency for this circuit after map,place & route.

Advantages:

1. This kind of a circuit is much faster than the binary implementation. This is especially true when it is combined with some sort of a machine controller since the decoding of the counters output will only be done on a single bit.

2. Easy to implement in VHDL. As can be seen from the code above, it takes only one line of code using the concatenation operation (&).

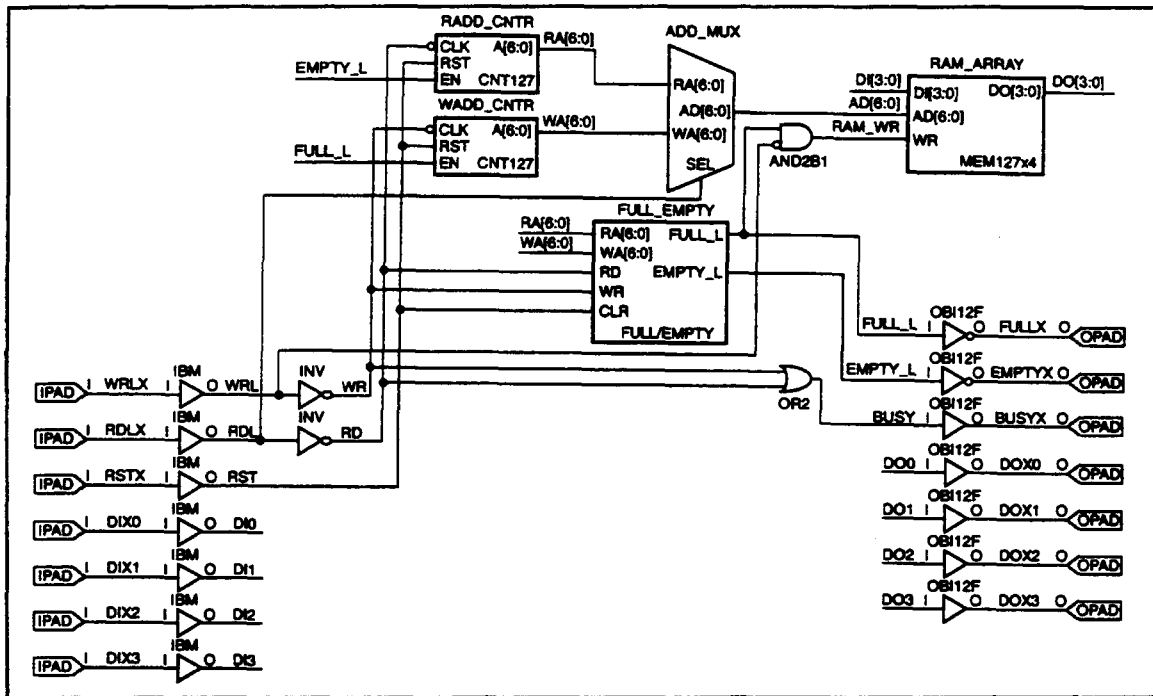
Disadvantages:

1. It consumes a large number of flipflops to implement. For example, assume that an N-bit counter is to be built, using this method the number of registers required to implement all of the counter states will be $[(2**N)+1]$.
2. This form of counters or statemachines does not take advantage of the FPGA's architectural features that would allow for a straight forward implementation of arithmetic functions.

An example:

The example shown below is going to discuss the implementation of a First in, First out (FIFO) memory block. The size of the FIFO is going to be 127 x 4. Shown in figure 3 is the block diagram of the FIFO. This circuit is going to be implemented in two methods, without instantiation of memory and with instantiation.

Figure 3: Block Diagram of a 127x4 FIFO



The FIFO is a single port, meaning that the memory array can only be read or written at a time, the FULL_L and EMPTY_L indicate the status of the FIFO. WRL and RDL are the active low write and read signals.

• **Method 1: A generic VHDL code for a 127x4 FIFO:**

For this method, the whole FIFO design was placed under one process and then synthesized as one block. Due to the limitation on the length of the paper, the code will not be shown. The following are the utilization results:

Utilization in a 2C15:

Number of FFs used: 528

Number of PFUs in design: 200 out of 400

Timing Report:

20MHz is the maximum frequency for this circuit after map,place & route.

• Method 2: An Instantiated VHDL code:

The same code that was used for method 1 was divided up into 2 blocks:

Block 1: Is a single process that has the following functions:

1. Calculates the WRITE and READ addresses depending on the address's previous values as well as the FULL_L, EMPTY_L, WRL and RDL signals.
2. Calculates a value named DIFF_PTR which gets incremented by 1 in a WRITE operation and decremented by the same value during a READ operation. This value gets used to set the FULL_L and EMPTY_L signals of the FIFO.

From figure 3 this block includes everything except the RAM_ARRAY entity. The process for this block is shown below:

```
process (clk,reset)
begin
    if reset = '1' then
        RD_ADD <= (others =>'0');
        WR_ADD <= (others => '0')
        DIFF_PTR <= 0;
    elsif (clk= '1' and clk'event) then
        if wr = '1' and (status=not_empty or status = empty) then
            WR_ADD <= WR_ADD + '1';
            DIFF_PTR <= DIFF_PTR + 1;
        elsif rd = '1' and (status=not_empty or status = full) then
            RD_ADD <= RD_ADD + '1';
            DIFF_PTR <= DIFF_PTR - 1;
        end if;
    end if;
end process;
status <= empty when DIFF_PTR = 0 else
    full when DIFF_PTR = depth else
    not_empty;
```

Block 2: An instantiated VHDL code:

In this entity there is an instantiation for eight RPP16x4z macros from the ORCA FPGA library. These macro were netlisted so as to create the 127x4 memory block of the FIFO (RAM_ARRAY).

Utilization in a 2C15:

Number of FFs used: 20

Number of PFUs in design: 25 out of 400

Timing Report:

29MHz is the maximum frequency for this circuit after map,place & route.

Conclusion

It is probably enough to write any kind of a generic HDL code, as long as it is synthesisable, for designs that are not speed and area intensive. However, for designs that are speed and area critical, a basic knowledge of the FPGA architecture and the correct HDL coding style for that architecture is a must. Synthesis vendors are currently working actively with the FPGA vendors in order to get the synthesis tools to a point where they would be able to take advantage of the continuously growing architectural features of these devices. For now, designers have to still apply their hardware digital design experiences while coding HDL in order to get the highest utilization out of the FPGAs.

References

B.Cohen, "VHDL Coding Styles and Methodologies", Kulwer Academic Publishers, M.S., 1995

D.Ott, and, T.Wilderotter, "A Designer's Guide to VHDL Synthesis", Kulwer Academic Publishers, M.S., 1994

AT&T Microelectronics, "AT&T Field-Programmable Gate Arrays Data Book", 1995.

AT&T Microelectronics, "ORCA's FPGAs HDL Design Guide", 1996.