Novel Area Optimization in FPGA Implementation Using Efficient VHDL Code
Zulfiqar
Abstract—A new novel method for area efficiency in FPGA implementation is presented. The method is realized through flexibility and wide capability of VHDL coding. This method exposes the arithmetic operations such as addition, subtraction and others. The design technique aim to reduce occupies area for multi stages circuits by selecting suitable range of all value involved in every step of calculations. Conventional and efficient VHDL coding methods are presented and the synthesis result is compared. The VHDL code which limits range of integer values is occupies less area than the one which is not. This VHDL coding method is suitable for multi stage circuits.

Keywords: Area optimization, FPGA, VHDL, Verilog, VHDL package.

I. INTRODUCTION

Huge number of FPGA applications during the last three decades have been influenced many scientists and engineers to develop a more and more fast and more capacity of Field Programmable Gate Arrays (FPGA) chips. However, the optimal speed and minimum area of a circuit when it is implemented on an FPGA chip may not be achieved if the HDL (Hardware Description Language) codes do not describe the nets properly.

In terms of area, most of the researchers have put their research focus on developing the more efficient basic FPGA blocks such as adder, multiplier and other basic circuits. Some important developments on adder circuit have been patented during last three decades [1]-[5].

Saha et al proposed various integer arithmetic algorithms [6]. Simultaneous design of multiplier-free filters and hardware implementation in FPGA had been proposed by Shajaan et al [7]. Some other recent developments on area efficiency of FPGA implementation had been reported; efficient FIR for high speed FPGA implementation [8], enhancing area efficiency of hard circuit using shadow clusters [9], a method of reducing area based on fully-parallel stochastic LDPC decoding [10], efficient convolution implementation [11], area efficient for logic element and efficient area for floating point [12].

For a huge and complex arithmetic calculation, it is better to perform all arithmetic operations in the integer format. However, it is required special care in decrypting the nets on the codes. Thus the efficient area using embedding block of optimal circuits may be obtained. The paper presents a novel method for area efficiency of arithmetic operations using IEEE (Numeric_std 1076.3) package library.

The rest of this work is organized as follows. In section II describes about FPGA programming. The following section explains the style of VHDL programming for area efficiency in arithmetic operations. Implementation results and discussions are covered in section IV. Calculation and possible future works are presented in section V. the VHDL codes of the designed circuit are listed at the end of this article.

II. FPGA PROGRAMMING

To configure the behavior of the FPGA, the designer provides an HDL or a schematic design. The HDL form is more suitable to work with large structures since it is possible to just specify them numerically rather than having to draw every piece by hand. Still, the schematic entry can allow for easier visualization of a design.

There are also many ways of how to program an FPGA chips through HDL. For area efficiency and faster speed
purposes, most of scientists and engineers prefer VHDL and Verilog languages. These codes provide more flexibility and capability for programmers to program the specific circuits and able to maintain (control) detail nets connections between basic gates [13]-[16].

A. Verilog

The most common used HDL language in the design, verification and implementation of digital chips at level of register transfer is Verilog. This language is also used for verification of analog and mixed signal circuits [13].

Verilog was invented by Phil Moorby and Prabhu Goel in 1983. This language is the first modern Hardware Description Language (HDL). Initially, the language was intended to explain (describe) and allow to run simulation. Later on, it was also supported for performing synthesis.

The code of a two simple flip-flop in Verilog as follows:

```verilog
module top level(clock, reset);
    input clock;
    input reset;
    reg ff1;
    reg ff2;
    always@(posedge reset or posedge clock)
        if(reset)
            begin
            ff1 <=0;
            ff2 <=1;
            end
        else
            begin
            ff1 <= flop2;
            ff2 <= flop1;
            end
    end module
```

B. VHDL

VHDL or VHSIC (Very High Speed Integrated Circuit) HDL is software (language) to describe hardware used in EDA (electronic design automation). The language is used to describe digital and mixed signal systems like IC (Integrated Circuit) and FPGA [13]-[16].

Generally, VHDL is used to write text modules to describe a logic circuit. Then a simulation program is required to test the logic design. In order to simulate the design, additional file called test bench is required. Some FPGA vendors provide a more user friendly test bench generation by providing it in terms of Graphical User Interface (GUI) [13],[14].

Xilinx ISE, Altera Quartus, Synopsys Symplyze and Menthor Graphics HDL Designer are some of major packet FPGA software. These packets require a VHDL code and its test bench only. The VHDL code of an AND gate as follows:

```vhdl
library IEEE;
useIEEE.std_logic_1164.all;
entity AND_gate is
    port(
        Input1 :in std_logic;
        Input2 :in std_logic;
        Output :out std_logic);
end entity AND_gate;
architecture AND_arc of AND_gate is
begin
    Output <= Input1 and Input2;
end architecture AND_arc;
```

C. VHDL Package

Every VHDL code must have at least one package or library. This basic library contains information of all basic gates required in for FPGA implementation. The package is IEEE.std_logic_1164.all, which is IEEE standard.

Arithmetic operations in VHDL are more easily to be performed in integer format. Many third party vendors provide packages for simplifying arithmetic operations. IEEE also provides this kind of library which is called Numeric_std 1076.3 (IEEE.NUMERIC_STD.ALL).

The package provides two ways of converting numbers between integer and standard logic vector either through signed or unsigned format. For certain purposes in arithmetic operations, the operations may also be performed in signed or unsigned formats only.

Table 1 shows conversion commands among integer, signed and unsigned format. Meanwhile, Table 2 views list of instructions for converting among standard logic vector, signed and unsigned format. By combining the commands in those tables, we are able to perform direct conversion between standard logic vector and integer format [17].

III. VHDL STYLE FOR AREA EFFICIENCY IN ARITHMETIC OPERATIONS

For illustrations, consider multi stages block circuit that shown in Fig. 1. The circuit operation may be explained by (1), (2) and (3). All word lengths size that used to connect all blocks is similar 6 bits.

\[ S_1 = (A \times B) \]  

<table>
<thead>
<tr>
<th>Table 1</th>
<th>LIST OF CONVERSION COMMANDS AMONG INTEGER, SIGNED AND UNSIGNED FORMATS</th>
</tr>
</thead>
<tbody>
<tr>
<td>From/ to</td>
<td>Integer Signed</td>
</tr>
<tr>
<td>Integer</td>
<td>Signed</td>
</tr>
<tr>
<td>Unsigned</td>
<td>to_integer()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
<th>LIST OF CONVERSION COMMANDS AMONG STANDARD LOGIC VECTOR, SIGNED AND UNSIGNED FORMATS</th>
</tr>
</thead>
<tbody>
<tr>
<td>From/ to</td>
<td>Standard logic vector Signed</td>
</tr>
<tr>
<td>Standard logic vector</td>
<td>std_logic_vector()</td>
</tr>
<tr>
<td>Signed</td>
<td>std_logic_vector()</td>
</tr>
<tr>
<td>Unsigned</td>
<td>std_logic_vector()</td>
</tr>
</tbody>
</table>
\[ S_2 = C + D \]  
\[ Z = S_1 - S_2 \]

For area efficiency, the word lengths of circuit shown in the Fig. 1 may be redesigned. Suppose word lengths of the circuit have been redesigned and the new word lengths are shown in Fig. 2.

The circuit in the Fig. 2 occupies less area when it is implemented into chips. This is due to the circuit in the Fig. 1 has similar and bigger word lengths. Therefore, it is required to design the efficient VHDL code that minimized area use. It can be done by assigning suitable and might be different word lengths block interconnects.

A design in VHDL consists at least an entity which describes the interface and an architecture which contains the actual implementation. Moreover, most circuit designs import library modules. Some circuit designs may also contain multiple architectures and configurations [14].

Any input and output data or numbers in VHDL are considered as bit, standard logic or standard logic vector format. Bit format provides less choice 0 or 1. However, standard logic format extend the choice up to 9 logic values (U, X, 0, 1, Z, W, H, L, -). A parallel data may be represented using standard logic vector format [15].

Inside the architecture, the values (data) might be converted to other formats such as signed, unsigned or integer. To simplify the code for a complex arithmetic operation, we prefer integer format for processing most of the arithmetic operations. For instance, a number (A) represented in 4-bit standard logic vector, may be converted into integer (B) and Vice Versa as follows:

\[ B \leftarrow \text{to integer} \left( \text{signed}(A) \right) \]
\[ A \leftarrow \text{std_logic_vector} \left( \text{to_signed}(B,4) \right) \]

All variables and signals that are defined in integer format inside architecture, may or may not be limited (specific range). For example,

signal Z : integer;
variable S : integer range -16 to 15;

IV. IMPLEMENTATIONS AND DISCUSSIONS

In order to show the performance of the proposed VHDL coding style, we choose two circuits; 4-bit addition and 8-bit counter. Both of the circuits have been synthesized using Xilinx ISE 9.2i.

A. Four-bit Addition

Fig. 3 shows a circuit for adding two values A and B controlled by clock. Both input numbers are represented in 4 bits (standard logic vector format). Meanwhile, clock may be represented using bit or standard logic format (the code listed in the Program 1).

Both input values are stored separately in two buffers (4 bit). Each times clock goes high, both of the stored values are adding up. The addition result is saved into a 5-bit buffer this is due to additional 1 bit because of addition process of two values. Fig. 4 shows behavior simulation result of 4-bit addition circuit. All values are viewed in signed number format.

Synthesis result below views report macro statistic of the circuit with no limitation of integer value.

```
HDL Synthesis Report (Macro Statistics)
# Adders/Subtractors : 1
  5-bit adder : 1
# Registers : 3
  4-bit register : 2
  5-bit register : 1
```

It can be seen that the second synthesis result require smaller register (5-bit) compare than the first one which is 32-bit register. This is the maximum register size available since the software was run under 32-bit windows.

B. Eight-bit Counter

An 8-bit Counter has been designed and synthesized using the proposed VHDL coding style (the codes are listed in the Program 2). Figs. 5 and 6 show behavior simulation results of 8-bit Counter circuit. Output is viewed in unsigned number format.

Synthesis results below views report macro statistic and device utilization summary of the circuit with no limitation.
of integer value (the first code in Program 2).

HDL Synthesis Report (Macro Statistics)
# Counters : 1
32-bit up counter : 1

Device utilization summary:
Number of Slices: 21 out of 768 2%
Number of Slice Flip Flops: 32 out of 1536 2%
Number of 4 input LUTs: 40 out of 1536 2%
Number of IOs: 9
Number of bonded IOBs: 9 out of 124 7%
Number of GCLKs: 1 out of 8 12%

HDL Synthesis Report (Macro Statistics)
# Counters : 1
8-bit up counter : 1

Device utilization summary:
Number of Slices: 6 out of 768 0%
Number of Slice Flip Flops: 8 out of 1536 0%
Number of 4 input LUTs: 11 out of 1536 0%
Number of IOs: 9
Number of bonded IOBs: 9 out of 124 7%
Number of GCLKs: 1 out of 8 12%

Synthesis results below views report macro statistic of the circuit by putting range for all values that defined in integer format (the modified code in Program 2).

Device utilization summary:
Number of Slices: 6 out of 768 0%
Number of Slice Flip Flops: 8 out of 1536 0%
Number of 4 input LUTs: 11 out of 1536 0%
Number of IOs: 9
Number of bonded IOBs: 9 out of 124 7%
Number of GCLKs: 1 out of 8 12%

Second (modified) VHDL code (second codes in Program 1 and Program 2) result far less area compare than the first one. Because there is no limitation of integer, the first VHDL code require 32-bit counter. In terms of device utilization, the first code requires four time large area than the first code.

V. CONCLUSIONS AND FUTURE WORKS
A novel method for area efficiency in FPGA implementation for multi stages circuits has been proposed. All arithmetic operations in the proposed method are done in integer format since it is suitable for huge number of operations.

Two types of VHDL coding have been simulated and
synthesized. The second VHDL code which limits integer range is better than the first one. These codes are suitable for implementing in multi stages circuit. Synthesize result for counter code shown that the second method utilizes only one-forth area compare than the first one.

Future works may be conducted, various arithmetic operations may be observed. Also for multiple stages calculation, analysis range of values involved may be done.

REFERENCES


Program 1: VHDL Code of 4-bit Addition:

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.NUMERIC_STD.ALL;
4 entity Addition is
5   Port ( clock : in std_logic;
6         A, B : in std_logic_vector (3 downto 0);
7         C : out std_logic_vector (4 downto 0));
8 end Addition;
9 architecture Behavioral of Addition is
10   signal buffer_A, buffer_B  : integer;
11   signal buffer_C  : integer;
12 begin
13   process(clock)
14     begin
15       if clock'event and clock = '1' then
16         buffer_A <= to_integer(signed(A));
17         buffer_B <= to_integer(signed(B));
18       end if;
19       if clock'event and clock = '0' then
20         buffer_C <= buffer_A + buffer_B;
21       end if;
22       C <= std_logic_vector(to_signed(buffer_C,5));
23   end process;
24 end Behavioral;
```

For representing circuit in the Fig.3, lines 10 and 11 have been modified for area efficiency:

```
10   signal buffer_A, buffer_B  : integer range -8 to 7;
11   signal buffer_C  : integer range -16 to 15;
```
Program 2: VHDL Code of 8-bit Counter:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Counterr is
  Port ( clock : in std_logic;
        Output : out std_logic_vector (7 downto 0));
end Counterr;

architecture Behavioral of Counterr is
  signal Count : integer;
begin
  process(clock)
  begin
    if clock'event and clock = '1' then
      Count <= Count + 1;
      if Count = 255 then
        Count <= 0;
      end if;
    end if;
    Output <= std_logic_vector(to_unsigned(Count,8));
  end process;
end Behavioral;
```

The code at line 9 has been modified for area efficiency:

```vhdl
9    signal Count : integer range 0 to 255;
```