HOMEWORK #1
RTL Design

Ali Alipour Fraydani
810101233
alipoura364@gmail.com

Ali
Alipour
(Fraydani)

We have an unsigned 8-bit multiplier (inA[7:0], inB[7:0], outW[15:0]) that we want to use for approximating multiplication of two 16-bit operands, opndA[15:0] and opndB[15:0]. The result is to become available on the 32-bit output, rsltW[31:0]. A simple and very inaccuruate way of doing the multiplication is to take eight most significant bits of opndA and opndB, multiply them using the 8-bit multiplier and use the result as the most significant 16 bits of rsltW with 16 zeros to its right.

VHDL, Datapath, Controller, VAM16, State Diagram

I. Introduction

In this homework we want to describe and 32 multiplier by 8 bit multiplier this multiplication is very inaccurate but the most of time we don't want accuracy but want to approach to real result.

II. Dathapath and Controller

A. *Datapath*

First one talk about design VAM16 (A Valuable 16-bit Approximate Multiplier). My design has a datapath and controller that I want to explain it, datapath shown in Figure 2.1, that has 3 registers that have different functionality, Register A have same functionality of Register B but concatenation Register is different. Data from ABus and BBus place to Register A and Register B by load command that issued by controller, therefore 16 bit send to a combinational logic named 1's Detector, 1's Detector, detect first one from the left hand side then 8-bit data select the input and send to its output to addition of special slice input counting number of zeros to arrive first 1, and it count send by Count_A or Count_B to the next level.

Datapath

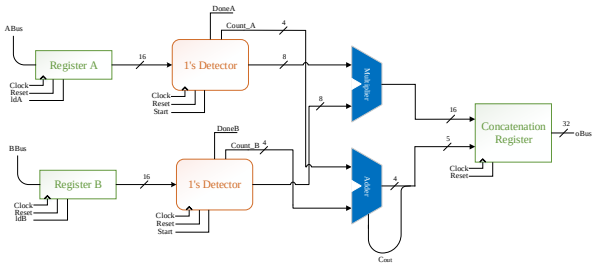

Figure II:I

8-bit of output of 1's Detector that send from Register A and Register B take to Multiplier and output of it is 16-bit that send to Concatenation Register.

Count that calculate by 1's Detector is number of zeros that we skipped, every two count must be add, this operation perform by Adder, because of addiction of two operands perhaps have beyond that 15 and overflow concatenate output by Cout and send to Concatenation Register. In Concatenation Register place result of Multiplier to output register but distance of left hand side specified by result of Adder Unit.

B. *Controller*

Now we talk about controller that designed. Controller shown in Figure 2.2

Controller

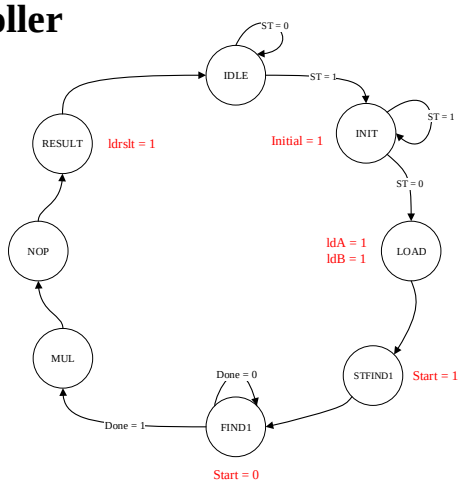

Figure II:II

In this controller we are in state IDLE, ST (start) have to toggle to one and toggle to zero for 1 cycle that controller started, after ST = 1 we are go to state INIT that initial signal has issued, initial signal is reset all register that apply to Datapath, then if ST = 0 go to LOAD state in LOAD, ldA and ldB issued that mean ABus and BBus load to output of its registers. STFIND1 active start for 1's Detector start its activity for detect 1, in state FIND1 we stay until Done = 1 that mean component find first 1 and go to state MUL that perform multiplication operation, one cycle delay for multiplication perform multiple completely that its purpose provide by NOP state that do not special operation in this state. Then result provided and we load this result in Concatenation register by state RESULT.

III. Code Description (Component)

First one we describe a register, Figure 3.1 shown entity of register.

```
entity Reg is

   GENERIC
   (
      busWidth :  INTEGER  := 16
   );

   PORT
   (
      Clock    :  IN    STD_LOGIC;
      Reset    :  IN    STD_LOGIC;
      load     :  IN    STD_LOGIC;
      inBus    :  IN    unsigned(busWidth-1 DOWNTO 0);
      outBus   :  OUT   unsigned(busWidth-1 DOWNTO 0)
   );

end Reg;
```

Figure III:I

Architecture of register shown in Figure 3.2 if load issued then input send to output and if Reset issued output is zero.

```
architecture Behavioral of Reg is

   Signal inBus_Int  :  unsigned(busWidth-1 DOWNTO 0) := (others=>'0');
   Signal outBus_Int :  unsigned(busWidth-1 DOWNTO 0) := (others=>'0');

begin

   outBus   <= outBus_Int;

   process(Clock)
   begin

      if rising_edge(Clock) then

         inBus_Int   <= inBus;

         if (load = '1') then
            outBus_Int  <= inBus_Int;
         end if;

         if (Reset = '1') then
            outBus_Int  <= (others=>'0');
         end if;

      end if;

   end process;

end Behavioral;
```

Figure III:II

Next component we describe is 1's Detector, every 1's Detector has four inputs Clock, Reset, Start and input and five output that names is sellower, selupper (this two port are optional remain from before my wrong design) Done, count_OUT and output, Figure 3.3 shown entity of 1's Detector.

```
entity Detect_1 is

   PORT
   (
      Clock       :  IN    STD_LOGIC;
      Reset       :  IN    STD_LOGIC;
      Start       :  IN    STD_LOGIC;
      sellower    :  OUT   STD_LOGIC;
      selupper    :  OUT   STD_LOGIC;
      Done        :  OUT   STD_LOGIC;
      Count_OUT   :  OUT   unsigned(03 DOWNTO 0);
      input       :  IN    unsigned(15 DOWNTO 0);
      output      :  OUT   unsigned(7  DOWNTO 0)
   );

end Detect_1;
```

Figure III:III

This module has an asynchronous Reset that by reset all signal and counter is zeros that shown in Figure 3.4.

```
if (Reset = '1')  then
   found          <= '0';
   Done_Int       <= '0';
   sellower_Int   <= '0';
   selupper_Int   <= '0';
   Count          <= (others=>'0');
end if;
```

Figure III:IV

If start is issued counter that show skip zeros is zero and module start to detect first 1. This state shown if Figure 3.5.

```
if (Start = '1') then
   Start_Int      <= '1';
   Done_Int       <= '0';
   count          <= (others=>'0');
   output_Int     <= (others=>'0');
end if;
```

Figure III:V

From right hand side if find first 1, found signal issued and start inactive because now we can start again and issued start for another set of numbers. Figure 3.6 show how to detect 1.

```
if (input(15-to_integer(Count)) = '1' AND Start_Int = '1') then
   found          <= '1';
   Start_Int      <= '0';
end if;
```

Figure III:VI

If do not found and start issued we have to walk to bits from left for this approach we have to a increment counter Figure 3.7 shown how to increment this counter.

```
if (input(15-to_integer(Count)) = '1' AND Start_Int = '1') then
   found          <= '1';
   Start_Int      <= '0';
end if;
```

Figure III:VII

If found issued, take Done means first 1 founded. If counter greater than 8 select (number of skip bit from the left is beyond 8 or first 1 is in LSB). Output take lower value 8-bit otherwise first 1 until 8 bit after assign to output. This statement shown in Figure 3.8.

```
if (found = '1' AND Done_Int = '0') then
   sellower_Int   <= found;
   selupper_Int   <= NOT found;
   Done_Int       <= '1';
   found          <= '0';
   if (count-1 > 8) then
      output_Int     <= input(7 DOWNTO 0);
   else
      output_Int     <= input((16-to_integer(Count)) DOWNTO (16-to_integer(Count)-7));
   end if;
end if;
```

Figure III:VIII

Multiply and Adder is very simplify that shown in Figure 3.9 and Figure 3.10.

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

entity Multiplier is

   PORT
   (
      InputA   :  IN    unsigned(07 DOWNTO 0);
      InputB   :  IN    unsigned(07 DOWNTO 0);
      Output   :  OUT   unsigned(15 DOWNTO 0)
   );

end Multiplier;

architecture Behavioral of Multiplier is

begin

   Output   <= InputA * InputB;

end Behavioral;
```

Figure III:IX

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

entity Adder is

   PORT
   (
      Cout     :  OUT   STD_LOGIC;
      InputA   :  IN    unsigned(3 DOWNTO 0);
      InputB   :  IN    unsigned(3 DOWNTO 0);
      Output   :  OUT   unsigned(3 DOWNTO 0)
   );

end Adder;

architecture Behavioral of Adder is

   Signal Sum  :  unsigned(4 DOWNTO 0) := (others=>'0');

begin

   Sum      <= resize(InputA, 5) + InputB;
   Output   <= Sum(3 DOWNTO 0);
   Cout     <= Sum(4);

end Behavioral;
```

Figure III:X

In Figure 3.10 used resize function because inputA and InputB are 4-bit but we want to assign to a 5-bit signal. VHDL take error this unlike Verilog.

In Concatenation Register we take result of Multiplier and Adder (number of first one position from two operand) and concatenate between 32-bit number like than Homework Question and Python code provided. Figure 3.11 shown how to describe Concatenation Register.

```
process(Clock)
begin

   if rising_edge(Clock) then
      input16_Int <= input16;
      input05_Int <= input05;
      if(input05(4) = '1') then
         input05_Int <= to_unsigned(16, 5);
      end if;
      if (load = '1') then
         Output32_Int((31-to_integer(input05_Int)) DOWNTO (16-to_integer(input05_Int))) <= input16_Int;
      end if;
      if (Reset = '1') then
         Output32_Int   <= (others=>'0');
      end if;
   end if;

end process;
```

Figure III:XI

IV. Code Description (Controller)

As you can see in Figure 4.1 we describe states by Constant in VHDL and take issued the signals by Controller design that shown in Figure 2.2.

```
Constant IDLE     :  unsigned(2 DOWNTO 0) := "000";
Constant INIT     :  unsigned(2 DOWNTO 0) := "001";
Constant LOAD     :  unsigned(2 DOWNTO 0) := "010";
Constant FIND1    :  unsigned(2 DOWNTO 0) := "011";
Constant STFIND1  :  unsigned(2 DOWNTO 0) := "100";
Constant MUL      :  unsigned(2 DOWNTO 0) := "101";
Constant RESULT   :  unsigned(2 DOWNTO 0) := "110";
Constant NOP      :  unsigned(2 DOWNTO 0) := "111";

Signal present_state :  unsigned(2 DOWNTO 0) := (others=>'0');
Signal next_state    :  unsigned(2 DOWNTO 0) := (others=>'0');
```

Figure IV:I

Figure 4.2 show transition state.

```
case (present_state) is

   WHEN IDLE   =>
      if (ST = '1') then
         next_state  <= INIT;
      else
         next_state  <= IDLE;
      end if;
   WHEN INIT   =>
      if (ST = '1') then
         next_state  <= INIT;
      else
         next_state  <= LOAD;
      end if;
   WHEN LOAD   =>
      next_state     <= STFIND1;
   WHEN STFIND1   =>
      next_state     <= FIND1;
   WHEN FIND1  =>
      if (Done = '1') then
         next_state  <= MUL;
      else
         next_state  <= FIND1;
      end if;
   WHEN MUL    =>
      next_state     <= NOP;
   WHEN NOP    =>
      next_state     <= RESULT;
   WHEN RESULT =>
      next_state     <= IDLE;
   WHEN OTHERS =>
      NULL;
end case;
```

Figure IV:II

Figure 4.3 shown how to issued signals.

```
process(present_state)
begin
   Initial  <= '0';
   ldA      <= '0';
   ldB      <= '0';
   Start    <= '0';
   ldrslt   <= '0';
   case (present_state) is

      WHEN INIT   =>
         Initial  <= '1';
      WHEN LOAD   =>
         ldA      <= '1';
         ldB      <= '1';
      WHEN STFIND1   =>
         Start    <= '1';
      WHEN FIND1  =>
         Start    <= '0';
      WHEN RESULT =>
         ldrslt   <= '1';
      WHEN OTHERS =>
         NULL;
   end case;
end process;
```

Figure IV:III

V. Code Description (Datapath)

Datapath create by connected components, note this modules instants by generating instantiation for example.

```
Detect_A_1: ENTITY WORK.Detect_1
PORT MAP(
    Clock       => Clock,
    Reset       => Reset,
    Start       => Start,
    sellower    => sellowerA,
    selupper    => selupperA,
    Done        => DoneA,
    Count_OUT   => Count_A,
    input       => AReg,
    output      => DetecA_OUT
);
```

Figure V:I

VI. Simulation Result

We define four set of number for apply to multiplier every multiply shown by Python sample.
Four sets shown in Figure 6.1.

```
Reset <= '1';
wait for Clock_period;
Reset <= '0';

ABus     <= to_unsigned(500, 16);
BBus     <= to_unsigned(718, 16);
wait for Clock_period;

ST <= '1';
wait for Clock_period;
ST <= '0';

wait for Clock_period*25;

Mymem(09)   <= resize(ABus, 32);
Mymem(10)   <= resize(BBus, 32);
Mymem(11)   <= oBus;
```

```
Reset <= '1';
wait for Clock_period;
Reset <= '0';

ABus     <= to_unsigned(35, 16);
BBus     <= to_unsigned(45, 16);
wait for Clock_period;

ST <= '1';
wait for Clock_period;
ST <= '0';

wait for Clock_period*25;

Mymem(06)   <= resize(ABus, 32);
Mymem(07)   <= resize(BBus, 32);
Mymem(08)   <= oBus;
```

```
Reset <= '1';
wait for Clock_period;
Reset <= '0';

ABus     <= to_unsigned(421, 16);
BBus     <= to_unsigned(411, 16);
wait for Clock_period;

ST <= '1';
wait for Clock_period;
ST <= '0';

wait for Clock_period*25;

Mymem(03)   <= resize(ABus, 32);
Mymem(04)   <= resize(BBus, 32);
Mymem(05)   <= oBus;
```

```
Reset <= '1';
wait for Clock_period;
Reset <= '0';

ABus     <= to_unsigned(25, 16);
BBus     <= to_unsigned(411, 16);
wait for Clock_period;

ST <= '1';
wait for Clock_period;
ST <= '0';

wait for Clock_period*25;

Mymem(00)   <= resize(ABus, 32);
Mymem(01)   <= resize(BBus, 32);
Mymem(02)   <= oBus;
```

Figure VI:I

Simulation result by ISIM shown in Figure 6.2.

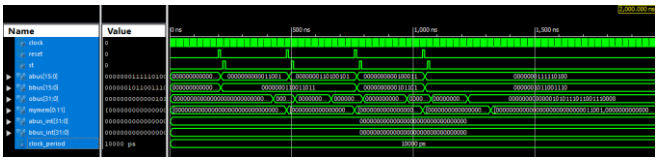

Figure VI:II

For better comparison we store inputs and outputs in a memory and compare by Python code than show in Figure 6.3.

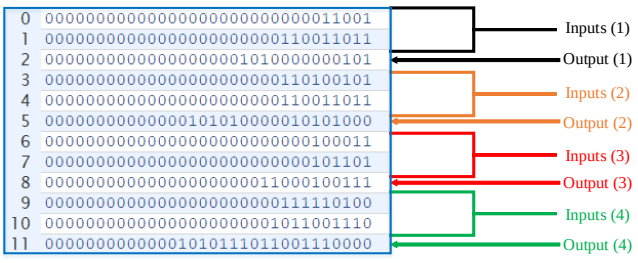

Figure VI:III

Result of Python code shown in Figure 6.4.

```
The Result is: 00000000000000000001010000000101

The Result is: 00000000000000101010000010101000

The Result is: 00000000000000000000011000100111

The Result is: 00000000000001010111011001110000
```

Figure VI:IV

VII. VAM16 (Top Module)

Top Level Module include Datapath and Controller that we describe Connect intersect port between Datapath and Controller Figure 7.1 show Top Module and connection between Datapath and Controller.

```
DP : ENTITY WORK.DataPath
PORT MAP
(
    Clock     => Clock,
    Reset     => Initial,
    ldA       => ldA,
    ldB       => ldB,
    ldrslt    => ldrslt,
    Start     => Start,
    Done      => Done,
    ABus      => ABus,
    BBus      => BBus,
    oBus      => oBus
);

CU : ENTITY WORK.Controller
PORT MAP(
    Clock     => Clock,
    Reset     => Reset,
    Done      => Done,
    ST        => ST,
    Initial   => Initial,
    ldA       => ldA,
    ldB       => ldB,
    ldrslt    => ldrslt,
    Start     => Start
);
```

Figure VII:I

EEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.