School of Engineering

Department of Electrical and Electronic Engineering

Microelectronics: Systems and Devices: MSc

Mod 3 Checking circuit for 8-bits ALU with VHDL Report

X.Liu

January 2020

Mod 3 Checking circuit for 8-bits ALU with VHDL Report

Xinjie Liu
School of Engineering
Newcastle University, NE1 7RU
United Kingdom

X.Liu81@newcastle.ac.uk

1) Introduction:

In this VHDL project, an 8-bit ALU and the Mod 3 Residue Checking circuit is built. The ALU focuses on the exact operation, and the Mod 3 Residue Checking circuit is made for approximate operation.

From this project, we can be familiar with VHDL Behavioral Level syntax and semantics. At the same time, Built-in Self Test, Test Pattern Generation, Behavioral Level Fault Modelling, Stuck-a-1/0 and other Design For Testability method are deployed in this project.

In the Technical Sections, I would explain how different blocks of the circuits' work, how circuit structures and how to use different methods to build it. After that, the Result Section will present the most important part of my codes and the waveform from each simulation. For the Discussion Section, I will discuss the advantages and disadvantages of my circuit. The debug method will also be included in this section. The conclusion is the last section in my report. I put some comparisons on the project aims and my result. The merit and limitation of my circuits are introduced in this part.

2) Technical Sections:

In this project, I divided the system into 7 parts; including ALU, Residue Generator, AND Gate and 3 Residue Generator, Selection Circuit, Mod 3 Adders and Comparator.

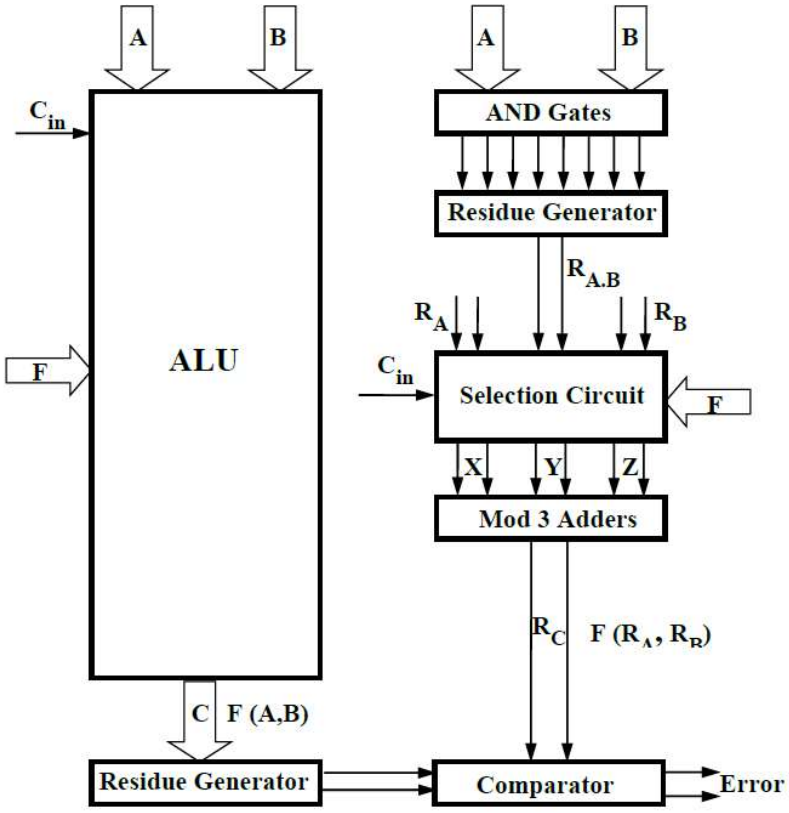

Figure 1: Project block diagram

I. ALU

This block I used case statement for selecting different modes of calculation. However, we noticed that this ALU had "addition" and "subtraction" operation, which must include the library "ieee.NUMERIC_STD.all". Another important thing is that the carry out operation needed to be deployed.

II. Residue Generator

The easiest approach to build a Residue Generator in VHDL was to use Behavioral Descriptions. In most of program languages, "Mod" is a function to create residue. Same with VHDL, I used mod 3 for generating residue; however, only integer can be "mod", so I needed to use "to_integer" and "to_unsigned" function to transform the type of data.

III. AND Gate and 3 Residue Generator

Same as Residue Generator, I used Mod 3 function in this block, but just adding an AND gate before doing modulus.

IV. Selection Circuit

This block is just a lite version of ALU. I also used case statement in this circuit. One thing needs to be noticed was that after all the cases, the other input should be set to "null" output. This can avoid some random mistakes while testing circuit.

V. Mod 3 Adders

Overall, the Mod 3 Adders is an addition circuit. Firstly, two of 2bits signal added together; If the result is 3 bits, take the carry out number down and doing the addition again; until there is only a 2bits signal. Secondly, if the result is "11", which is "3" in unsigned decimal, with the residue of 3, then that should be "0"; so, adding an if case makes the output from "3" to "0".

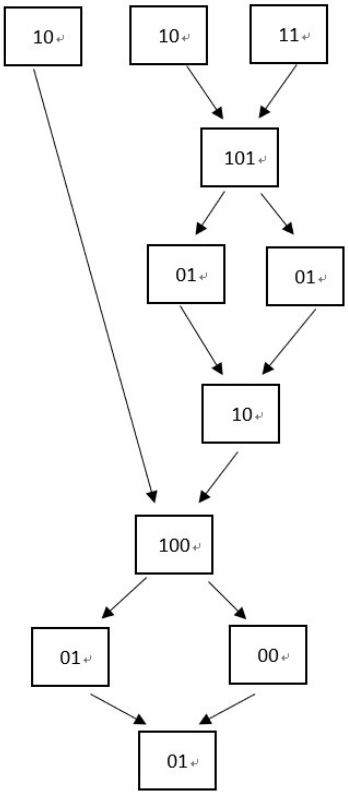

Figure 2: Mod 3 Adders example

VI. Comparator

Briefly, we used if statements in building the comparator, and there are 3 situations: Greater, Less and Equal. Each Comparator cell can provide one bit of compare feature; so I used 2 comparators and port map to link them together to build up the 2 bits comparator. When the Carry In is not zero, the output will be the Carry In input. When Carry In is zero, the output is determined by those two Inputs that compare with each other.

3) Results:

All the codes have been hosted by GitHub: https://git.io/JejM0

Phase 1: Basic ALU Implementation

In this ALU Design, we should notice the Carry Out; The & operation is used to concatenate a new element to an array. If we ignore to & a "0", when there is no Input, the circuit would result in error. For a 8bits ALU, the Carry Out is the ninth bit of the result; We can just use temp(8) to assign the Carry Out or temp(8 downto 3) to assign a 3bits logic vector.

```
architecture Behavioral of ALU is

begin
   process(A,B,ALU_Sel)
 begin
  case(ALU_Sel) is
  when "000" => -- Addition
   ALU_Result <= A + B ;
  when "001" => -- Subtraction
   ALU_Result <= A - B ;
  when "010" => -- Increase
   ALU_Result <= A + 1 ;
  when "011" => -- Decrease
   ALU_Result <= A - 1 ;
  when "100" => -- Logical and
   ALU_Result <= A and B;
  when "101" => -- Logical or
   ALU_Result <= A or B;
  when "110" => -- Logical xor
   ALU_Result <= A xor B;
  when "111" => -- Logical nor
   ALU_Result <= A nor B;
  when others => ALU_Result <= A + B ;
  end case;
 end process;
 ALU_Out <= ALU_Result; -- ALU out
  tmp <= ('0' & A) + ('0' & B);
  Carryout <= tmp(8); -- Carryout flag
end Behavioral;
```

Code 1: 'When' Case part in 8bits ALU

In the ALU Test Bench, we used For Loop to reduce the complexity of code. After testing 7 functions in ALU, I tested the availability of Carry Out.

```
  begin
     -- hold reset state for 100 ns.
     A <= "01100110";
     B <= "00011011";
     ALU_Sel <= "000";

  wait for 100 ns;

  for i in 0 to 7 loop
   ALU_Sel <= ALU_Sel + "1";
   wait for 100 ns;
  end loop;
     A <= "11111111";
     B <= "11111111";
     wait;
```

Code 2: ALU Test Bench Loop description

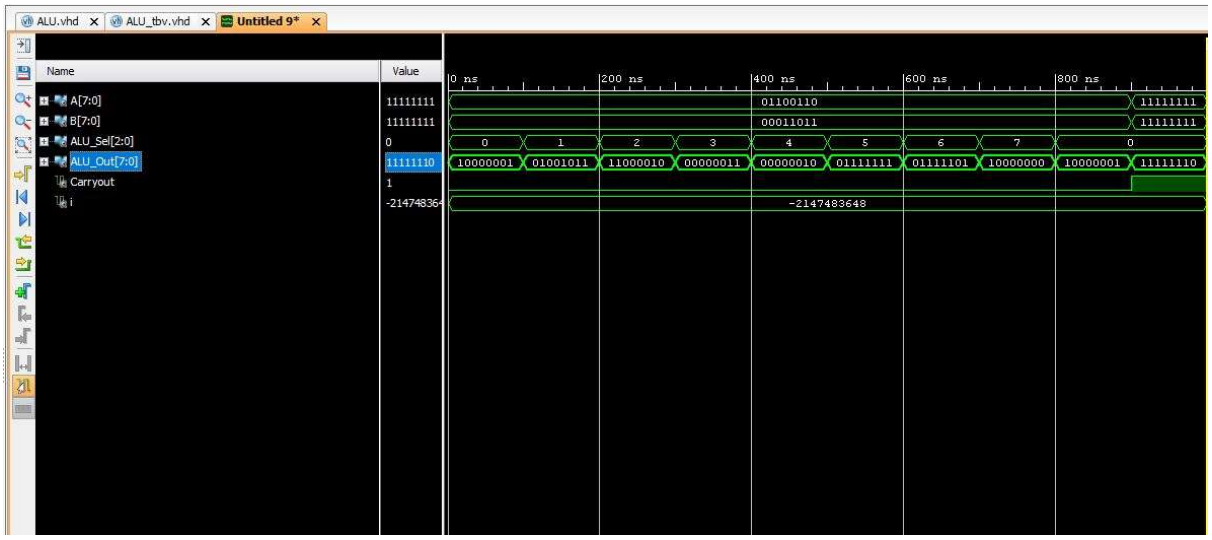

Figure 3: The correct waveform of an 8bits ALU

Phase 2: Behavioral Fault Injection

In this experiment, 3 different kinds of Behavioral Level Fault Modelling methods will be developed in the 8-bit ALU circuit, which are Assignment Control, Dead Process and Dead Clause.

I. Assignment Control

In Assignment Control, 2 highlighted Fault is injected, which are 001 and 100. In this Waveform, Aqua is the ALU output, Pink is the ALU Mode Selection Input. From it, after injecting the fault, the output signal is the same as the last action, with 000 and 001, 011 and 100 having the same output.

```
  when "000" => -- Addition
   ALU_Result <= A + B ;
  when "001" => -- Subtraction
   ALU_Result <= ALU_Result ;
  when "010" => -- Multiplication
   ALU_Result <= std_logic_vector(to_unsigned((to_integer(unsigned(A)) *
to_integer(unsigned(B))),8)) ;
  when "011" => -- Division
   ALU_Result <= std_logic_vector(to_unsigned(to_integer(unsigned(A)) /
to_integer(unsigned(B)),8)) ;
  when "100" => -- Logical and
   ALU_Result <= ALU_Result;
```

Code 3: Assignment Control Fault Inject

Figure 4: Assignment Control Fault Waveform

II. Dead Process

Dead Process is to replace the contents of the sensitivity list by a static signal. In this experiment, the Selection signal is set to 'Static_bit' and it is set to '0'. From the Waveform, all output is '0'.

The ALU changes from the following Code 4 to Figure 5.

```
process(A,B,ALU_Sel)
```

to

```
process(ALU_Sel)
```

```
begin
   -- hold reset state for 100 ns.
   A <= "01100110";
   B <= "00011011";
   ALU_Sel <= "000";

wait for 100 ns;

for i in 0 to 7 loop
 A <= A + "1";
 wait for 100 ns;
```

Code 4: Dead Process Testbench

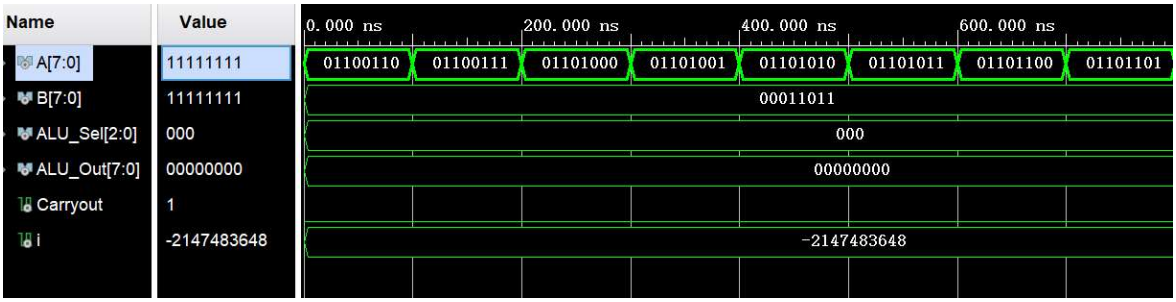

Figure 5: Dead Process Waveform

III. Dead Clause

This situation is one of the options in a 'case' statement failing to be executed. In this example, the addition and subtraction are removed. The Waveform shows ALU_Result having the same signal as A.

```
begin
  case(ALU_Sel) is
  when "000" => -- Addition
   ALU_Result <= A + B;
  when "001" => -- Subtraction
   ALU_Result <= A - B ;
```

Code 5: Origin Case Statement

```
begin
  case(ALU_Sel) is
  when "000" => -- Addition
   ALU_Result <= A ;
  when "001" => -- Subtraction
   ALU_Result <= A ;
```

Code 6: Dead Clause injected Case Statement

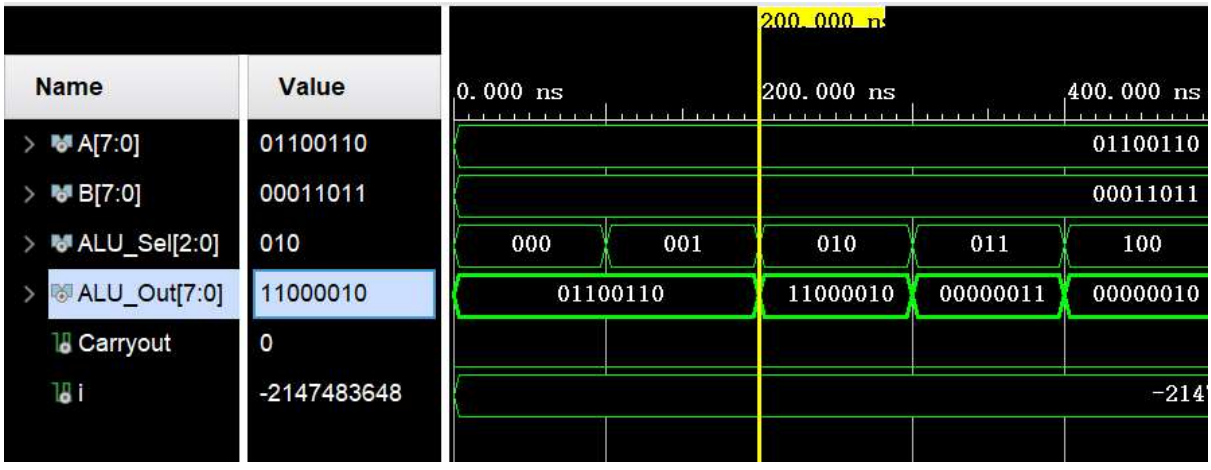

Figure 6: Dead Clause injected Waveform

Phase 3: Mod 3 Implementation

In my project, Mod 3 Residue Checking Circuit includes five blocks. They are Residue Generator, AND Gate and 3 Residue Generator, Selection Circuit, Mod 3 Adders and Comparator.

I. Residue Generator & AND gate and 3 Residue Generator

In this circuit, the type of signal needs to be recognized. Modulus function only works for integer signal, so we need to transform unsigned signal to integer.

```
      begin
    tmpAB <= A and B;
    tmpA <= A;
    tmpB <= B;
    RAB <= std_logic_vector(to_unsigned(((to_integer (unsigned(tmpAB)))mod
3),tmpRAB'length));  -- Residue for A AND B
    RA <= std_logic_vector(to_unsigned(((to_integer (unsigned(tmpA)))mod
3),tmpRA'length));  -- Residue for A
    RB <= std_logic_vector(to_unsigned(((to_integer (unsigned(tmpB)))mod
3),tmpRB'length));  -- Residue for B
    end;
```

Code 7: Residue Generator and AND gate VHDL code

```
      begin
          A <= X"35"; -- Giving Hexadecimal signal
          B <= X"3A";
      wait for 20 ns;
          A <= X"17";
          B <= X"DA";
      wait for 20 ns;
          A <= X"BF";
          B <= X"DE";
      wait;
```

Code 8: Testbench for Residue Generator and AND gate

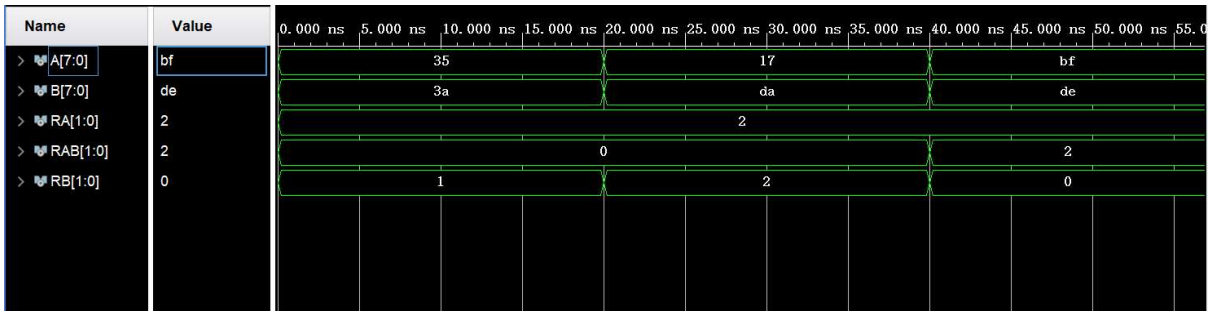

Figure 7: Residue Generator and AND gate Waveform

II. Selection Circuit

This circuit is choosing Component Residues for output. The simulation Waveform shows the output that meets the objectives.

ADD	Ra	Rb	Ci
SUB	Ra	Rb’	Ci
INCR.	Ra	00	01
DECR	Ra	00	10
AND	00	00	Ra.b
OR	Ra	Rb	Ra.b’
EXOR	Ra	Rb	Ra.b

Chart 1: Component Residues

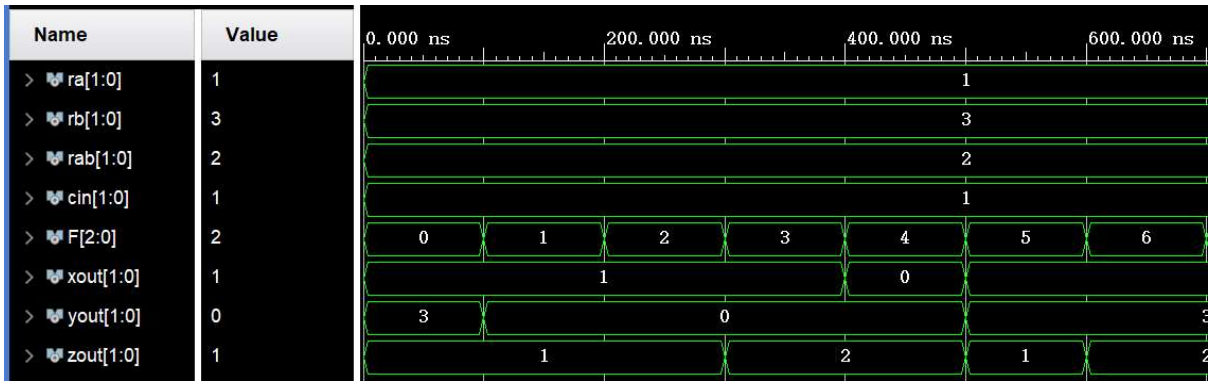

Figure 8: Selection Circuit Waveform

III. Mod 3 Adders

Briefly, this circuit is a 2 bits adder. When the adder has a third bit Carry Out, take it down and add it to the first and second bit. After the Addition, use When case to transform 3 to 0. The output Waveform then matches expectations.

```
architecture Behavior of Mod3adder is
------------------Temp for Input Signal---------------
signal tmpx, tmpy, tmpz :std_logic_vector(1 downto 0);

---------Storage for First and Second 2bits Result in Addition-------
signal tmp1, tmp2 :std_logic_vector(1 downto 0);

---------Storage for Result of Third bit + 2bits Result--------------
signal tmp13, tmp23 :std_logic_vector(1 downto 0);

---------3bits Result for 2bits + 2bits-------------------------
signal ltmp1,ltmp2 :std_logic_vector(2 downto 0);

   begin
      tmpx <= x;
      tmpy <= y;
      tmpz <= z;
      ltmp1 <= ('0' & tmpx) + ('0' & tmpy);
      tmp1 <= tmpx +tmpy;
      tmp13 <= tmp1 +ltmp1(2);     --Take the Carry Out bit + Result again
      ltmp2 <= ('0' & tmp13) + ('0' & tmpz);
      tmp2 <= tmp13 + tmpz;
      tmp23 <= tmp2 +ltmp2(2);     --Take the Carry Out bit + Result again

    process (tmp23) is             --Using case to force output 3 to 0
    begin                          --(3 Residue 3 should be 0)
    case tmp23 is
    when "11" => rc <= "00";
    when others => rc <= tmp23;
    end case;
    end process;
```

Code 9: 3 Input Mod 3 Adders code

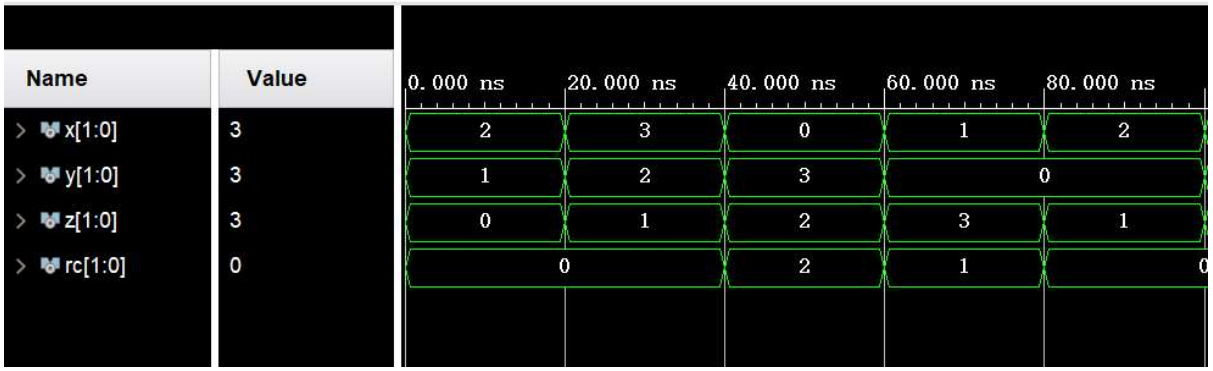

Figure 9: Selection Circuit Waveform

IV. Comparator

When A is larger than B, output is 1; when A is smaller than B, output is 2; when A is equal to B, output is 0.

```
architecture cmp_cell of cmp_cell is
    begin
    process (Cmpin,a,b) is
     begin
      if Cmpin = "10" then Cmpout <= "10";
      elsif Cmpin = "01" then Cmpout <= "01";
      elsif Cmpin = "00" and a=b then Cmpout <= "00";
      elsif Cmpin = "00" and a>b then Cmpout <= "01";
      elsif Cmpin = "00" and a<b then Cmpout <= "10";
      end if;
    end process;
end cmp_cell;
```

Code 10: Comparator Cell

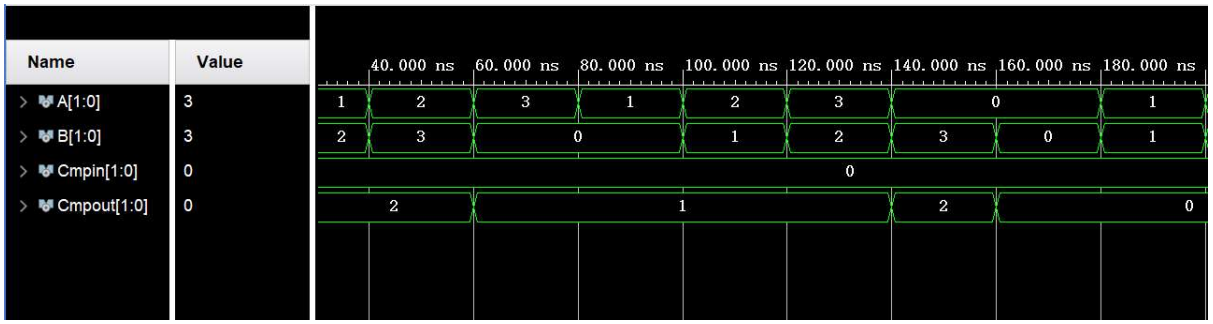

Figure 10: 2 bits Comparator Waveform

Phase 4: Complete Design Implementation

The last step for building this circuit is to connect those six circuits. Firstly, adding all the Component to one main circuit. Secondly, declaring Signal and using Port Map to connect Component with Signal. From the Error output signal, is 0 in any operation. That means ALU output is the same as Checking Circuit.

```
  --ALU to Residue Generator--
 signal AtoR :std_logic_vector(7 downto 0);
  --Residue Generator to Selection Circuit--
 signal RtoSa, RtoSb, RtoSab :std_logic_vector(1 downto 0);
  --Selection Circuit to Mod 3 Adders--
 signal StoMx, StoMy, StoMz :std_logic_vector(1 downto 0);
  --Mod 3 Adders to Comparator--
 signal MtoC :std_logic_vector(1 downto 0);
  --ALU Residue Generator to Comparator--
 signal RtoC :std_logic_vector(1 downto 0);
  --ALU Carry Out--
 signal cout :std_logic;
  --Comparator Carry In--
 signal ii :std_logic_vector(1 downto 0):=(others => '0');
begin


cell1:ALU port map (A ,B ,F ,AtoR ,cout);
cell2:residuenoand port map (AtoR ,RtoC);
cell3:residue_gene port map (A ,B ,RtoSa ,RtoSb ,RtoSab);
cell4:func_sel port map
(F ,cin ,RtoSa ,RtoSb ,RtoSab ,StoMx ,StoMy ,StoMz);
cell5:Mod3adder port map (StoMx ,StoMy ,StoMz ,MtoC);
cell6:comparator_2 port map (MtoC ,RtoC ,ii ,error);

end Behavioral;
```

Code 11: ALU and Checking Circuit

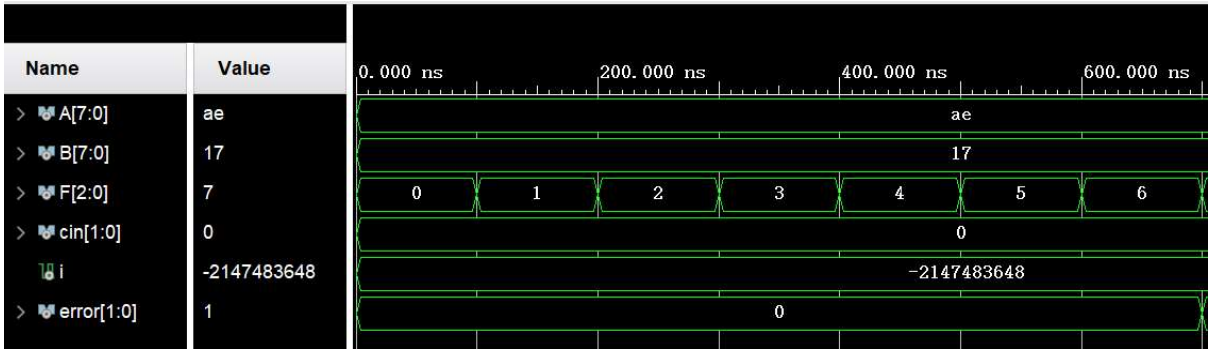

Figure 11: ALU and Checking Circuit checked Waveform

Fault injection for ALU and Checking Circuit

I. Comparator Stuck-then

Stuck-then fault will lead to a failure in 'else' statement. In this situation, the Process can only work after 'Then'. The Comparator Cell was injected in this experiment.

```
begin
 if Cmpin = "10" then Cmpout <= "10";
 elsif Cmpin = "01" then Cmpout <= "01";
 --elsif Cmpin = "00" and a=b then Cmpout <= "00";
 --elsif Cmpin = "00" and a>b then Cmpout <= "01";
 --elsif Cmpin = "00" and a<b then Cmpout <= "10";
 -----------------Stuck-Then-Fault-------------
 elsif (True) and a=b then Cmpout <= "00";
 elsif (True) and a>b then Cmpout <= "01";
 elsif (True) and a<b then Cmpout <= "10";
 ------------------------------------------
 end if
```

Code 12: Inject Stuck-then to Comparator

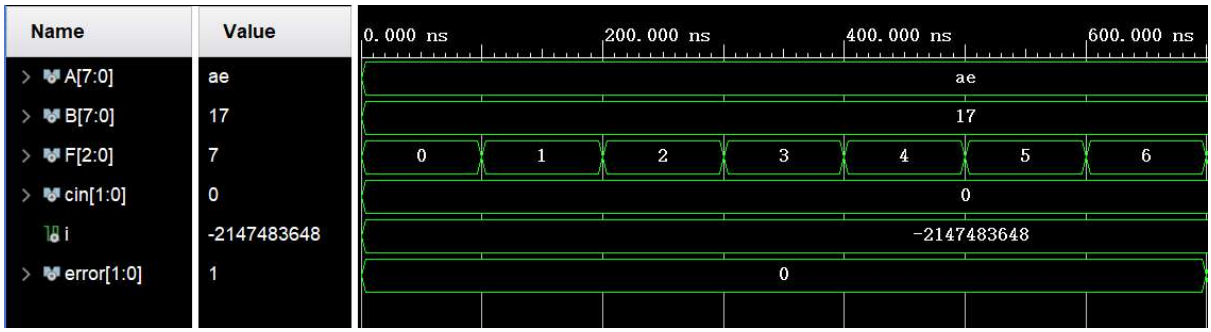

Figure 12: Inject Stuck-then to Comparator Waveform

II. Local Stuck

Local Stuck is the failure of a signal, variable or constant to contain the correct values in local cell. In this experiment, I injected 3 Local Stuck errors into Selection Circuit. Waveform shows Logical AND's output will not be affected. Increase and Decrease operations have Error output.

```
-- Increase
when "010" => xout <= ra ;
--yout <= "00" ;
yout <= "01" ; --Local Stuck
zout <= "01";

-- Decrease
when "011" => xout <= ra ;
--yout <= "00" ;
yout <= "01" ; --Local Stuck
zout <= "10";

-- Logical and
when "100" => xout <= "00" ;
--yout <= "00" ;
yout <= "11" ; --Local Stuck But do no affect to result
zout <= rab;
```

Code 13: Inject Local Stuck error to Selection Circuit

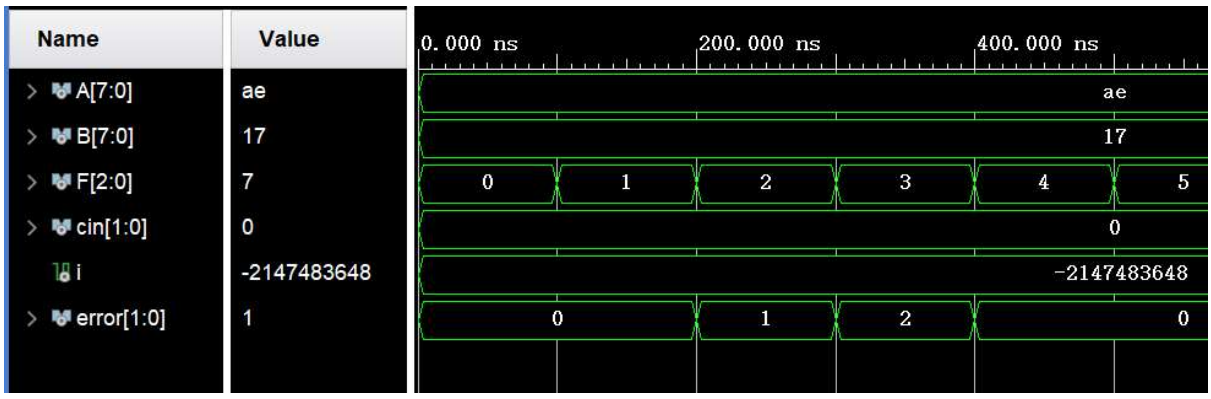

Figure 13: Inject Local Stuck error to Selection Circuit Waveform

III. Global Stuck

Global Stuck is also the failure of signal, variable or constant to contain the correct value. However, this value is Globally. Though there is no Global Signal in VHDL, I assumed the value in Top Cell is 'Global Value'. In this experiment, I injected 3 Global Stuck errors in my Top Cell, "main"; They are:

Signal from Residue Generator to Selection Circuit A

Signal from ALU Residue Generator to Comparator

Signal from Selection Circuit to Mod 3 Adders X

```
RtoSa <= "01";
RtoC <= "11";
StoMx <= "00";
```

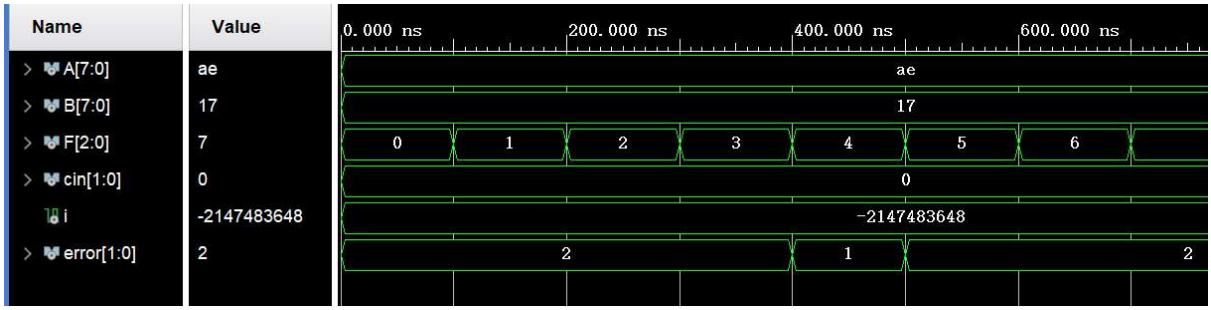

Figure 14: Inject Global Stuck error to Selection Circuit Waveform

Discussion:

The core part of Mod 3 checking circuit is the Component Residues (Chart 1). In an Addition mode, checking circuits calculate the 3 residues of A and B separately, and use Mod 3 Adders for addition. The second mode, Subtraction: it is using NOT logic operation to find B's ones' complement and working like addition. Increase and Decrease operation is the same as Addition; Increase is adding '1' to the residue and Decrease is to subtract '1' which is equal to add '2' in 2-bits adder.

In the logical operation, AND logic residue has the same generation method as ALU. The residue of OR logic is to sum up the '1' in signal A and B separately, and removes those having '1' both in the same bit in A and B. The XOR logic is just opposite to OR logic, removing those that do not have '1' both in the same bit in A and B.

When the Selection Circuit injects those Local Stuck errors (Code 13), the output Waveform shows in Logical AND case, the injected error does not affect the result (Figure 13). Because the Mod 3 Adders will only calculate the Residue of 3. If the error is 3 times the multiplier, the checking circuit cannot detect it. That is the drawback of Mod 3 Checking Circuit.

I then used Behavioral Level describe method to build my Mod 3 Adders. The advantage is that it is easy to code. Essentially, we could also use Full Adder and Half Adder to build a Mod 3 Adders circuits, which is easier to understand. From the Figure 2, each addition step needs a 2-bit Full Adder (Figure 15). By using this method, we do not need to care about which bit (Carry out) to extract for the next addition step.

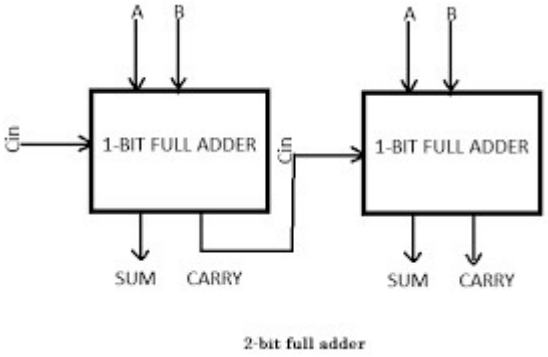

Figure 15: 2-bit full adder

There are also some improvements for the circuit. In the worst case, ALU can export 8-bits and 8-bits calculation result while the Mod 3 Checking circuit cannot handle it. Once the circuit needs Carry Out or Carry In, the Mod 3 Adders circuit will contain a different residue to ALU Residue Generator. Figure 16 shows the Addition overflow in Mod 3 Checking circuit.

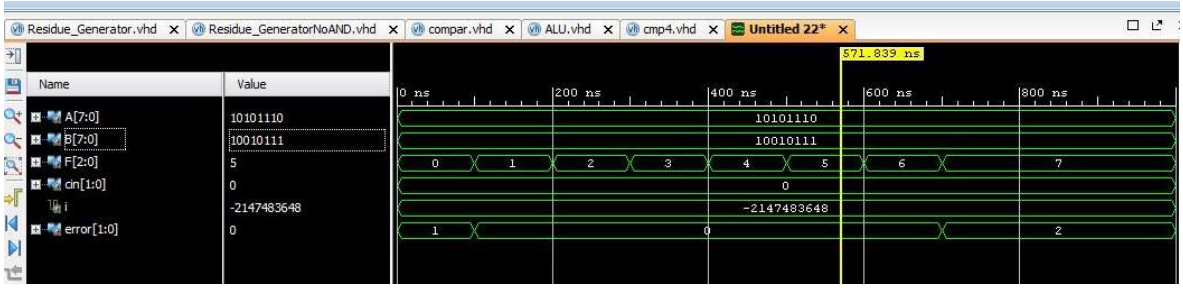

Figure 16: Worse case checking circuit Waveform

Conclusions:

In this report, a Mod 3 Checking circuit for 8-bit ALU and the Behavioral Level Fault Modelling is revisited and discusses the theory of Component Residues. With the respect to the former, some of modelling method can only apply to specific codes (cases or processes). Given this, we suggest finding out what kind of modelling method can be deployed on these circuits. Hence, this project presents an application of Behavioral Level Fault Modelling on ALU circuits.