การออกแบบวงจร

จากสิ่งที่ต้องการ คือ เราต้องการ เขียนโค้ด vending machine ที่มีการทำงาน ดังนี้ คือ

เมื่อเริ่มต้นการทำงาน ให้ แสดง ราคาสินค้า และเงินทอน เท่ากับ 0 และจากนั้น การกด ปุ่ม buy ครั้งแรกนั้น จะเป็นการเลือกสินค้า ซึ่ง สามารถปรับราคาได้ จาก slide switch ซึ่ง ใช้ 2 หลัก ซึ่งจะมีราคาทั้งหมด 4 ราคา คือ

Slide Switch	Price Value
00	48
01	38
10	18
11	9

และ จากนั้น หากเรากด ปุ่ม buy อีกครั้งจะเป็นการยกเลิกคำสั่งซื้อ ซึ่ง จะทำให้ reset ช่องแสดง ราคา เป็น 0 และ มีเสียง buzzer ดังยาวต่อเนื่องประมาณ 1 วินาที และ ถ้าหาก มีการหยอดเหรียญ ลงไปก่อนหน้านั้นแล้วจะทำการขึ้นโชว์ ที่หน้าทอนเงิน ตามจำนวนที่ได้หยอดไปก่อนหน้านั้นและกลับเข้าสู่ สภาวะเริ่มต้นดังเดิม

แต่ถ้าหากว่า ผู้ซื้อไม่ยกเลิก และได้จ่ายเงินจนครบ หรือ จ่ายเงินเกินราคาสินค้าที่เลือกแล้ว จะทำให้ ทำการโชว์ เงินทอนที่จ่ายเกิน และ มีเสียง buzzer ดังเป็นจังหวะ ประมาณ 1 วินาทีเช่นกัน และ ระบบ จะทำการเข้าสู่สภาวะเริ่มต้นดังเดิม

การกดปุ่มเพื่อหยอดเหรียญ นั้นมี 3 ปุ่ม ด้วยกัน ซึ่งประกอบ ด้วย เหรียญ 1 บาท 5 บาท และ 10 บาท หลังจาก ผู้ซื้อ กดปุ่ม จ่ายเงิน ใน ตอนที่ผู้ซื้อเลือก สินค้าเสร็จแล้ว จะทำให้ 7 segment ที่แสดง เงินที่ต้องจ่ายมีค่าลดลง ตามที่กดไป แต่ถ้าหากว่า ผู้ซื้อยังไม่กดปุ่มซื้อเพื่อเลือกสินค้า หรือว่า กดในขณะที่ buzzer ยังดังอยู่นั้น จะไม่มีผล ใดๆ เกิดขึ้น

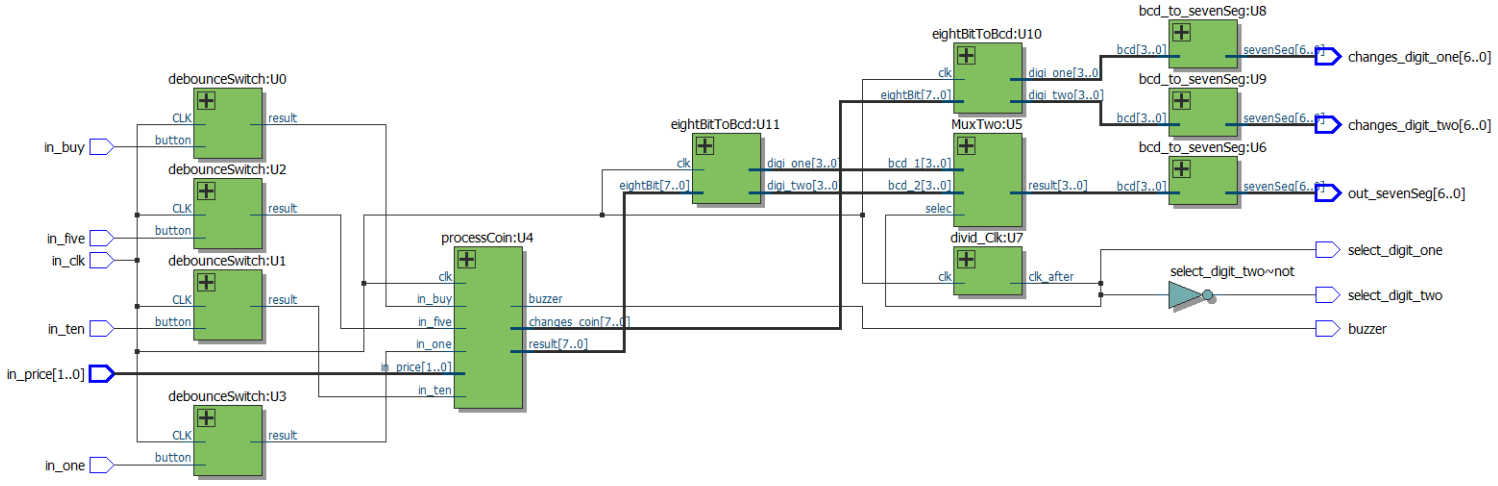
ซึ่งจากการพัฒนาโค้ด มาเรื่อยๆ นั้น โค้ดที่เสร็จสมบูรณ์ แล้ว จะมี component ดังนี้

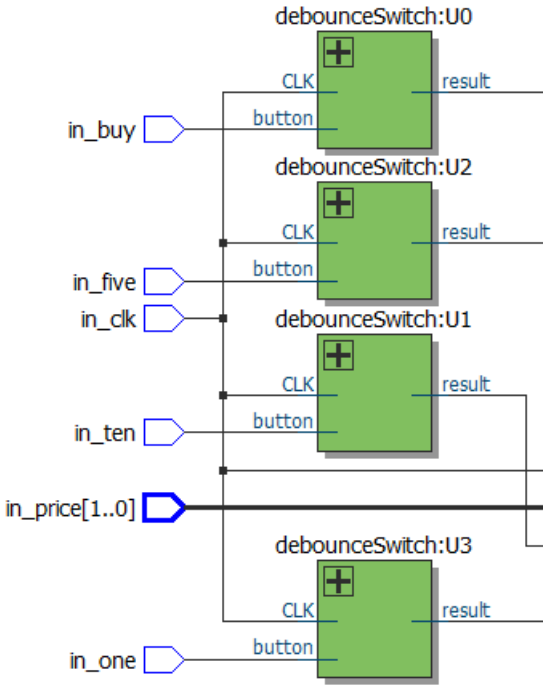
- ส่วนที่เป็นตัวกำจัด สัญญาณ รบกวนที่เกิดจาก ปุ่มกดหรือส่วนที่เรียกว่า debounceSwitch
- ส่วนที่เป็นตัวคำนวณราคาสินค้า และการทำงานส่วนใหญ่คือส่วนที่เรียกว่า processCoin
- ส่วนที่แปลงรหัส 8 bit ที่ถูกคำนวณจาก ส่วนคำนวณราคา ซึ่งเป็นรหัสที่หมายถึงราคาสินค้าหรือเงินทอน ให้กลายเป็น 4bitที่แสดงเป็นหลักหน่วยและ หลัก สิบ ซึ่งเรียกว่า eightBitToBcd
- ส่วนที่แปลงรหัส Bcd เป็น การแสดงผลบน seven segment ซึ่งเรียกว่า bcd_to_sevenSeg

- ส่วนที่ ใช้ในการเลือกสัญญาณที่ต้องการ จาก 2 สัญญาณ เหลือเพียง 1 โดยใช้ ตัว selecter เป็น clk เนื่องจากเราต้องการใช้ในการเลือกตัวแสดงผลหลักหน่วยและหลักสิบสลับกันเนื่องจาก 7 segment ที่ใช้นั้นมีการใช้ input ร่วมกัน ทั้ง 2 ตัว จึงจำเป็นต้องสลับ input แล้วใช้การเปิดปิดสลับกันผ่าน สัญญาณ clk ซึ่งทำให้สามารถมองเห็น ได้ถูกต้อง ซึ่งเรียกว่า MuxTwo
- ส่วนที่เป็นการหารสัญญาณ clk เพื่อใช้ในการสลับการเปิดปิดของ seven segment ซึ่ง เราต้องการให้มันกระพริบ น้อยลง(หรือถี่น้อยลง) ซึ่งส่วนที่ จะมี output ต่อไปถึง digit 1 และ มี output bar ต่อไปถึง digit 2 ซึ่งส่วนนี้เรียกว่า divid_Clk

ซึ่ง ทั้งหมดนี้ถูกเชื่อมต่อกันในไฟล์ vendingMach

รูปการ เชื่อมต่อกัน ของ component แบบภาพรวม





ส่วนแรกคือ การ debounceSwitch ซึ่งต่อให้ input ที่ มาจาก Switch ทั้งหมดผ่านการ debounce ทั้งหมดก่อน เพื่อจัดการสัญญาณรบกวน ที่เกิดจากระบบของ switch และมีการทำงาน แบบ Synchronous ซึ่ง ใช้สัญญาณ clk เป็นตัวกำหนดจังหวะการทำงาน

โค้ด debounceSwitch

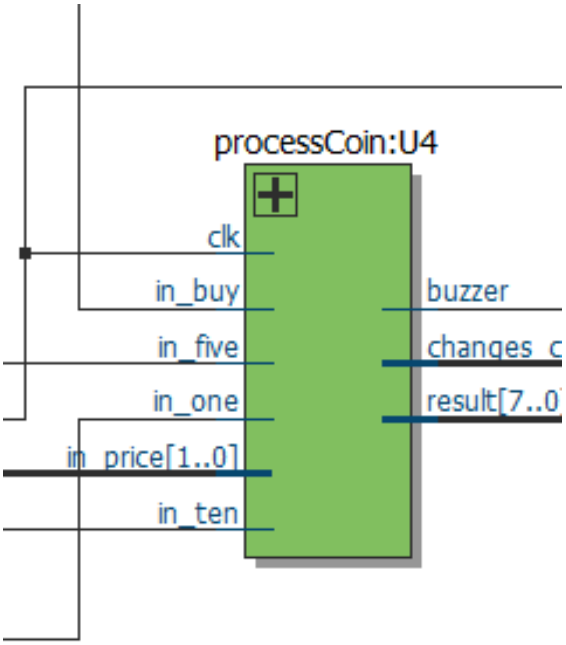
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity debounceSwitch is
6     port(CLK: in std_logic;
7          button: in std_logic;
8          result: out std_logic);
9   end debounceSwitch;
10
11  architecture behave of debounceSwitch is
12    signal inff : std_logic_vector(1 downto 0);
13    constant cnt_max : integer:=30000;
14    signal count: integer range 0 to cnt_max:=0;
15    signal keepResult: std_logic :='1';
16
17  begin
18    result <= not keepResult;
19    process(CLK)
20    begin
21      if(rising_edge(CLK)) then
22        inff <= inff(0) & button;
23          if(inff(0)/=inff(1))then
24            count <= 0;
25          elsif(count<cnt_max) then
26            count <= count +1;
27          else
28            keepResult <= inff(1);
29          end if;
30       end if;
31     end process;
32  end behave;
```

การทำงานคือ เมื่อตรวจจับของขาขึ้นของ clk ได้ ให้ทำการอัพเดทค่า inff โดยให้ เท่ากับ inff(0) , button(สถานะปุ่มปัจจุบัน) เช่น ถ้าหากว่า inff เดิม คือ 10 และในขณะที่ปุ่มกดกดอยู่ หรือ button เท่ากับ 1 ในช่วงที่ clk ขาขึ้นมา จะทำการอัพเดท inff = 11 และจากนั้นทำการเช็คว่าถ้าหากว่า inff(1) ไม่เท่ากับ inff(0) จะทำการ reset การนับ เป็น 0 แต่ถ้าเท่าจะนับเพิ่มเรื่อยๆ ซึ่งการนับนี้ นั้น ถ้าหากว่า นับไปจนถึงค่าที่เรากำหนดไว้ มันจะทำการอัพเดทค่า output ของ component เป็น inff(1) ซึ่ง การที่กำหนดให้ result เป็น not ของ ผลลัพ ก็เพราะว่า ต้องการเปลี่ยนจากการที่กดปุ่มแล้วมีค่าเป็น 0 เป็นกดแล้วมีค่า เป็น 1 แทน เพื่อ ให้สะดวกในการเขียนโค้ดที่เหลือต่อ

ส่วนที่ สอง คือ ส่วน processCoin

ที่เป็นส่วนที่ใช้ในการคำนวณ ราคาสินค้าที่เหลือ

เงินทอน หรือ แทบจะเป็นทุกอย่าง ของ วงจรนี้



ซึ่ง มีโค้ดดังนี้

```
5  entity processCoin is
6      port(in_ten :in std_logic;
7           in_five:in std_logic;
8           in_one :in std_logic;
9           in_buy :in std_logic;
10          in_price:in std_logic_vector(1 downto 0);
11               clk:in std_logic;
12            result:out std_logic_vector(7 downto 0);
13            buzzer:out std_logic;
14          changes_coin:out std_logic_vector(7 downto 0));
15 end processCoin;
16
17 architecture Behavioral of processCoin is
18     type state_type is (s0, s1, s2, s3, s4, s5);
19     signal state : state_type :=s0;
20     signal count_delay_changes:integer :=0;
21     signal count_delay_buzzer:integer :=0;
22     signal max_delay:integer := 50000000;
23     signal keep_re: std_logic_vector(7 downto 0);
24     signal changes: std_logic_vector(7 downto 0);
25     signal store_value: integer := 0;
26     signal price_value: integer := 0;
27 begin
28    result <= keep_re;
29    changes_coin <= changes;
30    process(clk)
31       begin
32          if(rising_edge(clk))then
33             case state is
```

มี input เป็น การหยอดเหรียญ ทั้ง 3 เหรียญ และปุ่มซื้อและ ยกเลิก มี input 2 bit ที่เป็นการกำหนดราคา มี clk และ output เป็น ราคาที่ผู้ใช้ต้องจ่าย และเงินทอน 8 bit รวมถึง buzzer ด้วย

ซึ่ง max_delay ใช้ในการ delay buzzer ให้วนตามจำนวน clk ที่กำหนดไว้ (component ตัวนี้จะได้รับ clk มาที่ 50Mhz) count_delay_changes นั้นเป็นตัวไว้นับให้เลขเงินทอนแสดงจนถึงการนับที่น้อยกว่า max_delay และ count_delay_buzzer จะเป็นตัวไว้นับเสียงที่เป็นจังหวะของ buzzer ในส่วนที่เป็นจังหวะ

Store_value ไว้เก็บจำนวนราคาที่ผู้ซื้อต้องจ่ายเพิ่ม

Price_value ไว้เก็บราคาสินค้าที่แท้จริง

Output ที่เป็น result นั้นจะให้มาจาก keep_re ตลอดเวลา และ ส่วนที่เป็นchanges _coin นั้นจะมาจาก changes ตลอด

มีทั้งหมด 6 stage โดยเริ่มจาก s0 และมีการทำงานตาม clk เป็น Synchronous

การทำงาน stage 0

```
when s0 =>
   buzzer <= '0';
   keep_re <= "00000000";
   changes <= "00000000";
   count_delay_changes <= 0;
   case in_price is
         when "00" => store_value <= 45;
         when "01" => store_value <= 38;
         when "10" => store_value <= 18;
         when "11" => store_value <= 9;
   end case;
   price_value <= store_value;
   if(in_buy = '1')then
      state <= s1;
   end if;
```

มีการทำงานคือ จะทำการ เซ็ทค่าเริ่มต้น ของวงจร ให้ทุกอย่างเป็น 0 และทำการอัพเดทราคาสินค้า ให้กับ store_value และทำการอัพเดท ราคาที่แท้จริงให้กับ price_value ซึ่งใน stage นี้ จะมีเงื่อนไขในการไป stage s1 คือ เมื่อปุ่มกดอยู่ในสภาวะ hight เท่านั้น

การทำงาน stage 1

```
when s1 =>
   if(in_buy = '0')then
      state <= s2;
   end if;
```

เป็น stage ที่ มีไว้เพื่อ ไม่ให้ การกดปุ่มค้างมีผลการทำงาน ซึ่งหาก มีการกดปุ่มค้างยังไม่ปล่อยก็จะวนใน stage นี้เรื่อยๆ จนกระทั่งปล่อย ก็จะเข้าสู่ stage s2

การทำงาน stage 2

```
when s2 =>
   if(in_buy = '1')then
      if((price_value - store_value) > 0)then
         store_value <= store_value - price_value;
      end if;
      state <= s3;
   end if;
   if(in_ten = '1')then
      store_value <= store_value -10;
      state <= s4;
   elsif(in_five = '1')then
      store_value <= store_value -5;
      state <= s4;
   elsif(in_one ='1')then
      store_value <= store_value -1;
      state <= s4;
   end if;
   if(store_value < 1)then
      changes <= std_logic_vector(to_signed(store_value*(-1), 8));
      keep_re <= "00000000";
      state <= s5;
   else
      keep_re <= std_logic_vector(to_signed(store_value, 8));
      changes <= "00000000";
   end if;
```

การทำงานใน stage นี้ นั้น จะเช็ค อยู่ 3 อย่าง คือมีการหยอดเหรียญมั้ยถ้าหากมีจะทำการอัพเดท เงินที่ต้องจ่ายให้ลดลงตามที่จ่าย และ อัพเดทค่าที่แสดงผลออกไปและทำการเข้าสู่ stage 4 ก่อนจากนั้นถ้าหากว่า ค่าที่ต้องจ่ายยังไม่น้อยกว่า 1 (หรือยังต้องจ่ายมากกว่า 0) ก็จะอัพเดทค่าตามปกติ แต่ถ้าหากว่า น้อยกว่าแล้วหรือหมายความว่า จ่ายเงินครบหรือเกินแล้วจะทำการย้ายไป stage 5 , และถ้าหากว่ามีการกดปุ่ม buy หรือปุ่ม buy มีสถานะเป็น hight ใน stage นี้ จะเช็คก่อนว่าราคาที่ต้องจ่ายนั้น มีค่าน้อยกว่าราคาสินค้าจริงมั้ย(ไว้เช็คว่าผู้ซื้อหยอดเหรียญมาหรือไม่)แล้วคำนวณเงินทอน จากนั้นเข้าสู่ stage 3

การทำงาน stage 3

```
when s3 =>
   keep_re <= "00000000";
   if(count_delay_changes < max_delay)then
      buzzer <= '1';
      if(store_value < 0)then
         changes <= std_logic_vector(to_signed(store_value*(-1), 8));
      else
         count_delay_changes <= max_delay;
      end if;
      count_delay_changes <= count_delay_changes + 1;
   elsif(in_buy /= '0')then
      count_delay_changes <= 0;
   else
      buzzer <= '0';
      state <= s0;
   end if;
```

เป็นส่วนที่ทำให้ buzzer ดังเป็นเสียงยาว ประมาณ 1 วินาที โดย มีการทำงานคือ ให้นับ ไปเรื่อยๆตาม clk ถ้าหากว่า ค่าที่นับยังน้อยกว่า ค่าที่กำหนดไว้ก็ให้ buzzer ดังไปเรื่อยๆ โดยมีการอัพเดทค่าตลอด ซึ่ง จะมีการเช็คอีกด้วยว่า ถ้าหากว่ามันเข้า stage นี้มาเนื่องจากสาเหตุอะไรถ้าหากว่ามาจากการที่ จ่ายเงินครบ จะทำการตั้งค่าเงินทอนแต่ถ้าไม่ใช่ ก็จะให้หลุดจาก loop ทันที และหลักจากที่เช็คว่านับครบแล้วก็เช็คอีกว่าถ้าหากยังไม่มีการปล่อยปุ่มที่กด ก็จะให้นับใหม่เรื่อยๆจาก 0 จนกระทั่งปล่อยและนับครบแล้วก็จะทำให้ buzzer ดับแล้วเข้าสู่ stage 0

การทำงาน stage 4

```
when s4 =>
   if(in_ten ='0' and in_five ='0' and in_one ='0')then
      state <= s2;
   end if;
```

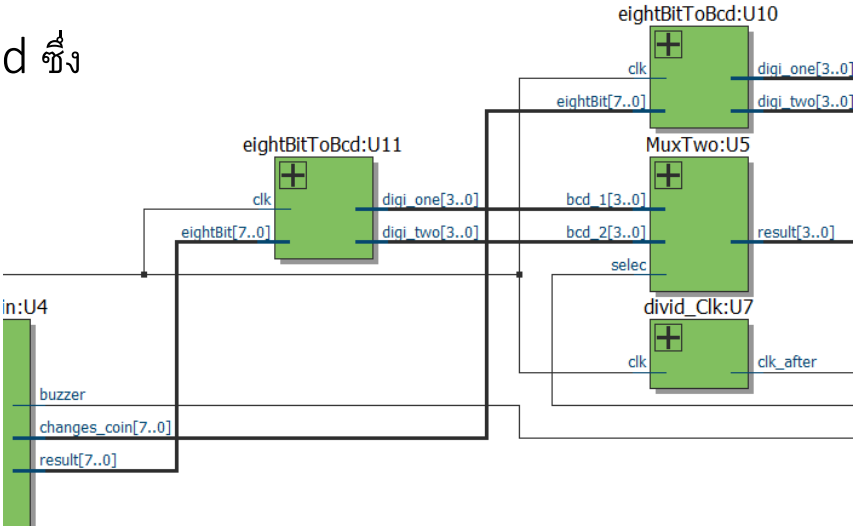
มีไว้เพื่อ ไม่ให้การกดปุ่มจ่ายเงินค้าง มีผลต่อเนื่อง คือจะหลุดจาก stage 4 ได้ก็ต่อเมื่อไม่มีการกดปุ่มจ่ายเงินใดๆเลย แล้วจะกลับเข้าสู่ stage 2

การทำงาน stage 5

```
when s5 =>
   if(count_delay_changes < max_delay)then
      buzzer <= '1';
      if(count_delay_buzzer > max_delay/10)then
         buzzer <= '0';
         count_delay_buzzer <= count_delay_buzzer +1;
         if(count_delay_buzzer > max_delay/5)then
            count_delay_buzzer <= 0;
         end if;
      else
         count_delay_buzzer <= count_delay_buzzer +1;
      end if;
      count_delay_changes <= count_delay_changes + 1;
    else
      count_delay_buzzer <= 0;
      state <= s0;
    end if;
end case;
```

การทำงานจะคล้ายกับ stage 3 คือ มีการนับ buzzer เหมือนกันแต่มีส่วนที่เพิ่มขึ้นมา คือ ส่วนที่เป็นการทำให้ buzzer ดับ ในขณะที่ มีค่ามากกว่า max_delay/10 และดังอีกครั้งเมื่อมากกว่า max_delay/5 เพราะ เมื่อมากกว่า max_delay/5 แล้ว มันจะทำการ reset การนับในส่วนที่ทำให้เกิดจังหวะ ของ buzzer ซึ่งคนละส่วนที่นับเวลา การดังของ buzzerซึ่งถ้าหาก นับครบแล้ว จะทำการเข้าสู่ stage 0 ดังเดิม

ส่วนที่ สาม คือ eightBitToBcd ซึ่งมีการใช้งาน 2 ครั้ง คือใช้แปลงค่าเงินที่ต้องจ่ายเพิ่มกับเงินทอน จากที่เป็น 8 bit ให้กลายเป็น 4bit ที่เป็นหลักหน่วยและหลักสิบ ของแต่ละตัว



ซึ่งมีโค้ดดังนี้

```
5  entity eightBitToBcd is
6      port(     clk:in std_logic;
7           eightBit:in std_logic_vector(7 downto 0);
8           digi_one:out std_logic_vector(3 downto 0);
9           digi_two:out std_logic_vector(3 downto 0));
10 end eightBitToBcd;
11
12 architecture Behavioral of eightBitToBcd is
13     signal value_input: integer :=to_integer(signed(eightBit));
14     signal value_digit_one: integer :=value_input/10;
15     signal value_digit_two: integer :=value_input mod 10;
16 begin
17     process(clk)
18     begin
19         if(rising_edge(clk))then
20             digi_one <= std_logic_vector(to_signed(value_digit_one, 4));
21             digi_two <= std_logic_vector(to_signed(value_digit_two, 4));
22         end if;
23    end process;
24 end Behavioral;
```

หลักการง่ายๆ คือทำการ หาร และ มอดหาเศษ ของค่าที่ได้รับ เช่น การหาหลักสิบ ของค่าที่ได้มา เช่น ค่าที่ได้รับมาเป็น 84 การที่ 84 หาร 10 นั้นจะได้ 8 เพราะการหาร จะปัดเศษลงเสมอ ซึ่งจะได้หลัก สิบมา ใช้นั้นเอง ส่วนหลักหน่วยใช้การ มอดหาเศษ ซึ่ง 84 มอด 10 จะได้ 4 ซึ่งเป็นหลักหน่วยนั้นเอง และนำไปใช้ฟังก์ชั่น แปลงค่าจาก integer ให้กลายเป็น bcd ส่งเป็น output

ส่วนที่ สี่ คือ เป็นส่วนที่ ใช้ในการแปลงค่า Bcd 4 bit ที่ได้รับมาให้แสดงผล เป็นตัวเลข บน 7 segment หรือก็คือ bcd_to_sevenSeg

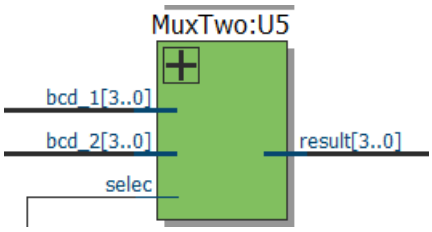
โดยมีโค้ดดังนี้

```
5  entity bcd_to_sevenSeg is
6    port(     bcd: in std_logic_vector (3 downto 0);
7         sevenSeg: out std_logic_vector (6 downto 0));
8  end bcd_to_sevenSeg;
9
10 architecture Structural of bcd_to_sevenSeg is
11 begin
12   process (bcd)
13     begin
14         case bcd is
15             when "0000" => sevenSeg <= "1000000";
16             when "0001" => sevenSeg <= "1111001";
17             when "0010" => sevenSeg <= "0100100";
18             when "0011" => sevenSeg <= "0110000";
19             when "0100" => sevenSeg <= "0011001";
20             when "0101" => sevenSeg <= "0010010";
21             when "0110" => sevenSeg <= "0000010";
22             when "0111" => sevenSeg <= "1111000";
23             when "1000" => sevenSeg <= "0000000";
24             when "1001" => sevenSeg <= "0011000";
25             when others => sevenSeg <= "0110000";
26          end case;
27     end process;
28 end Structural;
29
```

หลักการคือนำ input 4 bit ที่ได้รับมา มาเทียบ เป็น กรณี แล้วให้ output ตาม กรณี ที่ตรงตาม input ที่ได้รับมานั้นเอง

(ซึ่งoutput จะเรียงจาก G F... B A และ 0 หมายถึง ไฟติด 1 ไฟดับ เนื่องจากคุณสมบัติของ 7 segement ที่ใช้)

ส่วนที่ 5 คือ MuxTwo ใช้ในการเลือก output จาก 2 input



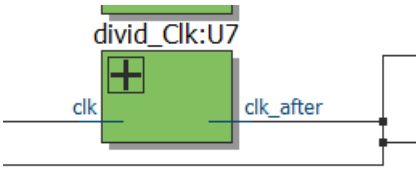
มีโค้ดดังนี้

```
5  entity MuxTwo is
6    port(bcd_1, bcd_2: in std_logic_vector(3 downto 0);
7                 selec: in std_logic;
8               result: out std_logic_vector(3 downto 0));
9  end MuxTwo;
10
11 architecture behave of MuxTwo is
12 begin
13   result <= bcd_1 when selec = '0' else bcd_2;
14 end behave;
15
```

มีการใช้ตัว selec เป็นตัวกำหนดว่าจะใช้ input 1 หรือ input 2 เป็น output ซึ่งในการใช้งานกับงานชิ้นนี้นั้น ได้ใช้ clk ที่ได้จากการหารสัญญาณ แล้วมาใช้งานในการเลือก

ส่วนที่ หก คือ devid_clk

เป็นส่วนที่ใช้ในการหารสัญญาณ ให้มีความถี่น้อยลง



โดยมีโค้ดดังนี้

```
 5  entity divid_Clk is
 6    port(      clk: in std_logic;
 7         clk_after: out std_logic);
 8  end divid_Clk;
 9
10  architecture Structural of divid_Clk is
11      signal max_count: integer := 499999;
12      signal counter: integer range 0 to max_count := 0;
13      signal result: std_logic :=clk;
14  begin
15    clk_after <= result;
16    process (clk)
17      begin
18        if(rising_edge(clk))then
19            if(counter < max_count)then
20                counter <= counter + 1;
21            else
22                if(result = '0')then
23                    result <= '1';
24                else
25                    result <='0';
26                end if;
27                counter <= 0;
28           end if;
29        end if;
30      end process;
31  end Structural;
32
```

หลักการการทำงานคือให้ทำงานทุกๆขอบขาขึ้นของ clk แล้วนับจำนวน clk ที่ผ่านไป จนถึง ค่าๆนึงจากนั้นให้ ทำการสลับ output จาก 1 เป็น 0 จาก 0 เป็น 1 ซึ่งกำหนดไว้ ที่ 499999 ซึ่งนับจาก 0 จะมีค่า เป็น 500,000 ต่อการเปลี่ยน 1 ครั้ง ซึ่งหมายความว่า การที่มันจะกลับเป็นค่าเดิมหรือครบ 1 คาบต้องใช้ 1 ล้าน clk

1M ต่อ 1 จาก input ที่ มี clk เป็น 50Mhz จะเหลือ 50 Hz นั้นเอง