CHAPTER 8

BINARY TO RESIDUE CONVERTER: VHDL CODING, SIMULATION AND VERIFICATION BY CPLD

8.1 Introduction

In the earlier chapters, various aspects of conversions between binary and residue numbers and the mathematical operations involving residue arithmetic have been thoroughly discussed. The circuits for such mathematical operations involving residue numbers are mainly digital ones. For all the residue based mathematical operations, verifications have been done using logic circuits implemented on bread board. Based on these principles, flowcharts have been developed so that all these operations can be performed by 8051 microcontroller. Although the present thesis involves simulation and microcontroller implementation, I feel it necessary to include, at least to some extent, the latest trend in digital circuit simulation using VHDL code for final verification by employing Complex Programmable Logic Device (CPLD). Among all the residue based circuits included in this thesis, I consider only the binary to residue converter of chapter 4 (Fig. 4.17) for functional verification using CPLD because of my time constraint to submit the thesis. The relevant software for writing the necessary VHDL code is Quartus - II, version 4.0 which is run in Windows XP environment. After writing the VHDL code, the simulation of the binary to residue converter is done following compilation. The simulation result
reflects the truth-table verification of the binary to residue converter. To enable the functional verification by the CPLD, the necessary Hex codes are generated by Quartus - II software. By connecting an interface card supported with SPDT switches and LEDs, the manual verification has been performed satisfactorily.

8.2 VHDL

The connection pattern of a digital circuit on a bread-board is described for its behavioral modelling by a language, often called Hardware Description Language. The two popular Hardware Description Languages are Verilog HDL [174] and VHDL [175]. The digital circuit synthesis and also the corresponding simulation are the two aspiring features of VHDL. The full form of VHDL stands for Very High Speed Integrated Circuit (V) Hardware (H) Description (D) Language (L) [176]. Some of the notable features [177] of VHDL are:

- Vendor independents
- Portable
- Reusable
- Concurrency

8.3 VHDL Codes

It has already been pointed out that only the combined circuit for binary to residue converter of moduli 15 and 17 has been chosen for VHDL coding and subsequent verification by CPLD for time constraint. The logic circuit implementation involves three numbers of 4-bit binary adders (7483 ICs), three numbers of 5-bit binary adders (constructed with two numbers of 7483 ICs), two numbers of multiplexers (each constructed with two numbers of 74157 ICs) and one 2-to-1 multiplexer constructed with logic gates. The VHDL codes for 4-bit binary adder 1 of Fig. 4.17 requires the VHDL codes for half adder and full adder. The corresponding codes for half adder, full adder and 4-bit binary Adder 1
are given below:

**VHDL Codes for Half Adder**

Library IEEE;
Use IEEE.std_logic_1164.all;
entity half_adder is
port(a,b:in bit;
       sum,carry:out bit);
end half_adder;
architecture df of half_adder is
begin
carry <= a and b;
sum <= a xor b;
end df;

**VHDL Codes for Full Adder**

Library IEEE;
Use IEEE.std_logic_1164.all;
entity full_adder is
port(a,b,cin: in bit;
       sum,cout:out bit);
end full_adder;
architecture struct of full_adder is
signal m,n,p:bit;
component half_adder is
port(a,b:in bit;
       sum,carry:out bit);
end component;
begin
ha1:half_adder port map(a,cin,m,n);
ha2:half_adder port map(b,m,sum,p);
cout <= n or p;
end struct;

VHDL Codes for 4-bit Binary Adder 1

Library IEEE;
Use IEEE.std_logic_1164.all;
entity adder_ripple is
port(a:in bit_vector(3 downto 0);
    b:in bit_vector(3 downto 0);
    cin:in bit;
    s:out bit_vector(3 downto 0);
    cout:out bit);
end adder_ripple;
architecture struct of adder_ripple is
signal s1,s2,s3:bit;
component full_adder is
port(a,b,cin: in bit;
    sum,cout:out bit);
end component;
begin
fa1:full_adder port map(a(0),b(0),cin,s(0),s1);
fa2:full_adder port map(a(1),b(1),s1,s(1),s2);
fa3:full_adder port map(a(2),b(2),s2,s(2),s3);
fa4:full_adder port map(a(3),b(3),s3,s(3),cout);
end struct;

The 4-bit binary adder 2 supported with exclusive OR gates can act as both adder as well as subtractor depending upon the value of the control input X. Similar is the case with 4-bit binary adder 3 of Fig. 4.17. The VHDL codes for both these adder/subtractor blocks are shown below as Adder Subtractor 1 and Adder Subtractor 2.

**VHDL Codes for Adder Subtractor 1**

Library IEEE;
Use IEEE.std_logic_1164.all;
entity adder_subtractor is
port(a:in bit_vector(3 downto 0);
    b:in bit_vector(3 downto 0);
    cin:in bit;
    s:out bit_vector(3 downto 0);
    cout:out bit);
end adder_subtractor;
architecture struct of adder_subtractor is
signal p:bit_vector(3 downto 0);
component adder_ripple is
port(a:in bit_vector(3 downto 0);
    b:in bit_vector(3 downto 0);
    cin:in bit;
    s:out bit_vector(3 downto 0);
    cout:out bit);
end component;
begin
  p(0)<= b(0) xor cin;
p(1)<= b(1) xor cin;
p(2)<= b(2) xor cin;
p(3)<= b(3) xor cin;
f1:adder_ripple port map(a,p,cin,s,cout);
end struct;

VHDL Codes for Adder Subtractor 2

Library IEEE;
Use IEEE.std_logic_1164.all;
entity adder_subtractor_1 is
port(a:in bit_vector(3 downto 0);
   b:in bit;
   cin:in bit;
   s:out bit_vector(3 downto 0);
   cout:out bit);
end adder_subtractor_1;
architecture struct of adder_subtractor_1 is
signal p:bit_vector(3 downto 0);
component adder_ripple is
port(a:in bit_vector(3 downto 0);
   b:in bit_vector(3 downto 0);
   cin:in bit;
   s:out bit_vector(3 downto 0);
   cout:out bit);
end component;
begin
p(0)<= b xor cin;
p(1)<= cin;
p(2)<= cin;
p(3)<= cin;
f1:adder_ripple port map(a,p,cin,s,cout);
end struct;

To enable the VHDL coding for 5-bit binary adder 1 and adder 2, it is necessary to write the VHDL codes for subtraction which is to be called by the VHDL codes associated with 5-bit binary adder 1 and adder 2. The listing of all these codes are mentioned below:

**VHDL Codes for Subtractor**

Library IEEE;
Use IEEE.std_logic_1164.all;
entity sub is
port(a,b,c:in bit;
    bor,dif:out bit);
end sub;
architecture df of sub is
signal m,n,o1,p:bit;
begin
m<=not a;
n<=m and b;
o1<=m and c;
p<=b and c;
bos<=m or o1 or p;
dif<a xor b xor c;
end df;
VHDL Codes for 5-bit Binary Adder 1

Library IEEE;
Use IEEE.std_logic_1164.all;
entity sub_5 is
  port(a1: in bit_vector(3 downto 0);
    t,c1: in bit;
    bor1: out bit;
    dif1: out bit_vector(4 downto 0));
end sub_5;
architecture struct of sub_5 is
  signal b_w: bit_vector(3 downto 0);
  signal p: bit;
  component sub is
    port(a,b,c: in bit;
      bor,dif: out bit);
  end component;
begn
  p<= '0';
  s1: sub port map(a1(0), t, c1, b_w(0), dif1(0));
  s2: sub port map(a1(1), t, b_w(0), b_w(1), dif1(1));
  s3: sub port map(a1(2), t, b_w(1), b_w(2), dif1(2));
  s4: sub port map(a1(3), t, b_w(2), b_w(3), dif1(3));
  s5: sub port map(t, p, b_w(3), bor1, dif1(4));
end struct;
VHDL Codes for 5-bit Binary Adder 2

Library IEEE;

Use IEEE.std_logic_1164.all;

entity s_5 is
port(a:in bit_vector(4 downto 0);
   t:in bit;
   s:out bit_vector(4 downto 0));
end s_5;

architecture behv of s_5 is
begin
    process(t,a)
    begin
        if t='1' then
            case a is
            when "00000" => s<= "11111";
            when "00001" => s<= "00000";
            when "00010" => s<= "00001";
            when "00011" => s<= "00010";
            when "00100" => s<= "00011";
            when "00101" => s<= "00100";
            when "00110" => s<= "00101";
            when "00111" => s<= "00110";
            when "01000" => s<= "00111";
            when "01001" => s<= "01000";
            when "01010" => s<= "01001";
            when "01011" => s<= "01010";
            when others => s<= "00000";
            end case;
        end if;
    end process;
end behv;
when "01100" => s <= "01011";
when "01101" => s <= "01100";
when "01110" => s <= "01101";
when "01111" => s <= "01110";
when "10000" => s <= "01111";
when "10001" => s <= "10000";
when "10010" => s <= "10001";
when "10011" => s <= "10010";
when "10100" => s <= "10011";
when "10101" => s <= "10100";
when "10110" => s <= "10101";
when "10111" => s <= "10110";
when "11000" => s <= "10111";
when "11001" => s <= "11000";
when "11010" => s <= "11001";
when "11011" => s <= "11010";
when "11100" => s <= "11011";
when "11101" => s <= "11100";
when "11110" => s <= "11101";
when "11111" => s <= "11110";
end case;
else s<=a;
end if;
end process;
end behv;
For the VHDL coding for the 5-bit binary adder 3 of Fig. 4.17, the VHDL code for 5-bit adder is extremely necessary to generate the corresponding VHDL code for 5-bit adder/subtractor. The code listings are furnished below:

**VHDL Codes for 5-bit Binary Adder**

Library IEEE;
Use IEEE.std_logic_1164.all;

entity adder_ripple1 is
port(a:in bit_vector(4 downto 0);
    b:in bit_vector(4 downto 0);
    cin:in bit;
    s:out bit_vector(4 downto 0);
    cout:out bit);
end adder_ripple1;

architecture struct of adder_ripple1 is
signal s1,s2,s3,s4:bit;

component full_adder is
port(a,b,cin: in bit;
    sum,cout:out bit);
end component;
beg
fa1:full_adder port map(a(0),b(0),cin,s(0),s1);
fa2:full_adder port map(a(1),b(1),s1,s(1),s2);
fa3:full_adder port map(a(2),b(2),s2,s(2),s3);
fa4:full_adder port map(a(3),b(3),s3,s(3),s4);
fa5:full_adder port map(a(4),b(4),s4,s(4),cout);
end struct;
VHDL Codes for 5-bit Adder/Subtractor

Library IEEE;
Use IEEE.std_logic_1164.all;
entity adder_subtractor_2 is
port(a:in bit_vector(4 downto 0);
     cin:in bit;
     s:out bit_vector(4 downto 0);
     cout:out bit);
end adder_subtractor_2;
architecture struct of adder_subtractor_2 is
signal p:bit_vector(4 downto 0);
component adder_ripple1 is
port(a:in bit_vector(4 downto 0);
     b:in bit_vector(4 downto 0);
     cin:in bit;
     s:out bit_vector(4 downto 0);
     cout:out bit);
end component;
between
p(0)<=not cin;
p(1)<= cin;
p(2)<= cin;
p(3)<= cin;
p(4)<= cin;
f1:adder_ripple1 port map(a,p,cin,s,cout);
end struct;
The VHDL codes for three multiplexers (MUX 1, MUX 2 and MUX 3) of Fig. 4.17 are also given below:

**VHDL Code for MUX 1**

Library IEEE;

Use IEEE.std_logic_1164.all;

entity mux_2 is

port(d0,d1: in bit_vector(4 downto 0);
    s0: in bit;
    y: out bit_vector(4 downto 0));

end mux_2;

architecture m21 of mux_2 is

begin

p0: process(s0,d0,d1)

begin

if (s0 = '0') then

y<=d0;

else

y<=d1;

end if;

end process p0;

end m21;

**VHDL Code for MUX 2**

entity mux_21 is

    port(d: in bit_vector(1 downto 0);
        s0: in bit;
        y: out bit);

end entity mux_21;
end mux_21;

architecture m21 of mux_21 is

signal p: bit_vector(1 downto 0);

signal m: bit;

begin

m<=not (s0);
p(0)<=m and d(0);
p(1)<=s0 and d(1);
y<=p(0) or p(1);

end m21;

VHDL Code for MUX 3

entity mux_22 is

port(d0: in bit_vector(4 downto 0);
d1: in bit_vector(2 downto 0);
s0: in bit;
y: out bit_vector(4 downto 0));

end mux_22;

architecture m21 of mux_22 is

signal n:bit_vector(4 downto 0);

begin

n(4)<='0'; n(3)<=d1(2); n(2)<=d1(1); n(1)<=d1(0); n(0)<='0';
p0:process(s0,d0,d1)

begin

if(s0='0')then

y<=d0;

else

end

end process p0;

end m21;
y<=n;
end if;
end process p0;
end m21;

The VHDL code listing for entire binary to residue converter for moduli 15 and 17 is given below:

**VHDL Code for Binary to Residue Converter of Moduli 15 and 17**

entity main is
  port(y2, y1, y0:in bit_vector(3 downto 0);
  x:in bit;
  o:out bit_vector(4 downto 0));
end main;
architecture struct of main is
signal a1,a2,a3:bit_vector(3 downto 0);
signal z,q,c1,c2,c3,m,p,discard,discard1,discard2:bit;
signal a4,a5,a6,res:bit_vector(4 downto 0);
signal n:bit_vector(1 downto 0);
signal r:bit_vector(2 downto 0);
component adder_ripple is
port(a:in bit_vector(3 downto 0);
  b:in bit_vector(3 downto 0);
  cin:in bit;
  s:out bit_vector(3 downto 0);
  cout:out bit);
end component;
component adder_subtractor is
port(a: in bit_vector(3 downto 0);
    b: in bit_vector(3 downto 0);
    cin: in bit;
    s: out bit_vector(3 downto 0);
    cout: out bit);
end component;
component adder_subtractor_1 is
port(a: in bit_vector(3 downto 0);
    b: in bit;
    cin: in bit;
    s: out bit_vector(3 downto 0);
    cout: out bit);
end component;
component mux_21 is
port(d: in bit_vector(1 downto 0);
    s0: in bit;
    y: out bit);
end component;
component sub_5 is
port(a1: in bit_vector(3 downto 0);
    t,cl: in bit;
    bor1: out bit;
    dif1: out bit_vector(4 downto 0));
end component;
component s_5 is
port(a: in bit_vector(4 downto 0);
  t: in bit;
  s: out bit_vector(4 downto 0));
end component;

component adder_subtractor_2 is
port(a: in bit_vector(4 downto 0);
  cin: in bit;
  s: out bit_vector(4 downto 0);
  cout: out bit);
end component;

component mux_22 is
port(d0: in bit_vector(4 downto 0);
  d1: in bit_vector(2 downto 0);
  s0: in bit;
  y: out bit_vector(4 downto 0));
end component;

component mux_2 is
port(d0,d1: in bit_vector(4 downto 0);
  s0: in bit;
  y: out bit_vector(4 downto 0));
end component;
beg
z<=not x;
n(1)<=c2 nand c3;
n(0)<=c2 or c3;
q<=a5(4) and a5(0);
\( r(2) \leq x \);  
\( r(1) \leq x \);  
\( r(0) \leq x \);  
\( t1 \): adder_ripple port map(y0,y2,x,a1,c1);  
\( t2 \): adder_subtractor port map(a1,y1,z,a2,c2);  
\( t3 \): adder_subtractor_1 port map(a2,c1,z,a3,c3);  
\( t4 \): mux_21 port map(n,x,m);  
\( t5 \): sub_5 port map(a3,m,discard1,discard2,a4);  
\( t6 \): s_5 port map(a4,x,res);  
\( t7 \): adder_subtractor_2 port map(res,x,a5,discard);  
\( t8 \): mux_22 port map(a5,r,q,a6);  
\( t9 \): mux_2 port map(res,a6,y2(3),o);  
end struct;  
The on screen listing of VHDL codes for the entire binary to residue converter with moduli 15 and 17 based on Quartus-II, version 4.0 software is shown in Fig. 8.1.  

### 8.4 Simulation Results

After writing of the proper VHDL codes for various blocks of binary to residue converter of moduli 15 and 17, the simulation of all these blocks has been performed using Quartus-II, Version 4.0 software in windows XP environment. The result of simulation of half adder is shown in Fig. 8.2 where the X axis represents time scale ranging from 0 pS to 1 nS. Similarly the timing diagram, which is the outcome of full adder, is shown in Fig. 8.3. Based on the simulation of the individual blocks, the final simulation result of the binary to residue converter of modulo 15 is shown in Fig. 8.4. It is to be noted that though the simulation diagram is basically a timing diagram, however the relationship between inputs and outputs can not be established with respect to time slots, because the input is 12-bit binary number while the output is a 5-bit binary number. So, only the bit patterns
Fig. 8.1: On screen listing of VHDL code for the entire binary to residue converter of moduli 15 and 17.

Fig. 8.2: Timing diagram of half adder.
<table>
<thead>
<tr>
<th>Name</th>
<th>Value at</th>
<th>0 ps</th>
<th>60,0 ns</th>
<th>160,0 ns</th>
<th>240,0 ns</th>
<th>320,0 ns</th>
<th>400,0 ns</th>
<th>480,0 ns</th>
<th>560,0 ns</th>
<th>640,0 ns</th>
<th>720,0 ns</th>
<th>800,0 ns</th>
<th>880,0 ns</th>
<th>960,0 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>cary</td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 8.3: Timing diagram of full adder.

Fig. 8.4: Simulation result of binary to residue converter of modulo 15.

in the input/output has been shown in Fig. 8.4 unlike the half/full adder. The result shown in Fig. 8.4 completely matches with the corresponding entries of the truth-table shown in Table 4.5.
8.5 Verification by CPLD

For the purpose of verification, we have considered the CPLD Max 3128 ATC 144 [178]. To enable the testing of all the entries of the truth-table of 12-bit positive and negative binary to residue converter of moduli 15 and 17, an LED based interfacing card has been coupled with the CPLD through I/O port 1. This I/O port 1 has been activated to function as output port from the CPLD by enabling SPDT switches. For the purpose of verification, the entire VHDL code for the binary to residue converter of moduli 15 and 17 has been simulated, and then converted to hexadecimal value using Quartus-II, Version 4.0 software. The hexadecimal codes are now being transferred into the referred CPLD through parallel port of the computer. All the entries of the truth-table of binary to residue converter of moduli 15 and 17 have been verified by observation through LEDs and thereby completing the hardware implementation of the said binary to residue converter.

8.6 Discussion

The VHDL coding, simulation and CPLD implementation have been studied for 12-bit positive and negative binary to residue converter of moduli 15 and 17. From the VLSI implementation point of view, this study is very useful and can be considered as a foot stepping to the rigorous features of VLSI design in terms of area occupied by the system, overall power consumption, overall propagation delay etc. All these features could not be studied due to time constraint. I feel seriously that the work carried out by me will attract future researchers to make significant contribution in this domain. It is also to be noted that due to time-constraint, the VHDL codes have been presented in Word-form unlike the major part of the thesis.