8 Design Example: A Division-by-Constant Combinational Circuit

8.1 A general case

A combinational circuit which divides n-bit binary number by a ‘small’ constant $\beta$ has a modular structure of an iterative 1-D array circuit, similar to the structure of an incrementer or an adder. The carry propagates from left to right, and its values are limited by the divisor, $\beta$.

$$
\begin{array}{c}
\begin{array}{c}
\text{dividend} \\
(2^n \cdot c_n + a)
\end{array} \\
\begin{array}{c}
\text{divisor} \\
\beta
\end{array} \\
\begin{array}{c}
\text{quotient} \\
s
\end{array} \\
\begin{array}{c}
\text{remainder} \\
c_0
\end{array}
\end{array}
\]

\[\text{Input/output variables are related by the following equation which links divider, divisor, quotient and remainder:}
\]

- The objective is to build a combinational circuit, which, given n-bit input $a$ and possibly $c_n$, will generate the quotient $s$ and the remainder, $c_0$.
- It is possible to build such a circuit using 1-bit cells.

8.2 Binary-to-decimal conversion

- The division-by-constant circuit can be used for binary-to-decimal conversion.
- It is the “division-by-target” radix method, therefore $\beta = 10$.

- In the example we consider conversion of a binary number to a 4-digit decimal number.
- The largest 4-digit decimal number, 9999, is represented by a 14-bit binary number.
- The first level division-by-10 circuit generate the first digit $d_3$ as a remainder and a 10-digit quotient that is equivalent to 3-digit decimal number not greater than 999.
- The final level division-by-10 circuit generate two last decimal digits, $d_3, d_2$.
8.3 A 1-bit division-by-constant circuit

- Consider as an example a division-by-3 circuit. In this case, we have:

\[ n = 1; \quad \beta = 3; \quad m = 2; \quad \{2^m \geq \beta\} \]

- Input and output carry signals, \( c, d \), are 2-bit numbers which are less than the divisor, \( \beta = 3 \), that is:

\[ c, d \in \{0, 1, 2\} \]

- The I/O equation is now of the following form:

\[
2c + a = 3s + d
\]

### Function Table

<table>
<thead>
<tr>
<th>( c )</th>
<th>( a )</th>
<th>( 2c + a )</th>
<th>( 3s + d )</th>
<th>( s )</th>
<th>( d )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2 0</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2 1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3 0</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3 1</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Truth Table

<table>
<thead>
<tr>
<th>( c_1 )</th>
<th>( c_0 )</th>
<th>( a )</th>
<th>( s )</th>
<th>( d_1 )</th>
<th>( d_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>0 1</td>
<td>1</td>
<td>0 0 1</td>
<td>0 0 1</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1</td>
<td>0 1 1</td>
<td>0 1 1</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>0</td>
<td>0 0 1</td>
<td>0 0 1</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>1</td>
<td>1 0 1</td>
<td>1 0 1</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>1 0</td>
<td>0</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

- From the tables the logic equations for the three outputs can be derived using the Karnaugh map technique.

- One possible \( \text{SoP} \) form is as follows:

\[
s = c_1 + a \cdot c_0
\]

\[
d_1 = \overline{a} \cdot c_0 + a \cdot c_1
\]

\[
d_0 = \overline{a} \cdot c_1 + \overline{a} \cdot \overline{c_1} \cdot \overline{c_0}
\]

A possible VHDL implementation of the 1-bit cell, \( \text{div3b1} \) based on the derived logic equations is as follows:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

-- div3b1, 1-bit div-by-3 Entity Description
ENTITY div3b1 IS
PORT(
    c : IN std_logic_vector (1 downto 0) ;
    a : IN std_logic ;
    d : OUT std_logic_vector (1 downto 0) ;
    s : OUT std_logic
);
END div3b1;

-- arch1 Architecture Description
ARCHITECTURE arch1 OF div3b1 IS
BEGIN
    s <= (c(1) or (a and c(0)));
    d(1) <= (not a and c(0)) or (a and c(1));
    d(0) <= (not a and c(1)) or (a and not c(1) and not c(0));
END arch1;
```

- At this stage we might like to avoid deriving the logic equations and use the truth table directly, as in the 1-bit adder example.

- We specify the truth table as a constant as it is illustrated in the next section.
8.4 An architecture with the truth table specification

- The truth table is simply specified as a constant array of binary words. The array consists of $2^3$ 3-bit words as in the truth table on page 8–3.

**ARCHITECTURE dv3tt OF div3b1 IS**

**TYPE arr_vec IS ARRAY (natural range <>) OF std_logic_vector(2 downto 0);**

**CONSTANT sd_ac : arr_vec := (**

--- sd1d0 ca
--------------- the truth table:
"000", -- 0
"001", -- 1
"010", -- 2
"100", -- 3
"101", -- 4
"110", -- 5
"---", -- 6
"---"; -- 7

**SIGNAL sd : std_logic_vector (2 downto 0) ;**

**BEGIN**

-- reading from the truth table :

sd <= sd_ac(conv_integer(unsigned(c & a)));

d <= sd (1 downto 0) ;

s <= sd (2) ;

**END dv3tt ;**

- The truth table is a constant sd_ac of the type arr_vec.
- The value of the constant is our truth table. Note that we can also specify the “don’t care” values, ‘–’.
- To read the values from the truth table we need an assignment statement of the form

sd <= sd_ac(ca);

- In the array specification we have implicitly specified that the address signal ac is of the type integer.
- Therefore, we first concatenate (c & a) into a 3-bit std_logic_vector, which is subsequently converted into a 3-bit unsigned vector that can be converted into an integer.

- The conversion functions are specified in the libraries ieee.std_logic_1164 and ieee.std_logic_arith.

A.P. Papliński

8–5

- We consider another architecture of the 1-bit division-by-3 circuit, based on the division and remainder operations.

Such operators are available in the numeric_std IEEE library.

- The division `/` and remainder rem, like all other arithmetic operators, do not operate on signals of the type std_logic_vector.

Instead, we can use signals of the type unsigned.

- The conversion function std_logic_vector converts signals back from the unsigned to std_logic_vector form.

Since ca is a 3-bit signal, the results of division and remainder operations are also 3-bit signals.
8.5 An n-bit division-by-three circuit

- In order to implement an n-bit division-by-3 circuit, we can instantiate the 1-bit component div3b1 using the port map and generate statements.

- The resulting structure of a 4-bit division-by-3 circuit has the following block-diagram:

```
  +---+  +---+  +---+  +---+  +---+
  | ec(1:0) | ea(3:0) | ed(1:0) | es(3:0) |
  +---+  +---+  +---+  +---+  +---+
      |        |        |         |
      |        |        |         |
      |        |        |         |
      |        |        |         |
      +---+  +---+  +---+  +---+  +---+
  +---+  +---+  +---+  +---+  +---+
  | cc(4) | cc(0) | cc(3) | cc(2) | cc(1) |
  +---+  +---+  +---+  +---+  +---+
```

- The numbers represented by the port signals are related by the following arithmetic equality:

$$16 \cdot ec + ea = 3 \cdot es + ed$$

- The internal carry signals, cc, form a 5 by 2 array of 2-bit signals.

The block diagram of the 4-bit division-by-3 circuit is equivalent to the following VHDL code:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY div3b4 IS --- 4-bit division-by-3
  GENERIC (N : natural := 3, m : natural := 1);
  PORT( 
    ec : IN std_logic_vector(m downto 0);
    ea : IN std_logic_vector(N downto 0);
    ed : OUT std_logic_vector(m downto 0);
    es : OUT std_logic_vector(N downto 0));
END div3b4;
ARCHITECTURE strctrl OF div3b4 IS
  COMPONENT div3b1
    PORT( 
      c : IN std_logic_vector(1 downto 0);
      a : IN std_logic;
      d : OUT std_logic_vector(1 downto 0);
      s : OUT std_logic
    );
  END COMPONENT div3b1;
  TYPE arr5b2 IS ARRAY(4 downto 0) OF std_logic_vector(1 downto 0);
  SIGNAL cc : arr5b2;
BEGIN
  cc(4) <= ec;
  glp: FOR i IN ea'range GENERATE
    U1: div3b1
      PORT MAP (c => cc(i+1),
                 a => ea(i),
                 d => cc(i),
                 s => ea(i));
  END GENERATE;
  ed <= cc(0);
END strctrl;
```

Note that

- In the generate statement ea'range is equivalent to 4 downto 0.
- In the component instantiation statement, port map, we have used the association-by-name method.