Shift-Add Multiplication

Full details can be found in the course text: Computer Systems: Organization & Architecture; Carpinelli, J. Section 8.1.2. A summary is given below

Multiplication can be implemented by repeated addition. For example the sum 2x3 can be converted to 2+2+2. For multi-digit values this can be a very long process. We can improve this through a combination of shift and add.

The hardware to implement shift-add for simple binary values is straightforward. We require a register to hold the running total, two registers to hold the operands and a counter to track the number of additions. Assuming the operands are held in registers **X** and **Y**, with the result stored in the registers **U** (high order half) and **V** (low order half), then for *n* bit values the algorithm is

```
U = 0;
FOR i = 1 to n DO
    {
     IF ( Y  = 1 )
            0
        {
           CU = U + X;
        }
     shift right CUV;
     rotate right Y;
    }
```

Test the least significant bit of operand Y,

if the least significant bit of Y is set then add X to our running total in U, carry out is held in C

then always do the following regardless of the value of the least significant bit of Y

shift CUV right one bit, $C \rightarrow U_{n-1} ... U_0 \rightarrow V_{n-1} ... V_0 \rightarrow$ discard.

rotate right Y, $Y_0 \rightarrow Y_{n-1} \rightarrow Y_{n-2} \rightarrow ... Y_1 \rightarrow Y_0 \rightarrow$

RTL Code

The abstract RTL code should be considered with reference to *figure 1*. The CPU will load the X and Y registers with the operands and then set the **start** signal high. The processor will then wait until the Finish flag is set, at which point it will read the result from registers UV. The state counter is decoded to provide signals **1, 2, 3**. The loop counter, i, generate the **z** signal when it is decremented to zero.

state **1** $U \leftarrow 0, i \leftarrow n$

state **2** $i \leftarrow i - 1$, if $(Y_0 = 1)$ then $CU \leftarrow U + X$

state **3**   SHR(CUV), ror(Y), if **z** = 0 then goto state 2 else Finish $\leftarrow 1$

Simple Multiplier

An example trace multiplying two 4-bit values: $13_{10} * 11_{10}$ 1101*1011

Function	i	C	U	V	Y	Comments
U = 0		0000	0000	1011		
if Y_0=1	1	0	1101			Y_0 = 1, add
SHR(CUV)		0	0110	1000		
ROR(Y)					1101	
if Y_0 = 1	2	1	0011			Y_0 = 1, add
SHR(CUV)		0	1001	1100		
ROR(Y)					1110	
if Y_0 = 1	3					Y_0 = 0
SHR(CUV)		0	0100	1110		
ROR(Y)					0111	
if Y_0 = 1	4	1	0001			Y_0 = 1, add
SHR(CUV)			1000	1111		
ROR(Y)					1011	Original value
DONE			1000	1111		Result = 143

RTL for Algorithm:

```
    1:  U ← 0, i ← n
Y₀2:  CU ←U + X                    ;  Y₀ == 1
    2:  i ← i − 1                    ;  Y₀ == x
    3:  SHR(CUV), ROR(Y)
¬Z3:  GOTO 2
```

Trace of the RTL code for the shift-add algorithm.

Conditions	υ-operations	i	C	U	V	Y	Z	Finish
START		x	x	xxxx	xxxx	1011		0
1	U←0, i←4	4		0000			0	
$Y_0 \wedge$ **2**, **2**	CU← U + X, i ←i − 1	0 3	0	1101			0	
3 ¬Z ∧ **3**	SHR(CUV), ROR(Y) GOTO **2**		0	0110	1xxx	1101		
$Y_0 \wedge$ **2**, **2**	CU← U + X, i ←i − 1	1 2	1	0011			0	
3 ¬Z ∧ **3**	SHR(CUV), ROR(Y) GOTO **2**		0	1001	11xx	1110		
2	i ←i − 1	1					0	
3 ¬Z ∧ **3**	SHR(CUV), ROR(Y) GOTO **2**		0	0100	111x	0111		
$Y_0 \wedge$ **2**, **2**	CU← U + X, i ←i − 1	1 0	1	0001			1	
3 Z ∧ **3**	SHR(CUV), ROR(Y) FINISH ← 1		0	1000	1111	1011		1

The microinstruction GOTO 2 is generated by driving the load input on the state counter. This loads a fixed value into the state counter which is decoded as state 2.

```
State counter    state
00               1
01               2
10               3
11               unused
```

The state signals, along with Y_0, Z, START and FINISH, drive the system.
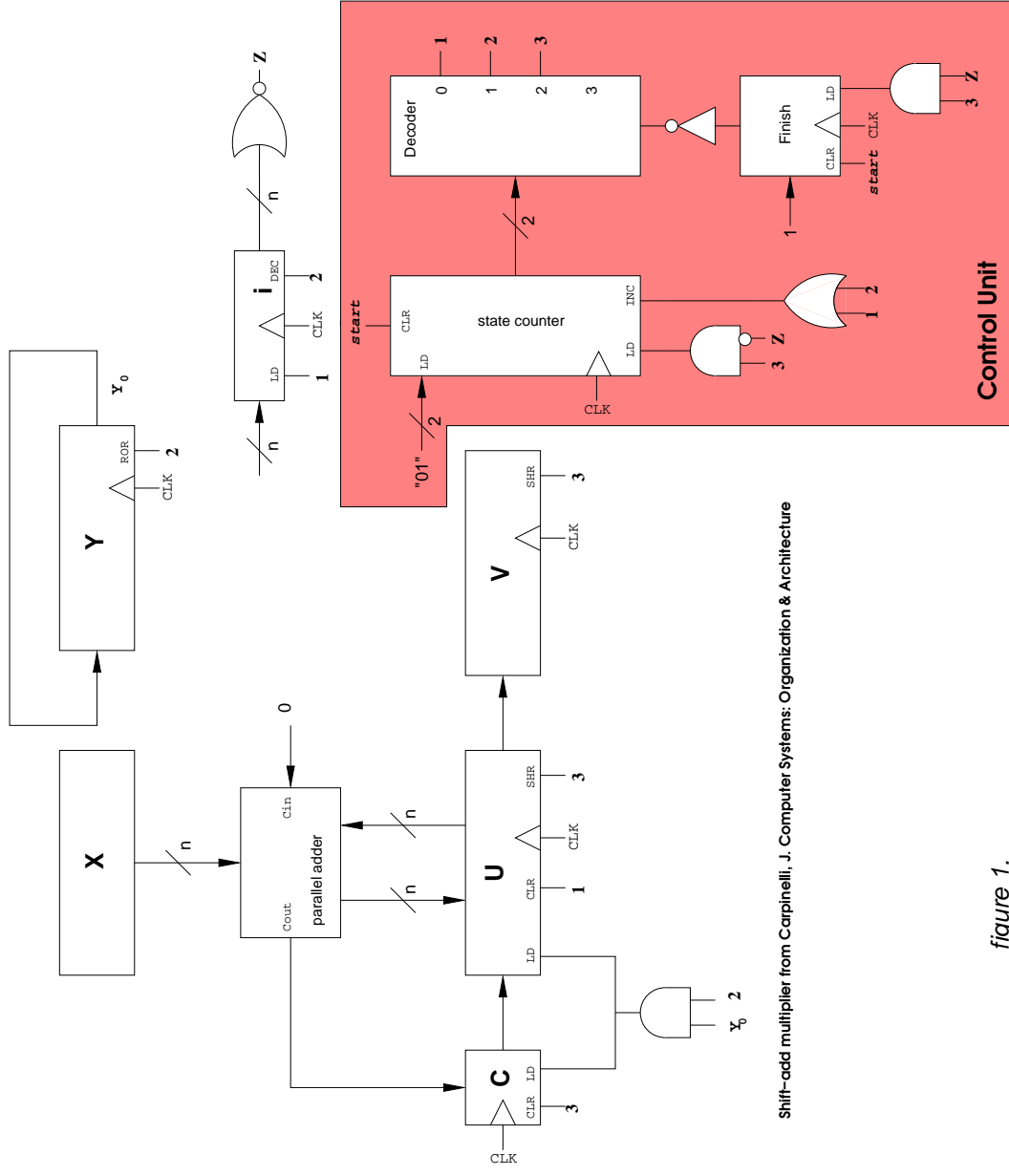
Shift–add multiplier from Carpinelli, J. Computer Systems: Organization & Architecture

figure 1.

```vhdl
-------------------------------------------------------------------------------
-- Title      : Example shift register for the shift-add multiplier
-- Project    :
-------------------------------------------------------------------------------
-- File       : sh_add_reg.vhd
-- Author     :   <ngunton@ptolome.cems.uwe.ac.uk>
-- Company    : DEDM, FET, UWE
-- Created    : 2010-04-23
-- Last update: 2010-04-23
-- Platform   : Alliance 5.0 on GNU/Linux
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: Alliance compliant register with right shift, msb in, lsb out
-------------------------------------------------------------------------------
-- Copyright (c) 2010 DEDM, FET, UWE
-------------------------------------------------------------------------------
-- Revisions  : 1
-- Date        Version  Author  Description
-- 2010-04-23  1.0      ngunton  Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity sh_add_reg is
  generic (
    n : natural := 8);
  port (
    clk, clr, ld, sh : in  std_logic;
    d                : in  std_logic_vector(n-1 downto 0);
    q                : out std_logic_vector(n-1 downto 0);
    msb_in           : in  std_logic;
    lsb_out          : out std_logic);
end sh_add_reg;
architecture dataflow of sh_add_reg is
  signal d_reg : std_logic_vector(n-1 downto 0);
begin  -- dataflow
  lsb_out <= d_reg(0);                    -- always make the lsb available
  q <= d_reg;                             -- and the q output
  -- purpose: load sum from adder
  -- type   : sequential
  -- inputs : clk, clr, d, ld
  -- outputs: q
  load: process (clk, clr)
  begin  -- process load
    if clk'event and clk = '1' then  -- rising clock edge
      if clr = '1' then                     -- synchronous reset (active high)
        d_reg <= (others => '0');
      elsif ld = '1' then
        d_reg <= d;
      elsif sh = '1' then
        d_reg <= msb_in & d_reg(n-1 downto 1);
      end if;
    end if;
  end process load;
end dataflow;
```