Sequential Circuits

- **Combinational** – output depends only on the input.
- **Sequential** – output depends on input and past behavior.
  - Require use of storage elements.
  - Contents of storage elements is called state.
  - Circuit goes through sequence of states as a result of changes in inputs.

![Figure 7.1](Images/figure7.1.png)  
Control of an alarm system

![Figure 7.2](Images/figure7.2.png)  
A simple memory element

![Figure 7.3](Images/figure7.3.png)  
A controlled memory element

![Figure 7.4](Images/figure7.4.png)  
A memory element with NOR gates
Figure 7.10 Master-slave D flip-flop

Figure 7.11 A positive-edge-triggered D flip-flop

Figure 7.12 Comparison of level-sensitive and edge-triggered

Figure 7.13 Master-slave D flip-flop with Clear and Preset

Figure 7.14 Positive-edge-triggered D flip-flop with Clear and Preset

Figure 7.15 Synchronous reset for a D flip-flop
Summary of Terminology

- Basic latch – cross-coupled NAND/NOR
- Gated latch – output changes when clk = 1.
  - Gated SR latch
  - Gated D latch
- Flip-flop – output changes only on clk edge.
  - Edge-triggered flip-flop
  - Master-slave flip-flop
Figure 7.21 A three-bit down-counter

Table 7.1 Derivation of the synchronous up-counter

Clock cycle | Q2 Q1 Q0 | Q2 changes
--- | --- | ---
0 | 0 0 0 | 0 0 0
1 | 0 0 1 | 1 0 0
2 | 0 1 0 | 0 1 0
3 | 0 1 1 | 1 1 1
4 | 1 0 0 | 0 0 0
5 | 1 0 1 | 1 1 1
6 | 1 1 0 | 0 0 0
7 | 1 1 1 | 1 1 1
8 | 0 0 0 | 0 0 0

Figure 7.22 A four-bit synchronous up-counter

Figure 7.23 Inclusion of enable and clear capability

Figure 7.24 A four-bit counter with D flip-flops

Figure 7.25 A counter with parallel load capability
Figure 7.26 A modulo-6 counter with synchronous reset

Figure 7.27 A modulo-6 counter with asynchronous reset

Figure 7.28 A two-digit BCD counter

Figure 7.29 Ring counter

Figure 7.30 Johnson counter

Figure 7.31 Three types of storage elements in a schematic
Figure 7.32  Gated D latch generated by CAD tools

Figure 7.33  Implementation of a circuit in a CPLD

Figure 7.34  Timing simulation of storage elements

Figure 7.35  Instantiating a D flip-flop from a package

Figure 7.36  Implied memory

Figure 7.37  Code for a gated D latch
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
  PORT ( D, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS (Clock)
  BEGIN
    IF Clock'EVENT AND Clock = '1' THEN
      Q <= D;
    END IF;
  END PROCESS;
END Behavior;

Figure 7.37 Code for a D flip-flop

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
  PORT ( D, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL Clock'EVENT AND Clock = '1';
    Q <= D;
  END PROCESS;
END Behavior;

Figure 7.38 Code for a D flip-flop using WAIT UNTIL

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
  PORT ( D, Resetn, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN
      Q <= '0';
    ELSIF Clock'EVENT AND Clock = '1' THEN
      Q <= D;
    END IF;
  END PROCESS;
END Behavior;

Figure 7.40 D flip-flop with asynchronous reset

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
  PORT ( D, Resetn, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL Clock'EVENT AND Clock = '1';
    IF Resetn = '0' THEN
      Q <= '0';
    ELSE
      Q <= D;
    END IF;
  END PROCESS;
END Behavior;

Figure 7.41 D flip-flop with synchronous reset

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
  PORT ( D, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL Clock'EVENT AND Clock = '1';
    IF D = '0' THEN
      Q <= '0';
    ELSE
      Q <= D;
    END IF;
  END PROCESS;
END Behavior;

Figure 7.42 The lpm_ff parameterized flip-flop module

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
  PORT ( D, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL Clock'EVENT AND Clock = '1';
    IF D = '0' THEN
      Q <= '0';
    ELSE
      Q <= D;
    END IF;
  END PROCESS;
END Behavior;

Figure 7.43 An adder with registered feedback
Timing Constraints

- 2 ns from register clocked to data output.
- 8 ns for adder to produce sum.
- 4 ns for sum to propagate to register input.
- 3 ns for setup time.
- Clock cycle time must be 17 ns.
Hierarchical code for a four-bit shift register

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY shift4 IS
  PORT ( R : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        L, w, Clock : IN STD_LOGIC;
        Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) );
END shift4;
ARCHITECTURE Structure OF shift4 IS
  COMPONENT muxdff
    PORT ( D0, D1, Sel, Clock : IN STD_LOGIC;
           Q : OUT STD_LOGIC );
  END COMPONENT;
  BEGIN
    Stage3: muxdff PORT MAP ( w, R(3), L, Clock, Q(3) );
    Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock, Q(2) );
    Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock, Q(1) );
    Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock, Q(0) );
  END Structure;
END;
```

Alternative code for a shift register

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY shift4 IS
  PORT ( R : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        Clock : IN STD_LOGIC;
        L, w : IN STD_LOGIC;
        Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) );
END shift4;
ARCHITECTURE Behavior OF shift4 IS
  BEGIN
    PROCESS
      BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF L = '1' THEN
          Q <= R;
        ELSE
          Q(0) <= Q(1);
          Q(1) <= Q(2);
          Q(2) <= Q(3);
          Q(3) <= w;
        END IF;
      END PROCESS;
    END;
END Behavior;
```

Code that reverses the ordering of statements

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY shift4 IS
  PORT ( R : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        Clock : IN STD_LOGIC;
        L, w : IN STD_LOGIC;
        Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) );
END shift4;
ARCHITECTURE Behavior OF shift4 IS
  BEGIN
    PROCESS
      BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF L = '1' THEN
          Q <= R;
        ELSE
          Q(3) <= w;
          Q(2) <= Q(3);
          Q(1) <= Q(2);
          Q(0) <= Q(1);
        END IF;
      END PROCESS;
    END;
END Behavior;
```

Code for an n-bit left-to-right shift register

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY shiftn IS
  GENERIC ( N : INTEGER := 8 );
  PORT ( R : IN STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
        Clock : IN STD_LOGIC;
        L, w : IN STD_LOGIC;
        Q : BUFFER STD_LOGIC_VECTOR(N - 1 DOWNTO 0) );
END shiftn;
ARCHITECTURE Behavior OF shiftn IS
  BEGIN
    PROCESS
      BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF L = '1' THEN
          Q <= R;
        ELSE
          Genbits : FOR i IN 0 TO N - 2 LOOP
            Q(i) <= Q(i+1);
          END LOOP;
          Q(N - 1) <= w;
        END IF;
      END PROCESS;
    END;
END Behavior;
```

Code for a four-bit up-counter

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY upcount IS
  PORT ( R : IN INTEGER RANGE 0 TO 15;
        Clock, Resetn, L : IN STD_LOGIC;
        Q : BUFFER INTEGER RANGE 0 TO 15 )
END upcount;
ARCHITECTURE Behavior OF upcount IS
  BEGIN
    PROCESS ( Clock, Resetn )
      BEGIN
        IF Resetn = '0' THEN
          Q <= 0;
        ELSIF Clock'EVENT AND Clock = '1' THEN
          Q <= L;
        END IF;
      END PROCESS;
END Behavior;
```

A four-bit counter with parallel load, using INTEGER signals

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY upcount IS
  PORT ( R : IN INTEGER RANGE 0 TO 15;
        Clock, Resetn, L : IN STD_LOGIC;
        Q : BUFFER INTEGER RANGE 0 TO 15 )
END upcount;
ARCHITECTURE Behavior OF upcount IS
  BEGIN
    PROCESS ( Clock, Resetn )
      BEGIN
        IF Resetn = '0' THEN
          Q <= 0;
        ELSIF Clock'EVENT AND Clock = '1' THEN
          Q <= L;
        END IF;
      END PROCESS;
END Behavior;
```

Figure 7.49

Figure 7.50

Figure 7.51

Figure 7.52

Figure 7.53

Figure 7.54

Figure 7.55

Figure 7.56
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY downcnt IS
    GENERIC (modulus : INTEGER := 8);
    PORT (Clock, L, E : IN STD_LOGIC;
        Q : OUT INTEGER RANGE 0 TO modulus - 1);
END downcnt;
ARCHITECTURE Behavior OF downcnt IS
    SIGNAL Count: INTEGER RANGE 0 TO modulus - 1;
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF E = '1' THEN
            IF L = '1' THEN
                Count <= modulus - 1;
            ELSE
                Count <= Count - 1;
            END IF;
        END IF;
    END PROCESS;
    Q <= Count;
END Behavior;

Register Swapping

- Consider 3 register example (R1, R2, R3) that swaps contents of R1 and R2 using R3.
- Three steps:
  - Contents of R2 transferred to R3.
  - Contents of R1 transferred to R2.
  - Contents of R3 transferred to R1.
A control circuit that does not require flip-flop preset inputs

Using multiplexers to implement a bus

Code for an n-bit register with enable

Code for an n-bit tri-state buffer

Code for the shift-register controller

Package and component declarations
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.components.all;

ENTITY swap IS
PORT ( Data : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  Resetn, w : IN STD_LOGIC;
  Clock, Extern : IN STD_LOGIC;
  RinExt : IN STD_LOGIC_VECTOR(1 TO 3);
  BusWires : BUFFER STD_LOGIC_VECTOR(7 DOWNTO 0) );
END swap;

ARCHITECTURE Behavior OF swap IS
SIGNAL Rin, Rout, Q: STD_LOGIC_VECTOR(1 TO 3);
SIGNAL R1, R2, R3: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
control: shiftr GENERIC MAP (K => 3)
PORT MAP (Resetn, Clock, w, Q);
Rin (1) <= RinExt (1) OR Q(3);
Rin (2) <= RinExt (2) OR Q(2);
Rin (3) <= RinExt (3) OR Q(1);
Rout (1) <= Q(2); Rout (2) <= Q(1); Rout (3) <= Q(3);
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.components.all;

ENTITY swapmux IS
PORT ( Data : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  Resetn, w : IN STD_LOGIC;
  Clock : IN STD_LOGIC;
  RinExt : IN STD_LOGIC_VECTOR(1 TO 3);
  BusWires : BUFFER STD_LOGIC_VECTOR(7 DOWNTO 0) );
END swapmux;

ARCHITECTURE Behavior OF swapmux IS
SIGNAL Rin, Q: STD_LOGIC_VECTOR(1 TO 3);
SIGNAL S: STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL R1, R2, R3: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
control: shiftr GENERIC MAP (K => 3)
PORT MAP (Resetn, Clock, w, Q);
Rin (1) <= RinExt (1) OR Q(3);
Rin (2) <= RinExt (2) OR Q(2);
Rin (3) <= RinExt (3) OR Q(1);
Rout (1) <= Q(2); Rout (2) <= Q(1); Rout (3) <= Q(3);
reg1: regn PORT MAP (BusWires, Rin (1), Clock, R1);
reg2: regn PORT MAP (BusWires, Rin (2), Clock, R2);
reg3: regn PORT MAP (BusWires, Rin (3), Clock, R3);
encoder:
WITH S SELECT
  S <= "00" WHEN "000",
       "10" WHEN "010",
       "01" WHEN "100",
       OTHERS WHEN OTHERS;
muxes : -- eight 4-to-1 multiplexers
WITH S SELECT
  BusWires <= Data WHEN "000",
              R2 WHEN "010",
              R1 WHEN "100",
              R3 WHEN OTHERS;
END Behavior;

ENTITY declaration not shown
ARCHITECTURE Behavior OF swapmux IS
SIGNAL Rin, Q: STD_LOGIC_VECTOR(1 TO 3);
SIGNAL R1, R2, R3: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
control: shiftr GENERIC MAP (K => 3)
PORT MAP (Resetn, Clock, w, Q);
Rin (1) <= RinExt (1) OR Q(3);
Rin (2) <= RinExt (2) OR Q(2);
Rin (3) <= RinExt (3) OR Q(1);
Rout (1) <= Q(2); Rout (2) <= Q(1); Rout (3) <= Q(3);
muxes:
WITH Q SELECT
  BusWires <= Data WHEN "000",
              R2 WHEN "010",
              R1 WHEN "100",
              R3 WHEN OTHERS;
END Behavior;

Figure 7.67a Using multiplexers to implement a bus

Figure 7.67b Using multiplexers to implement a bus (cont')
Processor Control Logic

Clear = $T_0 + $Done
FRin = $T_0$
Extern = $T_0$
Done = ($I_0 + I_1)T_1 + (I_2 + I_3)T_3$
A\_in = ($I_0 + I_1)T_1$
G\_in = ($I_2 + I_3)T_2$
G\_out = ($I_2 + I_3)T_3$
Add/Sub = $I_3$
R0\_in = ($I_0 + I_1)T_0X_0 + (I_2 + I_3)T_0X_0$
R0\_out = $I_1T_1Y_0 + (I_2 + I_3)(T_1X_0 + T_2Y_0)$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Function performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Re, F0</td>
<td>$R_1 \leftarrow $Data</td>
</tr>
<tr>
<td>M$\rightarrow$ R, R0</td>
<td>$R_1 \leftarrow $R0</td>
</tr>
<tr>
<td>Artif Re, R0</td>
<td>$R_1 \leftarrow $Re $+$ R0</td>
</tr>
<tr>
<td>MUL Re, R0</td>
<td>$R_1 \leftarrow $Re $-$ R0</td>
</tr>
</tbody>
</table>

Table 7.2: Operations performed in the processor

Table 7.3: Control signals asserted in each operation/time step
BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);
end spcount;
PORT MAP (Clear, Clock, Count);  
PORT MAP (Func, F & R, Y);
PORT MAP ( RegGENERIC M (N => 6));
end RegCnt;
END;

end spcount;

BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);  
end spcount;
PORT MAP (Clear, Clock, Count);
Tri0: trin PORT MAP ( R0, Rout(0), BusWires);
Tri1: trin PORT MAP ( R1, Rout(1), BusWires);
Tri2: trin PORT MAP ( R2, Rout(2), BusWires);
Tri3: trin PORT MAP ( R3, Rout(3), BusWires);
Reg3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
RegA: regn PORT MAP (BusWires, Ain, Clock, A);
RegG: regn PORT MAP (Sum, Gin, Clock, G);
RegP: regn PORT MAP (F, Rx, Ry);
triG: trin PORT MAP (G, Gout, BusWires);
END;

BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);  
end spcount;
PORT MAP (Clear, Clock, Count);
Tri0: trin PORT MAP ( R0, Rout(0), BusWires);
Tri1: trin PORT MAP ( R1, Rout(1), BusWires);
Tri2: trin PORT MAP ( R2, Rout(2), BusWires);
Tri3: trin PORT MAP ( R3, Rout(3), BusWires);
Reg3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
RegA: regn PORT MAP (BusWires, Ain, Clock, A);
RegG: regn PORT MAP (Sum, Gin, Clock, G);
RegP: regn PORT MAP (F, Rx, Ry);
triG: trin PORT MAP (G, Gout, BusWires);
END;

BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);  
end spcount;
PORT MAP (Clear, Clock, Count);
Tri0: trin PORT MAP ( R0, Rout(0), BusWires);
Tri1: trin PORT MAP ( R1, Rout(1), BusWires);
Tri2: trin PORT MAP ( R2, Rout(2), BusWires);
Tri3: trin PORT MAP ( R3, Rout(3), BusWires);
Reg3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
RegA: regn PORT MAP (BusWires, Ain, Clock, A);
RegG: regn PORT MAP (Sum, Gin, Clock, G);
RegP: regn PORT MAP (F, Rx, Ry);
triG: trin PORT MAP (G, Gout, BusWires);
END;

BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);  
end spcount;
PORT MAP (Clear, Clock, Count);
Tri0: trin PORT MAP ( R0, Rout(0), BusWires);
Tri1: trin PORT MAP ( R1, Rout(1), BusWires);
Tri2: trin PORT MAP ( R2, Rout(2), BusWires);
Tri3: trin PORT MAP ( R3, Rout(3), BusWires);
Reg3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
RegA: regn PORT MAP (BusWires, Ain, Clock, A);
RegG: regn PORT MAP (Sum, Gin, Clock, G);
RegP: regn PORT MAP (F, Rx, Ry);
triG: trin PORT MAP (G, Gout, BusWires);
END;

BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);  
end spcount;
PORT MAP (Clear, Clock, Count);
Tri0: trin PORT MAP ( R0, Rout(0), BusWires);
Tri1: trin PORT MAP ( R1, Rout(1), BusWires);
Tri2: trin PORT MAP ( R2, Rout(2), BusWires);
Tri3: trin PORT MAP ( R3, Rout(3), BusWires);
Reg3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
RegA: regn PORT MAP (BusWires, Ain, Clock, A);
RegG: regn PORT MAP (Sum, Gin, Clock, G);
RegP: regn PORT MAP (F, Rx, Ry);
triG: trin PORT MAP (G, Gout, BusWires);
END;

BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);  
end spcount;
PORT MAP (Clear, Clock, Count);
 Tri0: trin PORT MAP ( R0, Rout(0), BusWires);
Tri1: trin PORT MAP ( R1, Rout(1), BusWires);
Tri2: trin PORT MAP ( R2, Rout(2), BusWires);
Tri3: trin PORT MAP ( R3, Rout(3), BusWires);
Reg3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
RegA: regn PORT MAP (BusWires, Ain, Clock, A);
RegG: regn PORT MAP (Sum, Gin, Clock, G);
RegP: regn PORT MAP (F, Rx, Ry);
triG: trin PORT MAP (G, Gout, BusWires);
END;

BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);  
end spcount;
PORT MAP (Clear, Clock, Count);
Tri0: trin PORT MAP ( R0, Rout(0), BusWires);
Tri1: trin PORT MAP ( R1, Rout(1), BusWires);
Tri2: trin PORT MAP ( R2, Rout(2), BusWires);
Tri3: trin PORT MAP ( R3, Rout(3), BusWires);
Reg3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
RegA: regn PORT MAP (BusWires, Ain, Clock, A);
RegG: regn PORT MAP (Sum, Gin, Clock, G);
RegP: regn PORT MAP (F, Rx, Ry);
triG: trin PORT MAP (G, Gout, BusWires);
END;

BEGIN
Zero <= "00"; High <= '1';
Clear <= Reset OR Done OR (NOT T AND NOT Y);  
end spcount;
PORT MAP (Clear, Clock, Count);
Tri0: trin PORT MAP ( R0, Rout(0), BusWires);
Tri1: trin PORT MAP ( R1, Rout(1), BusWires);
Tri2: trin PORT MAP ( R2, Rout(2), BusWires);
Tri3: trin PORT MAP ( R3, Rout(3), BusWires);
Reg3: regn PORT MAP (BusWires, Rin(3), Clock, R3);
RegA: regn PORT MAP (BusWires, Ain, Clock, A);
RegG: regn PORT MAP (Sum, Gin, Clock, G);
RegP: regn PORT MAP (F, Rx, Ry);
triG: trin PORT MAP (G, Gout, BusWires);
END;
WHEN "00" => -- Load
WHEN "01" => -- Move
WHEN OTHERS => -- Add, Sub
Gout <= '1'; Rin <= X; Done <= '1';
END CASE;
END CASE;
END PROCESS;

reg0: regn PORT MAP (BusWires, Rin (0), Clock, R0);
reg1: regn PORT MAP (BusWires, Rin (1), Clock, R1);
reg2: regn PORT MAP (BusWires, Rin (2), Clock, R2);
reg3: regn PORT MAP (BusWires, Rin (3), Clock, R3);
regA : regn PORT MAP (Ain, Clock, A);
alu : WITH AddSub SELECT
Sum <= A + BusWires WHEN '0',
A - BusWires WHEN OTHERS;
regG : regn PORT MAP (Sum, Gin, Clock, G);
Sel <= Rout & Gout & Extern;
WITH Sel SELECT
BusWires <= R0 WHEN "100000",
R1 WHEN "010000",
R2 WHEN "001000",
R3 WHEN "000100",
G WHEN "000010",
Data WHEN OTHERS;
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY BCDcount IS
  PORT ( Clock : IN STD_LOGIC;
         Clear, E : IN STD_LOGIC;
         BCD1, BCD0 : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) );
END BCDcount;

ARCHITECTURE Behavior OF BCDcount IS
BEGIN
  PROCESS ( Clock )
  BEGIN
    IF Clock'EVENT AND Clock = '1' THEN
      IF Clear = '1' THEN
        BCD1 <= "0000"; BCD0 <= "0000";
      ELSE
        BCD1 <= BCD1 + '1';
      END IF;
      IF E = '1' THEN
        BCD0 <= BCD0 + '1';
      ELSE
        BCD0 <= '0000';
      END IF;
    END IF;
  END PROCESS;
END Behavior;
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reaction IS
PORT ( c9, Reset : IN STD_LOGIC;
       w, Pushn : IN STD_LOGIC;
       LEDn : OUT STD_LOGIC;
       Digit1, Digit0 : BUFFER STD_LOGIC_VECTOR(1 TO 7))
END reaction;

ARCHITECTURE Behavior OF reaction IS
BEGIN
  flipflop : PROCESS
  BEGIN
    WAIT UNTIL c9'EVENT AND c9 = '1';
    IF Pushn = '0' THEN
      LEDn <= '0';
    ELSIF w = '1' THEN
      LEDn <= '1';
    END IF;
  END PROCESS;
  LEDn <= NOT LEDn;
  counter: BCDcount PORT MAP (c9, Reset, LED, BCD1, BCD0);
  seg1: seg7 PORT MAP (BCD1, Digit1);
  seg0: seg7 PORT MAP (BCD0, Digit0);
END Behavior;

BEGIN
  flipflop: PROCESS
  BEGIN
    WAIT UNTIL c9'EVENT AND c9 = '1';
    IF Pushn = '0' THEN
      LEDn <= '0';
    ELSIF w = '1' THEN
      LEDn <= '1';
    END IF;
  END PROCESS;
  LEDn <= NOT LEDn;
  counter: BCDcount PORT MAP (c9, Reset, LED, BCD1, BCD0);
  seg1: seg7 PORT MAP (BCD1, Digit1);
  seg0: seg7 PORT MAP (BCD0, Digit0);
END Behavior;

Summary

• Latches / flip-flops
• Registers / counters
• VHDL
• Design examples