Digital System Design

Lecture 9
A Simple Processor

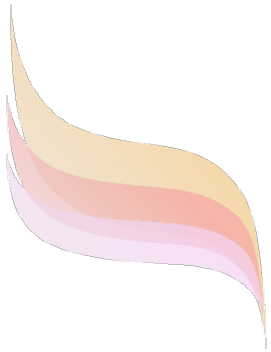
Reading Assignment:

- Brown, “Fundamentals of Digital Logic with VHDL, pp. 451 – 462

Learning Objective:

- Cover the basics of a simple computer system and present the design of an 8-bit system to illustrate the details of instruction execution.
- Provide an understanding of the basic principles of computer systems.

Register Swapping



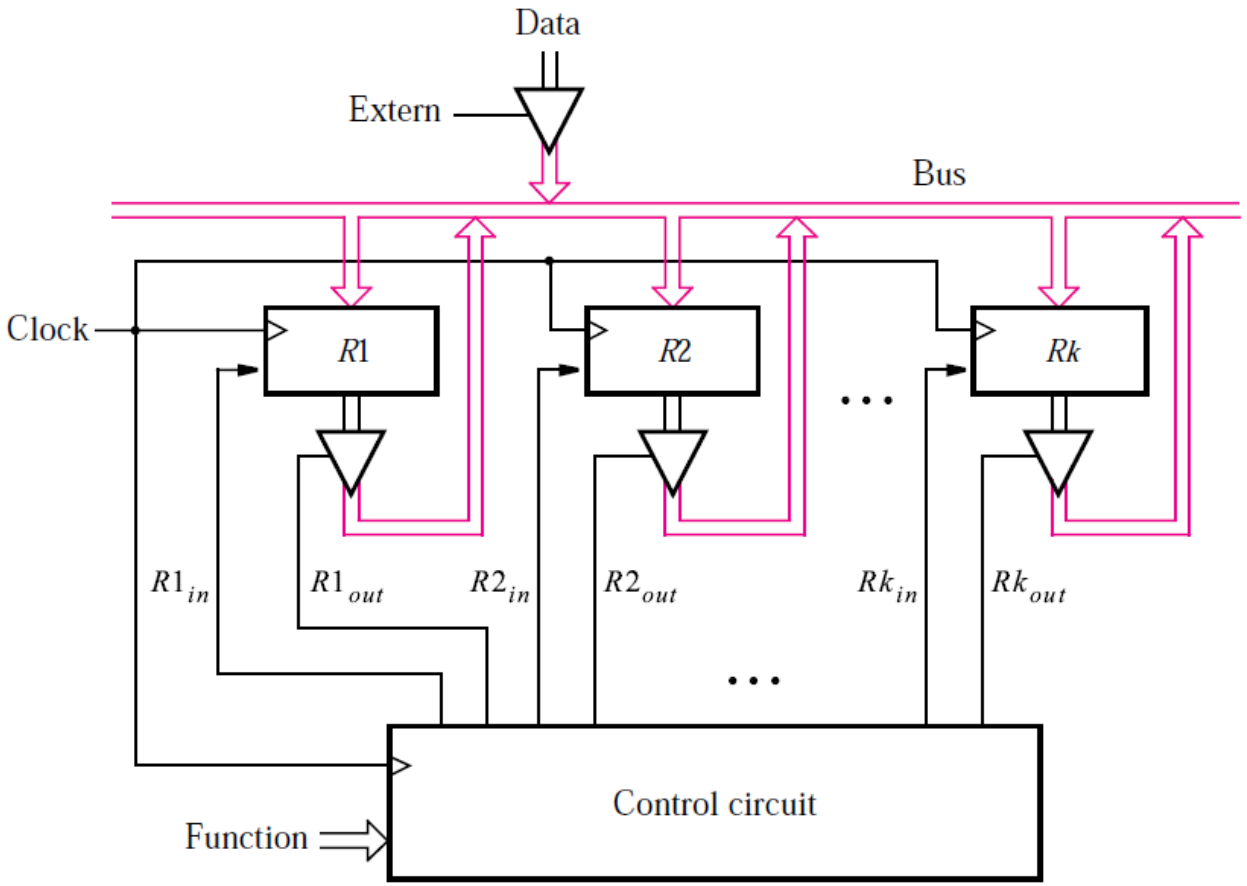
Register Swapping

Consider 3 register example (R1, R2, R3) that swaps contents of R1 and R2 using R3.

- The required swapping is done in three steps, each needing one clock cycle.
  - Contents of R2 transferred to R3.
  - Contents of R1 transferred to R2.
  - Contents of R3 transferred to R1.

A digital system with k registers

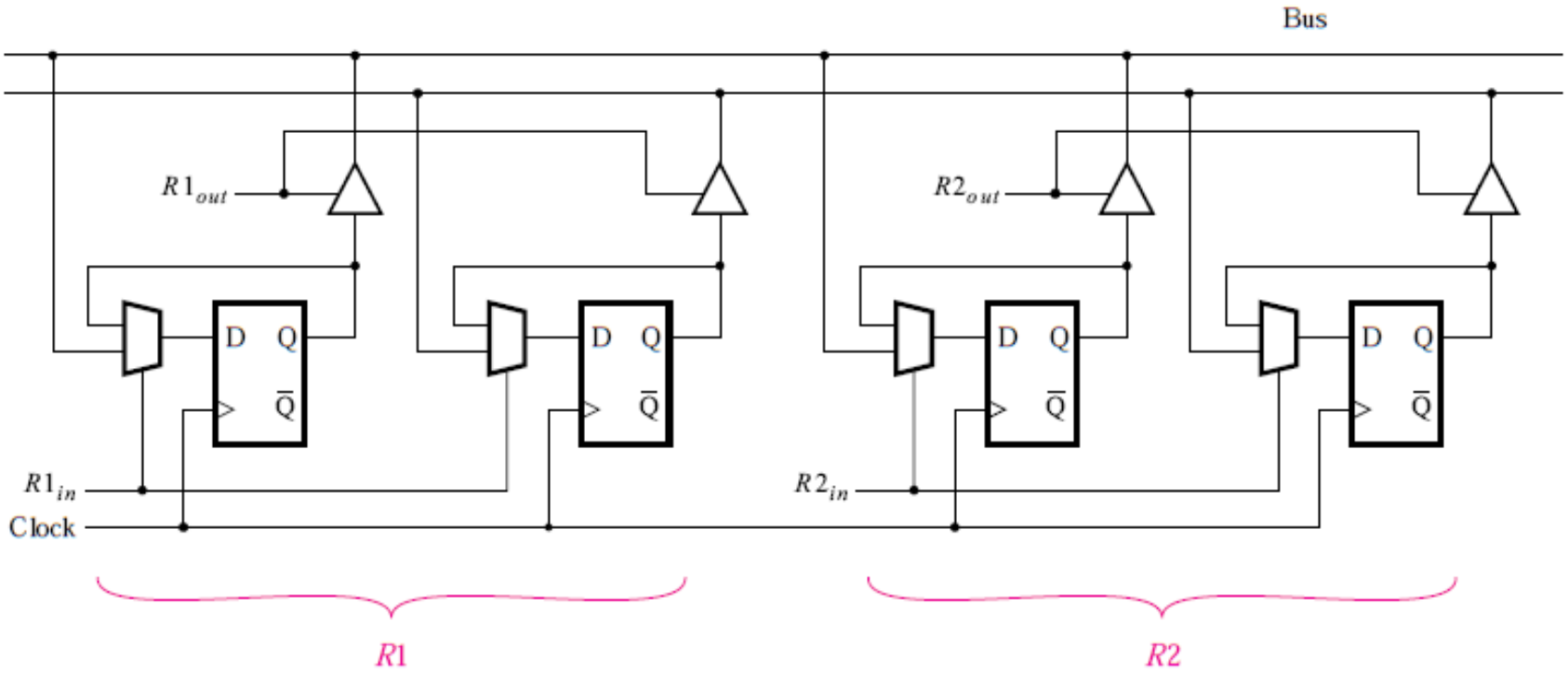
- A system with k *n-bit* registers, R_1 to R_k, each of which is connected to a common set of n wires, which are used to transfer data into and out of the registers.
- A control circuit is used to ensure that **only one** of the tri-state buffer enable inputs, $R1_{out}, \ldots, Rk_{out}$, is asserted at a given time. The control circuit also produces the signals $R1_{in}, \ldots, Rk_{in}$, which control when data is loaded into each register.
- an input signal Function instructs the control circuit to perform a particular task



A digital system with k registers

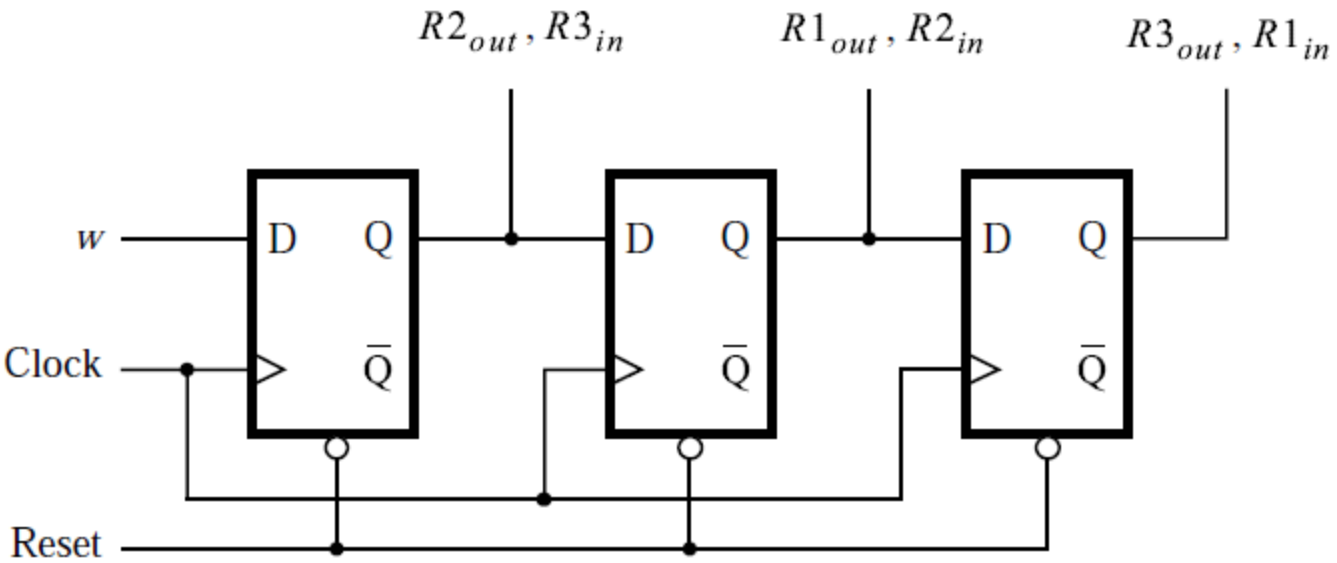
Details for connecting registers to a bus:

- To keep the picture simple, 2 two-bit registers are shown, but the same scheme can be used for larger registers.
- The D input on each flip-flop is connected to a 2-to-1 multiplexer, whose select input is controlled by $R1_{in}$.

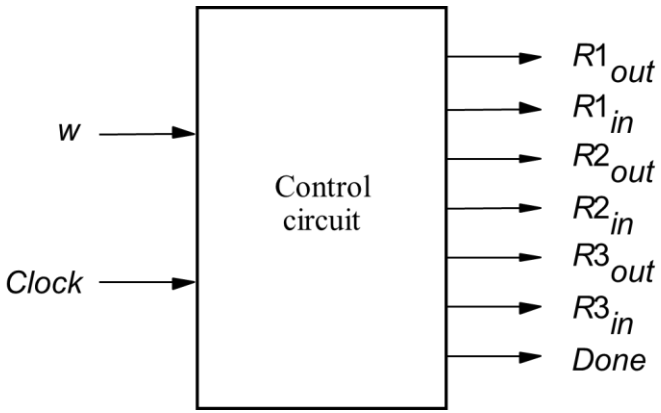


Register Swapping

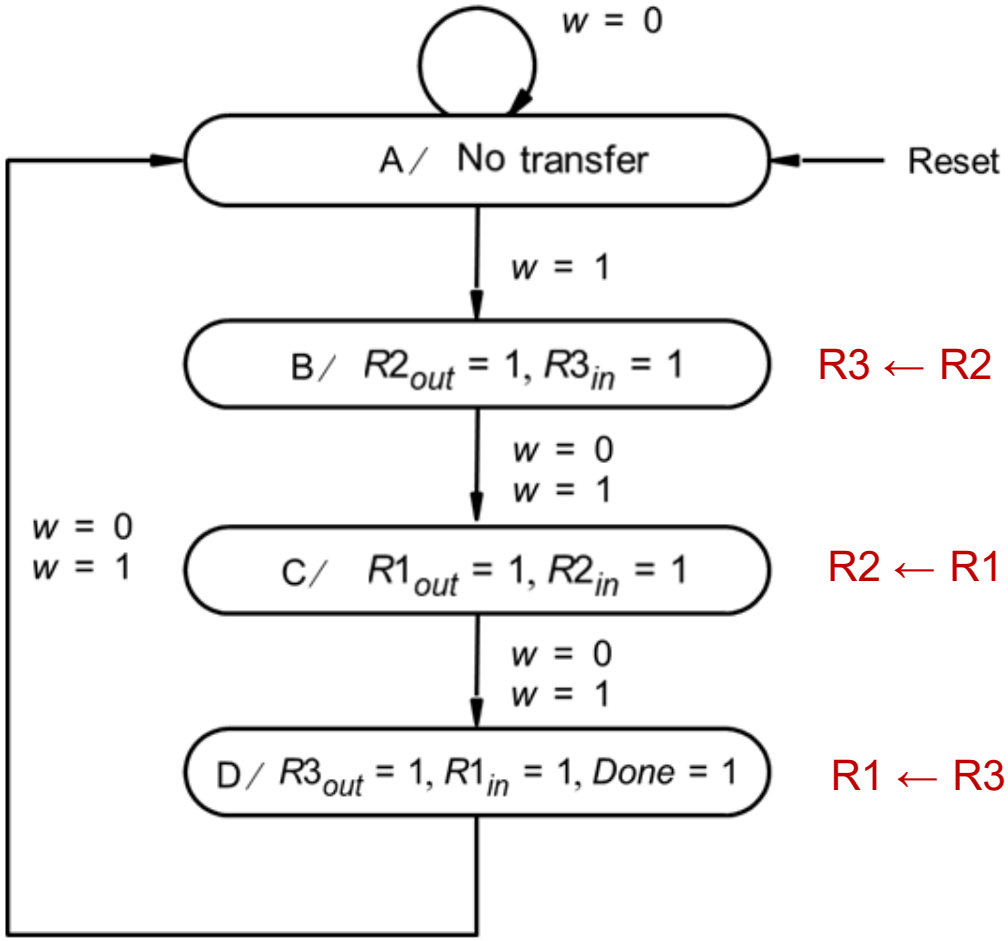
- There are many ways to design a suitable control circuit for the swap operation. One possibility is to use the left-to-right shift register.
  - The serial input w normally has the value 0.
  - We assume that changes in the value of w are synchronized to occur shortly after the active clock edge. This assumption is reasonable because w would normally be generated as the output of some circuit that is controlled by the same clock signal.
  - When the desired swap should be performed, w is set to 1 for one clock cycle, and then w returns to 0.



Finite State machine Controller for Register Swapping

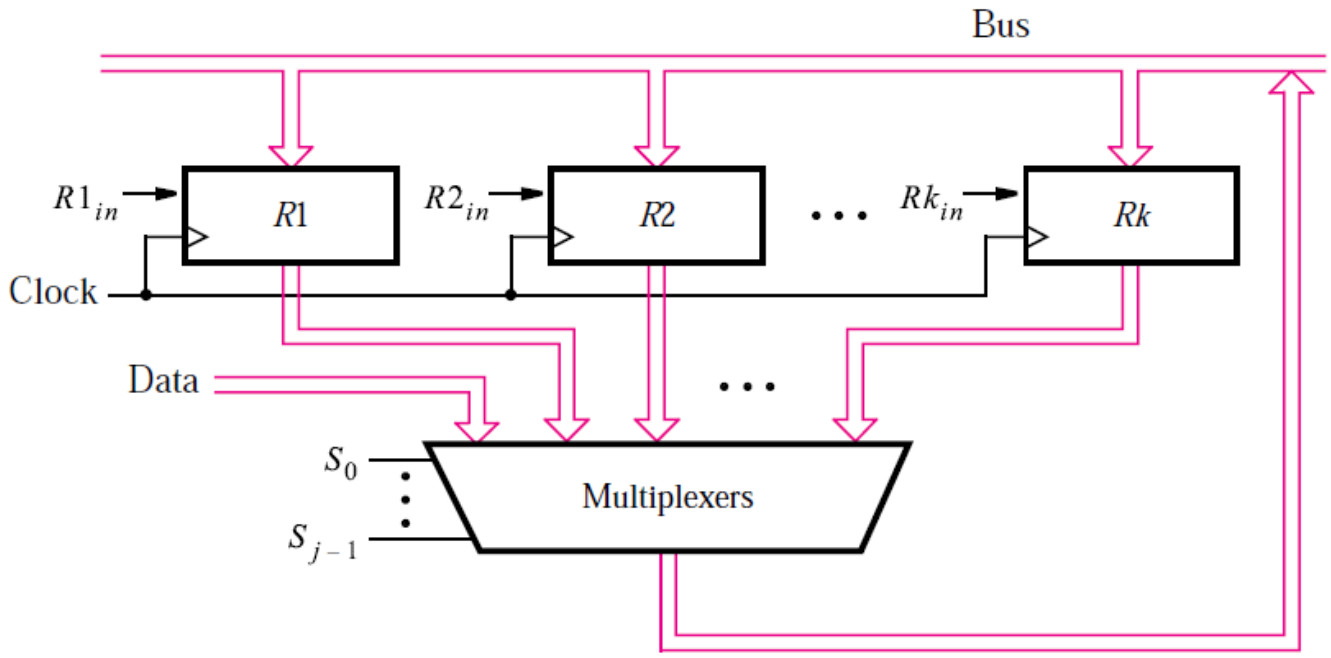


State diagram (Moore Machine)



Using a Multiplexer to Implement a Bus

- Instead of using tri-state buffers to control access to the bus, an alternative approach is to use multiplexers.
- The outputs of each register are connected to a multiplexer. The multiplexer's output is connected to the inputs of the registers, thus realizing the bus.
- In some types of chips, such as most PLDs, that do not contain a sufficient number of tri-state buffers to realize moderately large buses. In such chips the multiplexer based approach is the only practical alternative.



Subcircuits for Register Swapping

an n-bit register with enable

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

ENTITY regn IS
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (      R                   : IN        STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                    Rin, Clock : IN         STD_LOGIC ;
                    Q                   : OUT       STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE Behavior OF regn IS
BEGIN
        PROCESS
        BEGIN
                IF Clock'EVENT AND Clock = '1' THEN
                        IF Rin = '1' THEN
                                Q <= R ;
                        END IF ;
                END IF ;
        END PROCESS ;
END Behavior ;
```

Subcircuits for Register Swapping

an n-bit tri-state buffer

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

ENTITY trin IS
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (      X       : IN        STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                    E       : IN        STD_LOGIC ;
                    F       : OUT       STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END trin ;

ARCHITECTURE Behavior OF trin IS
BEGIN
        F <= (OTHERS => 'Z') WHEN E = '0' ELSE X ;
END Behavior ;
```

Subcircuits for Register Swapping

the shift-register controller

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY shiftr IS                    -- left-to-right shift register with async reset
        GENERIC ( K : INTEGER := 4 ) ;
        PORT (      Resetn, Clock, w        : IN            STD_LOGIC ;
                    Q                       : BUFFER        STD_LOGIC_VECTOR(1 TO K) ) ;
END shiftr ;

ARCHITECTURE Behavior OF shiftr IS
BEGIN
        PROCESS ( Resetn, Clock )
        BEGIN
                IF Resetn = '0' THEN
                        Q <= (OTHERS => '0') ;
                ELSIF Clock'EVENT AND Clock = '1' THEN
                        Genbits: FOR i IN K DOWNTO 2 LOOP
                                        Q(i) <= Q(i-1) ;
                        END LOOP ;
                        Q(1) <= w ;
                END IF ;
        END PROCESS ;
END Behavior ;
```

Package and component declarations for Register Swapping

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

PACKAGE components IS
    COMPONENT regn -- register
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (    R              : IN     STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                  Rin, Clock     : IN     STD_LOGIC ;
                  Q              : OUT    STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
    END COMPONENT ;

    COMPONENT shiftr -- left-to-right shift register with async reset
        GENERIC ( K : INTEGER := 4 ) ;
        PORT      (   Resetn, Clock, w     : IN       STD_LOGIC ;
                      Q                    : BUFFER   STD_LOGIC_VECTOR(1 TO K) ) ;
    END component ;

    COMPONENT trin -- tri-state buffers
        GENERIC ( N : INTEGER := 8 ) ;
        PORT (    X   : IN    STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                  E   : IN    STD_LOGIC ;
                  F   : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
    END COMPONENT ;
END components ;
```

VHDL for Register Swapping (using tris-tate buffers)

```
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 : INOUT   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) ;
```

VHDL for Register Swapping (using tris-tate buffers)

```
    tri_ext: trin PORT MAP ( Data, Extern, BusWires ) ;
    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 ) ;
    tri1: trin PORT MAP ( R1, Rout(1), BusWires ) ;
    tri2: trin PORT MAP ( R2, Rout(2), BusWires ) ;
    tri3: trin PORT MAP ( R3, Rout(3), BusWires ) ;
END Behavior ;
```

VHDL for Register Swapping (using multiplexers)

```
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 ) ;

        … con't
```

VHDL for Register Swapping (using multiplexers)

```
    Rin(1) <= RinExt(1) OR Q(3) ;
    Rin(2) <= RinExt(2) OR Q(2) ;
    Rin(3) <= RinExt(3) OR Q(1) ;
    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 Q SELECT
        S <= "00" WHEN "000",
             "10" WHEN "100",
             "01" WHEN "010",
             "11" WHEN OTHERS ;
    muxes: --eight 4-to-1 multiplexers
    WITH S SELECT
        BusWires <= Data WHEN "00",
                    R1 WHEN "01",
                    R2 WHEN "10",
                    R3 WHEN OTHERS ;
END Behavior ;
```

Note: When the shift register's contents are 000, the multiplexers select Data to be placed on the bus. This data is loaded into the register selected by RinExt. It is loaded into R1 if RinExt(1) = 1, R2 if RinExt(2) = 1, and R3 if RinExt(3) = 1.

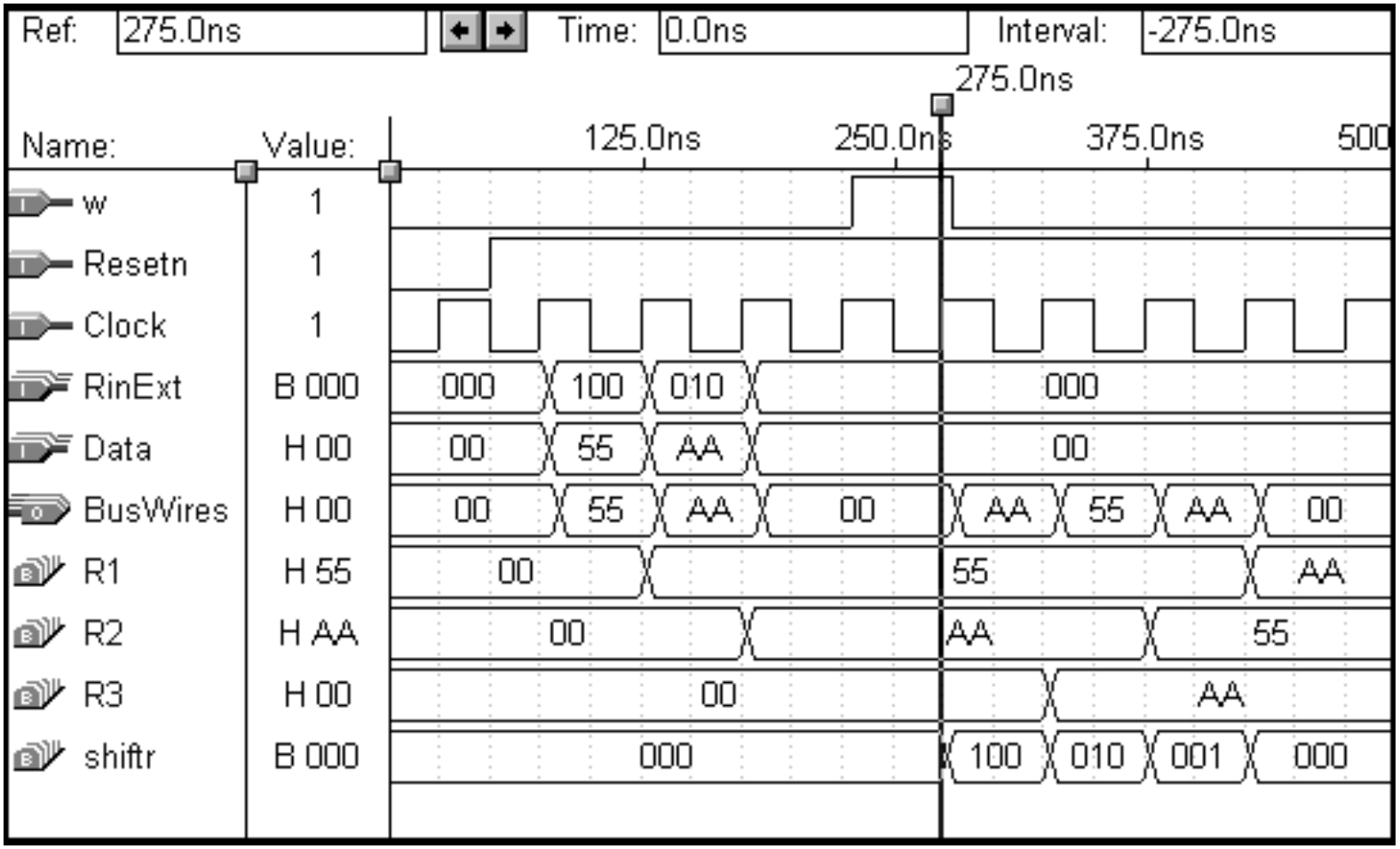
VHDL for Register Swapping (using multiplexers)

Simplified code for describing a bus

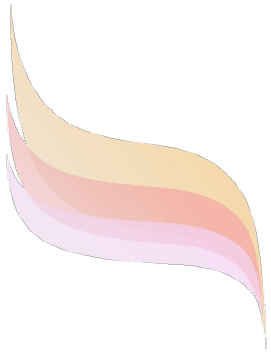
(ENTITY declaration not shown)

```
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) ;

    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 ) ;

    muxes:
    WITH Q SELECT
        BusWires <=  Data    WHEN "000",
                     R2      WHEN "100",
                     R1      WHEN "010",
                     R3      WHEN OTHERS ;
END Behavior ;
```

Timing simulation

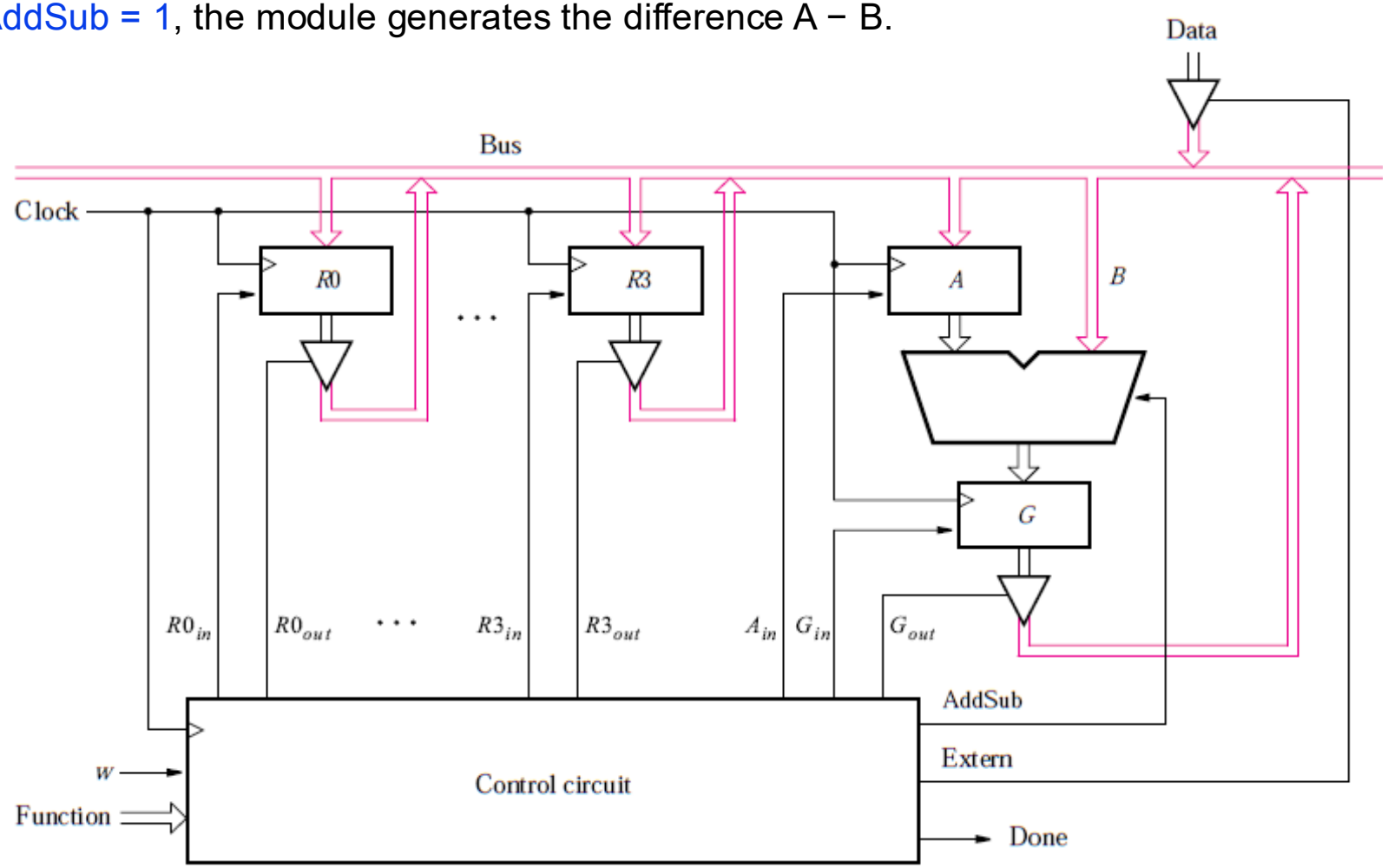


A Simple Processor



A Simple Processor

- It has four n-bit registers, R0, . . . , R3, that are connected to the bus using tri-state buffers.
- External data can be loaded into the registers from the n-bit Data input, which is connected to the bus using tri-state buffers enabled by the Extern control signal.
- It has an adder/subtractor module. If the AddSub = 0, the module generates the sum A + B; if AddSub = 1, the module generates the difference A − B.



Instructions performed in the processor

This system can perform different operations in each clock cycle, as governed by the control circuit.

- This circuit determines when particular data is placed onto the bus wires and it controls which of the registers is to be loaded with this data.
- For example, if the control circuit asserts the signals $R0_{out}$ and A_{in}, then the content of register R0 will be placed onto the bus and this data will be loaded by the next active clock edge into register A.

Operation	Function Performed
Load *Rx*, *Data*	$Rx \leftarrow Data$
Move *Rx*, *Ry*	$Rx \leftarrow [Ry]$
Add *Rx*, *Ry*	$Rx \leftarrow [Rx] + [Ry]$
Sub *Rx*, *Ry*	$Rx \leftarrow [Rx] - [Ry]$

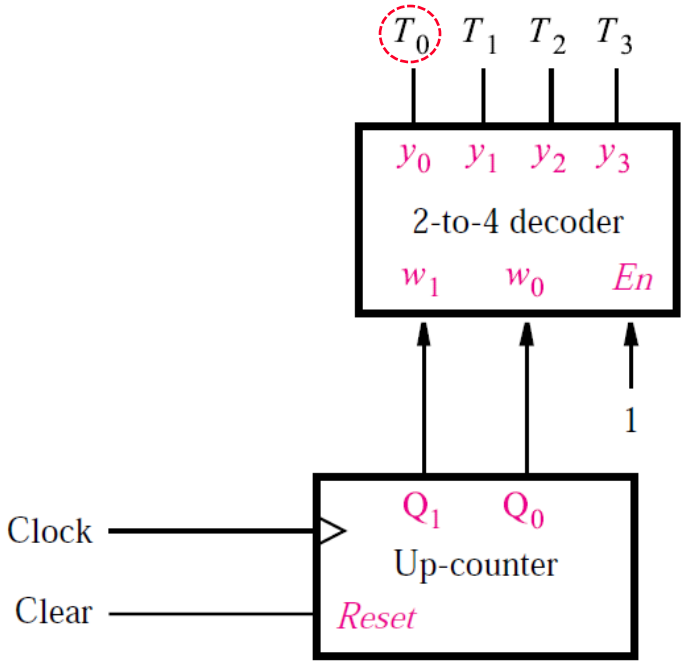
- the expression Rx ← Data indicates that the data on the external Data input is transferred across the bus into register Rx.
- The meaning of Rx ← [Ry] is the contents of register Ry are copied into register Rx.
- In the table the square brackets, as in [Rx], refer to the content of a register.

Control Signals

- Since only a single transfer across the bus is needed, both the Load and Move operations require only one step (clock cycle) to be completed.
- The Add and Sub operations require more than one clock cycle to complete, because multiple transfers have to be performed across the bus.
- The specific operation to be performed at any given time is indicated using the control circuit input named Function. The operation is initiated by setting the w input to 1, and the control circuit asserts the Done output when the operation is completed.
- Instead of using a shift register to design the control circuit, an alternate approach is based on a counter.

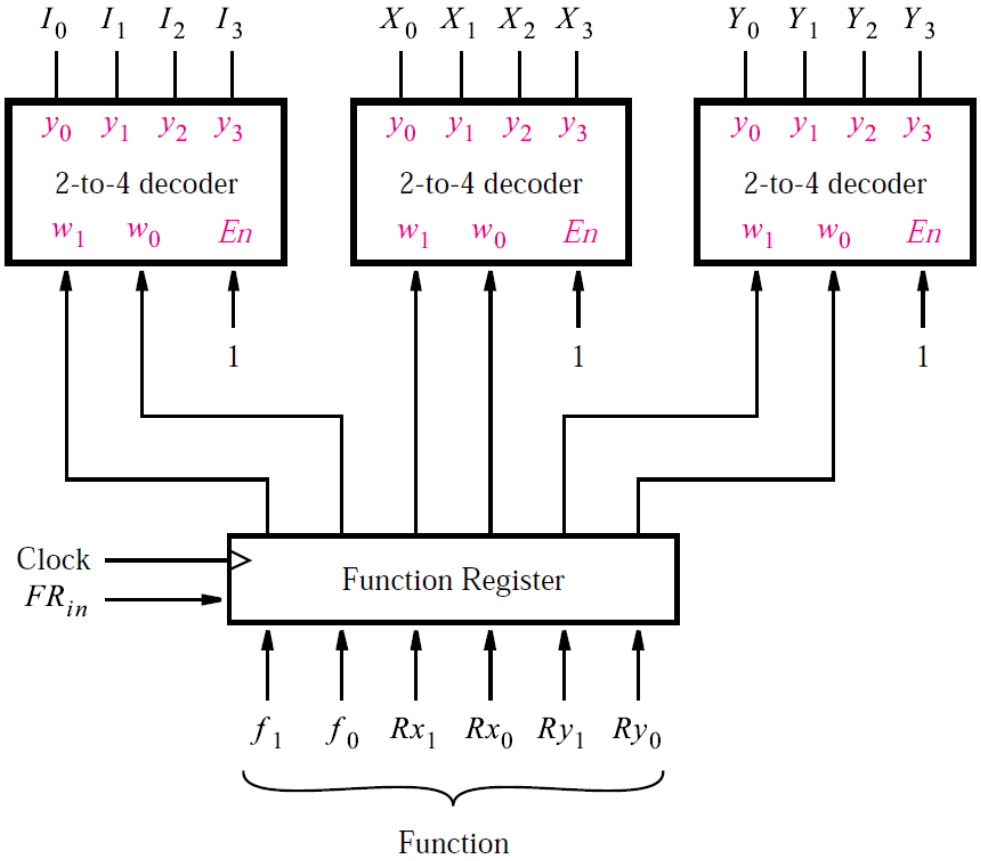
(T_o is initial state)

a two-bit up-counter with synchronous reset



Control Signals

- In each of steps T_0 to T_3, various control signal values have to be generated by the control circuit, depending on the operation being performed.
- The operation is specified with six bits, which form the Function input. The Function inputs are stored in a six-bit Function Register when the FRin signal is asserted.
- To represent Load, Move, Add, and Sub, we use the codes f_1f_0 = 00, 01,10, and 11, respectively. The inputs Rx_1Rx_0 are a binary number that identifies the Rx operand, while Ry_1Ry_0 identifies the Ry operand.

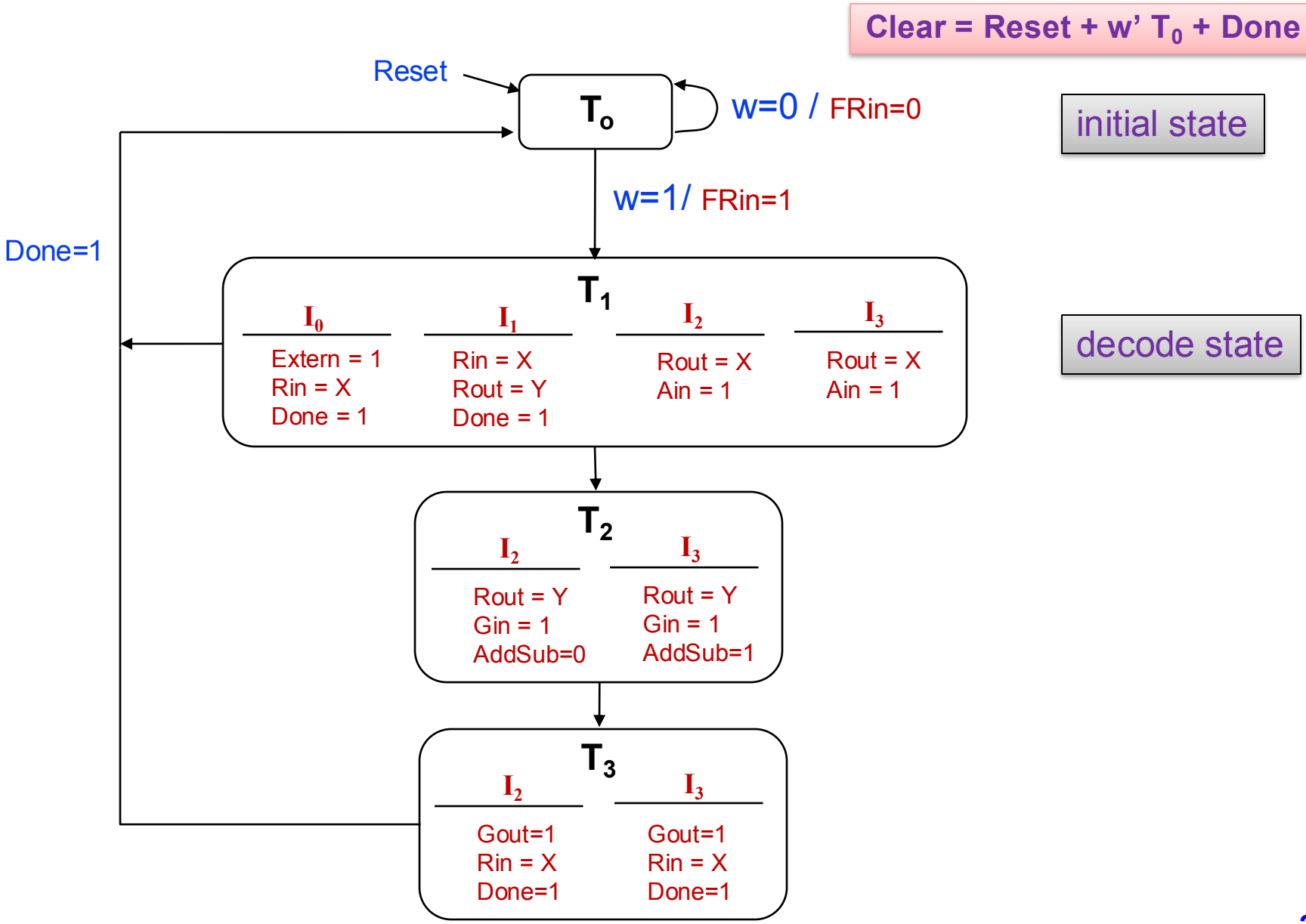


Control signals asserted in each instruction/time step

- The values that have to be generated for each signal are shown in the following table. Each row corresponds to a specific operation, and each column represents one time step.

	T_1	T_2	T_3
(Load): I_0	$Extern, R_{in} = X,$ $Done$		
(Move): I_1	$R_{in} = X, R_{out} = Y,$ $Done$		
(Add): I_2	$R_{out} = X, A_{in}$	$R_{out} = Y, G_{in},$ $AddSub = 0$	$G_{out}, R_{in} = X,$ $Done$
(Sub): I_3	$R_{out} = X, A_{in}$	$R_{out} = Y, G_{in},$ $AddSub = 1$	$G_{out}, R_{in} = X,$ $Done$

State Machine



Subcircuit for the Simple Processor

a two-bit up-counter with synchronous reset

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

ENTITY upcount IS
        PORT (      Clear, Clock      : IN          STD_LOGIC ;
                    Q                 : BUFFER      STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
        upcount: PROCESS ( Clock )
        BEGIN
                IF (Clock'EVENT AND Clock = '1') THEN
                        IF Clear = '1' THEN
                                Q <= "00" ;
                        ELSE
                                Q <= Q + '1' ;
                        END IF ;
                END IF;
        END PROCESS;
END Behavior ;
```

VHDL Code for the Simple Processor (using multiplexers)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
USE work.subccts.all ;

ENTITY proc IS
    PORT (  Data        : IN      STD_LOGIC_VECTOR(7 DOWNTO 0) ;
            Reset, w    : IN      STD_LOGIC ;
            Clock       : IN      STD_LOGIC ;
            F, Rx, Ry   : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            Done        : BUFFER  STD_LOGIC ;
            BusWires    : INOUT   STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END proc ;

ARCHITECTURE Behavior OF proc IS
    SIGNAL X, Y, Rin, Rout : STD_LOGIC_VECTOR(0 TO 3) ;
    SIGNAL Clear, High, AddSub : STD_LOGIC ;
    SIGNAL Extern, Ain, Gin, Gout, FRin : STD_LOGIC ;
    SIGNAL Count, Zero, T, I : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
    SIGNAL R0, R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
    SIGNAL A, Sum, G : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
    SIGNAL Func, FuncReg, Sel : STD_LOGIC_VECTOR(1 TO 6) ;
… con't
```

VHDL Code for the Simple Processor (using multiplexers)

```
BEGIN
    Zero <= "00" ; High <= '1' ;
    Clear <= Reset OR Done OR (NOT w AND NOT T(1) AND NOT T(0)) ;
    counter: upcount PORT MAP (Clear, Clock, Count ) ;
    T <= Count ;
    Func <= F & Rx & Ry ;
    FRin <= w AND NOT T(1) AND NOT T(0) ;
    functionreg: regn GENERIC MAP ( N => 6 )
        PORT MAP ( Func, FRin, Clock, FuncReg ) ;
    I <= FuncReg(1 TO 2) ;
    decX: dec2to4 PORT MAP ( FuncReg(3 TO 4), High, X ) ;
    decY: dec2to4 PORT MAP ( FuncReg(5 TO 6), High, Y ) ;
    PROCESS ( T, I, X, Y )
    BEGIN
        Extern <= '0' ; Done <= '0' ; Ain <= '0' ; Gin <= '0' ;
        Gout <= '0' ; AddSub <= '0' ; Rin <= "0000" ; Rout <= "0000" ;
        CASE T IS
            WHEN "00" => -- no signals asserted in time step T0
            WHEN "01" => -- define signals asserted in time step T1
                CASE I IS
                    WHEN "00" => -- Load
                        Extern <= '1' ; Rin <= X ; Done <= '1' ;

    … con't
```

control signals

VHDL Code for the Simple Processor (using multiplexers)

```
                WHEN "01" => -- Move
                        Rout <= Y ; Rin <= X ; Done <= '1' ;
                WHEN OTHERS => -- Add, Sub
                        Rout <= X ; Ain <= '1' ;
        END CASE ;

    WHEN "10" => -- define signals asserted in time step T2
        CASE I IS
                WHEN "10" => -- Add
                        Rout <= Y ; Gin <= '1' ;
                WHEN "11" => -- Sub
                        Rout <= Y ; AddSub <= '1' ; Gin <= '1' ;
                WHEN OTHERS =>      -- Load, Move
        END CASE ;
    WHEN OTHERS => -- define signals asserted in time step T3
        CASE I IS
                WHEN "00" => -- Load
                WHEN "01" => -- Move
                WHEN OTHERS => -- Add, Sub
                        Gout <= '1' ; Rin <= X ; Done <= '1' ;
        END CASE ;
    END CASE ;
END PROCESS ;
```

… con't

VHDL Code for the Simple Processor (using multiplexers)

```
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 ( BusWires, 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 ;
```

By default, Data is assigned to BusWires, even though extern is not set to 1.

Timing simulation of the processor