Digital System Design Based on Data Path and Control Unit Partitioning

Dr. Mahdi Abbasi

Computer Engineering Department

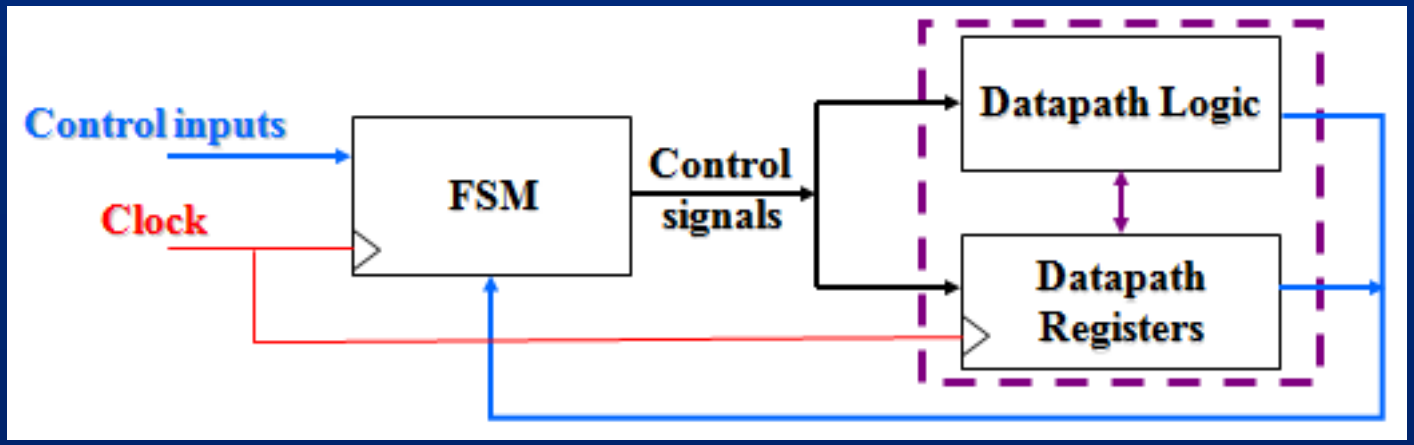
Bu Ali Sina University

Outline

- Data Path & Control Unit Partitioning
- Traffic Light Controller Design
- Algorithmic State Machine (ASM) Chart
- Design Examples

Digital Systems

- **Digital systems**
  - Control-dominated systems :
    being reactive systems responding to external events, such as traffic controllers, elevator controllers, etc.
  - Data-dominated systems :
    requiring high throughput data computation and transport such as telecommunications and signal processing
- **Sequential machines are commonly partitioned into data path units and control units**

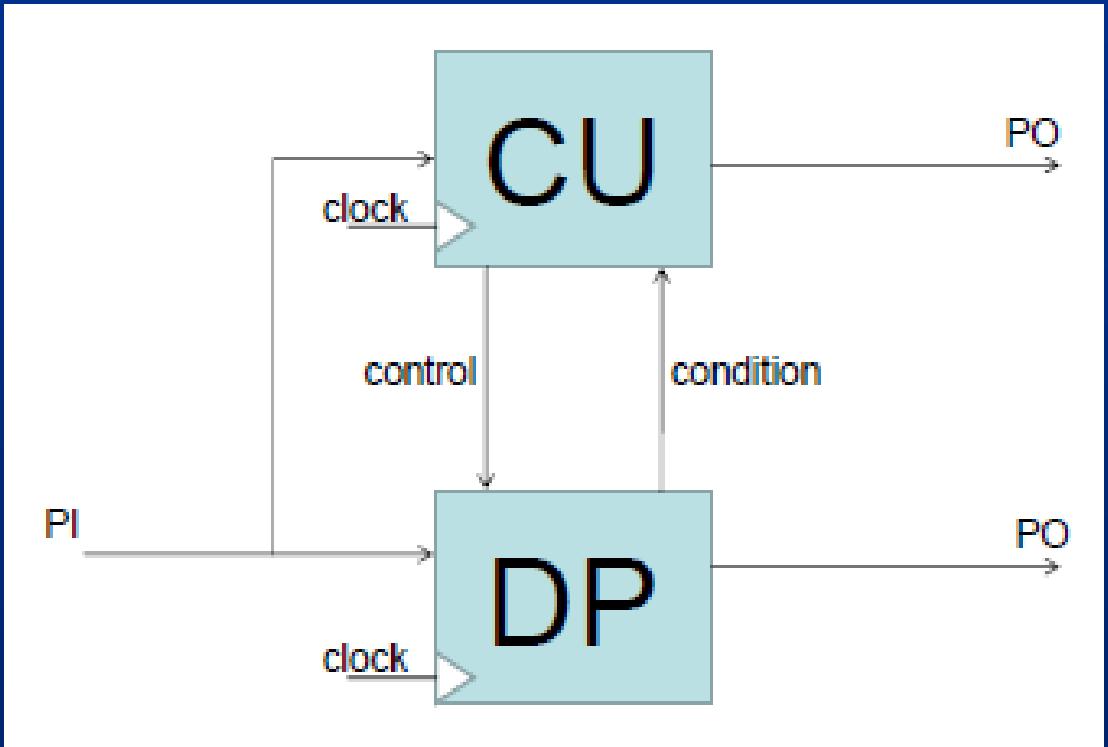


Data Path & Control Unit Partitioning

- **A common design practice decomposes the system in two parts:**
  - A Data Path (DP): a collection of interconnected modules that perform all the relevant computation on the data: it can use both combinational and sequential components
  - A Control Unit (CU) that coordinates the behavior of the Data Path by issuing appropriate control signals that guarantee the correct sequence of operations: it is typically designed as a single or cooperating FSMs
- **DP and CU communicate through 2 types of signals:**
  - Control signals are output of the CU to the DP and correctly synchronize the operations
  - Condition signals (or flags) are sent from the DP to the CU to indicate certain data dependent conditions (that could influence future behavior)

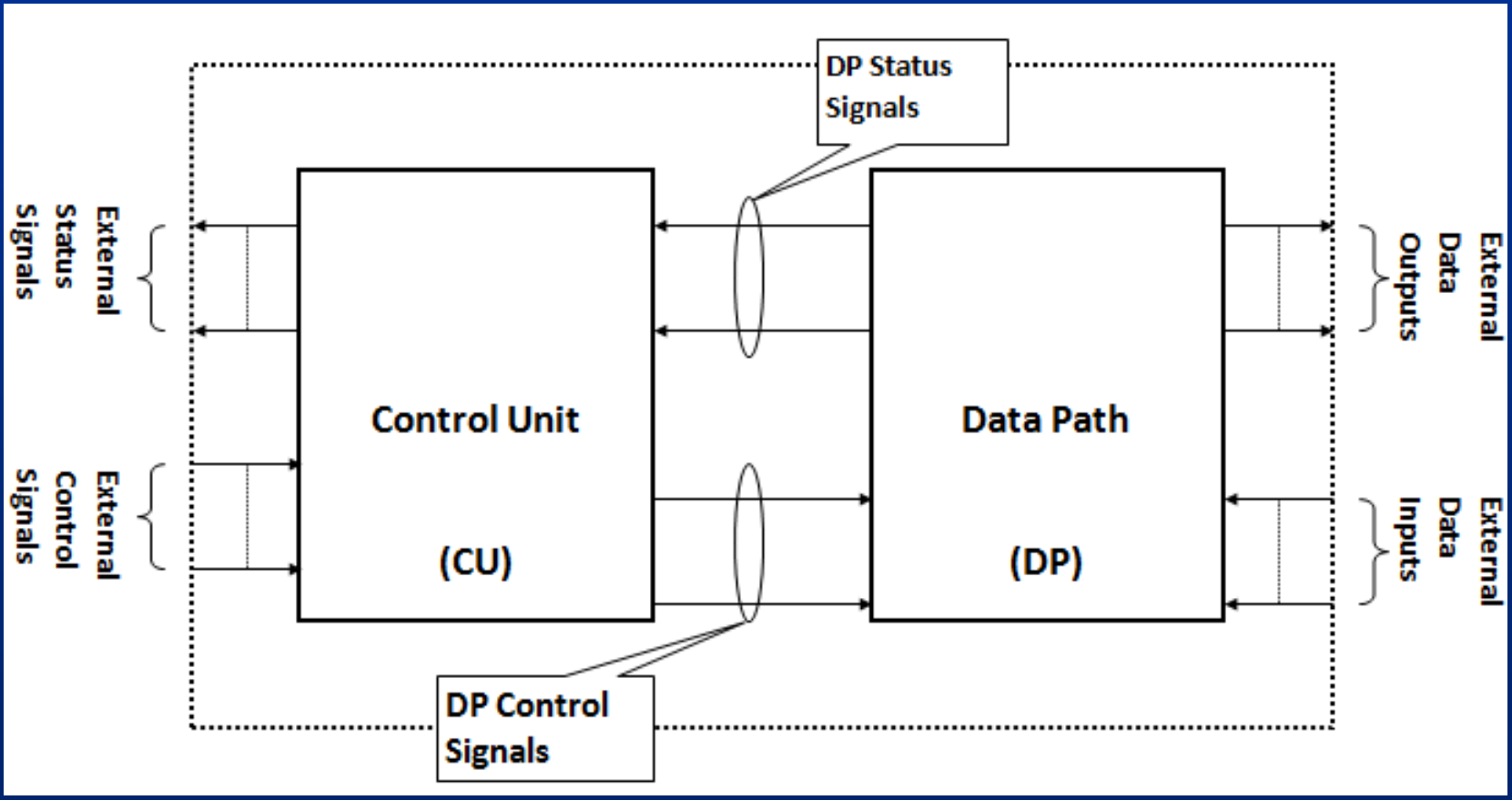
Data Path & Control Unit Partitioning

- Both DP and CU might receive the system's inputs (Primary inputs) and generate its outputs (Primary outputs).



Data Path & Control Unit Partitioning

- The general structure of a digital system that performs a specific task(s) is as follows:



Data Path & Control Unit Partitioning

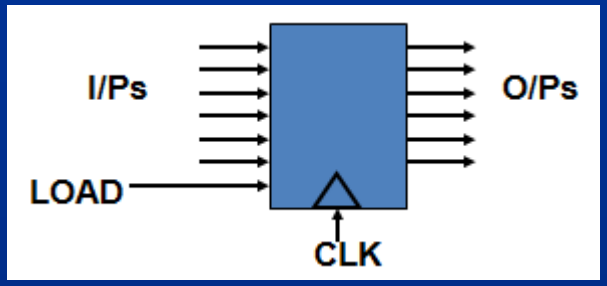
- **External Control Signals**: Specify the task required from the whole circuit (e.g. calculate the average of some integers)
- **External Status Signals**: Indicate the status of the whole circuit (e.g. finished processing, error or overflow ...etc.)
- **External Data Inputs/Outputs**: Data going into the circuit or out of it (e.g. the integers to be averaged and their average)
- **DP Control Signals**: Signals generated by the CU to control different blocks in the DP (e.g. Shift Registers, Counters, MUXs ...etc.)
- **DP Status Signals**: Signals that indicate the status of some blocks in the DP (e.g. when a counter reaches 7 or when an adder produces a carry or an overflow, or when the sign bit of the result is negative ...etc.)

Data Path Design

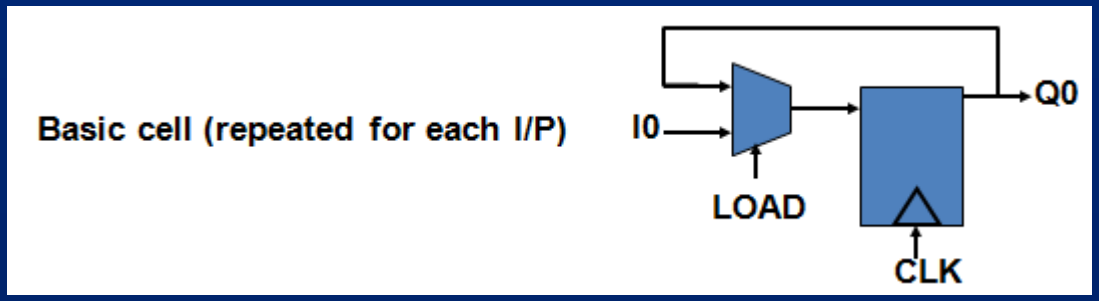
- **The data path contains blocks that only deal with data; they do not provide control to any other blocks and need to be controlled (by the CU).**
- **Data Path blocks can be viewed as the workers that perform certain tasks (on the data) who need to be managed by someone (in this case the CU is the manager that tells every 'worker' in the Data Path what to do).**
- **Examples of Data Path blocks:**
  - Registers, Counters, Multiplexors, Decoders, Logic Circuits (AND, OR, etc)
  - Arithmetic Circuits:
    - Adders, Subtractors, Comparators, Multipliers, Square root, etc.

Registers

- Parallel load registers to read data in parallel

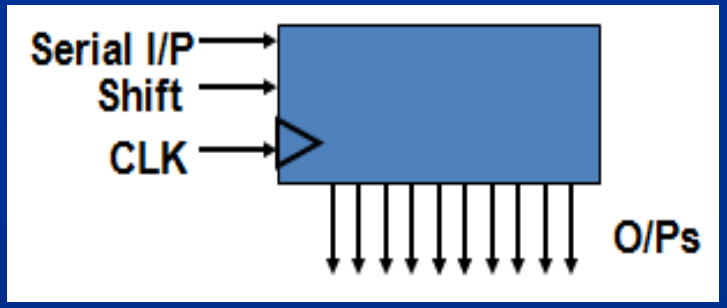


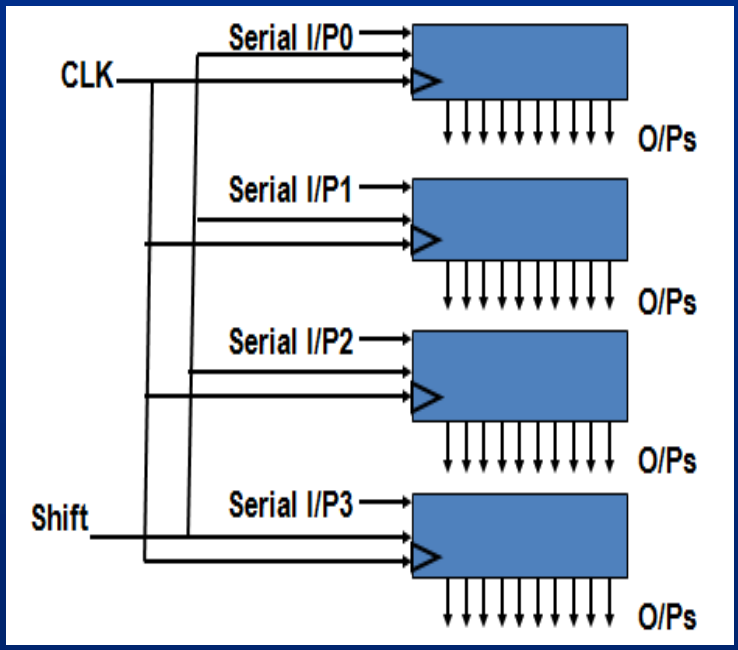
- Load is a synchronous control to control reading the data. When LOAD is inactive, the register keeps the data as is
- Used when we want to read data as fast as possible (in one clock cycle)
- Implemented using D-FFs and MUXs



Shift Registers

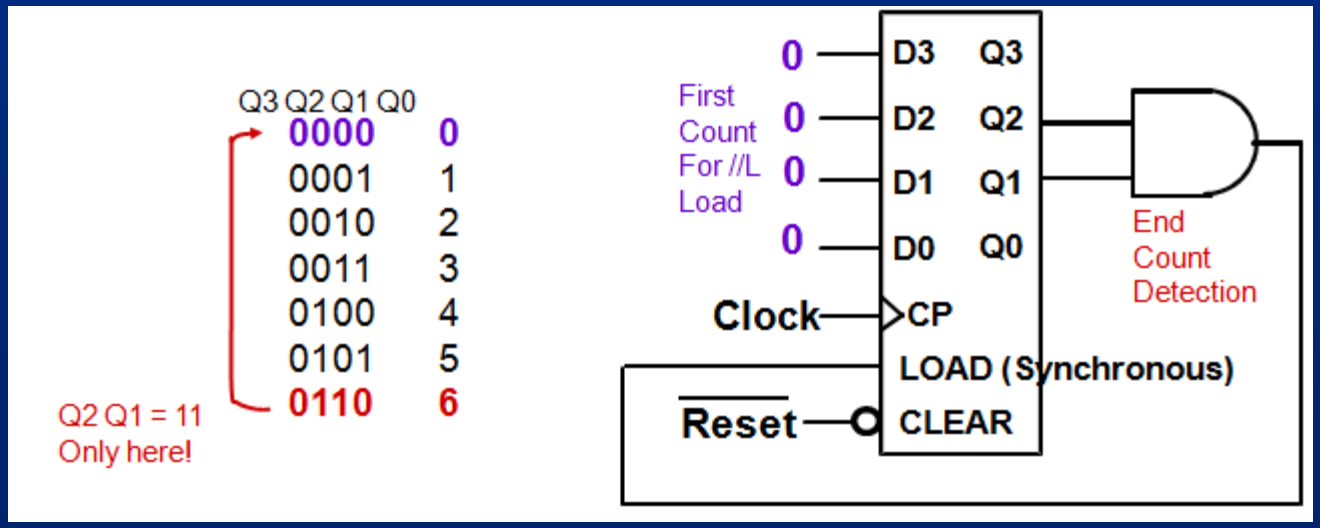
- Shift Registers to read data **serially** one bit at a time
- **Shift** is a synchronous control of shifting (register keeps data as is when **Shift** is not active)
- Digit serial registers that read data serially one digit at a time, where the digit size could be anything (e.g. 4-bits, 8-bits, 16-bits …etc.) → multiple of shift registers in parallel



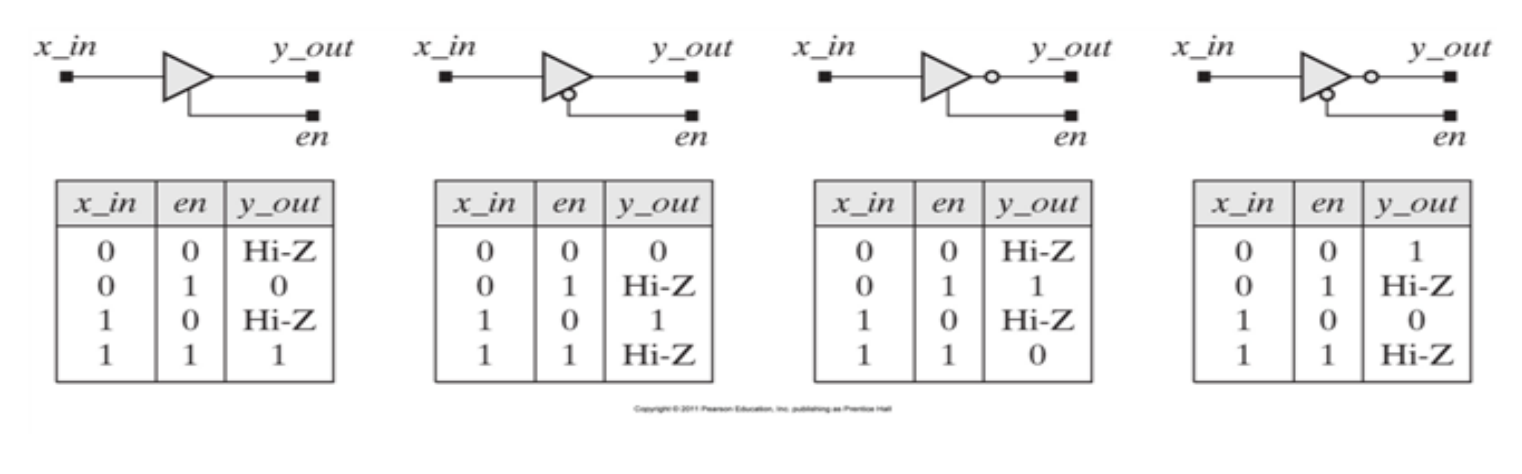


Modulo N (i.e. divide by N) Counters
N counting states: 0, 1, 2, …, (N-1)

- **The following techniques use an n-bit (2^n >= N) binary counter with synchronous clear or parallel load:**
  - Detect terminal count (N – 1) and use to synchronously Clear the counter to 0 (first count) on next clock pulse
  - Detect terminal count (N – 1) to synchronously Load in the value 0 (first count) on next clock pulse
- **Modulo 7 (0,1,…,6):**



Three-State Devices



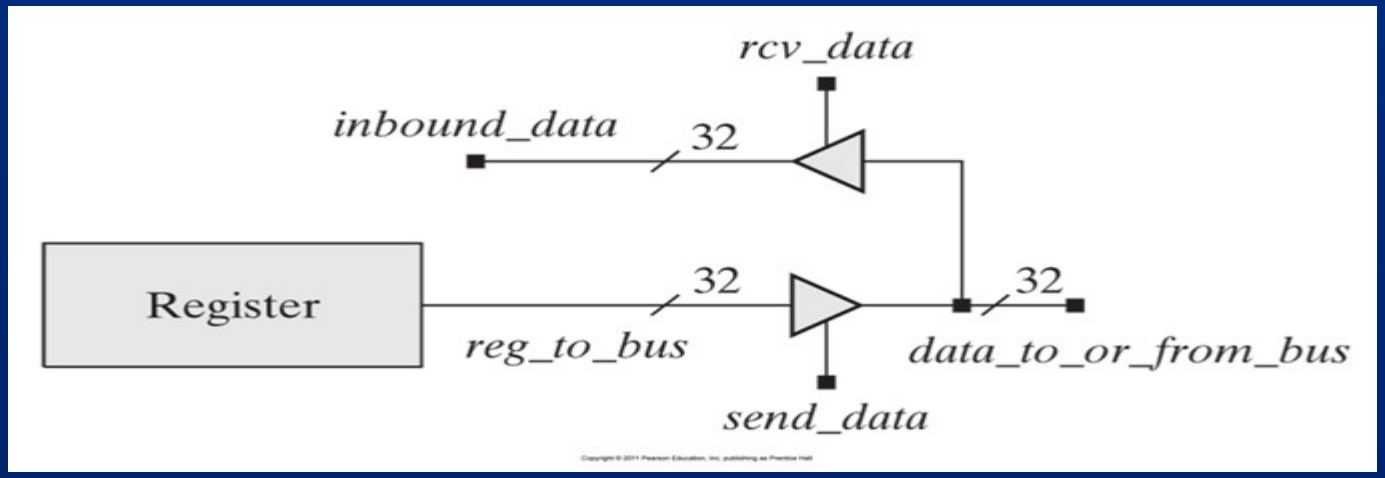
x_in	en	y_out
0	0	Hi-Z
0	1	0
1	0	Hi-Z
1	1	1

x_in	en	y_out
0	0	0
0	1	Hi-Z
1	0	1
1	1	Hi-Z

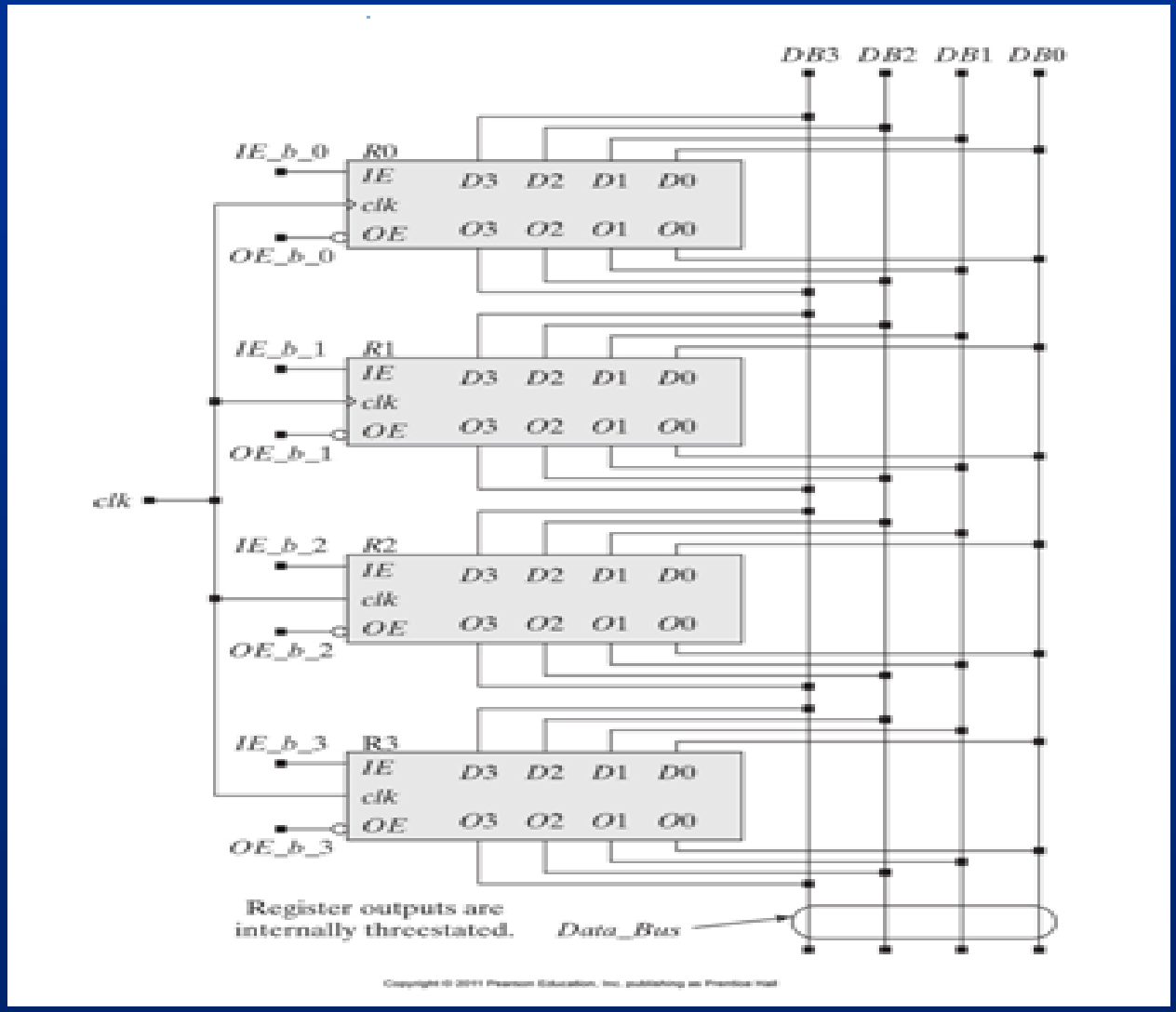
x_in	en	y_out
0	0	Hi-Z
0	1	1
1	0	Hi-Z
1	1	0

x_in	en	y_out
0	0	1
0	1	Hi-Z
1	0	0
1	1	Hi-Z

- Bus isolation with three-state devices



A Register Bank with a 4-bit Data Bus

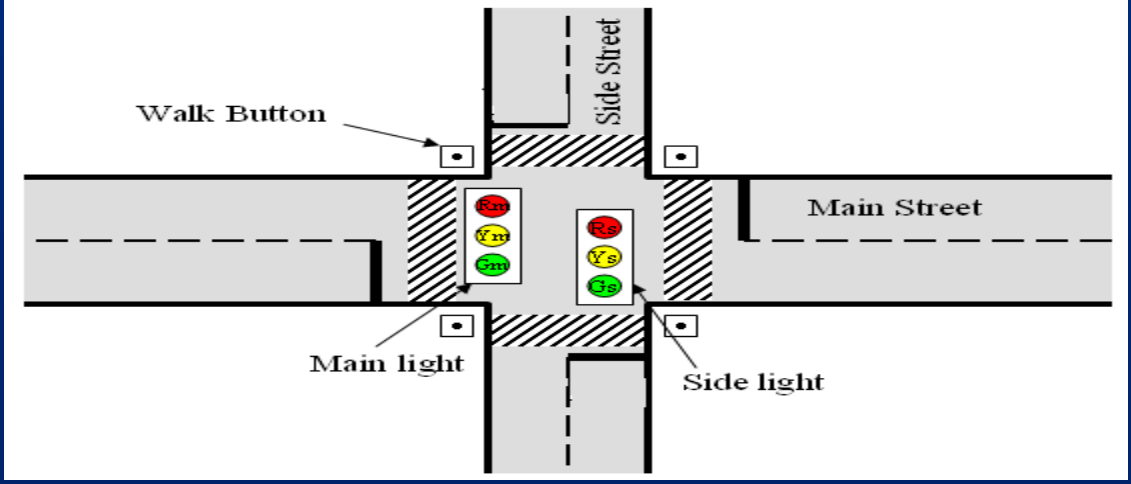


Design Steps

- Identify all inputs and outputs for the whole circuit. Identify, separately, data inputs/outputs and control inputs/outputs (external status).
- Identify the required Data Path blocks and their control signals and design them.
- Identify the input and output signals to the Data Path and Control unit.
- Design the control Unit (start by obtaining the state diagram, then the next state and output equations and finally the logic implementation) and connect it to the DP.

Example: Traffic Light Controller

- **Design a digital system that controls the traffic lights at an intersection:**
  - It receives inputs from all four corners indicating pedestrians that want to cross
  - In absence of crossing requests it should allow each direction 30 seconds of green light, followed by 5 seconds of yellow light while the other traffic light will be red light (i.e. for 35 seconds)
  - In presence of crossing requests at or after 15 seconds, immediately proceed with yellow
  - It is assumed that the clock frequency of the system is 1KHz



Example: Traffic Light Controller

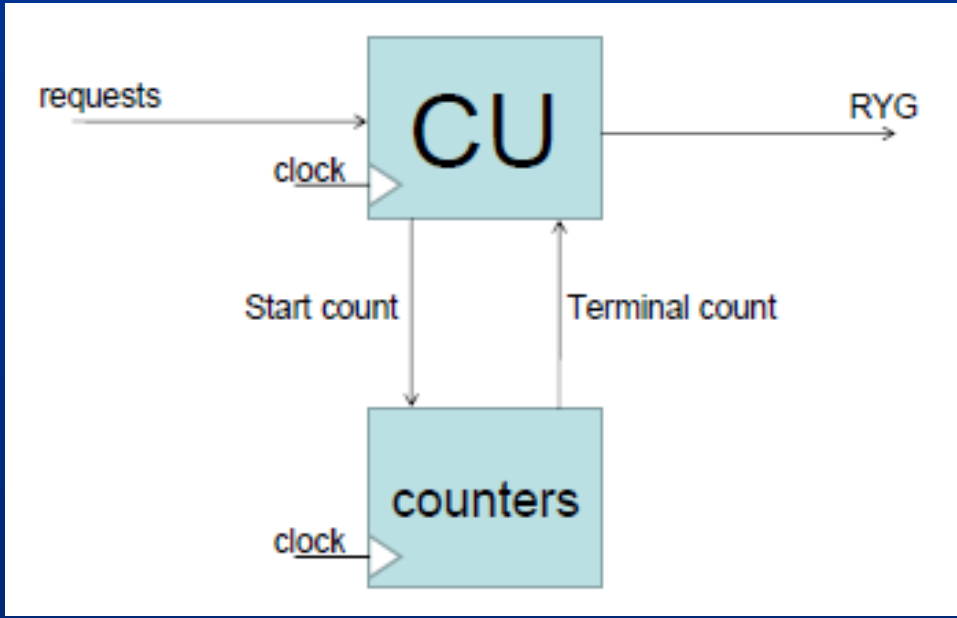
- **Is it possible to design the system as a single combinational circuit or an FSM?**
- **Because of the time delay, a combinational circuit will not work (we need to make sure that we waited 30 seconds, and then 5 seconds etc.)**
- **However, a sequential circuit is perfectly capable to deal with the problem.**
- **So why bother designing a more complex circuit?**
  - Let's look better at the specification

Example: Traffic Light Controller

- **If we were to design the system as a single FSM, we will have to include in it all the possible conditions.**
- **In particular, we will have to deal with the time delays by introducing delay states**
  - How many? With a 1KHz clock, we need 1000 clock cycles to measure one second, 30,000 to measure 30 seconds.
  - So, we will need more than 30,000 states!
  - The FSM design will be boring to say the least….
- **How can we handle the state explosion problem?**
- **We could use counters to measure the time intervals, and a control unit to coordinate their behavior.**

Example: Traffic Light Controller

- **DP/CU for the TLC:**

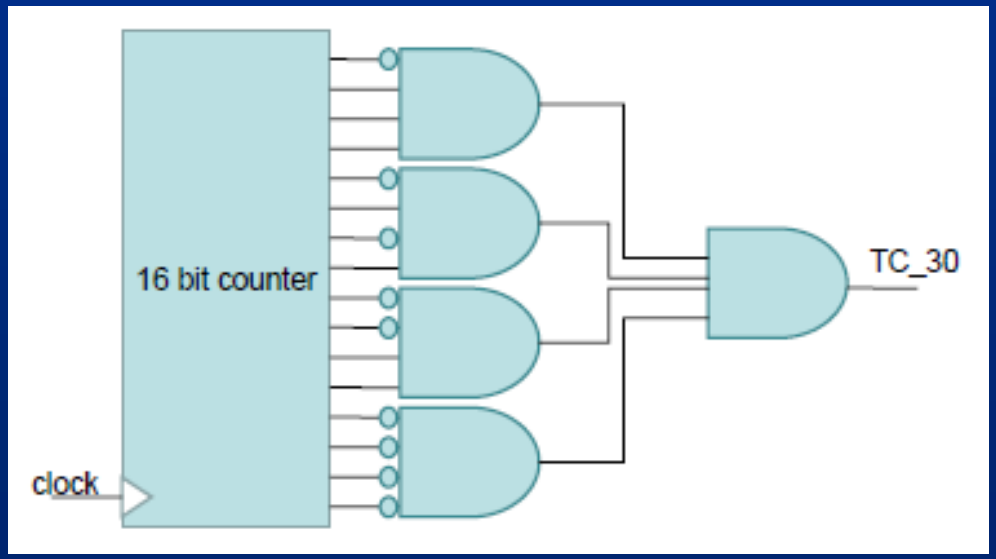


Example: Traffic Light Controller

- **DP design: we need to be able to count:**
  - 30 seconds (30,000 clock cycles) for the “regular green”
  - 15 seconds (15,000 clock cycles) for the “reduced green”
  - 5 seconds (5,000 clock cycles) for the “yellow light”
- **Typical counters have:**
  - A reset/clear input (asynchronous): that sets the count to “’0”
  - A count enable input: if 1 it allows the counter to increment
  - A multiple bit count output: reports the current state of the count
  - A single bit terminal count: indicates that a total number of clock cycles are elapsed and the counter is back to 0

Example: Traffic Light Controller

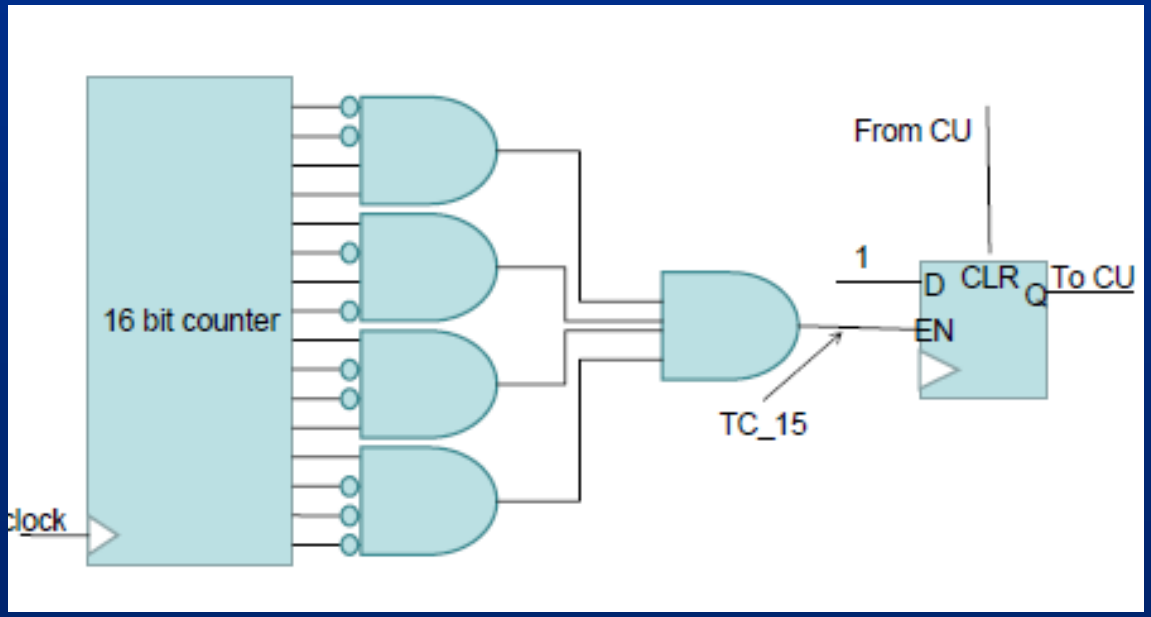
- In order to be able to count 30,000, we need a "power of 2" of at least 32,768 (=2^{15}).
- **30000=$(111\ 0101\ 0011\ 0000)_2$**. If the counter output bits are "1110101001100000" the terminal count is 1



Example: Traffic Light Controller

- Requests reduce the time to 15 seconds, but if 15 seconds have already elapsed you should know it

  $150000 = (11\ 1010\ 1001\ 1000)_2$



Example: Traffic Light Controller

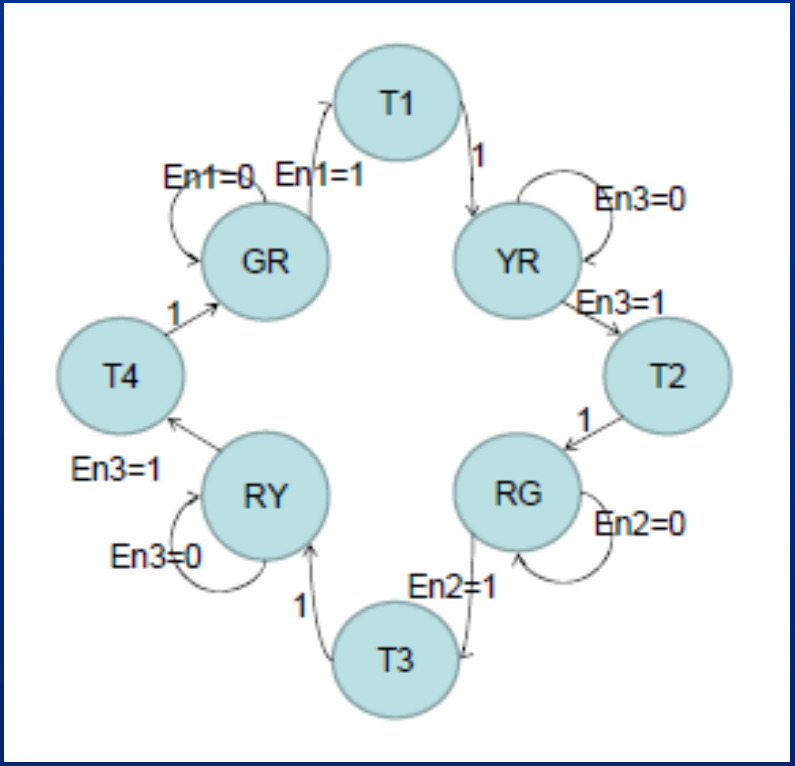
- **We can have three separate blocks in the DP (for the 30 sec, >=15 sec and 5 sec):**
  - DP composed of 3 16-bit counters, and a bunch of AND and NOT gates
- **However, we can do better:**
  - If you look at the specification, the intervals of 5, 15 or 30 seconds are either used in separate times (5 is only used during the “yellow” phase) or they can use the same starting point (the 30 and 15 – they actually have to use the same starting point)
  - Use just one counter, with three circuits for 5, 30 and >=15 sec.

Example: Traffic Light Controller

- **Now, we have a complete DP. Its interface with the CU:**
  - One counter reset input (asynchronous reset)
  - One FF reset input (asynchronous reset)
  - Three outputs (TC_30, TC_5, GE_15)
  - In this case, DP does not use PI nor produces PO (but it is just a special case)
- **CU design**
  - Once the DP is finished, the CU design can proceed, as in a usual FSM design
  - In this case, the CU has to:
    - Provide regular switching between green, yellow and red lights
    - Observe the request inputs and use them in combination with GE_15 to shorten green lights

Example: Traffic Light Controller

- **Possible Moore machine implementation**
  - EN1=TC_30 + GE_15 . cross1
  - EN2=TC_30 + GE_15 . cross2
  - EN3=TC_5
- **CU outputs (if not shown =0):**
  - State GR: G1=1, R2=1
  - State T1: Y1=1, R2=1
    - CNT_RES=1
  - State YR: Y1=1, R2=1
  - State T2: R1=1, G2=1
    - CNT_RES=1, FF_RES=1
  - State RG: R1=1, G2=1
  - State T3: R1=1, Y2=1
    - CNT_RES=1
  - State RY: R1=1, Y2=1
  - State T4: G1=1, R2=1
    - CNT_RES=1, FF_RES=1



Example: Traffic Light Controller

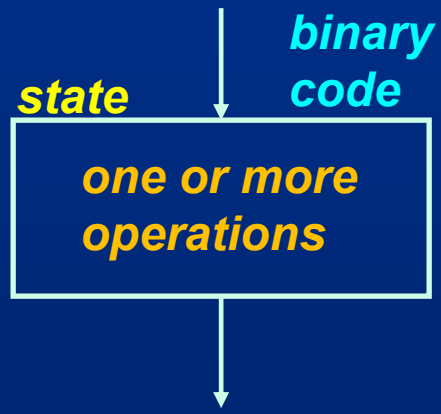
- **One more issue to verify:**
  - Are we actually keeping the lights 5, 15 or 30 seconds?
  - The counters act exactly, but the entire system has a certain delay that we need to take into account:
    - The counters start counting only the clock cycle after the reset (in this case, when the CU enters the states RG, GR YR and RY)
    - When they are done counting, the state changes the following clock tick, and the outputs with them - MOORE machine
  - In conclusion, the lights stay on for 1 more clock cycle: we need to take that into account by reducing the terminal count to 29,999, 4,999 and 14,999…

Algorithmic State Machine (ASM) Chart

- **Algorithmic State Machine (ASM) Chart is a high-level flowchart-like notation to specify the hardware algorithms in digital systems.**
- **Major differences from flowcharts are:**
  - uses 3 types of boxes: state box (similar to operation box), decision box and conditional box
  - contains exact (or precise) timing information; flowcharts impose a relative timing order for the operations.
- **From the ASM chart it is possible to obtain**
  - the control
  - the architecture (data processor)

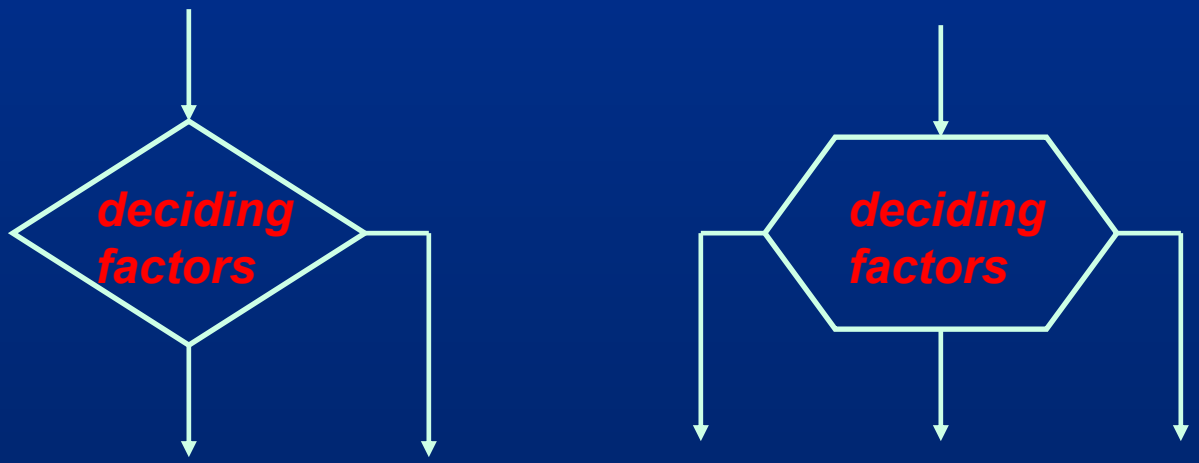
Components of ASM Charts

- The **state box** is rectangular in shape. It has at most one entry point and one exit point and is used to specify one or more operations which could be simultaneously completed in one clock cycle.



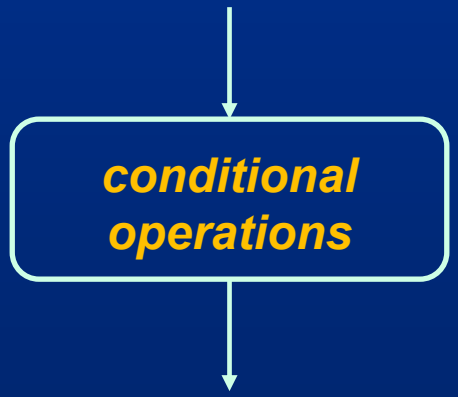
Components of ASM Charts

- The **decision box** is diamond in shape. It has one entry point but multiple exit points and is used to specify a number of alternative paths that can be followed.



Components of ASM Charts

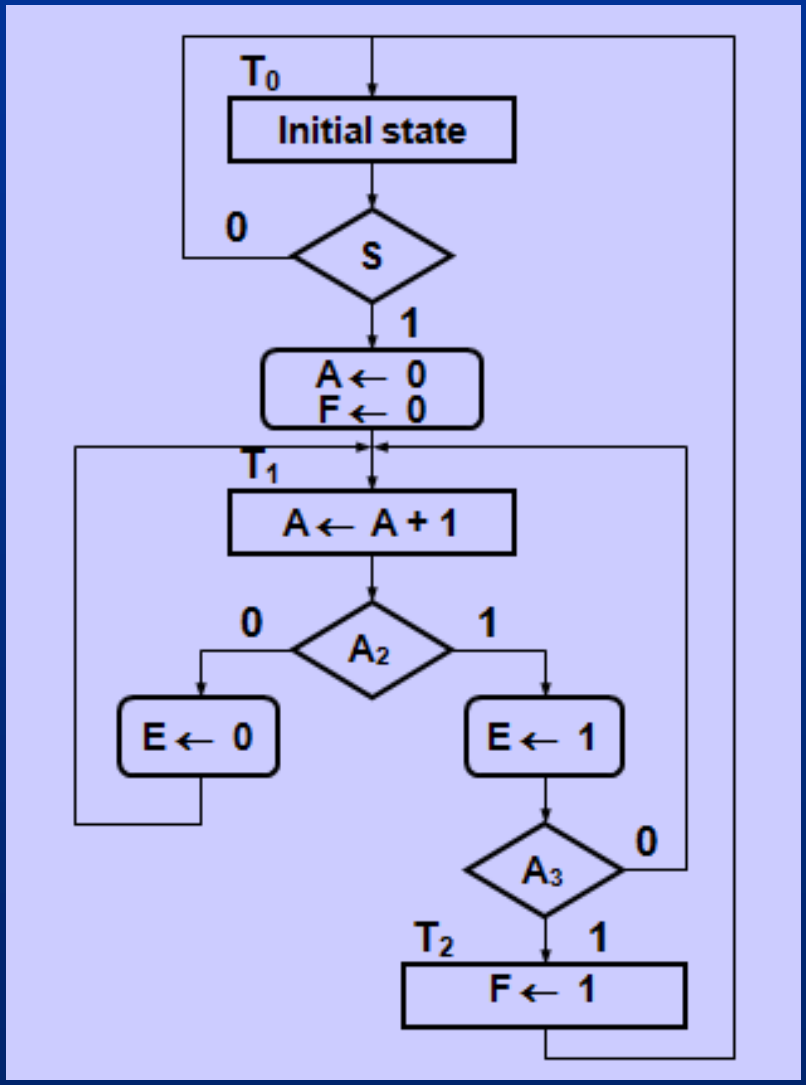
- The **conditional box** is represented by a rectangle with rounded corners. It always follows a decision box and contains one or more *conditional operations* that are only invoked when the path containing the conditional box is selected by the decision box.



ASM Charts: An Example

- A is a register;
- A_i stands for i^{th} bit of the A register.

  $A = A_4A_3A_2A_1$
- E and F are single-bit flip-flops.



Register Operations

- **Registers are present in the data processor for storing and processing data. Flip-flops (1-bit registers) and memories (set of registers) are also considered as registers.**
- **The register operations are specified in either the state and/or conditional boxes, and are written in the form:**

destination register ← *function*(*other registers*)

where the LHS contains a destination register (or part of one) and the RHS is some function over one or more of the available registers.

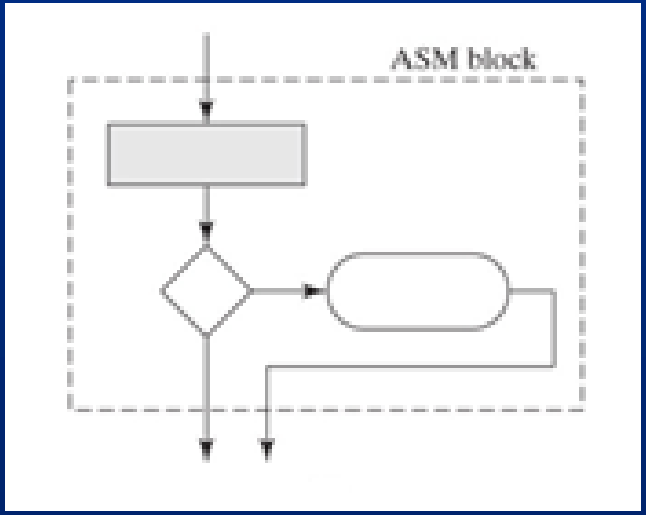
Register Operations

- **Examples of register operations:**

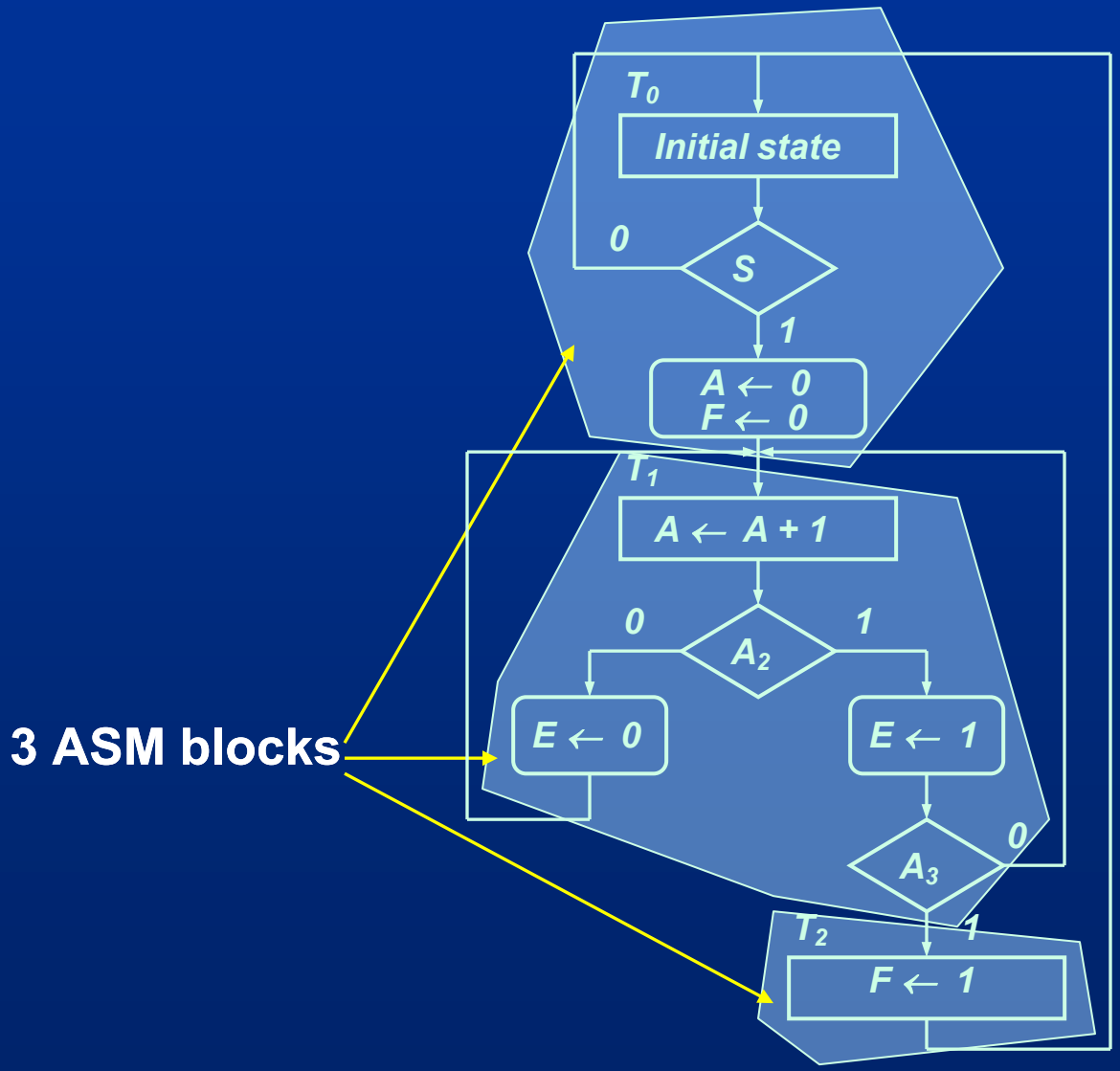
$A \leftarrow B$	Transfer contents of register B into register A.
$A \leftarrow 0$	Clear register A.
$A \leftarrow A - 1$	Decrement register A by 1.

Timing in ASM Charts

- Precise timing is implicitly present in ASM charts.
- Each *state box*, together with its immediately following *decision* and *conditional boxes*, occur within **one clock cycle**.
- A group of boxes which occur within a single clock cycle is called an **ASM block**.



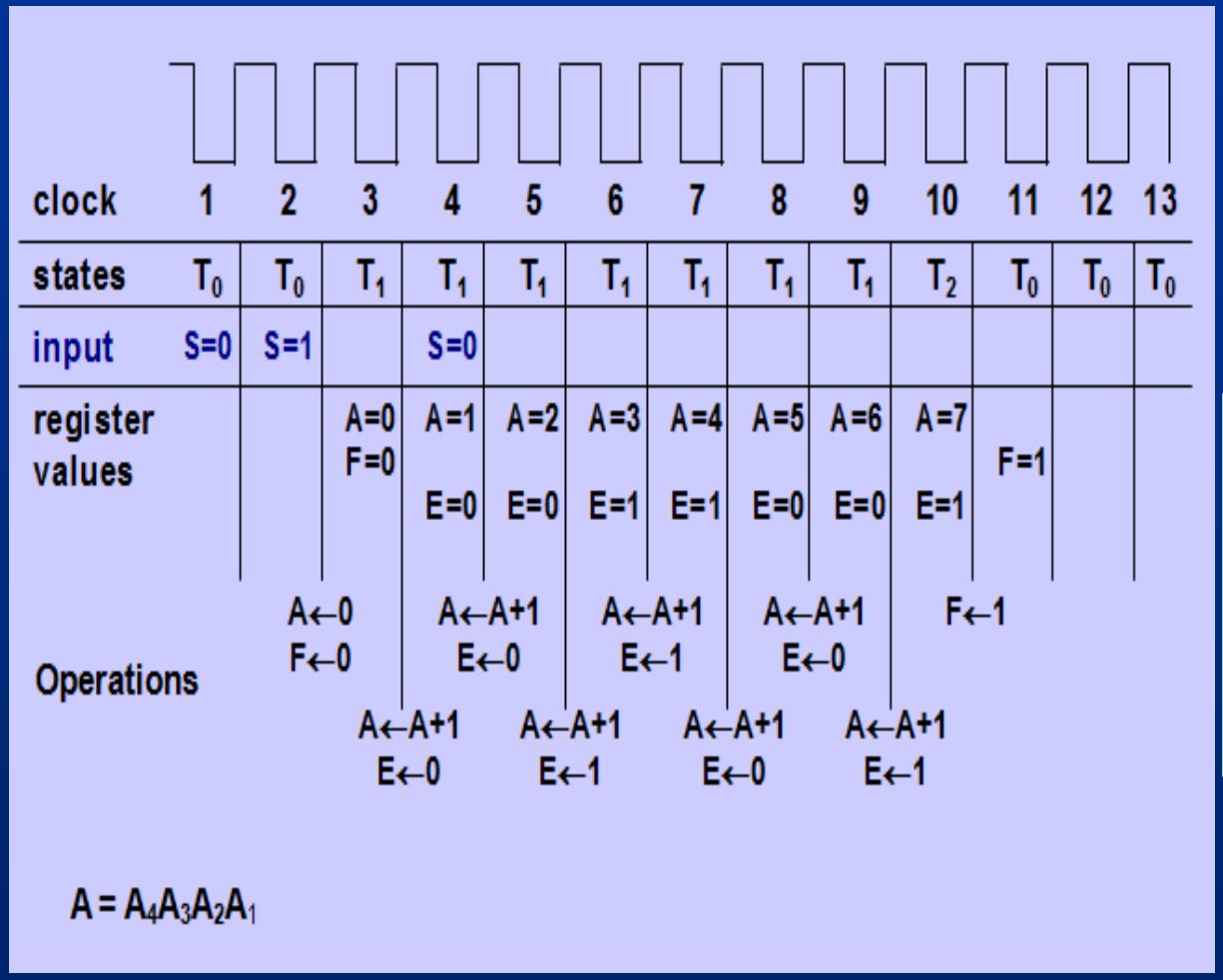
Timing in ASM Charts

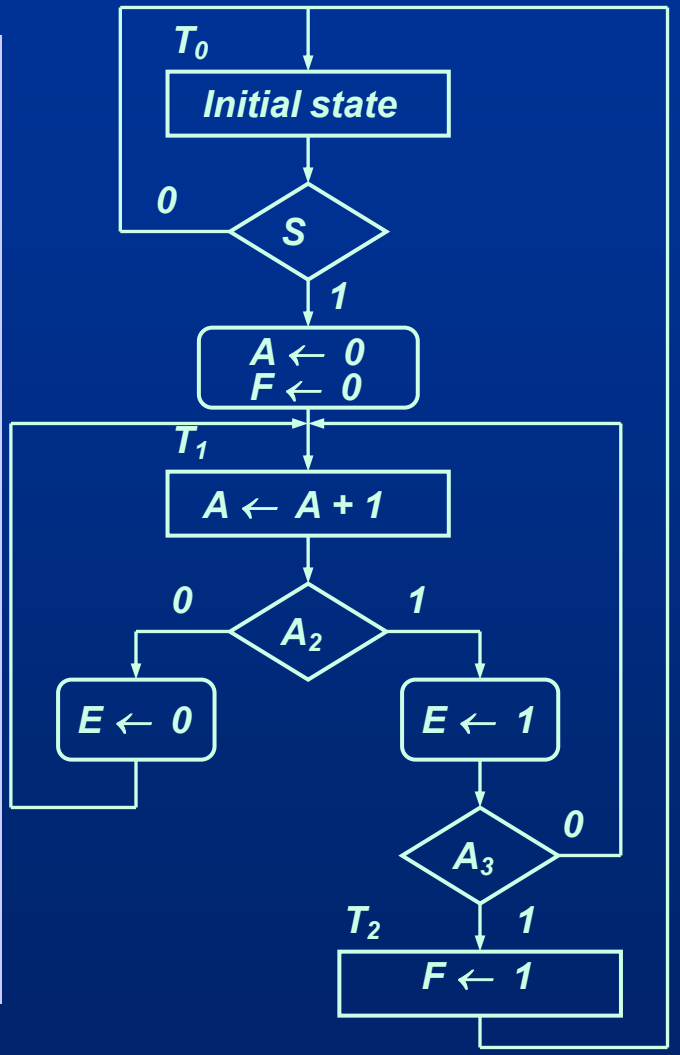


Timing in ASM Charts

- **Operations of ASM can be illustrated through a timing diagram.**
- **Two factors which must be considered are**
  - operations in an ASM block occur at the same time in *one clock cycle*
  - decision boxes are dependent on the status of the *previous clock cycle* (that is, they do not depend on operations of current block)

Timing in ASM Charts





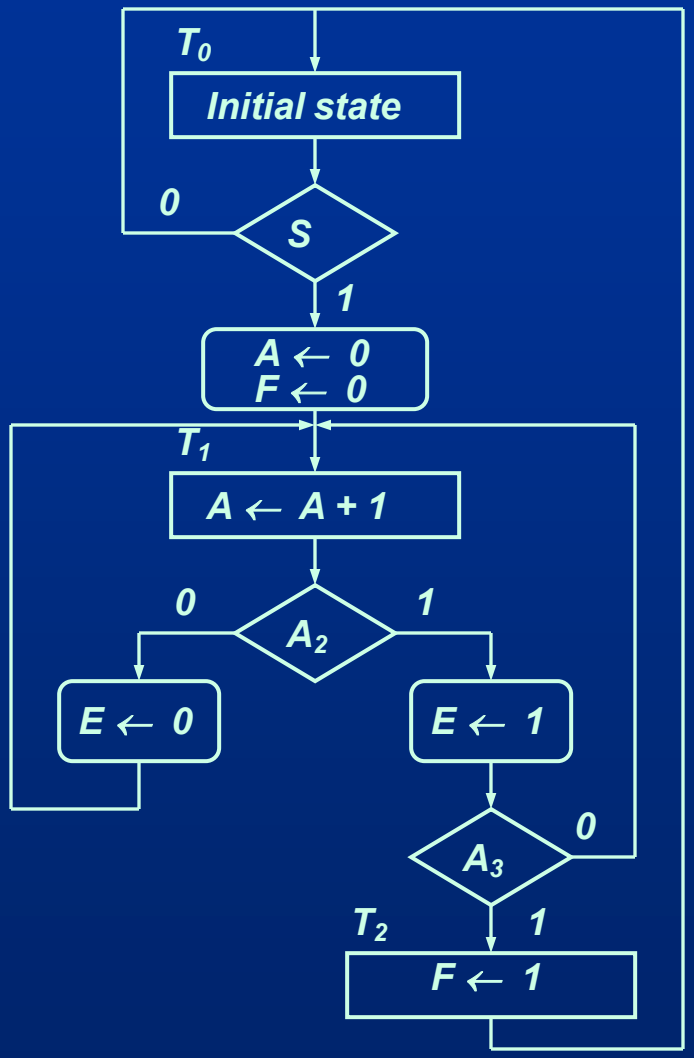
ASM Chart => Digital System

- **ASM chart describes a digital system. From ASM chart, we may obtain:**
  - Controller logic (via State Table/Diagram)
  - Architecture/Data Processor
- **Design of controller is determined from the decision boxes and the required state transitions.**
- **Design requirements of data processor can be obtained from the operations specified with the state and conditional boxes.**

ASM Chart => Controller

- **Procedure:**
  - Step 1: Identify all states and assign suitable codes.
  - Step 2: Formulate state table using
    State from state boxes
    Inputs from decision boxes
    Outputs from operations of state/conditional boxes.
  - Step 3: Obtain state/output equations and draw circuit.

ASM Chart => Controller



Assign codes to states:

$T_0 = 00$

$T_1 = 01$

$T_2 = 11$

Present state		inputs			Next state		outputs		
G_1	G_0	S	A_2	A_3	G_1^+	G_0^+	T_0	T_1	T_2
0	0	0	X	X	0	0	1	0	0
0	0	1	X	X	0	1	1	0	0
0	1	X	0	X	0	1	0	1	0
0	1	X	1	0	0	1	0	1	0
0	1	X	1	1	1	1	0	1	0
1	1	X	X	X	0	0	0	0	1

Inputs from conditions in decision boxes.
Outputs = present state of controller.

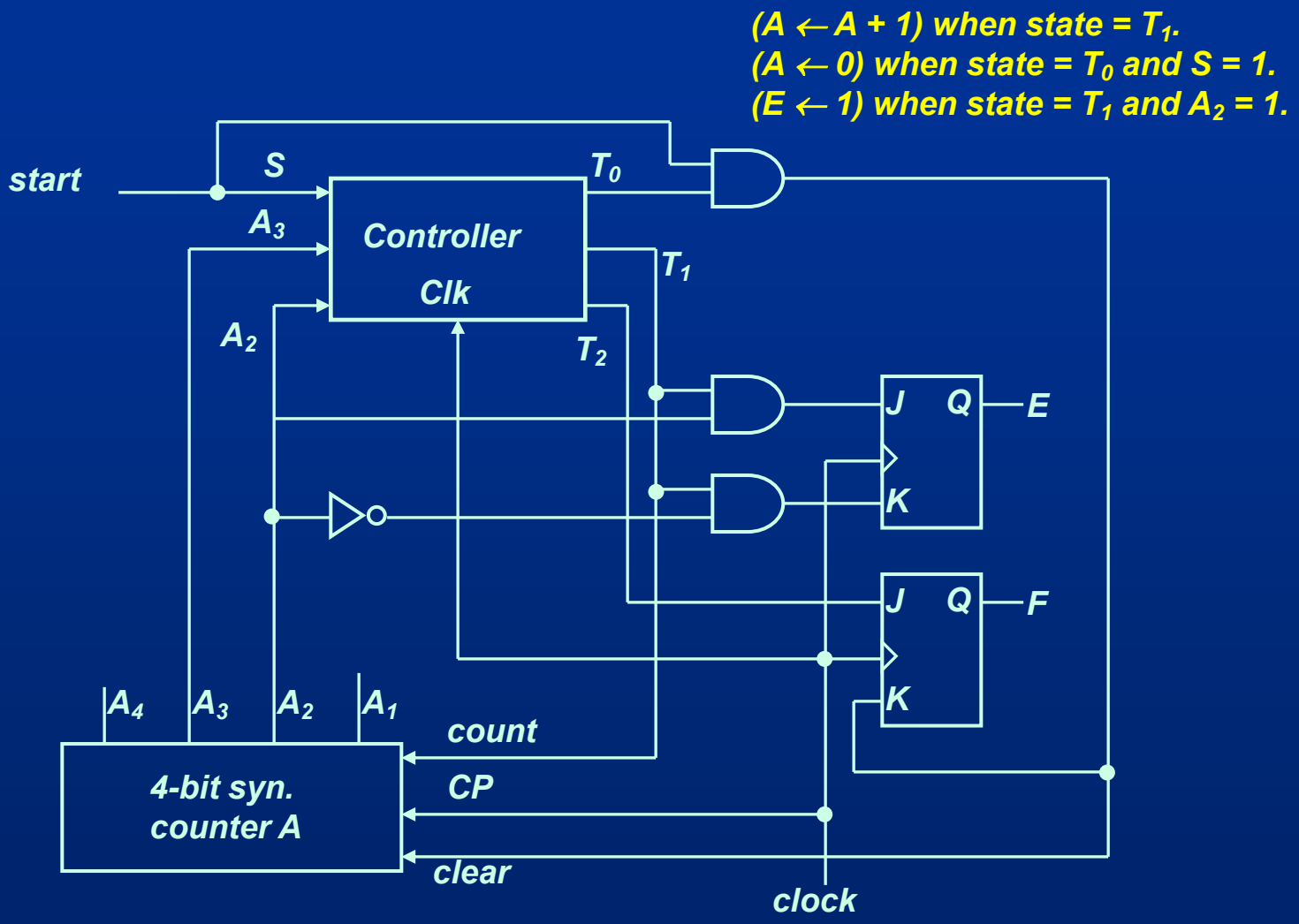
ASM Chart => Architecture/Data Processor

- **Architecture is more difficult to design than controller.**
- **Nevertheless, it can be deduced from the ASM chart. In particular, the operations from the ASM chart determine:**
  - What registers to use
  - How they can be connected
  - What operations to support
  - How these operations are activated.
- **Guidelines:**
  - always use high-level units
  - simplest architecture possible.

ASM Chart => Architecture/Data Processor

- **Various operations are:**
  - Counter incremented ($A \leftarrow A + 1$) when state = T_1.
  - Counter cleared ($A \leftarrow 0$) when state = T_0 and S = 1.
  - E is set ($E \leftarrow 1$) when state = T_1 and $A_2 = 1$.
  - E is cleared ($E \leftarrow 0$) when state = T_1 and $A_2 = 0$.
  - F is set ($F \leftarrow 0$) when state = T_2.
  - F is cleared ($F \leftarrow 0$) when state = T_0 and S = 1.
- **Deduce:**
  - One 4-bit register A (e.g.: 4-bit synchronous counter with clear/increment).
  - Two flip-flops needed for E and F (e.g.: JK flip-flops).

ASM Chart => Architecture/Data Processor

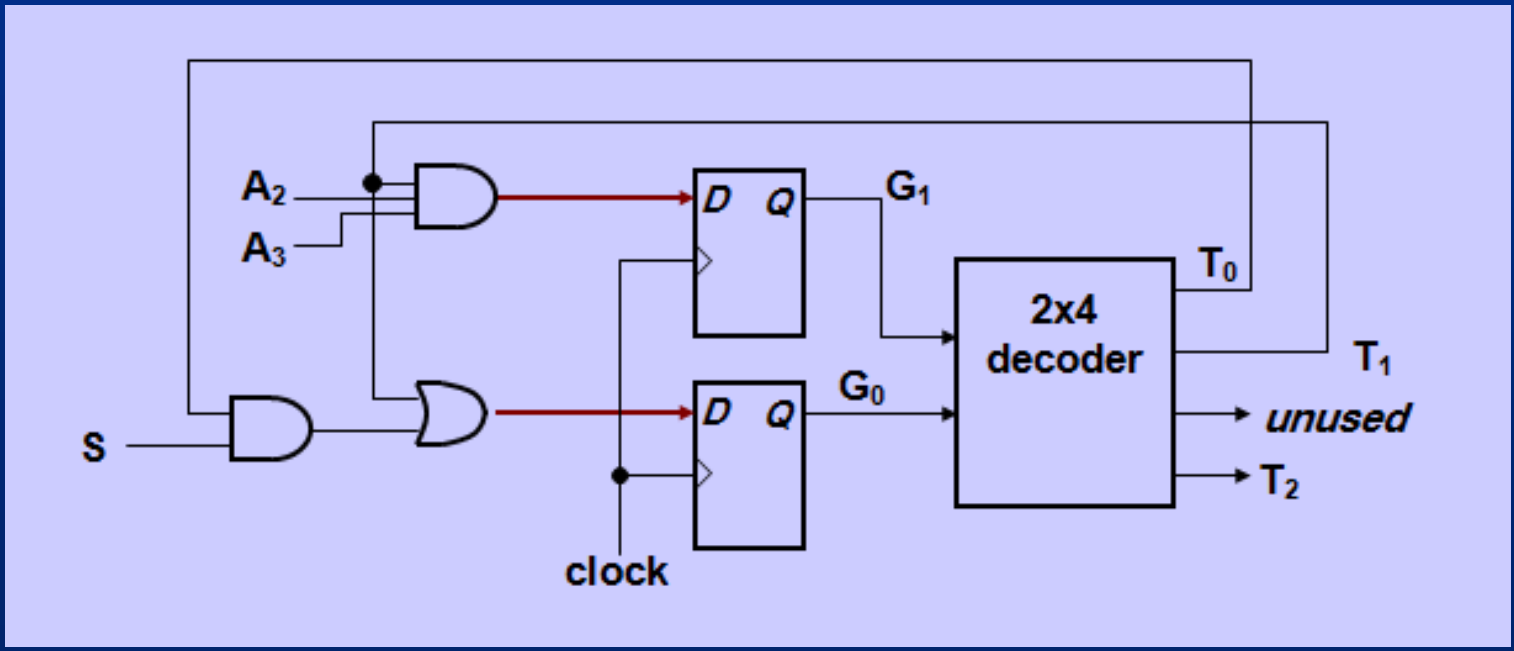


Implementing Controller: Decoder + D Flip-flops

- **Flip-flop input functions:**

$$DG_1 = T_1.A_2.A_3$$
$$DG_0 = T_0.S + T_1$$

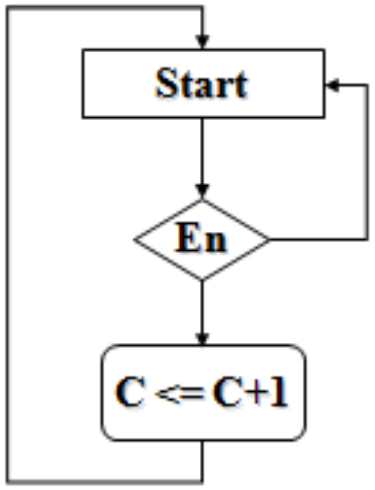
- **Circuit:**



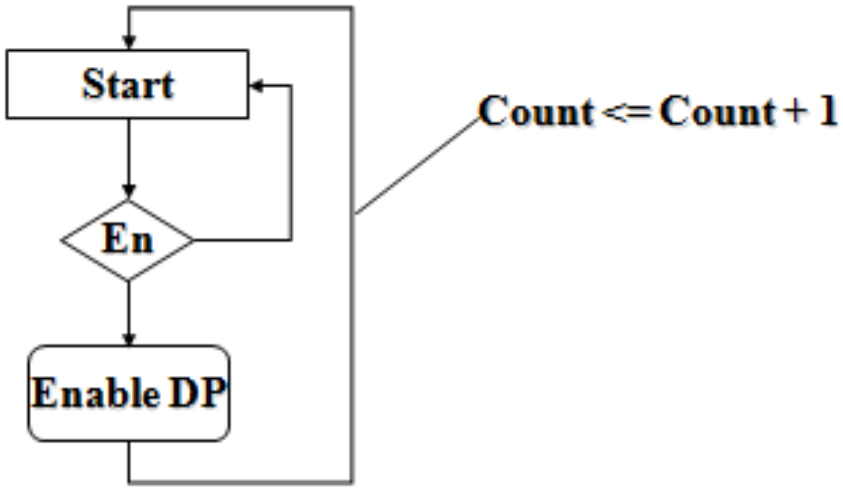
Algorithmic State Machine and DataPath (ASMD) Chart

- ASMD is different from ASM in that each of the transition path of an ASM is annotated with the associated concurrent *register* operations of datapath
- ASM vs. ASMD charts for a counter with enable

ASM chart representation



ASMD chart representation

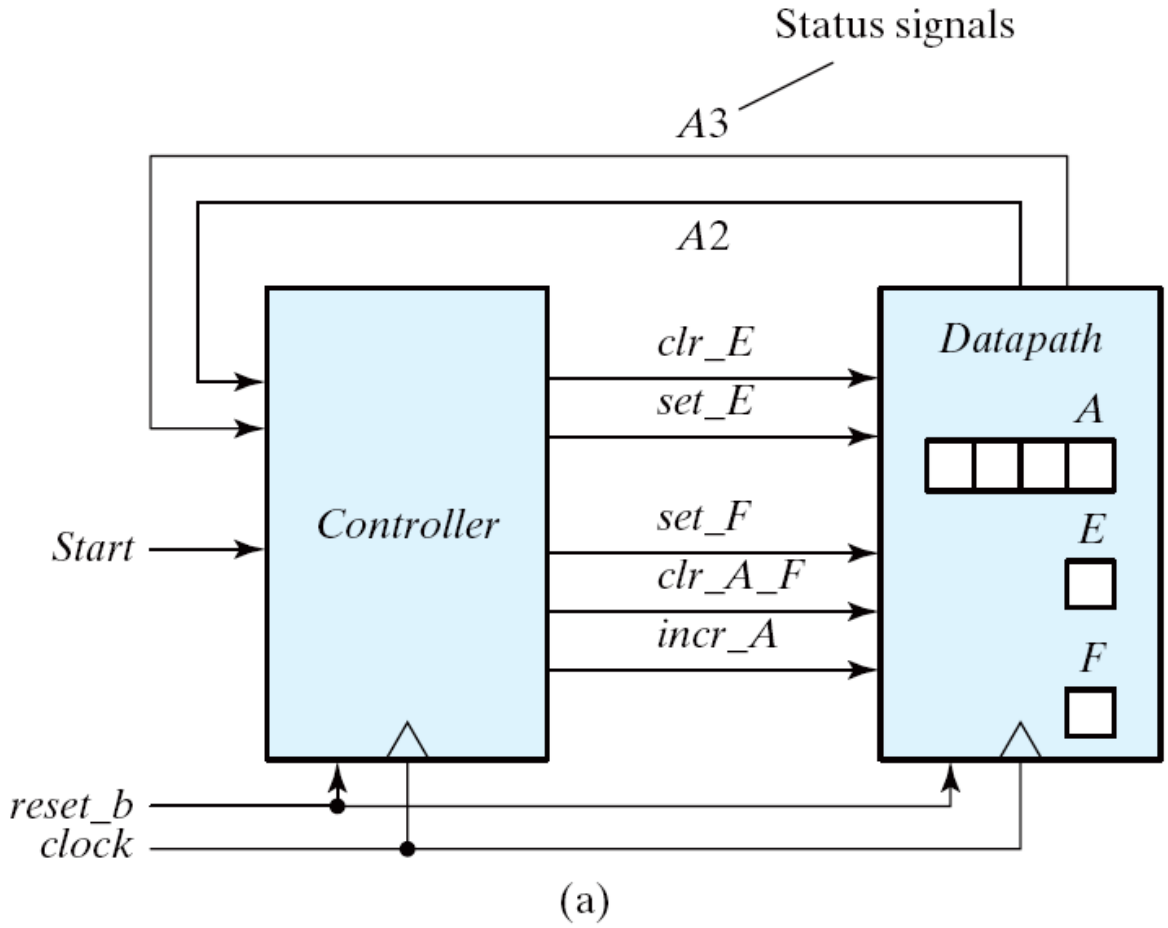


ASMD Chart

- **Contrasted between Algorithmic State Machine and Datapath (ASMD) charts & ASM charts.**
  - An ASMD chart are does not list register operations within a state box.
  - The edges of an ASMD charts are annotated with register operations that are concurrent with the state transition indicated by the edge.
  - An ASMD chart includes conditional boxes identifying the singles which control the register operations that annotate the edges of the chart.
  - An ASMD chart associates register operations with state transitions rather than with state.

- **Designed an ASMD chart have three-step:**
  - Form an ASM chart displaying only how the inputs to the controller determine its state transitions.
  - Convert the ASM chart to an ASMD chart by annotating the edges of ASM chart to indicate to the concurrent register operations of the datapath unit.
  - Modify the ASMD chart to identify the control singles that are generated by the controller and the cause the indicated register operations in the datapath unit.

8.5 Design Example



Note: A3 denotes A[3],
A2 denotes A[2],
<= denotes nonblocking assignment
reset_b denotes active-low reset condition

FIGURE 8.9 (a) Block diagram for design example

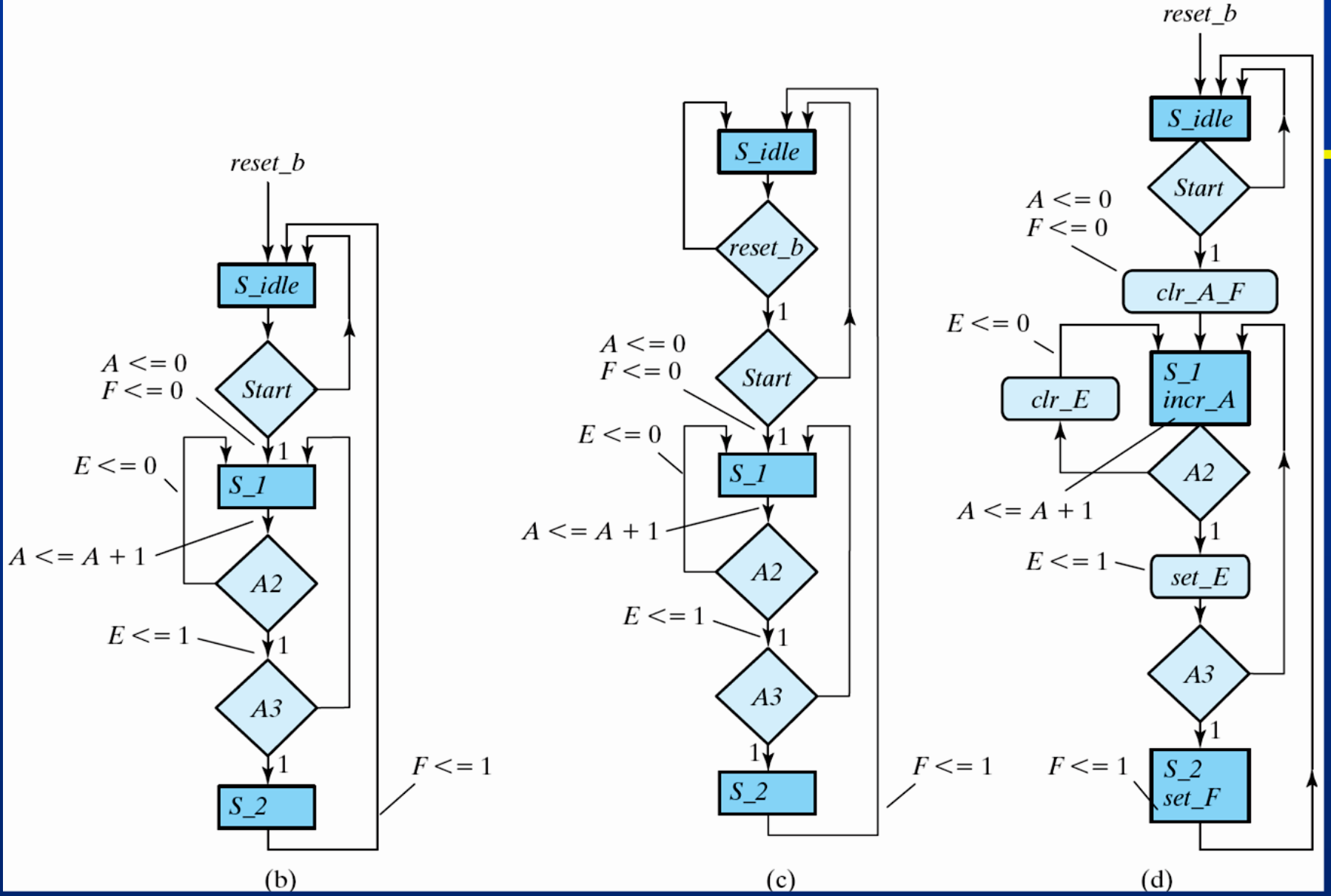


FIGURE 8.9 (b) ASMD chart for controller state transitions, asynchronous reset
(c)ASMD chart for controller state transitions, synchronous reset
(d)ASMD chart for a completely specified controller, asynchronous reset

Table 8.3
Sequence of Operations for Design Example

Counter				Flip-Flops			
A_3	A_2	A_1	A_0	E	F	Conditions	State
0	0	0	0	1	0	$A_2 = 0, A_3 = 0$	*S_1*
0	0	0	1	0	0		
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	$A_2 = 1, A_3 = 0$	
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	$A_2 = 0, A_3 = 1$	
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	$A_2 = 1, A_3 = 1$	
1	1	0	1	1	0		*S_2*
1	1	0	1	1	1		*S_idle*

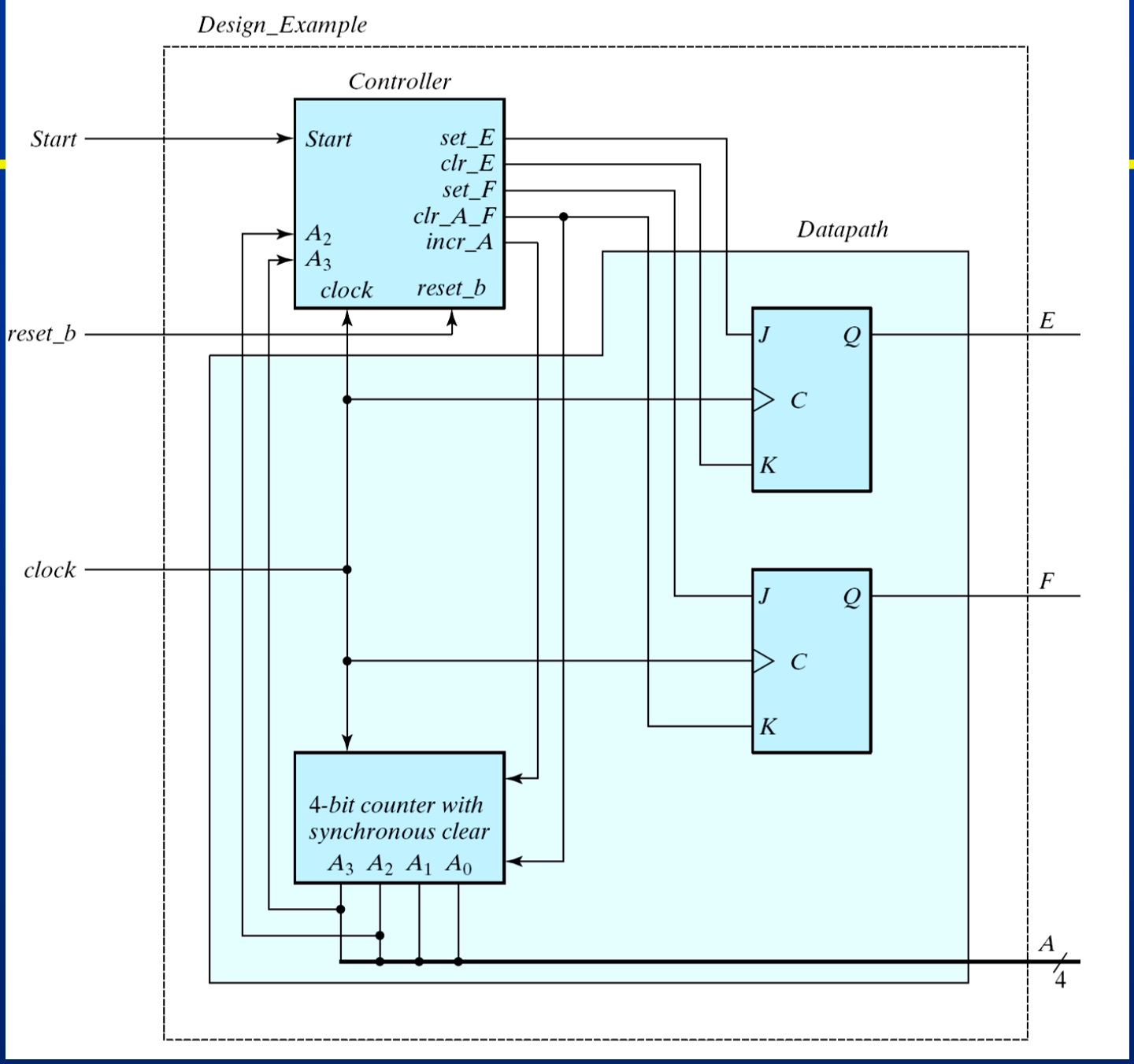
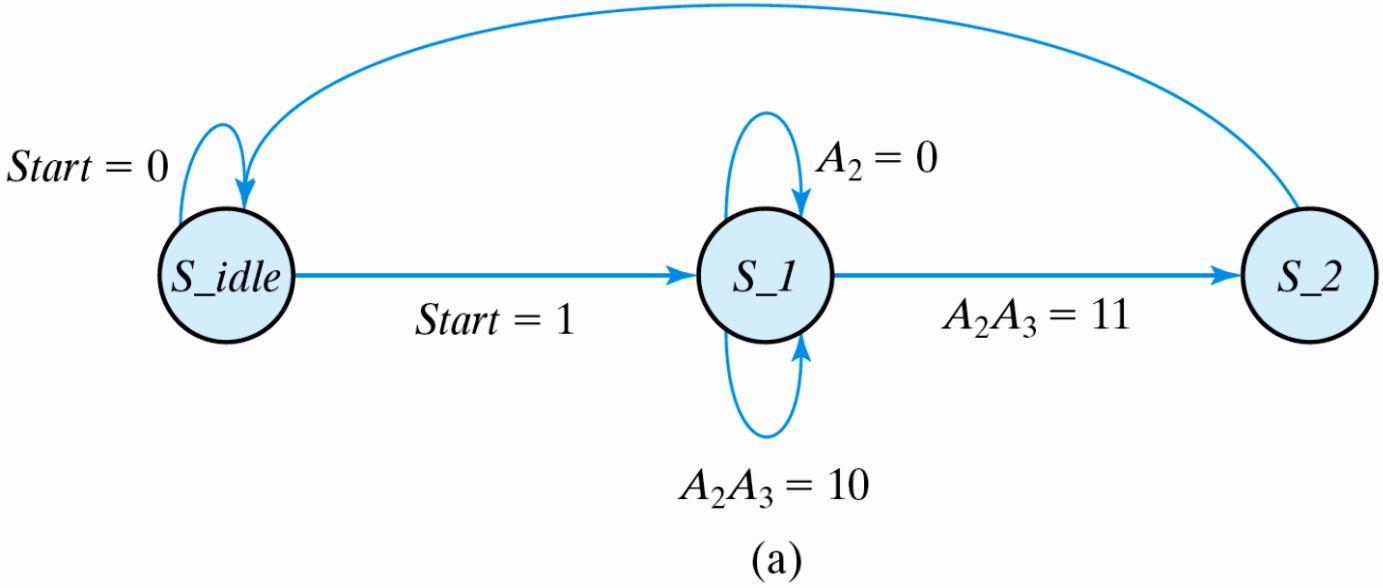


FIGURE 8.10 Datapath and controller for design example



(a)

S_idle ⟶	*S_1, clr_A_F:*	$A \leftarrow 0, F \leftarrow 0$
S_1 ⟶	*S_1, incr_A:*	$A \leftarrow A + 1$
	if $(A_2 = 1)$ then *set_E:*	$E \leftarrow 1$
	if $(A_2 = 0)$ then *clr_E:*	$E \leftarrow 0$
S_2 ⟶	*S_idle, set_F:*	$F \leftarrow 1$

(b)

FIGURE 8.11 Register transfer-level description of design example

Table 8.4
State Table for the Controller of Fig. 8.10

	Present State		Inputs			Next State		Outputs				
Present-State Symbol	G_1	G_0	*Start*	A_2	A_3	G_1	G_0	*set_E*	*clr_E*	*set_F*	*clr_A_F*	*incr_A*
S_idle	0	0	0	X	X	0	0	0	0	0	0	0
S_idle	0	0	1	X	X	0	1	0	0	0	1	0
S_1	0	1	X	0	X	0	1	0	1	0	0	1
S_1	0	1	X	1	0	0	1	1	0	0	0	1
S_1	0	1	X	1	1	1	1	1	0	0	0	1
S_2	1	1	X	X	X	0	0	0	0	1	0	0

- The *D* input of flip-flip G_1 must be equal to 1 during present state *S_1* when both inputs A_2 and A_3 are equal to 1. This condition is expressed with the D flip-flop input equation

$$D_{G1} = S\_1\ A_2\ A_3$$

Similarly, the next-state column of G_0 has four 1' s, and the condition for setting this flip-flop is

$$D_{G0} = Start\ S\_idle + S\_1$$

To derive the five output function, we can exploit the fact that state 10 is not used, which simplifies the equation for *clr_A_F* and enables us to obtain the following simplified set of output equations:

$$set\_E = S\_1\ A_2$$

$$clr\_E = S\_1\ A_2'$$

$$set\_F = S\_2$$

$$clr\_A\_F = Start\ S\_idle$$

$$incr\_A = S\_1$$

The logic diagram showing the internal detail of the controller of Fig. 8.10 is drawn in Fig. 8.12.

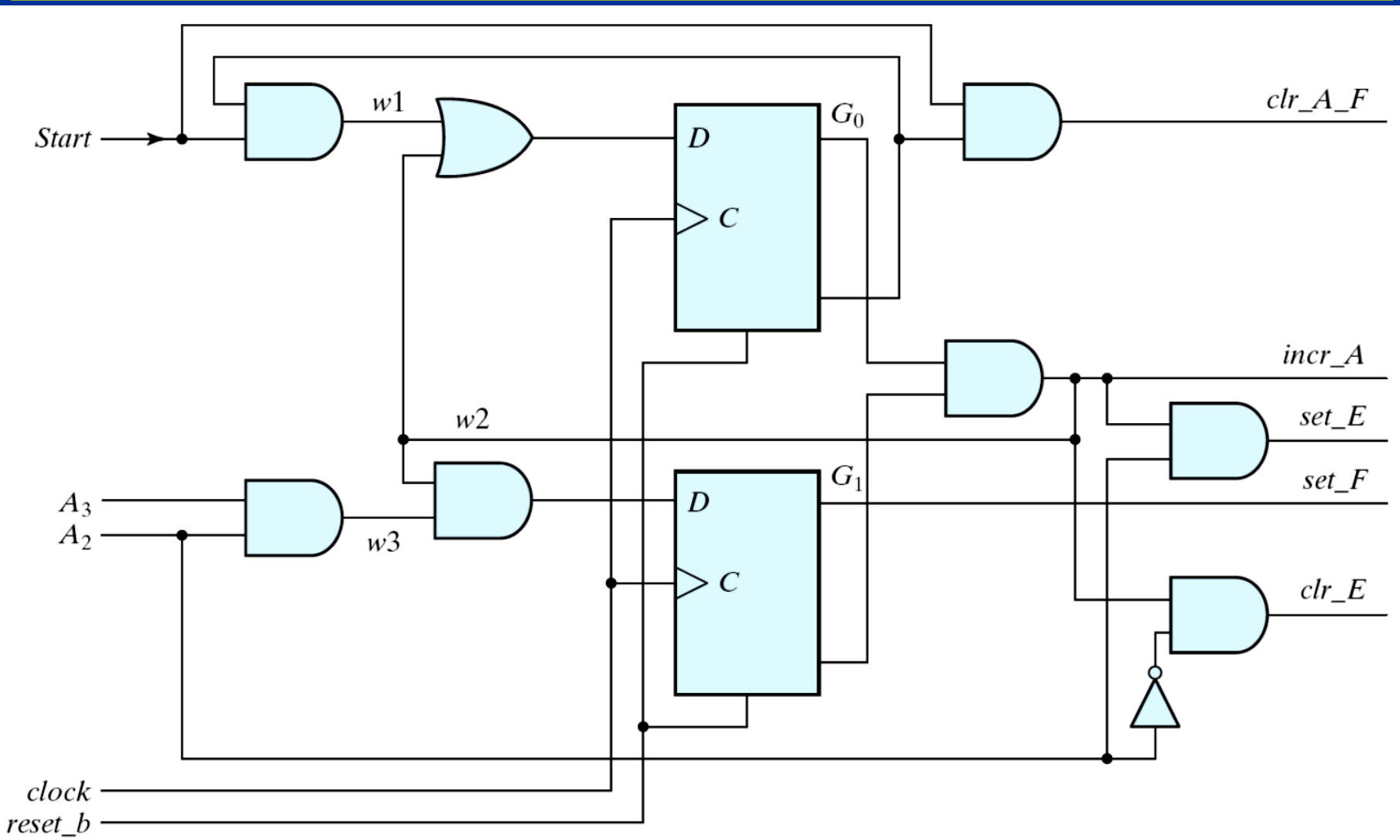
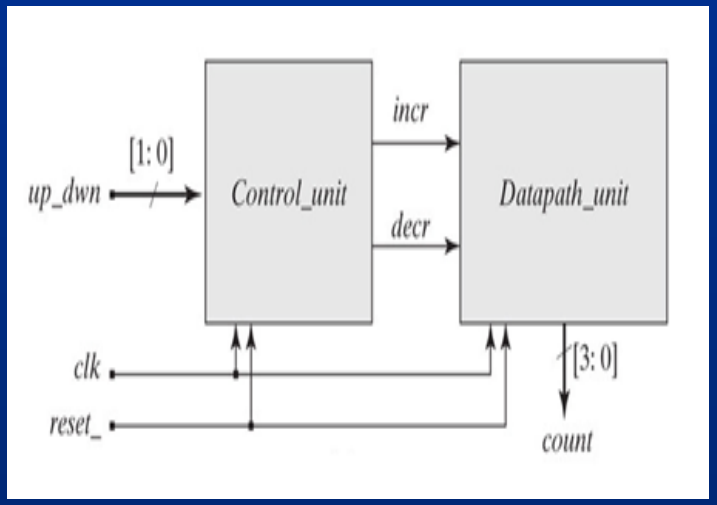
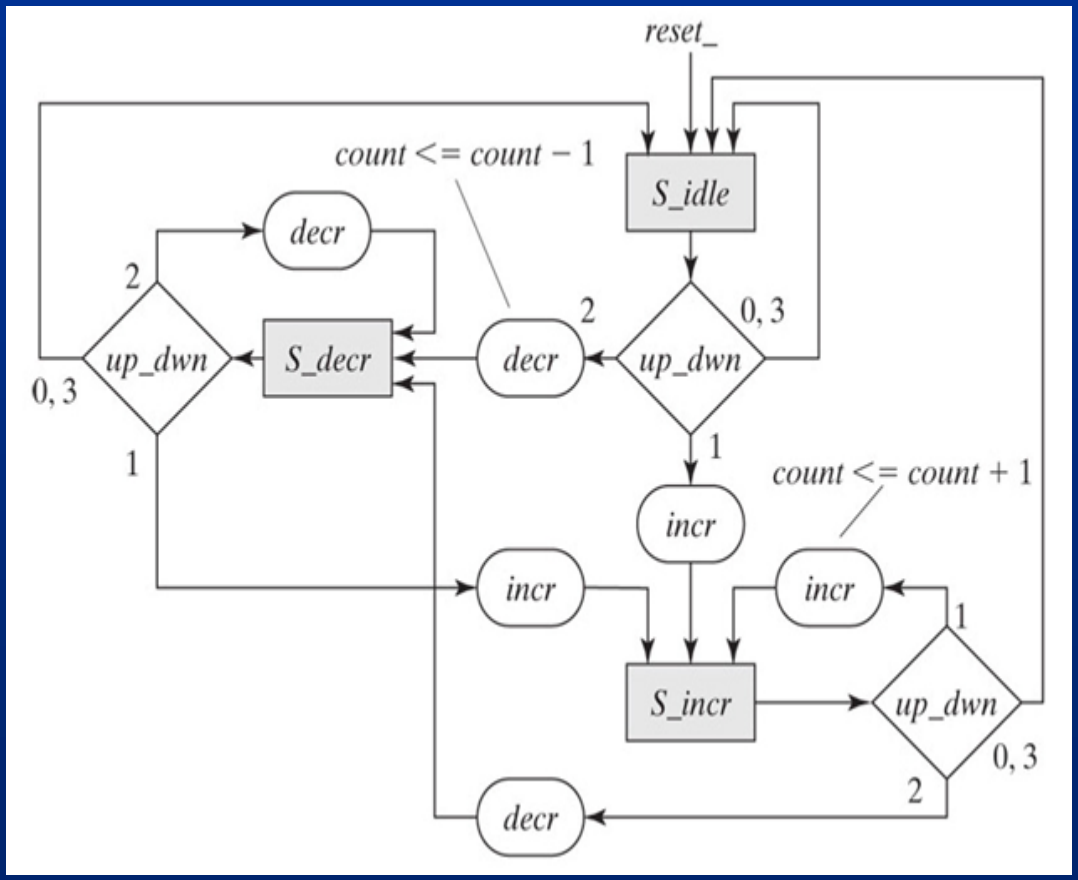


FIGURE 8.12 logic diagram of the control unit for Fig. 8.10

ASMD Chart for 4-bit Counter

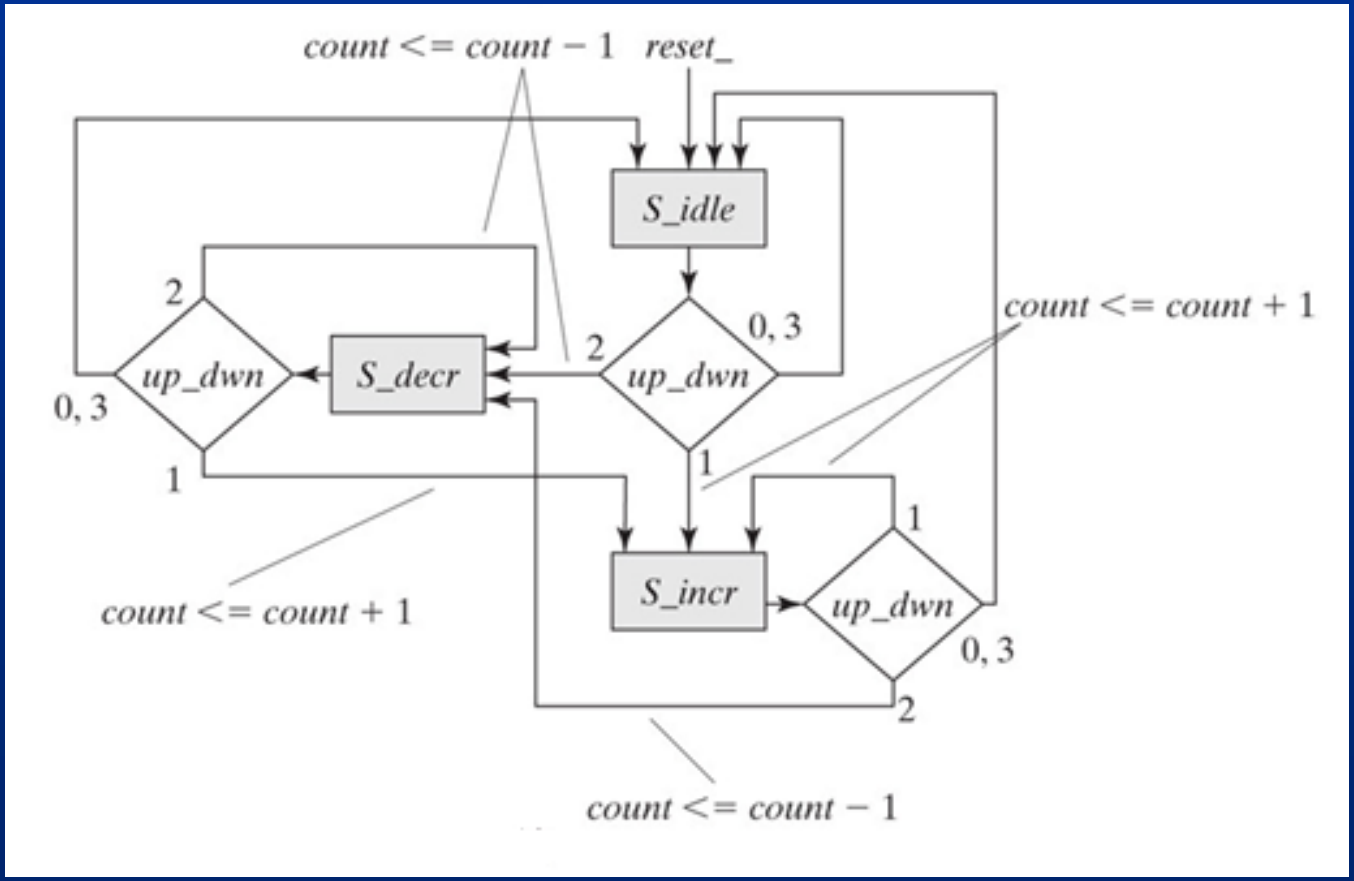
- A 4-bit counter that can count up, count down or hold the count





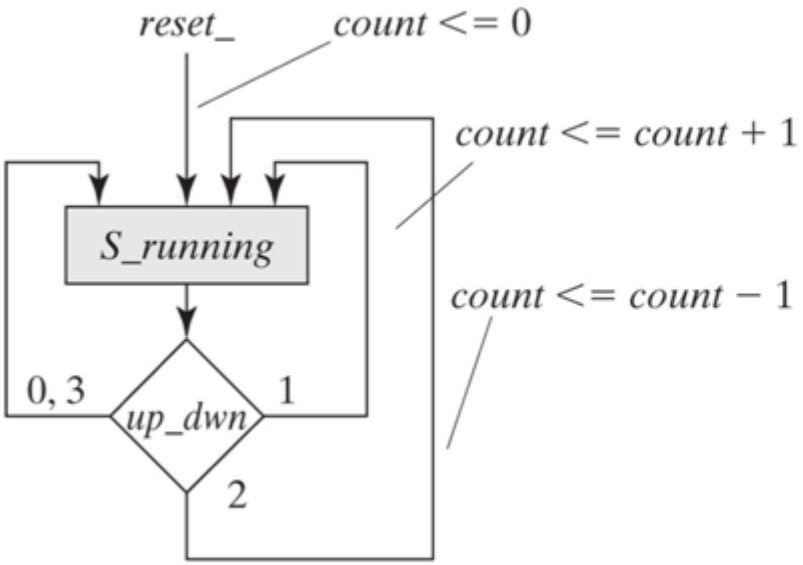
ASMD Chart for 4-bit Counter

Simplified ASMD Chart

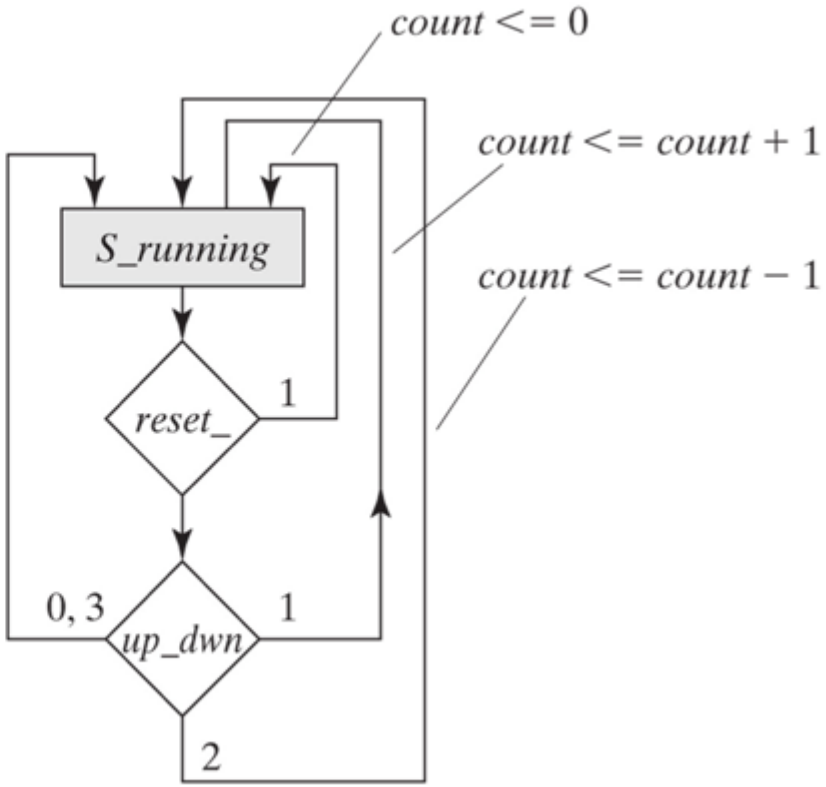


ASMD Chart for 4-bit Counter

Asynchronous Reset



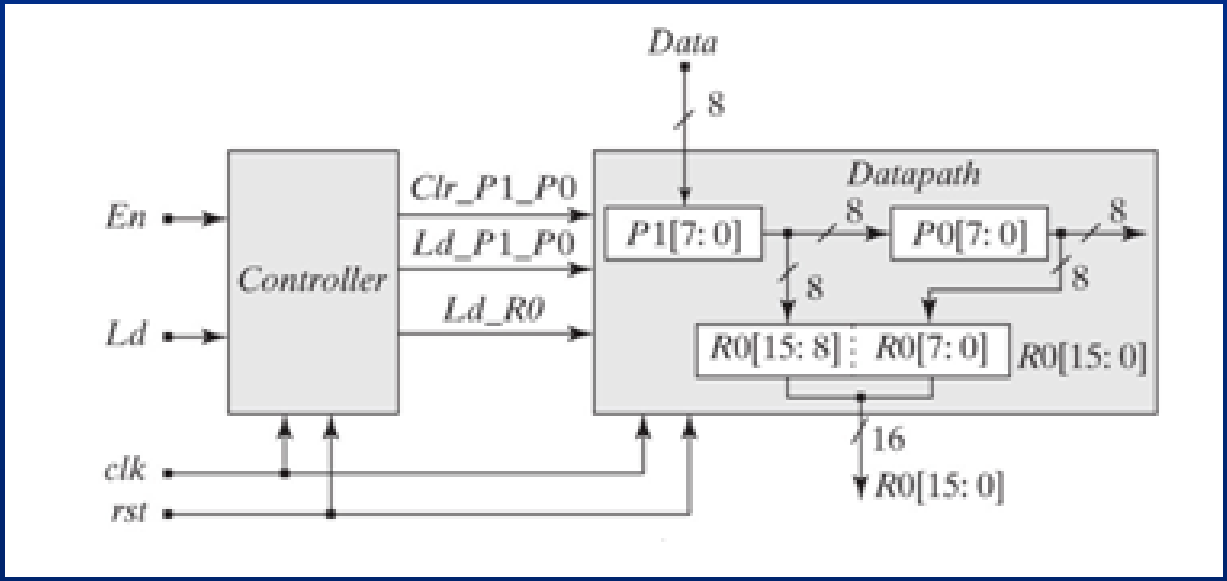
Synchronous Reset



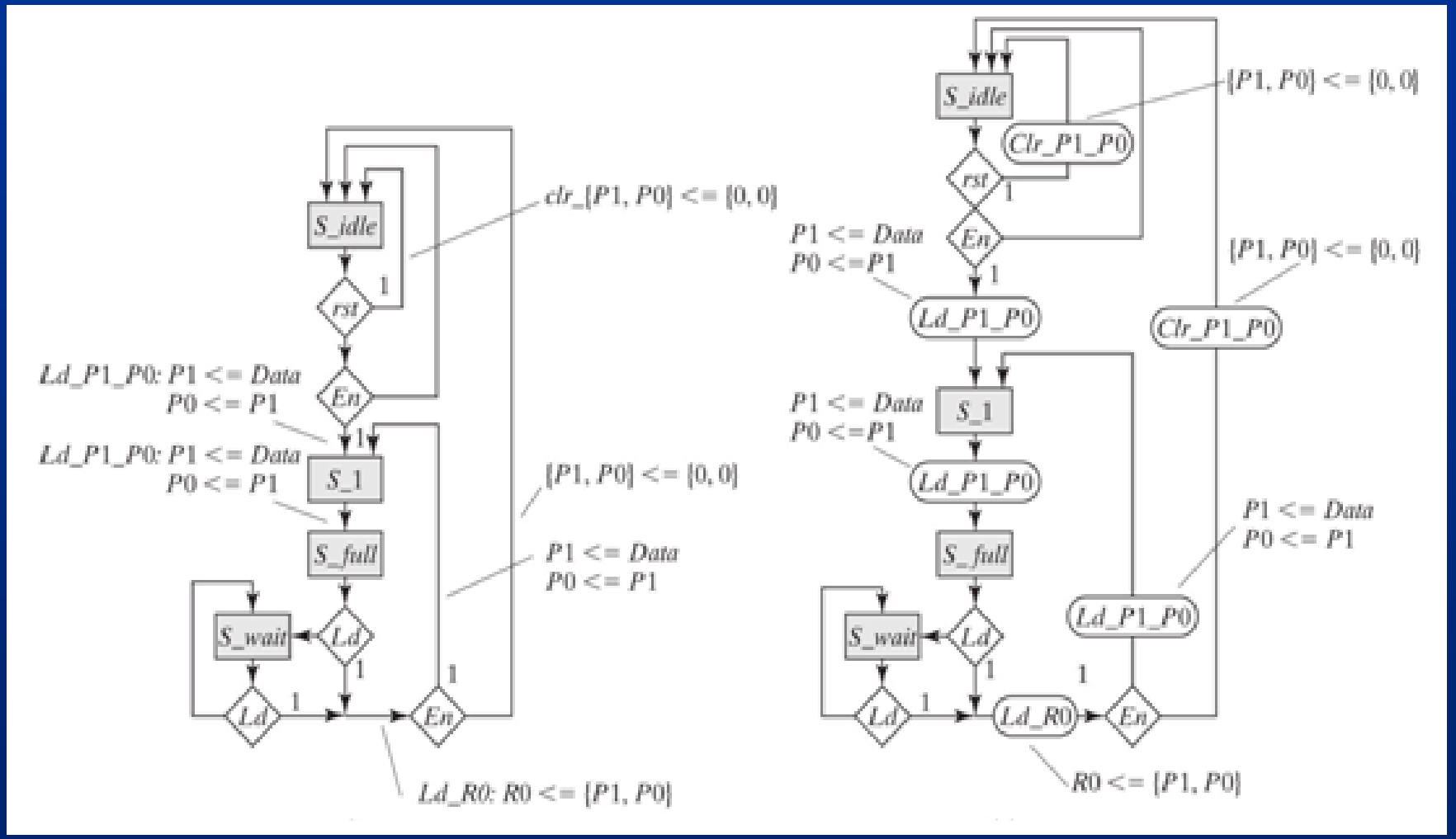
2:1 Decimator

- Decimators are used in digital signal processors to move data from a high-clock-rate datapath to a lower-clock-rate datapth.

Two-Stage Pipeline as 2:1 Decimator

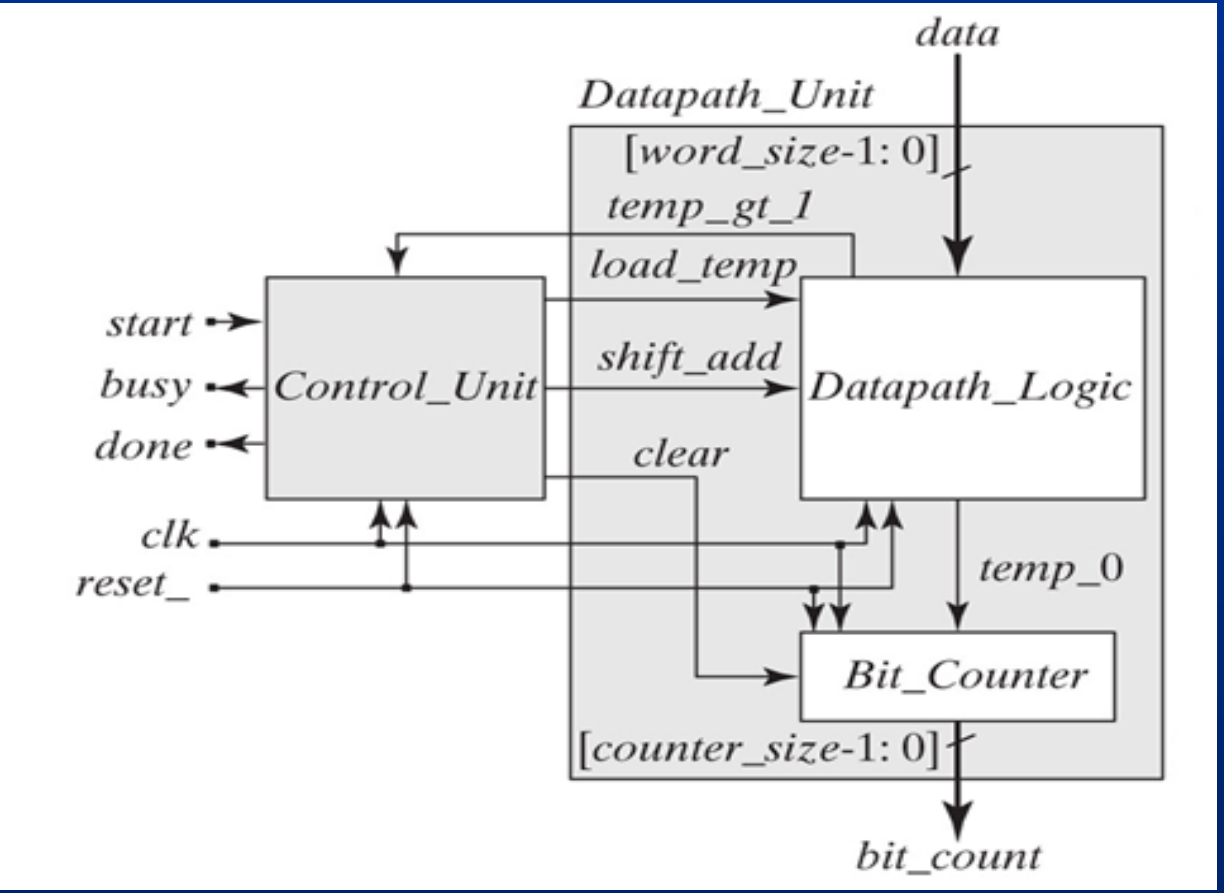


2:1 Decimator ASMD Chart

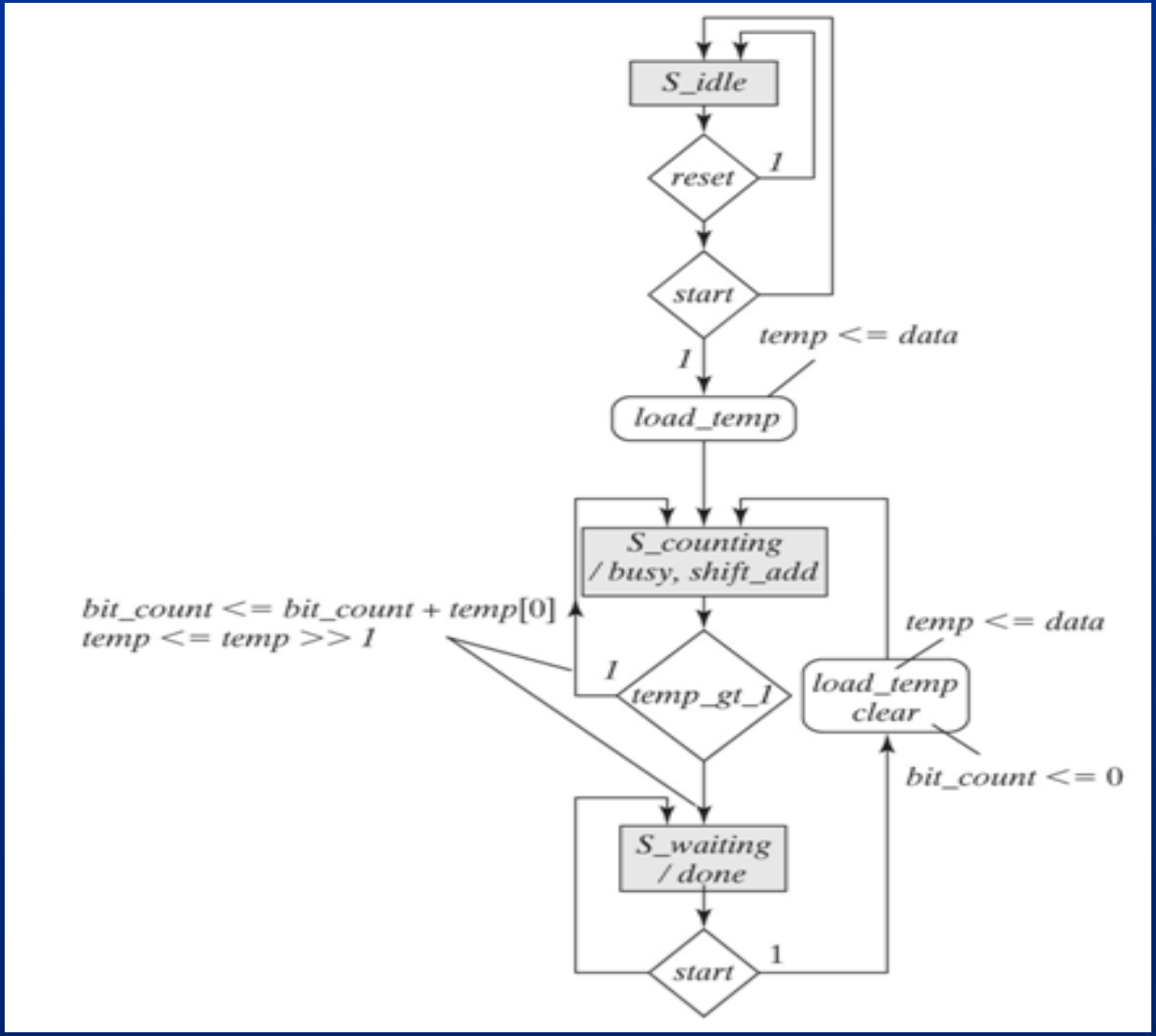


One's Count Circuit

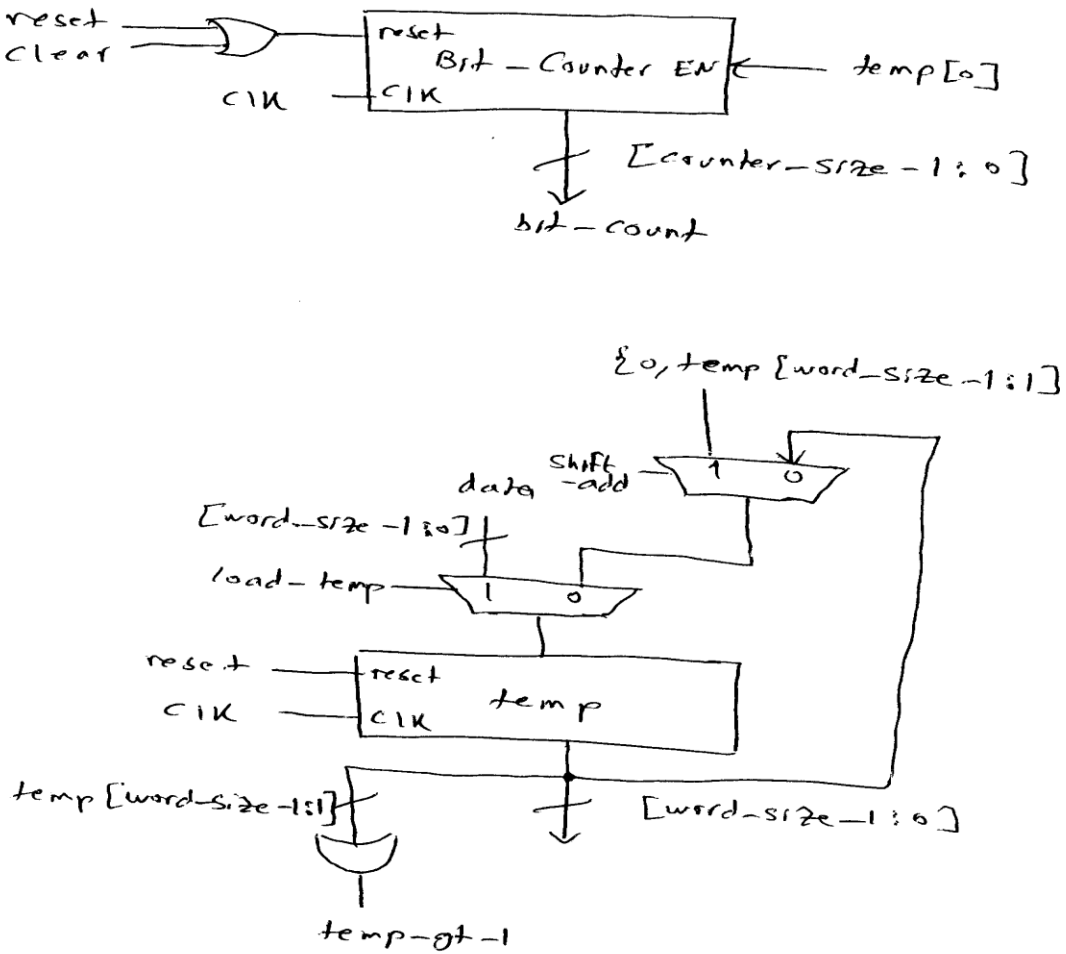
- A circuit that counts the 1's in a word and terminates activity as soon as possible.



One's Count Circuit

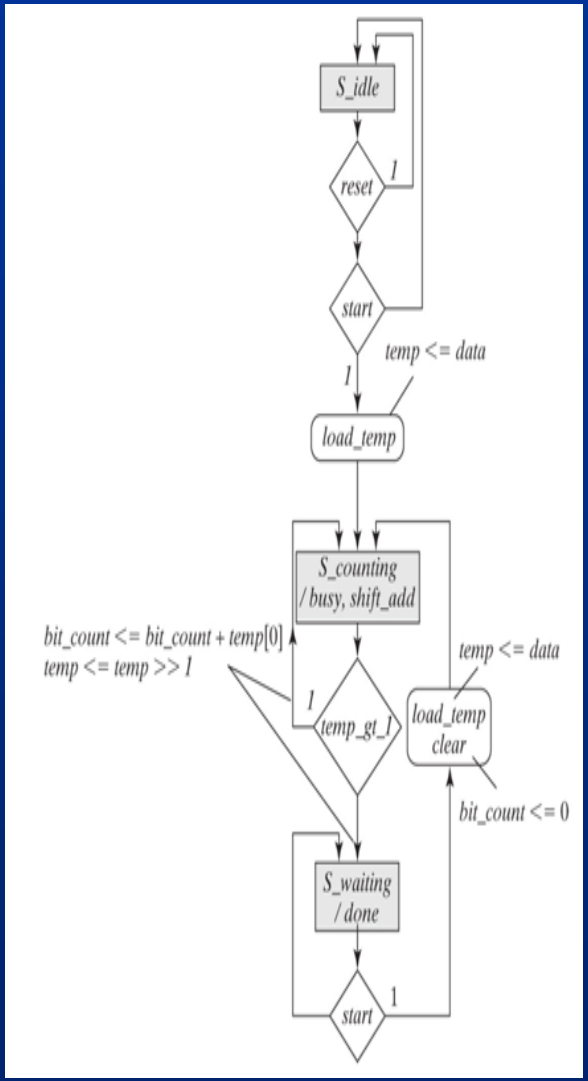


Data Path Unit



Control Unit

C.S.	Input		N.S.	Output				
	start	temp_gt_1		done	busy	load_temp	shift_add	clear
S_idle	0	x	S_idle	0	0	0	0	0
S_idle	1	x	S_counting	0	0	1	0	0
S_counting	x	1	S_counting	0	1	0	1	0
S_counting	x	0	S_waiting	0	1	0	1	0
S_waiting	1	x	S_counting	1	0	1	0	1
S_waiting	0	x	S_waiting	1	0	0	0	0



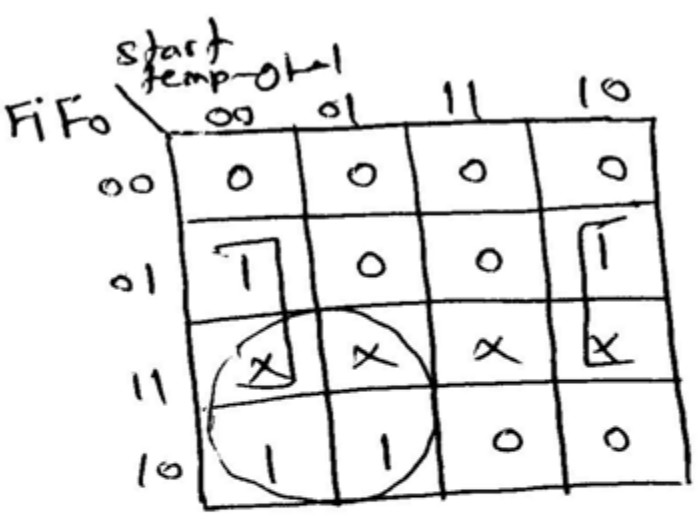
Control Unit

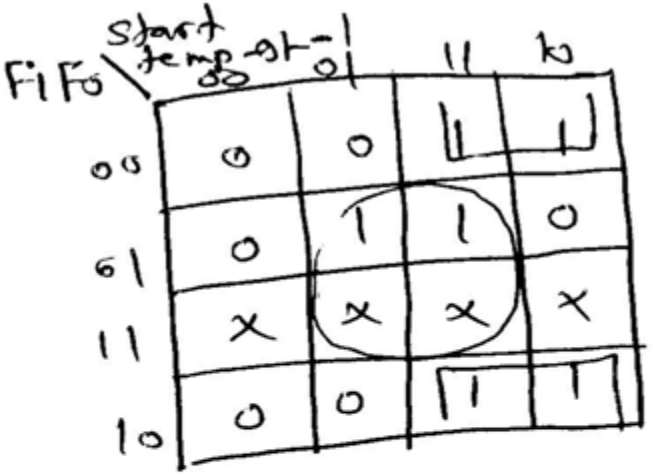
- **Assuming the two flip flops F1 and F0 for storing the machine state and the state assignment: *S_idle=00, S_counting =01*, and *S_waiting=10*, the control signals will be as follows:**

  - done = F1 F0'
  - busy = F1' F0
  - shift_add = F1' F0
  - load_temp = F0' start
  - clear = F1 F0' start

C.S.	Input		N.S.	Output				
	start	temp_gt_1		done	busy	load_temp	shift_add	clear
S_idle	0	x	S_idle	0	0	0	0	0
S_idle	1	x	S_counting	0	0	1	0	0
S_counting	x	1	S_counting	0	1	0	1	0
S_counting	x	0	S_waiting	0	1	0	1	0
S_waiting	1	x	S_counting	1	0	1	0	1
S_waiting	0	x	S_waiting	1	0	0	0	0

Control Unit





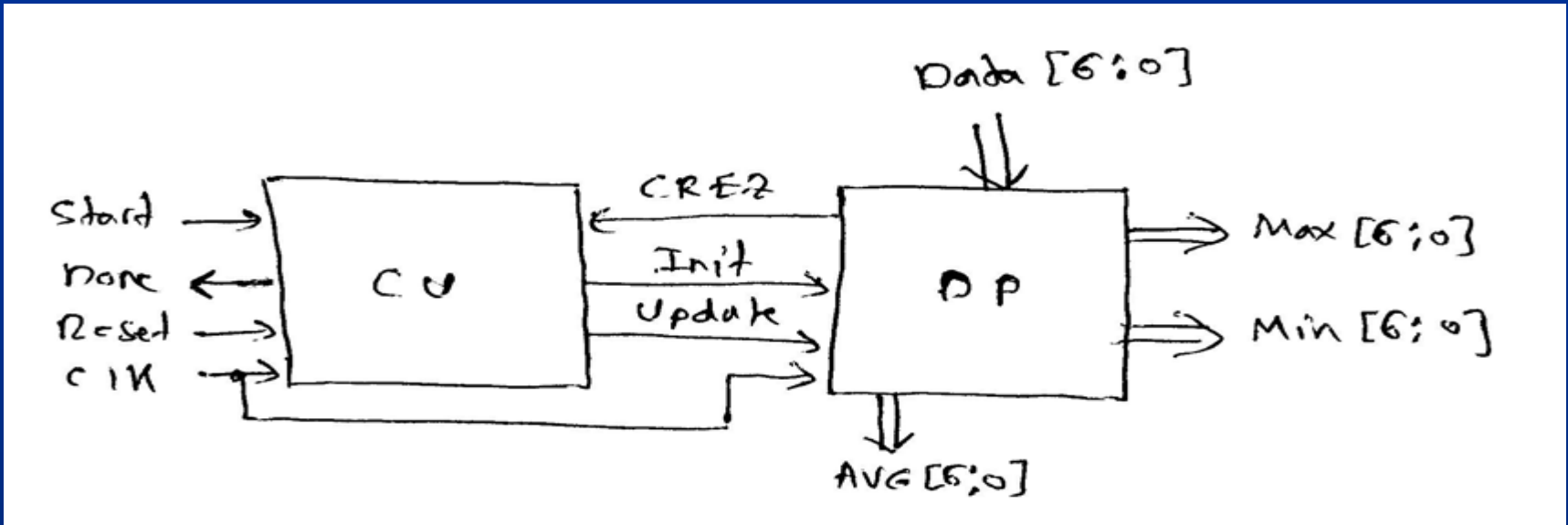
$$D_1 = F_1 \overline{Start} + F_0 \overline{temp\text{-}ot\text{-}1}$$

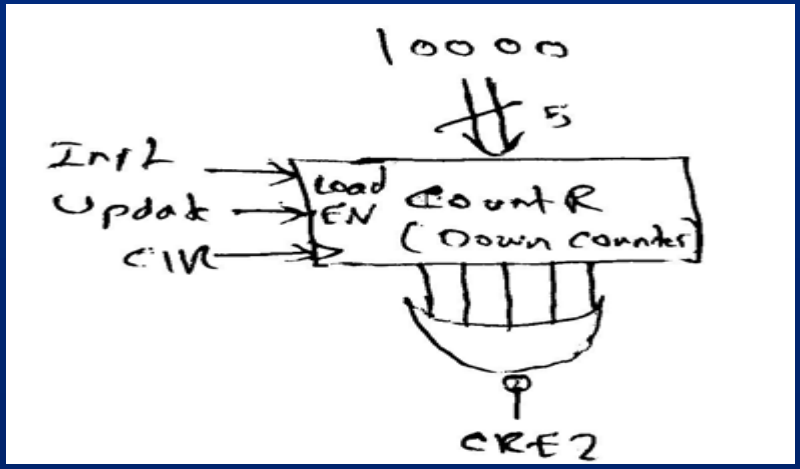
$$D_0 = \overline{F_0}\, start + F_0\, temp\text{-}ot\text{-}1$$

Scores Avg., Max. & Min.

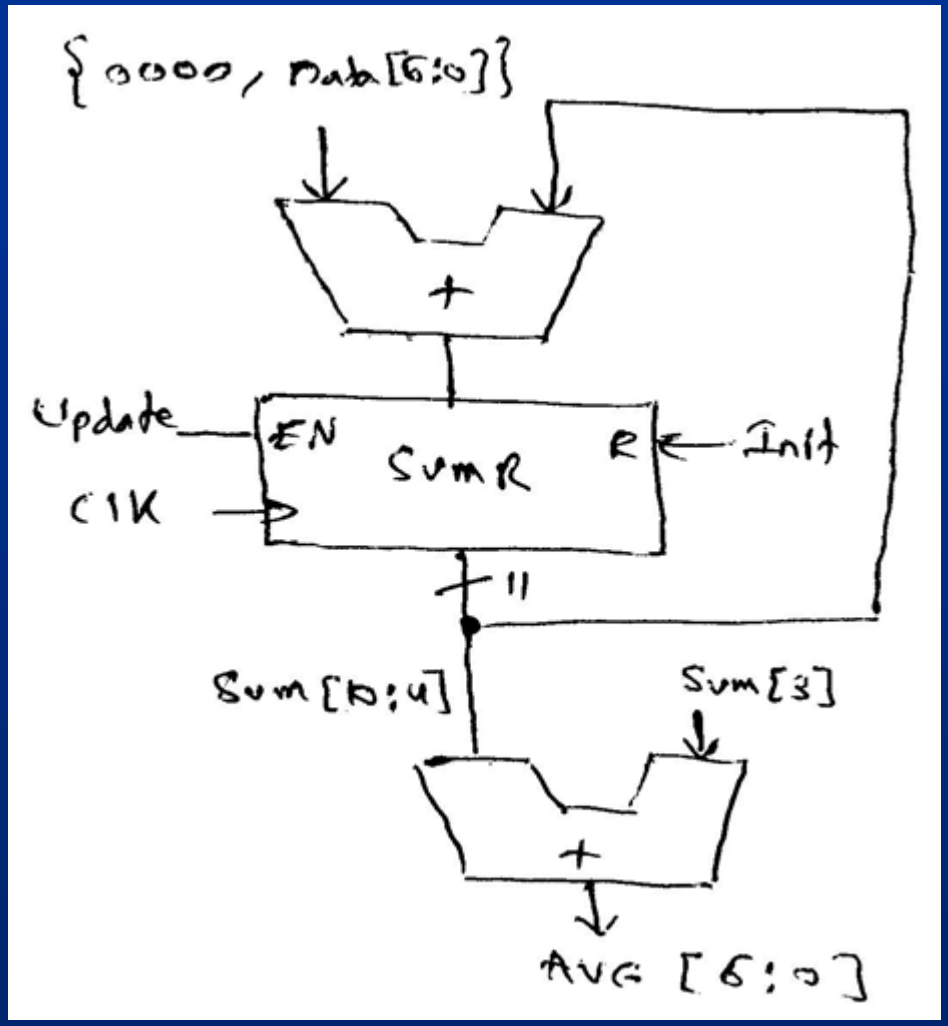
- Design a circuit that computes the average, maximum and minimum of 16 scores
- Each score has a value in the range [0,100]
- Scores will be fed to the circuit one score at a time the next cycle following the assertion of a ***Start*** input
- Once the circuit finishes computation, it will assert a ***Done*** signal and will generate the average, maximum and minimum scores
- The average will be shown as integer number resulting from dividing the sum by 16 with rounding the result to the nearest integer

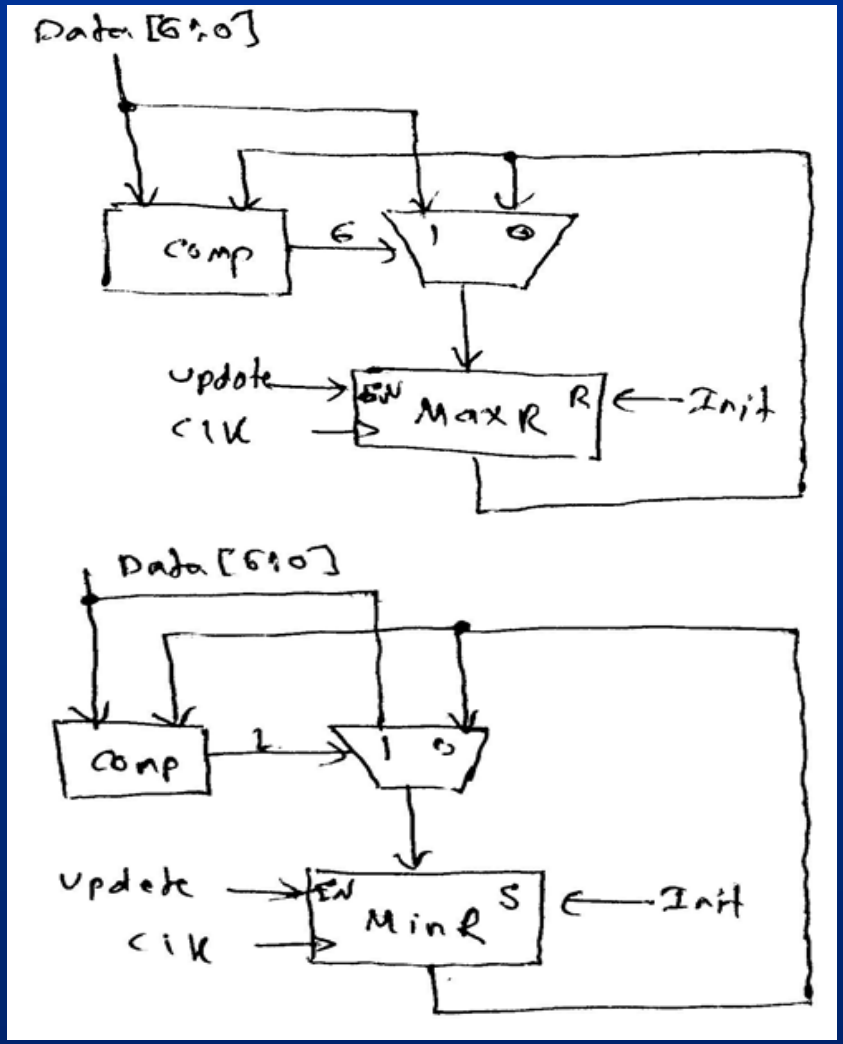
Design Block Diagram



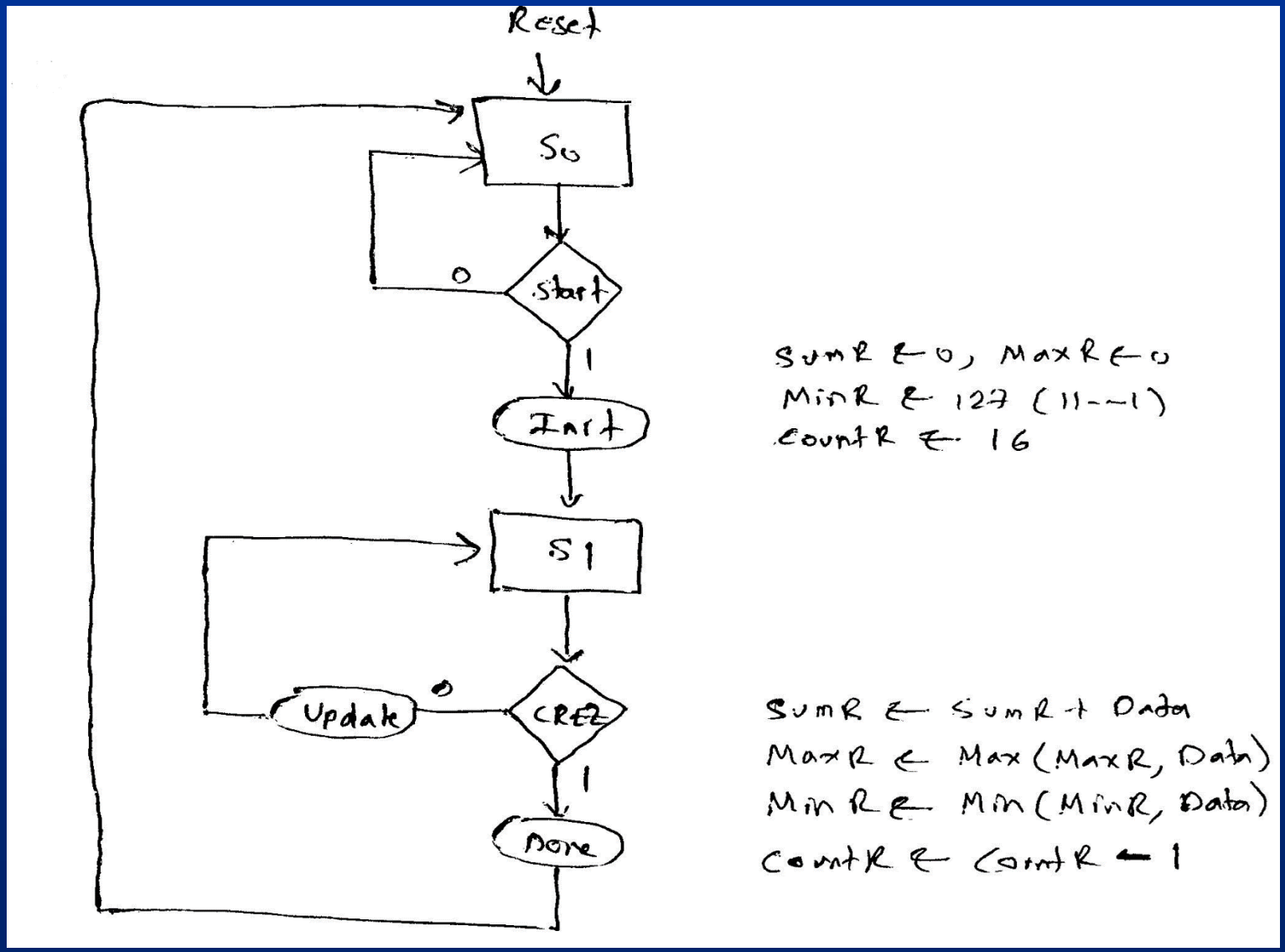


Data Path Design





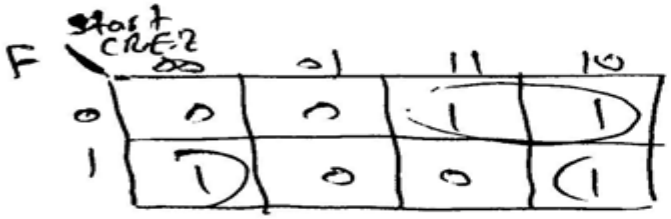
ASMD Chart


Reset
S0
Start
0
1
Init
S1
Updak
0
CREZ
1
Done
SumR ← 0, MaxR ← 0
MinR ← 127 (11--1)
CountR ← 16
SumR ← SumR + Data
MaxR ← Max (MaxR, Data)
MinR ← Min (MinR, Data)
CountR ← CountR − 1

Control Unit Design

C.S.	Start	CREZ	N.S.	Init	Updat	Done
S_0	0	–	S_0	0	0	0
S_0	1	–	S_1	1	0	0
S_1	–	0	S_1	0	1	0
S_1	–	1	S_0	0	0	1

Let $S_0 = 0$ and $S_1 = 1$



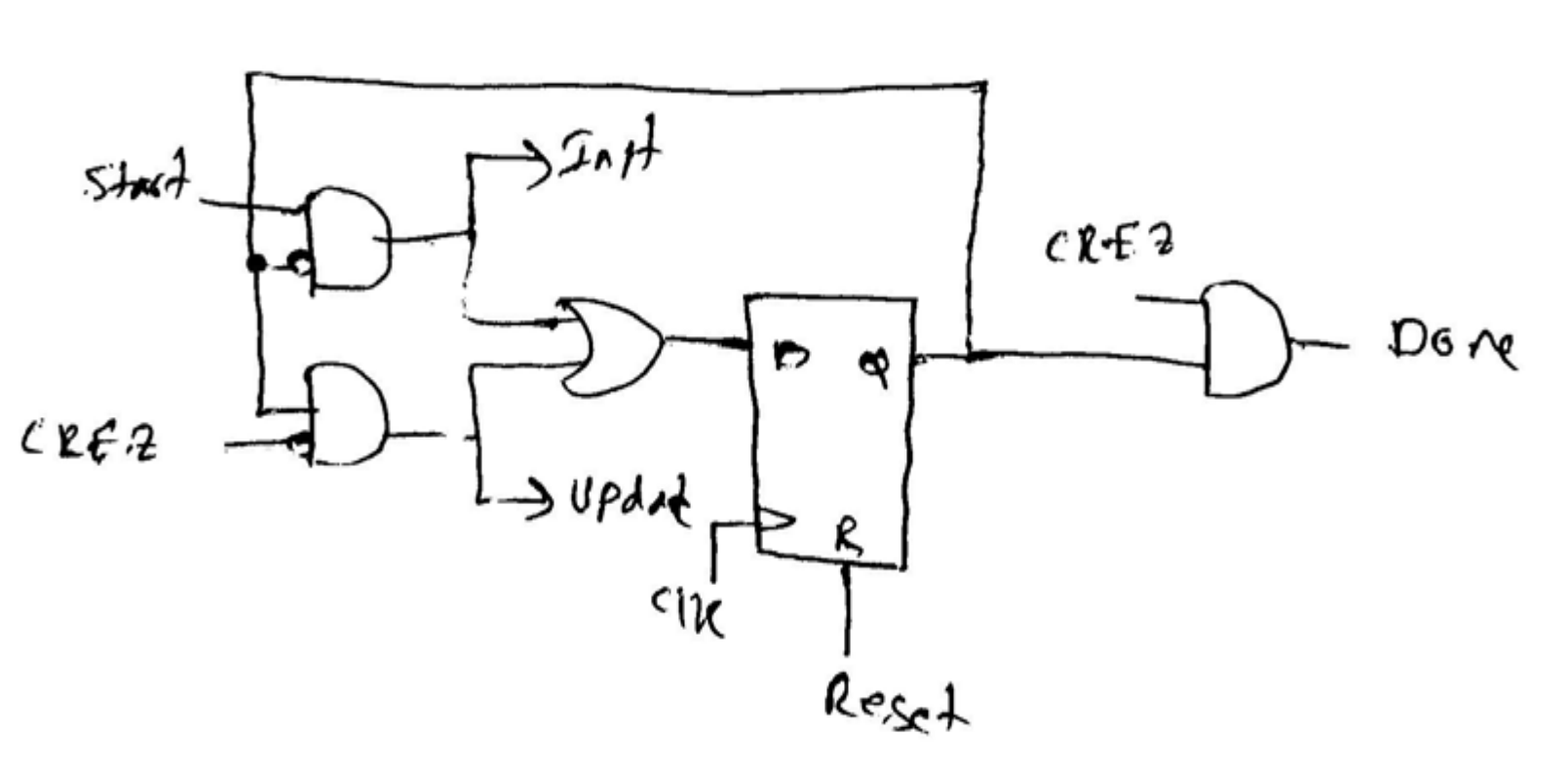
$$F^+ = \bar{F}\,Start + F\,\overline{CREZ}$$

$$Init = \bar{F} \cdot Start$$

$$Update = F \cdot \overline{CREZ}$$

$$Done = F \cdot CREZ$$

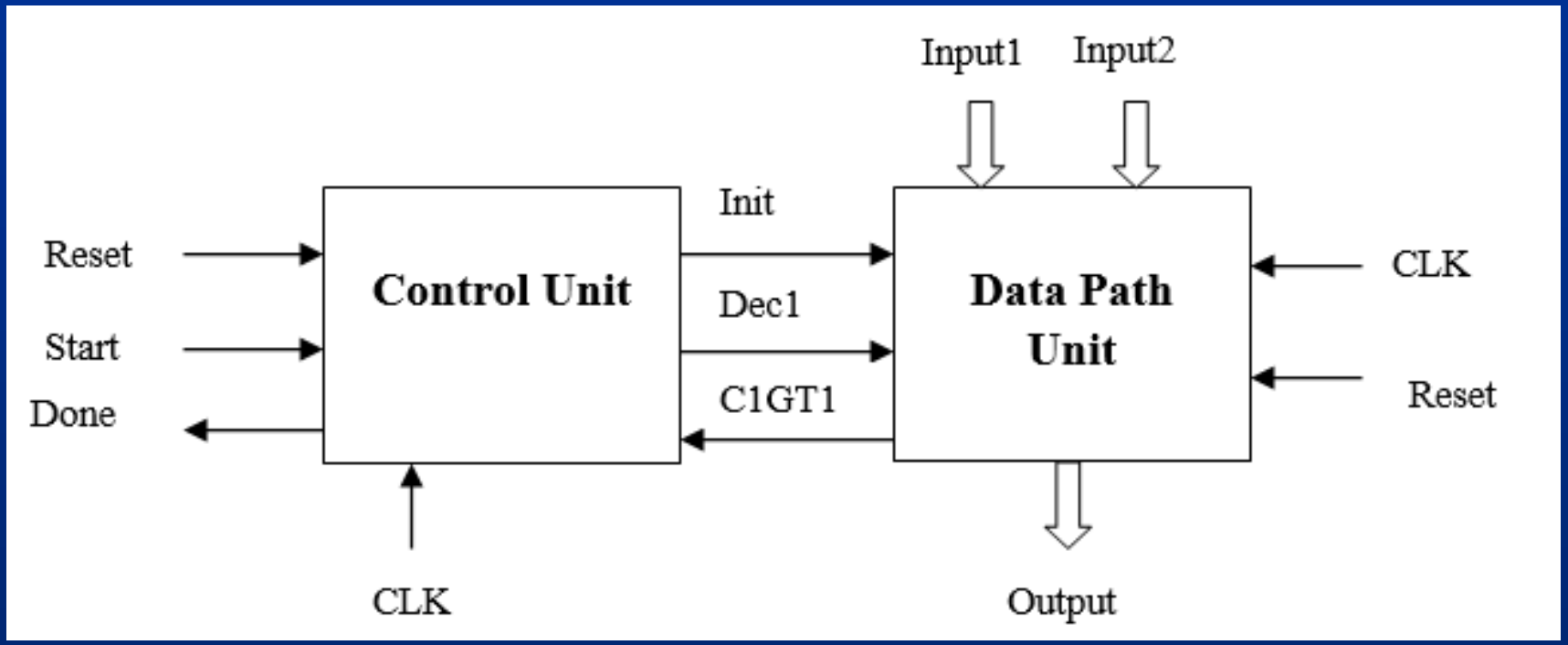
Control Unit Design



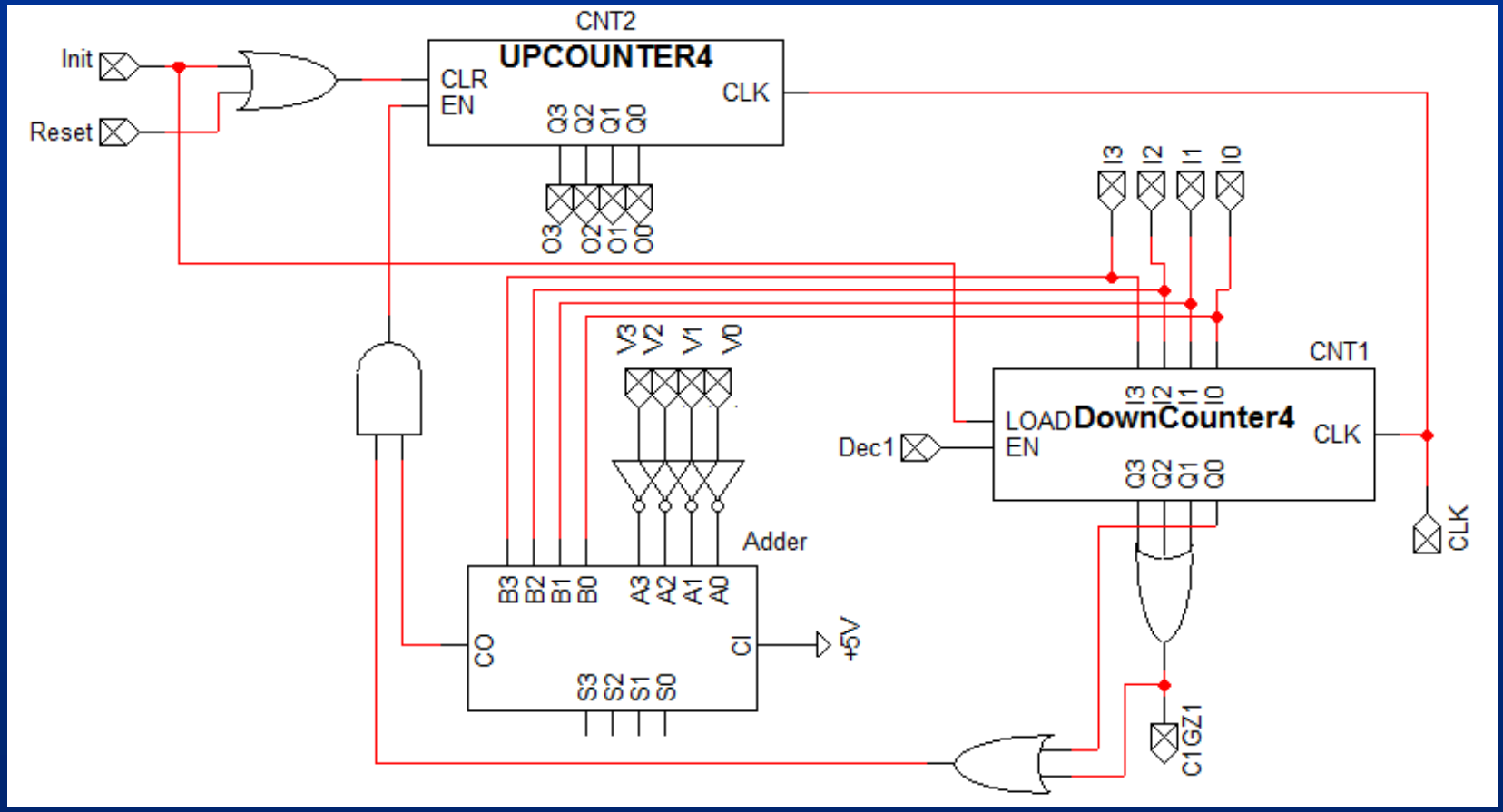
Counting Number of Elements ≥Target Value

- It is required to design a circuit that receives an array of n unsigned 4-bit numbers along with a 4-bit target value and counts the number of data elements that are greater than or equal to the target value.
- Assume that once the user presses ***Start*** he will also supply the number of elements in the array **n** (n≤15) through input1 and the target value through input2 in the same cycle.
- Assume that input2 will hold the target value until the circuit finishes its operation.
- In the next n consecutive cycles, the user will provide the n data elements through input1.
- Once the circuit finishes computation, it will assert a ***Done*** signal and will generate a 4-bit output indicating the count of the number of elements in the array ≥ target value.
- The ***Done*** signal and the result will remain valid unless the user resets the machine or asserts the ***Start*** signal.

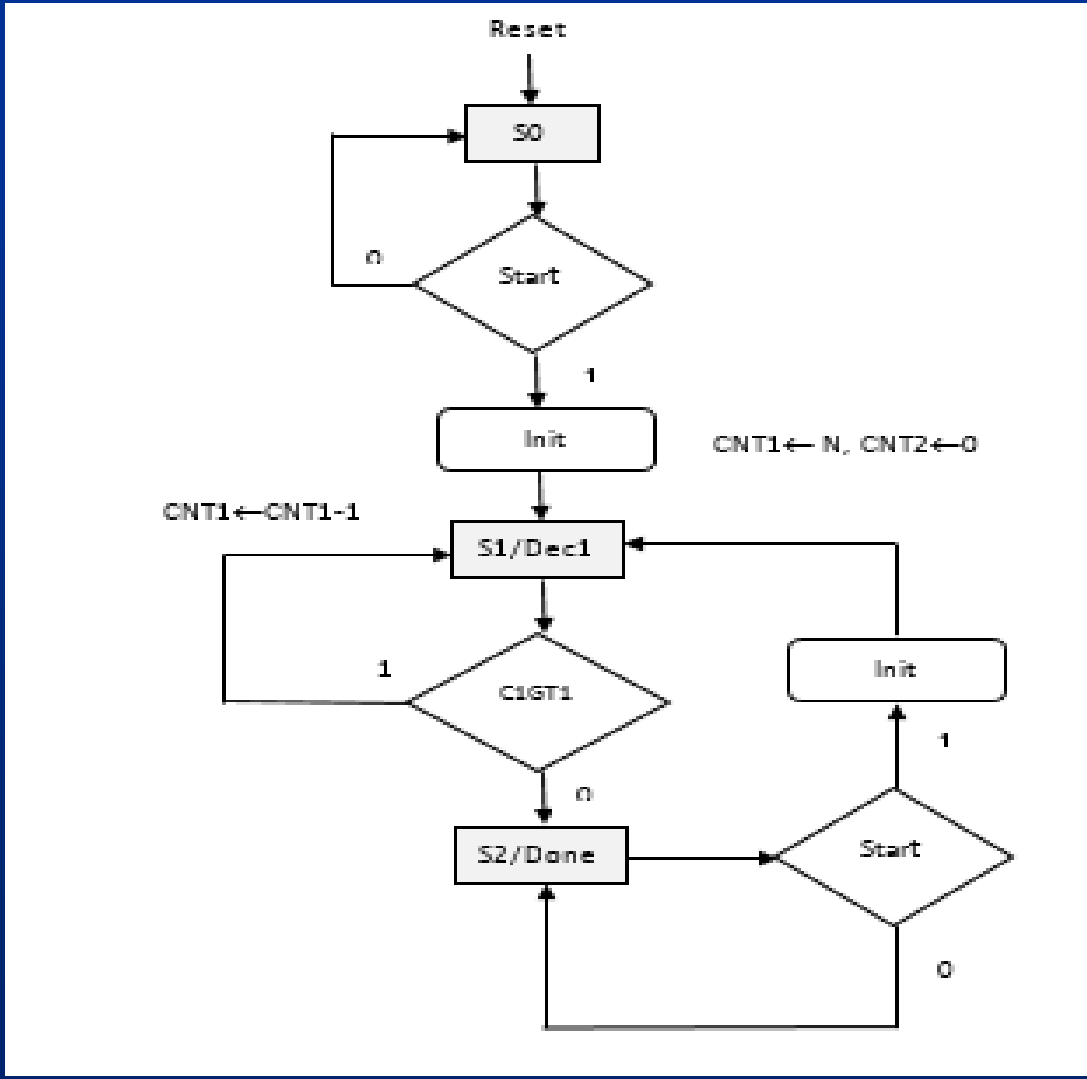
Data Bath & Control Unit Block Diagram



Data Path Design



ASMD Chart


Reset
S0
0
Start
1
Init
CNT1← N, CNT2←0
CNT1←CNT1-1
S1/Dec1
1
C1GT1
0
Init
1
S2/Done
Start
0

Control Unit Design

C.S.	Input		N.S.	Output		
	Start	C1GZ1		Init	Dec1	Done
S0	0	X	S0	0	0	0
S0	1	X	S1	1	0	0
S1	x	0	S2	0	1	0
S1	x	1	S1	0	1	0
S2	0	X	S2	0	0	1
S2	1	X	S1	1	0	1

- **We use the binary state assignment S0=00, S1=01, S2=10, and the two Flip Flops, F1 F0, to encode states.**
- **The output signals equations are as follows:**
  - Init = F0' Start
  - Dec1 = F0
  - Done = F1

Control Unit Design

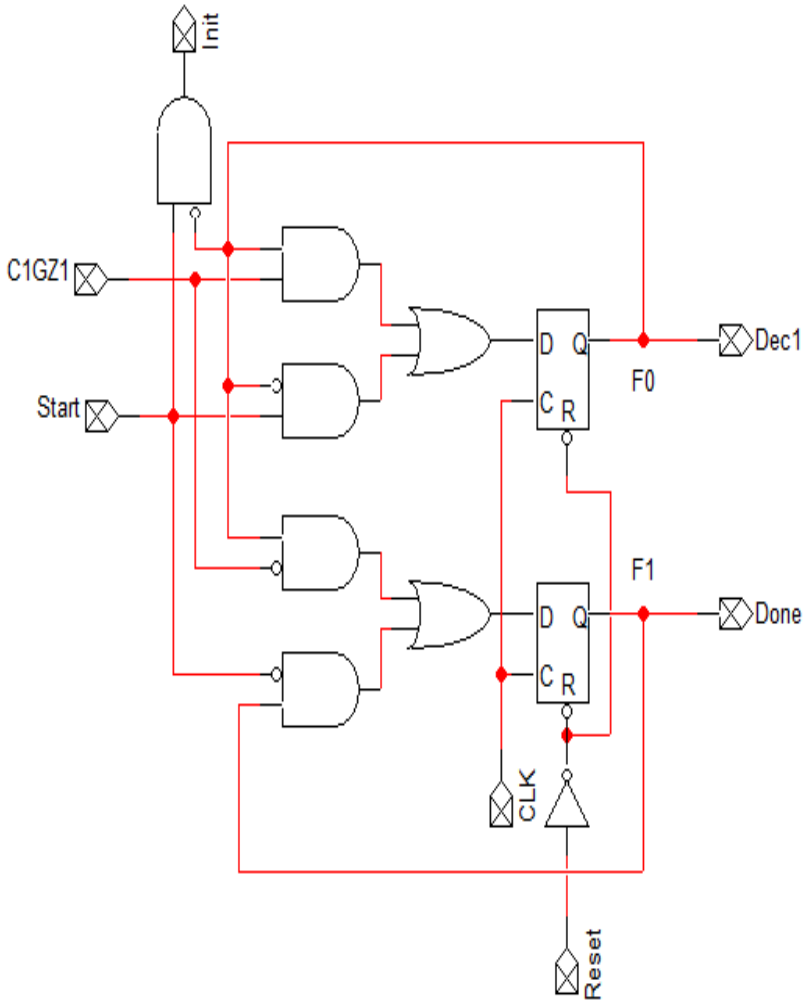
The next state equations are as follows:

	00	01	11	10
00	0 0	0 1	1 3	1 2
01	0 4	1 5	1 7	0 6
11	? 12	? 13	? 15	? 14
10	0 8	0 9	1 11	1 10

F0+= F0 C1GZ1 + F0' Start

	00	01	11	10
00	0 0	0 1	0 3	0 2
01	1 4	0 5	0 7	1 6
11	? 12	? 13	? 15	? 14
10	1 8	1 9	0 11	0 10

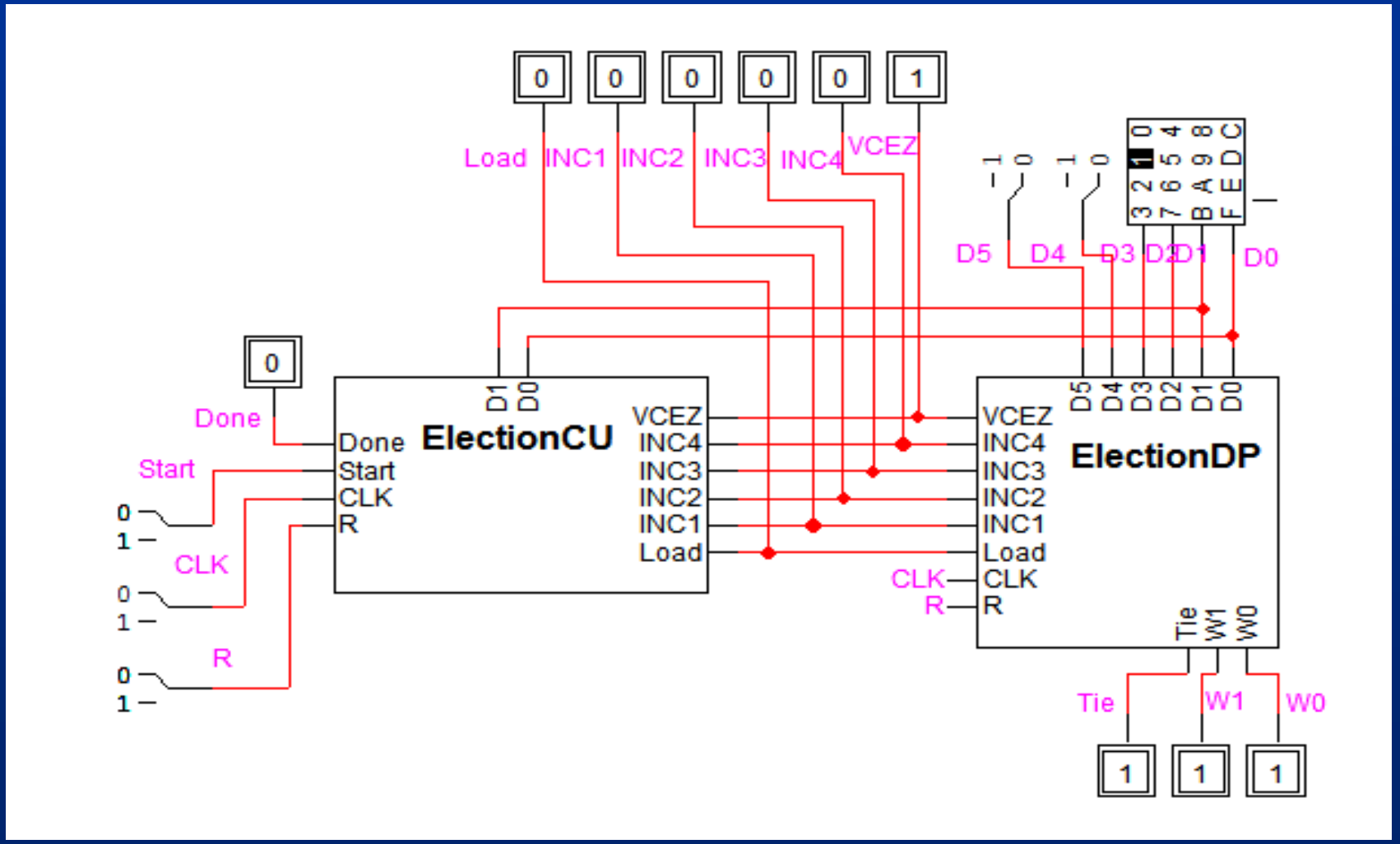
F1+ = F0 C1GZ1' + F1 Start'



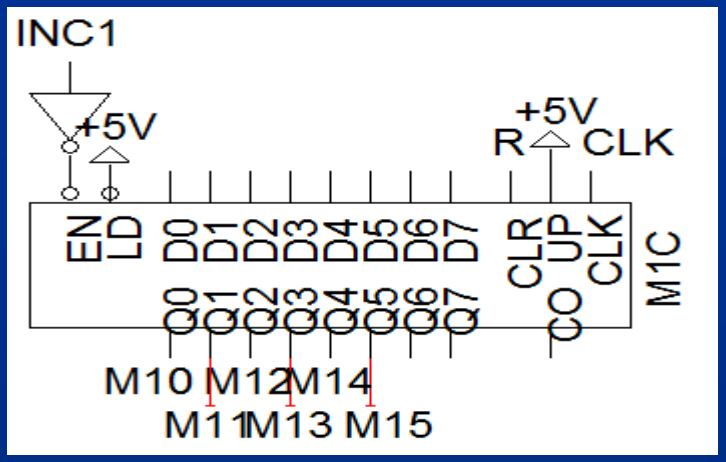
Election Circuit

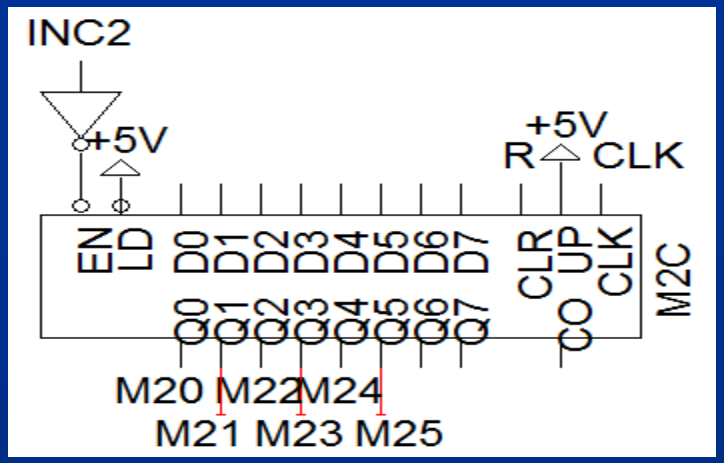
- It is required to design a circuit that computes the results of election and determines the winner.
- It is assumed that there are four members competing in the election with the following codes: Member 1: 00, Member 2: 01, Member 3: 10, and Member 4: 11.
- Assume that the number of votes to be counted will be given to the circuit when a *Start* input is set. Assume for simplicity that the maximum number of votes to be counted is 63.
- Assume that votes will be given to the circuit one vote at a time before the rising edge of each clock cycle.
- Once the circuit finishes computation, it will assert a *Done* signal and will generate a 2-bit output indicating the code of the winner. In case there is a tie, a Tie signal is set to 1.

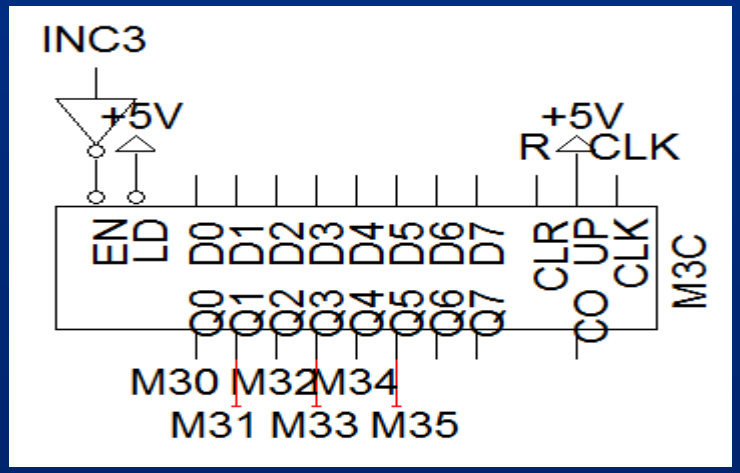
Block Diagram

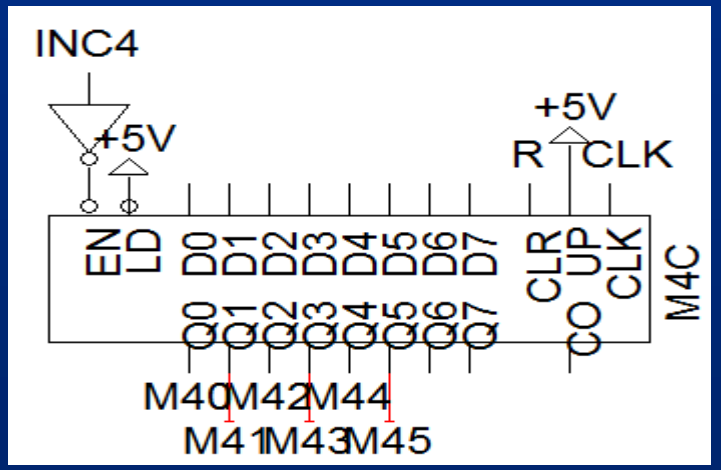


Data Path Design

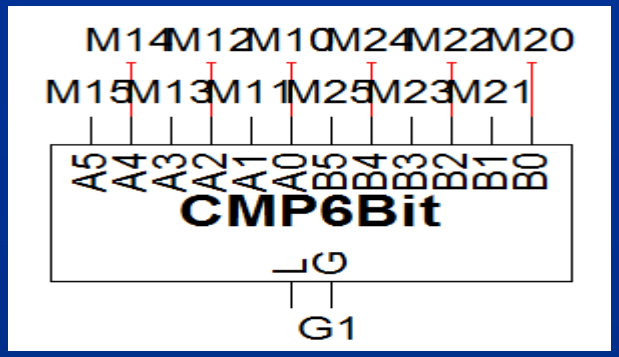


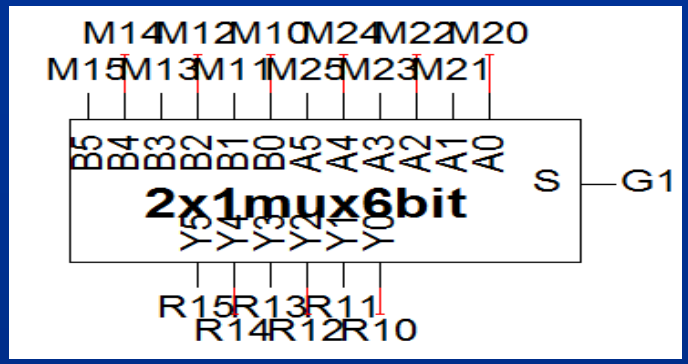


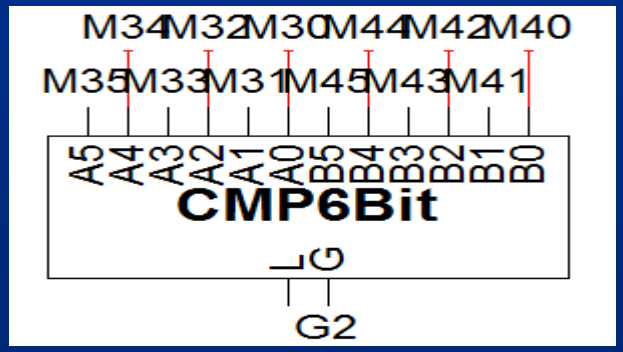


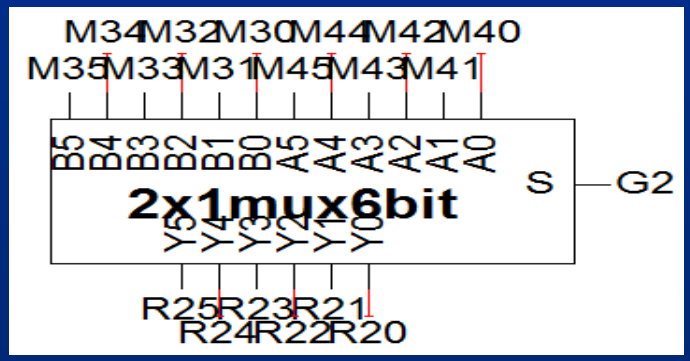


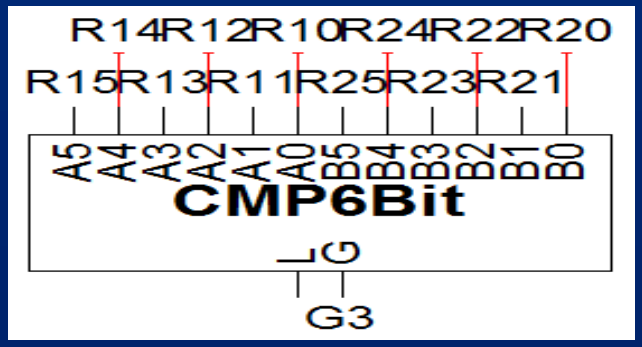
Data Path Design



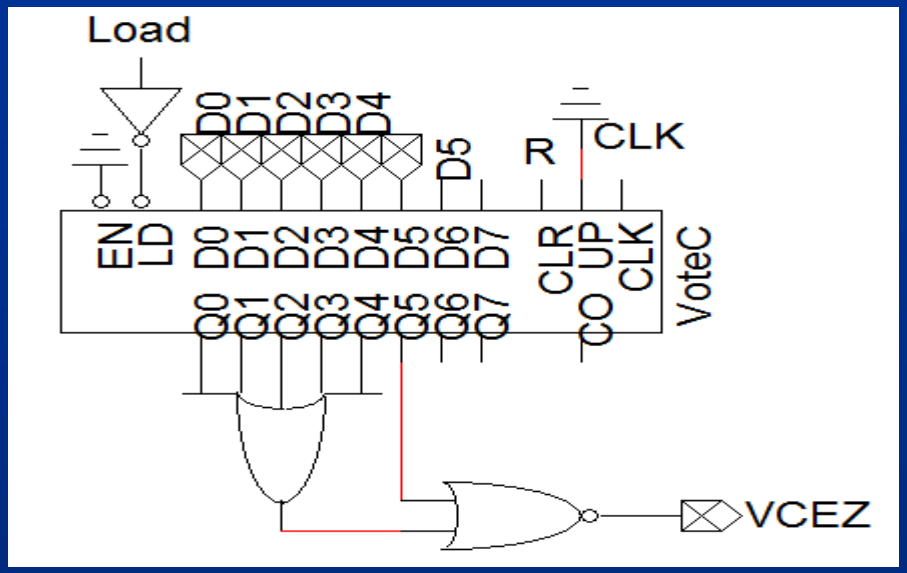


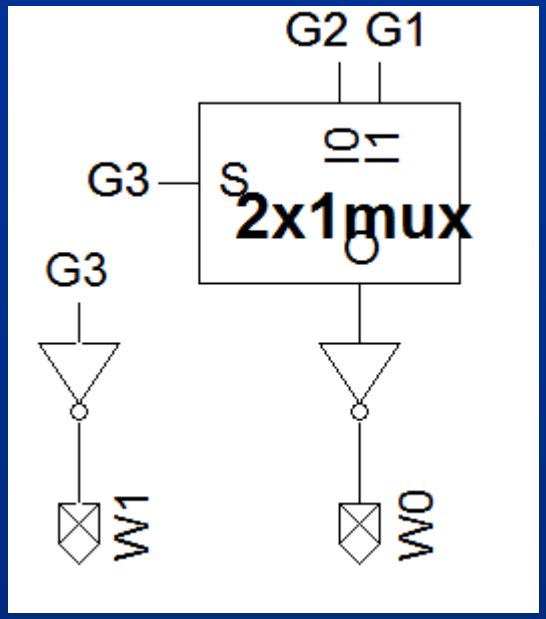


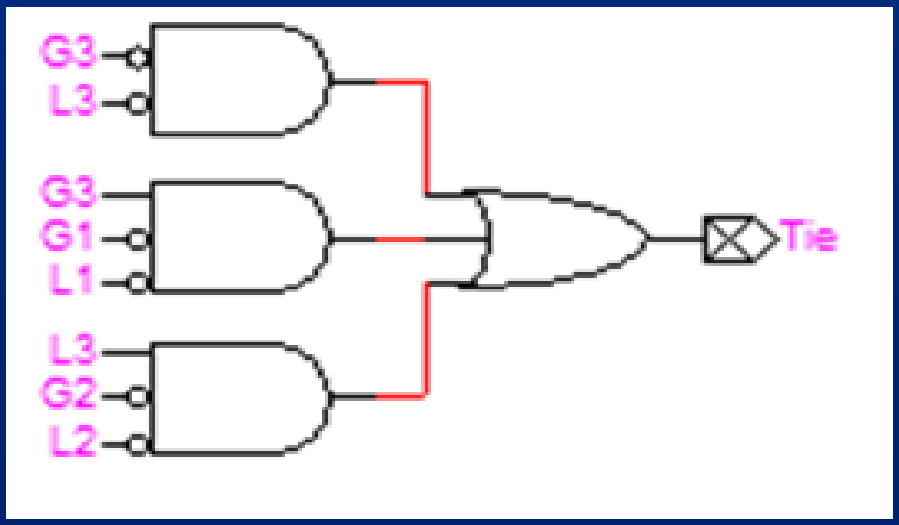




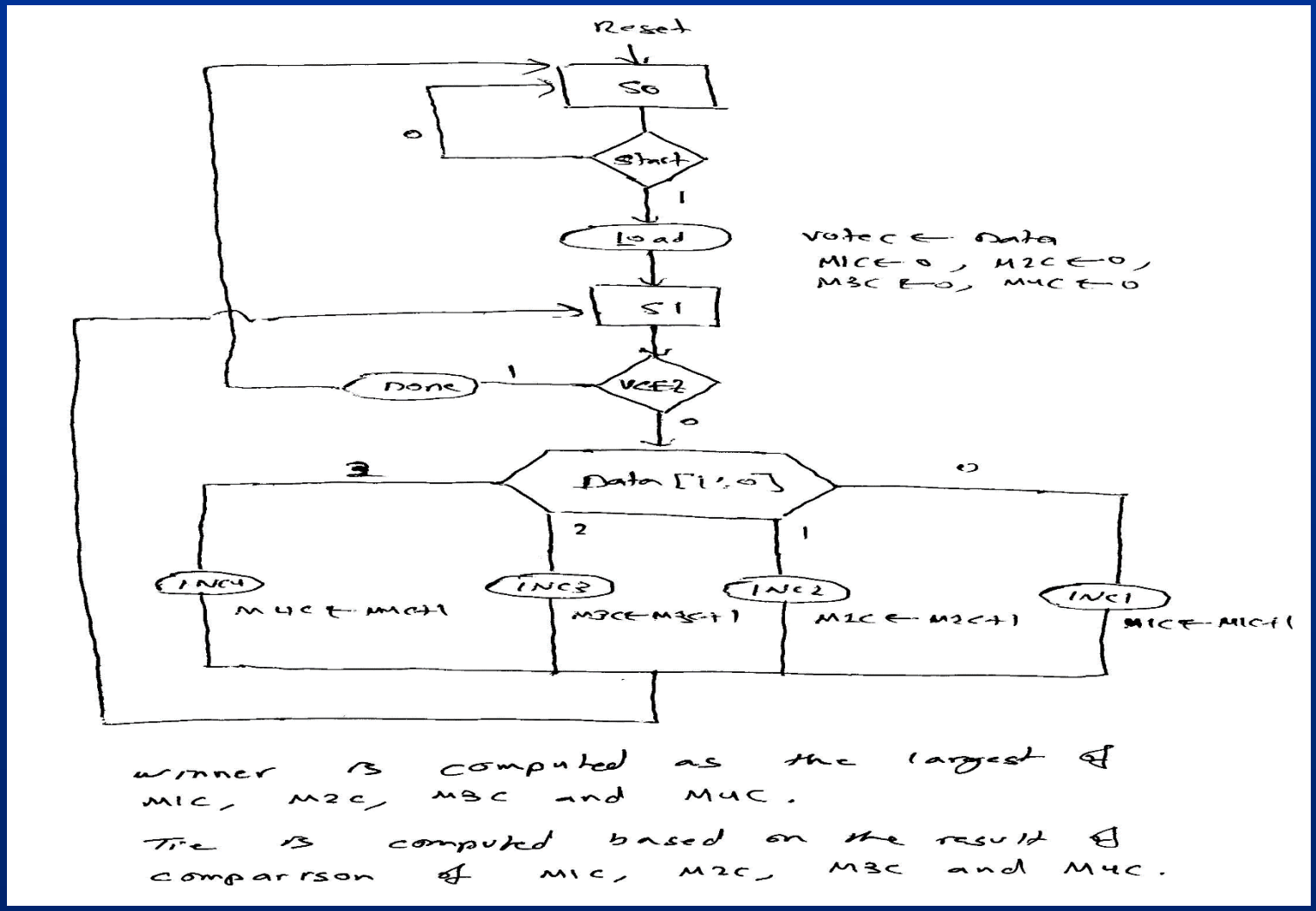
Data Path Design







ASMD Chart

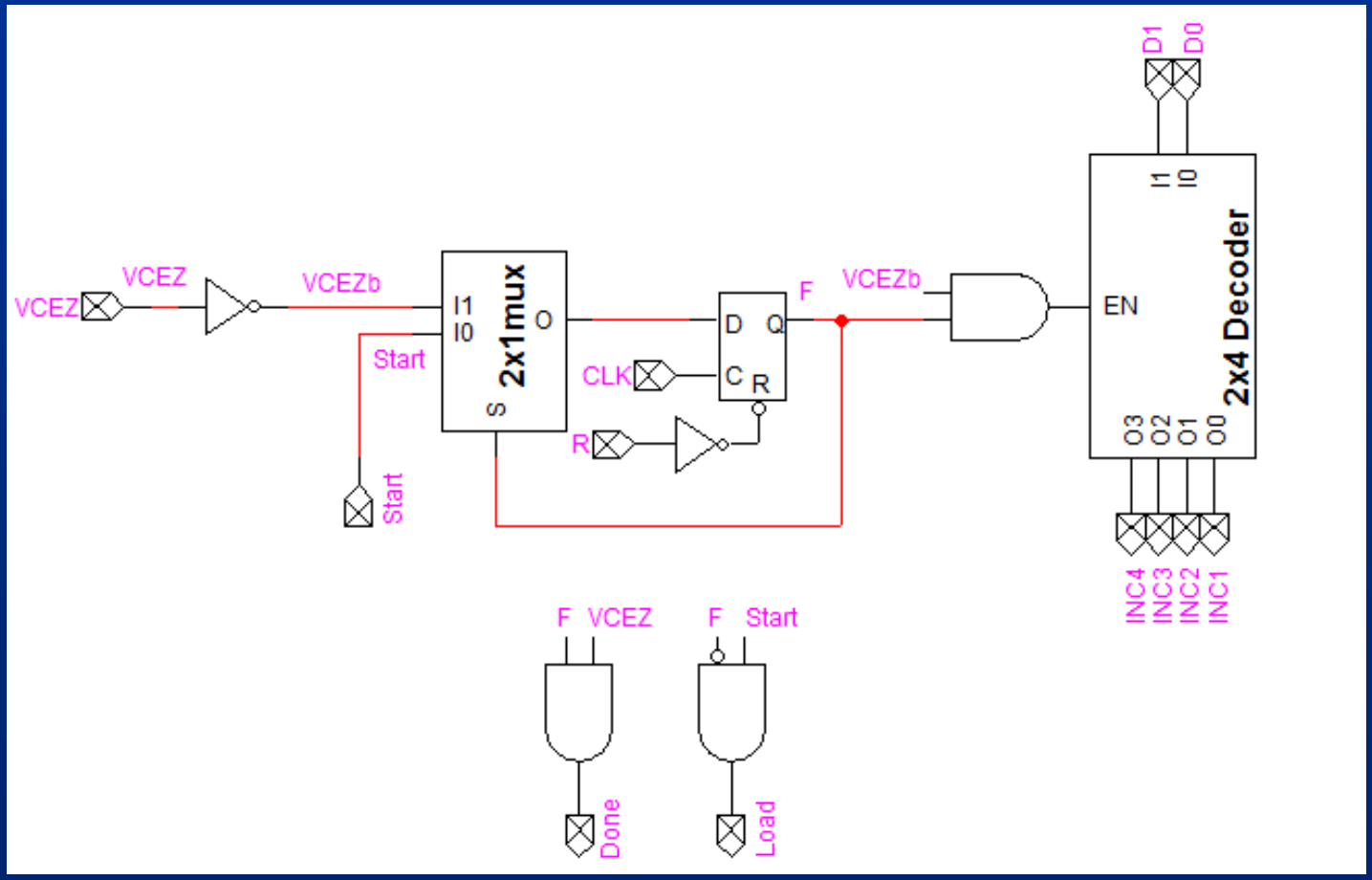

Reset
S0
0
Start
1
Load
votec ← Data
M1C ← 0, M2C ← 0,
M3C ← 0, M4C ← 0
S1
Done
1
VCE=2
0
3
Data[1:0]
0
2
1
INC4
M4C ← M4C+1
INC3
M3C ← M3C+1
INC2
M2C ← M2C+1
INC1
M1C ← M1C+1
winner is computed as the largest of M1C, M2C, M3C and M4C.
Tie is computed based on the result of comparison of M1C, M2C, M3C and M4C.

Control Unit Design

C.S.	Input				N.S.	Output					
	Start	Data[1]	Data[0]	VCEZ		Load	INC1	INC2	INC3	INC4	Done
S0	0	x	x	X	S0	0	0	0	0	0	0
S0	1	x	x	X	S1	1	0	0	0	0	0
S1	x	0	0	0	S1	0	1	0	0	0	0
S1	x	0	1	0	S1	0	0	1	0	0	0
S1	x	1	0	0	S1	0	0	0	1	0	0
S1	x	1	1	0	S1	0	0	0	0	1	0
S1	x	x	x	1	S0	0	0	0	0	0	1

Control Unit Design

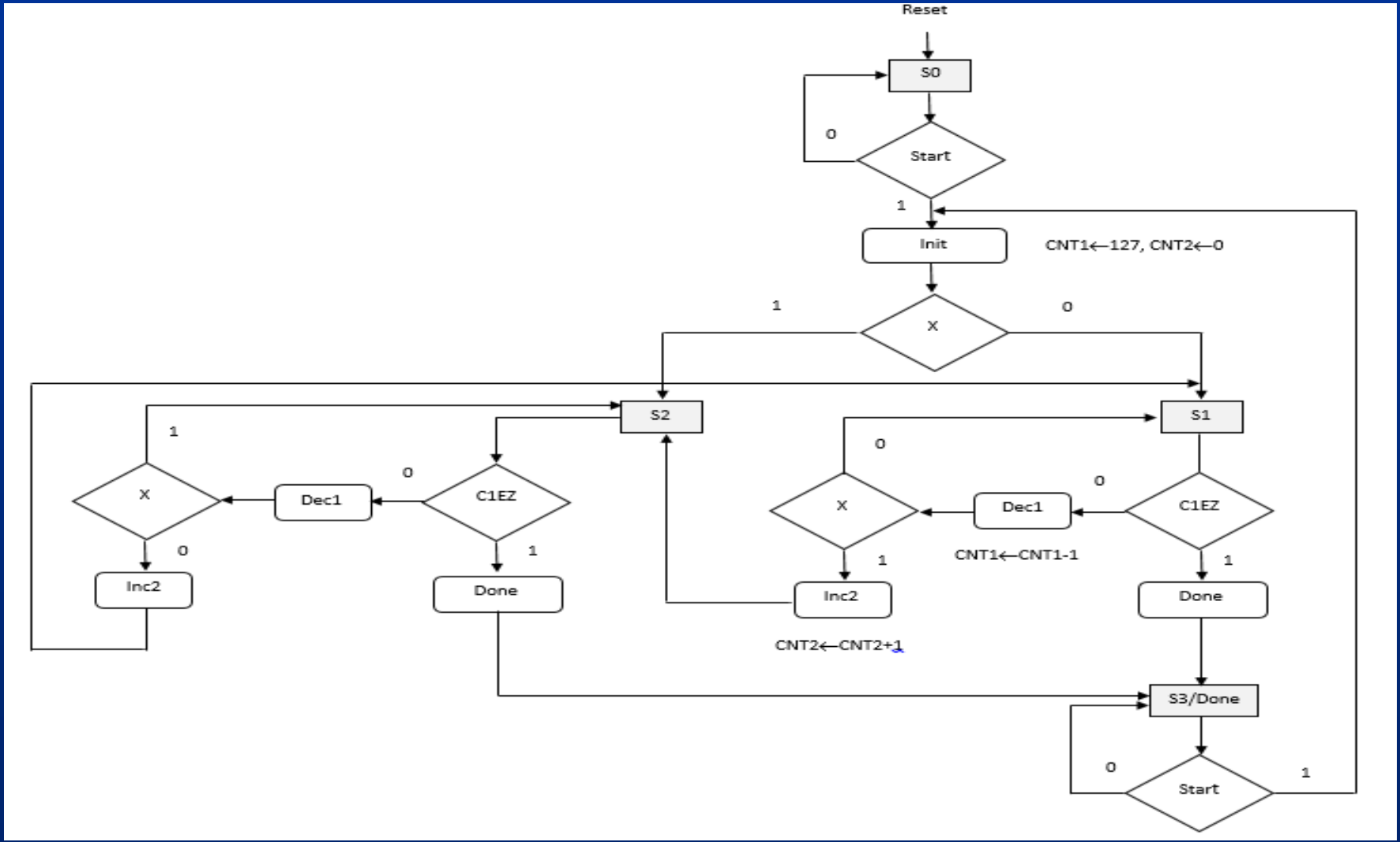
- State Assignment: S0=0, S1=1.



Transition Counting Circuit

- It is required to design a circuit that counts the number of data transitions (i.e. 0→1 and 1→0 data changes) through a stream of 128 bit data.
- The data is applied serially through an input *X* once the user presses a ***Start*** button, where the first bit is transmitted in the same cycle the *Start* button is asserted.
- Once the computation is finished, the machine asserts a ***Done*** signal which remains asserted until the user presses the Start button again or resets the machine.
- Assume that the machine has Asynchronous *Reset* input

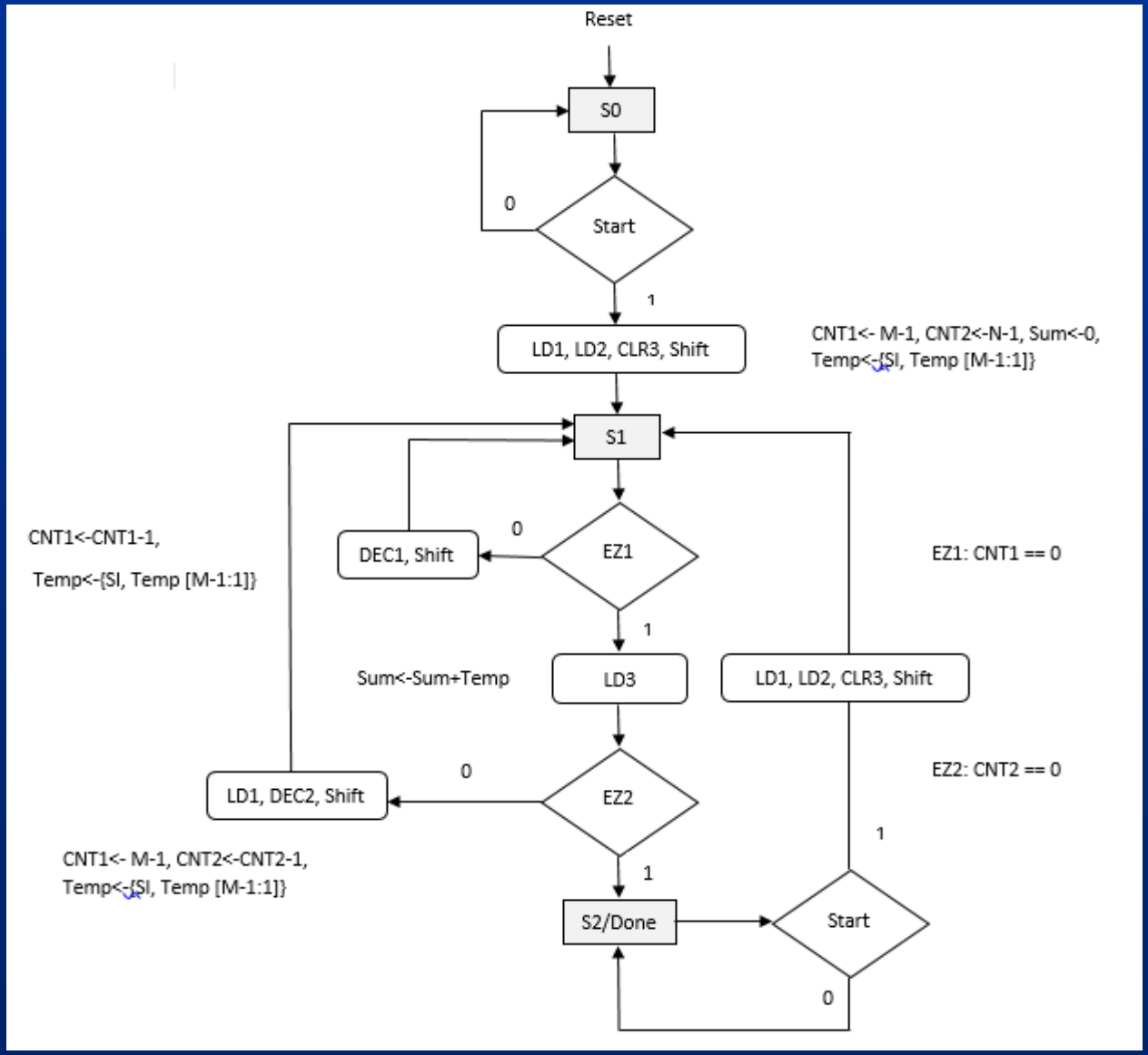
Transition Counting Circuit



Average Computation of Serial Data

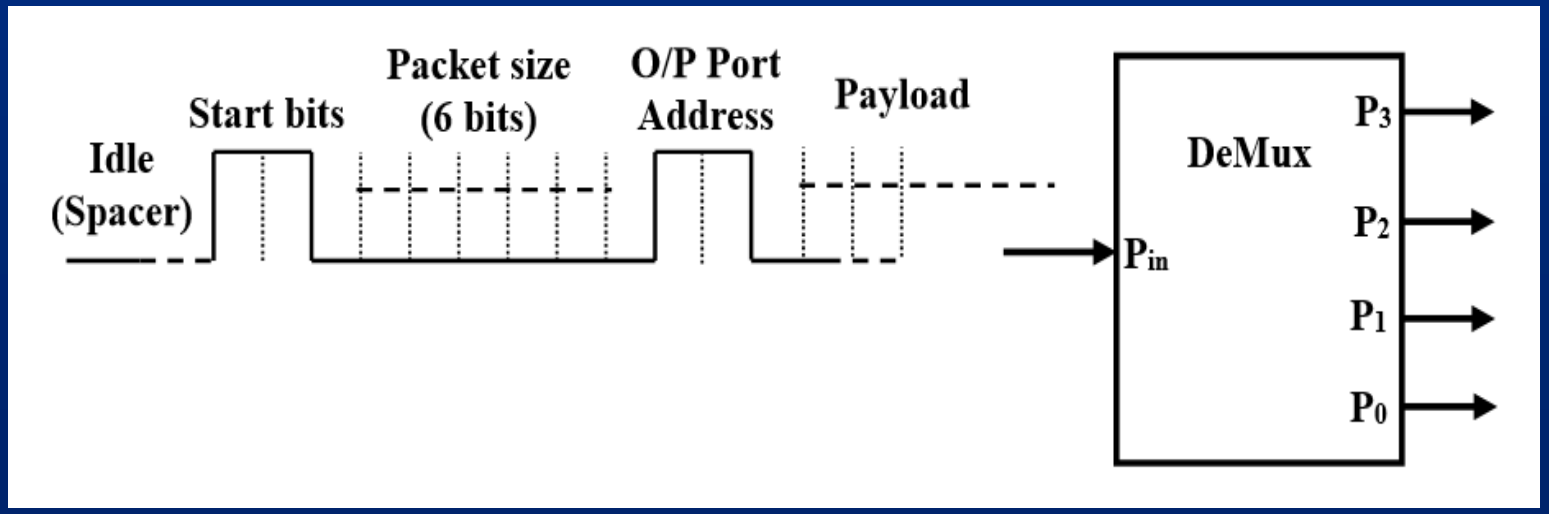
- **It is required to design a circuit that once a Start signal is asserted**
  - Reads N M-bit numbers serially (i.e. starting with first number, it is read 1-bit at a time (LSB to MSB) through a serial input SI, then the 2nd number and so on)
  - The first bit of the first number is applied along with the Start signal
  - Produces their average as integer by dividing their sum by N
  - Asserts a Done signal once the result is ready
- **Determine the number of bits required for the output**
- **Derive the ASMD Chart for the circuit**

Average Computation of Serial Data



A Simple Network DeMux

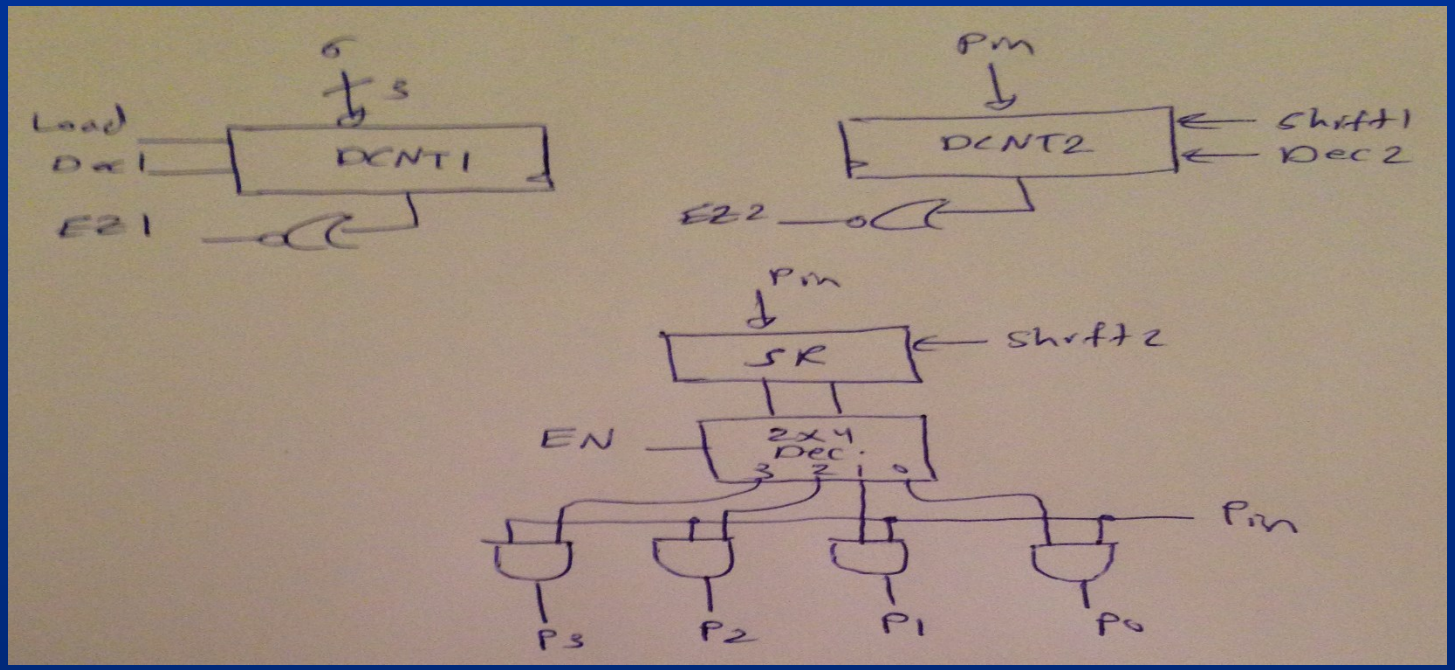
- **It is required to design a simple network DeMux. The circuit has one input and four outputs**
  - Packets have 10-bits headers
  - When input line is idle (i.e. no data is arriving), it is kept low
  - A packet has two start bits (11), followed by 6-bits specifying the packet size in bits (i.e. from 0 to 63 bits), followed by two bits specifying the address of the output port (0 to 3), and immediately followed by the payload (i.e., packet data)

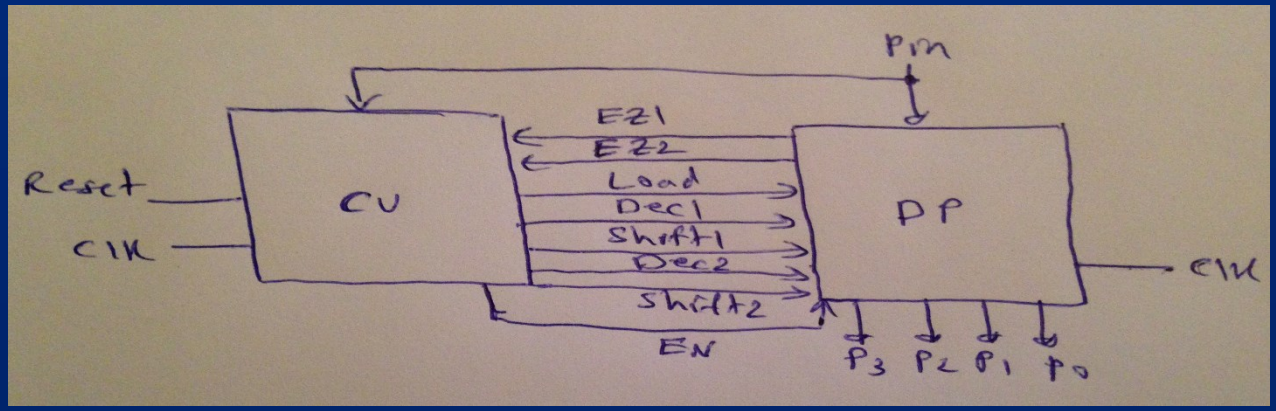


A Simple Network DeMux

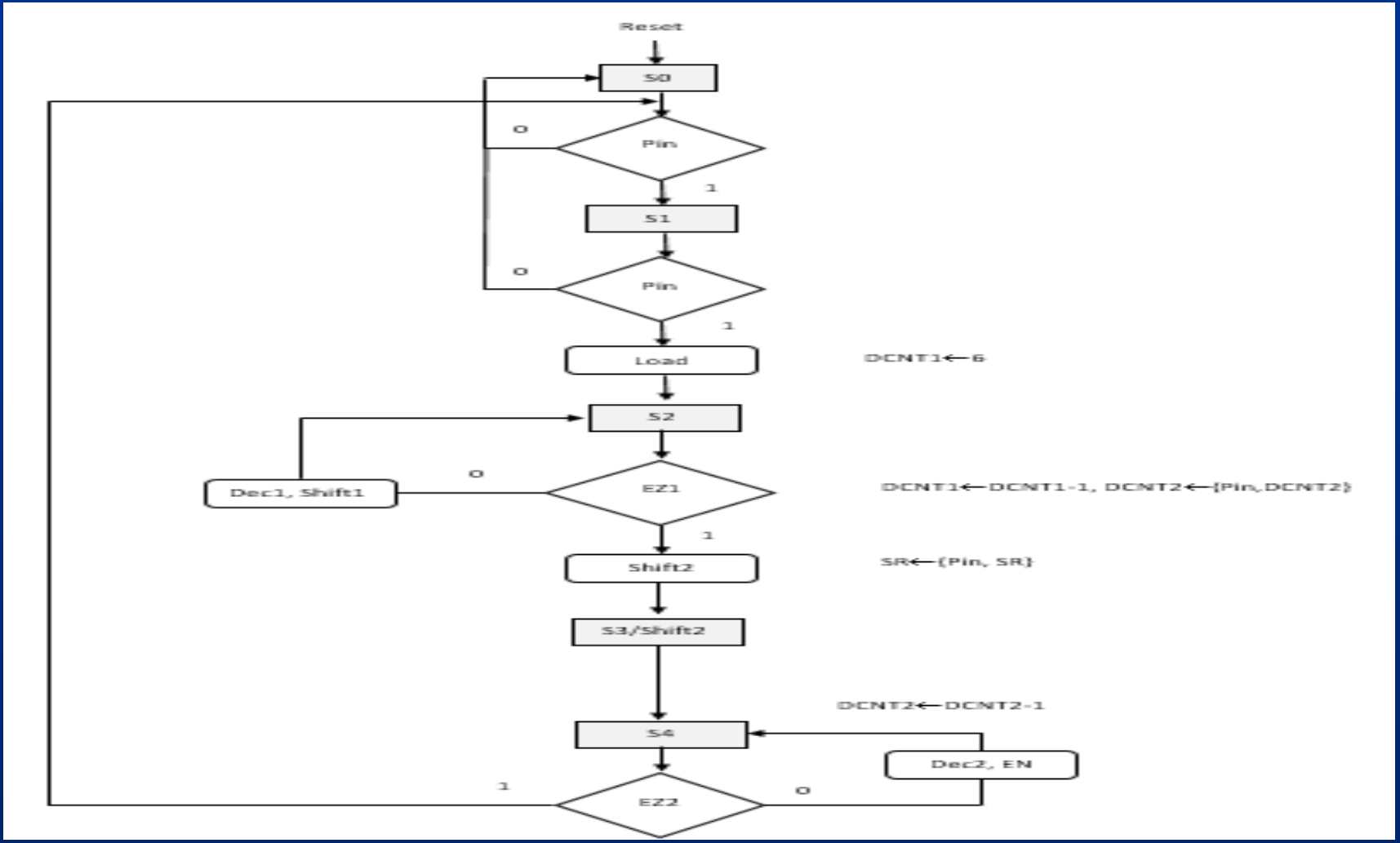
- The DeMux strips out the header before outputting the packet's payload
- Whenever data is not transmitted across any of the output ports, its output line will be 0
- Assume that packet size, port address and data will be transmitted from least significant to most significant bits
- The circuit should be fully synchronous with an external clock and have a master asynchronous reset input

Data Path


6
5
Load
Dec1
DCNT1
EZ1
Pm
DCNT2
Shift1
Dec2
EZ2
Pm
SR
Shift2
EN
2x4
Dec.
3 2 1 0
Pm
P3
P2
P1
P0

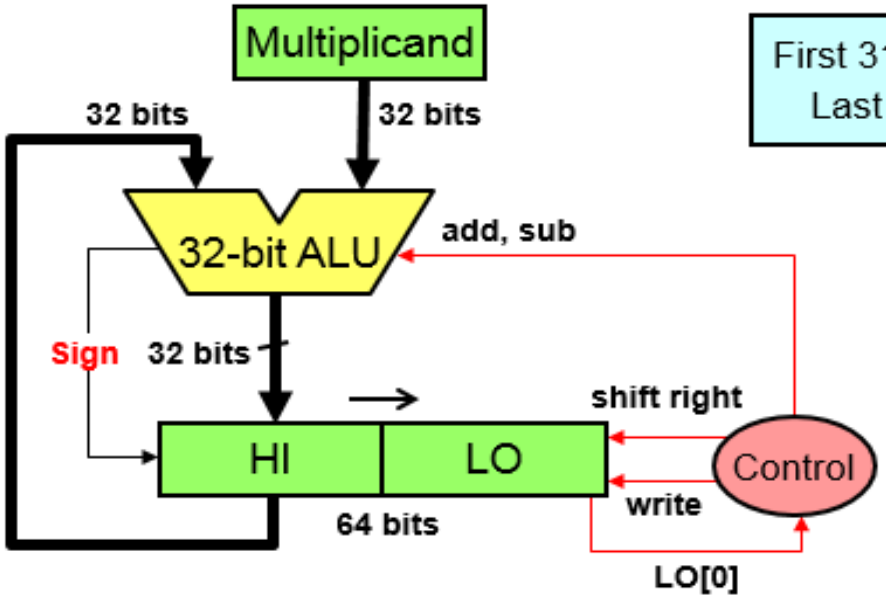

Pm
EZ1
EZ2
Load
Dec1
Shift1
Dec2
Shift2
EN
Reset
CLK
CU
DP
CLK
P3
P2
P1
P0

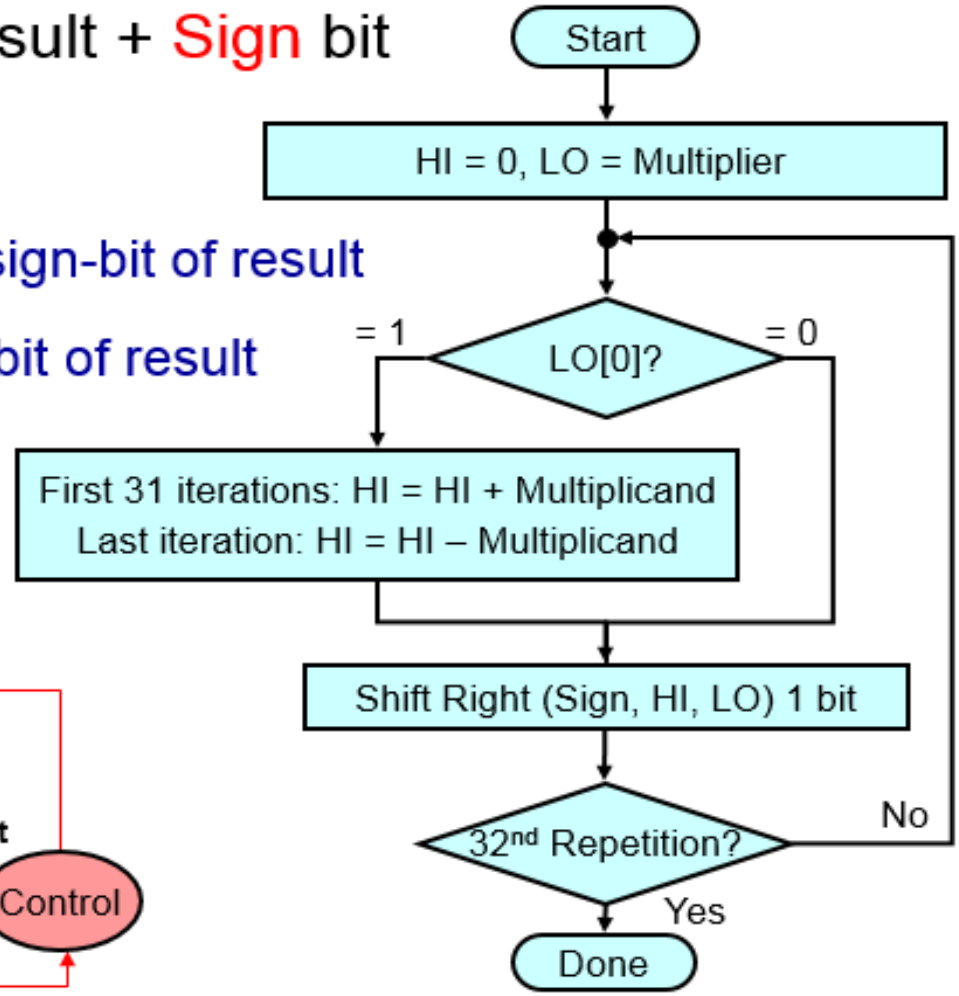
ASMD Chart


Reset
S0
0
Pin
1
S1
0
Pin
1
Load
DCNT1←6
S2
0
Dec1, Shift1
EZ1
DCNT1←DCNT1-1, DCNT2←{Pin,DCNT2}
1
Shift2
SR←{Pin, SR}
S3/Shift2
DCNT2←DCNT2-1
S4
Dec2, EN
1
EZ2
0

Sequential Signed Multiplier

- ALU produces 32-bit result + Sign bit
- Sign bit set as follows:
  - No overflow ➔ Extend sign-bit of result
  - Overflow ➔ Invert sign-bit of result



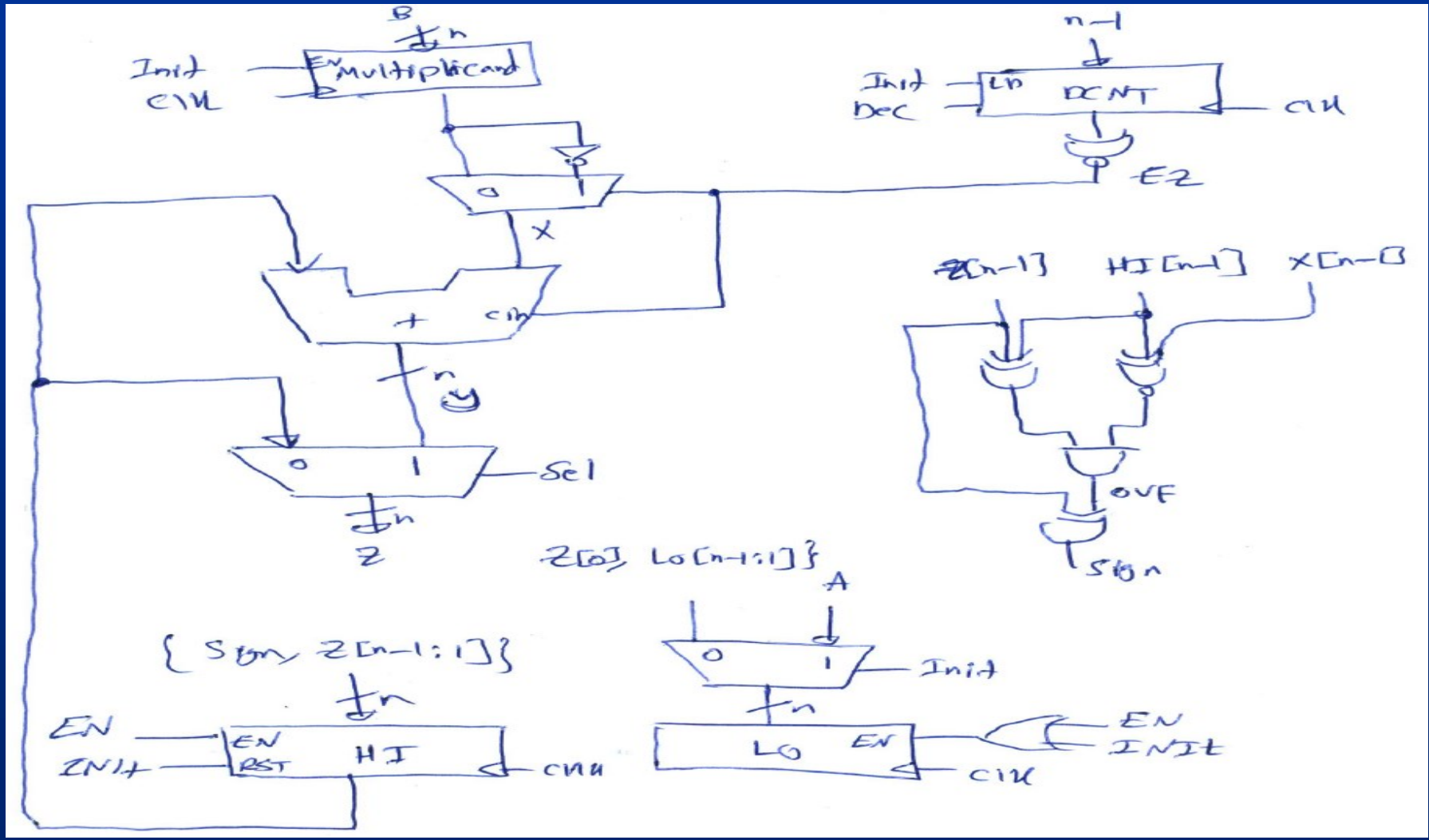


Multiplication Example

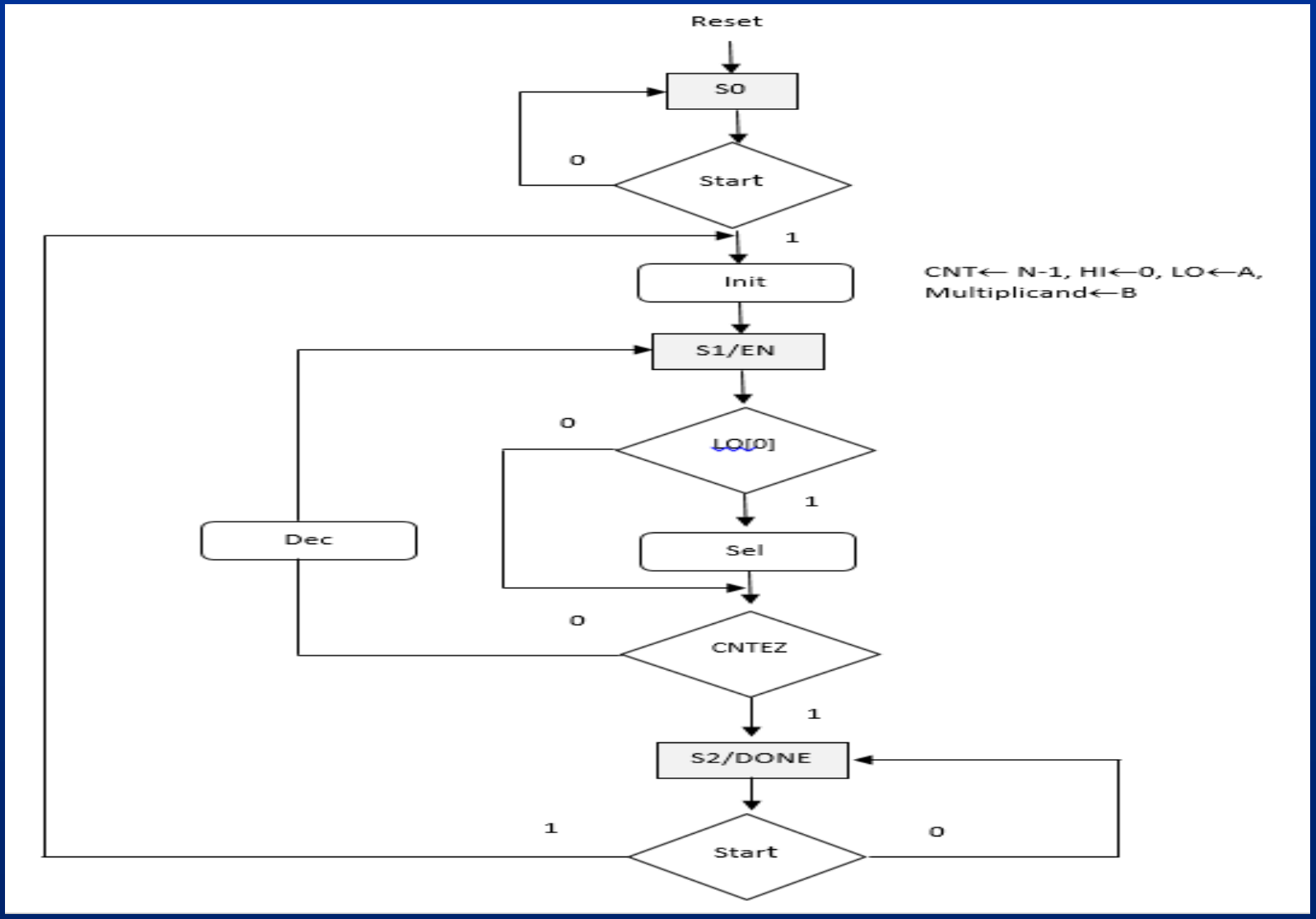
- Consider: 1010_2 (-6) × 1111_2 (-1), Product = 00000110_2 (+6)
- Check for overflow: No overflow ➔ Extend sign bit
- Last iteration: add 2's complement of Multiplicand

Iteration		Multiplicand	Sign	Product = HI, LO
0	Initialize (HI = 0, LO = Multiplier)	1 0 1 0		0 0 0 0 1 1 1 1
1	LO[0] = 1 => ADD		1	1 0 1 0 1 1 1 1
	Shift (Sign, HI, LO) right 1 bit	1 0 1 0		1 1 0 1 0 1 1 1
2	LO[0] = 1 => ADD		1	0 1 1 1 0 1 1 1
	Shift (Sign, HI, LO) right 1 bit	1 0 1 0		1 0 1 1 1 0 1 1
3	LO[0] = 1 => ADD		1	0 1 0 1 1 0 1 1
	Shift (Sign, HI, LO) right 1 bit	1 0 1 0		1 0 1 0 1 1 0 1
4	LO[0] = 1 => SUB (ADD 2's compl)	0 1 1 0	0	0 0 0 0 1 1 0 1
	Shift (Sign, HI, LO) right 1 bit			0 0 0 0 0 1 1 0

Data Path Design

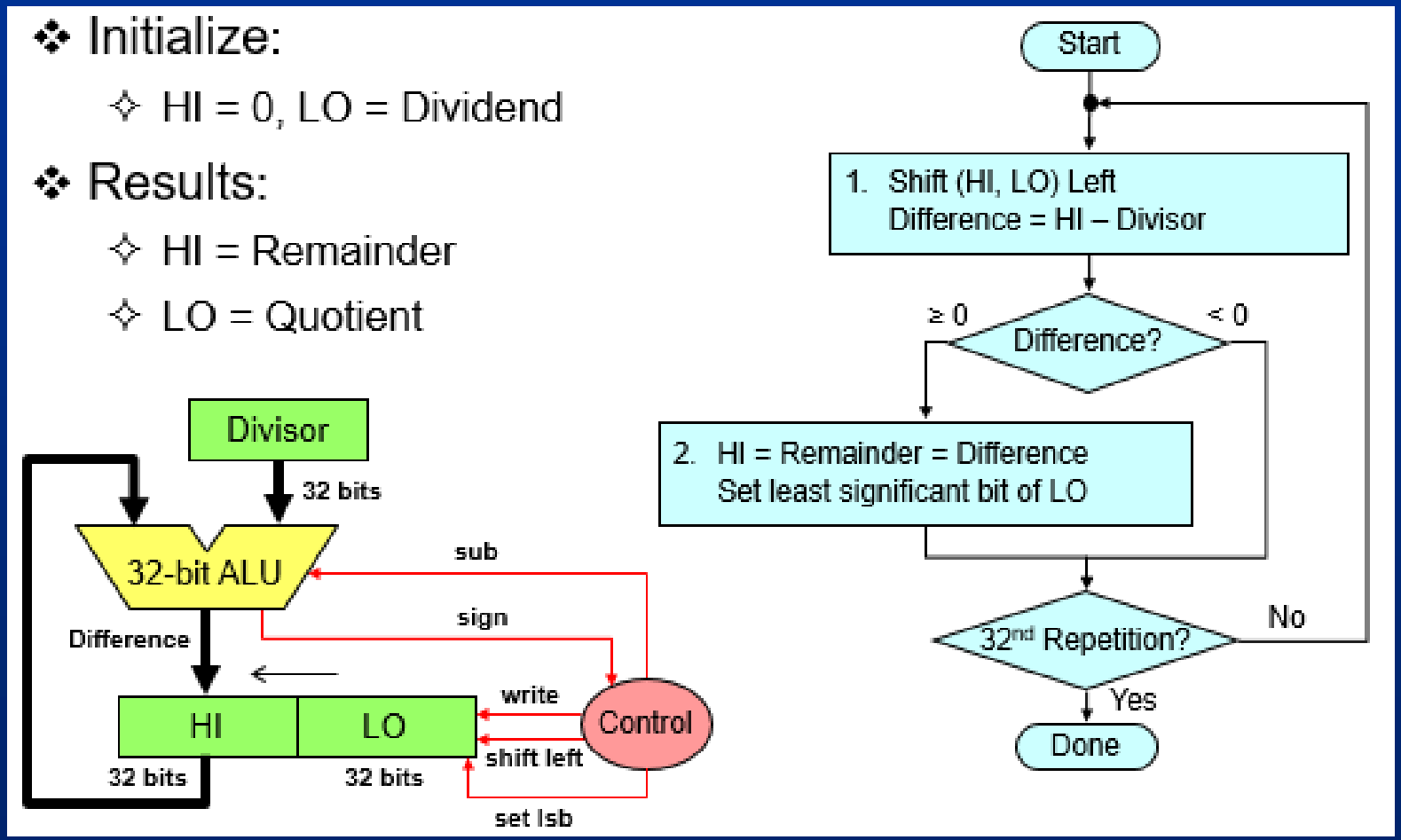


ASMD Chart for Signed Multiplier



Unsigned Divider Design

- **Unsigned Division Algorithm**



Unsigned Divider Design

- Example: 1110_2 / 0011_2 (4-bit dividend & divisor)
- Result Quotient = 0100_2 and Remainder = 0010_2
- 4-bit registers for Remainder and Divisor (4-bit ALU)

Iteration		HI	LO	Divisor	Difference
0	Initialize	0 0 0 0	1 1 1 0	0 0 1 1	
1	1: Shift Left, Diff = HI - Divisor	0 0 0 1 ←	1 1 0 0	0 0 1 1	1 1 1 0
	2: Diff < 0 => Do Nothing				
2	1: Shift Left, Diff = HI - Divisor	0 0 1 1 ←	1 0 0 0	0 0 1 1	0 0 0 0
	2: Rem = Diff, set **lsb** of LO	0 0 0 0	1 0 0 1		
3	1: Shift Left, Diff = HI - Divisor	0 0 0 1 ←	0 0 1 0	0 0 1 1	1 1 1 0
	2: Diff < 0 => Do Nothing				
4	1: Shift Left, Diff = HI - Divisor	0 0 1 0 ←	0 1 0 0	0 0 1 1	1 1 1 1
	2: Diff < 0 => Do Nothing				

ASMD Chart for Unsigned Divider