Verilog HDL: FSMD & ASMD

Pravin Zode

Outline

- Introduction
- Why Finite State Machine with Datapath (FSMD)
- Block diagram
- Components and functions of FSMD
- Comparison with FSM and FSMD
- FSMD Example
- ASMD

What is FSMD ?

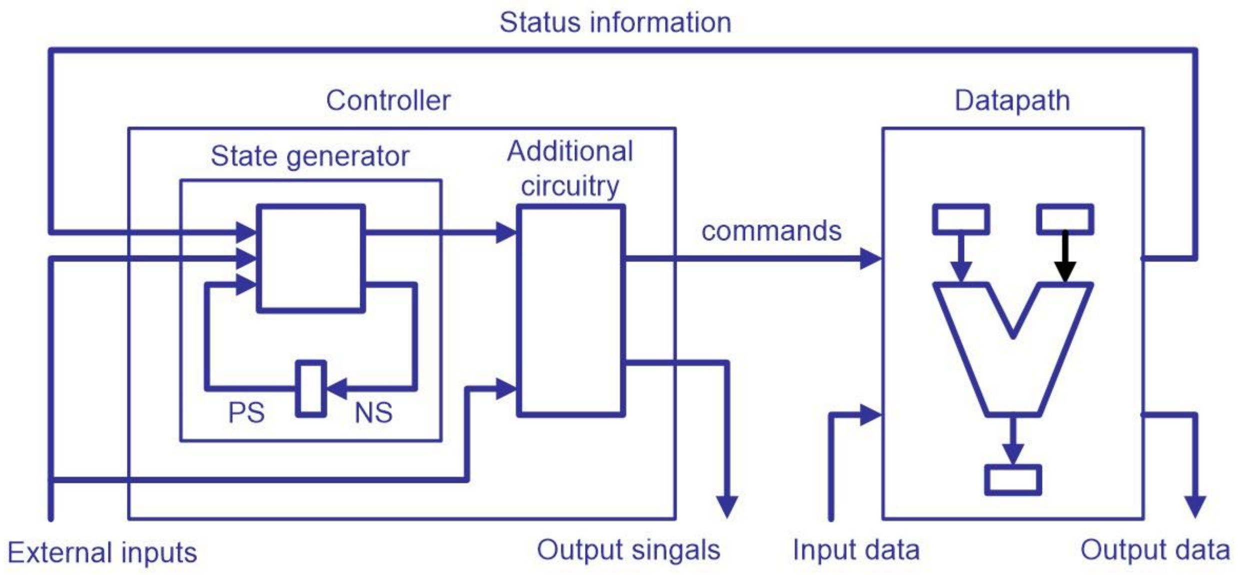
- A Finite State Machine with Datapath (FSMD) is an extension of a traditional FSM that includes a datapath along with the control logic
- Unlike traditional FSMs, FSMD can process arithmetic and logical operations alongside state transitions
- The control unit dictates operations on the datapath based on the current state and inputs
- Suitable for complex digital system designs that require computation and decision-making

Why FSMD ?

- Structured design: Separation of control and data
- Handles sequential operations effectively
- Suitable for hardware implementation (HDLs)
- Wide range of applications.

FSMD Blocks

- Control Unit (FSM): States, transitions, outputs
- Datapath: Registers, ALU, multiplexers, etc
- Control Signals: Arrows from Control Unit to Datapath
- Status Signals: Arrows from Datapath to Control Unit



Components of FSMD & Functions

- **Datapath:** Performs arithmetic, logic, and memory operations using an ALU, registers, multiplexers, and memory units
- **Control Path:** Generates control signals based on state transitions
- **Interaction:** The control unit determines the sequence of operations executed in the datapath

Comparison

Feature	FSM (Finite State Machine)	FSMD (FSM with Datapath)
Definition	Controls system behavior using states and transitions	Includes both control logic and arithmetic operations
Datapath	No arithmetic or logical computations	Contains ALU, registers, and multiplexers
Operations	Simple state transitions	Performs complex arithmetic and logical operations
Control	State-based transitions	State-based transitions + datapath control
Applications	Sequential logic control, vending machines	Digital signal processing, embedded systems
Example	Traffic light controller	FIR filter, encryption processor

Register Transfer (RT) Operation

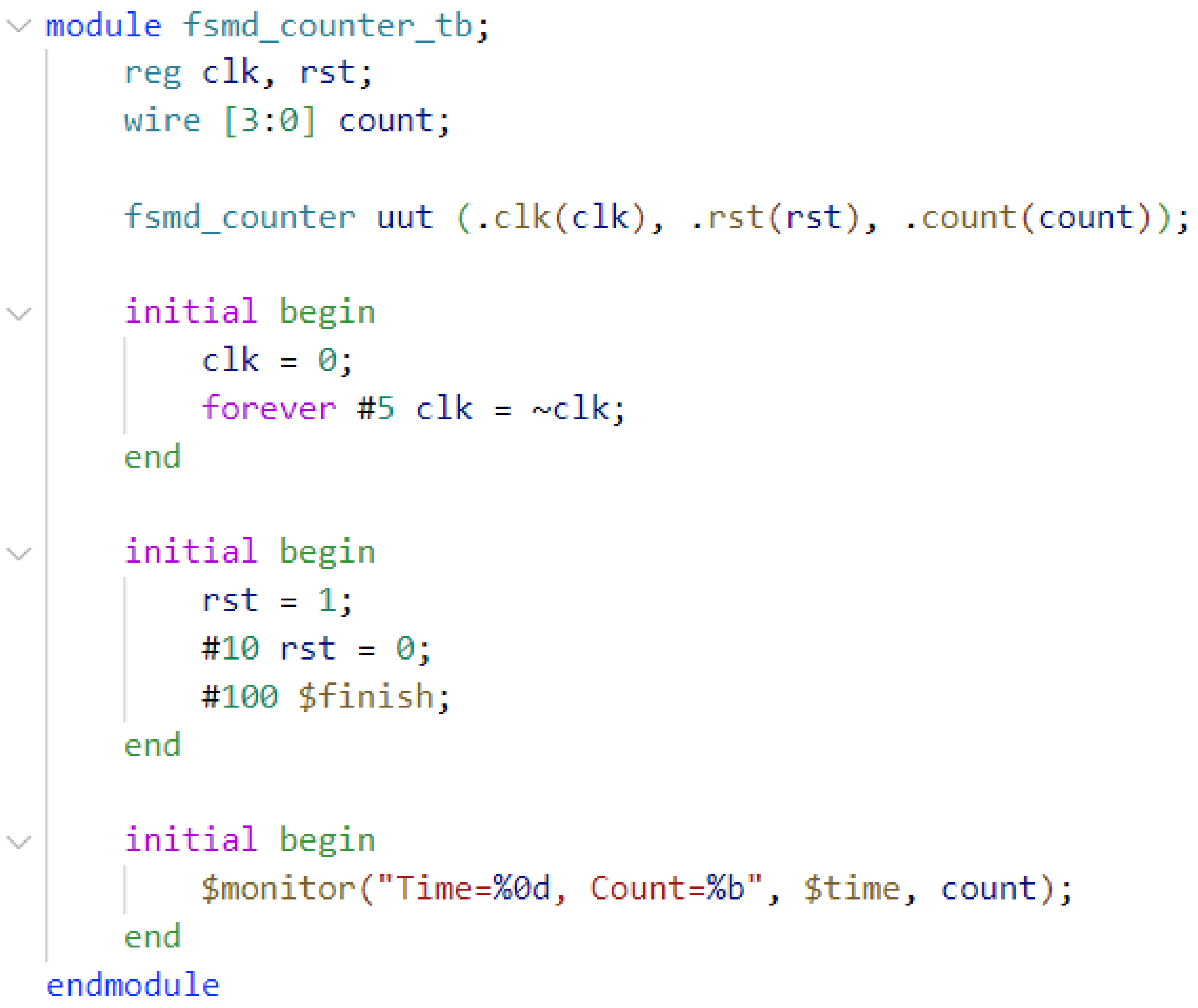
- The movement of data between registers and the operations performed on that data
- Representation:

    Use simple RT notation (e.g., R1 $\leftarrow$ R2 + R3)
- Example:
    - "Load data from memory to register R1“
    - "Add the contents of registers R2 and R3, store the result in R4“
    - "Shift the contents of register R5 left by one bit"

FSMD Example: Counter

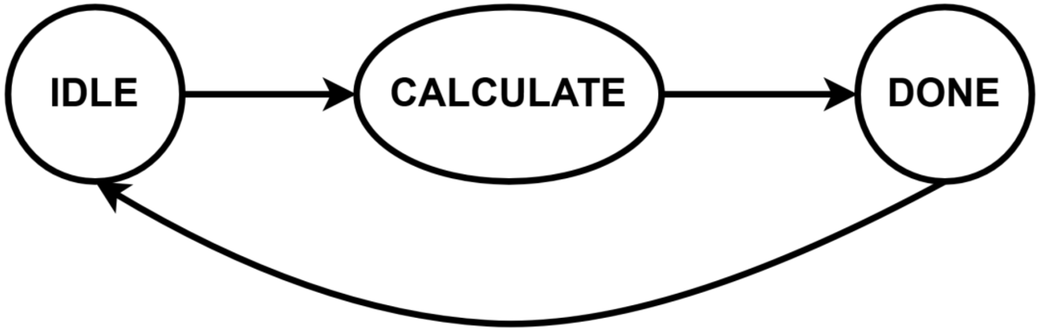
```
module fsmd_counter(
    input clk,
    input rst,
    output reg [3:0] count
);
    reg [1:0] state;
    parameter S0 = 2'b00, S1 = 2'b01;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            state <= S0;
            count <= 4'b0000;
        end else begin
            case (state)
                S0: if (count < 4'b1111) begin count <= count + 1; state <= S1; end
                S1: state <= S0;
            endcase
        end
    end
endmodule
```

FSMD Example: Counter_Testbench



```
module fsmd_counter_tb;
    reg clk, rst;
    wire [3:0] count;

    fsmd_counter uut (.clk(clk), .rst(rst), .count(count));

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        rst = 1;
        #10 rst = 0;
        #100 $finish;
    end

    initial begin
        $monitor("Time=%0d, Count=%b", $time, count);
    end
endmodule
```

FSMD Example: Adder_Subtractor



IDLE:

- Waits for a valid operation (00 or 01)
- Transitions to CALCULATE

CALCULATE:

- Performs the add/sub and stores in **temp_result**
- Moves to DONE

DONE:

- Transfers **temp_result** to final output result.
- Goes back to IDLE.

FSMD Code Breakdown: Adder_Subtractor

State Definitions

```
parameter IDLE = 2'b00;
parameter CALCULATE = 2'b01;
parameter DONE = 2'b10;
```

Control Unit

State Register (Clocked always block)

```
always @(posedge clk)
```

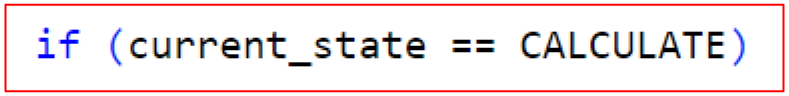
Updates current_state on every clock cycle
Resets to IDLE when reset = 1

Next State Logic

```
always @(*) begin
  case (current_state)
    IDLE:       → go to CALCULATE if operation is valid
    CALCULATE: → go to DONE
    DONE:       → go back to IDLE
  endcase
end
```

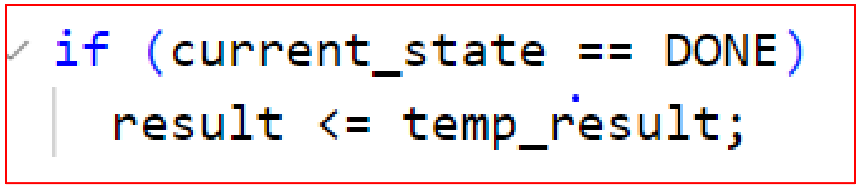
FSMD Code Breakdown: Adder_Subtractor

Datapath (Operations block)s



```
if (current_state == CALCULATE)
```

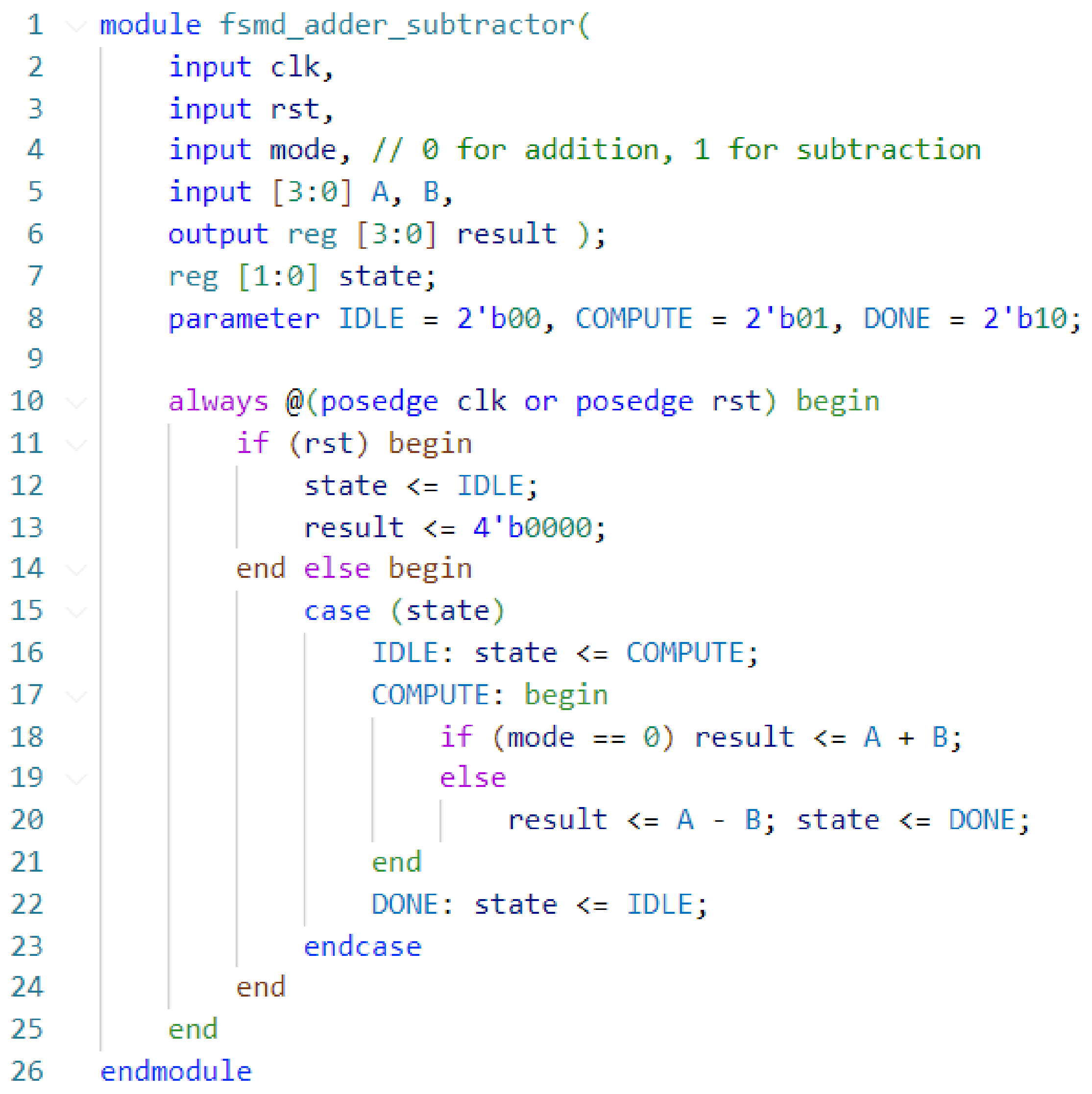
- ✓ Does the math (a + b or a - b)
- ✓ Stores in temp_result (a 9-bit register)



```
if (current_state == DONE)
  result <= temp_result;
```

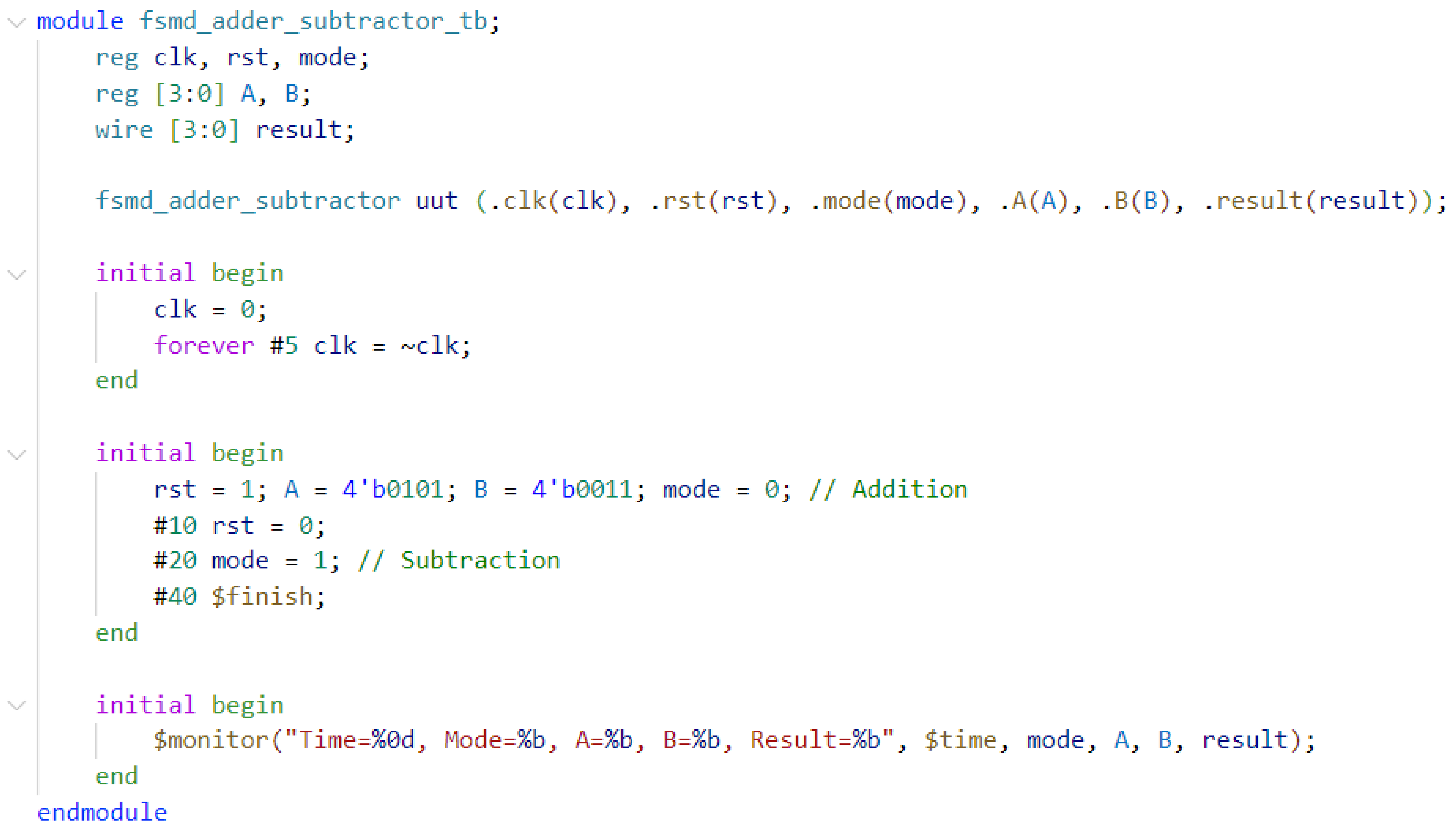
- ✓ On DONE state, updates the output result.

FSMD Example: Adder_Subtractor



```verilog
module fsmd_adder_subtractor(
    input clk,
    input rst,
    input mode, // 0 for addition, 1 for subtraction
    input [3:0] A, B,
    output reg [3:0] result );
    reg [1:0] state;
    parameter IDLE = 2'b00, COMPUTE = 2'b01, DONE = 2'b10;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            state <= IDLE;
            result <= 4'b0000;
        end else begin
            case (state)
                IDLE: state <= COMPUTE;
                COMPUTE: begin
                    if (mode == 0) result <= A + B;
                    else
                        result <= A - B; state <= DONE;
                end
                DONE: state <= IDLE;
            endcase
        end
    end
endmodule
```

FSMD Example: Adder_Subtractor_Testbench



```verilog
module fsmd_adder_subtractor_tb;
    reg clk, rst, mode;
    reg [3:0] A, B;
    wire [3:0] result;

    fsmd_adder_subtractor uut (.clk(clk), .rst(rst), .mode(mode), .A(A), .B(B), .result(result));

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        rst = 1; A = 4'b0101; B = 4'b0011; mode = 0; // Addition
        #10 rst = 0;
        #20 mode = 1; // Subtraction
        #40 $finish;
    end

    initial begin
        $monitor("Time=%0d, Mode=%b, A=%b, B=%b, Result=%b", $time, mode, A, B, result);
    end
endmodule
```

Summary

- FSMD provides a structured approach for digital design
- Used in high-performance computing and embedded systems
- FSM enables step-by-step control of operations
- Datapath performs actual computation
- FSM + Datapath = complete control + computation system

Verilog HDL:
ASMD

Introduction

- ASMD is graphical representation of a Finite State Machine with Datapath (FSMD)
- Describe and visualize the sequential operations of a digital system
- Bridges the gap between algorithmic description and hardware implementation
- Simplifies FSMD design, verification, and documentation

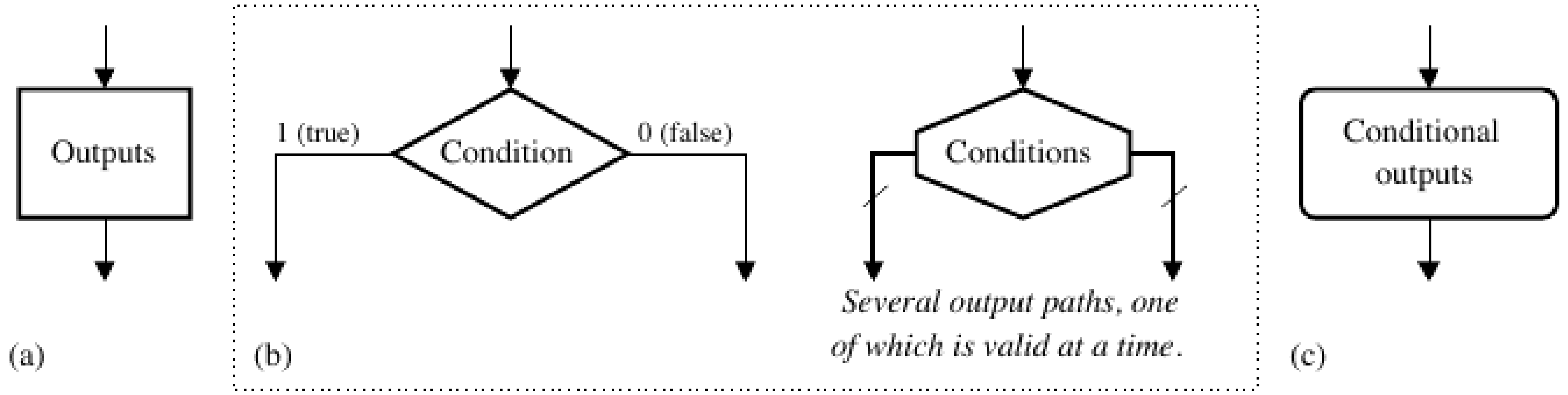
Why use ASMD Charts?

- Provides a clear visual representation of control flow and data operations
- Helps organize complex sequential algorithms
- Facilitates the design of both control and Datapath components
- Improves communication between designers

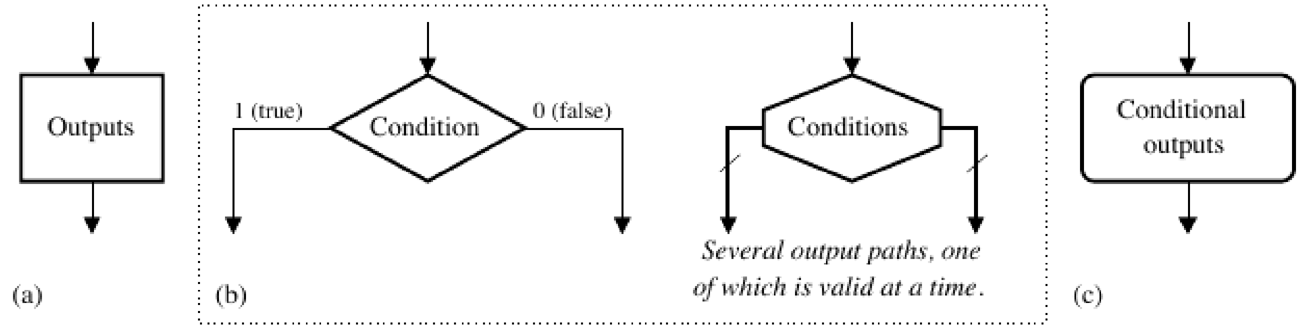
ASM Symbols

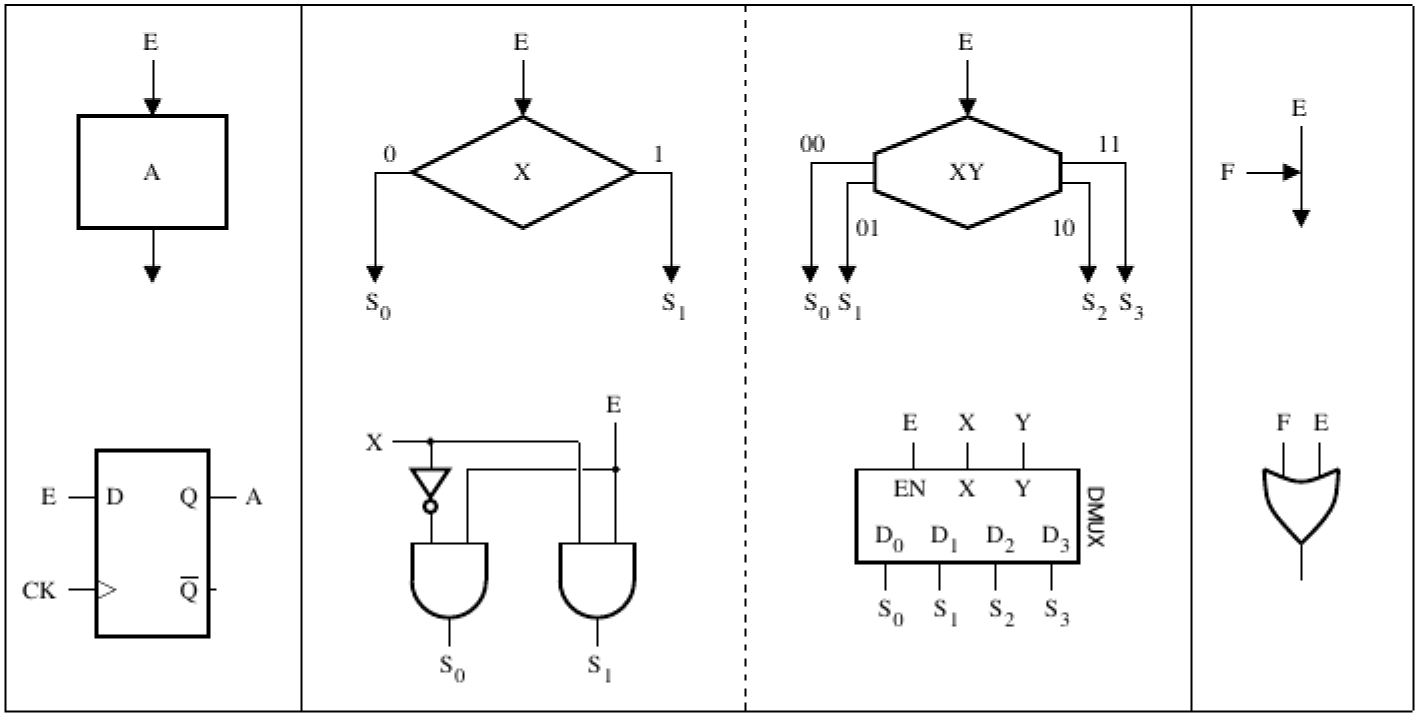
Three types of symbols used in ASM charts are as follows:

- Rectangle is used to represent outputs not dependent on input conditions, such as the flip-flop outputs
- Diamond or hexagon is associated with conditional execution of operations; for n inputs, 2n branches are possible
- Rectangle with rounded corners is used to yield the conditional outputs (dependent on inputs)

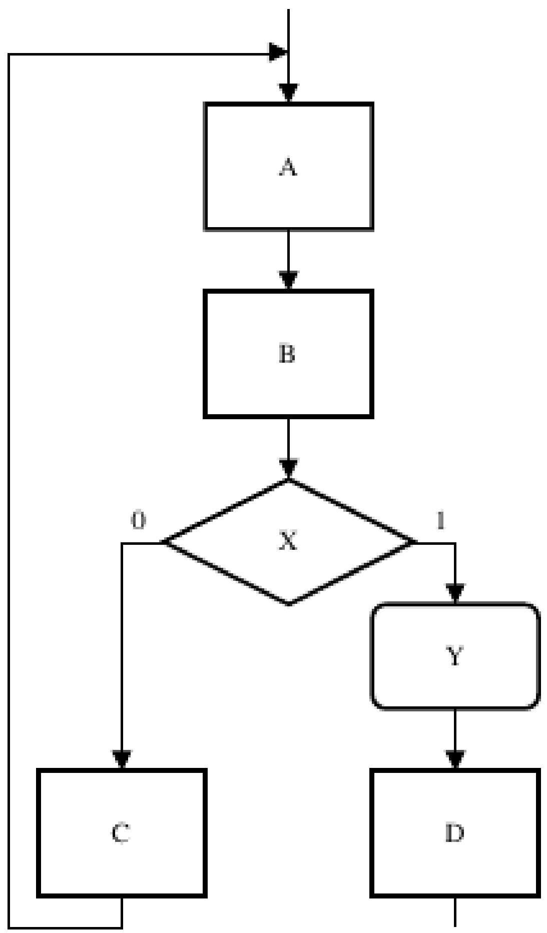


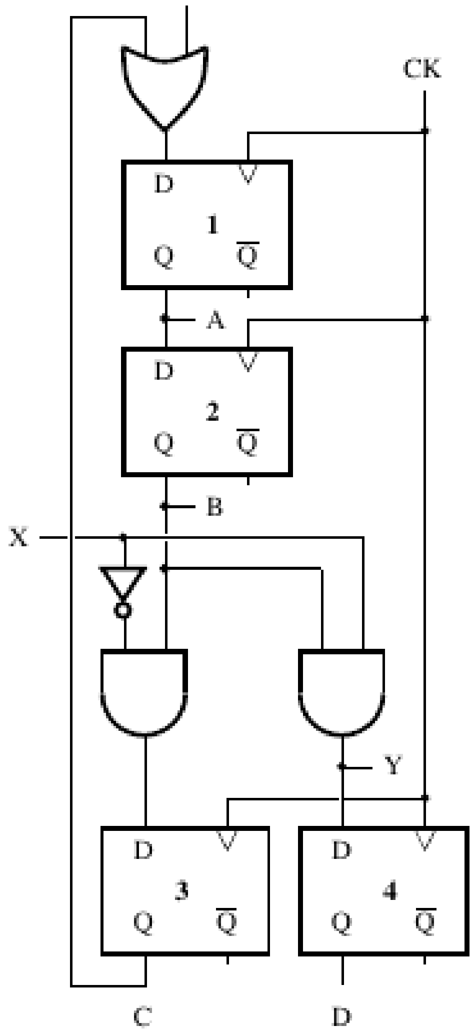
Circuits & ASM Symbols





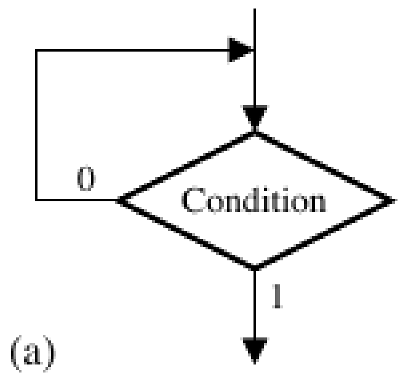
ASM chart and logic circuits

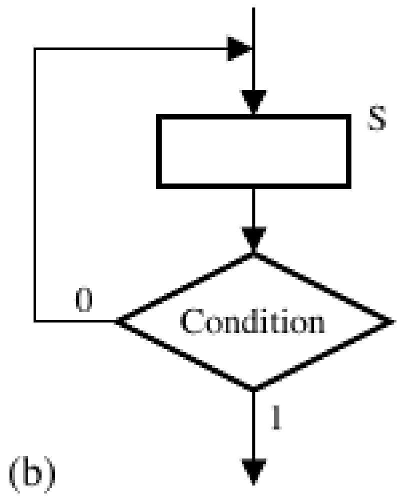




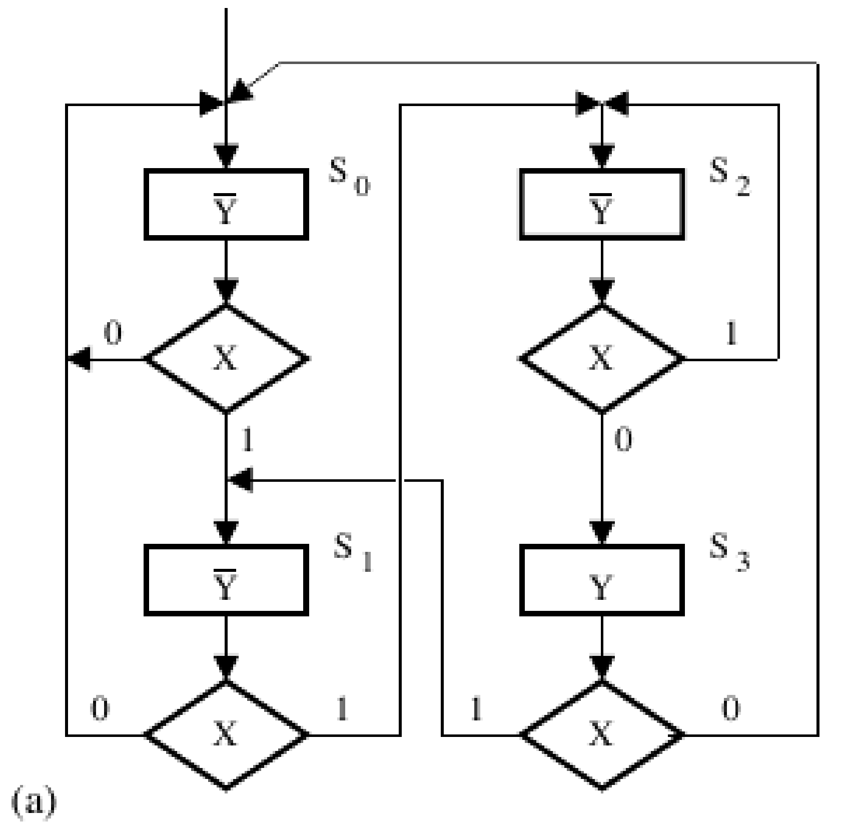
ASM chart and logic circuits

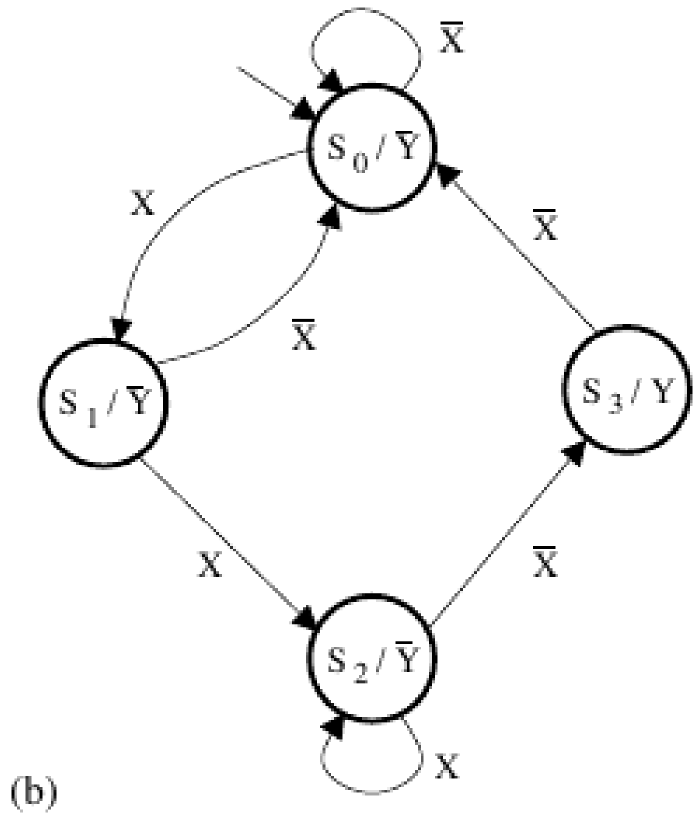
Caution needs to be exercised to avoid races; figure (a), below, admits races and should be avoided



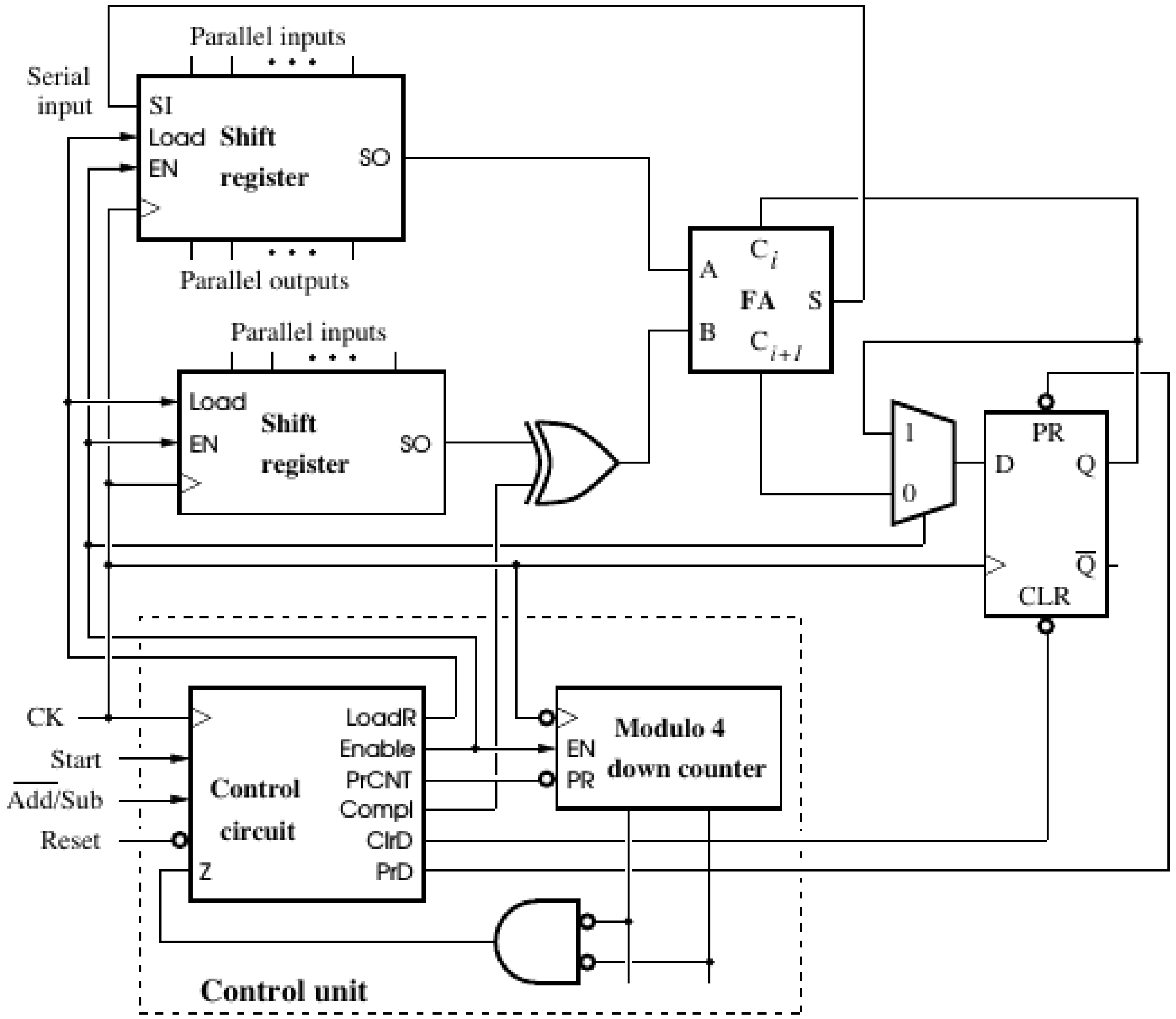


ASM chart and Moore m/c

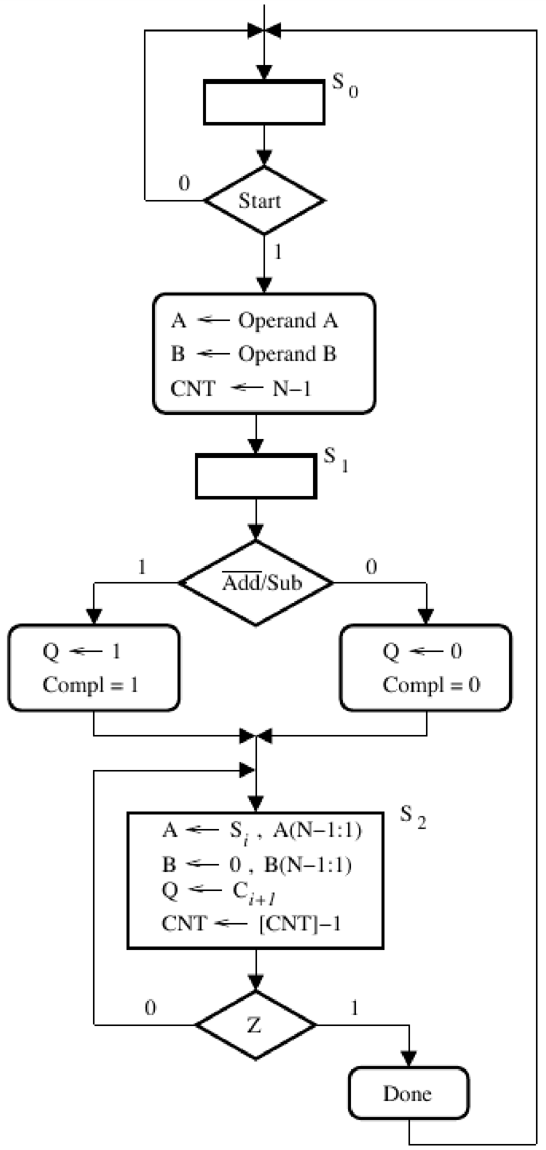




Example: Serial adder



Serial adder ASM chart



Thank you !

Happy Learning