University of Leicester

Programmable Electronics using FPGAs

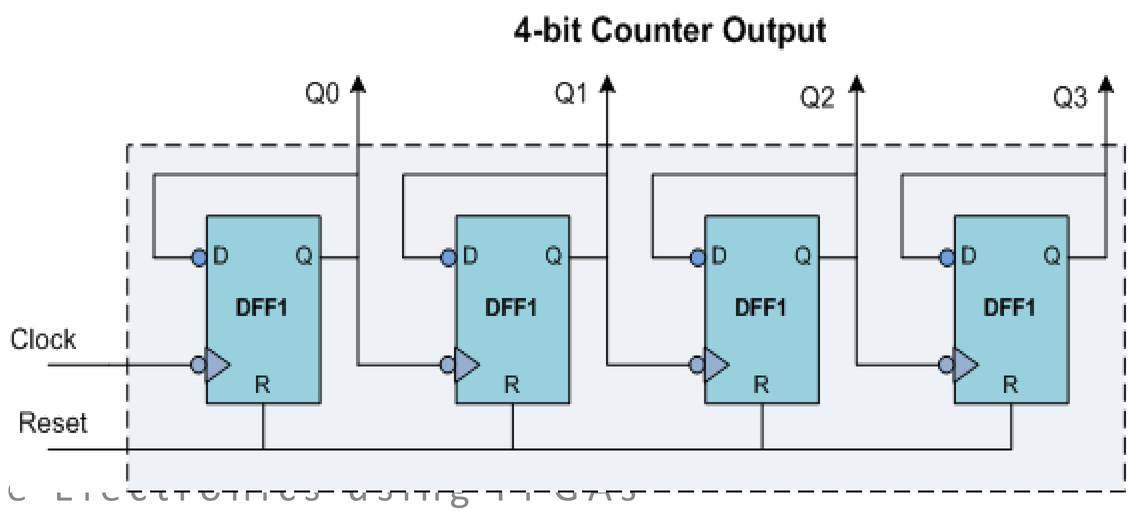
This week

- Recap: Sequential logic
- Finite state machine
- Mealy state machine
- Vending machine controller
- Assessment 2

Recap: Sequential logic

- Interconnection of Flip-Flops and Gates.
- Outputs depend on inputs and entire/some history of execution.
- It is based on:
  - Current state of input signal
  - Current state (stored in memory element) of the system
  - Next state of the system

For example :
A ripple carry counter



Finite state machine

- A finite state machine (FSM) is an efficient way to design sequential logic using a finite number of states.
  - System typically has only a limited number of unique configurations – known as states
  - For example a traffic light controller sequences (*GREEN*, *AMBER*, and *RED*) forever through fixed number of states
- We need a memory element to remember the current state of a system.
- The outputs and new state of a system is a function of both the inputs, and the current state.

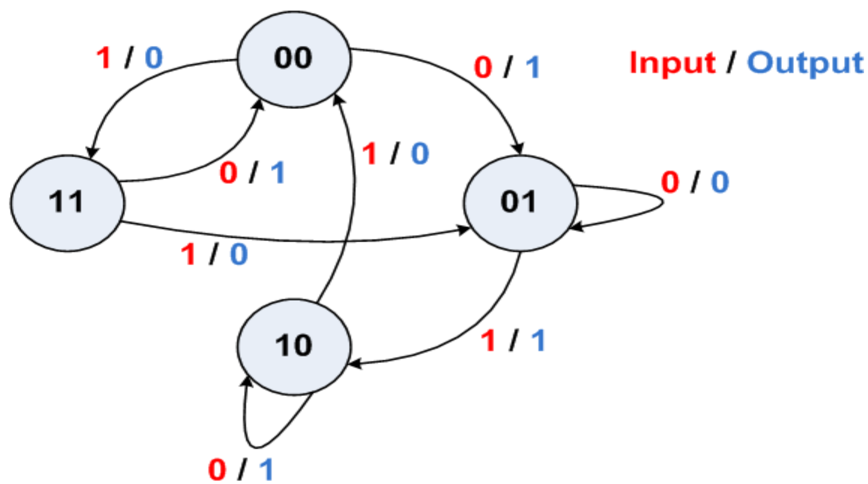
Finite state machine

University of Leicester

- A state transition table is used to express the relationship between inputs, outputs, current state, and next state of a system.

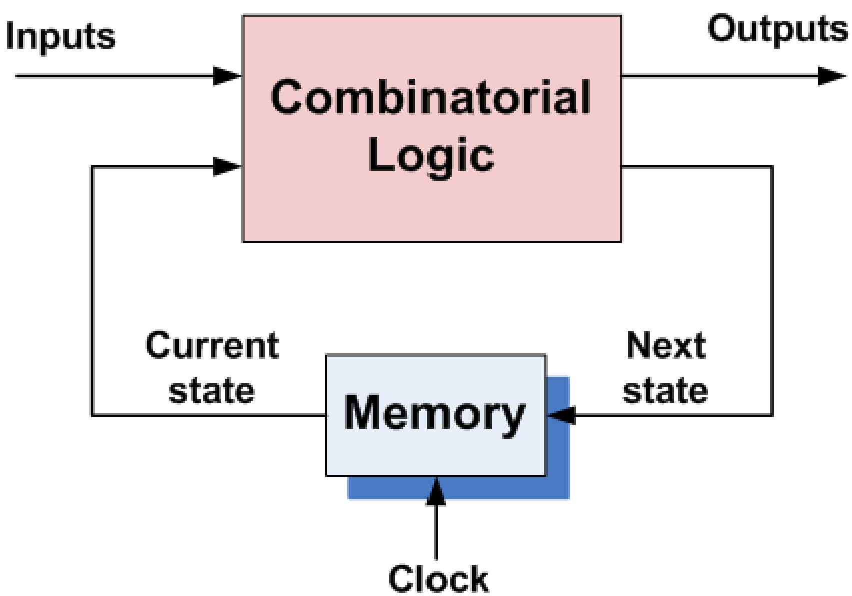
Current state Q1 Q2	Next state X = 0	Next state X = 1	Output X = 0	Output X = 1
0 0	0 1	1 1	1	0
0 1	0 1	1 0	0	1
1 0	1 0	0 0	1	0
1 1	0 0	0 1	1	0

- A state diagram is a graphical way to represent the state transition table. Various styles are adopted for state diagrams.



Finite state machine

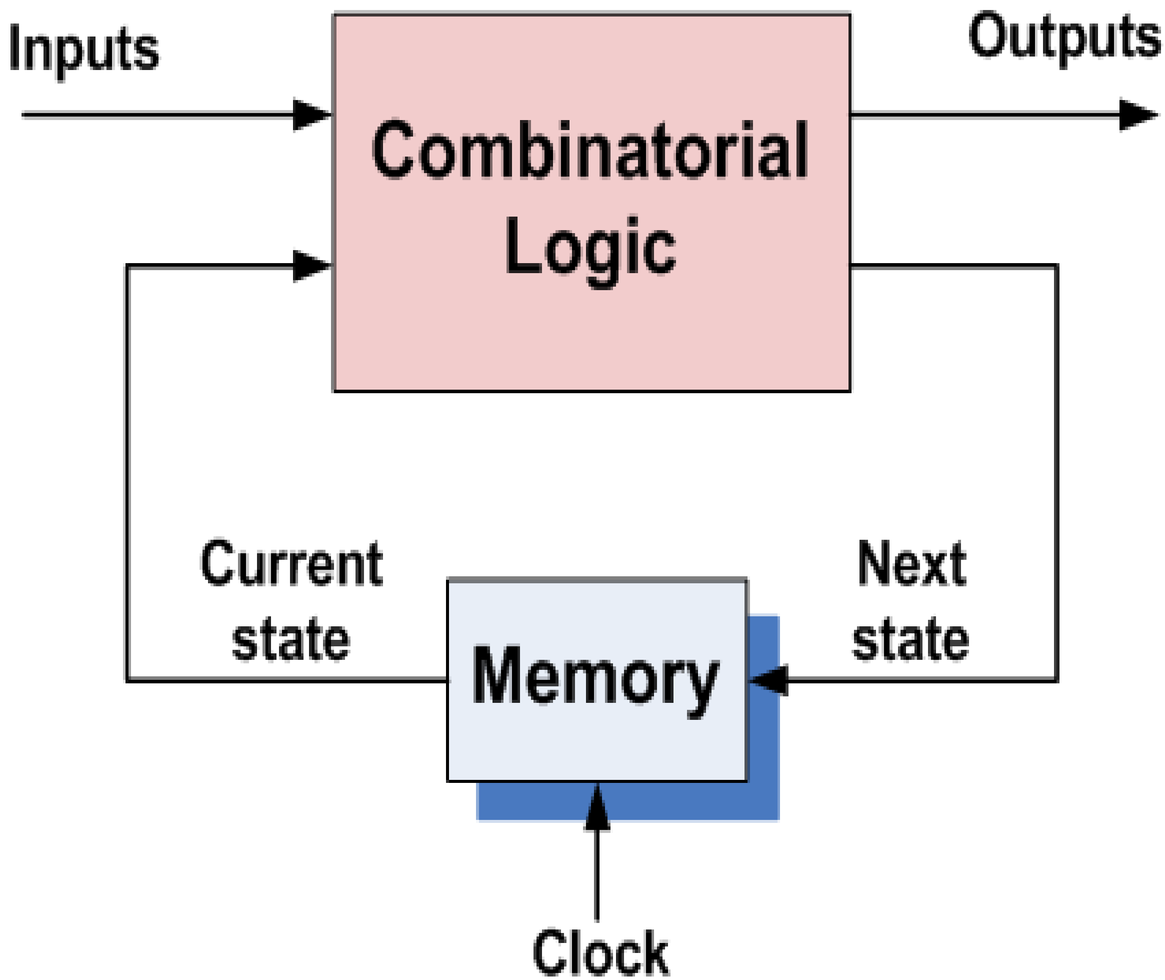
- A finite state machine is commonly used to build a controller for sequential logic.
- A state machine consists of
  - combinatorial logic,
  - delay elements,
  - feedback from the outputs of the delay elements, through combinatorial logic, back to the inputs of the delay elements.



Finite state machine

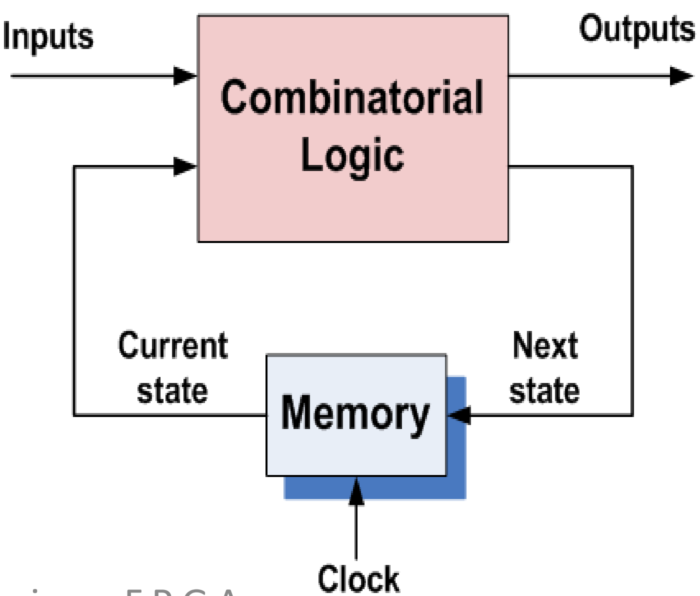
- A finite state machine can be divided into three parts:
  - State register
  - Next state logic
  - Output logic
- The following types of state machine are commonly used:
  - Mealy state machine
  - Moore state machine

Mealy state machine model



Mealy state machine

- Has the following features:
  - Outputs depend on both current state and inputs
  - May be fewer states
  - It has asynchronous outputs
    - If the input glitches, so does the output
    - Output available immediately
    - May be unstable output
    - Static/dynamic hazards!



Example: A vending machine

- Design specifications:
  - Our machine accepts 5p, 10p or 20p coins
  - Dispense the product when 20p has been paid
  - Return change if the payment exceeds 20p

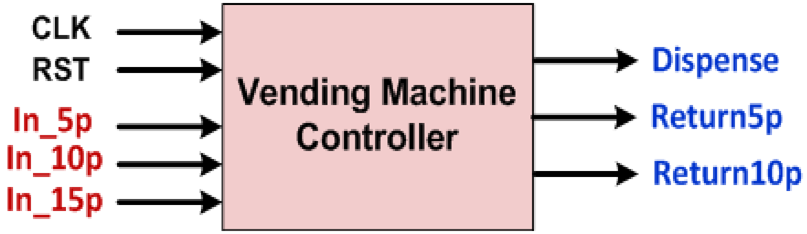
- We can use 4 states to represent all possible states:

S0 (00) = 0p	S1 (01) = 5p
S2 (10) = 10p	S3 (11) = 15p

  - We can use 2 bits each to give each of these states their own unique code

Example: A vending machine

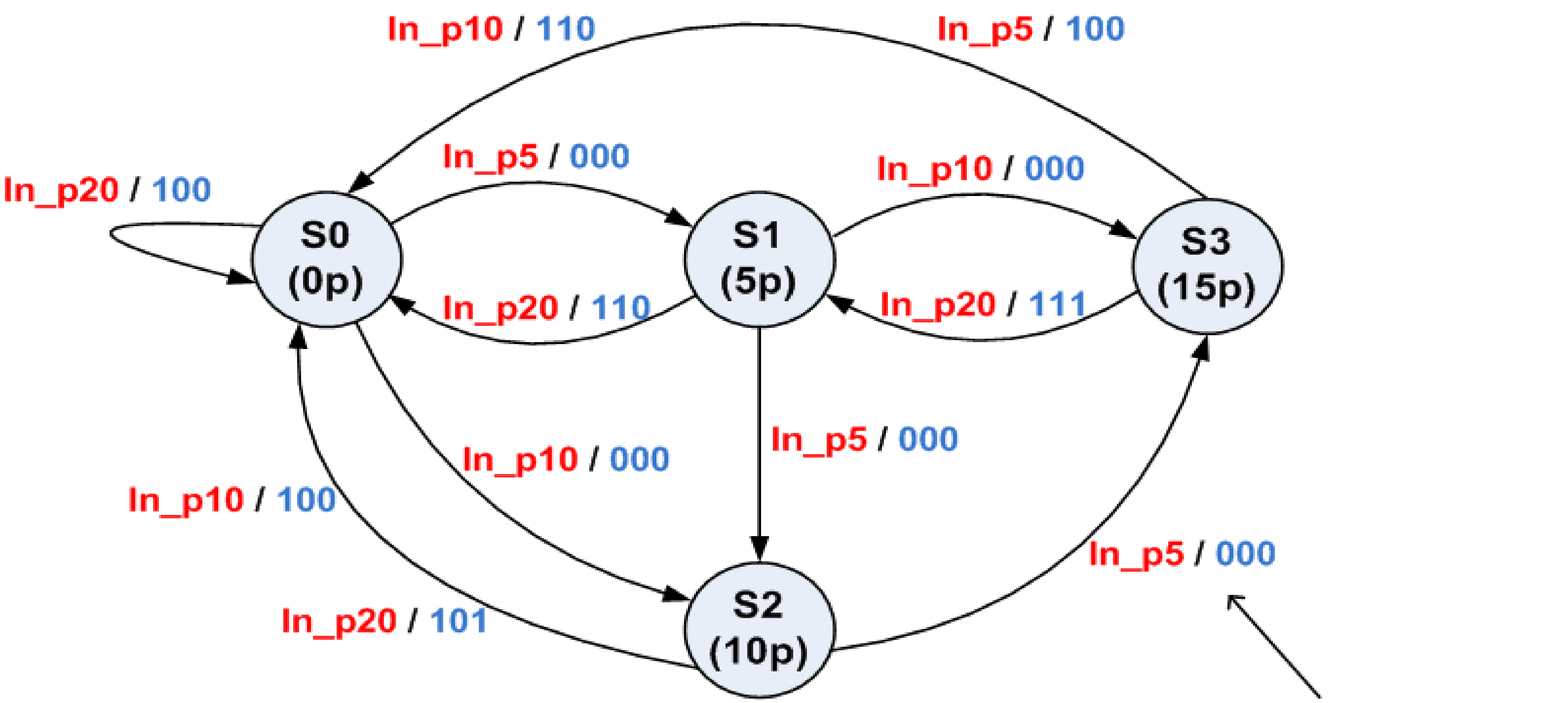
- 5 possible inputs
  - CLK, RST, In_5p, In_10p and In_20p
  - For instance, if we insert 10p we assign In_10p = ‘1’;



- 3 possible outputs
  - Dispense, Return5p and Return10p
  - Dispense = ‘1’ if the payment >= 20p, else Dispense = ‘0’
  - For instance:
    - If user inserts 10p then 20p, outputs will be Dispense = ‘1’, Return5p = ‘0’ and Return10p = ‘1’
    - If user inserts 5p, then 10p, then 10p, outputs will be Dispense = ‘1’, Return5p = ‘1’ and Return10p = ‘0’

Example: Vending machine controller using Mealy machine

- Express the controller as a state diagram:



Input = In_p5 | In_p10 | In_p20

$Output_3$ = **1** | **0** = Dispense / Do not dispense product

$Output_2$ = **1** | **0** = Return / Do not return a 5p in change

$Output_1$ = **1** | **0** = Return / Do not return a 10p in change

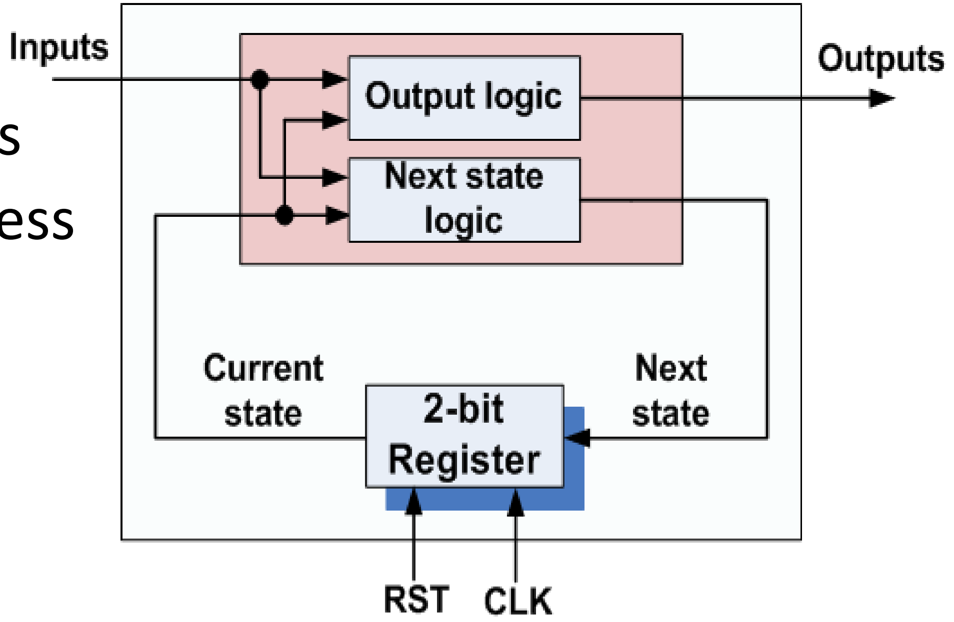
Example: Vending machine controller using Mealy machine

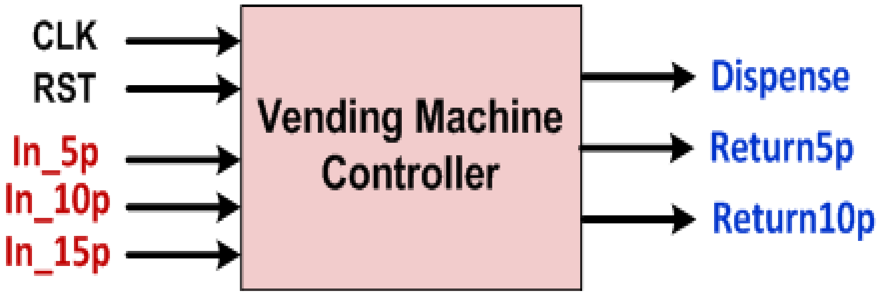
➢ Express the specification as a state transition table:

Current state Q0 Q1	Next state					Outputs		
	In_p5,In_p10,In_p20					Dispense,Return5p,Return10p		
(S0) 0 0	0	0	0	0	0	0	0	0
	0	0	1	0	0	1	0	0
	0	1	0	1	0	0	0	0
	1	0	0	0	1	0	0	0
(S1) 0 1	0	0	0	0	1	0	0	0
	0	0	1	0	0	1	1	0
	0	1	0	1	1	0	0	0
	1	0	0	1	0	0	0	0
(S2) 1 0	0	0	0	1	0	0	0	0
	0	0	1	0	0	1	0	1
	0	1	0	0	0	1	0	0
	1	0	0	1	1	0	0	0
(S3) 1 1	0	0	0	1	1	0	0	0
	0	0	1	0	1	1	1	1
	0	1	0	0	0	1	1	0
	1	0	0	0	0	1	0	0

The Mealy machine implementation

- FSM implementation consists of:
  - Entity declaration
  - Define state encoding
  - State register using a process
  - Next state logic using a process
  - Output logic using a process





The Mealy machine implementation

- Entity declaration and define state encoding

```
Entity  VFSM  is
        port  (  CLK, RST                          :  IN     std_logic;
                 In_5p, In_10p, In_20p             :  IN     std_logic;
                 Dispense, Return5p, Return10p     :  OUT  std_logic  );
end VFSM ;

architecture   My_FSM  of   VFSM   is
        type FSM_type  is  ( S0 ,  S1 ,  S2 ,  S3 ) ;
        signal Current_State,  Next_State  :  FSM_type;
begin
```

The Mealy machine implementation

- State register using a process block

```
process (CLK , RST) begin
    if  RST = '1'  then
        Current_State <= S0;
    elsif rising_edge (CLK)  then
        Current_State <= Next_State;
    else
        Current_State <= Current_State;
    end if;
end process;
```

The Mealy machine implementation

- Next state logic using a process block

```
process ( Current_State,  In_5p,  In_10p,  In_20p )  begin
    case   Current_State   is
        when S0   =>  if  In_5p  =  '1'  then Next_State <=  S1;
                      elsif   In_10p = '1'   then Next_State <=  S2;
                      elsif   In_20p = '1'   then  Next_State <=  S0;
                      else    Next_State <= S0;    end if;
        when S1   => …
        when  S2   =>  …
        when S3   => …
        when others => Next_State <= S0;
    end case;
end process;
```

Example: Vending machine controller

- Output logic using a process block

```
process ( Current_State,  In_5p,  In_10p,  In_20p )  begin
      case  Current_State  is
            when S0   =>  if  In_5p  =  '1'  then Dispense <= '0';   Return5p <= '0';    Return10p <= '0';
                              elsif   In_10p = '1'   then Dispense <= '0';  Return5p <= '0';   Return10p <='0';
                              elsif   In_20p = '1'   then Dispense <= '1';  Return5p <= '0';   Return10p <= '0';
                              else Dispense <= '0';   Return5p <= '0';    Return10p <= '0';    end if;
            when S1   => …
            when S2   =>   …
            when S3   =>   …
            when others  =>  Dispense <= '0';   Return5p <= '0';    Return10p <= '0';
      end case;
end process;
```

Assignment 2

- Will be issued this week
- Same structure as Assignment 1:
  - Individual implementation exercise
- Will require you to design and implement state machines!
  - Also, to explain how they work
- You might find it helpful to read further on Mealy and Moore state machines …