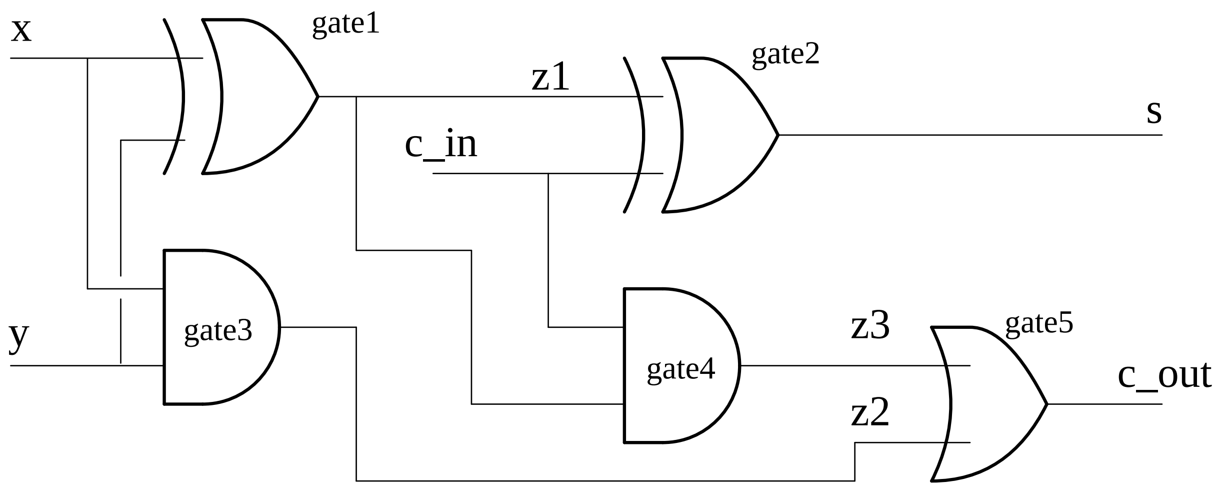
CMPT 250 : Week 2 (Sept 12 to Sept 17)

1. STRUCTURAL DESCRIPTIONS IN VHDL

Structural descriptions, equivalent to circuit diagrams can also be defined in VHDL. One possible implementation of the full adder with gates is given by the logic diagram:



This diagram can be described in a VHDL architecture body as follows:

```
architecture struct of FA is
signal z1, z2, z3: bit;
begin
   gate1: entity work.XOR2(behav)
        port map(a=>x, b=>y, z=>z1);
   gate2: entity work.XOR2(behav)
        port map(a=>z1, b=>z, z=>s);
   gate3: entity work.AND2(behav)
        port map(x=>x, y=>y, z=>z2);
   gate4: enityt work.AND2(behav)
        port map(x=>z, y=>z1, z=>z3);
   gate5: entity work.OR2(behav)
        port map(x=>z3, y=>z2, z=> z);
end struct;
```

Now this structural description in VHDL assumes the prior formal definition of entities `XOR2`, `AND2`, and `OR2`. These entities define a 2-input XOR gate, a 2 input AND gate, and a 2-input OR gate respectively. They can be defined as behavioral descriptions, and in each case the name of the defining architecture is `behav`

One possible definition for `XOR2` is:

```
entity XOR is
   port( a, b: in std_logic;
         z: out std_logic);
end XOR2;

Architecture behav of XOR2 is
begin
   Or_proc: process
   begin
        z <= x xor y;
        wait on x, y;
   end process;
end behav;
```

Definitions for `AND2` and `OR2` can be constructed in a similar way.

2. FORMAL SPECIFICATION OF SEQUENTIAL SYSTEMS

A *sequential system* is one whose output at any instant depends on the current value of the inputs **and the current content of all memory elements**, collectively called the *state* of the machine.

A sequential system is defined by:

1. An Input Set (I) : The set of values that can be input.
2. An Output Set (O) : Range of values that can occur as outputs.
3. A set of States (S) : The set of all values that can be assigned to the collective memory elements of the machine.
4. An Input/Output Function (G : S X I --> O) : The set of transformations performed by the system (NOTE that an output now depends on both the input and the current contents of memory).
5. A State Transition Function (H : S X I --> S) : defines how memory can be changed as a function of the current inputs and what was previously stored in memory.

In general, a sequential machine is required when a particular task is to be achieved in a sequence of steps. Each step constitutes an activity in which a combinational circuit performs a computation which is stored in memory, thus altering the state. A clock is used to partition time into a sequence of intervals, called the clock period. The clock period should be longer than the propagation delay of any combinational circuit that is enabled during that period.

Different ways exist for providing a behavioral specification of a sequential circuit:

- A state transition table (analgous to a function table for combinational circuits)
- A finite state machine diagram
- Algebraic equations that define the memory-input and output-signal functions.
- An algorithmic specification

EXAMPLE: Consider the problem of designing a mod-3 counter with two inputs:

- `r` resets the count to 0, provided it is enabled.
- `en` enables the circuit to either increment its count or be reset.

From the problem description, the following sets are identified:

Input set, I = {00. 01, 10, 11}, the four possible values that can be assigned to the input variables `en, r`.

Output set, O = {0, 1, 2}, the three possible values for the counter.

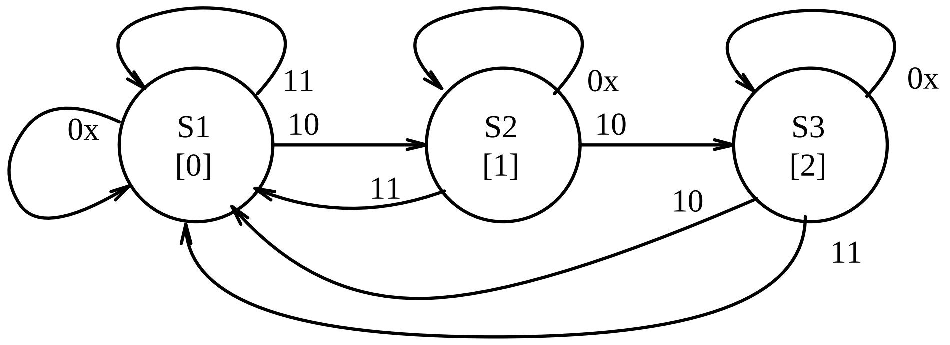
State set, S = {S1, S2, S3}. Three states are required to remember which one of the three possible values is the current value in the counter.

State transition table specification:

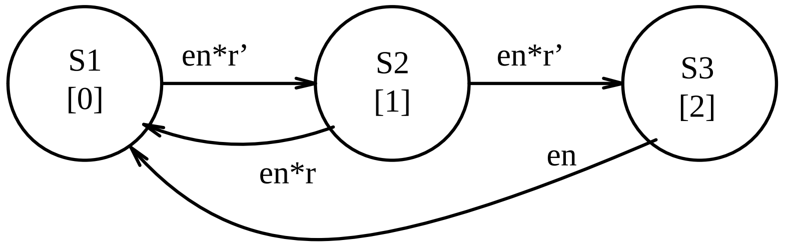
PRESENT	INPUT		NEXT	OUTPUT
STATE	en	r	STATE	
S1	0	x	S1	0
S1	1	0	S2	1
S1	1	1	S1	0
S2	0	x	S2	1
S2	1	0	S3	2
S2	1	1	S1	0
S3	0	x	S3	2
S3	1	x	S1	0

2.1. FINITE STATE MACHINE DIAGRAMS

A finite state machine diagram or *state transition diagram* is a graphical representation of the state transition table. This diagram is constructed from a set of vertices and directed edges. Each vertex corrsponds to one state, and is labelled by the state (current value of memory) that it represents. If appropriate, the output that occurs when the machine is in a given state is also included. A directed edge (arrow) is drawn from state S1 to state S2, if there exists a row in the state transition table with S1 in the Column labelled "PRESENT STATE" and S2 in the column labelled "NEXT STATE". The edge is labelled by the values of the external inputs that are specified on the same row of the characteristic table. For example, the mod 3 counter can be specified by:



Traditional State DIagram with all transitions explicitly shown



SImplified State Diagram with implied loops and boolean labels

Figure 2-1: Possible state transition diagrams for a mod 3 counter

2.2. SIMPLIFYING CONVENTIONS FOR STATE DIAGRAMS

Normally there should be a transition arrow from each state for evey possible input sequence. However labelling conventions are often adopted to permit state diagrams to be drawn more simply:

1. Self loops are eliminated. Any input sequence not explicitly identified in the state diagram is assumed to result in no change to memory, and therefore no state transition to a new state.
2. The labelling of inputs on the transition arrows can take several forms:
    - An equation stating explicitly the value assigned to each input variable.
    - A binary sequence that, together with a key indicating which bit is associated with which input variable, defines explicitly an assignment of values to all input variables.
    - An expression that is true only when the assignment of values to the variables will result in the state transition that is labelled by that expression. When only a unique assignment of values will cause the transition, this expression corresponds to a product term.
3. When more than one input sequence leads to the same state, the individual transition arrows can be combined into a single transition arrow whose label is the disjunction of expression labels associated with each original transition.

2.3. FINITE STATE MACHINES IN VHDL

For finite state machines that describe behavior to be implemented using a synchronous design, a *clock enable* input is required as an additional input to the black-box. This in turn means that a clock input should be included in the entity definition corresponding to the black-box. Finally, only the clk input needs to be included in the sensitivity list of the process within the architecture definition that describes the behavior. This is because a synchronous system can only change when the clock enable changes, on the rising edge (if positive edge triggered), or on the falling edge (if negative edge triggered).

The following behavioral description describes a finite state machine by its state-transition/output table:

PRES	INPUT	NEXT	OUTPUT
STATE	x	STATE	z
S1	0	S2	1
S1	1	S1	0
S2	0	S2	1
S2	1	S1	0

The black-box for this device would have one input, `x` and one output, `z`.

One way to define the behavior of this sequential circuit, that assumes a positive-edge triggered clock input, `clk`, in addition to `x` is as follows:

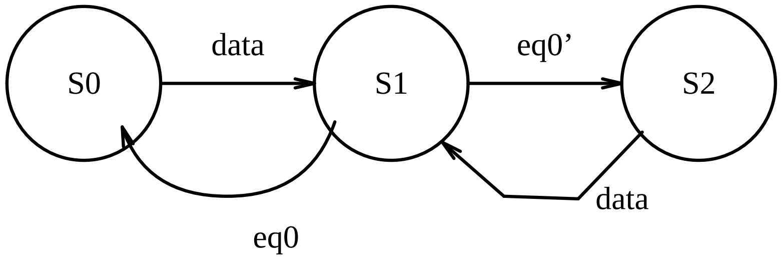
```
entity FSM is
   port(x, clk: in bit;
        z: out bit);
end FSM;

architecture behav of FSM is
begin
   PROC:process
   type state is (S1, S2);
   variable st: state := S1;
   begin
      if clk = ’1’ then
--
--  --The state transition function:
--
         if x = ’0’ and st = S1 then
            st := S2;
         elsif x = ’1’ and st = S2 then
            st := S1;
         end if;
--
--  --The output function:
--
         case st is
            when S1 => y <= 0;
            when S2 => y <= 1;
         end case;
--
      end if;
      wait on clk;
   end process;
end behav;
```

Notice how the positive edge trigger is achieved: The process PROC can only be activated by a change to `clk`. Once activated the process only does something useful if `clk = ’1’`. That is the `clk` must have changed from logic 0 to logic 1 when the process was invoked.

3. DESIGN FROM FINITE STATE MACHINES

As with combinational systems there are several possible ways to obtain the structural description (i.e., logic diagram) from the behavioral description.

The following state diagram will be used to illustrate the different methods:



3.1. STATE TRANSITION TABLE METHOD

The "classical" method employs the excitation table for the flip flops chosen to implement the state memory. The excitation table describes what inputs are necessary to cause a flip-flop to change state in a particular way or to remain in its current state.

For the example sequencer above, the states S0, S1, and S2 can be encoded as 00, 01, and 10 respectively. Choosing JK flip-flops to implement the state memory, the excitation table for the JK flip flop is:

```
Q Q+| J K
----|-----
0 0 | 0 X
0 1 | 1 X
1 0 | X 1
1 1 | X 0
```

From the state diagram, construct the state/transition table, and by examining the current and next state requirements, assign appropriate values to the inputs of the JK flip flops. Two are required to represent S0, S1, and S2:

Using K-maps, simplified sum-of-products boolean expressions can be obtained for the FF inputs:

```
J1 = Q0*eq0’, K1 = data

J0 = data, K0 = 1
```

This implementation assumed a natural binary encoding of the states by the first

Q1	Q0	eq0	data	Q1+	Q0+	J1	K1	J0	K0
0	0	0	0	0	0	0	X	0	X
0	0	0	1	0	1	0	X	1	X
0	0	1	0	0	0	0	X	0	X
0	0	1	1	0	1	0	X	1	X
0	1	0	0	1	0	1	X	X	1
0	1	0	1	1	0	1	X	X	1
0	1	1	0	0	0	0	X	X	1
0	1	1	1	0	0	0	X	X	1
1	0	0	0	1	0	X	0	0	X
1	0	0	1	0	1	X	1	1	X
1	0	1	0	1	0	X	0	0	X
1	0	1	1	0	1	X	1	1	X
1	1	X	X	X	X	X	X	X	X

Figure 3-1: State/Transition Table and FF Input Function Table

binary numbers. However it is often more convenient to adopt an encoding of the states by a bit vector, where each bit corresponds to one state. Then the state can be determined from the position of the single bit whose value is ’1’, with all other bits equal to ’0’. In the above example, the output that identifies the state is implemented so that a separate signal line is associated with each state. That is, three output lines are introduced S0, S1, and S2 so that:

```
S0 = Q1′*Q0′

S1 = Q1′*Q0

S2 = Q1*Q0′
```

These product terms can be generated using a 2x4 decoder, with the outputs Q1 and Q0 connected to the decoder select inputs, s1, and s0 respectively.