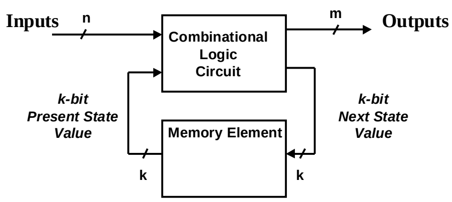
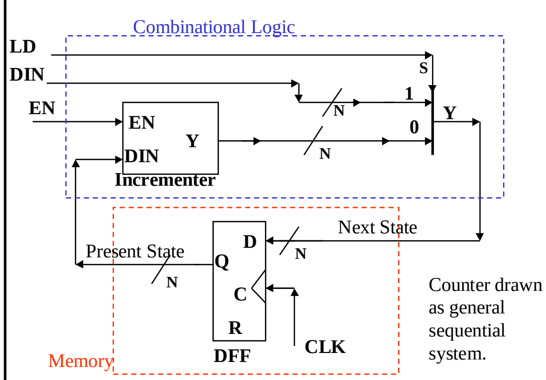
General Sequential Design

So far we have, we have looked at basic latches, FFs and common sequential building blocks.

All of these can be represented by a general block diagram:





Counter drawn as general sequential system.

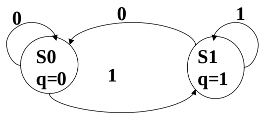
Describing Sequential Systems

- So far we have used Truth Tables to describe sequential systems
- Can also use Bubble Diagrams and Algorithmic State Machine Charts (ASM) to describe a sequential system.
- Another name for a sequential system is a ***Finite State Machine*** (FSM).
- A sequential system with N flip-Flop has 2^N possible states, so the number of possible states is FINITE.

DFF as a Finite State Machine

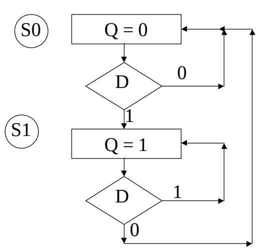
A DFF is a finite state machine with two possible states. Lets call these states S0 and S1. (*state enumeration*).

Furthermore, lets say when the Q output = ‘0’, then we are in State S0, and that when Q output = ‘1’, we are in State S1. This is called the *State Encoding*.



Bubble Diagram: States represented by bubbles. State transitions represented by arrows. Labeling on arrows represent input values (in this case, the D-input!). Labeling inside bubbles represent output values.

Algorithmic State Machine Chart for DFF



A Finite State Machine (FSM) can be described via either a Bubble diagram or an ASM chart.

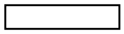
ASM charts are better for complex FSMs. We will use ASM charts in this class.

State S0 is usually the asynchronous Reset state.

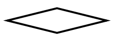
Algorithmic State Chart (ASM)

- An ASM chart can be used to describe FSM behavior

Only three action signals can appear within an ASM chart:



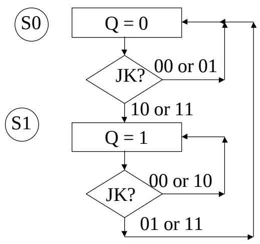
State box. Each box represents a state. Outputs within a state box is an UNCONDITIONAL output (always asserted in this state).



Decision box. A condition in this box will decide next state condition.

Conditional output box. If present, will always follow a decision box; output within it is conditional.

Algorithmic State Machine Chart for JKFF



Finite State Machine Implementation

Given an Algorithmic State Machine chart that describes a Finite State Machine, how do we implement it?????

Step #1: Decide on the State Encoding (how many Flip Flips do I use and how what should the FF outputs be for EACH state). The problem definition *may* decide the state encoding for you.

Step #2: Decide what kind of FFs to use! (We will always use DFFs in this class, but you could use JKFFs or TFFs if you wanted to).

Step #3: Write the State Transition Table.

Step #4: Write the FF input equations, and general output equations from the state transistion table.

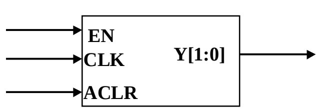
Problem Definition

Design a *Modulo three* counter. The count sequence is:

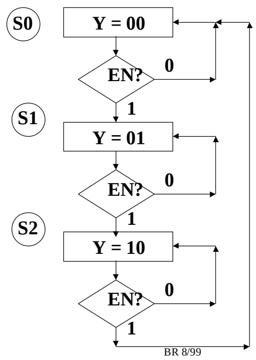
"00" → "01" → "10" → "00" → "01" → "10", etc.

There is an "en" input that should control counting (count when en=1, hold value when en=0). Assume ACLR line used to reset counter to "00".

How many states do we need? Well, we have three unique output values, so lets go with three states.



ASM Chart for Modulo Three Counter



BR 8/99 10

State Transition Table

State transition table shows next state, output values for present state, input values.

Inputs(EN)	Present State	Next State	Y
0	S0	S0	00
0	S1	S1	01
0	S2	S2	10
1	S0	S1	00
1	S1	S2	01
1	S2	S0	10

BR 8/99 11

Decisions

- **State encoding - will be based on number of FFs we use.**
  - **Three states means the minimum number of FFs we can use two FFs ($\log_2(3) = 2$).**
- **If we use two FFs, then could pick a state encodings like:**
  - **S0: 00, S1: 01, S2: 10 (binary counting order)**
  - **S0: 01, S1:01, S2: 11 (gray code - may result in less combinational logic)**
- **Could also use 1 FF per state (3 FFs) and use one hot encoding**
  - **S0:001, S1: 010, S2: 100 (may result in less combinational logic)**

BR 8/99 12

Decisions (cont.)

- **What type of FF to use?**
- **DFF - most common type, always available in programmable logic**
- **JKFF - sometimes available, will usually result in less combinational logic (more complex FF means less combinational logic external to FF)**

Lets use two FFs with state encoding S0=00, S1=01, S2=10.

Lets use DFFs.

New State Transition Table

Modify State Transition table to show what FF inputs need to be in order to get to that state. Also, use actual state encodings

Inputs(EN)	Present State (Q1Q0)	Next State (Q1Q)*	D1D0	Y
0	00	00	00	00
0	01	01	01	01
0	10	10	10	10
1	00	01	01	00
1	01	10	10	01
1	10	00	00	10

For DFFs, D inputs are simply equal to next state!!!!

D-input Equations, Y equations

Unoptimized equations:

D0 = EN’ Q1’Q0 + EN Q1’Q0’

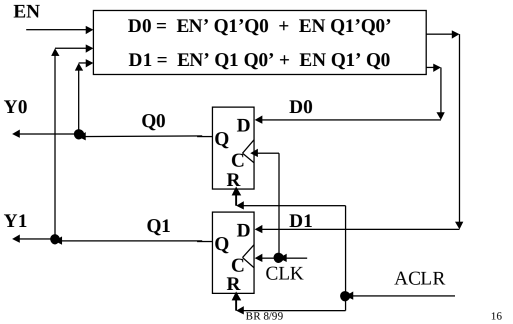
D1 = EN’ Q1 Q0’ + EN Q1’ Q0

Y0 = Q0

Y1 = Q1

The output Y is simply the DFF outputs! Here is one case where state encoding is affected by problem definition (does not make much sense to use a different state encoding, even though we could do it).

DFF Implementation



What if we used JKFFs?

Need to change State Transistion table to reflect JK input values.

Inputs EN	**Present State (Q1Q0)**	**Next State (Q1Q0)***	**J1 K1**	**J0 K0**	**Y**
0	**00**	**00**	**0 X**	**0 X**	**00**
0	**01**	**01**	**0 X**	**X 0**	**01**
0	**10**	**10**	**X 0**	**0 X**	**10**
1	**00**	**01**	**0 X**	**1 X**	**00**
1	**01**	**10**	**1 X**	**X 1**	**01**
1	**10**	**00**	**X 1**	**0 X**	**10**

JK FF Q transitions: 0→0 (J=0, K=X); 0→1 (J=1, K=X); 1→1 (J=X, K=0); 1→0 (J=X, K=1);

JK Input Equations, Output Equations

Unoptimized equations

J0 = EN Q1' Q0' **K0 = EN Q1' Q0**

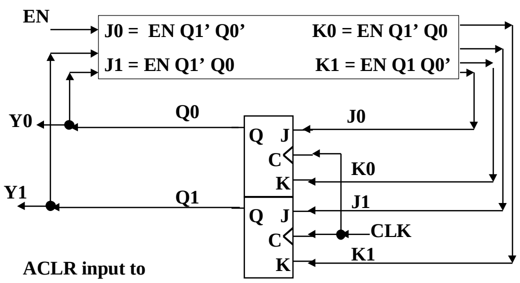
J1 = EN Q1' Q0 **K1 = EN Q1 Q0'**

Y0 = Q0

Y1 = Q1

Using JK FFs will mean simpler external optimized combinational logic because FFs are more complex (provide more functionality).

JK FF Implementation



ACLR input to
JKFFs not shown.

3 DFFs and One Hot Encoding

State encoding: S0 = 001, S1 = 010, S2 = 100

Inputs EN	Present State (Q2Q1Q0)	Next State (Q2Q1Q0)*	D2D1D0	Y
0	001	001	001	00
0	010	010	010	01
0	100	100	100	10
1	001	010	010	00
1	010	100	100	01
1	100	001	001	10

DFF input equations, Output Equations

D0 = EN’Q0 + ENQ2
D1 = EN’Q1 + ENQ0
D2 = EN’Q2 + ENQ1

Y0 = EN’Q1 + EN Q1 = Q1
Y1 = EN’Q2 + EN Q2 = Q2

In equations, because a FF Q will only be ‘1’ in a single state, do not have to include all FFs to define state!!
(Q2’Q1’Q0 = Q0!!, Q2’Q1Q0’ = Q1!, Q2Q1’Q0’ = Q2!!)
This is one of the advantages of one-hot encoding!

Generic Next State Equations

Generic next state equations can be written directly from the ASM chart as an alternative to the Transition table

S* = (conditions to remain in this state) + (conditions to enter state)

From ASM chart of modulo three counter:

S0* = EN' S0 + EN S2

S1* = EN' S1 + EN S0

S2* = EN'S2 + EN S1

If One hot encoding and DFFs are used, then Generic Next State equations ARE the specific next State Equations!!

D0 = EN'Q0 + EN Q2

D1 = EN'Q1 + EN Q0

D2 = EN' Q2 + EN Q1

BR 8/99 22