Outline

- Last time:
  - **Sequential Examples**
  - **Arithmetic Examples**
- This lecture:
  - Asynchronous Circuits
  - Moore and Mealy Standard Forms
  - Design Example: Word Problem
  - Race-free State Assignment
  - Excitation Equations
  - Design Example: Implementation
  - Summary of Asynchronous Design Process

Asynchronous Circuits (Feedback Sequential Circuits)

- Clocked Synchronous Circuits:
  - Change of state only occurs in response to a clock pulse
  - When this change of state requires that a number of flip-flops change their values, they do so simultaneously because they are synchronized by the common clock pulse.
  - Input changes are assumed to occur in between clock pulses and outputs may be read during or immediately before a clock pulse.
- Conditions where this model is too restrictive:
  - The network has inputs which may change at any time and cannot be synchronized by a clock.
  - Signal travel time down wires is significant and wire lengths in the circuit cannot be controlled
  - We want the network to operate as fast as possible
  - The power dissipation overhead of clocking signals that do not change is unacceptable (we need "event-driven" circuits).

Asynchronous Circuits

- Asynchronous Circuits:
  - Operation is not synchronized by a clock.
  - When an input change occurs, the state of the network can change almost immediately.
  - If several storage elements must change state, there is no guarantee they will do so at the same time.

- To simplify this challenging problem we consider only *fundamental mode* asynchronous circuits:
  - The input signals will only change when the circuit is in a stable condition (i.e. no internal signals are changing).
  - All of the input signals are considered to be levels, rather than pulses or edges.
  - Inputs may not change simultaneously
  - Asynchronous circuits may be structured as Mealy and Moore forms, as before, but delays in the feedback path are not clocked and may be different!

D Latch Example

delay

Excitation Equation for Y*:
Y* = CD + (CD' +Y')' = CD + C'Y + DY

D Latch Example: Transition Table

CD

Y	00	01	11	10
0	**0**	**0**	**1**	**0**
1	**1**	**1**	**1**	**0**

Y*

D Latch Example: State Table

CD

S	00	01	11	10
S0	**S0**	**S0**	**S1**	**S0**
S1	**S1**	**S1**	**S1**	**S0**

S*

○ = stable state

Transition Table

- ❍ A transition table has one row foreach possible combination of the state variables.
  - ➔ If the circuit has n feedback loops, it has 2^n rows in its transition table
- ❍ The table has one column for each input combination
  - ➔ A circuit with m inputs has 2^m columns in its transition table.
- ❍ The circuit is monitoring its inputs continuously

Total State

- ❍ The total state of a circuit is a particular combination of the internal state (values stored in the feedback loops) and input state (the current values of the circuit inputs).
- ❍ A *stable total state* is a combination of *internal state* and *input state* such that the next internal state predicted by the transition table is the same as the current internal state.
- ❍ If the next internal state is different, then the combination is an *unstable total state*.

D Latch Example: Output Equations

- $Q = CD + C'Y + DY$
  $/Q = CD' + Y'$
- Y is the only internal state variable
- Combined state and output table:

S	CD 00	01	11	10
S0	**S0,01**	**S0,01**	**S1,11**	**S0,01**
S1	**S1,10**	**S1,10**	**S1,10**	**S0,01**

S*, Q /Q

Races

- In a feedback sequential circuit, a *race* is said to occur when multiple internal variables change state as a result of a single input variable changing state.

- If the final state depends on the order in which the variables change, the race is said to be *critical*.

Analysis Example: Positive Edge-Triggered D Flip-Flop

Transition Table for D Flip-Flop

$Y_1Y_2Y_3$ \ CLK D	00	01	11	10
000	010	010	000	000
001	011	011	000	000
010	010	110	110	000
011	011	111	111	000
100	010	010	111	111
101	011	011	111	111
110	010	110	111	111
111	011	111	111	111

$Y_1{*}Y_2{*}Y_3{*}$

State and Output Table for D Flip-Flop

CLK D

S	00	01	11	10
S0	S2,01	S2,01	S0,01	S0,01
S1	S3,10	S3,10	S0,01	S0,01
S2	S2,01	S6,01	S6,01	S0,01
S3	S3,10	S7,10	S7,10	S0,01
S4	S2,01	S2,01	S7,11	S7,11
S5	S3,10	S3,10	S7,10	S7,10
S6	S2,01	S6,01	S7,11	S7,11
S7	S3,10	S7,10	S7,10	S7,10

S*

Flow and Output Table for the D Flip-Flop

CLK D

S	00	01	11	10
S0	S2,01	S6,01	S0,01	S0,01
S2	S2,01	S6,01	-,-	S0,01
S3	S3,10	S7,10	-,-	S0,01
S6	S2,01	S6,01	S7,11	-,-
S7	S3,10	S7,10	S7,10	S7,10

S*

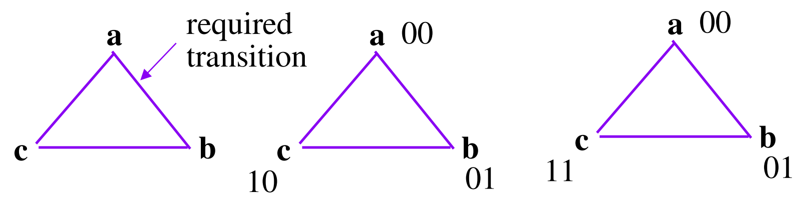
Example: After State Reduction and Merging

X1X2	00	01	11	10	00	01	11	10
a(1,2)	1	2	4	3	0	0	-	-
b(3,5,6)	1	6	5	3	-	0	0	0
c (4)	-	6	4	3	-	-	1	-

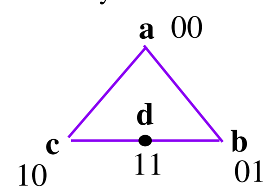
Example: States Labelled in Terms of Internal States

X1X2	00	01	11	10	00	01	11	10
a	a	a	c	b	0	0	-	-
b	a	b	b	b	-	0	0	0
c	-	b	c	b	-	-	1	-

Example: Race-Free State Assignment



- Line between states indicates required transition.
- Need assignment such that only one state variable changes during each state transition



Example: Critical-Race-Free State Assignment

X1X2

	00	01	11	10	00	01	11	10
a 00	00	00	10	01	0	0	-	0v
b 01	00	01	01	01	0u	0	0	0
d 11	-	01	-	-	-	-s	-	-
c 10	-	11	10	00	-	1t	1	-

- In col. 01, row C, 10->11->01: critical race avoided
- In col 10, could have directed transition via row d but already have transition to b in row a, so use it.
- Fill in outputs corresponding to unstable states to avoid momentary false outputs during transitions

Critical-Race-Free State Assignment via Shared-Row Assignments

	00	01	11	10
a	1	2	5	4
b	7	2	3	10
c	1	8	3	4
d	7	8	5	4
e	1	9	5	6
f	11	8	3	6

- **Required transitions:**
  col 00: e,c -> a; d->b
  col 01: a->b; c,f->d
  col 11: b,f->c; a,d->e
  col 10: a,c->d; e->f

Shared Row Assignment

- Consider required transitions: e,c -> a
- Could implement as: e->a, c->a; or e->c->a; or c->e->a; or any of the above going through an intermediate state (or states).
- To avoid critical races, must ensure that {a,c,e} are placed in adjacent squares on the assignment map. Similar for other constraints. Must satisfy:
  {a,c,e}, {b,d}, {a,b}, {c,d,f}, {b,c,f}, {a,d,e}, {a,c,d}, {e,f}

a	e
b	d
x	c
y	f

Steps to Asynchronous FSM Design

- Construct a *Primitive Flow Table* from the word statement of the problem.
- Derive a minimum-row primitive flow table or *Reduced Primitive Flow Table* by eliminating redundant, stable total-states.
- Convert the resulting table to Mealy form, if necessary, so that the output value is associated with the total state rather than the internal state.
- Derive a minimum-row flow table, or *Merged Flow Table*, by merging compatible rows of the reduced primitive flow table using a merger diagram. (Note: solution not necessarily unique)
- Perform race-free, or critical-race-free, state assignment, adding additional states if necessary.
- Complete the *Output Table* to avoid momentary false outputs when switching between stable total states.
- Draw *logic diagram* that shows ideal combinational next-state and output functions as well as necessary delay elements.

Design Example 1: Word Problem

- An asynchronous network has two inputs and one output. The input sequence X_1X_2 = 00, 01, 11 causes the output, Z, to become 1. The next input change then causes the output to return to 0. No other input sequence will produce a 1 output.

	X_1X_2 00	01	11	10	Z
(reset) 1	1	2	-	3	0
(00,01) 2		2			0
(00,10) 3				3	0

Fundamental mode, single input changes, so input can only change in stable total state and only one bit can change, so this total-state is impossible, since only stable total state is inputs 00 in this row.

Derivation of Primitive Flow Table: Cont.

	X_1X_2 00	01	11	10	Z
(reset) 1	1	2	-	3	0
(00,01) 2	1	2	4	-	0
(00,10) 3	1	-	5	3	0
(00,01,11) 4			4		1
(00,10,11) 5			5		0

Derivation of Primitive Flow Table: Cont.

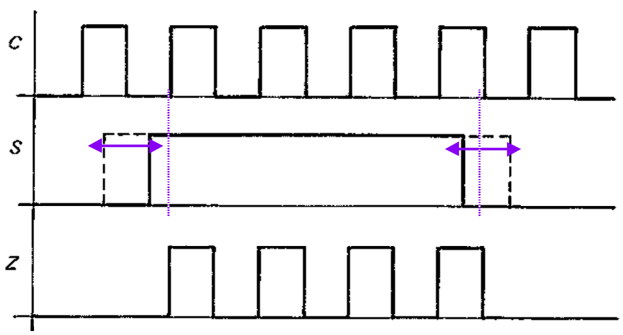
- *Primitive Flow Table: Only one stable state per row is permitted*, so every change in input must result in an internal state change as well as a toatl state change (by definition).

	X_1X_2 00	01	11	10	Z
(reset) 1	1	2	-	3	0
(00,01) 2	1	2	4	-	0
(00,10) 3	1	-	5	3	0
(00,01,11) 4	-	6	4	3	1
* 5	-	6	5	3	0
* 6	1	6	5	-	0

States marked * cannot lead to a 1 output without first resetting.

Design Example 2: Word Problem

- A clock signal (C) is to be gated on and off by another signal (S). The gating network must be such that only complete clock pulses appear at the output (Z) even though S may change in the middle of a clock pulse. S will always be on or off for at least two clock pulses.



Derivation of Primitive Flow Table: Cont.

CS

	00	01	11	10	Z
1	1	3	-	2	0
2	1	-		2	0
3		3	4	-	0
4	-	3	4		1

Derivation of Primitive Flow Table: Cont.

CS	00	01	11	10	Z
1	1	3	-	2	0
2	1	-	6	2	0
3	1	3	4	-	0
4	-	3	4	5	1
5	1	-	-	5	1
6	-	3	6	-	0

10->11 cannot occur in State 5 since S is assumed to be off for at least two clock pulses. Same for 11->10 in State 6.

Steps to Asynchronous FSM Design

- ✓ Construct a *Primitive Flow Table* from the word statement of the problem.
- ❍ Derive a minimum-row primitive flow table or *Reduced Primitive Flow Table* by eliminating redundant, stable total-states.
- ❍ Convert the resulting table to Mealy form, if necessary, so that the output value is associated with the total state rather than the internal state.
- ❍ Derive a minimum-row flow table, or *Merged Flow Table*, by merging compatible rows of the reduced primitive flow table using a merger diagram. (Note: solution not necessarily unique)
- ❍ Perform race-free, or critical-race-free, state assignment, adding additional states if necessary.
- ❍ Complete the *Output Table* to avoid momentary false outputs when switching between stable total states.
- ❍ Draw *logic diagram* that shows ideal combinational next-state and output functions as well as necessary delay elements.

Minimum-Row Primitive Flow Table: Eliminate Redundant Stable Total States

- *Redundant States*: Two stable total states are equivalent if:
  - (a) Their inputs are the same and
  - (b) Their outputs are the same and
  - (c) Their next-states are equivalent for each possible next input

In Example 1, possible candidates for combining are:
- {4,5} - no, because outputs not the same
- {2,6} - no, because next-states not equivalent under 11 input (states 4 and 5)

so no redundant states here.

In Example 2, possible candidates are:
- {4,6} - no, different outputs
- {2,5} - no, different outputs

so no redundant states here either.

Removal of Redundant States: Example

X1X2

	00	01	11	10	Z1Z2
1	1	7	-	4	1 1
2	2	5	-	4	0 1
3	-	7	3	11	1 0
4	2	-	3	4	0 0
5	6	5	9	-	1 1
6	6	7	-	11	0 1
7	1	7	14	-	1 0
8	8	12	-	4	0 1
9	-	7	9	13	0 1
10	-	7	10	4	1 0
11	8	-	10	11	0 0
12	6	12	9	-	1 1
13	8	-	14	13	1 1
14	-	12	14	11	0 0

Examine stable states in same column (same input) that have same output:

00: {2,6,8} -under input 01, {2,6} goe to different, not-equivalent next states. Similarly for {6,8}. {2,8} equiv. iff {5,12} equiv.

01: {5,12} - yes

11: {3,10} iff {4,11}

10: {4,11} iff {3,10} and {5,12}.

leads to:

{2,8}, {5,12}, {3,10}, {4,11}

so eliminate 8, 10, 11, 12

Removal of Redundant States: Example

X1X2

	00	01	11	10	Z1Z2
1	1	7	-	4	1 1
2	2	5	-	4	0 1
3	-	7	3	*4*	1 0
4	2	-	3	4	0 0
5	6	5	9	-	1 1
6	6	7	-	*4*	0 1
7	1	7	14	-	1 0
9	-	7	9	13	0 1
13	*2*	-	14	13	1 1
14	-	*5*	14	*4*	0 0

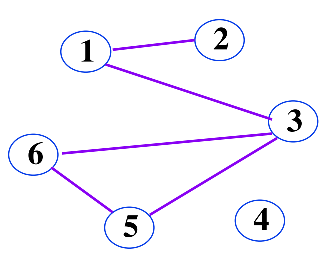
Steps to Asynchronous FSM Design

- ✓ Construct a *Primitive Flow Table* from the word statement of the problem.
- ✓ Derive a minimum-row primitive flow table or *Reduced Primitive Flow Table* by eliminating redundant, stable total-states.
- ❍ Convert the resulting table to Mealy form, if necessary, so that the output value is associated with the total state rather than the internal state.
- ❍ Derive a minimum-row flow table, or *Merged Flow Table*, by merging compatible rows of the reduced primitive flow table using a merger diagram. (Note: solution not necessarily unique)
- ❍ Perform race-free, or critical-race-free, state assignment, adding additional states if necessary.
- ❍ Complete the *Output Table* to avoid momentary false outputs when switching between stable total states.
- ❍ Draw *logic diagram* that shows ideal combinational next-state and output functions as well as necessary delay elements.

Conversion to Mealy Form for Merging: Example 1

		X1X2				Z				
		00	01	11	10	00	01	11	10	
1	1	2	-	3	0	-	-	-		2
	1	2	4	-	-	0	-	-		
3	1	-	5	3	-	-	-	0		
4	-	6	4	3	-	-	1	-		
5	-	6	5	3	-	-	0	-		
6	1	6	5	-	-	0	-	-		

Create Merger Diagram



What we are doing is choosing *internal* states that can have the *same* code, though the *total* states will still be different (since they will not have the same code for the same inputs; if that were possible they would have been redundant).

Two rows of a reduced primitive Mealy flow table are *compatible* and can be merged into a single row iff there are no state or output conflicts in any column.

- ➔ Draw a line between any pair of rows (states) which are compatible and can be merged.
- ➔ Choose the sub-groups of fully-connected rows that will result in the maximum reduction. In this case, {1,2} and {3,5,6} are best. We could also have chosen {1,3} and {5,6}, or just {3,6} in which case no others would be permitted.

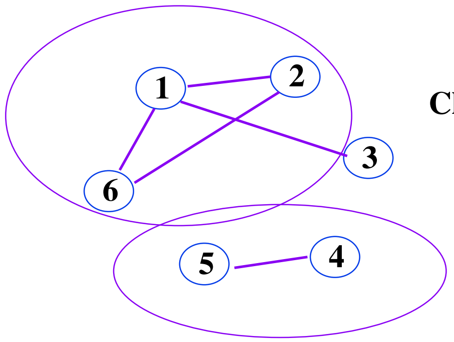
Example 1: After Merging

	X1X2 00	01	11	10	00	01	11	10
a(1,2)	1	2	4	3	0	0	-	-
b(3,5,6)	1	6	5	3	-	0	0	0
c (4)	-	6	4	3	-	-	1	-

Conversion to Mealy Form for Merging: Example 2

	CS 00	01	11	10	Z 00	01	11	10
1	1	3	-	2	0	-	-	-
2	1	-	6	2	-	-	-	0
3	1	3	4	-	-	0	-	-
4	-	3	4	5	-	-	1	-
5	1	-	-	5	-	-	-	1
6	-	3	6	-	-	-	0	-

Create Merger Diagram: Example 2



Choose {1,2,6} and {4,5}

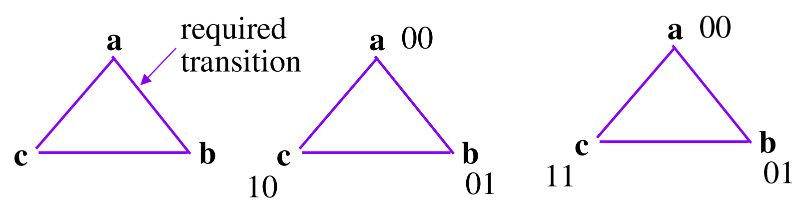
Steps to Asynchronous FSM Design

- ✓ Construct a *Primitive Flow Table* from the word statement of the problem.
- ✓ Derive a minimum-row primitive flow table or *Reduced Primitive Flow Table* by eliminating redundant, stable total-states.
- ✓ Convert the resulting table to Mealy form, if necessary, so that the output value is associated with the total state rather than the internal state.
- ✓ Derive a minimum-row flow table, or *Merged Flow Table*, by merging compatible rows of the reduced primitive flow table using a merger diagram. (Note: solution not necessarily unique)
- ❍ Perform race-free, or critical-race-free, state assignment, adding additional states if necessary.
- ❍ Complete the *Output Table* to avoid momentary false outputs when switching between stable total states.
- ❍ Draw *logic diagram* that shows ideal combinational next-state and output functions as well as necessary delay elements.

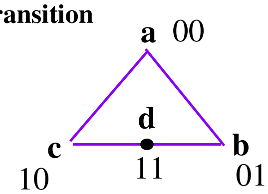
Example 1: States Labelled in Terms of Internal States

	X1X2 00	01	11	10	00	01	11	10
a	a	a	c	b	0	0	-	-
b	a	b	b	b	-	0	0	0
c	-	b	c	b	-	-	1	-

Example: Race-Free State Assignment



- Line between states indicates required transition.
- Need assignment such that only one state variable changes during each state transition



Example: Critical-Race-Free State Assignment

X1X2	00	01	11	10	00	01	11	10
a 00	00	00	10	01	0	0	-	0v
b 01	00	01	01	01	0u	0	0	0
d 11	-	01	-	-	-	-s	-	-
c 10	-	11	10	00	-	1t	1	-

- **In col. 01, row C, 10->11->01: critical race avoided**
- **In col 10, could have directed transition via row d but already have transition to b in row a, so use it.**
- **Fill in outputs corresponding to unstable states to avoid momentary false outputs during transitions**

- **Note: A race-free state assignment for any 3-row flow table can be found using two state variables.**

General Approach: Critical-Race-Free State Assignment via Shared-Row Assignments

	00	01	11	10
a	1	2	5	4
b	7	2	3	10
c	1	8	3	4
d	7	8	5	4
e	1	9	5	6
f	11	8	3	6

- **Required transitions:**
  col 00: e,c -> a; d->b
  col 01: a->b; c,f->d
  col 11: b,f->c; a,d->e
  col 10: a,c->d; e->f

Shared Row Assignment

- ❍ Consider required transitions: e,c -> a
- ❍ Could implement as: e->a, c->a; or e->c->a; or c->e->a; or any of the above going through an intermediate state (or states).
- ❍ To avoid critical races, must ensure that {a,c,e} are placed in adjacent squares on the assignment map. Similar for other constraints. Must satisfy:
  {a,c,e}, {b,d}, {a,b}, {c,d,f}, {b,c,f}, {a,d,e}, {a,c,d}, {e,f}

a	e
b	d
x	c
y	f

Steps to Asynchronous FSM Design

- ✓ Construct a *Primitive Flow Table* from the word statement of the problem.
- ✓ Derive a minimum-row primitive flow table or *Reduced Primitive Flow Table* by eliminating redundant, stable total-states.
- ✓ Convert the resulting table to Mealy form, if necessary, so that the output value is associated with the total state rather than the internal state.
- ✓ Derive a minimum-row flow table, or *Merged Flow Table*, by merging compatible rows of the reduced primitive flow table using a merger diagram. (Note: solution not necessarily unique)
- ✓ Perform race-free, or critical-race-free, state assignment, adding additional states if necessary.
- ❍ Complete the *Output Table* to avoid momentary false outputs when switching between stable total states.
- ❍ Draw *logic diagram* that shows ideal combinational next-state and output functions as well as necessary delay elements.

Completion of the Output Table

X1X2

	00	01	11	10	00	01	11	10
a 00	00	00	10	01	0	0	-	0v
b 01	00	01	01	01	0u	0	0	0
d 11	-	01	-	-	-	-s	-	-
c 10	-	11	10	00	-	1t	1	-

From State 00, input change 00->10 causes transition to stable state 01. To avoid glitch if logic synthesis assigns "don't care" in output to 1 for input value 10, make it a 0 (labelled v above).

From State 01, input change 01->00 causes transition to State 00, so avoid possible glitch by assigning output in State 01 for input 00 to 0 (u above)

From State 10, input change 11->01 causes transition to stable state 01, via 11. Since output goes from 1 to 0, choose output at t above to be 1, consistent with the starting value, but leave output at s a "don't care" since must make the transition somewhere and either before or after State 11 is the same.

Steps to Asynchronous FSM Design

- ✓ Construct a *Primitive Flow Table* from the word statement of the problem.
- ✓ Derive a minimum-row primitive flow table or *Reduced Primitive Flow Table* by eliminating redundant, stable total-states.
- ✓ Convert the resulting table to Mealy form, if necessary, so that the output value is associated with the total state rather than the internal state.
- ✓ Derive a minimum-row flow table, or *Merged Flow Table*, by merging compatible rows of the reduced primitive flow table using a merger diagram. (Note: solution not necessarily unique)
- ✓ Perform race-free, or critical-race-free, state assignment, adding additional states if necessary.
- ✓ Complete the *Output Table* to avoid momentary false outputs when switching between stable total states.
- ❍ Draw *logic diagram* that shows ideal combinational next-state and output functions as well as necessary delay elements.