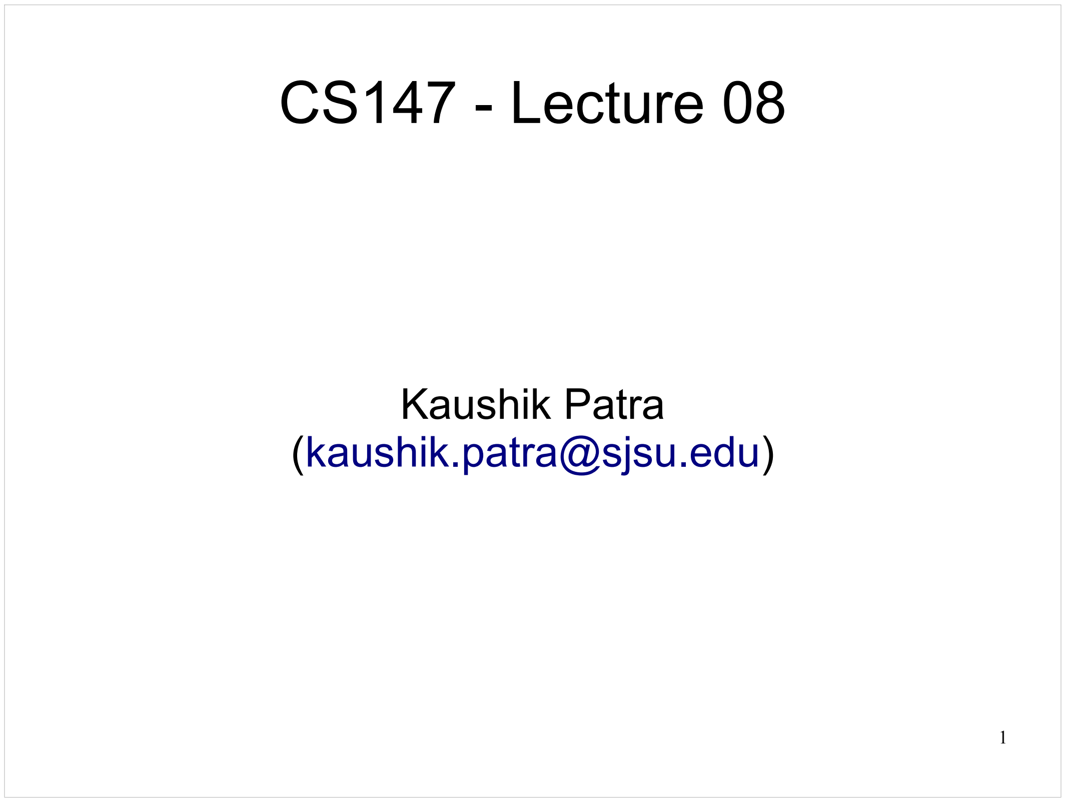


CS147 - Lecture 08

Kaushik Patra
(kaushik.patra@sjsu.edu)

1

- Sequential Circuit Design
- Digital Circuit Components

[Chapter 3, 5, 7, 9-4 of Logic & Computer Design Fundamentals, 4th Edition, M. Morris Mano, Charles R. Kime]

Sequential Circuit Design ...

Steps of Sequential Design

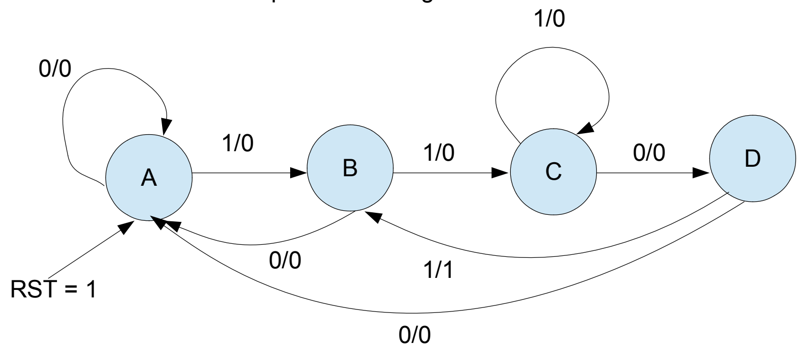
- **Specification**: Collect and write the specification
- **Formulation:** Obtain state diagram and state table.
- **State Assignment:** Assign binary code to the states and convert state table to truth table.
- **Output & Flipflop Input Equation:** Select flipflop type and determine the input equation (next state equation) for the flipflop.
- **Optimization**: Optimize the flipflop input equation and output equation.
- **Technology Mapping:** Map the obtained equation in terms of logic gates available for the technology.
- **Verification:** Verify the correctness of the final circuit.

Example Problem

- The sequence recognizer has one input X and one output Z. It has reset applied directly to the flipflop to initialize circuit to all zero. The circuit should recognize the occurrence of the sequenced bit of '1101' on incoming bit stream through X. Once a pattern 1101 is recognized output Z should produce 1, otherwise output will remain 0.

State Diagram

Sequence to recognize 1101



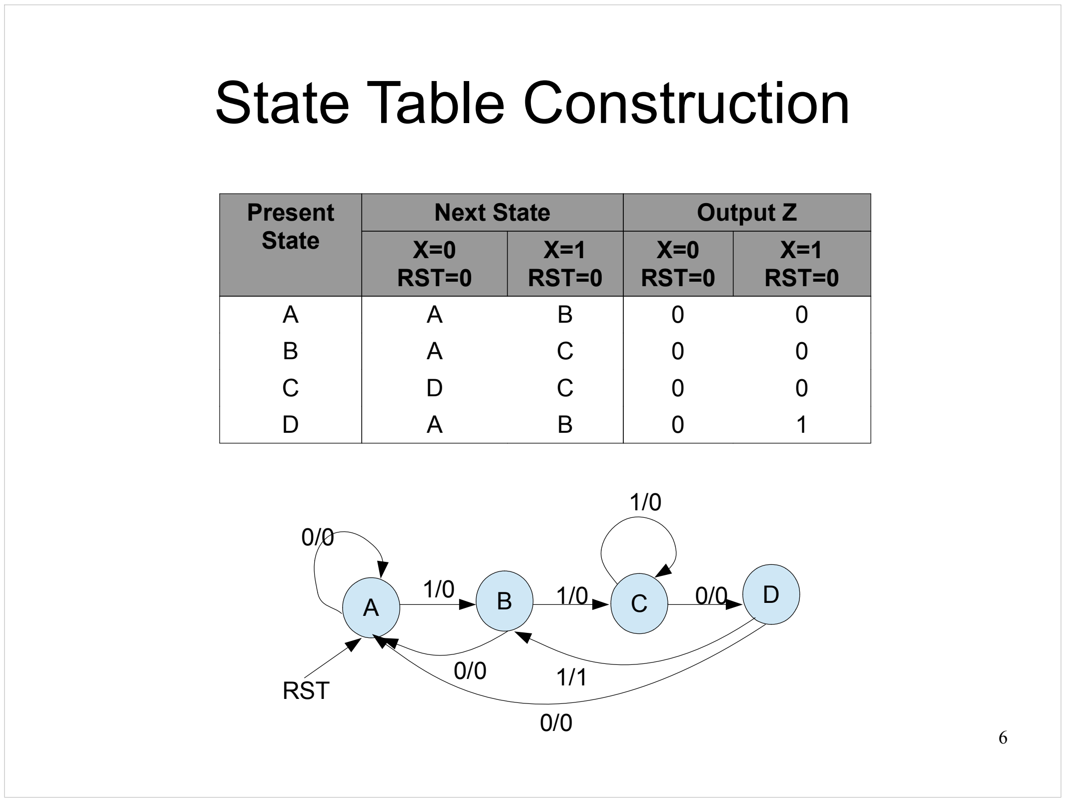
State A	'0' is seen.
State B	'1' is seen
State C	'11' is seen
State D	'110' is seen

- For the given problem, the circuit must remember what has been visited for last three times and decide if it is matching the given patter with the current value of X.

- A circle denotes a state of the machine, and transitions are marked with arrow to show state transitions. Each of this arrows or edges are marked with X/Z denoting what input caused the transition and what is the corresponding output value.

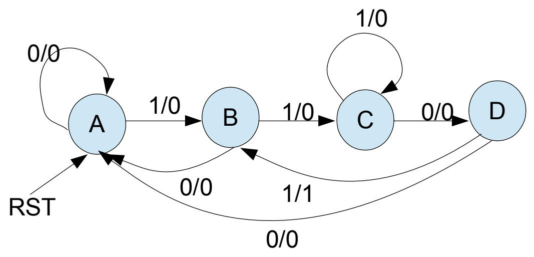
- The machine starts with state A which denotes that last value seen was 0. Upon receiving 0s it will stay at A and output will be 0. Once a 1 comes in, the state is changed to B and output is 0. The state B denotes that last value was 1. Upon receiving 0 in B, state changes to A since '10' can not be a start of '1101'. Upon receiving 1 in B, state goes to C which indicates that the last two values were 1 (a pattern '11' came). State will not be changed from C if further incoming values are 1, because getting more 1 means that at least the last 2 values were '11'. The state C will be changed to D upon receiving 0. This state D denotes that a pattern '110' has been seen. From this state D, transition will happen to B on getting 1 and output will 1 since it indicates that a pattern 1101 is seen. It goes to B because state B denotes that last value seen was '1'. If 0 is received at state D, the state will change to A because A denotes that the last value seen was 0.

- RST is the first state entry signal and it goes to state A with RST at 1. There is from state noted in the state diagram for RST=1. This means, the circuit will come to state A from what ever state it may be in.

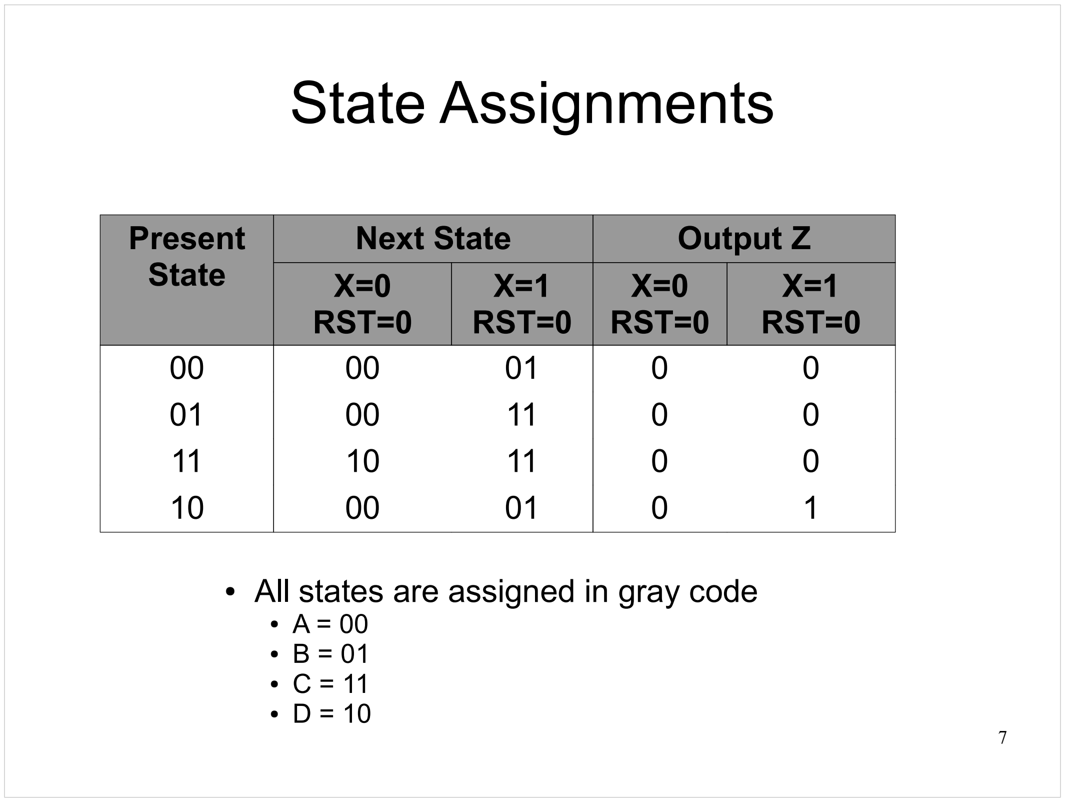


State Table Construction

Present State	Next State		Output Z	
	X=0 RST=0	X=1 RST=0	X=0 RST=0	X=1 RST=0
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1



- To construct state transition table, we list down the states in one column as present state of the machine. The next group of columns indicate the next state transition from the present state upon various combination of the primary input. There is another group of columns indicates the output value as a function of the present state and the input combinations.



State Assignments

Present State	Next State		Output Z	
	X=0 RST=0	X=1 RST=0	X=0 RST=0	X=1 RST=0
00	00	01	0	0
01	00	11	0	0
11	10	11	0	0
10	00	01	0	1

- All states are assigned in gray code
  - A = 00
  - B = 01
  - C = 11
  - D = 10

7

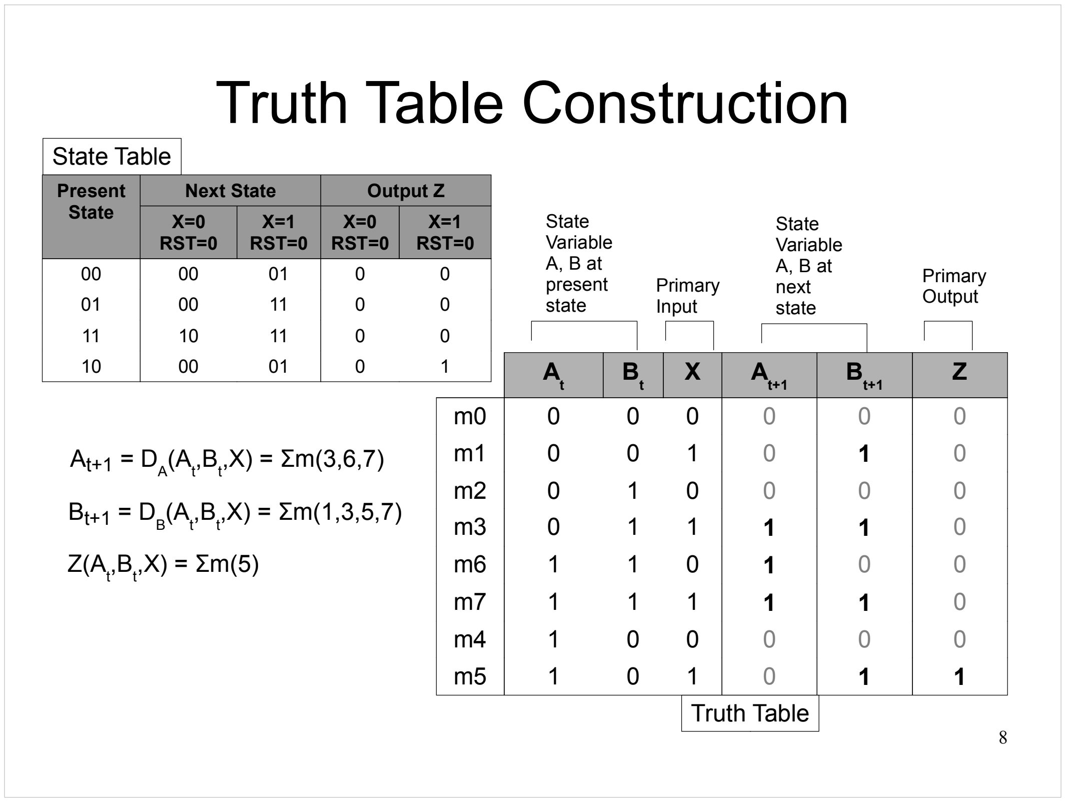
- At this state assignment steps we encode each state, denoted with letter, with some binary value. There are several way to assign binary value to a state. Some of them are as following.
  - Using counting order, e.g. A=00, B=01, C=10, D=11.
  - Using gray code, e.g. A=00, B=01, C=11, D=10.
  - Using one hot assignments, e.g. A=0001, B=0010, C=0100, D=1000

- If there are m number of states and minimum number of bit needed is n, then $2^n \geq m$.

- One hot assignment needs total m bit to denote m state. It needs maximum number of bits, but in many cases it reduce the logic circuit complexity for next state determination.

- There is no hard rules and methods to choose assignment strategies for state. A number of methods have been developed based on heuristics, however, there are beyond scope of this class.

- For this specific problem, coding has been done for the states using gray code. Intuitively, this coding method may work good since K-Map uses gray code, and we'll derive the next state logic using K-Map.



Truth Table Construction

State Table

Present State	Next State		Output Z	
	X=0 RST=0	X=1 RST=0	X=0 RST=0	X=1 RST=0
00	00	01	0	0
01	00	11	0	0
11	10	11	0	0
10	00	01	0	1

$A_{t+1} = D_A(A_t,B_t,X) = \Sigma m(3,6,7)$

$B_{t+1} = D_B(A_t,B_t,X) = \Sigma m(1,3,5,7)$

$Z(A_t,B_t,X) = \Sigma m(5)$

State Variable A, B at present state

Primary Input

State Variable A, B at next state

Primary Output

	A_t	B_t	X	A_{t+1}	B_{t+1}	Z
m0	0	0	0	0	0	0
m1	0	0	1	0	**1**	0
m2	0	1	0	0	0	0
m3	0	1	1	**1**	**1**	0
m6	1	1	0	**1**	0	0
m7	1	1	1	**1**	**1**	0
m4	1	0	0	0	0	0
m5	1	0	1	0	**1**	**1**

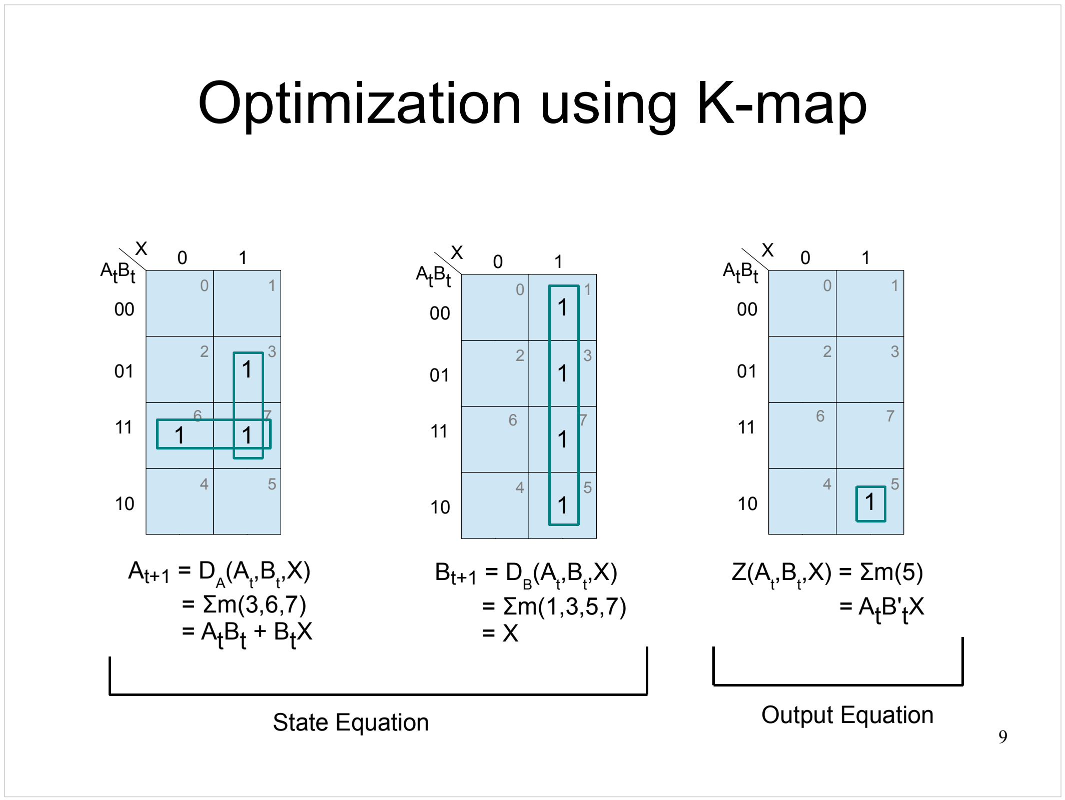
Truth Table

8

- Once the state assignments are done, the state table is rearranged to make truth table. We take each bit of the present state and the primary input as input for next state logic and the output logic.

- In this example, the present state bits are assumed in variables A_t and B_t (it is not related anyway to the state letter code A, B, C, D – we could have name these two variables A_t and B_t as K_t and V_t). These variables are also called state variable.The input for the truth tables are A_t , B_t and X. The outputs are A_{t+1} , B_{t+1} and Z. The bit combination A_{t+1} B_{t+1} will denote the encoded next state.

- From the truth table minterms are identified for each of the output variable and written in SOP format.



Optimization using K-map

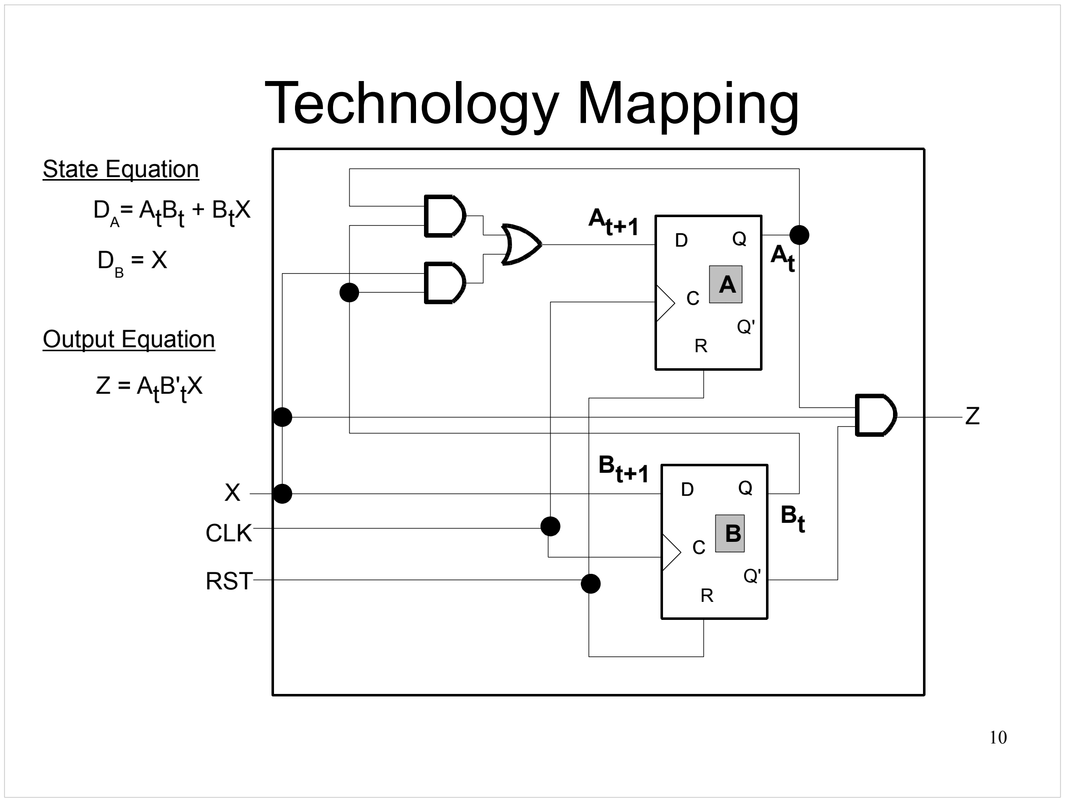
$A_{t+1} = D_A(A_t, B_t, X) = \Sigma m(3,6,7) = A_tB_t + B_tX$

$B_{t+1} = D_B(A_t, B_t, X) = \Sigma m(1,3,5,7) = X$

$Z(A_t, B_t, X) = \Sigma m(5) = A_tB'_tX$

- K-map optimization is used to determine reduced equation for each of the truth table output variable.

- The reduced equation for the state variable is also called state equation.



Technology Mapping

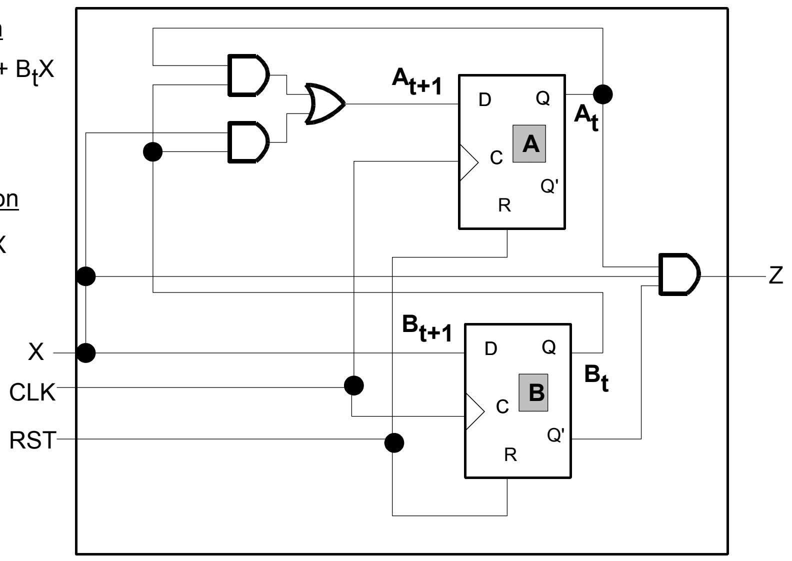
State Equation

$D_A = A_tB_t + B_tX$

$D_B = X$

Output Equation

$Z = A_tB'_tX$



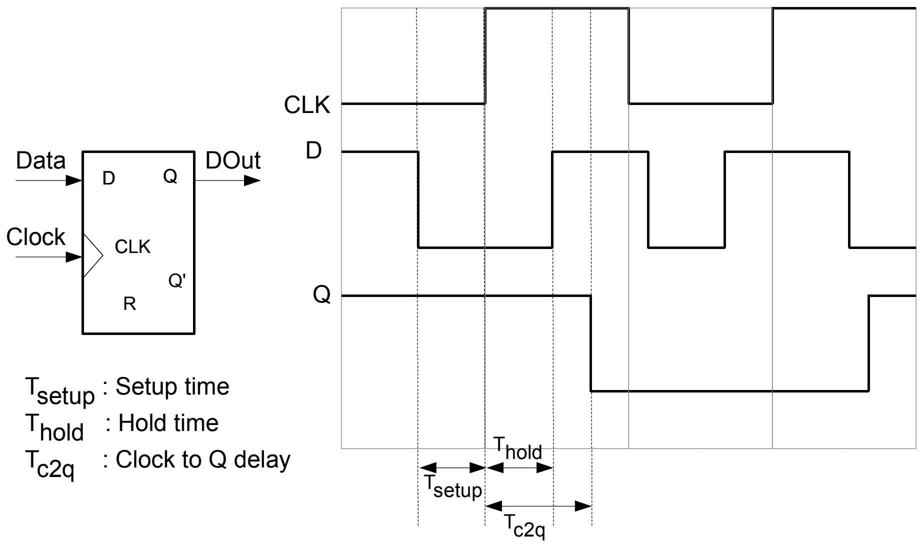
10

- We are assuming all types of gates are available for technology mapping of these equations.

- To store the two state bits we use two D-flipflops, one for state bit A and other for state bit B. The output of these two flops provide the current state. Based on current state and the primary input, the next state is determined through the combinational Logic circuit. This next state information will be available on the D-pins of the state storage flops which will be latched in the next clock cycle.

- The output Z is also determined using the current state bits and the primary input bit.

Flipflop Time Constraints



T_{setup} : Setup time
T_{hold} : Hold time
T_{c2q} : Clock to Q delay

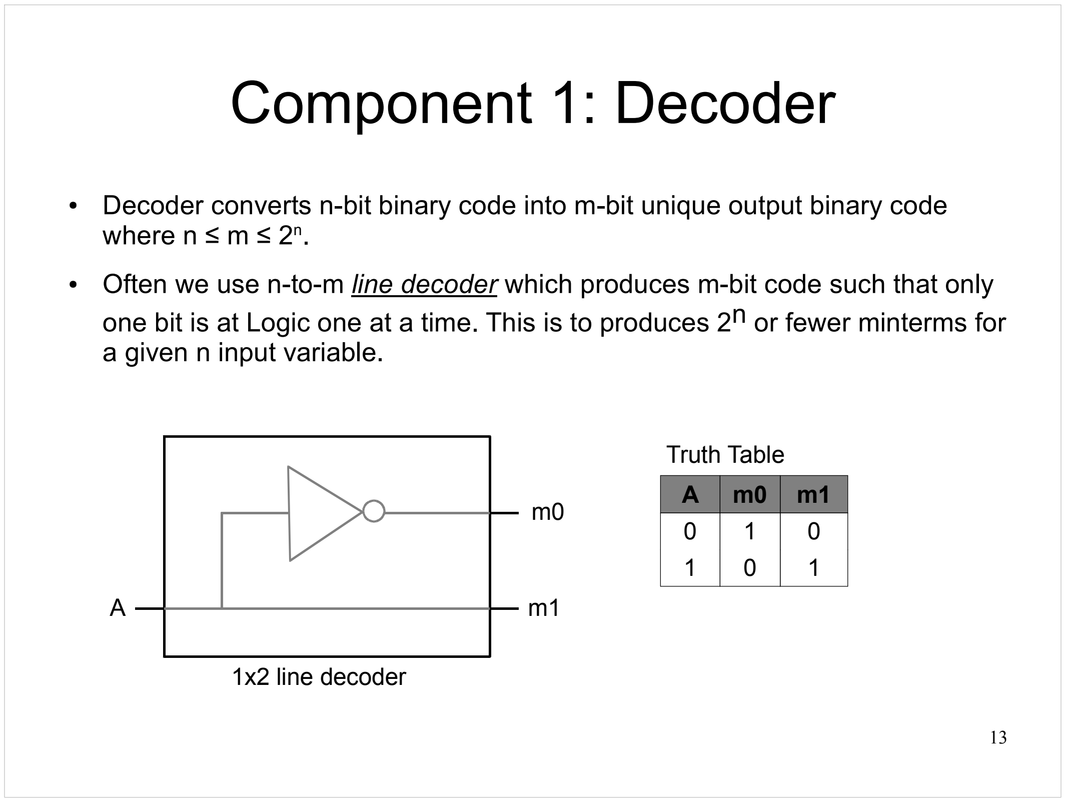
11

- Setup time is the minimum amount of time, with respect to the active clock edge, before which input data must be at a stable logic value.

- Hold time is the minimum amount of time, with respect to the active clock edge, for which input data must be at stable logic value after the clock edge.

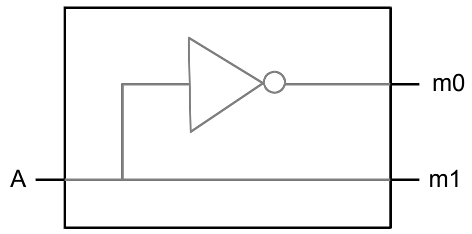
- C2Q or clock-2-q delay is the amount of time from the active clock edge, after which new data appears at the Q (the output) pin.

Digital Circuit Components ...



Component 1: Decoder

- Decoder converts n-bit binary code into m-bit unique output binary code where $n \le m \le 2^n$.
- Often we use n-to-m *line decoder* which produces m-bit code such that only one bit is at Logic one at a time. This is to produces 2^n or fewer minterms for a given n input variable.



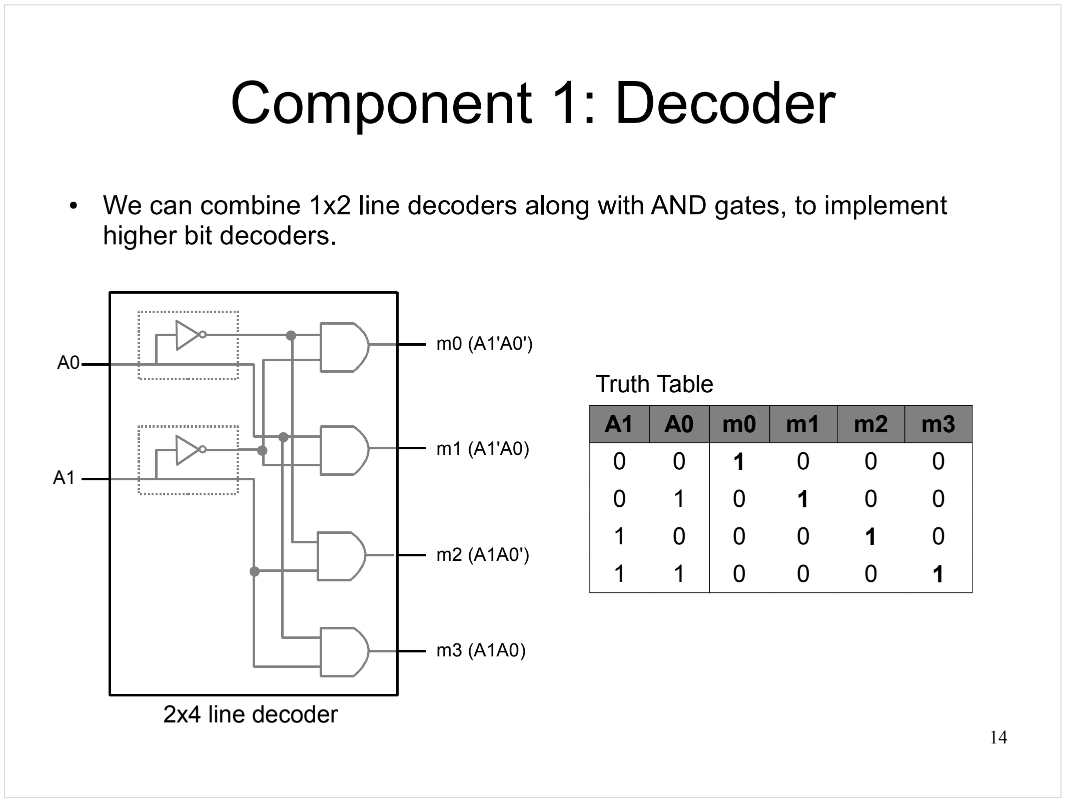
1x2 line decoder

Truth Table

A	m0	m1
0	1	0
1	0	1

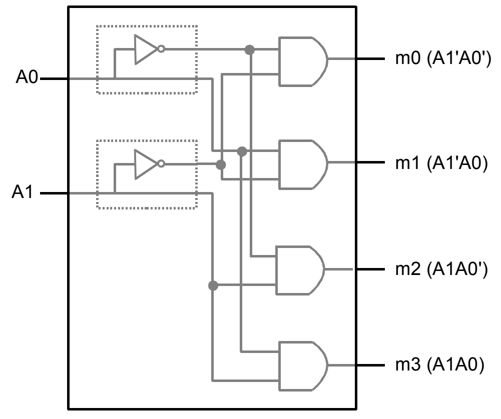
13

- For a 1x2 line decoder one inverter is sufficient to derive two minterms – A and A' in this example. If A=0, it is the m0 minterm and the decoder will give m0 a 1 and m1 as 0. On the other hand if A=1, it is the m1 minterm and the decoder will give m0 as 0 and m1 as 1.



Component 1: Decoder

- We can combine 1x2 line decoders along with AND gates, to implement higher bit decoders.



2x4 line decoder

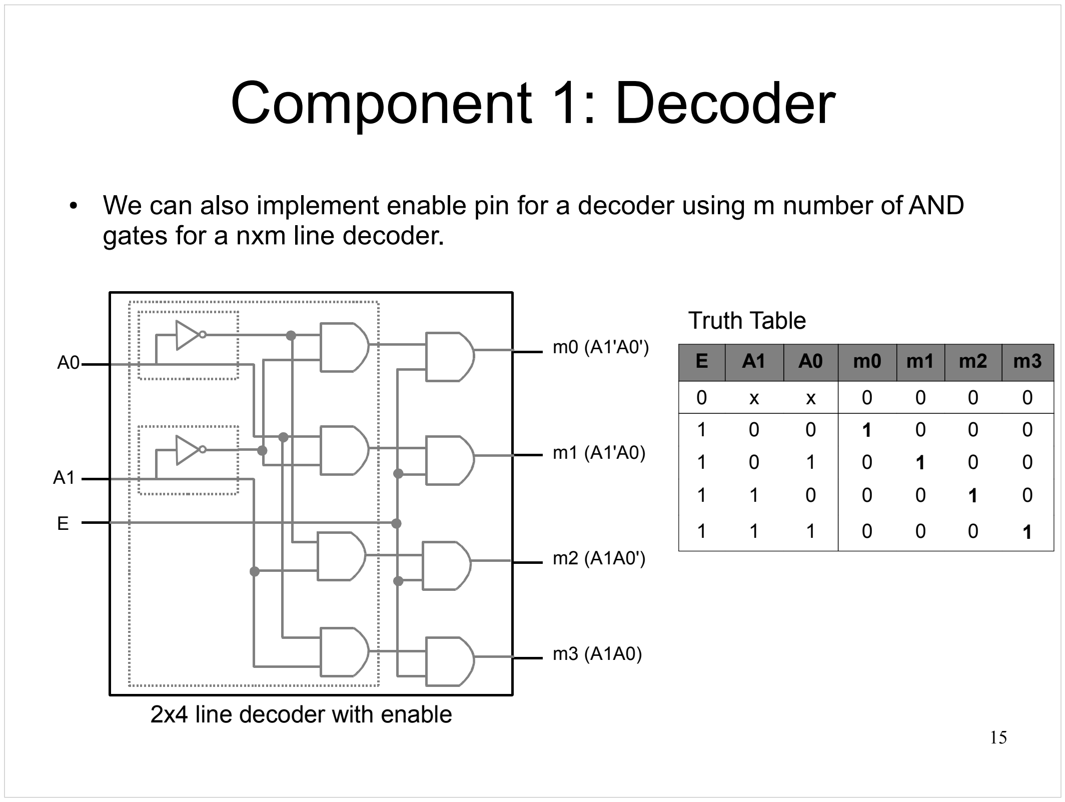
Truth Table

A1	A0	m0	m1	m2	m3
0	0	**1**	0	0	0
0	1	0	**1**	0	0
1	0	0	0	**1**	0
1	1	0	0	0	**1**

14

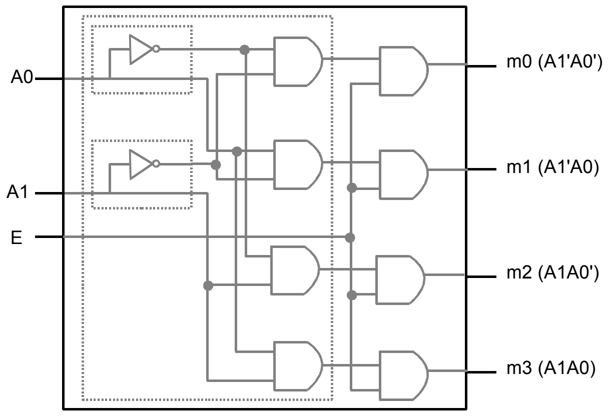
- The output of the 2x4 line decode is one-hot. This means out of 4 output only single output will be at logic 1 and all other will be at logic 0 for a given input. The output pattern is captured in the truth table.

- 2x4 line decoder is implemented using two 1x2 decoder. Each decoder will give the minterm for single variable connected to it. Hence two decoder will produce the term A0, A0' , A1, A1'. These four terms are then combined by four AND gates with get the terms A1'.A0', A1'.A0, A1.A0' and A1.A0. These are nothing but the minterms m0, m1, m2 and m3 of the combined Boolean variable A1 and A0.



Component 1: Decoder

- We can also implement enable pin for a decoder using m number of AND gates for a nxm line decoder.



2x4 line decoder with enable

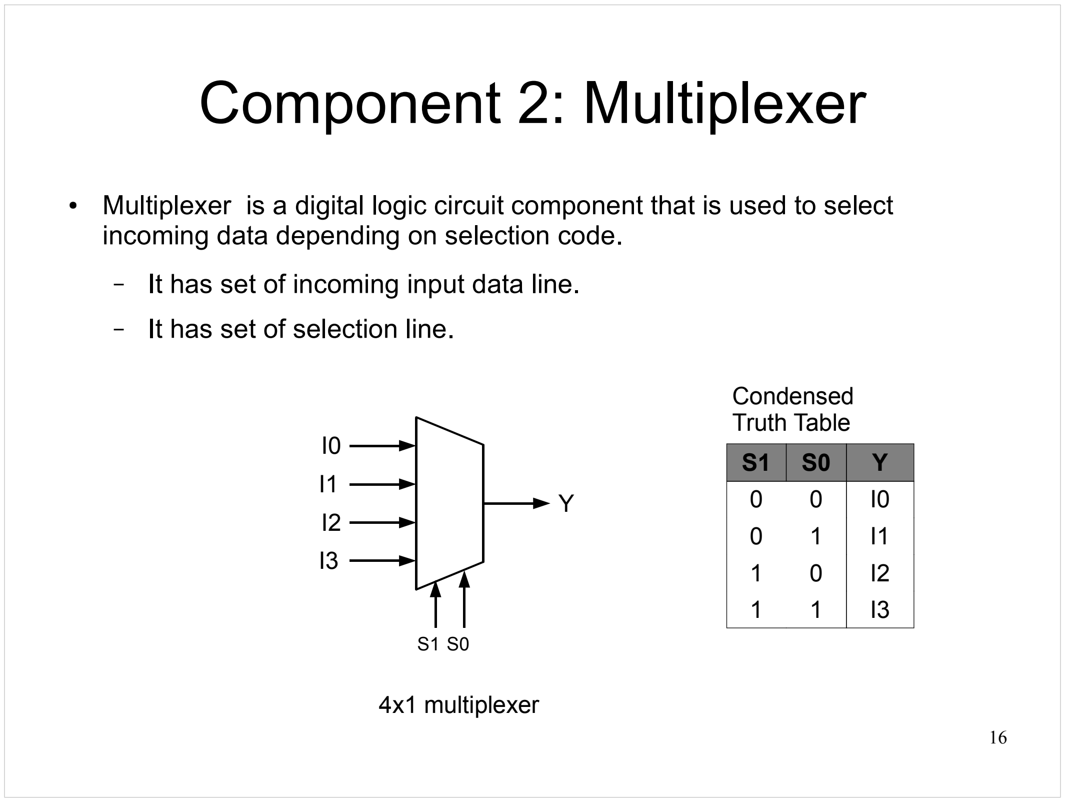
Truth Table

E	A1	A0	m0	m1	m2	m3
0	x	x	0	0	0	0
1	0	0	**1**	0	0	0
1	0	1	0	**1**	0	0
1	1	0	0	0	**1**	0
1	1	1	0	0	0	**1**

15

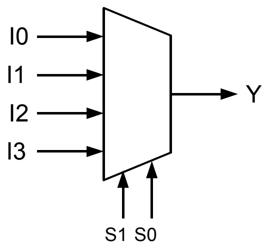
- We can cascade another four AND gates to implement enable signal E for the decoder. If E is 0, the output of the decoder should be all 0 no matter what the other inputs are. Once E is set to 1, decoder starts functioning and produce correct result at the outputs.

- The output of the decoder of the slide 13 is gated by signal E using AND gate. Once the E is one, the value of the decoding will pass to the final output. Final output will remain 0 otherwise.



Component 2: Multiplexer

- Multiplexer is a digital logic circuit component that is used to select incoming data depending on selection code.
  - It has set of incoming input data line.
  - It has set of selection line.



4x1 multiplexer

Condensed Truth Table

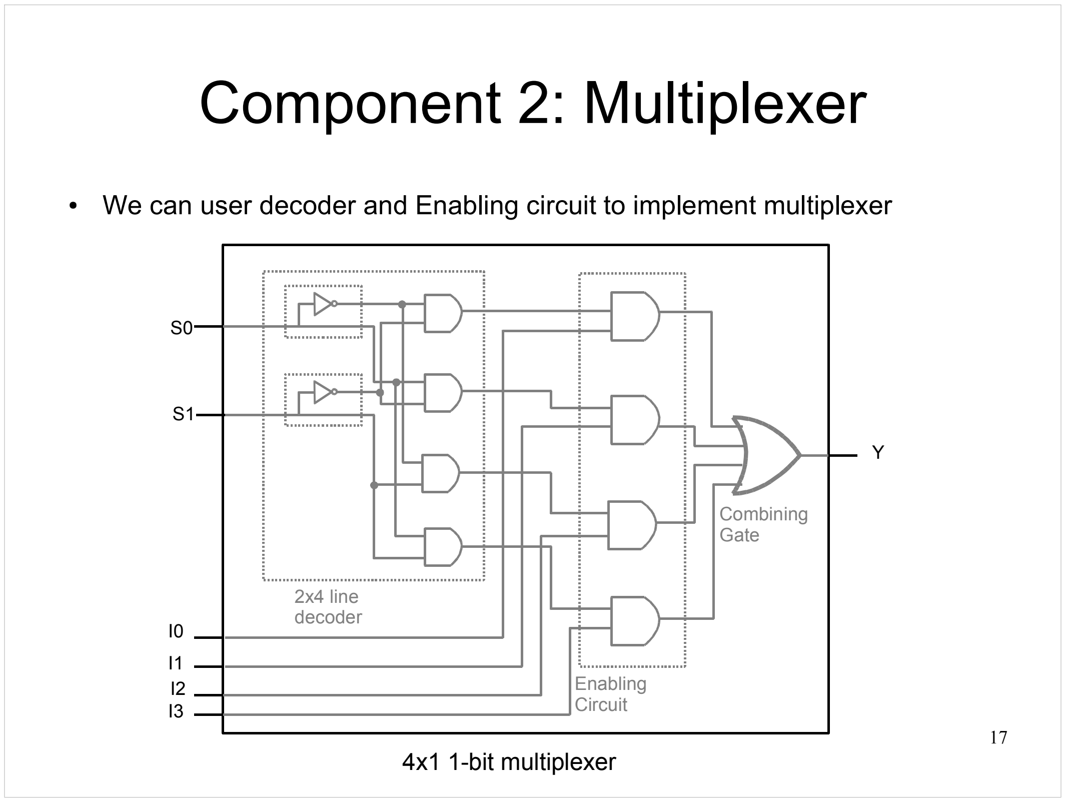
S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

16

- Conceptually multiplexer implements decision algorithm. The example mux (short form of multiplexer) in this slide, implements simple case statement like following.

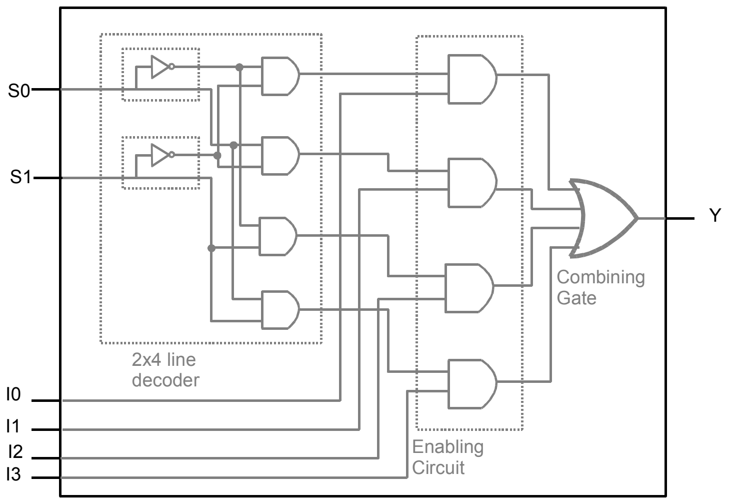
```
Case (S1 S0)
Begin
   00: Y = I0;
   01: Y = I1;
   10: Y = I2;
   11: Y = I3;
End
```

- This is one of the most important component in computer architecture. This gate enable controlling of right control flow and data flow through a computer.



Component 2: Multiplexer

- We can user decoder and Enabling circuit to implement multiplexer



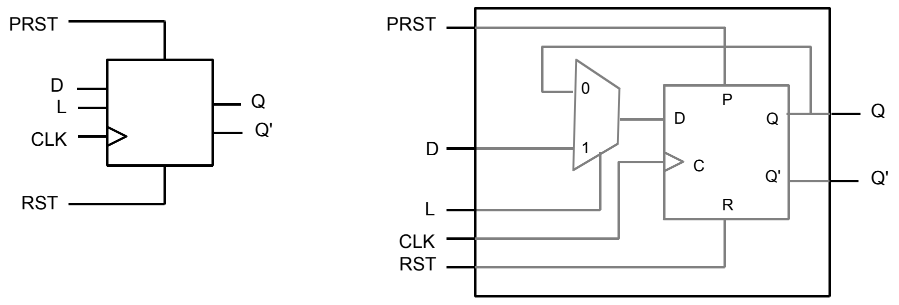
4x1 1-bit multiplexer

17

- To implement mux, decoder and enabling circuit can be re-used with minor modification. Since the line decoder produces one hot output corresponding the input binary pattern, the decoder output can be used to select one of the input data to send for final output. The outputs from the enabling circuit is connected to OR gate in next stage. Since, except for the enabled data, all the other outputs from the enable circuit will be logic 0, the OR gate will pass Boolean logic value of the selected signal.

Component 3: Register

- Registers are the sequential component which can store logic value and new logic value can be loaded upon request.

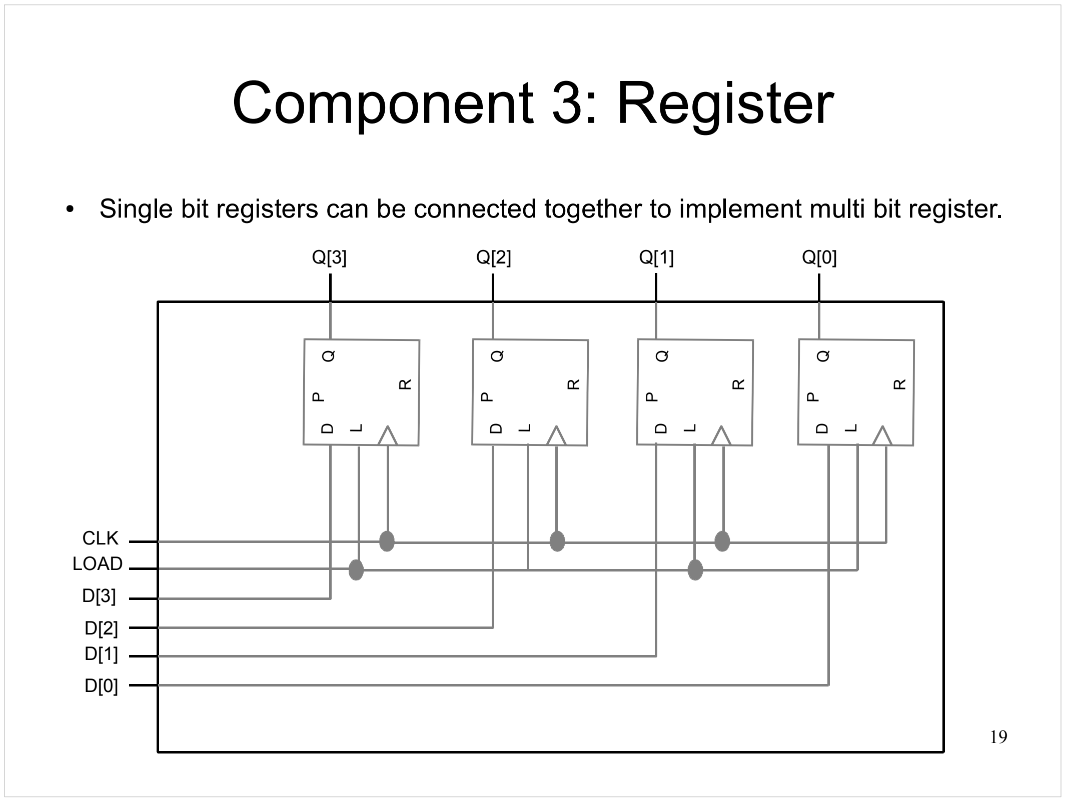


18

- In a computer operation, it is not always desired to load whatever value comes into the data input of a flipflop. Register provides one extra control 'L' (or sometime it is explicitly called 'load') for selectively loading data into a flipflop. Storage element with selectable load is called a register.

- The selective load feature is implemented using a combination of flipflop and a 2x1 mux. The output of the mux is connected to the data input of the flop. One of the input for the mux is the output of the flop and the other input is the incoming data input. The mux selection control is the load input value.

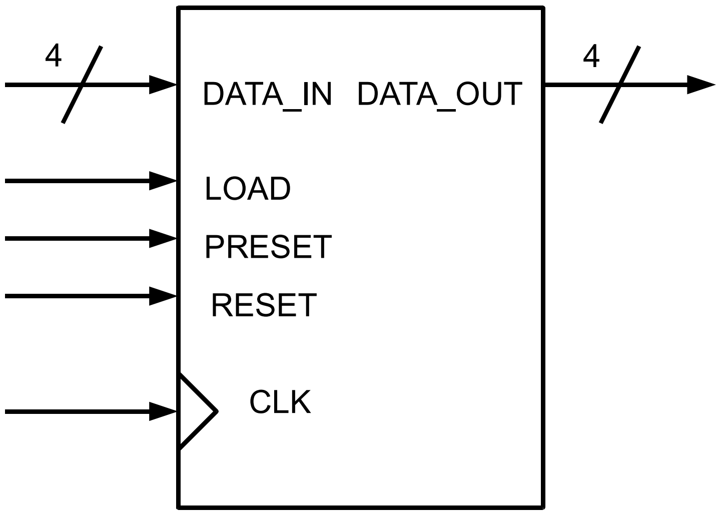
- If the load control logic 'L' is 0, the output of the flop is feedback into its data input. As a result, with L=0, output in next clock cycle is same as the value at previous clock cycle (data is held). With L=1, the incoming data value is placed on the data input of the flop and thus the flop is loaded with new value that is placed on the register's data input.

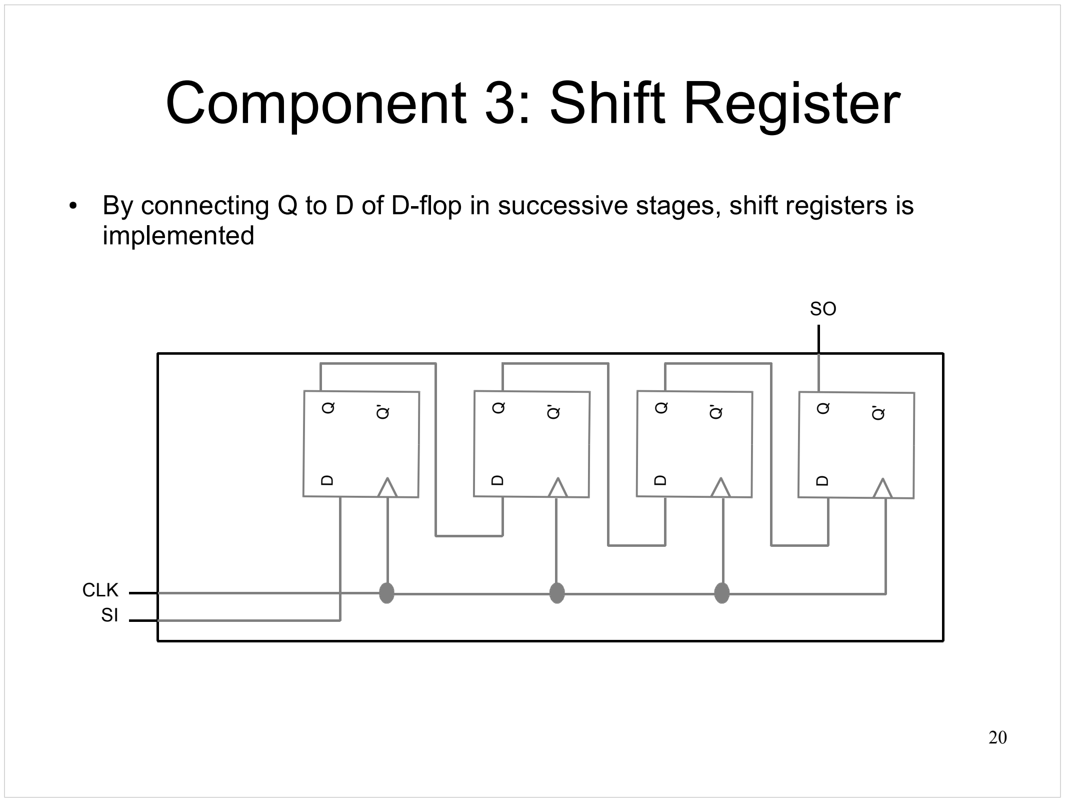


Component 3: Register

- Single bit registers can be connected together to implement multi bit register.

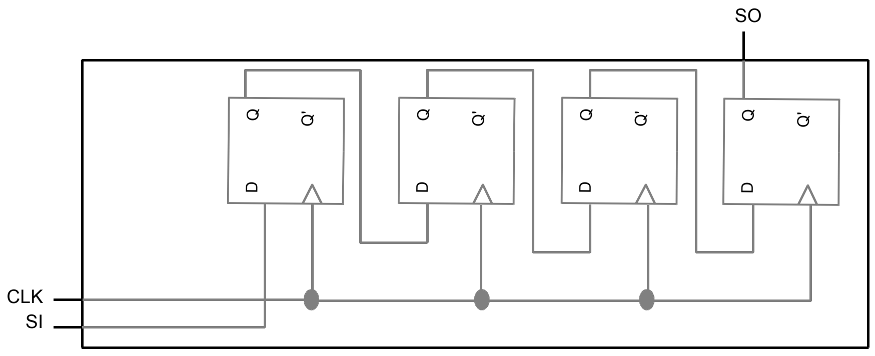
- Multiple registers can be grouped together to form a multi-bit register. The clock and load signal is shared among the flops in the group. The following is a schematic diagram for register. The PRESET and RESET signals are asynchronous. PRESET=1 will set the output to 0xF and RESET=1 will set the output to 0x0. Both the input and output are 4-bit wide.





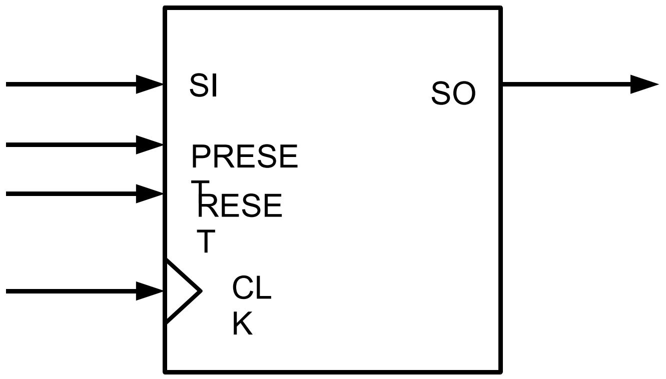
Component 3: Shift Register

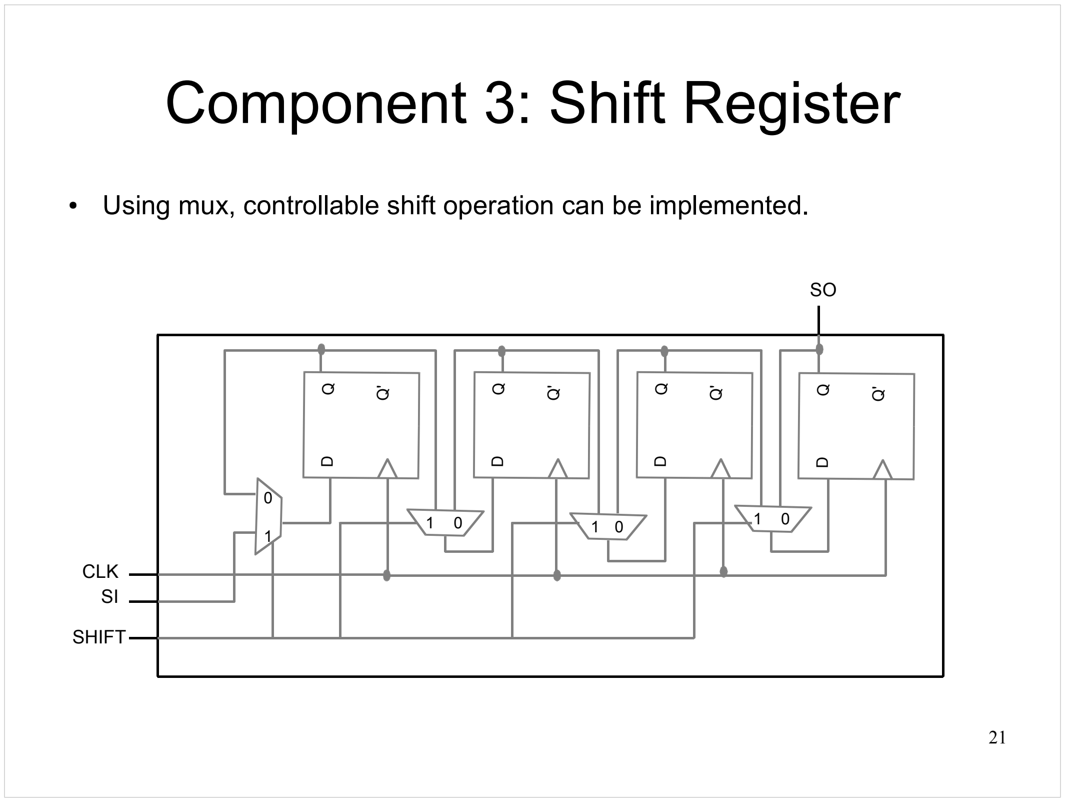
- By connecting Q to D of D-flop in successive stages, shift registers is implemented



20

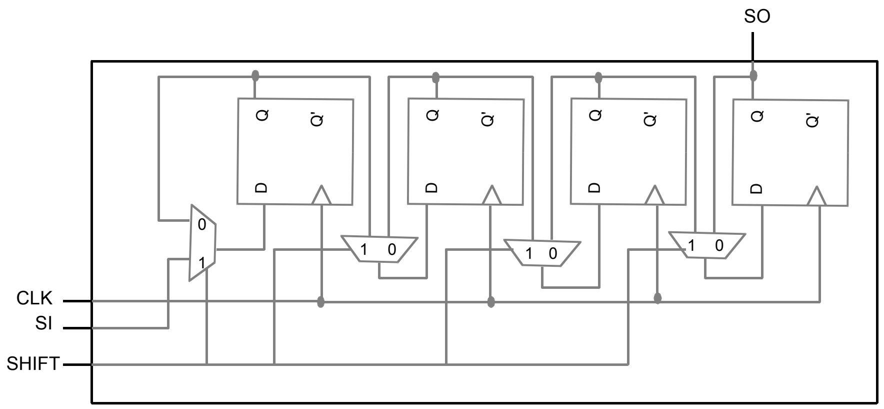
- If the output of 1-bit register is connected to input of the register in the next bit position, we get a shift register. There is one serial input SI and serial output SO. A continuous bit stream comes onto SI and after 4 clock cycle (in this example) a continuous bit stream occurs in FIFO (First In First Out) manner. At each clock cycle data from one flop is transferred to the next flop in the stage. For a n-bit shift register, it takes n clock cycle for the first bit of SI to be appeared at the SO. However, with this arrangement there is no control of when to start the shift and when to stop it.





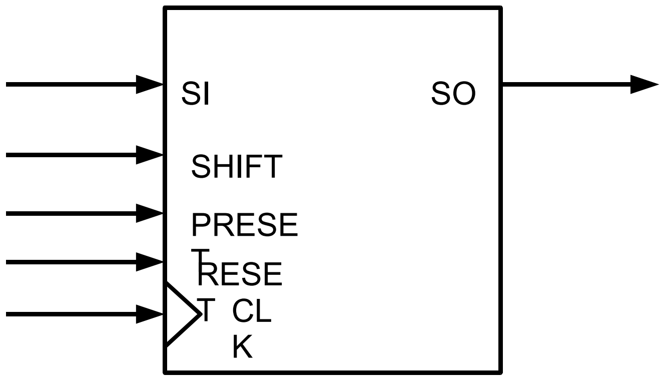
Component 3: Shift Register

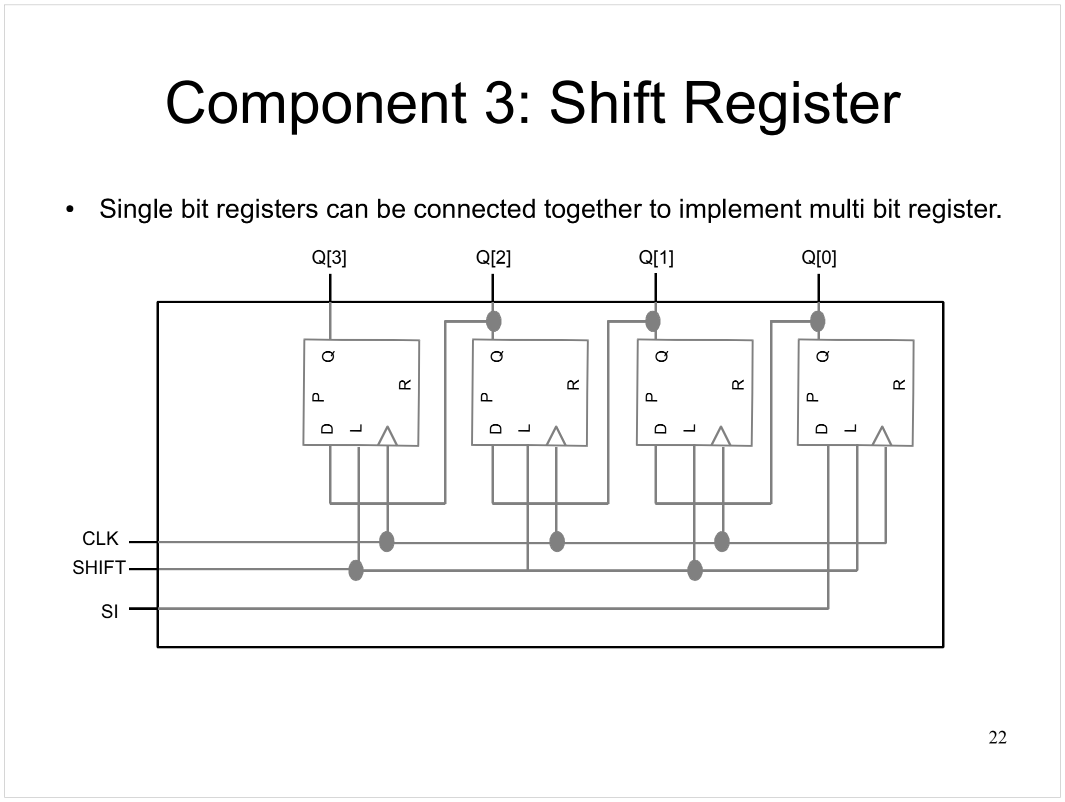
- Using mux, controllable shift operation can be implemented.



21

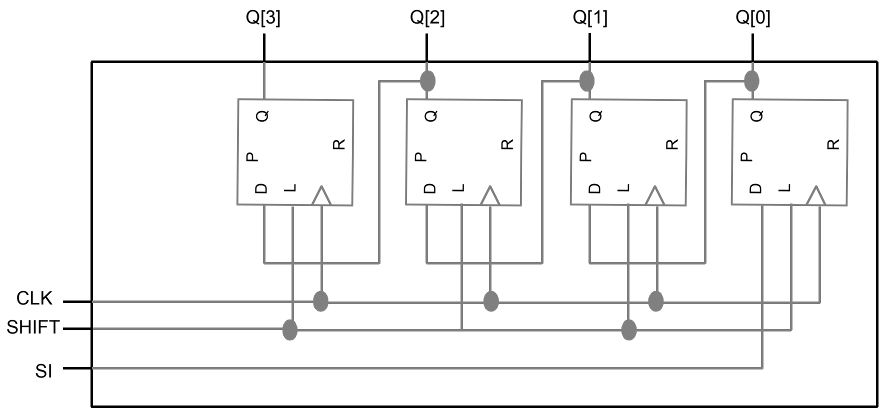
- Mux based shift control can be implemented. In this case, an external control signal 'SHIFT' is added. If SHIFT=0, the data holds and not shift operation is performed. If SHIFT=1, shift operation resumes. Incoming data on a register is controlled by a mux for each register, so that data source can either be itself (feedback from its own output) or from previous flop or SI (in case of the the first register). If the SHIFT is 1, data on a register comes from the previous stage register or SI and hence the shift operation resumes. If SHIFT is 0, data comes from itself for a register and thus hold is performed. However, for this circuit, there is no way to load a predefined bit pattern in one clock cycle.





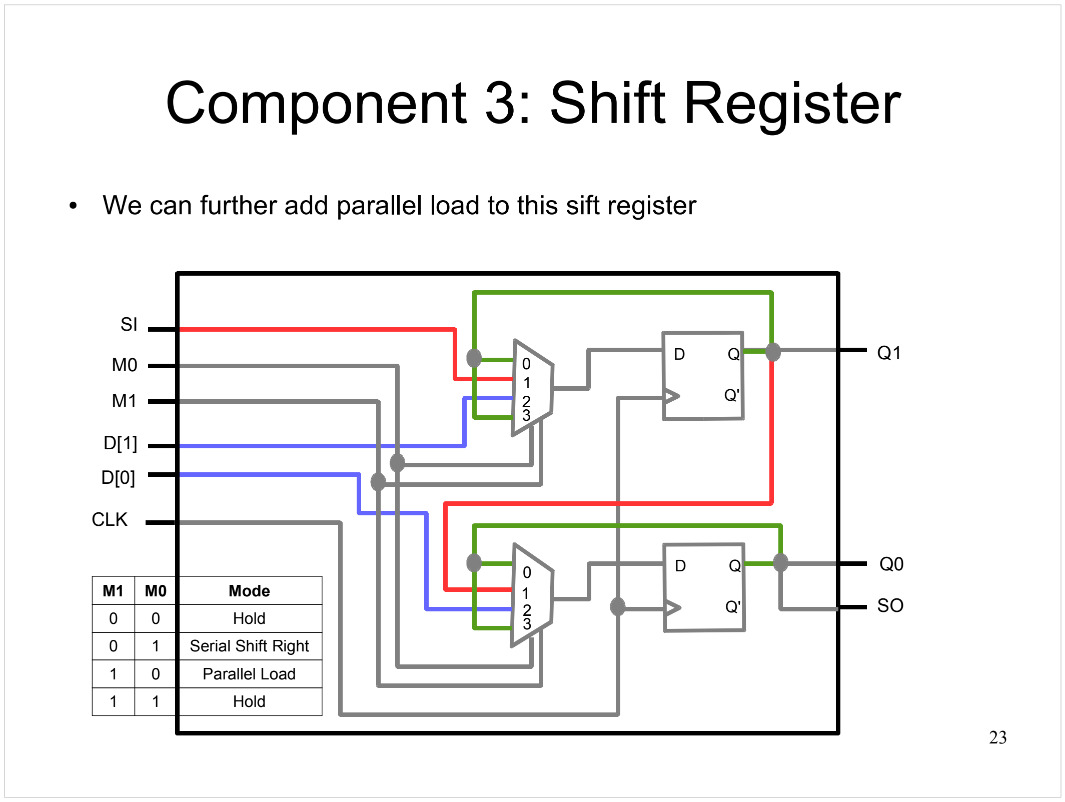
Component 3: Shift Register

- Single bit registers can be connected together to implement multi bit register.



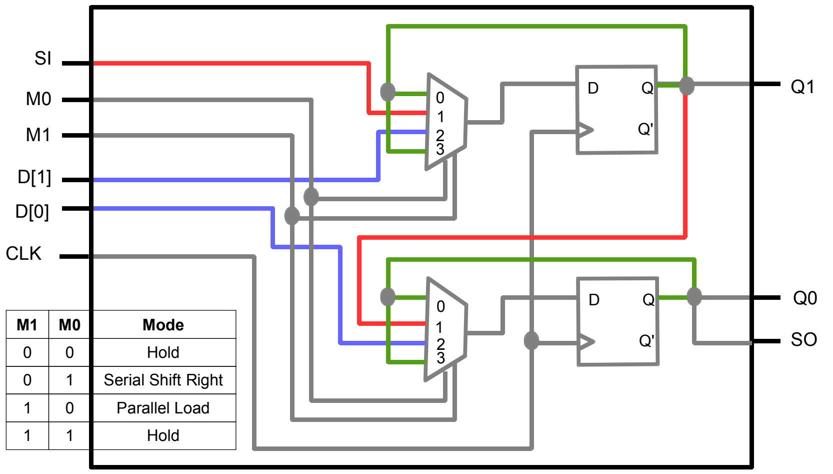
22

- Mux and D flipflop combination is nothing but placing 'register' elements in the shift register logic circuit.



Component 3: Shift Register

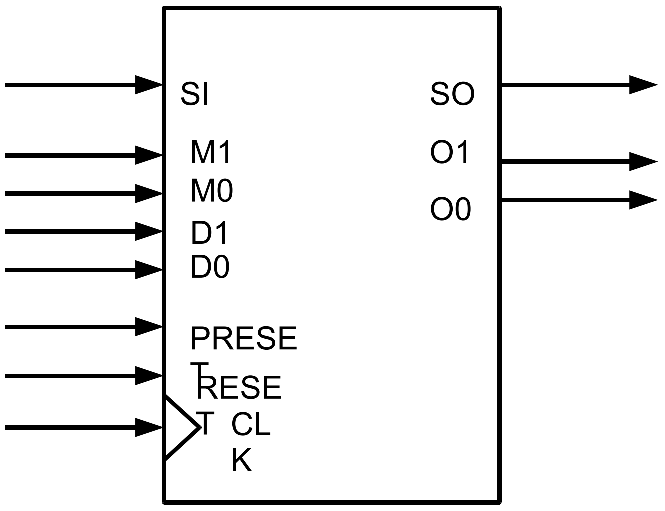
- We can further add parallel load to this sift register

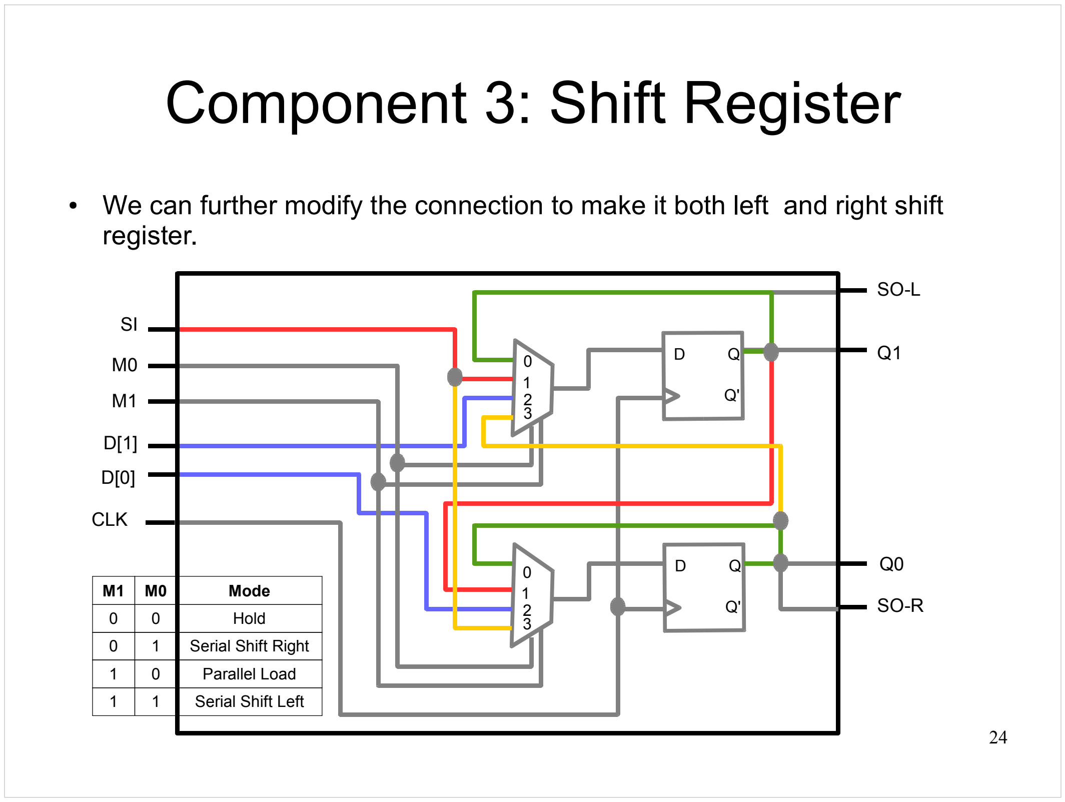


M1	M0	Mode
0	0	Hold
0	1	Serial Shift Right
1	0	Parallel Load
1	1	Hold

23

- Using a 4x1 mux, a parallel loadable shift register can be implemented. In this case, it has two control signal (or mode signal) bit M1 and M0. The data sources for a single bit register is either it self (hold operation), or serial in / data from previous register (for shift operation) or independent input assigned for that register (parallel load). In the above circuit, M1M0 value '00' implements hold operation, where '01' resumes shift operation and '10' will load independent data from input. The feedback data is also connected to the 4th selected point on the mux to make sure '11' control pattern does not do any unexpected operation. Since it is connected to the feedback of the register, '11' will perform hold operation.





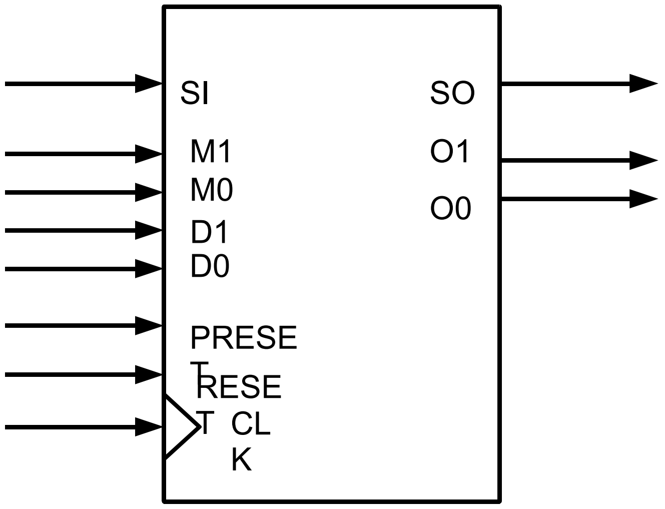
Component 3: Shift Register

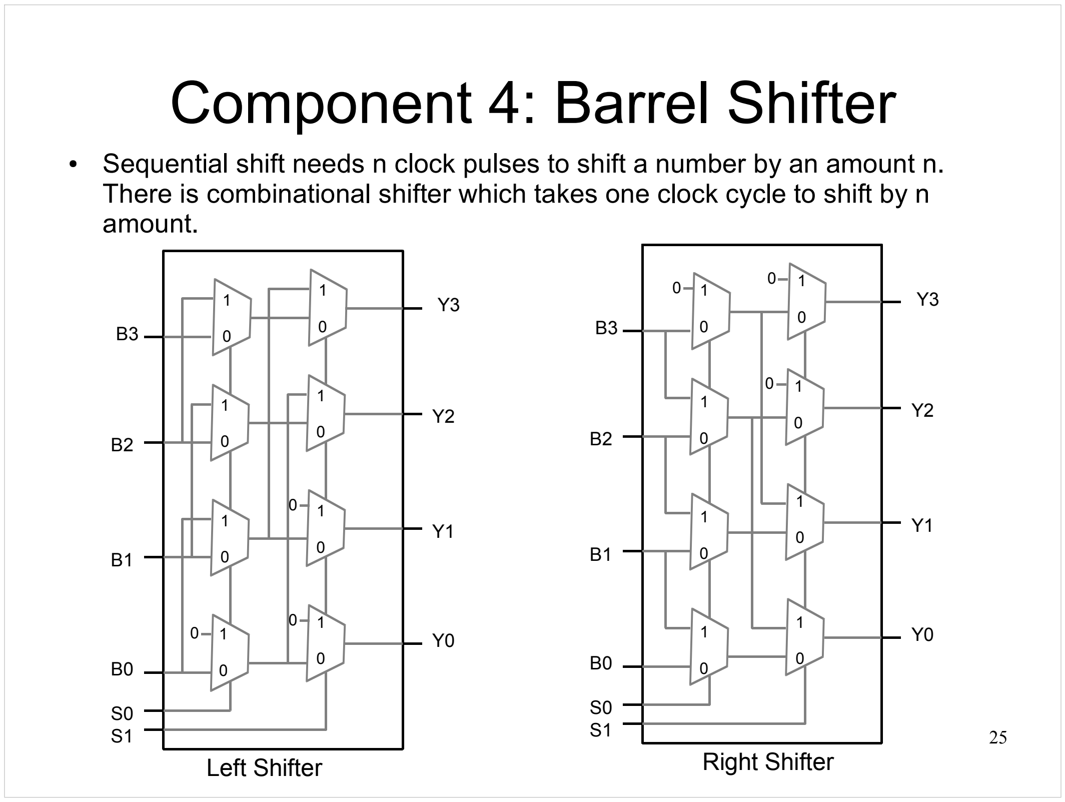
- We can further modify the connection to make it both left and right shift register.

M1	M0	Mode
0	0	Hold
0	1	Serial Shift Right
1	0	Parallel Load
1	1	Serial Shift Left

24

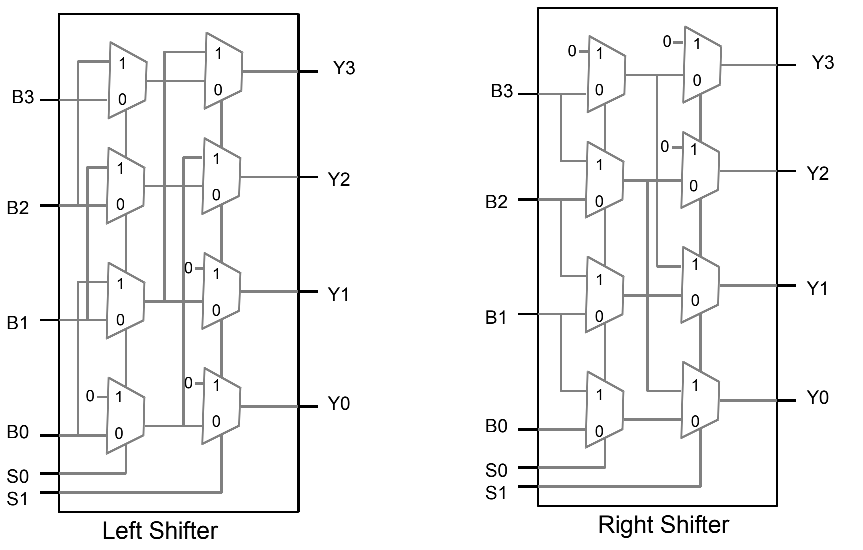
- If we connect the register data input to the selection mux from next stage to previous stage we can implement both left shift and right shift. In this case M1M0 as '11' with perform left shift where '00' will perform the right shift. The interface does not have any change with respect to the previous register.





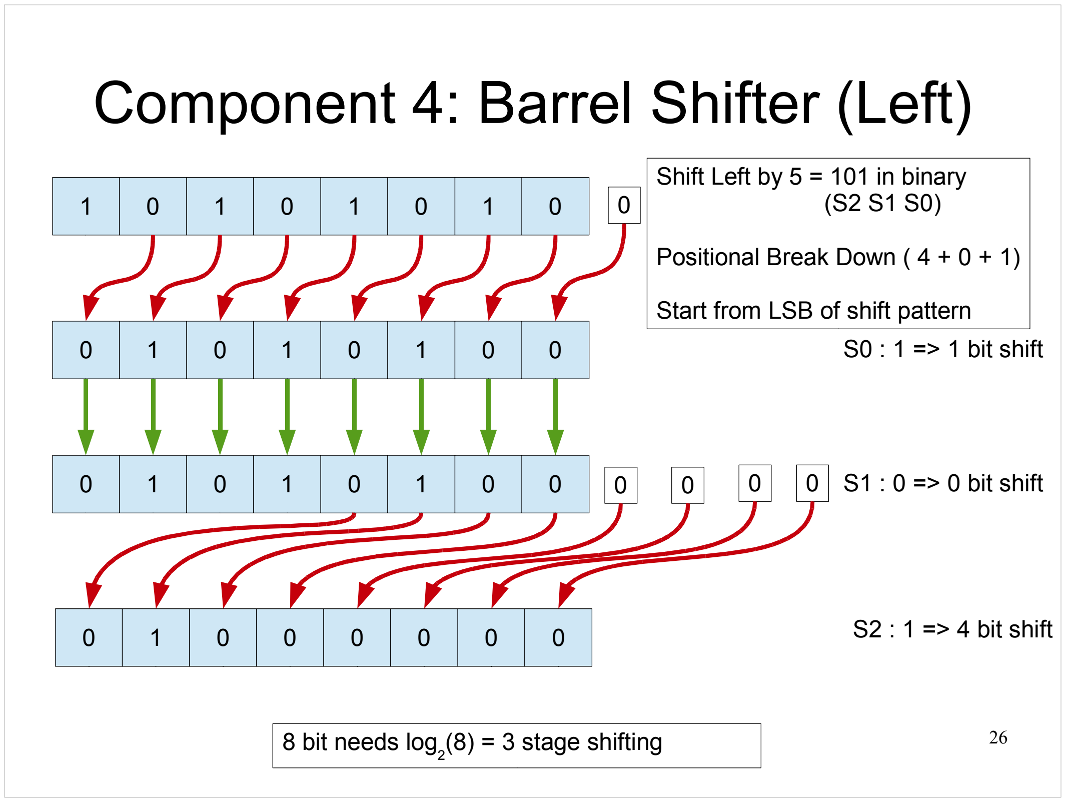
Component 4: Barrel Shifter

- Sequential shift needs n clock pulses to shift a number by an amount n. There is combinational shifter which takes one clock cycle to shift by n amount.



25

- The register based shifter takes n clock cycle for shifting a data by n bit position. We can implement mux based combinational shifter for a faster turn around. The idea is that the shift amount is encoded in a binary form. For example, a 2-bit shift amount 11 means 3 bit position shift. If we review closely, this 3 comes from each bit position value 2 + 1. Hence for a 1st bit position value 1 we need to shift the number by 1 position and for a 2nd bit position with value 1 we need to shift by 2 position. For a m-bit selection code, we arrange m number of mux group, each group containing n number of mux (assuming the number to be shifted is n-bit number). Hence it needs (m*n) number of mux to implement n-bit shifter with m-bit shift amount code. The input for mux are such arranged that for the first stage it shifts the number by 1 position if needed, for the 2nd stage it shifts the number by 2 position, for the 3rd stage it shifts the number by 4 position and so on. There can be a separate left shifter and right shifter based what type of shift operation is performed. For the shift operation, in this case, the vacant bit positions are filed with 0.



Component 4: Barrel Shifter (Left)

Shift Left by 5 = 101 in binary (S2 S1 S0)

Positional Break Down (4 + 0 + 1)

Start from LSB of shift pattern

8 bit needs $\log_2(8)$ = 3 stage shifting

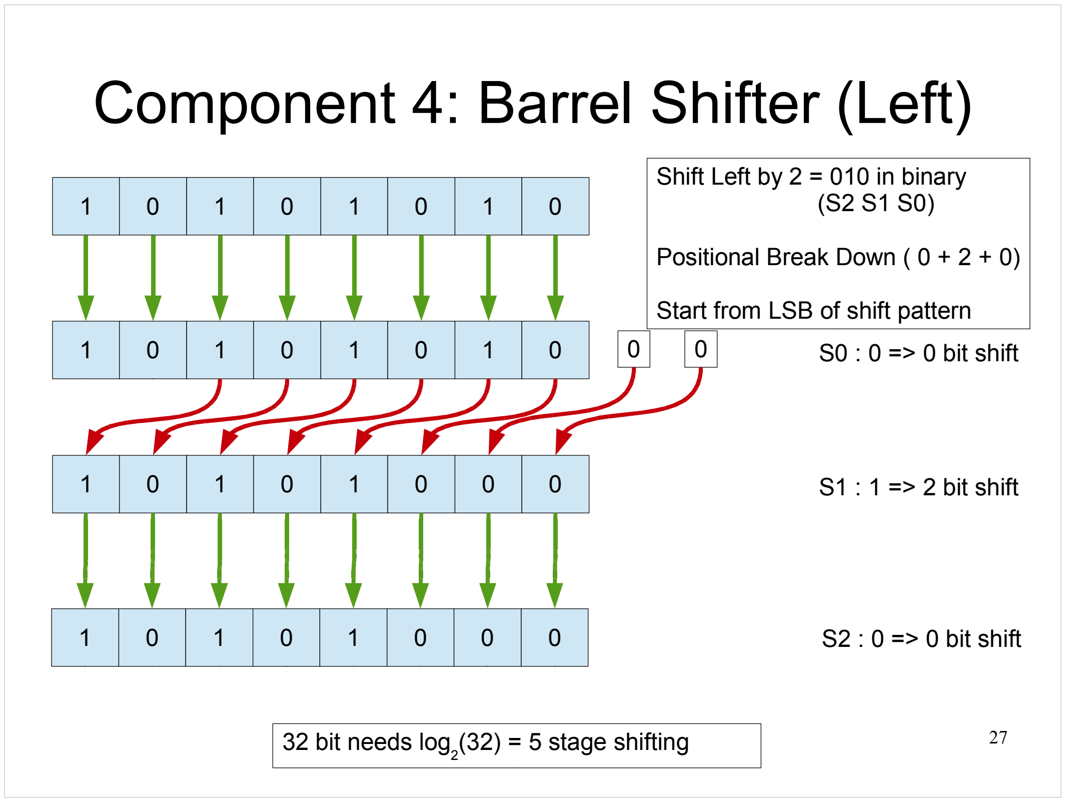
26

- Lets take this example of shifting a bit pattern 10101010 by 5. 5 can be represented as 101 in binary. Lets assume this three bit position of 101 is represented with Boolean variable S2, S1, S0.

- Using positional weight (2^n , where n is the bit position starting from 0 at LSB)
  - S0=1 will cause the pattern to shift by its positional weight 1. If S0=0, there will be no shifting of this pattern.
  - Similarly S1=1 will cause 2 bit shift of the pattern obtained from stage of S0 and S1=0 will cause no shift.
  - Likewise, S2=1 will cause 4 bit shift of the pattern obtained from stage of S1 and S2=0 will cause no shift.

- For this shift amount 5 (101 in binary) The pattern will be shifted at S0 stage by 1 bit and at S2 stage by 4 bit (resulting total 5 bit shift).

- For a 8-bit number max number of bit needed to represent shift amount is 3 ($\log_2 8 = 3$) which can represent a bit shift amount 0-7. Any larger amount will cause the entire result to be 0.



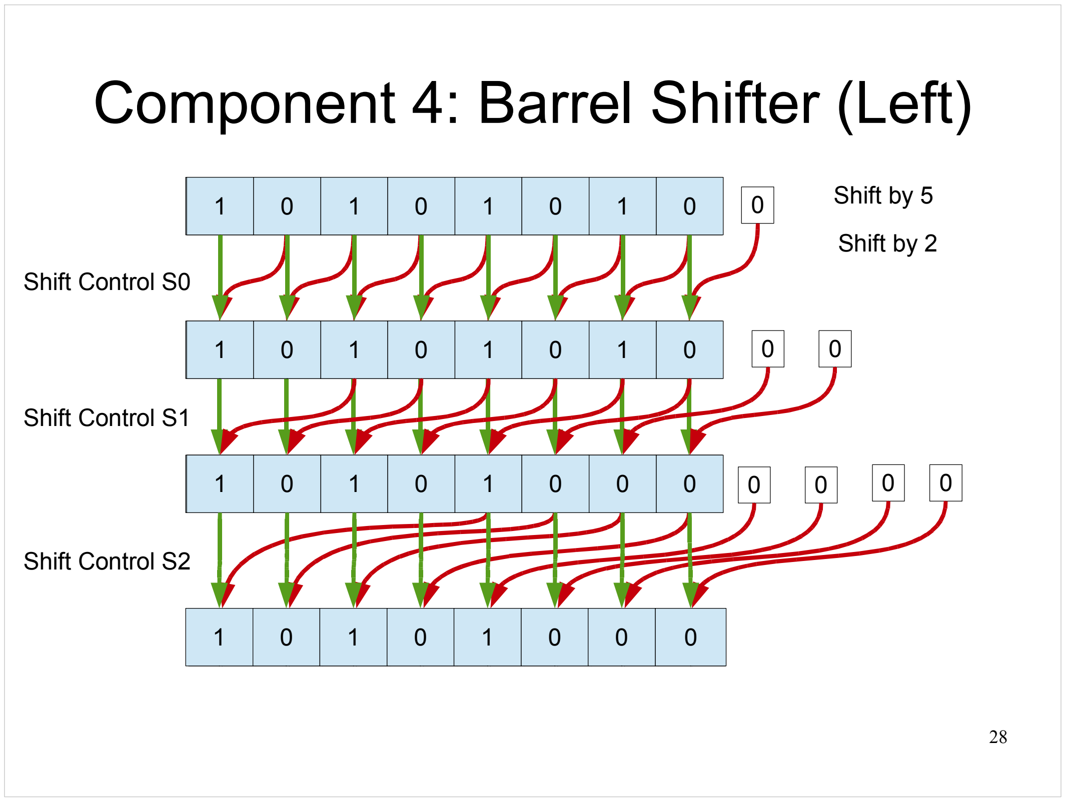
Component 4: Barrel Shifter (Left)

Shift Left by 2 = 010 in binary (S2 S1 S0)
Positional Break Down (0 + 2 + 0)
Start from LSB of shift pattern
1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0
0
0
S0 : 0 => 0 bit shift
1 0 1 0 1 0 0 0
S1 : 1 => 2 bit shift
1 0 1 0 1 0 0 0
S2 : 0 => 0 bit shift
32 bit needs log2(32) = 5 stage shifting
27

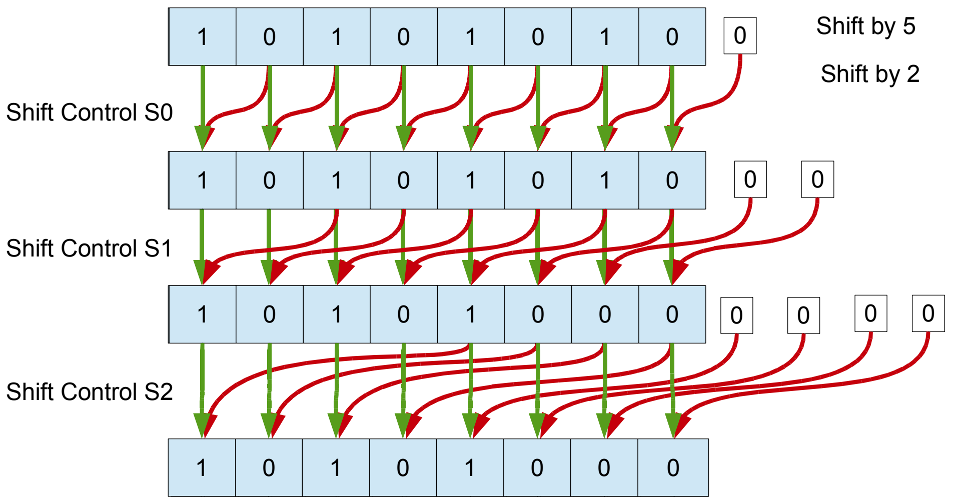
- Lets take similar example of shifting a bit pattern 10101010 by 2. 2 can be represented as 010 in binary.

- For this shift amount 2 (010 in binary) the pattern will be shifted at S1 stage only by 2 bit.

- For a 32-bit number max number of bit needed to represent shift amount is 5 ($\log_2 32 = 5$) which can represent a bit shift amount 0-31. Any larger amount will cause the entire result to be 0.



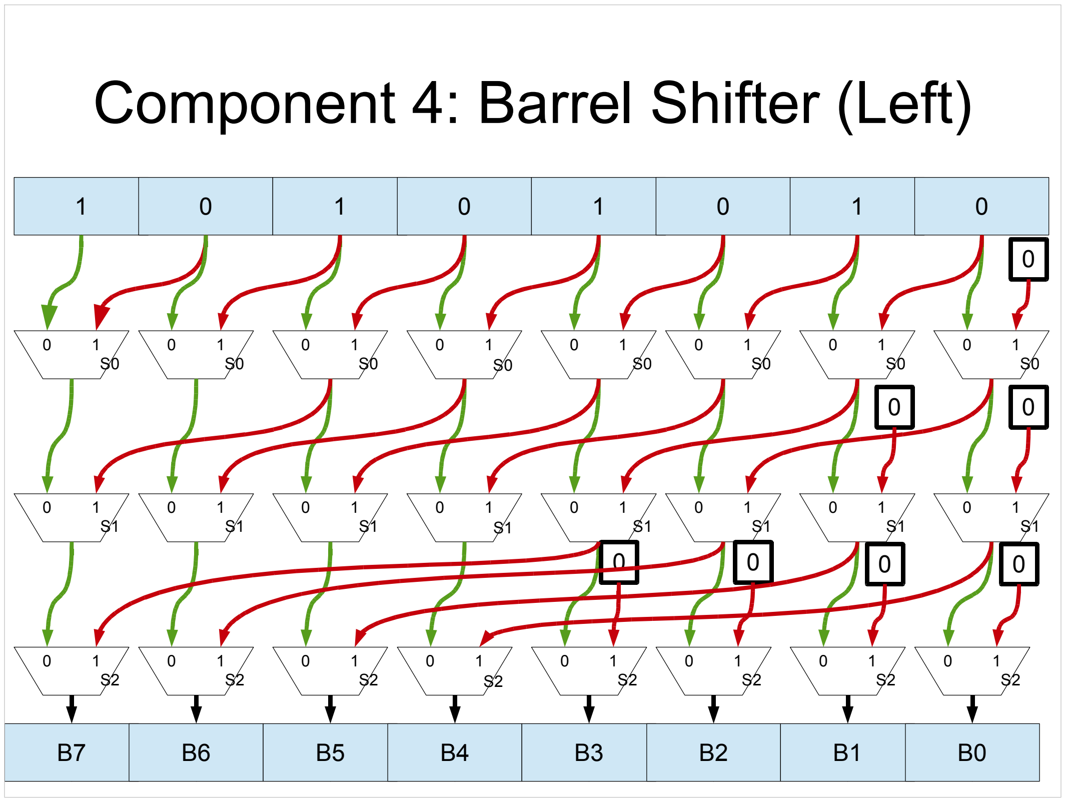
Component 4: Barrel Shifter (Left)



28

- If we superpose previous two examples, we can observe that at any stage bit source can be either from the same bit position of the previous stage or certain fixed bit position relative to current position. For example, at stage S1, bit 6 source can be either bit 6 of stage S0 (no shift case) or bit 4 of stage S0 (shift case). Similarly at S2, bit 6 source can be either bit 6 of stage S1 (no shift case) or bit 2 of stage S0 (shift case).

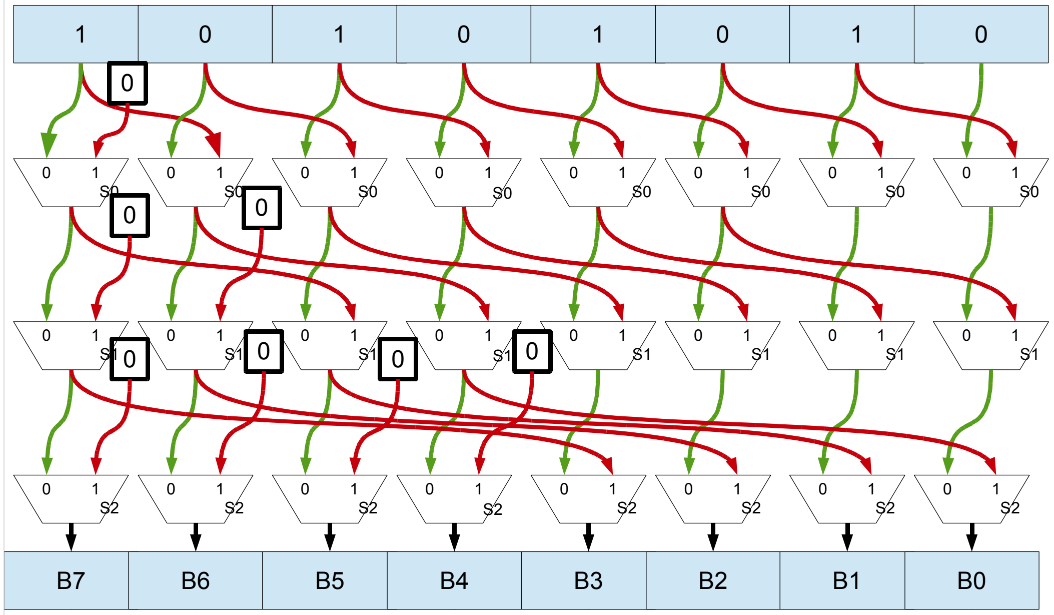
- In digital circuit, each conflicting source path for data must be mitigated using multiplexer.



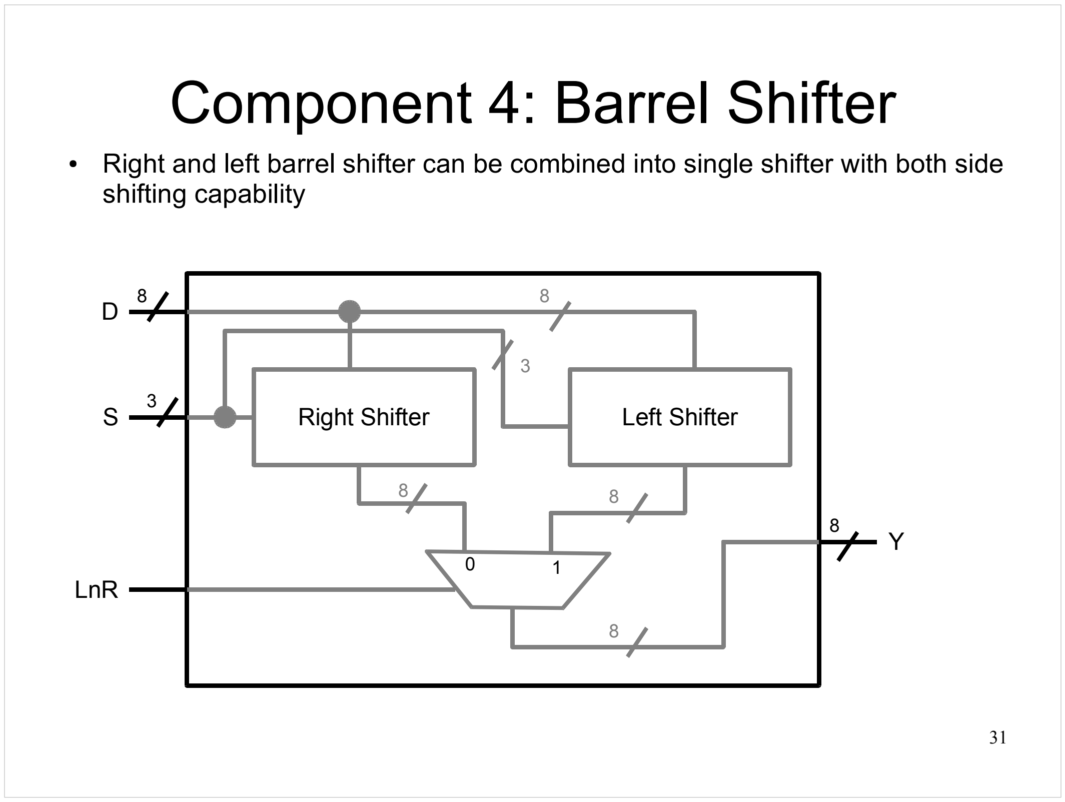
Component 4: Barrel Shifter (Left)

- In this 8-bit left barrel shifter, all the conflict of sources are mitigated using 2x1 1-bit multiplexer circuit. Each of the three stages have 8 mux each (total 3*8=24 mux). Each stage mux has selection control of selection bit (S2, S1, S0).

Component 4: Barrel Shifter (Right)

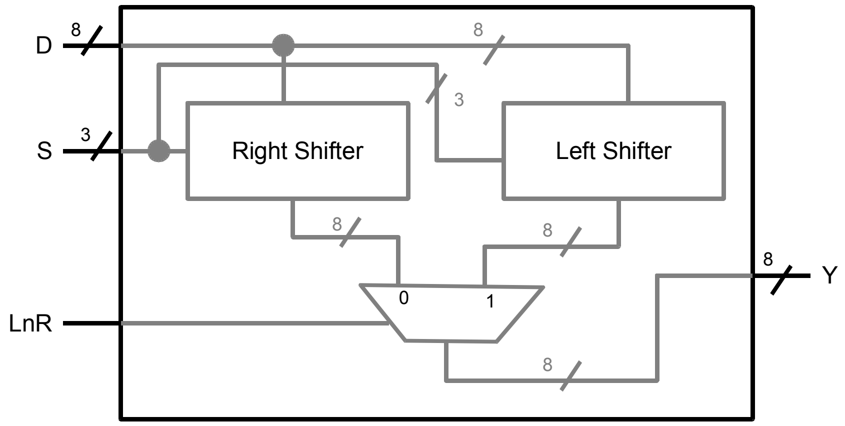


- Same list of components can be configured into 8-bit right shifter just by alternating the connection.
- It is possible to superpose these two type of barrel shifter into one by introducing more mux and selection using left-right shift.



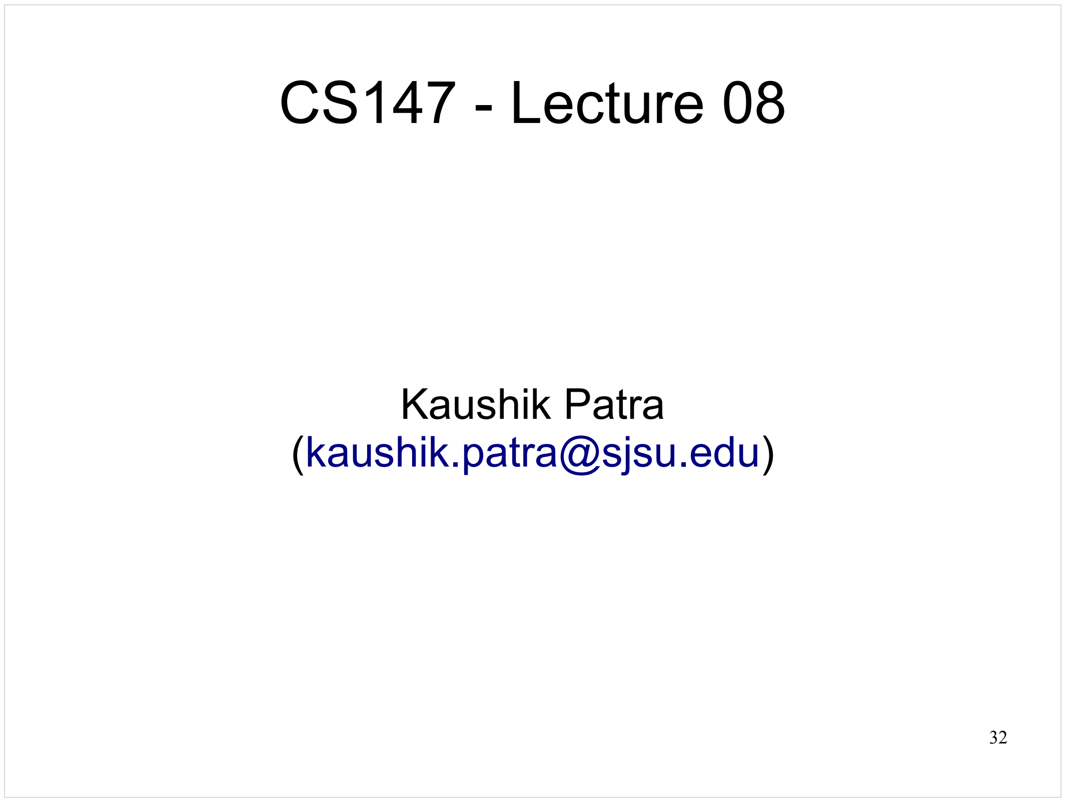
Component 4: Barrel Shifter

- Right and left barrel shifter can be combined into single shifter with both side shifting capability



31

- To keep circuit simple we can also keep distinct type of shifter (left and right). Once we have both right and left barrel shifter, we can use another mux to select between right shift and left shift result depending on a control signal LnR. This means LnR=1 will give the left shift result and LnR=0 will give the right shift result.



CS147 - Lecture 08

Kaushik Patra
(kaushik.patra@sjsu.edu)

32

- Sequential Circuit Design
- Digital Circuit Components

[Chapter 5 of Logic & Computer Design Fundamentals, 4th Edition, M. Morris Mano, Charles R. Kime]