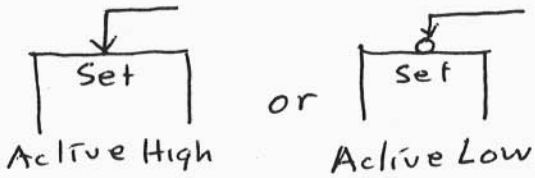
Other types of F-F's

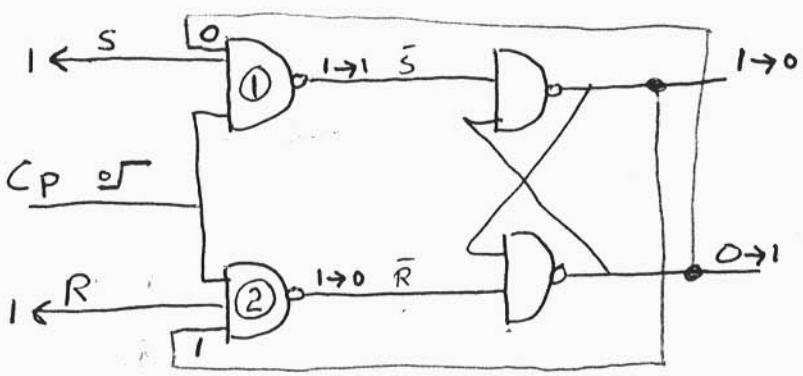
The basic SR-FF considered sofar has some features, either missing or not quite what we want, that have led to the development of other types.

1. Need (want) ability to initialize the FF to a given state. All FF's now have "Preset" and "Preclear" inputs which override all other controls. May be active high or active low indicated on symbol as



2. A desired operating mode is missing, namely the "toggle" mode which is a basic feature of counters. Toggle mode is mode where output changes state on each succesive input clock pulse.
3. later

If thought about it for awhile, might propose following scheme to implement toggling: Postulate:



At certain time ($t=0$) have conditions as shown. Note that gate ② is ennabled while ① is disabled (ie ① will not respond to Cp). When Cp goes high, output of gate ② goes 1→0 while output of ① stays 1. Hence latch outputs change state. Now have ① ennabled and ② disabled so on next clock latch outputs will again change. Key point is that the cross-coupling ennables only the input

gate which can cause an output state change — voila — toggle mode.

Wrong

If C_p stays high for a time longer than the propagation delay thru the gates then each input gate will be successively enabled - result is a "burst" oscillator output (circuit will in general give peculiar results).

Need to effectively introduce delay in the cross-coupling paths. Done by using a "dual-rank" approach in which two latches are used. First one, called the "master" is controlled by "clock", second one, called the "slave" is " " $\overline{\text{clock}}$.

When clock goes high, master changes (or not) as dictated by S-R inputs. As clock goes low master retains values. Slave is ennabled by $\overline{\text{clock}}$ going high and acquires values from master and holds on until next cycle & slave acts as a D type data latch.) Arrangement is then called a "master-slave S-R Flip-Flop" or a "dual-rank F-F".

Now can incorporate the cross-coupling feature and get proper toggling mode.

However, if put in cross-coupling we change the names S and R to J and K. Hence with cross-coupling built in we have a J-K type F-F which is very common and very versatile.

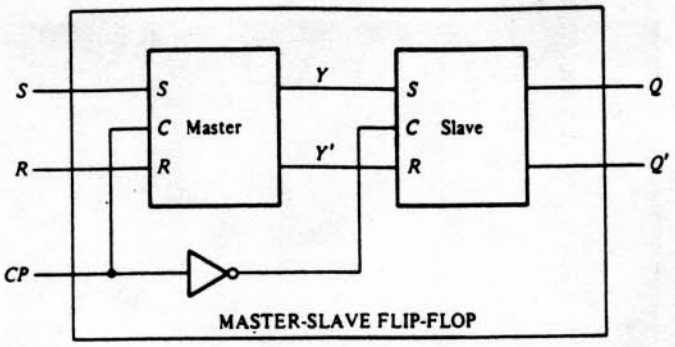


FIGURE 6-9
Logic diagram of a master–slave flip-flop

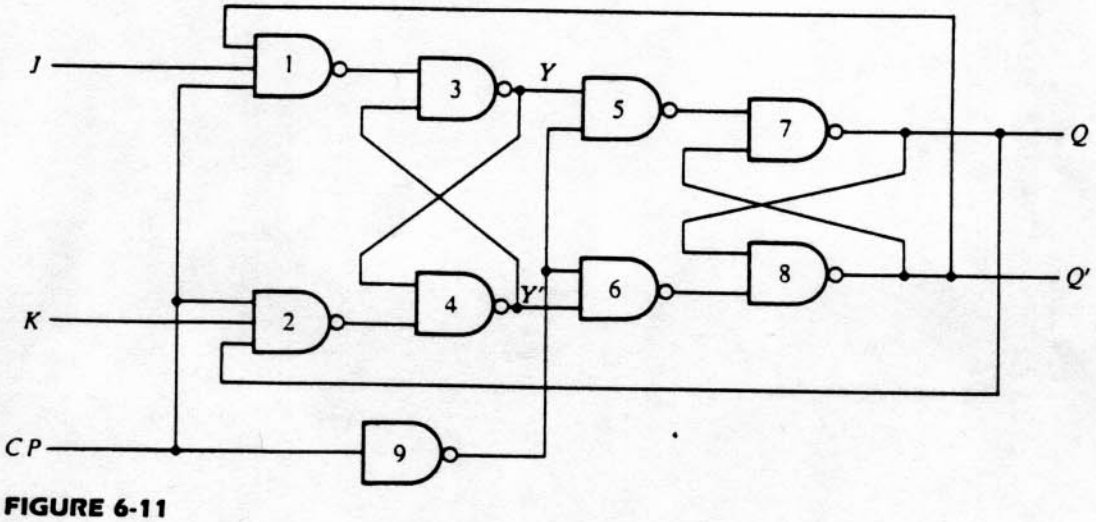
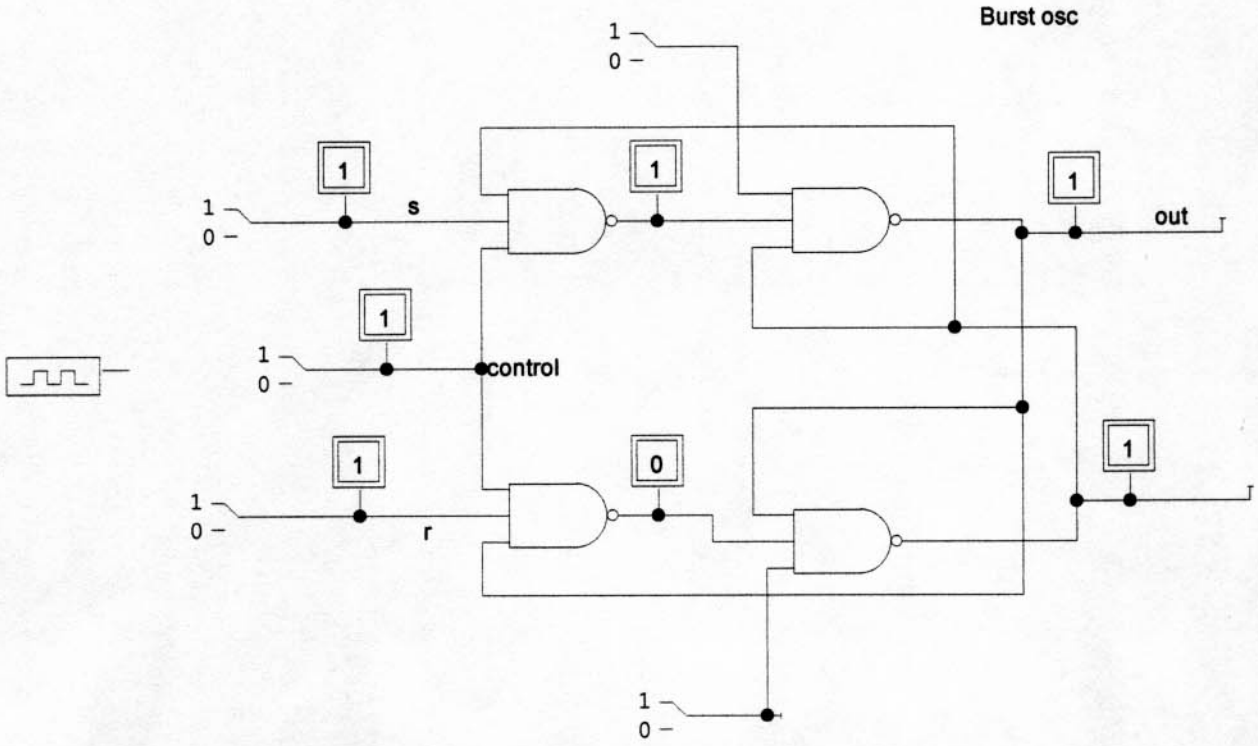
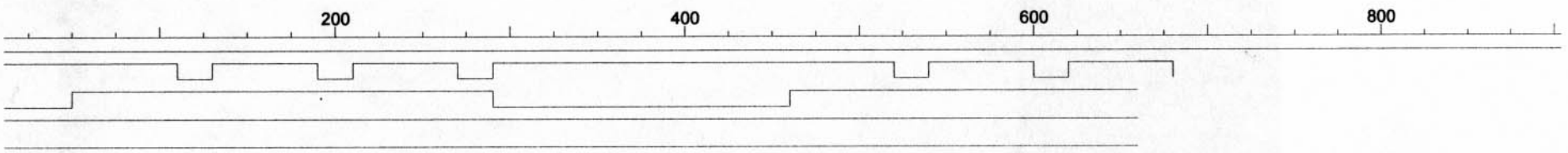


FIGURE 6-11
Clocked master–slave *JK* flip-flop


Burst osc
1
0
1
1
s
1
out
1
control
1
0
1
1
r
0
1
1
0

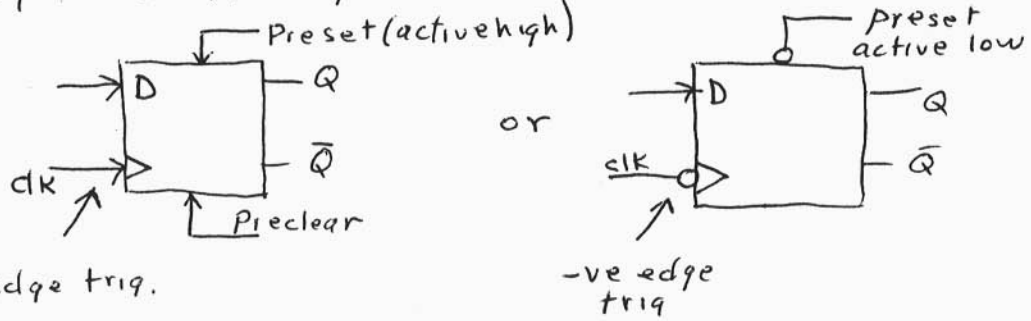

200
400
600
800
ut
ontrol

3. Plain latches, ordinary SR-FF's and MS-FF's are all "transparent" i.e. respond to inputs during all the time clock is high - can lead to catching and holding on to glitchs.

Prompted development of another type called "D type edge triggered F-F". This type has one input only, called D (plus of course clock, preset, and preclear. Its characteristic is that it captures the state of the D input which exists just prior to the rising edge of the clock and holds it until the next rising edge where it may take on new value. Hence it is not transparent. Internally it uses three latchs.

As you would expect most modern F-F's are either D type edge-triggered or J-K's built to incorporate edge-triggering.

Edge-triggering is indicated on the symbol by a ▷ (triangle)



Characteristic and excitation tables:

Convenient to summarize behaviour:

Char. table ⇒ given these inputs what is result

excitation table = if you want this to happen what must be inputs.

Flip-Flops Summary

Characteristic Table

S-R

S	R	Q_{n+1}
0	0	Q_n (no change)
0	1	0 (reset)
1	0	1 (set)
1	1	dis. {0,0 NOR; 1,1 NAND}

J-K

J	K	Q_{n+1}
0	0	Q_n (no change)
0	1	0 (reset)
1	0	1 (set)
1	1	$\bar{Q}_n$ (toggle)

D

D	Q_{n+1}
0	0 (reset)
1	1 (set)

T

T	Q_{n+1}
0	Q_n (no change)
1	$\bar{Q}_n$ (toggle)

Excitation Table

S-R

Q_n	Q_{n+1}	S	R	
0	0	0	ϕ	(no change or reset)
0	1	1	0	(set)
1	0	0	1	(reset)
1	1	ϕ	0	(no change or set)

J-K

Q_n	Q_{n+1}	J	K	
0	0	0	ϕ	(no change or rest)
0	1	1	ϕ	(set or toggle)
1	0	ϕ	1	(reset or toggle)
1	1	ϕ	0	(no change or set)

D

Q_n	Q_{n+1}	D_n
0	0	0
0	1	1
1	0	0
1	1	1

T

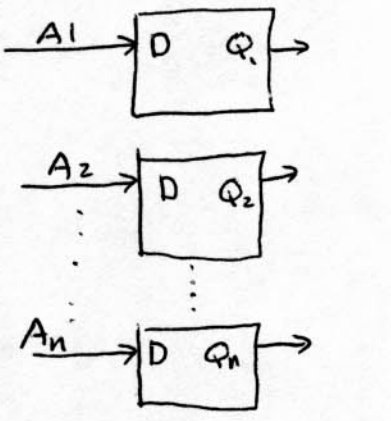
Q_n	Q_{n+1}	T	
0	0	0	(no change)
0	1	1	(toggle)
1	0	1	(toggle)
1	1	0	(no change)

Registers

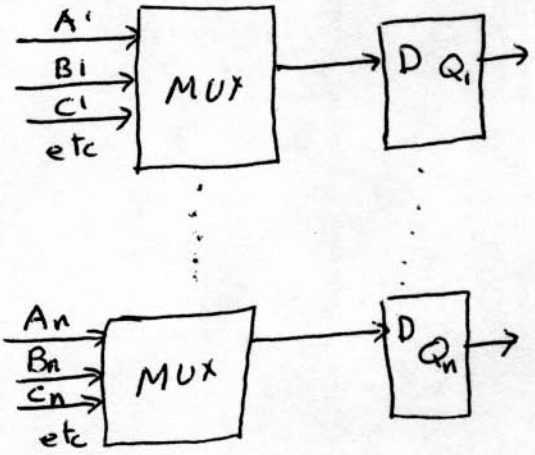
Register = group of F-F's acting as one entity.

Common application = <u>storage registers</u>

Ex:

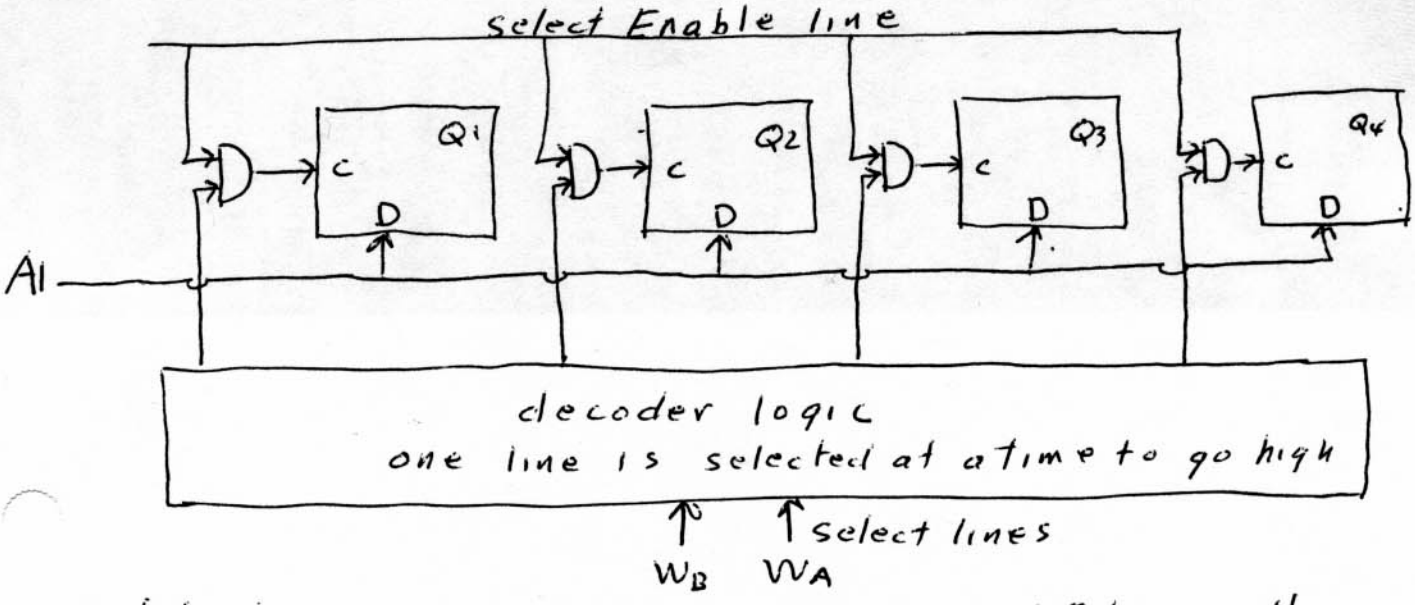


acquire and store bits of word A



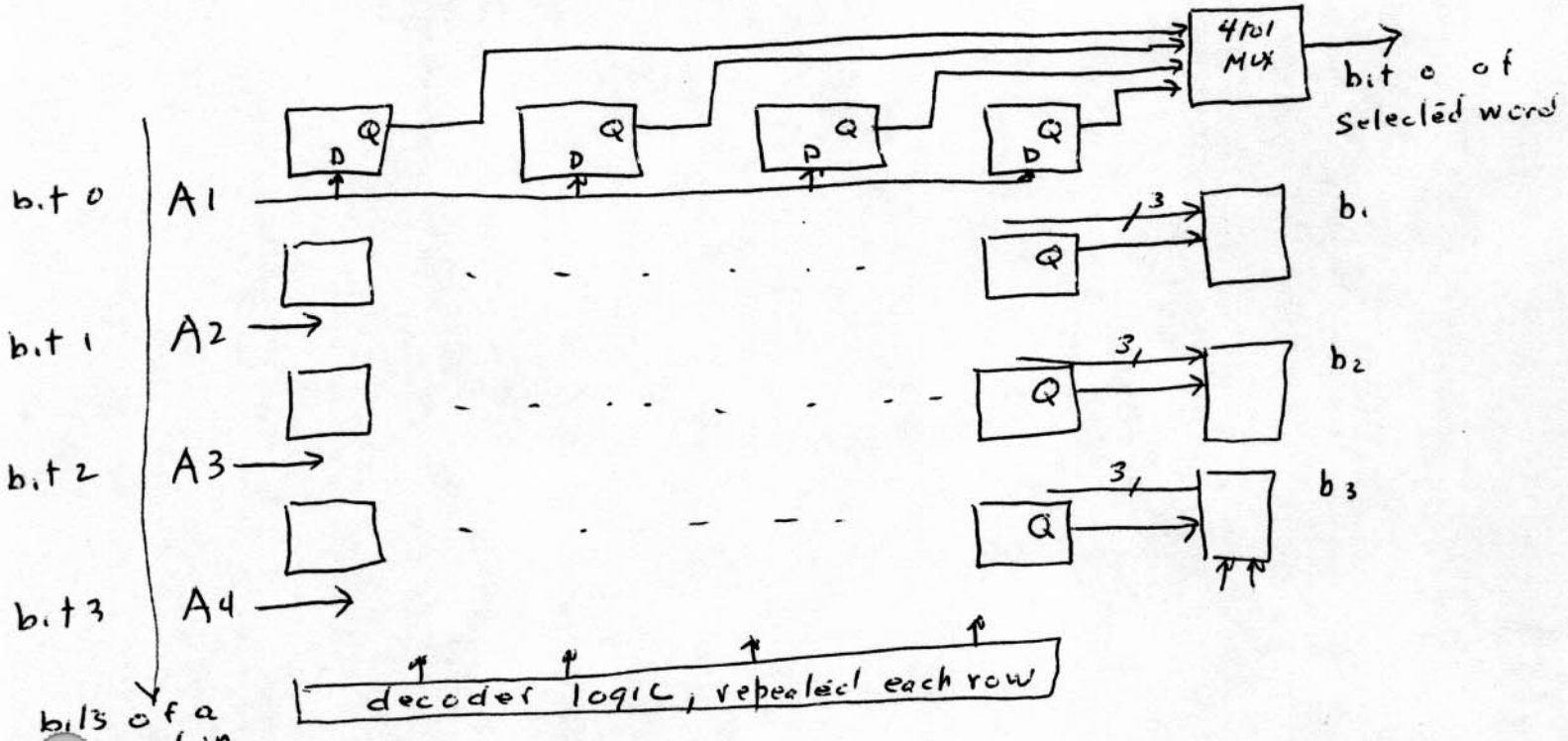
Select and store bits of word A or word B etc

Can rearrange topology so that can select which of a number of latches A1 etc is loaded into as in



depending on $W_B W_A$ one of the four latches will acquire A1 when register is ennabled.

Add 3 more identical rows:



WRITE control lines

EN (Gw) — W_B W_A determine in which column the bits of the input word will be stored

READ control lines

EN (GR) — RB RA determine which column, i.e. which word will be read out

Above is called a 4x4 register file (74170) or RAM

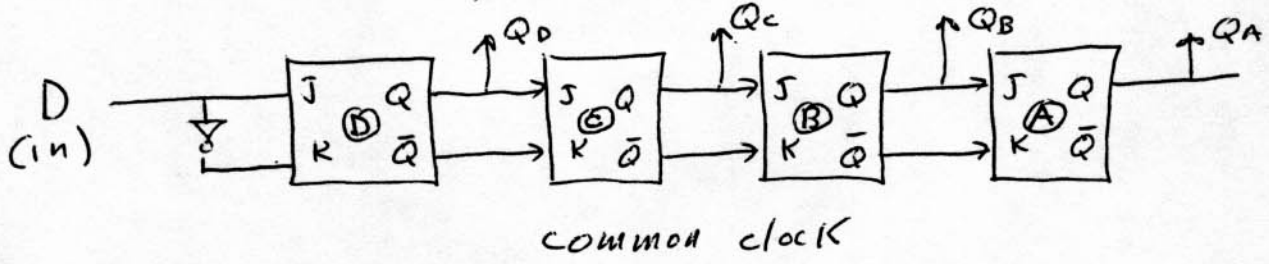
7489 is a 16 × 4 RAM.

Much (very much) larger sizes are common.

Shift registers

Basic function is to move the bit pattern contained over one position on each clock. Several types exist:

SIPO (serial in parallel out)

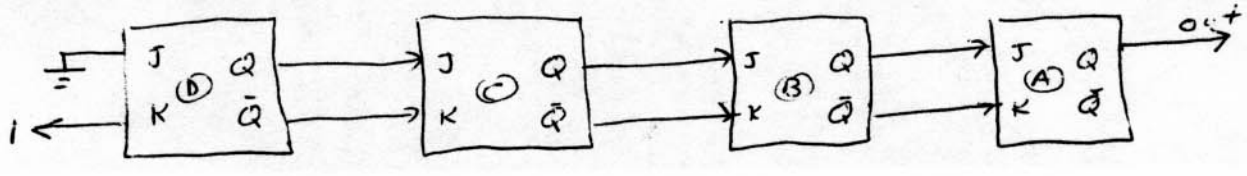


here have J-K's acting as D types. Preset, clear not shown

	D	C_P	Q_D	Q_C	Q_B	Q_A	
reset		0	0	0	0	0	
	1	1	1	0	0	0	
	0	2	0	1	0	0	
	1	3	1	0	1	0	
	0	4	0	1	0	1	← parallel output

serial input

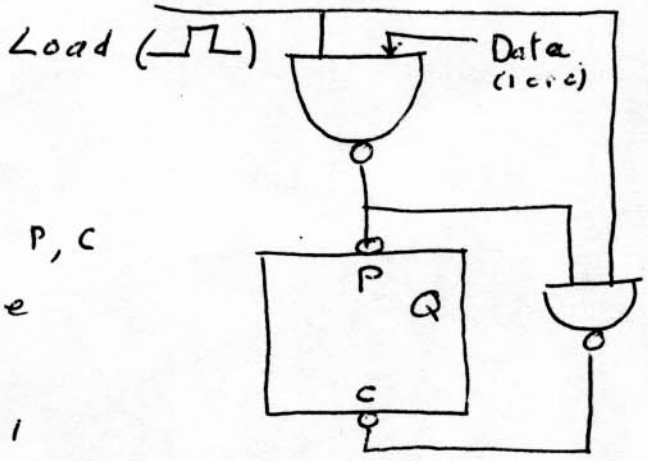
PISO (parallel in serial out)



	C_P	Q_D	Q_C	Q_B	Q_A	
preset	0	1	0	1	0	→ Serial out
	1	0	1	0	1	→
	2	0	0	1	0	→
	3	0	0	0	1	→
	4	0	0	0	0	

Presetting

One way is:



(jam transfer)

Load low: both P, C are 1 = inactive

Load high, D = 1

P = 0, C = 1

sets Q to 1

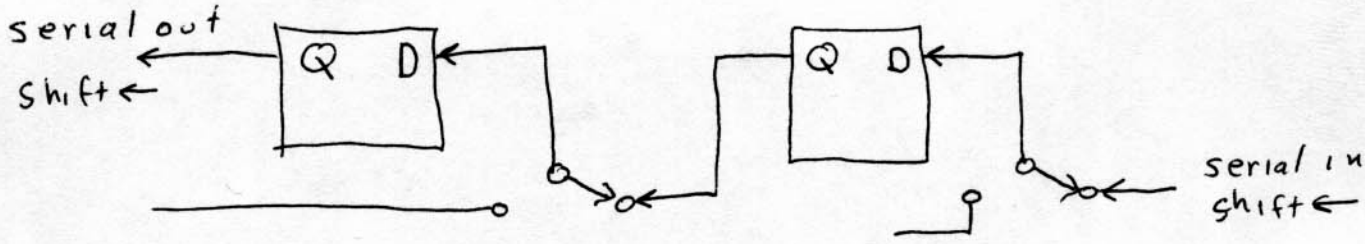
Load high, D = 0

P = 1, C = 0

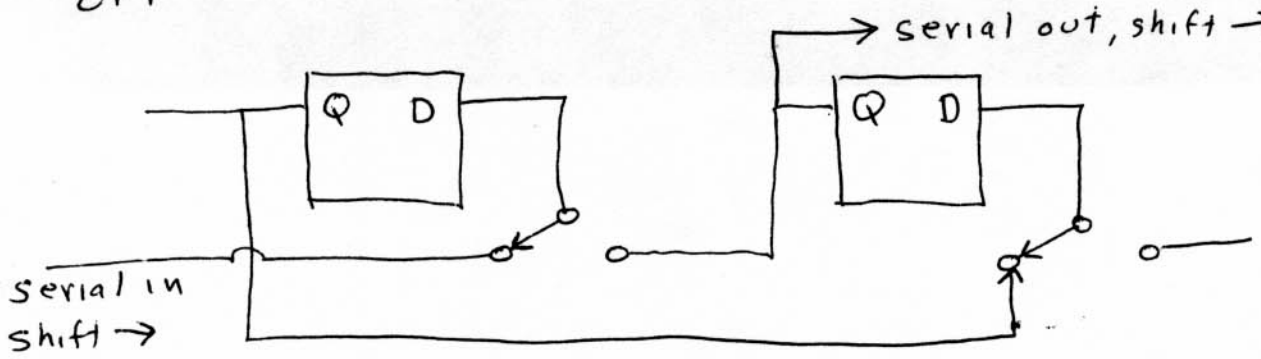
resets Q to 0

Shift left, shift right

Can arrange, using switchs, to cause pattern shifting to go either left or right:



or:



Bidirectional Shift Register with Parallel Load

Shift registers can be used for converting serial data to parallel data, and vice versa. If we have access to all the flip-flop outputs of a shift register, then information entered serially by shifting can be taken out in parallel from the outputs of the flip-flops. If a parallel-load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stored in the register.

Some shift registers provide the necessary input and output terminals for parallel transfer. They may also have both shift-right and shift-left capabilities. The most general shift register has all the capabilities listed below. Others may have only some of these functions, with at least one shift operation.

1. A *clear* control to clear the register to 0.
2. A *CP* input for clock pulses to synchronize all operations.
3. A *shift-right* control to enable the shift-right operation and the *serial input* and *output* lines associated with the shift right.
4. A *shift-left* control to enable the shift-left operation and the *serial input* and *output* lines associated with the shift left.
5. A *parallel-load* control to enable a parallel transfer and the n input lines associated with the parallel transfer.
6. n parallel output lines.
7. A control state that leaves the information in the register unchanged even though clock pulses are continuously applied.

A register capable of shifting both right and left is called a *bidirectional shift register*. One that can shift in only one direction is called a *unidirectional shift register*. If the register has both shift and parallel-load capabilities, it is called a *shift register with parallel load*.

The diagram of a shift register that has all the capabilities listed above is shown in Fig. 7-9. It consists of four D flip-flops, although RS flip-flops could be used provided an inverter is inserted between the S and R terminals. The four multiplexers (MUX) are part of the register and are drawn here in block diagram form. (See Fig. 5-16 for the logic diagram of the multiplexer.) The four multiplexers have two common selection variables, s_1 and s_0. Input 0 in each MUX is selected when $s_1s_0 = 00$, input 1 is selected when $s_1s_0 = 01$, and similarly for the other two inputs to the multiplexers.

The s_1 and s_0 inputs control the mode of operation of the register as specified in the function entries of Table 7-2. When $s_1s_0 = 00$, the present value of the register is applied to the D inputs of the flip-flops. This condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock pulse transfers into each flip-flop the binary value it held previously, and no change of state occurs. When $s_1s_0 = 01$, terminals 1 of the multiplexer inputs have a path to the D inputs of the flip-flops. This causes a shift-right operation, with the serial input transferred into flip-flop A_4. When $s_1s_0 = 10$, a shift-left operation results, with the other serial input going into

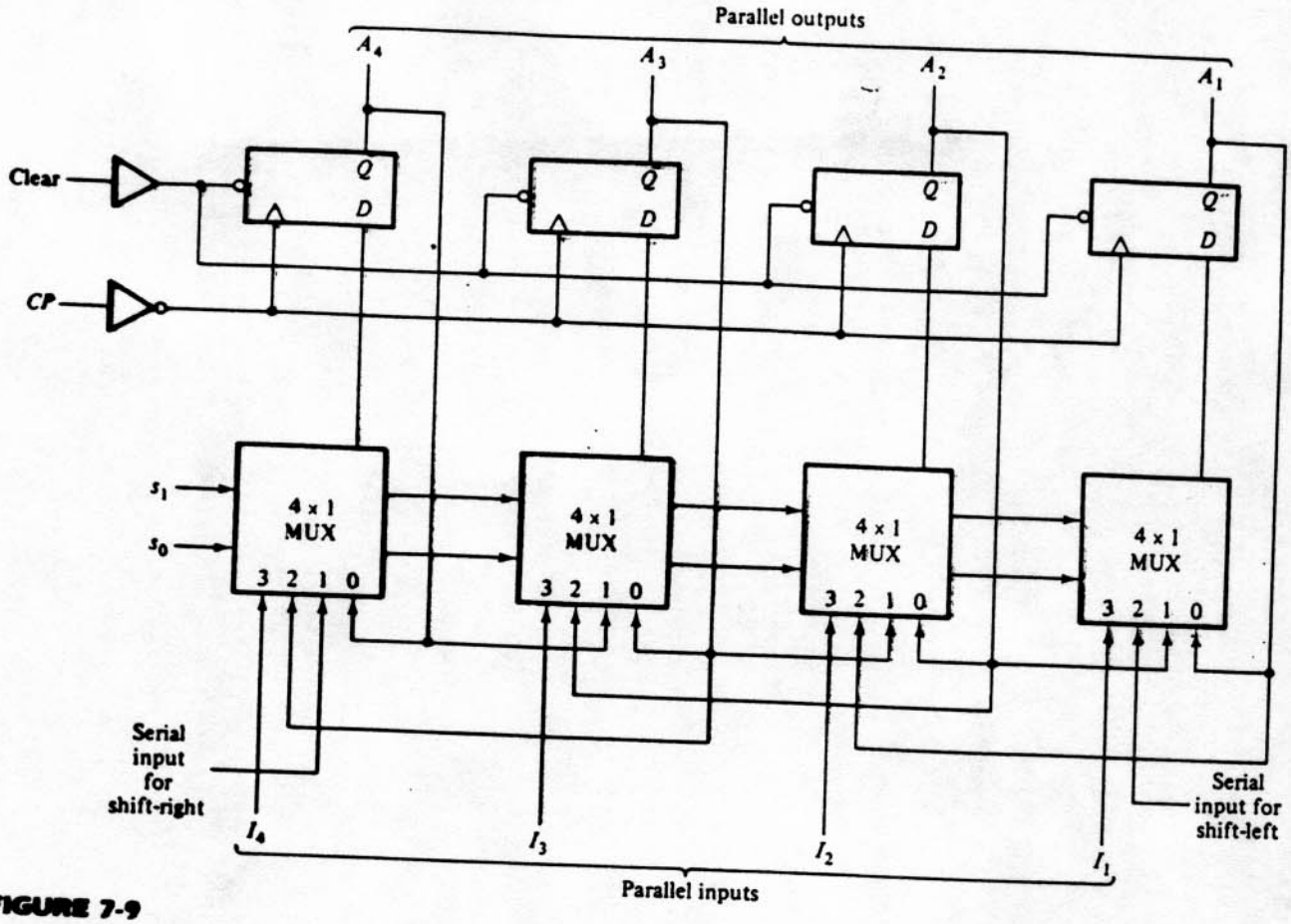


FIGURE 7-9
4-bit bidirectional shift register with parallel load

flip-flop A_1. Finally, when $s_1s_0 = 11$, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock pulse.

A bidirectional shift register with parallel load is a general-purpose register capable of performing three operations: shift left, shift right, and parallel load. Not all shift registers available in MSI circuits have all these capabilities. The particular application dictates the choice of one MSI shift register over another.

TABLE 7-2
Function Table for the Register of Fig. 7-9

Mode Control		
s_1	s_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

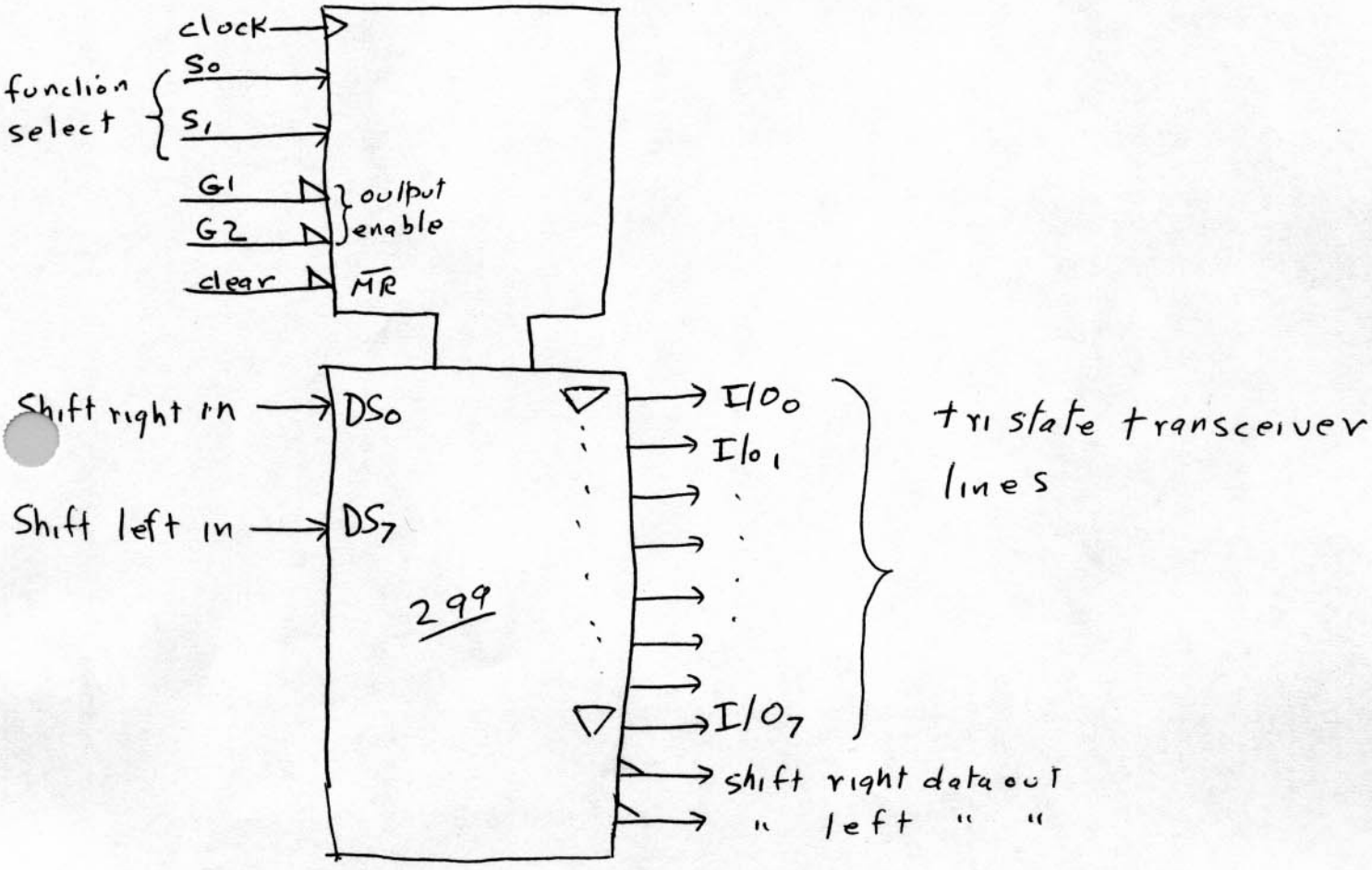
Typical shift registers

74LS 164 = SIPO 8 bits

74LS 165 = PISO "

Universal type

74LS 299 (very versatile)



Modes

$\overline{MR}$	S_1	S_0	
0	x	x	Reset $Q_0 \rightarrow Q_7$ to 0
1	0	0	Hold
1	0	1	Shift right $DS_0 \rightarrow Q_0 \cdots Q_7$
1	1	0	Shift left $DS_7 \rightarrow Q_7 \cdots Q_0$
1	1	1	Parallel load