CALIFORNIA STATE UNIVERSITY
LOS ANGELES

Department of Electrical and Computer Engineering

EE-2449 Digital Logic Lab

EXPERIMENT 12

Shift Registers

Text: Mano and Ciletti, *Digital Design*, *5th Edition*, Chapter 6
Required chips: one **7474**, with 2 D-Flip-flops; One **7486** XOR; One **74179** Shift Register.

12.1 Introduction.

A shift register is an n-bit register with provision for shifting its stored data by one position at each clock pulse. The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops have a common clock. In Fig. 1, a positive pulse at CLK ↑ will cause the contents of each flipflop to be shifted one position to the right, bringing the serial input bit (SW1) in at the left and pushing the bit stored in L4 out at the right. That bit is lost unless it is saved externally. Although Fig. 1 shows a shift-right register, the same register can obviously be used for left shifts simply by reversing the sense of the bits. Most shift registers have provision for shifting only in one direction, but some have a control input that allows either left or right shifting to be specified at each clock.

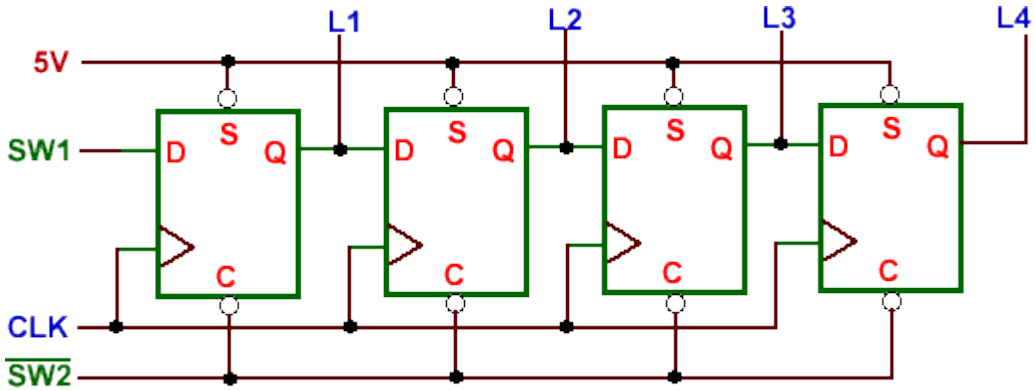


Figure 1. 4-bit shift register.

One way to load *n* bits of data into the flip-flop chain is to load the data one bit each clock cycle using the serial input. Some shift registers also have parallel inputs that can be used to load all *n* bits with one clock pulse. The output of a shift register can be observed one bit at a time at the serial output, but some shift registers also have parallel outputs for observing all *n* bits at once.

Shift registers are classified according to three basic considerations: their method of data handling (serial-in serial-out, serial-in parallel-out, and parallel-in serial-out), their direction of data movement (shift right, shift left, and bidirectional), and their bit length. One of the important applications of shift register circuits is in serial computation. Compared to parallel computation, where all bits in a word are processed at the same cycle, serial computations process words in one bit per cycle. Therefore, serial computation is slower, but it has the advantage of requiring less hardware and wiring. A serial adder will be studied later in this experiment as an example.

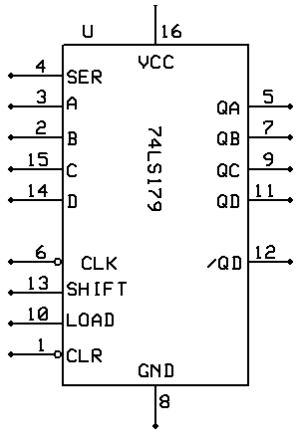
12.2* Shift Register using 7474 D Flip-Flops

Use two 7474 dual flip-flops to connect a serial-in, parallel-out shift register as shown in Fig. 1. Draw the wiring diagram in your lab notebook (refer to the IC specifications document for pin numbers). Connect L1–L4 to four LEDs (from the LED bar graph), SW1 and SW2 to switches, and CLK to a normally low pulser. Implement your design on your breadboard. Test the behavior of the circuit as follows. (1) Initially, set SW2 to logic 0 to clear all flip-flops (diagram shows SW2 as active-low). (2) Now set both SW2 and SW1 to logic 1; i.e. flipflops are no longer being cleared and serial input is high. (3) Push the pulser button several times to allow a series of logic 1's to be shifted into the shift register. (4) Change SW1 to logic 0 and repeat the experiment.
Record your observations in your laboratory notebook. You will use this circuit in part 12.4 so do not disassemble it.

12.3* Shift Register using 74179

	/CLR	LOAD	SHIFT	CLK	FUNCTION
1	L	X	X	X	CLEAR: (0000) → (QA...QD).
2	H	L	L	X	HOLD: (no change).
3	H	X	H	↓	SHIFT: (SER,QA,QB,QC) → (QA,QB,QC,QD).
4	H	H	L	↓	LOAD: (A...D) → (QA...QD).

Study the behavior of the 74179 shift register in this function table. It shows that when CLR is active (L), the other control inputs have no effect. And when CLR is not active (H) but SHIFT is active (H), a negative clock pulse ↓ shifts the serial input SER into QA, pushing QD out at the right. This experiment uses only these two conditions; i.e. lines 1 and 3 of the function table (so LOAD and parallel inputs ABCD are ignored.). Here you will implement the circuit in Figure 1 using the 74179. Note that outputs L1…L4 appear here as QA…QD. Connect SER and CLR to switches, SHIFT to Vcc (it will not change), and CLK to a pulser. Also, connect outputs QA…QD to LEDs.



Draw the wiring diagram in your lab notebook. Then build the circuit and test it following the steps described in part 12.2 (clock in some 1's at SER, then some 0's.). Discuss results in your lab notebook.

12.4* Pseudo-random Sequence Generator

Now use the circuit of 12.1 to build a pseudo-random binary sequence generator as shown in Fig. 2. This binary sequence generator will display a pseudo-random output (not truly random because the bit patterns repeat every 2^n–1 bits, where n is the number of flip-flops used in the shift register). The IC 7486 provides the exclusive-OR needed in the circuit. Again, draw the wiring diagram in your lab notebook and then build the circuit on your bread board.

4-BIT PSEUDO RANDOM PATTERN

	L1	L2	L3	L4
1	0	0	0	1
8	1	0	0	0
4	0	1	0	0
2	0	0	1	0
9	1	0	0	1
12	1	1	0	0
6	0	1	1	0
11	1	0	1	1
5	0	1	0	1
10	1	0	1	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
7	0	1	1	1
3	0	0	1	1

Go through the following steps to test your circuit.

(1) To initialize the sequence generator, set SW1 to 0 (active) which will clear L1, L2, L3 to 0's and set L4 to 1. Resulting pattern is 0001, the top row of the table.

(2) Change SW1 to logic 1 in order to inactivate the C (clear) and S (set) controls so circuit can now shift.

(3) Now pulse the clock repeatedly and record the resulting sequence in a table in your lab notebook.

(4) Continue pulsing a bit more to see if the sequence begins to repeat.

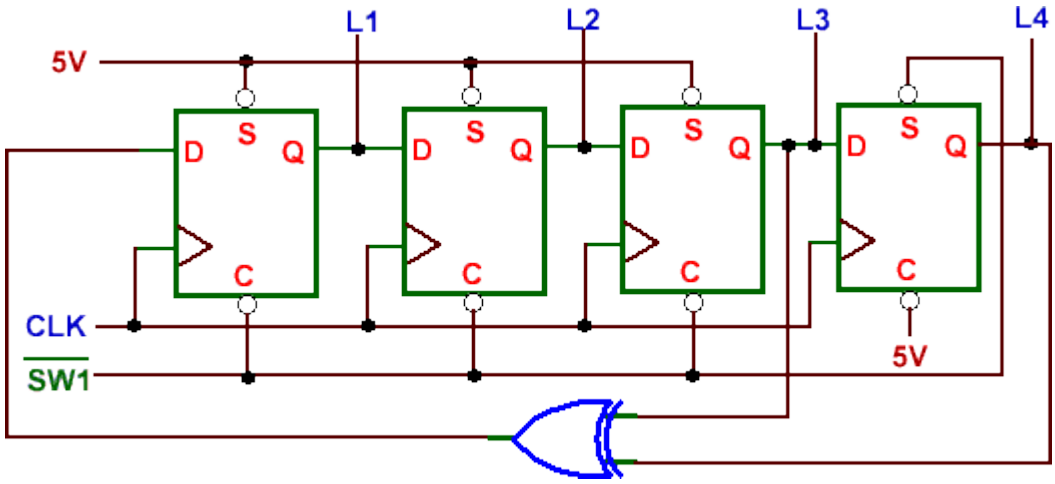


Figure 2. Pseudo-random binary sequence generator.

Questions:

a) Does your sequence match that in the table? Looking at the decimal values, does it appear random?

b) Why would you not want to start the sequence with 0000?

c) Does the sequence repeat after 15 clock pulses? If so, this justifies the term "pseudo (false)-random.

12.5 Serial Adder

A serial adder adds a pair of binary numbers serially with a simple full adder. The carry out of the full adder is transferred into a D flip-flop and the output of this carry flip-flop is then used as the input carry for the next pair of significant bits. Fig. 3 illustrates an example of serial addition.

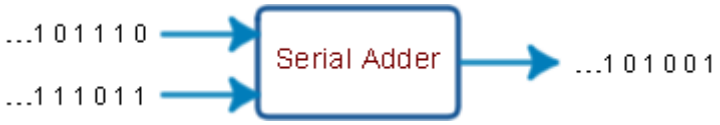


Figure 3. Serial addition.

Note: when we do addition by hand, we also use the serial method. We have a number on the top, the "augend", to which we add the number on the bottom, the "addend". The augend is so named because it is being added to (i.e. *aug*mented) by the addend. Starting at the right-hand column, we add the two digits, place the sum below the column and bring any carry (0 or 1) into the next column at the left and repeat. So we don't parallel add all columns at once. Nor does a serial adder; it also adds one column at a time, except that instead of moving left through the columns, it shifts them to the right, one at a time, into a single adder.

Fig. 4. shows the block diagram design of a serial adder. Registers A and B are shift registers (maybe 74179's). Shifting goes from QA toward QD.

- Both QD outputs will be fed into the full adder along with the output of the D flipflop.
- A pulser is connected to the clock inputs of both registers and the flip-flop.
- The adder output sum, S, is connected to register A's serial input and is shifted into QA when the circuit is clocked
- The output of the D flip-flop, which is the stored carry *from the previous addition*, is also input to the adder. The adder output carry (C) is stored into D, also when the circuit is clocked.
- The serial input to register B comes from an external source.

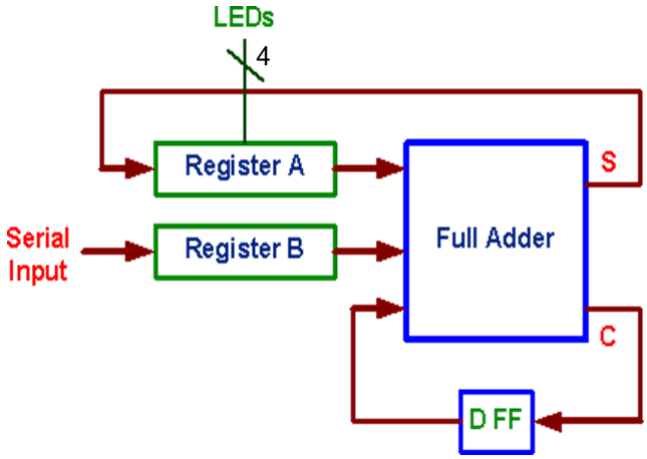


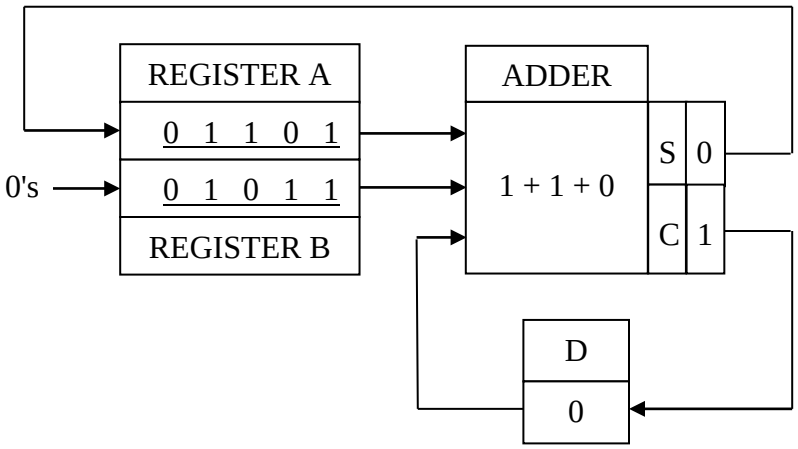
Figure 4. Block diagram of a serial adder.

Here's how the circuit works. In the following example, we assume that register A is already loaded with the augend and register B with the addend. The augend could have been sent serially into register B, through the adder and then into register A. The addend would have immediately followed the augend bits and filled register B.

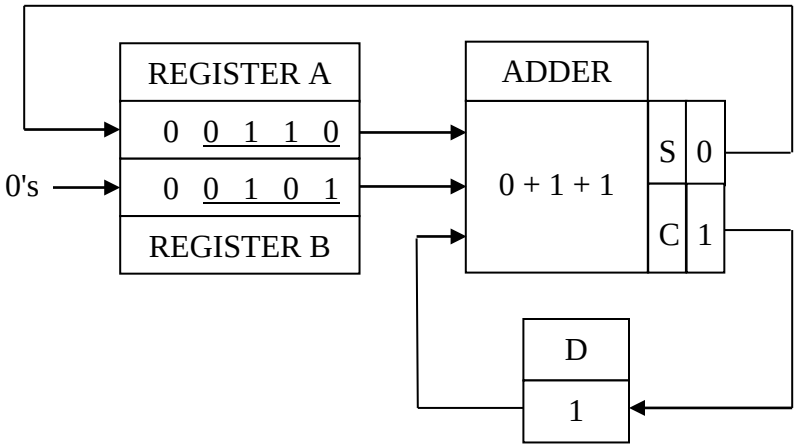
Alternatively, both registers could have simultaneously parallel-loaded their contents (as with the 74179's A,B,C,D inputs).

The first diagram shows the situation after initializing the contents of the registers and clearing the carry flip-flop D. The right-hand bits of the registers have entered the adder along with the stored carry in D. Initially, of course, D = 0 since there was no previous addition to generate a carry. Addition then proceeds with one shift per clock pulse.

Pre-clocking) Register A contains the augend, B contains the addend, and the carry flip-flop D is cleared.



1st clock) When the clock is triggered, the S = 0 from the previous state is brought around and shifted into the left-hand bit of register A, and the C = 1 from the previous diagram is stored into D, replacing the 0 that was there.



(The new adder output C = 1 waits for the next clock to be stored into D.)

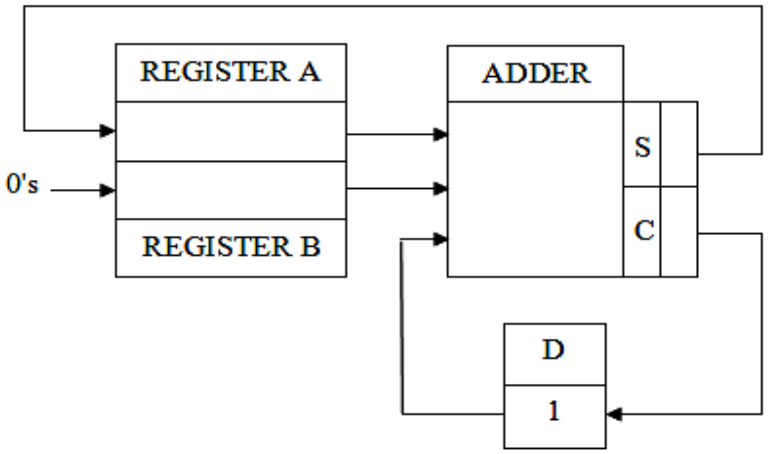
Also at this time, a 0 is shifted into register B. The bits shifted into reg. B could be 0's or 1's--they will not affect the final sum stored in reg. A since by the time they reach the adder, the sum (augend+addend) is already in register A and the addition is finished--no more clocks.

Fill in the diagrams for the next 4 shifts on the next page. Cut and paste into your lab notebook. (You may want to copy the page, shrink it to the size of your notebook, and print it out.)

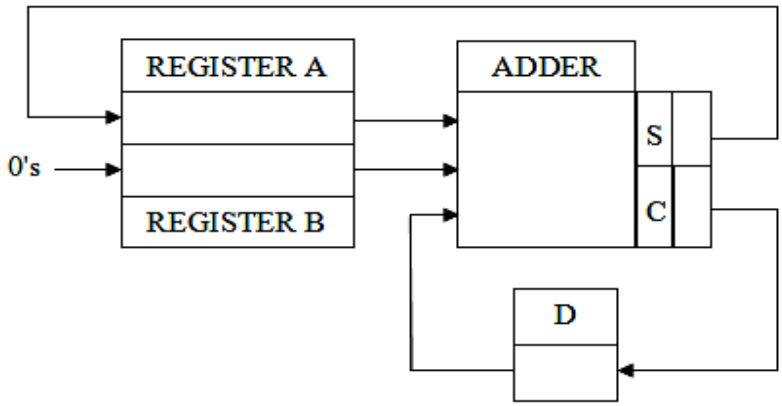
After the 5th shift, the sum of the augend and addend should be in register A.

Do a hand calculation of the sum and compare. Obviously, they should be the same.

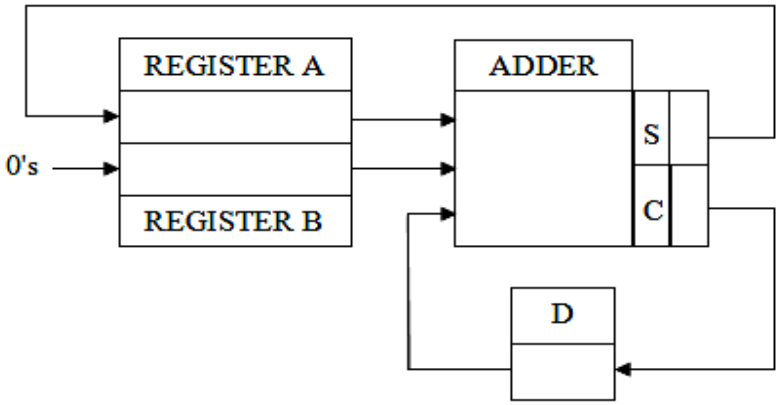
2nd clock)



3rd clock)



4th clock)



5th clock)