11 Sequential logic circuits – registers

11.1 Objectives

The registers are defined and classified based on their functionality. The implementation of the universal shift register, based on flip-flops and multiplexers, is studied. Two representative integrated circuits having register functionality are presented. Several numerical sequence generators are implemented using the analyzed registers.

11.2 Theoretical considerations

The registers are sequential logic circuits capable of storing or shifting binary data. They are implemented using as many bistables as the number of bits. For simplicity, the registers are often implemented with D flip-flops. The number of stored bits represent the *register capacity*. According to their functionality, the registers can be classified as:

- storage registers – can load and store data every clock cycle;
- shift registers – can shift data one position every clock cycle, while serially loading one bit of data: SerialBit→Q_A→Q_B→Q_C→ …;
- combined storage and shift registers – combines the functionality of the previous categories;
- universal shift registers – adds bidirectional shift to a combined storage and shift register.

11.2.1 The universal shift register

Figure 11. 1 highlights the implementation of a 4-bit universal shift register using D flip-flops and multiplexers. Each output bit is associated with a flip-flop and a multiplexer defining the various functionalities. The flip-flops can be reset asynchronously.

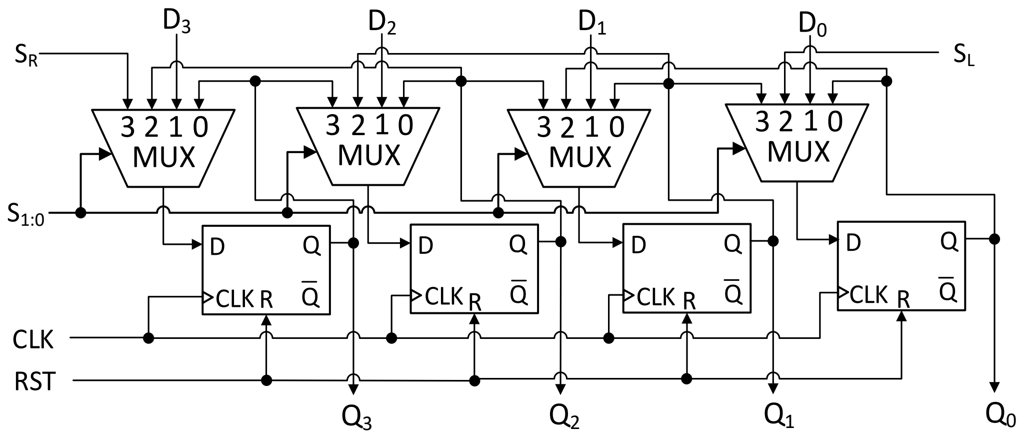


Figure 11. 1 Implementation of a 4-bit universal shift register using D flip-flops

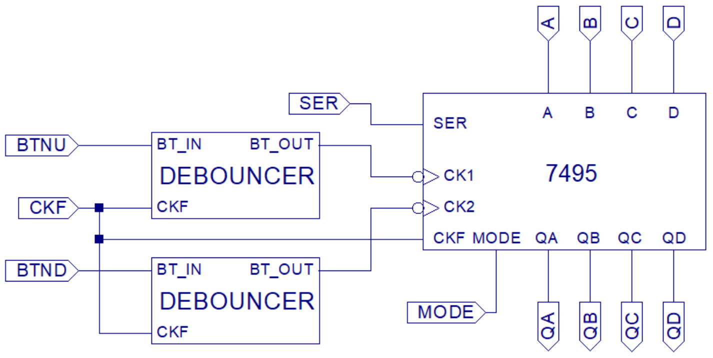
The selections S_1, S_0 define the functionality, as follows:

- $S_{1:0}$=00 – storage: $Q_3Q_2Q_1Q_0{}^{t+1} = Q_3Q_2Q_1Q_0{}^{t}$;
- $S_{1:0}$=01 – parallel load: $Q_3Q_2Q_1Q_0{}^{t+1} = D_3D_2D_1D_0$;
- $S_{1:0}$=10 – serial load on S_L (Serial Left) with left shift: $Q_3Q_2Q_1Q_0{}^{t+1} = Q_2Q_1Q_0S_L{}^{t}$;
- $S_{1:0}$=11 – serial load on S_R (Serial Right) with right shift: $Q_3Q_2Q_1Q_0{}^{t+1} = S_RQ_3Q_2Q_1{}^{t}$.

Note: Serial load refers to filling one bit of data each clock cycle. Filling a 4-bit register entirely requires 4 consecutive clock cycles. Alternatively, a parallel load fills the register in 1 clock cycle.

11.2.2 The shift register 7495

The 4-bit shift register 7495 has the following functions: load, storage and data shift. The testing circuit and the function table are highlighted in Figure 11. 2. The register has two functions: if MODE=0, every falling edge of the clock signal CK_1, the data is shifted with serial load on input SER (shift direction: SER→Q_A→Q_B→Q_C→Q_D); if MODE=1, the binary value $(ABCD)_2$ is loaded synchronously on the CK_2 falling edge.



MODE	CK_1	CK_2	SER	$Q_{A:D}^{t+1}$
0	$\downarrow$	X	0	0 Q_A^t Q_B^t Q_C^t
0	$\downarrow$	X	1	1 Q_A^t Q_B^t Q_C^t
1	X	$\downarrow$	X	A B C D

Figure 11. 2 Testing circuit for register 7495 (left) and the function table (right)

When cascading several 7495 registers, the same functionality is extended to a multiple of 4 bits. An 8-bit register is implemented from two 4-bit registers, as shown in Figure 11. 3.

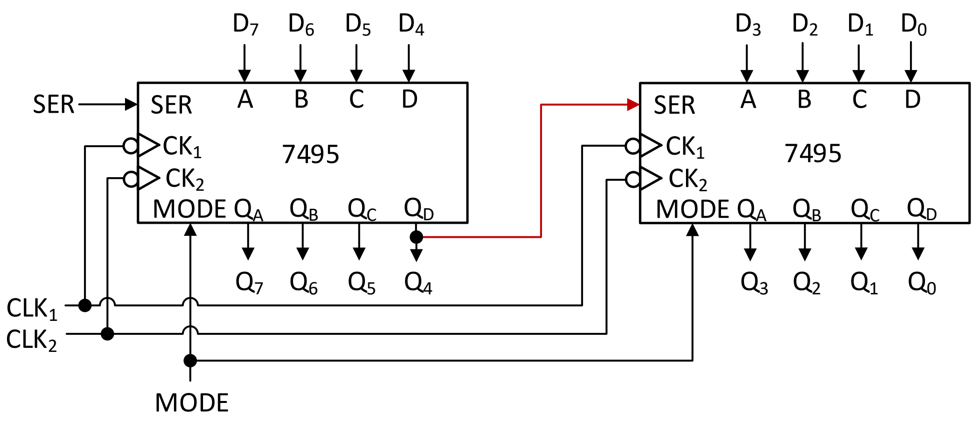


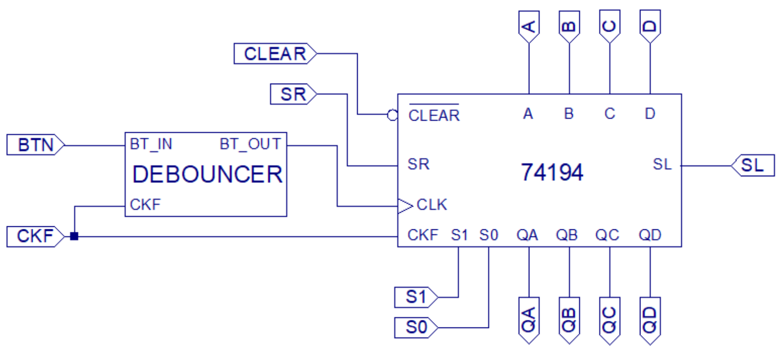
Figure 11. 3 Cascading two 7495 registers

The clock and MODE signals are shared by all registers. To enable the shift of data bits from one register to its right neighbor, the output Q_D is connected to the serial input SER of the neighbor, thus implementing the shift from Q_4 to Q_3.

11.2.3 The universal shift register 74194

Circuit 74194 is a 4-bit universal shift register with synchronous operations on the clock rising edge. The active low input $\overline{\text{CLEAR}}$ performs asynchronous reset and has priority over other operations. Selection signals S_1 and S_0, define 4 synchronous operations, highlighted in the function table from Figure 11. 4:

- $S_{1:0}$=00 – for storage;
- $S_{1:0}$=01 – for right shift with serial load on S_R;
- $S_{1:0}$=10 – for left shift with serial load on S_L;
- $S_{1:0}$=11 – for load with binary value $(ABCD)_2$.



$\overline{CLR}$	S_1	S_0	CLK	$Q_{A:D}^{t+1}$			
0	X	X	X	0	0	0	0
1	0	0	↑	Q_A^t	Q_B^t	Q_C^t	Q_D^t
1	0	1	↑	S_R	Q_A^t	Q_B^t	Q_C^t
1	1	0	↑	Q_B^t	Q_C^t	Q_D^t	S_L
1	1	1	↑	A	B	C	D

Figure 11. 4 Testing circuit for register 74194 (left) and the function table (right)

To extend the capacity, several 74194 registers can be cascaded. Figure 11. 5 highlights two cascaded registers implementing an 8-bit universal shift register. Both 74914 units share the same clock signal, the asynchronous command $\overline{\text{CLEAR}}$, and the selections S_1, S_0. Consequently, the registers have the same function regime. To shift the data bits between registers, the output Q_D is connected to the serial input S_R ($Q_4 \rightarrow Q_3$), and the output Q_A is connected to the serial input S_L ($Q_3 \rightarrow Q_4$).

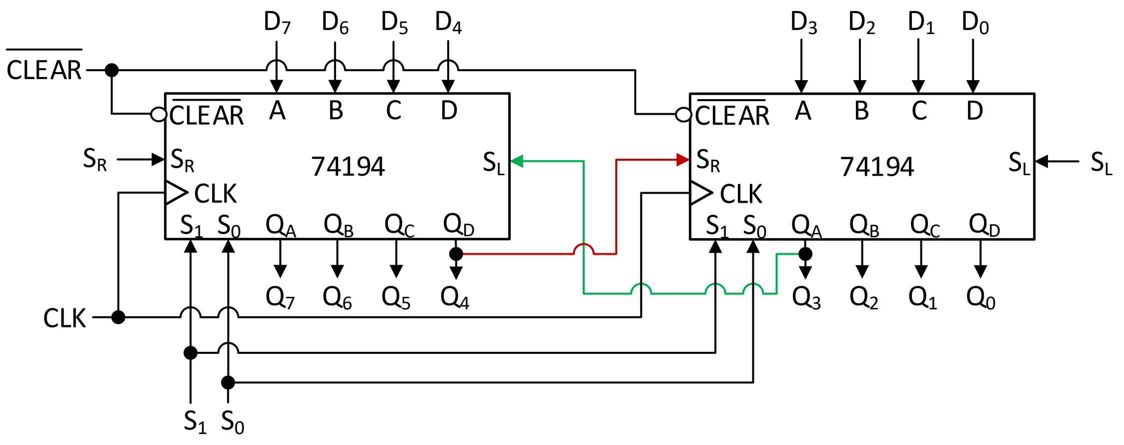


Figure 11. 5 Cascading two 74194 registers

11.2.4 Numerical sequence generators

Numerical sequences are arrays of values, which repeat in the same order. Such sequences can be easily generated using shift registers and additional combinational logic. A sequence containing 2^n-1 non-zero random *n*-bit values is called a *pseudo-random sequence*. Such a sequence can be generated using a register, initially loaded with any non-zero value. The register must perform data shift, while serially loading the XOR result between the most significant output bit and the least significant output bit. Figure 11. 6 shows two versions for implementing a 4-bit pseudo-random sequence generator using register 74194. The Initial value 0001_2 is loaded at startup, by setting $S_{1:0}$=11, for at least 1 clock cycle.

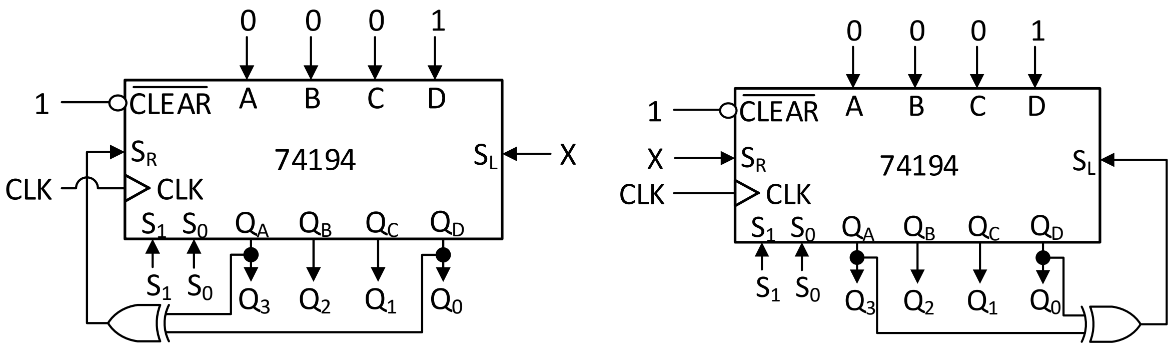


Figure 11. 6 The 4-bit pseudo-random sequence generator, implemented using register 74194, with serial load on S_R (left) and with serial load on S_L (right)

In the first case (Figure 11. 6 – left), after loading the initial value, the functionality is changed to right shifting with serial load on S_R, by setting $S_{1:0}$=01. The sequence generated on $Q_{3:0}$ contains the following 15 values: 0001 → 1000 → 1100 → 1110 → 1111 → 0111 → 1011 → 0101 → 1010 → 1101 → 0110 → 0011 → 1001 → 0100 → 0010. After the last value the sequence restarts, while keeping the same order.

In the second case (Figure 11. 6 – right), turning $S_{1:0}$=10 after the initial load, the register enables the left shift regime with serial load on S_L. The recurrent sequence generated on outputs becomes: 0001 → 0011 → 0111 → 1111 → 1110 → 1101 → 1010 → 0101 → 1011 → 0110 → 1100 → 1001 → 0010 → 0100 → 1000.

The implementation of the first sequence using register 7495 is highlighted in Figure 11. 7.

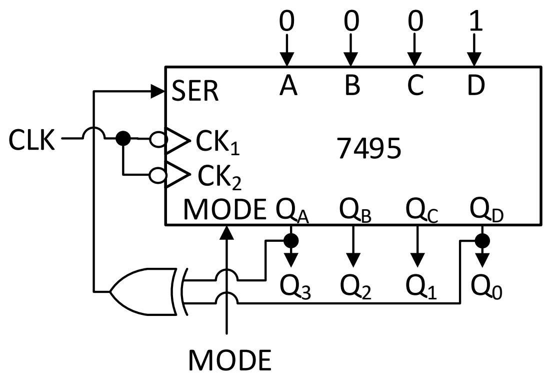


Figure 11. 7 The 4-bit pseudo-random sequence generator, implemented using 7495

The command MODE is set to MODE=1, for at least one clock cycle, and value $(0001)_2$ gets loaded into the register. Afterwards, MODE is set to MODE=0, and the rest of sequence is generated. Every transition between values takes place on the clock rising edge.

Note: A zero value should never be loaded, because the output will not change afterwards.

To generate pseudo-random sequences on a larger number of bits, the registers must be cascaded, as in Figure 11. 8. After loading the non-zero value, the ensemble transitions to data shifting regime. To implement a 6-bit sequence, 4 bits can be generated by one register, and 2 bits by another register.

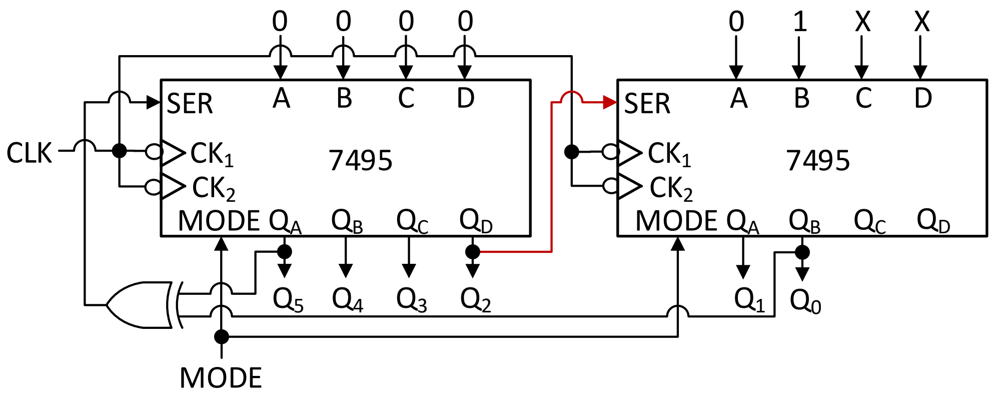


Figure 11. 8 A 6-bit pseudo-random sequence generator, implemented using cascaded 7495 registers

11.3 Assignments

1. Implement on the board the 7495 register. Test the commands acting on the clock falling edge. Considering Q_A is the most significant bit, load the following values, using parallel load and serial load: 13, 15, 17.
2. Implement on the board the 74194 register. Test the asynchronous reset, and the commands synchronous on the clock rising edge. Considering Q_A is the most significant bit, after an asynchronous reset, load the following values, using parallel load and serial load (in both directions): 10, 12, 6.
3. Implement and test in Logisim a 4-bit pseudo-random sequence generator starting with value 0001_2, loaded at startup. Implement using the 7495 register.
4. Implement and test in Logisim a 6-bit pseudo-random sequence generator starting with value 000001_2, loaded at startup. Implement using 7495 registers.
5. Implement in Logisim an 8-bit universal shift register, by cascading two 74194 registers. Test the asynchronous reset, and the commands synchronous on the clock rising edge. After an asynchronous reset, load the following values, using parallel load and serial load (in both directions): 17, 31, 67.