

Logic Design – Lab 6: “Shift registers, Adders, Comparators”

For the report, please complete the tasks marked in green boxes only (ver. 2020)

1. Shift registers

In digital circuits, a shift register is a cascade of flip flops, sharing the same clock, in which the output of each flip-flop is connected to the 'data' input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the 'bit array' stored in it, 'shifting in' the data present at its input and 'shifting out' the last bit in the array, at each transition of the clock input.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as 'serial-in, parallel-out' (SIPO) or as 'parallel-in, serial-out' (PISO). There are also types that have both serial and parallel input and types with serial and parallel output. There are also 'bidirectional' shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected to create a 'circular shift register'.

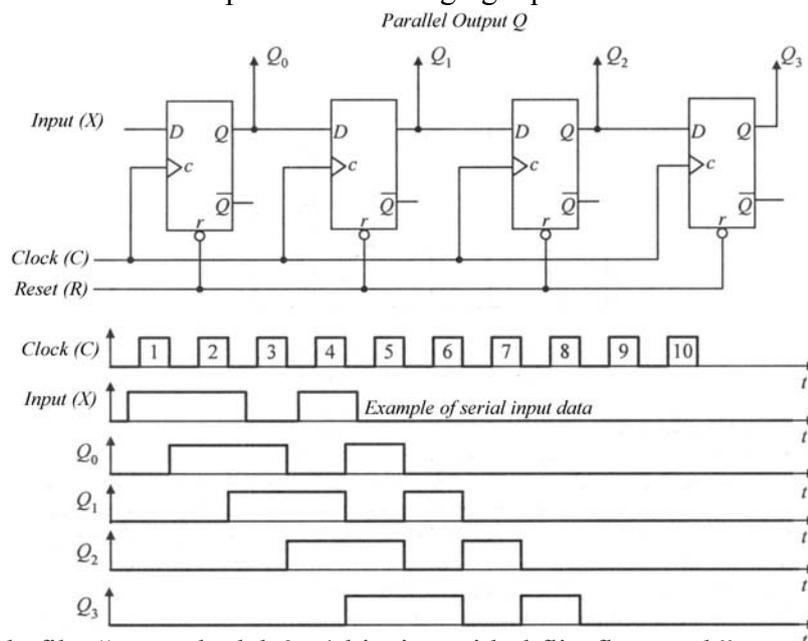
4-Bit Serial-in, parallel-out (SIPO) register

This configuration allows conversion from serial to parallel format. Data is input serially. Once the data has been clocked in, it may be either read off at each output simultaneously, or it can be shifted out.

In this configuration, each flip-flop is edge triggered. The initial flip-flop operates at the given clock frequency. Each subsequent flip-flop halves the frequency of its predecessor, which doubles its duty cycle. As a result, it takes twice as long for the rising/falling edge to trigger each subsequent flip-flop; this staggers the serial input in the time domain, leading to parallel output.

In general, the practical application of the serial-in/parallel-out shift register is to convert data from serial format on a single wire to parallel format on multiple wires.

Lab Task 1: With use of Electronic Workbench (EWB) construct the given SIPO register circuits and observe how it operates. Note states of all register outputs for successive clock pulses with changing input data bit X.



Tip: see example file: “example_lab6 - 4-bit sipo with d flip-flops.ewb”

2. Adders

An adder, also called summer, is a digital circuit that performs addition of numbers. In many computers and other kinds of processors, adders are used not only in the arithmetic logic units, but also in other parts of the processor, where they are used to calculate addresses, table indices, increment and decrement operators, and similar operations.

Although adders can be constructed for many numerical representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement or ones' complement is being used to represent negative numbers, it is trivial to modify an adder into an adder-subtractor. Other signed number representations require more logic around the basic adder.

Half adder

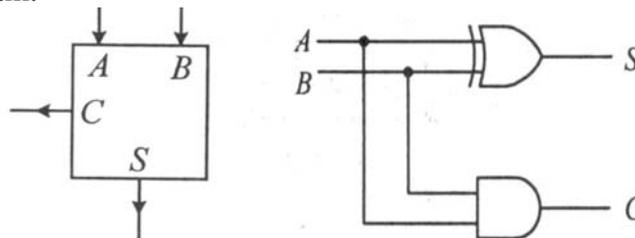
The half adder adds two single binary digits A and B. It has two outputs, sum (S) and carry (C). The carry signal represents an overflow into the next digit of a multi-digit addition. The value of the sum is $2C + S$. The simplest half-adder design, pictured on the right, incorporates an XOR gate for S and an AND gate for C. With the addition of an OR gate to combine their carry outputs, two half adders can be combined to make a full adder. The half adder adds two input bits and generates a carry and sum, which are the two outputs of a half adder. The input variables of a half adder are called the augend and addend bits. The output variables are the sum and carry. The truth table for the half adder is as follow:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A \ B	0	1
0	0,0	1,0
1	1,0	0,1

S, C

Half adder logic diagram:



Full adder

A full adder gives the number of 1s in the input in binary representation. Schematic symbol for a 1-bit full adder with C_{in} and C_{out} drawn on sides of block to emphasize their use in a multi-bit adder

A full adder adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as A, B, and C_{in} ; A and B are the operands, and C_{in} is a bit carried in from the previous less-significant stage. The full adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. bit binary numbers. The circuit produces a two-bit output, output carry and sum typically represented by the signals C_{out} and S, where $sum = 2 \times C_{out} + S$.

A full adder can be implemented in many different ways such as with a custom transistor-level circuit or composed of other gates. One example implementation is with

$$S = A \oplus B \oplus C_{in} \text{ and } C_{out} = (A \cdot B) + (C_{in} \cdot A) + (C_{in} \cdot B).$$

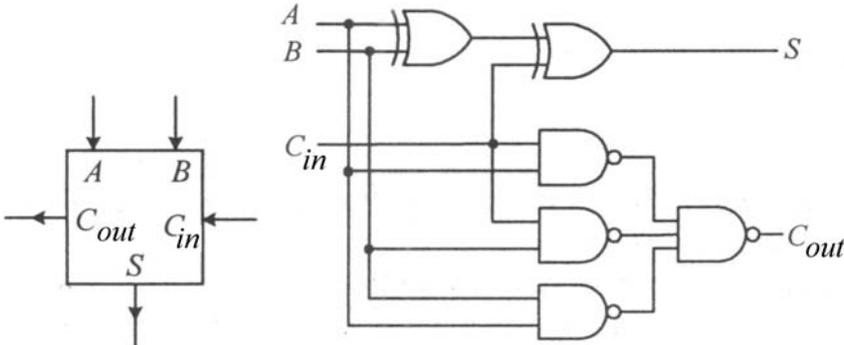
is as follow:

A	B	C _{in}	S	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

		AB			
		00	01	11	10
C _{in}	0	0,0	1,0	0,1	1,0
	1	1,0	0,1	1,1	0,1

S, C_{out}

Full adder logic diagram:



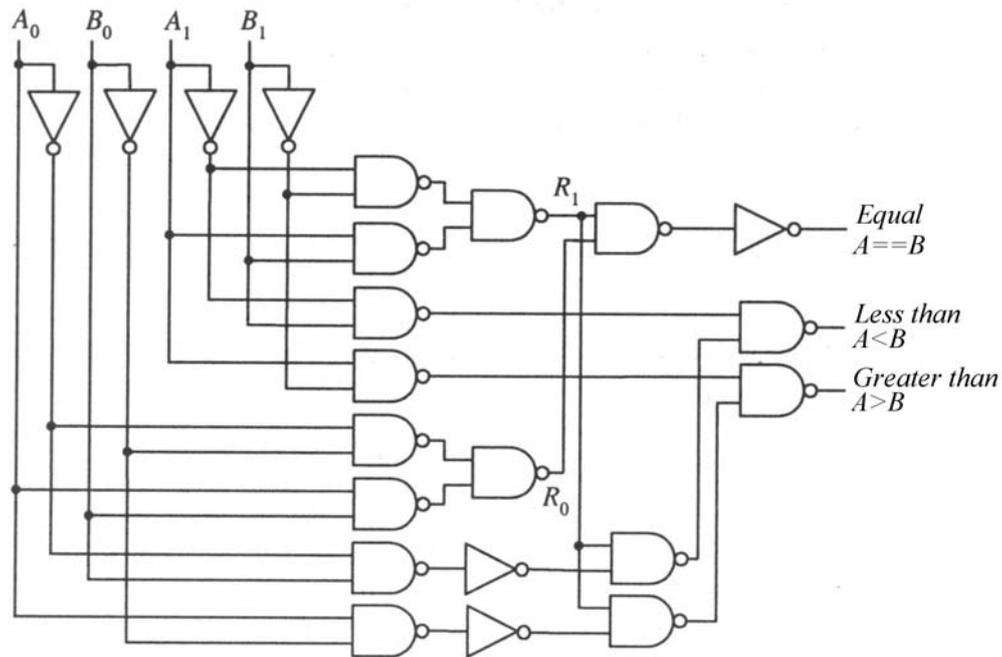
Lab Task 2: With use of Electronic Workbench (EWB) construct the half and full adders and verify how they doing.

3. Digital Comparators

A digital comparator or magnitude comparator is a hardware electronic device that takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number.

(Note: An XNOR single gate is the simplest 1-bit comparator, because its output is "1" only if its two input bits are equal.)

2-bit numbers full comparator logic circuit built with NAND gates only:



where A and B are 2-bits input numbers to compare ($A = A_1 A_0$, $B = B_1 B_0$).

The truth tables for the 2-bit full comparator:

$A_1 A_0$ \ $B_1 B_0$	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$A = B$

$A_1 A_0$ \ $B_1 B_0$	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$A < B$

$A_1 A_0$ \ $B_1 B_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$A > B$

Lab Task 3: Using of Electronic Workbench (EWB) construct 2-bit full comparator and make comparison for different values of A and B (where A, B can be 0, 1, 2, 3 in decimal).