NTNU Department of Electronics and Telecommunications	**TFE4141 Design of Digital Systems 1**

Assignment 1: Terms and definitions used among digital designers

Q1: What is $01000_2 + 01001_2$?

$$\begin{array}{r} 01000 \\ +\ 01001 \\ \hline 10001 \end{array}$$

Q2: Which number do 1111_2 represent?

a) In 4-bit unsigned format

b) In 4-bit signed format

a. 4-bit unsigned format of 1111_2 is:
   $= 1.2^3 + 1.2^2 + 1.2^1 + 1.2^0$
   $= 8 + 4 + 2 + 1$
   $= 15$ (in decimal)
b. 4-bit signed format of 1111_2 is:
   $= -(1.2^2 + 1.2^1 + 1.2^0)$
   $= -(4 + 2 + 1)$
   $= -7$ (in decimal)

Q2: Which number do 1111_2 represent?

a) In 4-bit unsigned format

b) In 4-bit signed format (2's complement)

a. 4-bit unsigned format of 1111_2 is:
   $= 1.2^3 + 1.2^2 + 1.2^1 + 1.2^0$
   $= 8 + 4 + 2 + 1$
   $= 15$ (in decimal)
b. 4-bit signed format of 1111_2 (2's complement) is:
   1's complement of 1111_2 is:
   $= 1000_2$ [keeping the first bit constant as it is the signed bit]

2's complement of 1000_2 is:

$$\begin{array}{r} 1000 \\ +\ 1 \\ \hline 1001 \end{array}$$

= - 1 (in decimal)

Q3: What does "dynamic range" in the context of number representation mean?

Dynamic range is the ratio of the maximum absolute value representable and the minimum positive (i.e., non-zero) absolute value representable.

Q4: What has highest dynamic range of a 32 bit floating point number and a 32 bit fixed point number?

Dynamic range for a fixed point 32-Bit (unsigned) number is:
$2^N - 1 = 2^{31} - 1 = 2.15 \times 10^9$
Dynamic range for a floating point 32-bit number (as per IEEE Standard 754, single-precision) is:
$(2-2^{-23}) \times 2^{127} = 3.402823 \times 10^{38}$

The highest value, as mentioned above, is for the 32-bit floating point number. Therefore, single-precision binary floating-point is used due to its wider range over fixed point (of the same bit-width), even if at the cost of precision.

Q5: How are floating point numbers and fixed point numbers spaced across the dynamic range?

With fixed-point notation, the gaps between adjacent numbers always equal a value of one, whereas in floating-point notation, gaps between adjacent numbers are not uniformly spaced – the gap between any two numbers is approximately ten million times smaller than the value of the numbers (ANSI/IEEE Std. 754 standard format), with large gaps between large numbers and small gaps between small numbers.

Q6: Add the numbers from Q1 in the same way as thought at school.

$$\begin{array}{r} 01000 \\ +\ 01001 \\ \hline 10001 \end{array}$$

Q7: A boolean function can be constructed for computing the LSB in the addition of two fixed-point numbers (as done in Q6).

a) What is a Boolean function
b) Create a truth table for this Boolean function.
c) Create the Boolean function based on the truth table

a) Boolean function-
A Boolean function is described by an algebraic expression called a Boolean expression that contains binary variables (0 and 1) and logic operation symbols.

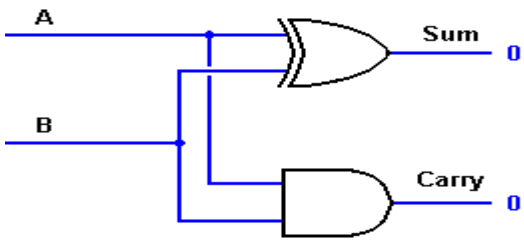
b) Truth table of the above Boolean function, for computing the LSB, results to:

A	B	F(A, B)
0	0	0
0	1	1
1	0	1
1	1	1

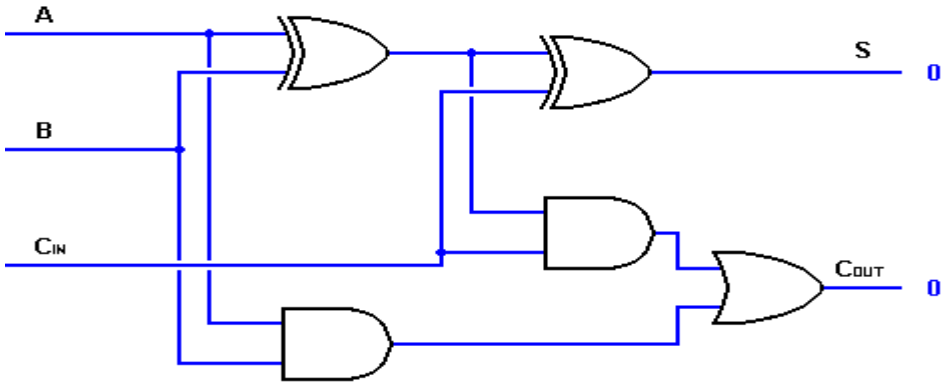
c) Based on the truth table the Boolean function is:
$F\ (A, B) = A + B$
Where,
A = 0
B = 1
LSB of the binary numbers mentioned in Q6.

Q8: What is a half-adder and what is a full-adder?

Half Adder:
With the help of half adder, we can design circuits that can perform simple addition with the help of logic gates.



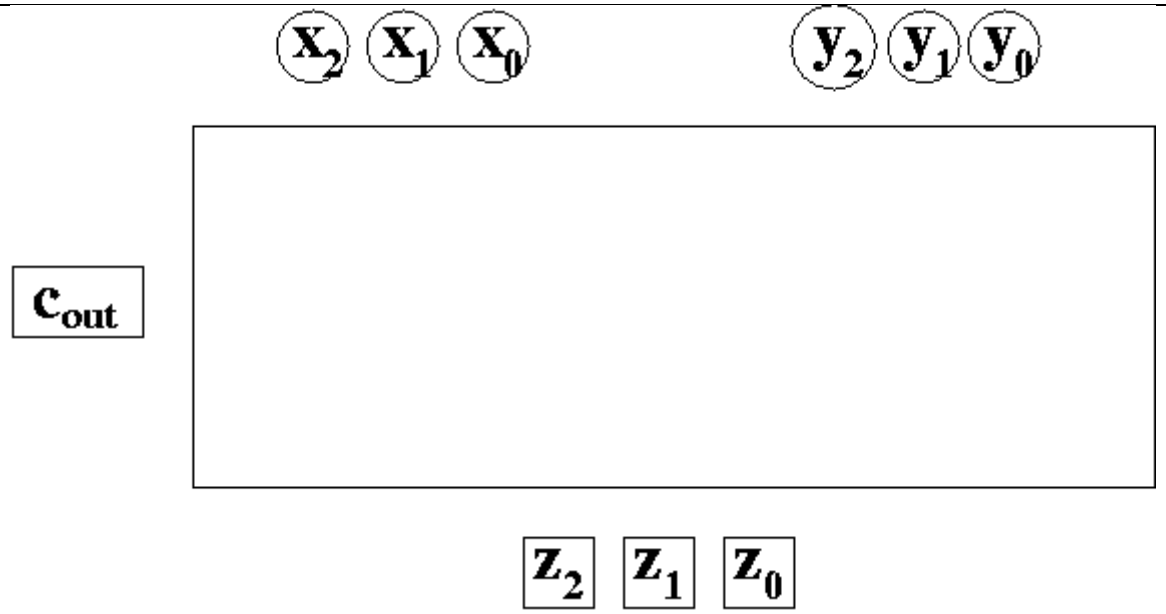
Full Adder:
This type of adder is a little more difficult to implement than a half-adder. The main difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs. The first two inputs are A and B and the third input is an input carry designated as CIN. When a full adder logic is designed we will be able to string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.



Q9: Make something cool/useful out of full-adders and half-adders. Draw a diagram

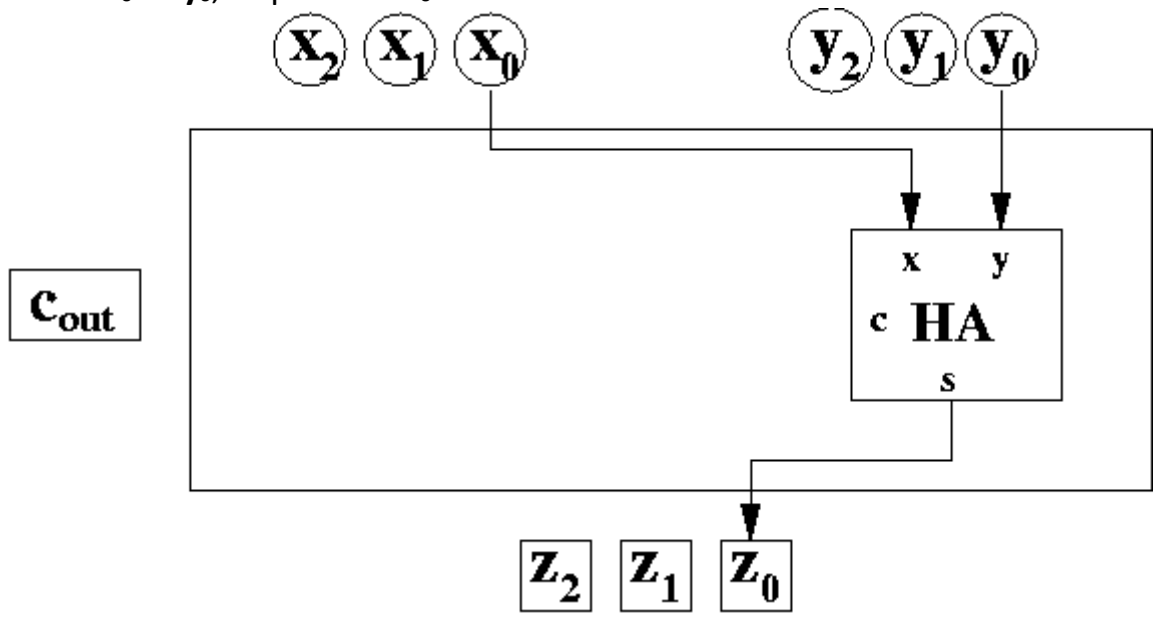
A Ripple Carry Adder can be prepared using half adder and full adder circuits. A ripple carry adder allows you to add two k-bit numbers. We use the half adders and full adders and add them a column at a time.

Before Adding



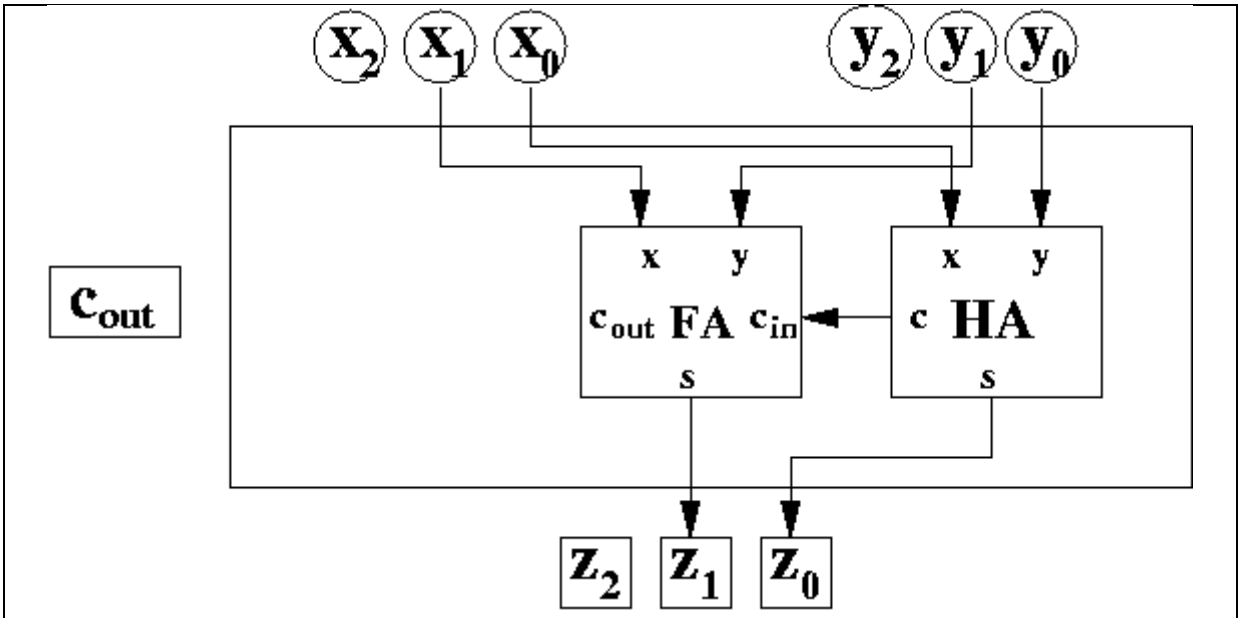
Adding Column 0

We add x_0 to y_0, to produce z_0.



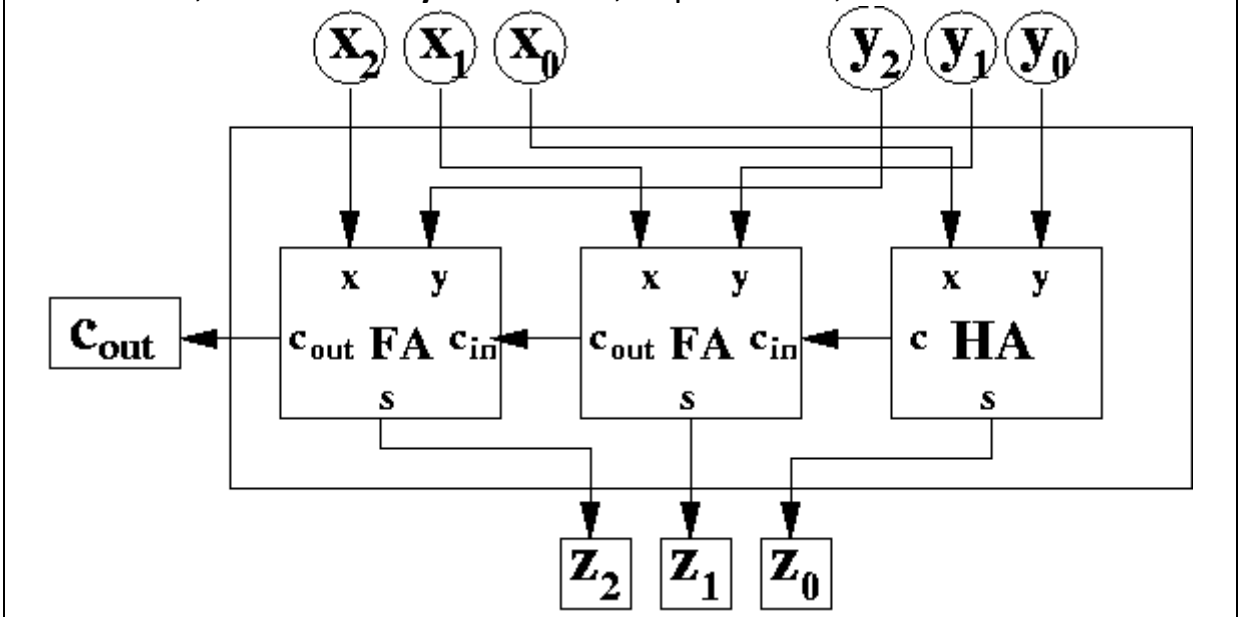
Adding Column 1

In column 1, We add x_1 to y_1 and c_1 and, to produce z_1, and c_2, where c_i is the carry-in for column **i** and c_{i+1} is the carry-out of column **i**.



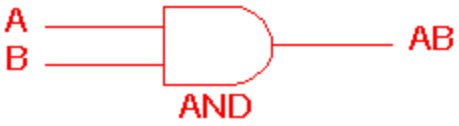
Adding Column 2

In column 2, We add x_2 to y_2 and c_2 and, to produce z_2, and c_3.



Q10: Draw symbols for all the logic gates you can think of.

AND gate

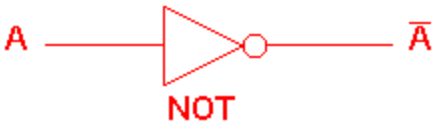


2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

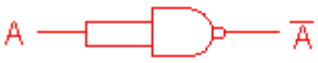
OR gate

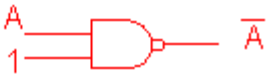
2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate



NOT gate	
A	$\overline{A}$
0	1
1	0

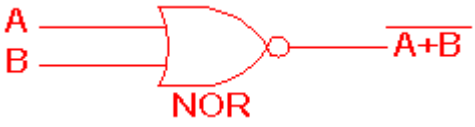




NAND gate

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate

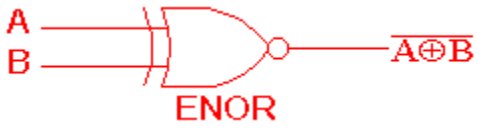


2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

EXOR gate

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

EXNOR gate



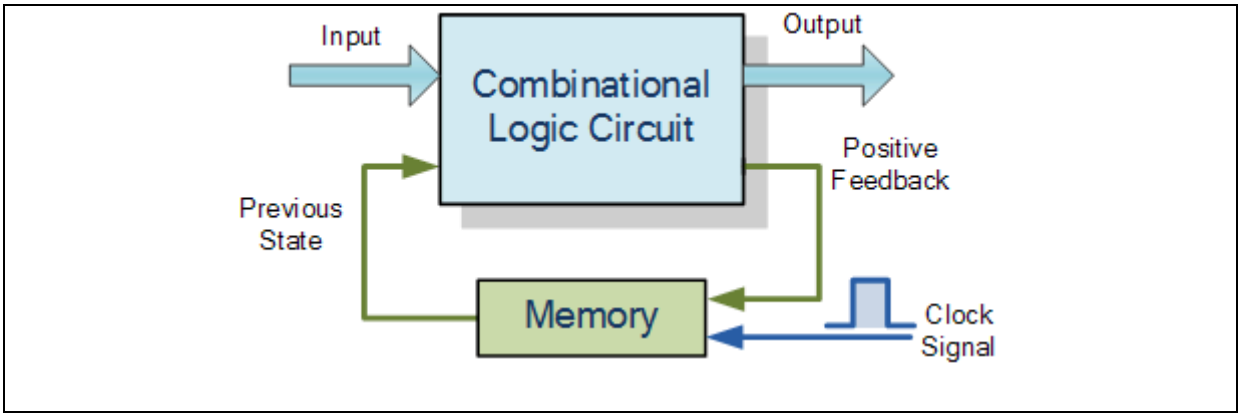
2 Input EXNOR gate		
A	B	$\overline{A⊕B}$
0	0	1
0	1	0
1	0	0
1	1	1

Q11: What is sequential logic?

Sequential logic circuits are able to take into account their previous input state as well as those actually present, a sort of “before” and “after” effect is involved with sequential circuits. Sequential Logic circuits have some form of inherent “Memory” built in.

The output state of a “sequential logic circuit” is a function of the following three states, the “present input”, the “past input” and/or the “past output”. Sequential Logic circuits remember these conditions and stay fixed in their current state until the next clock signal changes one of the states, giving sequential logic circuits “Memory”.

Sequential logic circuits are generally termed as two state or Bistable devices which can have their output or outputs set in one of two basic states, a logic level “1” or a logic level “0” and will remain “latched” (hence the name latch) indefinitely in this current state or condition until some other input trigger pulse or signal is applied which will cause the bistable to change its state once again.

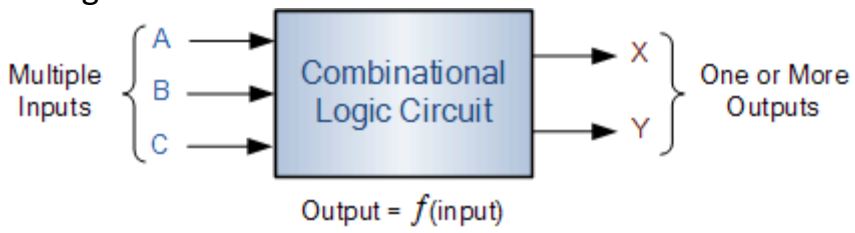


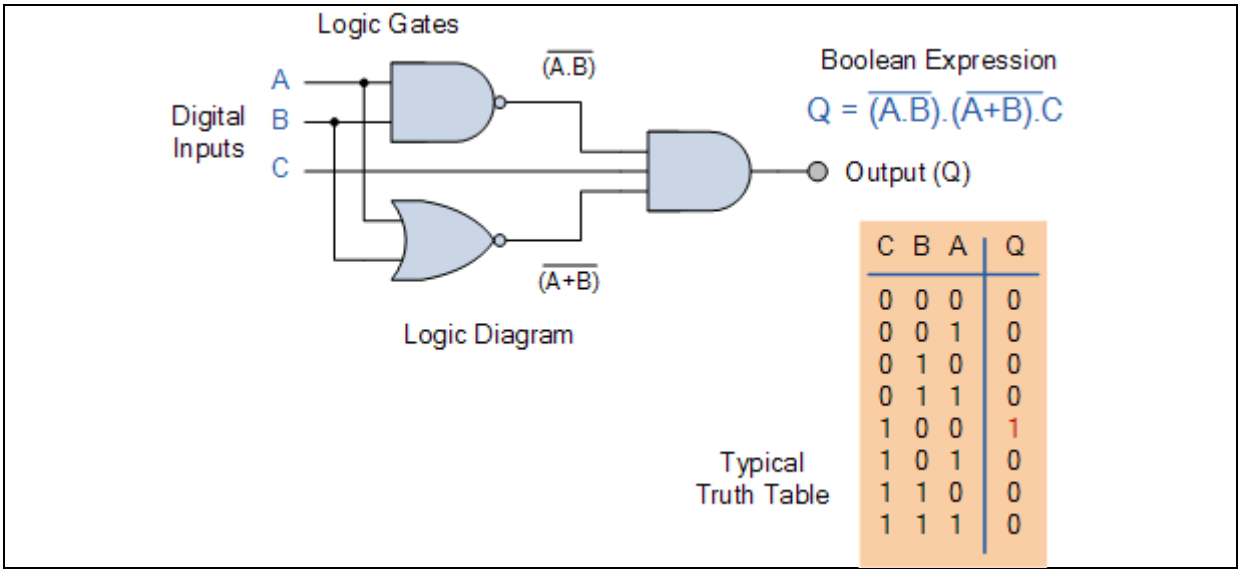
Q12: What is combinational logic?

In digital circuit theory, combinational logic (sometimes also referred to as time-independent logic[1]) is a type of digital logic which is implemented by Boolean circuits, where the output is a pure function of the present input only.

The outputs of Combinational Logic Circuits are only determined by the logical function of their current input state, logic “0” or logic “1”, at any given instant in time.

The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a Combinational Logic Circuit, the output is dependant at all times on the combination of its inputs. So if one of its inputs condition changes state, from 0-1 or 1-0, so too will the resulting output as by default combinational logic circuits have “no memory”, “timing” or “feedback loops” within their design.
Combinational Logic Circuits are made up from basic logic NAND, NOR or NOT gates that are “combined” or connected together to produce more complicated switching circuits. These logic gates are the building blocks of combinational logic circuits.

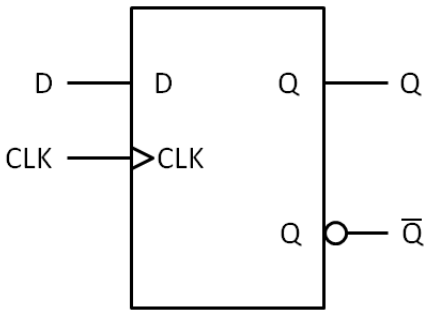




Q13: D flip flop

a) What is a D flip-flop?
b) Draw the symbol commonly used for representing a D flip-flop.
c) Why do some flip-flops have reset?
d) Find VHDL code for a D flip-flop with synchronous reset.

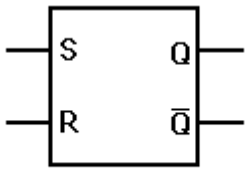
a) The D flip-flop tracks the input, making transitions with match those of the input D. The D stands for "data"; this flip-flop stores the value that is on the data line. It can be thought of as a basic memory cell. A D flip-flop can be made from a set/reset flip-flop by tying the set to the reset through an inverter. The result may be clocked.



b)

c) The set/reset type flip-flop is triggered to a high state at Q by the "set" signal and holds that value until reset to low by a signal at the Reset input. This can be implemented as a NAND gate latch or a NOR gate latch and as a clocked version.

One disadvantage of the S/R flip-flop is that the input S=R=0 gives ambiguous results and must be avoided. The J-K flip-flop gets around that problem.



Set/Reset Type

d) Following are the code lines for D Flip-Flop with Synchronous Reset, Set and Clock Enable:

```
--library declaration for the module.
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--This is a D Flip-Flop with Synchronous Reset,Set and
Clock Enable(posedge clk).
--Note that the reset input has the highest priority,Set
being the next highest
--priority and clock enable having the lowest priority.
entity example_FDRSE is
   port(
      Q : out std_logic;       -- Data output
      CLK :in std_logic;       -- Clock input
      CE :in std_logic;     -- Clock enable input
      RESET :in std_logic;  -- Synchronous reset input
      D :in  std_logic;       -- Data input
      SET : in std_logic    -- Synchronous set input
   );
end example_FDRSE;
architecture Behavioral of example_FDRSE is  --architecture
of the circuit.
begin  --"begin" statement for architecture.
process(CLK) --process with sensitivity list.
begin  --"begin" statment for the process.
  if ( rising_edge(CLK) ) then  --This makes the process
synchronous(with clock)
    if (RESET = '1') then
         Q <= '0';
     else
           if(SET = '1') then
             Q <= '1';
           else
              if ( CE = '1') then
               Q <= D;
             end if;
           end if;
      end if;
  end if;
end process;  --end of process statement.
end Behavioral;
```

Q14: Latch (The evil cousin of the D flip-flop)

a) What is a latch.
b) Try googling “unwanted latches”.

Make sure you know everything there is to know about “unwanted latches” before you write any RTL code.

a) A latch is an electronic logic circuit that has two inputs and one output. One of the inputs is called the SET input; the other is called the RESET input.

Latch circuits can be either active-high or active-low. The difference is determined by whether the operation of the latch circuit is triggered by HIGH or LOW signals on the inputs.

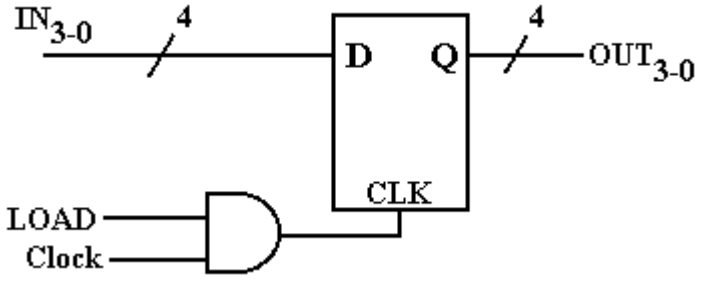
b) Unwanted Latches:
- When developing synchronous digital designs, latches can generate asynchronous circuits that become unstable.
- One of the main reasons that latches are not used in designs is due to combinational loop back and subsequent oscillation which creates super problems in simulation, and also simulation Vs synthesis mismatches.
- Due to controllability issues in latches unlike FF's, they can significantly harm STA/DFT and thereby reduce coverage in both cases.
- The major factor for the above uncontrollability are Glitches in the enable pin of the latches that can cause unrecoverable failure
- Inferred latches after synthesis due to bad coding styles.

Q15: What is a register and how are registers related to D flip-flops?

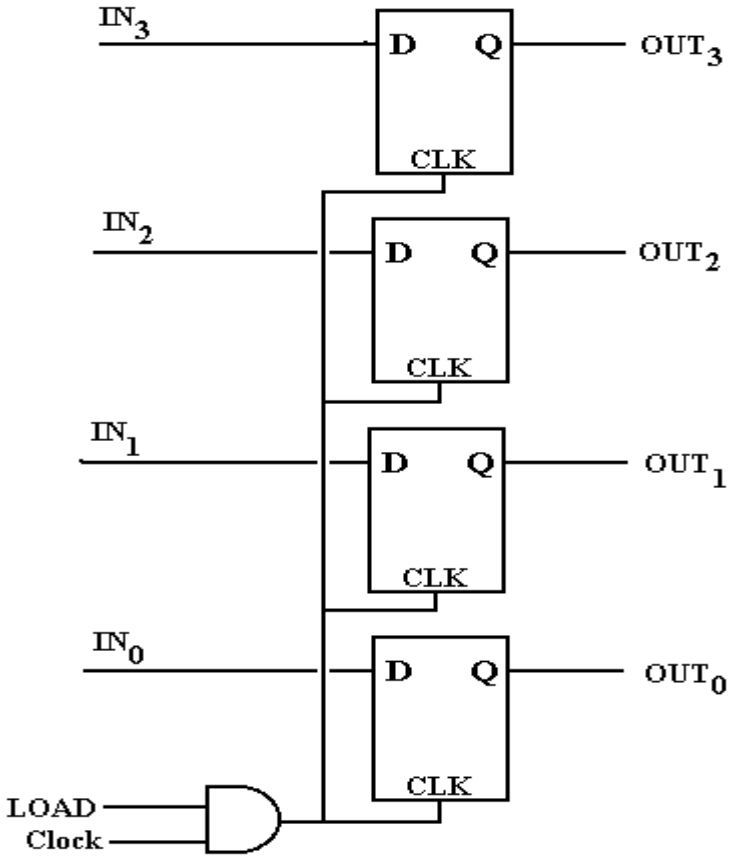
A register is a storage device capable of holding binary data. It is best viewed as a collection of flip-flops, usually D flip-flops. To store N bits, a register must have N flip-flops, one for each bit to be stored. We show a design for a four-bit register with a synchronous LOAD.

The figure at right shows a short-hand notation used when drawing registers that contain a number of flip-flops identically configured.

It should be obvious that the figure represents a 4-bit register.



Q16: Draw a 4-bit register

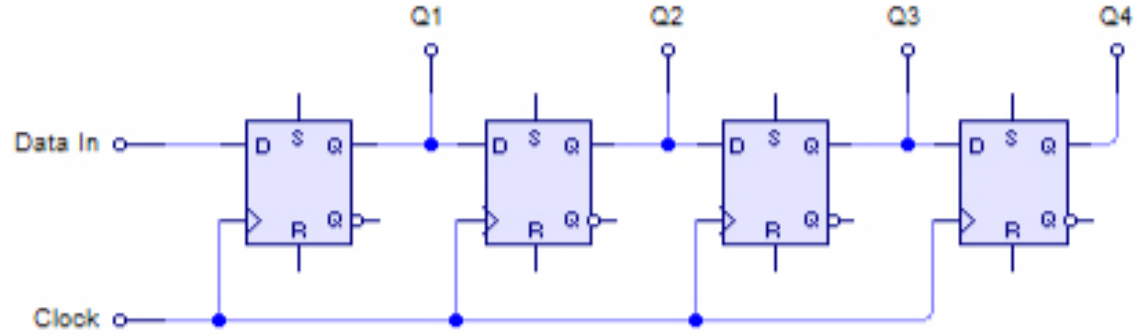


Note the input CLK comes from an AND gate that puts out the logical AND of the system clock (Clock) and the LOAD signal. When LOAD is 0, the flip-flops are cut off from the input and do not change state in response to the input. The design calls for LOAD to be 1 for almost one clock pulse, so that the system clock and LOAD are both high for 1/2 clock cycle. At this time, the register is loaded.

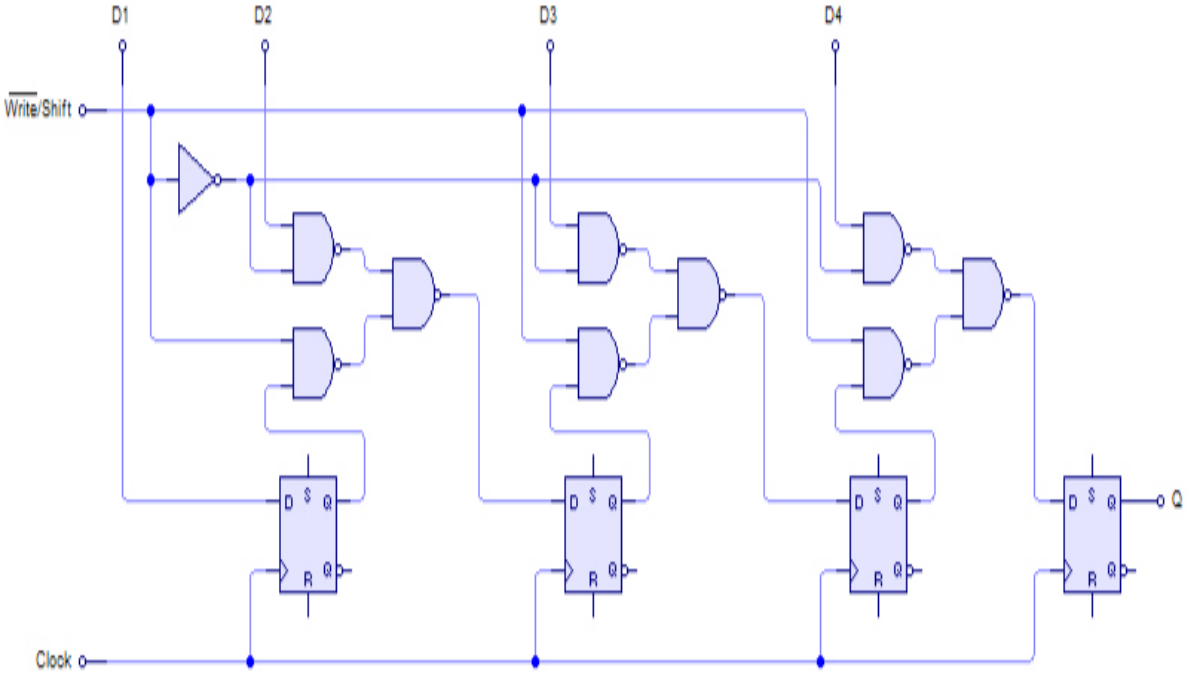
Q17: Draw a 4-bit shift-register

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).

SIPO:



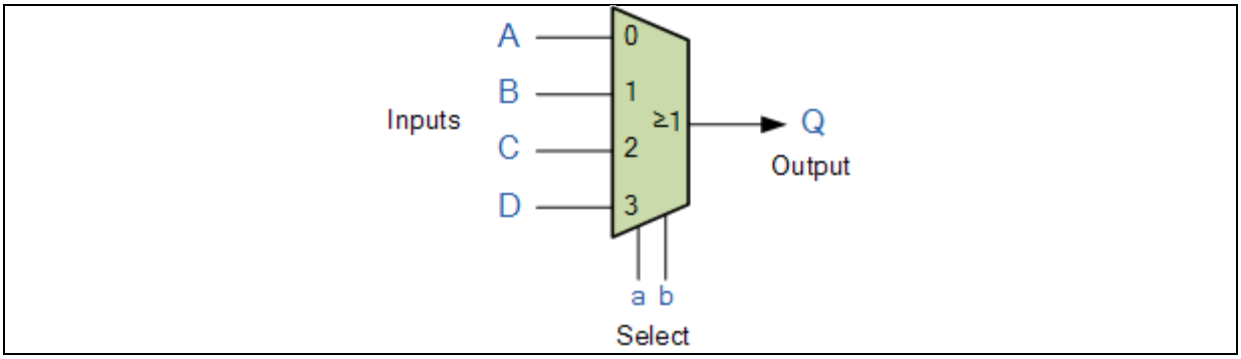
PISO



Q18: What is a Mux?

Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a Multiplexer (MUX in short).

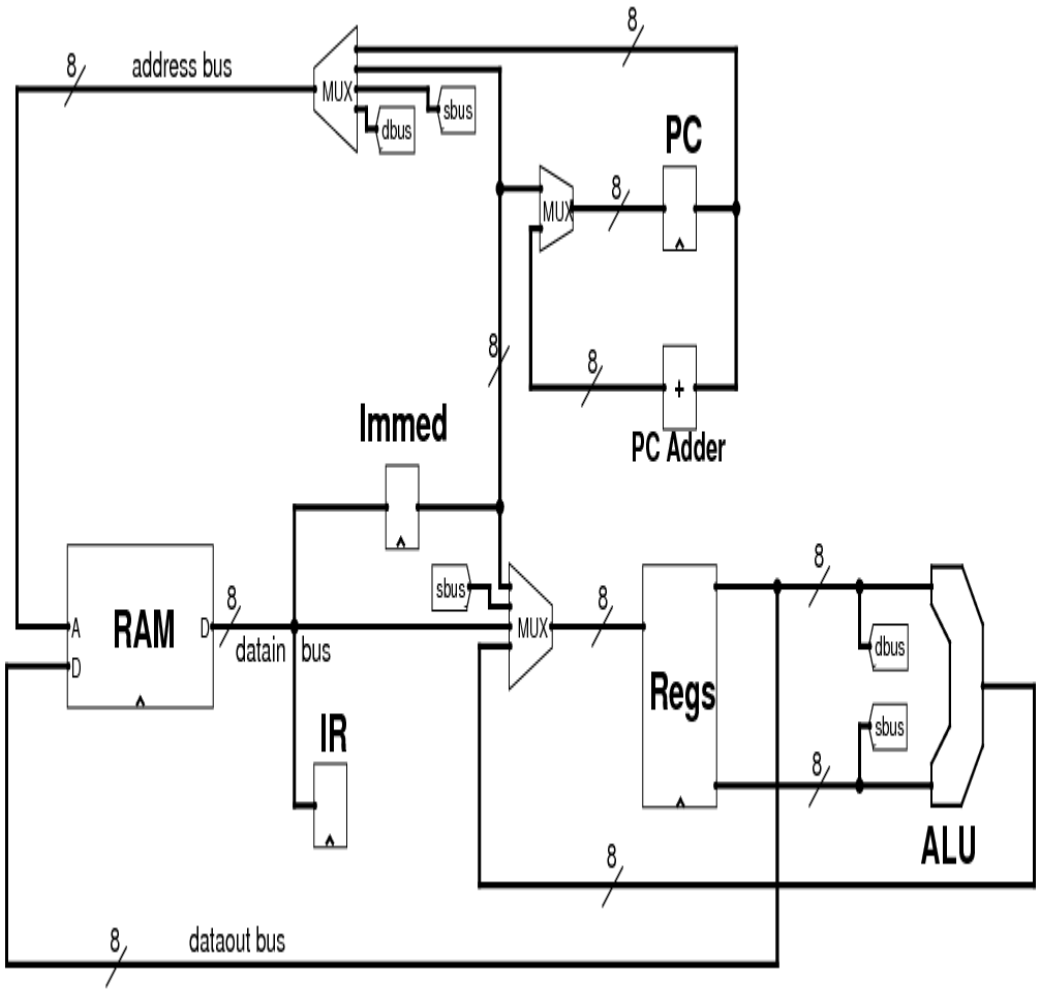
Symbol of MUX:



Q19: Google “CPU datapath”, print out one example Datapath and explain what we are looking at.

A datapath is a collection of functional units (such as arithmetic logic units or multipliers, that perform data processing operations), registers, and buses. Along with the control unit it composes the central processing unit (CPU).

The following diagram shows the datapaths in the CPU:



- The *dbus* and *sbus* labels indicate the lines coming out from the register file which hold the value of the destination and source registers.
- Note the data loop involving the registers and the ALU, whose output can only go back into a register.
- The dataout bus is only connected to the *dbus* line, so the only value which can be written to memory is the destination register.
- Also note that there are only 3 multiplexors:
    - the address bus multiplexor can get a memory address from the PC, the immediate register (for direct addressing), or from the source or destination registers (for register indirect addressing).
    - the PC multiplexor either lets the PC increment, or jump to the value in the immediate register.
    - the multiplexor in front of the registers determines where a register write comes from: the ALU, the immediate register, another register or the data bus.

Q20: What is the purpose of pipelining in computing?

In computers, a pipeline is the continuous and somewhat overlapped movement of instruction to the processor or in the arithmetic steps taken by the processor to perform an instruction. Pipelining is the use of a pipeline. Without a pipeline, a computer processor gets the first instruction from memory, performs the operation it calls for, and then goes to get the next instruction from memory, and so forth. While fetching (getting) the instruction, the arithmetic part of the processor is idle. It must wait until it gets the next instruction. With pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing arithmetic operations, holding them in a buffer close to the processor until each instruction operation can be performed. The staging of instruction fetching is continuous. The result is an increase in the number of instructions that can be performed during a given time period.

Pipelining is sometimes compared to a manufacturing assembly line in which different parts of a product are being assembled at the same time although ultimately there may be some parts that have to be assembled before others are. Even if there is some sequential dependency, the overall process can take advantage of those operations that can proceed concurrently.

Computer processor pipelining is sometimes divided into an instruction

pipeline and an arithmetic pipeline. The instruction pipeline represents the stages in which an instruction is moved through the processor, including its being fetched, perhaps buffered, and then executed. The arithmetic pipeline represents the parts of an arithmetic operation that can be broken down and overlapped as they are performed.

Pipelines and pipelining also apply to computer memory controllers and moving data through various memory staging places.

Q21: What is logic synthesis?

Logic synthesis is the process of converting a high-level description of design into an optimized gate-level representation. Logic synthesis uses a standard cell library which have simple cells, such as basic logic gates like and, or, and nor, or macro cells, such as adder, muxes, memory, and flip-flops. Standard cells put together are called technology library. Normally the technology library is known by the transistor size (0.18u, 90nm).

Q22: What is high-level synthesis?

High-level synthesis (HLS), sometimes referred to as C synthesis, electronic system-level (ESL) synthesis, algorithmic synthesis, or behavioural synthesis, is an automated design process that interprets an algorithmic description of a desired behaviour and creates digital hardware that implements that behaviour.

Q23: What does EDA stand for in the context of digital design?

Electronic design automation (EDA), also referred to as electronic computer-aided design (ECAD),[1] is a category of software tools for designing electronic systems such as integrated circuits and printed circuit boards. The tools work together in a design flow that chip designers use to design and analyze entire semiconductor chips. Since a modern semiconductor chip can have billions of components, EDA tools are essential for their design.

Q24: What does “place & route” refer to?

Place and route is a stage in the design of printed circuit boards, integrated circuits, and field-programmable gate arrays. As implied by the name, it is composed of two steps, placement and routing. The first step, placement, involves deciding where to place all electronic components, circuitry, and logic elements in a generally limited amount of space. This is followed by routing, which decides the exact design of all the wires needed to connect the placed components. This step must implement all the desired connections while following the rules and limitations of the manufacturing process.

Q25: What is static timing analysis (STA)?

Static timing analysis (STA) is a simulation method of computing the expected timing of a digital circuit without requiring a simulation of the full circuit.

Q26: Is negative slack a good thing?

Yes, because negative slack implies that a path is too slow, and the path must be sped up (or the reference signal delayed) if the whole circuit is to work at the desired speed.

Q27: What does “critical path” mean?

Critical path simply means the path that violates the timing the most. Usually that is the path with the longest delay.

Q28: What does the terms “setup-time” and “hold-time” for a flip-flop mean?

Setup time is defined as the minimum amount of time before the clock's active edge that the data must be stable for it to be latched correctly. Any violation may cause incorrect data to be captured, which is known as setup violation.
Hold time is defined as the minimum amount of time after the clock's active edge during which data must be stable. Violation in this case may cause incorrect data to be latched, which is known as a hold violation. Note that setup and hold time is measured with respect to the active clock edge only.

Q29: What does “design for testability” (DFT) mean?

Design for testing or design for testability (DFT) consists of IC design techniques that add testability features to a hardware product design. The added features make it easier to develop and apply manufacturing tests to the designed hardware. The purpose of manufacturing tests is to validate that the product hardware contains no manufacturing defects that could adversely affect the product’s correct functioning.

Q30: What is a scan chain?

Scan chain is a technique used in design for testing.
The objective is to make testing easier by providing a simple way to set and observe every flip-flop in an IC.
The basic structure of scan includes the following set of signals in order to control and observe the scan mechanism.:

- Scan_in and scan_out define the input and output of a scan chain. In a full scan mode usually each input drives only one chain and scan out observe one as well.
- A scan enable pin is a special signal that is added to a design. When this signal is asserted, every flip-flop in the design is connected into a long shift register.
- Clock signal which is used for controlling all the FFs in the chain during shift phase and the capture phase. An arbitrary pattern can be entered into the chain of flip-flops, and the state of every flip-flop can be read out.

Q31: What is formal verification?

In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.
Formal verification can be helpful in proving the correctness of systems such as: cryptographic protocols, combinational circuits, digital circuits with internal memory, and software expressed as source code.
Verification is one aspect of testing a product's fitness for purpose.

Q32: What is a state-machine and what role do they play in digital circuits?

A finite-state machine (FSM) or finite-state automaton (FSA, plural: automata), finite automaton, or simply a state machine, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSM can change from one state to another in response to some external inputs; the change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state, and the conditions for each transition.

In a digital circuit, an FSM may be built using a programmable logic device, a programmable logic controller, logic gates and flip flops or relays. More specifically, a hardware implementation requires a register to store state variables, a block of combinational logic that determines the state transition, and a second block of combinational logic that determines the output of an FSM. One of the classic hardware implementations is the Richards controller.
In a Medvedev machine, the output is directly connected to the state flip-flops minimizing the time delay between flip-flops and output.
Through state encoding for low power state machines may be optimized to minimize power consumption.