EDA322/ DIT797: Digital Design
Exam - June 2018

Date: June 8, 2018

Time: **14:00-18:00**

Examiner: Ioannis Sourdis

Department: Computer Science and Engineering

Inquiries: Ioannis Sourdis (extension 1744); will visit the room at **15:30** and at **17:00**

Results and grading review: room 4128 EDIT on **June 29th at 11:00**.

Duration: 4 hours

Grading scale: 100 points in total

Chalmers:
0: 0%-49%, 3: 50%-64%, 4: 65%-84%, 5: 85%-100%
GU:
Fail (U): 0%-49%, Pass (G): 50%-79%, Pass with Distinction (VG): 80%-100%

Available references: a calculator is allowed. No textbooks or lecture notes, etc. allowed.

General: Submit your solutions, in English, on blank paper sheets. Write legibly; feel free to use figures to get your point across.

The order of answering the questions does not matter (start with the easiest ones).

Please start the solutions for each problem on a new sheet. Please number the sheets so that the solutions are in numerical order.

Note that it is possible to receive partial credit for an answer even if it is not 100% correct.

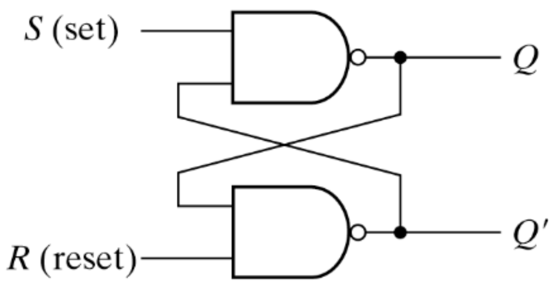
Your personal identity code is required on each submitted sheet!

Good luck!

Question 1 Sequential Circuits: (10 points)
Draw the (i) gatelevel block diagram, (ii) the truth table and (iii) the state-diagram of a NAND-based SR-latch.

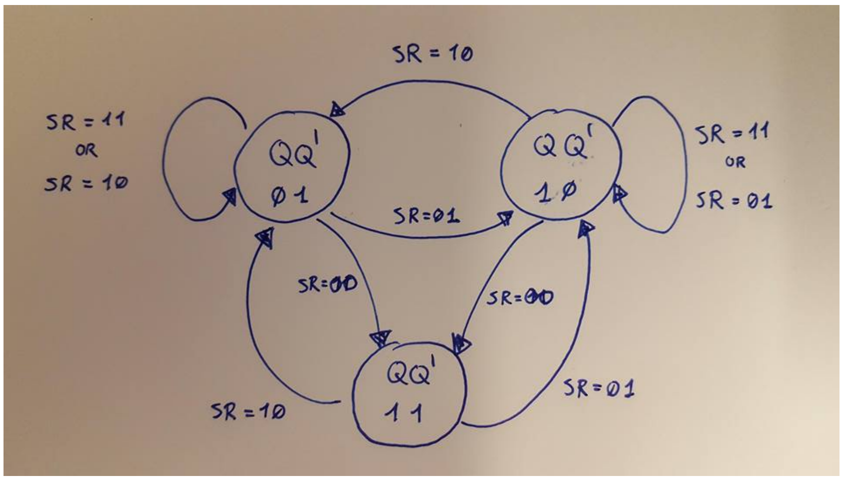
Answer

The block diagram can be observed on the left side of the following figure. Its behavior is described in the truth table on the right.



S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(after $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(after $S = 0, R = 1$)
0	0	1	1	

The state diagram of the NAND-SR latch:



When SR = 11, there's no state change. For SR = 00, the 'not-allowed' state QQ' = 11 is reached. When SR = 10, we have QQ' = 01, i.e. the state moves to QQ' = 01 if it was in QQ' = 10, otherwise it stays in QQ' = 01. The opposite reasoning applies when SR = 01.

Question 2 Interfaces: (10 points)
Give one example of a digital module that uses each of the following interfaces:

(i) a flow control interface,
(ii) a push flow control interface, and
(iii) a pull flow control interface.

Explain your choices.

Answer
The input of a FIFO is usually a flow control interface. It receives a request signal that indicated data to be enqueued are "valid" and sends back to the sender a "ready" ("not full") signal to receive new data.

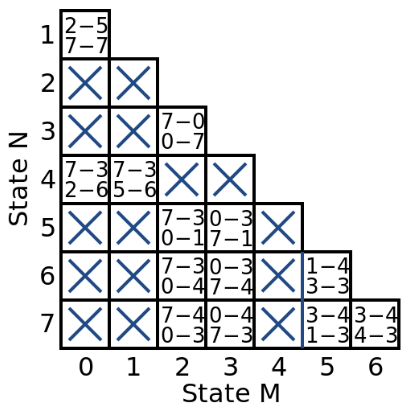
The write port of a memory is usually a push flow control interface. Data are ready to be written in a memory location when the write signal is set (valid) and there is no signal from the memory to the sender indicating whether the memory is ready to receive the data or not.

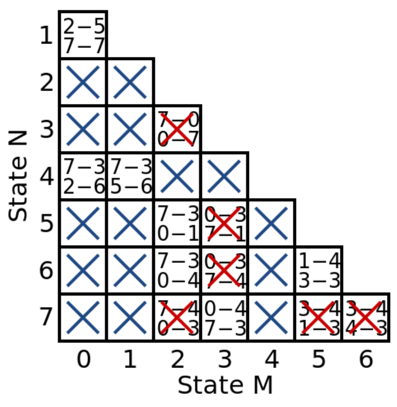
The output of a synchronous memory (that has no read enable signal) uses a pull interface as it always provides valid content (the content of the memory entry pointed by the read address in the previous cycle). Another example could be the output of a sensor (e.g. a temperature sensor)

Question 3 FSMs: (10 points)
Minimize the states of the FSM described by the following state table.

Current State	Next State		Output	
	Input X		*Input X*	
	0	1	0	1
0	7	2	0	0
1	7	5	0	0
2	7	0	1	0
3	0	7	1	0
4	3	6	0	0
5	3	1	1	0
6	3	4	1	0
7	4	3	1	0

Answer





Current State	Next State		Output	
	Input X		Input X	
	0	1	0	1
0	3	2	0	0
2	3	0	1	0
3	0	3	1	0

Question 4 Asynchronous: (10 points)
a) Detect any race conditions in the asynchronous circuit defined by the following state table:

State	Code (c,a,b)	Next (in) 0	1	Out (a,b)
A	000	Ⓐ	B	00
B	110	C	Ⓑ	10
C	100	Ⓒ	D	00
D	001	A	Ⓓ	01

b) Make the necessary modifications to eliminate race conditions

Answer

Lecture on asynchronous slides 33, 34

a) Detect the race condition

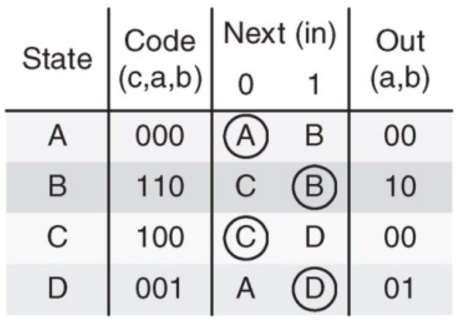
Transition from A (cab=000) to B (110) has 2 state bits changing, "a" and "c".

If "a" changes first then state will be 010 and then, when "c" changes, it will be 110 which is state B.
If "c" changes first then the end state will be C (100) which then changes to D

b) Fix the race condition

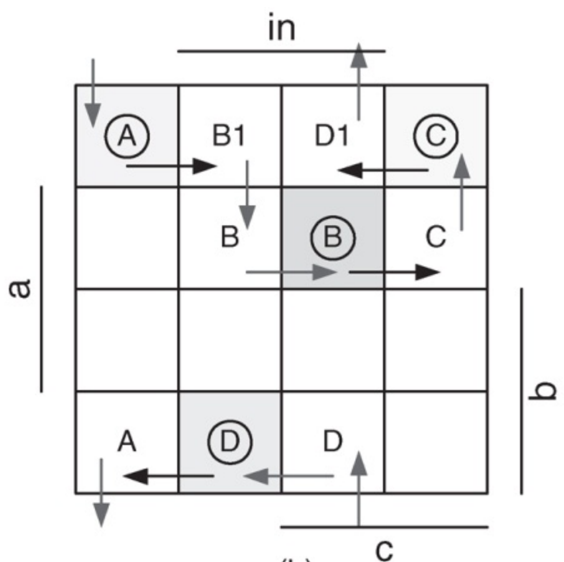
To fix the race condition we need to introduce two new states:
State B1 (010): when input changes to 1 in state A we go to state B1 and then to B.

State D1 (101) for transition from state C (100) to state D (001)



State	Code (c,a,b)	Next (in) 0	Next (in) 1	Out (a,b)
A	000	A	B	00
B	110	C	B	10
C	100	C	D	00
D	001	A	D	01

(a)



(b)

Question 5 Pipelining: (10 points)
Consider that an instruction requires to go through (all) the following microprocessor steps in order to be executed:

1. **Fetch**: instruction read
2. **Read:** registers (operands) read
3. **ALU:** execute instruction
4. **Mem:** Data memory access
5. **Write:** write back result to registers

Consider the following delays:

- Fetch 1 ns,
- Read 0,8 ns,
- ALU 1,1 ns,
- Mem 1 ns,
- Write 0,8 ns
- Setup time of a flipflop 0,1ns
- Propagation time of a flipflop 0.1ns
- Hold time of a flipflop 0,05ns

What would be the latency and throughput of the microprocessor if
a) the above steps are performed in one cycle (unpipelined).
b) the microprocessor is pipelined and each of the above 5 steps is a separate pipeline stage.

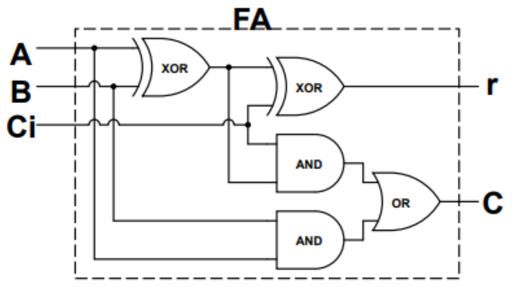
Answer:
a) latency = 1+0,8+1,1+1+0,8 = 4,7ns,
throughput = 1 instruction per 4,7 ns = 212 Minstr/sec

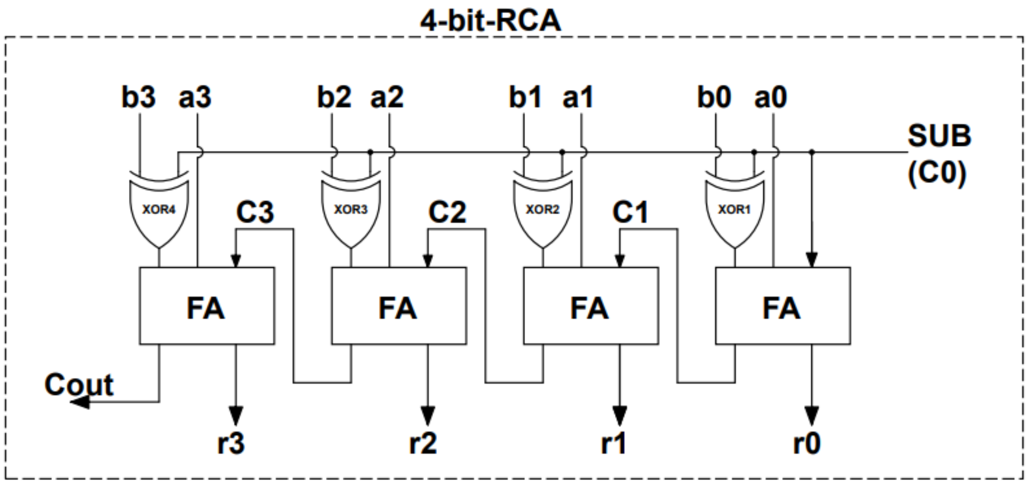
b) latency = 5*(longest stage) = 5*(1,1+0,1+0,1) = 5*1,3= 6,5 ns, throughput = 1 instr. per the longest stage delay = 1instr. /1,3ns = 769 Minstr/sec

Question 6 Testing & Timing: (10 points)
Draw the gate-level block diagram of a 4-bit ripple carry adder.

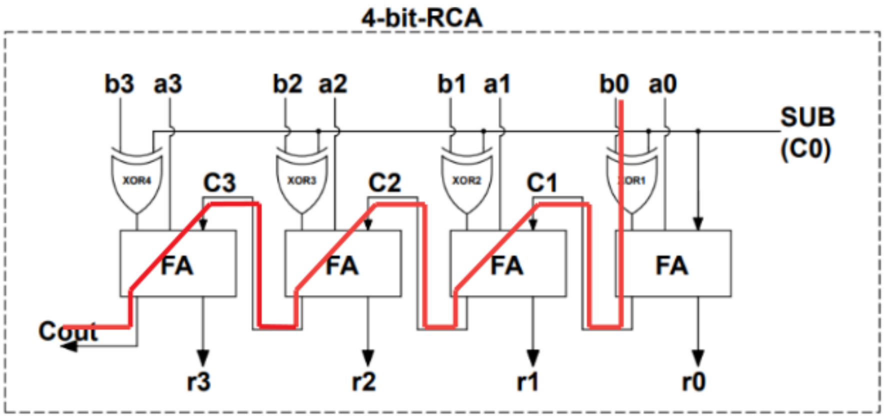
a) How would you modify the adder to also support subtraction?
b) Find the critical path delay of the circuit in (a) if a 2-input XOR gate has a latency of 2 ns and a 2-input OR or AND gate has a latency of 1 ns.
c) Find two input test vectors for the circuit in (a), one testing for a stuck-at-1 and the other for a stuck-at-0 in the carry-in signal of the 4th full adder.

Answer
a) One of the inputs XORed with a signal indicating subtraction, that signal also drives the carry-in of the adder.





b) The critical path is 12ns, starting from b[0] which is Xor-ed with the SUB signal to Cout or r[3].



XOR1 = 2ns
1st FA = (XOR)2ns + (AND)1ns + (OR)1ns = 4ns
2nd FA = (AND)1ns + (OR)1ns = 2ns
3rd FA = (AND)1ns + (OR)1ns = 2ns
4th FA = (AND)1ns + (OR)1ns = 2ns
TOTAL = 12ns

c) You can apply the path sensitization algorithm to activate and propagate the fault. Consider the inputs SUB(C0), a[0-3] and b[0-3] of the 4bit adder.

Stuck at 0 in the carry in of the 4th full adder (C3):

Then if a[2]=1 and b[2]=1 then C3 should be 1, which is the opposite than the stuck-at-0 value so it activates the fault.

To propagate the fault, we need to propagate the C3 to the Cout of the 4th full adder. As we know from the carry lookahead equation for carry propagation a[3] and b[3] should have different values, so either a[3]=1, b[3]=0 or a[3]=0, b[3]=1

Stuck at 1 in the carry in of the 4th full adder (C3):
Then if a[2]=0 and b[2]=0 then C3 should be 0, which is the opposite than the stuck-at-1 value so it activates the fault.
The propagation of the faulty value is the same as in the stuck-at-0 case.

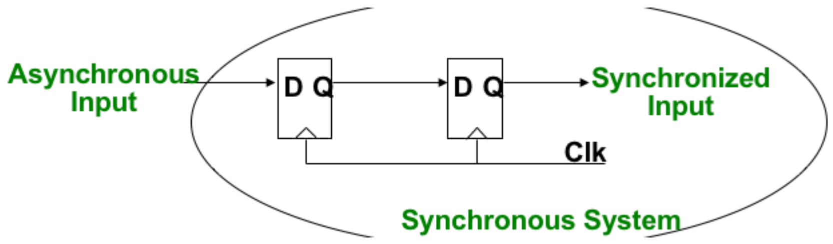
Question 7 Timing: (10 points)

a) What is clock skew and how can it be avoided?
b) What is metastability and how can it be avoided in a flip flop with asynchronous input?

Answer
a) the clock does not tick at every flipflop of a design exactly at the same time. It can be fixed by careful layout of the clock tree so that the distance between the clock generator and the clock tree leaves (that give clock to the flipflops) is almost the same. This can be done using a H-tree structure of a clock tree.

b) When a flipflop input changes too close to clock edge (after the setup-time and before the hold time), the flipflop may enter the metastable state: neither a logic 0 nor a logic 1. It may stay in this state an indefinite amount of time, although this is not likely in real circuits.

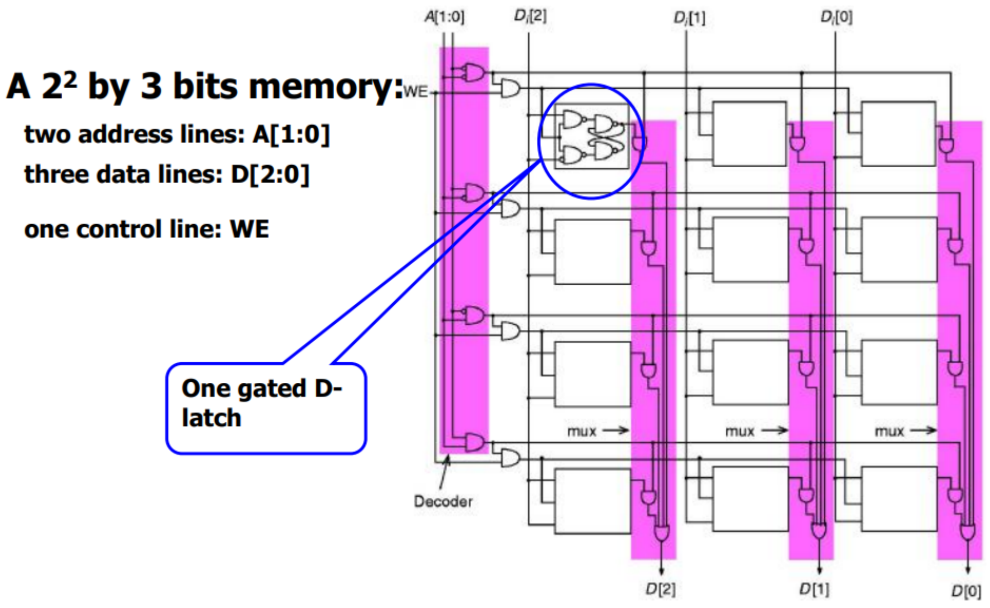
The probability of failure can never be 0, but it can be reduced by :
- slowing down the system clock. This gives the synchronizer more time to decay into a steady state.
- using fastest possible logic in the synchronizer. This makes for a very sharp "peak" upon which to balance S or AS TTL D-FFs are recommended
- cascade two synchronizers



Question 8 Memories: (10 points)
a) Draw the gatelevel block diagram of a 4x3 memory block (4 entries of 3 bits wide) without showing the internals of a single bit memory cell.
b) Create a 8x6 memory made out of the above 4x3 memory blocks. Draw the new gatelevel block diagram (no need to draw again the internals of the 4x3 block). What is(are) the name(s) of the mechanism(s) used to create the new memory?

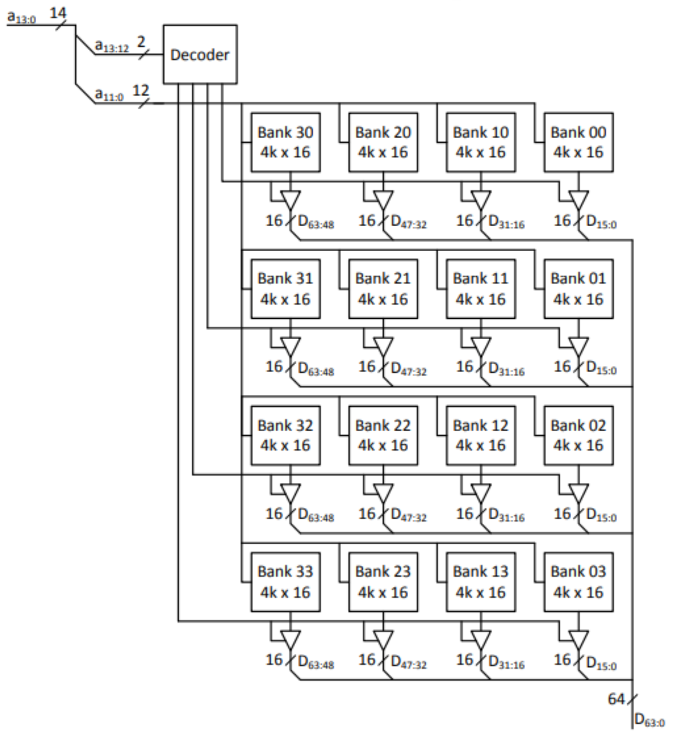
Answer

(a)



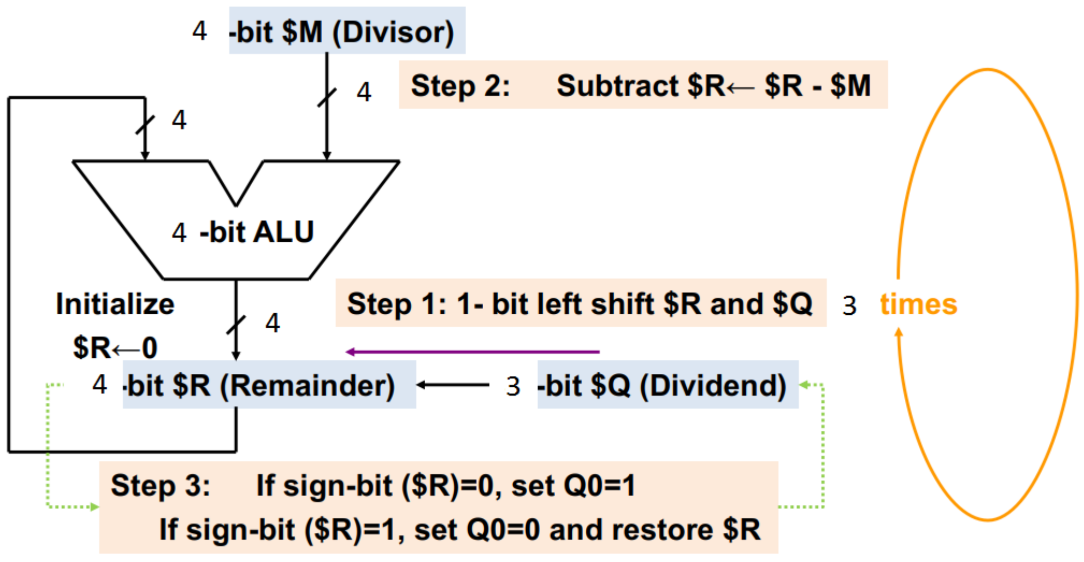
(b)

Bit slicing & banking



Question 9 Arithmetic: (10 points)
Draw the block diagram and describe the functionality of a 3-bit sequential divider. Show how you can use it to perform the division 7/3.

Answer



4 -bit $M (Divisor)
4
Step 2: Subtract $R← $R - $M
4
4 -bit ALU
Initialize $R←0
4
Step 1: 1- bit left shift $R and $Q
3 times
4 -bit $R (Remainder)
3 -bit $Q (Dividend)
Step 3: If sign-bit ($R)=0, set Q0=1
If sign-bit ($R)=1, set Q0=0 and restore $R

M and $R should be 4 bits, $Q should be 4 bits), number of iterations 3.

Action	**n**	**$R**	**$Q**	**$M**
Init	3	0000	111	0011
Shift Left {R,Q}	3	0001	110	0011
Add -M to R	3	1110	110	0011
Restore	3	0001	110	0011
Shift Left {R,Q}	2	0011	100	0011
Add -M to R	2	0000	100	0011
R>0, Q0 → 1	2	0000	101	0011
Shift Left {R,Q}	1	0001	010	0011
Add -M to R	1	1110	010	0011
Restore	1	0001	010	0011
	0			

7/3 = 2 with remainder $R=1

Question 10 Reconfigurable Hardware: (10 points)
Consider the implementation of the following function:

$$F = A_0A_1A_3 + A_1A_2\bar{A}_3 + \bar{A}_0\bar{A}_1\bar{A}_2$$

in 3 different FPGAs, each composed of logic cells with either (i) 4-input LUTs, (ii) 3-input LUTs, or (iii) 2-input LUTs.

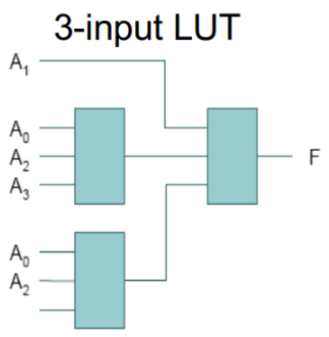
a) Show the mapping of the logic of function F in each of the 3 types of FPGAs. How many LUTs are needed in each case? How many bits of SRAM memory needed in total for each case?

b) Which is the most efficient choice of LUT size in terms of area? What happens to the delay of the function implementation in each case?

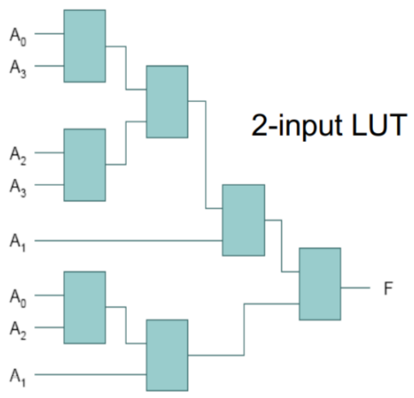
Answer

a)

Only 1 4-input LUT.
Required SRAM memory: $2^4 * 1 = 16$ bits
================================



3 LUTs.Required SRAM memory: 2^3 *3 = 24 bits
Required SRAM memory: 2^3 *3 = 24 bits
=================================



7 LUTs.
Required SRAM memory: 2^2*7 = 28 bits
=================================

b)

The FPGA with 4-input LUTs is the best choice. As illustrated in the figure in part (a), this option requires only one LUT and much less interconnects which makes it the most efficient choice in terms of area.
On the other hand, although larger LUTs are slower than smaller LUTs, the circuit depth of function F using 4-input LUT is only one. Therefore, due to having lower interconnects delay, 4-input LUT implementation will also be the best choice in terms of speed.

In general, the minimum LUT area is shown to be for K=4, but for different functions the most efficient size of LUT inputs depends on the typical complexity and structure of the logic. Bigger LUTs can handle complicated functions more efficiently and require less interconnects, but are slower than smaller LUTs and are less area efficient for regular functions.

END of EXAM