EDA321: Digital Design

Solutions of Final Exam - Spring 2013 study period 3

Question 1: (8 points)

a) Derive the canonical form of Sum of Products (SoP) for the following function F= a'b+abc+a'b'd (5 points)

b) Then, derive the canonical form of the Product of Sums (PoS) for the function F. (3 points)

Solution:

Can be solved in 3 different ways:

1. Making the truth table of the function and then finding the minterms (the sum of which is the SoP canonical form of F) and the maxterms (the product of which is the PoS of F). (similar to Lecture 2 slide 12)
2. Through Shannon Theorem (similar to Lecture 2 slide 16)
3. Expanding each product (similar to Lecture 2 slide 17)
4. After step 2. or 3. The PoS is derived (similar to Lecture 2 slide 12)

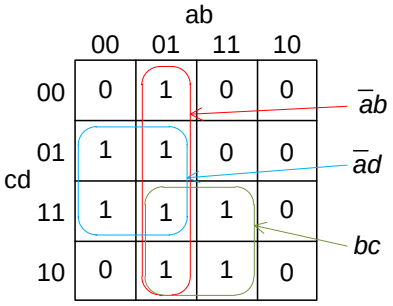
(Above mentioned lecture slides can be found at the end of this document)

a)

$$f = \bar{a} \cdot b + a \cdot b \cdot c + \bar{a} \cdot \bar{b} \cdot d$$

using Karnaugh or (truthtable) you can extract the minterms of the function

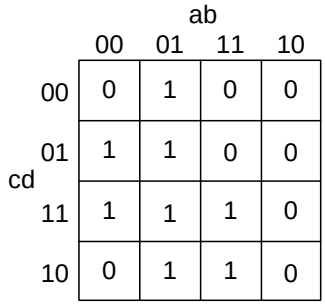
$$f = \bar{a} \cdot b + b \cdot c + \bar{a} \cdot d$$



Figur E1.1 Karnaugh-diagram förenklad SOP

The minterms are:

$$f(a, b, c, d) = \sum m(1, 3, 4, 5, 6, 7, 14, 15)$$



Figur E1.2 Karnaugh-diagram

So the canonical form of the SoP for F is:

$$f(a,b,c,d) = \bar{a}\cdot\bar{b}\cdot\bar{c}\cdot d + \bar{a}\cdot\bar{b}\cdot c\cdot d + \bar{a}\cdot b\cdot\bar{c}\cdot\bar{d} + \bar{a}\cdot b\cdot\bar{c}\cdot d + \\ + \bar{a}\cdot b\cdot c\cdot\bar{d} + \bar{a}\cdot b\cdot c\cdot d + a\cdot b\cdot c\cdot\bar{d} + a\cdot b\cdot c\cdot d$$

another way to solve it is option (3):

$$f = \bar{a}\cdot b + a\cdot b\cdot c + \bar{a}\cdot\bar{b}\cdot d = \bar{a}\cdot b\cdot(\bar{c}+c) + a\cdot b\cdot c\cdot(\bar{d}+d) + \bar{a}\cdot\bar{b}\cdot(\bar{c}+c)\cdot d = \\ = \bar{a}\cdot b\cdot\bar{c}\cdot(\bar{d}+d) + \bar{a}\cdot b\cdot c\cdot(\bar{d}+d) + a\cdot b\cdot c\cdot\bar{d} + a\cdot b\cdot c\cdot d + \bar{a}\cdot\bar{b}\cdot\bar{c}\cdot d + \bar{a}\cdot\bar{b}\cdot c\cdot d = \\ = \bar{a}\cdot b\cdot\bar{c}\cdot\bar{d} + \bar{a}\cdot b\cdot\bar{c}\cdot d + \bar{a}\cdot b\cdot c\cdot\bar{d} + \bar{a}\cdot b\cdot c\cdot d + a\cdot b\cdot c\cdot\bar{d} + a\cdot b\cdot c\cdot d + \bar{a}\cdot\bar{b}\cdot\bar{c}\cdot d + \bar{a}\cdot\bar{b}\cdot c\cdot d$$

b)

knowing the minterms, the easiest way for the PoS is to get it get the maxterms as explained in slide 13 Lecture 2. So,

$$f(a,b,c,d) = \prod M(0,2,8,9,10,11,12,13)$$

so

$$f(a,b,c,d) = (a+b+c+d)\cdot(\bar{a}+\bar{b}+c+d)\cdot(\bar{a}+b+c+d)\cdot(\bar{a}+\bar{b}+c+\bar{d})\cdot \\ (\bar{a}+b+c+\bar{d})\cdot(\bar{a}+b+\bar{c}+\bar{d})\cdot(a+b+\bar{c}+d)\cdot(\bar{a}+b+\bar{c}+d)$$

or else

$$\overline{f(a,b,c,d)} = \bar{a}\cdot\bar{b}\cdot\bar{c}\cdot\bar{d} + a\cdot b\cdot\bar{c}\cdot\bar{d} + a\cdot\bar{b}\cdot\bar{c}\cdot\bar{d} + a\cdot b\cdot\bar{c}\cdot d + \\ + a\cdot\bar{b}\cdot\bar{c}\cdot d + a\cdot\bar{b}\cdot c\cdot d + \bar{a}\cdot\bar{b}\cdot c\cdot\bar{d} + a\cdot\bar{b}\cdot c\cdot\bar{d}$$

$$f(a,b,c,d) = \overline{\overline{f(a,b,c,d)}} = \overline{\bar{a}\cdot\bar{b}\cdot\bar{c}\cdot\bar{d} + a\cdot b\cdot\bar{c}\cdot\bar{d} + a\cdot\bar{b}\cdot\bar{c}\cdot\bar{d} + a\cdot b\cdot\bar{c}\cdot d + }$$

$$\overline{+ a\cdot\bar{b}\cdot\bar{c}\cdot d + a\cdot\bar{b}\cdot c\cdot d + \bar{a}\cdot\bar{b}\cdot c\cdot\bar{d} + a\cdot\bar{b}\cdot c\cdot\bar{d}} =$$

$$= \overline{\overline{a}\cdot\overline{b}\cdot\overline{c}\cdot\overline{d}}\cdot\overline{a\cdot b\cdot\overline{c}\cdot\overline{d}}\cdot\overline{a\cdot\overline{b}\cdot\overline{c}\cdot\overline{d}}\cdot\overline{a\cdot b\cdot\overline{c}\cdot d}\cdot\overline{a\cdot\overline{b}\cdot\overline{c}\cdot d}\cdot\overline{a\cdot\overline{b}\cdot c\cdot d}\cdot\overline{\overline{a}\cdot\overline{b}\cdot c\cdot\overline{d}}\cdot\overline{a\cdot\overline{b}\cdot c\cdot\overline{d}} =$$

$$= \left(\overline{\overline{a}}+\overline{\overline{b}}+\overline{\overline{c}}+\overline{\overline{d}}\right)\cdot\left(\overline{a}+\overline{b}+\overline{\overline{c}}+\overline{\overline{d}}\right)\cdot\left(\overline{a}+\overline{\overline{b}}+\overline{\overline{c}}+\overline{\overline{d}}\right)\cdot\left(\overline{a}+\overline{b}+\overline{\overline{c}}+\overline{d}\right)\cdot$$

$$\left(\overline{a}+\overline{\overline{b}}+\overline{\overline{c}}+\overline{d}\right)\cdot\left(\overline{a}+\overline{\overline{b}}+\overline{c}+\overline{d}\right)\cdot\left(\overline{\overline{a}}+\overline{\overline{b}}+\overline{c}+\overline{\overline{d}}\right)\cdot\left(\overline{a}+\overline{\overline{b}}+\overline{c}+\overline{\overline{d}}\right)=$$

$$= (a+b+c+d)\cdot\left(\overline{a}+\overline{b}+c+d\right)\cdot\left(\overline{a}+b+c+d\right)\cdot\left(\overline{a}+\overline{b}+c+\overline{d}\right)\cdot$$

$$\left(\overline{a}+b+c+\overline{d}\right)\cdot\left(\overline{a}+b+\overline{c}+\overline{d}\right)\cdot\left(a+b+\overline{c}+d\right)\cdot\left(\overline{a}+b+\overline{c}+d\right)$$

Question 2: (17 points)

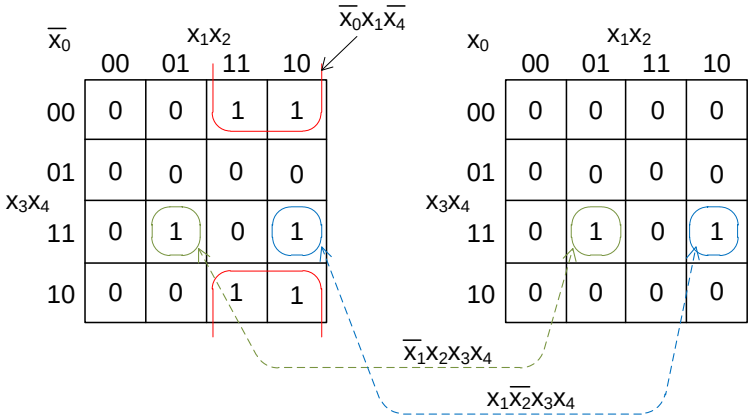
a) *Minimize the cost of function F(x0,x1,x2,x3,x4)=Σ(7,8,10,11,12,14,23,27). Measure the cost of the minimized function by counting the total number of 2-input gates of the circuit (e.g. a*b + c*d, has cost of 3). You can use either Karnaugh or Quine-McCluskey. (13 points)*

b) *Can you minimize the function even further using Boolean algebra and other logic gates? (4 points)*

Solution:

$$f(x_0, x_1, x_2, x_3, x_4) = \sum m(7, 8, 10, 11, 12\,14, 23, 27)$$

using Karnaugh



Figur E2.1 Förenkling via Karnaugh-diagram

$$f(x_0, x_1, x_2, x_3, x_4) = \overline{x_0}\cdot x_1\cdot\overline{x_4} + \overline{x_1}\cdot x_2\cdot x_3\cdot x_4 + x_1\cdot\overline{x_2}\cdot x_3\cdot x_4$$

using Quine-McCluskey:

7	0 0 1 1 1
8	0 1 0 0 0
10	0 1 0 1 0
11	0 1 0 1 1
12	0 1 1 0 0
14	0 1 1 1 0
23	1 0 1 1 1
27	1 1 0 1 1

Figur E2.2 Ursprungstabell

8	0 1 0 0 0	√
10	0 1 0 1 0	√
12	0 1 1 0 0	√
7	0 0 1 1 1	√
11	0 1 0 1 1	√
14	0 1 1 1 0	√
23	1 0 1 1 1	√
27	1 1 0 1 1	√

Figur E2.3 Sorterad tabell

8,10	0 1 0 x 0	√
8,12	0 1 x 0 0	√
10,11	0 1 0 1 x	p_1
10,14	0 1 x 1 0	√
12,14	0 1 1 x 0	√
7,23	x 0 1 1 1	p_2
11,27	x 1 0 1 1	p_3

Figur E2.4 Första hopparing

8,10, 12, 14	0 1 x x 0	p_4

Figur E2.5 Tredje sortering

Prime implicant	*Minterm* 7	8	10	11	12	14	23	27
$p_1 = 0\,1\,0\,1\,x$			√	√				
$p_2 = x\,0\,1\,1\,1$	√						√	
$p_3 = x\,1\,0\,1\,1$				√				√
$p_4 = 0\,1\,x\,x\,0$		√	√		√	√		

Figur E4.23c Initial primär täckningstabell

P2, P3, and P4 are essential prime implicants and cover the function F resulting in the following form:

$$f(x_0, x_1, x_2, x_3, x_4) = \overline{x_0} \cdot x_1 \cdot \overline{x_4} + \overline{x_1} \cdot x_2 \cdot x_3 \cdot x_4 + x_1 \cdot \overline{x_2} \cdot x_3 \cdot x_4$$

This form has a cost of 10.

b)

we can further reduce the function, using the distributive law since x3*x4 are common in 2 of the products, then what is left is x1'x2+x1x2' which is a XOR between x1 and x2:

$$f(x_0, x_1, x_2, x_3, x_4) = \overline{x_0} \cdot x_1 \cdot \overline{x_4} + \left(\overline{x_1} \cdot x_2 + x_1 \cdot \overline{x_2}\right) \cdot x_3 \cdot x_4 =$$
$$= \overline{x_0} \cdot x_1 \cdot \overline{x_4} + (x_1 \oplus x_2) \cdot x_3 \cdot x_4$$

$$f(x_4, x_3, x_2, x_1, x_0) = \overline{x_4} \cdot x_3 \cdot \overline{x_0} + (x_3 \oplus x_2) \cdot x_1 \cdot x_0$$

This form has a cost of 6.

Question 3: (20 points)

Describe a (1-bit) full adder.

a) What are its inputs and outputs? (1 point)
b) What is its truth table? (1 point)
c) What are the Boolean expressions that describe it? (1points)
d) What is its critical path? (1 point)

Ripple carry adder:

e) Describe a 4-bit ripple carry adder using the above full adder. (2 points)
f) What is the critical path of the ripple carry adder? (1 point)

How is a (1-bit) full adder modified for a carry-lookahead adder?

*g) Which are the new Boolean functions **p** and **g** (explain also how the outputs of the 1-bit full adder are calculated based on **p** and **g**)? (2points)*
*h) What do **p** and **g** mean? (2 points)*

Carry-lookahead adder:

i) Construct a 4-bit carry lookahead adder.
  - *Draw the block diagram of the carry lookahead using the above (modified) 1-bit full adder as a blackbox[1] and showing the connections between them and the additional top-level Boolean expressions required. (6points)*

g) How does the critical path of a carry-lookahead adder increase with regard to the number of bits added N? How does this compare to the critical path of a ripple carry adder? (3 point)

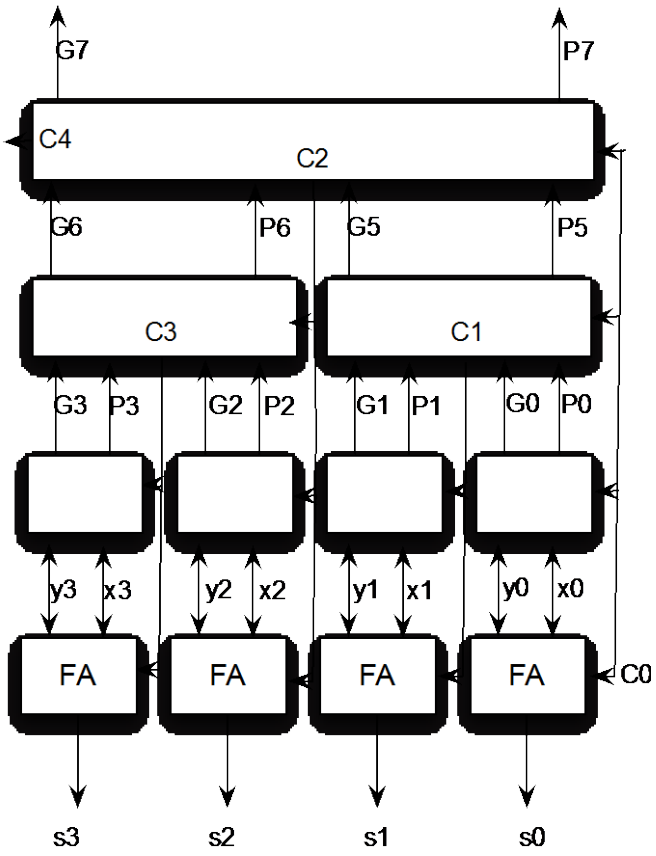
Solution:

a) Answer in lecture 2 Slide 71
b) Answer in lecture 2 Slide 71
c) Answer in lecture 2 Slide 72
d) Answer in lecture 2 Slide 72
e) Answer in lecture 2 Slide 73
f) Answer in lecture 2 Slide 75
g) Answer in lecture 2 Slide 81
h) Answer in lecture 2 Slide 81

(Above mentioned lecture slides can be found at the end of this document)

[1] No need to specify again the Boolean expressions for the 1-bit adder, only the inputs and outputs should be visible.

a)



$p_0 = x_0 \oplus y_0$

$g_0 = x_0\, y_0$

$p_1 = x_1 \oplus y_1$

$g_1 = x_1\, y_1$

$p_2 = x_2 \oplus y_2$

$g_2 = x_2\, y_2$

$p_3 = x_4 \oplus y_4$

$g_3 = x_4\, y_4$

$c_3 = g_2 + p_2 c_2$

$c_1 = g_0 + p_0 c_0$

$P_5 = p_0 p_1$

$G_5 = g_1 + p_1 g_0$

$c_2 = G_5 + P_5 c_0$

$P_6 = p_2 p_3$

$G_6 = g_3 + p_3 g_2$

$P_7 = P_5 P_6$

$G_7=G_6+P_6G_5$

$c_4=G_7+P_7c_0$

b) The critical path of the CLA increases logarithmically with the number of bits added N (T α log(N)), while for the RCA it increases linearly (T α N)

Question 4: (20 points)

Design a synchronous FSM (Finite State Machine) for an automatic machine that sells drinks and snacks. The machine stays idle and ready until somebody presses the button "Buy-drink" or the button "Buy-snack", then the machine expects to receive 2 coins to give a drink and 3 coins for a snack. If it receives the right number of coins it gives the drink or snack and waits idle again. If it does not get the right number of coins, it needs to wait until it gets the right number of coins. At any point if one presses the button "EXIT" the machine need to abandon the transaction and return to idle (any coins received up to that point -without giving the right product- need to be returned back to the client).

The FSM has the following inputs:

- ***RST**: FSM reset (if '1' go to initial state)*
- ***B-Dr**: Buy-drink (if '1' a client wants to buy a drink)*
- ***B-Sn**: Buy-snack (if '1' a client wants to buy a snack)*
- ***Coin**: coin inserted (if '1' a client inserted a coin)*
- ***Exit**: (if '1' a client wants to exit and maybe start all over again)*

The FSM will produce the following outputs:

- ***INST**: insert-coin ('1' when the customer is asked to insert another coin)*
- ***Dout**: give drink (if '1' the machine gives a drink and keeps the coins it received)*
- ***Sout**: give-snack (if '1' the machine gives a snack and keeps the coins it received)*
- ***Cout**: give-back-coins (if '1' the machine gives back any coins it has receives during this transaction)*
- ***RDY**: Ready (if '1' the machine is ready to serve the next client)*

a) Draw the state diagram of your FSM. (10 points)
   - *Values of Outputs should be written either in the states or in the transitions (depending on the FSM choice). Omitted (not mentioned) outputs are assumed to be zero.*
   - *On arrows that denote transition from one state to another indicate the input values that cause this transition. In case the combination of multiple inputs cause a transition, write the (Boolean) expression that causes the transition:*
     e.g. Input1 or (Input2 and Input3)

b) Choose a state encoding and fill the following tables defining the outputs and next-state of the FSM. (6 points)
   - *Add to the table as many rows as needed to cover all combinations of present states and inputs that cause a different state transition and/or output.*
   - *One state may need multiple rows to indicate different state transitions.*
   - *Values of Inputs in rows that don't affect the next state or the output can be marked with '-' (don't care).*

Question 4 (continued):

Present state	***Inputs***					***Next state***	***Outputs***				
	RST	*BDr*	*BSn*	*Coin*	*Exit*		*INST*	*Dout*	*Sout*	*Cout*	*RDY*

c) *Explain your choice of FSM type. What would be different if you had chosen a different FSM type? (2 points)*

d) *Explain your state encoding. What would be different if you had chosen a different encoding? Is there an encoding that would simplify the (combinational) logic of the FSM? (2 points)*

Note-1: inputs "B-Dr" and "B-Sn" cannot be "1" at the same time.

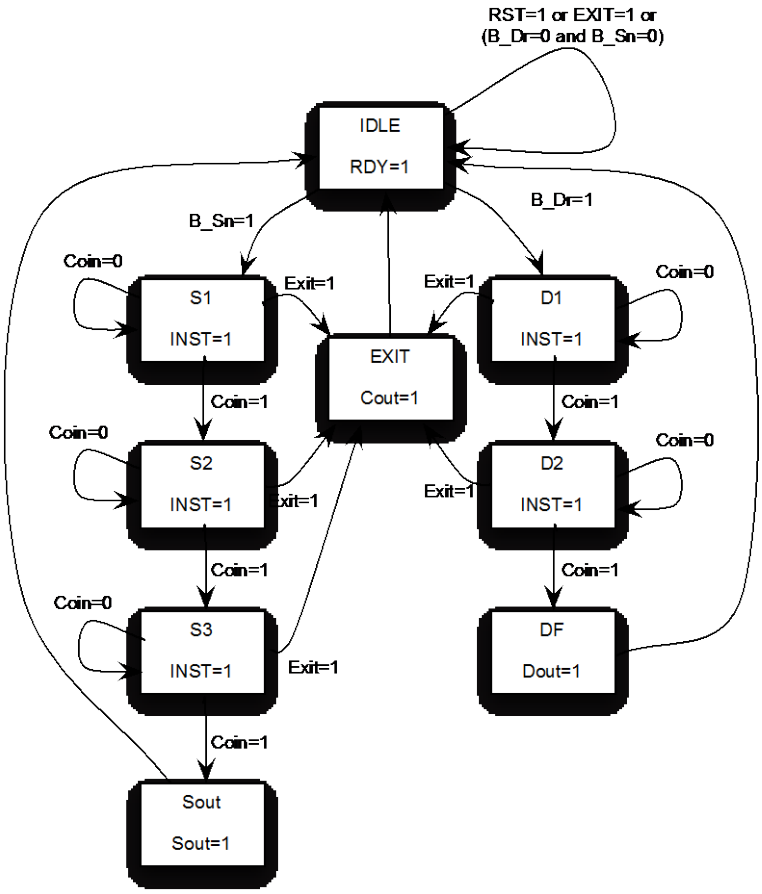
Note-2: it is not required to minimize the number of states of the FSM or find the best state encoding.

Note-3: if there is any information missing in the above description of the FSM, feel free to define it yourselves. If so, explain it in your answer.

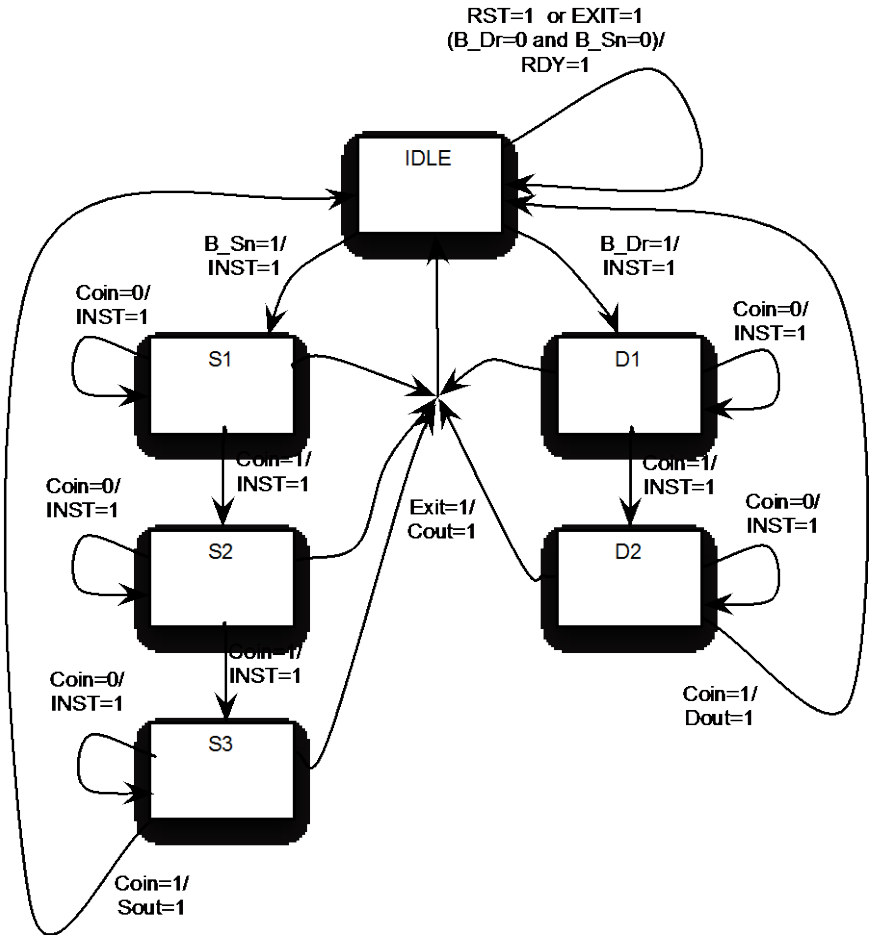
Solution:

There can be many different correct solutions.

a) Moore solution:



Mealy solution:



b)

encoding:

Moore encoding

STATE	ENCODING
IDLE	0000
D1	0001
D2	0010
DF	0011
S1	0101
S2	0110
S3	0111
SF	0100
EXIT	1000

Mealy encoding

STATE	ENCODING
IDLE	0000
D1	0001
D2	0010
S1	0101
S2	0110
S3	0111
EXIT	1000

Moore:

Present state	***Inputs***					***Next state***	***Outputs***				
	RST	*BDr*	*BSn*	*Coin*	*Exit*		*INST*	*Dout*	*Sout*	*Cout*	*RDY*
IDLE	1	-	-	-	-	IDLE	0	0	0	0	1
IDLE	-	0	0	-	-	IDLE	0	0	0	0	1
IDLE	0	-	-	-	1	IDLE	0	0	0	0	1
IDLE	0	1	0	-	0	D1	0	0	0	0	1
IDLE	0	0	1	-	0	S1	0	0	0	0	1
D1	0	-	-	0	0	D1	1	0	0	0	0
D1	0	-	-	1	0	D2	1	0	0	0	0
D1	0	-	-	-	1	EXIT	1	0	0	0	0
D2	0	-	-	0	0	D2	1	0	0	0	0
D2	0	-	-	1	0	DF	1	0	0	0	0
D2	0	-	-	-	1	EXIT	1	0	0	0	0
DF	-	-	-	-	-	IDLE	0	1	0	0	0
S1	0	-	-	0	0	S1	1	0	0	0	0
S1	0	-	-	1	0	S2	1	0	0	0	0
S1	0	-	-	-	1	EXIT	1	0	0	0	0
S2	0	-	-	0	0	S2	1	0	0	0	0
S2	0	-	-	1	0	S3	1	0	0	0	0
S2	0	-	-	-	1	EXIT	1	0	0	0	0
S3	0	-	-	0	0	S3	1	0	0	0	0
S3	0	-	-	1	0	SF	1	0	0	0	0
S3	0	-	-	-	1	EXIT	1	0	0	0	0
DF	-	-	-	-	-	IDLE	0	0	1	0	0
EXIT	-	-	-	-	-	IDLE	0	0	0	1	0

Mealy:

Present state	***Inputs***					***Next state***	***Outputs***				
	RST	*BDr*	*BSn*	*Coin*	*Exit*		*INST*	*Dout*	*Sout*	*Cout*	*RDY*
IDLE	1	-	-	-	-	IDLE	0	0	0	0	1
IDLE	-	0	0	-	-	IDLE	0	0	0	0	1
IDLE	0	-	-	-	1	IDLE	0	0	0	0	1
IDLE	0	1	0	-	0	D1	1	0	0	0	0
IDLE	0	0	1	-	0	S1	1	0	0	0	0
D1	0	-	-	0	0	D1	1	0	0	0	0
D1	0	-	-	1	0	D2	1	0	0	0	0
D1	0	-	-	-	1	IDLE	0	0	0	1	0
D2	0	-	-	0	0	D2	1	0	0	0	0
D2	0	-	-	1	0	IDLE	0	1	0	0	0
D2	0	-	-	-	1	IDLE	0	0	0	1	0
S1	0	-	-	0	0	S1	1	0	0	0	0
S1	0	-	-	1	0	S2	1	0	0	0	0
S1	0	-	-	-	1	IDLE	0	0	0	1	0
S2	0	-	-	0	0	S2	1	0	0	0	0
S2	0	-	-	1	0	S3	1	0	0	0	0
S2	0	-	-	-	1	IDLE	0	0	0	1	0
S3	0	-	-	0	0	S3	1	0	0	0	0
S3	0	-	-	1	0	IDLE	0	0	1	0	0
S3	0	-	-	-	1	IDLE	0	0	0	1	0

c) A Mealy FSM has fewer states but you may have longer critical path or spikes since the outputs depend both on the present state and the inputs. A Moore FSM would have more states but possibly shorter path and definitely stable outputs since the output depends only on the current state (which is registered)

d) In general binary and gray encoding need fewer flip-flops than one-hot (logN vs. N for N states). One hot may have simpler logic, while gray may have simpler logic than binary since only one bit changes from one state to another (this is true only when an FSM changes states "in some fixed order", like in a counter)

The encoding chosen is:

Moore encoding

STATE	ENCODING
IDLE	0000
D1	0001
D2	0010
DF	0011
S1	0101
S2	0110
S3	0111
SF	0100
EXIT	1000

Mealy encoding

STATE	ENCODING
IDLE	0000
D1	0001
D2	0010
S1	0101
S2	0110
S3	0111
EXIT	1000

The two LSbits are implementing a counter in both the Drink and the Snack case, then when counting coins a simpler logic will be "active". The 3rd bit distinguishes between drink and snack (easy to set when the "Buy-drink" or "buy-snack" button is pushed), the 4th bit (easy to set when the "EXIT" button is pushed, and simpler logic for the output).

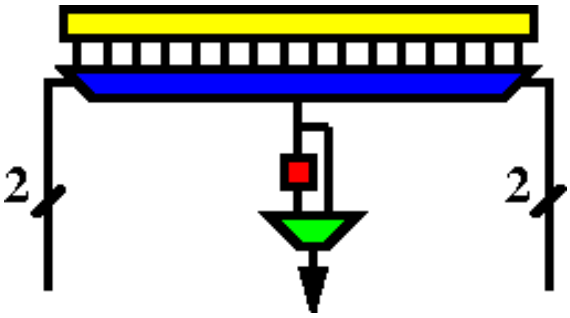
Question 5: (10 points)

a) *Draw a programmable FPGA logic cell, which has 4 inputs and 1 output. Consider that the output has the option to be either registered or not registered. (6 points)*

b) *Explain how this FPGA logic cell will be configured to implement an OR gate without a flip-flop in the output and how it will be configured to implement an AND gate with a registered output. (4 points)*

Solution:

a)



The yellow box is 16 bits of memory (or flip-flops), which together with the multiplexer implement a 4-input lookup table. The 4 bits inputs go to the select of the multiplexer. The output of the LUT goes to a flip-flip, and 2-to-1 multiplexer selects either the registered output or the one directly coming from the LUT.

b) for a 4-input OR gate without registered output the LUT would have in the position "0000" the value "0" and in the rest of the 15 positions the value "1", in addition the select of the 2-to-1 multiplexer would choose the LUT output and not the registered one.

For a 4-input AND gate with registered output the LUT would have in the position "1111" the value "1" and in the rest of the 15 positions the value "0", in addition the select of the 2-to-1 multiplexer would choose the registered output coming from the flip-flop and not the LUT output.

Question 6: (5 points)

a) *If you consider the following design and production costs for FPGA and ASIC technologies:*
   - *FPGA design cost: 1MSEK,*
   - *ASIC design cost: 100MSEK,*
   - *cost per FPGA chip produced: 10 kSEK,*
   - *cost per ASIC chip produced: 100 SEK,*

   how many chips do you need to sell in order to choose ASIC technology for your implementation, and what is the number of chips to sell that makes FPGA technology more attractive? Your only selection criterion is cost (design and production costs combined). Hint: find the breakpoint first; that is the number of chips to sell with both technologies having similar cost. (2 points)

b) *What is the difference between standard-cell and full-custom ASICs? (2 points)*

Solution:

FPGA_design_cost + x*(FPGA_production_cost) > ASIC_design_cost + x*(ASIC_production_cost) ⇔

1MSEK+x*10kSEK > 100MSEK + x*100SEK ⇔

$10^6 + x*10^4 > 10^8 + x*10^2$ ⇔

$x*10^4 - x*10^2 > 10^8 - 10^6$ ⇔

9900*x > 99,000,000 ⇔

x > 10,000

If one sells more than 10,000 chips the FPGA cost is higher and therefore ASICs are better, for less than 10,000 chips FPGAs are better.

b)

Standard cell ASICs use prefixed cells implementing a single gate or a flip-flip (or bigger blocks, then called block-based ASICs). Tools are then used to place and wire these standard cells. Cells are described in libraries of some vendor that the hardware designer uses.

In full-custom ASICs, every gate and flip-flop is designed in detail down to the transistor level and even lower. Full custom ASICs get better performance, lower power consumption, and better area efficiency compared to standard cell ASICs, but they are more difficult and expensive to design. Full-custom ASICs are meant for critical blocks (e.g. blocks that cause performance bottlenecks)

Question 7: (5 points)

a) *What is the difference between a volatile and a non-volatile memory? (3points)*
b) *Give two examples for each category. (2 points)*

Solution:

a) Volatile memory is the one that loses its state when the power goes off, while a non-volatile is the one that keeps its state even when the power goes off.

b)

Volatile memory: DRAM, SRAM

Non-Volatile memory: Mask ROM, EPROM, EEPROM, FLASH

Question 8: (5 points)

*a) Draw the block diagram of a memory with 4096 entries/rows and 32-bit elements (4k*32-bits), which is composed of other memory blocks, which are 1024 entries and have 8 bit elements (1k*8bits). (3 points)*

*b) How many 1k*8-bit memory elements do you need? (1 point)*

*c) How many address bits are needed for the 4k*32-bits? (1point)*

Solution:

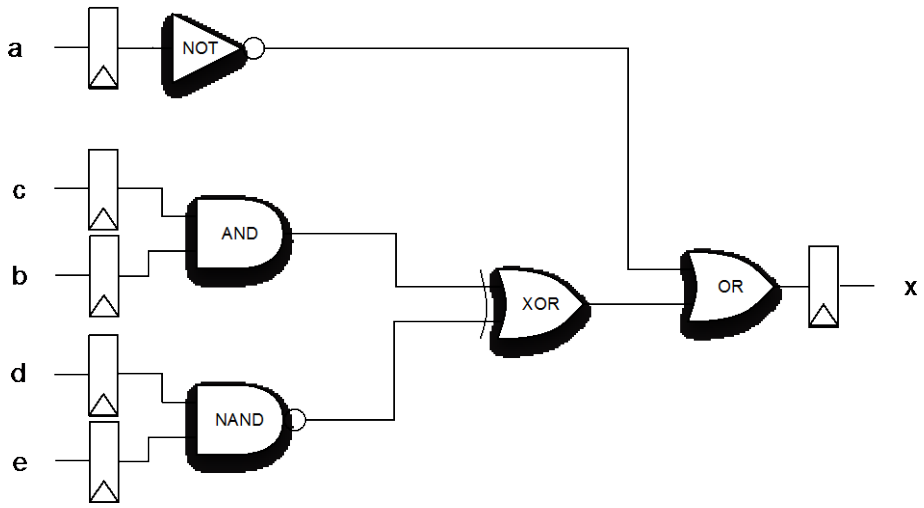
a) Similar to figure in lecture 11 slide 36 (only with 4 columns of memory blocks instead of 2 columns). Each block is a 1K x 8 bits, which needs10 bits to be addressed (A9-A0) the decoder needs 2 more additional bits (A11-A10).
(Above mentioned lecture slides can be found at the end of this document)

b) 16

c) 12

Question 9: (5 points)
Calculate the maximum delay of the circuit (critical path delay), considering the following:

- *a NOT gate has a delay of 1 ns*
- *an AND gate has a delay of 3 ns*
- *a NAND gate has a delay of 2 ns*
- *a XOR gate has a delay of 6 ns*
- *an OR gate has a delay of 3 ns*
- *Propagation time (clock to output) for a flip-flop 1ns*
- *Setup time for a flip-flop 2ns*
- *Wires have zero delay*
- *All inputs (a, b, c, d, e) have zero delay*



Solution:

The Critical path is from the output of the flip-flops connected to "c" or "d" to the input of the flip-flop at the output.

Max Delay = (Propagation delay of the flip-flop) + (Delay of the AND gate) + (Delay of the XOR gate) + (Delay of the OR gate) + (Setup-time of the flip-flop) ⇔ Max Delay = 1ns + 3ns + 6ns + 3ns +2ns = 15ns

Question 10: (5 points)

a) *How many test-inputs would a 32-bit adder need to be exhaustively tested? (1 point)*

b) *Why exhaustive testing of all combinations of inputs is not a realistic solution for large circuits? (1 point)*

c) *What is the definition of yield? (1 point)*

d) *Which are the parameters that affect manufacturing yield and how they affect it (provide the formula that connects yield with its parameters) (2 points)*

Solution:

a) A 32-bit adder has two 32-bit inputs (plus the carry). To exhaustively test it, one needs to test all possible combinations of the inputs. That is $2^{32+32}=2^{64}$ (or $2^{32+32+1}=2^{65}$)

b) Applying all possible inputs in a large circuit is unrealistic (both in terms of time and cost). Even in the above adder, $2^{64} = 10^{19}$ combination of inputs is a huge number. Taking 1 ns for testing each one of these inputs would require a huge amount of time (585 years)

c) Yield = (number of good chips)/(total number of chips produced)

d) Yield = $(1+ A*d/\alpha)^{-\alpha}$

where: A is the chip area

d is the defect density

α is the fault clustering parameter

Lecture Slides

Question 1

Algebraic Forms of Logic Functions

- A ***minterm*** is a product term in which all the variables appear exactly once either complemented or uncomplemented.
- ***Canonical Sum of Products* (*canonical SOP*)**:
  - Represented as a sum of minterms only.
  - ***Example***: $f_1(A,B,C) = A'BC' + ABC' + A'BC + ABC$
  - Minterms of three variables:

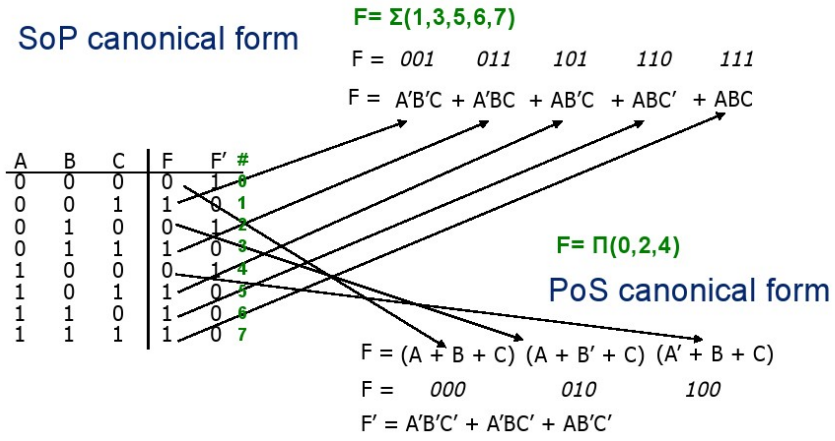
Minterm	Minterm Code	Minterm Number
$A'B'C'$	000	m_0
$A'B'C$	001	m_1
$A'BC'$	010	m_2
$A'BC$	011	m_3
$AB'C'$	100	m_4
$AB'C$	101	m_5
ABC'	110	m_6
ABC	111	m_7

Algebraic Forms of Logic Functions

- A ***maxterm*** is a sum term in which all the variables appear exactly once either complemented or uncomplemented.
- ***Canonical Product of Sums* (*canonical POS*)**:
  - Represented as a product of maxterms only.
  - ***Example***: $f_2(A,B,C) = (A+B+C)(A+B+C')(A'+B+C)(A'+B+C')$
- Maxterms of three variables:

Maxterm	Maxterm Code	Maxterm Number
$A+B+C$	000	M_0
$A+B+C'$	001	M_1
$A+B'+C$	010	M_2
$A+B'+C'$	011	M_3
$A'+B+C$	100	M_4
$A'+B+C'$	101	M_5
$A'+B'+C$	110	M_6
$A'+B'+C'$	111	M_7

Canonical Forms



canonical form ≠ minimal form

Conversion between canonical forms

Relation between maxterms and minterms of a function

$$F(x,y,z) = \Sigma(1,3,5,6,7) = \Pi(0,2,4)$$

… and for the inverted function:

$$F'(x,y,z) = \Pi(1,3,5,6,7) = \Sigma(0,2,4)$$

Shannon's expansion theorem

- ***Shannon's expansion theorem***

  (a). $f(x_1, x_2, \ldots, x_n) = x_1 * f(1, x_2, \ldots, x_n) + x_1' * f(0, x_2, \ldots, x_n)$

  (b). $f(x_1, x_2, \ldots, x_n) = [x_1 + f(0, x_2, \ldots, x_n)]\ [x_1' + f(1, x_2, \ldots, x_n)]$

Example:

$$F(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3$$
$$= x_1'F(0, x_2, x_3) + x_1F(1, x_2, x_3)$$
$$= x_1'\,(0x_2 + 0x_3 + x_2x_3) + x_1(1x_2 + 1x_3 + x_2x_3)$$
$$= x_1'\,(x_2x_3) + x_1(x_2 + x_3 + x_2x_3)$$
$$= x_1'\,(x_2x_3) + x_1(x_2 + x_3(1 + x_2))$$
$$= x_1'\,(x_2x_3) + x_1(x_2 + x_3)$$

Derivation of Canonical Forms

- Derive canonical PoS or SoP using Boolean algebra.
- ***Shannon's expansion theorem***

  (a). $f(x_1, x_2, \ldots, x_n) = x_1 * f(1, x_2, \ldots, x_n) + x_1' * f(0, x_2, \ldots, x_n)$

  (b). $f(x_1, x_2, \ldots, x_n) = [x_1 + f(0, x_2, \ldots, x_n)]\ [x_1' + f(1, x_2, \ldots, x_n)]$

Convert a function to SoP

1. Apply Shannon Theorem until you get sum of minterms
2. Get the function to a SoP form, as follows:
   - if a product of literals is a minterm, keep it
   - for every variable x_i that doesn't exist in a product we add $(x_i + x_i')$, e.g.:
     - $x_1 * x_2 = x_1 * x_2 * (x_3 + x_3') = x_1 * x_2 * x_3 + x_1 * x_2 * x_3'$
   - Continue and delete redundant products

Derivation of Canonical Forms

- Derive canonical PoS or SoP using Boolean algebra.
- ***Shannon's expansion theorem***

  (a). $f(x_1, x_2, \ldots, x_n) = x_1 * f(1, x_2, \ldots, x_n) + x_1' * f(0, x_2, \ldots, x_n)$

  (b). $f(x_1, x_2, \ldots, x_n) = [x_1 + f(0, x_2, \ldots, x_n)]\,[x_1' + f(1, x_2, \ldots, x_n)]$

- ***Example***: $f(A,B,C) = AB + AC' + A'C$
  - $f(A,B,C) = AB + AC' + A'C = A\,f(1,B,C) + A'\,f(0,B,C)$
    $= A(1\cdot B + 1\cdot C' + 1'\cdot C) + A'(0\cdot B + 0\cdot C' + 0'\cdot C) = A(B + C') + A'C$
  - $f(A,B,C) = A(B + C') + A'C = B[A(1+C') + A'C] + B'[A(0 + C') + A'C]$
    $= B[A + A'C] + B'[AC' + A'C] = AB + A'BC + AB'C' + A'B'C$
  - $f(A,B,C) = AB + A'BC + AB'C' + A'B'C$
    $= C[AB + A'B\cdot 1 + AB'\cdot 1' + A'B'\cdot 1] + C'[AB + A'B\cdot 0 + AB'\cdot 0' + A'B'\cdot 0]$
    $= ABC + A'BC + A'B'C + ABC' + AB'C'$

Derivation of Canonical Forms

- Derive canonical PoS or SoP using Boolean algebra.
- ***Shannon's expansion theorem***

  (a). $f(x_1, x_2, \ldots, x_n) = x_1 * f(1, x_2, \ldots, x_n) + x_1' * f(0, x_2, \ldots, x_n)$

  (b). $f(x_1, x_2, \ldots, x_n) = [x_1 + f(0, x_2, \ldots, x_n)]\,[x_1' + f(1, x_2, \ldots, x_n)]$

- ***Example***: $f(A,B,C) = AB + AC' + A'C$

  $f(A,B,C) = AB + AC' + A'C =$

  $= AB(C+C') + AC'(B+B') + A'C(B+B')$

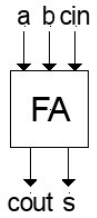
  $= ABC + ABC' + ABC' + AB'C' + A'BC + A'B'C$

  $= ABC + ABC' + AB'C' + A'BC + A'B'C$

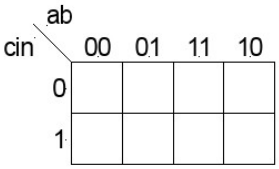
Question 3

Adders

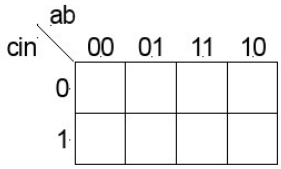
Full-adder cell (FA) revisited:



a b cin	cout	s
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	1



cout



s

Boolean functions for the full adder

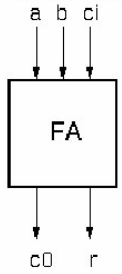
R = AB'Cin' + A'BCin' + ABCin + A'B'Cin

= Cin'(AB'+A'B) + Cin(AB+A'B')

=Cin'(AB'+A'B) + Cin (AB'+A'B)'

= Cin'(A xor B) + Cin (A xor B)'

=Cin xor (A xor B)

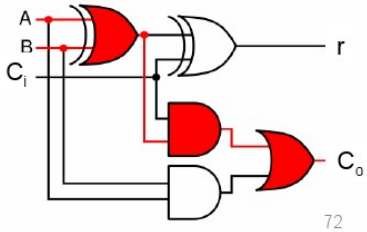


Critical path:
The longest path from an input to an output

Cout=AB'Cin + A'Bcin + AB =

=Cin(AB'+A'B) + AB

=Cin(A xor B) + AB

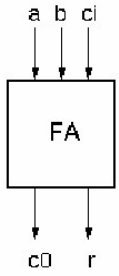


Ripple-Carry Adder

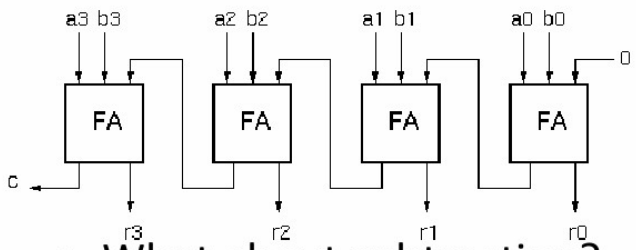
- Each cell:

  $r_i = a_i \text{ XOR } b_i \text{ XOR } c_{in}$

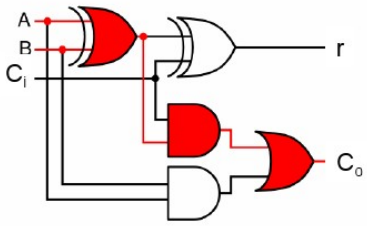
  $c_{out} = c_{in}(a_i \text{ XOR } b_i) + a_i b_i$



- 4-bit adder:



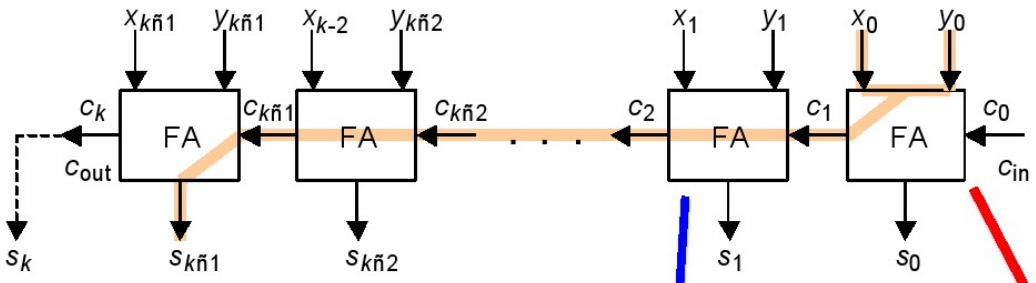
"Full adder cell"



- What about subtraction?

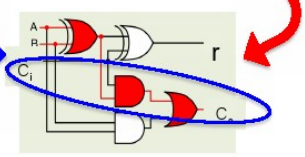
Critical Path Through a Ripple-Carry Adder

$$T_{\text{ripple-add}} = T_{FA}(x,y \rightarrow c_{out}) + (k-2) \times T_{FA}(c_{in} \rightarrow cout) + T_{FA}(c_{in} \rightarrow s)$$



How do we make it faster, perhaps with more cost?

Mechanical adder: https://youtu.be/GcDshWmhF4A



Carry Look-ahead Adders

- In general, for n-bit addition best we can achieve is
  delay α log(n)
- How do we arrange this? (think trees)
- First, reformulate basic adder stage:

a b c_i	c_{i+1}	s
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	1

p

g

carry "propagate"
$p_i = a_i \oplus b_i$
Exactly one of the 2 inputs is 1, then propagate the carry you received from the previous stage

carry "generate"
$g_i = a_i\, b_i$
If both inputs are 1, then no matter what carry-in you received, generate a carry

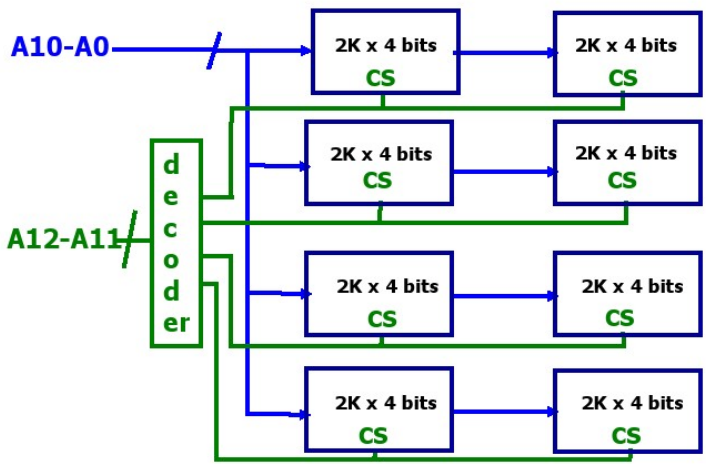
$c_{i+1} = g_i + p_i c_i$
Carry-out is one if you generate a carry or you have a carry-in to propagate

$s_i = p_i \oplus c_i$
Result is 1 either when exactly 1 of the Inputs (a, b) is 1 (carry-propagate) or have a carry-in

Question 8

Using Memory Building Blocks

– Building an 8K byte memory using chips that are 2K by 4 bits.



- **CS = chip select: when set, it enables the addressing, reading and writing of that chip.**

This is an 8KB byte addressable memory