DIGITAL TECHNICS

Dr. Bálint Pődör

Óbuda University,
Microelectronics and Technology Institute

6. LECTURE (ANALYSIS AND SYNTHESIS OF SYNCHRONOUS SEQUENTIAL CIRCUITS)

1st (Autumn) term 2018/2019

1

6. LECTURE

Analysis and synthesis of

synchronous sequential circuits:

Design examples and case studies

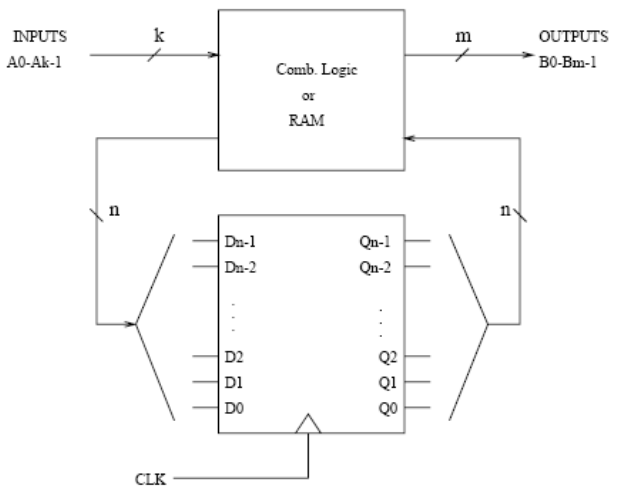
2

SYNTHESIS: GENERAL CONCEPTS

Synchronous sequential circuits synthesis procedure

Word description of problem (hardest; art, not science)
Derive state diagram and state table
Minimize (moderately hard)
Assign states (very hard)
Produce state and output transition tables
Determine what FFs to use and find their excitation maps
Derive output equations/K-maps
Obtain the logic diagram

STATE MACHINE



General scheme of a state machine.

4

STATE MACHINE SYNTHESIS

The strategy for applying this scheme to a given problem consists of the following:

1. Identify the number of required states, m. The number of bits of memory (e.g. number of flip-flops) required to specify the m states is at minimum n = $\log_2(m)$.

2. Make a state diagram which shows all states, inputs, and outputs.

3. Make a truth table for the logic section. The table will have n + k inputs and n + m outputs.

4. Implement the truth table using combinational logic techniques.

5

SYNTHESIS OF SEQUENTIAL CIRCUIT: A CASE STUDY

- Synthetize a network which determines the parity of a four bit serial code word.

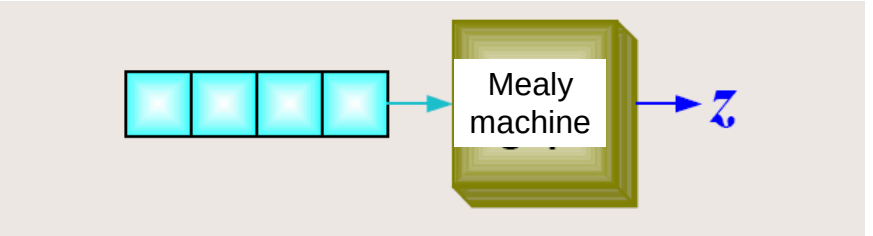
- Should indicate the parity of the incoming code word after receiving the 4-th bit as
  - 1 if the parity is odd,
  - 0 if the parity is even.

- The output is irrelevant (don't care) during the first three cycle of the period.

This design project being rather elementary will be skipped now, however it can be read in the Moodle files.
Instead of it a more complex problem, a traffic control system will be presented and discussed.

6

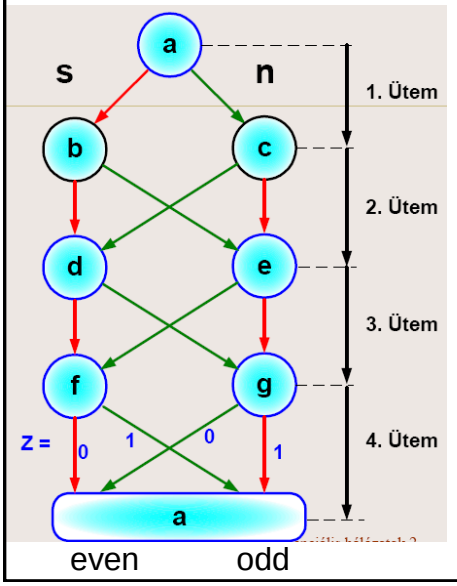
4-BIT PARITY INDICATOR



- When checking the parity the order of the bits is irrelevant.
- Construct the state transition diagram of the Mealy-machine.

7

4-BIT PARITY INDICATOR: STATE TRANSITION DIAGRAM



left (s)- even
right (n) - odd

red - incoming bit 0
green - incoming bit 1

The output Z is defined only in the fourth cycle, otherwise it is ”don’t care”.

For the code word 1011
a → c → d → f → a

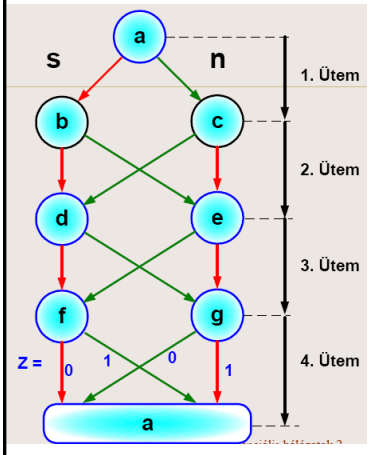
8

CHARACTERISTICS

- Because there are two input conditions, two connecting lines emanate from each node.
- The network returns to its initial state after the fourth cycle.
- The operation of the network is cyclic, the length of the period is four cycles.

9

STATE TRANSITION TABLE AND DIAGRAM



q^n	q^{n+1}		Z	
	X=0	X=1	X=0	X=1
a	b	c	x	x
b	d	e	x	x
c	e	d	x	x
d	f	g	x	x
e	g	f	x	x
f	a	a	0	1
g	a	a	1	0

even odd

10

THE NUMBER OF INTERNAL STATES AND THEIR ENCODING

- Total number of internal states: seven
- Three flip-flops (Q_1, Q_2, Q_3) are necessary and enough for the encoding.
- The actual state encoding greatly influences the complexity and structure of the network.
- Here we use the final (optimal) state encoding.

11

STATE ENCODING

q^n	Q_1	Q_2	Q_3
a	0	0	x
b	0	1	0
c	0	1	1
d	1	1	0
e	1	1	1
f	1	0	0
g	1	0	1

- In the firs row, we make use of the redundancy.
- To the states in the same level of the state transition diagram, the same Q1 and Q2 codes are ascribed.
- Q1, Q2: cycle counters.
- Q3: indicates whether the system is in the even or on the odd branch of the state transition diagram.

12

STATE FUNCTIONS AND THE OUTPUT FUNCTION

i	n-edik ütem					(n+1)-edik ütem				Z
	X	Q_1	Q_2	Q_3	q^n	q^{n+1}	Q_1	Q_2	Q_3	
0	0	0	0	0	**a**	**b**	0	1	0	x
1		0	0	1	**a**	**b**	0	1	0	x
2		0	1	0	**b**	**d**	1	1	0	x
3		0	1	1	**c**	**e**	1	1	1	x
4		1	0	0	**f**	**a**	0	0	x	0
5		1	0	1	**g**	**a**	0	0	x	1
6		1	1	0	**d**	**f**	1	0	0	x
7		1	1	1	**e**	**g**	1	0	1	x

STATE FUNCTIONS AND THE OUTPUT FUNCTION

i	n-edik ütem					(n+1)-edik ütem				Z
	X	Q_1	Q_2	Q_3	q^n	q^{n+1}	Q_1	Q_2	Q_3	
8	1	0	0	0	**a**	**c**	0	1	1	x
9		0	0	1	**a**	**c**	0	1	1	x
10		0	1	0	**b**	**e**	1	1	1	x
11		0	1	1	**c**	**d**	1	1	0	x
12		1	0	0	**f**	**a**	0	0	x	1
13		1	0	1	**g**	**a**	0	0	x	0
14		1	1	0	**d**	**g**	1	0	1	x
15		1	1	1	**e**	**f**	1	0	0	x

STATE FUNCTIONS AND THE OUTPUT FUNCTION

$Q_1^{n+1} = \Sigma^4(2,3,6,7,10,11,14,15);$

$Q_2^{n+1} = \Sigma^4(0\text{-}3,8\text{-}12);$

$Q_3^{n+1} = \Sigma^4(3,7,8,9,10);\ x:(4,5,12,13);$

$Z^n = \Sigma^4(5,12);\ x:(0\text{-}3,6\text{-}11,14,15);$

The weighing of the variables:

X^n	8
Q_1^n	4
Q_2^n	2
Q_3^n	1

15

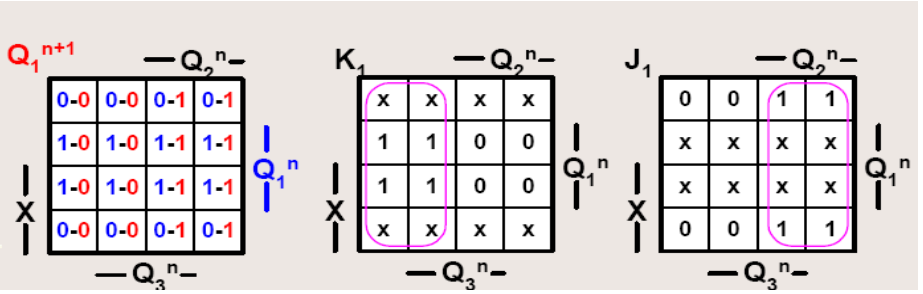
EXCITATION TABLE OF THE JK FLIP-FLOP

The logic synthesis is based on the excitation table of the flip-flop chosen for the implementation.

Q^n	→	Q^{n+1}	J	K
0	→	0	0	X
0	→	1	1	X
1	→	0	X	1
1	→	1	X	0

16

CONTROL OF FLIP-FLOP Q_1

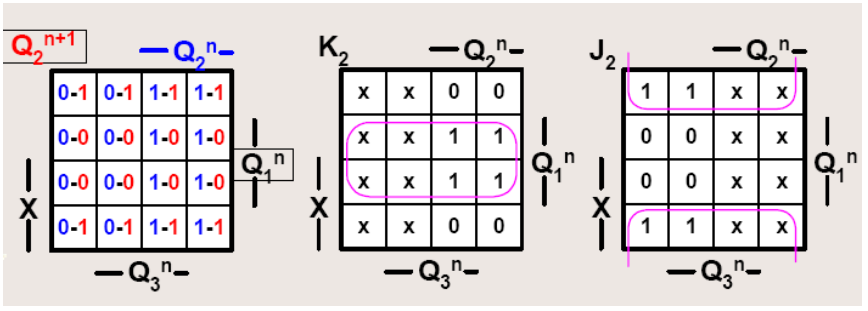


$K_1 = \overline{Q}_2$ $J_1 = Q_2$

Note the role of the don't care terms in the minimization.

17

CONTROL OF FLIP-FLOP Q_2

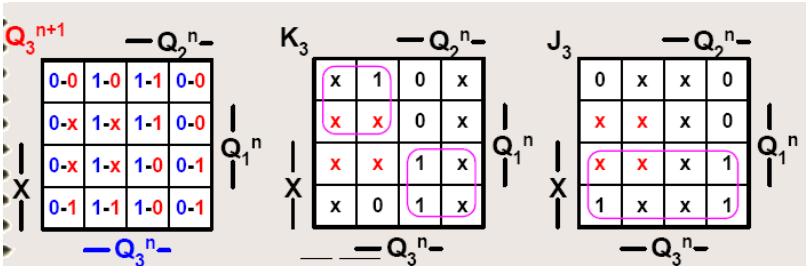


$K_2 = Q_1$ $J_2 = \overline{Q}_1$

Due to the proper state-encoding, the X input variable is not present in the control equations of Q_1 és Q_2 .These two flip-flops act as cycle counter.

18

CONTROL OF FLIP-FLOP Q_3

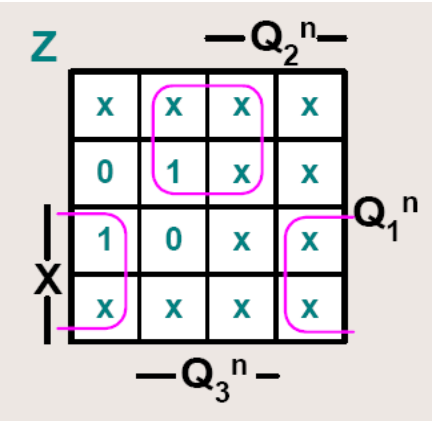


$$K_3 = \overline{X}\,\overline{Q}_2 + X\,Q_2 = X \oplus \overline{Q}_2 \qquad J_3 = X$$

The X input is among the variables controlling the flip-flop. The state of Q_3 will represent the actual parity. Q_3 will "remember" then parity of the input sequence.

19

THE OUTPUT FUNCTION Z



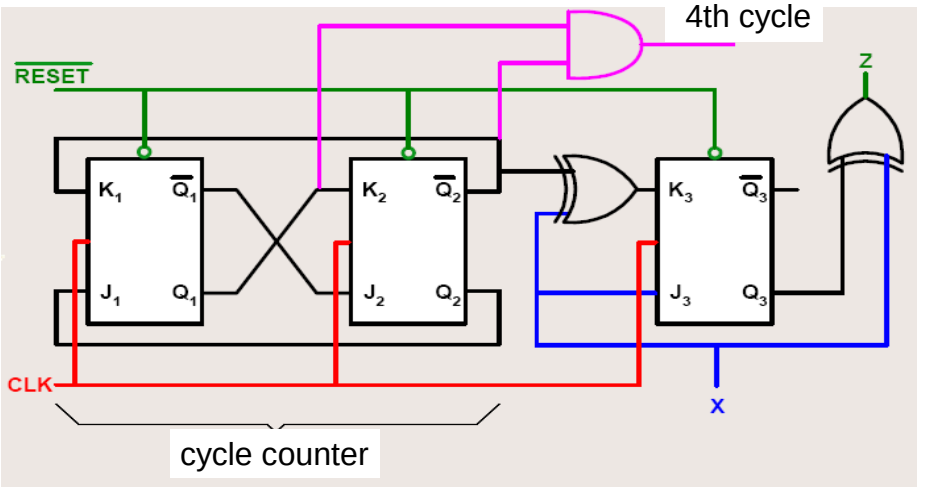
Note the chessboard pattern!
This implies XOR function:

$$Z = \overline{X}\,Q_3 + X\,\overline{Q}_3 =$$

$$= X \oplus Q_3$$

20

THE LOGIC DIAGRAM OF THE PARITY CHECK CIRCUIT



21

IMPLEMENTATION ALTERNATIVE USING D FLIP-FLOPS

$D_1 = Q_2$ $\quad\quad$ $D_2 = \overline{Q}_1$

$D_3 = X\,\overline{Q}_1 + X\,\overline{Q}_3 + \overline{X}\,Q_1\,Q_2$

Due to the ”clever” sate encoding, the control of the two flip-flops acting as the cycle counter corresponds to the usual one. However the control network of the third flip-flop is somewhat more complex than in the former implementation.

22

IMPLEMENTATION USING T FLIP-FLOPS

The feedback network is somewhat more complicated than in the case of D flip-flops.
Main reason: Counting in Gray code with T flip-flops needs more gates for the feedback.

Perhaps somebody might check a design with T flip-flops, the cycle counter operating in the simple binary code…

23

8-BIT PARITY INDICATOR

Generalization to 8 bit s is straightforward.

Design procedure and the state transition diagram is similar.

There will be 15 states, therefore four flip-flops are necessary. If the encoding is the same as previously, then three FFs form the cycle counter, and the fourth will store the information concerning the parity.

24

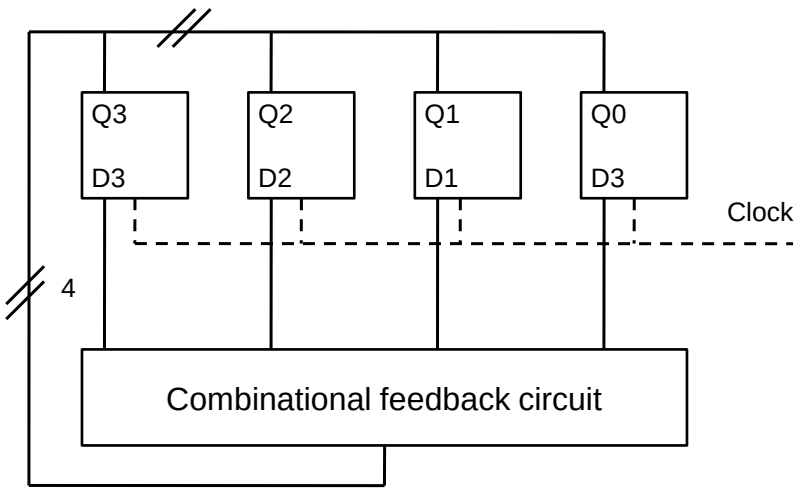
SYNCHRONOUS COUNTER DESIGN EXAMPLE AND CASE STUDY

Consider the synthesis of a 4-bit up-counter in Gray-code using D flip-flops.

A Gray-code counter using D flip-flops can be designed by finding the appropriate function of each D terminal. Given a present state of the counter, the D terminal of each flip-flop should be made equal to the value of the same bit position of the next-number in the Gray code.

25

4-BIT GRAY CODE COUNTER: CONCEPTUAL DIAGRAM



26

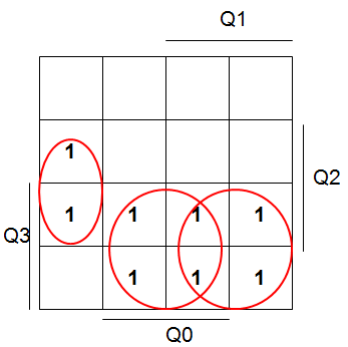
STATE TRANSITION TABLE

Minterm index	$Q3^n$	$Q2^n$	$Q1^n$	$Q0^n$	$Q3^{n+1}$ D3	$Q2^{n+1}$ D2	$Q1^{n+1}$ D1	$Q0^{n+1}$ D0
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	1
3	0	0	1	1	0	0	1	0
2	0	0	1	0	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	0	1	0	1
5	0	1	0	1	0	1	0	0
4	0	1	0	0	1	1	0	0
12	1	1	0	0	1	1	0	1
13	1	1	0	1	1	1	1	1
15	1	1	1	1	1	1	1	0
14	1	1	1	0	1	0	1	0
10	1	1	1	0	1	0	1	1
11	1	1	1	1	1	0	0	1
9	1	1	0	1	1	0	0	0
8	1	0	0	0	0	0	0	0

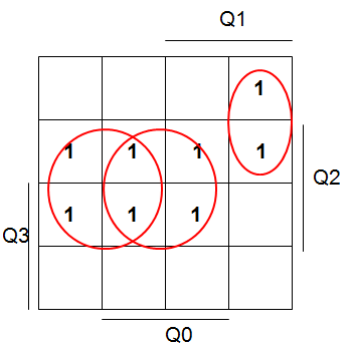
27

KARNAUGH MAPPING

D3



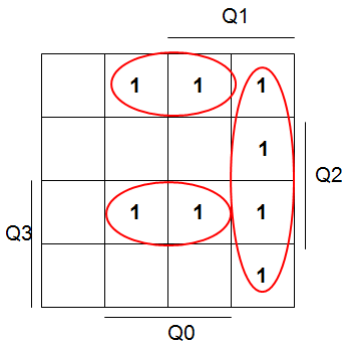
D2



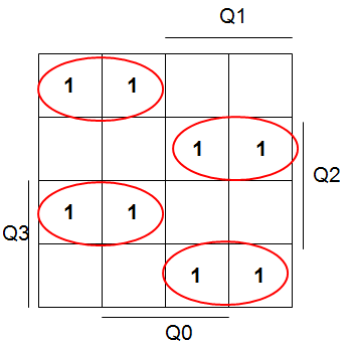
28

KARNAUGH MAPPING

D1



D0



29

FLIP-FLOP CONTROL EQUATIONS

$Q3^{n+1} = D3 = Q3Q0 + Q3Q1 + Q2\overline{Q1}\,\overline{Q0}$

$Q2^{n+1} = D2 = Q2\overline{Q1} + Q2Q0 + \overline{Q3}Q1\overline{Q0}$

$Q1^{n+1} = D1 = Q1\overline{Q0} + \overline{Q3}\,\overline{Q2}Q0 + Q3Q2Q0$

$Q0^{n+1} = D0 = \overline{Q3}Q2\overline{Q1} + Q3Q2\overline{Q1} + \overline{Q3}Q2Q1 + Q3\overline{Q2}Q1$

Implementation options: two-level AND-OR (13 AND, 4 OR) in modular logic or PLA, or two-level NAND-NAND in modular logic, or PROM.

30

FLIP-FLOP CONTROL EQUATIONS

Design alternative: D1 and D0 controls can be implemented in AND-OR-XOR LOGIC too.

$Q1^{n+1} = D1 = Q1\overline{Q0} + \overline{Q3}\,\overline{Q2}Q0 + Q3Q2Q0 =$

$Q1\overline{Q0} + (Q3 \oplus \overline{Q2})Q0$

$Q0^{n+1} = D0 = \overline{Q3}Q2\overline{Q1} + Q3Q2\overline{Q1} + \overline{Q3}Q2Q1 + Q3\overline{Q2}Q1 =$

$Q3 \oplus \overline{Q2} \oplus Q1$

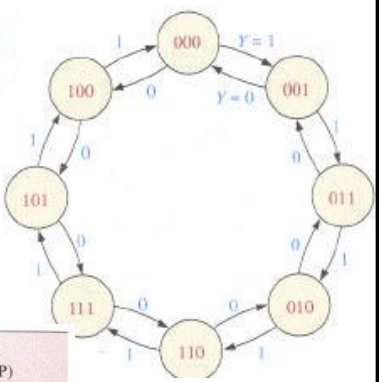
Gives a three-level combinational network (7 AND, 3 OR, 2 XOR, and 1 INV).

UP/DOWN 3-BIT GRAY CODE COUNTER

State transition diagram

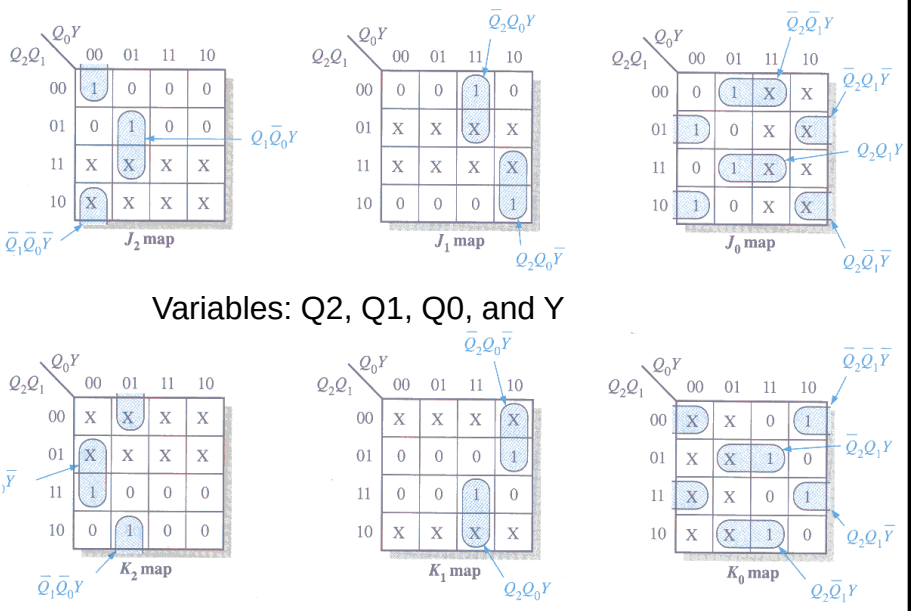
Next-state table

$\overline{\text{UP/DOWN}}$ control input: Y



Present State			Next State, Y = 0 (DOWN)			Next State, Y = 1 (UP)		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	1
0	1	1	0	0	1	0	1	0
0	1	0	0	1	1	1	1	0
1	1	0	0	1	0	1	1	1
1	1	1	1	1	0	1	0	1
1	0	1	1	1	1	1	0	0
1	0	0	1	0	1	0	0	0

UP/DOWN 3-BIT GRAY CODE COUNTER



Variables: Q2, Q1, Q0, and Y

UP/DOWN 3-BIT GRAY CODE COUNTER

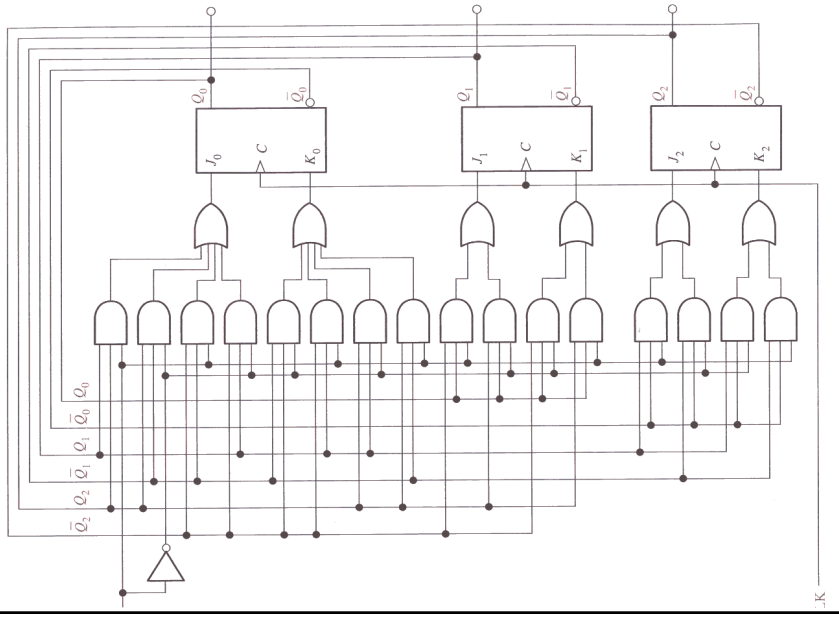
$$J_0 = Q_2Q_1Y + Q_2\overline{Q}_1\overline{Y} + \overline{Q}_2Q_1\overline{Y} + \overline{Q}_2\overline{Q}_1Y$$
$$J_1 = \overline{Q}_2Q_0Y + Q_2Q_0\overline{Y}$$
$$J_2 = Q_1\overline{Q}_0Y + \overline{Q}_1\overline{Q}_0\overline{Y}$$

$$K_0 = \overline{Q}_2\overline{Q}_1\overline{Y} + \overline{Q}_2Q_1Y + Q_2Q_1\overline{Y} + Q_2\overline{Q}_1Y$$
$$K_1 = \overline{Q}_2Q_0\overline{Y} + Q_2Q_0Y$$
$$K_2 = Q_1\overline{Q}_0\overline{Y} + \overline{Q}_1\overline{Q}_0Y$$

Logic expressions for flip-flop control

34

UP/DOWN 3-BIT GRAY CODE COUNTER



35

4-BIT BI-DIRECTIONAL GRAY CODE COUNTER

Features of design provided by one of the students of my previous course.

Compared designs using D or T flip-flops.

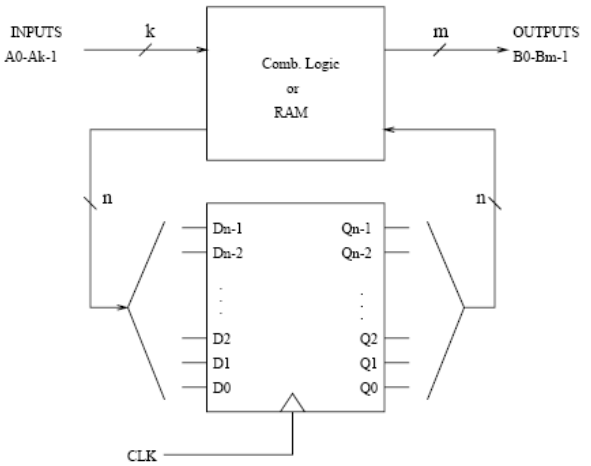
Using T flip-flops, some several common terms could be realized by XOR gate or XOR gate and inverter, leading to further simplification of the feedback circuit.

Complexity: 16 NAND gates (2,3 or 4 inputs), 2 XOR gates and 2 inverters.

Estimated the maximum clock frequency of the counter when using high speed CMOS logic components.

36

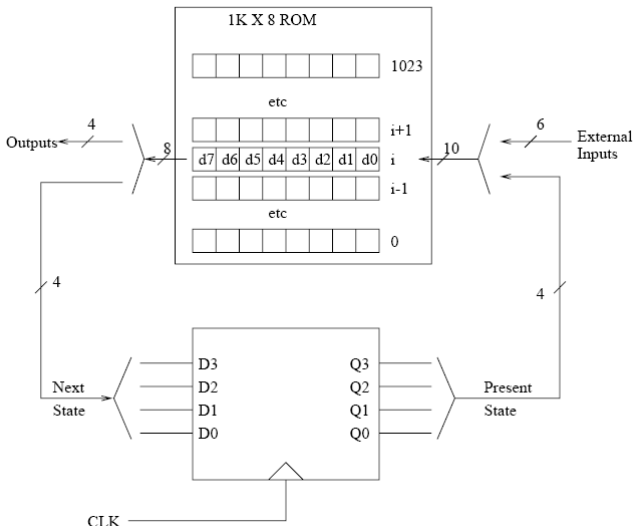
STATE MACHINE WITH MEMORY



The standard state machine configuration

37

STATE MACHINE WITH MEMORY



Toward a microprocessor: Replacing the combinational logic with a memory.

38

STATE MACHINE WITH MEMORY

To start with, let's assume a state machine with no external inputs or outputs. Then the state machine's *present state* (PS) becomes an address which is input to the ROM. The data word stored in the ROM at that address then corresponds to the *next sta*te (NS). This correspondence had been initially programmed into the ROM, just as the specic combinational logic in an old state machine had to be pre-determined. So if the PS as defined at the data register are, for example, 1001, then the ROM data word at address 1001 will be the NS which is then passed back to the register. When there are also external inputs, as there will be for most anything of interest, these are combined with the PS bits to form a longer address for the ROM. Similarly, any external outputs are combined with the NS bits in the data word.

39

EXAMPLE: DIVIDE BY 2 OR 3 COUNTER

Design a counter which either divides by 2 or by 3, depending upon the value of an external input bit P.
3 states are required, use 2 bits, describe four states:

A	00
B	01
C	10
D	11

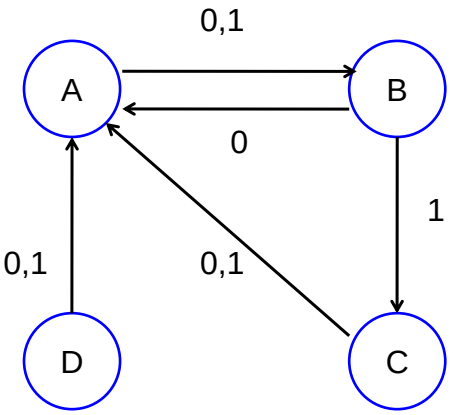
P = 0	divide by 2
P = 1	divide by 3

	Present State		Next State		
p	Q_1Q_0			D_1D_0	r
0	00	A	B	01	0
0	01	B	A	00	1
0	10	C	A	00	0
0	11	D	A	00	0
1	00	A	B	01	0
1	01	B	C	10	1
1	10	C	A	00	0
1	11	D	A	00	0

Output R =1 if present state is B, otherwise R = 0.
State D is normally unused.

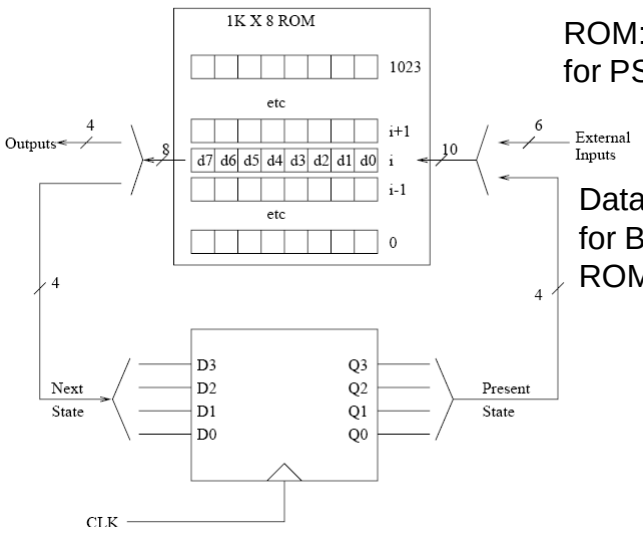
40

DIVIDE BY 2 OR 3 COUNTER: STATE TRANSITION DIAGRAM



41

EXAMPLE: DIVIDE BY 2 OR 3 COUNTER



ROM: 3 address bits (2 for PS, 1 for input P).

Data word length 3 bit (2 for BS 1 for output R). ROM size 8x3=24 bits.

42

EXAMPLE: DIVIDE BY 2 OR 3 COUNTER

The programming of the ROM is straightforward and can be read directly from the truth table. Addresses are encoded as PQ_1Q_0 and the data words as D_1D_0R. For example take the 5th row of the truth table. The address would be 100 and the data word at this address would be 010. The remaining bits of the ROM would be programmed in the same way. So one would initially ”burn in" these bit patterns into the ROM and put it into the circuit.

	Present State		Next State		
p	Q_1Q_0			D_1D_0	r
0	00	A	B	01	0
0	01	B	A	00	1
0	10	C	A	00	0
0	11	D	A	00	0
1	00	A	B	01	0
1	01	B	C	10	1
1	10	C	A	00	0
1	11	D	A	00	0

43

GENERALIZATION TO MICROPROCESSORS

A state machine with zero input bits can perform a counter-like function, but not more: its next state is limited to be a function only of the present state. A single input bit can be used to ”program" the state machine to behave in one of two possible ways for each present state, as was illustrated with the examples.

E.g. in an up/down counter.

On the other hand, with n inputs, the machine can perform 2^n different operations. So, e.g. with n = 8 the machine can perform one of 256 different operations on each clock cycle. This allows for tremendous potential and flexibility.

44

GENERALIZATION TO MICROPROCESSORS

The input bits can themselves be sequenced and stored externally in a specific sequence which is then applied step by step to the state machine inputs on successive clock cycles. Such a stored sequence of operations is a program and the 256 operations represent the programming operations.

Here we have essentially configured a simple micro-processor. The inputs and outputs would need to be connected to buses (via 3-state buffers where appropriate), which in turn are also connected to memories which store the program and any output or input data. The buses would also be connected to various input/output devices, mass storage devices, etc.

45

SYNTHESIS OF SYNCHRONOUS CIRCUITS: RECAPITULATION

1. Constructing the state transition diagram.
2. Selection or specifying the encoding of the states.
3. Constructing the state transition tables. It gives for each cycle the next-state of each flip-flop in the function of the previous states of all flip-flops and in the function of the control conditions (up/down).
4. Selection or specifying the type of flip-flop used in the implementation. Excitation table of the flip-flop type.
5. Determination of the logic functions of the control input(s) of each flip-flop. Performing the necessary or appropriate minimization.
6. Selection of the types of logic gates to be used and implementation of the feedback/control network.

46

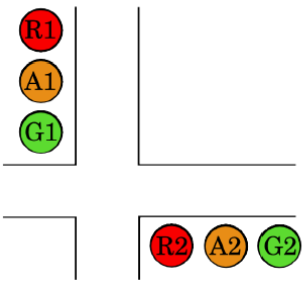
END OF THIS PART

Supplement follows: Traffic light control design example and case study.

47

TRAFFIC LIGHTS DESIGN EXAMPLE

Design a synchronous digital circuit, a Moore machine, which operates a traffic light at two types of road crossing.

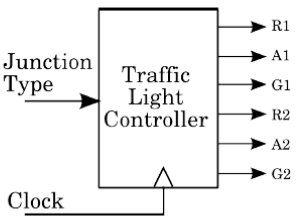


Quiet Junction

Red	Green
Red	Amber
Green	Red
Amber	Red

Busy Junction

Red	Green
Red	Amber
Red	Red
Green	Red
Amber	Red
Red	Red

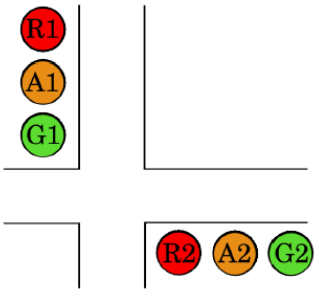


48

TRAFFIC LIGHTS DESIGN EXAMPLE

There are six lights to operate. The Red, Amber, and Green lights in the North-South direction will be designated as R1, A1, G1. Similarly, the lights in the East-West direction will be called R2, A2, and G2. When the digital signals are in the Logic-1 state they turn their respective lights on, otherwise the lights are off.

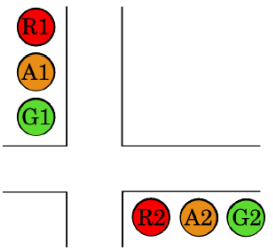
A digital clock signal will be supplied and at each clock pulse the lights should change according the schedule given above. The design of the circuit that produces the clock pulses at appropriate times will not be considered here.



49

TRAFFIC LIGHTS DESIGN EXAMPLE

There are two types of road crossing: quiet crossings that use a simple sequence, and busy crossings require a longer (delayed green) sequence. Some junctions may use the busy sequence during the day and the quiet sequence at night. One digital input signal called J (for junction type) will indicate whether the road crossing is considered quiet. J=0 denotes a busy junction and J=1 a quiet one. Thus, we have a one-input, six-output synchronous system to design.

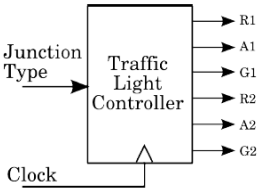


Quiet Junction

Red	Green
Red	Amber
Green	Red
Amber	Red

Busy Junction

Red	Green
Red	Amber
Red	Red
Green	Red
Amber	Red
Red	Red



50

FORMALIZATION OF THE PROBLEM

Most problems are first specified in a loose verbal form which must be made more rigorous.

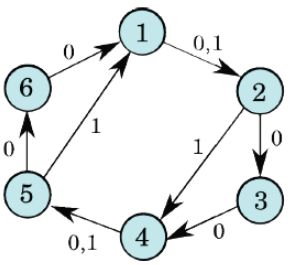
A good first step in this direction is to determine the number of states required.

Sometime the determination of the minimum number of states may be very difficult. However, our problem is simple enough to determine the states easily.

51

NUMBER OF STATES

State	R1	A1	G1	R2	A2	G2
1	1	0	0	0	0	1
2	1	0	0	0	1	0
3	1	0	0	1	0	0
4	0	0	1	1	0	0
5	0	1	0	1	0	0
6	1	0	0	1	0	0



Looking at the original specification, we see that there are six states (light patterns) for the busy intersection, and four states in the quiet junction. However we do not need teen states because all four states required for the quiet junction are also used in the busy junction. We need only six states. Let us number them 1 to 6 as shown in the table.

Quiet Junction

Red	Green
Red	Amber
Green	Red
Amber	Red

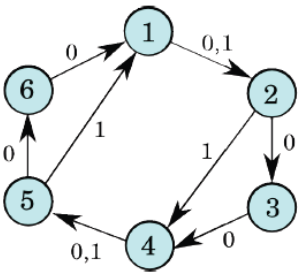
Busy Junction

Red	Green
Red	Amber
Red	Red
Green	Red
Amber	Red
Red	Red

52

NUMBER OF STATES

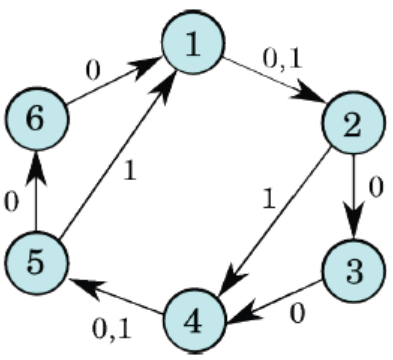
State	R1	A1	G1	R2	A2	G2
1	1	0	0	0	0	1
2	1	0	0	0	1	0
3	1	0	0	1	0	0
4	0	0	1	1	0	0
5	0	1	0	1	0	0
6	1	0	0	1	0	0



Two states (3 and 6) have exactly the same traffic light outputs. Could they be merged as one state? The answer is no, unfortunately, because the state after 3 is 4 while the state after 6 is 1.

53

STATE TRANSITION DIAGRAM



(The outputs (lamp controls) are ignored yet here.)

The state transition diagram is very simple. For the busy junction the states just cycle through in order. For the quiet junction states 3 and 6 are skipped. The state transition diagram is a finite state machine model and now represents a formal specification of what the circuit is required to do.

54

FLIP-FLOPS: TYPE AND NUMBER

Since the number of states is equal to six, the minimum number of flip-flops, which can support six states, is three.

The maximum number of flip-flops one may use is six (one flip-flop per state), though this implementation would clearly be wasteful and so we will use three D-type flip-flops. There will be two unused states.

55

STATE ASSIGNMENTS

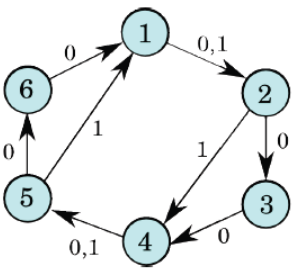
There are some heuristic rules for assigning states to flip-flop outputs, but they are difficult to apply and do not guarantee a minimum circuit.

One rule is to maximize the number of 1s. The idea is that a large number of 1s may provide easier minimization in the Karnaugh maps for the state sequencing logic. On this basis we will not use the states 000 and 001. The rest of the flip-flop outputs are assigned in order while constructing the transition table.

56

STATE TRANSITION TABLE

J	State(t)	Q1	Q2	Q3	State(t+1)	D1	D2	D3
0	7	0	0	0	X	X	X	X
0	8	0	0	1	X	X	X	X
0	1	0	1	0	2	0	1	1
0	2	0	1	1	3	1	0	0
0	3	1	0	0	4	1	0	1
0	4	1	0	1	5	1	1	0
0	5	1	1	0	6	1	1	1
0	6	1	1	1	1	0	1	0
1	7	0	0	0	X	X	X	X
1	8	0	0	1	X	X	X	X
1	1	0	1	0	2	0	1	1
1	2	0	1	1	4	1	0	1
1	3	1	0	0	X	X	X	X
1	4	1	0	1	5	1	1	0
1	5	1	1	0	1	0	1	0
1	6	1	1	1	X	X	X	X



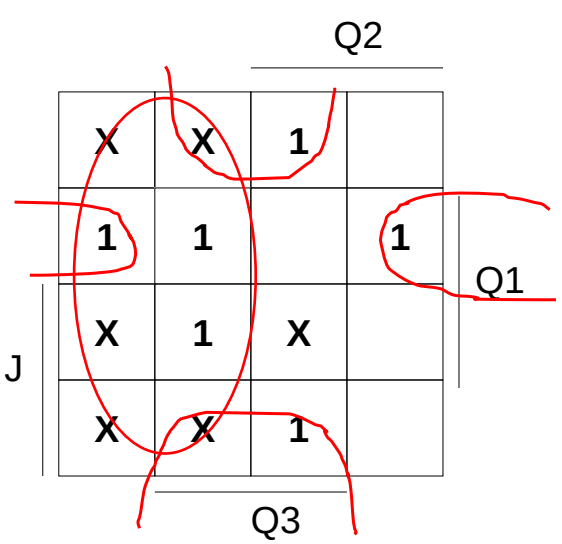
57

MINIMIZATION ON KARNAUGH MAP

The next step is to determine the required logic expressions for the three flip-flop inputs D1, D2, and D3.

58

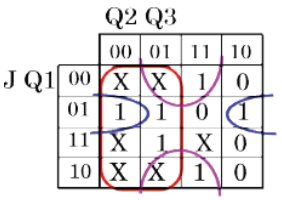
EXAMPLE: MINIMIZATION OF D1



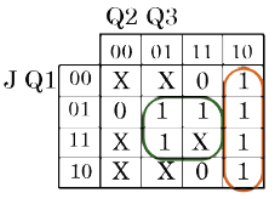
J	State(t)	Q1	Q2	Q3	State(t+1)	D1
0	7	0	0	0	X	X
0	8	0	0	1	X	X
0	1	0	1	0	2	0
0	2	0	1	1	3	1
0	3	1	0	0	4	1
0	4	1	0	1	5	1
0	5	1	1	0	6	1
0	6	1	1	1	1	0
1	7	0	0	0	X	X
1	8	0	0	1	X	X
1	1	0	1	0	2	0
1	2	0	1	1	4	1
1	3	1	0	0	X	X
1	4	1	0	1	5	1
1	5	1	1	0	1	0
1	6	1	1	1	X	X

59

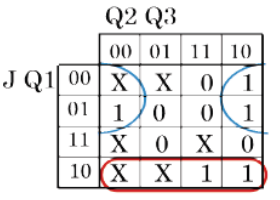
COMPLETE MINIMIZATION



D1 = Q2'+ Q3Q1' + J'Q1Q3'



D2 = Q1Q3 + Q2Q3'



D3 = J'Q3' + J Q1'

60

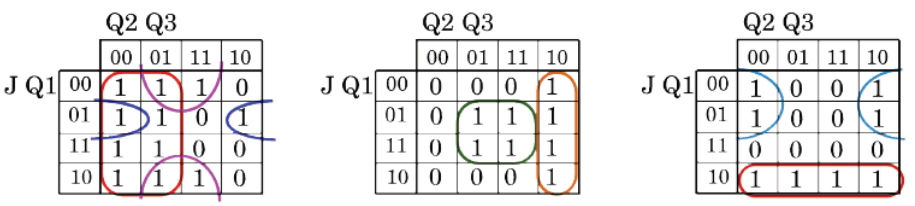
IMPLEMENTATION

Once the minimization has been done, we can replace the ”don’t care” outputs in the Karnaugh maps with the actual values we will get out of the circuit. Any don’t care inside a circle is replaced with 1, and any outside all circles is replaced with 0.

We have now a completely defined a sequential circuit and we should check whether the system behaves correctly even if it starts from one of the unused states. A convenient way of checking this is by constructing the complete transition diagram in which the unused states 7 and 8 are also included.

61

IMPLEMENTATION

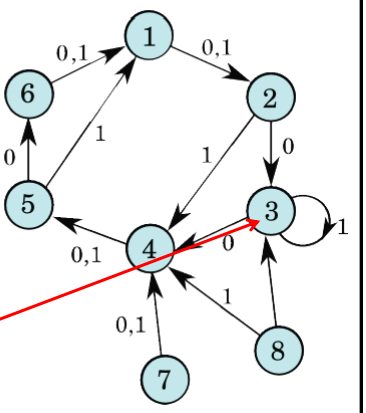


The Karnaugh maps now have a value for each of the don’t care states we used in the design, so we can rebuild the state transition table from them and draw the finite state machine that includes the unused states.

62

IMPLEMENTATION: DISASTER !

J	State(t)	Q1	Q2	Q3	State(t+1)	D1	D2	D3
0	7	0	0	0	4	1	0	1
0	8	0	0	1	3	1	0	0
0	1	0	1	0	2	0	1	1
0	2	0	1	1	3	1	0	0
0	3	1	0	0	4	1	0	1
0	4	1	0	1	5	1	1	0
0	5	1	1	0	6	1	1	1
0	6	1	1	1	1	0	1	0
1	7	0	0	0	4	1	0	1
1	8	0	0	1	4	1	0	1
1	1	0	1	0	2	0	1	1
1	2	0	1	1	4	1	0	1
1	3	1	0	0	3	1	0	0
1	4	1	0	1	5	1	1	0
1	5	1	1	0	1	0	1	0
1	6	1	1	1	1	0	1	0



Disaster strikes ! The system can be stuck in state 3 if J = 1 !

63

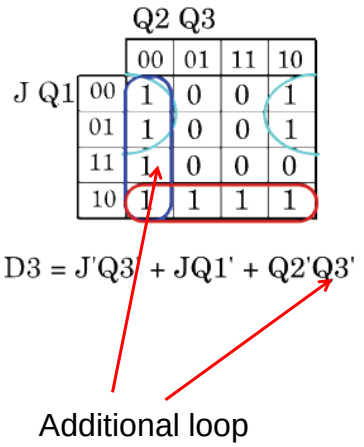
PROBLEM ANALYSIS

Disaster strikes! If the J input is logic 1 (quiet crossing) and the system finds itself in state 3, for example when switched on, or when it changes from busy mode to quiet mode, then it will be stuck in state 3. We have to go back and change some ”don’t care” bit(s) to fix the problem. This will mean revising our circles and the minimal equations.

The problem occurs at Karnaugh map entry 1100. Looking at the three maps, we can see that by changing the 0 indicated for D3 to a 1 will cause the circuit to go to state 4 from state 3 when the safe input is 1. This fixes the problem and causes minimal damage (i.e. will add one extra term to the expression).

64

REMEDY



65

OUTPUT CIRCUITS: LAMP CONTROLS

For each state we need to generate the signals that light the correct traffic light bulbs. There are six such circuits but fortunately they have three inputs only (Q1,Q2,Q3). Their K-Maps can be filled out by the requirements of lights to be either ON or OFF for each given state. Here again we will start by ignoring the two unused states which will provide ”don’t care” outputs to find the minimized circuits. Again, filling out the K-maps with the selected 1s and 0s will give us the actual operation of the lights for states 7 and 8. We will have to look at these whether they are safe. From the original specification we can fill in the values for the bulb outputs.

66

FINAL CIRCUIT SPECIFICATION

In summary we have the following circuits to build. The common terms are underlined. They need only be implemented once in hardware.

$D1 = Q2' + J' \cdot \underline{Q1 \cdot Q3'} + \underline{Q1' \cdot Q3}$
$R1 = Q1' + \underline{Q2' \cdot Q3'} + Q2 \cdot Q3$
$R2 = Q1$

$D2 = Q1 \cdot Q3 + \underline{Q2 \cdot Q3'}$
$A1 = Q2 \cdot \underline{Q1 \cdot Q3}$
$A2 = \underline{Q1' \cdot Q3}$

$D3 = \underline{Q2' \cdot Q3'} + J' \cdot Q3' + J \cdot Q1'$
$G1 = \underline{Q2' \cdot Q3}$
$G2 = Q1' \cdot Q3'$

67

FURTHER MODIFICATIONS: DIFFERENT STATE ASSIGNMENTS

If we want to try to find a simpler overall circuit, we may try different flip-flop assignments for the states. One idea is to minimize the output circuitry. We could, for example, make R1=Q1 and R2=Q2, if these simple assignments will give us a correct complete state assignment. The third output, Q3 has to be assigned such that all used states are distinct. One possible set of assignments are shown below:

68

A DIFFERENT STATE ASSIGNMENT

State	Q1	Q2	Q3	R1	A1	G1	R2	A2	G2
1	1	0	0	1	0	0	0	0	1
2	1	0	1	1	0	0	0	1	0
3	1	1	1	1	0	0	1	0	0
4	0	1	1	0	0	1	1	0	0
5	0	1	0	0	1	0	1	0	0
6	1	1	0	1	0	0	1	0	0
7	0	0	0	X	X	X	X	X	X
8	0	0	1	X	X	X	X	X	X

The output circuits are quite a lot simpler and smaller, but of course, we have to redesign the state sequencing logic circuitry with the new flip-flop state assignments.

69

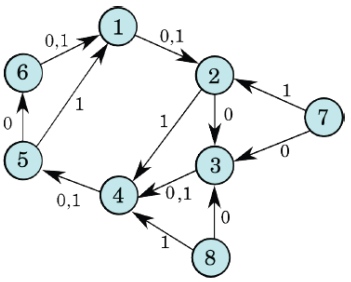
NEW STATE TABLES

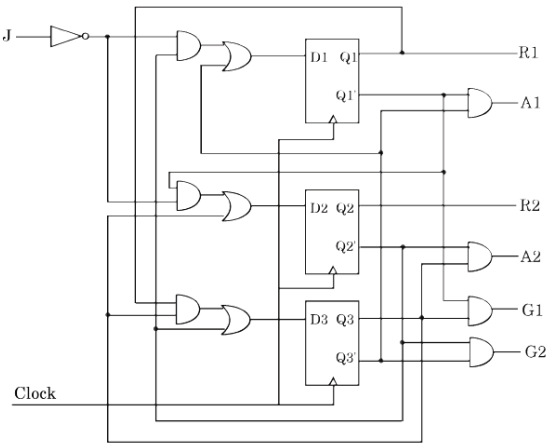
J	State(t)	Q1	Q2	Q3	State(t+1)	D1	D2	D3
0	7	0	0	0	X	X	X	X
0	8	0	0	1	X	X	X	X
0	1	1	0	0	2	1	0	1
0	2	1	0	1	3	1	1	1
0	3	1	1	1	4	0	1	1
0	4	0	1	1	5	0	1	0
0	5	0	1	0	6	1	1	0
0	6	1	1	0	0	1	0	0
1	7	0	0	0	X	X	X	X
1	8	0	0	1	X	X	X	X
1	1	1	0	0	2	1	0	1
1	2	1	0	1	4	0	1	1
1	3	1	1	1	X	X	X	X
1	4	0	1	1	5	0	1	0
1	5	0	1	0	1	1	0	0
1	6	1	1	0	X	X	X	X

The resulting circuit is simpler than the first one, and if we check the don't care states - which you can do as an exercise - it turns out to be safe. The final state diagram and circuit looks like this:

70

FINAL CIRCUIT





71

REVISION QUESTIONS

1. Describe the main steps involved in the synthesis of a synchronous sequential circuit/finite state machine.

2. Describe and illustrate with a simple example the operation of a state machine with memory.

72

PROBLEMS AND EXERCISES

1. In a complete state machine, all possible transitions between states of a finite state machine (FSM) should be specified. Your state machine has two inputs, A and B, and two states, S0 and S1. You are told that your FSM behaves as follows:
The FSM moves from S0 to S1 if and only if A = 1.
The FSM moves from S1 to S0 if and only if A = 0 and B = 1.
Draw the complete state transition diagram of your FSM.

2. Analyze the operation of the type 7474 edge triggered D flip-flop and give its full operational/truth table for asynchronous and synchronous control.

73

PROBLEMS AND EXERCISES

3. Design a finite state machine which determines whether the two 4-bit binary numbers arriving simultaneously on the two inputs are equal or not. If not, it should also indicate which is the greater. The code-words in pairs arrive cyclically to the **X** and Y inputs of the circuit. The MSBs arrive at first to the input.

4. Four-bit codewords representing normal BCD coded decimal digits arrive cyclically to the **X** input of a synchronous sequential circuit. The MSB arrives first. Design a synchronous sequential circuit which indicates with 1 on its **Z** output if the arriving 4-bit codeword presumably representing a normal BCD digit is invalid.

74

PROBLEMS AND EXERCISES

5. Design a synchronous sequential circuit to control a bottled drink vending machine. A bottle of drink costs 200 HUF. The machine accepts 50, 100, and 200 HUF coins. When the amount of money inserted equals or exceeds the price of the merchandize, the machine vends a bottle and returns change if any, then waits for the next transaction.

6. Design a synchronous counter according to the specifications given below:

Encoding: Excess-3 (Stibitz) code
Counting direction: up or down, externally controllable
Mode of operation: self-correcting (returns to the counting cycle from the invalid states).

75

END OF LECTURE

76