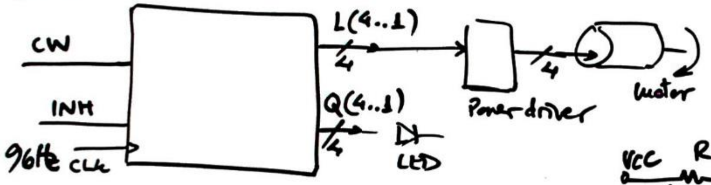
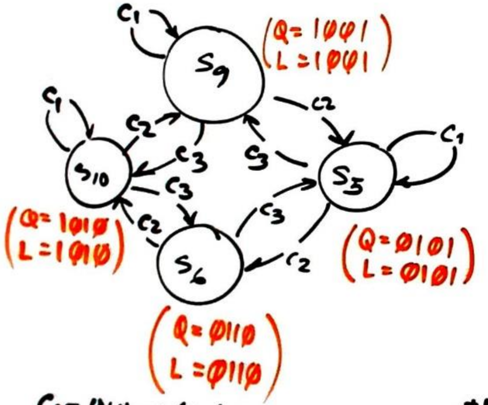
PROBLEM 3. Standard procedure and solution using microcontrollers in CSD style to implement FSM. Similar circuits and applications in P10 and P12.

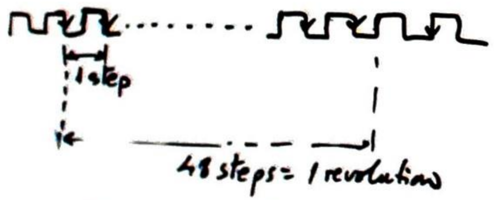
CW=1 (as count up)
CW=0 (as count down)



INH is like count disable

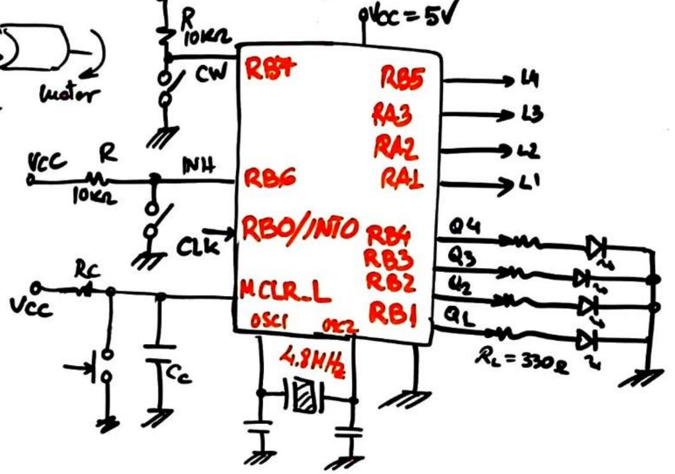


C1 = INH = 1 (inhibit)
C2 = INH = 0 and CW = 1 (clockwise) rotation
C3 = INH = 1 and CW = 0 (counterclockwise) rotation



FSM (P6 project) It is like a reversable counter that counts only 9 → 5 → 6 → 10 → 9 → ... CW
... ← 10 ← 9 ← 5 ← 6 ← 10 ← 9 CCW

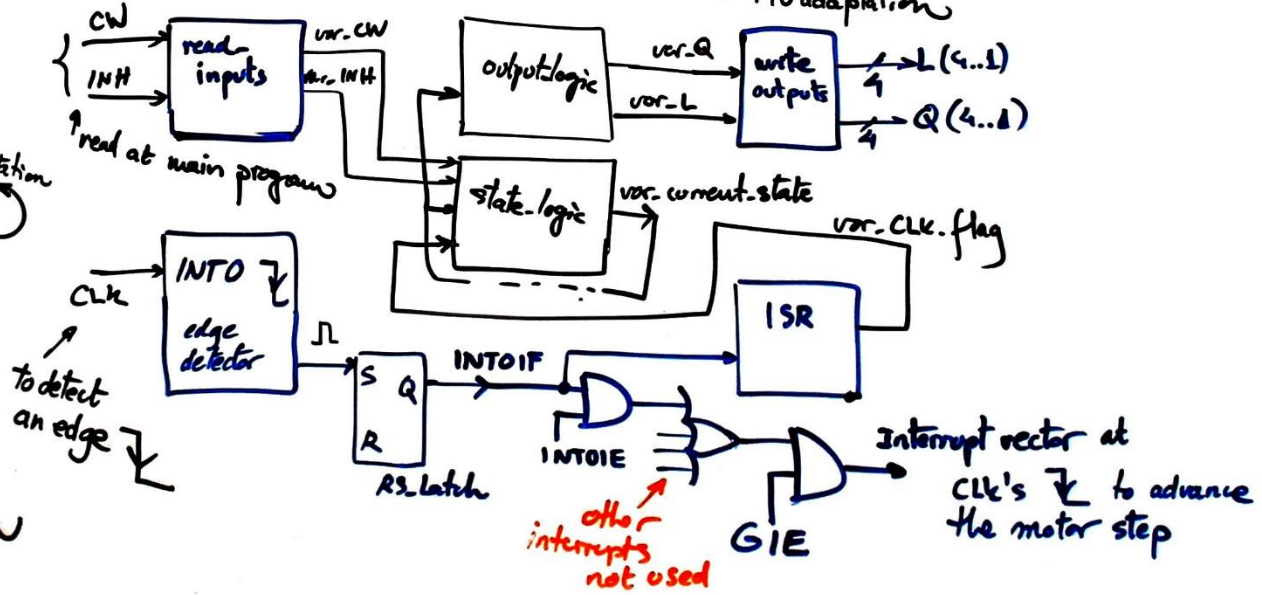
1. Example schematic



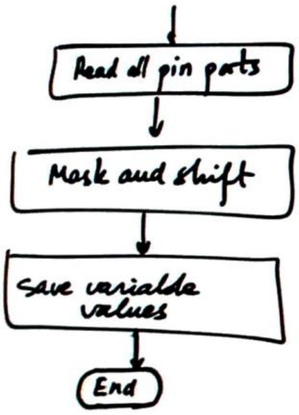
3. RAM variables

7	6	5	4	3	2	1	0	
								var_current_state (ASCII code A, B, C, D)
Ø	Ø	Ø	Ø	Ø	Ø	Ø	INH	var_INH
Ø	Ø	Ø	Ø	Ø	Ø	Ø	CW	var_CW
Ø	Ø	Ø	Ø	Ø	Ø	Ø	•	var_CLK_flag
Ø	Ø	Ø	Ø	Q4	Q3	Q2	Q1	var_Q
Ø	Ø	Ø	Ø	L4	L3	L2	L1	var_L
7	6	5	4	3	2	1	0	

2. Hardware-software diagram → FSM architecture P10 adaptation

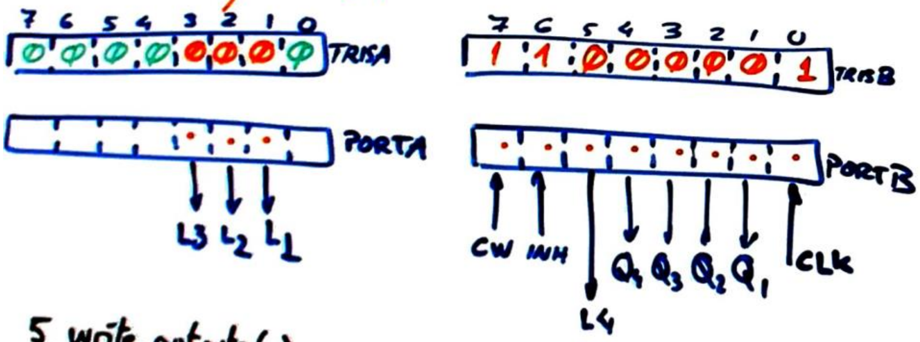


4. Read_inputs() to poll level sensitive inputs

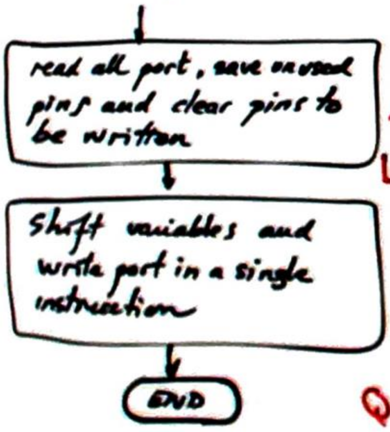


```
var_buf = PORTB & 11000000;
var_buf2 = (var_buf & 01000000);
var_INH = var_buf2 >> 6;
var_buf2 = (var_buf & 10000000);
var_CW = var_buf2 >> 7;
```

* init_system()



5. write_outputs()


read all port, save unused pins and clear pins to be written
Shift variables and write port in a single instruction
END

L

```
var_buf = PORTA & 0b11110001;
var_buf2 = (var_L & 0b00000111) << 1;
PORTA = var_buf | var_buf2;
var_buf2 = (var_L & 0b00001000) << 2;
var_buf = PORTB & 0b11011111;
PORTB = var_buf | var_buf2;
```

Q

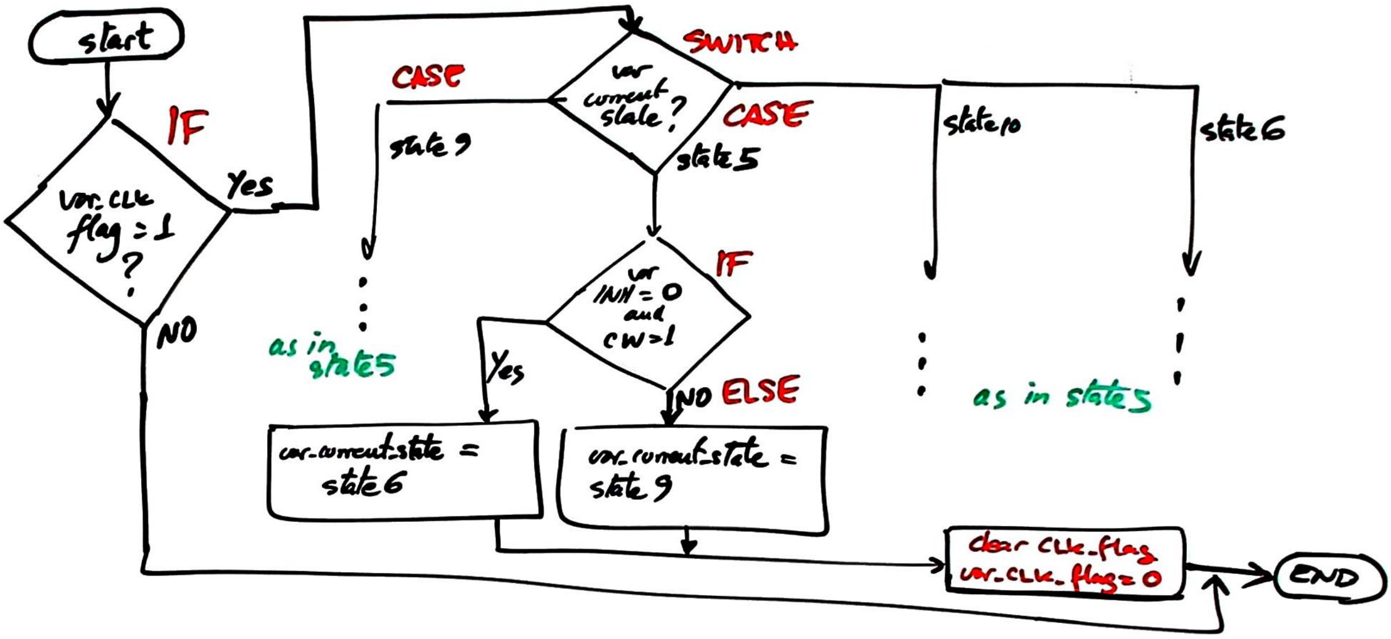
```
var_buf = PORTB & 0b11100001;
var_buf2 = var_Q << 1;
PORTB = var_buf2 | var_buf;
```

6. state_logic truth table and flowchart

⇒ (only when var_CLK_flag = 1)

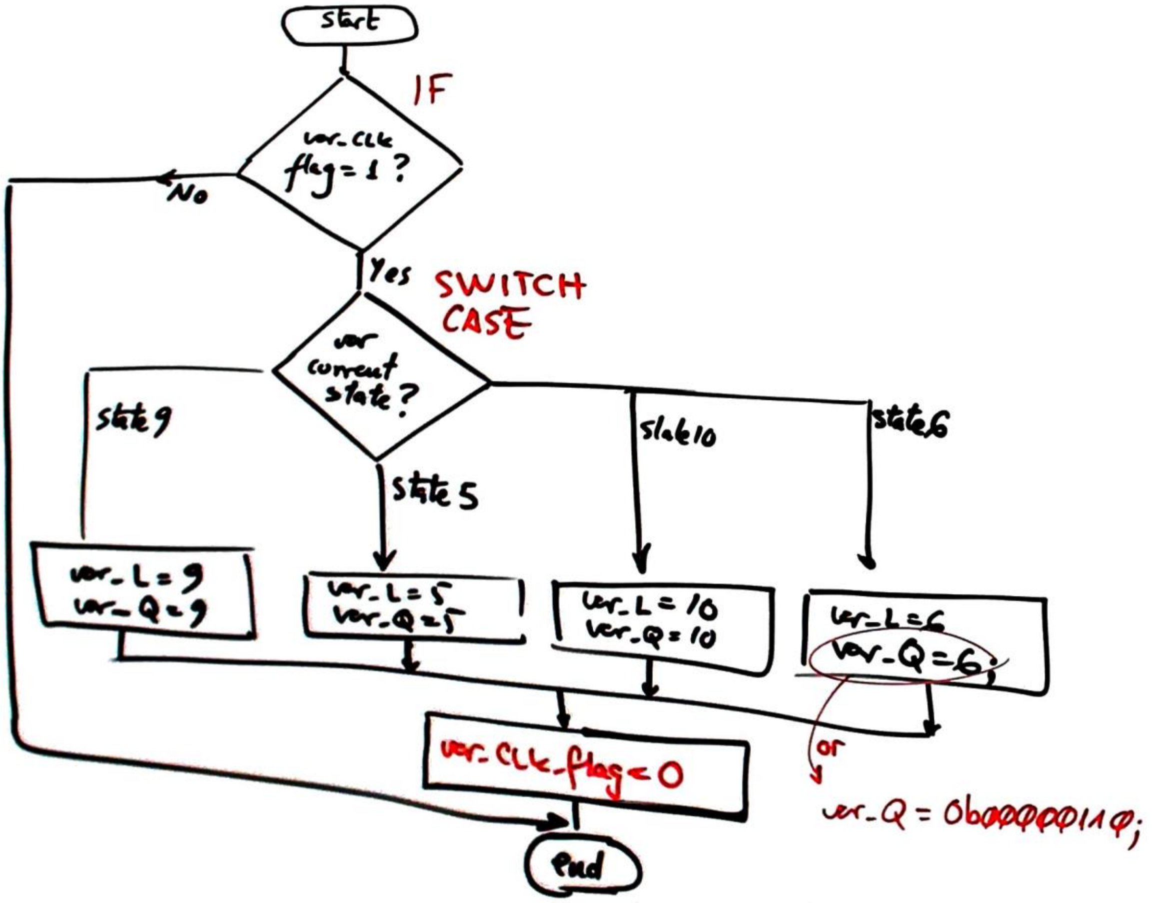
var_INH	var_CW	var_current_state	var_current_state+
1	x	state 9	state 9
0	1	state 9	state 5
0	0	state 9	state 10
	⋮		⋮
1	x	state 10	state 10
0	1	state 10	state 9
0	0	state 10	state 6

← after executing the C instruction

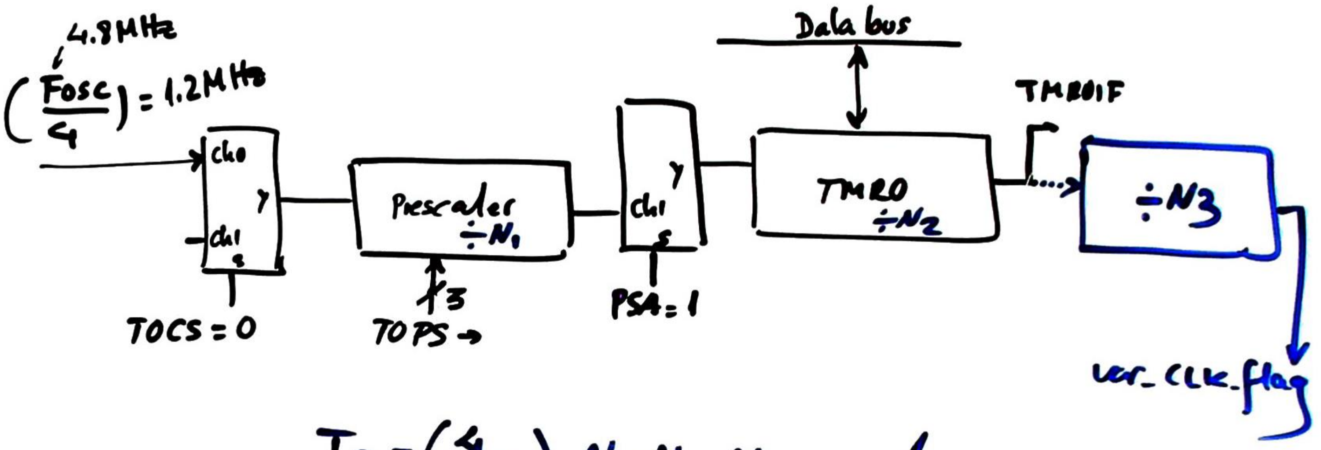


7. output_logic( ) truth table and flowchart

var_current_state	var_Q	var_L
state 5	5	5
state 10	10	10
state 9	9	9
state 6	6	6



8. TMR0 to generate var_CLK_flag



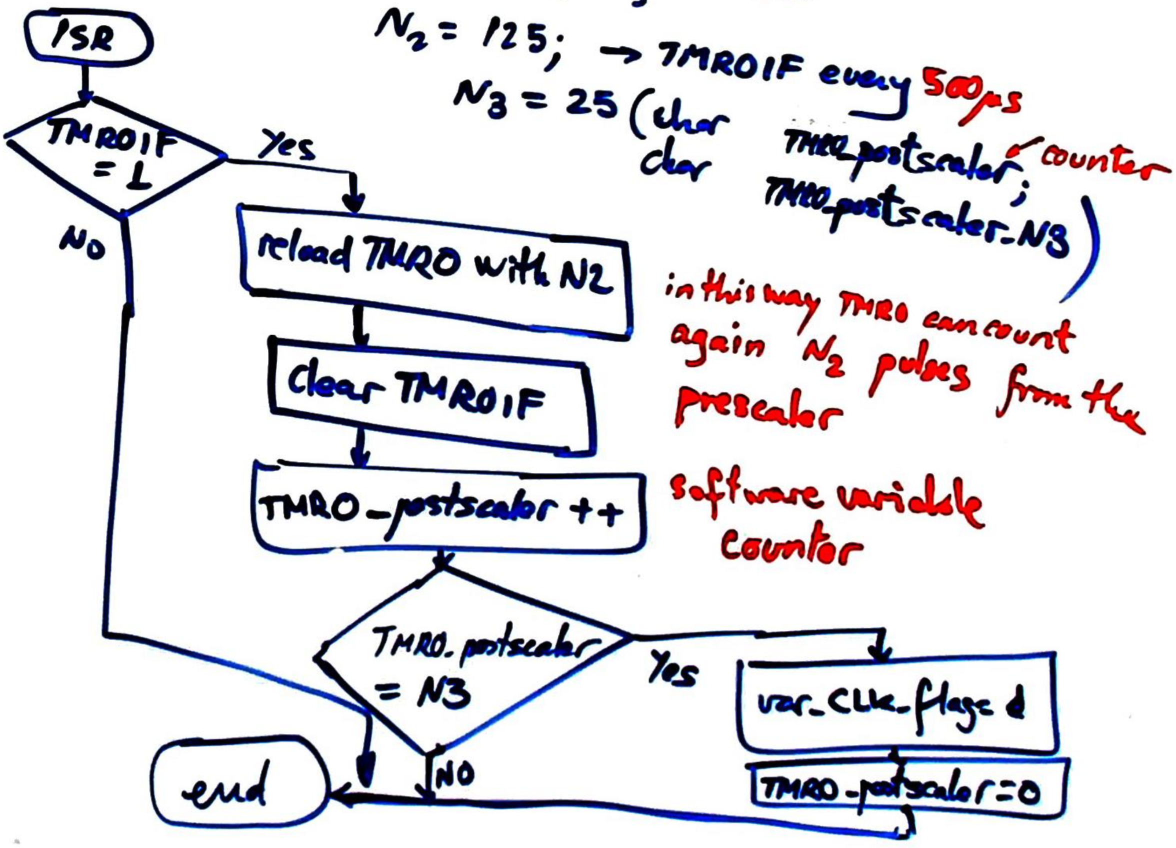
$$T_p = \left(\frac{4}{Fosc}\right) \cdot N_1 \cdot N_2 \cdot N_3 = \frac{1}{96 Hz}$$

$$N_1 \cdot N_2 \cdot N_3 = \frac{1.2M}{96} = 12500$$

$N_1 = 4$; $N_2 \cdot N_3 = 3125$

$N_2 = 125$; → TMR0IF every 500µs

$N_3 = 25$ (char TMR0_postscaler; ← counter
char TMR0_postscaler_N3)



Using this planning, it is straightforward to obtain the C code and the Proteus schematic copying and adapting files from another P10 example project in *digsys.upc.edu*.
In addition, this stepper motor controller can be written as another example of learning materials in:
P10. Phase #1: Stepper motor controller using an external CLK.
P12. Phase #2: Stepper motor controller using TM0 to generate CLK, as an improvement from phase #1.

Logically, even previously to µC adaptations, the same project steeper motor controller can be organised in P6 as a canonical FSM using VHDL and EDA tools for a FPGA/CPLD target chip. A programmable *CLK_generator* can be implemented in P8 so that the motor can operate at different rotation angular speeds.