Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

Departament d'Enginyeria Electrònica

DIGITAL CIRCUITS AND SYSTEMS

P_Ch3: 6-bit Johnson counter

Cooperative group

TEAM NUMBER: 2 GXX_____
DUE DATE: _____**17/06/2019**_____ 1st review due date: _____
STUDY TIME:

Study time (in hours)	Group work	Classroom and laboratory sessions		Sessions out of classroom	
	Individual	Student 1			
		Student 2			
		Student 3			

STATEMENT:
My signature below indicates that I have (1) made equitable contribution to the application project as a member of the group, (2) read and fully agree with the contents (i.e., results, conclusions, analyses, simulations) of this document, and (3) acknowledged by name anyone outside this group who assisted this learning team or any individual member in the completion of this document.

Today's date: 06/13/2019

Active members
(1) Ricardo Tomas Castro
(2) Jordi Figueras García
(3) Carlos Garcia

Acknowledgement of individual(s) who assisted this group in completing this document:
(1) Lecturer's office time to solve some questions
(2) _____

Abstract

In this project we divide it in three phases; in the first phase we saw how a Johnson counter worked in essence, and also, we added a way to count upwards or downwards by adding a switch to our chip. In phase two we added a trigger in form of a start and stop button and lcd to see how an interrupt affected the counter and seeing the change in status. Finally, in phase three we added a timer, to replace the external clock, being sure that the clock from the timer won't be different from the external clock thus allowing us to reducing the size of our chip.

Table of content

1 Project specifications

1.1 Specifications

The idea is to design a system able to count from zero to eleven using an external clock or an internal one (in form of a timer).

1.1.1 Truth table

CE	*UD_L*	*ST_SP*	*Current_state*	*Next_state*
0	x	x	Current_state	Current_state
1	1	1	Idle	Num_0
1	0	1	Idle	Num_11
1	x	0	Idle	Idle
1	1	x	Num_0	Num_1
1	0	1	Num_0	Idle
1	0	0	Num_0	Num_11
1	1	x	Num_1	Num_2
1	0	x	Num_1	Num_0
1	1	x	Num_2	Num_3
1	0	x	Num_2	Num_1
1	1	x	Num_3	Num_4
1	0	x	Num_3	Num_2
1	1	x	Num_4	Num_5
1	0	x	Num_4	Num_5
1	1	x	Num_5	Num_6
1	0	x	Num_5	Num_4
1	1	x	Num_6	Num_7
1	0	x	Num_6	Num_5
1	1	x	Num_7	Num_8
1	0	x	Num_7	Num_6
1	1	x	Num_8	Num_9
1	0	x	Num_8	Num_7
1	1	x	Num_9	Num_10
1	0	x	Num_9	Num_8
1	1	x	Num_10	Num_11
1	0	x	Num_10	Num_9
1	1	1	Num_11	Idle
1	1	0	Num_11	Num_0
1	0	x	Num_11	Num_10

CC1 Truth Table

Current_state	*Q5*	*Q4*	*Q3*	*Q2*	*Q1*	*Q0*	*TC*
Idle	0	0	0	0	0	0	0
Num_0	0	0	0	0	0	0	1
Num_1	1	0	0	0	0	0	1
Num_2	1	1	0	0	0	0	1
Num_3	1	1	1	0	0	0	1
Num_4	1	1	1	1	0	0	1
Num_5	1	1	1	1	1	0	1
Num_6	1	1	1	1	1	1	0
Num_7	0	1	1	1	1	1	0
Num_8	0	0	1	1	1	1	0
Num_9	0	0	0	1	1	1	0
Num_10	0	0	0	0	1	1	0
Num_11	0	0	0	0	0	1	0

CC2 Truth Table

CC1's truth table stays relatively the same through the three phases, the only time it may differ is in the first phase, since it doesn't have an idle state and ST_SP button it just transfers through from Num_0 to Num_11 and vice versa depending if it's counting up or down as shown below:

CE	*UD_L*	*Current_state*	*Next_state*
0	x	Current_state	Current_state
1	1	Num_11	Num_0
1	0	Num_0	Num_11
1	1	Num_0	Num_1
1	1	Num_1	Num_2
1	0	Num_1	Num_0
1	1	Num_2	Num_3
1	0	Num_2	Num_1
1	1	Num_3	Num_4
1	0	Num_3	Num_2
1	1	Num_4	Num_5
1	0	Num_4	Num_5
1	1	Num_5	Num_6
1	0	Num_5	Num_4
1	1	Num_6	Num_7
1	0	Num_6	Num_5
1	1	Num_7	Num_8
1	0	Num_7	Num_6
1	1	Num_8	Num_9
1	0	Num_8	Num_7
1	1	Num_9	Num_10
1	0	Num_9	Num_8
1	1	Num_10	Num_11
1	0	Num_10	Num_9
1	0	Num_11	Num_10

CC1 Phase 1

Current_state	*Q5*	*Q4*	*Q3*	*Q2*	*Q1*	*Q0*	*TC*
Num_0	0	0	0	0	0	0	1
Num_1	1	0	0	0	0	0	1
Num_2	1	1	0	0	0	0	1
Num_3	1	1	1	0	0	0	1
Num_4	1	1	1	1	0	0	1
Num_5	1	1	1	1	1	0	1
Num_6	1	1	1	1	1	1	0
Num_7	0	1	1	1	1	1	0
Num_8	0	0	1	1	1	1	0
Num_9	0	0	0	1	1	1	0
Num_10	0	0	0	0	1	1	0
Num_11	0	0	0	0	0	1	0

CC2 Phase 1

1.1.2 State diagram

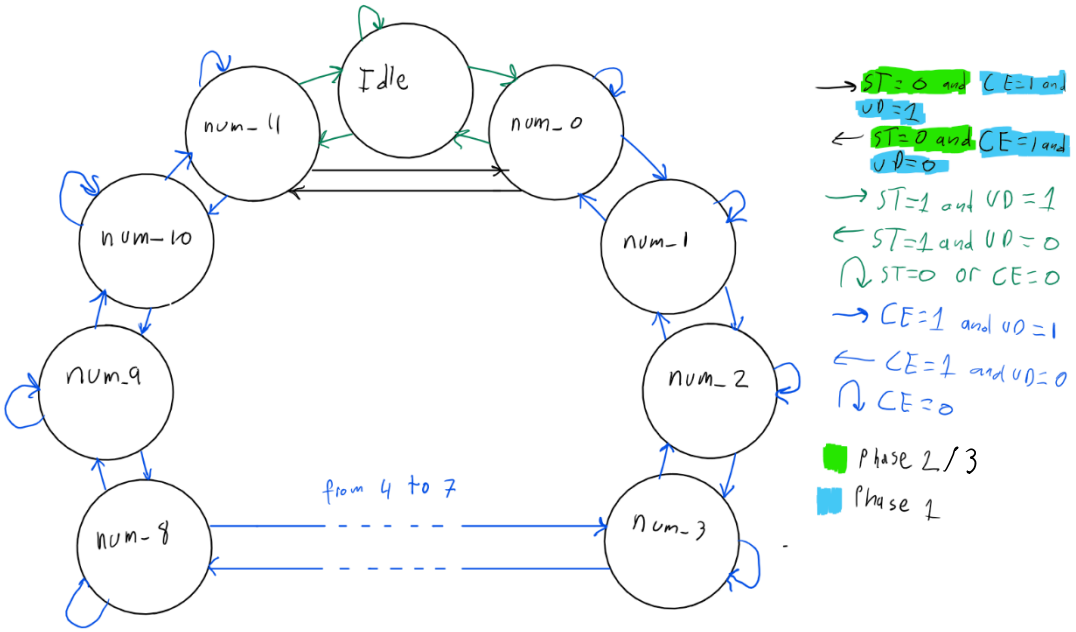


Fig. 1 state diagram showing several operations of the circuit.

1.2 Theory

A circuit which is used to count the number of times an event occurs is called a counter. In digital sense, these circuits comprise of bi-stable devices called flip-flops arranged in a particular fashion. Such a chain can be regarded to be a shift register, due to which counters can be considered as an application of shift registers. Either D or JK type flip-flops are preferred while designing the counter circuits. On (Electrical4U, 2018) such counter designed using D flip-flop chain is shown by Figure 1 and is called **Johnson counter**.

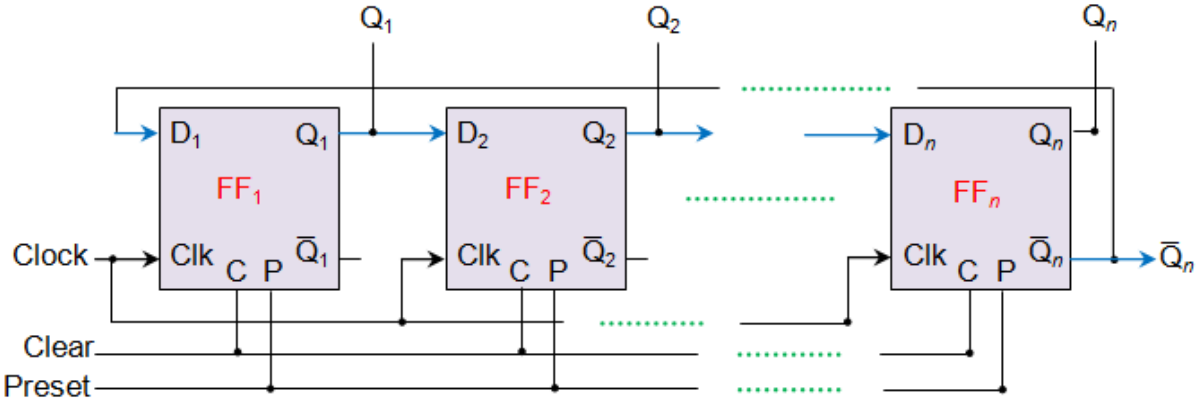


Figure 1 *n*-bit Johnson Counter Designed Using D Flip-Flops

Fig. 1 This is an example of picture caption.

The schematic shows a cascaded arrangement of n flip-flops in which the output of the preceding flip-flop is fed as an input to the immediate next flip-flop. However, it is to be noted that the complement output of the last flip-flop $\bar{Q}n$ is back fed to the first flip-flop in the chain. This arrangement results in a closed loop due to which

the bits within the counter continuously circulate within it. Further, the schematic shown in the figure is seen to comprise of n flip-flops due to which it is called n-bit Johnson counter. Further, the counter has preset (P) and clear (C) pins meant to initialize and reset the counter, respectively.

(Electrical4U, 2018)

1.3 How Our Circuit Works

In the final design we use three switches to control the count enable, the count up or down option and the option to use the timer two as an internal clock signal instead of the external one. Then we also have two buttons, one for the clear direct and one for the start or stop that functions as an interrupt. Finally, the external clock generator. Those were our inputs, for our outputs we Q (zero to five) that will be connected to leds, TC12_SQ that is also connected to a different color of led from the Qs while being connected to an oscilloscope, lastly we also have an lcd connected in the output of our chip.

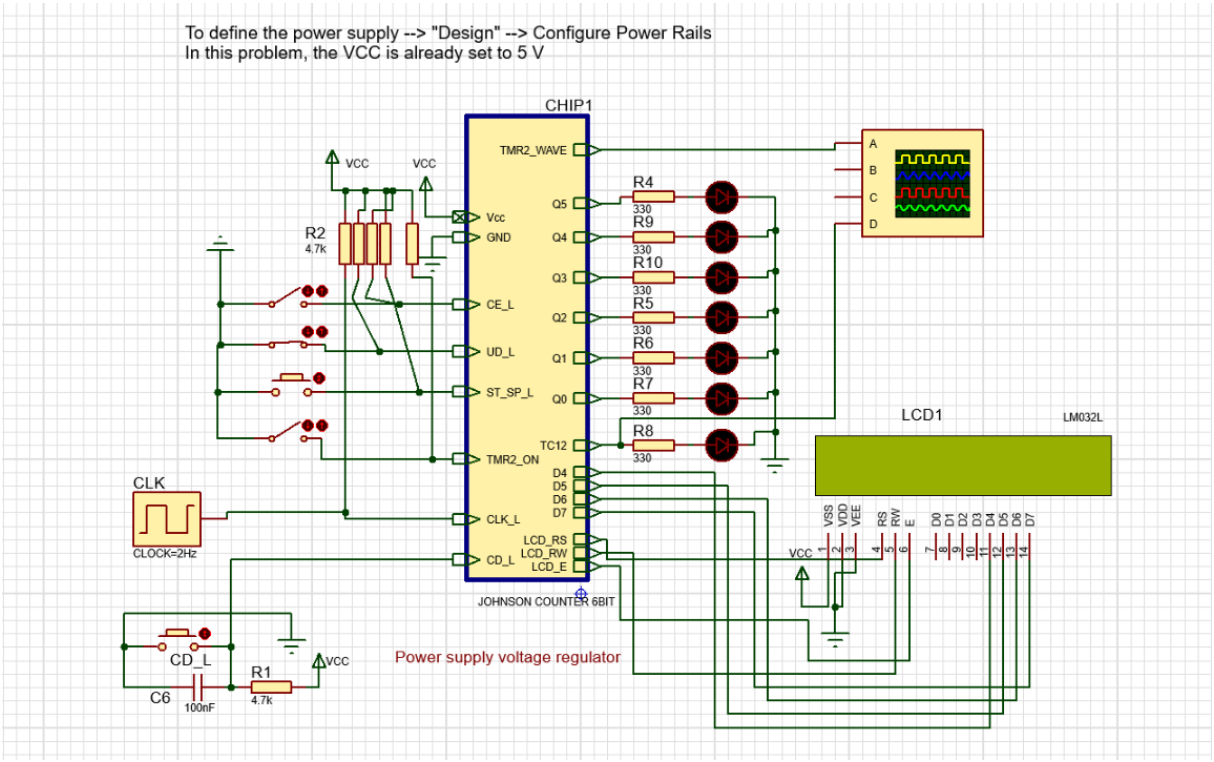
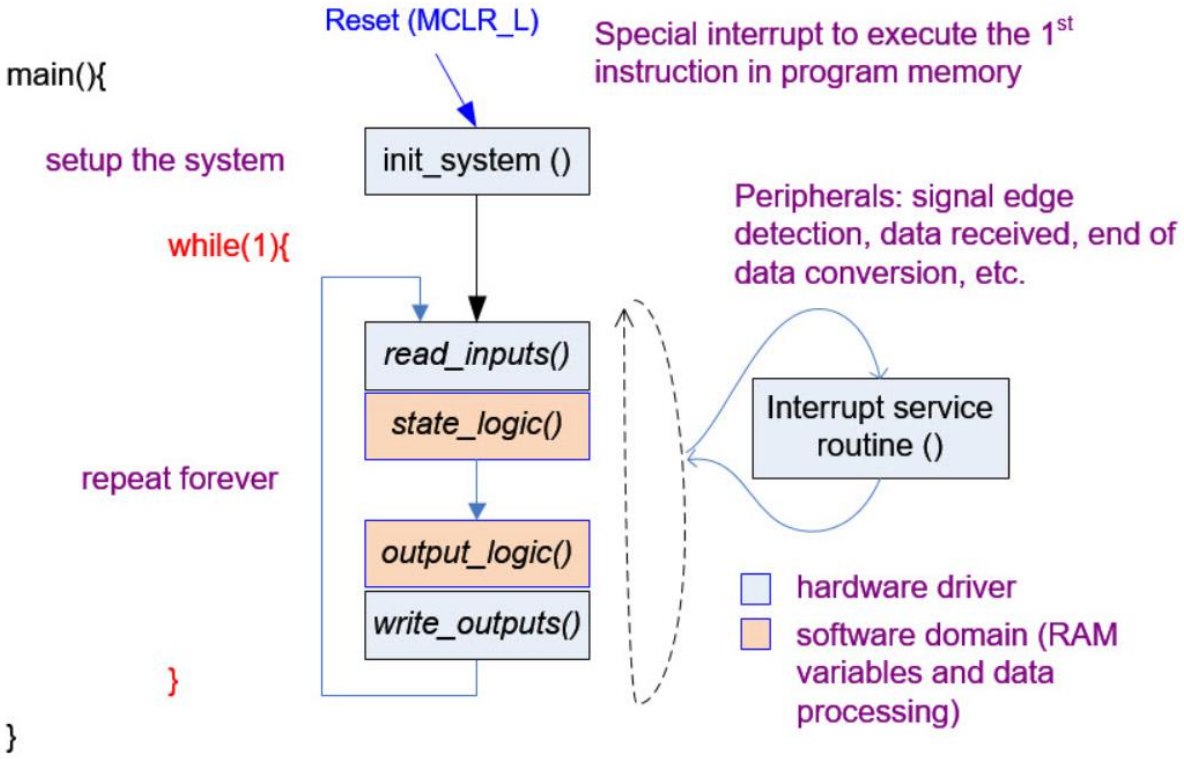


Figure 2: how our chip is connected to other components simulated in proteus

The way our chip works, when booted up it will start in an idle state. It will remain idle until the count enable switch is open and when the user presses the ST_SP_L button. This will enable the chip to jump form idle to certain number depending if UD_L is open or closed (open=0, closed=1). During the counting operation you can see the leds lighting up and also lcd will print out a message telling the user what number the leds are representing and in which direction the chip is counting (all outputs can be better understood by looking at the truth table of cc1 and cc2). The user can also choose which source to use for the clock (external or the timer as an internal clock), note the clock is always the same. Then we also have an oscilloscope connected to the TC12 and to our output of the TMR2_wave (you will be able to see the output of the TMR2_wave if TMR2_ON switch is open). The CD button is only used to reset the chip.

2 Planning

2.1 Software diagram

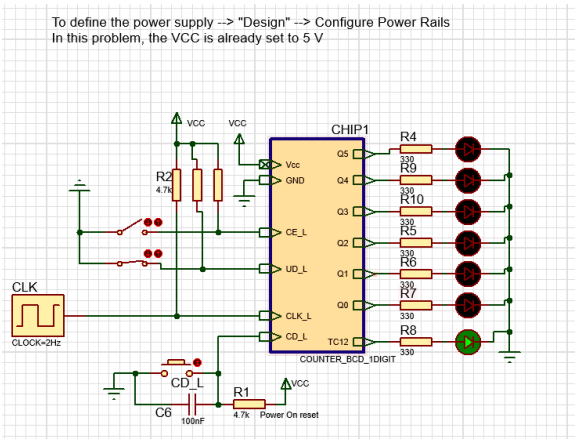


(J. Jordana; Robert, Francesc J., 2001)

2.2 Hardware diagram

2.2.1 Phase 1

- In this phase we have count enable (CE_L signal switch) and up and down (UD_L signal switch).
- Advances synchronously on the falling edge of an external clock input of 2Hz.
- TC (terminal count)
- CD clear direct (reset)



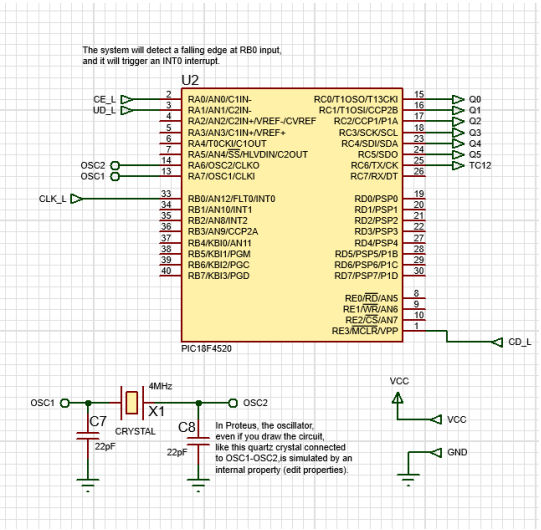
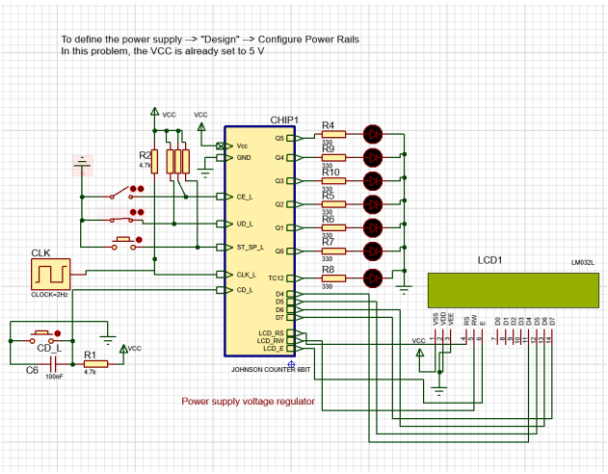


Figure i: phase 1 hardware configuration

2.2.2 Phase 2

- Count enable (CE_L).
- Up and down (UD_L).
- External clock of 2Hz.
- TC12 (terminal count).
- CD (clear direct).
- We add ST_SP (start and stop) button.
- We add an LCD.



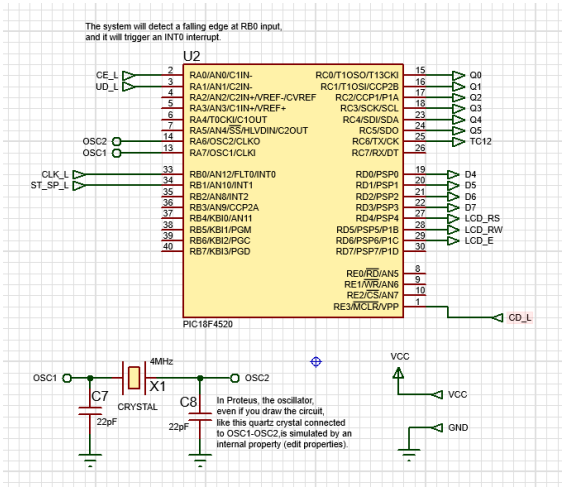
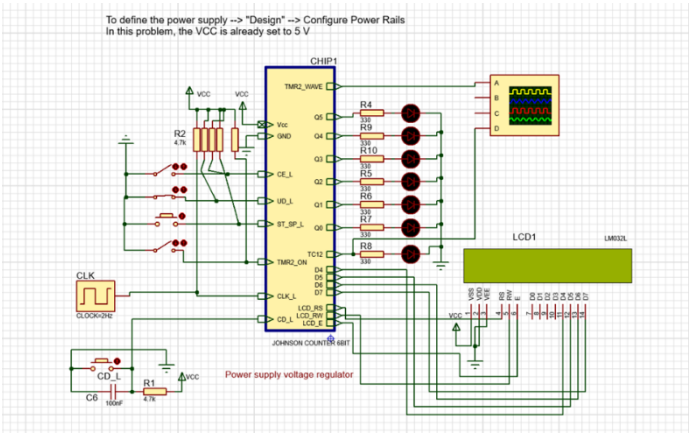


Figure ii: phase 2 hardware configuration

2.2.3 Phase 3

- Count enable (CE_L).
- Up and down (UD_L).
- External clock of 2Hz.
- TC12 (terminal count) that we connect to an oscilloscope.
- CD (clear direct).
- ST_SP (start and stop) button.
- LCD.
- We add TMR2_ON (switch).
- We add TMR2_wave, an output that is connected to an oscilloscope.



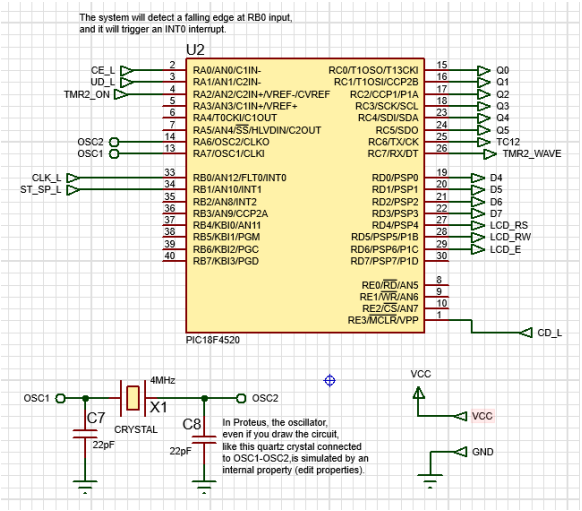
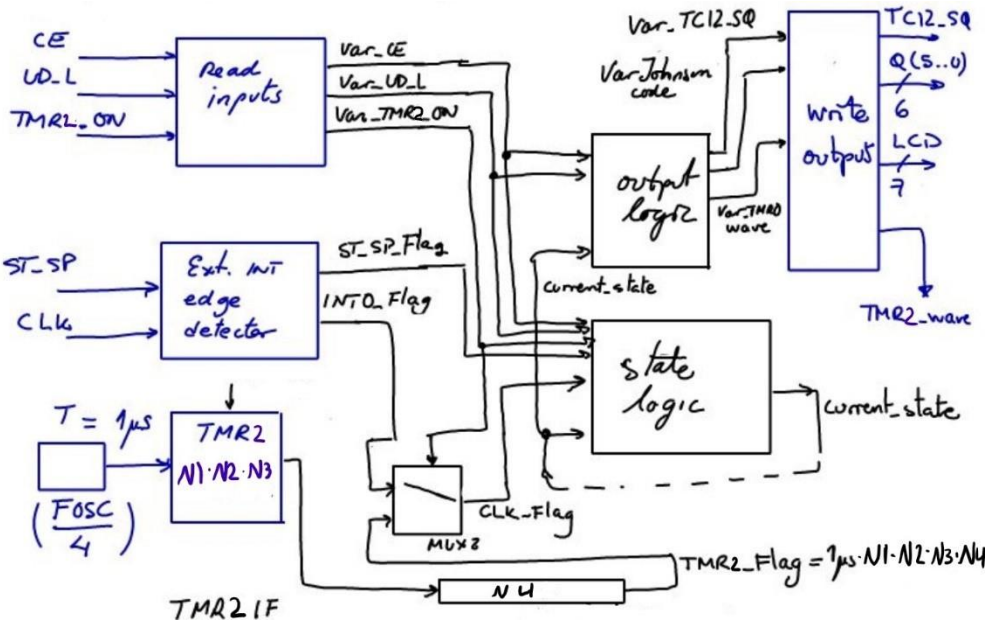


Figure iii: phase 3 hardware configuration

2.3 Hierarchical block diagram

This Fig. is an example of picture representing a plan or a diagram. It can be scanned from an sketch in a sheet of paper.



(Robert, 2018)

Fig. 4 hardware and software blocks involved in the final design.

2.4 Bit configuration of init_system(), read inputs and write outputs

2.4.1 init_system()

as shown (Figure a), we simply must initialize each port and, in each TRIS, we prime its value by giving it a binary value depending on which pin in the port on our chip (PIC18F4520) will be used as input or output.

init_system()									
inputs = '1'		PORTA = 0x00							
outputs = '0'			A7	A6	A5	A4	A3	A2	A1	A0
		TRISA = 0b	0	0	0	0	0	0	1	1
		PORTB = 0x00								
			B7	B6	B5	B4	B3	B2	B1	B0
		TRISB = 0b	0	0	0	0	0	0	1	1
		PORTC = 0x00								
			C7	C6	C5	C4	C3	C2	C1	C0
		TRISC = 0b	0	0	0	0	0	0	0	0
		PORTD = 0x00								
			D7	D6	D5	D4	D3	D2	D1	D0
		TRISD = 0b	0	0	0	0	0	0	0	0

Figure iv

2.4.2 read_inputs()

as shown (Figure b), when we read Port A its first three bits will have information (represented by ones in the image). To obtain the variables that we want we apply a mask to Port A by doing an and bitwise to the port and if bit is not in the right position, we shift them right or left depending where we want the bit in the final variable.

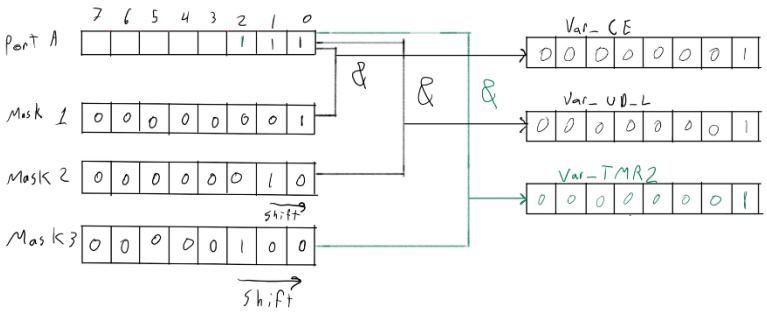


Figure v: black lines are for Phase 1 and 2, the black and green lines are for phase 3

2.4.3 write_outputs()

as shown (Figure c), we have the variables ready to be outputted. Depending in which phase we are in there is a condition, that if meet we will write on out or another, in any case we will write in Port C the result of doing an or bitwise of Var_Q, Var_TC12 after its bits being shifted six positions to the right and TMR2_wave after its bits being shifted seven positions to the right.

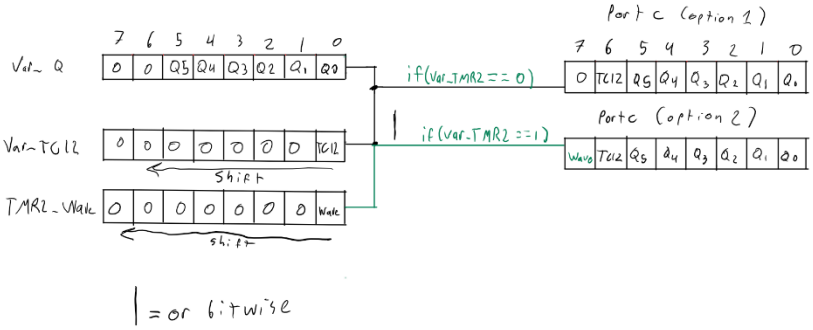


Figure vi: black lines are for Phase 1 and 2, the black and green lines are for phase 3

2.4.4 flow charts (state and output logic)

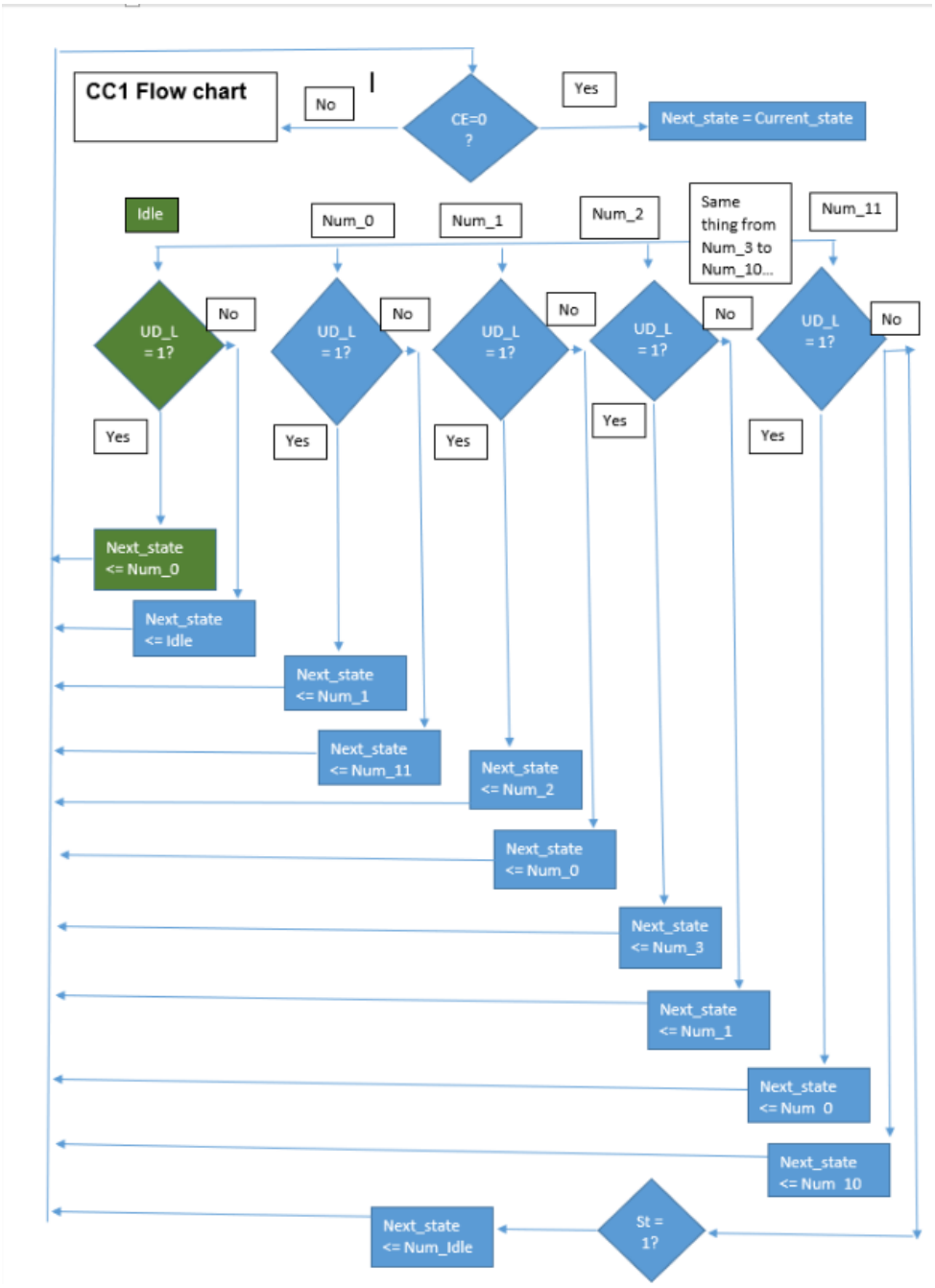


Figure vii

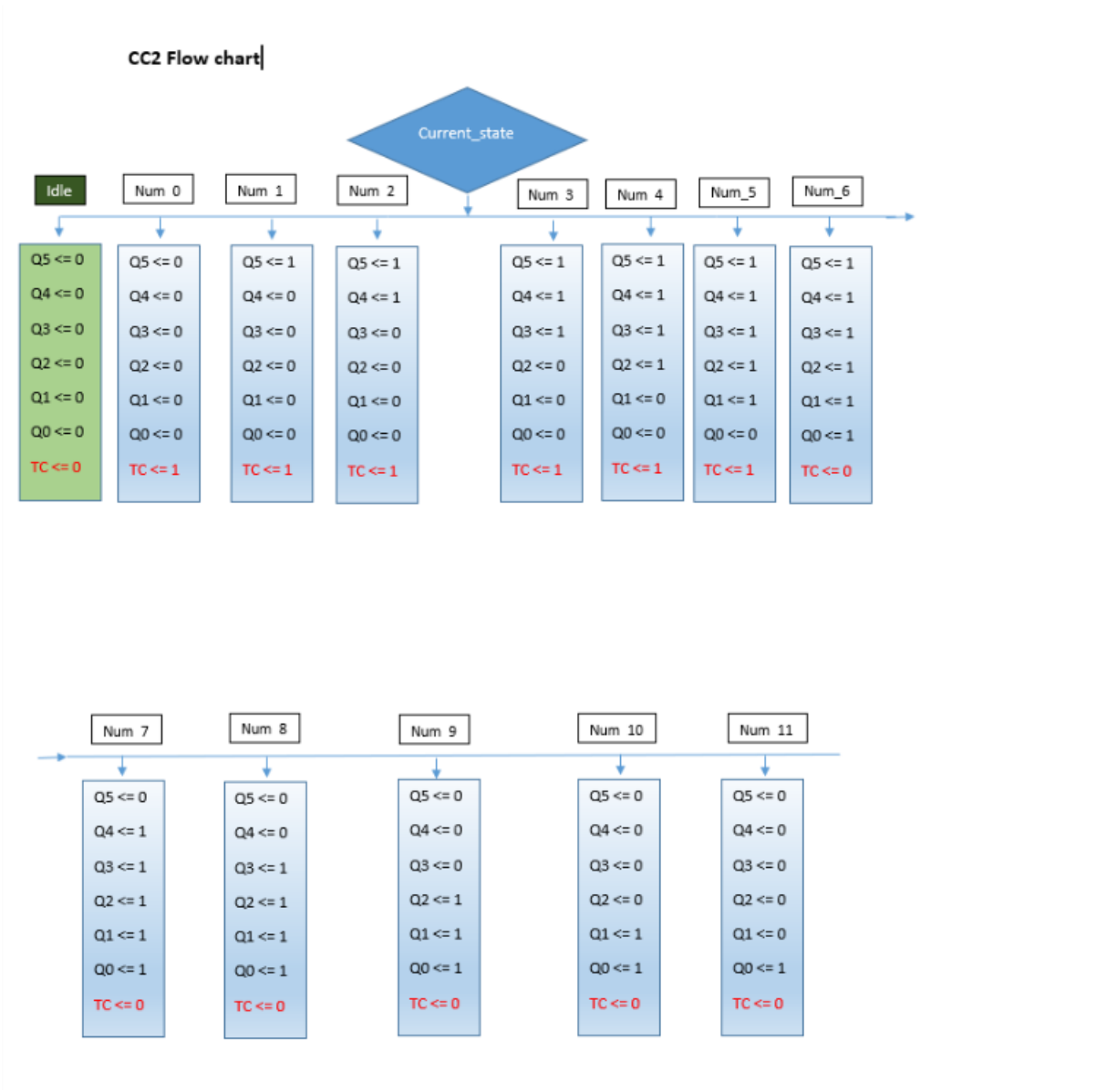

CC2 Flow chart
Current_state
Idle
Num 0
Num 1
Num 2
Num 3
Num 4
Num_5
Num_6
Q5 <= 0
Q4 <= 0
Q3 <= 0
Q2 <= 0
Q1 <= 0
Q0 <= 0
TC <= 0
Q5 <= 0
Q4 <= 0
Q3 <= 0
Q2 <= 0
Q1 <= 0
Q0 <= 0
TC <= 1
Q5 <= 1
Q4 <= 0
Q3 <= 0
Q2 <= 0
Q1 <= 0
Q0 <= 0
TC <= 1
Q5 <= 1
Q4 <= 1
Q3 <= 0
Q2 <= 0
Q1 <= 0
Q0 <= 0
TC <= 1
Q5 <= 1
Q4 <= 1
Q3 <= 1
Q2 <= 0
Q1 <= 0
Q0 <= 0
TC <= 1
Q5 <= 1
Q4 <= 1
Q3 <= 1
Q2 <= 1
Q1 <= 0
Q0 <= 0
TC <= 1
Q5 <= 1
Q4 <= 1
Q3 <= 1
Q2 <= 1
Q1 <= 1
Q0 <= 0
TC <= 1
Q5 <= 1
Q4 <= 1
Q3 <= 1
Q2 <= 1
Q1 <= 1
Q0 <= 1
TC <= 0
Num 7
Num 8
Num 9
Num 10
Num 11
Q5 <= 0
Q4 <= 1
Q3 <= 1
Q2 <= 1
Q1 <= 1
Q0 <= 1
TC <= 0
Q5 <= 0
Q4 <= 0
Q3 <= 1
Q2 <= 1
Q1 <= 1
Q0 <= 1
TC <= 0
Q5 <= 0
Q4 <= 0
Q3 <= 0
Q2 <= 1
Q1 <= 1
Q0 <= 1
TC <= 0
Q5 <= 0
Q4 <= 0
Q3 <= 0
Q2 <= 0
Q1 <= 1
Q0 <= 1
TC <= 0
Q5 <= 0
Q4 <= 0
Q3 <= 0
Q2 <= 0
Q1 <= 0
Q0 <= 1
TC <= 0

Figure e

2.5 Design phases

To obtain the final phase we first had to pass through three phases.

Phase 1:

1. Place all files in the appropriate folders and directories.
2. Organize the hardware by defining which pins on our chip are going to be used for CE, UD, Q(0..5), etc.
3. Organize the C code.
4. Test.

Phase 2:

1. Add pins ST_SP and LCD to the hardware.
2. Add ST_SP and LCD to the C code (with the lcd also and its libraries).
3. Test.

Phase 3:

1. Add pins TMR2_ON and TMR2_wave to the hardware.
2. Add TMR2_ON and TMR2_wave to the C code.
3. Test.

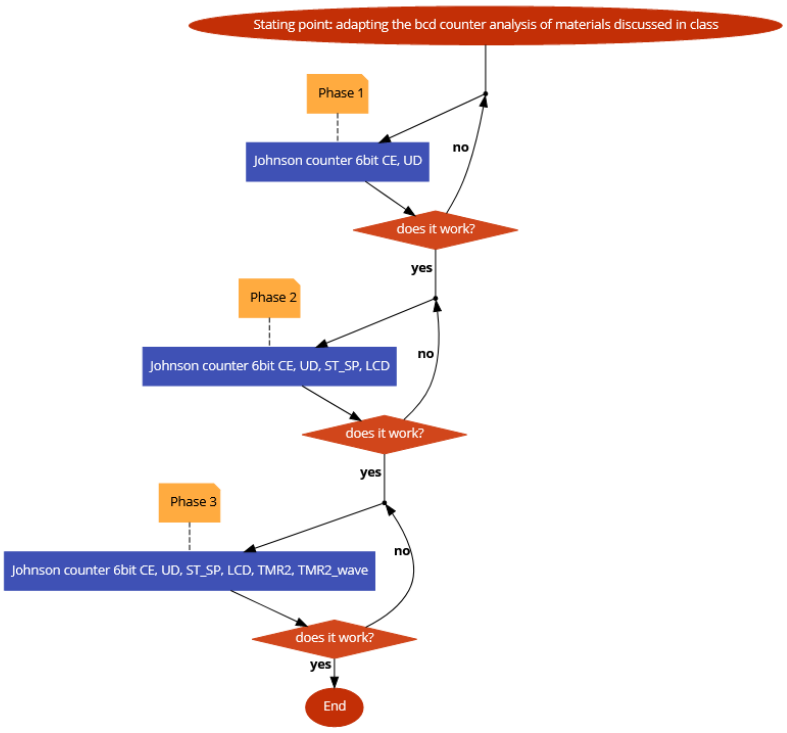


Figure f

3 Development

3.1 Phase 1

```
/*
================================================================================
 UPC - EETAC - CSD - http://digsys.upc.edu
Phase 1
An example on how to implement a sequential system like an 1-digit BCD counter
using "current_state" state-type variable.
Thus, this example can be used to adapt other FSM containing a small number of states.

 * Print in color this file, and analyse it and use it as 'theory' class notes.
================================================================================
 */

// This general header file includes the device specific headers like
// "pic18f4520.h". This file contains all the microcontroller declarations.
#include <xc.h>

/*
================================================================================
    Definitions

    ================================================================================
     */
// Our counter has 13 states defined as follows:
#define     Num_0           '0' //States enumeration. "Char"
#define     Num_1           '1'
#define     Num_2           '2'
#define     Num_3           '3'
#define     Num_4           '4'
#define     Num_5           '5'
#define     Num_6           '6'
#define     Num_7           '7'
#define     Num_8           '8'
#define     Num_9           '9'
#define     Num_10          'A'
#define     Num_11          'B'


/*
================================================================================
    Function prototypes

    ================================================================================
     */
// The three basic functions to organise the code to emulate a FSM
void init_system (void);
char state_logic(void);
char output_logic(void);

//The interrupt service routine (ISR) to detect the Var_CLK_Flag (external INT0)
// or the ST_SP signal (RB interrupt, or the TMR0 interrupt
static void interrupt ISR(void);

// The auxiliary functions to read/poll and write port pins and be able to
// organise the software variables, to make the main program hardware
```

```
48  // independent and compatible for many microcontrollers.
49  void write_outputs (void);
50  void read_inputs(void);
51
52  /*
    ================================================================================
53      Global variables
54
        ================================================================================
         */
55  // Even if some variables are 'bit' type, we use 'char' (byte) to make them
    easy
56  // to watch while debugging.
57  static char Var_CE;
58  static char Var_UD_L;
59  static char Var_TC12;
60  static char Var_Q;
61
62  //The flag that will be set from the interrupt service routine to indicate
    that
63  // a new CLK_Flag has arrived
64  static char Var_CLK_Flag;
65
66  static char current_state = Num_0;
67
68
69  /*
    ================================================================================
70      Main function
71
        ================================================================================
         */
72  void main(void){
73      init_system ();
74      while(1) {                      /* loop forever the FSM */
75          if (state_logic()) break;
76          if (output_logic()) break;
77      }
78      while (1){
79          __nop();//The system has crashed in an illegal state and needs
            attention
80                  //This second infinite loop is never reached in normal
                    operation
81      }           // The only way to scape is clicking reset (CD)
82  }
83
84  /*
    ================================================================================
85      Function definitions
86
        ================================================================================
         */
87
88  //*****************************************************************************
    **
89  //State variable logic routine  (The CC1)
90  //*****************************************************************************
    **
```

```
91  char state_logic(void){
92      char error = 0;
93
94      read_inputs(); //sample input values so that the CC1 can decide
        transitions
95
96
97      if (Var_CLK_Flag == 1){
98
99          switch (current_state) {
100             case Num_0:
101                 if((Var_CE == 1) && (Var_UD_L == 0)){
102                     current_state = Num_11;
103                 }
104                 else if((Var_CE == 1) && (Var_UD_L == 1)){
105                     current_state = Num_1;
106                 }
107                 else{
108                     current_state = Num_0;
109                 }
110             break;
111
112             case Num_1:
113                 if ((Var_CE == 1) && (Var_UD_L == 1)){
114                     current_state = Num_2;
115                 }
116                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
117                     current_state = Num_0;
118                 }else{
119                     current_state = Num_1;
120                 }
121             break;
122
123             case Num_2:
124                 if ((Var_CE == 1) && (Var_UD_L == 1)){
125                     current_state = Num_3;
126                 }
127                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
128                     current_state = Num_1;
129
130                 }else{
131                     current_state = Num_2;
132                 }
133             break;
134
135             case Num_3:
136                 if ((Var_CE == 1) && (Var_UD_L == 1)){
137                     current_state = Num_4;
138                 }
139                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
140                     current_state = Num_2;
141                 }else{
142                     current_state = Num_3;
143                 }
144             break;
145             case Num_4:
146                 if ((Var_CE == 1) && (Var_UD_L == 1)){
147                     current_state = Num_5;
```

```
148                 }
149                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
150                     current_state = Num_3;
151                 }else{
152                     current_state = Num_4;
153                 }
154             break;
155             case Num_5:
156                 if ((Var_CE == 1) && (Var_UD_L == 1)){
157                     current_state = Num_6;
158                 }
159                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
160                     current_state = Num_4;
161                 }else{
162                     current_state = Num_5;
163                 }
164             break;
165             case Num_6:
166                 if ((Var_CE == 1) && (Var_UD_L == 1)){
167                     current_state = Num_7;
168                 }
169                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
170                     current_state = Num_5;
171                 }else{
172                     current_state = Num_6;
173                 }
174             break;
175             case Num_7:
176                 if ((Var_CE == 1) && (Var_UD_L == 1)){
177                     current_state = Num_8;
178                 }
179                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
180                     current_state = Num_6;
181                 }else{
182                     current_state = Num_7;
183                 }
184             break;
185             case Num_8:
186                 if ((Var_CE == 1) && (Var_UD_L == 1)){
187                     current_state = Num_9;
188                 }
189                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
190                     current_state = Num_7;
191                 }else{
192                     current_state = Num_8;
193                 }
194             break;
195             case Num_9:
196                 if ((Var_CE == 1) && (Var_UD_L == 1)){
197                     current_state = Num_10;
198                 }
199                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
200                     current_state = Num_8;
201                 }else{
202                     current_state = Num_9;
203                 }
204             break;
205             case Num_10:
```

```
206                 if ((Var_CE == 1) && (Var_UD_L == 1)){
207                     current_state = Num_11;
208                 }
209                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
210                     current_state = Num_9;
211                 }else{
212                     current_state = Num_10;
213                 }
214             break;
215             case Num_11:
216                  if((Var_CE == 1) && (Var_UD_L == 1)){
217                     current_state = Num_0;
218                 }
219                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
220                     current_state = Num_10;
221                 }else{
222                     current_state = Num_11;
223                 }
224             break;
225 
226             default:
227             // If no states have been selected, there is an error
228             // and the systems must be stopped
229                 error = 1;
230                 break;
231         }
232         Var_CLK_Flag = 0;
233     }
234 
235     return (error);
236 }
237 
238 //*****************************************************************************
    **
239 //Output logic routine (The combinational circuit CC2)
240 //*****************************************************************************
    **
241 char output_logic(void){
242 unsigned char error = 0;
243     switch (current_state) {
244         case Num_0:
245             Var_Q = 0;//0
246             Var_TC12 = 1;
247             break;
248         case Num_1:
249             Var_Q = 0b00100000;//1
250             Var_TC12 = 1;
251             break;
252         case Num_2:
253             Var_Q = 0b00110000;//2
254             Var_TC12 = 1;
255             break;
256         case Num_3:
257             Var_Q = 0b00111000;//3
258             Var_TC12 = 1;
259             break;
260         case Num_4:
261             Var_Q = 0b00111100;//4
```

```
262             Var_TC12 = 1;
263             break;
264         case Num_5:
265             Var_Q = 0b00111110;//5
266             Var_TC12 = 1;
267             break;
268         case Num_6:
269             Var_Q = 0b00111111;//6
270             Var_TC12 = 0;
271             break;
272         case Num_7:
273             Var_Q = 0b00011111;//7
274             Var_TC12 = 0;
275             break;
276         case Num_8:
277             Var_Q = 0b00001111;//8
278             Var_TC12 = 0;
279             break;
280         case Num_9:
281             Var_Q = 0b00000111;//9
282             Var_TC12 = 0;
283             break;
284         case Num_10:
285             Var_Q = 0b00000011;//A
286             Var_TC12 = 0;
287             break;
288         case Num_11:
289             Var_Q = 0b00000001;//B
290             Var_TC12 = 0;
291             break;
292         default:
293             error = 1;  // If no states have been selected, there is an error
294                         // and the systems must be stopped
295             break;
296     }
297
298     write_outputs(); // The interface with the hardware to convert variables
299                      // into voltages at a given microcontroller pin.
300
301     return (error);
302
303 }
304
305 /*
    ================================================================================
306  Scan inputs and process the bits so that we can set our convenient
    variables
307  many of the variables are "char" because we can easily monitor them using
308  the watch window for debugging purposes.
309
      ================================================================================
       */
310 void read_inputs (void){
311
312 // Like in P9, read the port, mask the bit of interest and set the variable
313     Var_CE = (PORTA & 0b00000001);
314     Var_UD_L = (PORTA & 0b00000010)>>1;
315 }
```

```
316
317 /*
    ================================================================================
318     Driving the output pins. Let's drive the output ports, converting
319     microcontroller variables into electrical signals.
320
    ================================================================================
    */
321 void write_outputs(void){
322
323     //To write a given output pin:
324     PORTC = Var_Q | (Var_TC12 << 6);
325 }
326
327 //*****************************************************************************
328 //Interrupt Service Routine
329 //*****************************************************************************
330
331 static void interrupt ISR (void){   //"interrupt" is the key word here
332     GIE = 0;        //Disable interrupts while attending one of them ...
333     if(INT0IF == 1) {       // Is it an INT0 interrupt ?  PHASE 1
334     //GIE = 0;
335 // If the code execution reach this point is because a falling edge signal
336 // in RB0 has been detected, so let's set the flag and the FSM will
337 // acknowledge it
338         Var_CLK_Flag = 1;   //Set the "software flag"
339         INT0IF = 0; // clear the "hardware" INT0 interrupt flag
340         //GIE = 1;
341     }
342     GIE = 1;        //Enable interrupts
343 }
344
345 //*****************************************************************************
    **
346 // Initialise the microcontroller system
347 //*****************************************************************************
    **
348 void init_system (void){
349 // Configuration bits
350
351 // These are the configuration bits translated to the XC8 Compiler
352 // Microcontroller general configuration bits. You don't have to change them.
353 // PIC18F4520 Configuration Bit Settings. They can be generated
    automatically
354 // in the MPLAB X IDE
355
356 // CONFIG1H
357 #pragma config OSC = XT       // Oscillator Section bits (XT oscillator)
358 #pragma config FCMEN = OFF
359 // Fail-Safe Clock Monitor E bit (Fail-Safe Clock Monitor disabled)
360 #pragma config IESO = OFF
361 // Internal/External Osc Switch-over bit (Oscillator Switch-over mode
    disabled)
362
363 // CONFIG2L
364 #pragma config PWRT = ON       // Power-up Timer E bit (PWRT enabled)
365 #pragma config BOREN = SBORDIS
366 // Brown-out Reset E bits (Brown-out Reset enabled in hardware
```

```
367   // only (SBOREN is disabled))
368   #pragma config BORV = 3    // Brown Out Reset Voltage bits (Minimum setting)
369
370   // CONFIG2H
371   #pragma config WDT = OFF
372   // Watchdog Timer E bit (WDT disabled (control is on the SWDTEN bit))
373   #pragma config WDTPS = 32768
374   // Watchdog Timer Postscale Sect bits (1:32768)
375
376   // CONFIG3H
377   #pragma config CCP2MX = PORTC
378   // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
379   #pragma config PBADEN = OFF
380   // PORTB A/D E bit (PORTB<4:0> pins are configured
381   // as digital I/O on Reset)
382   #pragma config LPT1OSC = OFF
383   // Low-Power Timer1 Oscillator E bit (Timer1 configured for
384   // higher power operation)
385   #pragma config MCLRE = ON
386   // MCLR Pin E bit (MCLR pin enabled; RE3 input pin disabled)
387
388   // CONFIG4L
389   #pragma config STVREN = ON
390   // Stack Full/Underflow Reset E bit (Stack full/underflow will cause Reset)
391   #pragma config LVP = OFF
392   // Single-Supply ICSP E bit (Single-Supply ICSP disabled)
393   #pragma config XINST = OFF
394   // Extended Instruction Set E bit (Instruction set extension and Indexed
395   // Addressing mode disabled (Legacy mode))
396
397   // CONFIG5L
398   #pragma config CP0 = OFF
399   // Code Protection bit (Block 0 (000800-001FFFh) not code-protected)
400   #pragma config CP1 = OFF
401   // Code Protection bit (Block 1 (002000-003FFFh) not code-protected)
402   #pragma config CP2 = OFF
403   // Code Protection bit (Block 2 (004000-005FFFh) not code-protected)
404   #pragma config CP3 = OFF
405   // Code Protection bit (Block 3 (006000-007FFFh) not code-protected)
406
407   // CONFIG5H
408   #pragma config CPB = OFF
409   // Boot Block Code Protection bit
410   // (Boot block (000000-0007FFh) not code-protected)
411   #pragma config CPD = OFF
412   // Data EEPROM Code Protection bit (Data EEPROM not code-protected)
413
414   // CONFIG6L
415   #pragma config WRT0 = OFF
416   // Write Protection bit (Block 0 (000800-001FFFh) not write-protected)
417   #pragma config WRT1 = OFF
418   // Write Protection bit (Block 1 (002000-003FFFh) not write-protected)
419   #pragma config WRT2 = OFF
420   // Write Protection bit (Block 2 (004000-005FFFh) not write-protected)
421
422   #pragma config WRT3 = OFF
423   // Write Protection bit (Block 3 (006000-007FFFh) not write-protected)
424
```

```
425  // CONFIG6H
426  #pragma config WRTC = OFF
427  // Configuration Register Write Protection bit (Configuration registers
428  // (300000-3000FFh) not write-protected)
429  #pragma config WRTB = OFF
430  // Boot Block Write Protection bit (Boot block (000000-0007FFh)
431  // not write-protected)
432  #pragma config WRTD = OFF
433  // Data EEPROM Write Protection bit (Data EEPROM not write-protected)
434
435  // CONFIG7L
436  #pragma config EBTR0 = OFF
437  // Table Read Protection bit (Block 0 (000800-001FFFh) not protected from
438  // table reads executed in other blocks)
439  #pragma config EBTR1 = OFF
440  // Table Read Protection bit (Block 1 (002000-003FFFh) not protected from
441  // table reads executed in other blocks)
442  #pragma config EBTR2 = OFF
443  // Table Read Protection bit (Block 2 (004000-005FFFh) not protected from
     table
444  // reads executed in other blocks)
445  #pragma config EBTR3 = OFF
446  // Table Read Protection bit (Block 3 (006000-007FFFh) not protected from
     table
447  // reads executed in other blocks)
448
449  // CONFIG7H
450  #pragma config EBTRB = OFF
451  // Boot Block Table Read Protection bit (Boot block (000000-0007FFh)
452  // not protected from table reads executed in other blocks)
453
454      /* -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+*/
455  // Initialise starts here:
456
457  /* All the PORTA, PORTB and PORTE is set to be digital I/O (The can be
458       * defined as analogue inputs*/
459      ADCON1  = 0x0F; // read the datasheet chapter on the A/D converter
460  // All unused pins are defined as outputs and reset
461
462      PORTA   = 0x00;         // Reset all Flip-Flops at PORTB
463      LATA    = 0x00;
464      TRISA   = 0b0000011;    // PORTB1 and PORTB0 are inputs; PORTB2 is
       output
465
466      PORTB   = 0x00;         // Reset all Flip-Flops at PORTB
467      LATB    = 0x00;
468      TRISB   = 0b0000001;
469
470      PORTC   = 0x00;         // Reset all Flip-Flops at PORTD
471      LATC    = 0x00;
472      TRISC   = 0b00000000;   // PORTD(7..4) are outputs for the code
473
474      PORTD   = 0x00;         // Reset all Flip-Flops at PORTD
475      LATD    = 0x00;
476      TRISD   = 0b00000000;
477                              // the other bits are also defined as outputs
478
479  // INT0 edge selection: '0' --> on falling edge
```

```
480   // (interrupt after clicking the pushbutton)
481       INTEDG0 = 0;
482       INT0IE  = 1; // Enable INT0 interrupts
483
484       PEIE = 0;   // Disable all the other sources of interrupt; PHASE 1 and 2
485       GIE = 1;        // Global interrupts allowed
486
487       Var_CLK_Flag = 0;   // Reset the flag to detect an interrupt INT
488   }
```

3.2 Phase 2

```
/*
================================================================================
 UPC - EETAC - CSD - http://digsys.upc.edu
Phase 2
An example on how to implement a sequential system like an 1-digit BCD counter
using "current_state" state-type variable.
Thus, this example can be used to adapt other FSM containing a small number of states.

 * Print in color this file, and analyse it and use it as 'theory' class notes.
================================================================================
 */

// This general header file includes the device specific headers like
// "pic18f4520.h". This file contains all the microcontroller declarations.
#include <xc.h>
#include <stdio.h>
#include "lcd.h"
/*
================================================================================
    Definitions

    ================================================================================
     */
// This structure allows to access to a given bit into an 8-bit register
#define     PORTBIT(adr, bit) ( (unsigned)(&adr)*8 + (bit) )
// Our counter has 13 states defined as follows:
#define     Idle            'I' //States enumeration. "Char"
#define     Num_0           '0'
#define     Num_1           '1'
#define     Num_2           '2'
#define     Num_3           '3'
#define     Num_4           '4'
#define     Num_5           '5'
#define     Num_6           '6'
#define     Num_7           '7'
#define     Num_8           '8'
#define     Num_9           '9'
#define     Num_10          'A'
#define     Num_11          'B'



/*
================================================================================
    Function prototypes

    ================================================================================
     */
// The three basic functions to organise the code to emulate a FSM
void init_system (void);
char state_logic(void);
char output_logic(void);

//The interrupt service routine (ISR) to detect the Var_CLK_Flag (external INT0)
```

```
48  // or the ST_SP signal (RB interrupt, or the TMR0 interrupt
49  static void interrupt ISR(void);
50
51  // The auxiliary functions to read/poll and write port pins and be able to
52  // organise the software variables, to make the main program hardware
53  // independent and compatible for many microcontrollers.
54  void write_outputs (void);
55  void read_inputs(void);
56
57  /*
    ================================================================================
58      Global variables
59
        ================================================================================
         */
60  // Even if some variables are 'bit' type, we use 'char' (byte) to make them
    easy
61  // to watch while debugging.
62  static char Var_CE;
63  static char Var_UD_L;
64  static char Var_TC12;
65  static char Var_Q;
66
67  //The flag that will be set from the interrupt service routine to indicate
    that
68  // a new CLK_Flag has arrived or when ST_SP button is pushed or when the
    has to
69  //write something new
70  static char Var_CLK_Flag;
71  static char St_Sp_flag;
72  static char LCD_Flag;
73
74  static char current_state = Idle;   //initiolize the current state
75
76  /*
    ================================================================================
77      Main function
78
        ================================================================================
         */
79  void main(void){
80      init_system ();
81      while(1) {                      /* loop forever the FSM */
82          if (state_logic()) break;
83          if (output_logic()) break;
84      }
85      while (1){
86          __nop();//The system has crashed in an illegal state and needs
            attention
87                  //This second infinite loop is never reached in normal
                    operation
88      }           // The only way to scape is clicking reset (CD)
89  }
90
91  /*
    ================================================================================
92      Function definitions
93
```

```
    ================================================================================
     */
94
95  //*****************************************************************************
    **
96  //State variable logic routine  (The CC1)
97  //*****************************************************************************
    **
98  char state_logic(void){
99      char error = 0;
100
101     read_inputs(); //sample input values so that the CC1 can decide
        transitions
102
103     if (Var_CLK_Flag == 1){
104
105         switch (current_state) {
106             case Idle:
107                 if((Var_CE == 1) && (Var_UD_L == 0) && (St_Sp_flag == 1)){
                    // if the ST_SP has been pushed
108                     current_state = Num_11;
                        //then it will perimit to change state
109                     St_Sp_flag=0;   //reset flag after operation
                        //depending in the direction (up or down)
110                     LCD_Flag=1;     //due to change of state we right
                        somthing new in the lcd
111                 }
112                 else if((Var_CE == 1) && (Var_UD_L == 1)&& (St_Sp_flag ==
                    1)) {
113                     current_state = Num_0;
114                     St_Sp_flag=0;
115                     LCD_Flag=1;
116                 }else{
117                     current_state = Idle;
118                 }
119             break;
120             case Num_0:
121                 if((Var_CE == 1) && (Var_UD_L == 0) && (St_Sp_flag == 1)){
122                     current_state = Idle;
123                     St_Sp_flag=0;
124                     LCD_Flag=1;
125                 }
126                 else if ((Var_CE == 1) && (Var_UD_L == 1)){
127                     current_state = Num_1;
128                     LCD_Flag=1;
129                 }
130                 else if((Var_CE == 1) && (Var_UD_L == 0) && (St_Sp_flag ==
                    0)) {
131                     current_state = Num_11;
132                     LCD_Flag=1;
133
134                 }else{
135                     current_state = Num_0;
136                 }
137             break;
138
139             case Num_1:
140                 if ((Var_CE == 1) && (Var_UD_L == 1)){
```

```
141                     current_state = Num_2;
142                     LCD_Flag=1;
143                 }
144                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
145                     current_state = Num_0;
146                     LCD_Flag=1;
147                 }else{
148                     current_state = Num_1;
149                 }
150             break;
151 
152             case Num_2:
153                 if ((Var_CE == 1) && (Var_UD_L == 1)){
154                     current_state = Num_3;
155                     LCD_Flag=1;
156                 }
157                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
158                     current_state = Num_1;
159                     LCD_Flag=1;
160                 }else{
161                     current_state = Num_2;
162                 }
163             break;
164 
165             case Num_3:
166                 if ((Var_CE == 1) && (Var_UD_L == 1)){
167                     current_state = Num_4;
168                     LCD_Flag=1;
169                 }
170                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
171                     current_state = Num_2;
172                     LCD_Flag=1;
173                 }else{
174                     current_state = Num_3;
175                 }
176             break;
177             case Num_4:
178                 if ((Var_CE == 1) && (Var_UD_L == 1)){
179                     current_state = Num_5;
180                     LCD_Flag=1;
181                 }
182                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
183                     current_state = Num_3;
184                     LCD_Flag=1;
185                 }else{
186                     current_state = Num_4;
187                 }
188             break;
189             case Num_5:
190                 if ((Var_CE == 1) && (Var_UD_L == 1)){
191                     current_state = Num_6;
192                     LCD_Flag=1;
193                 }
194                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
195                     current_state = Num_4;
196                     LCD_Flag=1;
197                 }else{
198                     current_state = Num_5;
```

```
199                 }
200             break;
201             case Num_6:
202                 if ((Var_CE == 1) && (Var_UD_L == 1)){
203                     current_state = Num_7;
204                     LCD_Flag=1;
205                 }
206                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
207                     current_state = Num_5;
208                     LCD_Flag=1;
209                 }else{
210                     current_state = Num_6;
211                 }
212             break;
213             case Num_7:
214                 if ((Var_CE == 1) && (Var_UD_L == 1)){
215                     current_state = Num_8;
216                     LCD_Flag=1;
217                 }
218                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
219                     current_state = Num_6;
220                     LCD_Flag=1;
221                 }else{
222                     current_state = Num_7;
223                 }
224             break;
225             case Num_8:
226                 if ((Var_CE == 1) && (Var_UD_L == 1)){
227                     current_state = Num_9;
228                     LCD_Flag=1;
229                 }
230                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
231                     current_state = Num_7;
232                     LCD_Flag=1;
233                 }else{
234                     current_state = Num_8;
235                 }
236             break;
237             case Num_9:
238                 if ((Var_CE == 1) && (Var_UD_L == 1)){
239                     current_state = Num_10;
240                     LCD_Flag=1;
241                 }
242                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
243                     current_state = Num_8;
244                     LCD_Flag=1;
245                 }else{
246                     current_state = Num_9;
247                 }
248             break;
249             case Num_10:
250                 if ((Var_CE == 1) && (Var_UD_L == 1)){
251                     current_state = Num_11;
252                     LCD_Flag=1;
253                 }
254                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
255                     current_state = Num_9;
256                     LCD_Flag=1;
```

```
257                 }else{
258                     current_state = Num_10;
259                 }
260             break;
261             case Num_11:
262                  if((Var_CE == 1) && (Var_UD_L == 1) && (St_Sp_flag == 1)){
263                     current_state = Idle;
264                     St_Sp_flag=0;
265                     LCD_Flag=1;
266                 }
267                  else if ((Var_CE == 1) && (Var_UD_L == 1) && (St_Sp_flag
                    == 0)){
268                     current_state = Num_0;
269                     LCD_Flag=1;
270                 }
271                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
272                     current_state = Num_10;
273                     LCD_Flag=1;
274                 }else{
275                     current_state = Num_11;
276                 }
277             break;
278 
279             default:
280             // If no states have been selected, there is an error
281             // and the systems must be stopped
282                 error = 1;
283                 break;
284         }
285         Var_CLK_Flag=0; // Clear the Var_CLK_Flag detection flag
286     }
287     return (error);
288 }
289 
290 //*************************************************************************
    **
291 //Output logic routine (The combinational circuit CC2)
292 //*************************************************************************
    **
293 char output_logic(void){
294 unsigned char error = 0;
295     switch (current_state) {
296         case Idle:
297             Var_Q = 0;//idle
298             Var_TC12 = 0;
299              if (LCD_Flag == 1) {           //write only if flag activates
300                 lcd_clear();
301                 lcd_home();                     // select line 1
302                 lcd_puts("IDLE mode ...");
303                 lcd_home2();            // select line 2 to write the result
304                 lcd_puts(" ... push ST_SP ...");
305                 }
306             LCD_Flag = 0;
307             break;
308         case Num_0:
309             Var_Q = 0;//0
310             Var_TC12 = 1;
311             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {   //depending if counting
```

```
            up`or down
312             lcd_clear();                        //the lcd will give a
                different output
313             lcd_home();
314             lcd_puts("0 ...");
315             lcd_home2();
316             lcd_puts(" ... count up ...");
317             }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
318             lcd_clear();
319             lcd_home();
320             lcd_puts("0 ...");
321             lcd_home2();
322             lcd_puts(" ... count down ...");
323             }
324         LCD_Flag = 0;
325         break;
326       case Num_1:
327         Var_Q = 0b00100000;//1
328         Var_TC12 = 1;
329         if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
330             lcd_clear();
331             lcd_home();
332             lcd_puts("1 ...");
333             lcd_home2();
334             lcd_puts(" ... count up ...");
335             }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
336             lcd_clear();
337             lcd_home();
338             lcd_puts("1 ...");
339             lcd_home2();
340             lcd_puts(" ... count down ...");
341             }
342         LCD_Flag = 0;
343         break;
344       case Num_2:
345         Var_Q = 0b00110000;//2
346         Var_TC12 = 1;
347         if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
348             lcd_clear();
349             lcd_home();
350             lcd_puts("2 ...");
351             lcd_home2();
352             lcd_puts(" ... count up ...");
353             }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
354             lcd_clear();
355             lcd_home();
356             lcd_puts("2 ...");
357             lcd_home2();
358             lcd_puts(" ... count down ...");
359             }
360         LCD_Flag = 0;
361         break;
362       case Num_3:
363         Var_Q = 0b00111000;//3
364         Var_TC12 = 1;
365         if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
366             lcd_clear();
367             lcd_home();
```

```
368                 lcd_puts("3 ...");
369                 lcd_home2();
370                 lcd_puts(" ... count up ...");
371                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
372                 lcd_clear();
373                 lcd_home();
374                 lcd_puts("3 ...");
375                 lcd_home2();
376                 lcd_puts(" ... count down ...");
377                 }
378             LCD_Flag = 0;
379             break;
380         case Num_4:
381             Var_Q = 0b00111100;//4
382             Var_TC12 = 1;
383             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
384                 lcd_clear();
385                 lcd_home();
386                 lcd_puts("4 ...");
387                 lcd_home2();
388                 lcd_puts(" ... count up ...");
389                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
390                 lcd_clear();
391                 lcd_home();
392                 lcd_puts("4 ...");
393                 lcd_home2();
394                 lcd_puts(" ... count down ...");
395                 }
396             LCD_Flag = 0;
397             break;
398         case Num_5:
399             Var_Q = 0b00111110;//5
400             Var_TC12 = 1;
401             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
402                 lcd_clear();
403                 lcd_home();
404                 lcd_puts("5 ...");
405                 lcd_home2();
406                 lcd_puts(" ... count up ...");
407                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
408                 lcd_clear();
409                 lcd_home();
410                 lcd_puts("5 ...");
411                 lcd_home2();
412                 lcd_puts(" ... count down ...");
413                 }
414             LCD_Flag = 0;
415             break;
416         case Num_6:
417             Var_Q = 0b00111111;//6
418             Var_TC12 = 0;
419             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
420                 lcd_clear();
421                 lcd_home();
422                 lcd_puts("6 ...");
423                 lcd_home2();
424                 lcd_puts(" ... count up ...");
425                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
```

```
426                 lcd_clear();
427                 lcd_home();
428                 lcd_puts("6 ...");
429                 lcd_home2();
430                 lcd_puts(" ... count down ...");
431                 }
432             LCD_Flag = 0;
433             break;
434         case Num_7:
435             Var_Q = 0b00011111;//7
436             Var_TC12 = 0;
437             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
438                 lcd_clear();
439                 lcd_home();
440                 lcd_puts("7 ...");
441                 lcd_home2();
442                 lcd_puts(" ... count up ...");
443                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
444                 lcd_clear();
445                 lcd_home();
446                 lcd_puts("7 ...");
447                 lcd_home2();
448                 lcd_puts(" ... count down ...");
449                 }
450             LCD_Flag = 0;
451             break;
452         case Num_8:
453             Var_Q = 0b00001111;//8
454             Var_TC12 = 0;
455             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
456                 lcd_clear();
457                 lcd_home();
458                 lcd_puts("8 ...");
459                 lcd_home2();
460                 lcd_puts(" ... count up ...");
461                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
462                 lcd_clear();
463                 lcd_home();
464                 lcd_puts("8 ...");
465                 lcd_home2();
466                 lcd_puts(" ... count down ...");
467                 }
468             LCD_Flag = 0;
469             break;
470         case Num_9:
471             Var_Q = 0b00000111;//9
472             Var_TC12 = 0;
473             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
474                 lcd_clear();
475                 lcd_home();
476                 lcd_puts("9 ...");
477                 lcd_home2();
478                 lcd_puts(" ... count up ...");
479                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
480                 lcd_clear();
481                 lcd_home();
482                 lcd_puts("9 ...");
483                 lcd_home2();
```

```
484                 lcd_puts(" ... count down ...");
485                 }
486             LCD_Flag = 0;
487             break;
488         case Num_10:
489             Var_Q = 0b00000011;//A
490             Var_TC12 = 0;
491             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
492                 lcd_clear();
493                 lcd_home();
494                 lcd_puts("10 ...");
495                 lcd_home2();
496                 lcd_puts(" ... count up ...");
497                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
498                 lcd_clear();
499                 lcd_home();
500                 lcd_puts("10 ...");
501                 lcd_home2();
502                 lcd_puts(" ... count down ...");
503                 }
504             LCD_Flag = 0;
505             break;
506         case Num_11:
507             Var_Q = 0b00000001;//B
508             Var_TC12 = 0;
509             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
510                 lcd_clear();
511                 lcd_home();
512                 lcd_puts("11 ...");
513                 lcd_home2();
514                 lcd_puts(" ... count up ...");
515                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
516                 lcd_clear();
517                 lcd_home();
518                 lcd_puts("11 ...");
519                 lcd_home2();
520                 lcd_puts(" ... count down ...");
521                 }
522             LCD_Flag = 0;
523             break;
524         default:
525             error = 1;  // If no states have been selected, there is an error
526                         // and the systems must be stopped
527             break;
528     }
529
530     write_outputs(); // The interface with the hardware to convert variables
531                      // into voltages at a given microcontroller pin.
532
533     return (error);
534
535 }
536
537 //*****************************************************************************
538 //Interrupt Service Routine
539 //*****************************************************************************
540
541 static void interrupt ISR (void){   //"interrupt" is the key word here
```

```
542       GIE = 0;        //Disable interrupts while attending one of them ...
543       if(INT0IF == 1) {       // Is it an INT0 interrupt ?  PHASE 1
544       //GIE = 0;
545   // If the code execution reach this point is because a falling edge signal
546   // in RB0 has been detected, so let's set the flag and the FSM will
547   // acknowledge it
548           Var_CLK_Flag = 1;   //Set the "software flag"
549           INT0IF = 0; // clear the "hardware" INT0 interrupt flag
550           //GIE = 1;
551       }
552       if(INT1IF ==1){         // Is it an INT1 interrupt ?  PHASE 2
553
554           St_Sp_flag=1;
555           INT1IF =0;
556
557       }
558
559       GIE = 1;        //Enable interrupts
560   }
561
562
563   /*
      ================================================================================
564    Scan inputs and process the bits so that we can set our convenient
       variables
565    many of the variables are "char" because we can easily monitor them using
566    the watch window for debugging purposes.
567
        ================================================================================
        */
568   void read_inputs (void){
569
570   // Like in P9, read the port, mask the bit of interest and set the variable
571       Var_CE = (PORTA & 0b00000001);
572       Var_UD_L = (PORTA & 0b00000010)>>1;
573   }
574
575   /*
      ================================================================================
576       Driving the output pins. Let's drive the output ports, converting
577       microcontroller variables into electrical signals.
578
       ================================================================================
        */
579   void write_outputs(void){
580
581       //To write a given output pin:
582
583       PORTC = Var_Q | (Var_TC12 << 6);
584   }
585
586
587   //*************************************************************************
      **
588   // Initialise the microcontroller system
589   //*************************************************************************
      **
590   void init_system (void){
```

```
591  // Configuration bits
592
593  // These are the configuration bits translated to the XC8 Compiler
594  // Microcontroller general configuration bits. You don't have to change them.
595  // PIC18F4520 Configuration Bit Settings. They can be generated
     automatically
596  // in the MPLAB X IDE
597
598  // CONFIG1H
599  #pragma config OSC = XT       // Oscillator Section bits (XT oscillator)
600  #pragma config FCMEN = OFF
601  // Fail-Safe Clock Monitor E bit (Fail-Safe Clock Monitor disabled)
602  #pragma config IESO = OFF
603  // Internal/External Osc Switch-over bit (Oscillator Switch-over mode
     disabled)
604
605  // CONFIG2L
606  #pragma config PWRT = ON       // Power-up Timer E bit (PWRT enabled)
607  #pragma config BOREN = SBORDIS
608  // Brown-out Reset E bits (Brown-out Reset enabled in hardware
609  // only (SBOREN is disabled))
610  #pragma config BORV = 3    // Brown Out Reset Voltage bits (Minimum setting)
611
612  // CONFIG2H
613  #pragma config WDT = OFF
614  // Watchdog Timer E bit (WDT disabled (control is on the SWDTEN bit))
615  #pragma config WDTPS = 32768
616  // Watchdog Timer Postscale Sect bits (1:32768)
617
618  // CONFIG3H
619  #pragma config CCP2MX = PORTC
620  // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
621  #pragma config PBADEN = OFF
622  // PORTB A/D E bit (PORTB<4:0> pins are configured
623  // as digital I/O on Reset)
624  #pragma config LPT1OSC = OFF
625  // Low-Power Timer1 Oscillator E bit (Timer1 configured for
626  // higher power operation)
627  #pragma config MCLRE = ON
628  // MCLR Pin E bit (MCLR pin enabled; RE3 input pin disabled)
629
630  // CONFIG4L
631  #pragma config STVREN = ON
632  // Stack Full/Underflow Reset E bit (Stack full/underflow will cause Reset)
633  #pragma config LVP = OFF
634  // Single-Supply ICSP E bit (Single-Supply ICSP disabled)
635  #pragma config XINST = OFF
636  // Extended Instruction Set E bit (Instruction set extension and Indexed
637  // Addressing mode disabled (Legacy mode))
638
639  // CONFIG5L
640  #pragma config CP0 = OFF
641  // Code Protection bit (Block 0 (000800-001FFFh) not code-protected)
642  #pragma config CP1 = OFF
643  // Code Protection bit (Block 1 (002000-003FFFh) not code-protected)
644  #pragma config CP2 = OFF
645  // Code Protection bit (Block 2 (004000-005FFFh) not code-protected)
646  #pragma config CP3 = OFF
```

```
647  // Code Protection bit (Block 3 (006000-007FFFh) not code-protected)
648
649  // CONFIG5H
650  #pragma config CPB = OFF
651  // Boot Block Code Protection bit
652  // (Boot block (000000-0007FFh) not code-protected)
653  #pragma config CPD = OFF
654  // Data EEPROM Code Protection bit (Data EEPROM not code-protected)
655
656  // CONFIG6L
657  #pragma config WRT0 = OFF
658  // Write Protection bit (Block 0 (000800-001FFFh) not write-protected)
659  #pragma config WRT1 = OFF
660  // Write Protection bit (Block 1 (002000-003FFFh) not write-protected)
661  #pragma config WRT2 = OFF
662  // Write Protection bit (Block 2 (004000-005FFFh) not write-protected)
663
664  #pragma config WRT3 = OFF
665  // Write Protection bit (Block 3 (006000-007FFFh) not write-protected)
666
667  // CONFIG6H
668  #pragma config WRTC = OFF
669  // Configuration Register Write Protection bit (Configuration registers
670  // (300000-3000FFh) not write-protected)
671  #pragma config WRTB = OFF
672  // Boot Block Write Protection bit (Boot block (000000-0007FFh)
673  // not write-protected)
674  #pragma config WRTD = OFF
675  // Data EEPROM Write Protection bit (Data EEPROM not write-protected)
676
677  // CONFIG7L
678  #pragma config EBTR0 = OFF
679  // Table Read Protection bit (Block 0 (000800-001FFFh) not protected from
680  // table reads executed in other blocks)
681  #pragma config EBTR1 = OFF
682  // Table Read Protection bit (Block 1 (002000-003FFFh) not protected from
683  // table reads executed in other blocks)
684  #pragma config EBTR2 = OFF
685  // Table Read Protection bit (Block 2 (004000-005FFFh) not protected from
     table
686  // reads executed in other blocks)
687  #pragma config EBTR3 = OFF
688  // Table Read Protection bit (Block 3 (006000-007FFFh) not protected from
     table
689  // reads executed in other blocks)
690
691  // CONFIG7H
692  #pragma config EBTRB = OFF
693  // Boot Block Table Read Protection bit (Boot block (000000-0007FFh)
694  // not protected from table reads executed in other blocks)
695
696      /* -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+*/
697  // Initialise starts here:
698
699  /* All the PORTA, PORTB and PORTE is set to be digital I/O (The can be
700       * defined as analogue inputs*/
701      ADCON1  = 0x0F; // read the datasheet chapter on the A/D converter
702  // All unused pins are defined as outputs and reset
```

```
703
704       PORTA   = 0x00;         // Reset all Flip-Flops at PORTB
705       LATA    = 0x00;
706       TRISA   = 0b0000011;    // PORTB1 and PORTB0 are inputs; PORTB2 is
          output
707
708       PORTB   = 0x00;         // Reset all Flip-Flops at PORTB
709       LATB    = 0x00;
710       TRISB   = 0b0000011;
711
712       PORTC   = 0x00;         // Reset all Flip-Flops at PORTD
713       LATC    = 0x00;
714       TRISC   = 0b00000000;   // PORTD(7..4) are outputs for the code
715
716       PORTD   = 0x00;         // Reset all Flip-Flops at PORTD
717       LATD    = 0x00;
718       TRISD   = 0b00000000;
719                               // the other bits are also defined as outputs
720
721   // INT0 edge selection: '0' --> on falling edge
722   // (interrupt after clicking the pushbutton)
723       INTEDG0 = 0;
724       INT0IE  = 1; // Enable INT0 interrupts
725       INT1IE  = 1; // Enable INT1 interrupts
726
727
728       PEIE = 0;   // Disable all the other sources of interrupt; PHASE 1 and 2
729       GIE = 1;        // Global interrupts allowed
730
731       Var_CLK_Flag = 0;   // Reset the flag to detect an interrupt INT
732       St_Sp_flag  = 0;    // Reset the flag to detect an interrupt INT
733
734       lcd_init(FOURBIT_MODE);     /* Initialise the LCD module  */
735       LCD_Flag = 1;
736   }
```

3.3 Phase 3

```
/*
================================================================================
 UPC - EETAC - CSD - http://digsys.upc.edu
Phase 3
An example on how to implement a sequential system like an 1-digit BCD
counter
using "current_state" state-type variable.
Thus, this example can be used to adapt other FSM containing a small number
of states.

 * Print in color this file, and analyse it and use it as 'theory' class
 notes.
================================================================================
 */

// This general header file includes the device specific headers like
// "pic18f4520.h". This file contains all the microcontroller declarations.
#include <xc.h>
#include <stdio.h>
#include "lcd.h"
/*
================================================================================
    Definitions

    ================================================================================
     */
// This structure allows to access to a given bit into an 8-bit register
#define     PORTBIT(adr, bit) ( (unsigned)(&adr)*8 + (bit) )
// Our counter has 13 states defined as follows:
#define     Idle            'I' //States enumeration. "Char"
#define     Num_0           '0'
#define     Num_1           '1'
#define     Num_2           '2'
#define     Num_3           '3'
#define     Num_4           '4'
#define     Num_5           '5'
#define     Num_6           '6'
#define     Num_7           '7'
#define     Num_8           '8'
#define     Num_9           '9'
#define     Num_10          'A'
#define     Num_11          'B'

 //Other constants:
#define     Counter_var_Max_Count   10      //(for 20 s )

#define     Timer2_period_Reg       250

/*
================================================================================
    Function prototypes

    ================================================================================
     */
// The three basic functions to organise the code to emulate a FSM
void init_system (void);
char state_logic(void);
char output_logic(void);
```

```
49
50  //The interrupt service routine (ISR) to detect the Var_CLK_Flag (external
    INT0)
51  // or the ST_SP signal (RB interrupt, or the TMR0 interrupt
52  static void interrupt ISR(void);
53
54  // The auxiliary functions to read/poll and write port pins and be able to
55  // organise the software variables, to make the main program hardware
56  // independent and compatible for many microcontrollers.
57  void write_outputs (void);
58  void read_inputs(void);
59
60  /*
    ================================================================================
61      Global variables
62
        ================================================================================
         */
63  // Even if some variables are 'bit' type, we use 'char' (byte) to make them
    easy
64  // to watch while debugging.
65  static char Var_CE;
66  static char Var_UD_L;
67  static char Var_TC12;
68  static char Var_Q;
69  static char Var_TMR2;
70  static char TMR2_wave;
71
72  //The flag that will be set from the interrupt service routine to indicate
    that
73  // a new INT0 or Timmer has arrived or when ST_SP button is pushed or when
    the has to
74  //write something new
75  static char Var_CLK_Flag;
76  static char St_Sp_flag;
77  static char LCD_Flag;
78  static char Timer_Flag;
79  static char Var_INT0_Flag;
80
81  static int Counter_var = 0;     // The auxiliary timer variable to enlarge
    timing periods
82  static int Count_wave=0;
83
84  static char current_state = Idle;
85
86  /*
    ================================================================================
87      Main function
88
        ================================================================================
         */
89  void main(void){
90      init_system ();
91      while(1) {                      /* loop forever the FSM */
92          if (state_logic()) break;
93          if (output_logic()) break;
94      }
95      while (1){
```

```
96          __nop();//The system has crashed in an illegal state and needs
            attention
97                  //This second infinite loop is never reached in normal
                    operation
98      }           // The only way to scape is clicking reset (CD)
99  }
100
101 /*
    ============================================================================
102     Function definitions
103
        ============================================================================
         */
104
105 //****************************************************************************
    **
106 //State variable logic routine  (The CC1)
107 //****************************************************************************
    **
108 char state_logic(void){
109     char error = 0;
110
111     read_inputs(); //sample input values so that the CC1 can decide
        transitions
112
113     if(Var_TMR2==1){                //depending if TMR2 is on
114         Var_CLK_Flag = Timer_Flag;  //the CLK flag will be the
115     }else{                          //INT0 or the Timer
116          Var_CLK_Flag = Var_INT0_Flag;
117     }
118
119     if (Var_CLK_Flag == 1){
120
121         switch (current_state) {
122             case Idle:
123                 if((Var_CE == 1) && (Var_UD_L == 0) && (St_Sp_flag == 1)){
                    // if the ST_SP has been pushed
124                     current_state = Num_11;
                        //then it will perimit to change state
125                     St_Sp_flag=0;   //reset flag after operation
                        //depending in the direction (up or down)
126                     LCD_Flag=1;     //due to change of state we right
                        somthing new in the lcd
127                 }
128                 else if((Var_CE == 1) && (Var_UD_L == 1)&& (St_Sp_flag ==
                    1)) {
129                     current_state = Num_0;
130                     St_Sp_flag=0;
131                     LCD_Flag=1;
132                 }else{
133                     current_state = Idle;
134                 }
135             break;
136             case Num_0:
137                 if((Var_CE == 1) && (Var_UD_L == 0) && (St_Sp_flag == 1)){
138                     current_state = Idle;
139                     St_Sp_flag=0;
140                     LCD_Flag=1;
```

```
141                 }
142                 else if ((Var_CE == 1) && (Var_UD_L == 1)){
143                     current_state = Num_1;
144                     LCD_Flag=1;
145                 }
146                 else if((Var_CE == 1) && (Var_UD_L == 0) && (St_Sp_flag ==
                    0)) {
147                     current_state = Num_11;
148                     LCD_Flag=1;
149 
150                 }else{
151                     current_state = Num_0;
152                 }
153             break;
154 
155             case Num_1:
156                 if ((Var_CE == 1) && (Var_UD_L == 1)){
157                     current_state = Num_2;
158                     LCD_Flag=1;
159                 }
160                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
161                     current_state = Num_0;
162                     LCD_Flag=1;
163                 }else{
164                     current_state = Num_1;
165                 }
166             break;
167 
168             case Num_2:
169                 if ((Var_CE == 1) && (Var_UD_L == 1)){
170                     current_state = Num_3;
171                     LCD_Flag=1;
172                 }
173                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
174                     current_state = Num_1;
175                     LCD_Flag=1;
176                 }else{
177                     current_state = Num_2;
178                 }
179             break;
180 
181             case Num_3:
182                 if ((Var_CE == 1) && (Var_UD_L == 1)){
183                     current_state = Num_4;
184                     LCD_Flag=1;
185                 }
186                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
187                     current_state = Num_2;
188                     LCD_Flag=1;
189                 }else{
190                     current_state = Num_3;
191                 }
192             break;
193             case Num_4:
194                 if ((Var_CE == 1) && (Var_UD_L == 1)){
195                     current_state = Num_5;
196                     LCD_Flag=1;
197                 }
```

```
198                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
199                     current_state = Num_3;
200                     LCD_Flag=1;
201                 }else{
202                     current_state = Num_4;
203                 }
204             break;
205             case Num_5:
206                 if ((Var_CE == 1) && (Var_UD_L == 1)){
207                     current_state = Num_6;
208                     LCD_Flag=1;
209                 }
210                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
211                     current_state = Num_4;
212                     LCD_Flag=1;
213                 }else{
214                     current_state = Num_5;
215                 }
216             break;
217             case Num_6:
218                 if ((Var_CE == 1) && (Var_UD_L == 1)){
219                     current_state = Num_7;
220                     LCD_Flag=1;
221                 }
222                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
223                     current_state = Num_5;
224                     LCD_Flag=1;
225                 }else{
226                     current_state = Num_6;
227                 }
228             break;
229             case Num_7:
230                 if ((Var_CE == 1) && (Var_UD_L == 1)){
231                     current_state = Num_8;
232                     LCD_Flag=1;
233                 }
234                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
235                     current_state = Num_6;
236                     LCD_Flag=1;
237                 }else{
238                     current_state = Num_7;
239                 }
240             break;
241             case Num_8:
242                 if ((Var_CE == 1) && (Var_UD_L == 1)){
243                     current_state = Num_9;
244                     LCD_Flag=1;
245                 }
246                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
247                     current_state = Num_7;
248                     LCD_Flag=1;
249                 }else{
250                     current_state = Num_8;
251                 }
252             break;
253             case Num_9:
254                 if ((Var_CE == 1) && (Var_UD_L == 1)){
255                     current_state = Num_10;
```

```
256                     LCD_Flag=1;
257                 }
258                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
259                     current_state = Num_8;
260                     LCD_Flag=1;
261                 }else{
262                     current_state = Num_9;
263                 }
264             break;
265             case Num_10:
266                 if ((Var_CE == 1) && (Var_UD_L == 1)){
267                     current_state = Num_11;
268                     LCD_Flag=1;
269                 }
270                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
271                     current_state = Num_9;
272                     LCD_Flag=1;
273                 }else{
274                     current_state = Num_10;
275                 }
276             break;
277             case Num_11:
278                  if((Var_CE == 1) && (Var_UD_L == 1) && (St_Sp_flag == 1)){
279                     current_state = Idle;
280                     St_Sp_flag=0;
281                     LCD_Flag=1;
282                 }
283                  else if ((Var_CE == 1) && (Var_UD_L == 1) && (St_Sp_flag
                     == 0)){
284                     current_state = Num_0;
285                     LCD_Flag=1;
286                 }
287                 else if((Var_CE == 1) && (Var_UD_L == 0)) {
288                     current_state = Num_10;
289                     LCD_Flag=1;
290                 }else{
291                     current_state = Num_11;
292                 }
293             break;
294 
295             default:
296             // If no states have been selected, there is an error
297             // and the systems must be stopped
298                 error = 1;
299                 break;
300         }
301         Var_INT0_Flag=0;    // Clear the Var_CLK_Flag detection flag
302         Timer_Flag=0;
303     }
304     return (error);
305 }
306 
307 //*****************************************************************************
    **
308 //Output logic routine (The combinational circuit CC2)
309 //*****************************************************************************
    **
310 char output_logic(void){
```

```
311  unsigned char error = 0;
312      switch (current_state) {
313          case Idle:
314              Var_Q = 0;//idle
315              Var_TC12 = 0;
316               if (LCD_Flag == 1) {           //write only if flag activates
317                  lcd_clear();
318                  lcd_home();                       // select line 1
319                  lcd_puts("IDLE mode ...");
320                  lcd_home2();             // select line 2 to write the result
321                  lcd_puts(" ... push ST_SP ...");
322                  }
323              LCD_Flag = 0;
324              break;
325          case Num_0:
326              Var_Q = 0;//0
327              Var_TC12 = 1;
328              if ((LCD_Flag == 1)&&(Var_UD_L==1)) {   //depending if counting
                 up`or down
329                  lcd_clear();                        //the lcd will give a
                     different output
330                  lcd_home();
331                  lcd_puts("0 ...");
332                  lcd_home2();
333                  lcd_puts(" ... count up ...");
334                  }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
335                  lcd_clear();
336                  lcd_home();
337                  lcd_puts("0 ...");
338                  lcd_home2();
339                  lcd_puts(" ... count down ...");
340                  }
341              LCD_Flag = 0;
342              break;
343          case Num_1:
344              Var_Q = 0b00100000;//1
345              Var_TC12 = 1;
346              if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
347                  lcd_clear();
348                  lcd_home();
349                  lcd_puts("1 ...");
350                  lcd_home2();
351                  lcd_puts(" ... count up ...");
352                  }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
353                  lcd_clear();
354                  lcd_home();
355                  lcd_puts("1 ...");
356                  lcd_home2();
357                  lcd_puts(" ... count down ...");
358                  }
359              LCD_Flag = 0;
360              break;
361          case Num_2:
362              Var_Q = 0b00110000;//2
363              Var_TC12 = 1;
364              if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
365                  lcd_clear();
366                  lcd_home();
```

```
367                 lcd_puts("2 ...");
368                 lcd_home2();
369                 lcd_puts(" ... count up ...");
370                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
371                 lcd_clear();
372                 lcd_home();
373                 lcd_puts("2 ...");
374                 lcd_home2();
375                 lcd_puts(" ... count down ...");
376                 }
377             LCD_Flag = 0;
378             break;
379         case Num_3:
380             Var_Q = 0b00111000;//3
381             Var_TC12 = 1;
382             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
383                 lcd_clear();
384                 lcd_home();
385                 lcd_puts("3 ...");
386                 lcd_home2();
387                 lcd_puts(" ... count up ...");
388                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
389                 lcd_clear();
390                 lcd_home();
391                 lcd_puts("3 ...");
392                 lcd_home2();
393                 lcd_puts(" ... count down ...");
394                 }
395             LCD_Flag = 0;
396             break;
397         case Num_4:
398             Var_Q = 0b00111100;//4
399             Var_TC12 = 1;
400             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
401                 lcd_clear();
402                 lcd_home();
403                 lcd_puts("4 ...");
404                 lcd_home2();
405                 lcd_puts(" ... count up ...");
406                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
407                 lcd_clear();
408                 lcd_home();
409                 lcd_puts("4 ...");
410                 lcd_home2();
411                 lcd_puts(" ... count down ...");
412                 }
413             LCD_Flag = 0;
414             break;
415         case Num_5:
416             Var_Q = 0b00111110;//5
417             Var_TC12 = 1;
418             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
419                 lcd_clear();
420                 lcd_home();
421                 lcd_puts("5 ...");
422                 lcd_home2();
423                 lcd_puts(" ... count up ...");
424                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
```

```
425                 lcd_clear();
426                 lcd_home();
427                 lcd_puts("5 ...");
428                 lcd_home2();
429                 lcd_puts(" ... count down ...");
430                 }
431             LCD_Flag = 0;
432             break;
433         case Num_6:
434             Var_Q = 0b00111111;//6
435             Var_TC12 = 0;
436             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
437                 lcd_clear();
438                 lcd_home();
439                 lcd_puts("6 ...");
440                 lcd_home2();
441                 lcd_puts(" ... count up ...");
442                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
443                 lcd_clear();
444                 lcd_home();
445                 lcd_puts("6 ...");
446                 lcd_home2();
447                 lcd_puts(" ... count down ...");
448                 }
449             LCD_Flag = 0;
450             break;
451         case Num_7:
452             Var_Q = 0b00011111;//7
453             Var_TC12 = 0;
454             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
455                 lcd_clear();
456                 lcd_home();
457                 lcd_puts("7 ...");
458                 lcd_home2();
459                 lcd_puts(" ... count up ...");
460                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
461                 lcd_clear();
462                 lcd_home();
463                 lcd_puts("7 ...");
464                 lcd_home2();
465                 lcd_puts(" ... count down ...");
466                 }
467             LCD_Flag = 0;
468             break;
469         case Num_8:
470             Var_Q = 0b00001111;//8
471             Var_TC12 = 0;
472             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
473                 lcd_clear();
474                 lcd_home();
475                 lcd_puts("8 ...");
476                 lcd_home2();
477                 lcd_puts(" ... count up ...");
478                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
479                 lcd_clear();
480                 lcd_home();
481                 lcd_puts("8 ...");
482                 lcd_home2();
```

```
483                 lcd_puts(" ... count down ...");
484                 }
485             LCD_Flag = 0;
486             break;
487         case Num_9:
488             Var_Q = 0b00000111;//9
489             Var_TC12 = 0;
490             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
491                 lcd_clear();
492                 lcd_home();
493                 lcd_puts("9 ...");
494                 lcd_home2();
495                 lcd_puts(" ... count up ...");
496                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
497                 lcd_clear();
498                 lcd_home();
499                 lcd_puts("9 ...");
500                 lcd_home2();
501                 lcd_puts(" ... count down ...");
502                 }
503             LCD_Flag = 0;
504             break;
505         case Num_10:
506             Var_Q = 0b00000011;//A
507             Var_TC12 = 0;
508             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
509                 lcd_clear();
510                 lcd_home();
511                 lcd_puts("10 ...");
512                 lcd_home2();
513                 lcd_puts(" ... count up ...");
514                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
515                 lcd_clear();
516                 lcd_home();
517                 lcd_puts("10 ...");
518                 lcd_home2();
519                 lcd_puts(" ... count down ...");
520                 }
521             LCD_Flag = 0;
522             break;
523         case Num_11:
524             Var_Q = 0b00000001;//B
525             Var_TC12 = 0;
526             if ((LCD_Flag == 1)&&(Var_UD_L==1)) {
527                 lcd_clear();
528                 lcd_home();
529                 lcd_puts("11 ...");
530                 lcd_home2();
531                 lcd_puts(" ... count up ...");
532                 }else if((LCD_Flag == 1)&&(Var_UD_L==0)){
533                 lcd_clear();
534                 lcd_home();
535                 lcd_puts("11 ...");
536                 lcd_home2();
537                 lcd_puts(" ... count down ...");
538                 }
539             LCD_Flag = 0;
540             break;
```

```
541             default:
542                 error = 1;  // If no states have been selected, there is an error
543                             // and the systems must be stopped
544                 break;
545         }
546
547         write_outputs(); // The interface with the hardware to convert variables
548                          // into voltages at a given microcontroller pin.
549
550         return (error);
551
552     }
553
554     //***************************************************************************
555     //Interrupt Service Routine
556     //***************************************************************************
557
558     static void interrupt ISR (void){   //"interrupt" is the key word here
559         GIE = 0;        //Disable interrupts while attending one of them ...
560         if(INT0IF == 1) {       // Is it an INT0 interrupt ?  PHASE 1
561         //GIE = 0;
562     // If the code execution reach this point is because a falling edge signal
563     // in RB0 has been detected, so let's set the flag and the FSM will
564     // acknowledge it
565             Var_INT0_Flag = 1;  //Set the "software flag"
566             INT0IF = 0; // clear the "hardware" INT0 interrupt flag
567             //GIE = 1;
568         }
569         if(INT1IF ==1){         // Is it an INT1 interrupt ?
570            // GIE = 0;
571             St_Sp_flag=1;
572             INT1IF =0;
573             //GIE = 1;
574         }
575         if(TMR2IF == 1) {           // Is it a Timer2 overflow?
576             Counter_var ++;
577             if (Counter_var == Counter_var_Max_Count){
578                 Timer_Flag = 1;     // Set the flag to match the external clock
579                 Counter_var=0;
580                 if(Count_wave == 2){  //we double the time to give a visual aid
                    in the oscilloscope
581                     TMR2_wave=0;
582                     Count_wave =0;
583                 }else{
584                     TMR2_wave=1;
585                     Count_wave++;
586                 }
587
588             }
589             TMR2IF = 0;         // Clear the flag
590         }
591         GIE = 1;        //Enable interrupts
592     }
593
594
595     /*
        ============================================================================
596      Scan inputs and process the bits so that we can set our convenient
```

```
 variables
597  many of the variables are "char" because we can easily monitor them using
598  the watch window for debugging purposes.
599
      ================================================================================
       */
600 void read_inputs (void){
601
602 // Like in P9, read the port, mask the bit of interest and set the variable
603     Var_CE = (PORTA & 0b00000001);
604     Var_UD_L = (PORTA & 0b00000010)>>1;
605     Var_TMR2= (PORTA & 0b00000100)>>2;
606 }
607
608 /*
    ================================================================================
609     Driving the output pins. Let's drive the output ports, converting
610     microcontroller variables into electrical signals.
611
       ================================================================================
        */
612 void write_outputs(void){
613
614     //To write a given output pin:
615     if(Var_TMR2==1){
616     PORTC = Var_Q | (Var_TC12 << 6)|(TMR2_wave <<7);
617     }else{
618     PORTC = Var_Q | (Var_TC12 << 6);
619     }
620 }
621
622
623 //*************************************************************************
    **
624 // Initialise the microcontroller system
625 //*************************************************************************
    **
626 void init_system (void){
627 // Configuration bits
628
629 // These are the configuration bits translated to the XC8 Compiler
630 // Microcontroller general configuration bits. You don't have to change them.
631 // PIC18F4520 Configuration Bit Settings. They can be generated
    automatically
632 // in the MPLAB X IDE
633
634 // CONFIG1H
635 #pragma config OSC = XT       // Oscillator Section bits (XT oscillator)
636 #pragma config FCMEN = OFF
637 // Fail-Safe Clock Monitor E bit (Fail-Safe Clock Monitor disabled)
638 #pragma config IESO = OFF
639 // Internal/External Osc Switch-over bit (Oscillator Switch-over mode
    disabled)
640
641 // CONFIG2L
642 #pragma config PWRT = ON       // Power-up Timer E bit (PWRT enabled)
643 #pragma config BOREN = SBORDIS
644 // Brown-out Reset E bits (Brown-out Reset enabled in hardware
```

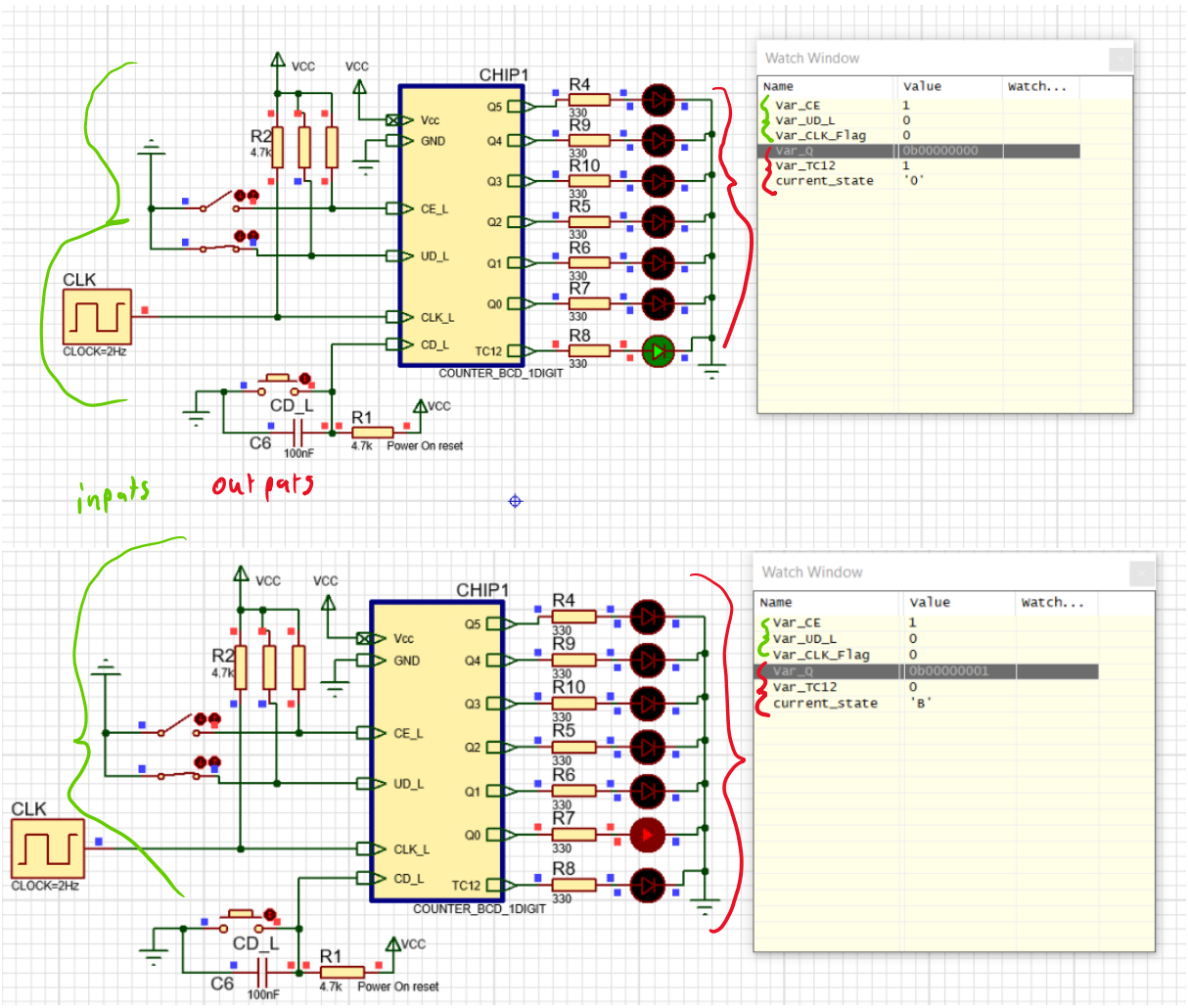
```
645  // only (SBOREN is disabled))
646  #pragma config BORV = 3    // Brown Out Reset Voltage bits (Minimum setting)
647
648  // CONFIG2H
649  #pragma config WDT = OFF
650  // Watchdog Timer E bit (WDT disabled (control is on the SWDTEN bit))
651  #pragma config WDTPS = 32768
652  // Watchdog Timer Postscale Sect bits (1:32768)
653
654  // CONFIG3H
655  #pragma config CCP2MX = PORTC
656  // CCP2 MUX bit (CCP2 input/output is multiplexed with RC1)
657  #pragma config PBADEN = OFF
658  // PORTB A/D E bit (PORTB<4:0> pins are configured
659  // as digital I/O on Reset)
660  #pragma config LPT1OSC = OFF
661  // Low-Power Timer1 Oscillator E bit (Timer1 configured for
662  // higher power operation)
663  #pragma config MCLRE = ON
664  // MCLR Pin E bit (MCLR pin enabled; RE3 input pin disabled)
665
666  // CONFIG4L
667  #pragma config STVREN = ON
668  // Stack Full/Underflow Reset E bit (Stack full/underflow will cause Reset)
669  #pragma config LVP = OFF
670  // Single-Supply ICSP E bit (Single-Supply ICSP disabled)
671  #pragma config XINST = OFF
672  // Extended Instruction Set E bit (Instruction set extension and Indexed
673  // Addressing mode disabled (Legacy mode))
674
675  // CONFIG5L
676  #pragma config CP0 = OFF
677  // Code Protection bit (Block 0 (000800-001FFFh) not code-protected)
678  #pragma config CP1 = OFF
679  // Code Protection bit (Block 1 (002000-003FFFh) not code-protected)
680  #pragma config CP2 = OFF
681  // Code Protection bit (Block 2 (004000-005FFFh) not code-protected)
682  #pragma config CP3 = OFF
683  // Code Protection bit (Block 3 (006000-007FFFh) not code-protected)
684
685  // CONFIG5H
686  #pragma config CPB = OFF
687  // Boot Block Code Protection bit
688  // (Boot block (000000-0007FFh) not code-protected)
689  #pragma config CPD = OFF
690  // Data EEPROM Code Protection bit (Data EEPROM not code-protected)
691
692  // CONFIG6L
693  #pragma config WRT0 = OFF
694  // Write Protection bit (Block 0 (000800-001FFFh) not write-protected)
695  #pragma config WRT1 = OFF
696  // Write Protection bit (Block 1 (002000-003FFFh) not write-protected)
697  #pragma config WRT2 = OFF
698  // Write Protection bit (Block 2 (004000-005FFFh) not write-protected)
699
700  #pragma config WRT3 = OFF
701  // Write Protection bit (Block 3 (006000-007FFFh) not write-protected)
702
```

```
703  // CONFIG6H
704  #pragma config WRTC = OFF
705  // Configuration Register Write Protection bit (Configuration registers
706  // (300000-3000FFh) not write-protected)
707  #pragma config WRTB = OFF
708  // Boot Block Write Protection bit (Boot block (000000-0007FFh)
709  // not write-protected)
710  #pragma config WRTD = OFF
711  // Data EEPROM Write Protection bit (Data EEPROM not write-protected)
712
713  // CONFIG7L
714  #pragma config EBTR0 = OFF
715  // Table Read Protection bit (Block 0 (000800-001FFFh) not protected from
716  // table reads executed in other blocks)
717  #pragma config EBTR1 = OFF
718  // Table Read Protection bit (Block 1 (002000-003FFFh) not protected from
719  // table reads executed in other blocks)
720  #pragma config EBTR2 = OFF
721  // Table Read Protection bit (Block 2 (004000-005FFFh) not protected from
     table
722  // reads executed in other blocks)
723  #pragma config EBTR3 = OFF
724  // Table Read Protection bit (Block 3 (006000-007FFFh) not protected from
     table
725  // reads executed in other blocks)
726
727  // CONFIG7H
728  #pragma config EBTRB = OFF
729  // Boot Block Table Read Protection bit (Boot block (000000-0007FFh)
730  // not protected from table reads executed in other blocks)
731
732      /* -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+*/
733  // Initialise starts here:
734
735  /* All the PORTA, PORTB and PORTE is set to be digital I/O (The can be
736       * defined as analogue inputs*/
737      ADCON1  = 0x0F; // read the datasheet chapter on the A/D converter
738  // All unused pins are defined as outputs and reset
739
740      PORTA   = 0x00;         // Reset all Flip-Flops at PORTB
741      LATA    = 0x00;
742      TRISA   = 0b0000111;    // PORTB1 and PORTB0 are inputs; PORTB2 is
       output
743
744      PORTB   = 0x00;         // Reset all Flip-Flops at PORTB
745      LATB    = 0x00;
746      TRISB   = 0b0000011;
747
748      PORTC   = 0x00;         // Reset all Flip-Flops at PORTD
749      LATC    = 0x00;
750      TRISC   = 0b00000000;   // PORTD(7..4) are outputs for the code
751
752      PORTD   = 0x00;         // Reset all Flip-Flops at PORTD
753      LATD    = 0x00;
754      TRISD   = 0b00000000;
755                              // the other bits are also defined as outputs
756
757  // INT0 edge selection: '0' --> on falling edge
```

```
758  // (interrupt after clicking the pushbutton)
759      INTEDG0 = 0;
760      INT0IE  = 1; // Enable INT0 interrupts
761      INT1IE  = 1; // Enable INT1 interrupts
762
763      T2CKPS0 = 1;
764      T2CKPS1 = 1;    // Prescaler is 16
765
766      TOUTPS0 = 1;
767      TOUTPS1 = 1;
768      TOUTPS2 = 0;
769      TOUTPS3 = 1;    // Postscaler is 12
770
771      TMR2ON  = 1;
772      TMR2IE  = 1;    // Timer2 interrupts not allowed when initiating the
     microcontroller
773                      // Activating the Timer2 interrupts will be done in
                         "Set_Timer2" state
774
775      IPEN = 0;   // Interrupt priority disabled
776
777      GIE = 1;        // Global interrupts allowed
778      PEIE    = 1;    // Enable interrups from other peripherals ( Timer 2 in
     this example)
779
780      Var_CLK_Flag = 0;   // Reset the flag to detect an interrupt INT
781      St_Sp_flag  = 0;    // Reset the flag to detect an interrupt INT
782
783      lcd_init(FOURBIT_MODE);     /* Initialise the LCD module  */
784      LCD_Flag = 1;
785  }
```

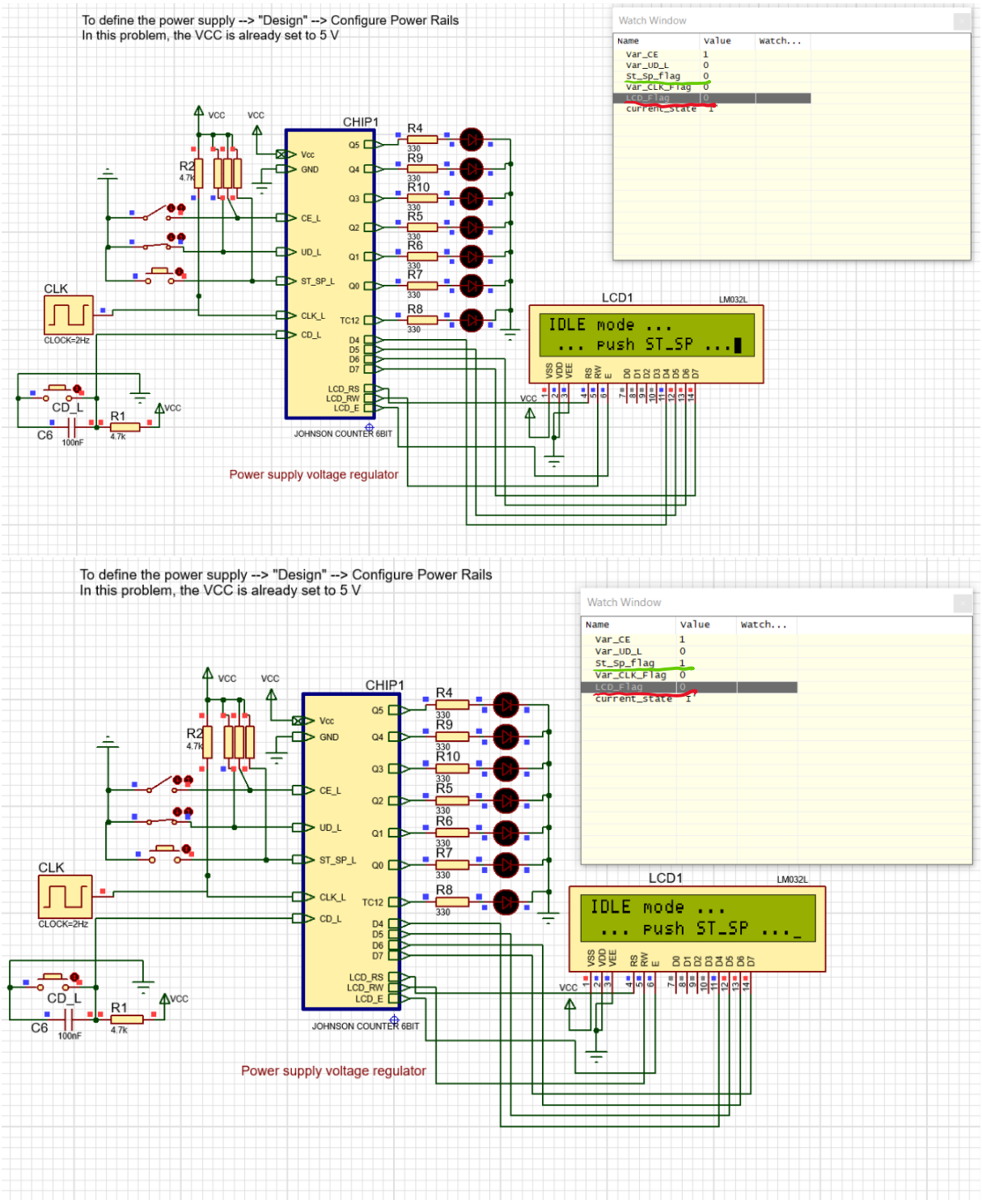
4 Test and verification

4.1 Phase 1

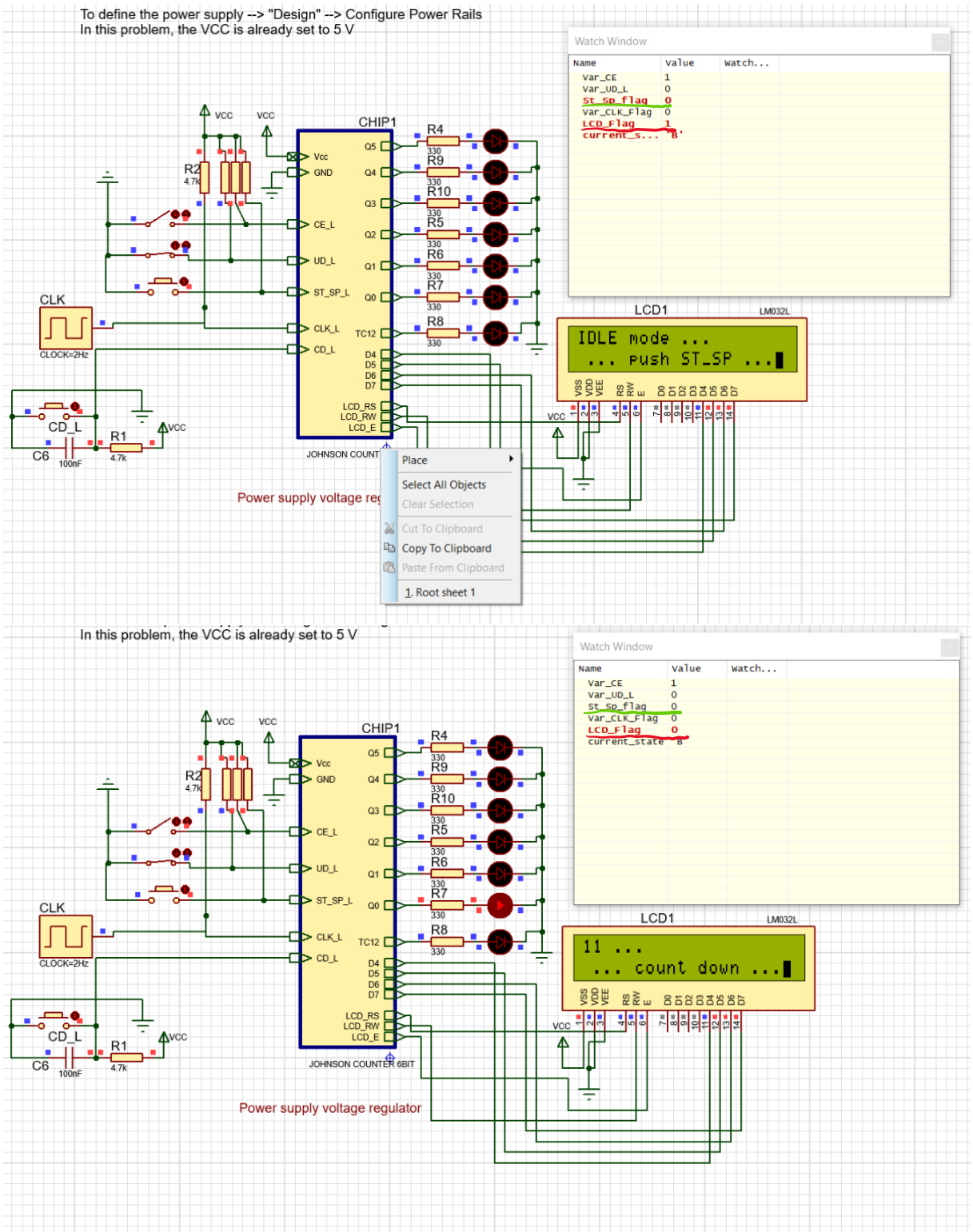


In phase one we mainly wanted to se if the base Johnson counter would work so we mainly checked if the inputs and outputs were working correctly.

4.2 Phase 2

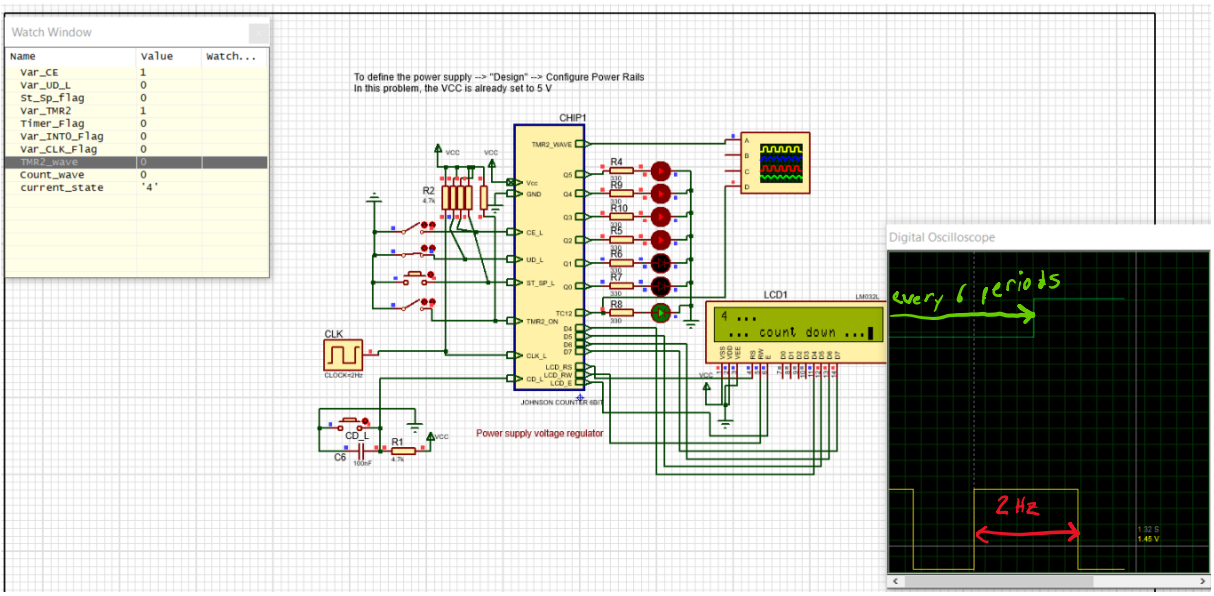

To define the power supply --> "Design" --> Configure Power Rails
In this problem, the VCC is already set to 5 V
Watch Window
Name
Value
Watch...
Var_CE 1
Var_UD_L 0
St_Sp_flag 0
Var_CLK_Flag 0
LCD_Flag 0
current_state 1
CHIP1
Vcc
GND
CE_L
UD_L
ST_SP_L
CLK_L
CD_L
Q5
Q4
Q3
Q2
Q1
Q0
TC12
D4
D5
D6
D7
LCD_RS
LCD_RW
LCD_E
JOHNSON COUNTER 6BIT
R2
4.7k
R4
R9
R10
R5
R6
R7
R8
330
CLK
CLOCK=2Hz
CD_L
R1
4.7k
C6
100nF
VCC
LCD1
LM032L
IDLE mode ...
... push ST_SP ...
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
Power supply voltage regulator

To define the power supply --> "Design" --> Configure Power Rails
In this problem, the VCC is already set to 5 V
Watch Window
Name
Value
Watch...
Var_CE 1
Var_UD_L 0
St_Sp_flag 1
Var_CLK_Flag 0
LCD_Flag 0
current_state 1
CHIP1
Vcc
GND
CE_L
UD_L
ST_SP_L
CLK_L
CD_L
Q5
Q4
Q3
Q2
Q1
Q0
TC12
D4
D5
D6
D7
LCD_RS
LCD_RW
LCD_E
JOHNSON COUNTER 6BIT
R2
4.7k
R4
R9
R10
R5
R6
R7
R8
330
CLK
CLOCK=2Hz
CD_L
R1
4.7k
C6
100nF
VCC
LCD1
LM032L
IDLE mode ...
... push ST_SP ...
VSS
VDD
VEE
RS
RW
E
D0
D1
D2
D3
D4
D5
D6
D7
Power supply voltage regulator



In phase 2 we added ST_SP interrupt and the lcd. In this phase, since the inputs and outputs were working correctly, we focused on getting lcd to print the correct message and by pushing the ST_SP button, depending on which direction the chip was counting to go to or exit from idle.

4.3 Phase 3



In phase 3 we added TMR2_ON switch that if activated, the chip would use the TMR2 as an internal clock that had the same period as the external clock and output that frequency to TMR2_wave and we also connected TC12 to the oscilloscope. In this phase we mainly checked if TMR2 was generating the same clock as the external clock.

5 Conclusions

After going through the various phase of the project, where we first started adapting and modifying a bcd counter, then little by little we kept adding other components like a ST_SP button, lcd screen and finally, we added the TMR2_ON switch (that allowed us to change from using the external clock or the timer2 as an internal clock with same period as the external one) and TMR2_wave (activates only when the timer2 is used as the clock), which is connected to an oscilloscope that way the user can see wave generated from the timer. In conclusion we saw that one can make this chip smaller by removing the external clock and just use a timer as an internal clock.

6 References

ideas_P12 [Document] / auth. Robert Francesc J.. - 2018.
Johnson counter: electrical4u [Online] / auth. Electrical4U // electrical4u. - July 31, 2018. - may 29, 2019. - https://www.electrical4u.com/johnson-counter/.
Project P10: Let's program a µC in C like a FSM [Online] / auth. J. Jordana; Robert, Francesc J. // digsys. - Setember 2001. - June 6, 2019. - https://digsys.upc.edu/csd/P10/P10.html.