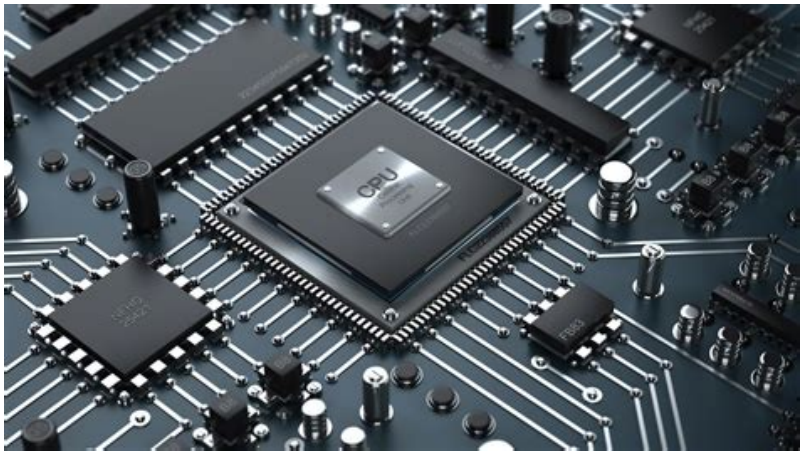
Trabajo Práctico Número 5

Informe técnico

Un trabajo presentado para la materia de
Aplicaciones de electrónica digital



Krapp Ramiro – Golmar Elias – Pisacane Juan Cruz
Instituto tecnológico San Bonifacio
Departamento de electrónica
26 de noviembre de 2021

Hecho en LaTeX

Índice

1. Actividad **2**
1.1. Se pide: 2

2. Circuito Eléctrico **3**

3. Diagrama de flujo **4**

4. Código en C **14**

5. Bitacora **20**

6. Simulaciones en Proteus **21**
6.1. C 21

Actividad

Se pide:

a) Dibujar circuito eléctrico.

b) Realizar diagrama de flujo.

c) Realizar código en lenguaje C.

d) Realizar bitácoras personales

e) Simular en PROTEUS.

Circuito Eléctrico

Este es el circuito electrico:

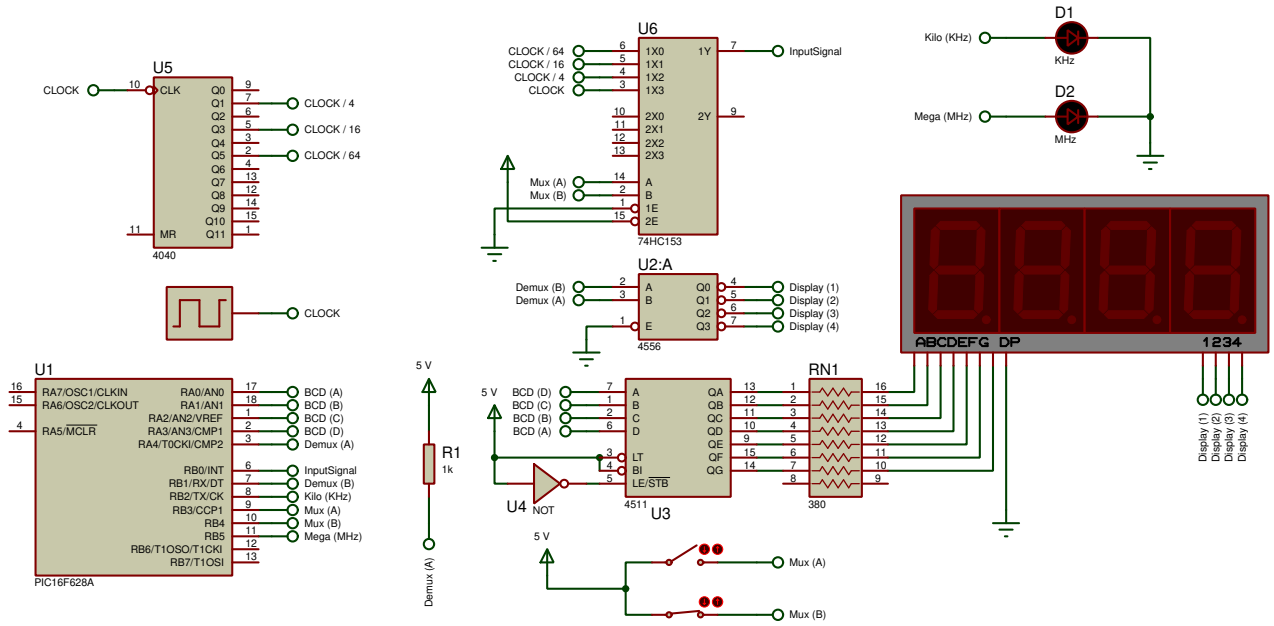
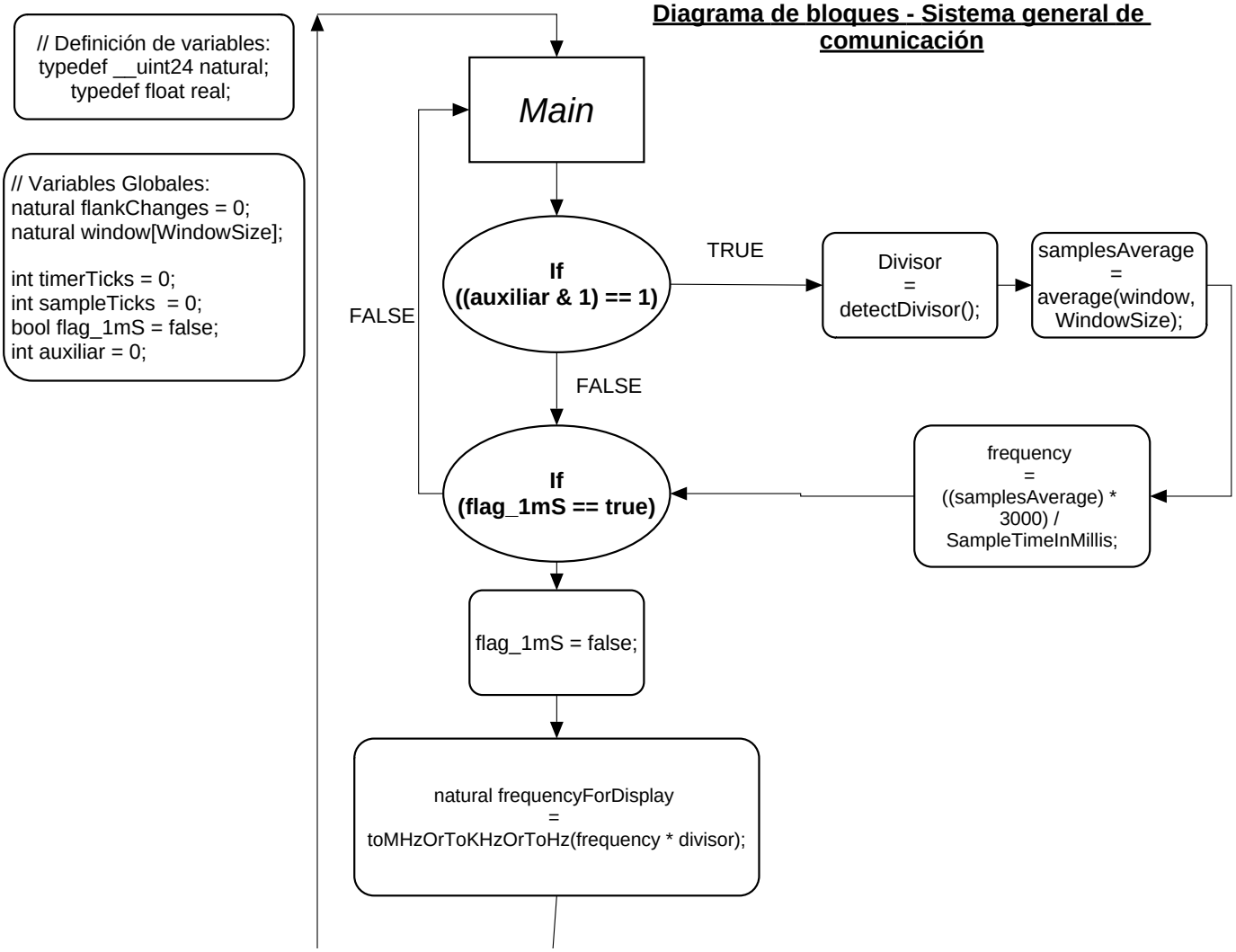


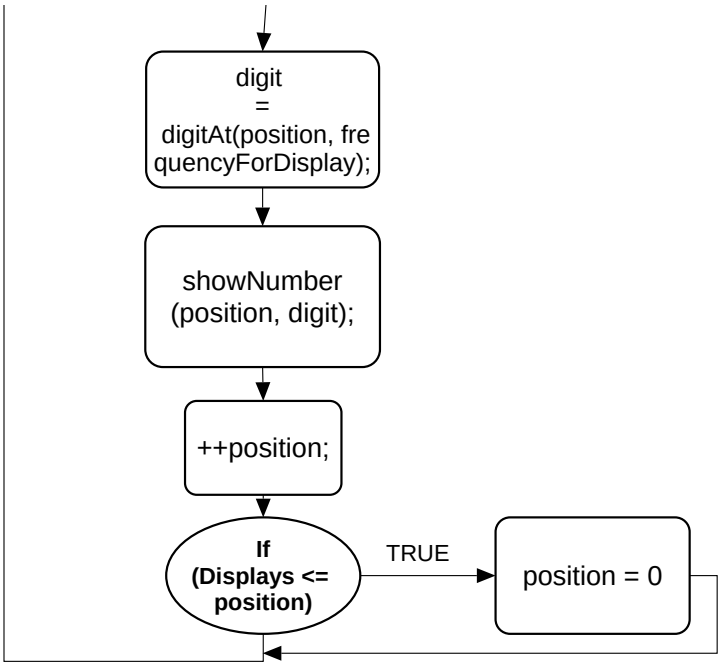
Figura 1: Diagrama esquematico hecho en Proteus

Diagrama de flujo

Este es el diagrama de flujo

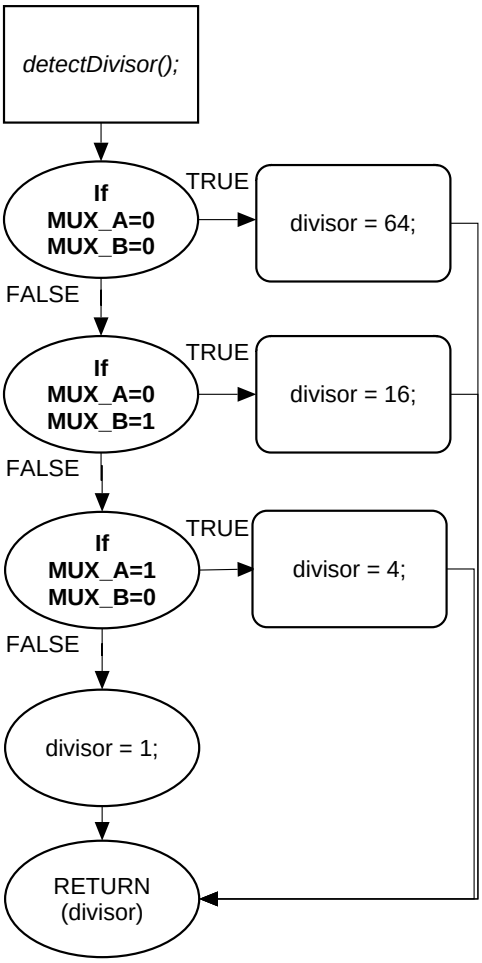
Diagrama de bloques - Sistema general de comunicación

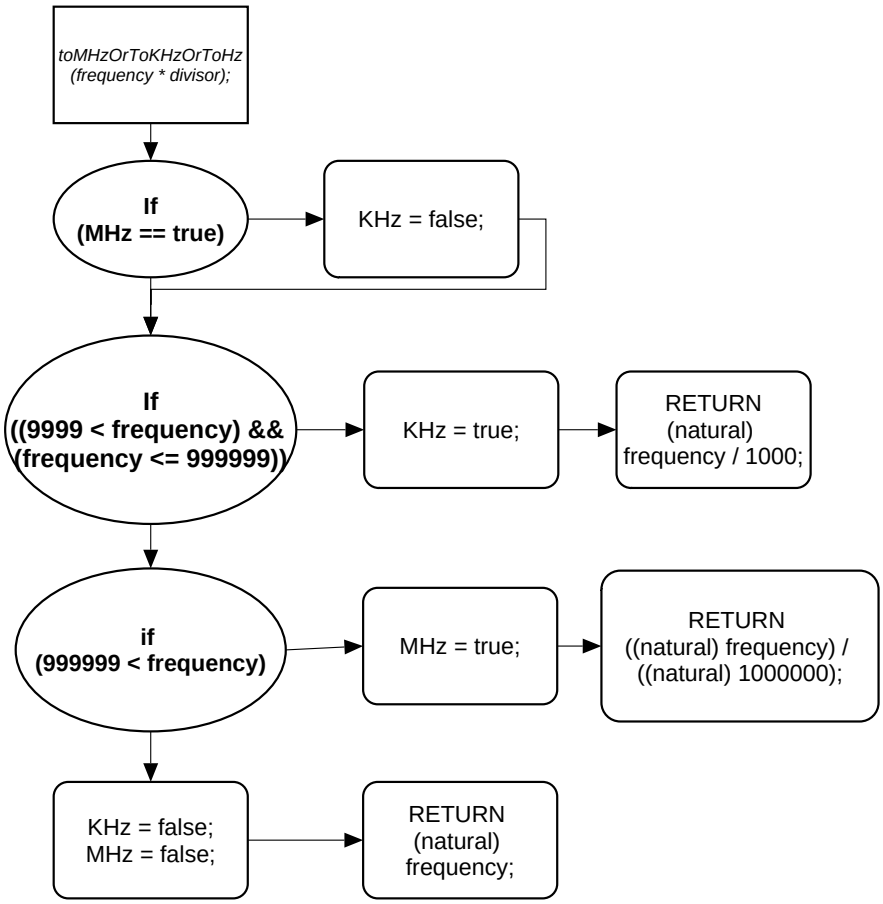


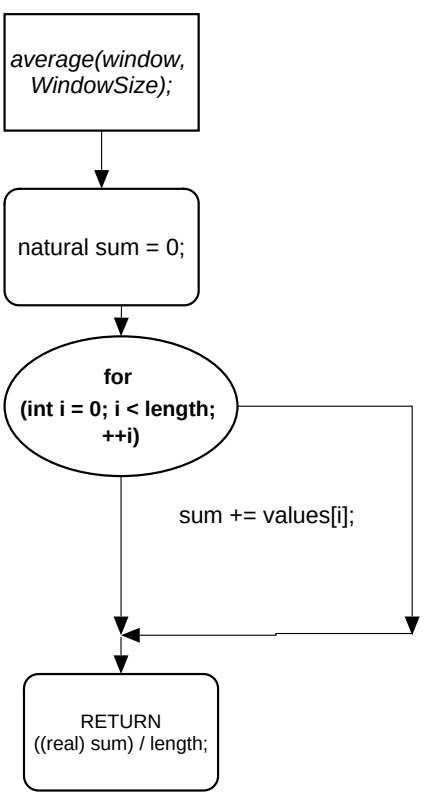


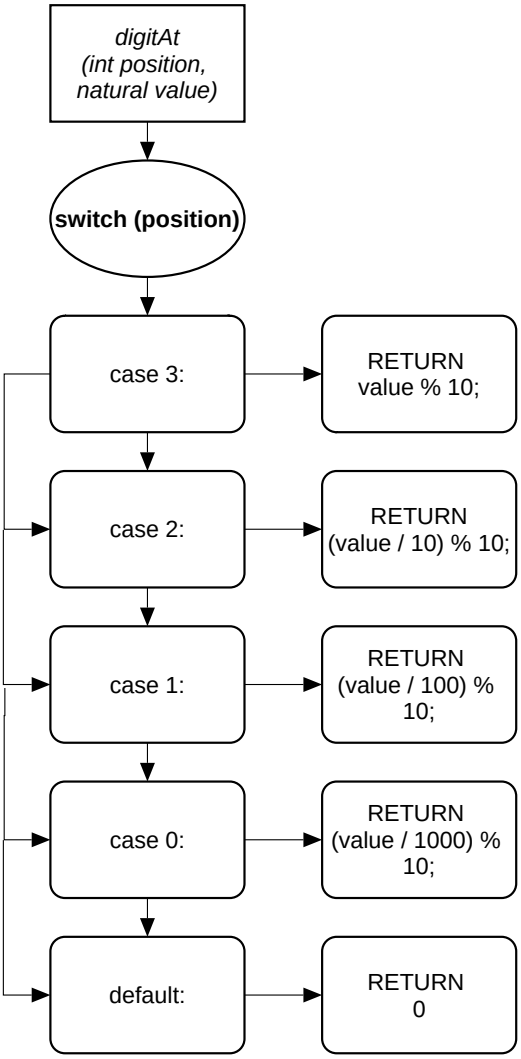
configuration
(void)

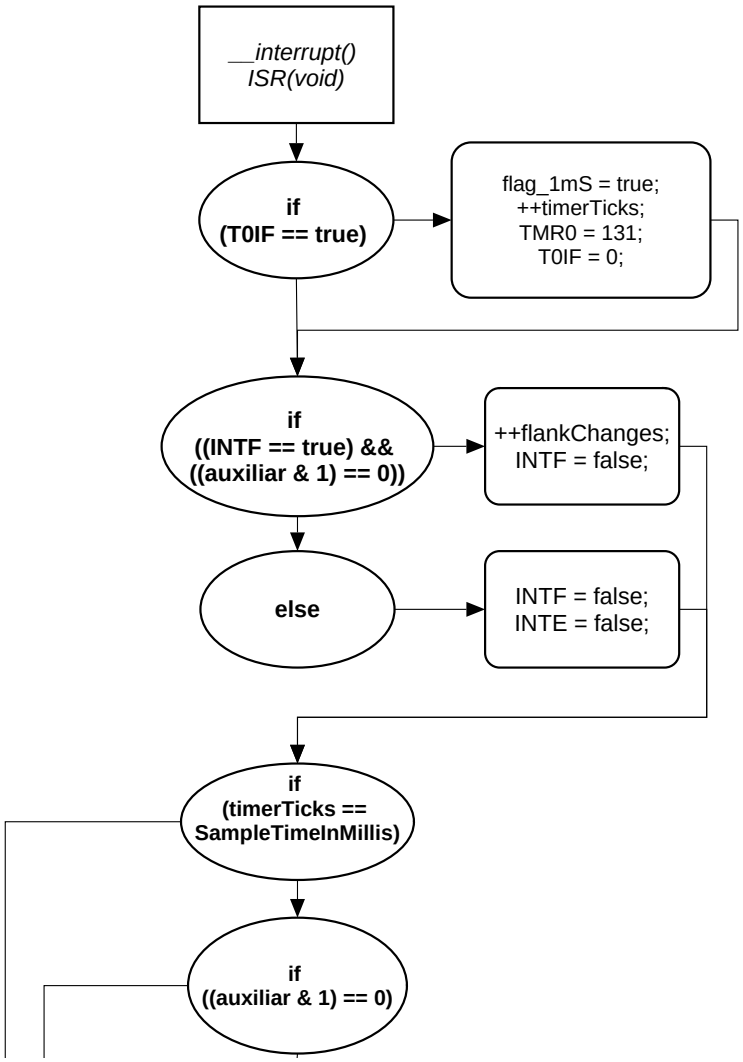
```
KHz = false;
MHz = false;

TRISB = 0x19;
PORTB = 0x00;
TRISA = 0x00;
PORTA = 0x00;

GIE = 1;

T0IE = 1;
T0CS = 0;
TMR0 = 131;
PSA = 0;
PS0 = 1;
PS1 = 1;
PS2 = 0;

INTE = 1;
INTEDG = 1;
```

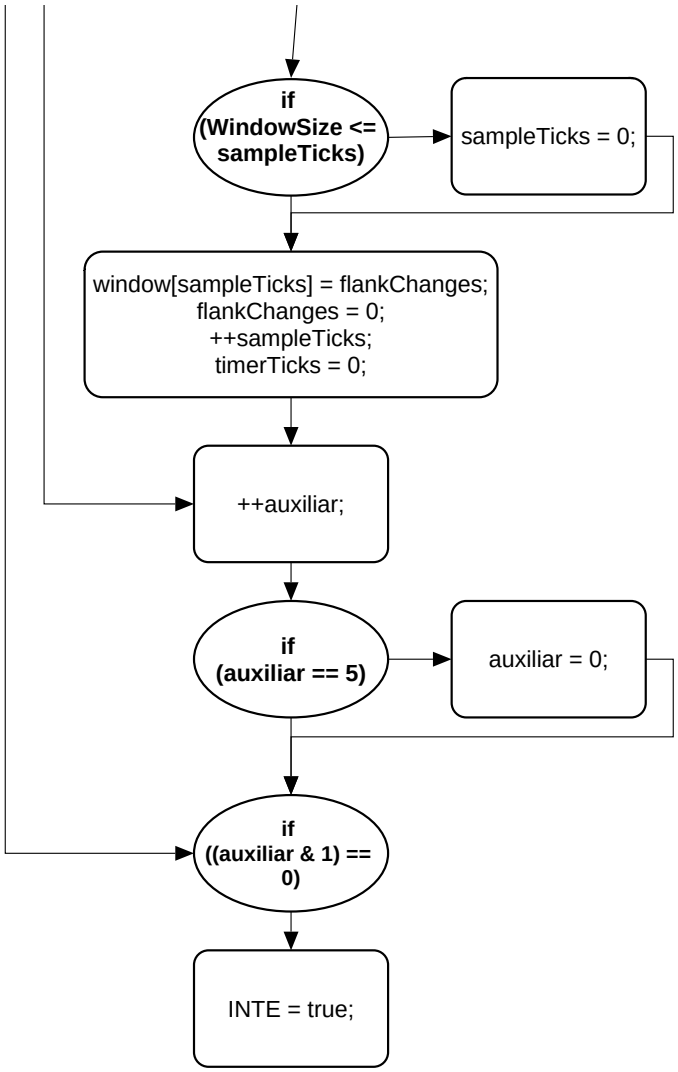

detectDivisor();
If
MUX_A=0
MUX_B=0
TRUE
divisor = 64;
FALSE
If
MUX_A=0
MUX_B=1
TRUE
divisor = 16;
FALSE
If
MUX_A=1
MUX_B=0
TRUE
divisor = 4;
FALSE
divisor = 1;
RETURN
(divisor)

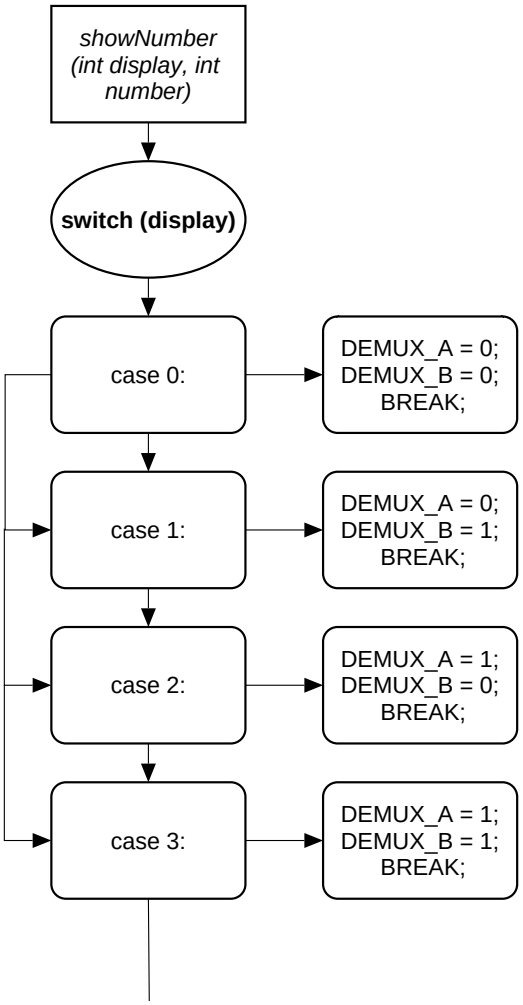

toMHzOrToKHzOrToHz
(frequency * divisor);
If
(MHz == true)
KHz = false;
If
((9999 < frequency) &&
(frequency <= 999999))
KHz = true;
RETURN
(natural)
frequency / 1000;
if
(999999 < frequency)
MHz = true;
RETURN
((natural) frequency) /
((natural) 1000000);
KHz = false;
MHz = false;
RETURN
(natural)
frequency;

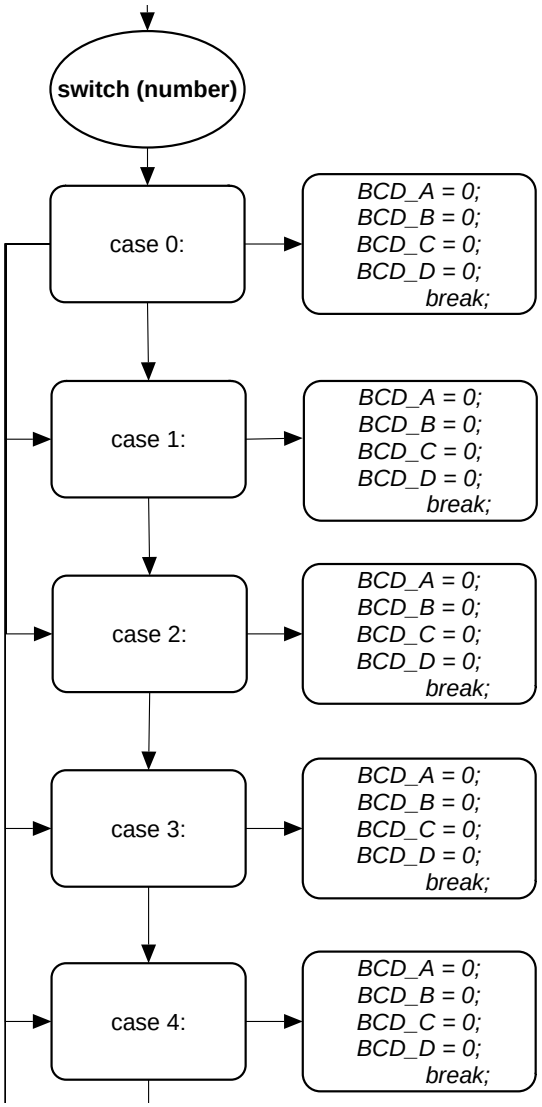

average(window, WindowSize);
natural sum = 0;
for (int i = 0; i < length; ++i)
sum += values[i];
RETURN ((real) sum) / length;

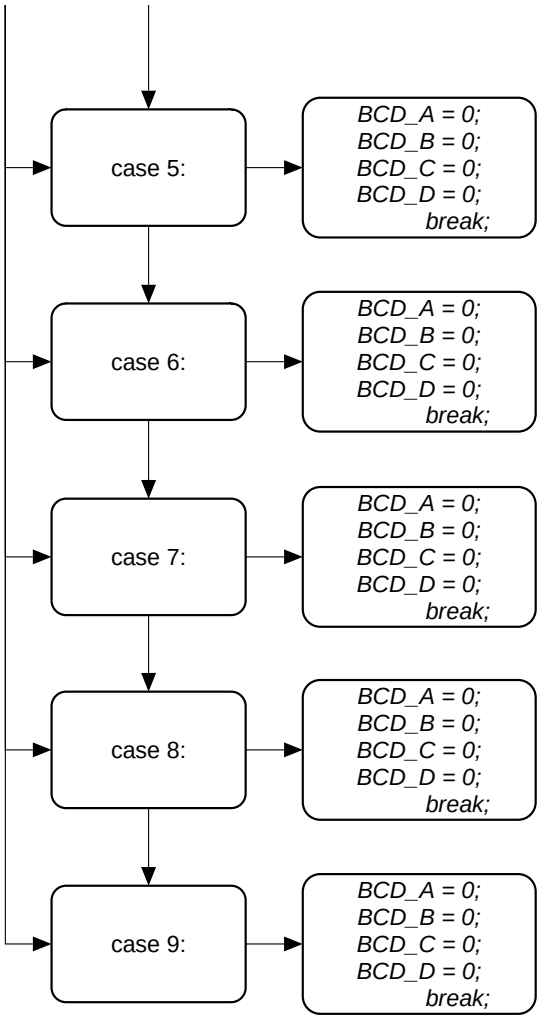

digitAt (int position, natural value)
switch (position)
case 3:
RETURN value % 10;
case 2:
RETURN (value / 10) % 10;
case 1:
RETURN (value / 100) % 10;
case 0:
RETURN (value / 1000) % 10;
default:
RETURN 0


__interrupt()
ISR(void)
if
(T0IF == true)
flag_1mS = true;
++timerTicks;
TMR0 = 131;
T0IF = 0;
if
((INTF == true) &&
((auxiliar & 1) == 0))
++flankChanges;
INTF = false;
else
INTF = false;
INTE = false;
if
(timerTicks ==
SampleTimeInMillis)
if
((auxiliar & 1) == 0)


if
(WindowSize <=
sampleTicks)
sampleTicks = 0;
window[sampleTicks] = flankChanges;
flankChanges = 0;
++sampleTicks;
timerTicks = 0;
++auxiliar;
if
(auxiliar == 5)
auxiliar = 0;
if
((auxiliar & 1) ==
0)
INTE = true;


showNumber
(int display, int
number)
switch (display)
case 0:
DEMUX_A = 0;
DEMUX_B = 0;
BREAK;
case 1:
DEMUX_A = 0;
DEMUX_B = 1;
BREAK;
case 2:
DEMUX_A = 1;
DEMUX_B = 0;
BREAK;
case 3:
DEMUX_A = 1;
DEMUX_B = 1;
BREAK;


switch (number)
case 0:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;
case 1:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;
case 2:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;
case 3:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;
case 4:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;


case 5:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;
case 6:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;
case 7:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;
case 8:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;
case 9:
BCD_A = 0;
BCD_B = 0;
BCD_C = 0;
BCD_D = 0;
break;

Código en C

Codigo principal

```
/*
 *  RECORDATORIO:
 *  Tabla de Verdad - relacion entre MUX y Frecuencia
 *  MUX_B  MUX_A // Frecuencia
 *  0      0        100 Hz a 1,920 MHz
 *  0      1        100 Hz a 480 KHz
 *  1      0        100 Hz a 120 KHz
 *  1      1        100 Hz a 30 KHz
 *
 * NOTA: el limite inferior de 100 Hz no es tan asi, puede llegar en teoria al divisor como limite inferior.
 * Es decir, para divisor de 64, frecuencia minima de 64 Hz o para divisor de 4, frecuencia minima de 4 Hz
 * Pero es teorico, en la practica puede variar un poco. Por eso lo mejor es minimo 100 Hz.
 */



#include <xc.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "ConfigurationBitsC.h"

#define _XTAL_FREQ          8000000
#define BCD_A               RA0
#define BCD_B               RA1
#define BCD_C               RA2
#define BCD_D               RA3
#define DEMUX_A             RA4
#define DEMUX_B             RB1
#define KHz                 RB2
#define MUX_A               RB3
#define MUX_B               RB4
#define MHz                 RB5
#define Displays            4
#define WindowSize          3
#define SampleTimeInMillis  166

//----------------------------
// Definición de variables:
typedef __uint24 natural; // Numeros enteros sin signo (16 bits).
typedef float real; // Numeros con decimal (24 bits).

//----------------------------
// Variables Globales:
natural flankChanges = 0;       // Cantidas de pulsos o flancos detectados por RB0.
natural window[WindowSize];     // Muestras para el calculo de la frecuencia.
int timerTicks = 0;             // Auxiliar para el traspaso a Hz.
int sampleTicks  = 0;           // Auxiliar para el tamaño de window (WindowSize).
bool flag_1mS = false;          // Booleano para detectar el paso de un 1 ms.
int auxiliar = 0;

//----------------------------
// Funcion de Interrupción:
void __interrupt() ISR(void) {
   // Clasificación de interrupciones:
   if (T0IF == true) { // Pregunto si se activo la interrupcion de 1ms
      flag_1mS = true;
      ++timerTicks;
      //--------------
      TMR0 = 131; // Contador del Timer cero (0).
      T0IF = 0;
    }
    if ((INTF == true) && ((auxiliar % 2) == 0)) {
      //Esto de aca solamente se ejecuta cuando el valor de auxiliar es par, esto es
      //para tener una ventana para calcular la frecuencia y otra para mostrar en display.
      //Tambien necesita que se haya activado la interrupcion por cambio de flanco
```

```
68          ++flankChanges;
69        //--------------
70        INTF = false;
71      }
72      else {
73        INTF = false;
74        INTE = false;
75      }
76
77
78    if (timerTicks == SampleTimeInMillis) {   // pregunto si la cantidad de interrupciones
79       //del timer 0 es igual a 166, esto es para dividir en 6 ventanas de tiempo a lo
80       //largo de 1 segundo
81       if ((auxiliar % 2) == 0) {
82          //Esto de aca solamente se ejecuta cuando el valor de auxiliar es par, esto es
83          //para tener una ventana para calcular la frecuencia y otra para mostrar en display.
84          if (WindowSize <= sampleTicks) {
85             // esto de aca hace que el valor de sample ticks loopee desde 0 a 5, y no se vaya
86             // a valores superiores a las ventanas de tiempo que tenemos en 1ms
87             sampleTicks = 0;
88          }
89          window[sampleTicks] = flankChanges; //esto de aca es fundamental, guarda
90          //las mediciones hasta el momento en una posicion del array window.
91          // Si se suman todos los valores de window, se obtienen la cantidad de
92          // cambios de flancos
93          flankChanges = 0;
94          ++sampleTicks;
95          timerTicks = 0;
96       }
97       ++auxiliar;
98       if (auxiliar == 5) {
99          auxiliar = 0;
100       }
101     }
102     if ((auxiliar % 2) == 0) {
103       INTE = true;
104   }
105 }
106
107 void configuration(void) {
108     KHz = false;
109     MHz = false;
110    //--------------
111     TRISB = 0x19; //0001-1001 // el truco para entender hexa es saber que
112    // 1 -> 0001
113    // 9 -> 1001
114    // si al hexa le paso 0x19, es lo mismo que pasar 0b 0001 1001 como haciamos en binario
115     PORTB = 0x00;
116     TRISA = 0x00;
117     PORTA = 0x00;
118     //nRBPU = 0;
119     //--------------
120     GIE = 1;
121     //--------------
122    T0IE = 1; // Habilito la interrupcion por Timer cero (0).
123     T0CS = 0; // Aumenta el Timer por cada ciclo de intruccion.
124     //T0IF = 1; // Flag que se activa cuando se desborda el contador del Timer cero (0).
125     TMR0 = 131; // Contador del Timer cero (0).
126     PSA = 0; // Asigno la preescala al Timer cero (0).
127     PS0 = 1; // Preescala = 16 (veces).
128     PS1 = 1;
129     PS2 = 0;
130     //--------------
131     INTE = 1; // Habilito el detector de flancos.
132     INTEDG = 1; // Habilito el flanco ascendente.
133     //INTF = 1; // Flag que se activa cuando ocurre una interrupcion por RB0.
134 }
135
136 void showNumber(int display, int number) {
137    /*
138     * Esta funcion recibe dos argumentos, uno es a que display se quiere imprimir,
```

```
139       * otro es que numero se quiere imprimir
140       */
141       switch (display) {
142           case 0:
143               DEMUX_A = 0;
144               DEMUX_B = 0;
145               break;
146           case 1:
147               DEMUX_A = 0;
148               DEMUX_B = 1;
149               break;
150           case 2:
151               DEMUX_A = 1;
152               DEMUX_B = 0;
153               break;
154           case 3:
155               DEMUX_A = 1;
156               DEMUX_B = 1;
157               break;
158       }
159       switch (number) {
160           case 0:
161               BCD_A = 0;
162               BCD_B = 0;
163               BCD_C = 0;
164               BCD_D = 0;
165               break;
166           case 1:
167               BCD_A = 0;
168               BCD_B = 0;
169               BCD_C = 0;
170               BCD_D = 1;
171               break;
172           case 2:
173               BCD_A = 0;
174               BCD_B = 0;
175               BCD_C = 1;
176               BCD_D = 0;
177               break;
178           case 3:
179               BCD_A = 0;
180               BCD_B = 0;
181               BCD_C = 1;
182               BCD_D = 1;
183               break;
184           case 4:
185               BCD_A = 0;
186               BCD_B = 1;
187               BCD_C = 0;
188               BCD_D = 0;
189               break;
190           case 5:
191               BCD_A = 0;
192               BCD_B = 1;
193               BCD_C = 0;
194               BCD_D = 1;
195               break;
196           case 6:
197               BCD_A = 0;
198               BCD_B = 1;
199               BCD_C = 1;
200               BCD_D = 0;
201               break;
202           case 7:
203               BCD_A = 0;
204               BCD_B = 1;
205               BCD_C = 1;
206               BCD_D = 1;
207               break;
208           case 8:
209               BCD_A = 1;
```

```
210             BCD_B = 0;
211             BCD_C = 0;
212             BCD_D = 0;
213             break;
214         case 9:
215             BCD_A = 1;
216             BCD_B = 0;
217             BCD_C = 0;
218             BCD_D = 1;
219             break;
220     }
221 }
222
223 int digitAt(int position, natural value) {
224    /*
225     * Esta funcion hace dos cosas:
226     * con % 10 saca el resto,
227     * y con el (value / 10,100, o 1000) hace que se puedan imprimir valores
228     * con el formato 1000 en el display,
229     * o sea, con case 0 pasa un 2000 a 2, y te permite imprimir ese 2 en la posicion
230     * del 1000 en el display
231     */
232     switch (position) {
233         case 3: return value % 10;
234         case 2: return (value / 10) % 10;
235         case 1: return (value / 100) % 10;
236         case 0: return (value / 1000) % 10;
237         default: return 0;
238     }
239 }
240
241 real average(natural values[], int length) {
242    /* esta funcion hace un promedio entre todos los valores de un array, como
243    * argumentos se necesita pasar el array y la longitud del mismo
244    *
245    * La misma retorna el promedio
246    */
247
248     natural sum = 0;
249     for (int i = 0; i < length; ++i) {
250         sum += values[i];
251     }
252     return ((real) sum) / length;
253 }
254
255 natural toMHzOrToKHzOrToHz(real frequency) {
256    /*
257     * Esta funcion recibe como argumento una frecuencia en Hz, y
258     * calcula si debe ser retornada como MHz, KHz o Hz
259     *
260     * Tambien es la encargada de prender y apagar los LED's de MHz y KHz
261     */
262     if (MHz == true) {
263         KHz = false;
264     }
265    if ((frequency > 9999) && (frequency <= 999999)) { // si la frecuencia es mayor a lo
266       // que puede mostrar el display en Hz, y no llega a ser un Mhz, prende el led de
267       // KHz y retorna la frecuencia en KHz
268       KHz = true;
269       return (natural) frequency / 1000;
270    }
271     else if (frequency > 999999) {
272       // Si la frecuencia es mayor a lo que puede mostrar el display en KHz, es un MHz, entonces
273       // prende el led de MHz y retorna el valor en MHz
274       MHz = true;
275       return ((natural) frequency) / ((natural) 1000000);
276     }
277     else {
278       // en caso de que la frecuencia no llega a ser ni un KHz, ni un MHz, apaga los leds
279       // de KHz y MHz y retorna la frecuencia
280       KHz = false;
```

```
281         MHz = false;
282         return (natural) frequency;
283     }
284 }
285 
286 int detectDivisor() {
287     /*
288      * esta funcion es fundamental para el divisor de frecuencia,
289      * detecta si se tocaron los botones MUX_A y/ o MUX_B, y en base a eso
290      * sabe internamente por cuanto se esta dividiendo la frecuencia.
291      *
292      * Este valor de divisor se usa internamente para luego multiplicar la frecuencia
293      * calculada por este valor, antes de pasarlo al display LCD
294      */
295     int divisor;
296     if ((!MUX_A) && (!MUX_B)) {
297         divisor = 64;
298     }
299     else if ((MUX_A) && (!MUX_B)) {
300         divisor = 16;
301     }
302     else if ((!MUX_A) && (MUX_B)) {
303         divisor = 4;
304     }
305     else {
306         divisor = 1;
307     }
308     return divisor;
309 }
310 
311 //-------------------------------------------------------------------------------------
312 // Función Principal:
313 int main() {
314     configuration();
315     //----------------------------
316     // Declaracion de variables:
317     real samplesAverage = 0;
318     real frequency = 0;
319     int position = 0;
320     int digit = 0;
321     int divisor = 1;
322     for (int i = 0; i < WindowSize; ++i) {
323         window[i] = 0;
324     }
325 
326     //----------------------------
327     while (true) {
328     // Codigo para actualizar samplesAverage (promedio) cada 322 mS.
329        if ((auxiliar % 2) == 1) {
330           /*
331           * Esto se ejecuta en las ventanas de tiempo en las que no se esta calculando
332           * la frecuencia actual
333           */
334           divisor = detectDivisor();
335           samplesAverage = average(window, WindowSize);
336           frequency = ((samplesAverage) * 3000) / SampleTimeInMillis; // Convierto en Hz.
337        }
338        //------------------------
339        // Codigo para actualizar el display cada 1 mS.
340        if (flag_1mS == true) {
341           flag_1mS = false;
342           // la siguiente linea calcula la frecuencia que hay que imprimir en el display
343           natural frequencyForDisplay = toMHzOrToKHzOrToHz(frequency * divisor);
344           // la siguiente calcule el valor que corresponde imprimir en el display,
345           // en base a la posicion que hay que imprimir
346           digit = digitAt(position, frequencyForDisplay);
347           // la siguiente linea va a imprimir en el display el numero digit, en la posicion position
348           showNumber(position, digit);
349           // la siguiente linea va a permitir que se imprima en los 4 displays LCD, se va a ir
350           // loopeando position por position, el mismo va a loopear entre 0 y 3
351           ++position;
```

```
352             if (Displays <= position) {
353                 position = 0;
354             }
355         }
356         //------------------------
357     }
358      return (EXIT_SUCCESS);
359 }
```

Bitacora

Empezamos este código armando las funciones principales del proyecto. Las funciones que nos resultaron más fundamentales fueron digitAt() y detectDivisor(). Tambien usamos typedef para definir nuestro propio tipo de dato.

Algo de lo que nos dimos cuenta fue de que a mayor cantidad de mediciones, mayor precisión de la medición

Simulaciones en Proteus

C

- https://www.loom.com/share/27a263f557e74fb1976f201afc8397d8