Customer counter and Watch

Final Project Report

December 7, 2006

E155

Jordan Bonnet and Benoit Courtade

Abstract:

The shopkeepers of a growing store have decided to count the number of customers visiting each day in order to determine the variation of the number of customers according to the time. Moreover, they want to install a new system of alarm based on the speed of eventual robbers. This device will be positioned in a corridor in which only one person may enter or exit.

For such a requirement, the device will consist in a PIC18F452 that receives signals from detectors. The PIC will determine the speed of each person and trigger an alarm if a potential robber is running. The data of the attendance of the store, as well as the warnings, will be stored in memory. Also, a FPGA linked to the PIC will receive the parameters for the device from a keypad. Finally, the measures of the device will be displayed on a LCD screen and stored in memory for statistic data.

Introduction

The ‘customer counter and watch’ is a device whose purpose is to store the current number of customers of a store in the Flash memory of the PIC according to the time. Consequently, the shopkeepers may determine whether they should either hire or fire employees, or change their schedules. The number of customers according to the time may displayed as a plot on a computer using software programs such as Excel. The curves obtained may saved or printed.

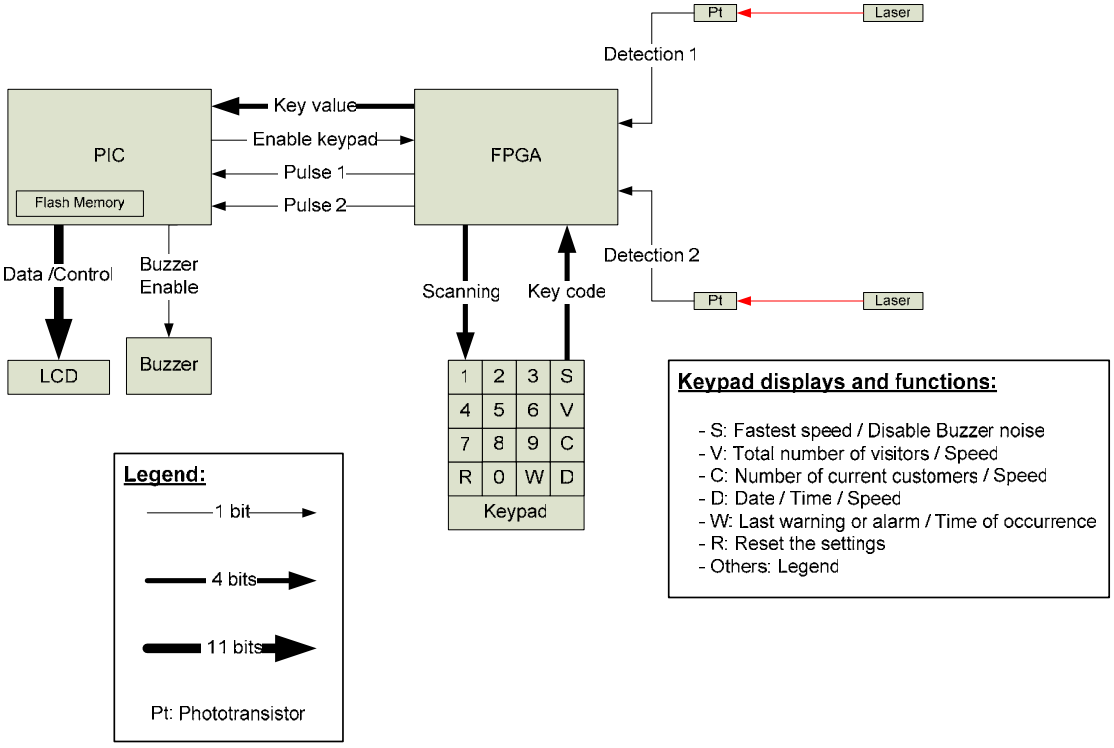


Figure 1: Block Diagram

Two lasers with two respective phototransistors make it possible to detect that someone is either entering or exiting the store. Thus, the counter for the current number of customers in the store may either incremented or decremented. In addition, the total number of entrances may also be counted. The output signals of the phototransistors, which are at a low level when the lasers are pointing on it and high level if not, are sent to the FPGA which processes the signals. As hands passing across the field of detection are noises, it detects whether the detection is long enough to be considered as a person. If the signal remains high at least for 70 ms, a pulse is sent to the PIC.

The device uses the Timer0 of the PIC to generate a clock so that time and date may be displayed on the LCD. The Timer0 is also used to determine the amount of time spent between the two detection pulses sent by the FPGA when someone is entering or exiting the store. Given the length between the two phototransistors, the speed of the person passing across the field of detection may be estimated.

The device includes a warning mode and an alarm mode. The warning is enabled by the PIC either when a customer is entering too quickly into the store or when the current number of customers is too large. These two events make respective LEDs switch on. The alarm is enabled either when a customer is exiting the store too quickly in ‘day mode’ or when a person is detected by the sensors, while the store is closed, in ‘night mode’. The alarm event is indicated by lighting a LED and by activating a buzzer that may be disabled by pressing the S key of the keypad.

The LCD module is used during the initialization or at any moment to set the parameters of the device: date, time, mode (night or day), limit of number of customers, speed limit, and delay. A maximum of ten customers may be between the two lasers at the same time. Consequently, the detection time of each sensor is stored for an amount of time equal to the period of delay, making it possible to avoid counting errors.

New Hardware

A LCD module permits the users to have an interface to set all the parameters and to get the inputs of the device. The LCD module used is the CrystalFontz CFAH2002A available in the Microprocessor's room.

The supply voltage for logic must be in a range between 4.5 V and 5.5 V. As the supply voltage for LCD ($V_{DD} – V_0$ has to be 4.5 V), it is an easy solution to connect V_0 to ground and to set the supply voltage at 4.5 V. Consequently, it would not be necessary to get a 0.5 V voltage for V_0.

It is useful to connect the Data Bus to PORTD (connected to LEDs) because it simplifies the debugging. The $D_{<7:0>}$ bus makes it possible to transmit either the instructions or the data to the LCD. The Register Selector (RS) indicates to the LCD whether you are writing into the Data Register (DR) when RS=1 or into the Instruction Register (IR) when RS=0.

Unless you want to read data from the Display Data RAM (DDRAM) or Characters Generator RAM (CGRAM), or read the busy flag and address counter, you do not need the Read part of the Read/Write (R/W) signal. However, to display characters, you need the Write signal (R/W=0) in order to write data to the DDRAM. While writing instructions to the IR, the R/W signal must also be set to 0.

After each instruction or data sent on the bus, you must pulse the enable signal. That is, you must set it high for a minimum amount of time indicated in the Timing Characteristics section of the data sheet. The new instruction is executed when the enable signal returns to low. Then, an execution time, indicated in the data sheet, is necessary for each instruction. If you do not want to see the display changing between two messages, you should use execution time as close to the time indicated in the data sheet as possible.

Schematics

The schematic of the breadboard is as following:

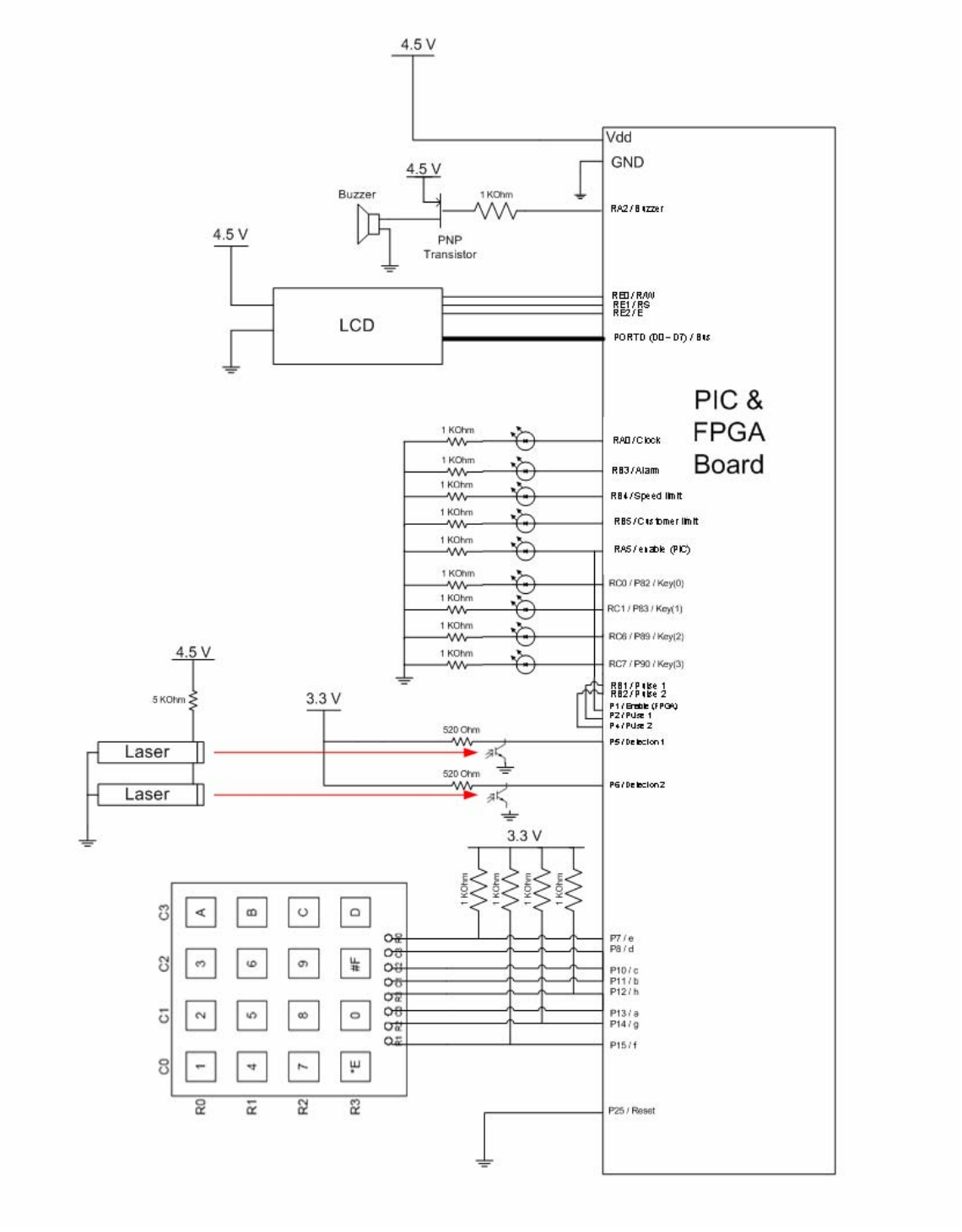


Figure 2: Schematic of the breadboard

Microcontroller Design

In our project, we use a PIC18F452 that is the master of the system. It receives inputs from the FPGA in order to set up the values like the time, the date and establish parameters. Then, it detects changes on the sensors, calculates the speed of the customers, determines warning levels and stores customer data. Finally, the PIC drives the LCD screen displaying the interface.

The design is based on an infinite loop which is interrupted every second by a high priority interrupt that is the reference for the time. Moreover, when there is a change on the signal coming from one of the receivers, as well as a reset from the keypad, there is a low priority interrupt in order to process the information received.

- *void main(void):*

  This function initializes functions for the date and the parameters of the device that the user can see through the LCD and the keypad. It sets up low priority interrupts on falling edge of the signal from the phototransistors or a reset from the keypad (on RB<2:0>) and sets up a high priority interrupt when there is an overflow of the timer 0. Then, this function sets the timers 1, 2 and 3 for timing measurements for the other functions. Finally, it runs an infinite loop displaying the clock and the different screens on the LCD.

- *void hisr(void):*

  This function occurs each second, incrementing the clock and call functions to write the Flash Program Memory each minute when there are changes in the frequenting of the store.

- *void lisr(void):*

  A detected edge on a signal coming from the sensors means that a customer has crossed the locational point of the receiver. Depending on the case, this function calls other functions in order to process the speed of the customer and also updates the data.

- *void create_tablein(void),void create_tableout(void):*

These functions initialize the table for the data of time of the two different sensors in and out at 0xFF. In the library '*stdlib'* of the compiler mcc18, there is no function "*memory allocation"* defined; consequently, a static table contains pointers on data of time for the sensors.

- *void storein(void),void storeout(void):*

It is assumed that the people move in only one direction at time and that a maximum of 10 persons may be between the two detectors, so that ten people can be detected by a first sensor before being detected by the other sensor. The data is stored in the tables seen above.

The time of each customer crossing the signal of a detector is stored in the table of the corresponding sensor at a certain step out of ten. Then, when the customer is detected by the other sensor, the data is also stored in the other table at the same step. At each increment, the table of the other sensor is tested at the same step and the data is either processed or not, as showed in the 'Detection Data Storage' figure 3. When the step reaches nine, it is reset to zero.

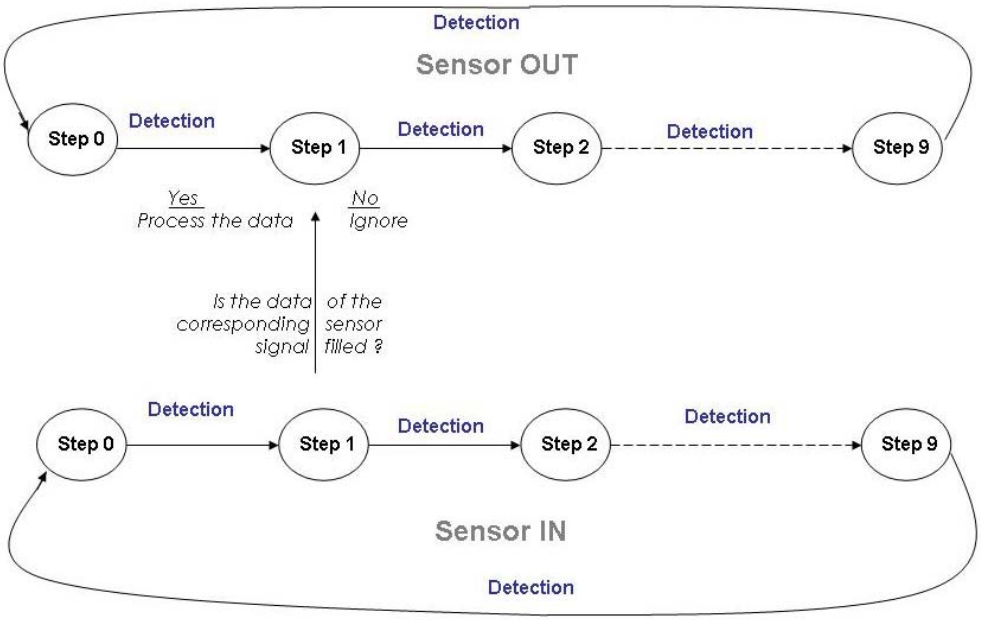


Figure 3: Detection Data Storage

- *void convert(int direction,int step):*

The data of each customer detected contains the value of the seconds and the value of *timer 0* for each sensor. If the function is called, it converts the data in speed according to the different possible cases. Given the problem of reference units across countries, the displayed speed is in meters per second.

- void warning (void):

There are four warnings. When someone is running out of the store, the alarm starts (Buzzer) and a red LED is switched (on RB3). When someone entering is running, there is a green LED switched (on RB4). When there are too many people in the store, a yellow LED is switched (on RB5). These warnings are also displayed on the LCD screen with the time when they occurred, except for the *"Too many people"* warning which is displayed only once before the counter decreases under the limit.

- *void storedata(void):*

This function stores the data in the Flash Program Memory. The data is stored each minute if there are warnings or changes in the number of current customers in the store. It contains the high and low bytes of the number of customers in the store, the minute, hour, day, month, speed limit for the detection and the eventual warnings. The information stored contains 8 bytes.

After each writing of the memory, the table pointer is incremented of 16 bytes instead of 8 bytes as it is then easier to read and process the data for statistics about the store or the µP's Lab in this way.

- *void write(unsigned short long ad, char * data):*

This function writes the Flash Program Memory with the content of the table at the program memory's address indicated by the function storedata(). It writes the values to store in holding registers, values which are then written in the Flash Program Memory. The program's code stops at the address 0x317F, there are thus 0x4870 = 20 kB of

memory available until the address 0x7FFF, which is equivalent to more than 40 days if the data is stored each ten minutes and the store opened 10 hours a day.

- The functions read_pm() and erase_pm(unsigned short long ad) are not used but are useful to learn how to write the memory.

- *void data_fpga(void), void check_fpga(void), int init (void):*

See detail in PIC-FPGA protocol

At first, we intended to use the I²C function of the PIC with an external serial EEPROM and we had to find another way to communicate with the FPGA.

In order to acquire values, the PIC is linked to the FPGA that sends values typed on the keypad when it is enabled by the PIC (RA5). The function data_fpga() updates the value coming through 4 wires (RC<7:6>:RC<1:0>). When there is no value, the incoming value is 0xF. The transmission is made of a succession of 0xF and numbers, and the function check_fpga() waits for the appropriated values to be typed on the keypad.

Each byte containing the data is made of two numbers. Instead of shifting the first number of four bits, the incoming number is converted in a decimal value, then multiplied by ten and added to the second number by the function init().

- *void reset(void) :*

This function was created to solve the problems encountered when a customer is detected by a sensor, changes his mind between the two sensors and then exists the store without being detected by the other detector. It processes to a reset of the data of the sensors in and out after a delay typed during the initializing period : *void init_parameters(void)*.

- *void init_parameters(void) :*

This function allows the user to enter the following parameters: month, day, hour, minute, mode day or night, speed limit for the speed detection, the threshold number of customers before the warning and the delay before resetting the sensors' tables.

Finally, we experienced some problems in compiling because the size of all our initialized values was too important. It appeared that the variables initialized are stored in a memory called idata (udata for the other values non-initialized) and that we had to divide this memory in different sections so that the variables fit the space in these sections. It is possible that we divided the memory a bit too much, but it works with the code: “pragma idata name_of_variable” inserted before the variable.

LCD Design

The design of the LCD is made in such a way that it is easy to display different messages and easy to integrate to the first part of the PIC program. The idea is to type the message we want to display and store it in a string structure as following:

messageXX[ ] = “Message to display $r Variable: $v”;

When the pointer of the message scans each character in order to store it into a global variable named message[ ], the variables or functions are detected with the ‘$’ character. The function that stores the characters is easy to modify in order to add variables to display. While calling the function that stores the digits of the variables into the message[ ] string, it is possible to choose whether one wants insignificant zeros of the number to be displayed or not. Indeed, you can choose between displaying 100 as “100” or “0100” with a maximum of 4 digits. Functions can also be used or added, such as ‘$r’ that makes the display go to the second row.

FPGA Design

The hardware on the FPGA is divided in two parts: the keypad and the detection process.

1. Keypad

The FPGA makes it possible to send the hexadecimal code of the key pressed to the PIC when the enable signal is *set high.*

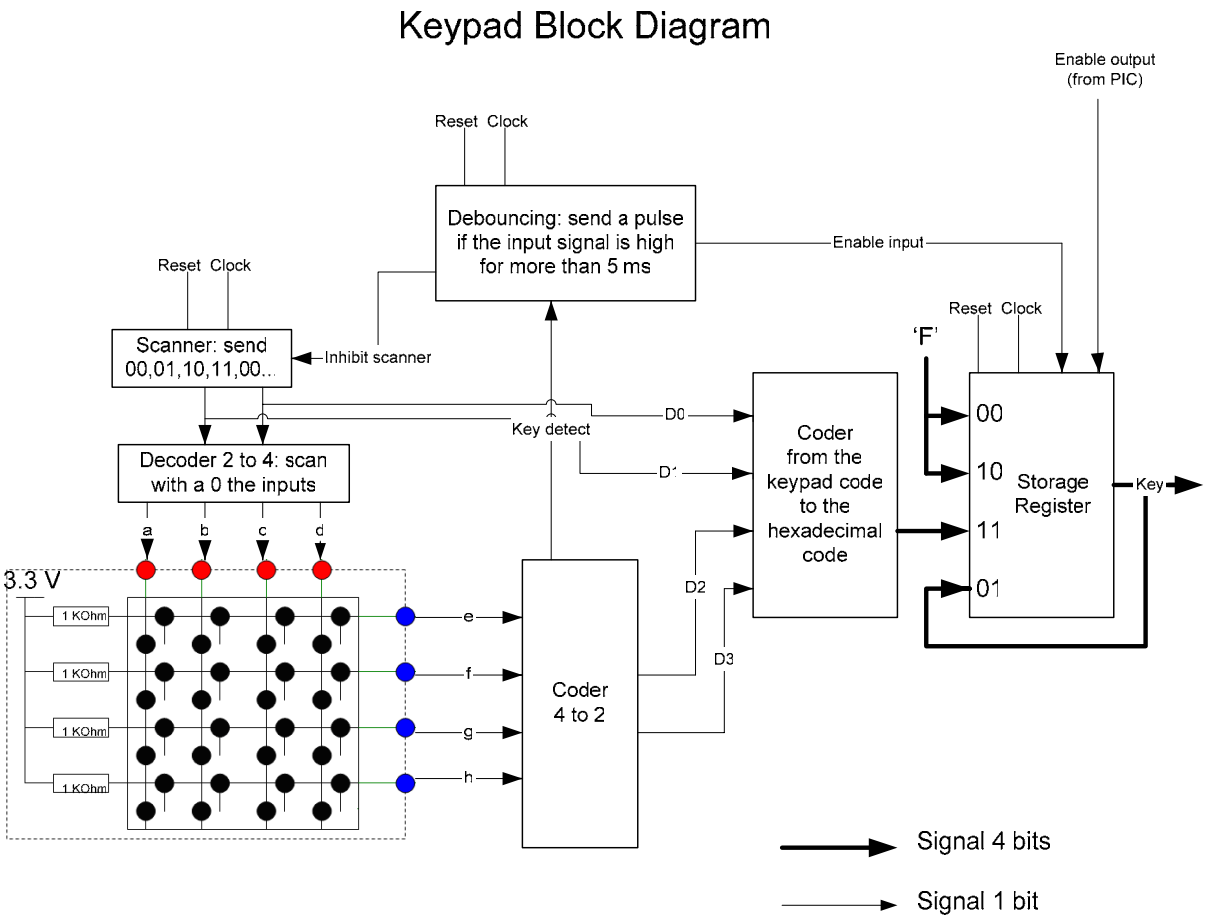


Figure 4: Block Diagram of the Keypad

The storage register module makes it possible for the FPGA and the PIC to communicate. 'Enable input' tells the storage register that the hexadecimal code of the key pressed is

available. First, ‘enable output’ is set high and the FPGA is sending ‘F’. When a key is pressed, the FPGA waits for ‘enable input’ to be ‘1’ and then sends the new value to the PIC. The latter copies the value and sets ‘enable output’ to ‘0’, which makes the FPGA send ‘F’. As the PIC receives ‘F’ now, it sets ‘enable output’ to ‘1’, which makes the storage register module in the ‘01’ mode. This mode makes the FPGA send the last value stored (‘F’) until a new key is pressed. This procedure makes it possible for the PIC to determine when a new key is pressed, even if it is the same one.

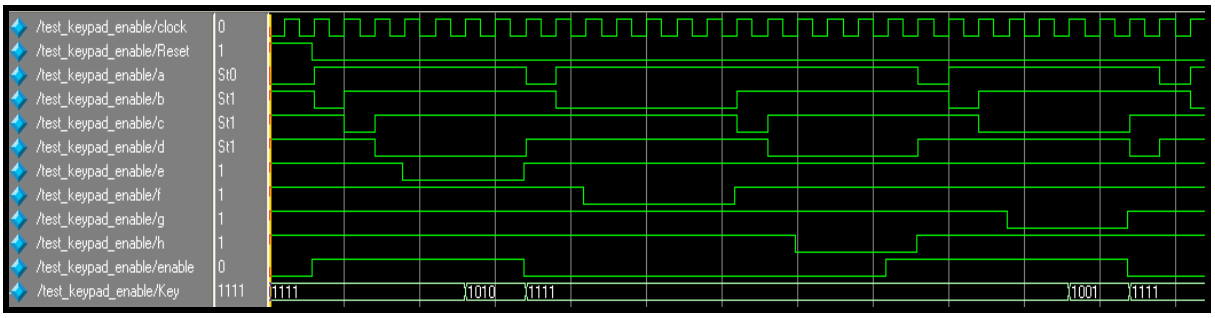


Figure 5: Chronogram of the communication between the FPGA and the PIC

2. Sensors debouncing

The two signals coming from the phototransistors are processed by the FPGA in order to prevent anything other than a person from being considered as so. In particular, the hands passing across the field of detection could be detected as another person. We realized that a relatively slow movement of the hands can reach a 50 ms signal width, whereas someone running quickly is around a 75 ms signal width. Consequently, the FPGA sends a pulse to the PIC when the width of the signal from the detectors is greater than 70 ms.

Results

1. EEPROM Memory

In our first project proposal, we intended to store the data in an external serial EEPROM. The idea was that the user would take only the EEPROM after use, and plug it in a parallel port. However, we encountered problems because of the protections of the external ports with Windows XP in one hand, and on the other hand because of difficulties to drive the EEPROM. Indeed, the MSSP protocol of the PIC was supposed to set interruption flags and send some bits automatically with a buffer, but these things randomly happened. Moreover, we were not able to find the synchronization time to release the Serial Data Wire at high impedance to receive the acknowledge bit of the EEPROM.

2. Flash Program Memory

Looking for another place to store the data, we chose to put it in the Flash Program Memory, in the free space after the code's program. We chosethis type of memory because we needed several Kbytes of space; furthermore, we wanted to be able to read the data from a computer. We had already used data in the program memory during the previous labs, but it was significantly more difficult to write in it. The values are first written in 8 one byte holding registers before being copied by block of 8 bytes in the program memory. Thus, the information about the customers can be read through MPLAB and exported to a file to plot curves of the attendance. This purpose was one of the final goals of this project, and we are really happy to be able to display a small curve of the attendance.

The Flash Program Memory values are readable in MPLAB in the Program Memory's view, selecting the option Opcode Hex.

3. Status

The final device of our project is composed of a main utility board covered by a blue paper cover that is linked to two lasers pointer and two phototransistors. The device performs the functions described in this report and works well, as long as people do not stay in front of the lasers while other people are walking. In this case, the device has to wait for a delay entered at the beginning before being readjusted.

References

Datasheet PIC18F452's, Microchip,

http://ww1.microchip.com/downloads/en/devicedoc/39564b.pdf

Environment MPLAB IDE, Microchip,

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002

LCD CFAH2002A Datasheet, CrystalFontz,

http://www.crystalfontz.com/products/2002a/CFAH2002AYYHJP.pdf

Parts List

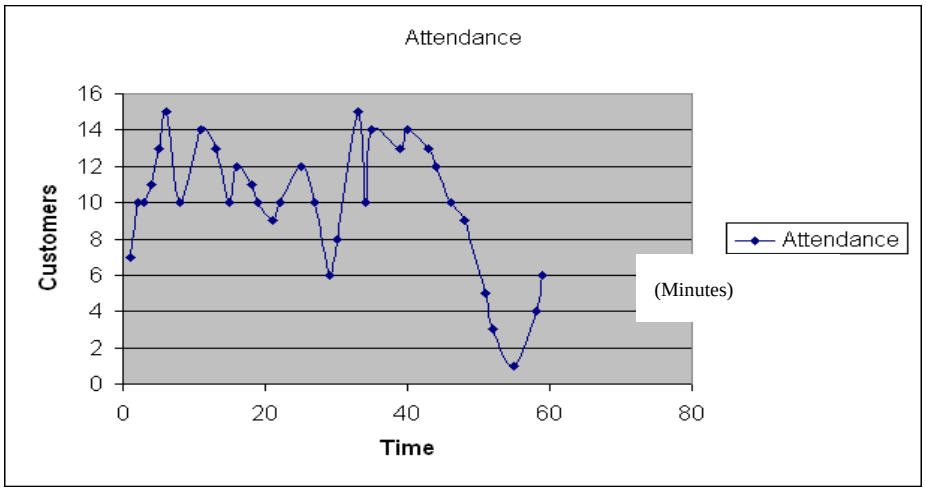
Parts	Source	Vendor Part #	Price	Q^{ty}
Microcontrolleur	Utility Board	Microchip PIC18F452	_	1
FPGA	Utility Board	SPARTAN 3	_	1
LCD	µP's Room	FAH2002A	_	1
Laser Pointers	All Electronics	LP-506	$4.75	2
DC Buzzer-3V	R-Vac	_	$3.61	1
Phototransistors	_	_	_	2
Divers Components (Resistors, Capacitors, Keypad, LED)	µP's Room	_	_	_

Appendices

1. Statistics

Month	Day	Hour	Minute	Speed limit	Warning	CounterH	CounterL	Minute	Counter
12	4	21	1	3	0	0	7	1	07
12	4	21	2	3	0	0	10	2	010
12	4	21	3	3	0	0	10	3	010
12	4	21	4	3	0	0	11	4	011
12	4	21	5	3	0	0	13	5	013
12	4	21	6	3	0	0	15	6	015
12	4	21	8	3	0	0	10	8	010
12	4	21	11	3	0	0	14	11	014
12	4	21	13	3	0	0	13	13	013
12	4	21	15	3	0	0	10	15	010
12	4	21	16	3	0	0	12	16	012
12	4	21	18	3	0	0	11	18	011
12	4	21	19	3	0	0	10	19	010
12	4	21	21	3	0	0	9	21	09
12	4	21	22	3	0	0	10	22	010
12	4	21	25	3	0	0	12	25	012
12	4	21	27	3	0	0	10	27	010
12	4	21	29	3	0	0	6	29	06
12	4	21	30	3	0	0	8	30	08
12	4	21	33	3	0	0	15	33	015
12	4	21	34	3	0	0	10	34	010
12	4	21	35	3	0	0	14	35	014
12	4	21	39	3	0	0	13	39	013
12	4	21	40	3	0	0	14	40	014
12	4	21	43	3	0	0	13	43	013
12	4	21	44	3	0	0	12	44	012
12	4	21	46	3	0	0	10	46	010
12	4	21	48	3	0	0	9	48	09
12	4	21	51	3	0	0	5	51	05
12	4	21	52	3	0	0	3	52	03
12	4	21	55	3	0	0	1	55	01
12	4	21	58	3	0	0	4	58	04
12	4	21	59	3	0	0	6	59	06

Figure 6: Excel Table of the Data stored and corresponding Plot



2. Pictures

Figure 7: Picture of the Device with its cover

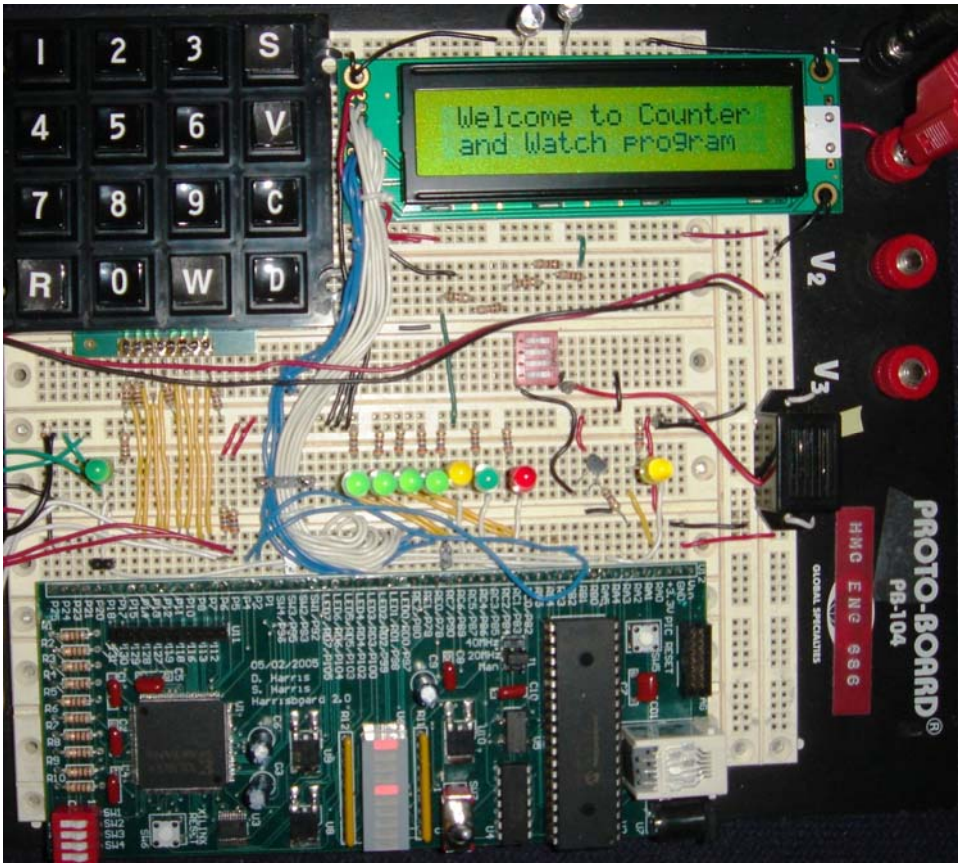


Figure 8: Picture of the Device without its cover

Figure 9: Setup during the Presentation day

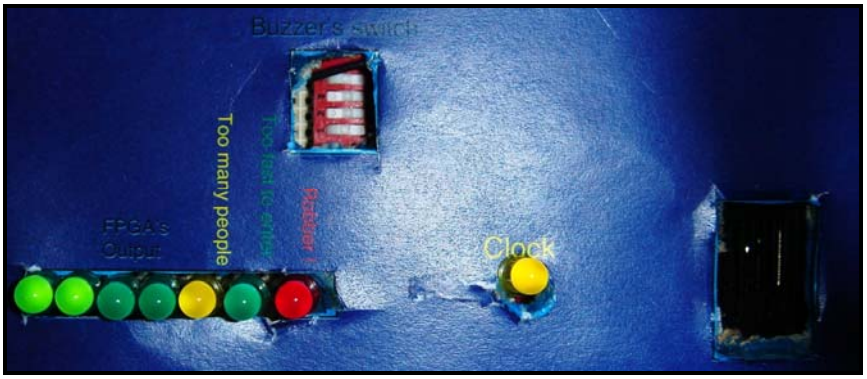


Figure 10: Information given by the LEDs

Figure 11: Keypad

Figure 12: Examples of Displays

3. Microcontroller code

```
/* 1. Library */



#include <p18F452.h>
#include <timers.h>
#include <stdio.h>
//#include <string.h>
//#include <stdlib.h>

unsigned short long adress_pm_end=0x3200;   //adress of the end of the program in Pgm memory
                                                                                    //has to be a
multiple of 64

//LCD constants
const int Init=0x30;
const int Interface=0x38;
const int D_mode=0x06;
const int D_off=0x08;
const int D_clear=0x01;
const int D_on=0x0C;
const int C_blink=0x0F;
const int Row1=0x00;
const int Row2=0x40;

//variables
int secondes=0;
int in=0;
int j=0,k=0;                                                    //j for in and k for out
int direction=0;
int test=0;
int counter=0;
char number=0;
char enable=0;
int init_d=0;
int number_d=0;
int number_temp=0x10;
int change=1;
int warn=0;
int warning_hours=0;
int warning_minutes=0;
int timo=0;
int timi=0;
```

```
int last_interupt=0;
int variation=0;
double best_speed=0;
char data[8];
unsigned short long adress_pm=0x3200;              //adress of the Pgm memory to write to
store data


#pragma idata message0
char message0[]="Total Visitors: $C  $rSpeed: $s* m/s      ";
#pragma idata message1
char message1[]="$D***          $T***$rSpeed: $s* m/s      ";
#pragma idata message2
char message2[]=" Welcome to Counter $r and Watch program  ";
#pragma idata message3
char message3[]="Date: 00/00         $r                    ";
#pragma idata message4
char message4[]="Time: 00:00         $rMilitary hour (0-24)";
#pragma idata message5
char message5[]="Mode Day:00/Night:01$rChoice: __          ";
#pragma idata message6
char message6[]="Speed limit: 0.0 m/s$rAdvised speed: 3 m/s";
#pragma idata message7
char message7[]="Maximum number of   $rcustomers: __       ";
#pragma idata message8
char message8[]="Delay: __ secondes  $r                    ";
#pragma idata message9
char message9[]="Customers: $c       $rSpeed: $s* m/s      ";
#pragma idata message10
char message10[]="D:Date    V:Visitors$rC:Current Customers ";
#pragma idata message11
char message11[]="WARNING!!! Customers$rlimit reached  $W***";
#pragma idata message12
char message12[]="WARNING!!! Speed    $rlimit reached  $W***";
#pragma idata message13
char message13[]="ALAAAAAAARRRRMMMM!!!$rRobber!!!      $W***";
#pragma idata message14
char message14[]="  Best Speed is :   $r    $B* m/s   !!!!  ";
#pragma idata warning_message
char warning_message[]="                                          ";

#pragma idata variables1
char message[]="                                            ";
int address=0x0000;
#pragma idata variables2
int i;
int t;
```

```
//Initialization LCD
int months=13,minutes=60,hours=24,days=32,delay=-1,mode=2,counter_max=-1,speed_limit_i=110;
double speed_limit;

//No Human initialization
//int months=0,minutes=0,hours=0,days=0,delay=10,mode=0,counter_max=1,speed_limit=3;

//Constant
double length=0.88;



//structures

//data for detector in
typedef struct
                      {
                      int timer;
                      int seconde;
                      }
                      detectionin;
typedef detectionin *watchin;

//malloc
detectionin din0,din1,din2,din3,din4,din5,din6,din7,din8,din9;
watchin  in0=&(din0),in1=&(din1),in2=&(din2),in3=&(din3),in4=&(din4);
watchin in5=&(din5),in6=&(din6),in7=&(din7),in8=&(din8),in9=&(din9);



//data for detector out
typedef struct
                      {
                      int timer;
                      int seconde;
                      }
                      detectionout;
typedef detectionout *watchout;
//malloc
detectionout dout0,dout1,dout2,dout3,dout4,dout5,dout6,dout7,dout8,dout9;
watchout out0=&(dout0),out1=&(dout1),out2=&(dout2),out3=&(dout3),out4=&(dout4);
watchout out5=&(dout5),out6=&(dout6),out7=&(dout7),out8=&(dout8),out9=&(dout9);

//reference for warning
typedef struct
                      {
                      double speed;
                      int way;
```

```
                    }
                    robber;
robber current;



//contains the adresses of the structures with the time of the detection of corresponding
sensor
watchin tablein[3];
watchout tableout[3];

int test0,test1,test2,test3,test4,test5,test6,test7,test8,test9;
int testo0,testo1,testo2,testo3,testo4,testo5,testo6,testo7,testo8,testo9;
int test;




//Function Prototypes
void main(void);
void lisr(void);
void hisr(void);
void storein(void);
void storeout(void);
void create_tablein(void);
void create_tableout(void);
void convert(int step);
void storedata(void);
void warning (void);
void write(void);
void data_fpga(void);
void check_fpga(void);
int init(void);
void reset(void);
void init_parameters(void);
void erase_pm(void);
void write_pm(unsigned short long ad, char * data);
char read_pm(unsigned short long ad);

/*LCD prototypes*/
void copy(char var_mess[],char dest_mess[]);
void main(void);
void time(int wait);
void Enpulse(void);
void Display_message(void);
void store_variable(int variable,int choice);
void Init_lcd(void);
int Rounding(double var_double);
void clear_display(void);
```

```
void blinking(void);
void set_digit(void);
void Display_on(void);
void go_address(void);
void Display_char(void);
```

```
/* 2. Program */
                      /******************************************/
                      /*            Benoit Courtade             */
                      /*             Jordan Bonnet              */
                      /*     Counter and Watch PIC program      */
                      /******************************************/

#include "library.h"

/*******************************************************************************/
/* set the interrupts high and low at the adresses 0x08 and 0x18 in the memory */
/*******************************************************************************/

#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh (void)
{
  _asm
   goto hisr
  _endasm
}
#pragma code

#pragma code low_vector=0x18
void low_interrupt(void)
{
	_asm
	goto lisr
	_endasm
}

#pragma code

/******************************************************/
/*                   Main function                    */
/******************************************************/

void main(void)
{

/******************************************************/
```

```
/*                Port configuration                */
/**************************************************/


       ADCON1=0x07;                                                                    //
digital inputs


       TRISA=0x00;                                                                     //
set PORTA as output
       PORTAbits.RA2=1;                                                                //
Clock on RA0, Buzzer on RA2 active at 0, Enable FPGA (Keypad) on RA5


       PORTB=0x00;
       // Warning LEDs on RB<5:3>
       TRISB=0b00000111;                                                               //
RB0 to change LCD parameters from FPGA

              // RB1 and RB2 for IR detectors Falling Edge Interrupt


       PORTC=0x00;
       // I2C for EEPROM on RC<4:3>
       TRISC=0b11000011;                                                               //
Data from FPGA on RC<7:6>:<1:0>

              // RC2 PWM to drive the IR emitters

       TRISD=0x00;                                                                     //
set PORTD as output for LCD data
       PORTD=0x00;
       // 8 bit data PORT


       TRISE=0x00;
       // PORTE output for LCD display
       PORTE=0x00;
       // Control LCD display







/*************************************************************************/
/*          Create tables containing the data of the sensors              */
/*************************************************************************/


       create_tableout();
       create_tablein();
       j=0;
       k=0;
       current.speed=0;


/*************************************************************************/
/*           Initialization of the interrupts for the sensors            */
/*************************************************************************/
```

```
                                        /* initialize interrupt */


       RCONbits.IPEN=1;                                                                //
enable priroty levels on interrupts

       INTCONbits.GIEH=1;                                                              //
enable high priority interrupts

       INTCONbits.GIEL=1;                                                              //
enable low priority interrupts


    /* initialise low priority interrupts when falling edge on RB1 and RB2 */


       INTCON2bits.INTEDG1=0;                                                  // enable
interrupt when falling edge on RB1

       INTCON2bits.INTEDG2=0;                                                  // enable
interrupt when falling edge on RB2

       INTCON3bits.INT1IP=0;                                                   // set as low
priority interrupt

       INTCON3bits.INT2IP=0;                                                   // set as low
priority interrupt

       INTCON3bits.INT1IF=0;                                                   // clear
interrupt flag on RB1

       INTCON3bits.INT2IF=0;                                                   // clear
interrupt flag on RB2

       INTCON3bits.INT1IE=1;                                                   // enable
interrupt on RB1

       INTCON3bits.INT2IE=1;                                                   // enable
interrupt on RB2


/************************************************************************/

/*                                          CLOCK
*/

/*
                                         */

/*
                                         */

/*                                      Timer0: 1 sec
                            */

/*
                                         */

/*         - 16-bit counter
                     */

/*         - prescale of 128, 4 cycles by incrementation                            */

/*             - 1 operation each 1/20,000,000
                  */

/*             - interrupt every 1 s
                     */

/*             - 1=1/fosc x 128 x 4 x number to count
                 */

/*             - counterFFFF=2^16
                            */

/*             - number to count is 39062=0x9896
                 */

/*             - 65536-number to count=>counter start=26474=0x6769                  */

/************************************************************************/
```

```
       INTCONbits.TMR0IE=1;                                                    // enable
interrupt when overflow on Timer 0
       INTCON2bits.TMR0IP=1;                                                   // set Timer
0 interrupt as high priority interrupt


                       /* initialize T0CON */


       T0CON=0x00;                                                                //
set the timer as a 16 bit counter, no prescale value
       T0CONbits.PSA=0;                                                           //
assign prescaler
       T0CONbits.T0PS2=1;                                                         //
set prescaler as 1:128
       T0CONbits.T0PS1=1;
       T0CONbits.T0PS0=0;
       T0CONbits.TMR0ON=1;                                                        //
enables timer0


/*********************************************************************/
/*                          SENSOR OUT TIMER                         */
/*
                                          */
/*
                                          */
/*                       Timer1: 0.104856 sec
        */
/*
                                          */
/*             - 16-bit counter
                            */
/*          - prescale of 8, 4 cycles by incrementation
        */
/*      - 1 operation each 1/20,000,000
               */
/*      - interrupt = 1/fosc x 8 x 4 x 0xFFFF
        */
/*      - counterFFFF=2^16
                    */
/*********************************************************************/


       PIR1bits.TMR1IF=0;                                                         //
clear interrupt flag of timer1
       PIE1bits.TMR1IE=0;                                                         //
disable interrupt when overflow on Timer 1
       IPR1bits.TMR1IP=0;                                                         //
set Timer 1 interrupt as low priority interrupt


                                         /* initialize T1CON */


       T1CON=0xB4;                                                                //
set the timer as a 16 bit counter, prescale value:8,counter off


/*********************************************************************/
/*                          SENSOR IN TIMER                          */
```

```
/*                                                                      */

/*                                                                      */

/*                      Timer3: 0.104856 sec
        */

/*                                                                      */

/*            - 16-bit counter
                            */

/*          - prescale of 8, 4 cycles by incrementation
        */

/*      - 1 operation each 1/20,000,000
                */

/*      - interrupt = 1/fosc x 8 x 4 x 0xFFFF
        */

/*      - counterFFFF=2^16
                        */

/************************************************************************/


       PIR2bits.TMR3IF=0;                                                              //
clear interrupt flag of timer 3

       PIE2bits.TMR3IE=0;                                                              //
disable interrupt when overflow on Timer 3

       IPR2bits.TMR3IP=0;                                                              //
set Timer 3 interrupt as low priority interrupt


                                          /* initialize T3CON */


       T3CON=0xB4;
       // set the timer as a 16 bit counter, prescale value:8, timer off


/************************************************************************/

/*                               LCD                                    */

/*                                                                      */

/*                                                                      */

/*                      Timer2:  sec
        */

/*                                                                      */

/*            - 8-bit counter
                                */

/*          - prescale of 16, 4 cycles by incrementation                                */


/*      - 1 operation each 1/20,000,000 = 50 ns
*/

/*      - one incrementation = 0.05 us * 4 * 16 = 0.0000032 s =    3.2 us   */

/*      - number of incrementations = time wanted (us)/3.2                      */

/************************************************************************/

                                          /* initialize T2CON */


       T2CON=0x02;
       // set the timer with a prescale value:16, timer off
```

```
	TMR2=0x00;
	// initialize Timer 2 register



/*************************************************************************/
/*            Interface initialization with LCD and Keypad                */
/*************************************************************************/


//	erase_pm();
	//erase the end of the program memory to store data later
	init_parameters();                                                    //
User enters parameters for the device



/*************************************************************************/
/*                          Change Parameters                           */
/*************************************************************************/


                         /* Interrupt 0 */


	INTCON2bits.INTEDG0=0;                                                // enable
interrupt when falling edge on RB0
	INTCONbits.INT0IF=0;                                                  // clear
interrupt flag on RB0
	INTCONbits.INT0IE=1;                                                  // enable
interrupt on RB0


/*************************************************************************/
/*                                RUN
            */
/*************************************************************************/


	while(1)
	{
                                        /* Clock Display */


		if (secondes%2==0)
			PORTA=0xFE & PORTA;


		if (secondes%2==1)
			PORTA=0x01 | PORTA;



                        /* Messages display */



		data_fpga();                                                  //
copy into number the value of the last key pressed on the keypad
		if(number_temp!=number)                                       //
change the display if the user presses another key (Mode TV, Mode C, Mode D or Mode W)
```

```
                    change=1;


              if(change==1)                                                    //
the display must be updated
              {
                    change=0;
       // set change to 0 so that the display is only updated when a change occurred
                    switch(number)
                    {


                           case 0x0A:            PORTAbits.RA2=1;             // if A is
pressed => Stop Buzzer and also display the best speed
                                                        clear_display();
                                                        copy(message14,message);
                                                        Display_message();
                                                        number_temp=number;
                                                        break;
                           case 0x0D:            clear_display();             // if D is
pressed => Display Date, time and speed
                                                        copy(message1,message);
                                                        Display_message();
                                                        number_temp=number;
                                                        break;
                           case 0x0B:            clear_display();             // if TV (the
actual B) is pressed => Display Total number of Visitors and speed
                                                        copy(message0,message);
                                                        Display_message();
                                                        number_temp=number;
                                                        break;
                           case 0x0C:            clear_display();             // if C is
pressed => Display the number of current Customers and speed
                                                        copy(message9,message);
                                                        Display_message();
                                                        number_temp=number;
                                                        break;
                           case 0x0F:            clear_display();             // if W (the
actual F) is pressed => Display the last Warning
                                                        copy(warning_message,message);
                                                        Display_message();
                                                        number_temp=number;
                                                        break;
                           default:              clear_display();             // if any
other key is pressed => Display Keypad Legend
                                                        copy(message10,message);
                                                        Display_message();
                                                        number_temp=number;
                                                        break;
                    }
              }
       }
```

```
}


/*****************************************************************************/
/*                            END OF RUN
        */
/*****************************************************************************/




/*****************************************************************************/
/*                           Low Interrupt
*/
/*****************************************************************************/


#pragma interruptlow lisr
void lisr(void)
{
       if(INTCONbits.INT0IF==1)                                                 // Interrupt
0 enables low priority interrupt
       {
              INTCON3bits.INT1IF=0;                                             // through
INT1
              INTCONbits.INT0IF=0;                                              // clear INT0
interrupt flag
              PORTAbits.RA5=0;                                                         //
disable RA5 (disable keypad)
              init_parameters();                                                       //
retype parameters
              INTCONbits.INT0IE=1;                                              // reenable
INT0
       }


       if(PIR1bits.TMR1IF==1)
              {
              /* 0.1 sec since detector out has been disabled, reenable this sensor */
              T1CON=T1CON & 0xFE;                                                      //
clear bit 0 =>timer 1 off
              PIR1bits.TMR1IF=0;                                                       //
clear IF
              TMR1L=0x00;
              TMR1H=0x00;
              T1CON=T1CON | 0x01;                                                      //
Timer1 on
              PIR1bits.TMR1IF=0;                                                       //
clear IF
              PIE1bits.TMR1IE=1;                                                       //
enable interrupt on timer 1 which will happen in .1 sec
              timo++;
              if(timo>4)
                     {
                     timo=0;
```

```
                        PIE1bits.TMR1IE=0;                                              //
disable interrupt on timer 1
                        INTCON3bits.INT1IF=0;                                   // clear the
IF of sensor out
                        INTCON3bits.INT1IE=1;                                   // enable the
interrupt on the sensor out
                        }
                }


        else
                if (INTCON3bits.INT1IF==1)                                      // RB1
detector out
                        {
                        INTCON3bits.INT1IE=0;                                   // disable
the interrupt on the sensor out
                        INTCON3bits.INT1IF=0;                                   // clear
interrupt flag on RB1
                        TMR1L=0x00;
                        TMR1H=0x00;
                        T1CON=T1CON | 0x01;                                             //
Timer1 on
                        PIR1bits.TMR1IF=0;                                              //
clear IF
                        PIE1bits.TMR1IE=1;                                              //
enable interrupt on timer 1 which will happen in .1 sec


                        direction=0;
                        last_interupt=secondes;
                        k++;
                        if(k>2)
                                k=0;
                        storeout();
                        if(tablein[k]->seconde!=0xFF)                   //check that the
corresponding step on the detector in is filled
                                convert(k);
                        }


        if(PIR2bits.TMR3IF==1)
                {


                /* 0.1 sec since detector in has been disabled, reenable this sensor */


                T3CON=T3CON & 0xFE;                                                     //
clear bit 0 =>timer 3 off
                PIR2bits.TMR3IF=0;                                                      //
clear IF
                TMR3L=0x00;
                TMR3H=0x00;
                T3CON=T3CON | 0x01;                                                     //
Timer 3 on
                PIR2bits.TMR3IF=0;                                                      //
clear IF
```

```
		PIE2bits.TMR3IE=1;                                                    //
enable interrupt on timer 3 which will happen in .1 sec
		timi++;
		if(timi>4)
			{
			timi=0;
			PIE2bits.TMR3IE=0;                                            //
disable interrupt on timer 3
			INTCON3bits.INT2IF=0;                                 // clear the
IF of sensor in
			INTCON3bits.INT2IE=1;                                 // enable the
interrupt on the sensor in
			}
		}


	else
		if (INTCON3bits.INT2IF==1)                                    // RB2
detector in
		{
			INTCON3bits.INT2IE=0;                                 // disable
the interrupt on the sensor in
			INTCON3bits.INT2IF=0;                                 // clear
interrupt flag on RB2
			TMR3L=0x00;
			TMR3H=0x00;
			T3CON=T3CON | 0x01;                                           //
Timer 3 on
			PIR2bits.TMR3IF=0;                                            //
clear IF
			PIE2bits.TMR3IE=1;                                            //
enable interrupt on timer 3 which will happen in .1 sec


			direction=1;
			last_interupt=secondes;
			j++;
			if(j>2)
				j=0;
			storein();
			if(tableout[j]->seconde!=0xFF)                        // check that
the corresponding step on the detector in is filled
				convert(j);
		}


}


/*************************************************************************/
/*                          - High Interrupt -
        */
/*                                                 occurs every second
                            */
/*************************************************************************/
```

```
#pragma code
#pragma interrupt hisr
void hisr (void)
{

/***********************************************/
/* INT0 is always a high priority interrupt       */
/* INT0 occured => start a low priority interrupt */
/***********************************************/

if(INTCONbits.INT0IF==1)
	{
	INTCON3bits.INT1IF=1;                                                    // enable a
low priority interrupt on sensor out which wont be taken in consideration
	INTCONbits.INT0IE=0;                                                     // INT0
disabled but INT0IF not cleared
	}
else
	{
	T0CONbits.TMR0ON=0;                                                            //
stop timer0
	INTCONbits.TMR0IF=0;                                                     // stop
interruption flag for timer0
	INTCONbits.TMR0IE=0;                                                     // disable
interrupt on timer 0
	secondes++;
	if (secondes==60)
		{
		// clear the warnings at the end of the minute
		PORTB=PORTB & 0b11000111;                                        // clear
warning LEDs on RB<5:3>
		PORTAbits.RA2=1;                                                         //
Disable Buzzer on RA2(unactive at 1)
		PORTD=0xF9 & PORTD;                                                      //
clear display of detection on sensors 1 & 2 at the end of the minute
		change=1;
	// update of the LCD display
		// write Flash memory each minute if there is a change of the acquaintance
		if (counter!=variation || (PORTBbits.RB5 | PORTBbits.RB4 |PORTBbits.RB3) )
			storedata();
		variation=counter;
		minutes++;
		secondes=0;
		if (minutes==60)
	//update clock
			{
				hours++;
				minutes=0;
				if(hours==24)
					{
					hours=0;
```

```
                                        days++;
                                        if(months==1 ||months==3 ||months==5 || months==7
||months==8 ||months==10 || months==12)
                                                {
                                                if(days==32)
                                                        days=1;
                                                        months++;
                                                }
                                        if(months==4 ||months==6 ||months==9 || months==11)
                                                {
                                                if(days==31)
                                                        days=1;
                                                        months++;
                                                }
                                        if(months==2)
                                                {
                                                if(days==29)
                                                        days=1;
                                                        months++;
                                                }
                                        }
                                }
                        }
        reset();
        // test if there were detections recently, else reset the data of sensors


                        /* start timer 0 at 0x6769 */


        TMR0H=0x67;
        TMR0L=0x69;


        T0CONbits.TMR0ON=1;                                                             //
enables timer0
        INTCONbits.TMR0IE=1;                                                    // enable
interrupt on timer 0
        }
}


/*****************************************************************************************/
/* The function CONVERT returns the speed of the customer between the two sensors in m/s */
/* It also increments or decrements the current customer counter and increment the total */
/* visitor counter if the customer is going out.
                                                                                        */
/*
                                                                                */
/*          - 1 incrementation of timer
                                        */
/*          - 0.0000256=1/fosc x 128 x 4 x 1
                        */
```

```
/*                    - two detectors at "length" of each other
                              */
/*                    - the event2 is assumed to occur after event1
                              */
/*****************************************************************************************/


void convert(int step)
{
        unsigned int seci=tablein[step]->seconde;
        unsigned int seco=tableout[step]->seconde;
        unsigned int timeri=tablein[step]->timer;
        unsigned int timero=tableout[step]->timer;
        double tempo1;
        double tempo2;
        double delta;


        tempo1=timeri*0.0000256;                                                    // convert
the value of timeri into seconds
        tempo1=tempo1+seci;                                                                 //
store the time in seconds when the detection of the sensor in occurred
        tempo2=timero*0.0000256;                                                    // convert
the value of timero into seconds
        tempo2=tempo2+seco;                                                                 //
store the time in seconds when the detection of the sensor out occurred


        if (direction==0)                                                                   //
customer is going out
        {
                if (seco<seci)
                {
                        tempo2=tempo2+60;

                        delta=tempo2-tempo1;
                }
                else
                {
                        delta=tempo2-tempo1;
                }

                current.way=0;
                counter--;
        // number of current customers in the store decremented
                if(counter<0)
                        counter=0;
        }



        if (direction==1)                                                                   //
customer is going inside
        {
```

```
		if (seci<seco)
		{
			tempo1=tempo1+60;
			delta=tempo1-tempo2;
		}
		else
		{
			delta=tempo1-tempo2;
		}
		current.way=1;
		counter++;
	// number of current customers in the store incremented
		in++;
	// total number of visitors of the day incremented
	}


	current.speed=length/delta;                                          // speed of
the customer stored


	if(current.speed>9.9)                                                // 9.9 m/s is
the maximum possible by running
		current.speed=9.9;
	warning();
	tablein[step]->seconde=0xFF;                                    // clear values used
	tableout[step]->seconde=0xFF;
	if(current.speed>best_speed)
		best_speed=current.speed;
	change=1;
	// the display must be updated
}


/*****************************************************************************/
/*         The function STOREIN stores the data of the sensor in:                  */
/*         seconds and value of Timer 0 corresponding to detection                         */
/*****************************************************************************/


void storein(void)
{
	tablein[j]->seconde=secondes;


	/* TMR0H is not directly readable, the value is updated after a read of TMR0L */


	test=TMR0L;
	tablein[j]->timer=TMR0H;
	tablein[j]->timer=tablein[j]->timer<<8;
	tablein[j]->timer=TMR0L+tablein[j]->timer;
}
```

```
/*************************************************************************/
/*         The function STOREOUT stores the data of the sensor out:                        */
/*         seconds and value of Timer 0 corresponding to detection                         */
/*************************************************************************/


void storeout(void)
{
        tableout[k]->seconde=secondes;

        /* TMR0H is not directly readable, the value is updated after a read of TMR0L */

        test=TMR0L;
        tableout[k]->timer=TMR0H;
        tableout[k]->timer=tableout[k]->timer<<8;
        tableout[k]->timer=TMR0L+tableout[k]->timer;
}

/*************************************************************************/
/*      The function STOREDATA is called during the High interrupt
         */
/*      It stores in the Flash Program Memory the data for future statistics        */

/*************************************************************************/


void storedata(void)
{

        char counterH_c=0, counterL_c=0;
        int counter_tp;
        int warning_c=0;
        counter_tp= counter;

        /* convert data of counter from 16 bits to 8 */

        counterL_c=counter_tp;
        counter_tp=0xFF00 & counter_tp;
        counter_tp=counter_tp>>8;
        counterH_c=counter_tp;
        warning_c=PORTB && 0b00111000;       // store the warnings (too many people,too fast to
enter and too fast to exit) on RB<5:3>

        data[1]=months;                                      // stored in program memory in
words
        data[0]=days;                               // first Lower bits stored on the right
and then Upper bits
        data[3]=hours;
        data[2]=minutes;
        data[4]=warning_c;
```

```
	data[5]=(char)speed_limit;
	data[7]=counterH_c;
	data[6]=counterL_c;

	write_pm(adress_pm,data);
	adress_pm+=16;                              // Normally should increment of 8, but
easier to read columns in Pgm memory
	if(adress_pm>0x7FF0)                // Pgm Memory Displayed by range from 0 to F for
the bytes
		adress_pm=adress_pm_end+8;    // If arriving at the end of the memory, do the
other column of the Pgm memory

}



/*****************************************************************************/
/*                           - The function WRITE_PM -                       */
/*     writes 8 bytes in after the end of the pgm in the flash program memory      */
/*****************************************************************************/

void write_pm(unsigned short long ad, char * data)
{

	/*
		When a TBLWT is executed, the three LSbs of the Table
		Pointer (TBLPTR<2:0>) determine which of the eight
		program memory holding registers is written to.
		When the timed write to program memory (long write) begins,
		the 19 MSbs of the Table Pointer, TBLPTR
		(TBLPTR<21:3>), will determine which program memory
		block of 8 bytes is written to.
	*/

	int i;

	INTCONbits.GIE=0;                                  //Disable all interrupts
	TBLPTR = ad;
	EECON1bits.EEPGD = 1;         //Access Flash Program memory instead EEPROM

	for(i = 0;i < 8;i++)
		{
		TABLAT = data[i];     // put a char into the table latch register
		_asm
		TBLWTPOSTINC                // write to holding register and increment TBLPTR
(corresponding to TBLWT*+)
		_endasm
```

```
		}

		// after writing 8 bytes in the holding registers
		//write the registers in the Flash Memory

			TBLPTR = ad;
			EECON1bits.EEPGD = 1;		//Access Flash Program memory instead
EEPROM
			EECON1bits.CFGS = 0;		//Select Flash/EEPROM access instead of
configuration registers
			EECON1bits.WREN = 1;		//Allows write cycle
			EECON2 = 0x55;			//values to enter to write (EECON2
non physical register)
			EECON2 = 0xAA;
			EECON1bits.WR = 1;			//start write cycle(automaticly
cleared)
			while(EECON1bits.WR==1){};	//wait end of writing
			EECON1bits.WREN = 0;		//Disable writing

			INTCONbits.GIE=1;			//Reenable all interrupts
}

/*************************************************************************/
/*                          - The function ERASE_PM -
                             */
/*		erases the pgm memory by block of 64 bytes
                */
/*		until the end of the flash program memory
                */
/*************************************************************************/



void erase_pm(void)
{
	//Bits TBLPTR<21:6>
	unsigned short long ad_pm_tp;
	ad_pm_tp=adress_pm;

	INTCONbits.GIE=0;				//Disable all interrupts
	while((ad_pm_tp%64)!=0)			//check that it is a multiple of 64
		ad_pm_tp++;
	while(ad_pm_tp<=0x7FBF)			//Pgm Memory goes until 0x7FFF, -
64=0x7BFF
		{
		TBLPTR=ad_pm_tp;
		EECON1bits.EEPGD=1;			//Access Flash Program memory
instead EEPROM
		EECON1bits.CFGS=0;			//Select Flash/EEPROM access
instead of configuration registers
		EECON1bits.WREN=1;			//Allows write cycle
```

```
                EECON1bits.FREE=1;                          //Erase the memory rows adressed by
TBLPTR<21:6>
                EECON2=0x55;
                EECON2=0xAA;
                EECON1bits.WR=1;                            //start erasing
                while(EECON1bits.WR==1){};          //wait end of erase
                ad_pm_tp+=64;
                }
        INTCONbits.GIE=1;                                   //Reenable all interrupts
}

/*************************************************************************/
/*                            - The function READ_PM -
                                 */
/*              is not used but is useful for debugging
                 */
/*              it is also possible to view the content of the Pgm Mem
          */
/*              in selecting in the View Display the Opcode Hex
                 */
/*************************************************************************/

char read_pm(unsigned short long ad)
{
        TBLPTR=ad;
        _asm
        TBLRDPOSTINC
        _endasm
        return TABLAT;
}







/*************************************************************************/
/* The function WARNING scans all the variables that can generate a warning. */
/*   => 3 warnings:
                                    */
/*   - too fast to go out: Buzzer RA2 + red LED on RB3
       */
/*   - too fast to enter: green LED on RB4
                */
/*   - too many people: yellow LED on RB5
                */
/*************************************************************************/

void warning(void)
{
```

```
	if(mode==0)
	// day mode
	{
		if(current.speed>speed_limit)
		{
			if (direction==0)                                                    //
customer going out
				{
				PORTBbits.RB3=1;                                             //
red LED,enable RB3
				PORTAbits.RA2=0;                                             //
start buzzer, disable RA2 (on at 0)
				copy(message13,warning_message);
				warning_hours=hours;
				warning_minutes=minutes;                                // store the
time when the warning occurred
				PORTAbits.RA5=0;                                             //
Disable Keypad=>number=0xF
				PORTAbits.RA5=1;                                             //
Enable Keypad
				change=1;
	// the display must be updated
				}
			if (direction==1)                                                    //
custommer going in
				{
				PORTBbits.RB4=1;                                             //
green LED,enable RB4
				copy(message12,warning_message);
				warning_hours=hours;
				warning_minutes=minutes;                                // store the
time when the warning occurred
				PORTAbits.RA5=0;                                             //
Disable Keypad=>number=0xF
				PORTAbits.RA5=1;                                             //
Enable Keypad
				change=1;
	// the display must be updated
				}
		}
		if(counter>counter_max)
			{
			PORTBbits.RB5=1;                                                     //
enable yellow LED on RB5
			if(warn==0)
				{
				copy(message11,warning_message);
				warning_hours=hours;
				warning_minutes=minutes;                                // store the
time when the warning occurred
				PORTAbits.RA5=0;                                             //
Disable Keypad=>number=0xF
				PORTAbits.RA5=1;                                             //
Enable Keypad
```

```
                        change=1;
        // the display must be updated
                        warn=1;
        // display once the warning message
                        }
                }
                if(counter<=counter_max)
                {
                PORTBbits.RB5=0;                                                //
disable yellow LED on RB5
                if(warn==1)
                        warn=0;
        // reset warn. Consequently, if a new warning occurs, it will be displayed
                }
                }
        if(mode==1)
                {
                if(current.speed>0)                                             //
night mode
                {
                PORTBbits.RB3=1;                                                //
switch on red LED(RB3)
                PORTAbits.RA2=0;                                                //
start buzzer(RA2)
                copy(message13,warning_message);
                warning_hours=hours;
                warning_minutes=minutes;                                // store the
time when the warning occurred
                PORTAbits.RA5=0;                                                //
Disable Keypad=>number=0xF
                PORTAbits.RA5=1;                                                //
Enable Keypad
                change=1;
        // the display must be updated
                }
                }


}


/*****************************************************************************/
/* The function DATA_FPGA copies into the global variable number the value of  */
/* the data sent by the FPGA after having pressed a key on the keypad          */
/*****************************************************************************/


void data_fpga(void)
{
        number=0x00;
        if(PORTCbits.RC0==1)                                                    //
LSB
                number=0x01;
        if(PORTCbits.RC1==1)
```

```
                number=0x02 | number;
        if(PORTCbits.RC6==1)
                number=0x04 | number;
        if(PORTCbits.RC7==1)                                                    //
MSB
                number=0x08 | number;
}



/*****************************************************************************/
/* The function CHECK_FPGA first checks that the value sent by the FPGA is 'F'   */
/* to make sure that the FPGA is well set to 'F'. Then, the function enables     */
/* the FPGA and consequently makes it possible to wait for a new value to be     */
/* typed. When a new value is detected, it is copied into number and the keypad  */
/* is disabled (this makes the FPGA send 'F').
*/
/*****************************************************************************/


void check_fpga(void)
{
        int number_temp;
        data_fpga();                                                            //
update the value from FPGA
        while(number!=0x0F)                                                 // wait until
the value sent is 0b1111
                data_fpga();
        PORTAbits.RA5=1;                                                        //
enable RA5=>enable FPGA
        while (number==0x0F)                                                // wait for a
value to be typed on the keypad
                data_fpga();
        number_temp=number;                                                     //
prevent the case when the number was set during its update in data_fpga
        data_fpga();
        if (number!=number_temp)
                data_fpga();
        PORTAbits.RA5=0;                                                        //
disable RA5 (disable keypad)
}


/*****************************************************************************/
/* The function INIT makes it possible to get a decimal value from the two digits */
/* typed on the keypad
                                    */
/*****************************************************************************/


int init(void)
{
        number=10;
        // makes sure to enter the while condition
```

```
	while(number>9)
	// prevents the user from using the key A,B,C,D,E or F key
		check_fpga();                                                    //
wait for a new key to be pressed and copy its value into number
	set_digit();                                                     //
display the digit typed on the LCD
	init_d=(int)number;                                              //
first number for the tens
	init_d=init_d*10;                                                //
first number times 10
	number=10;
	// makes sure to enter the while condition
	while(number>9)
// prevents the user from using the key A,B,C,D,E or F key

		check_fpga();                                                    //
wait for a new key to be pressed and copy its value into number
	set_digit();                                                     //
display the digit typed on the LCD
	init_d=(int)number + init_d;                        // add the second number to the
first number
	return init_d;                                                   //
return the decimal number made with the two digits typed
}



/*****************************************************************************/
/* The function CREATE_TABLEOUT
/*****************************************************************************/


void create_tableout(void)
{
	//no malloc in mcc18 (not present in the library stdlib of the compiler)
	//static value, a maximum of ten customers between the 2 sensors is assumed
	//table of pointers on the data of detector out
	tableout[0]=out0;
	tableout[1]=out1;
	tableout[2]=out2;
/*	tableout[3]=out3;
	tableout[4]=out4;
	tableout[5]=out5;
	tableout[6]=out6;
	tableout[7]=out7;
	tableout[8]=out8;
	tableout[9]=out9;
*/
	for(k=0; k<3; k++)
	{
	tableout[k]->seconde=0xFF;
	}
	k=0;
```

```
}

/*****************************************************************************/
/* The function CREATE_TABLEIN
/*****************************************************************************/

void create_tablein(void)
{
	//no malloc in mcc18 (not present in the library stdlib of the compiler)
	//static value, a maximum of ten customers between the 2 sensors is assumed
	//table of pointors on the data of detector in
	tablein[0]=in0;
	tablein[1]=in1;
	tablein[2]=in2;
/*	tablein[3]=in3;
	tablein[4]=in4;
	tablein[5]=in5;
	tablein[6]=in6;
	tablein[7]=in7;
	tablein[8]=in8;
	tablein[9]=in9;
*/
	for(j=0; j<3; j++)
	{
	tablein[j]->seconde=0xFF;
	}
	j=0;
}

/*****************************************************************************/
/* The function RESET
/*****************************************************************************/

void reset(void)
{
//test if enter and exit without passing the other sensor
	if((last_interupt+delay)<60)
		{
		if((last_interupt+delay)<=secondes)		// bizarre d'avoir un if
suivi d'un if. Tu peux regrouper les deux conditions dans le 1er if avec un &&
			{
			create_tableout();
			create_tablein();
			j=0;
			k=0;
			}
		}
```

```
        else if((last_interupt+delay)<=secondes+60)        // 2 'r' a interrupt
                        {
                        create_tableout();
                        create_tablein();
                        j=0;
                        k=0;
                        }
}

/*****************************************************************************/
/* The function INIT_PARAMETERS makes it possible to set all the variables used
/* while the program runs.
/*****************************************************************************/

void init_parameters(void)
{
        Init_lcd();                                                                     //
initialize the LCD
        clear_display();                                                        // clear the
display
        copy(message2,message);

        Display_message();                                                      // display
the greeting message

/*****************************************************************************/
/* These variables are initialized to make sure to enter all the while conditions */
/*****************************************************************************/

months=13,minutes=60,hours=24,days=32,delay=-1,mode=2,counter_max=-1,speed_limit_i=110;

                                                        /* Month */
        check_fpga();                                                           // wait for
any key to be pressed
        clear_display();                                                        // clear the
display
        copy(message3,message);
        Display_message();                                                      // display
the message to set month and day
        blinking();                                                                     //
start blinking to know which value the user is going to set
        while(months>12 | months <1)                                    // wait for the value to
correspond to a possible month
        {
                address=0x0006;
                go_address();                                                   // go to the
address where the first digit of month must be set
                months=init();                                                  // store the
decimal typed
        }
```

```
                                /* Day */


       while(days>31 | days <1)                                          // wait for the
value to correspond to a possible day
       {
              address=0x0009;
              go_address();                                                    // go to the
address where the first digit of day must be set
              days=init();                                                     // store the
decimal typed
       }
       Display_on();                                                           // stop
blinking
                                                        /* Hour */
       check_fpga();                                                    // wait for any
key to be pressed
       clear_display();                                                        // clear the
display
       copy(message4,message);
       Display_message();                                                      // display
the message to set hour and minute
       blinking();                                                                    //
start blinking to know which value the user is going to set
       while(hours>23 | hours <0)                                    // wait for the value to
correspond to a possible hour
       {
              address=0x0006;

              go_address();                                                    // go to the
address where the first digit of hour must be set
              hours=init();                                                    // store the
decimal typed
       }


                                                        /* Minute */


       while(minutes>59 | minutes <0)                                    // wait for the
value to correspond to a possible minute
       {
              address=0x0009;
              go_address();                                                    // go to the
address where the first digit of minute must be set
              minutes=init();                                                         //
store the decimal typed
       }
       Display_on();                                                           // stop
blinking
                                                        /* Mode */
       check_fpga();                                                           // wait for
any key to be pressed
       clear_display();                                                        // clear the
display
       copy(message5,message);
```

```
       Display_message();                                                    // display
the message to set the mode


       while(mode<0 | mode>1)                                          // wait for the value to
correspond to a possible mode
       {
              address=0x0048;
              go_address();                                                 // go to the
address where the first digit of mode must be set
              mode=init();                                                  // store the
decimal typed
       }
       Display_on();                                                        // stop
blinking


       if(mode==0)                                                                 //
day mode
       {
                           /* Speed limit */


              check_fpga();                                                 // wait for
any key to be pressed
              clear_display();                                          // clear the
display
              copy(message6,message);
              Display_message();                                            // display
the message to set the speed limit
              blinking();                                                         //
start blinking to know which value the user is going to set
              address=0x000D;


              go_address();                                                 // go to the
address where the first digit of speed limit must be set
              speed_limit_i=init();                                   // store the decimal
typed
              speed_limit=speed_limit_i;                                 // value int
speed_limit_i typed by user as N.N
              speed_limit=speed_limit/10;                                // and stored as
NN=> /10 in double speed_limit
              Display_on();                                                 // stop
blinking


                                                  /* Customer limit */


              check_fpga();                                                 // wait for
any key to be pressed
              clear_display();                                              // clear the
display
              copy(message7,message);
              Display_message();                                            // display
the message to set the customer limit
              blinking();                                                         //
start blinking to know which value the user is going to set
              address=0x004B;
```

```
                go_address();                                                   // go to the
address where the first digit of customer limit must be set
                counter_max=init();                                             // store the
decimal typed
                Display_on();                                                   // stop
blinking



                                                        /* Delay */


                check_fpga();                                                   // wait for
any key to be pressed
                clear_display();                                                // clear the
display
                copy(message8,message);
                Display_message();                                              // display
message to set the delay
        blinking();                                                             // start
blinking to know which value the user is going to set
                while(delay<0 |       delay>59)                                 // wait for
the value to correspond to a possible delay
                {
                        address=0x0007;
                        go_address();                                           // go to the
address where the first digit of delay must be set
                        delay=init();                                           // store the
decimal typed
                }
                Display_on();                                                   // stop
blinking
                check_fpga();                                                   // wait for
any key to be pressed
        }
        else                                                                            //
night mode
        {
                speed_limit=3;
                counter_max=0;
                delay=30;                                                               //
settings of the night mode
        }


        PORTAbits.RA5=1;                                                        // enable
keypad
        copy(message10,warning_message);                        // initialize the next
message by using the 'F' sent by the FPGA
}



        /*************************************************************************/
        /*                                                 #         ###  ###
                                    */
```

```
        /*                                                  #    #     #  #
*/
        /*                                                  #    #     #  #
*/
        /*                           ####  ###  ###                              */
        /*****************************************************************************/



/*******************************************************************************************/
/* The function TIME makes it possible to the LCD to have the necessary time for each  */
/* instruction to be executed                                                          */
/*******************************************************************************************/


void time(int wait)
{
        int b=0;

        T2CONbits.TMR2ON=1;

        for(b=0;b<wait;b++)
                {
                // each incrementation lasts 3.2 us
                while(TMR2<=0x11);                                                  //
17*3.2=54 us => 17=0x11
                TMR2=0x00;
                }
        T2CONbits.TMR2ON=0;
        TMR2=0x00;
}


/*******************************************************************************************/
/* The function ENPULSE generates a pulse on the Enable pin of the LCD so that the  */
/* LCD can know that a new instruction is available                                  */
/*******************************************************************************************/


void Enpulse(void)
{
        _asm
                NOP
                NOP
        _endasm
        // make sure that the Enable signal remains low for at least 230 ns


        PORTEbits.RE2 = 1;                                                          //
set enable high
        _asm
                NOP
                NOP
        _endasm
        // make sure that the Enable signal remains high for at least 230 ns
```

```
	PORTEbits.RE2 = 0;                                                              //
set enable low
}

/*************************************************************************/
/* This function displays the message stored into the message[i] string. */
/* To display variables, this function calls the function store_variable */
/*************************************************************************/

void Display_message(void)
{
	i=0;
	// 'i' is initialized to 0 to display the first character first
	while(message[i]!=0x00)                                                         //
run until the last character is displayed
	{

		/*********************************
	*************************************************/
		/* - if choice in 'store_variable(int number, int choice)' is 1, a number like
'09'  */
		/* will be displayed as '9' without the '0'.
*/
		/* if choice is 0, the numbers will be displayed as they are, for example '05'
*/
		/* will remain '05'
*/
		/* - store_variable function copies the value of the variable into message[i]
and    */
		/* makes i point on the first character that has to be displayed
*/

	/*****************************************************************************/

		if(message[i]=='$' && message[i+1]=='c')           // if message[i] contains $c
		{
			store_variable(counter,1);                                  // store the
current number of customer into message [i]
			if(counter<10) message[i+1]=' ';                  // erase the 'c' of
'$c' if only one character (0-9) is displayed
		}
		else if(message[i]=='$' && message[i+1]=='s')   // if message[i] contains $s
		{
			int speed;
			speed=Rounding(current.speed);


	/*************************************************************************/
			/* round the variable current.speed (double) and convert it into a int.
*/
			/* for example: 3.26 gives 33 without comma. The comma is displayed
after. */
```

```
	/*************************************************************************/


                        store_variable(speed,0);                                        // store the
value of the speed into messsage [i]
                        message[i+2]=message[i+1];                                      // shift the
second digit on the right to let a space for the comma
                        message[i+1]='.';                                                       //
display the comma
                }
                else if(message[i]=='$' && message[i+1]=='r')   // if message[i] contains $r
                {
                        address=Row2;
                        go_address();                                                           // go
to the beginning of the second row (0x40).
                        i=i+2;
        // shift 'i' twice on the right because of the space taken by '$r'
                }
                else if(message[i]=='$' && message[i+1]=='D')   // if message[i] contains $D
                {


                        store_variable(months,0);                               // store the
value of the month into message[i]
                        i=i+2;
        // shift 'i' twice on the right to display '/'
                        message[i]='/';
        // display '/'
                        i++;
        // shift 'i' on the right to store the value of the day into message[i]
                        store_variable(days,0);                                                 //
store the value of the days into message[i]
                        i=i-3;
        // make 'i' point on the first digit of the month
                }


                else if(message[i]=='$' && message[i+1]=='T')   // if message[i] contains $T
                {
                        store_variable(hours,0);                                        // store the
value of the hours into message[i]
                        i=i+2;
        // shift 'i' twice on the right to display ':'
                        message[i]=':';
        // display ':'
                        i++;
        // shift 'i' on the right to store the value of the minute into message[i]
                        store_variable(minutes,0);                                      // store the
value of the minutes into message[i]
                        i=i-3;
        // make 'i' point on the first digit of the hours
                }


                else if(message[i]=='$' && message[i+1]=='W')   // if message[i] contains $W
                {
```

```
	/*************************************************************************/
			/* when a warning occured, the time it occured is stored into message[i]
*/
	/*************************************************************************/

			store_variable(warning_hours,0);			// store the value
of the hours into message[i]
			i=i+2;
	// shift 'i' twice ont he right to display ':'
			message[i]=':';
	// display ':'
			i++;
	// shift 'i' on the right to store the value of the minute into message [i]
			store_variable(warning_minutes,0);			// store the value
of the minutes [i]
			i=i-3;
	// make 'i' point on the first digit of the hours
		}
		else if(message[i]=='$' && message[i+1]=='C')   // if message[i] contains $C
		{
			store_variable(in,1);					// store the
number of visitors into message[i]
			if(in<10) message[i+1]=' ';				// erase the
'C' of '$C' if only one character (0-9) is displayed
		}
		else if(message[i]=='$' && message[i+1]=='B')   // if message[i] contains $B
		{
			int Best_speed;
			Best_speed=Rounding(best_speed);


	/*************************************************************************/
			/* round the variable Best_speed (double) and convert it into a int.
*/
			/* for example: 3.26 gives 33 without comma. The comma is displayed
after. */
	/*************************************************************************/

			store_variable(Best_speed,0);			// store the value
of the speed into messsage [i]
			message[i+2]=message[i+1];				// shift the
second digit on the right to let a space for the comma
			message[i+1]='.';						//
display the comma
		}



		Display_char();
	// display the character on which 'i' points and then increment 'i'
	}
```

```
}


/*********************************************************************************************/
/* The function COPY makes it possible to copy a variable message (var_mess[]} into another */
/* message (dest_mess[]) such as message[i] or others.                                       */
/*********************************************************************************************/


void copy(char var_mess[],char dest_mess[])
{
       int g=0;                                                          // 'g' is
initialized to 0 to copy the first character first
       while(var_mess[g]!=0x00)                                             // run until
the last character is copied
       {
              dest_mess[g]=var_mess[g];                                     // the
character in var_mess[g] is copied into dest_mess[g]
              g++;
       // g is incremented to scan the string
       }
}



/*********************************************************************************************/
/* The function STORE_VARIABLE makes it possible to store the variables into message[i] */
/* so that they can be displayed after                                                       */
/*********************************************************************************************/


void store_variable(int variable,int choice)
{
       int temp;
       int calc;
    int count=0;                                                      // count is used to
know how many unsignificant '0' there are in the number
       int n=0;                                                             // 'n' is
used to point on the right character depending on the value of choice


       /*********************************************************************************
**************/
       /* Each digit of the variable is processed to know if the variable has 1 or 2 or 3 or 4
digits.      */
       /* First, the variable is divided by 1000 to store the most significant digit and to
know if it      */
       /* is a 4 digits variable. Then, the number found times 1000 is subtracted to the
variable so that   */
       /* one can keep only the 3 others digits.
                                                                  */
       /* The same process is made to store the 3 others digits.
*/
       /* Depending on 'choice' the unsignificant '0' will be replaced or not.
*/
       /* '48' is added to get the ASCII code.
                                                                */
```

```
	/* An example is used to understand the process.
*/
	/* Let us imagine that our variable=108 and i=5 and choice=1.
                                                */
	/* 'Variable' is an int in this function so the divisions give also int!
*/
	/*****************************************************************************
***************/


	if(choice==1)                                                                  // if
choice=1, the unsignificant '0' will be replaced by the most significant digits
	{
		message[i]=variable/1000 + 48;                                      //
message[5]=(108/1000)+48=48 and 48 is character '0' in ASCII code
		calc=message[i]-48;                                                        //
calc=48-48=0
		calc=calc*1000;
	// calc=0*1000=0
		temp=variable-calc;                                                        //
temp=108-0=108
		if(message[i]=='0')                                                        //
TRUE because message[5]='0'
		{
			i--;
	// i=4
			count++;
	// count=1
		}
		message[i+1]=temp/100 + 48;                                 //
message[5]=(108/100)+48=49 and 49 is character '1' in ASCII code
		calc=message[i+1]-48;                                              //
calc=message[5]-48=1
		calc=calc*100;                                                             //
calc=1*100=100
		temp=temp-calc;
	// temp=108-100=8
		if(message[i+1]=='0' && count==1)                          // FALSE.
message[5]='1'
		{


	/*****************************************************************************/
		/* Count==1 makes it possible to prevent this function from replacing the '0'
*/
		/* in 1023 for example. This condition will only replace the '0' of the third
digit  */
		/* if the fourth digit is also a '0' such as 0012.
                                */

	/*****************************************************************************/


			i--;
			count++;
		}
		n=2;
```

```
	}
	else temp=variable;
	message[i+n]=temp/10 + 48;                                          //
message[4+2]=message[6]=(8/10)+48=48  => '0'
	calc=message[i+n]-48;                                               // calc=48-
48=0
	calc=calc*10;                                                               //
calc=0*10=0
	temp=temp-calc;
	// temp=8-0=8
	if(message[i+n]=='0' && count==2 && choice==1)             // FALSE because count=1
	{
		i--;
		count++;
	}
	message[i+n+1]=temp + 48;                                           //
message[7]=8+48=56  => '8'
	i=i+count;
	// i=4+1=5

	/*****************************************************************/
	/* Finally, i=5, message[5]='1', message[6]='0', message[7]='8' */
	/*****************************************************************/
}

/*************************************************************************/
/* The function ROUNDING takes a double such as 5.6783 and convert it into  */
/* a two digits int rounded.                                             */
/* Let us try with 5.6783                                                */
/*************************************************************************/

int Rounding(double var_double)
{
	int temp1=var_double*100;                                           // get the
three most significant digits.   temp1=5.6783*100=567
	int temp2=var_double*10;                                            // get the
two most significant digits.     temp2=56
	temp1=temp1-temp2*10;                                               // get the
value of the third digit.        temp1=567-56*10=7
	if(temp1>=5) temp2=temp2 + 1;                       // if the third digit>=5, add one.
temp2=56+1=57
	return temp2;                                                               //
return the rounded value.                       return 57
}


/*****************************************************************/
/* The function INIT_LCD makes it possible to initialize the LCD */
/* by following the necessary steps indicated in the datasheet   */
/*****************************************************************/
```

```
void Init_lcd(void)

{

       PORTD = Init;                                                                  //
function set

       PORTE = 0x00;                                                                  //
set RS low, indicating an instruction

       Enpulse();
       // tell the LCD that a new instruction is there

       time(76);
       // 4100 us/54 us = 75.9 => wait for a few more 4.1 ms


       PORTD = Init;                                                                  //
function set

       PORTE = 0x00;                                                                  //
set RS low, indicating an instruction

       Enpulse();
       // tell the LCD that a new instruction is there

       time(2);
       // 100 us/54 us = 1.85  => wait for more than 100 us


       PORTD = Init;                                                                  //
function set

       PORTE = 0x00;                                                                  //
set RS low, indicating an instruction

       Enpulse();
       // tell the LCD that a new instruction is there


       time(1);
       // 39 us/54 us = 0.72 => wait for a few more than 39 us


       PORTD = Interface;                                                             // 2
lines and 5x8 dots

       PORTE = 0x00;                                                                  //
set RS low, indicating an instruction

       Enpulse();
       // tell the LCD that a new instruction is there


       time(1);
       // 39 us/54 us = 0.72 => wait for a few more than 39 us


       PORTD = D_off;
       // switch off the display

       PORTE = 0x00;                                                                  //
set RS low, indicating an instruction

       Enpulse();
       // tell the LCD that a new instruction is there


       time(1);
       // 39 us/54 us = 0.72  => wait for a few more than 39 us


       clear_display();                                                               //
clear the display


       PORTD = D_mode;
       // cursor shifts on the right

       PORTE = 0x00;                                                                  //
set RS low, indicating an instruction
```

```
        Enpulse();
        // tell the LCD that a new instruction is there

        time(1);
        // 39 us/54 us = 0.72 => wait for a few more than 39 us


        Display_on();                                                                   //
switch on the display

}


/*********************************************************/

/* The function CLEAR_DISPLAY clears the display of the LCD */

/* generally to display another message                    */

/*********************************************************/


void clear_display(void)

{

        PORTD = D_clear;                                                                //
clear the display

        PORTE = 0x00;                                                                   //
set RS low, indicating an instruction

        Enpulse();
        // tell the LCD that a new instruction is there
                                    //

        time(29);
        // 1530 us/54 us = 28.3 => wait for a few more than 1.53 ms

}


/*****************************************************************************/

/* The function SET_DIGIT makes it possible to enter the value of the variables     */

/* with the keypad during the initialization of the program.                         */

/*****************************************************************************/


void set_digit(void)

{

        go_address();                                                                   //
set the address counter to the address stored in the global variable 'address'

        if(address>=0x0040)

                i=address-0x002A;                                                       //
2nd row begins at 0x0040 for the address counter and 0x0016 for the pointer 'i'

        else i=address;
        // 'i' must point where address points so that Display_char() will display the digit at
the right address

        message[i]=number+0x0030;                                                   // 'number'
corresponds to the digit typed on the keypad. 0x0030 is added to get the ASCII code of
'number'

        Display_char();
        // display the digit pointed by 'i' and increment 'i' and increment the address counter

        if(message[i]=='/' || message[i]==':' || message[i]=='.')

        {


        /*********************************************************************/

        /* if the character pointed by 'i' is '/' or ':' or '.', 'address' is  */
```

```
        /* incremented by 2 to point on the address just after the separator.  */
        /* Thus, the address counter is incremented by 1.                      */
        /* For example, in 12:35 the ':' will not be replaced but after having */
        /* replace '2', the next digit to be replaced will be '3'              */
        /*********************************************************************/


                address=address+0x0002;
                go_address();
                i++;
        // Like the address counter, 'i' is incremented by 1 to point on the digit after the
separator
        }
        if(address>=0x0040)                                                          // if
the variable is on the second row
                address=i+0x002A;                                                    //
set 'address' where 'i' is pointing on.
        else address=i;
}



/*********************************************/
/* The fonction BLINKING makes the cursor blink */
/*********************************************/
void blinking(void)
{
        PORTD = C_blink;
        PORTE = 0x00;                                                                //
set RS low, indicating an instruction
        Enpulse();
        // tell the LCD that a new instruction is there
        time(1);
        // 39 us/54 us = 0.72 => wait for a few more than 39 us
                                                    // wait for
}


/*************************************************/
/* The fonction DISPLAY_ON switches on the display */
/* Cursor off, blinking off                      */
/*************************************************/
void Display_on(void)
{
        PORTD = D_on;                                                                //
switch on the display
        PORTE = 0x00;                                                                //
set RS low, indicating an instruction
        Enpulse();
        // tell the LCD that a new instruction is there
        time(1);
        // 39 us/54 us = 0.72 => wait for a few more than 39 us
}
```

```
/*****************************************************************/
/* The fonction GO_ADDRESS makes it possible for the address counter */
/* to point on the address stored in the global variable 'address'   */
/*****************************************************************/
void go_address(void)
{
       address=address+0x0080;                                                    //
Set the address counter to 'address' (D7 must be 1)
       PORTD = address;
       PORTE = 0x00;                                                              //
set RS low, indicating an instruction
       Enpulse();
       // tell the LCD that a new instruction is there
       time(1);
       // 39 us/54 us = 0.72 => wait for a few more than 39 us
       address=address-0x0080;                                                    //
reset 'address' to its original value
}


/*************************************************************/
/* The fonction DISPLAY_CHAR makes it possible to display the */
/* character pointed by 'i' in message[i]                     */
/*************************************************************/
void Display_char(void)
{
       PORTD = message[i];                                                        //
copy data onto the DDRAM
       PORTE = 0x02;                                                              //
set RS high, indicating a data
       Enpulse();
       // tell the LCD that a new data is there
       time(1);
       // 43 us/54 us = 0.80 => wait for a few more than 43 us
       i++;
       // increment 'i' so that it can point on the next character to display
}
```

4. VHDL code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity top_keyboard is
    Port (    clock : in std_logic;
              Reset : in std_logic;
              a,b,c,d: out std_logic;
              e,f,g,h: in std_logic;
               enable : in std_logic;
               Interrupt : out std_logic;
               Laser1 : in std_logic;
               Laser2 : in std_logic;
              Detection1 : out std_logic;
              Detection2 : out std_logic;
               Key : out std_logic_vector(3 downto 0));

end top_keyboard;

architecture top_keyboard_arch of top_keyboard is
signal d0, d1, d2, d3,inhibit_int, key_detect_int, data_available_int: std_logic;

begin
Scanner: entity work.scanner(scanner_arch)
        port map(clock=>clock, Reset=>Reset, D0=>d0, D1=>d1, en_scan=>inhibit_int);
Decoder_2to4: entity work.decoder_2_to_4(decoder_2_to_4_arch)
        port map(D0=>d0, D1=>d1, a=>a, b=>b, c=>c, d=>d);
Coder_4to2: entity work.coder_4to2(coder_4to2_arch)
        port map(D2=>d2, D3=>d3, e=>e, f=>f, g=>g, h=>h, key_detect=>key_detect_int);
Register_storage: entity work.Register_storage(Register_storage_arch)
        port map(clock=>clock, Interrupt=>Interrupt, Reset=>Reset, D0=>d0, D1=>d1,D2=>d2,
D3=>d3, Y0=>Key, en_out=>enable, en_in=>data_available_int);
Debouncing: entity work.bounce_detect(debouncing_arch)
        generic map(n=>17,p=>100000)
        port map(clock=>clock,Reset=>Reset, key_detect=>key_detect_int, inhibit=>inhibit_int,
data_available=>data_available_int);
Laser1: entity work.laser(laser_arch)
       generic map(n=>21,p=>1400000)
       port map(clock=>clock,Reset=>Reset,edge_detect=>laser1,detection=>detection1);
Laser2: entity work.laser(laser_arch)
       generic map(n=>21,p=>1400000)
       port map(clock=>clock,Reset=>Reset,edge_detect=>laser2,detection=>detection2);
end top_keyboard_arch;
```

▪▪

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.numeric_std.ALL;

entity scanner is
    Port (    clock : in std_logic;
              Reset : in std_logic;
              en_scan : in std_logic;
              D0 : out std_logic;
              D1 : out std_logic);
end scanner;

architecture scanner_arch of scanner is

signal d, q: std_logic_vector(1 downto 0);
begin
D1<=q(1);
D0<=q(0);
q<="00" when Reset='1' else d when rising_edge(clock);
d<=std_logic_vector(unsigned(q)+1) when en_scan='1' else q;

end scanner_arch;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity decoder_2_to_4 is
    Port (       D1 : in std_logic;
              D0 : in std_logic;
              a : out std_logic;
              b : out std_logic;
              c : out std_logic;
              d : out std_logic);
end decoder_2_to_4;

architecture decoder_2_to_4_arch of decoder_2_to_4 is
signal Din: std_logic_vector(1 downto 0);
signal code: std_logic_vector(3 downto 0);

begin
Din<=D1&D0;
with Din select
code <=              "0111" when "00",
                      "1011" when "01",
                      "1101" when "10",
                      "1110" when "11",
```

```
                      "1111" when others;

a<=code(3);
b<=code(2);
c<=code(1);
d<=code(0);

end decoder_2_to_4_arch;
```

■■■

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity coder_4to2 is
    Port (    e : in std_logic;
              f : in std_logic;
              g : in std_logic;
              h : in std_logic;
              D2 : out std_logic;
              D3 : out std_logic;
              key_detect : out std_logic);
end coder_4to2;

architecture coder_4to2_arch of coder_4to2 is
signal code: std_logic_vector(1 downto 0);

begin
D3<=code(1);
D2<=code(0);
code<=    "00" when e='0' else
         "01" when f='0' else
         "10" when g='0' else
         "11";

key_detect<='1' when (e and f and g and h)='0' else '0';

end coder_4to2_arch;
```

■■■

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.numeric_std.ALL;

entity register_storage is
    Port (    D3 : in std_logic;
              D2 : in std_logic;
              D1 : in std_logic;
              D0 : in std_logic;
              en_out : in std_logic;
              en_in : in std_logic;
                              clock : in std_logic;
              Reset : in std_logic;
                              Interrupt : out std_logic;
                              Y0 : out std_logic_vector(3 downto 0));

end register_storage;

architecture register_storage_arch of register_storage is

signal code,Din,Yint,Yout: std_logic_vector(3 downto 0);
signal en: std_logic_vector(1 downto 0);
signal Interrupt_int,pulse: std_logic;

begin
       Din<=D3&D2&D1&D0;
       en<=en_in&en_out;
       Yout<= "1111" when Reset='1'
              else Yint when rising_edge(clock);
       Y0<=Yout;
       Interrupt_int<='1' when Yout=x"e" else '0';
       Interrupt<= Interrupt_int and (not pulse);
       pulse<= '0' when Reset='1' else Interrupt_int when rising_edge(clock);

with en select
Yint<=         code when "11",
                              x"f" when "00",
                              x"f" when "10",
                              Yout when others;

with Din select
code <=        x"0" when "1101",
                              x"1" when "0000",
                              x"2" when "0001",
                              x"3" when "0010",
                              x"4" when "0100",
                              x"5" when "0101",
                              x"6" when "0110",
```

```
                              x"7" when "1000",
                              x"8" when "1001",
                              x"9" when "1010",
                              x"a" when "0011",
                              x"b" when "0111",
                              x"c" when "1011",
                              x"d" when "1111",
                              x"e" when "1100",
                              x"f" when "1110",
                              x"0" when others;

end register_storage_arch;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity bounce_detect is generic(n:natural:=17;p:natural:=100000);
    Port (    clock : in std_logic;
              Reset : in std_logic;
              key_detect : in std_logic;
              inhibit : out std_logic;
              data_available : out std_logic);
end bounce_detect;

architecture debouncing_arch of bounce_detect is
signal dint,sint,rint: std_logic_vector((n-1) downto 0);
signal fint, gint: std_logic;

begin
inhibit<= not key_detect;
sint<= dint when key_detect='1' else (others=>'0');
rint<= (others=>'0') when Reset='1' else sint when rising_edge(clock);
dint<= rint when fint='1' else std_logic_vector(unsigned(rint)+1);
fint<= '1' when unsigned(rint)>=p else '0';
data_available<= fint and (not gint);
gint<= '0' when Reset='1' else fint when rising_edge(clock);

end debouncing_arch;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.numeric_std.ALL;

entity laser is generic(n:natural:=21;p:natural:=1400000);
       Port (           clock : in std_logic;
              Reset : in std_logic;
              edge_detect : in std_logic;
              detection : out std_logic);
end laser;

architecture laser_arch of laser is

signal dint,sint,rint: std_logic_vector((n-1) downto 0);
signal fint, gint: std_logic;

begin

sint<= dint when edge_detect='1' else (others=>'0');
rint<= (others=>'0') when Reset='1' else sint when rising_edge(clock);
dint<= rint when fint='1' else std_logic_vector(unsigned(rint)+1);
fint<= '1' when unsigned(rint)>=p else '0';
detection<= fint and (not gint);
gint<= '0' when Reset='1' else fint when rising_edge(clock);

end laser_arch;
```