

“UNIVERSIDAD AUTÓNOMA DE QUERÉTARO”

ENGINEERING FACULTY

MICROSYSTEMS LAB

PRACTICE # 1 COUNTER 0-99 WITH ONE STEP PUSH BUTTON

TEACHER:

MC. MOISES MARTÍNEZ HERNÁNDEZ

STUDENTS:

MORONES COLLADO HOMERO

MOJICA MORALES ISRAEL

LONA IVÁN

DATE:29/JAN/2019

**OBJECTIVE**

The purpose of the practice is to distinguish the use of registers, in what moments and circumstances make use of them with the right implementation depending on the needs. Since it is intended to be a practical process and expand our knowledge of the management of the microcontroller PIC 18F4550and the register tools we have at our disposal.

**DESCRIPTION OF THE PRACTICE**

The 18f4550 microcontroller will be used, the software used to program the microcontroller will be the PIC C Compiler, for the practice it is defined to perform a zero to ninety-nine counter, using two 7-segment displays, with the freedom to choose a 7-digit display. cathode or anode segments, the increase will be caused by a push botton, therefore the increase of the aforementioned display will be of a unit value, within the body of the program it is required to make only use of "for" cycles so the use of "Delays" is discarded in the body of the program, in the same program should be made use of the records seen in the previous class of Microsystems laboratory, which are the records "standard\_io, fixed\_io and fast\_io"

**THEORETICAL FRAMEWORK**

-Pic 18F4550: Microcontroller ideal for low power (nanoWatt) and connectivity applications that benefit from the availability of three serial ports: FS-USB(12Mbit/s), I²C™ and SPI™ (up to 10 Mbit/s) and an asynchronous (LIN capable) serial port (EUSART). Large amounts of RAM memory for buffering and Enhanced Flash program memory make it ideal for embedded control and monitoring applications that require periodic connection with a (legacy free) personal computer via USB for data upload/download and/or firmware updates.

-#USE STANDARTD\_io (Port)

This directive affects the code that the compiler will generate for the input and output instructions. This quick method of doing I / O causes the compiler to perform I / O without programming the address register.

Example: #USE STANDARTD\_io (B)

-#USE FIXED\_IO (port\_OUTPUTS = pin\_x #, pin\_x # ...)

This directive affects the code that the compiler will generate for the input and output instructions. The fixed method of doing I / O will cause the compiler to generate code to make an I / O pin input or output each time it is used. This saves the RAM byte used in normal I / O.

Example: #use fixed\_io (a\_outputs = PIN\_A2, PIN\_A3)

-#USE STANDARD\_IO (port)

This directive affects the code that the compiler generates for the input and output instructions. The standard method of doing I / O will cause the compiler to generate code to make an I / O pin input or output each time it is used. In the 5X series processors this requires a byte of RAM for each port established as standard I / O.

Example: #use standard\_io (A)

**DEVELOPING**

For the developement of the practice first we made a simulation in the software Proteus so before doing any physical conection with the simulation we can see if the programmed code will work as pleased or theres a major mistake so the system does not work, the circuit shown in the next figure 1.0 is the representation o the desired final result.

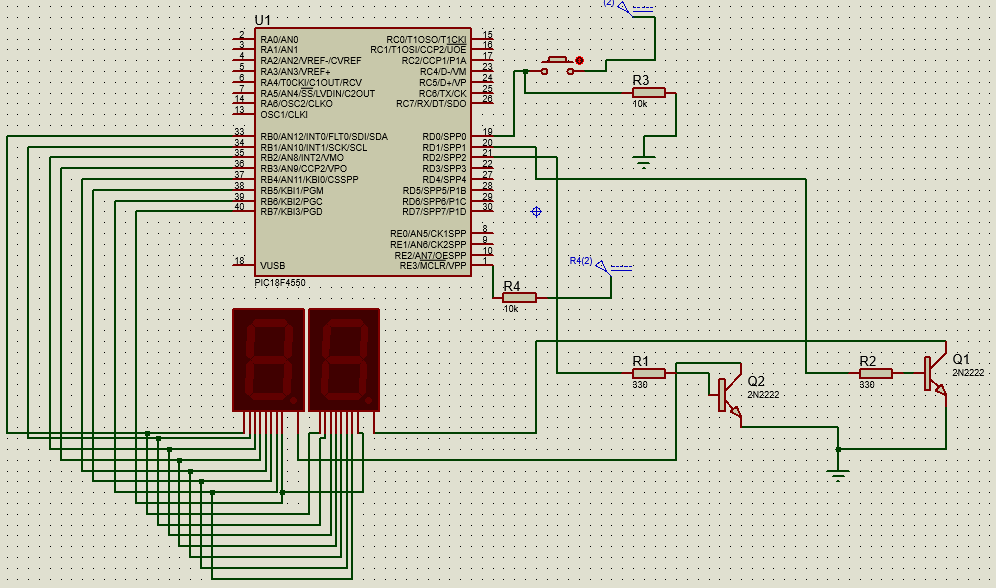


Figure 1.0 Circuit scheme in software Proteus.

Once we make all the conections in Proteus we start programming our code in CCS Compiler so we can start simulating and testing if the code is in the right path or if there must be a change. The final code which performed as desired is shown below.

Code:

#include <18F4550.h>

#fuses HS,NOPROTECT,NOWDT

#use delay(clock=2M, internal)

#use fast\_io(D)

//#use standard\_io(B)

//#use fixed\_io(D\_OUTPUTS=PIN\_D0,PIN\_D1)

Int num[10]={0b00111111,0b00000110,0b01011011,0b01001111,0b01100110,0b01101101,0b01111101,0b00000111,0b01111111,0b01101111},unidad=0,decena=0,contador=0;

void multiplex();

void incremento();

void main()

{

set\_tris\_D(0b00000000);

while(true)

{

if(input(pin\_b0))

{

incremento();

while(input(pin\_b0))

multiplex();

}

multiplex();

}//fin del ciclo infinito

}//fin del main

void incremento()

{

contador++;

if(contador>99)

contador=0;

decena=contador/10;

unidad=contador-decena\*10;

}//fin del incremento

void multiplex()

{

output\_high(pin\_c1); //display de la derecha

output\_d(num[decena]);

delay\_ms(10);

output\_low(pin\_c1);

output\_high(pin\_c0); //display de la izquierda

output\_d(num[unidad]);

delay\_ms(10);

output\_low(pin\_c0);

}//fin del multiplex

The code works as desired and as it seems that theres no problem so we proceed building the circuit physically.

**RESULTS**

For the easy and fast montage of the circuit we use a protobard, the next figure 2.0 shows the circuit on the protoboard with all the other components such as the push button, both 7-segment displays, transistors for rapid transition and the counter starting at zero. Theres also the Master-Prog connected as programmer and power supply.

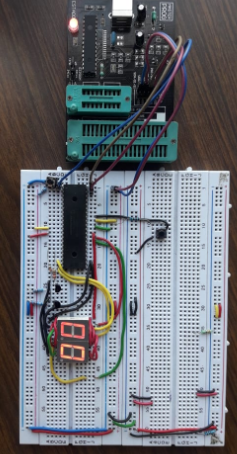


Figure 2.0 Final circuit

The final result was a complete success, we achieve the goal using the fast\_io register, also the code perfomed very well with no breaking points.

**CONCLUSIONS**

-Morones Collado Homero: The practice was completed with satisfactory results, the usage of fast registers showed me that is important to understand the needs of the proyect you want to make many times you try to fix a problem by reprogramming your code when the real problema is to dont have a previos knowledge of the microcontroller and his registers. By nowing the registers and how they work will result in many times the solution of a problem of the desired project.

-Mojica Morales Israel: For the realization of the practice we used a command or instruction called fast\_io, which allow us the ejecution stablished of the program with determinated inputs and outputs for its formal ejecution. For the rest it kept being a summary of things seen for the incoming concepts of the course.

-Lona Iván: To finalize the practice already makes use of certain resources that in a certain way help to fine-tune details in the programming of the memory of our microcontroller, we learned the different functions of the policy records that were seen in the class as the #use faster\_io , fixed\_io and standard\_io, that for example, briefly the faster \_io uses inputs or outputs without configuring, it is necessary to indicate the port, fixed\_io is to be forced exclusively to be as input or outputs so this register ignores the set\_tris instruction and The standard\_io is for simple operations and for issues that are not provided.