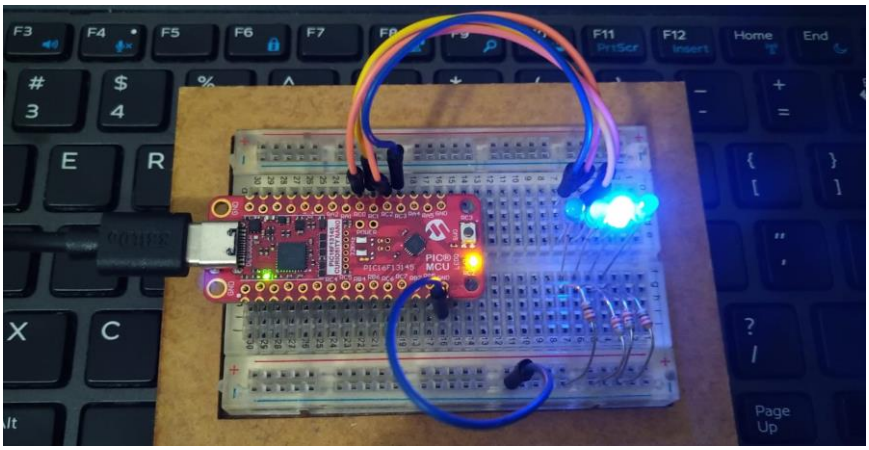
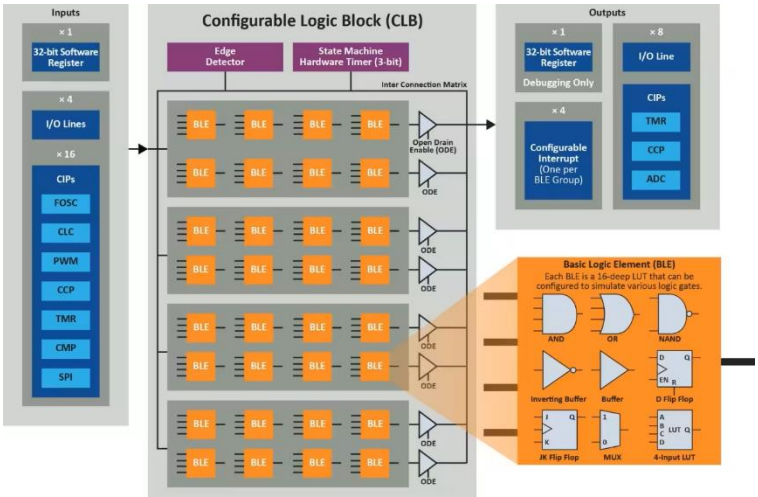


PIC16F13145 and the Gray Code

https://www.linkedin.com/pulse/pic16f13145-gray-code-francesco-sacco-3jmpe?utm_source=rss&utm_campaign=articles_sitemaps

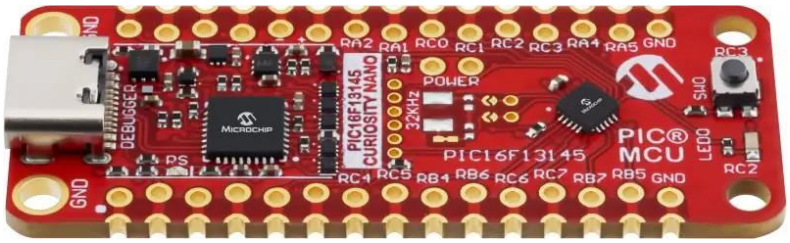
Francesco Sacco
Opublikowano 10 kwi 2024

Recently Microchip launched a new microcontroller named PIC16F13145. This new component introduces the CLB (Configurable Logic Block) peripheral, which is capable of incorporating customized logic blocks into your project. This new peripheral can reduce or eliminate the need for external logic, reducing project costs, for example.



Configurable Logic Block (CLB) Module

In this article we will present this new component using the PIC16F13145 Curiosity Nano Evaluation Kit development platform. We will also present an example of converting binary to Gray Code in three different ways. You can find the files of this project in GitHub.



PIC16F13145 Curiosity Nano Evaluation Kit

The Basis of the Project

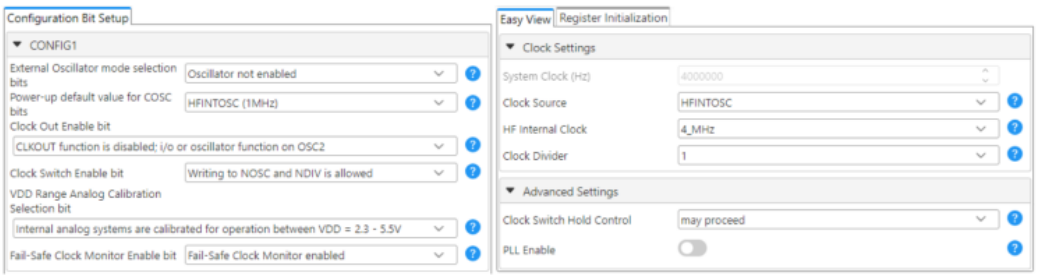
This project was developed using the MPLAB-X IDE platform in version v6.20. Installation of the XC8 compiler is also required. The softwares can be downloaded from the Microchip website at the links below.

Link to MPLAB®-X.

Link to MPLAB®-XC8 Compiler.

We start creating the project in MPLAB-X IDE in "File->New Project". We wish to develop an application project. The device to be selected is the PIC16F13145. When selecting the compiler, we chose XC8 in version v2.46, as it was installed.

Once the project is created, it is necessary to configure the oscillator in the "Configuration Bit" tab. We will use the 4MHz internal oscillator as below.

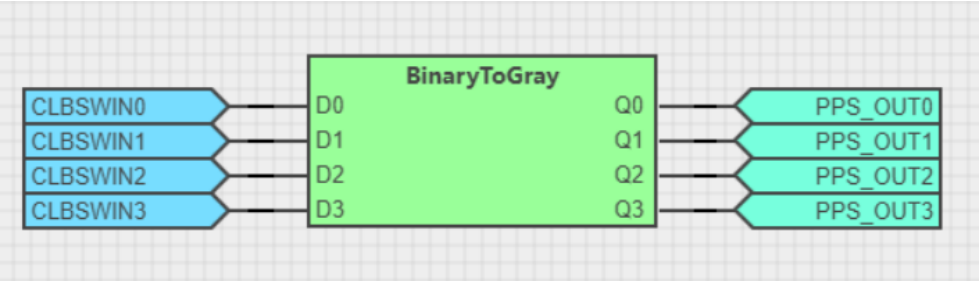


Selecting the oscillator

Adding the CLB peripheral

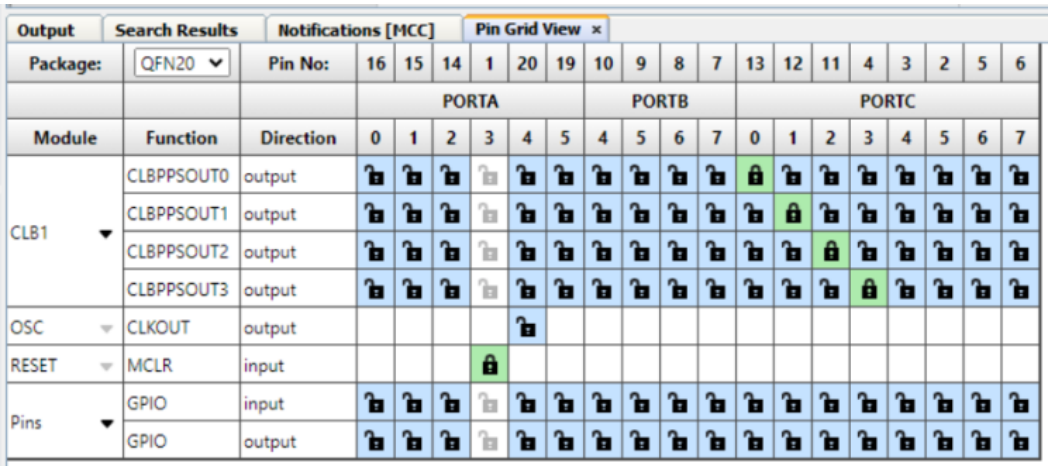
The next step is to add the CLB peripheral to the project. By doing this, MPLAB automatically opens a CLB Synthesizer editing window. This editor allows you to create your logic in three ways, through Lookup Tables (LUT), using logic gates, or writing your module in Verilog. We will explore these three ways of developing the synthesis of logic blocks.

The first step is to define the inputs and outputs. For inputs, we use the CLBSWIN bits. It is a register accessible by the processor and we will write to it through our C program. For the outputs, we select PPS_OUT, which we will then link to a GPIO on the microcontroller.



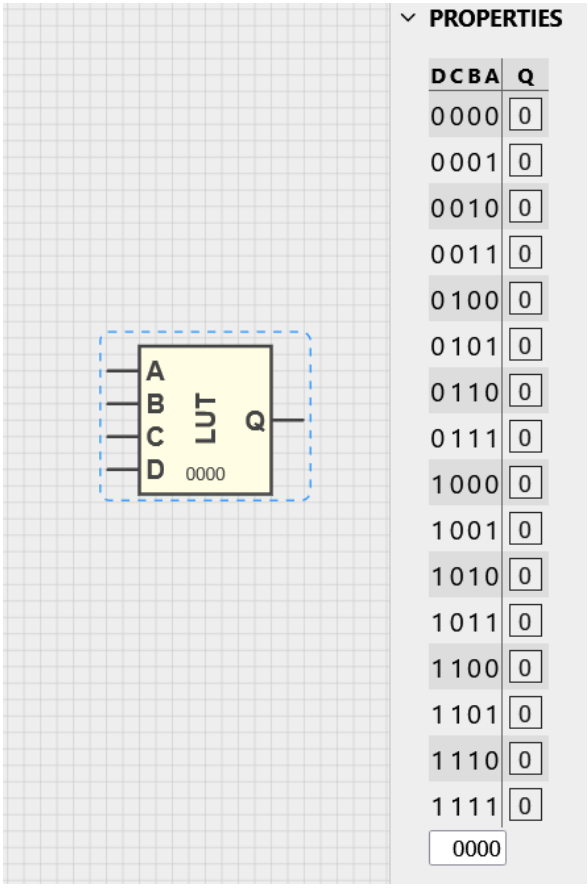
The module to implement the Gray Code converter.

In the "Grid Pin View" we will assign a GPIO for each CLBPPSOUT we use. In this case, it is very convenient to assign Port C0 to Port C3 for CLBPPSOUT0 to CLBPPSOUT3 respectively.



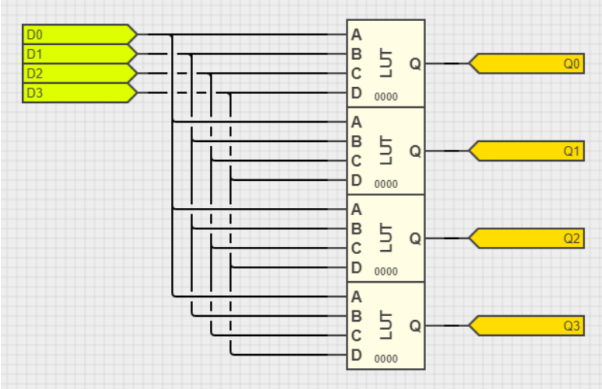
Implementation of Gray Code convert using LUT

The first implementation we will make of the Binary to Gray Code converter is using Lookup Tables (LUT). Each LUT has 4 input bits and 1 output bit, and it is possible to define the desired output for each combination of input data.



LUT Component and it's properties

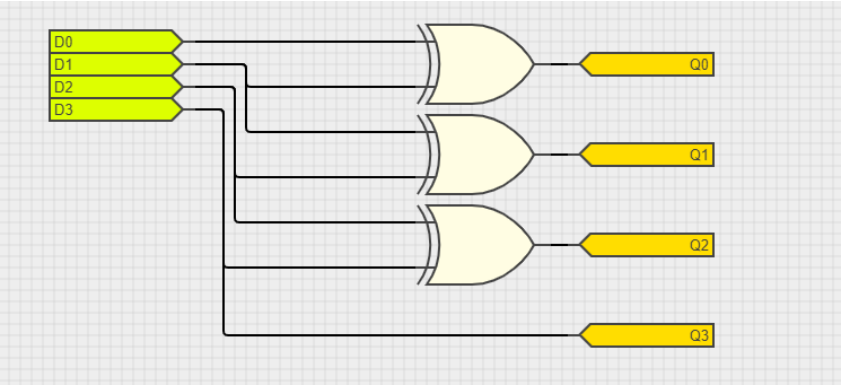
In our final solution, a LUT was placed for each output bit. The table for each bit was implemented representing the expected Gray code for each input binary data.



The content of the module Binary to Gray using LUT.

Implementation of Gray Code convert using Logic Gates

To develop the module using logic gates, you can start from the table used in the previous solution and perform the calculations using Boolean logic and Karnaugh Maps. Doing this exercise you will notice that three XOR gates perform the necessary operations.



The content of the module Binary to Gray using Logic Gates.

Implementation of Gray Code convert using Verilog

The solution using verilog is exactly the same as what we did using logic gates. The difference is that the XOR logical operation is written according to the standard of this hardware description language.

```
module BinaryToGray( d0, d1, d2, d3, q0, q1, q2, q3);
    input d0, d1, d2, d3;
    output q0, q1, q2, q3;

    assign q0 = d0 ^ d1 ;
    assign q1 = d1 ^ d2 ;
    assign q2 = d2 ^ d3 ;
    assign q3 = d3 ;

endmodule
```

The content of the module Binary to Gray using Verilog.

C Program Content

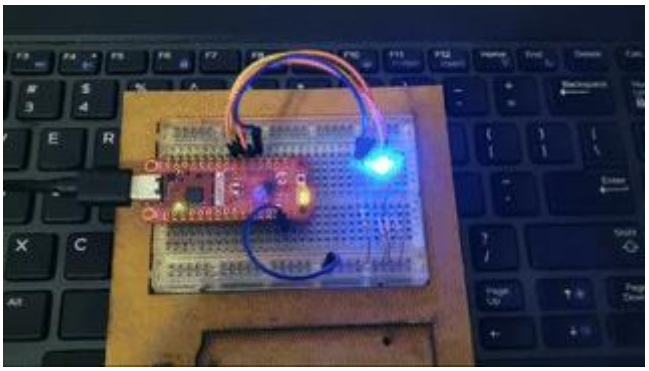
In the main file we will implement two actions, a count and a delay of 1 second. The count is transferred to the CLBSWINL register, which we defined in the logic synthesis. The 4 least significant bits of this register are the input data to our converter.

```
59      CLBSWINL = 0;
60
61      for( unsigned char cnt = 0x00 ; /* EVER */ ; cnt++ )
62      {
63          __delay_ms(1000);
64          CLBSWINL = cnt & 0x0F ;
65      }
```

This program is the same for any of the three logic implementations adopted.

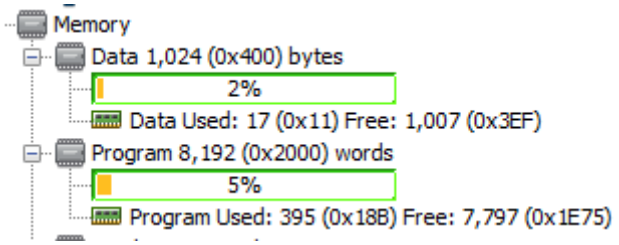
Results and Conclusion

For any solution adopted, the result is the same, the output presented follows the Gray Code counting sequence.



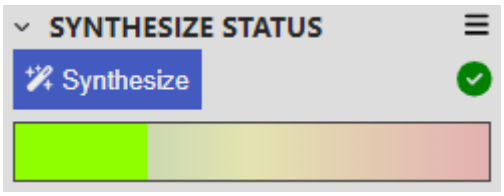
The execution of the example.

Since the C code is the same, all solutions have the same memory usage.



Memory usage.

The use of logical fabric also presented the same use for the three solutions presented.



Use of logic blocks.

```
Number of wires:                 16
Number of wire bits:             16
Number of public wires:           8
Number of public wire bits:       8
Number of memories:               0
Number of memory bits:            0
Number of processes:              0
Number of cells:                 11
  IB                              4
  LUT2                            3
  OB                              4
```

Logical block usage statistics.