HOW TO TURN ON A LED

WITH A STM32F407G BOARD

JULIO EMANUEL GONZALEZ GONZALEZ

In this document will be described how to turn on a LED with the board previously mentioned step by step.

This activity consisted of turning on a LED of the STM32, no matter which model we are working on. In my case I have been working with the STM32F407.

In the following, the steps which were carried out to turn on a LED will be described.

Step 1.

Make research about the documentation of the board we are working with.

In this step we had to search the internet for everything related to the STM32F407 board, such as the datasheet, the reference manual, the user's manual, and the schematic diagram of the board.

Step 2.

As a second step we had to analyze the schematic diagram of the board.

Here what we had to do was to analyze the complete schematic diagram of the board and identify if the board we were working with had LEDs available to turn on. Once we had identified the LEDs, we only had to look at which port they were connected to and which pin of that port to move on to the next step.

Step 3.

In the third step, we had to analyze the block diagram of the board.

Once we had identified the port and pin to which the LED is connected, we went to analyze the block diagram of the board to find the GPIOs and then observe to find which bus they are connected to. In this way, it could be seen that in the case of the STM32F407 board the GPIOs ports are connected to AHB1.

Step 4.

In our fourth step we had to analyze the memory map of the board.

When we had identified both the port of the GPIOs and the Bus they are connected to, we went to the memory map to find which register memories we had to work on.

In the next step, we had to identify the registers for the reset and clock control (RCC).

Step 5.

For this step, the first thing we did in the memory map was to find where the Bus is which the GPIO ports are connected, which was AHB1. After finding it in the memory map, we had to identify the RCC

of that Bus and get the starting address. Once the main address was obtained, we had to find the specific address to activate the enable of port D which was the one we had to use. When we find that address, we add the offset of 0x30 to the initial address of the RCC and with that we would have our address of the register on which we were going to work with respect to the RCC.

Step 6.

As next step we had to identify the GPIO registers.

Once the GPIO registers were identified, we had to see which MODER or PIN was the one we were going to work on, in my case it was 14. Similarly, there we found the bits we had to modify which were 28 and 29 which had to be set to 1 and 0 respectively to configure that pin as an output. This case it was not required an addition of the offset to the base address because of the first register of the GPIOs is to configure the mode of the MODER.

Step 7.

The next step was to go to the GPIO port output data register to configure the one.

for the pin I was going to use, which was 14.

We also had to add the offset to the initial register of the GPIOs to get to the address of the port output data register and modify the registers we needed. This offset was of 0x14 to add to the base address of 0x4002 0C00.

Step 8.

Once we had identified all the registers we needed, we went to the STM32CubeIDE to program the registers.

Once in the software, we had to create 3 pointers, one for each register address we needed. The first one to modify the RCC, another one for the GPIO mode and the last one for the Output data register.

Consequently, we had to make the activation of the bits corresponding to each address that we saved in each of the pointers we created.

For the first pointer that stores the address of the RCC enable, the bit we had to set was bit 3 corresponding to the port D we are working on.

For the next pointer that saved the GPIO mode address we had to set bit 28 corresponding to modern 14 or pin 14 on which we are working, and the corresponding bits for that pin are 28 and

29, but to configure that pin as an output bit 28 is a 1 and bit 29 is a 0.

For the last pointer that stores the address of the output data register of the GPIO, we had to set bit 14 corresponding to pin 14 that we are using.

Step 9.

As a last step, we only had to build the program, configure the Debug, and test it on the board.