G070RB_LowPower_Demo

(Github Repository Link: https://github.com/PriyamMascharak/G070RB_LowPower_Demo.git)
(Shortened Report: https://github.com/PriyamMascharak/G070RB_LowPower_Demo/blob/main/README.md)

This is a mini project to demonstrate the **Low Power Capabilities** of a **STM32G070RB**. This is essentially an interview assignment project I was given. This is the problem statement:

The Problem Statement

Q Switch & LED Blinking with STM32G070RB: There is a tactile switch (only one switch) and LED connected to the microcontroller in a **NUCLEO-G070RB** board. The LED is off initially. Depending on switch press, the LED blinks in the following way.

1st Switch press: LED blinks at frequency of **0.5 Hz**. 2nd Switch press: LED blinks at frequency of **1 Hz**. 3rd Switch press: LED blinks at frequency of **2 Hz**. 4th Switch press: LED turns off. 5th Switch press: considered 1st switch press and the LED blinking cycle repeats.

Create circuit and code for target microcontroller. Assume relevant functions. Also, optimize for power consumption. Use micro-controllers **low power modes** to achieve it.

Solution Approach

The above program is essentially a blinky with a button input which sounds pretty dull in the beginning. The simplest way to achieve it is toggling the LED in a while loop while polling the button i/p. Once a button press is detected we change the delay values in an if/switch statement

However this becomes interesting when we have to achieve the lowest power possible in the **µA range**. To achieve that in the above program above we need several setup choices. The first thing that comes to mind is the fact that we don't need the cpu to blink the LEDs,

Blinking the LED

So how do we blink the LEDs without the CPU? The answer is to make use of the inbuilt **Hardware TIMERS** inside the STM32G070RB. More specifically the **PWM Channel** inside the **General Purpose Timers** provided with the MCU. The reason for using the PWM is the fact that we need different time durations for the **Ton** and the **Toff** when the LED blinks.

I have chosen the **TIM3 Channel 1** for the PWM Channel. The output pin for the channel is the pin **PA6**.

We configure the PWM by setting three values: The **Prescaler**, the **Pulse** and the **Period**. To change the frequency of the blinks we essentially have to change the OFF time(Toff) of the LED.

To achieve this I used a constant pulse and prescaler values and changed the timer period based on the frequency needed.

To set these values we need the **PWM Formula**

Here the Update Event represents the frequency with which the PWM counter Register is reset. So if we want to get a specific frequency of blinks, for a given timer clock we set a fixed prescaler value and adjust the pulse and period values based on the Ton and Toff we need respectively.

Since we are setting low power modes, the Clock provided to the timers will change later so i will give the calculation demo for it later.

Because we will be changing period values a lot we enable the **auto-reload preload register** of the timer

Low Power Modes:

The **G series** STM microcontrollers are kind of middle of the road when it comes to low power optimizations. They provide more options compared to the **F series** microcontrollers however not as much as the **L-series**.

The G-Series provides us with 7 low power modes: Out of them we are interested in the **Low Power Run mode** and the **Low Power sleep mode**

The **Low Power Run Mode** lets us use the **Low Power Regulator** while still running the CPU. However we only activate this mode when we need to change the period values in the timer register after a button press.

The **Low Power Sleep Mode** is the mode we will stay at the majority of the time. This mode lets us shut off the CPU and while still having all the peripherals in the **VCORE domain** powered by the low power regulator. This includes the timers we want to use.

The Low Power mode also caps the max frequency at **2Mhz** to reduce power but this is less relevant to us as we will see later

We also enable the **Sleep-On-Exit** feature by setting the **SleepOnExit** bit in the **SCB->SCR** register. This enables us to automatically exit sleep mode whenever an interrupt is fired and enter sleep mode again when the interrupt is done with its processing task.

The **Stop0 mode** provided by the G0 is even more suitable for a sleep mode however since it turns off the VCORE domain which powers our Timer we can't use it. It could have been used if we had the **LPTIM** which are generally provided in the stm32 L series MCUs.

Clock Configurations

It is a general rule that lower clock speed equates to low performance but even lower power consumption. In our case we just need to change one period register and keep track of the number of times a button has been pressed.

Thus we can afford to put the clock speed as low as possible. The **Low Power Run mode** we are using already caps our clock speed at **2 Mhz** which reduces power consumption significantly, but the fact of the matter is we can go even lower!

The lowest possible clock speed in the g070 is provided by the **Low Speed Internal clock**. While it is not as accurate as the **Low Speed External clock**, it is slightly lower power and since we don't really need the accuracy we can use it to power the sysclock. Thus running the entire system at a very low **32KHz**.

We set the timer values accordingly using the PWM formula with a prescaler of **31** and period values of:

- **499** for 0.5 Hz
- **999** for 1Hz
- **1999** for 2Hz

This enables us to shed off computing power we dont need

Button Press Detection AND Changing Blink Frequencies

Because we need the CPU only when we want to change the blink frequencies we can put all of our minimal code in the interrupt itself. Thus we don't write anything in the main loop enabling us to shut off the CPU when a button is not being pressed.

I will be using the **PC13** pin for the button i/p. We are going to set up the PC13 pin in **EXTI interrupt mode**. Whenever the button is pressed the **NVIC** causes the interrupt to fire, waking the processor up from low power sleep mode to the low_power_run_mode.

To ensure our MCU wakes up from sleep mode using the **WFI()** instruction.

We also have to implement **debouncing** in our interrupt detection to prevent the interrupt from triggering multiple times while the user is still pressing the button.

To implement this we set up another interrupt using the **TIM6** basic timer in the MCU to fire another interrupt when the after the timer period has elapsed. If the button is still high after the period elapsed we register it as a valid button press and move on with our CPU processing step. I have set up the TIM6 such that it fires an interrupt after **0.2 seconds** preventing multiple accidental button presses.

Changing the Timer Period

We change the TIM3 timer period in the **TIM6 interrupt callback function** after a valid button press has been detected.

We keep track of the current frequency by keeping track of a global variable **'mode'**, 'mode' has a value between **0,1,2 and 3** depending on the off, 0.5Hz, 1Hz and 2Hz modes respectively.

We use a **switch case** to check the current mode, change the **AUTO_RELOAD** register to the relevant value and update the mode variable. In off mode we turn the timer off and start it again when the next button is pressed.

Assumptions

- Assumed that we are free to use any pin for the led to maximize low power performance
- Accuracy is not the priority for this blinking led output

Choices

- Mostly used in-built **STHAL macros** and functions instead of direct register manipulation to increase readability and portability.
- More power could have been saved by moving the vector table to the **SRAM** and disabling flash all together but that seemed like a bit overkill considering we are already using the low-power-regulator

Program Flow:

1) Initialization

- Initialize **Flash Interface** and **Systick**.
- Configure System Clock by using the **High Speed Internal(HSI)** oscillator to complete initialization quickly
- We set up the GPIO pin **PC13** in **Interrupt Rising Mode**.
- We set up **TIM3** with a 32kHz clock source in mind an enable PWM in channel 1 with the channel output in **PA6**.
- We setup **TIM6** with a 32Khz clock source in mind with a period of **200ms**
- We enable the power interface clock by using the macro **__HAL_RCC_PWR_CLK_ENABLE()** ;
- We prepare to enter Low Power mode by entering **LOW_POWER_CLK_CONFIG()** where we
    - Initialize Typedef structs to
        - Initialize **LSI**
        - Disable **HSI**
        - Update **SysClk** source
    - Change the **Voltage Scaling** to **Scale 2**

    - Initialize LSI and wait for LSI turn on
    - Setup LSI as **SYSCLK**
    - Disable HSI
    - Disable **PLL**
- We then return to the main function to Suspend the **SYSTICK Interrupt**
- Enable **Sleep on Exit**
- Enter **Low Power Sleep Mode** by using the **WFI** instruction. Using the HAL function also enables the Low Power Run mode for us before going to the Low-Power Sleep Mode

2) Running

- After Entering the Sleep mode the MCU listens for interrupts
- Once an interrupt is triggered by the PC13 button, the **NVIC** calls the **EXTI callback function** and starts the CPU.
- Inside the callback function we start **TIM6**
- TIM6 has a 200ms period after which it fires another interrupt.
- In the **TIM6 Period Elapsed Callback** we first stop the Timer6 and Check if the GPIO PC13 is still set
- If it is set we register it as a button press.
- We go into the switch statement and check the mode. If it is **Mode zero** we Start the timer. The default period is set to 499 with equates to an LED blink of 0.5Hz
- As more button presses are detected we update the period by using the **__HAL_TIM_SET_AUTORELOAD()** macro to update the TIM3 auto-reload register
- When we reach **Mode 3** we update the auto-reload register to 499 and then stop the TIM3 to stop the LED blinking
- After we process the switch statement we exit the Interrupt and the Cortex Core goes back to sleep because we enabled **SleepOnExit()**.