System-on-chip Architecture

Lab 2: STM32F0-Discovery wave generation

Robert Margelli, 224854

Contents

1. Purpose of the lab session

The purpose of this lab sessions was to get familiar with the DMA, ADC and TIM2 to implement a wave generator program. Additionally, the program was integrated with knowledge acquired in the previous lab session, in fact the micro-controller was intermittently forced in a low power state.

2. Program behavior

After initializations the program evolution is triggered by the user's interaction with the board. The program can cycle through the generation of 4 waveforms: sine, escalator, square (pulse) and triangle waves. The on-board LEDs notify the user which wave is being emitted on pin PA4 (of course by connecting it to an oscilloscope it is possible to see the actual wave); by pressing the User Button while an LED is on the wave can be switched.

The wave to LED correspondence is as follows:

1. Green LED On: sinewave;
2. Blue LED On: escalator wave;
3. Green LED On (after the blue one, ie second occurrence of green): pulse wave;
4. Green and Blue LED On: triangle wave.

Figure 1 reports pictures of the 4 waveforms, extracted by using an Arduino UNO board [1] as a very simple oscilloscope. Although highly inaccurate, this setup proved to be enough to demonstrate the effectiveness of the program.

3. Implementation details

Between each wave period the StandbyRTC low-power mode is reached and the CPU is kept in such state for 1 second, after which the automatic wakeup and a system reset occur. To understand which waveform should be emitted after each reset, I resorted to the RTC_BKP_DR0 backup register which in the program is meant to store the value of the SelectedWaveForms variable (see `EXTI0_1_IRQHandler` in `stm32f0xx_it.c`). As the program re-executes this value is then restored in the SelectedWaveForms variable and correct wave generation occurs.

Overall, this mechanism is enabled by properly configuring the 3rd channel of the DMA. An interrupt handler (refer to `DMA1_Channel2_3_IRQHandler` in `stm32f0xx_it.c`) has been added and the NVIC has also been enabled. The interrupt request is triggered each time the DMA finishes the transfer of data corresponding to a single wave and the its handler responds by calling the `StandbyRTCMode_Measure()` function defined in `stm32f0xx_lp_modes.c`.

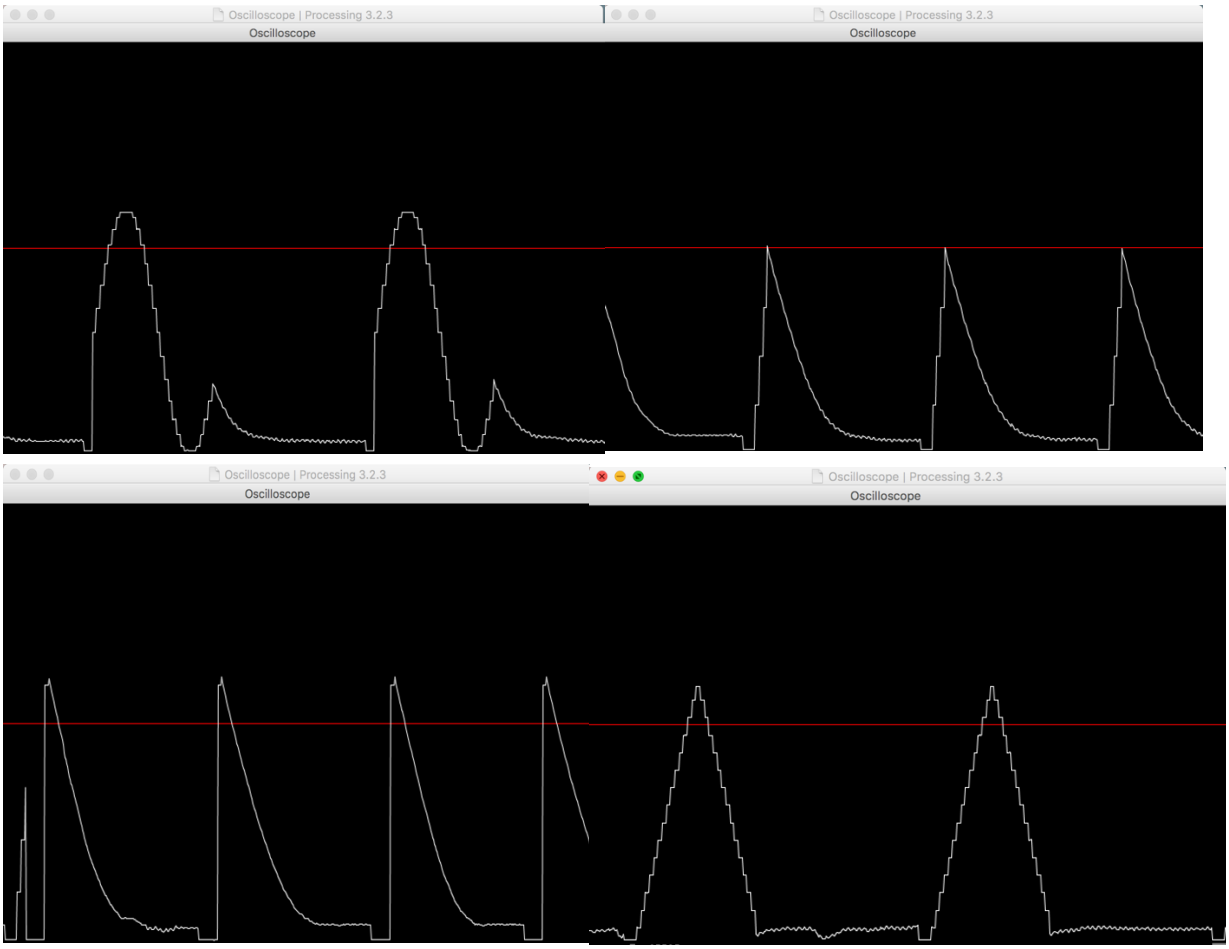

Figure 1 – Waveforms. From top left to bottom right: sinewave, escalator, pulse, triangle.

3. Conclusions

The main difficulty I encountered in this lab session was figuring out how to integrate a low-power state in the provided program. I opted for the StandbyRTC mode because it sends a hard reset at wakeup, Sleep and Stop modes would also have been suitable but the used peripherals would have to be manually turned on again at wakeup. I found my solution to be the easiest to implement and also the Standby state is the least power hungry among the low-power states.
In conclusion, this program provides a basic skeleton to which new functionalities can be easily added (e.g. new waveforms).

4. References

[1] http://www.instructables.com/id/Arduino-Oscilloscope-poor-mans-Oscilloscope/