Politecnico di Torino

System-on-chip Architecture

Laboratory 2

Report

Master degree in Computer Engineering

Referents: Prof. Andrea Acquaviva, Prof. Alberto Macii

Author: Luca Mocerino, 229006

January 31, 2017

Summary

In this laboratory we have managed (Digital to Analog Converter) DAC,(Direct Memory Access) DMA , timer and low-power mode. We can divide the laboratory work in two main phases:

- Using datasheets in order to learn the configuration of all elements;
- Providing a C code for handling and swichting among three signals: square wave, sin and escalator wave. After each period of the signal the system goes to Standby mode. The switch among different waves happens using user button.

Settings

In this phase we describe the different settings and configurations of the board for the mentioned purpose.

Timer

It was used TIM2, a 16-bit timer, one of the different general purpos timer on the board, since it was one possible source of trigger for DAC. I set timer prescaler and clock period in order to make simpler the observation on the oscilloscope. The timer is basically a counter in which you can load the maximum value it can counts. Every time the timer reaches the maximum value the DAC is triggered. The following picture provides the DAC block diagram and TIM2.

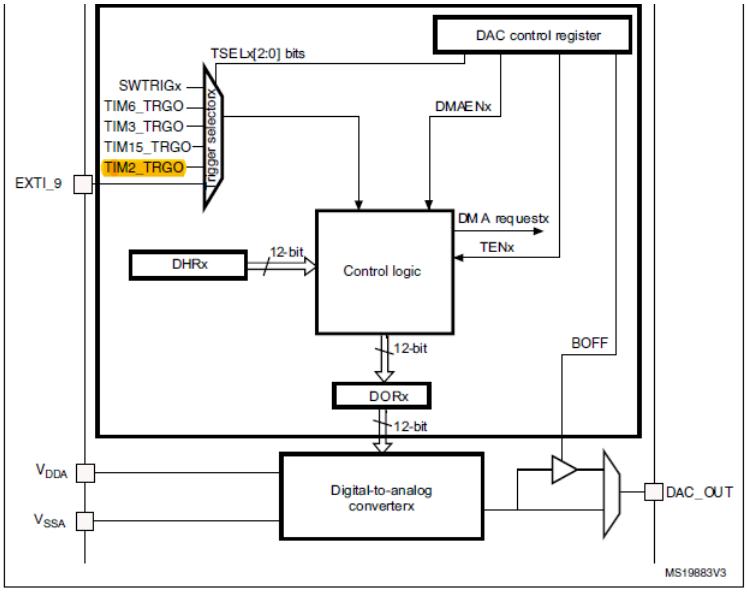

Figure 1: DAC block diagram

DAC

On out board we have a 12-bit DAC and the convertion in this case is generated using TIM2. We use DMA to comunicate between DAC and memory in which digital samples are stored. PA4 is used as analog output to read the waveform generation.

DMA

The DMA trasfers data from memory to peripheral (DAC) using their addresses. In our case we use also a DMA interrupt in order to switch in low power mode, after the end of transfers.

Standby mode

The Standby mode allows to achieve the lowest power consumption. There are two different types of standby mode depending on the mode to resume: using RTC or with external interruption. In the former RCT is clocked by LSI, in the latter RTC is off. In this mode we have the following configuration:

- RTC Clocked by LSI (low-power clock source around 40kHz).The HSI and the HSE oscillators are disabled;
- All peripherals are disabled;
- No DEBUG available since Cortex-M0 is no longer clocked;
- All I/O pins are high impedance except:reset pad, PC13, PC14 and PC15 if configured by RTC or LSE and WKUPx pins;
- After waking up from Standby mode, program execution restarts in the same way as after a Reset (boot pin sampling, option bytes loading, reset vector is fetched, etc.).

One important thing is that we use backup register in order to store the value for selecting, after reset, the previous wave.

Code & Results

main.c

In this section was descripted the main module. We can divide the code in several parts:

- **Initializaion phase**:
  1. Timer configuration with a certain period and prescaler. It was used to trigger the DAC
  2. DAC configuration setting up APB bus clock and GPIO PA.04 (DAC_OUT1) as analog output.
  3. DMA configuration with input output buffer in circular mode. We add a DMA interrupt in order to implement the request.
- **Infinite Loop part**: in that part we have three sections of similar code that configure DMA and DAC in order to take the correct wave.
- **Additional part I**: We add another wave, in this case a square wave . So the program allows to switch among those three waves adding that additional code in the user button (PA0) interrupt handler:

```
void EXTI0_1_IRQHandler(void)
{
  if(EXTI_GetITStatus(USER_BUTTON_EXTI_LINE) != RESET)
  {
    /* Change the wave */
    WaveChange = !WaveChange;

    if(SelectedWavesForm == 2) {
      SelectedWavesForm =0;
    }else SelectedWavesForm++;


    /* Clear the Right Button EXTI line pending bit */
    EXTI_ClearITPendingBit(USER_BUTTON_EXTI_LINE);
  }
}
```

Figure 2: User button interrupt handler

- **Additional part II**: In that part after each transmission of DMA, it generates and interrupt that allows Discovery Board to go in Stanby Mode using the funtion **StandbyRTC-Mode_Measure(void)**. In order to recover the previous state after the Standby mode, that reset the Discovery Board and restart execution from beginning we have used the availables *backup registers*. In that register we save the values of the current wave in the way that after one period of the selected wave the execution resumes in the same state. Moreover in that code

is possible to switch among waves. The main part that allows what I have described before is the DMA interrupt handler:

```
//DMA interrupt routine
void DMA1_Channel2_3_IRQHandler(void)
{

  if(DMA_GetITStatus(DMA1_IT_TC3)!= RESET)
  {
  StandbyRTCMode_Measure(); //When DMA ends its tranfers, go in this mode
  /* Clear the DMA1 channel 3 trasfert complete pending bit */
  DMA_ClearITPendingBit(DMA1_IT_GL3);
  }

}
```

Figure 3: DMA interrupt handler

Moreover we choose that low power mode among the others availabes. Why ?

About **Standby mode** we know that the behaviour after wakeup is the reset of the MCU after a certain time using the RTC. For that reason we store values in Backup register in order to store the desidered values. Reguarding **Stop mode** we need to enable all peripherals after wakeup using RTC. Finally **Sleep mode** we have only the CPU main clock off (48MHz) and , as before we have to enable again all peripherals after wakeup.

Results

In this section are shown the results of our waves :

1) Waves without low-power mode:

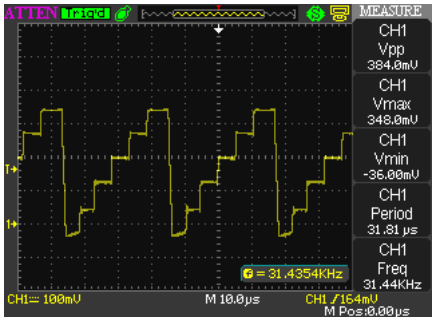

Figure 4: Escalator

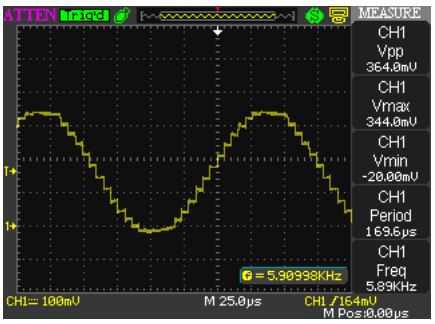

Figure 5: Sine

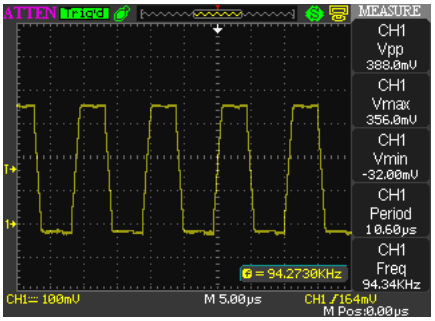

Figure 6: Square wave

2) Waves with low-power mode:

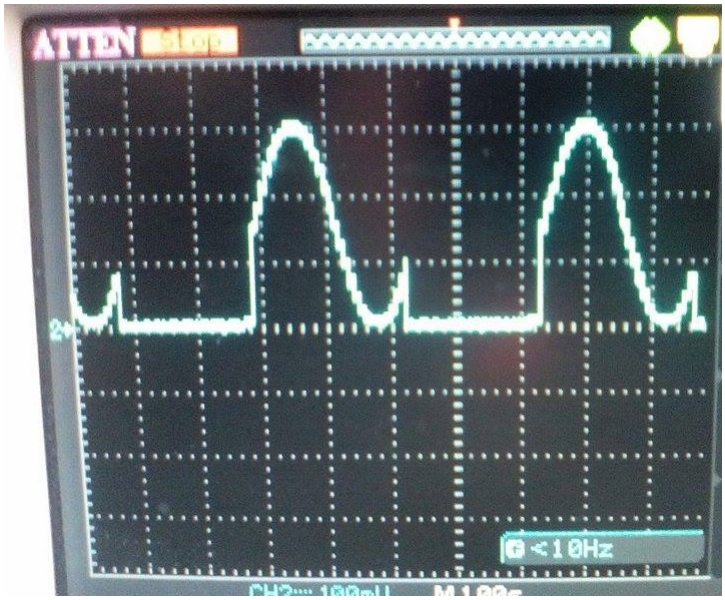

Figure 7: Sine wave

Figure 8: Escalator

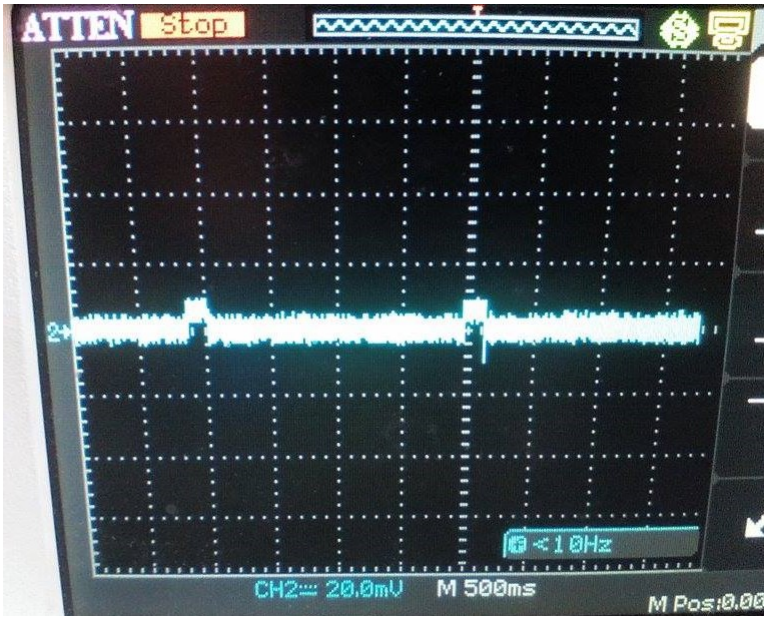

Figure 9: Square wave

(For the high noise on the square wave, the resolution is not optimal)

Moreover, we have measured the current of the normal mode and standby and are compatible with datasheet.