System on Chip Architecture
Lab 2 Report

Daniele Castro S253244
System-on-chip architecture
Politecnico of Turin

CONTENTS

LIST OF FIGURES

System on Chip Architecture Lab 2 Report

***Abstract*—In this lab I have implemented a simple function generator using the MCU STM32F051R8 on the discovery board. Doing it I have tried, as much as possible, to use MCU hardware support to make the final work perform at its best. Of course, the output signal has some analog hardware limitations: it can only output signals in the range of 0v and 3.3v. To shift and amplify it, for instance, between -1v and +1v, an analog output circuitry is required and I have made it.**

I. Introduction

Here is a small figure that explains better what this function generator does using hardware support:

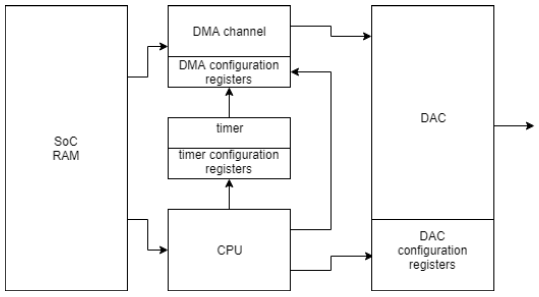

DMA channel
DMA configuration registers
timer
timer configuration registers
SoC RAM
CPU
DAC
DAC configuration registers

Fig. 1. block diagram

It is easy to understand that CPU only configures these peripherals. This means that, after the initialization stage, it can be switched off to drain less current. That is why I have put the CPU in STOP mode.

II. Background

The STM32F051R8 has 12-bit DAC (Digital to Analg Converter). It means that it can quantize an analog signal with $2^{12} - 1$ digital unsigned values. DMA channel 13 moves signal frames stored in memory to the DAC without CPU usage and timer TIM2 triggers the DAC to output a frame at $1/(1/(48 * (10^6)) * ((2^8) - 1)) = 188.2352941\text{MHz}$. Since prescaler and postscaler are disabled, TIM2 increments its count register at system frequency, that is 48MHz.

III. Proposed Solution

Of course, depending on the application, this laboratory can be made in different ways. For instance, there is the possibility not to use both timer and DMA and to do everything with CPU. This implementation can have some advantages like power saving because we are using less peripherals and it can be also implemented in cheaper hardware that has not any DMA channel like some 8-bit micro controllers. But its disadvantage is that it is limited in the maximum frequency at which samples can be output and code occupies much more memory. So, normally, a good approach is to choose hardware to be used depending on what we must do and, when it is present, use all the possible peripherals instead of emulating them in software. Stop mode is the best to put the CPU in after the initialization process because it does not disable peripherals and switches off only the CPU core. In terms of written code I have only made an header file called ”waveform” that contains samples of numerical values that I have calculated using a MATLAB script:

```
clear
clc
unit = ((2*pi)/1024);
y = 0:unit:(2*pi)-unit;
t = sin(y);
t = ((t+max(t))/(2*max(t)))*hex2dec('FFF');
numel(t)
t = int16(t);
plot(t);
M = vec2mat(t, 16)
numel(M)
F = string(M);
F(:, :, 1) = F(:, :, 1)+','+' ';
H = char(F);
dlmwrite('values_1024.txt', H,'delimiter','');
```

This specific script outputs, in a file sample, unsigned values for a full sine wave period so that, being DMA configured in circular mode, at the end of the sample, windows DMA pointer will start again from the beginning of the window. Changing `sin(y)` to `square(y)` this script will generate samples for a square wave. Both of these waves samples are saved in the file ”waveform”. Here is shown their definitions:

```
#include <stdint.h>

//#define SQUARE //comment for sinewave

#ifdef SQUARE
extern const uint16_t Square12bit[1024];
#else
extern const uint16_t Sine12bit[1024];
#endif
```

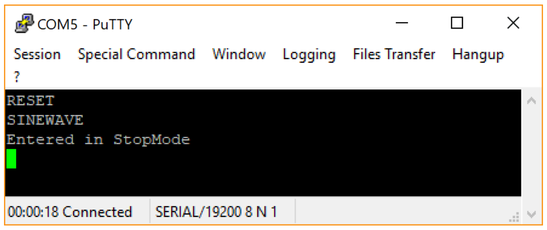
`#ifdef` statements are put only to have the possibility to choose, only on compile time, either to output a square wave or to output a sine wave. At run time only Escalator wave form can be changed with the current one previously selected by pressing the user button, but since the button pressure is CPU handled when we go into stop mode this event cannot be handled anymore and waveform cannot be changed at run time. Here is the code I have written:

```
...
else
{
        printf("Entered in StopMode\n\r");
        StopMode_Measure();
        USART2_init();
        printf("Exited from StopMode\n\r");
}
...
```

other small modifications I have made are in the DMA buffer lenght. I have put some defines and I recalculated period samples with the MATLAB script below:

```
...
                #ifdef SQUARE
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&
    Square12bit;
                #else
DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&
    Sine12bit;
                #endif
...
DMA_InitStructure.DMA_BufferSize = 1024;
...
```

I have also configured the serial port to output debug messages so that when waveform is going to be output can be known even before plugging the oscilloscope:



```
RESET
SINEWAVE
Entered in StopMode
```

Fig. 2. serial console

IV. Results and Discussion

The oscilloscope waveform output of the three waveforms are here displayed. Of course, there are noise distortions but signal is precise as we have lots of samples representing a single period:

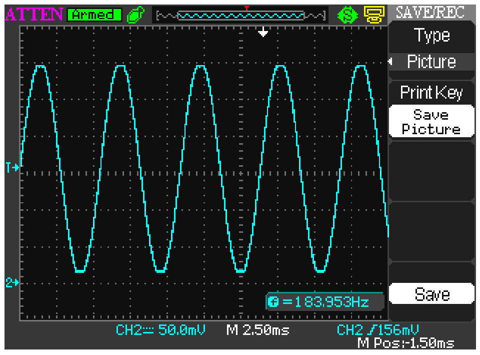


Fig. 3. sine waveform

I have also made tests with my hand-made analog voltage shifter I have talked in the abstract:

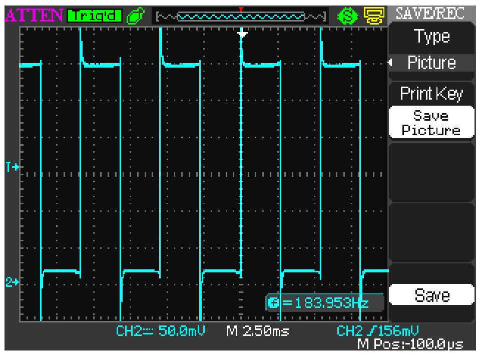


Fig. 4. square waveform

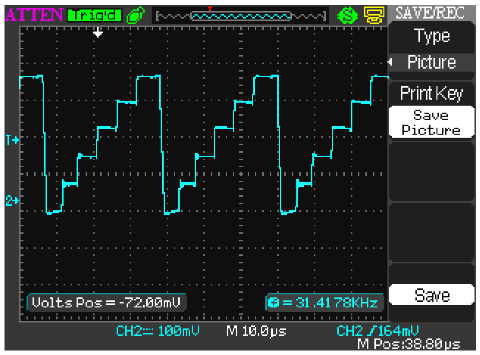


Fig. 5. escalator waveform

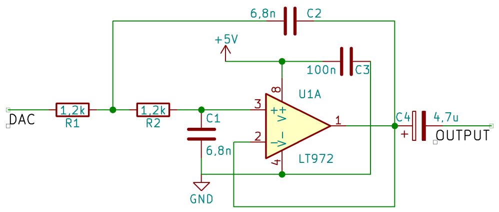


Fig. 6. analog circuit diagram

V. Results and Discussion

In this way I can connect everything to my PCs audio acquisition peripheral and print signals in the range of 20MHz - 20KHz. I have seen that waves I have modified are at 183Hz

VI. Conclusions

In this discussion I have matched the assignment requests and moreover, I have also made an analog voltage shifting circuitry.

References

[1] STM32F051R8 Reference Manual (RM0091), [pdf] Available at: Link.
Accessed on: Feb. 10, 2019.

[2] STM32F0DISCOVERY User Manual (UM1525), [pdf] Available at: Link.
Accessed on: Feb. 10, 2019.