Wolfson School of
Mechanical, Electrical and Manufacturing Engineering

Loughborough University

WSC055 – Laboratory 2

DAC to ADC

Introduction

The purpose of this laboratory is to engage the in-built analogue to digital converter (ADC) in the STM32F3 discovery board. This will be tested by using the inbuilt Digital to Analogue Converter (DAC) as a signal generator. You will make use of your previous timer-based interrupt code to generate the output from the DAC.

The tasks required to be completed for this laboratory are listed on **page 5.**

During the lab you will have to refer to your lecture notes (or the STM32F303x Reference manual) regarding the bit locations within the registers.

Dr. Peter Hubbard - *p.d.hubbard3@lboro.ac.uk*

An introduction to the STM32F303 ADC and DAC

Digital to Analogue Conversion

The on-board DAC on the STM32F3x is a fairly straightforward device with few configuration options. The fundamental principle is that data is written to a register within the DAC, that is converted to an analogue output in the next clock pulse cycle. As with any peripheral the DAC will require a clock input before we can interact with it.

Start-up Procedure:

1. Enable clock connection
    - The DAC is connected to the system clock via the APB1 peripheral clock bus. The connection is made by:

```
RCC->APB1ENR |= RCC_APB1ENR_DAC1EN;
```

2. Disable the 'buffer' function in the DAC control register

```
DAC1->CR |= DAC_CR_BOFF1;
```

3. Enable DAC peripheral

```
DAC1->CR |= DAC_CR_EN1;
```

The complete register details can be found in chapter 16 of the reference manual [1].

Write procedure

1. Write data to relevant data holding register. On the next clock cycle the DAC receives, it automatically selects the populated data holding register and converts the signal.
    - The DAC has a register for each resolution and data alignment option available:

Register name	Function
DAC1_DHR8R1	Holding Register for Right aligned, 8-bit data
DAC1_DHR12R1	Holding Register for Right aligned, 12-bit data
DAC1_DHR12L1	Holding Register for Left aligned, 12-bit data

The DAC analogue output is between 0 and 3.3V or 3V (depending on the V_{ref} used).

Analogue to Digital Conversion

In contrast to the DAC, the ADC peripheral presents many different options for configuration, as has been discussed in lecture notes. In this lab, the ADC will be configured to work in a simple format, and data will be read in a software-triggered manner with no real regard for sample time. Remember that the peripherals onboard are essentially 'unwired' so all connections and options need to be defined by setting appropriate options in registers. The key 'bits' within registers that we need to interact with have been covered in the lectures [2], but the complete list of registers and associated functions can be found in the reference manual [1].

Start-Up Procedure

The start-up procedure for the ADC is as follows:

1. Enable the Voltage regulator using the ADCx_CR register.
    - The minimum 10 usecond wait required can be achieved by a while loop that simple counts to 100 before exiting.
2. Calibrate the ADC using the ADCx_CR
3. Enable the clock connection to the ADC peripheral
4. Select input pin and configure necessary GPIO port
5. Configure ADC using the ADCx_CFGR register to have
    - 8-bit resolution
    - Right hand data alignment
    - Not continuous operation
6. Configure the Multiplexing Options
    - Write the channel number to be sampled in the SQR1 section of ADCx_SR
    - Define the length of the sequence to be '1'.
    - Define the sample time for the sampled channel by setting the value in SMP1 in the ADCx_SMPR1 register
7. Enable the ADC
8. Wait for the ADRDY flag to go 'high'.

Read procedure

In this mode for the ADC, the read functions are reasonably straight forward. Code is written that starts each individual conversion, then waits for the conversion to be complete before collecting the data. This can written in the main while loop (aka super loop) or as an ISR.

1. Start the conversion by setting the ADSTART bit high
2. Wait for the end of conversion (reported in the ADCx_ISR register by the EOC bit going high)
3. Read data from the data register (ADCx_DR). Doing this resets the EOC flag.

A Note on Sub-functions

The ADC start-up procedure is long-winded and it is sensible to isolate this from other code to enable debugging. This can be done using a sub function. In the C language, subfunctions need three parts:

1. **A Function prototype** - this goes before the 'main' function and defines any variables to be sent and returned.
2. **A Function definition** - goes after 'main' function and contains the activities required
3. **A Function call** - goes within the 'main' function. This starts the function and waits for a return)

An example of this in an embedded system is as follows:

```
void SubFunction(void); // Function prototype. The 'void suggests
that no variables are sent to this function and no variables are
returned

/* Main Function */
// Calls sub function then enters superloop
void main(){

SubFunction(); // Function call

while(1);
}

/* Function Definition of SubFunction */
// Counts from 0-9 then returns
Void SubFunction(void){
      int j = ;
      while(j<10){
            j++;
            }
      return;
      }
```

Laboratory Tasks

Task 1: DAC output

The task here is to configure the DAC to output an 8-bit value in response to a timer based interrupt. We will make use of the LEDs to show the value written to the DAC data register.

1.1. Start with the timer based interrupt code for the counter generated in Lab 1
1.2. Write code to engage the peripheral clock to the DAC, and configure an 8bit output on a chosen pin.
1.3. In the interrupt service routine developed in Lab 1, write the counter value to the data register for the DAC. Each second should therefore generate a new analogue output. Allow the LEDs to show the bitwise value currently being written to the DAC.

Task 2: ADC input

The ADC initialisation is fairly long winded. It therefore makes sense to write a sub-function to contain this to prevent confusion in the main function.
Make sure that you annotate your code sensibly during development

2.1. Write a sub-function that initialises an 8-bit ADC on a defined pin.
2.2. Modify the superloop in your code to include a read from the ADC peripheral, and to output the bitwise values on the LEDs. Remove the LED output from the DAC part of your code.
2.3. Collect a jumper lead and connect the output of the DAC to the input of the ADC. The lights should continue to count from 0x00 to 0xFF, with some noise and offset effects.

Questions:

- Is the counter value preserved during the conversion?
- What affects do you notice?

References

[1] ST, “STM32F3030x Reference Manual,” vol. DocID02255, no. April, 2007.
[2] P. D. Hubbard, “Lecture 7 - ADC Interfacing.” Loughborough University, 2017.