EEE3096S - Practical 4
2022

8 BCOS 3 - DACs and (more) PWM

8.1 Overview

8.1.1 Design overview

In the last practical we dealt with analog-to-digital convertors (ADCs). In this practical, we will be looking at their counterparts, digital-to-analog convertors, or DACs.

A DAC converts a digital code into an analog voltage or current and works on the general premise that $V_{OUT}/I_{OUT} = k * DigitalInput$, where k is a proportionality factor and the digital input is a number in the range of 0 to $2^{bits} - 1$.

DACs have a variety of uses, with the most important among them converting digital signals to analog audio in just about every single electronic audio system in the world.

The STM32F051 Discovery Board already has a build in DAC. Setting it up is trivial and there are plenty of online resources available if you wish to test it out. Instead, in this practical we will be making a simple DAC using PWM and a low-pass filter. You will need to build the circuit you designed for Tutorial 4, generate lookup tables for a few waves and set up the two timers needed for generating the PWM signal.

All of the code needed to initialize the peripherals used in this practical is automatically generated by STMCubeIDE using the .ioc file. If you want to see how these peripherals were configured, you can open up the configuration GUI by double-clicking on the .ioc file included in the project folder.

8.2 Outcomes and Knowledge Areas

You will learn about the following aspects:

- DACs
- PWM
- Filtering
- You should acquaint yourself with the STM32 HAL/LL documentation, available here.

8.3 Deliverables

For this practical, you must:

- Demonstrate your working implementation to a tutor
- Push your code to your shared repository.

8.4 Hardware Required

- STM32 Discovery Board
- Breadboard
- Your low-pass filter
- Dupont/Jumper Wires
- UART to USB cable
- Oscilloscope

8.5 Walkthrough

1. Clone or download the git repository.
   `$ git clone {https://github.com/UCT-EE-OCW/EEE3096S-2022`

2. Copy the STMCubeIDE (/EEE3096S-2022/WorkPackage4/EEE3096S_2022_Prac_4_PWM_DAC_Student_Version) project folder into your workspace and open up STMCubeIDE. If the project doesn't immediately show up you need to import the project by going to File - Import.. - Import existing Projects into Workspace and selecting the project. It will probably be the only project not greyed out in your workspace. Tick on the checkbox next to it and hit Finish. Open up the main.c file.

   ***Note about STMCubeIDE projects: The IDE provides a GUI which can be used to set up the clocks and peripherals (such as GPIO, UART, I2C) and then automatically generates the code required to do it in the main.c file. The configuration is stored in a .ioc file, which opens up the GUI if you try to open it. We have provided the .ioc file so that you can see how the pins are configured but please do not make/save any changes as this will regenerate the code in your main.c file.

   For the same reason, we also limit our code to the sections that begin with /*USER CODE BEGIN XXX */ and end with /* USER CODE END XXX */. For example, all our private functions are declared between /* USER CODE BEGIN PFP */ and /* USER CODE END PFP */ (PFP - Private Function Prototypes) and their implementations are between /* USER CODE BEGIN 4 */ and /* USER CODE END 4 */ near the end of the file.

3. **TASK 1:** Your first task will be to generate lookup tables (LUTs) for a single cycle of a sinusoid, a sawtooth wave and a triangular wave. Your lookup table should have a minimum of 128 values ranging from 0 to 1023. Plot your LUTs using Matlab or Excel to ensure you have the correct wave shape. Copy your LUTs into your main.c file.

4. **TASK 2:** Assign values to **NS**, **TIM2CLK** and **F_signal**. **NS** is the number of samples in your LUT, **TIM2CLK** is the frequency of the STM32 clock and **F_signal** is the frequency we want our analog signal to have. **F_signal** should not exceed 5kHz or the cutoff frequency of your filter, whichever is lower.

5. **TASK 3:** Calculate **TIM2_Ticks**. This is the number of cycles that the PWM duty cycle will be changed and depends on the clock frequency **TIM2CLK**, number of samples **NS** and output frequency **F_signal**.

   **Note: We will be using Direct Memory Access to change the PWM duty cycle. Direct memory access (DMA) is used in order to provide high-speed data transfer between peripherals and memory as well as memory to memory. Data can be quickly moved by DMA without any CPU actions. Check the .ioc file to see how the DMA is configured (as well as the other peripherals!).

6. **TASK 4:** In the **main()** function:

   - Start TIM3 in PWM mode on channel 1
   - Start TIM2 in Output Compare (OC) mode on channel 1.
   - Start the DMA in interrupt (IT) mode. The source address is one of the LUTs you created earlier. The destination address is the CCR1 register for TIM3.
   - Use **__HAL_TIM_ENABLE_DMA(htim2, TIM_DMA_CC1);** to start the DMA transfer.

   HINT: The 3 functions you need for TASK 4 begin with either HAL_DMA or HAL_TIM. Press Ctrl + Space to see the options once you have typed out the first few letters.

7. **TASK 5:** An interrupt has been configured on the blue push-button, and **EXTI0_1_IRQHandler(void)** is called when it is pressed. Write code that changes from one wave-form to the next when the button is pressed. Remember to debounce your button presses sensibly or risk losing 2 marks! Debouncing should eliminate noise from bouncing, but not make the response seem sluggish if the button is presses multiple times shortly after each other. Use **__HAL_TIM_DISABLE_DMA(htim2, TIM_DMA_CC1);** to stop the DMA transfer before the source address.

8. **TASK 6:** Test your filter using a signal generator and oscilloscope. Confirm that your filter attenuates frequencies above your cutoff frequency (we will be using a max of 5kHz but you're free to experiment).

9. **TASK 7:** Connect PA6 (TIM3_CH1) to the input of your filter. Make sure that the STM32 and your filter share a ground line. Scope the output of your filter. Try with different frequencies for **F_signal** and test all three of your wave forms.

10. **TASK 8:** Demo your working PWMed DAC to a tutor.

    ***Note about UART: Since there is no LCD connected to the Discovery board we will be sending our text outputs via the serial port using UART which we can monitor on a laptop or PC using Putty. You can download Putty here. UART connections are as follows: 5V-5V GND-GND RXD-PA2 TXD-PA3. Open device manager and go to Ports. Plug in the USB connector with the STM powered on. Check the port number (COMx). Open up Putty and create a new Serial session on COMx with baud rate of 9600. You may need to install the CP2102 driver from here if no new ports show up. If you'd prefer, you can use this python script to read from the serial port. You will need to change the COM port and baud rate and pip install pyserial.

    The UART comms has already been configured in the .ioc and main.c files. To send data via UART we create a char buffer, use sprintf to add our text to the buffer and then use the command **HAL_UART_Transmit(huart2, buffer, sizeof(buffer), 1000)** to send the buffer. All of this has already been provided to you. All you need to do is un-comment the lines of code and replace the contents of the buffer with the data you want to send. You can use this in conjunction with the debugger to test your code.

    Check out the HAL documentation if you want to learn more about how this works. There are also plenty of online resources available for working with the STM32 boards and their peripherals.

11. Upload your main.c file to your shared repository. Your file structure should resemble this: STDNUM001_STDNUM002_EEE3096S/Prac4/main.c assuming your shared repository is called STDNUM001_STDNUM002_EEE3096S. Add a link to your shared repository in your PDF submission.

8.6 Some Hints

1. Read the documentation of the HAL libraries.
2. The source code provided to you has a lot of implementation in it already. Ensure you read and understand it before embarking out on the practical.

8.7 Submission

Please submit a scanned copy or image of the mark-sheet that you will receive when you demonstrate your working code/circuit.