EEE3096S - Practical 4
2023

DACs, Filtering and (more) PWM

1 Overview

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

A DAC converts a digital code into an analog voltage or current and works on the general premise that $V_{OUT} = k * DigitalInput$, 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 STM Board already has a built-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 low-pass filter circuit, generate look-up tables (LUTs) for a few different waveforms, set up **two** timers for generating the PWM signal and cycling through the LUT values, and then use an oscilloscope to monitor the output from your filter.

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. But be careful not to edit anything or re-generate the code!

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.

3 Deliverables

For this practical, you must:

- Push your code to your shared repository on GitHub, then submit a PDF of your main.c code and GitHub repo link on Amathuba/Gradescope. Do this by copy-pasting your code (and GitHub link) into a document and saving it as a PDF; do NOT submit screenshots as the text needs to be searchable.
- After you have submitted your PDF, demonstrate your working implementation to a tutor in the lab; they will open your submission on Amathuba and mark your demo in real time.
- You will have two weeks to complete this practical, and as such, you will be allowed to conduct your demo during any lab session within these two weeks. By the deadline, you should have submitted your PDF, pushed your code to GitHub, and had your demo marked.

4 Hardware Required

- UCT STM Board
- Breadboard
- A low-pass filter on the breadboard
- Oscilloscope

5 Walkthrough

1. Clone or download the git repository.
   $ git clone https://github.com/UCT-EE-OCW/EEE3096S-2023
2. /EEE3096S-2023/WorkPackage4/Prac4_student is the project folder that you will need; open up STMCubeIDE and then go to File --> Import --> Existing Code as Makefile Project --> Next --> Browse to the project folder and then select "MCU ARM GCC" as the Toolchain --> Finish.
3. Open up the main.c file under the Core/src folder and complete the Tasks below. Note: All code that you need to complete is marked with a "TODO" comment; do not edit any of the other provided code.
4. **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.
5. **TASK 2**: Assign values to NS, TIM2CLK and Fsignal. NS is the number of samples in your LUT, TIM2CLK should be set to 8 MHz (which you can see in the .ioc file), and Fsignal is the frequency that we want our analog signal to have. Fsignal should **not** exceed 1 kHz or the cutoff frequency of your filter, whichever is lower.

6. **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 (DMA) to change the PWM duty cycle. DMA is used 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!).

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

- Start TIM3 in PWM mode on channel 3
- 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 CCR3 register for TIM3_CH3. Start with the Sine wave LUT.
- Write the waveform type to the LCD screen, i.e., "Sine".
- Make use of __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.

8. **TASK 5**: An interrupt has been configured on the PA0 pushbutton, 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; debouncing should eliminate noise from bouncing, but not make the response seem sluggish if the button is pressed multiple times shortly after each other. Use __HAL_TIM_DISABLE_DMA(htim2, TIM_DMA_CC1) and HAL_DMA_Abort_IT to stop the DMA transfer before changing the source address, then re-enable DMA.

9. **TASK 6**: As part of your interrupt, write the current waveform type to the LCD screen, e.g., "Sine", "Sawtooth", or "Triangular" depending on which LUT is selected after pressing the pushbutton.

10. **TASK 7:** 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 5 kHz, but you're free to experiment).

11. **TASK 8**: Connect PB0 (TIM3_CH3) to the input of your filter. Make sure that the STM32 and your filter share a ground line. Scope the output of your filter and test all three of your waveforms.

12. **TASK 9**: Demo your work to a tutor.

13. 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).

6 Some Hints

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

7 Submission

A PDF submission is required for this practical:

1. The PDF document must comprise your main.c code in text form, not screenshots. Also include a link to your GitHub repo.
2. After submitting the PDF, demo the aforementioned functionalities to a tutor in the lab. Your PDF must already be submitted by the time you call a tutor to your bench for the demo, as they will then be able to open your submission on Amathuba/Gradescope and assign marks accordingly.
3. All marks for this practical will come from your demo and your submitted code.