EEE3096S - Practical 3
2022

6 BCOS 2 - ADCs, Interrupts and PWM

6.1 Overview

6.1.1 Design overview

Many embedded systems rely on the use of analog devices for various reasons. Certain sensors, for example, output an analog voltage or current which the digital brain of the system needs to be able to read. Similarly, embedded devices often allow users to adjust parameters by turning the knob of a rotary potentiometer. In order to be able to process these analog signals we need to make use of an Analog to Digital Convertor, or ADC for short.

Embedded systems typically also have functionalities that have to happen at precise times, such as when a button is pressed. In simple cases, polling a GPIO register and waiting for a bit to change is often enough, but as complexity increases this approach becomes less useful. Think of an E-stop on a dangerous bit of machinery, for example. We would want the system to shut down as soon as the E-stop is pressed. With polling, the system would only be shut down once the lines to poll the GPIO pins are executed, wasting what could be valuable milliseconds if things go wrong. So, interrupts are used instead. By now you should already be familiar with interrupts, and this practical will guide you through setting up interrupts for the GPIO peripheral on the STM32.

In this prac, you will be using C to read the voltage on a potentiometers wiper using the onboard ADC on the STM32. You will then use that to calculate the duty cycle of a PWM signal which will be used to control the brightness of the blue built-in LED. You will also use interrupts to control when certain events occur.

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.

6.2 Outcomes and Knowledge Areas

You will learn about the following aspects:

- ADCs
- PWM
- Interrupts

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

6.3 Deliverables

For this practical, you must:

- Demonstrate your working implementation to a tutor
- Push your code to your shared repository and submit a PDF of your main.c file to Gradescope. Please copy and paste your code. DO NOT submit screenshots. The tutors should be able to CTRL+F their way through your PDF. All files submitted to Vula/Gradescope need to be in the format: pracnum_studnum1_studnum2.fileformat (although if you are submitting a zip file, you do not need to have all the sub-files named according to the student numbers, but do please have the report filename containing student numbers to ensure the marker is aware that it is a team submission).

6.4 Hardware Required

- STM32 Discovery Board
- 10K Potentiometer
- Breadboard
- Dupont/Jumper Wires
- UART to USB cable

6.5 Walkthrough

1. Clone or download the git repository.
   `$ git clone {https://github.com/UCT-EE-OCW/EEE3096S-2022`
2. Copy the STMCubeIDE (/EEE3096S-2022/WorkPackage3/EEE3096S_2022_Prac_3_ADCs_Interrupts_and_PWM_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:** For the first task you need to switch the frequency of the flashing LED between 1Hz and 2Hz when the blue push button is pressed. The function **void EXTI0_1_IRQHandler(void)** is called when an interrupt is generated on EXTIO (ie the blue pushbutton). Complete the function body. The function has already been declared and an empty function body has been provided. You may notice that occasionally the frequency does not seem to change when the button is pressed. Explain why this happens and find a way to fix the issue.

   HINT: The HAL_GetTick() function may be useful.

4. Connect the potentiometer. Connections are as follows: Wiper-PA7 Outer Pins-GND,3V (Connect the outer pins so that a CW rotation increases the wiper voltage). Use a breadboard and Dupont wires if needed.

5. **TASK 2:** With our potentiometer connected and the ADC configured, your next task is to complete the function **uint32_t pollADC(void)**. As before, the function has already been declared and the empty function body provided. This function should start, poll and stop the ADC and return the most recent ADC value. Test your function in the main loop with a 500ms delay (you can use the HAL_Delay(ms) function or implement your delay function from Prac 2). Display the ADC value on your laptop using the UART protocol and Putty.

   HINT: Many of the HAL functions you need for Tasks 2 and 3 begin with HAL_ADC. The functions also require a pointer to an ADC_HandleTypeDef structure which was automatically generated when the ADC was set up using the GUI.

   ***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.

6. **TASK 3:** Next, the function **uint32_t ADCtoCRR(uint32_t adc_val)** which converts an ADC value to a PWM duty cycle value (or raher, the CRR value that needs to be set for a specific duty cycle). Since the ADC is configured in 12-bit mode, the functions input would be integer value between 0 and 4096. TIM3 is configured to use channel 4 (PC9, Green LED) with the prescaler set to 0 and the auto-reload register set to 47999. This corresponds to a 1Khz frequency for the PWM signal. The duty cycle is the CCR (Capture/Compare Register)/ARR (Auto Reload Register). The macro **__HAL_TIM_SetCompare(htim3, TIM_CHANNEL_4, ccr_val)** can be used to set the CCR register to ccr_val. As before, the TIM_HandleTypeDef structure has already been created.
7. **TASK 4:** Complete the **int main(void)** function. Your program should:
   - Toggle the state of the blue LED.
   - Poll the ADC.
   - Set the PWM duty cycle of the green LED
   - Display the ADC and duty cycle values to the Serial port.
   - Toggle the delay interval when the blue push-button is pressed.
8. Demonstrate your working code to a tutor.
9. Upload your main.c file to your shared repository. Your file structure should resemble this: STDNUM001_STDNUM002_EEE3096S/Prac2/main.c assuming your shared repository is called STDNUM001_STDNUM002_EEE3096S. Add a link to your shared repository in your PDF submission.

6.6 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.

6.7 Submission

Please ensure that the following is included in your PDF submission.

1. A Github link to your shared repository which should contain your main.c file.
2. A copy of your main.c file (This is for marking purposes in case the tutors are unable to access your repository). Make sure that you copy the actual text. Do NOT submit screenshots of your code.