EEE3096S - Practical 2
2022

4 Intro to BCOS – Delays and I2C

4.1 Overview

One of the key features of many embedded operating systems is the ability to keep track of time. Being able to keep track of time is useful for many things, from simply displaying it (like most microwaves ovens) to using it to schedule operations as is needed, for example, in other household appliances such as ovens, dishwashers and MANY others.

From working with the STM32 previously, you should already be aware that there is no built-in means to keep track of time, besides using its own internal clock that could become rather inaccurate after a long time. This is because the STM32 doesn't have a built-in RTC (real time clock), or access to the internet, and thus the NTPD (Network Time Protocol Daemon) to fetch, set and store the date and time. There are a few ways of getting around this issue. The first is to manually set the time when flashing the board, and then use the available timers to keep track of the time. This, as you might expect, is not ideal: it requires the user to manually set the time each time the board is power cycled (turned off and on) and also requires a timer that is perpetually on. We could also add internet connectivity using something like a WIZ5500 Ethernet Controller. However, this is a very involved process which is beyond the scope of this course. Instead, the approach we will use is to add an external RTC which we will interface with using I2C. The RTC also needs to be set manually, but it has its own battery and maintains the time even if the STM32 is power cycled (as long as you dont overwrite the time!).

Hopefully by now you should already be familiar with the concept of UNIX time, epoch time or POSIX time and understand the benefits of using it. With that in mind, we will also write code for converting the time we get from the RTC to an equivalent epoch timestamp (with some adjustments to make things a bit easier).

4.1.1 Design overview

In this prac, you will be using C to write to time to an RTC module. You will also have an LED that will flicker on and off every second, as well as publish the time (and corresponding epoch time) using UART to a serial monitor on your PC/Laptop.

IMPORTANT: You cannot use any time libraries in your script, i.e. make sure you are using the I2C communication protocol between the RTC and STM32. Marks

4.2 Outcomes and Knowledge Areas

You will learn about the following aspects:

- I2C
- UART
- Real Time Clocks (DS3231 Documentation)
- Delays
- You should acquaint yourself with the STM32 HAL/LL documentation, available here.

4.3 Deliverables

For this practical, you must:

- Demonstrate your working implementation to a tutor. See the marking guide available on Vula. Demonstrations will need to be booked on Vula. The demos will likely happen in the week after this practical is released. The demonstration is to be semi-formal - you should introduce yourself, explain and summarize what you are going to show and be prepared to answer questions about the practical.
- 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).

4.4 Hardware Required

- STM32 Discovery Board
- DS3231 Module
- Breadboard
- Dupont/Jumper Wires
- UART to USB cable

4.5 Walkthrough

1. Clone or download the git repository.
   `$ git clone {https://github.com/UCT-EE-OCW/EEE3096S-2022`
2. Copy the STMCubeIDE (/EEE3096S-2022/WorkPackage2/EEE3096S_2022_Prac_2_Delays_and_I2C_Student_Version) project folder into your

workspace and open up the 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:** Ensure that only the line needed to toggle the state of the GPIO pin is un-commented in **int main(void)**. Scope the pin that is being toggled (it's the same pin that the blue LED is connected to) and take note of the frequency. Take a photo or screenshot of the scope to add to your PDF submission. Here is a guide on how to save screenshots from most of the scopes in White lab to a USB flash drive. Make sure that you can clearly see the shape of the graph.

4. **TASK 2:** Your next order of business will be to create a function that implements a delay by wasting clock cycles in a nested for loop. Give values to DELAY1 and DELAY2 in the main.c file (they both are currently defined as 0 under /* USER CODE BEGIN PD */ (PD - Private Define). (HINT: we are interested in DELAY1*DELAY2). Complete the implementation of **void pause_sec(float x)**. It has already been declared and the empty function body has been provided under /* USER CODE BEGIN 4 */. Test that your delay function works as expected by toggling the LED state with a 1 second and 60 second delay. Time the 60 second interval and comment on your results in your PDF submission.

5. **TASK 3:** The RTC uses BCD to store the time so we need to create functions that convert from BCD to decimal and vice versa. Complete the implementation for **uint8_t decToBcd(int val)** and **int bcdToDec(uint8_t val)**. They have already been declared and the empty function bodies provided.

***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: red-5V black-GND white(TX)-PA2 green(RX;unused)-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. 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. Connect the RTC Module. Connections are as follows (+)-5V (-)-GND D-PB7 (I2C1_SDA) C-PB6 (I2C1_SCL). Use a breadboard and Dupont wires if needed.

7. **TASK 4:** Next, define functions for setting and getting the RTC time. Unsurprisingly, the functions you need to implement are **void getTime (void)** and **setTime (uint8_t sec, uint8_t min, uint8_t hour, uint8_t dow, uint8_t dom, uint8_t month, uint8_t year)**. They have already been declared and the partially completed function bodies provided. A TIME structure has also been created and a global variable declared for use in your **getTime()** function. The HAL commands are provided, but you will need to fill out some of the parameters by reading the RTC datasheet and HAL documentation.

8. **TASK 5:** Implement a function that takes in a time HH-MM-DD hh:mm:ss and returns the epoch time. The function is defined as **int epochFromTime(TIME time)** and the empty function body provided. You have been given the epoch time (1640988000) for Saturday, January 1, 2022 12:00:00 AM GMT+02:00 as a starting point. Ignore leap seconds/years.

9. **TASK 6:** Complete the **int main(void)** function. You need to set the RTC time and then continuously read the time from the RTC, convert it to a UNIX epoch time and write both the time and epoch time to the serial port with a frequency of 1Hz.

10. Demonstrate your working code to a tutor.
11. 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.

4.6 Some Hints

1. Read the Docs. This includes the datasheet for the RTC (which you will absolutely have to do), as well as 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.

4.7 Submission

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

1. A photo/screenshot of your oscilliscope output (TASK 1)
2. Comments on your timer accuracy (TASK 2)
3. A Github link to your shared repository which should contain your main.c file.
4. 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.