Lab 4: I^2C Peripherals and OS

Kaicheng Wu
Department of Electrical and Computer Engineering
McGill University
Montreal, Canada
kaicheng.wu@mail.mcgill.ca
260892789

Tian Feng
Department of Electrical and Computer Engineering
McGill University
Montreal, Canada
tian.feng@mail.mcgill.ca
260913386

***Abstract*—In this lab, we implemented a program that reads I^2C sensor values and prints them to the terminal through UART, using button presses as user input to change mode.**

***Index Terms*—I^2C sensors, UART, RTOS, STM32**

I. Introduction

I^2C serves as a standard interface for peripherals, where each peripheral's register is designated a distinct address. STM's board support package provides us with a number of useful functions that initialize and uses the peripherals, while also taking care of scaling the sensor outputs.

UART is mainly used to establish a user interface for device configuration as well as for communication between computers and peripherals. In our lab, we will use it to redirect sensor data to the terminal.

RTOS allows us to break our program into a number of tasks and use OS directives to put them to sleep, to wake them up, or to coordinate between them.

For this experiment, we will develop a program that uses FreeRTOS to coordinate the following tasks: reading up to 4 different I^2C sensors; transmitting acquired sensor data over UART to a terminal; and changing the mode of the application to output data from the next sensor in the sequence by detecting a button press.

II. Implementation

A. Reading I^2C Sensors

We carefully reviewed the BPS driver document and moved the HTS221-related source code and header files to the project so that we could read the humid and temperature sensors using the provided function. Those read functions access the sensor's I2C address (in the case (u8)0xBE) and return a float i to the memory. the result is already calculated into real units.

B. Displaying Sensor Values on UART

Then we first overwrite the "fputc" function used in the stdio library, the "printf" function uses the function except for displaying floats, so could we use "printf" to simply print the UART messages to the terminal via the virtual com port. To make the device print data at 10Hz, one needs to comment out the dead loop at line 188 of the ECSE444Lab4(1).

To solve the problem that the UART can only send a byte at a time, we print the float temperature and humidity data by

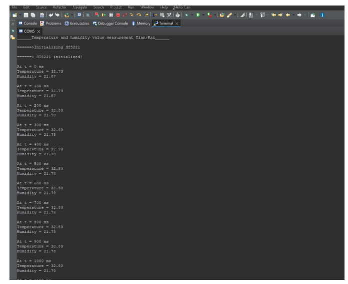

Fig. 1. The displaying result of the sensor reading at 10Hz, Baud rate =115200hz, word size =8, stop bit =1, parity = none

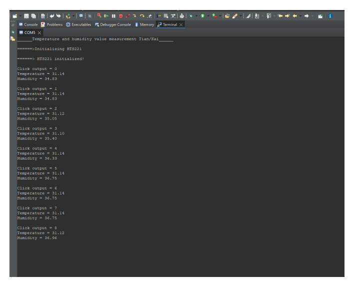

Fig. 2. Button driven outputs

converting the fraction part into integers, and all the numbers can be sent in chars.

C. Changing Sensors with the Push-button

When the dead loop at line 188 is enabled, the device will not display the readings automatically but when the button interrupt is detected. The interrupt callback function is implemented at line 109. The interrupt callback logic is not distinct from the auto-refresh one except that there's no *HAL_Delay()*.

D. FreeRTOS

We created 4 asynchronous threads to let the device output a different sensor recursively every time the button is pressed.

1) The Button_Click_Poll(): polls the button at 200Hz and changes the working sensor variable if not in Sensor_Change_Cooldown. This thread function has more than normal priority.

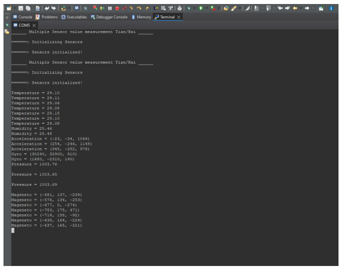

Fig. 3. RTOS Asynchronous Output

2) The Reset_Flags(): rests the Sensor_Change_Cooldown flag when the flag is on for 500ms.

3) The Sensor_SW_Poll(): The Sensor_SW_Poll() polls the sensors at the same 100Hz frequency and can be easily modified to assign each sensor to a different frequency.

4) The UART_Print_Data() : prints the Data the user button controls every 1000ms.

III. Limitations and Further Improvements

1) Mutex:: We can add a mutex to avoid the use of global variables and make the code more robust and readable.

2) Dynamic Memory Allocation:: The lab currently uses a static memory allocation which is safer. However, the code is not very agile for example if we had the array of dynamic length, the UART would not have needed to transfer those empty characters to improve performance.

3) Thread stack size:: The program can run on the current thread stack size configuration but any change can't be made on the configuration(640B). We may further need a bigger stack size.

4) Override of fputc() ineffective:: In the first part of the lab, we override fputc() to make the printf function able to use UART. However, in the second part, the modified printf function only works outside of thread functions. This problem needs further research.

IV. Demo

A video is contained in the zip file to show the demo.

V. Conclusion

To conclude, we implemented a program that uses FreeRTOS to coordinate sensor reading, output display, and button interrupts. Also, an interrupt-driven single-board program is implemented in the first part.

References

[1] RM0432 Reference manual STM32L4+ Series advanced Arm®-based 32-bit MCUs.(n.d.).
Retrieved from https://www.st.com/resource/en/reference_manual/rm0432-stm32l4-series-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[2] STM32L4S5xx STM32L4S7xx STM32L4S9xx Ultra-low-power Arm Cortex-M4 32-bit MCU+FPU, 150DMIPS, up to 2MB Flash, 640KB SRAM, LCD-TFT & MIPI DSI, AES+HASH. (n.d.).
Retrieved from https://www.farnell.com/datasheets/2602792.pdf

[3] ECSE444 Lab4 documents
Retrieved from https://mycourses2.mcgill.ca/