KETTERING UNIVERSITY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CE-426-01

Text-Based Messaging in a UART Interface

Darek Konopka

Breanna Krywko

Seth Hanft

June 17, 2021

Table of Contents

1. Objectives

The objective which we chose to accomplish was implementing two UART terminals to emulate a text messaging system between two users. In a lab assignment, we were asked to do something similar; send a character when it was entered into the UART terminal. This project expanded on that by asking us to record the characters pressed, but without sending them until a specific key (Enter) is registered. When 'Enter' is pressed, we want our code to send the message in the current UART terminal to the next terminal

Additional objectives provided by the assignment was to use special strings to display transmitted and received messages; determine a maximum message size, which we chose to be 50 characters; implement a low-power mode when the terminals are idle, i.e. when they are not sending or receiving any messages.

We also came up with our own additional objectives for this project. In our implementation, we wanted to use an additional UART terminal (UART#3) to act as a log file of the entire conversation, which users may not edit directly. We also wanted to have use of the 'Backspace' key, where it deletes the last character in the current UART terminal if it is pressed.

2. Purpose

The purpose of this project was to demonstrate and solidify our understanding of the concepts learned in Real-Time Embedded Systems. In order to build this, we had to use various techniques that were introduced to us throughout the labs and lectures of the course. There were some problems that we had to overcome which were not explicitly taught to us, such as initializing UART 3, implementing the Enter key so that no message is sent until it is pressed, and implementing the Backspace key so that it can edit keystrokes already entered into a terminal. The use of the Enter and Backspace keys is what pushes us to use one of the final concepts we learned in lecture; message queues.

We founds ourselves using the CMSIS-RTOS Reference Manual quite often, especially in order to implement UART#3. This pushed us to learn how to use the resources that are available to us more effectively.

3. Implementation

3.1 Threads

In our implementation we make use of several threads. These threads handle our first, second, and third person interfaces and make use of UART 1, 2, and 3 respectively.

3.2 Interrupt Request Handlers

We have 3 IRQs that handle the keystroke for each UART interface. These IRQS are also used to manually send a message telling the other thread that Person1/Person2 has sent a text message. On each interrupt the thread determines what to do depending on the character.

3.3 Mutexes

Our project has 1 implemented mutex for the UART terminal, which is shared by the first and second person threads. This secures the mutex while the threads are printing either their input for the user or the message they are receiving from the other person.

3.4 osEvents

In the current version of our project we have 3 osEvents that capture the result of a certain IRQ handler. We use these results to determine what is sent between the IRQs and act accordingly.

3.5 Message Queues

In our project we have 4 message queue. Each message queue is related to sending the text between UARTS. UART1toUART2 & UART1toUART3 are for sending the text message to both person2 and the logger (UART3). The same applies to UART2toUART1 and UART2toUART3.

3.6 Methods

The method that was chosen to complete the task at hand is to operate under 3 main threads. The two main threads are person1 and person2. These threads are used to handle the communication between the UARTs. Each thread operates by waiting for an IRQ signal before they execute the code at hand. While both threads are waiting for their respective IRQ signal thread 3, the os_idle_demon thread is executing to preserve power. This is our low power mode. There are two ways for an IRQ message to be sent, option A is from the IRQ handler that sends the character every time an input is detected in a UART. Option B however is a manual message sent into an IRQ. For option A the IRQ is a keystroke register from the UART. The thread receiving the interrupt then looks at the IRQ message, if the message shows that enter is pressed the thread sends the manual IRQ to the other person and sends them a text message. On receiving backspace the thread deletes the previous character, all other characters are published to the UART for the user to see what they type and saved in a message queue to use when sending the text. If the IRQ receives the specified manual message sent the thread will then print the message queue corresponding to that text received. On every keystroke the UART3 acts as a log and saves each keystroke by each person. Figure 1 is the block diagram for a visual representation of our method.

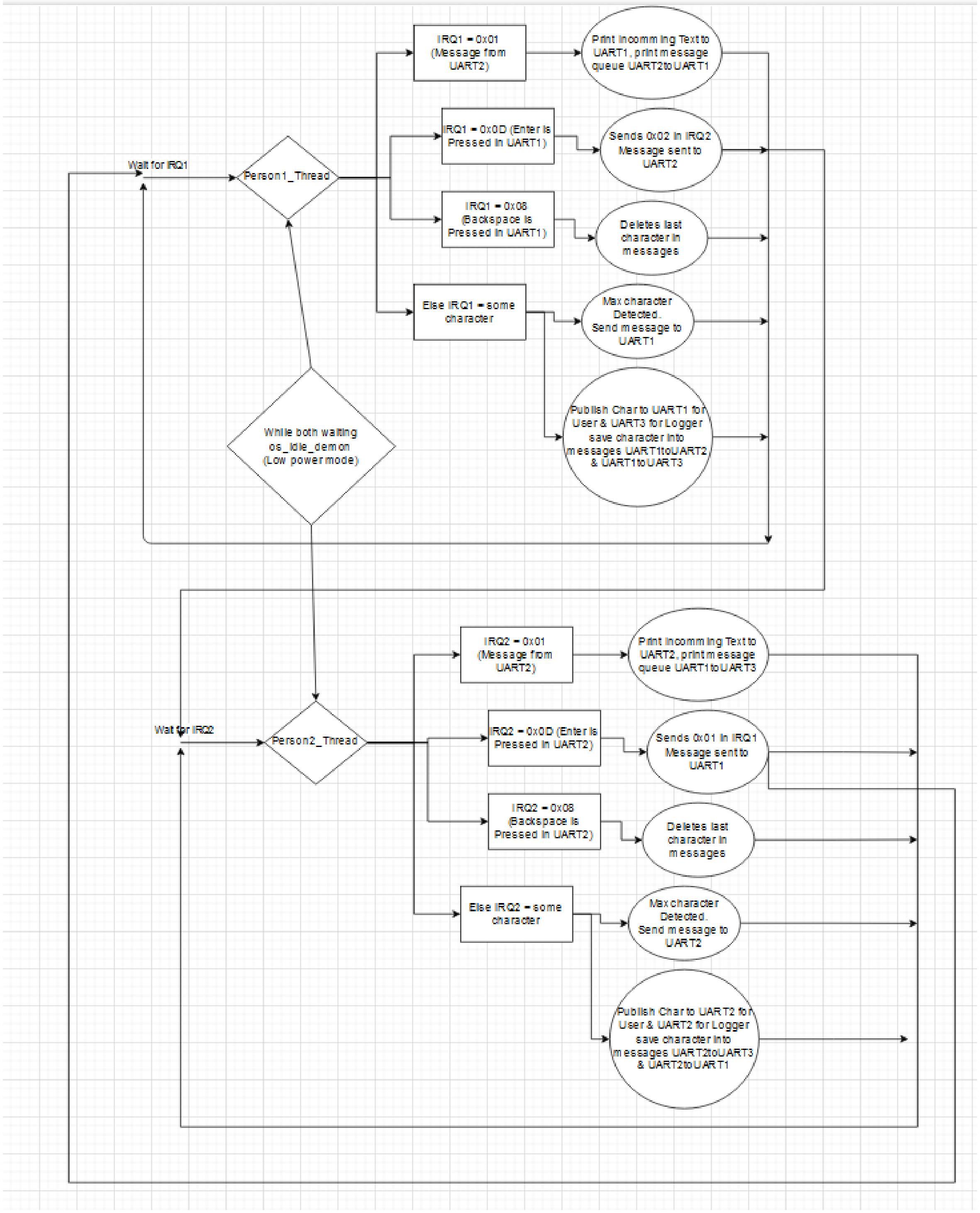
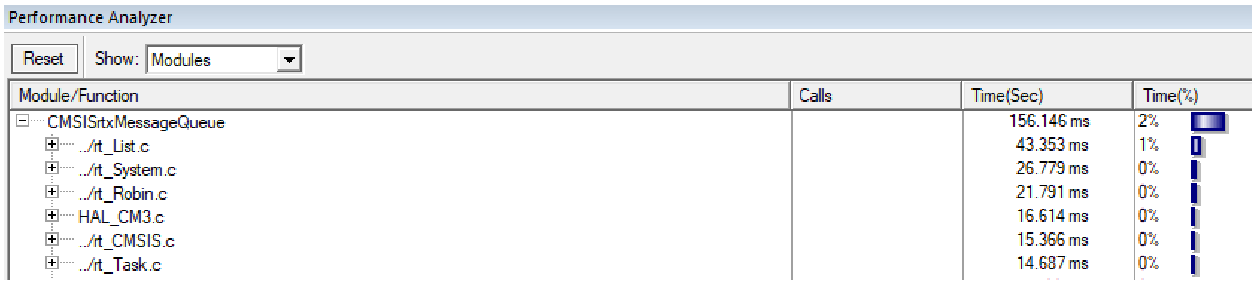

Wait for IRQ1
Person1_Thread
IRQ1 = 0x01 (Message from UART2)
Print Incomming Text to UART1, print message queue UART2to UART1
IRQ1 = 0x0D (Enter is Pressed in UART1)
Sends 0x02 in IRQ2 Message sent to UART2
IRQ1 = 0x08 (Backspace is Pressed in UART1)
Deletes last character in messages
Else IRQ1 = some character
Max character Detected. Send message to UART1
Publish Char to UART1 for User & UART3 for Logger save character into messages UART1toUART2 & UART1toUART3
While both waiting os_idle_demon (Low power mode)
Wait for IRQ2
Person2_Thread
IRQ2 = 0x01 (Message from UART2)
Print Incomming Text to UART2, print message queue UART1to UART3
IRQ2 = 0x0D (Enter is Pressed in UART2)
Sends 0x01 in IRQ1 Message sent to UART1
IRQ2 = 0x08 (Backspace is Pressed in UART2)
Deletes last character in messages
Else IRQ2 = some character
Max character Detected. Send message to UART2
Publish Char to UART2 for User & UART2 for Logger save character into messages UART2toUART3 & UART2toUART1

Figure 1: Block Diagram

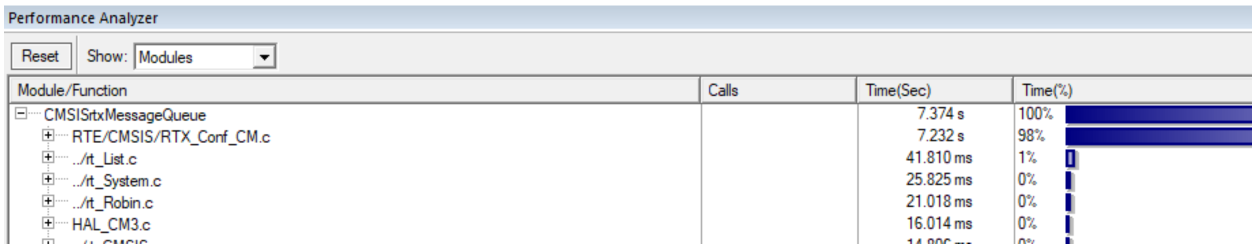
3.7 os_idle_demon

When none of the user-defined threads are ready to run, the system lets the idle thread to run. The os_idle_demon has been modified to keep the code in low-power mode when no interrupts are occurring. As a reminder, in this code, interrupts happen when a key is pressed, and the UART interrupt is enabled. To prove this works, below are two images of the performance analyzer when the code has low-power mode implemented and when the code does not have low-power mode implemented.



Module/Function	Calls	Time(Sec)	Time(%)
CMSISrtxMessageQueue		156.146 ms	2%
../rt_List.c		43.353 ms	1%
../rt_System.c		26.779 ms	0%
../rt_Robin.c		21.791 ms	0%
HAL_CM3.c		16.614 ms	0%
../rt_CMSIS.c		15.366 ms	0%
../rt_Task.c		14.687 ms	0%

Figure 2: Code with low-power Mode Implementation



Module/Function	Calls	Time(Sec)	Time(%)
CMSISrtxMessageQueue		7.374 s	100%
RTE/CMSIS/RTX_Conf_CM.c		7.232 s	98%
../rt_List.c		41.810 ms	1%
../rt_System.c		25.825 ms	0%
../rt_Robin.c		21.018 ms	0%
HAL_CM3.c		16.014 ms	0%

Figure 2: Code without low-power Mode Implementation

4. References

CMSIS-RTOS Tutorial

John kneen microcontrollers tutorial

https://sites.google.com/site/johnkneenmicrocontrollers/sci_rs232/fci_f107?tmpl=%2Fsystem%2Fapp%2Ftemplates%2Fprint%2F&showPrintDialog=1

Used to figure out how to initialize UART2

STM Reference manual for STM32F103xx

https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Used to figure out how to initialize UART3