Lab 5

Kyle Jeffrey

Abstract—The fifth lab uses the PSoC Micro and the Raspberry PI and UARTS for communication between the two.

1 Introduction

This lab covers a software implementtion of UART communications. The Raspberry Pi is used as a transmission device with provided code. The PSoC 6 will use Free Real Time Operating System(Free RTOS) to make two Software UARTS.

2 Part 1: Single Soft UART with Error Monitoring

A single Software instance of UART is developed using the Free RTOS tasks. 4 Free RTOS Tasks were used to accomplish capturing the throughput, printing to the terminal, reading from a selected RX Pin, and turning on LED if errors are detected. The UART needs to also check for a parity error and for a framing error. The framing error check is as simple as checking that the stop bit is a 1. The parity error check needs to make sure that the number of 1's in the 9 bits was odd. From this, an LED was set to half brightness to represent a parity error, and full brightness for a framing error.

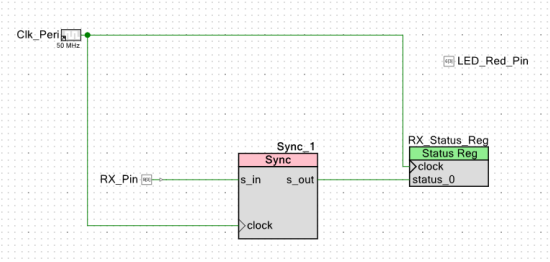

Fig. 1. Top Design

2.1 UART Software Read

In order to accomplish good sampling of the data for the software, the software loop, displayed below that, is implemented as a freeRTOS task:

1. Sample RX serial input and wait for negative edge
   1.1 Wait for 1.5x bit time to align with middle of first data bit.
   1.2 Sample data bit and add to byte. If all 8 data bits done, exit to step 2.
   1.3 Wait for 1x bit time and got to step 1.2.
2. Wait for 1x time and sample parity bit
3. Wait for 1x time and sample stop bit.
4. If stop bit ≠ 1, set framing error. Else if parity bit ≠ computed parity, set parity error.
5. Return data and error status

Fig. 2. Software Pseudocode

Using this software loop accomplishes reading the UART like so:

Fig. 3. UART Read

2.2 LED Display

The LED Display is a task that runs every 10 tick cycles. The error flags are set in the UART read task. To accomplish half brightness for the parity error, task delay is used between turning the LED off and on.

2.3 Testing

It was helpful to lower the baudrate to 50 for testing purposes to have a much easier time

reading out the data. The Raspberry Pi was configured to send a single constant byte repeatedly for ease of checking the data validity. The AD2 also includes a logic analyzer that can be configured to read UART. This was extremely helpful to read the output of the Raspberry Pi, to assure the data being sent was the data desired at the output.

2.4 Top Level Design

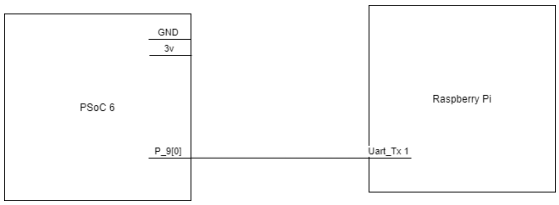

Fig. 4. Top Level Design

3 Dual Software Uarts with Error Monitoring

The dual software UART is just a doubling of the single UART of part 1. The only thing that needs to be added is a second LED, and LED task for displaying errors on the second UART. One error that was run into was creating too many tasks and using up all the heap space, so the tasks size need to be monitored.

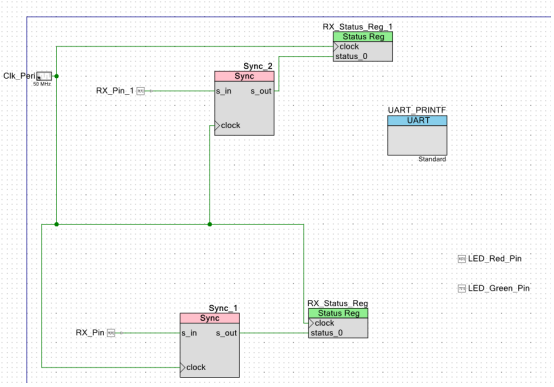

Fig. 5. UART Read

3.1 Hardware

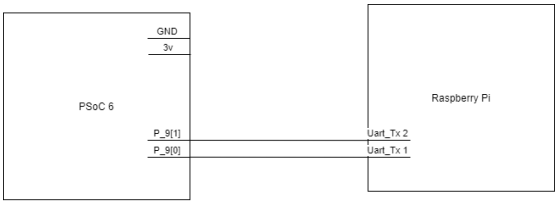

Fig. 6. Top Level Design

4 Conclusion

This lab explored using freeRTOS to handle complicated programs. Without using freeRTOS, handling the programs would be far more complicated, and nearly impossible to maintain the proper timing scheme required to maintain proper Baud Rates on a UART communications.

Kyle Jeffrey is a Senior Robotics Engineering Student at 6the University of California Santa Cruz. He is the Secretary of the Engineering Fraternity Tau Beta Pi and the lead Hardware Engineer at the on campus startup Yektasonics.