Parker Piedmont

ECE 443

9/24/2021

Project 3 Report (project as of 10pm on Tuesday, 9/22/2021)

What Works

- UART RX interrupt
- EEPROM reads/writes
- EEPROM FIFO order
- LEDA/LEDB
- Heartbeat
- LCD printout (sort of – can be fixed by changing one character)

What Doesn't Work

- LCD scrolling
- Queued button presses

Implementation

Tasks

- printToLCD: When BTN1 is pressed, this task reads a message from the EEPROM and displays it on the LCD. Every time it runs, it frees one slot in the EEPROM. This task is normally blocked but can be unblocked by giving a semaphore.
- writeToEEPROM: This task receives a character from a queue and appends it to a string. When it receives a carriage return, it replaces the '\r' with a null character and writes the string to the EEPROM. Every time it runs, it consumes one slot in the EEPROM.
- isEEPROMFull: Checks the number of free slots in the EEPROM. If the EEPROM is not full, it lights LEDA. If the EEPROM is empty, it lights LEDB.
- toggleLEDC: Toggles LEDC every millisecond.

Inter-Task Communication

- Queue: Transmits data from the UART receive ISR to writeToEEPROM. Every time the UART ISR runs, it sends one character to the queue, and every time writeToEEPROM runs, it receives all characters in the queue, one at a time.
- Semaphore: Controls the execution of printToLCD. When BTN1 is pressed, the CN ISR will give the semaphore, unblocking printToLCD.
- Global integer: Counts the number of free message slots in the EEPROM. writeToEEPROM uses it to determine whether a message can be stored in the EEPROM, printToLCD uses it to determine whether there is a message to read from the EEPROM, and isEEPROMFull

uses it to determine whether to light LEDA and LEDB. writeToEEPROM decrements the counter, and printToLCD increments it.

EEPROM Configuration

To implement the five-message FIFO, I treated the EEPROM as a circular buffer. I set a starting address for the first message in the EEPROM (0x10) and an offset (0x20) that would be added to the address for each additional message. I used two counters to keep track of which address was up next to be read or written and used modular arithmetic to ensure the counters would never exceed four.

When communicating with the EEPROM, I always wrote or read an 81-character string (80-character message plus null terminator) and filled the slack space after the last character of the message with null characters. I did this particularly for writes to ensure that writing a short message to the EEPROM would delete the extra characters left over from a longer message. In hindsight, this was not the best approach, as I could have modified my EEPROM library to stop at null characters (I actually tried that but ran into trouble), which would result in faster transactions.

LCD Configuration

The LCD was the weakest part of my implementation. I wanted to make sure all the communication worked properly before adding more advanced features, and I knew focusing on message breaking and LCD scrolling would get in the way of setting up the fundamentals. I wasn't able to finish these features by the deadline, so my program simply prints the whole message to the LCD using LCD_puts().

BTN1 Configuration

I configured BTN1 to generate a CN interrupt, which would unblock a task that prints to the LCD. I disabled the interrupt at the start of the ISR and waited until after the button finished bouncing to re-enable the interrupt and clear the interrupt flag. This ensured that the ISR would only run once per button press or release. Unfortunately, I didn't think to wait for the button to be released before re-enabling the interrupt, so the ISR will run when the button is pressed and when it is released. I also forgot to change the binary semaphore to a counting semaphore before submitting my project, which would have enabled queued button presses.

Challenges

Other than implementing LCD scrolling, my greatest challenge in this project was communicating with the EEPROM. I started my development process by simply writing a message to the EEPROM, reading it back, and checking to see if they matched. Since I knew my EEPROM library worked, I expected the read and write to match. This was the case if I ran my project outside debug

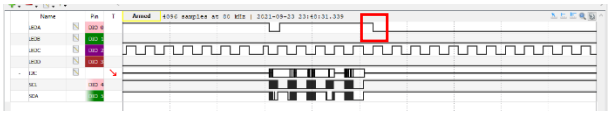
mode, but not if I ran it inside debug mode. I remembered having trouble debugging I2C communications in ECE 341, so I ditched the debugger and verified the transmissions using LEDs and the logic analyzer.

Another challenge I encountered was in displaying messages on the LCD. I noticed that sometimes a message or two would fail to print if multiple were queued up, but the following messages would print successfully. I discovered after submitting my project that this occurred because I stored my messages too close together. Because I wrote null characters in the slack space of each message, sometimes the null characters would overflow from one message into the next and delete part of it. I was able to fix this by increasing the offset between message start addresses to 0x60.

Results

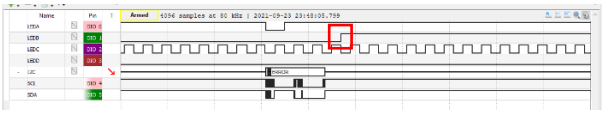
Other than some messages failing to print to the LCD (too bad I didn't catch that until last night), the features I implemented in this project worked as I expected them to. The instrumentation LEDs (shown below) lit up at the appropriate times, and the heartbeat was consistent.

LEDA on – EEPROM not full



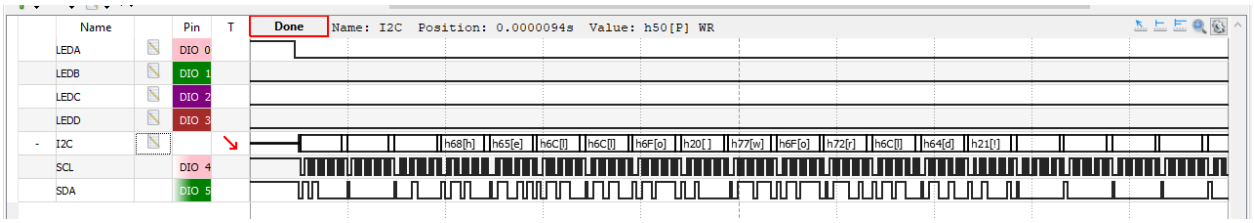
LEDA off – EEPROM full

LEDB off – EEPROM not empty

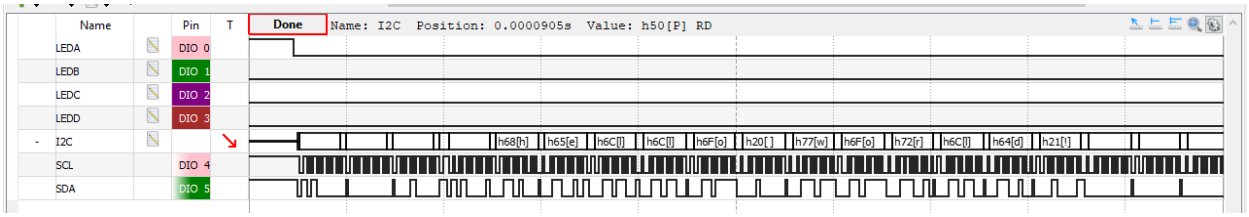


LEDB on – EEPROM empty

Communication with the EEPROM also worked properly. I tested this by sending "hello world!" to the PIC32 from PuTTY and recording the I2C transactions for the EEPROM read and write. The screenshots below show that my program sent and received the message correctly (followed by null character slack space).



EEPROM write



EEPROM read

Tracealyzer showed some important characteristics of my project. One is that the CPU usage was pinned at 100%. This is because writeToEEPROM and isEEPROMFull have no conditions for executing, so they will run whenever they get a chance to. This means that they will alternate back and forth while other tasks are blocked, as shown below. Between the scheduling of each task, toggleLEDC ran once.

3.722.068	EEPROM	Context switch on CPU 0 to EEPROM	
3.723.004	EEPROM	OS Tick: 3723	
3.723.017	EEPROM	Actor Ready: Toggle	
3.723.032	Toggle	Context switch on CPU 0 to Toggle	
3.723.045	Toggle	vTaskDelay(1)	
3.723.068	EEPROM (2)	Context switch on CPU 0 to EEPROM	(2)
3.724.004	EEPROM (2)	OS Tick: 3724	
3.724.017	EEPROM (2)	Actor Ready: Toggle	
3.724.032	Toggle	Context switch on CPU 0 to Toggle	
3.724.045	Toggle	vTaskDelay(1)	

Tracealyzer also showed how long communicating with the EEPROM took. Since I always wrote the full 81-character string to the EEPROM, I was guaranteed to have to wait for page writes, which would increase write time. This was reflected in Tracealyzer, as the write duration was about 11ms, but the read duration was only 2ms.

3.718.193	EEPROM	[EEPROM] EEPROM not full
3.728.956	EEPROM	[EEPROM] Wrote to EEPROM

EEPROM write duration

4.181.146	LCD Pri	[EEPROM] EEPROM not empty
4.183.478	LCD Pri	[EEPROM] Read from EEPROM

EEPROM read duration

Tracealyzer showed the amount of time my project spent in each ISR as well. Both ISRs took only about 100µs to execute, as shown below.

1.759.000	EEPROM (2)	[UART RX ISR] Entered C portion	
1.759.029	EEPROM (2)	[UART RX ISR] Echoed to terminal	
1.759.059	ISR using Queue #1	Context switch on CPU 0 to ISR using…	
1.759.059	ISR using Queue #1	xQueueSendFromISR(Queue #1)	
1.759.059	EEPROM (2)	Context switch on CPU 0 to EEPROM	(…
1.759.071	EEPROM (2)	[UART RX ISR] Sent char to queue	
1.759.097	EEPROM (2)	[Interrupts] Cleared UART int flags	
1.759.126	EEPROM (2)	[UART RX ISR] Exiting	

UART RX ISR

4.264.043	Toggle	[CN ISR] Entered C portion	
4.264.069	Toggle	[Interrupts] Disabled CN interru…	
4.264.102	ISR using Semaphore #1	Context switch on CPU 0 to ISR u…	
4.264.102	ISR using Semaphore #1	xSemaphoreGiveFromISR(Semaphore…	
4.264.102	Toggle	Context switch on CPU 0 to Toggle	
4.264.115	Toggle	Actor Ready: LCD Pri	
4.264.124	Toggle	[CN ISR] Gave semaphore	
4.264.146	Toggle	[CN ISR] Exiting	

CN ISR