Senior Design Progress Report

|  |  |  |  |
| --- | --- | --- | --- |
| **Student**: | Brian Dye | **Team**: | 20 ENIGMA |
| **Semester**: | Spring 2022 | **Position**: | Team Leader |
| **Week**: | **5** | **Hours**: | 25 |

# Progress Description

This was a successful week. My team and I were able to interface with the RFM69HCW radio module using SPI. This is the most important PSCC we have and it’s the most important interface to successfully create a mobile ad-hoc network. While I was setting up the SPI connection between the RFM69 and STM32 microcontroller I wanted to create objectives that ensure the SPI configuration is correct and accurate. So, I gave my team the objective of 1) successfully writing to a register on the RFM69 and 2) successfully reading from a register on the RFM69. Achieving these two goals will allow us to utilize all the functionality that the RFM69 provides such as sending data, receiving data, determining signal strength, etc.

I created a system that would allow us to test our achievement of these goals. I set up a process that attempted to write an 8-bit value to the **address register** in the RFM69. Then immediately read from the same register. The process looks as follows:

**char** \* address = "ABCDE";

**for** (**int** i = 0; i < 5; i++) {

RFM69\_setAddress((uint8\_t) address[i]);

regValue = RFM69\_readReg(REG\_NODEADRS);

}

If successful, on the first iteration of the loop the STM32 should write the **value 0x65 (‘A’)** to register **0x39** on the RFM69 and then read the **value 0x65** from **register 0x39**. On the second iteration of the loop, the STM32 would write the **value 0x66 (‘B’)** to **register 0x39** on the RFM69 and then read the **value 0x66** from **register 0x39** on the RFM69.

The issue can be further demonstrated using the following table:

|  |  |  |
| --- | --- | --- |
| **Iteration** | **Value written to address register** | **Value read after writing address value** |
| 1 | ‘A’ (0x65) | NULL (0x00) |
| 2 | ‘B’ (0x66) | ‘A’ (0x65) |
| 3 | ‘C’ (0x67) | ‘B’ (0x66) |
| 4 | ‘D’ (0x68) | ‘C’ (0x67) |
| 5 | ‘E’ (0x69) | ‘D’ (0x68) |

So, every time I attempted to write a value to a register, I wouldn’t be able to read the correct byte until I sent another byte to the register. I probed the SPI channel using an oscilloscope, and I came to understand that even though the SPI is configured for an 8-bit word, the SPI channel is generating a clock signal for another 8-bits after 8 bits of data is sent out. This issue is demonstrated in the following figure:

A picture containing timeline

Description automatically generated

We fixed the issue using a special cast on the data register of the SPI channel:

\*(uint8\_t \*)&(SPIx->DR) = txBuffer[i];

This told the compiler to treat the 16-bit SPI data register as an unsigned 8-bit value. This transformed the SPI channel activity into **exactly what we need**:

Diagram

Description automatically generated with low confidence

Now we can successfully write to an RFM69 register and read from an RFM69 register which gives us complete access to all the functionality that the RFM69HCW provides.

# Bluetooth Module HM-19

I worked extensively with Henry and Nathanial to configure the HM-19 bluetooth module using UART. We setup the UART pin connection to the STM32. As well as setup the UART configuration registers on the STM32. However, there is still work that needs to be done. We were unable to receive a response from the HM-19. I believe the issue lies in the configuration of the UART on the STM32. It’s diffcult to understand the required configurations of the HM-19 and what it expects from the UART connection because the documentation isn’t very good. This weekend, and next week I plan on using the oscilloscope to gain an understanding of the electrical oscillations that are occuring during the UART communication to debug the problem.

I’ve been contemplating using the HC-05 Bluetooth module. It has much better documentation. However, it’s less power efficient. I haven’t been able to find much information online for the HM-19 and the manufacturer information is sparse.