Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/17/2021

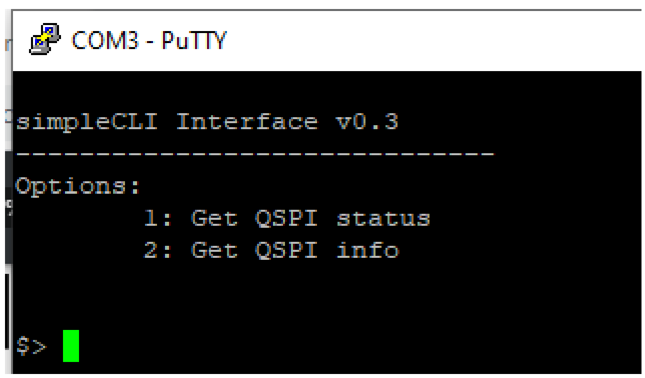
Assignment 5: SPI Hands On

The following will document completion of the fifth assignment for ECE-40293, using the onboard serial hardware to complete the following user stories:

1. **Create a CLI (Command Line Interface) on UART1 that prompts you to enter as follows:**
2. **BSP_QSPI_GetStatus(). When the user selects 1, display the results of calling BSP_QSPI_GetStatus() on the console.**
3. **BSP_QSPI_GetInfo(). When the user selects 2, display the results of calling BSP_QSPI_GetStatus() on the console.**

1. **Create a CLI (Command Line Interface) on UART1 that prompts you to enter as follows:**

To start, we will create a new default project as done in previous assignments, and will also reuse the code that was used in previous CLI-based User Stories, with modifications as necessary to update the "menu" to reflect the SPI options described above (see following page for code excerpt).



Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/17/2021

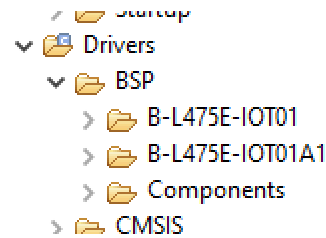
```c
  // Header info for CLI
  char* cliHeader = "\nsimpleCLI Interface v0.3\n------------------------------\n";
  HAL_UART_Transmit(&huart1, (uint8_t*) cliHeader, strlen(cliHeader), 1000);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
      // Define strings to structure prompt around
      char* cliPrompt = "Options:\n\t1: Get QSPI status\n\t2: Get QSPI info\n\t\r\n$> ";
      char* cliResponse = "Invalid input!\r\n";

      // Issue prompt
      HAL_UART_Transmit(&huart1, (uint8_t*) cliPrompt, strlen(cliPrompt), 1000);

      // Get the user selection and echo it on the terminal
      char cliInput;
      HAL_UART_Receive(&huart1, (uint8_t*) &cliInput, 1, HAL_MAX_DELAY);
      HAL_UART_Transmit(&huart1, (uint8_t*) &cliInput, 1, 1000);

      // Evaluate input
      switch (cliInput)
      {
        case '1':
            cliResponse = "\r\nQSPI status request:\n";
            HAL_UART_Transmit(&huart1, (uint8_t*) cliResponse, strlen(cliResponse), 1000);
            do_qspi_status();
            break;

        case '2':
            cliResponse = "\r\nQSPI info request:\n";
            HAL_UART_Transmit(&huart1, (uint8_t*) cliResponse, strlen(cliResponse), 1000);
            do_qspi_info();
            break;

        default:
            HAL_UART_Transmit(&huart1, (uint8_t*) cliResponse, strlen(cliResponse), 1000);
            break;
      }

  }
  /* USER CODE END 3 */
}
```

Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/17/2021

2. BSP_QSPI_GetStatus(). When the user selects 1, display the results of calling BSP_QSPI_GetStatus() on the console.

With the console interface developed, we will need to develop the BSP-related function calls to get the device status and info. As the process to import the BSP files was covered in a dedicated assignment for ECE-40291, I won't go into great detail on that process other than to say that the appropriate BSP files for the correct firmware version will need to be manually imported, the project's includes path will need to be updated, and the necessary header files will need to be added to the main.c User Includes section, as seen here:




```c
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stm32l475e_iot01.h"
#include "stm32l475e_iot01_qspi.h"

#include <stdio.h>
#include <string.h>
```

Additionally, we will need to call the *BSP_QSPI_Init()* function prior to entering the *while(1)* loop:

```c
  /* USER CODE BEGIN 2 */

  BSP_QSPI_Init();

```

With all of the support files added, we can define the wrapper function *do_qspi_status()*, which will call *BSP_QSPI_GetStatus()* and report the results over the UART to our serial console.

Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/17/2021

```
void do_qspi_status(void)
{
    uint8_t status = BSP_QSPI_GetStatus();

    char buffer[100];
    snprintf(buffer, sizeof(buffer), "QSPI status: %d\r\n", status);

    HAL_UART_Transmit(&huart1, (uint8_t *) buffer, strlen(buffer), 1000);
}
```

When this option is called from the user interface, we see the following result:

```
$> 1

QSPI status request:
QSPI status: 0

Options:
        1: Get QSPI status
        2: Get QSPI info


$>
```

Tracing back the function definition for *BSP_QSPI_GetStatus()* within stm32l475e_iot01_qspi.h, we find that the value of 0 is associated with a macro for QSPI_OK, indicating that we did have a successful status read.

```
/** @defgroup STM32L475E_IOT01_QSPI_Exported_Constants QSPI Exported Constants
  * @{
  */
/* QSPI Error codes */
#define QSPI_OK            ((uint8_t)0x00)
#define QSPI_ERROR         ((uint8_t)0x01)
#define QSPI_BUSY          ((uint8_t)0x02)
#define QSPI_NOT_SUPPORTED ((uint8_t)0x04)
#define QSPI_SUSPENDED     ((uint8_t)0x08)
```

Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/17/2021

4. BSP_QSPI_GetInfo(). When the user selects 2, display the results of calling BSP_QSPI_GetStatus() on the console.

Moving to the definition of *do_qspi_info()*, we see it is another simple wrapper function for the BSP call to *BSP_QSPI_GetInfo()*, which stores the result in an instance of the typedef QSPI_Info. We then see the flash size, erase sector size, and program page size printed out on the console.

```c
void do_qspi_info(void)
{
    QSPI_Info info = {0};

    BSP_QSPI_GetInfo(&info);

    char buffer[100];
    snprintf(buffer, sizeof(buffer), "Flash size: %lu\r\n"
            "Erase Sector Size: %lu\r\n"
            "Prog Page Size: %lu\r\n",
            info.FlashSize,
            info.EraseSectorSize,
            info.ProgPageSize);

    HAL_UART_Transmit(&huart1, (uint8_t *) buffer, strlen(buffer), 1000);
}
```

```
$> 2

QSPI info request:
Flash size: 8388608

Erase Sector Size: 4096

Prog Page Size: 256

Options:
         1: Get QSPI status
         2: Get QSPI info


$> 
```

Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/17/2021

Closing Thoughts

I've played around a bit with SPI before, specifically with using cheap monochrome LCD displays as well as with programming members of the Atmel (now Microchip) AVR family, such as the ATMega328 or ATTiny85. I don't know that I would call SPI at a lower level or even on bare metal super complicated but even then, the BSP functions used in this assignment was orders of magnitude easier to deploy and have results in just a few minutes. I guess that would be a perk of having had some team of software engineers at ST develop the dozens of support files necessary to this ease of use on our behalf. After seeing this during this assignment, I'll have to see if I can find a way to incorporate other SPI hardware into future projects.

On another note, I own the I2C mini version of the SPI Driver used in the slide show and even though I haven't had many use cases for it yet, it is extremely handy to have the ability to use that direct interface along with C and Python APIs to automate troubleshooting or hardware testing in addition to the functionality you would get from using a logic analyzer.