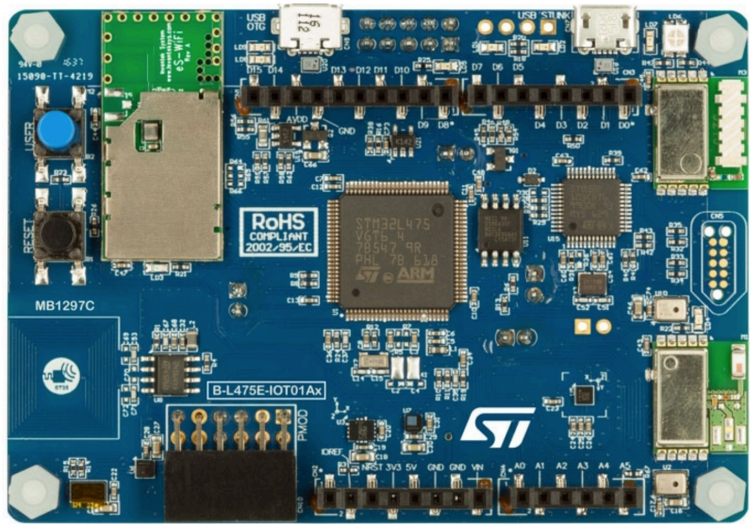
Embedded Systems
Hardware Interfacing
SPI

Norman McEntire



Contents

- Concepts
- Data Sheet - STM32L475
- User Manual - UM2153 - STM32L Discovery Kit for IoT
- Schematics - STM32L Discovery Kit for IoT
- API - STM32L HAL and BSP
    - Data Structures
    - Functions
- Hands-On Project

References

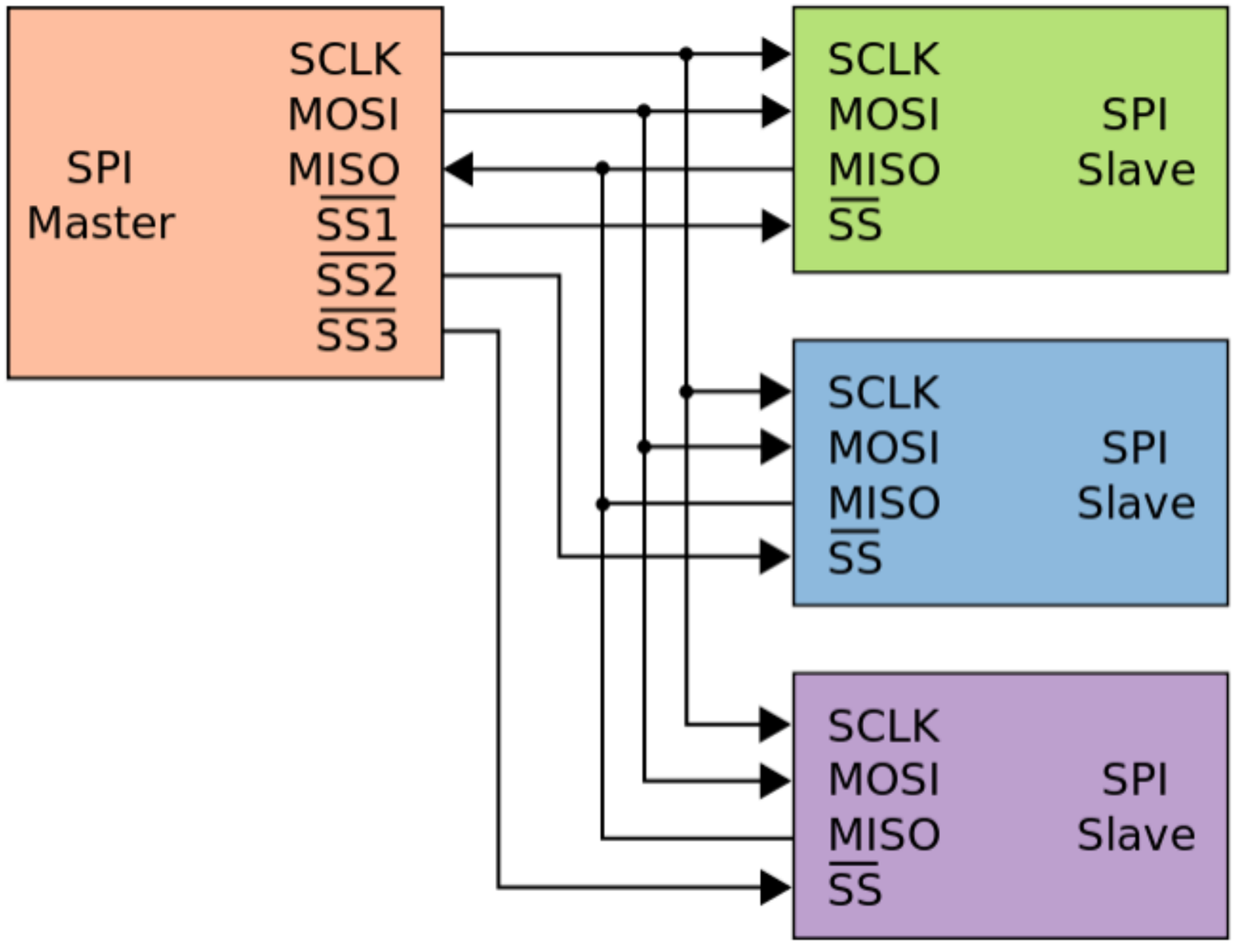
- https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
- https://en.wikipedia.org/wiki/Galvanic_isolation
- https://www.st.com/resource/en/schematic_pack/b-l475e-iot01ax_sch.zip
- https://www.st.com/resource/en/user_manual/dm00347848-discovery-kit-for-iot-node-multichannel-communication-with-stm32l4-stmicroelectronics.pdf
- https://www.st.com/resource/en/datasheet/stm32l475vg.pdf

SPI

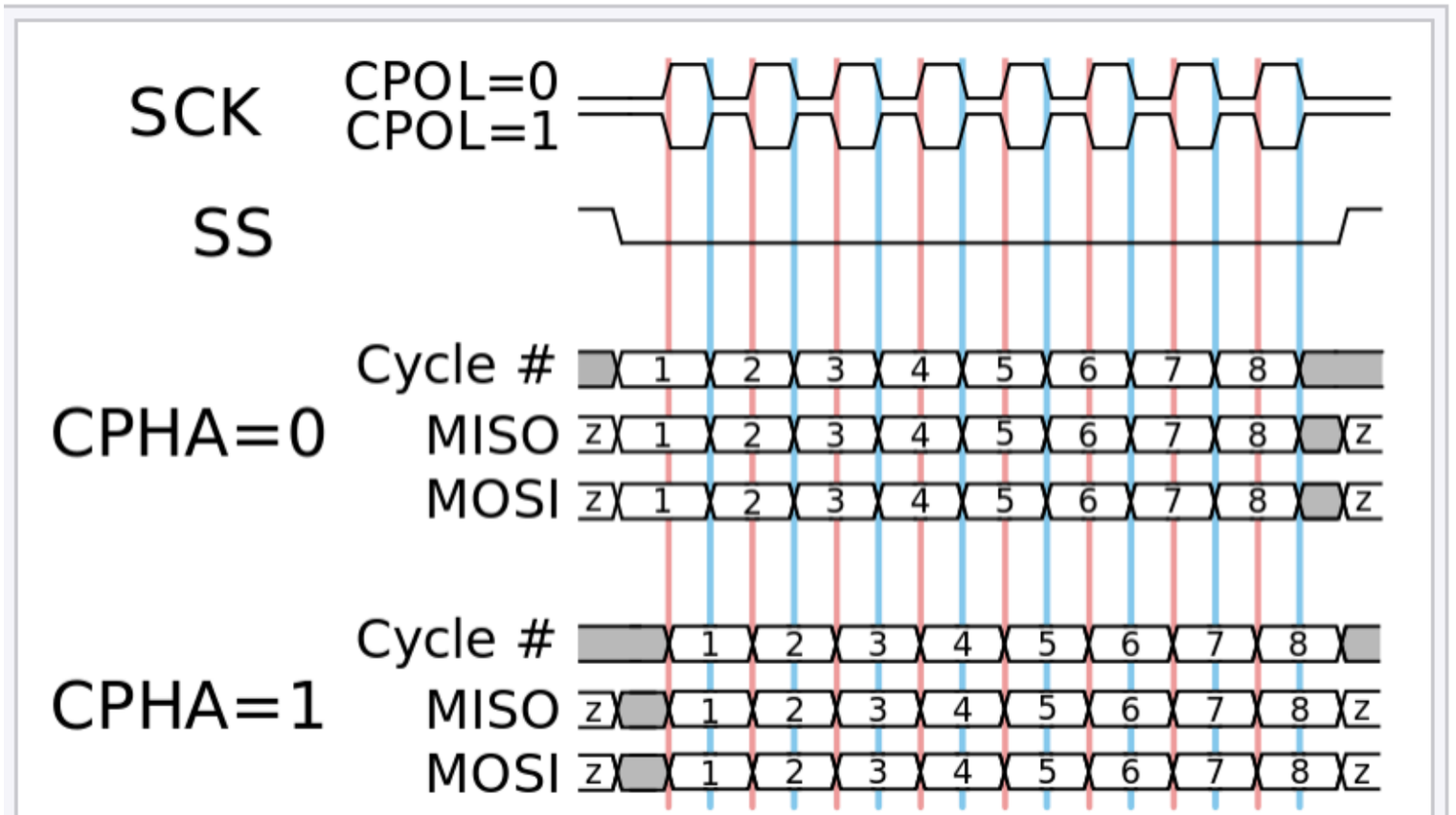
SPI Concepts - Part 1

- Nearly all SOCs include one or more SPI Ports
- SPI - Serial Peripheral Interface
- **Four** Wire Interface (recall I2C was 2 wire, SCL, SDA)
    - SCLK - Serial Clock
    - MOSI - Master Out, Slave In
    - MISO - Master In, Slave Out
    - SS - Slave Select (sometimes called Chip Select)
- **Full** Duplex (recall I2C was half-duplex)

SPI Diagram



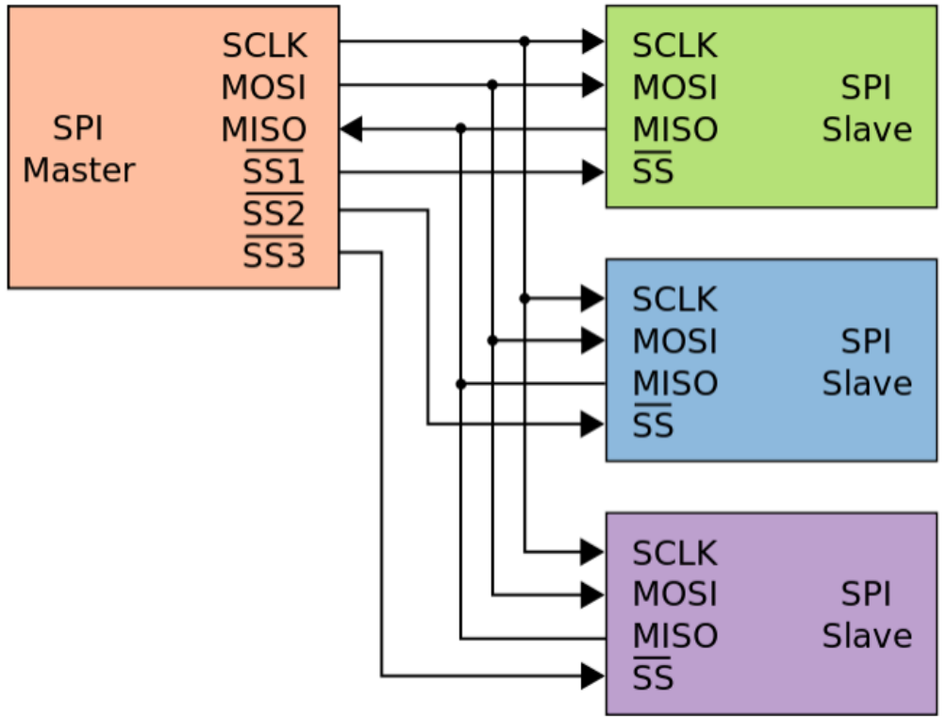
SPI Concepts
Timing Diagram



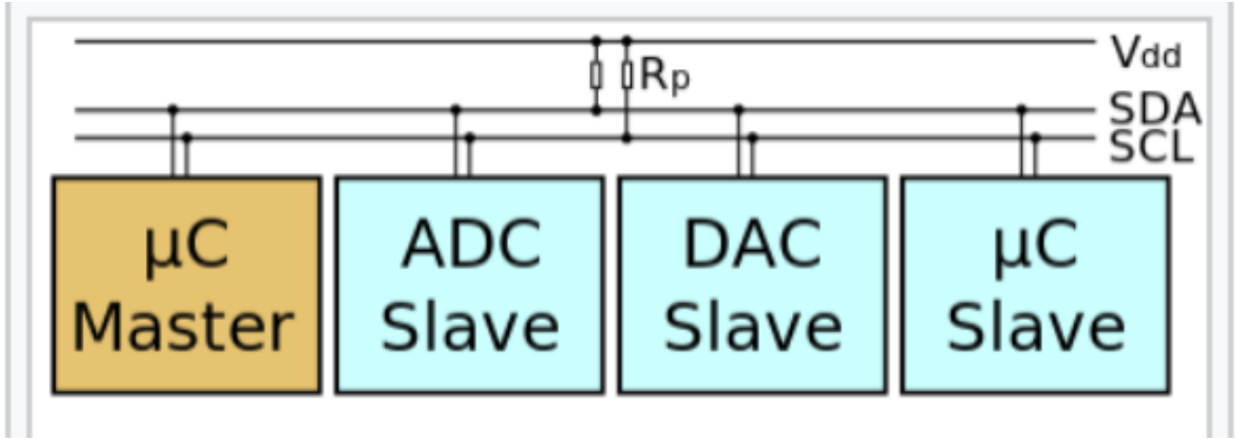
A timing diagram showing clock polarity and phase. Red lines denote clock leading edges, and blue lines, trailing edges.

SPI Compared to I2C Diagrams

SPI
4-Wire
Full Duplex
Slave Select

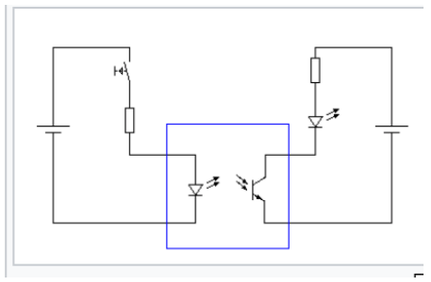


I2C
2-Wire
Half Duplex
7-bit address

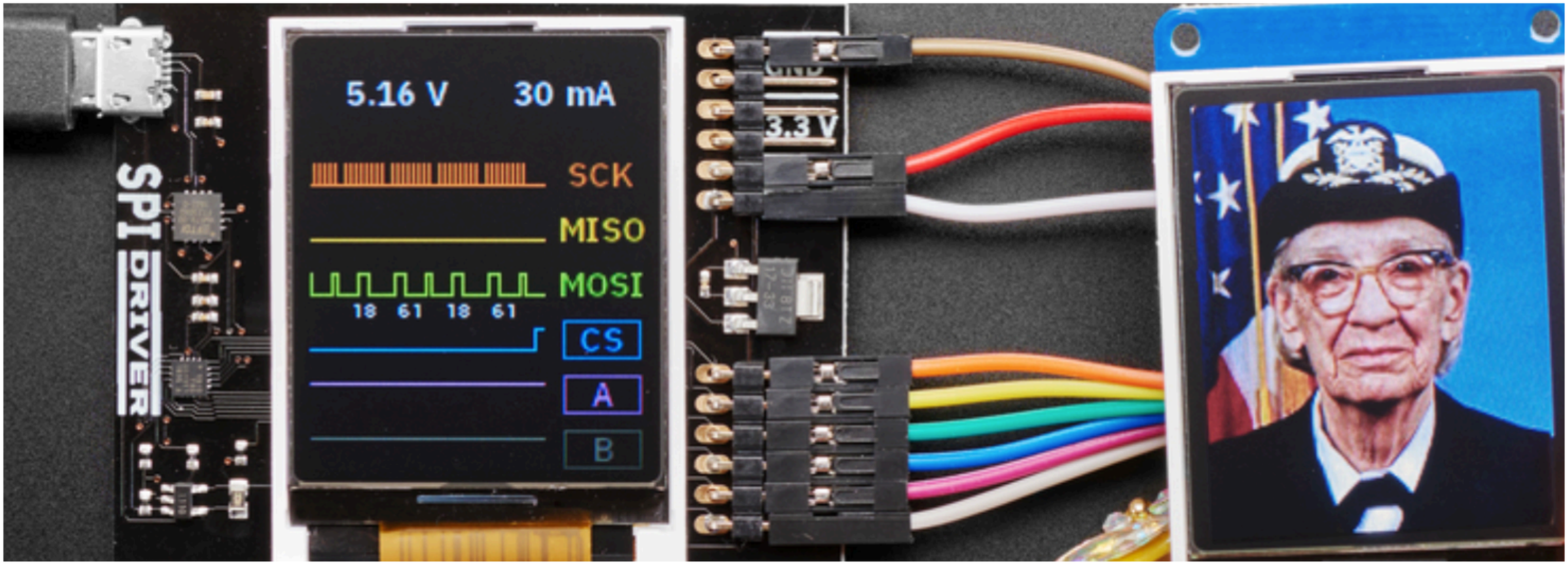


SPI Compared to I2C

- SPI is Full Duplex - I2C is Half Duplex
- SPI is 4 Wire - I2C is 2 wire
- SPI uses Push-Pull Drivers - I2C uses open drain
    - So SPI can be driven faster
- SPI uses SS (Slave Select) - I2C uses addressing
- SPI signals are one-way - I2C bidirectional
    - Easier Galvanic Isolation (isolation of current flow)
    - NOTE: I have destroyed systems due to NOT having galvanic isolation!



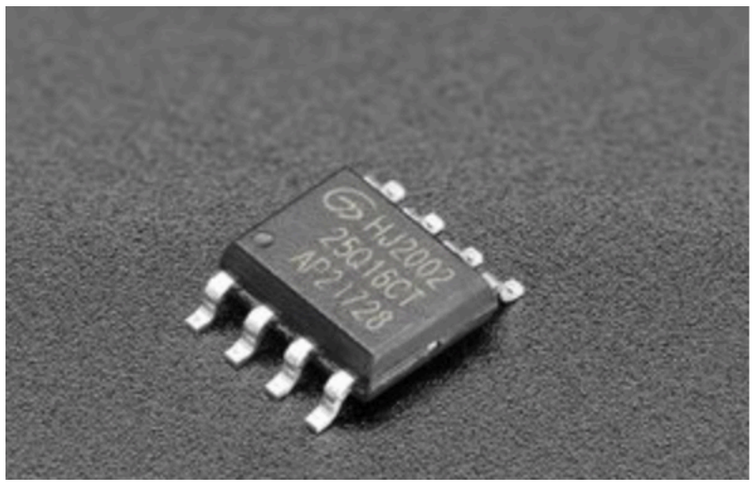
SPI Example
SPI Driver
https://www.adafruit.com/product/4268



A Live SPI Logic Analyzer!

Grace Hopper of COBOL Fame!

SPI Example
2MB SPI Flash
https://www.adafruit.com/product/4763



GD25Q16 - 2MB SPI Flash in 8-Pin SOIC package

PRODUCT ID: 4763

These little chips are like miniature SSD drives for your electronics. When you don't need something with as much storage as a micro SD card, but an EEPROM is too small, SPI (or QSPI) Flash chips give you on-the-order-of megabytes, with little cost and complexity. We use these chips all the...

SPI Example
MX25R6435F (on Discovery Kit)

MACRONIX
INTERNATIONAL CO., LTD.

MX25R6435F
(J Grade)

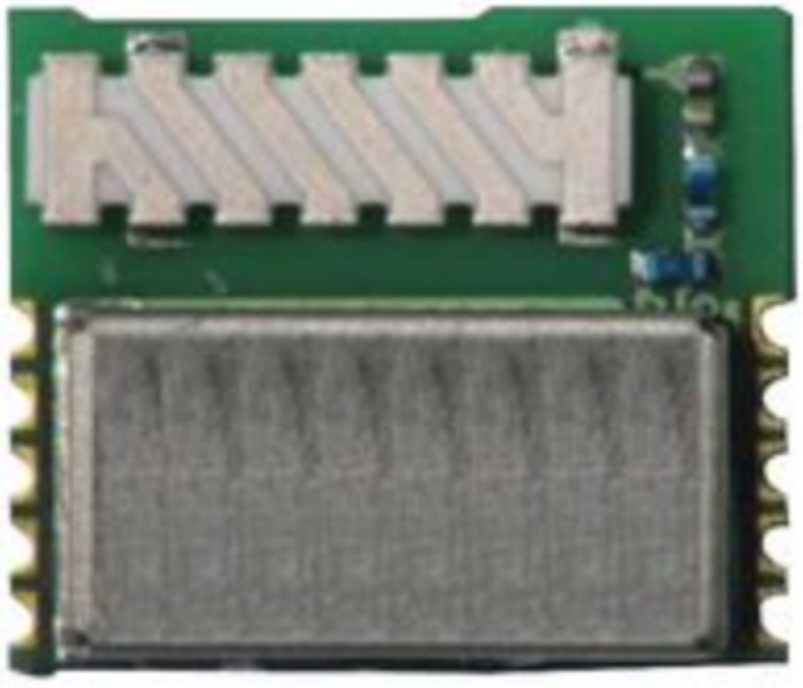
Ultra Low Power 64M-BIT [x 1/x 2/x 4] CMOS MXSMIO® (SERIAL MULTI I/O) FLASH MEMORY

HARDWARE FEATURES

- SCLK Input
  - Serial clock input
- SI/SIO0
  - Serial Data Input or Serial Data Input/Output for 2 x I/O read mode and 4 x I/O read mode
- SO/SIO1
  - Serial Data Output or Serial Data Input/Output for 2 x I/O read mode and 4 x I/O read mode
- WP#/SIO2
  - Hardware Write Protection or Serial Data Input/ Output for 4 x I/O read mode
- RESET#/SIO3 * or HOLD#/SIO3 *
  - Hardware Reset pin or Serial Data Input/Output for 4 x I/O read mode

or

SPI Example
SPSGRF-915 (On Discovery Kit)



SPSGRF-915

Meter Reading Module, 2FSK, GFSK, MSK, GMSK, OOK, ASK, 500Kbps, 928MHz, -118dBm, 1.8V to 3.6V, SPI

SPI Example
SPSGRF-915 (On Discovery Kit)

Product Overview

The SPSGRF-915 from STMicroelectronics is a sub-GHz 915MHz low power programmable RF transceiver module. The SPSGRF-915 is easy to use, low power sub-GHz module based on the SPIRIT1 RF transceiver, operating respectively in the 915MHz ISM band. The module provides a complete RF platform in a tiny form factor. The SPSGRF series enables wireless connectivity in electronic devices, requiring no RF experience or expertise for integration into the final product. As an FCC, IC and CE certified solution, the SPSGRF series optimizes the time to market of end applications. The SPSGRF-915 is an FCC certified module (FCC ID:S9NSPSGRF) and IC certified (IC 8976CSPSGRF). The module is designed for maximum performance in minimal space and includes 4 programmable I/O pins and SPI serial interfaces.

- Based on sub-1GHz SPIRIT1 transceiver and BALF-SPI-01D3 integrated balun
- 2-FSK, GFSK, MSK, GMSK, OOK, ASK modulation schemes
- Air data rate from 1Kbps to 500Kbps, on-board antenna
- Operating temperature range from -40°C to 85°C
- Receiver sensitivity is -118dBm
- Programmable RF output power up to +11.6dBm
- SPI host interface
- Up to 32 programmable I/O functions on 4 GPIO programmable module pins

Applications

RF Communications, Metering, Building Automation, Industrial, Wireless, Security

sub-GHz - 915 Mhz ISM
(ISM = Industrial Scientific Medical)
“Module provides a complete RF platform”
FCC, IC, CE Certified Module

Data Sheet

STM32l475

STM32L475 Data Sheet

STM32L475xx

Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, analog, audio

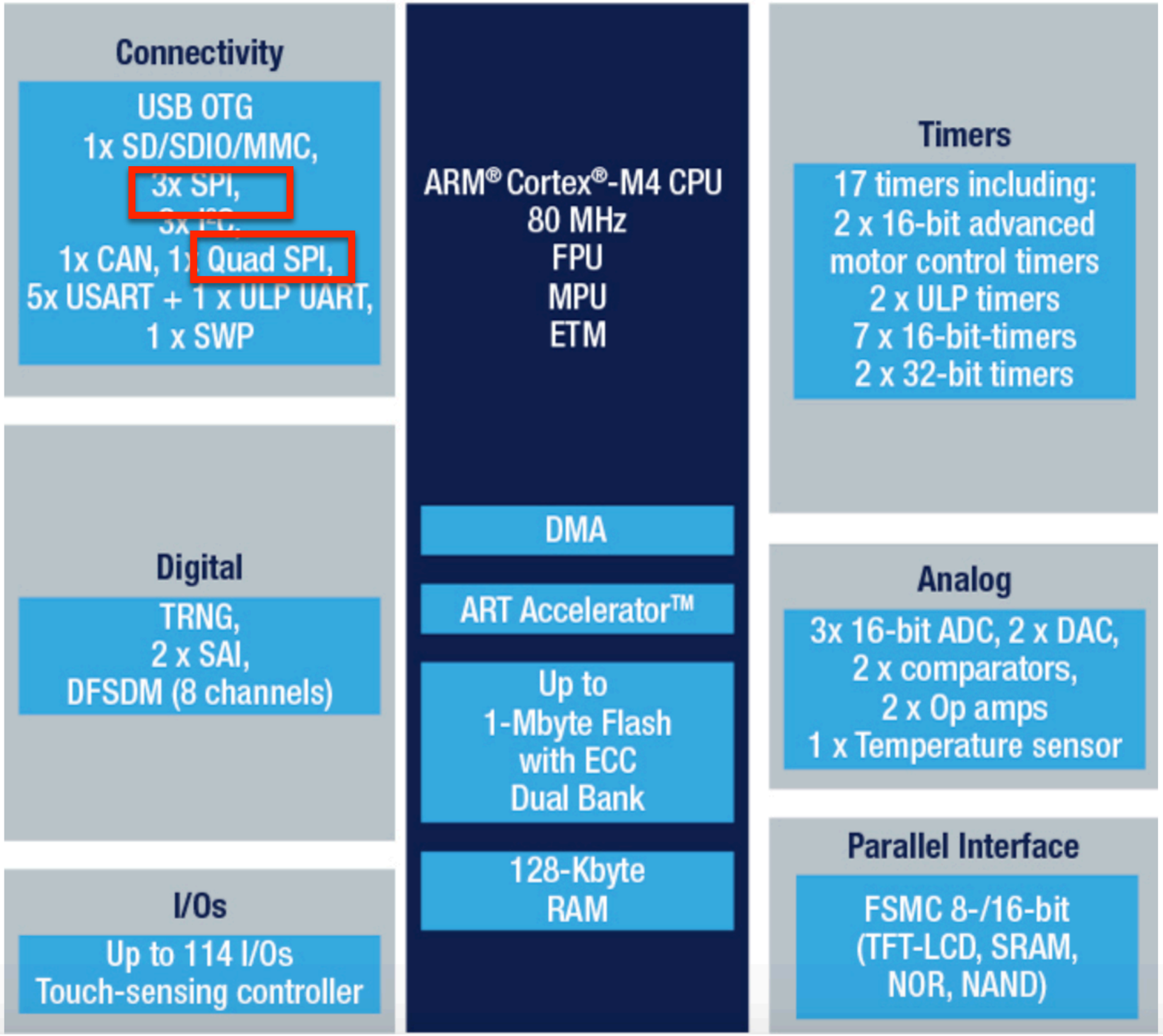
Datasheet - production data

- 20x communication interfaces
  - USB OTG 2.0 full-speed, LPM and BCD
  - 2x SAIs (serial audio interface)
  - 3x I2C FM+(1 Mbit/s), SMBus/PMBus
  - 5x USARTs (ISO 7816, LIN, IrDA, modem)
  - 1x LPUART (Stop 2 wake-up)
  - 3x SPIs (and 1x Quad SPI)
  - CAN (2.0B Active) and SDMMC interface
  - SWPMI single wire protocol master I/F
  - IRTIM (Infrared interface)

USART - Universal Serial/Async Receiver/Transmitter

LPUART - Low PowerUniversal Async Receiver/Transmitter

STM32L475



STM32L475 Data Sheet

STM32L475xx

life.augmented

Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, up to 1MB Flash, 128 KB SRAM, USB OTG FS, analog, audio

Datasheet - production data

3.28 Serial peripheral interface (SPI)

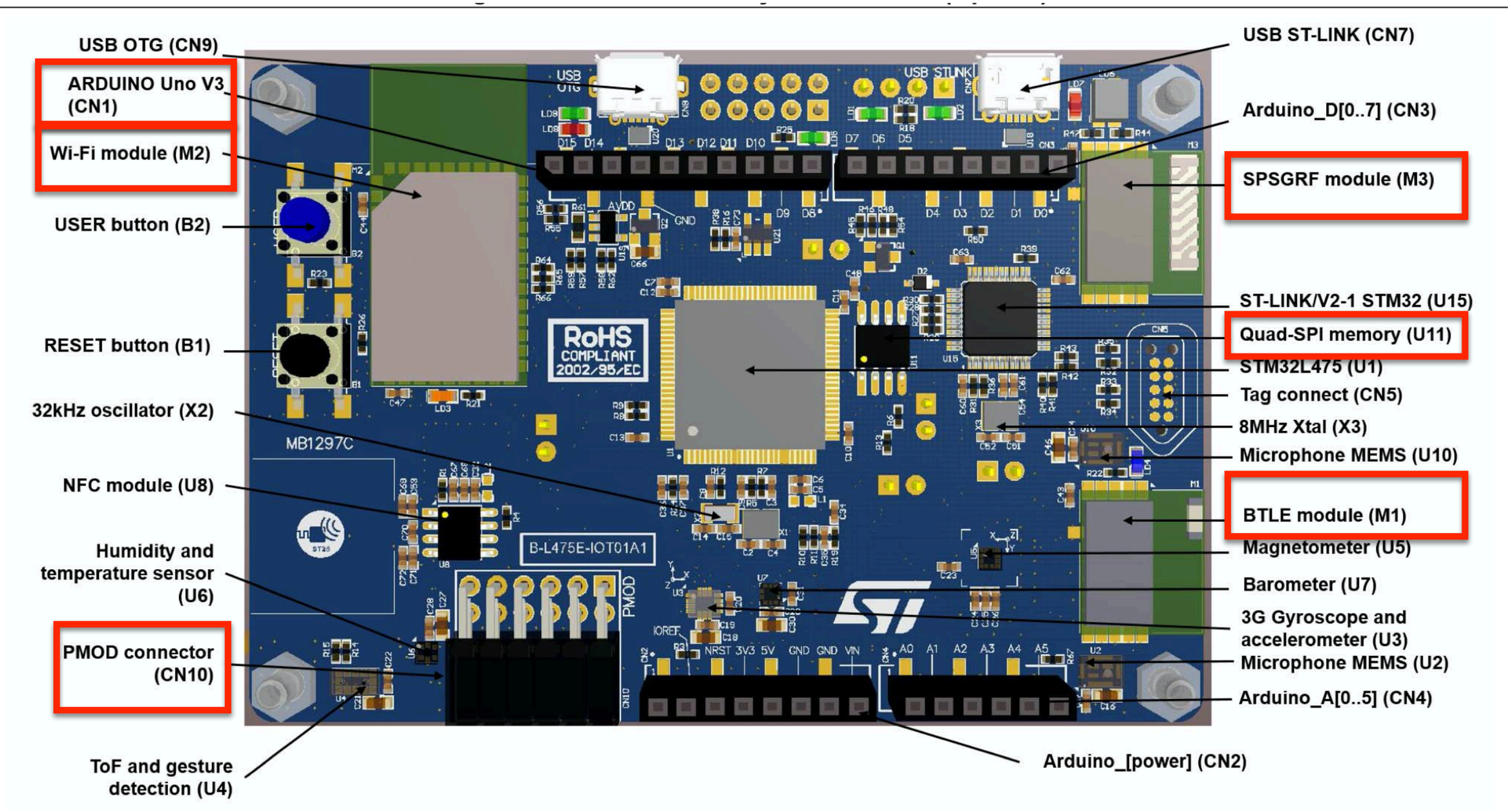
Three SPI interfaces allow communication up to 40 Mbits/s in master and up to 24 Mbits/s slave modes, in half-duplex, full-duplex and simplex modes. The 3-bit prescaler gives 8 master mode frequencies and the frame size is configurable from 4 bits to 16 bits. The SPI interfaces support NSS pulse mode, TI mode and Hardware CRC calculation.

All SPI interfaces can be served by the DMA controller.

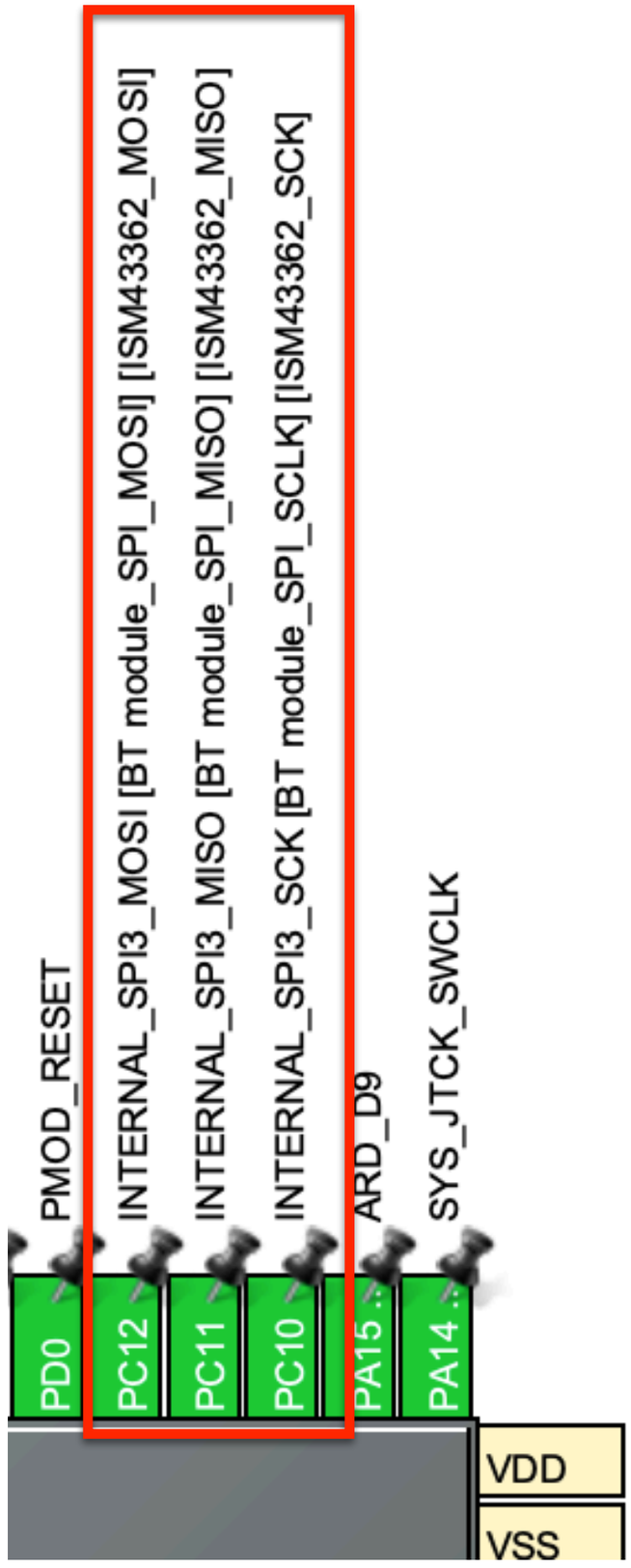
User Manual

STM32L Discovery Kit IoT Node SPI

Use of SPI



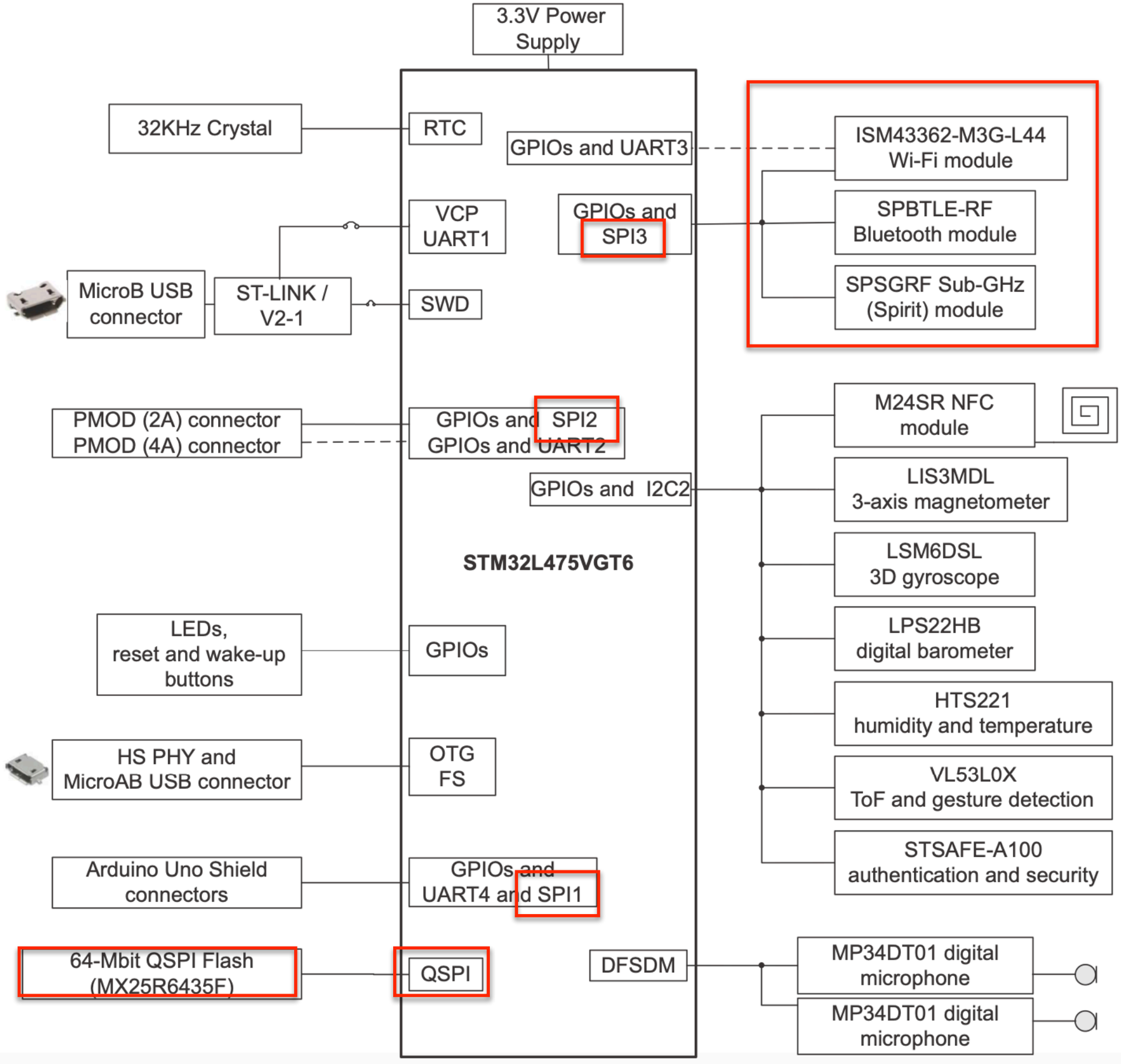
SPI3 - RF Internal



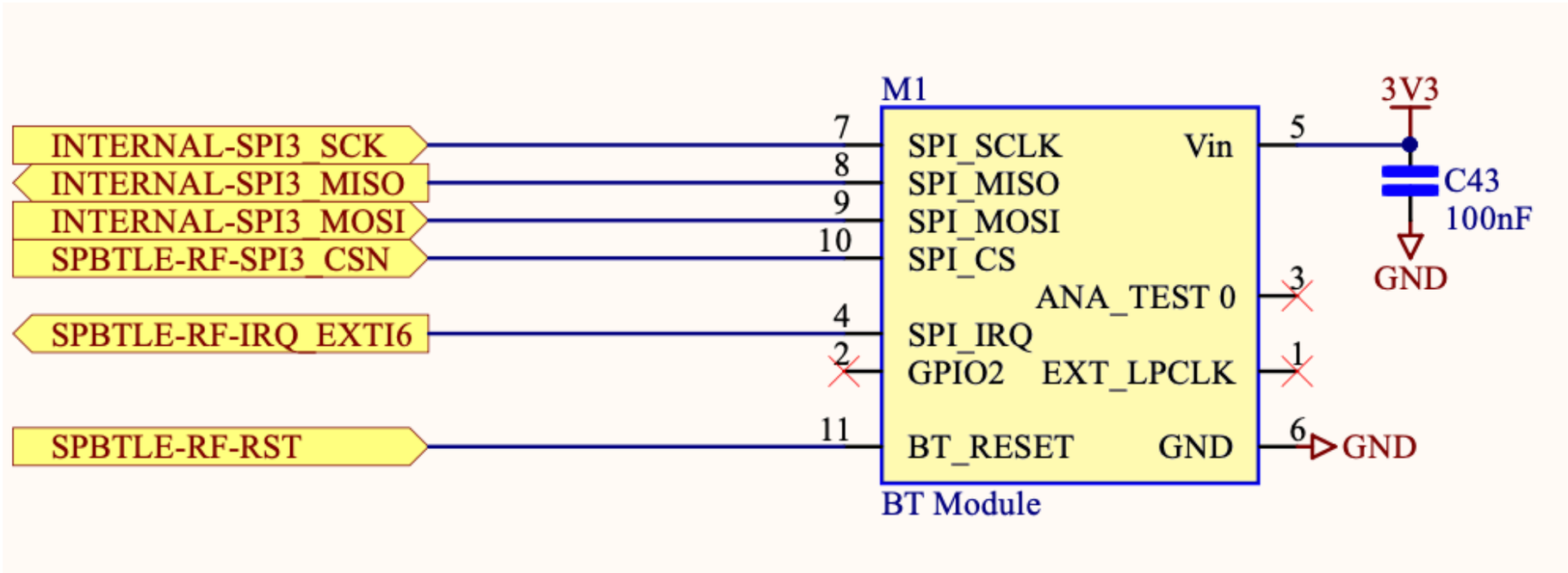
QuadSPI - Internal

Schematics

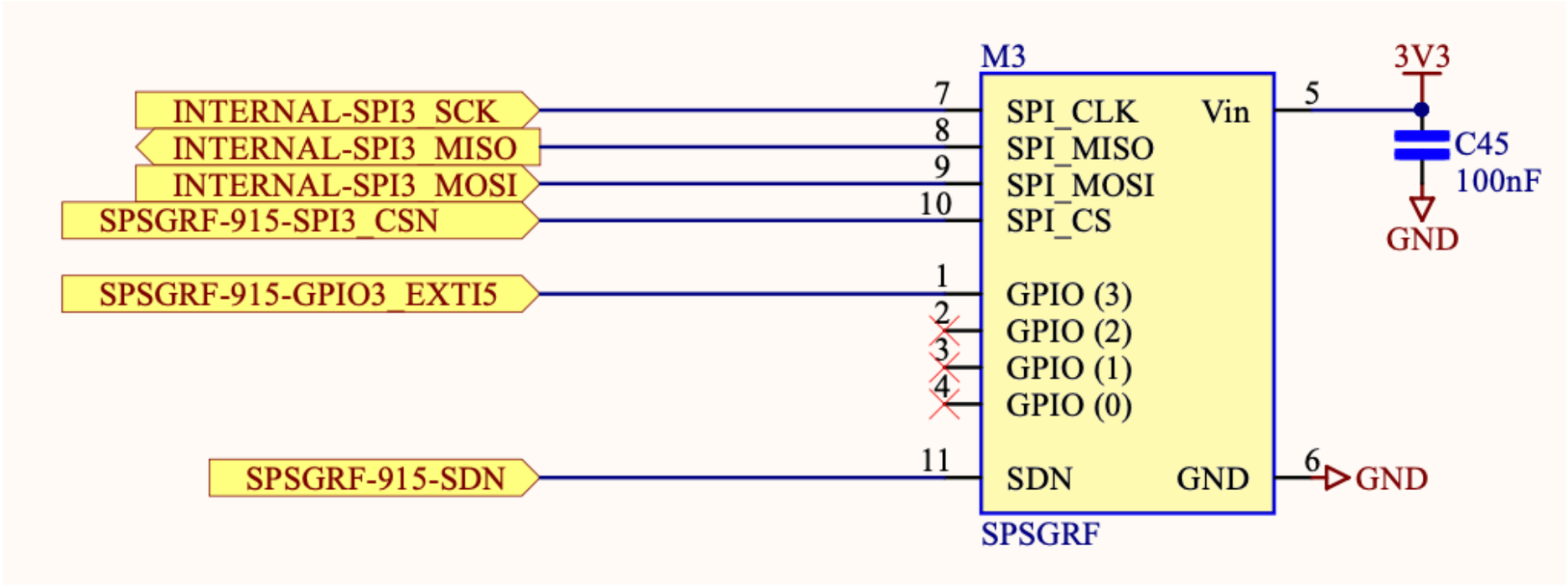
STM324L Discovery Kit IoT Node



STM32L Discovery Kit Bluetooth Module - SPI3

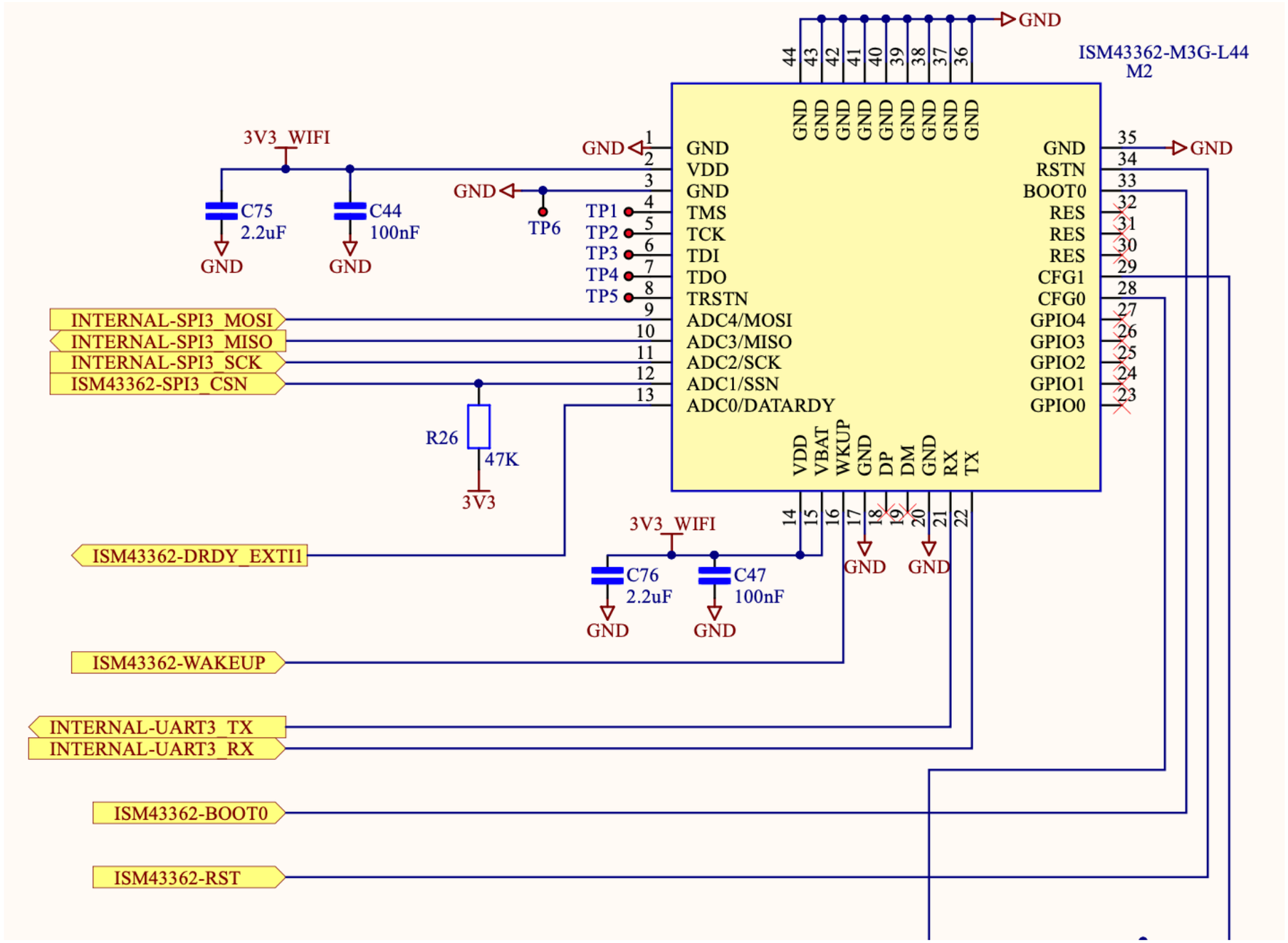


STM32L Discovery Kit
SPSGRF - 915Mhz - SPI



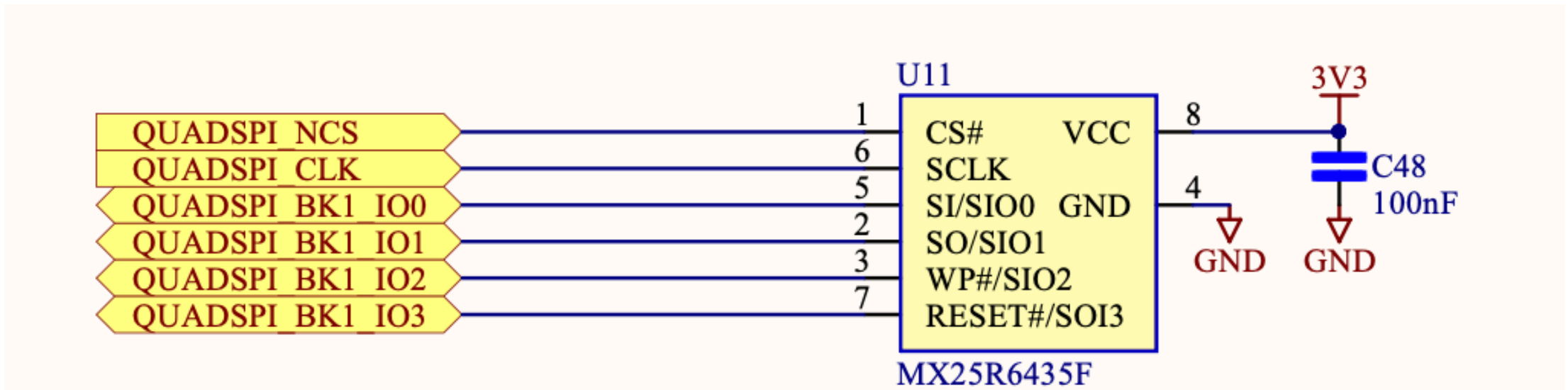
SPSGRF-868 and SPSGRF-915 modules

STM32L Discovery Kit
ISM43362-M3G-L44 - Wifi - SPI



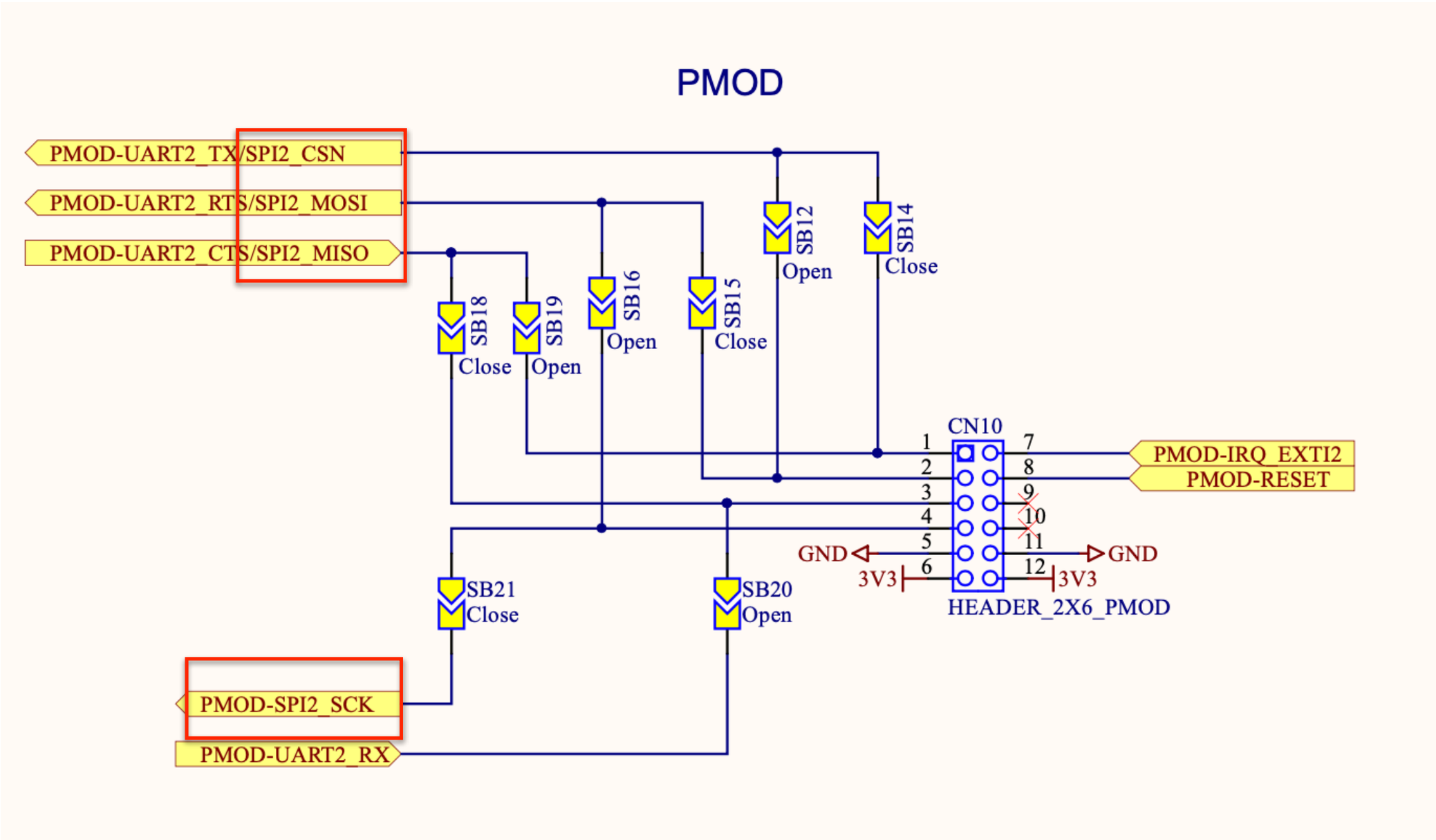
STM32L Discovery Kit
MX25R6435F - QSPI

Ultra Low Power 64M-BIT [x 1/x 2/x 4] CMOS MXSMIO® (SERIAL MULTI I/O) FLASH MEMORY

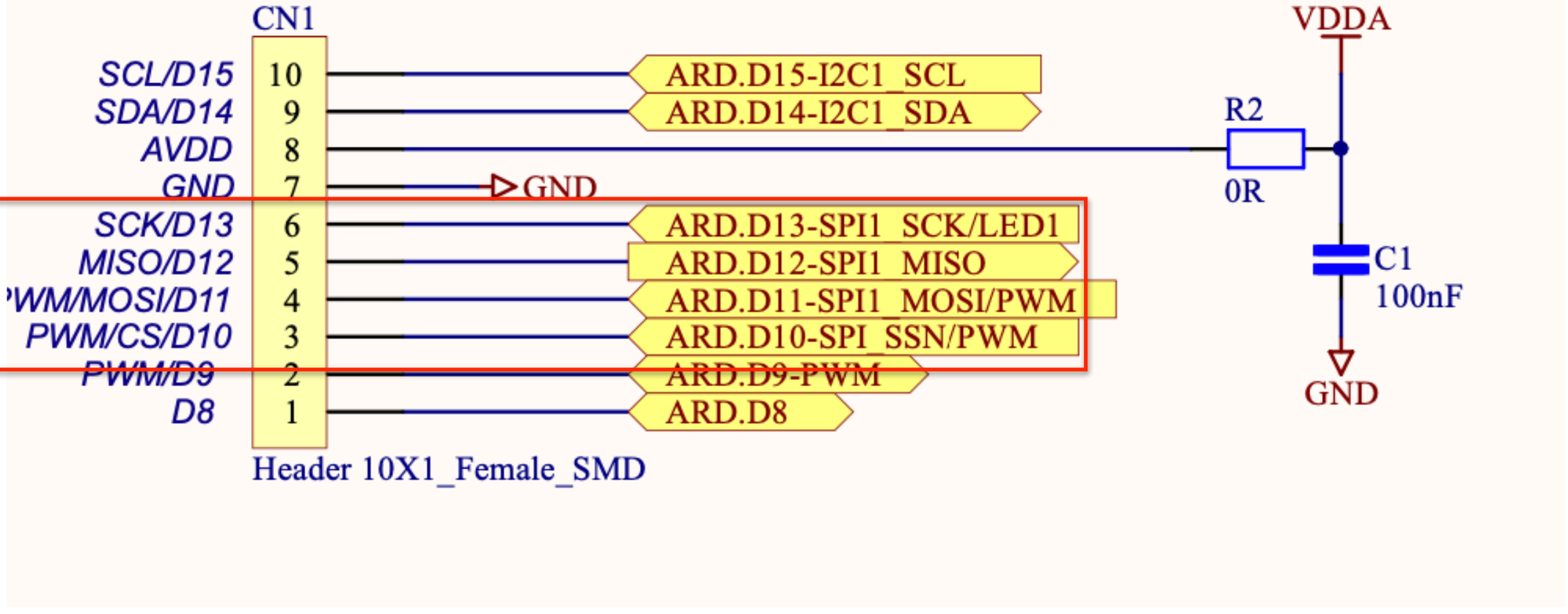


Note use of 4 Data
SIO0, 1, 2, 3

Use of SPI
PMOD Connector



Use of SPI
Arduino Connector



HAL SPI

Data Structures

Default Project Has SPI

SPI_InitTypeDef

```
44  /**
45    * @brief  SPI Configuration Structure definition
46    */
47  typedef struct
48  {
49    uint32_t Mode;                /*!< Specifies the SPI operating mode.
50                                       This parameter can be a value of @ref SPI_Mode */
51
52    uint32_t Direction;           /*!< Specifies the SPI bidirectional mode state.
53                                       This parameter can be a value of @ref SPI_Direction */
54
55    uint32_t DataSize;            /*!< Specifies the SPI data size.
56                                       This parameter can be a value of @ref SPI_Data_Size */
57
58    uint32_t CLKPolarity;         /*!< Specifies the serial clock steady state.
59                                       This parameter can be a value of @ref SPI_Clock_Polarity */
60
61    uint32_t CLKPhase;            /*!< Specifies the clock active edge for the bit capture.
62                                       This parameter can be a value of @ref SPI_Clock_Phase */
63
64    uint32_t NSS;                 /*!< Specifies whether the NSS signal is managed by
65                                       hardware (NSS pin) or by software using the SSI bit.
66                                       This parameter can be a value of @ref SPI_Slave_Select_management */
```

HAL_SPI_StateTypeDef

```
 98 /**
 99   * @brief  HAL SPI State structure definition
100   */
101 typedef enum
102 {
103   HAL_SPI_STATE_RESET      = 0x00U,    /*!< Peripheral not Initialized                         */
104   HAL_SPI_STATE_READY      = 0x01U,    /*!< Peripheral Initialized and ready for use           */
105   HAL_SPI_STATE_BUSY       = 0x02U,    /*!< an internal process is ongoing                     */
106   HAL_SPI_STATE_BUSY_TX    = 0x03U,    /*!< Data Transmission process is ongoing               */
107   HAL_SPI_STATE_BUSY_RX    = 0x04U,    /*!< Data Reception process is ongoing                  */
108   HAL_SPI_STATE_BUSY_TX_RX = 0x05U,    /*!< Data Transmission and Reception process is ongoing */
109   HAL_SPI_STATE_ERROR      = 0x06U,    /*!< SPI error state                                    */
110   HAL_SPI_STATE_ABORT      = 0x07U     /*!< SPI abort is ongoing                               */
111 } HAL_SPI_StateTypeDef;
112
```

HAL_SPI_HandleTypeDef

```
113 /**
114   * @brief  SPI handle Structure definition
115   */
116 typedef struct __SPI_HandleTypeDef
117 {
118   SPI_TypeDef                *Instance;      /*!< SPI registers base address               */
119
120   SPI_InitTypeDef            Init;          /*!< SPI communication parameters             */
121
122   uint8_t                    *pTxBuffPtr;    /*!< Pointer to SPI Tx transfer Buffer        */
123
124   uint16_t                   TxXferSize;    /*!< SPI Tx Transfer size                     */
125
126   __IO uint16_t              TxXferCount;   /*!< SPI Tx Transfer Counter                  */
127
128   uint8_t                    *pRxBuffPtr;    /*!< Pointer to SPI Rx transfer Buffer        */
129
```

HAL_SPI_ERROR_...

```
198
199  /** @defgroup SPI_Error_Code SPI Error Code
200    * @{
201    */
202  #define HAL_SPI_ERROR_NONE              (0x00000000U)   /*!< No error                                    */
203  #define HAL_SPI_ERROR_MODF              (0x00000001U)   /*!< MODF error                                  */
204  #define HAL_SPI_ERROR_CRC               (0x00000002U)   /*!< CRC error                                   */
205  #define HAL_SPI_ERROR_OVR               (0x00000004U)   /*!< OVR error                                   */
206  #define HAL_SPI_ERROR_FRE               (0x00000008U)   /*!< FRE error                                   */
207  #define HAL_SPI_ERROR_DMA               (0x00000010U)   /*!< DMA transfer error                          */
208  #define HAL_SPI_ERROR_FLAG              (0x00000020U)   /*!< Error on RXNE/TXE/BSY/FTLVL/FRLVL Flag */
209  #define HAL_SPI_ERROR_ABORT             (0x00000040U)   /*!< Error during SPI Abort procedure       */
210  #if (USE_HAL_SPI_REGISTER_CALLBACKS == 1U)
211  #define HAL_SPI_ERROR_INVALID_CALLBACK  (0x00000080U)   /*!< Invalid Callback error                 */
212  #endif /* USE_HAL_SPI_REGISTER_CALLBACKS */
213  /**
```

SPI API

Functions

SPI Init

```
769
770 /** @addtogroup SPI_Exported_Functions_Group1
771   * @{
772   */
773 /* Initialization/de-initialization functions  **********************
774 HAL_StatusTypeDef HAL_SPI_Init(SPI_HandleTypeDef *hspi);
775 HAL_StatusTypeDef HAL_SPI_DeInit(SPI_HandleTypeDef *hspi);
776 void HAL_SPI_MspInit(SPI_HandleTypeDef *hspi);
777 void HAL_SPI_MspDeInit(SPI_HandleTypeDef *hspi);
778
```

SPI Polling Mode

```
790   */
791  /* I/O operation functions  ***************************************************/
792  HAL_StatusTypeDef HAL_SPI_Transmit(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size, uint32_t Timeout)
793  HAL_StatusTypeDef HAL_SPI_Receive(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size, uint32_t Timeout);
794  HAL_StatusTypeDef HAL_SPI_TransmitReceive(SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData, uint1
795                                            uint32_t Timeout);
```

SPI Interrupt Mode

```
795                                                uint32_t Timeout);
796 HAL_StatusTypeDef HAL_SPI_Transmit_IT(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size);
797 HAL_StatusTypeDef HAL_SPI_Receive_IT(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size);
798 HAL_StatusTypeDef HAL_SPI_TransmitReceive_IT(SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData,
799                                              uint16_t Size);
```

SPI DMA Mode

```
799                                                uint16_t Size);
800  HAL_StatusTypeDef HAL_SPI_Transmit_DMA(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size);
801  HAL_StatusTypeDef HAL_SPI_Receive_DMA(SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size);
802  HAL_StatusTypeDef HAL_SPI_TransmitReceive_DMA(SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData,
803                                                uint16_t Size);
804  HAL_StatusTypeDef HAL_SPI_DMAPause(SPI_HandleTypeDef *hspi);
805  HAL_StatusTypeDef HAL_SPI_DMAResume(SPI_HandleTypeDef *hspi);
806  HAL_StatusTypeDef HAL_SPI_DMAStop(SPI_HandleTypeDef *hspi);
```

BSP_QSPI_...
Board Support Package
Quad SPI
MX25R6435f

QSPI Error Code

```
45  /* Exported constants --------------------------------------------------------
46  /** @defgroup STM32L475E_IOT01_QSPI_Exported_Constants QSPI Exported Constants
47    * @{
48    */
49  /* QSPI Error codes */
50  #define QSPI_OK            ((uint8_t)0x00)
51  #define QSPI_ERROR         ((uint8_t)0x01)
52  #define QSPI_BUSY          ((uint8_t)0x02)
53  #define QSPI_NOT_SUPPORTED ((uint8_t)0x04)
54  #define QSPI_SUSPENDED     ((uint8_t)0x08)
55
```

QSPI_Info

```
60  /* Exported types ------------------------------------------------------------*/
61  /** @defgroup STM32L475E_IOT01_QSPI_Exported_Types QSPI Exported Types
62    * @{
63    */
64  /* QSPI Info */
65  typedef struct {
66    uint32_t FlashSize;           /*!< Size of the flash */
67    uint32_t EraseSectorSize;     /*!< Size of sectors for the erase operation */
68    uint32_t EraseSectorsNumber;  /*!< Number of sectors for the erase operation */
69    uint32_t ProgPageSize;        /*!< Size of pages for the program operation */
70    uint32_t ProgPagesNumber;     /*!< Number of pages for the program operation */
71  } QSPI_Info;
72
```

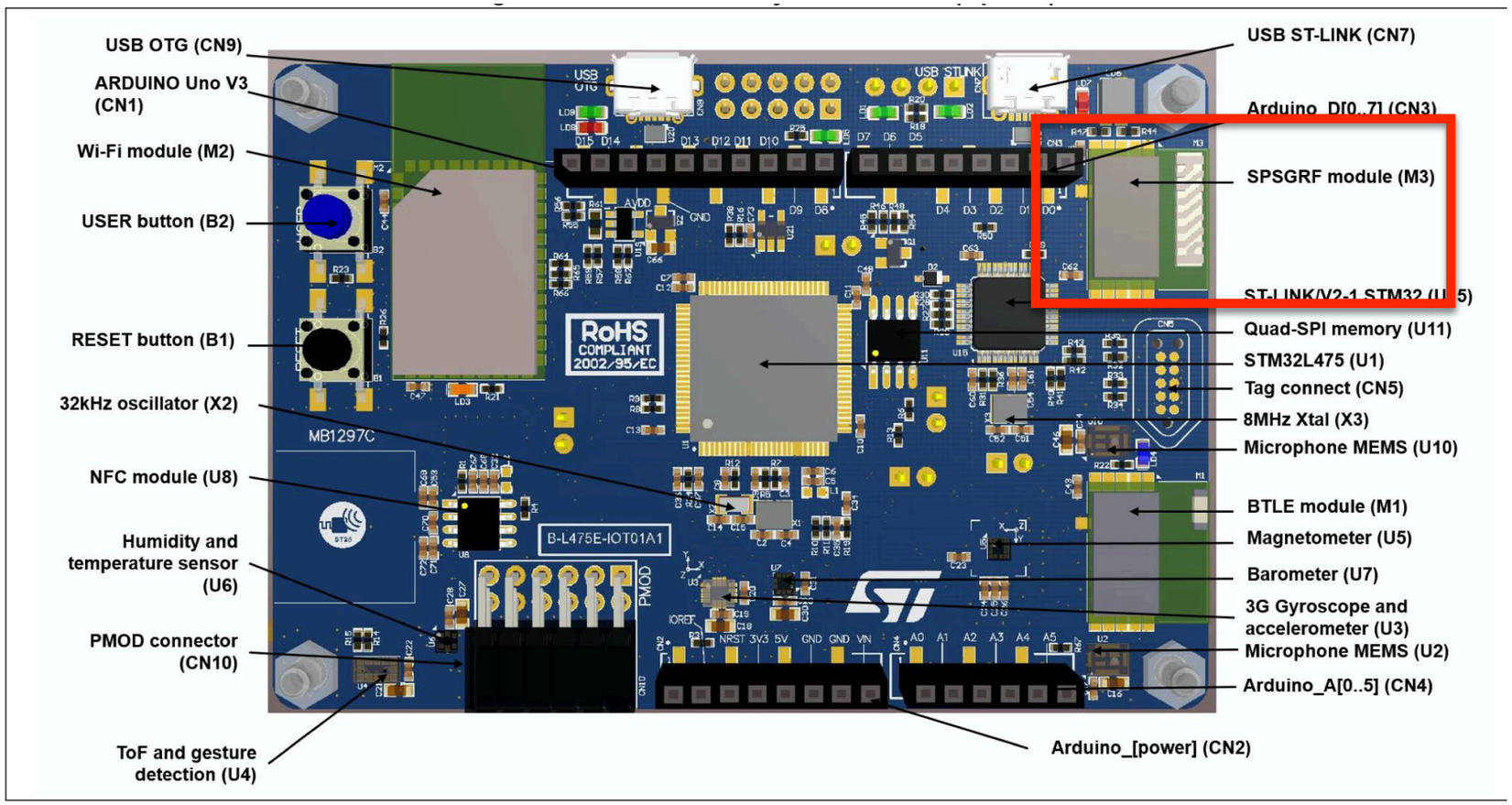
BSP_QSPI_...

```
77  /* Exported functions ---------------------------------------------------*/
78  /** @defgroup STM32L475E_IOT01_QSPI_Exported_Functions QSPI Exported Functions
79    * @{
80    */
81  uint8_t BSP_QSPI_Init                    (void);
82  uint8_t BSP_QSPI_DeInit                  (void);
83  uint8_t BSP_QSPI_Read                    (uint8_t* pData, uint32_t ReadAddr, uint32_t Size);
84  uint8_t BSP_QSPI_Write                   (uint8_t* pData, uint32_t WriteAddr, uint32_t Size);
85  uint8_t BSP_QSPI_Erase_Block             (uint32_t BlockAddress);
86  uint8_t BSP_QSPI_Erase_Sector            (uint32_t Sector);
87  uint8_t BSP_QSPI_Erase_Chip              (void);
88  uint8_t BSP_QSPI_GetStatus               (void);
89  uint8_t BSP_QSPI_GetInfo                 (QSPI_Info* pInfo);
90  uint8_t BSP_QSPI_EnableMemoryMappedMode(void);
91  uint8_t BSP_QSPI_SuspendErase            (void);
92  uint8_t BSP_QSPI_ResumeErase             (void);
93  uint8_t BSP_QSPI_EnterDeepPowerDown      (void);
94  uint8_t BSP_QSPI_LeaveDeepPowerDown      (void);
95
```

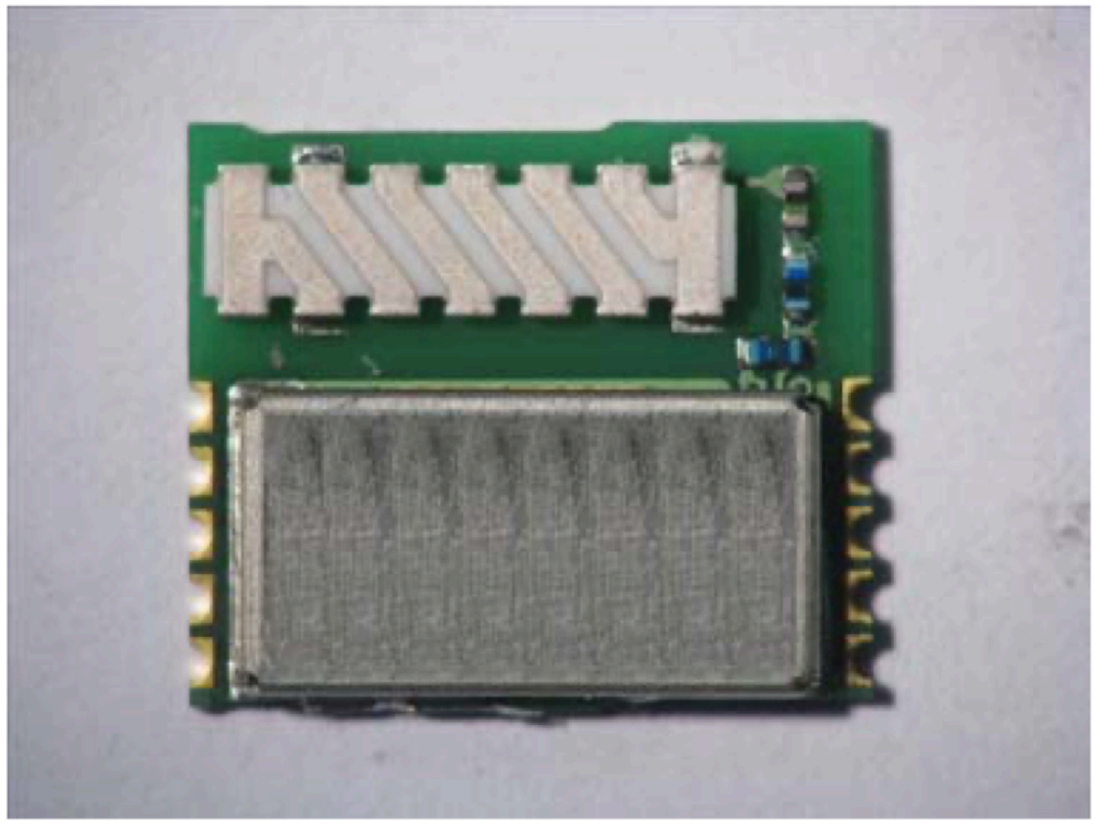
SPSGRF

Sub GHz RF Module

Sub-GHz (868 or 915 MHz) low power programmable RF transceiver modules



Overview



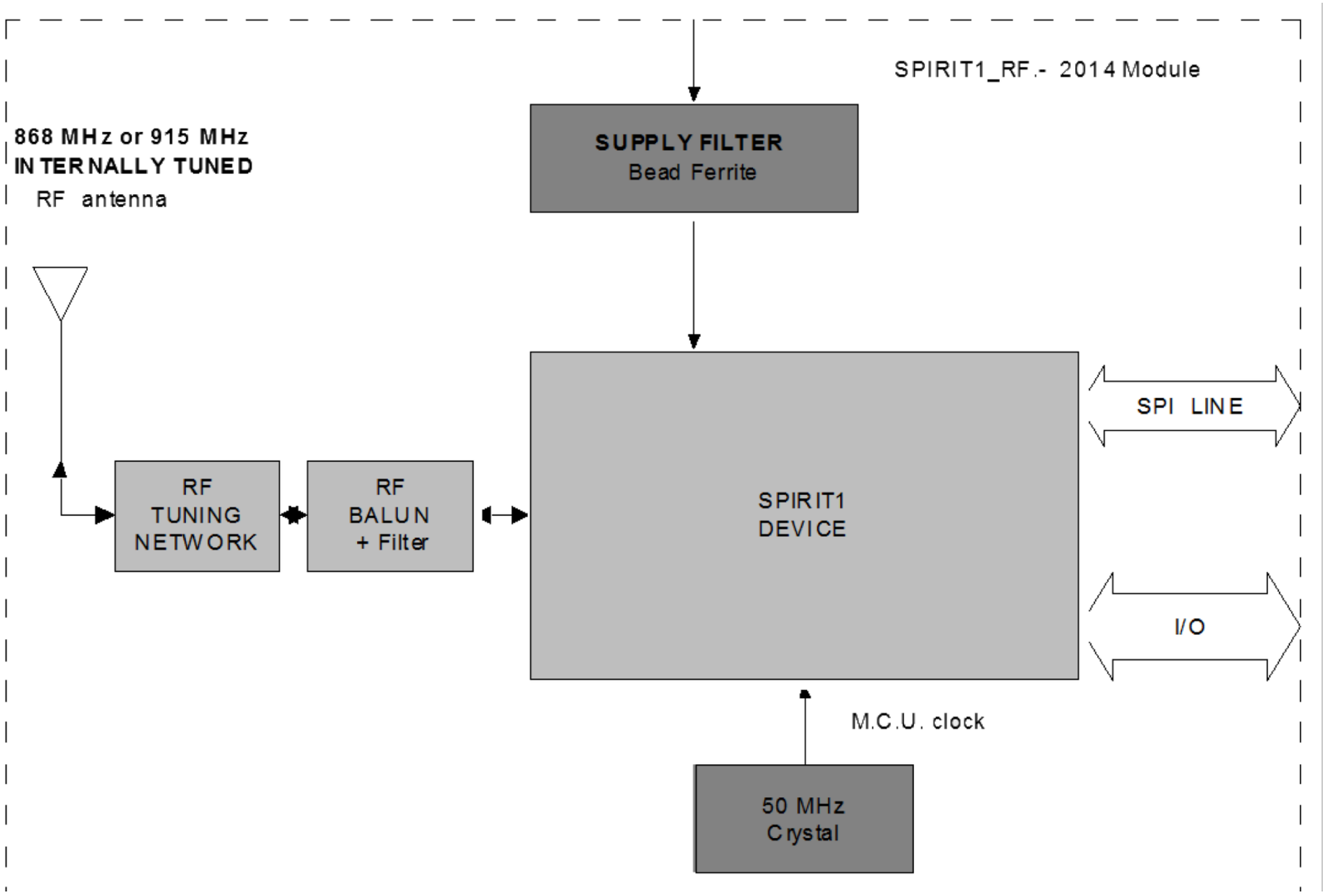
Applications

- AMR (automatic meter reading)
- Home and building automation
- WSN (wireless sensor network)
- Industrial monitoring and control
- Wireless fire and security alarm systems
- Point-to-point wireless link

Features

- Programmable radio features
  - Based on Sub-1GHz SPIRIT1 transceiver and integrated Balun (BALF-SPI-01D3)
  - Modulation schemes: 2-FSK, GFSK, MSK, GMSK, OOk, ASK
  - Air data rate from 1 to 500 kbps
  - On-board antenna
  - Operating temperature range from -40 °C to 85 °C
- RF features
  - Receiver sensitivity: -118 dBm
  - Programmable RF output power up to +11.6 dBm
- Host interface
  - SPI

Block Diagram



Pin Description

Table 9. Pin assignment

Name	Type	Pin#	Description	Alt function	V max. tolerance	Initial state
SPI interface						
SPI_CLK	I	7	SPI CLOCK (Max. 8 MHz)		V_{in}	
SPI_MISO	O	8	SPI MISO (MASTER in / SLAVE out)		V_{in}	
SPI_MOSI	I	9	SPI MOSI (MASTER out SLAVE in)		V_{in}	
SPI_CS	I	10	SPI “Chip Select” (SPI slave select)		V_{in}	
Power and ground						
V_{in}		5	V_{in}		(1.8V - 3.6V max.)	
GND		6	GND			
Module shutdown						
SDN	I	11	Shutdown input (active high)		(1.8V - 3.6V max.)	
GPIO - general purpose input/output						
GPIO [0]	I/O	4	Programmable input / output & analog temperature output		(1.8V - V_{in} max.)	Digital output. Low power
GPIO [1]	I/O	3	Programmable input / output		(1.8V - V_{in} max.)	Digital output. Low power
GPIO [2]	I/O	2	Programmable input / output		(1.8V - V_{in} max.)	Digital output. Low power
GPIO [3]	I/O	1	Programmable input / output		(1.8V - V_{in} max.)	Digital output. Low power

Hardware Design Notes

The SPSGRF module supports SPI hardware interfaces.

Note: *- All unused pins should be left floating; do not ground.*

- All GND pins must be well grounded.

- The area around the module should be free of any ground planes, power planes, trace routings, or metal for 6 mm from the module antenna position, in all directions.

- Traces should not be routed underneath the module

FCC Certification

6.1 FCC certification

The SPSGRF-915 module has been tested and found compliant with the FCC part 15 rules. These limits are designed to provide reasonable protection against harmful interference in approved installations. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications.

However, there is no guarantee that interference may not occur in a particular installation.

This device complies with part 15 of the FCC rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

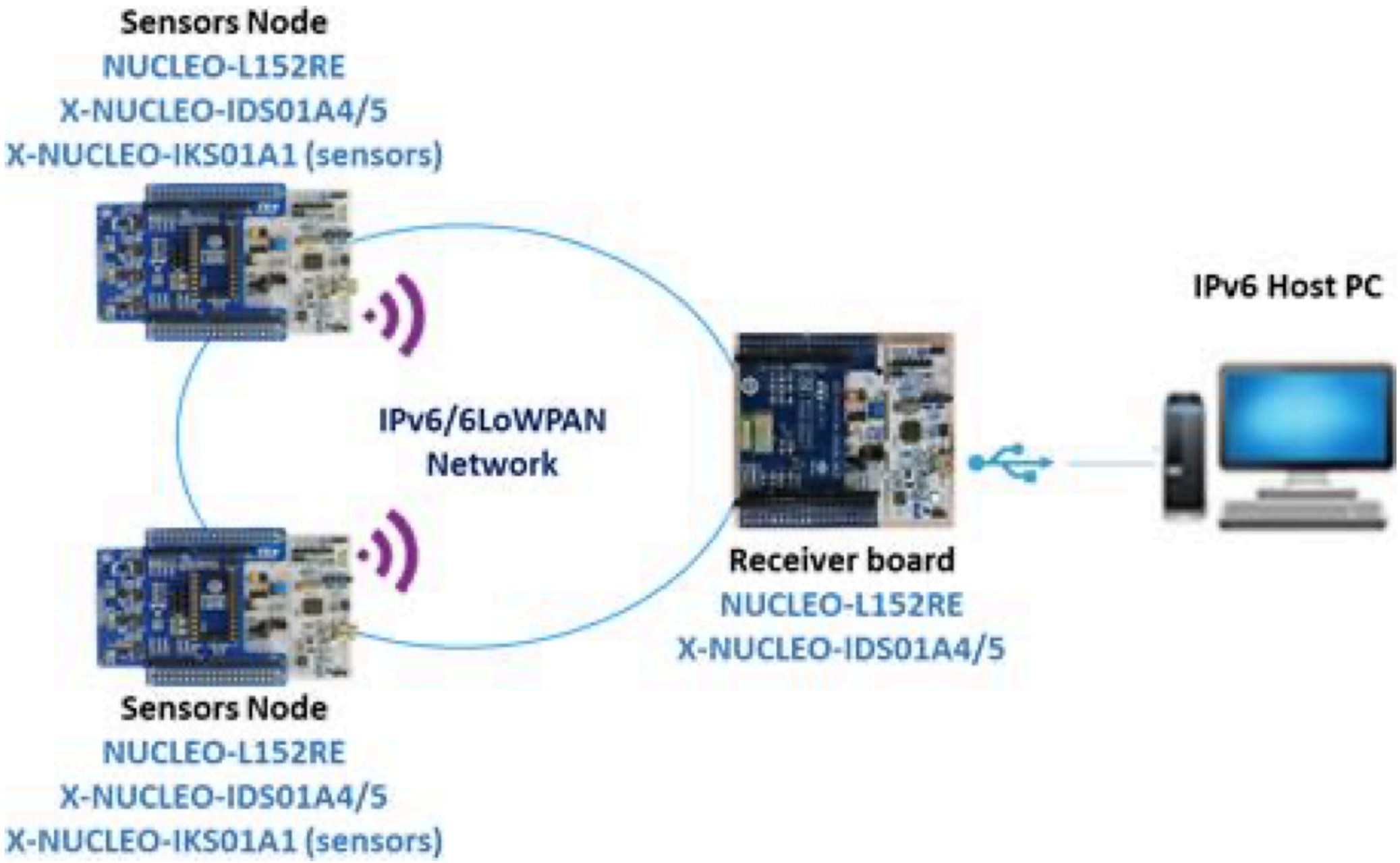
Modifications or changes to this equipment not expressly approved by STMicroelectronics may render void the user's authority to operate this equipment.

Modular approval

FCC ID: S9NSPSGRF

In accordance with FCC part 15, the SPSGRF-915 is listed as a modular transmitter device.

Example Solution



SPI

Hands-On Project

SPI Hands-On Project Overview

- The goal of this project is to give you hands-on experience with Quad SPI using the BSP_QSPI_ API
  - BSP_QSPI_Init()
  - BSP_QSPI_GetStatus()
  - BSP_QSPI_GetInfo(QSPI_Info *pInfo)
- To confirm your experience, you will create a **PDF document** that you will submit for grading
  - The PDF document will capture the major steps you perform to complete this project
  - See example PDF posted with assignment for example PDF format

SPI Hands-On Project User Stories

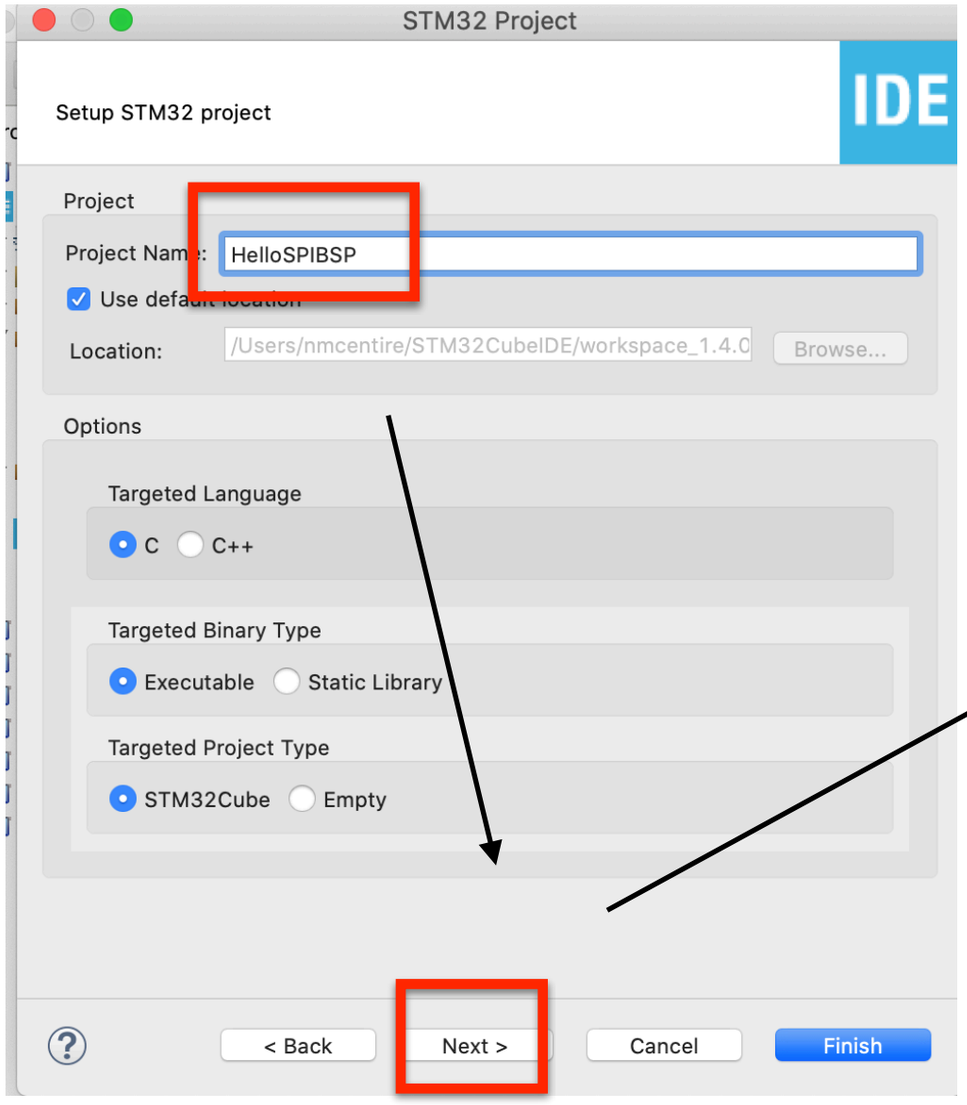
- **User Story 1** - Use HAL Polling API to send command prompt to UART1. Read single number command from console which is User Story number. Command codes are:
  - 1 = BSP_QSPI_GetStatus, 2 = BSP_QSPI_GetInfo()

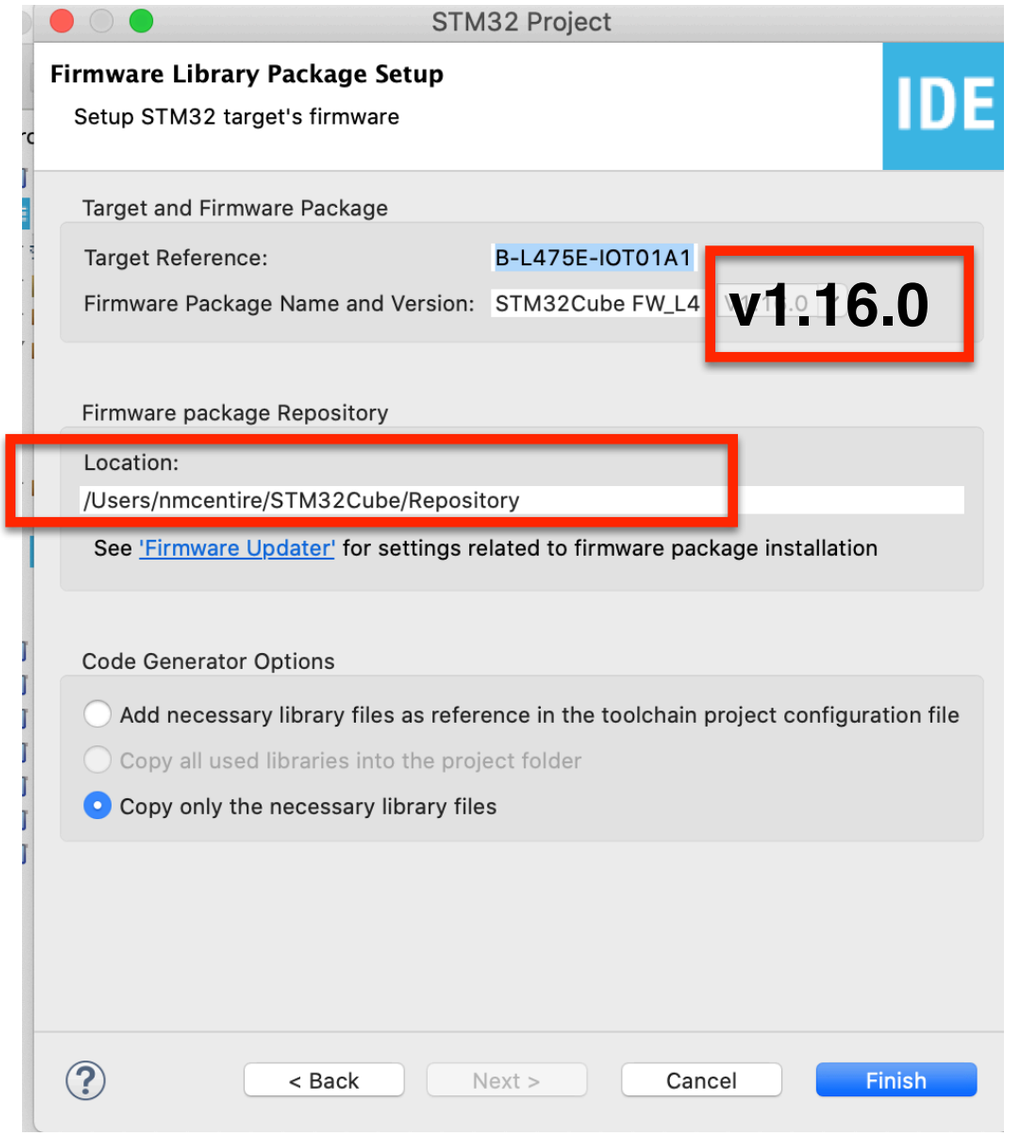
- **User Story 2** - Display results of calling BSP_QSPI_GetStatus()

- **User Story 3** - Display results of calling BSP_QSPI_GetInfo()

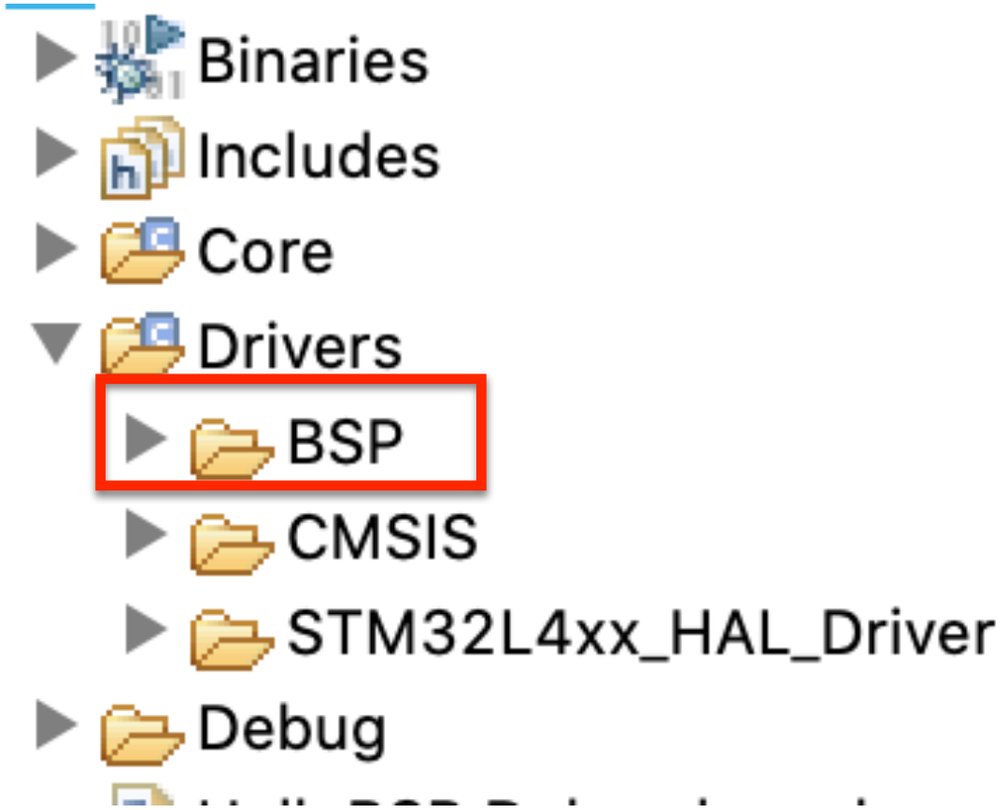
Project Setup - Part 1
Keep Defaults

Create New Project and Observe Firmware Version

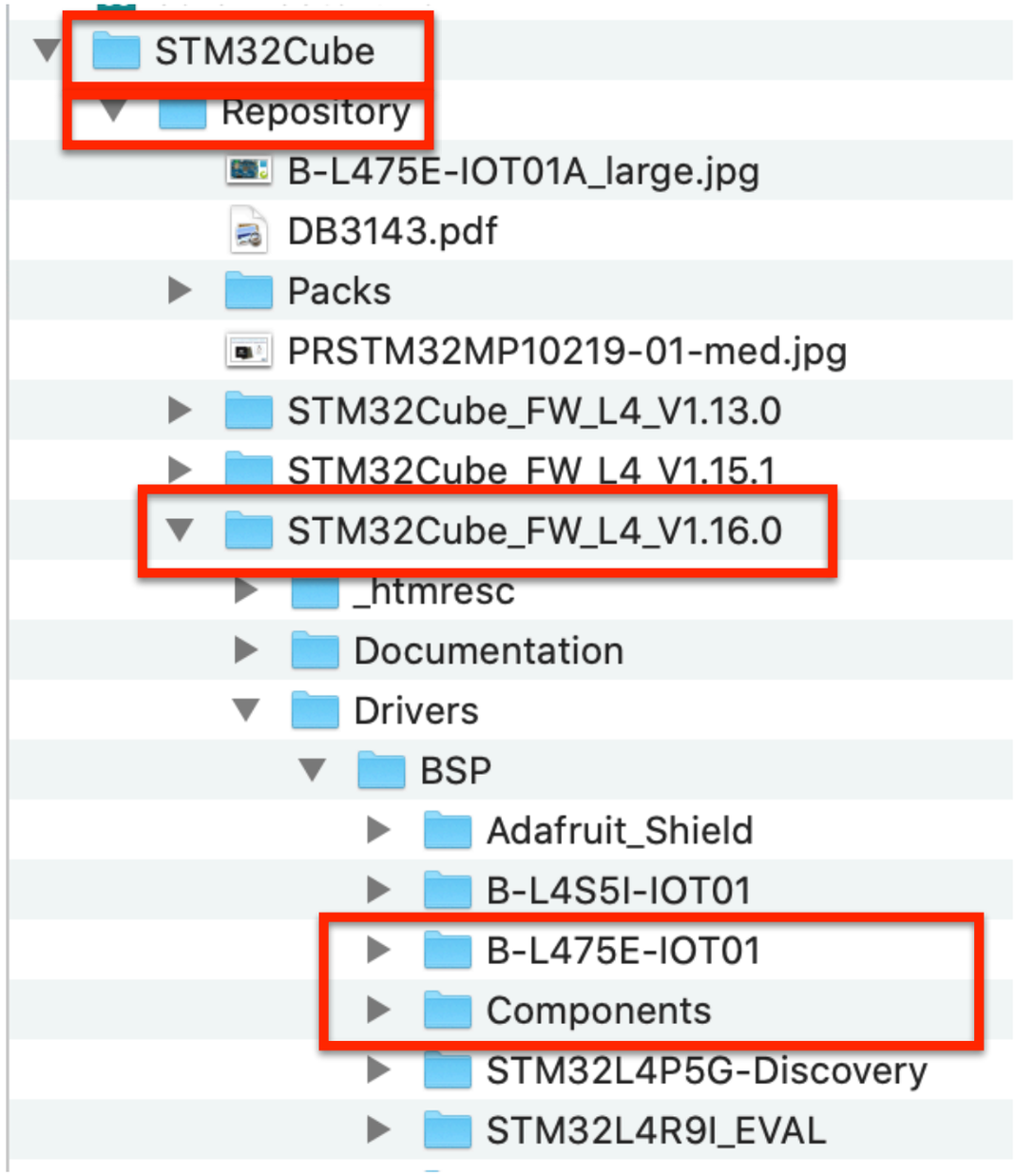




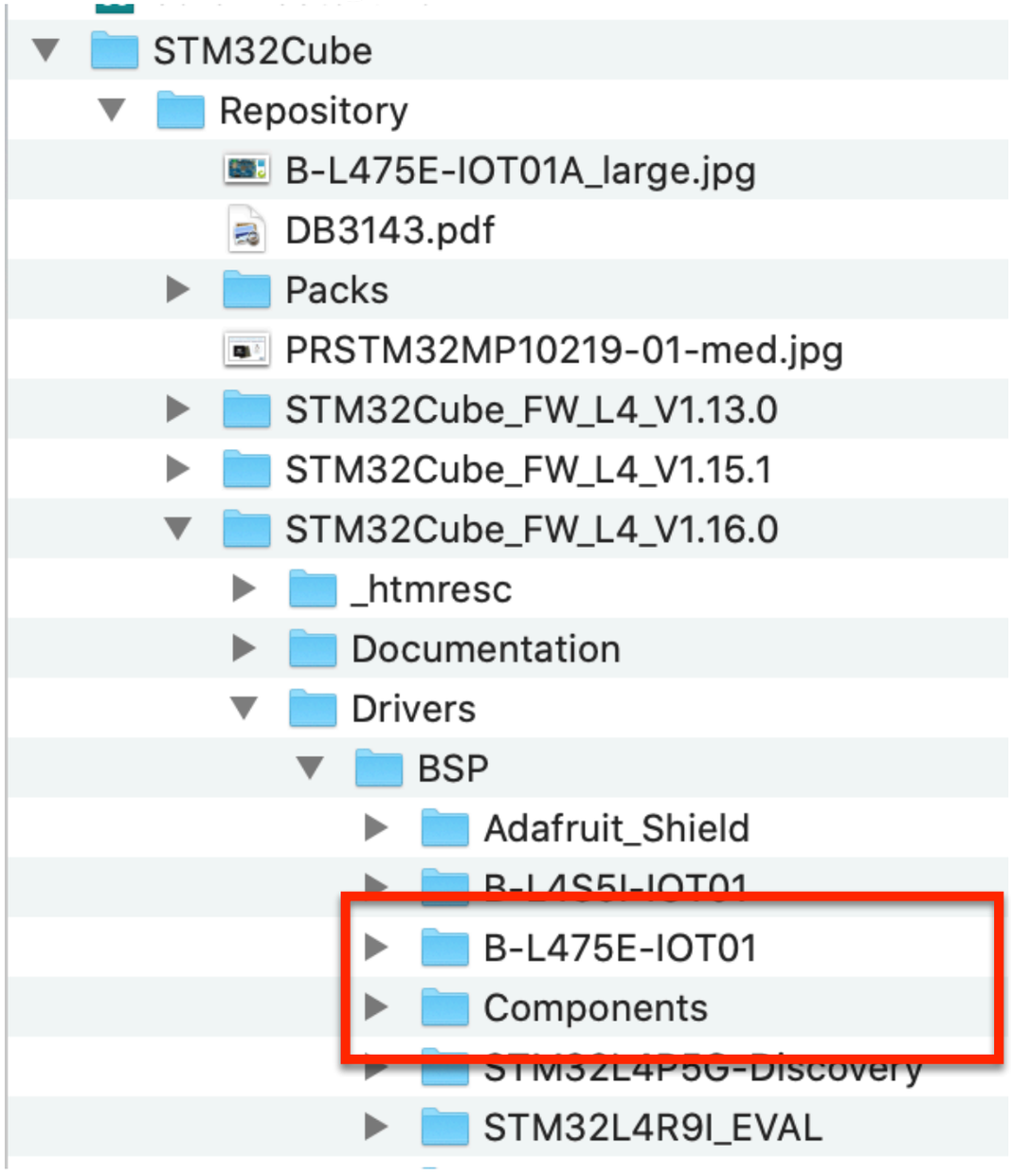
Create **BSP** Folder In Your Project Under Drivers

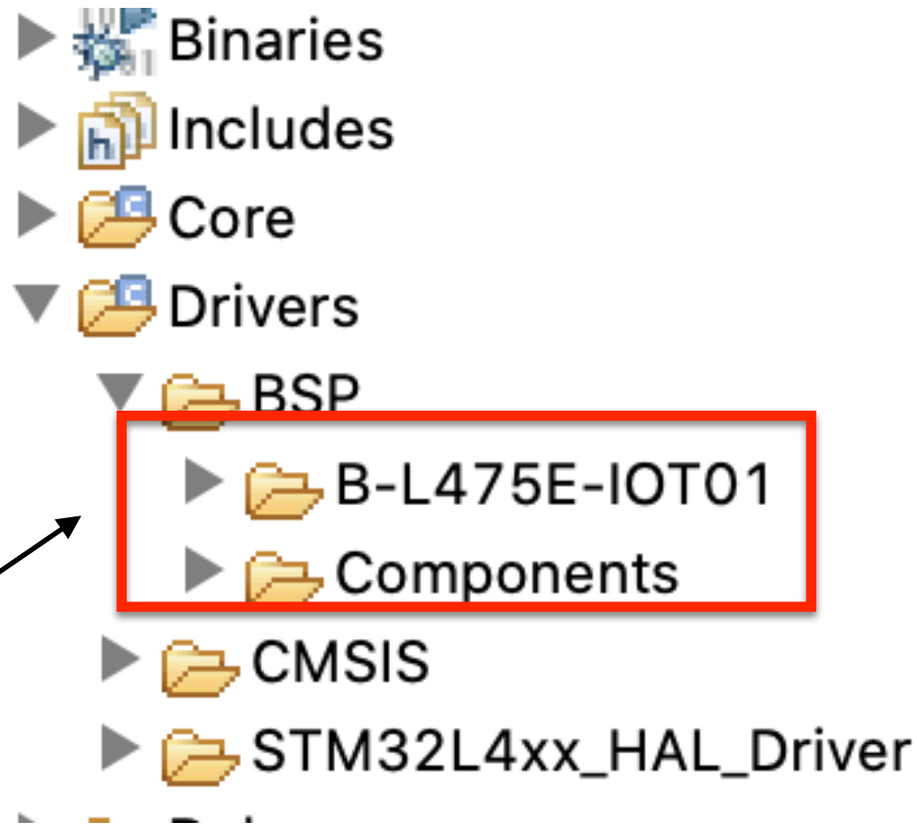


Find This Location on Your File System (in your home directory)



Drag/Drop Files from Folder to Project (Select **COPY FILES**)

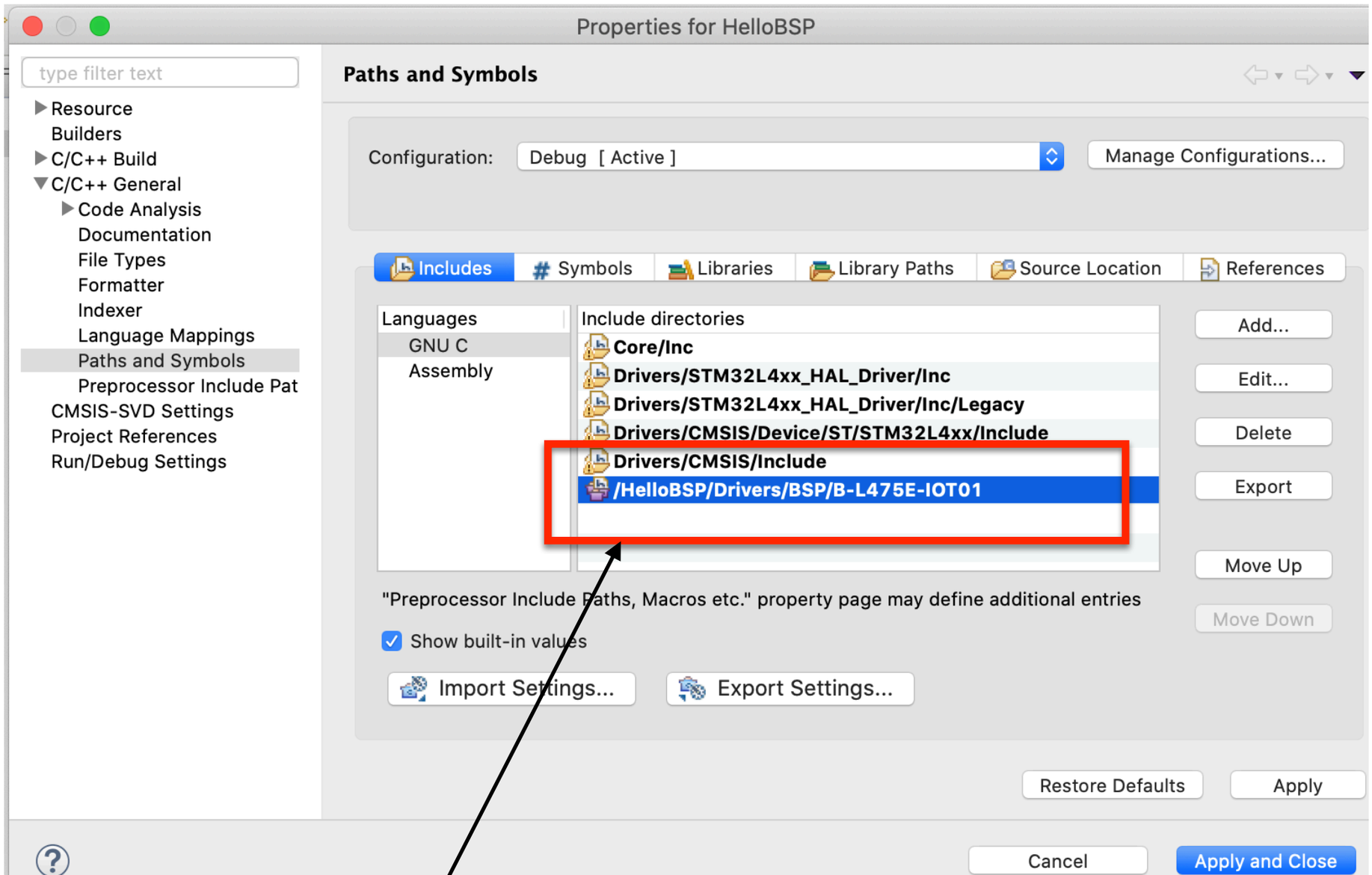




Add header file to main.c

```
22
23  /* Private includes ----------------
24    /* USER CODE BEGIN Includes */
25    #include "stm32l475e_iot01.h"
26
```

Add Path To Project



Add **Workspace** Path

User Story 1 Code
Add code to while(1)

```
155     /* USER CODE BEGIN WHILE */
156     while (1)
157     {
158         // Issue command prompt
159         char *prompt = "1=BSP_QSPI_Status(), 2=BSP_QSPI_Info()\n\rcmd> ";
160         HAL_UART_Transmit(&huart1, (uint8_t *)prompt, strlen(prompt), 1000);
161
162         // Wait for a single number entry
163         char ch;
164         HAL_UART_Receive(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
165
166         char *msg = "What?";
167         switch (ch) {
168             case '1': {
169                 msg = "\r\nBSP_QSPI_Status()\r\n";
170                 HAL_UART_Transmit(&huart1, (uint8_t *) msg, strlen(msg), 1000);
171                 do_qspi_status();
172                 break;
173             }
174             case '2': {
175                 msg = "\r\nBSP_SQPI_Info()\r\n";
176                 HAL_UART_Transmit(&huart1, (uint8_t *) msg, strlen(msg), 1000);
177                 do_qspi_info();
178                 break;
179             }
180             default: {
181                 char *msg = "\r\nWhat?\r\n";
182                 HAL_UART_Transmit(&huart1, (uint8_t *) msg, strlen(msg), 1000);
183             }
184         }
185
```

User Story 1 Code
Add stubbed out code

```
/* Private user code ----------
/* USER CODE BEGIN 0 */

void do_qspi_status() {

}

void do_qspi_info() {

}

/* USER CODE END 0 */
```

User Story 1 Code
Add Header Files

```
24  /* USER CODE BEGIN Includes */
25  #include "stm32l475e_iot01.h"
26  #include "stm32l475e_iot01_qspi.h"
27
28  #include <string.h>
29  #include <stdio.h>
30
31  /* USER CODE END Includes */
```

User Story 1
Running Code - CLI

```
cmd> 1=BSP_QSPI_Status(), 2=BSP_QSPI_Info()
cmd>
```

User Story 2

BSP_QSPI_Status()

User Story 2 Code
Add Code

```
81
82  void do_qspi_status() {
83
84      uint8_t status = BSP_QSPI_GetStatus();
85
86      char buf[100];
87      snprintf(buf, sizeof(buf), "status: %d\r\n", status);
88
89      HAL_UART_Transmit(&huart1, (uint8_t *) buf, strlen(buf), 1000);
90
91  }
92
```

User Story 2
Running

```
cmd> 1=BSP_QSPI_Status(), 2=BSP_QSPI_Info()
cmd>
BSP_QSPI_Status()
status: 1
```

User Story 3 Code

```
92
93  void do_qspi_info() {
94
95      QSPI_Info info = { 0 };
96
97      BSP_QSPI_GetInfo(&info);
98
99      char buf[100];
100     snprintf(buf, sizeof(buf), "FlashSize: %lu\r\n"
101         "EraseSectorSize: %lu\r\n"
102         "ProgPageSize: %lu\r\n",
103         info.FlashSize,
104         info.EraseSectorSize,
105         info.ProgPageSize);
106
107     HAL_UART_Transmit(&huart1, (uint8_t *) buf, strlen(buf), 1000);
108
109 }
110
111 /* USER CODE END 0 */
```

User Story 3 Code Running

```
1=BSP_QSPI_Status(), 2=BSP_QSPI_Info()
cmd>
BSP_SQPI_Info()
FlashSize: 8388608
EraseSectorSize: 4096
ProgPageSize: 256
```

8388608.0 / 1024.0 / 1024.0 = 8MB

Ultra Low Power 64M-BIT [x 1/x 2/x 4] CMOS MXSMIO® (SERIAL MULTI I/O) FLASH MEMORY

HARDWARE FEATURES

64MB-bit / 8bits/byte = 8MB

Summary

- Concepts
- Data Sheet - STM32L475
- User Manual - UM2153 - STM32L Discovery Kit for IoT
- Schematics - STM32L Discovery Kit for IoT
- API - STM32L HAL and BSP
    - Data Structures
    - Functions
- Hands-On Project