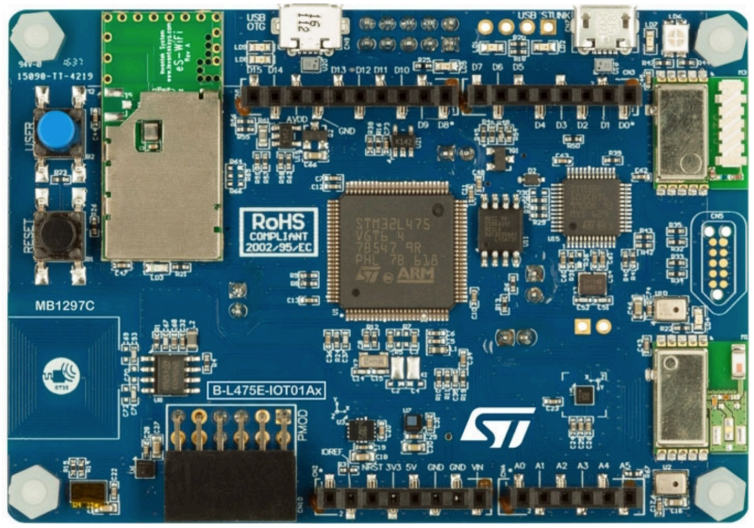
Embedded Systems Hardware Interfacing

Serial/UART

Norman McEntire



Contents

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

References

- https://en.wikipedia.org/wiki/Asynchronous_serial_communication
- https://en.wikipedia.org/wiki/RS-232
- https://en.wikipedia.org/wiki/Serial_port
- 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

Serial/UART

Serial Concepts

- Nearly all SOCs include multiple Serial I/O options
- UART
  - Universal Async Receiver/Transmitter
- USART
  - Universal Synchronous/Asynchronous Receiver/Transmitter

Serial Concepts
Serial Signal Level Options

- TTL - 5VDC
- TTL - 3.3 VDC
- RS-232C

Serial Example
USB to TTL Serial Cable

USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi

PRODUCT ID: 954

The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at the end of the 36" cable are four wire - red power, black ground, white RX into USB port, and green TX out of...

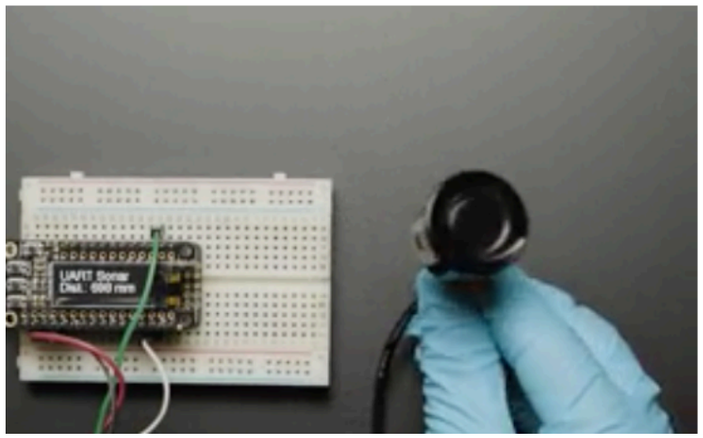
Serial Example
Termal Printer

Tiny Thermal Receipt Printer - TTL Serial / USB

PRODUCT ID: 2751

Add a really small printer to any microcontroller project with this very cute thermal printer. Thermal printers are also known as receipt printers, they're what you get when you go to the ATM or grocery store. Now you can embed a little printer of your own into an enclosure. This printer is...

Serial Example
Ultrasonic Sensor



IN STOCK

Panel Mount Ultrasonic (Sonar) Ranging Module with UART Output

PRODUCT ID: 4665

This sleek looking 'button' is actually an ultrasonic sensor, ready for panel mount use. Inside is a module that uses high frequency signals to determine distance to a surface. It's elegant looking and is more durable than the 'raw' ultrasonics we sell a lot of, so it would do...

Serial Example
Serial LCD

USB + Serial Backpack Kit with 16x2 RGB backlight positive LCD - Black on RGB

PRODUCT ID: 782

Adding a character display to your project or computer has never been easier with the new Adafruit USB or TTL serial backpack! This custom-designed PCB sits on the back of our 'standard' character LCD (16x2 or 20x4 sized) and does everything you could want: printing text, automatic...

Serial Example
Serial GPS

Serial Example
Serial-to-Wifi

ISM43362-M3G-L44-E/U Serial-to-WiFi Module

The ISM43362-M3G-L44-E/U is an embedded 2.4 GHz Wi-Fi module. The Wi-Fi module hardware consists of a Cypress CYW43362, an integrated antenna or optional external antenna, and a STM32F205 host processor that has a SPI or UART interface capability.

The Wi-Fi module requires no operating system and has an integrated TCP/IP stack that only requires a simple AT command set to establish connectivity for your wireless product. The module has a very small 14.5mm x 30mm surface mount footprint and has full FCC, IC, Japan and CE module certification.

Data Sheet

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

- 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 Data Sheet

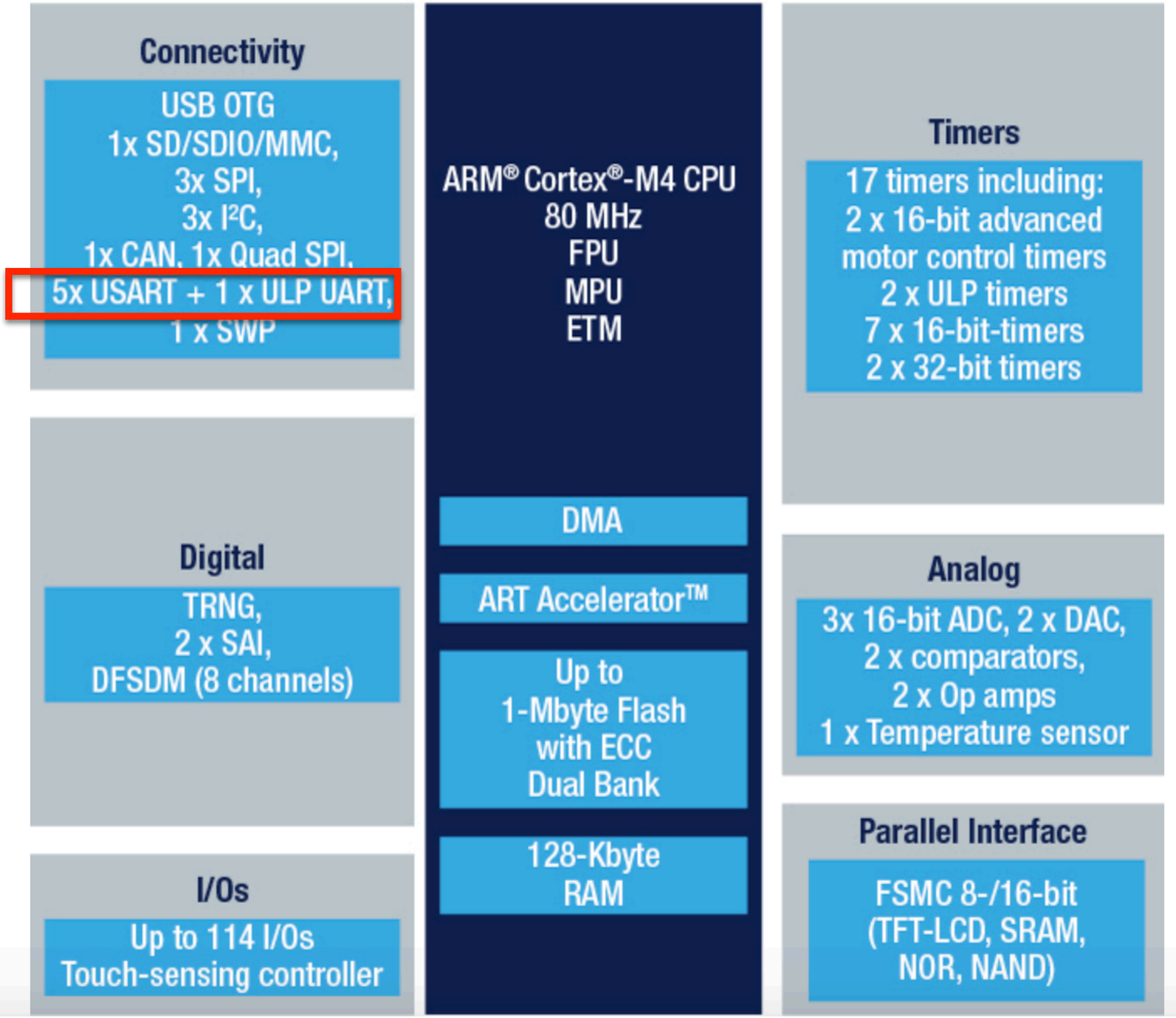
life.augmented

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

STM32L475



3.26 Universal synchronous/asynchronous receiver transmitter (USART)

The STM32L475xx devices have three embedded universal synchronous receiver transmitters (USART1, USART2 and USART3) and two universal asynchronous receiver transmitters (UART4, UART5).

These interfaces provide asynchronous communication, IrDA SIR ENDEC support, multiprocessor communication mode, single-wire half-duplex communication mode and have LIN Master/Slave capability. They provide hardware management of the CTS and RTS signals, and RS485 Driver Enable. They are able to communicate at speeds of up to 10Mbit/s.

USART1, USART2 and USART3 also provide Smart Card mode (ISO 7816 compliant) and SPI-like communication capability.

All USART have a clock domain independent from the CPU clock, allowing the USARTx (x=1,2,3,4,5) to wake up the MCU from Stop mode using baudrates up to 204 Kbaud. The wake up events from Stop mode are programmable and can be:

- Start bit detection
- Any received data frame
- A specific programmed data frame

All USART interfaces can be served by the DMA controller.

CTS - Clear to Send
RTS - Request To Send

Table 13. STM32L475xx USART/UART/LPUART features

USART modes/features(1)	USART1	USART2	USART3	UART4	UART5	LPUART1
Hardware flow control for modem	X	X	X	X	X	X
Continuous communication using DMA	X	X	X	X	X	X
Multiprocessor communication	X	X	X	X	X	X
Synchronous mode	X	X	X	-	-	-
Smartcard mode	X	X	X	-	-	-
Single-wire half-duplex communication	X	X	X	X	X	X
IrDA SIR ENDEC block	X	X	X	X	X	-
LIN mode	X	X	X	X	X	-
Dual clock domain	X	X	X	X	X	X
Wakeup from Stop 0 / Stop 1 modes	X	X	X	X	X	X
Wakeup from Stop 2 mode	-	-	-	-	-	X
Receiver timeout interrupt	X	X	X	X	X	-
Modbus communication	X	X	X	X	X	-
Auto baud rate detection	X (4 modes)					-
Driver Enable	X	X	X	X	X	X
LPUART/USART data length	7, 8 and 9 bits					

1. X = supported.

3.27 Low-power universal asynchronous receiver transmitter (LPUART)

The device embeds one Low-Power UART. The LPUART supports asynchronous serial communication with minimum power consumption. It supports half duplex single wire communication and modem operations (CTS/RTS). It allows multiprocessor communication.

The LPUART has a clock domain independent from the CPU clock, and can wakeup the system from Stop mode using baudrates up to 220 Kbaud. The wake up events from Stop mode are programmable and can be:

- Start bit detection
- Any received data frame
- A specific programmed data frame

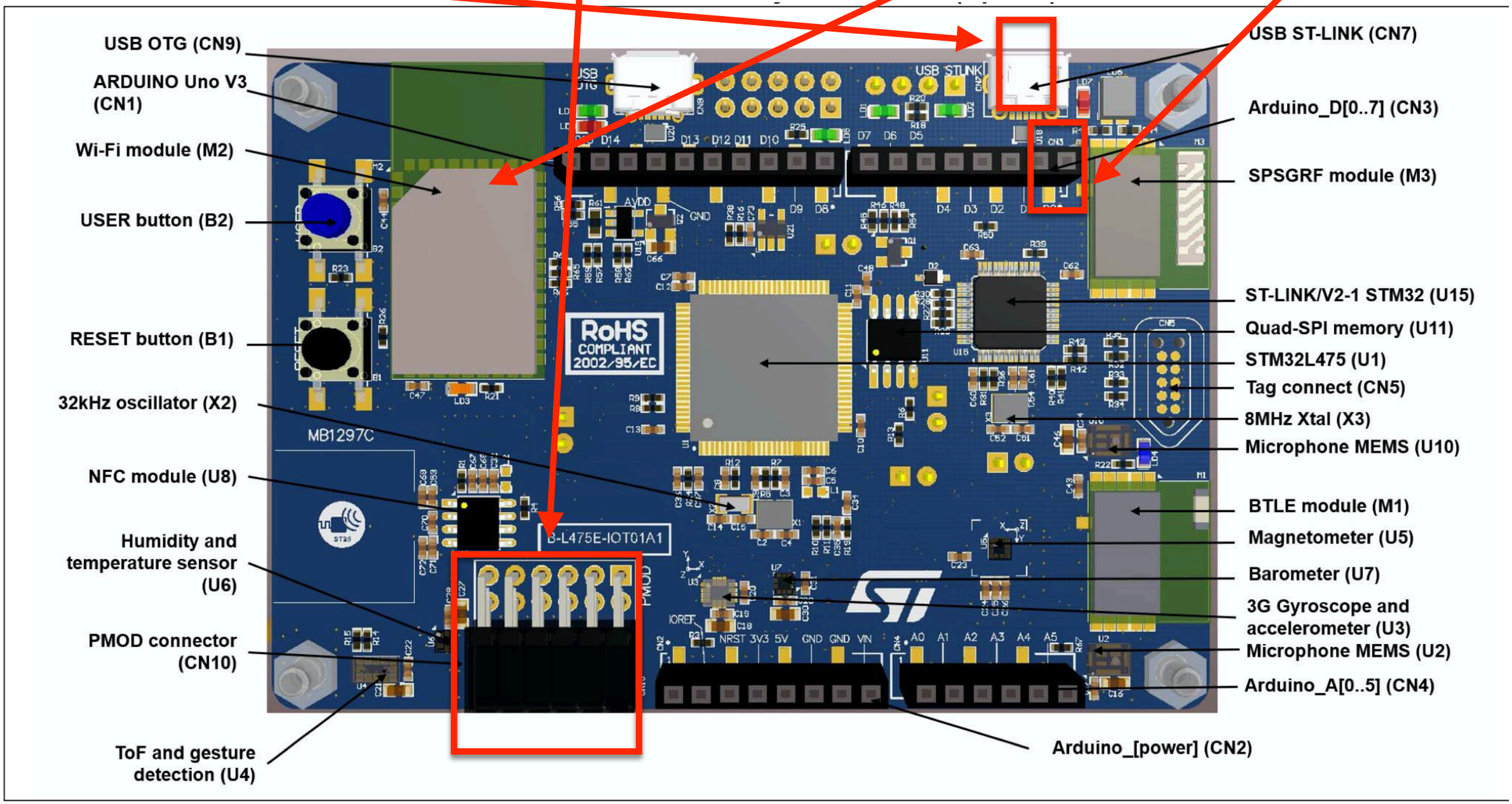
Only a 32.768 kHz clock (LSE) is needed to allow LPUART communication up to 9600 baud. Therefore, even in Stop mode, the LPUART can wait for an incoming frame while having an extremely low energy consumption. Higher speed clock can be used to reach higher baudrates.

LPUART interface can be served by the DMA controller.

User Manual

STM32L Discovery Kit IoT Node Serial/UART

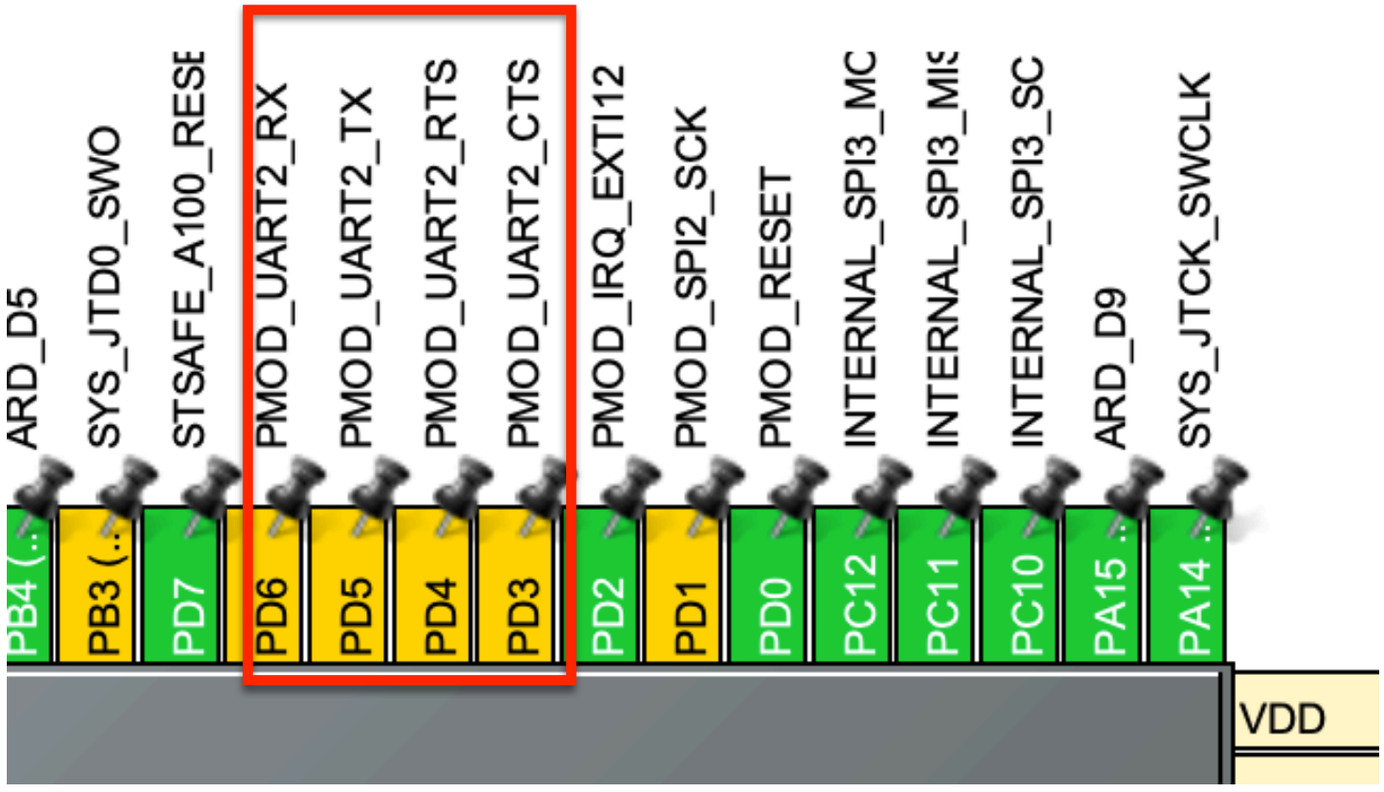
Use of Serial I/O
UART1, UART2, UART3, UART4



UART1

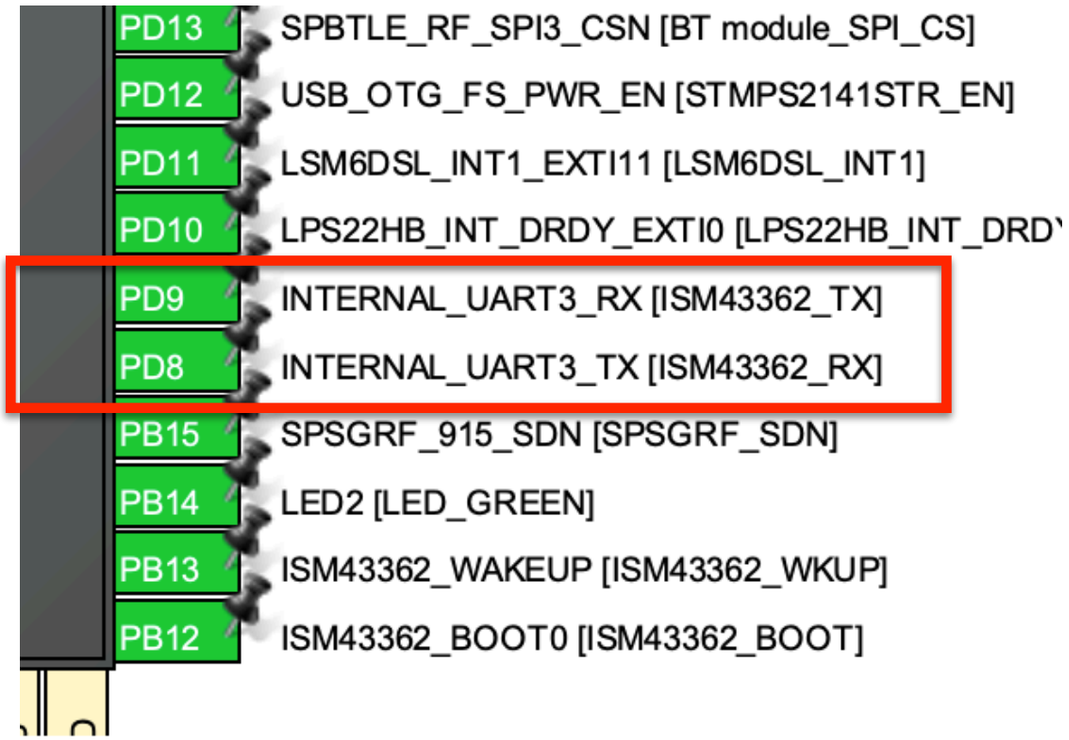
58	92	PB6	I/O	FT_fa	-	LPTIM1_ETR, TIM4_CH1, TIM8_BKIN2, I2C1_SCL, DFSDM1_DATIN5, USART1_TX, TSC_G2_IO3, TIM8_BKIN2_COMP2, SAI1_FS_B, TIM16_CH1N, EVENTOUT	COMP2_INP
59	93	PB7	I/O	FT_fa	-	LPTIM1_IN2, TIM4_CH2, TIM8_BKIN, I2C1_SDA, DFSDM1_CKIN5, USART1_RX, UART4_CTS, TSC_G2_IO4, FMC_NL, TIM8_BKIN_COMP1, TIM17_CH1N, EVENTOUT	COMP2_INM, PVD_IN

UART2



-	84	PD3	I/O	FT	-	SPI2_MISO, DFSDM1_DATIN0, USART2_CTS, FMC_CLK, EVENTOUT	-
-	85	PD4	I/O	FT	-	SPI2_MOSI, DFSDM1_CKIN0, USART2_RTS_DE, FMC_NOE, EVENTOUT	-
-	86	PD5	I/O	FT	-	USART2_TX, FMC_NWE, EVENTOUT	-
-	87	PD6	I/O	FT	-	DFSDM1_DATIN1, USART2_RX, FMC_NWAIT, SAI1_SD_A, EVENTOUT	-

UART3

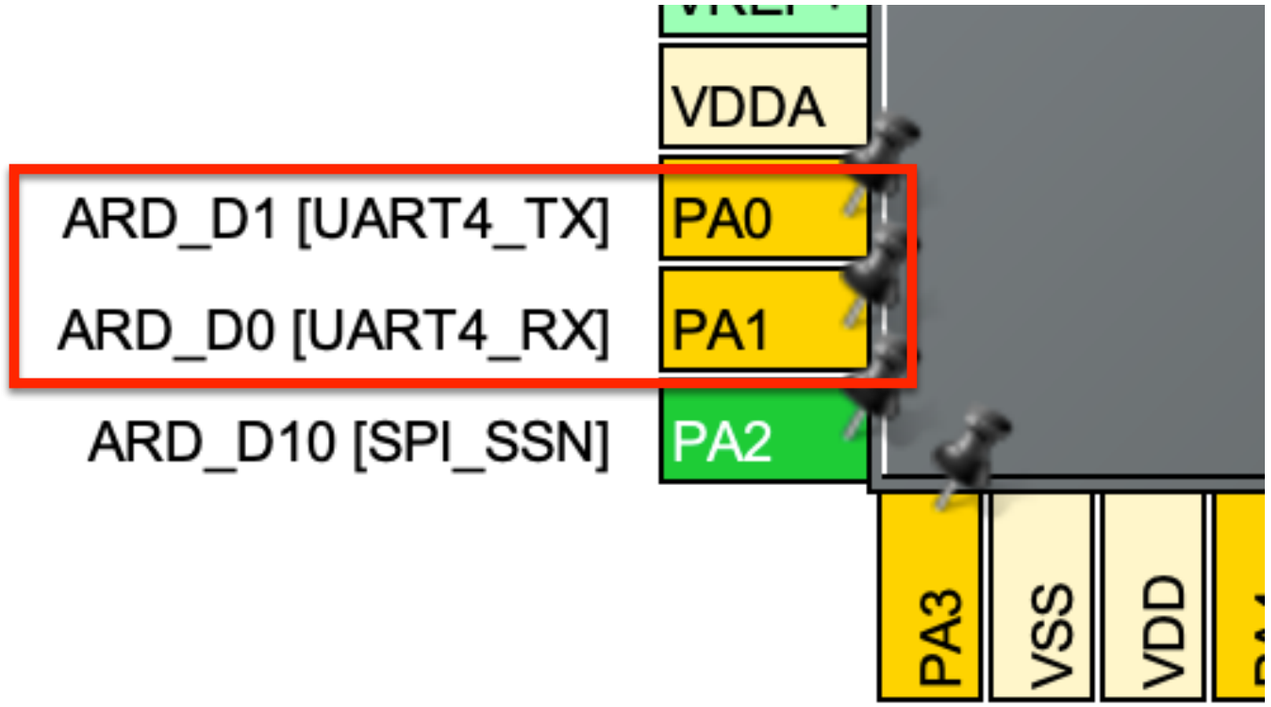


-	55	PD8	I/O	FT	-	USART3_TX, FMC_D13, EVENTOUT	-
-	56	PD9	I/O	FT	-	USART3_RX, FMC_D14, SAI2_MCLK_A, EVENTOUT	-

ISM43362-M3G-L44-E/U Serial-to-WiFi Module (More on this when we cover Wifi later in the course)

UART4

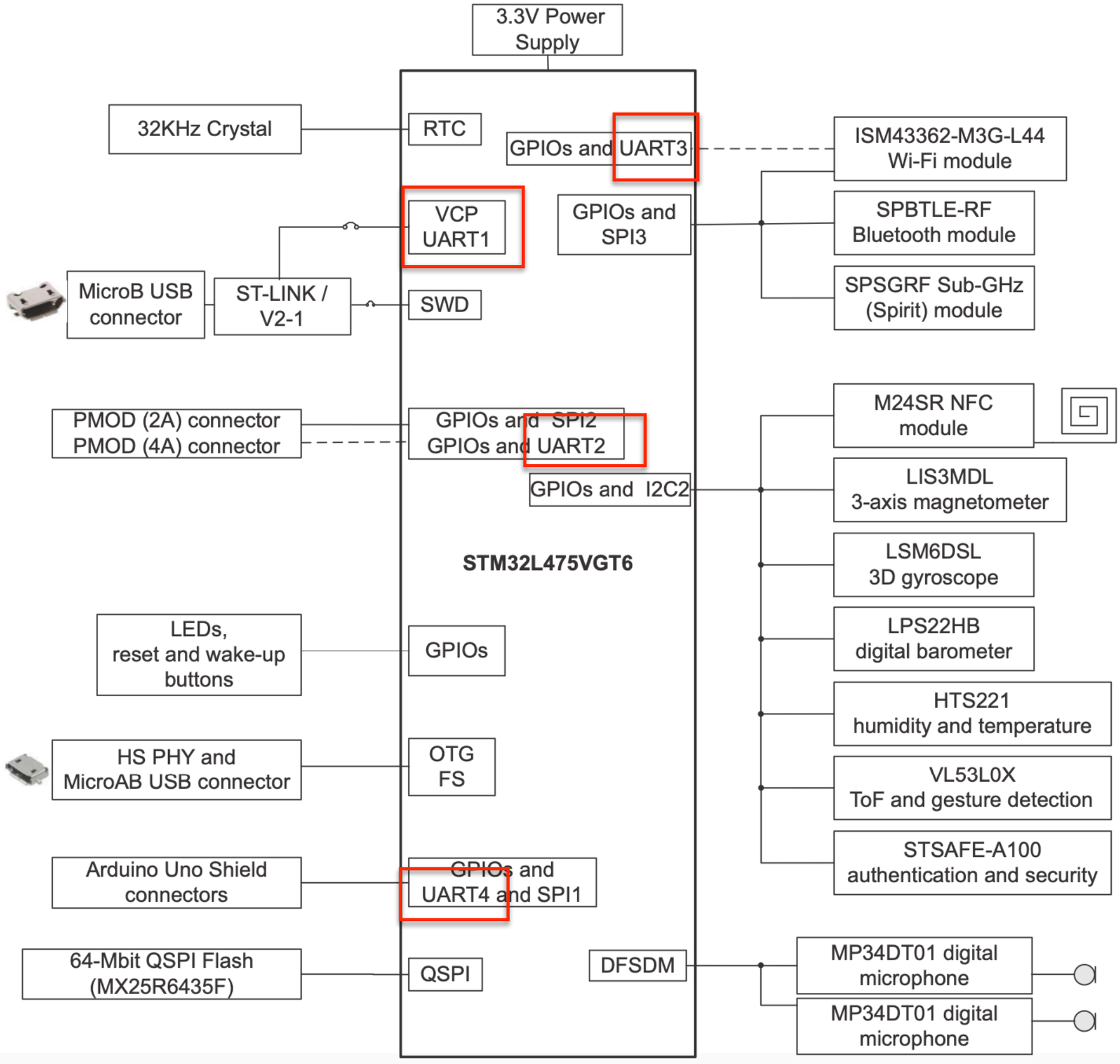
To Arduino Connector



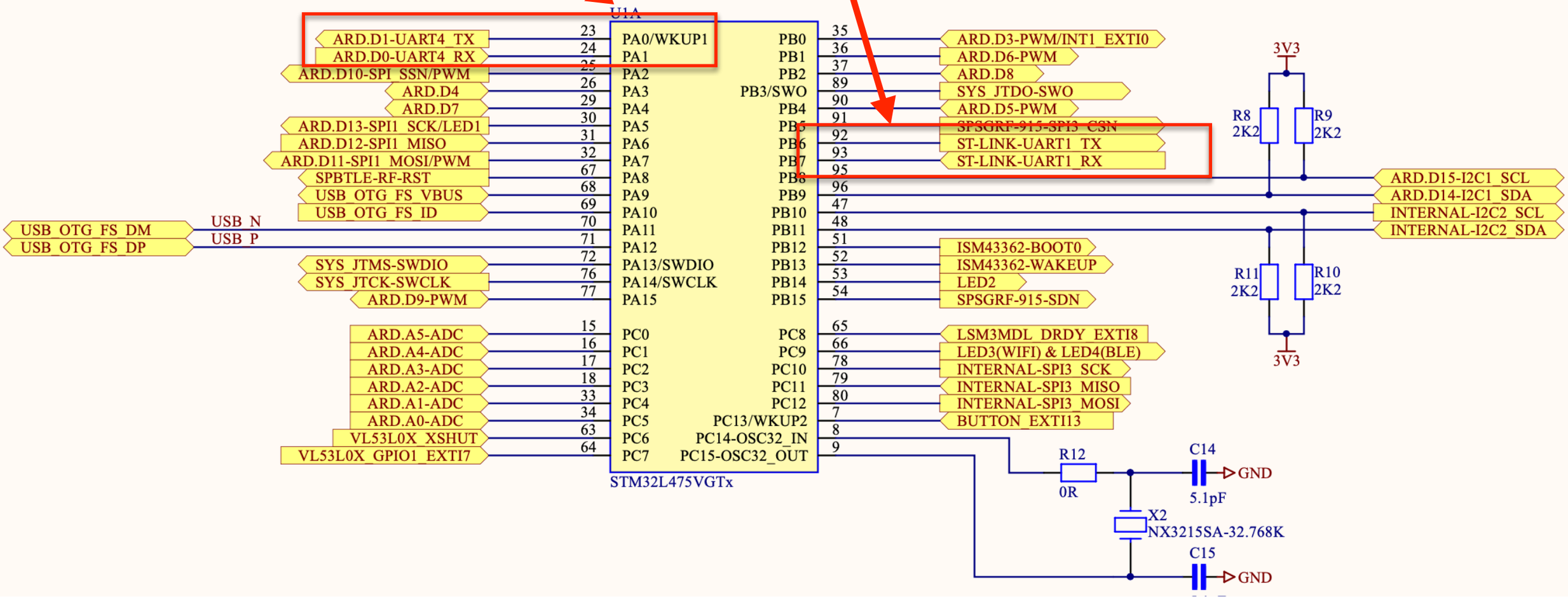
14	23	PA0	I/O	FT_a	-	TIM2_CH1, TIM5_CH1, TIM8_ETR, USART2_CTS, UART4_TX, SAI1_EXTCLK, TIM2_ETR, EVENTOUT	OPAMP1_VINP, ADC12_IN5, RTC_TAMP2/WKUP1
15	24	PA1	I/O	FT_a	-	TIM2_CH2, TIM5_CH2, USART2_RTS_DE, UART4_RX, TIM15_CH1N, EVENTOUT	OPAMP1_VINM, ADC12_IN6

Schematics

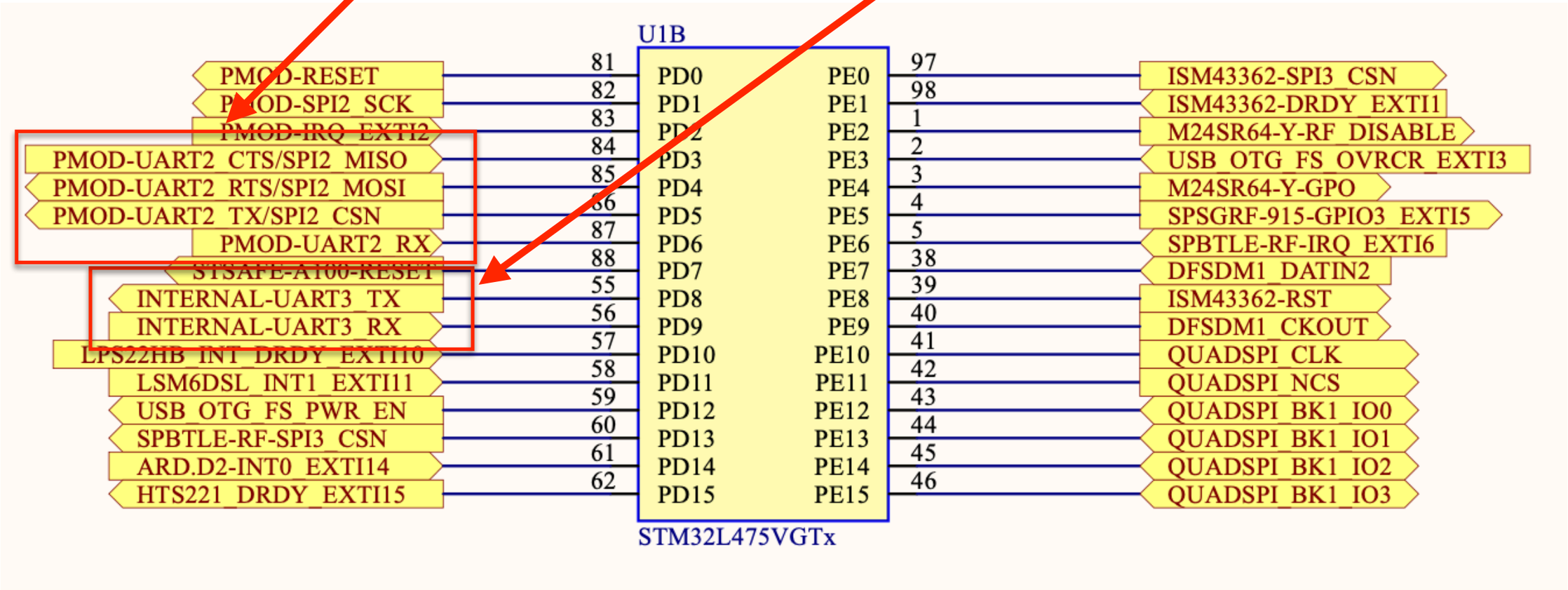
STM324L Discovery Kit IoT Node



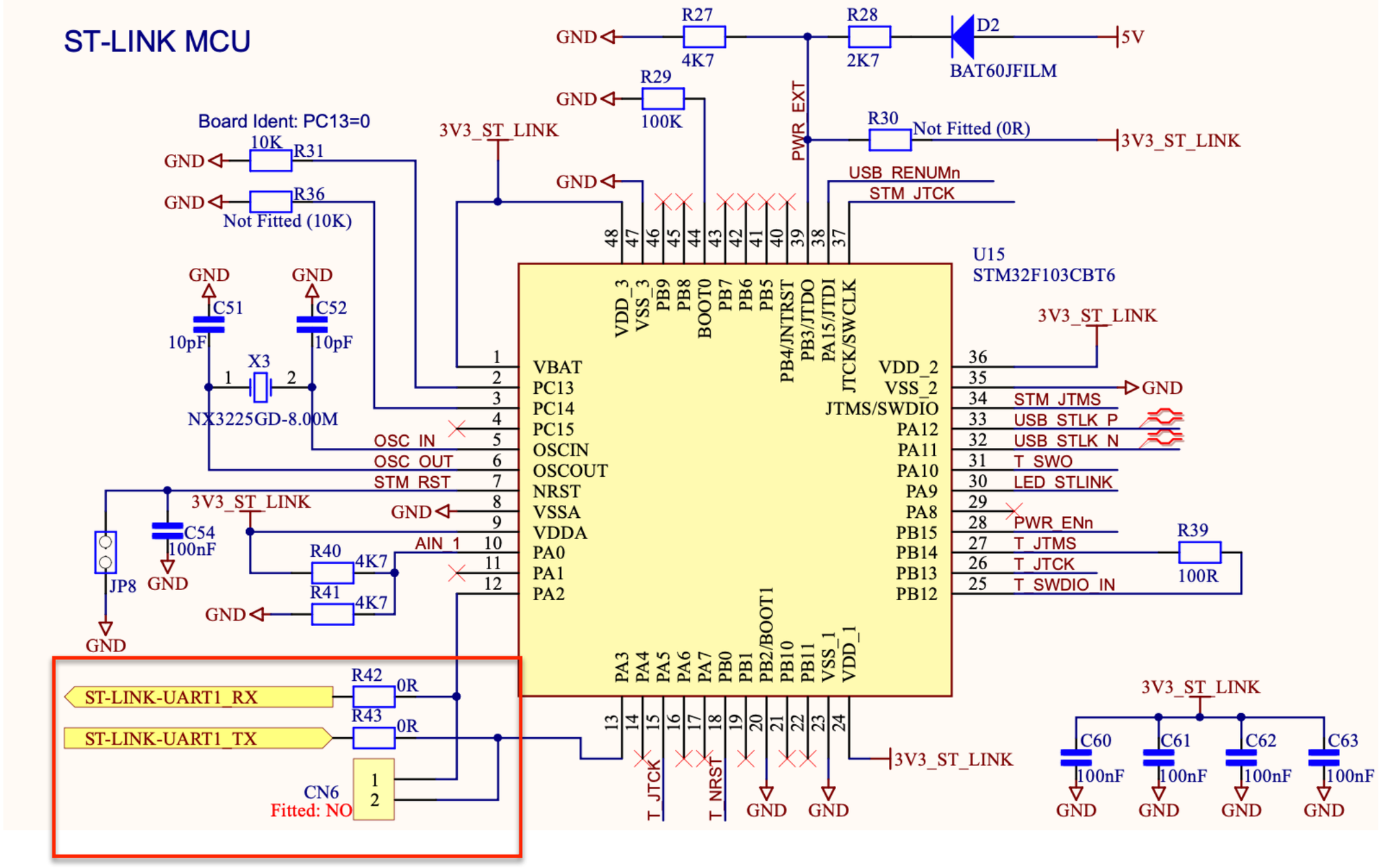
STM32L Discovery Kit IoT Node - UART1, UART4



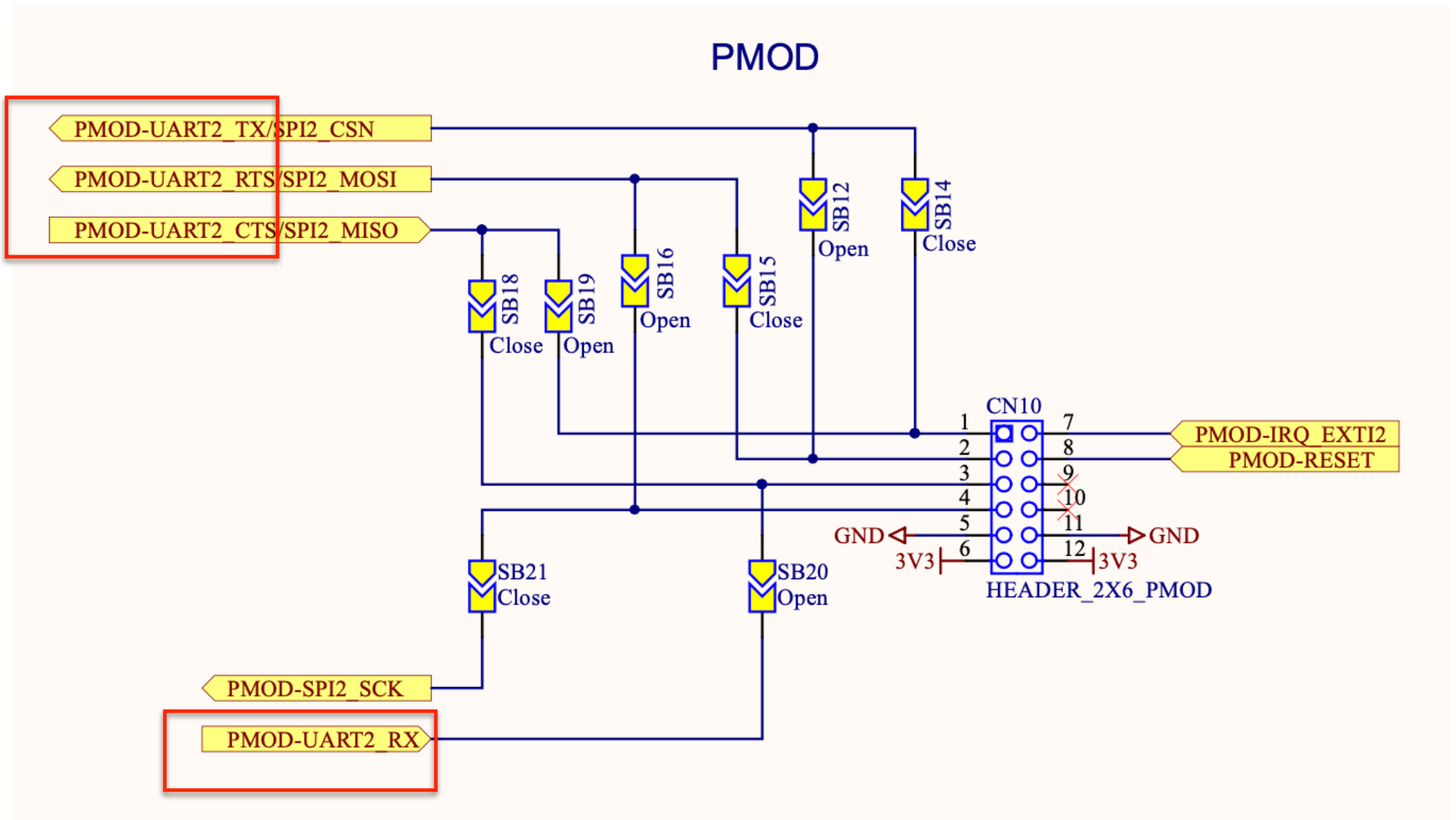
STM32L Discovery Kit IoT UART2 and UART3



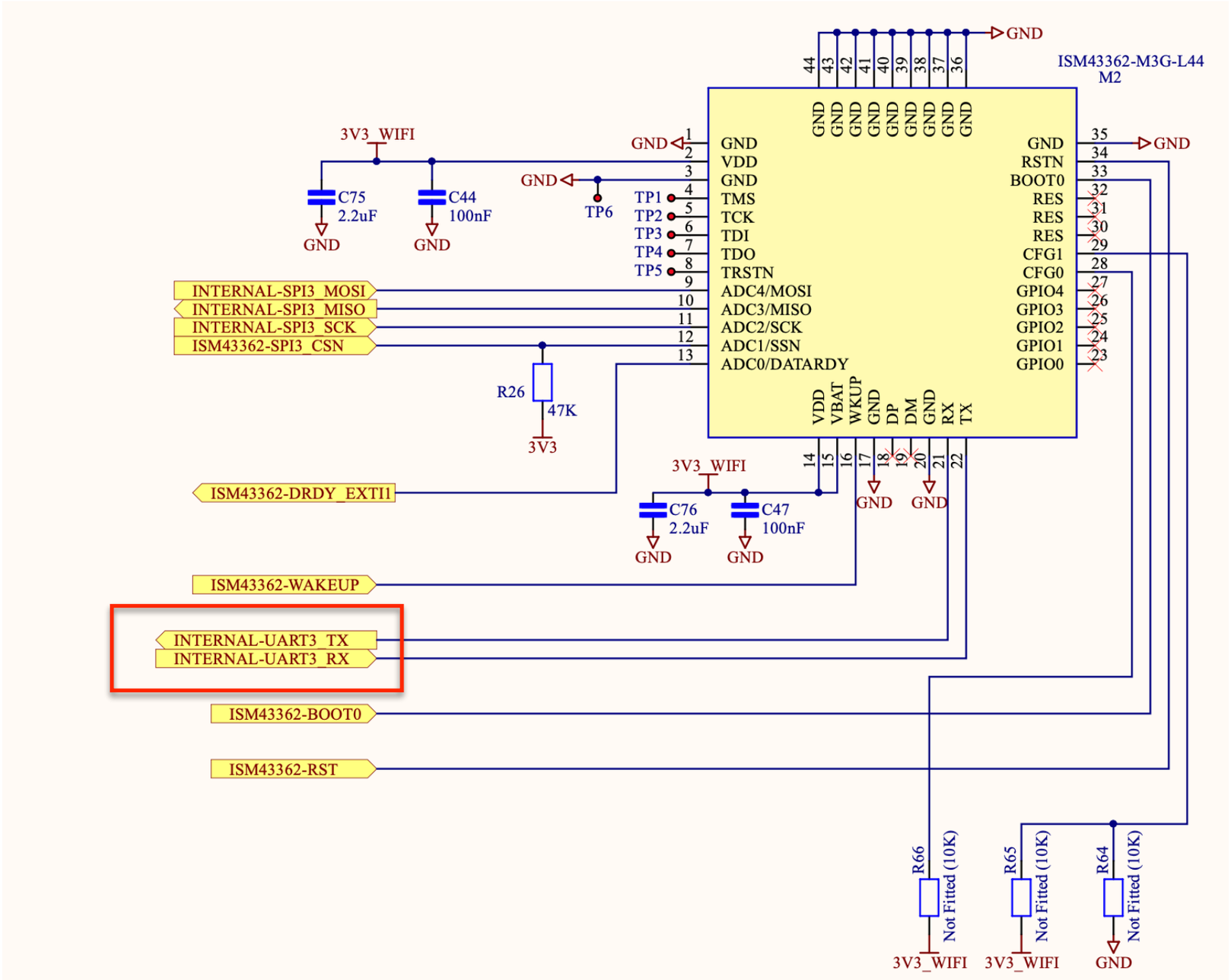
Use of UART1 Virtual Connector (ST-LINK)



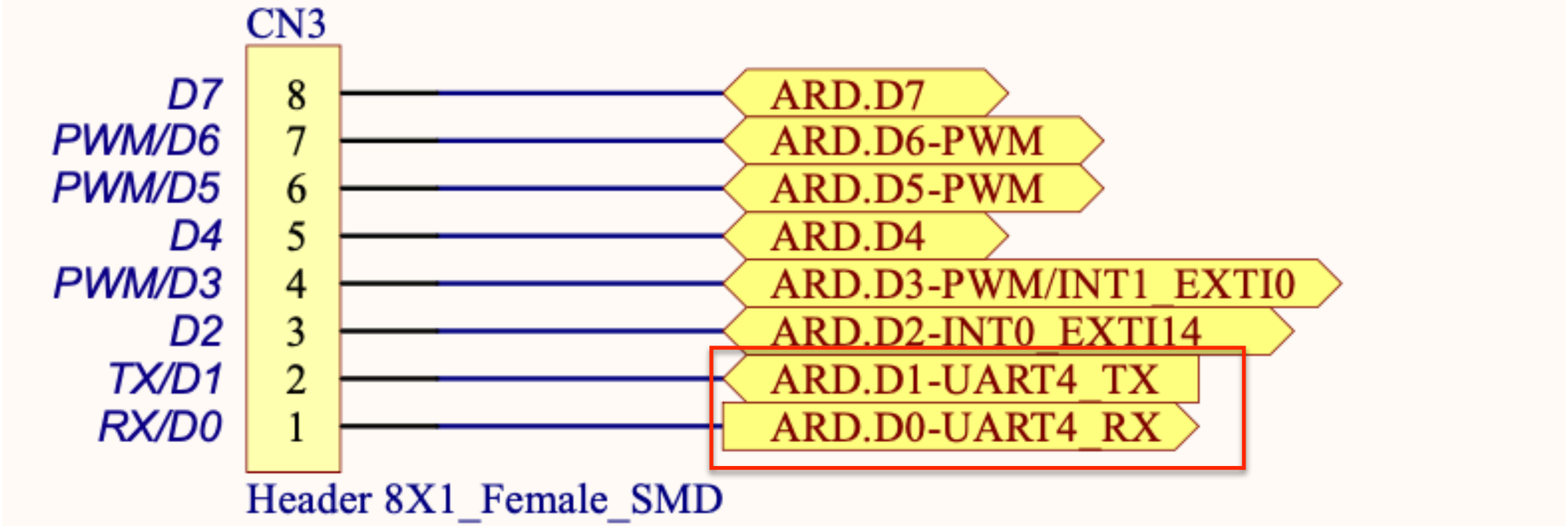
Use of UART2
PMOD Connector



Use of UART3 Wifi Module



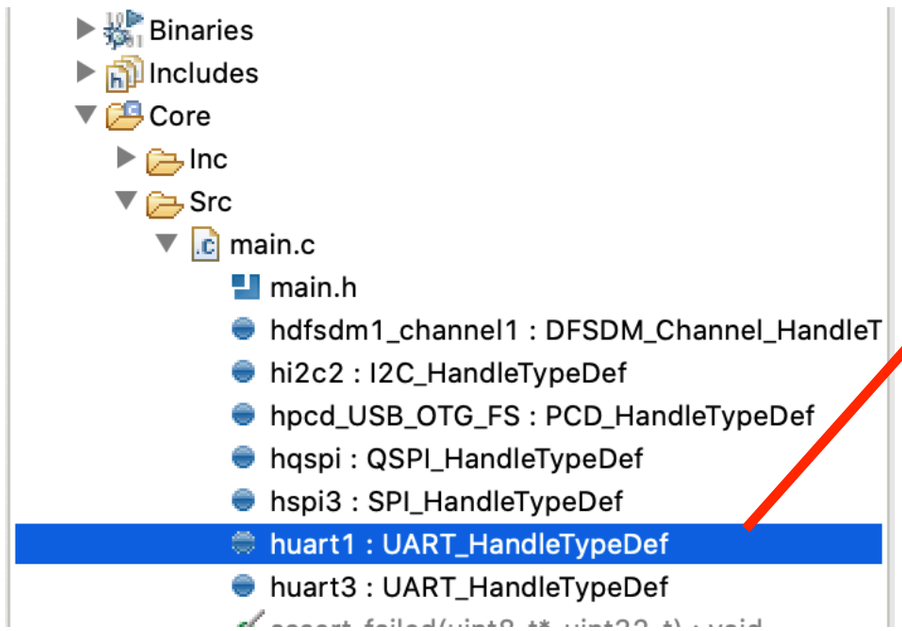
Use of UART4
Arduino Connector



UART API

Data Structures

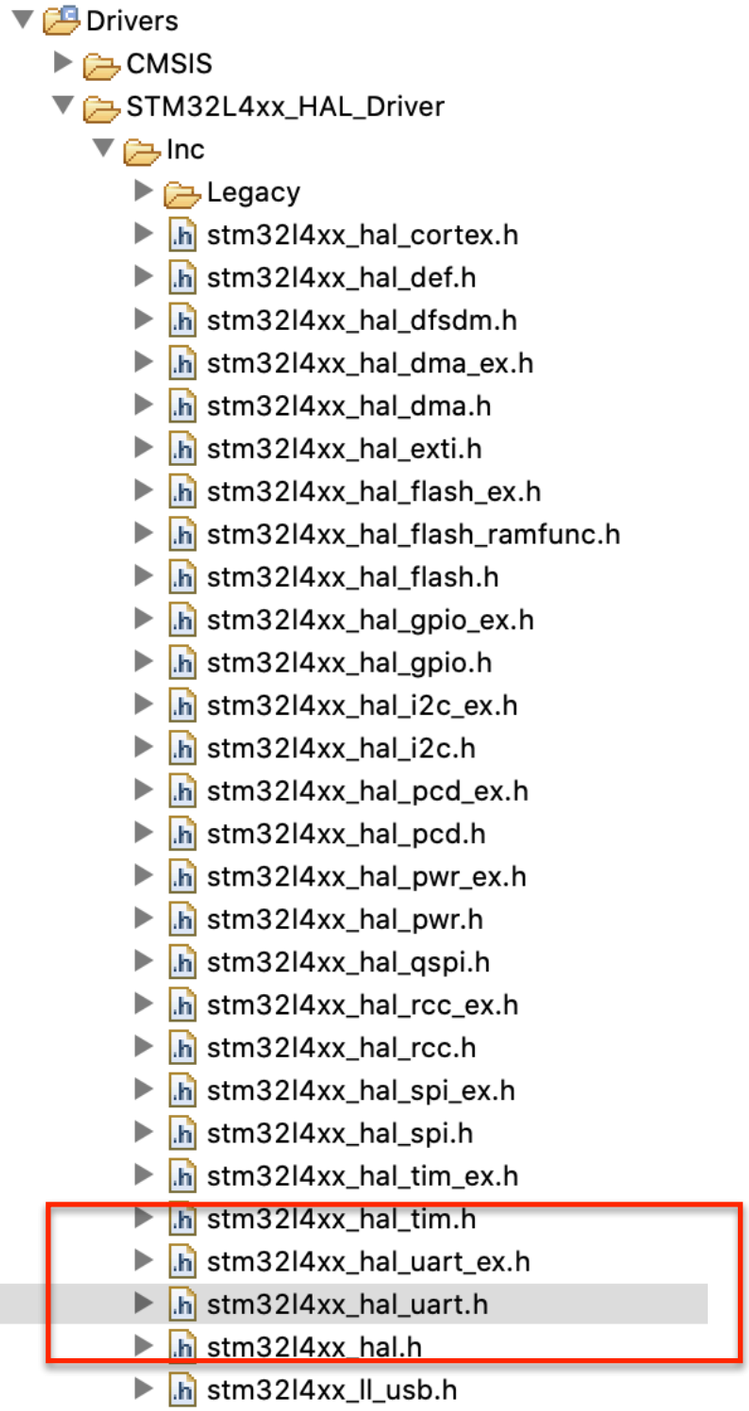
Default Project Has UART1




```
50
51  UART_HandleTypeDef huart1;
52  UART_HandleTypeDef huart3;
```

```
381 static void MX_USART1_UART_Init(void)
382 {
383
384   /* USER CODE BEGIN USART1_Init 0 */
385
386   /* USER CODE END USART1_Init 0 */
387
388   /* USER CODE BEGIN USART1_Init 1 */
389
390   /* USER CODE END USART1_Init 1 */
391   huart1.Instance = USART1;
392   huart1.Init.BaudRate = 115200;
393   huart1.Init.WordLength = UART_WORDLENGTH_8B;
394   huart1.Init.StopBits = UART_STOPBITS_1;
395   huart1.Init.Parity = UART_PARITY_NONE;
396   huart1.Init.Mode = UART_MODE_TX_RX;
397   huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
398   huart1.Init.OverSampling = UART_OVERSAMPLING_16;
399   huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
400   huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
401   if (HAL_UART_Init(&huart1) != HAL_OK)
402   {
403     Error_Handler();
404   }
405   /* USER CODE BEGIN USART1_Init 2 */
406
407   /* USER CODE END USART1_Init 2 */
408
409 }
```

stm32l4xx_hal_uart.h



UART_HandleTypeDef (partially shown)

```
191
192 /**
193   * @brief  UART handle Structure definition
194   */
195 typedef struct __UART_HandleTypeDef
196 {
197   USART_TypeDef            *Instance;              /*!< UART registers base address       */
198
199   UART_InitTypeDef         Init;                   /*!< UART communication parameters     */
200
201   UART_AdvFeatureInitTypeDef AdvancedInit;         /*!< UART Advanced Features initialization param
202
203   uint8_t                  *pTxBuffPtr;            /*!< Pointer to UART Tx transfer Buffer */
204
205   uint16_t                 TxXferSize;             /*!< UART Tx Transfer size              */
206
207   __IO uint16_t            TxXferCount;            /*!< UART Tx Transfer Counter           */
208
209   uint8_t                  *pRxBuffPtr;            /*!< Pointer to UART Rx transfer Buffer */
210
211   uint16_t                 RxXferSize;             /*!< UART Rx Transfer size              */
212
213   __IO uint16_t            RxXferCount;            /*!< UART Rx Transfer Counter           */
214
215   uint16_t                 Mask;                   /*!< UART Rx RDR register mask          */
216
```

UART API

Functions

stm32l4xx_hal_uart.h Initialization

```
1591  /* Initialization and de-initialization functions  ******************************/
1592  HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart);
1593  HAL_StatusTypeDef HAL_HalfDuplex_Init(UART_HandleTypeDef *huart);
1594  HAL_StatusTypeDef HAL_LIN_Init(UART_HandleTypeDef *huart, uint32_t BreakDetectLength);
1595  HAL_StatusTypeDef HAL_MultiProcessor_Init(UART_HandleTypeDef *huart, uint8_t Address, uint32_t WakeUp
1596  HAL_StatusTypeDef HAL_UART_DeInit(UART_HandleTypeDef *huart);
1597  void HAL_UART_MspInit(UART_HandleTypeDef *huart);
```

stm32l4xx_hal_uart.h
Polling Mode

```
1616  HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_
1617  HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t
```

stm32l4xx_hal_uart.h Interrupt Mode

```
1618  HAL_StatusTypeDef HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
1619  HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
```

stm32l4xx_hal_uart.h
DMA Mode

```
1620  HAL_StatusTypeDef HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
1621  HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
1622  HAL_StatusTypeDef HAL_UART_DMAPause(UART_HandleTypeDef *huart);
1623  HAL_StatusTypeDef HAL_UART_DMAResume(UART_HandleTypeDef *huart);
1624  HAL_StatusTypeDef HAL_UART_DMAStop(UART_HandleTypeDef *huart);
```

stm32l4xx_hal_uart.h
Callbacks for Interrupt and DMA Modes

```
1633  void HAL_UART_IRQHandler(UART_HandleTypeDef *huart);
1634  void HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef *huart);
1635  void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart);
1636  void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart);
1637  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);
1638  void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart);
1639  void HAL_UART_AbortCpltCallback(UART_HandleTypeDef *huart);
1640  void HAL_UART_AbortTransmitCpltCallback(UART_HandleTypeDef *huart);
1641  void HAL_UART_AbortReceiveCpltCallback(UART_HandleTypeDef *huart);
1642
```

Serial/UART

Hands-On Project

UART Hands-On Project Overview

- The goal of this project is to give you hands-on experience with UART using polled, interrupt driven, and DMA operation
  - Using STM32L4 Discovery Kit IoT Node
  - Using STM32CubeIDE
- 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

UART Hands-On Project User Stories

- User Story 1 - UART1 - 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 = polling, 2 = interrupt, 3 = DMA
- User Story 2 - UART4 - Init to 115200. Connect ARD-D0 to ARD-D1 (loopback of TX/RX on UART4. Use HAL polling UART APIs to send/ receive a constant string of “abcd….z” out the port and then read back in.
    - How much data can you send out of UART4 and then read back in from UARRT4 without loss of data?
- User Story 3 - UART4 - Repeat User Story #2 but use interrupts
- User Story 4 - UART4 - Repeat User Story #2 but use DMA

User Story 1 Code

```
159
160     /* Infinite loop */
161     /* USER CODE BEGIN WHILE */
162     while (1)
163     {
164         // Issue command prompt
165         char *prompt = "Options: 1=polling, 2=interrupt, 3=DMA\n\rcmd> ";
166         HAL_UART_Transmit(&huart1, (uint8_t *)prompt, strlen(prompt), 1000);
167
168         // Wait for a signle number entry
169         char ch;
170         HAL_UART_Receive(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
171
172         char *msg = "What?";
173         switch (ch) {
174         case '1': msg = "\r\nPolling\r\n"; do_polling(); break;
175         case '2': msg = "\r\nTODO: Interrupt\r\n"; break;
176         case '3': msg  = "\r\nTODO: DMA\r\n"; break;
177         // Fall through if none
178         }
179         HAL_UART_Transmit(&huart1, (uint8_t *) msg, strlen(msg), 1000);
180
181       /* USER CODE END WHILE */
```

User Story 1
Running Code - CLI

```
Options: 1=polling, 2=interrupt, 3=DMA
cmd>
```

User Story 2 Code Polling - Part 1

```
77
78  /* Private user code ---------------------------------------------------------*/
79  /* USER CODE BEGIN 0 */
80
81  static char tx_buf[] = "abcdefghijklmnoqarstuvwzyz\r\n";
82  static char rx_buf[] = "abcdefghijklmnoqarstuvwzyz\r\n";
83
84  static void do_polling() {
85
86      char * ptx_buf = tx_buf;
87      char * prx_buf = rx_buf;
88
89      // Set rx buf to known pattern
90      for (int i = 0; i < sizeof(rx_buf); i++ ) rx_buf[i] = '?';
91
92      do {
93          // Let UART4 know we are active
94          char ch = '.';
95          HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 100);
96
97          // Send a char
98          HAL_UART_Transmit(&huart4, (uint8_t *)ptx_buf, 1, 100);
99
100         // Receive a char (we are in loopback)
101         HAL_UART_Receive(&huart4, (uint8_t *)prx_buf, 1, 100);
102
```

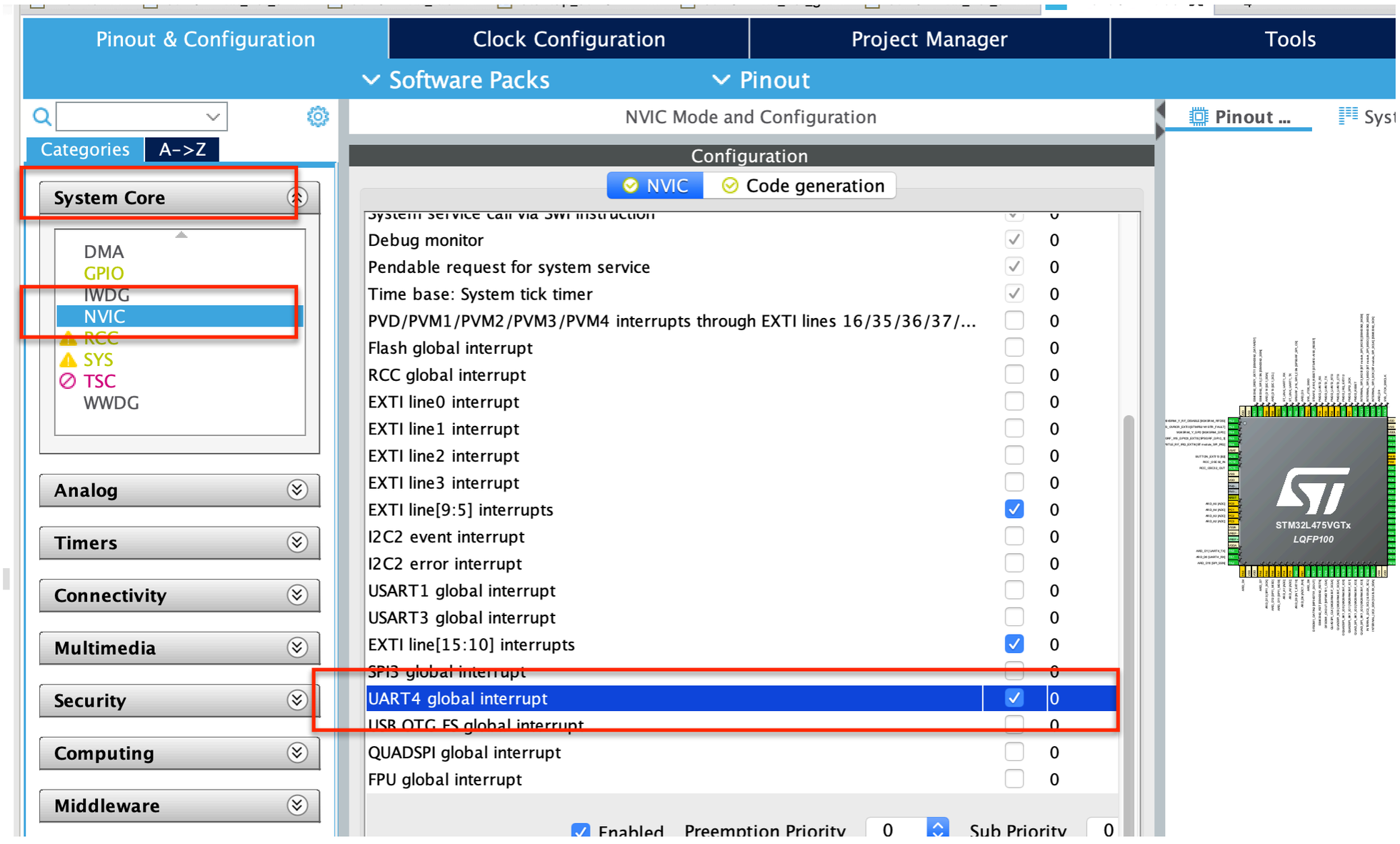
User Story 2 Code Polling - Part 2

```
102
103         // Confirm they are the same
104         if (*ptx_buf != *prx_buf) {
105             char buf[100];
106             snprintf(buf, sizeof(buf), "\r\nError: 0x%02x != 0x%02x\r\n", *ptx_buf, *prx_buf);
107             HAL_UART_Transmit(&huart1, (uint8_t *)&buf, sizeof(buf), 100);
108             return;
109         }
110
111         // Point to next location
112         ptx_buf++;
113         prx_buf++;
114
115     } while (ptx_buf < (tx_buf + sizeof(tx_buf)));
116 }
117
```

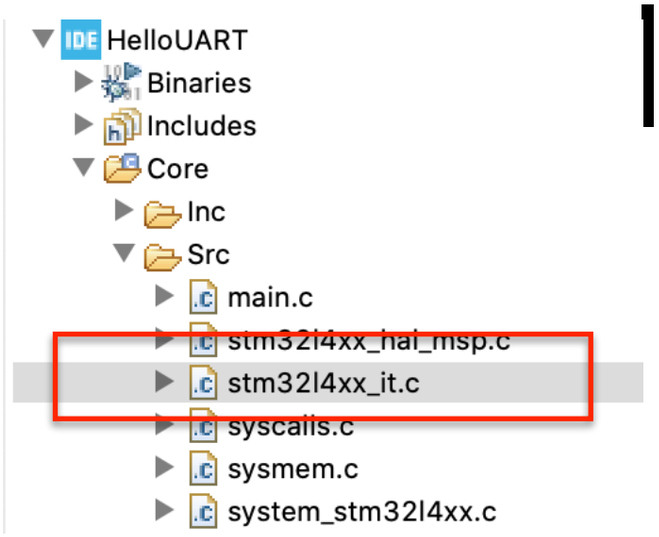
User Story 2
Running Code - Polling

```
Options: 1=polling, 2=interrupt, 3=DMA
cmd> ............................
Polling
Options: 1=polling, 2=interrupt, 3=DMA
cmd>
```

User Story 3 Code Interrupt - Part 1



User Story 3 Code Interrupt - Part 2




```
235 /**
236   * @brief This function handles UART4 global interrupt.
237   */
238 void UART4_IRQHandler(void)
239 {
240   /* USER CODE BEGIN UART4_IRQn 0 */
241
242   /* USER CODE END UART4_IRQn 0 */
243   HAL_UART_IRQHandler(&huart4);
244   /* USER CODE BEGIN UART4_IRQn 1 */
245
246   /* USER CODE END UART4_IRQn 1 */
247 }
248
```

User Story 3 Code Interrupt - main.c - Part 3

```
117
118 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
119
120     // Show we made it here
121     char ch = 'T';
122     HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 100);
123 }
124
125 static int do_interrupt_done = 0;
126
127 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
128
129     // Show we made it here
130     char ch = 'R';
131     HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 100);
132
133     // Confirm we received all data
134     HAL_UART_Transmit(&huart1, (uint8_t *)rx_buf, sizeof(rx_buf), 100);
135
136     // All done!
137     do_interrupt_done = 1;
138 }
```

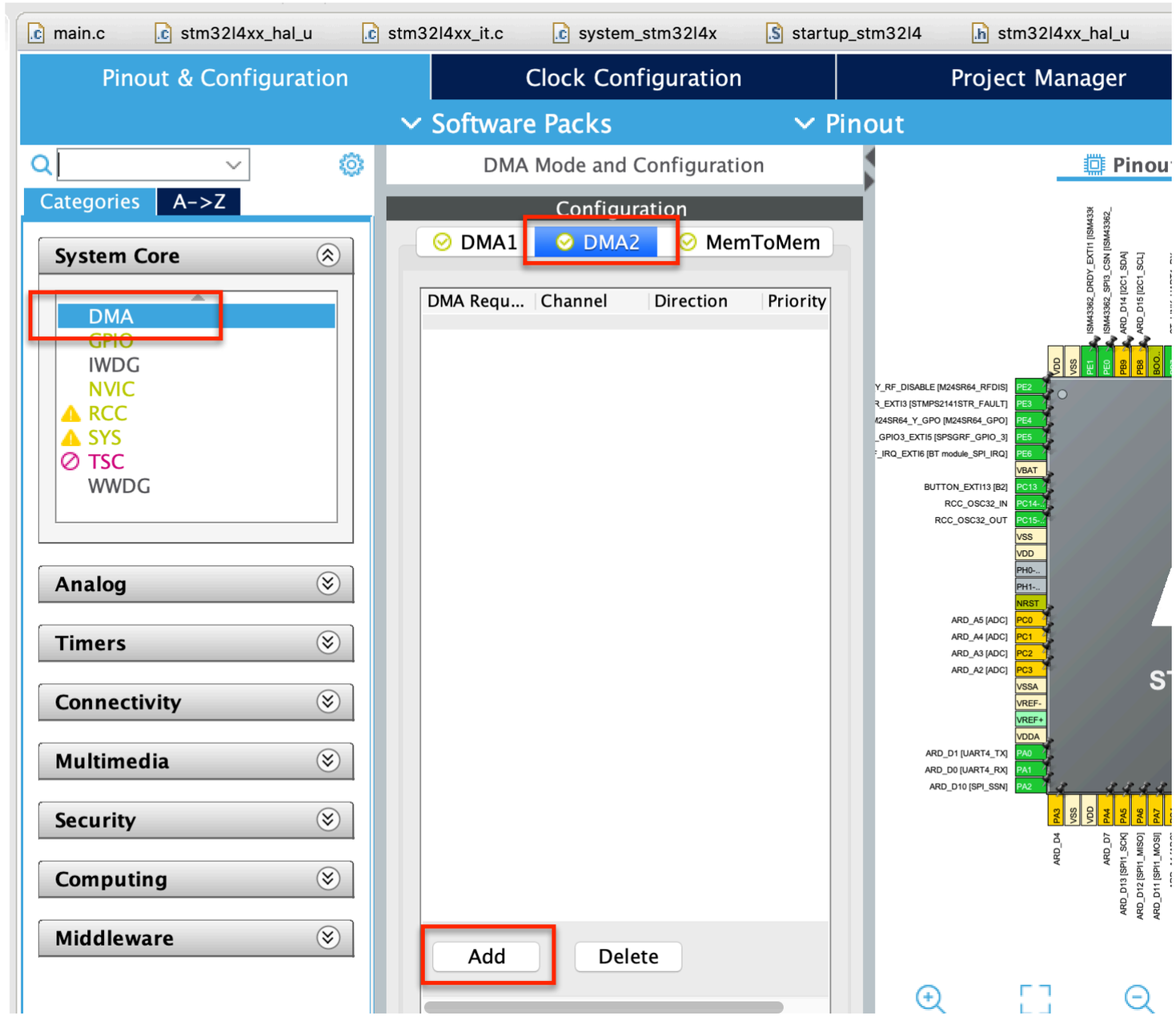
User Story 3 Code Interrupt - main.c - Part 4

```
139
140 static void do_interrupt() {
141
142     // Set rx buf to known pattern
143     for (int i = 0; i < sizeof(rx_buf); i++ ) rx_buf[i] = '?';
144
145
146     // Let UART1 know we are active
147     char ch = '.';
148     HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 100);
149
150     // Clear flag to know whne interrupts done
151     do_interrupt_done = 0;
152
153     // Receive the buffer using interrupts
154     //HAL_UART_Receive(&huart4, (uint8_t *)prx_buf, 1, 100);
155     HAL_UART_Receive_IT(&huart4, (uint8_t *)rx_buf, sizeof(rx_buf));
156
157     // Send the complete buffer using interrupts
158     //HAL_UART_Transmit(&huart4, (uint8_t *)ptx_buf, 1, 100);
159     HAL_UART_Transmit_IT(&huart4, (uint8_t *) tx_buf, sizeof(tx_buf));
160
161     // Interrupt Tx and RX does the work...we just wait
162     while (!do_interrupt_done) {
163         // Let UART1 know we are active
164         char ch = '~'; //In flight
165         HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 100);
166         HAL_Delay(100);
167     }
168
169 }
170
```

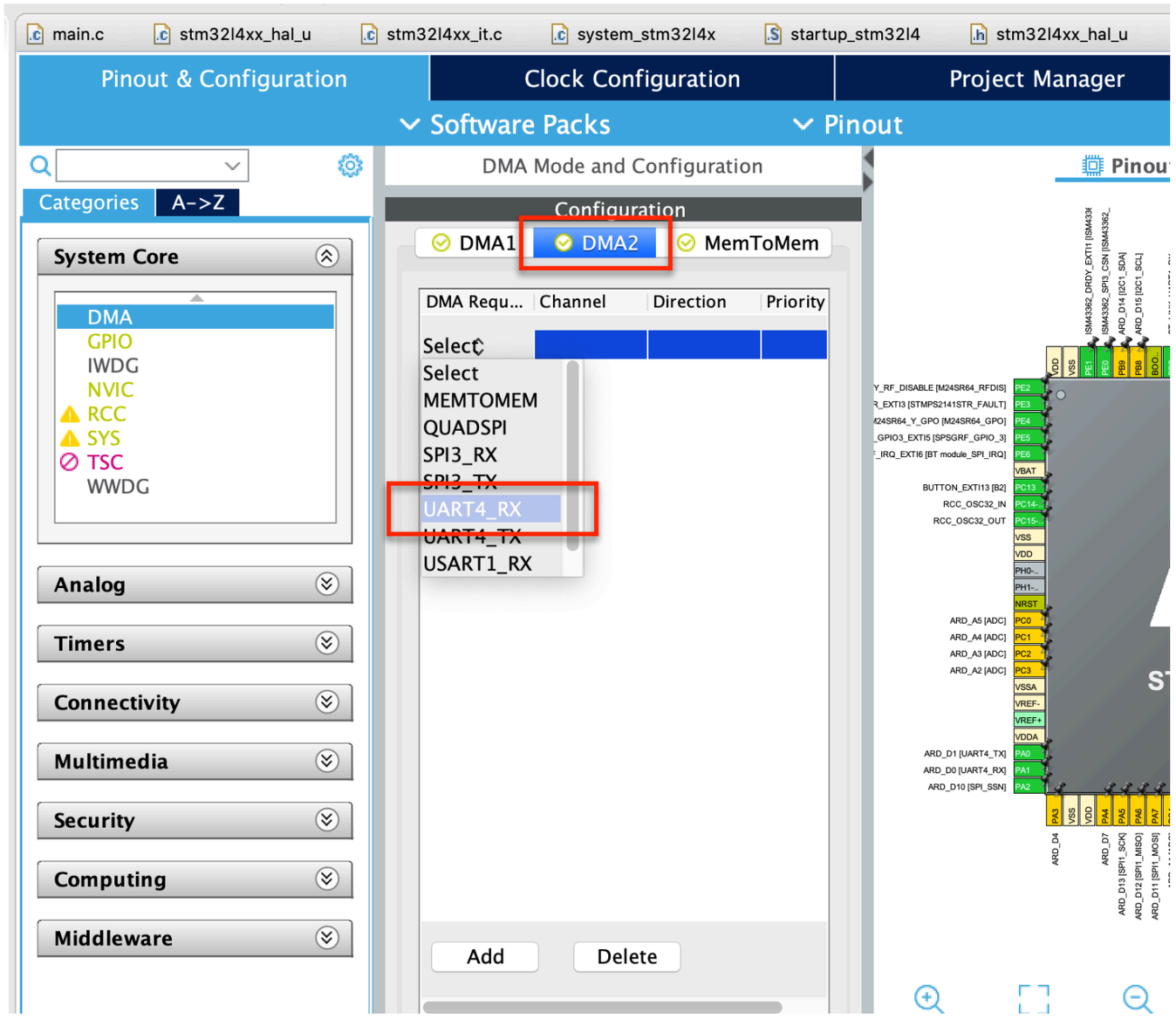
User Story 3
Running Code - Interrupts

```
Options: 1=polling, 2=interrupt, 3=DMA
cmd> .~Rabcdefghijklmnoqarstuvwzyz
T
Interrupt
Options: 1=polling, 2=interrupt, 3=DMA
cmd>
```

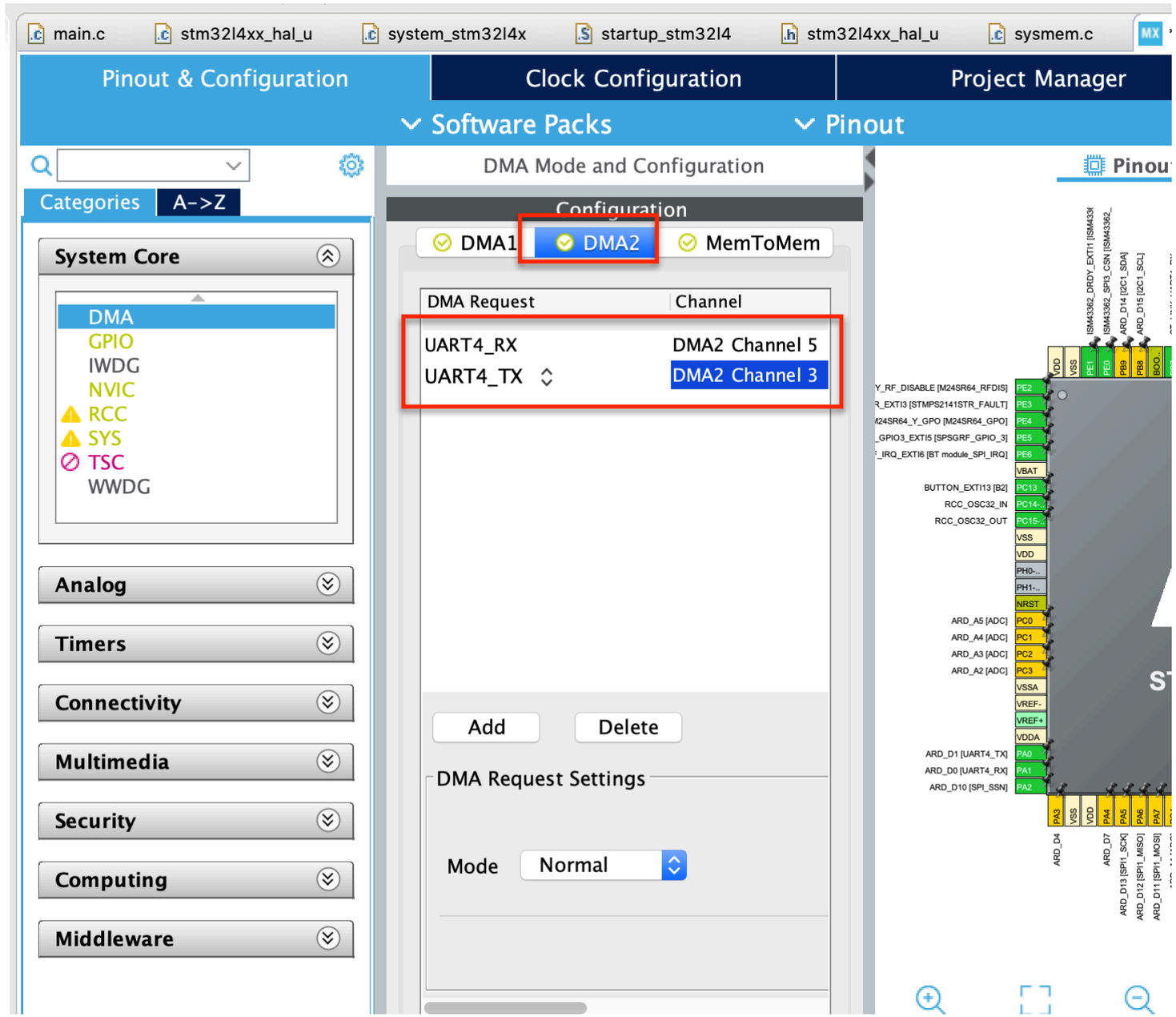
User Story 4 Code
DMA - Part 1 - DMA2



User Story 4 Code
DMA - Part 2 - UART4_RX



User Story 4 Code
DMA - Part 3 - UART4_TX



User Story 4 Code
DMA - Part 4 - main.c

After
Code
Generation

main.c

```
44  /* Private variables ----------------------
45  DFSDM_Channel_HandleTypeDef hdfsdm1_channel
46
47  I2C_HandleTypeDef hi2c2;
48
49  QSPI_HandleTypeDef hqspi;
50
51  SPI_HandleTypeDef hspi3;
52
53  UART_HandleTypeDef huart4;
54  UART_HandleTypeDef huart1;
55  UART_HandleTypeDef huart3;
56  DMA_HandleTypeDef hdma_uart4_rx;
57  DMA_HandleTypeDef hdma_uart4_tx;
58
59  PCD_HandleTypeDef hpcd_USB_OTG_FS;
60
61  /* USER CODE BEGIN PV */
```

User Story 4 Code DMA - Part 5 - main.c

NOTE: Uses Sample Tx/RX Complete Interrupts As With Interrupt Version

```
173
174 static void do_dma() {
175
176     // Set rx buf to known pattern
177     for (int i = 0; i < sizeof(rx_buf); i++ ) rx_buf[i] = '?';
178
179     // Let UART1 know we are active
180     char ch = '.';
181     HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 100);
182
183     // Clear flag to know when DMA done
184     do_interrupt_done = 0;
185
186     // Receive the buffer using DMA
187     HAL_UART_Receive_DMA(&huart4, (uint8_t *)rx_buf, sizeof(rx_buf));
188
189     // Send the complete buffer using DMA
190     HAL_UART_Transmit_DMA(&huart4, (uint8_t *) tx_buf, sizeof(tx_buf));
191
192     // Interrupt Tx and RX does the work...we just wait
193     while (!do_interrupt_done) {
194         // Let UART1 know we are active
195         char ch = '~'; //In flight
196         HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 100);
197         HAL_Delay(100);
198     }
199
200 }
```

User Story 4
Running Code - DMA

```
Options: 1=polling, 2=interrupt, 3=DMA
cmd> .~TRabcdefghijklmnoqarstuvwzyz

DMA
Options: 1=polling, 2=interrupt, 3=DMA
cmd>
```

Summary

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