Blue Sky Transmission Protocol

btcp.h btcp.c buart.h buart.c

Contents

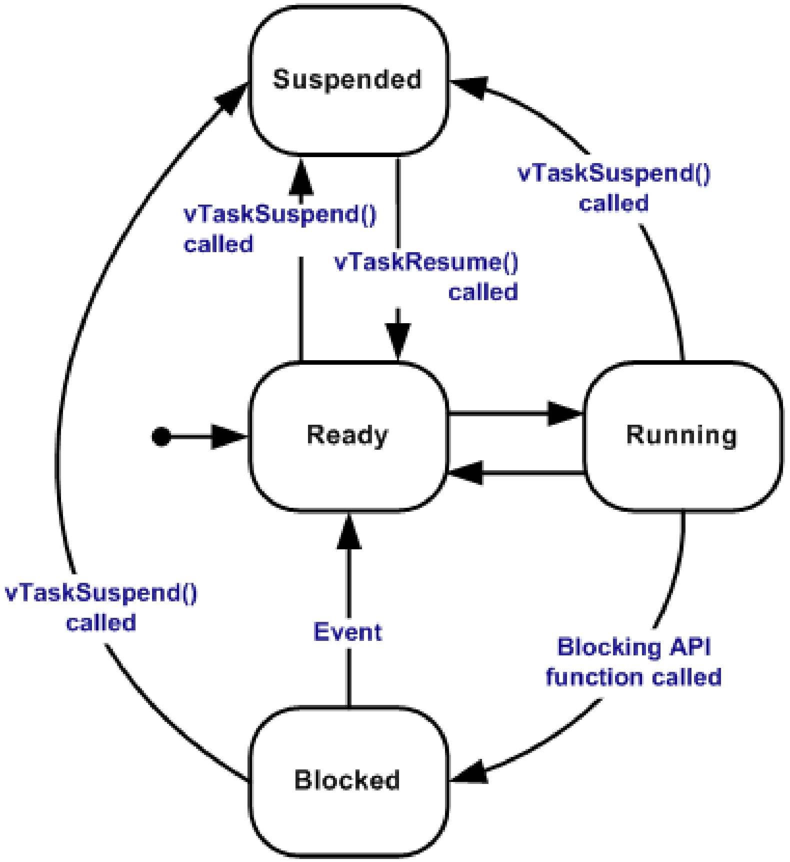
- Terminologies
- Purpose of Blue Sky Protocol
- How to use Blue Sky Protocol
- Software architecture
    - Motherboards
    - UART Hub

- Uncertainties and improvements
- GitHub discussion

Terminologies

In FreeRTOS:

- Task = Thread
- Block = sleep
  - Task blocks for 100 ms -> thread sleeps for 100 ms
  - a FreeRTOS function that blocks is NOT the same as a blocking function!
- Sender ID = TCP ID
- Might use the nomenclature interchangeably



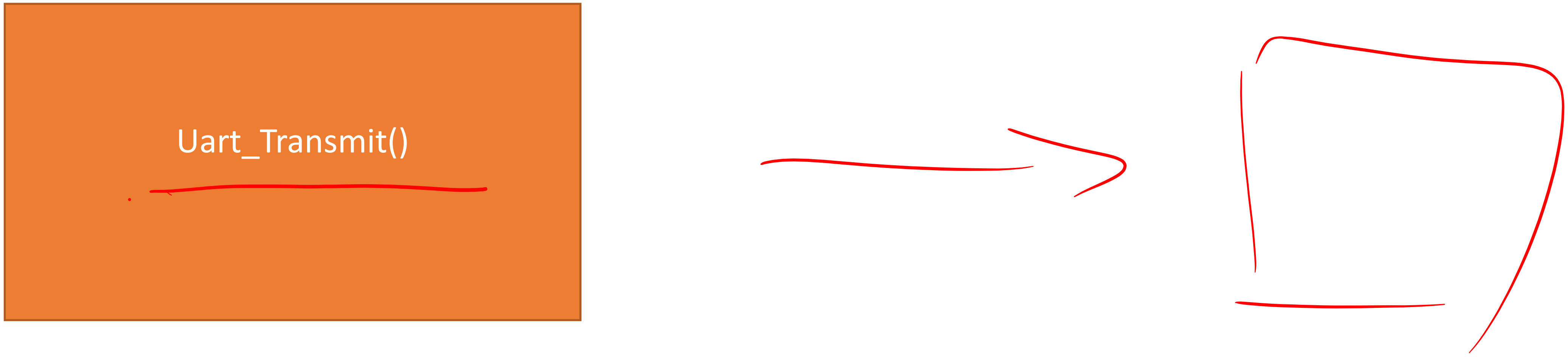
Valid task state transitions

So why use Blue Sky Protocol

- Supports Parallel Programming
  - Safely send messages from multiple threads through the same UART ports
  - Ensures no race conditions
- Includes CRC (cyclic redundancy check)
  - Rejects data that have been corrupted
- Additional functionality like Sequence and Acknowledgement Number
  - Allows us to check if a packet has been lost
  - Haven’t see this used in our system

So why use Blue Sky Protocol

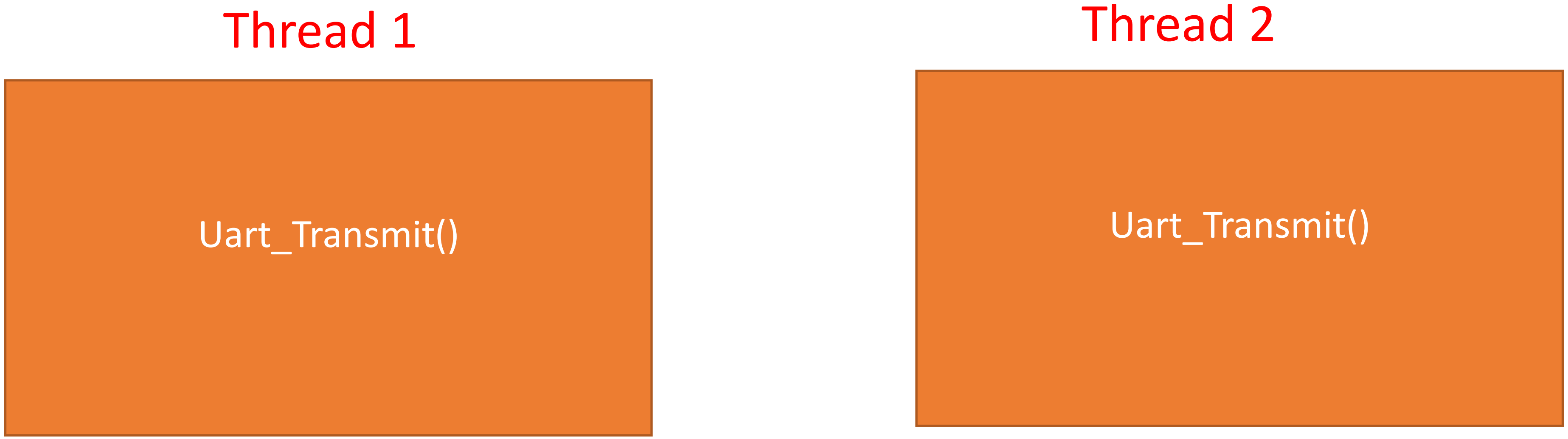
Single Thread



Uart_Transmit()

So why use Blue Sky Protocol

- Race condition!



Thread 1

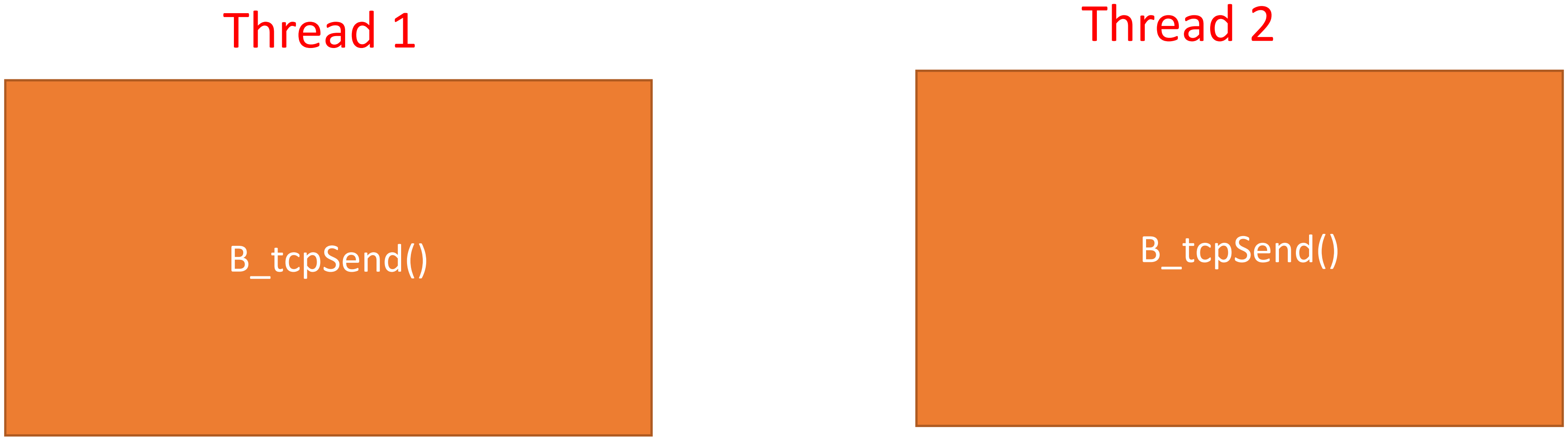
Uart_Transmit()

Thread 2

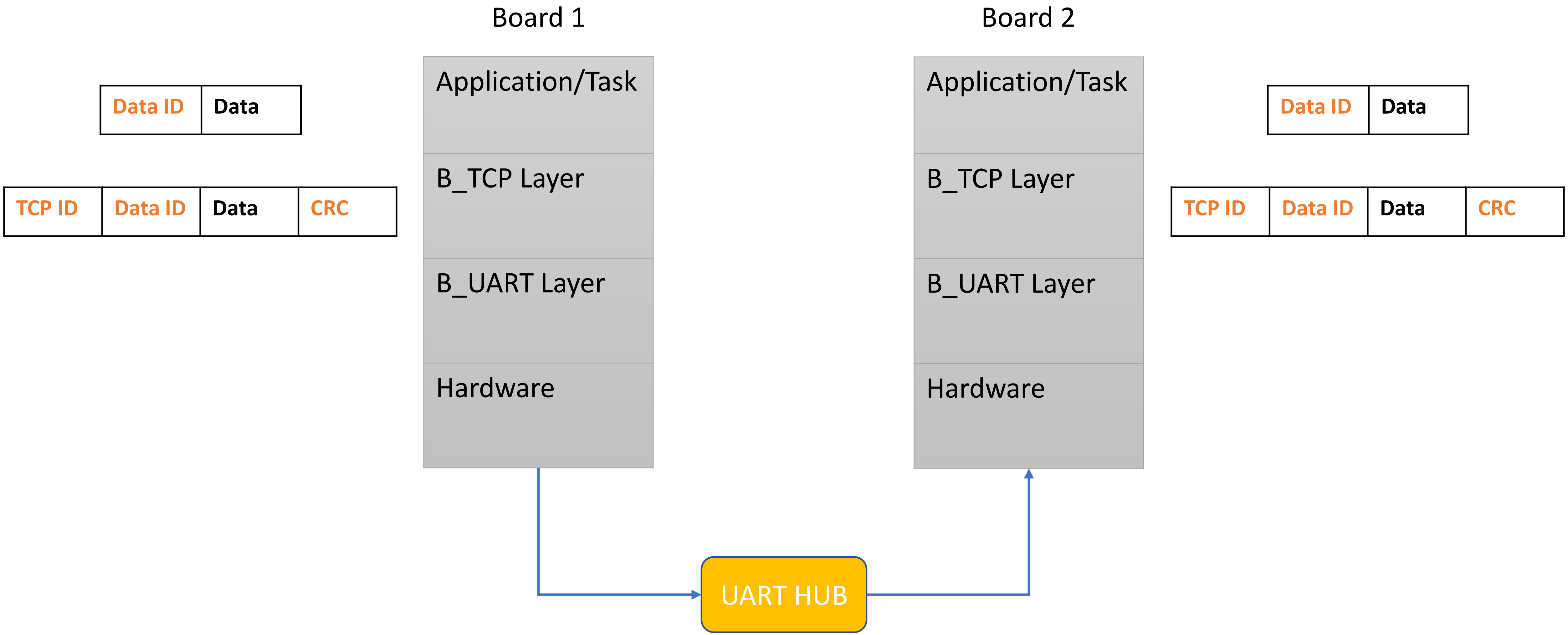
Uart_Transmit()

So why use Blue Sky Protocol

- Use B_tcpSend() instead



Blue Sky Protocol Layering



Board 1

Data ID	Data

TCP ID	Data ID	Data	CRC

Application/Task

B_TCP Layer

B_UART Layer

Hardware

UART HUB

Board 2

Application/Task

B_TCP Layer

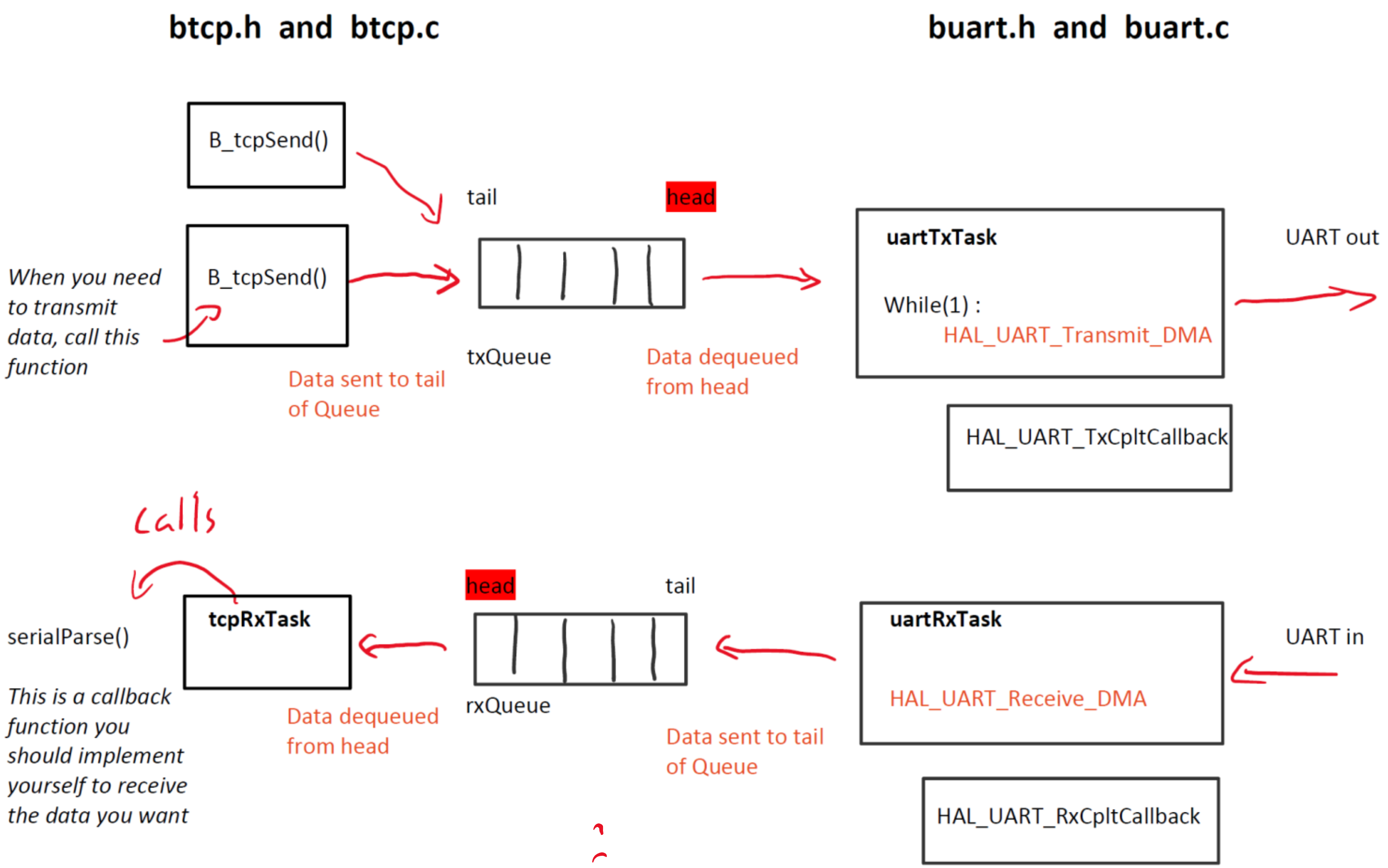
B_UART Layer

Hardware

Data ID	Data

TCP ID	Data ID	Data	CRC

Motherboard Software Architecture



When you need to transmit data, call this function

Data sent to tail of Queue

Data dequeued from head

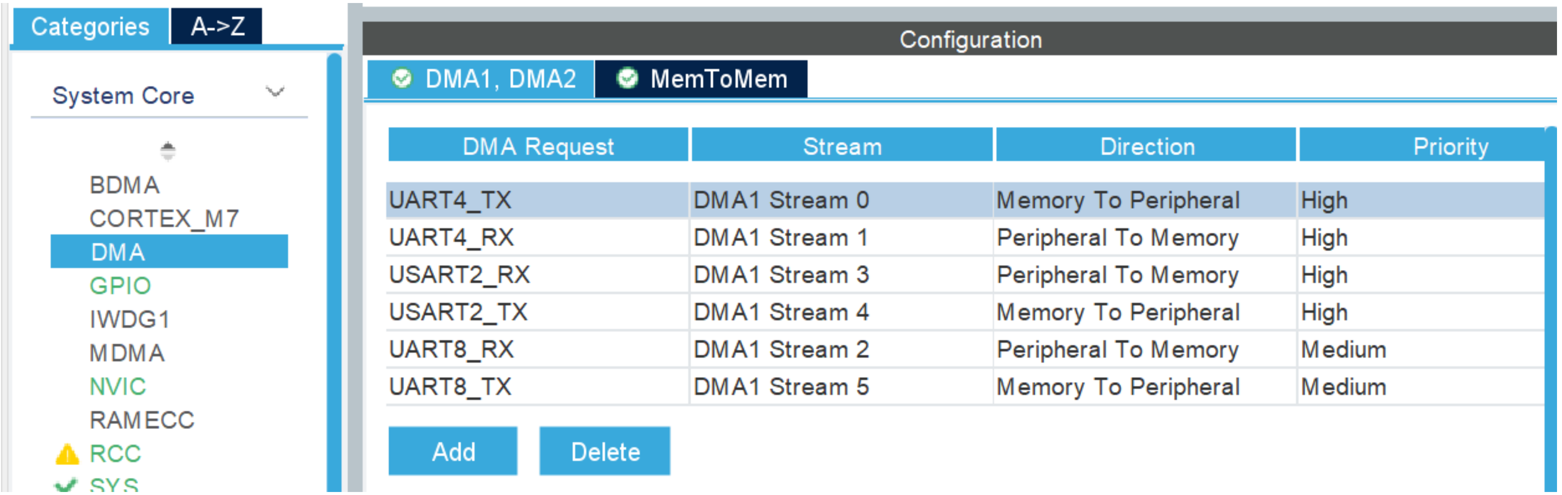
This is a callback function you should implement yourself to receive the data you want

Data dequeued from head

Data sent to tail of Queue

CubeMX setup

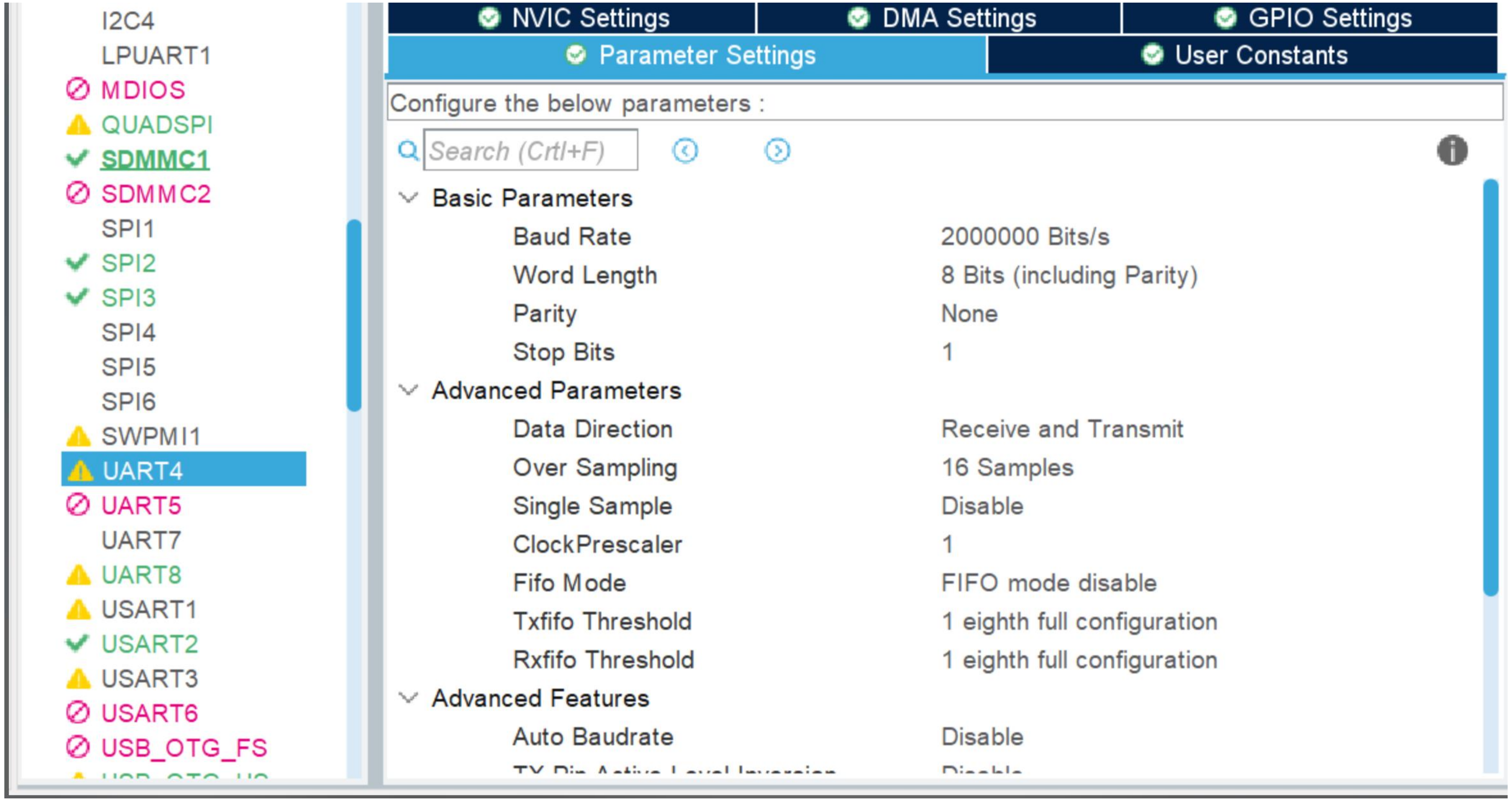
1. Set up FreeRTOS: [set up FreeRTOS from cubeMX - Google Docs]
    1. Make sure the
2. Set the UART Ports you want to use through the Blue Sky Protocol to DMA and set priority to high according to the screenshot



DMA Request	Stream	Direction	Priority
UART4_TX	DMA1 Stream 0	Memory To Peripheral	High
UART4_RX	DMA1 Stream 1	Peripheral To Memory	High
USART2_RX	DMA1 Stream 3	Peripheral To Memory	High
USART2_TX	DMA1 Stream 4	Memory To Peripheral	High
UART8_RX	DMA1 Stream 2	Peripheral To Memory	Medium
UART8_TX	DMA1 Stream 5	Memory To Peripheral	Medium

Configure the UART baud rate and other parameters

For MotherBoards: UART 4 is used for board to board transmission. USART2 is used for debugging



Configure the below parameters :

Basic Parameters

Baud Rate	2000000 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

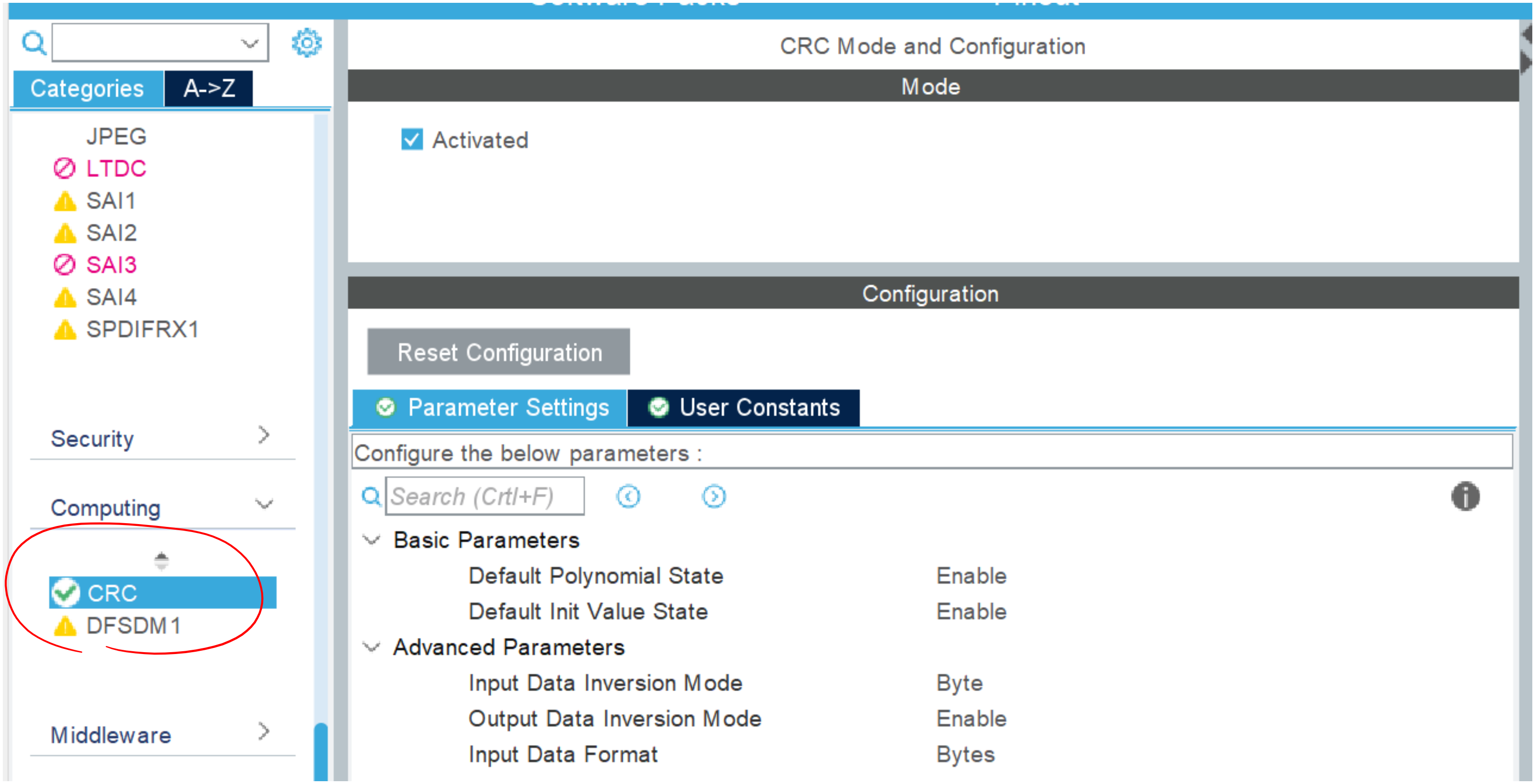
Data Direction	Receive and Transmit
Over Sampling	16 Samples
Single Sample	Disable
ClockPrescaler	1
Fifo Mode	FIFO mode disable
Txfifo Threshold	1 eighth full configuration
Rxfifo Threshold	1 eighth full configuration

Advanced Features

Auto Baudrate	Disable
TX Pin Active Level Inversion	Disable

How to Use

3. Set up CRC



CRC Mode and Configuration

Mode

☑ Activated

Configuration

Configure the below parameters :

Basic Parameters

Default Polynomial State	Enable
Default Init Value State	Enable

Advanced Parameters

Input Data Inversion Mode	Byte
Output Data Inversion Mode	Enable
Input Data Format	Bytes

Code setup

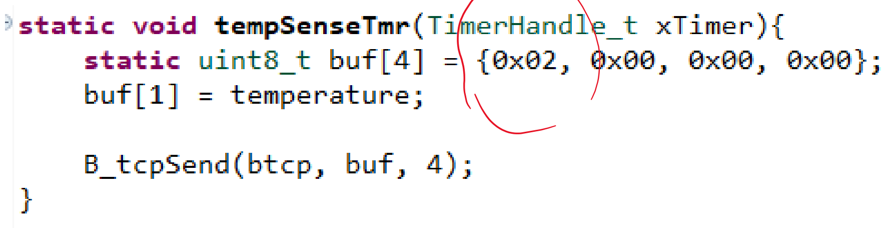
More detailed instructions: https://www.notion.so/blueskysolar/Using-BlueSky-s-UART-Protocol-7013663e777e43378d4e90f64c27ab88

1. In main.h, #define the TCP ID (ex: for MCMB, `#define TCP_ID 0x03`)
2. In code: set up the btcp and buart handles using B_uartStart() and B_tcpStart()
    1. This will create buart TX and RX task, and a btcp RX task
3. To send: call B_tcpSend()
4. To receive: Implement your own serialParse()
    1. Check for TCP ID (board ID)
    2. Check for Data ID

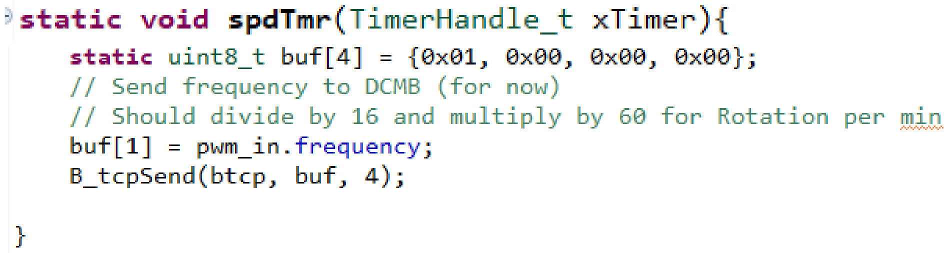
Example

Sending side (MCMB)

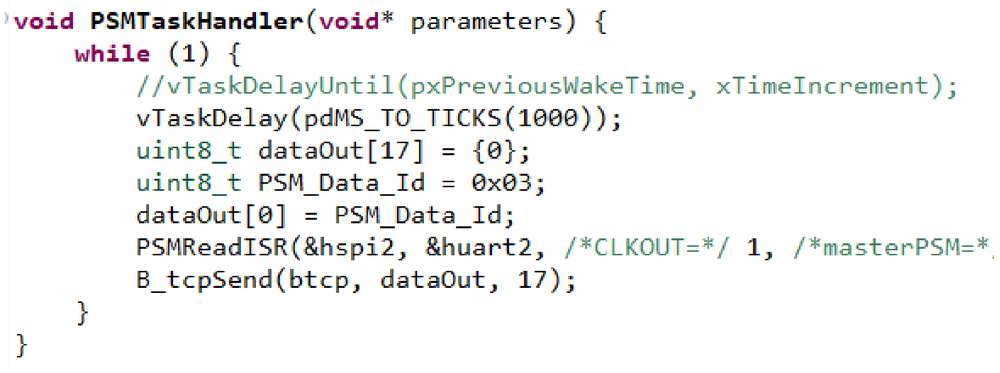
Data_ID



```
static void tempSenseTmr(TimerHandle_t xTimer){
    static uint8_t buf[4] = {0x02, 0x00, 0x00, 0x00};
    buf[1] = temperature;

    B_tcpSend(btcp, buf, 4);
}
```

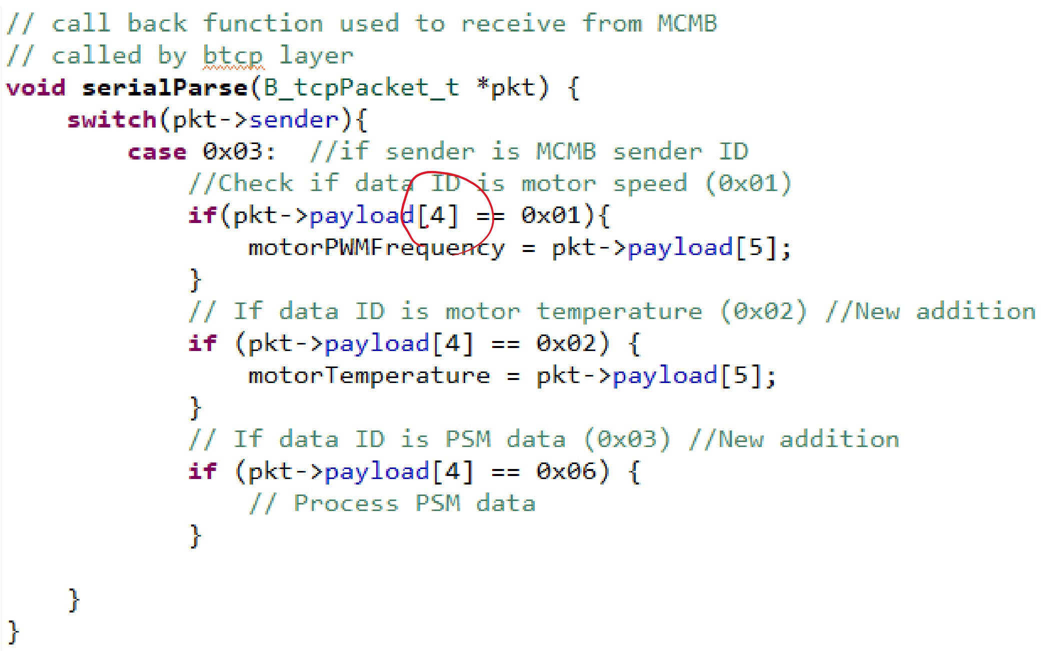


```
static void spdTmr(TimerHandle_t xTimer){
    static uint8_t buf[4] = {0x01, 0x00, 0x00, 0x00};
    // Send frequency to DCMB (for now)
    // Should divide by 16 and multiply by 60 for Rotation per min
    buf[1] = pwm_in.frequency;
    B_tcpSend(btcp, buf, 4);

}
```



```
void PSMTaskHandler(void* parameters) {
    while (1) {
        //vTaskDelayUntil(pxPreviousWakeTime, xTimeIncrement);
        vTaskDelay(pdMS_TO_TICKS(1000));
        uint8_t dataOut[17] = {0};
        uint8_t PSM_Data_Id = 0x03;
        dataOut[0] = PSM_Data_Id;
        PSMReadISR(&hspi2, &huart2, /*CLKOUT=*/ 1, /*masterPSM=*
        B_tcpSend(btcp, dataOut, 17);
    }
}
```

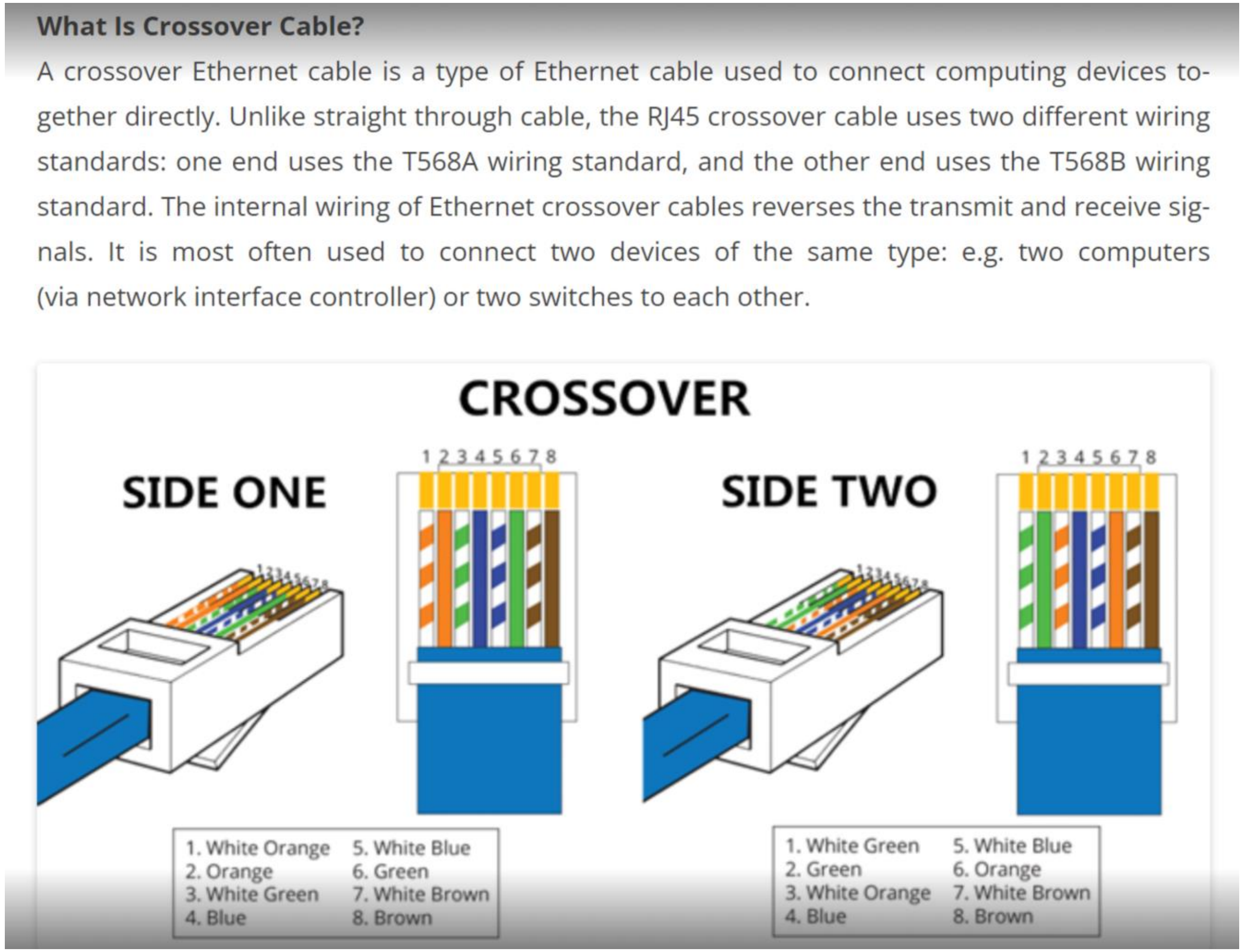
Receiving side (DCMB)



```
// call back function used to receive from MCMB
// called by btcp layer
void serialParse(B_tcpPacket_t *pkt) {
    switch(pkt->sender){
        case 0x03:  //if sender is MCMB sender ID
            //Check if data ID is motor speed (0x01)
            if(pkt->payload[4] == 0x01){
                motorPWMFrequency = pkt->payload[5];
            }
            // If data ID is motor temperature (0x02) //New addition
            if (pkt->payload[4] == 0x02) {
                motorTemperature = pkt->payload[5];
            }
            // If data ID is PSM data (0x03) //New addition
            if (pkt->payload[4] == 0x06) {
                // Process PSM data
            }


    }
}
```

What cable to use between boards

- Not standard ethernet cable!



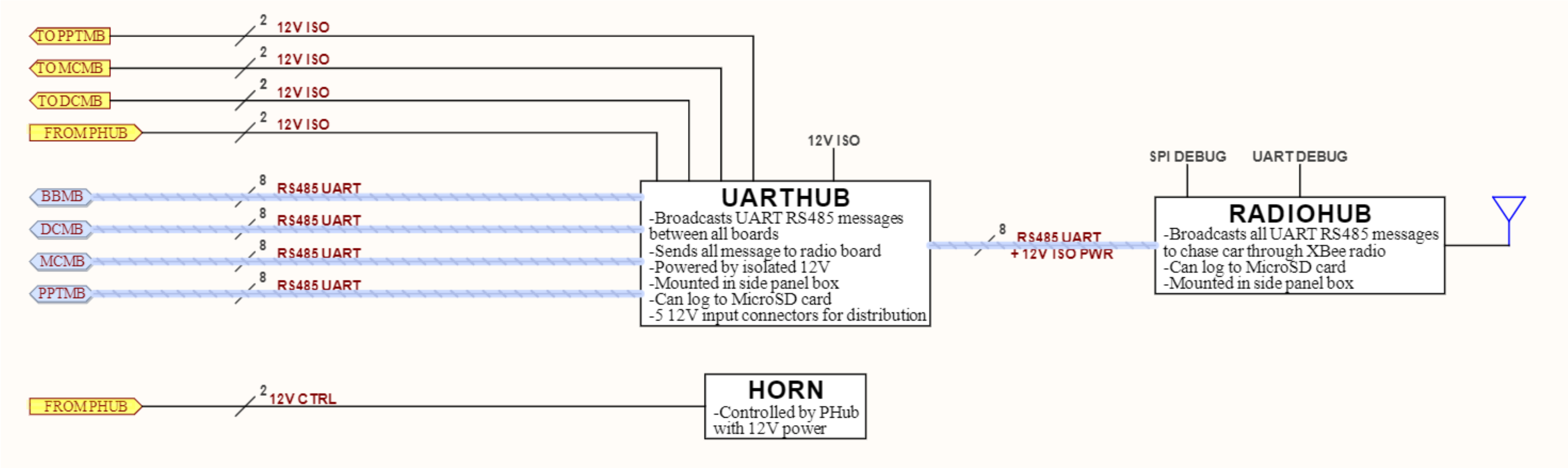
What Is Crossover Cable?

A crossover Ethernet cable is a type of Ethernet cable used to connect computing devices together directly. Unlike straight through cable, the RJ45 crossover cable uses two different wiring standards: one end uses the T568A wiring standard, and the other end uses the T568B wiring standard. The internal wiring of Ethernet crossover cables reverses the transmit and receive signals. It is most often used to connect two devices of the same type: e.g. two computers (via network interface controller) or two switches to each other.

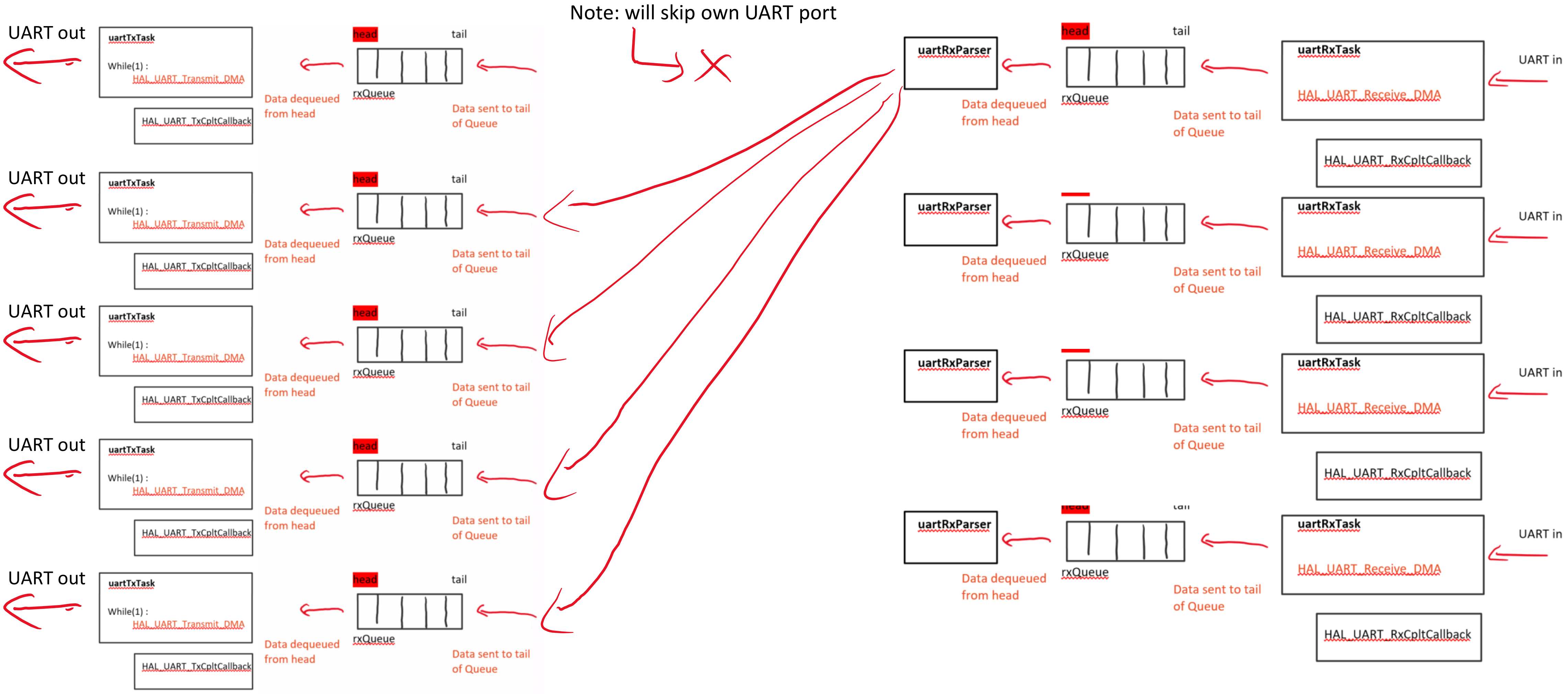
1. White Orange
2. Orange
3. White Green
4. Blue
5. White Blue
6. Green
7. White Brown
8. Brown

1. White Green
2. Green
3. White Orange
4. Blue
5. White Blue
6. Orange
7. White Brown
8. Brown

UART HUB


TO PPTMB
TO MCMB
TO DCMB
FROM PHUB
2
12V ISO
BBMB
DCMB
MCMB
PPTMB
8
RS485 UART
12V ISO
UARTHUB
-Broadcasts UART RS485 messages between all boards
-Sends all message to radio board
-Powered by isolated 12V
-Mounted in side panel box
-Can log to MicroSD card
-5 12V input connectors for distribution
RS485 UART
+ 12V ISO PWR
SPI DEBUG
UART DEBUG
RADIOHUB
-Broadcasts all UART RS485 messages to chase car through XBee radio
-Can log to MicroSD card
-Mounted in side panel box
FROM PHUB
12V CTRL
HORN
-Controlled by PHub with 12V power

UART HUB Software Architecture



Note: will skip own UART port

UART out
uartTxTask
While(1) :
HAL_UART_Transmit_DMA
HAL_UART_TxCpltCallback
Data dequeued from head
head
tail
rxQueue
Data sent to tail of Queue

UART out
uartTxTask
While(1) :
HAL_UART_Transmit_DMA
HAL_UART_TxCpltCallback
Data dequeued from head
head
tail
rxQueue
Data sent to tail of Queue

UART out
uartTxTask
While(1) :
HAL_UART_Transmit_DMA
HAL_UART_TxCpltCallback
Data dequeued from head
head
tail
rxQueue
Data sent to tail of Queue

UART out
uartTxTask
While(1) :
HAL_UART_Transmit_DMA
HAL_UART_TxCpltCallback
Data dequeued from head
head
tail
rxQueue
Data sent to tail of Queue

UART out
uartTxTask
While(1) :
HAL_UART_Transmit_DMA
HAL_UART_TxCpltCallback
Data dequeued from head
head
tail
rxQueue
Data sent to tail of Queue

uartRxParser
Data dequeued from head
head
tail
rxQueue
Data sent to tail of Queue
uartRxTask
HAL_UART_Receive_DMA
HAL_UART_RxCpltCallback
UART in

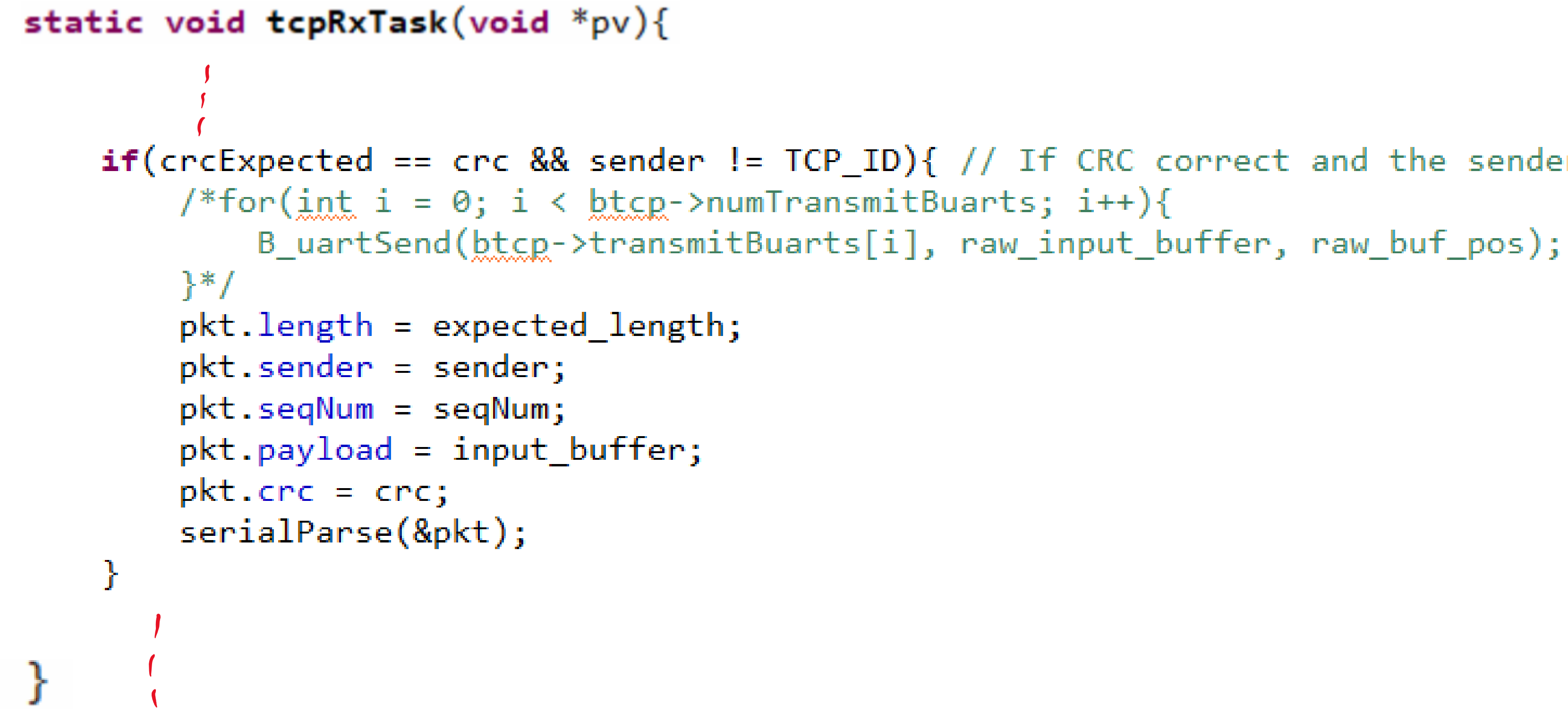
uartRxParser
Data dequeued from head
rxQueue
Data sent to tail of Queue
uartRxTask
HAL_UART_Receive_DMA
HAL_UART_RxCpltCallback
UART in

uartRxParser
Data dequeued from head
rxQueue
Data sent to tail of Queue
uartRxTask
HAL_UART_Receive_DMA
HAL_UART_RxCpltCallback
UART in

uartRxParser
Data dequeued from head
head
tail
rxQueue
Data sent to tail of Queue
uartRxTask
HAL_UART_Receive_DMA
HAL_UART_RxCpltCallback
UART in

Changes from last Gen

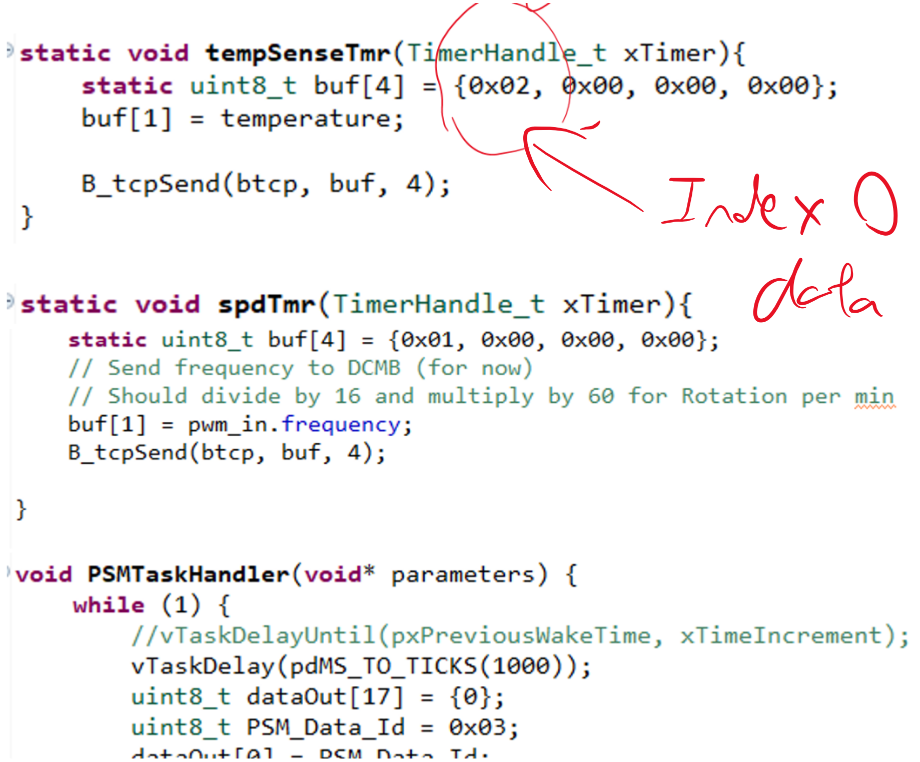
- Fixed some warnings, added comments for GEN11 btcp.c, btcp.h
- Commented out the Daisy Chain code (since we are not using it)



```
static void tcpRxTask(void *pv){

    if(crcExpected == crc && sender != TCP_ID){ // If CRC correct and the sender is not this motherboard
        /*for(int i = 0; i < btcp->numTransmitBuarts; i++){
            B_uartSend(btcp->transmitBuarts[i], raw_input_buffer, raw_buf_pos);
        }*/
        pkt.length = expected_length;
        pkt.sender = sender;
        pkt.seqNum = seqNum;
        pkt.payload = input_buffer;
        pkt.crc = crc;
        serialParse(&pkt);
    }

}
```

daisy chain
Commented out

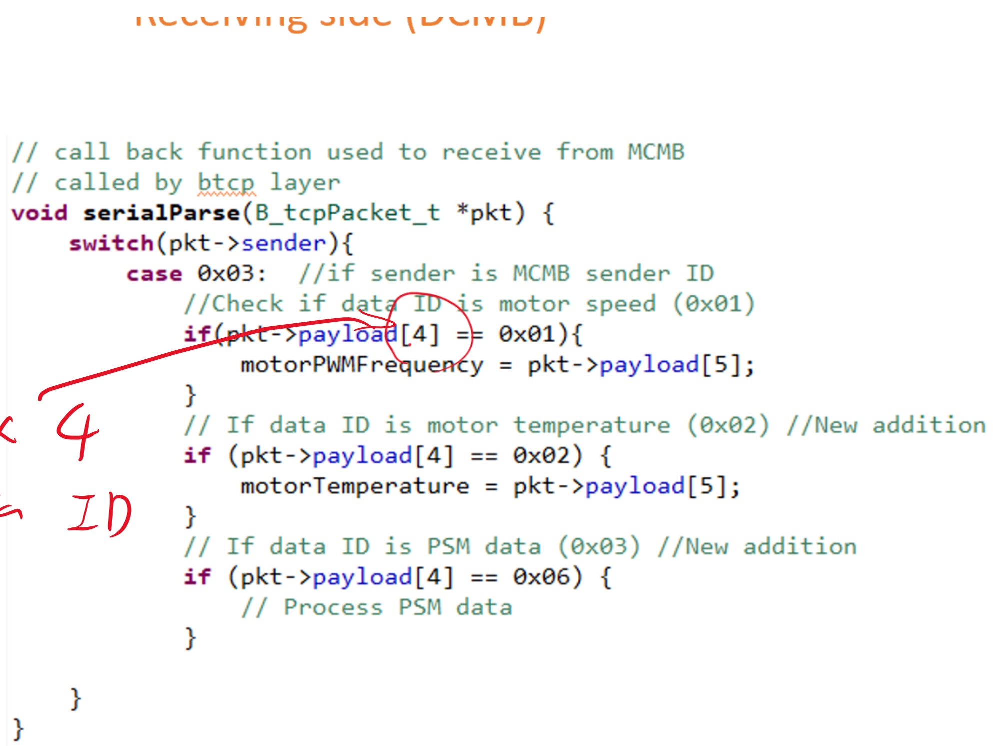
Uncertainties/Improvements



```
static void tempSenseTmr(TimerHandle_t xTimer){
    static uint8_t buf[4] = {0x02, 0x00, 0x00, 0x00};
    buf[1] = temperature;

    B_tcpSend(btcp, buf, 4);
}

static void spdTmr(TimerHandle_t xTimer){
    static uint8_t buf[4] = {0x01, 0x00, 0x00, 0x00};
    // Send frequency to DCMB (for now)
    // Should divide by 16 and multiply by 60 for Rotation per min
    buf[1] = pwm_in.frequency;
    B_tcpSend(btcp, buf, 4);

}

void PSMTaskHandler(void* parameters) {
    while (1) {
        //vTaskDelayUntil(pxPreviousWakeTime, xTimeIncrement);
        vTaskDelay(pdMS_TO_TICKS(1000));
        uint8_t dataOut[17] = {0};
        uint8_t PSM_Data_Id = 0x03;
        dataOut[0] = PSM_Data_Id;
```

Index 0 is data ID

Receiving side (DCMB)



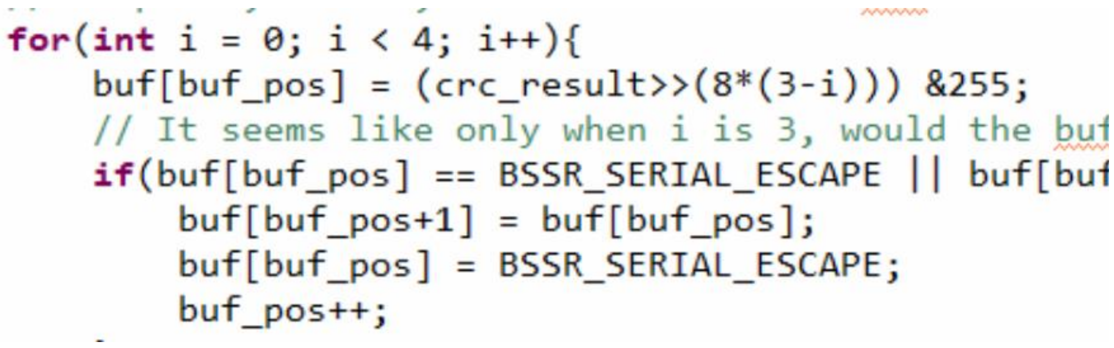
```
// call back function used to receive from MCMB
// called by btcp layer
void serialParse(B_tcpPacket_t *pkt) {
    switch(pkt->sender){
        case 0x03:  //if sender is MCMB sender ID
            //Check if data ID is motor speed (0x01)
            if(pkt->payload[4] == 0x01){
                motorPWMFrequency = pkt->payload[5];
            }
            // If data ID is motor temperature (0x02) //New addition
            if (pkt->payload[4] == 0x02) {
                motorTemperature = pkt->payload[5];
            }
            // If data ID is PSM data (0x03) //New addition
            if (pkt->payload[4] == 0x06) {
                // Process PSM data
            }


    }
}
```

Index 4 is data ID

Uncertainties/Improvements

- TCP_ID is defined in main.h (hard to find)
  - Pass it as parameter in main.c?

- CRC code works but looks strange. Some parts are useless?



```
for(int i = 0; i < 4; i++){
    buf[buf_pos] = (crc_result>>(8*(3-i))) &255;
    // It seems like only when i is 3, would the buf
    if(buf[buf_pos] == BSSR_SERIAL_ESCAPE || buf[buf
        buf[buf_pos+1] = buf[buf_pos];
        buf[buf_pos] = BSSR_SERIAL_ESCAPE;
        buf_pos++;
```

GitHub?

- Store all Btcp and buart files as a single instance on github
  - Not sure how this is done
- Might need to do some special setup in IDE to link some external repository