Table of Contents

[UART Functions 1](#_Toc472007001)

[UART Examples 2](#_Toc472007002)

[UART Functions 2](#_Toc472007003)

# UART Functions

bool init (int ID, int BAUDRATE);

Initialize the UART bus.

**ID** indicate which UART to be used (0 for UART0 and 4 for UART4, etc.).

**BAUDRATE** indicate which Baud rate will the device be communicating at. If the input is not one of the standard Baud rate, the Baud rate will default to 9600.

Return False if initialization fail.

Return True if initialization success.

int peek (void);

Check to see how many bytes are in the receiving buffer (if there are data available).

No Input

Return number of bytes available to read. This is an integer number from 0 to anything.

int send (uint8\_t \*output, int len);

Send binary data through this UART device.

**output** is a pointer to array of bytes to be sent.

**len** is the number of bytes to in **output** buffer.

Return 0 at all time. (Need to modify the function to provide more feedback).

int receive (uint8\_t \*output, int len);

Receive binary data through this UART device.

**input** is a pointer to array of bytes to store the receiving data.

**len** is the number of bytes to read (NOTE: never test what if len is more than number of bytes in UART device.

Return number of bytes received from UART.

int printf (const char \*format, …);

Send formatted string through UART. This function behave identical to printf except with a character limit of 96.

**input** is a pointer to array of bytes to store the receiving data.

**len** is the number of bytes to read (NOTE: never test what if len is more than number of bytes in UART device.

Return 0 at all time (Need to modify the function to provide more debug information).

# UART Examples

#include "UART.h"

int main (void) {

**// Initialize UART**

UART Serial;

Serial.init(4, 9600);

**// Transmit 3 string of text**

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

Serial.printf("Hello World! %d \n\r", i);

}

**// Transmit 3 bytes in uint8\_t**

uint8\_t\* tx\_buff = {0xF5, 0x77, 0x90};

Serial.send(tx\_buff, 3);

**// Check available bytes, and then read it. 0 is possible and will not give error.**

uint8\_t rx\_buff[96];

int bytes = Serial.peak();

int ret = Serial.receive(rx\_buff, bytes);

return 0;

}

# SPI Functions

bool init (void);

Initialize the SPI bus.

**No Input – To be developed into custom Chip-Selection mode.**

Return False if initialization fail.

Return True if initialization success.

bool setSPIMode (uint8\_t mode);

set the SPI bus to be operating on one of the 4 possible modes.

**mode** is a 1 byte numeric data for mode. Can be ranging from 0 to 3.

Return true if setting success.

Return false if setting fail.

bool setSPISpeed (uint32\_t speed);

set the SPI bus to be operating on different speed. This is the Clock Rate.

**speed** can be any numeric within 4 bytes limit.

Return true if setting success.

Return false if setting fail.

bool setSPIBitLength (uint8\_t bit);

set the Bit Length of the SPI bus. I don’t really know the implication of this value.

**bit** can be any 1 byte number.

Return true if setting success.

Return false if setting fail.

int transfer (uint8\_t \*tx, uint8\_t \*rx, int length);

Perform transfer action. Transmit the bytes and receive from the other side.

**tx** is a pointer to array of bytes to be sent.

**rx** is a pointer to array of bytes that hold the receiving data

**len** is the number of bytes to input output buffer.

Return 0 at all time. (Need to modify the function to provide more feedback).

# SPI Examples

#include "SPI.h"

int main (void) {

**// Initialize UART**

SPI Self;

if (!Self.init()) {

printf(“Error Initializing SPI Device. Is Device Tree Overlay completed?\n”);

return 1;

}

// 500kHz speed with SPI Mode 0. Please do checking like init, in this example I am being lazy.

setSPIMode(0);

setSPISpeed(500000);

setSPIBitLength(8);

uint8\_t tx[5] = {0x05, 0x44, 0xE5, 0xFF, 0xF1};

uint8\_t rx[5] = {0.};

transfer(tx, rx, 5);

// Print the returned result in Hex format.

for (ret = 0; ret < 5; ret++) {

if (rx[ret] != tx[ret]) {

printf("%.2X ", rx[ret]);

}

}

return 0;

}