2 Way COMM Manual

How it works: the logic behind the functioning of the two way comm system is to keep both the ground and balloon systems ready to receive data at all times while periodically breaking from what is the receive mode for

Setting up buffers

1. **Define buffer size**: At the beginning of the code two constants are defined in order to allocate in order to allocate memory for the buffer. These buffers are known as **maxi**, and **pL**.
   1. **maxi** is the maximum number of packets that can be stored in the TX and RX buffers at a time
   2. **pL** represents the maximum packet length.
2. **Understanding the buffer variables**:
   1. TXS and RXS and RXS are 2 dimensional arrays used to allocate memory space for the buffered data packets
   2. TX\_Buffer and RX\_Buffer and variables of class type Node and will be considered the array of actual packets them selves
      1. The Node class has two members content and size this will help with organizing transmitted and received data.
3. **En-queuing to the buffers**: because the buffers are arrays there are some indexing variables needed to properly place items into the buffers.
   1. In the **TX\_Buffer** we use tsp and tsi, and to check if the buffer is full we use the variable fulT.
      1. Example1: this is just the basic example of putting a 3 byte packet in the buffer this example Is from putting something on a buffer on the ground

//placing the message into the allocated memory space

TXS[tsp][0]=4;

TXS[tsp][1]=0;

TXS[tsp][2]=2;

updateT();//updating the status of the buffer

if(!fulT)//cheking if te buffer full

{//putting the message in the buffer

TX\_Buffer[tsi].content=TXS[tsp];

TX\_Buffer[tsi].size=3;

}

tsp=inc(tsp);//going to the next allocated space

fulT=enqueueT();//updating other indexes

* + 1. Example2: this example is putting something on the buffer from the balloon and collecting 3 bytes from the

TXS[tsp][0]=1;//Identifying itself at the beginning of the packet the first byte is for source

TXS[tsp][1]=10;//futher identification

//TXS[tsp][2]=

collect2(5,3,&TXS[tsp][2]);//getting 3 bytes from I^2c peripheral 5 to be transmitted updateT();//updating buffer status of the trasmission buffer

if(!fulT)//checking if the buffer

{

TX\_Buffer[tsi].content=TXS[tsp];//pushing the packet onto the buffer

TX\_Buffer[tsi].size=5;

//Serial.println("pushed1");

}

tsp=inc(tsp);//updating the allocated TX space index

fulT=enqueueT(); //updating the TX buffer status and indexes

* + 1. Example3: although the user doesn’t need to worry about being the one to en-queue data to the received data buffer this is how it is done in the code

emT=dequeueT();//updating the TX indexes for an effective data popping

if(CC110L.CheckReceiveFlag())// Checking if there is data recieved

{

int size=CC110L.ReceiveData(RXS[rsp]);// recieves the data

updateR();//updating the status of the buffer

if(!fulR)

{

RX\_Buffer[rsi].content=RXS[rsp];//pushing the recieved data into the queue

RX\_Buffer[rsi].size=size;// also recording its size

}

rsp=inc(rsp);//going to the next allocated space for recieved signals

fulR=enqueueR();// updating the RX indexes for an effective data pushing

CC110L.SetReceive();//setting back to recieve mode

Serial.println(size);

//Serial.println("got");

}

1. DE-queuing from buffer: de-queuing is primarily done by the user when a message that has been received and about to be processed for example
   1. De-queuing received messages:
      1. EXmple1: an example of a dequeue from the rx buffer in the balloon code updateR();//updatiing the space status on the Rx biffer

if(!emR)//if RX Buffer is not empty pops what is on it

{

//Serial.println("split");

if(( RX\_Buffer[rpi].content[0]==4))//4 is for request

source= RX\_Buffer[rpi].content[2];//getting the destination of the request

}

emR=dequeueR();//popping of the next item on RX buffer

* + 1. Example2: This example comes from the ground code and later revision will have the same method of received message retrieval for both the balloon and ground

updateR();//updatiing the space status on the Rx biffer

if(!emR)//if RX Buffer is not empty pops what is on it

{

//Serial.println("rx");

source= RX\_Buffer[rpi].content[0];

message=RX\_Buffer[rpi].content[2];

currentr=RX\_Buffer[rpi].content;

}

rsp=inc(rsp);//going to the next allocated space for received signals

fulR=enqueueR();// updating the RX indexes for an effective data pushing

The main Loop and the ISR and the radio modes

1. The Main Loop: The main loop of the program has two purposes
2. Servicing Radio modes
   1. At the start of the program the radio is set to receive mode and then it will always check for two things, first it checks if the TX Buffer is empty if it is not the radio is momentarily switched from receive mode to transmit mode and then the next item on the buffer is transmitted, afterwards the radio is switched back to receive mode.
   2. The next important radio task is to check if the radio has received any data packets and place the data packet into the RX\_buffer.
3. Checking Other flags: there are some timer and activity flags that will be set in the ISR , that will be shown later when these flags are set the 2 way come device will do a task such as transmitting a message from a peripheral or a command
4. ISR: the ISR is currently set to interrupt every ½ second it also has 2 sets of tasks
5. Set flags that are related to received data
   1. In this step data is usually removed from the rx buffer to be stored at a temporary location the first byte of the data packet is stored in a variable named source where it is used to trigger an activity in the ISR or in part 2 of the main loop
6. Set timer flags and does other periodic activities and updates the time since the program started by counting every time the program has been interrupted

August 8 flight example excerpts

|  |  |
| --- | --- |
| Main loop(excluding radio code ) | ISR |
| Part1: {radio code }  Part 2  **if(auto1==1**) //checking if there is a message from source flag  {  **auto1=0;// sets the flag back to zero**  Serial.print("light:");  byte temp[3];  //preparing to tranmit back in response to a recieved message  TXS[tsp][0]=1;//Identifying itself at the beginning of the packet the first byte is for source  TXS[tsp][1]=10;//futher identification  //TXS[tsp][2]=  collect2(5,3,&TXS[tsp][2]);//getting 1 byte frome preipherral 5 to be transmitted through I2C interface  updateT();//updating buffer status of the trasmission buffer  if(!fulT)//checking if the buffer  {  TX\_Buffer[tsi].content=TXS[tsp];//pushing the packet onto the buffer  TX\_Buffer[tsi].size=5;  //Serial.println("pushed1");  }  tsp=inc(tsp);//updating the allocated TX space index  fulT=enqueueT(); //updating the TX buffer status and indexes  } | **Part1: Received messages**  updateR();  if(!emR) {  if(( RX\_Buffer[rpi].content[0]==4  **source= RX\_Buffer[rpi].content[2];**  }  emR=dequeueR();  (notice that source is theflag tat was set when a message was recieved)  **if(source==1**)//checking the source for the request  {  source=0;// resetting the source  if(allow==1)  allow=2;  else allow=1;  }  **if(source==2**)//checking the source for the request  {  source=0;// resetting the source  if(allow==0)  allow=1;  else allow=0;  }  **Part2: timmer activities**  **if(isc%20==0&&allow==1)** {  /\*This is an example of a tranmission message that can be done in a Interrupt because it does not contain the use of the I2C sub system every 10 seconds\*/  TXS[tsp][0]=3;  TXS[tsp][1]=1;  TXS[tsp][2]=collect(3); //collecting data from SPI periferral enabled by pin 3  updateT();  if(!fulT)  {  TX\_Buffer[tsi].content=TXS[tsp];  TX\_Buffer[tsi].size=3;  //Serial.println("pushed");  }  tsp=inc(tsp);  fulT=enqueueT();  }  /\*\*\*below is an example of how to activate flags for periodic messages with function calls no allowed in an interrupt\*\*/  **if(isc%30==0&&allow==2)**  {  /\* this sets the flag for an automatic message in the main loop to activate every 15  seconds which must be set back to zero after the if statement of the flag is read  \*/  **auto1=1;//setting the flag**  }  isc++;//updating the time  seconds=isc/2; |

Peripheral communication

1. I2C: <http://arduino.cc/en/reference/wire>

Warning to communicate with an 12C slave the user must set up the statement inside the main loop never inside the ISR as show before this can be doene by setting up flags in the ISR then checking the flags in the main loop.

* 1. Collecting the data: to collect data via I2c sue the collect2() function provided .

byte collect2(int slave ID ,int data length, byte\* memory)

this function talks to an i2c peripheral with the given id and collects the byte into an array that the user provides if the requested data is only 1 byte it will be returned by the function because the function always returns the last byte it read from the peripheral otherwise all collected day is stored in the array given by user in the parameters.

Note part b of the main loop code given above

“collect2(5,3,&TXS[tsp][2]);”

To see the code in the slave that responds to this look in the I2C slave example 2 fIle

1. SPI: <http://arduino.cc/en/Reference/SPI>
   1. The spi code is not yet complete but it also features a way to communicate with although at this time one byte at a time use the collect(int expnum) this function as it stands takes in the pin number the slave select for a peripheral of an experiment is connected to and reads one byte from it the only active pins for now are pins 3-7 as pin 2 is dedicated to the transceiver

Below is a schematic of the controlling arduino connected to some I2c and spi slaves