Wireless File Transfer Program Design

**Project Leader:** Patrick Mamaid **Members:**Kevin Le

Kevin Hsieh

Fei Yan

# Table of contents

# Program states ………………………………………………………………………….. 4

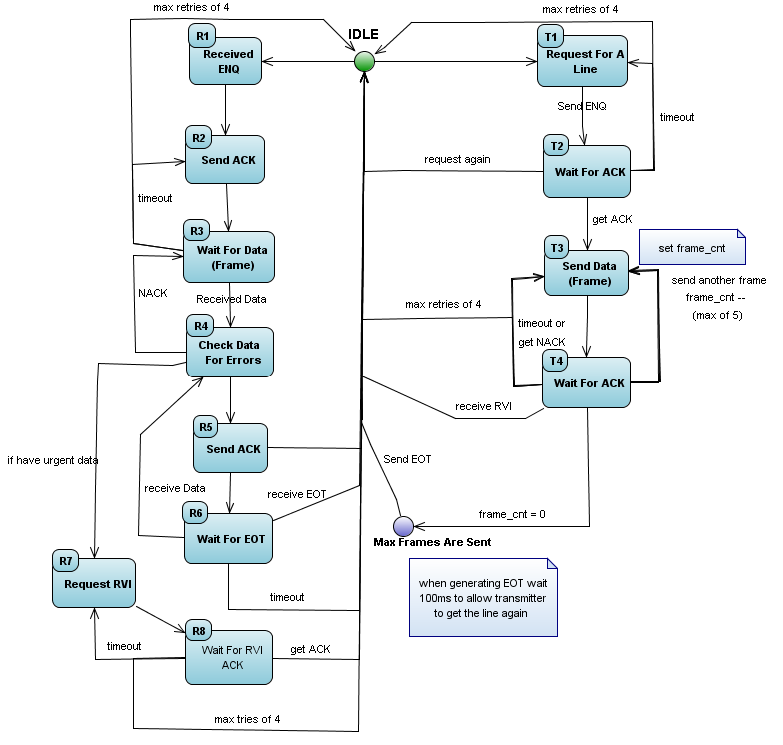
# Pseudocode Modular design ………………………………………………………..5

# Pseudocode Linear design ……………………………………………………………9

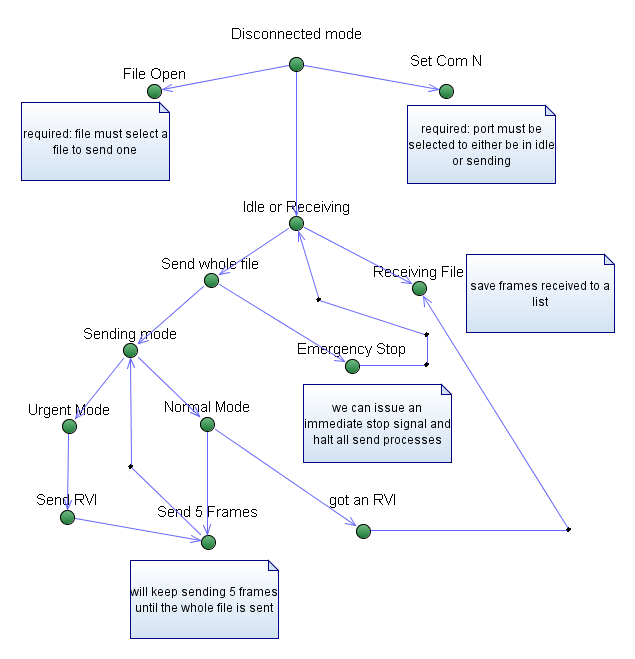
# Desired Program Design …………………………………………………………….11

# Design Aspect & Completion Scheduling …………………………………….12

# Program designed according to the following radio modem protocol:



# Program States:



Pseudo code modular design:

To indicate the beginning of our file we will embed a dc1 to the front and dc2 to indicate end of file in our data part of the frame.

Global variables:

Incomingfile 🡨 incoming file from wireless

Mode 🡨 urgent or normal

nFramesSent 🡨 initially zero

vector or list<string> frames 🡨 A list of frames to send.

programState 🡨disconnected, idle(connected to port), sending, receiving

whoHasLine 🡨me(our program), you(the other program over the wireless channel) , nobody

Open file(){//menu item file 🡪 open

Clear list of frames , nFramesSent to zero,   
 open file dialog window  
 user points to input file  
 break file into frames()  
 respond with file opened successfully or not

}

Connect(“COM”(n)){ //menu item file 🡪 connect 🡪 COM1 … COM10

Sets com n for reading & writing  
enable option for disconnecting in file>menu   
program now ready for outgoing & incoming connections  
start idleRecieving() loop  
set program to idle

}

emergencyStop(){//menu item file 🡪 send 🡪 emergency stop

finish up the last few frames to send  
when the ETX is set nframessent += 5  
do not send anymore frames   
do not flush the nframessent yet. User may want to recontinue sending

}

Break file into frames(){//works with open file method

Make a temp buffer 🡨 the text goes here   
insert a DC1 at beginning of the temp buffer  
insert a DC2 at the end of the temp buffer  
cut the temp buffer so that it is separated into frames  
add them to list of frames. //do not apply padding here, we will do it on the fly  
//since we are implementing RVI we will not know when other side will send it  
//when we do capture RVI during a send frame session, we may have   
//sent 3 frames

}

FileSave(){//menu item file 🡪 save

Check if file has been successfully downloaded from receiver  
(check the last frame received if it has DC2 in it)  
notify user about incomplete file   
save the file using a dialog box  
flush the incoming file buffer  
set program to idle

}

txFrames(){//sends 5 frames

counter 🡨 5(frames to send)  
frameRetryCount 🡨 4

check if program has line if(whohasline == nobody)  
send ENQ – send again if no response 4x retries  
get ACK set who has line to me

loop until counter Is zero  
send frame //including pad frames   
wait for ACK  
max wait 🡪 250ms   
do not decrement counter if we didn’t get ACK, decrement retrys--  
decrement counter—if we got ACK, reset frame retry count  
if we got RVI release loop  
  
when loop done send ETX  
increase nframessent global variable according to how many frames sent  
set who has line to nobody if RVI set it to you

}

fileSend(){ //menu item file 🡪 send 🡪 send file

loop until all frames have been sent  
if nobody has the line  
txFrames() send 5 frames  
if they have the line wait till they are done

}

idleRecieving(){

loop this method until disconnected

if program state is idle   
periodically check port for ENQ or RVI //however there is no reason to send RVI if nobody has the line right? Just check for it just in case.   
  
when we encounter ENQ set program state to receiving  
send out ACK **or send out RVI if program mode is urgent**wait for frames until ETX **or wait for ACK for our RVI**set program to idle or sending depending if sent RVI or not.

}

displayData(){

display on screen # of packets transferred  
 display bit error rate #  
 #ack sent  
 #nack sent  
 #enq sent  
 # frames sent

}

Pseudo code linear design

IDLE STATE

* Wants to send
* Call RequestLine()
* RequestLine()
* {
  + Send ENQ
  + Set retriesCount -> 4
  + WaitForACKRequest()
* }
* WaitForACKRequest ()
* {
  + Set ACK timer to 250ms
  + ReadData()
  + If
    - ACK received
      * SendData(Data)
    - ENQ received
      * Set ENQ timer to 1000ms before RequestLine() can be called again
      * ReadLine() //idle state
    - ACK timer elapses with neither a ACK response or ENQ response
      * If
        + retriesCount > 0

decrement retriesCount;

RequestLine()

* + - * + else //4 retries already

ReadLine() //idle state

* + - * Endif
  + Endif
  + End ACK timer
* SendData(Data)
* {
  + Send 1 frame of data (size = 2kb)
  + Set frameCount -> 5
  + Set retriesCount -> 4
  + WaitForACKFrame()
* }

WaitForACKFrame()

* {
  + Set ACK timer to 250ms
  + ReadData()
  + If
    - ACK received
      * If
        + frameCount > 0 //can send more frames

decrement frameCount

SendData(Data) //send more

* + - * + else //5 frames sent

ReadLine() //idle state

Set ENQ timer to 100ms before RequestLine() can be called again

* + - * Endif
    - NAK received //unsuccessful
      * SendData(Data)
    - RVI received
      * Send ACK
      * ReadLine() //idle state
      * Set ENQ timer to 100ms before RequestLine() can be called again
    - ACK timer elapses with neither a ACK, NAK or RVI response
      * If
        + retriesCount > 0

decrement retriesCount;

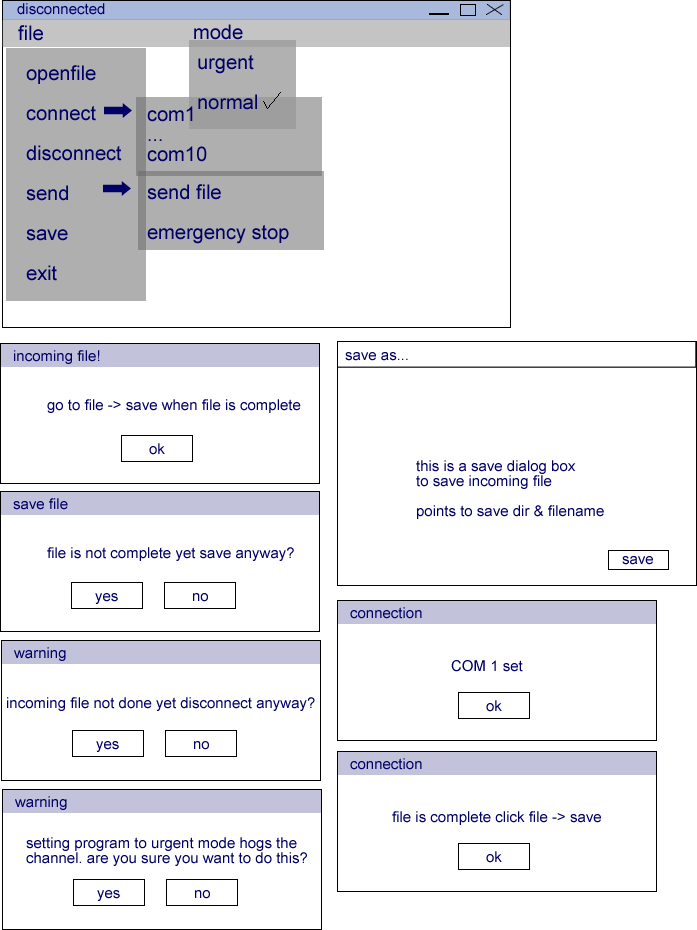
SendData(Data)

* + - * + else //4 retries already

ReadLine() //idle state

* + - * Endif
  + Endif
  + End ACK timer
* }

Desired program design:



Design aspect & completion scheduling

