Simplex Control and Message/ACK Exchange
ECE4305 Final Project Module 6

by

Renato Iida, Le Wang, Rebecca Copper

A Project Report
Submitted to Prof. Wyglinski
in partial fulfillment of the requirements of
ECE 4305
in
Electrical and Computer Engineering

Abstract

Put your abstract here.

Acknowledgements

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Put your introduction here

Chapter 2

Final prototype design

This chapter describe the evolution of frame format proposed in A. Also, describe the reason of each change based on better understand of need of others team and the problems in the development

2.1 FrameObj Evolution

The proposed frame format in A had all the necessary fields and some removed to keep the model simple The final frame format is shown in **??**. The figure 2.1 is the final state machine used to shows how the receiver will work

Frame Type	Receiver UE	Sender UE	Data Size	Header CRC	Data	Data CRC
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	1 to 234 Bytes	1 Byte

Table 2.1: Final Frame Format

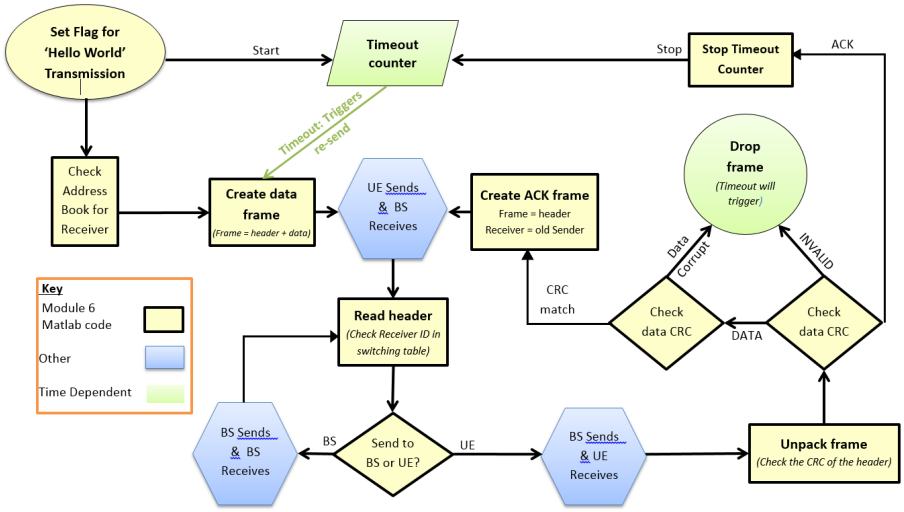


Figure 2.1: State Machine of transmission of package

Chapter 3

Experimental results

Put your Experimental results

Chapter 4

Conclusions

Put your conlusion here.

Bibliography

Appendix A

Design Proposal

COURSE DESIGN REPORT

Module 6: Simplex Control and Message/ACK Exchange

Renato Iida, Le Wang, Rebecca Cooper
Software Defined Radio System and Analysis

Abstract

In this project, we are going to create a simplex control mechanism to coordinate the communications between the nodes. First, we standardize the format of the frames. All the frames transmitted in the system have to be created based on the format. Then, we will utilize CRC-8 to verify the received frame and send the ACK frame if the verification is correct. Finally, we describe the steps and the interfaces with other groups to complete this project.

Problem Statement

In SDRSaA we are creating a Cellular Network with USRPs using Matlab. This network will be small, made up of only three users (UE 1, UE 2, and UE 3) connected to two base stations (BS 1 and BS 2). There will be three different standards used to communicate between nodes. As shown in Figure 1, UE 1 and UE 2 are associated with BS 1 and BS 1 connects to BS 2. BS 2 only has one client, UE 3. We are assuming all the communications between these nodes are on different channels. UE 1 talks to BS 1 through channel 1, UE2 talks to BS1 on channel 2. There will be at least 4 channels for the system. The USRPs run with Matlab cannot handle full duplex mode so each segment of communication will be in simplex mode. We are also assuming that the network will be static; none of the users will change base stations.

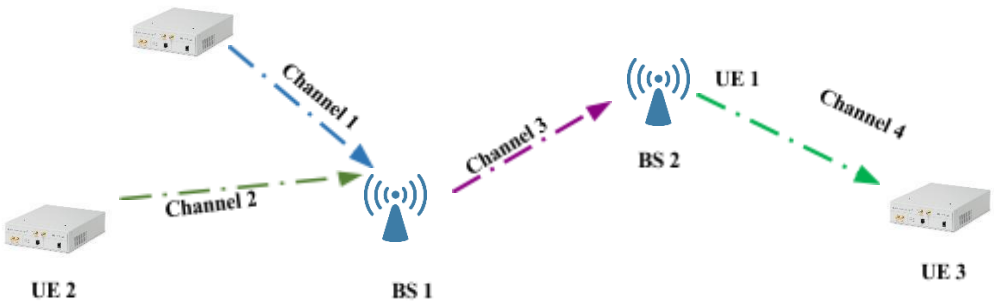


Figure 1: Overview of the cellular network architecture

As Team 6 we will be establishing end-to-end network resource allocation management as well as exchanging of control information and message/ACK forwarding between the UEs and the BS units. Other teams will be creating the three standards and the channels used to transmit messages as well as initializing the network and allocating the resources of each of the base stations.

Proposed Solution

To complete Module 6 and interface with the other teams we will need to create a simple MAC layer protocol, which should deal with routing protocol and checksum and create the initial messages that will be transmitted in the network.

Routing Protocol

There are two routing solutions that we came up with, static and dynamic, the one that we use will depend on the design choices of the other teams. The routing process will only be implemented by the base stations.

Static version

In this case, the frame leaves the sender, passes the base stations and reaches the destination receiver, through a static routing protocol. Based on the problem statement, we can draw static routing tables for both base stations. A draft of the routing tables is listed below:

Routing Table for BS1:

Src ID	Dst ID	Channel
UE 1	UE 2	Channel 2
UE 1	UE 3	Channel 3
UE 2	UE 1	Channel 1
UE 2	UE 3	Channel 3
UE 3	UE 1	Channel 1
UE 3	UE 2	Channel 2

Routing Table for BS2:

Src ID	Dst ID	Channel
UE 1	UE 3	Channel 4
UE 2	UE 3	Channel 4
UE 3	UE 1	Channel 3
UE 3	UE 2	Channel 3

For example, if UE 2 sends a frame to UE 3, first it will send the frame on channel 2, where only BS 1 can receive it. After receiving it, BS 1 will check the receiver address in the header, which appears as UE 3. Then BS 1 will send the frame on Channel 3. Similarly, only BS 2 can receive it. After checking the (Rcv ID) of the frame, BS 2 knows it is for UE 3, so BS 2 will send it on channel 4.

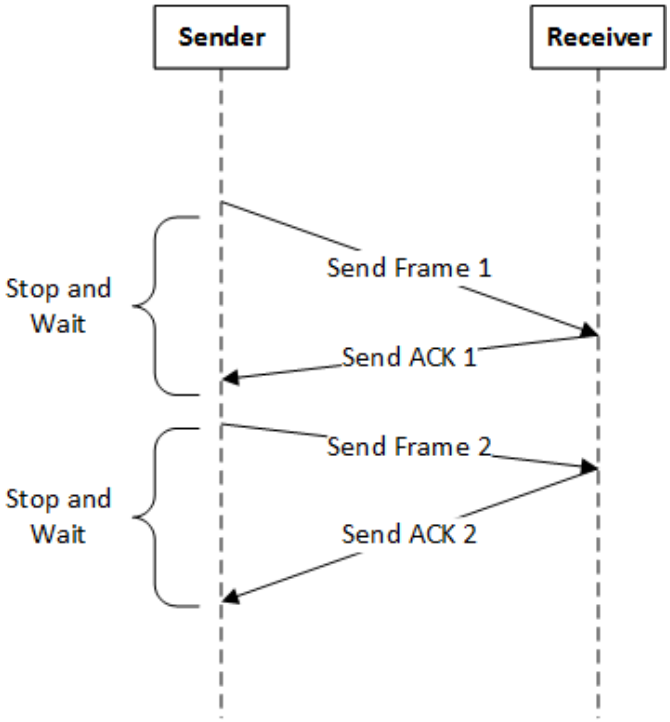
Dynamic Version

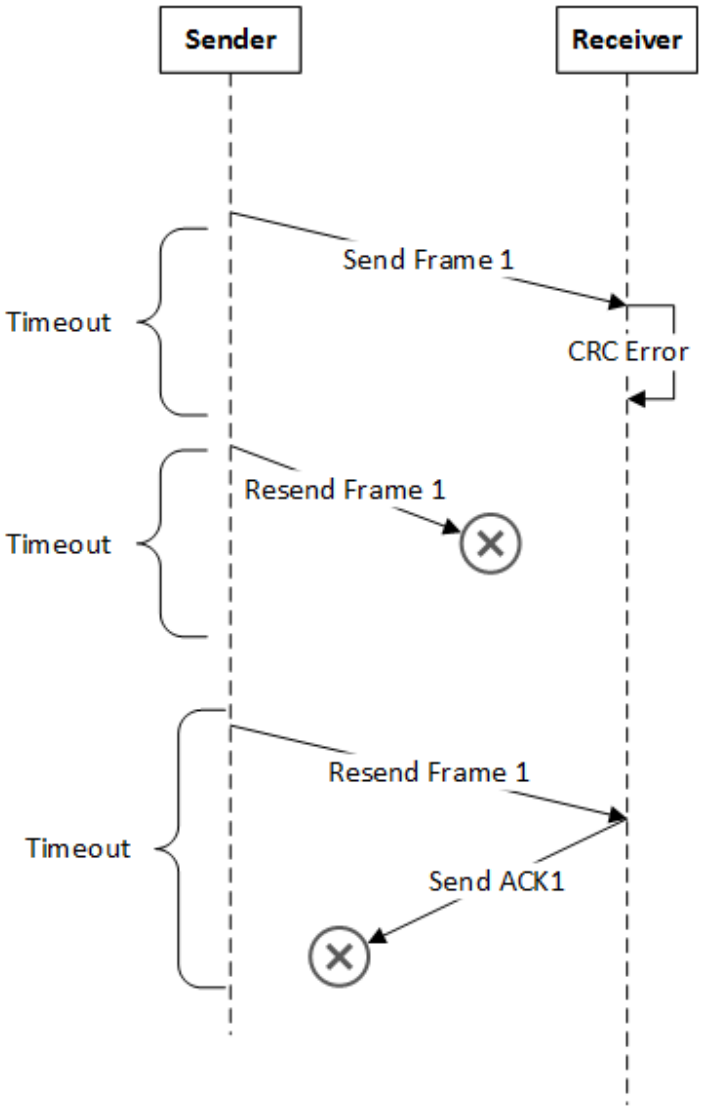
The dynamic algorithm is an open issue; the detail of this implementation will be discussed with team 4 and 5 in the development of this project.

Checksum Mechanism and ACK

We will build a checksum mechanism. Preliminarily, we adopt CRC-8 (Cyclic Redundancy Check). When a UE receives a data frame, it will perform CRC-8 on the payload. If the checksum returns a correct value, the receiver will reply with a control frame, ACK (acknowledge). CRC is simple and fast, however it does not have the ability to correct errors. The sender has to retransmit the frame if the CRC doesn't match instead of correcting locally. The whole system is in 'stop-and-wait' mode until the ACK is received by the UE.

'Stop-and-wait', also known as "positive acknowledgement with retransmission" is the fundamental technique to provide reliable transfer under unrealizable packet delivery system. In this mode, sender sends one data frame, waits for acknowledgement (ACK) from the receiver before proceeding to transmit next frame.





After broadcasting a frame, the sender will start a timer. If the sender cannot receive the ACK before timeout, it will retransmit. The reasons for such situation are listed below:

1. The frame reaches the receiver, however it is corrupted, which cannot pass the checksum mechanism. So no ACK will be replied, the sender will retransmit the frame after timeout.
2. The frame is lost during the transmission process. In other word, the receiver never received the frame. So there is no ACK, the sender will retransmit the frame after timeout.
3. The frame reaches the receiver and it is intact. The receiver did reply ACK, however this ACK frame is lost. The sender cannot receive ACK, so has to retransmit the frame after timeout.

In the worst case scenario there will be 6 transmissions between the initial transmission and the reception of the ACK by the UE. This is vital to determine how long to wait before the function will timeout and resend the frame.

Frame Format

When sending messages we will use at least two types of frames, the Data frame, and ACK frame. The frame type will distinguish between them. The data frame carries data and the ACK frame has no data payload. Additional types of frames can be added based on the needs of other teams.

The Rcv ID is the second part of the header so the base station only needs to check the first two bytes and it can make a decision on where to transfer the frame.

Based on the theory mentioned above, we design the format of frame as below:

Frame type	Rcv ID	Snd ID	SN	Data Size	CRC-8	Data (Payload)
1 byte	1 byte	1 byte	1 byte	2 bytes	1 byte	65,536 bytes

Rcv ID: The identification number of the receiver

Snd ID: The identification number of the sender.

SN: The sequence number is optional if it is necessary to deal with the situation when ACK is corrupt or lost.

Data size: Indicates the length of the payload.

CRC-8: The checksum result.

Data: The payload of the frame.

Implementation

We will create Matlab functions to complete our module's functionality.

- Function *Framing ( )*
    - Get the information such as Destination ID, Channel number from broadcast table.
    - Calculate the CRC of the data.
    - Create the header of the frame based on the information above and append the header to the payload to form the frame.
    - Pass the frame to the teams 1, 2 or 3 to send the data over the air.

- Function *Extract ( )*
    - Occurs after we receive the frame from teams 1, 2, or 3.
    - Extract the payload from it.
    - Call the Checksum function to calculate the CRC and check if it is correct.
    - Interface with Team 4 to send ACK.
- Function *Checksum ( )*
    - Calculate the CRC of the data and check it with the one from the header.
    - Return correct if they are equal.
- Function *Routing ( )*
    - Implement routing algorithm according to the routing table.

Prototype Evaluation Strategies & Logistics

- Milestone 1 (02/06):
  We will create a script in matlab that will create the message and it calls the function *Framing ( )*. This script will include placeholders for the functions that have not yet been developed from all the teams. The other milestones will replace them with functional code. It will make the integration processes more efficient and incremental with this team and the others. The *Framing ( )* function will be evaluated on whether it can create a frame of data from our placeholders. We will be working with the other teams to determine if our placeholders are appropriate.

- Milestone 2 (02/13):
  For this milestone we will add the verification of the received packet to the script developed in the Milestone 1. At this point we will have worked with team 4 to determine how the ACK will be sent and add that to our script.

- Milestone 3 (02/20):
  Here we will add the routing algorithm and interface to send and received the frame from team 1, 2, and 3. It will evaluate the end-to-end communication of the frame. The test case will follow the scenarios described in the Course design project listed below:

    • Extra-cellular communications between UEs:

      US1 -> BS1 -> BS2 -> US3

      US3 -> BS2 -> BS1 -> US1

      US2 -> BS1 -> BS2 -> US3

      US3 -> BS2 -> BS1 -> US2

    • Intra-cellular communications between UEs:

      US1 -> BS1 -> US2

      US2 -> BS1 -> US1

Appendix B

Source Code

Listing B.1: FrameObj Source Code

```
classdef FrameObj
    %FRAMEOBJ has 2 input configuraions; 4 inputs mean you are using the
    %frame requirements to create a FrameObj, 1 input means you are using
    %the bits
    %   In the first configuration (4 inputs) the first 3 inputs must be
    %   numbers and it is recommended that the constant properties of
    %   FrameObj are used to ensure accuracy. The last input depends on the
    %   frameType.
    %    -DATAFRAME: the last input must be a string and only the first 234
    %       characters will be included.
    %    -ACKFRAME: the last input does not matter but must exist.
    %    -POLLFRAME
    %    -REQFRAME
    %    -TABLEFRAME
    %   In the second configuration (1 input) the input must be a binary
    %   nx1 array with a supported frame type in the first byte. To ensure
    %   that there is no indexing outside the dimensions of the input array
    %   it is best if all inputs have a size of 1920x1
```

```
properties (Constant)
    %for sndID and rcvID
    IDBS1 = 100;
    IDUE1 = 101;
    IDUE2 = 102;
    IDUE3 = 203;
    IDBS2 = 200;
    IDALLUE = 000;

    % for frameTypes    %these numbers are chosen to resist flips and
    % shifts. It is the most important that frameType is correct as
    % very wrong frames can be dropped based on frameType
    DATAFRAME = 240;    %1111 0000
    ACKFRAME  = 255;    %1111 1111
    POLLFRAME = 202;    %1100 1010
    REQFRAME  =  83;    %0101 0011
    TABLEFRAME = 15;    %0000 1111
    INVALID = 0;        %0000 0000

    % for classUse
    ENCODE = 1;
    DECODE = 2;

    CHUE1BS1 = 1;
    CHUE2BS1 = 2;
    CHBS1BS2 = 3;
    CHUE3BS2 = 4;

    CRCOK = 1;
    CRCFAIL = 2;

    ACKRECEIVED = 3;
    TIMEOUT = 4;

    MAXDATA = 234; % Probably the only constant that has meaning
    MAXBYTES = 240;
end
```

```
properties
    classUse    %Identifies which configuration we are in
    frameType   %Identifies the type of frame that is being used
    rcvID       %The identification number of the destination receiver
    sndID       %The identification number of the sender.
    data        %The data field (cuts off after more than 234 bytes)

end
properties (Dependent)
    dataSize    %Indicates the length of the payload in bytes.
    header      %The array of the frame header with hCRC8
    hCRC8       %CRC-8 code verfication of the header field
    dCRC8       %CRC-8 code verfication of the data field
    frameArray  %The frame as an n*1 array
end

methods
    %This function sets the inputs to the property functions below.
    %Those functions define the actual properties, this is where we
    %call those functions.
    function obj = FrameObj(inputType,inputrcvID,inputsndID,inputData)

        % create a FrameObj with 4 inputs from the frame requirements
        if nargin == 4
            obj.classUse = inputType;
            %test if the frame type is valid
            obj.frameType = inputType;
            obj.rcvID = inputrcvID;
            obj.sndID = inputsndID;
            obj.data = inputData;

        %create a FrameObj with 1 array of bits
        elseif nargin == 1
            %Not sure if we need to do this. It might just be a
            %reminder that inputType is not the data that is being used
            %by the class properties in this case.
```

```
            bitwise = inputType;

            %size check
            [size_in, ¬] = size(bitwise);
            if (size_in ≥ 40)
                % hCRC check

                % needed for the crc calculation
                hDetect = comm.CRCDetector([8 7 6 4 2 0]);
                % detects if there is an error in the CRC of the header
                [¬, err] = step(hDetect, bitwise(1:5*8,1));

                if (err ==0)
                    obj.classUse = bitwise;

                    %We are actually converting the array of bits here
                    %and then passing the pretty decimal numbers we get
                    %to the propertyfunctions.
                    obj.frameType = bi2de(bitwise(1:8,1)','left-msb');
                    obj.rcvID = bi2de(bitwise(1+8:2*8,1)','left-msb');
                    obj.sndID =bi2de(bitwise(1+2*8:3*8,1)','left-msb');

                    %Whether there is data or not depends on frameType.
                    %The location of data in the frame is dependent on
                    %dataSize so we pass the unaltered FrameObj input
                    %to obj.data
                    obj.data = bitwise;
                else
                    % the crc does not match and the header is junk
                    obj.frameType = FrameObj.INVALID;
                end
            else
                % the data we recdived is not long enough to check the
                % header crc
                obj.frameType = FrameObj.INVALID ;
            end
        else %incorrect number of inputs
```

```
                error('That is not a valid number of inputs')
            end
        end

%classUse
        %This property enables us to distingush between the two uses of the
        %class FrameObj though the length of the first (or only) input.
        %This is only used by obj.data which takes both string type inputs
        %and arrays of bits and needs to be able to distinguish them
        function obj = set.classUse(obj,inputframeType)
            [array_or_vector,¬] = size(inputframeType);
            if array_or_vector == 1
                obj.classUse = FrameObj.ENCODE;
            else
                obj.classUse = FrameObj.DECODE;
            end
        end

%frameType
        function obj = set.frameType(obj,inputframeType)
            %Using the switch statement in this way ensures that a
            %supported data type is used
            switch inputframeType
                case FrameObj.DATAFRAME %DATA
                    obj.frameType = uint8(inputframeType);
                case FrameObj.ACKFRAME  %ACK
                    obj.frameType = uint8(inputframeType);
                case FrameObj.POLLFRAME %POLL
                    obj.frameType = uint8(inputframeType);
                case FrameObj.REQFRAME  %REQ
                    obj.frameType = uint8(inputframeType);
                case FrameObj.INVALID   %INVALID
                    obj.frameType = uint8(inputframeType);
                otherwise % also INVALID
                    obj.frameType = uint8(FrameObj.INVALID);
            end
        end
```

```
%rcvID
        function obj = set.rcvID(obj,inputrcvID)
            obj.rcvID = uint8(inputrcvID);
        end

%sndID
        function obj = set.sndID(obj,inputsndID)
            obj.sndID = uint8(inputsndID);
        end

%data
        %frameType dependent
        %classUse dependent
        %Data actually refers to the data and the CRC8 number in an n*1
        %binary array
        function obj = set.data(obj,datainput)
            %These variables mean we can vary the size of MAXBYTES or the
            %header without and data will still be functional.
            header_bits = (FrameObj.MAXBYTES-(FrameObj.MAXDATA+1))*8;
            max_data_bits = FrameObj.MAXDATA*8;
            switch obj.frameType
                case FrameObj.DATAFRAME %DATA
                    if obj.classUse == FrameObj.ENCODE;
                        %This converts the datainput into an array of bits
                        temp_bin = reshape(dec2bin(datainput,8)',1,[]);

                        %Define the length of temp_data for speed
                        % the length of temp_data is limited by MAXBYTES
                        if size(temp_bin,2)≥max_data_bits
                            temp_data = zeros(1,max_data_bits);
                        else
                            temp_data = zeros(1,size(temp_bin,2));
                        end

                        for j=1:size(temp_data,2)
                            temp_data(1,j) = str2num(temp_bin(1,j));
```

```
            end

            crcGen = comm.CRCGenerator([8 7 6 4 2 0]);

            %Calculates the CRC and adds it to the end of data
            obj.data =  step(crcGen, temp_data');

        elseif obj.classUse == FrameObj.DECODE;
            data_bits = size(datainput, 1)-header_bits -8;
            %This seperates the data from the rest of the array
            %using dataSize.

            %First seperate dataSize then convert to decimal
            %Cast to double and convert from bytes to bits
            Temp =  bi2de(datainput(1+3*8:4*8,1)','left-msb');
            ds = double(Temp*8);

            %This allows FrameObj to not exceed the dimensions
            %of inputdata in case the dataSize was corrupted to
            %be or larger than the length of the input array
            %and passed the hCRC.
            if ds ≥ data_bits
                ds = data_bits
            end
            % or larger than MAXDATA
            if ds ≥ max_data_bits
                ds = max_data_bits
            end

            %Seperate data using the start of the data and the
            %length+crc
            bits = datainput(header_bits+1:header_bits+ds+8,1);
            %cast to double
            obj.data = double(bits);
        end
    case FrameObj.ACKFRAME %ACK
        obj.data = ''; %could be anything
```

```
                %if there is no data you should not try to access tha
                %ACK data but data is always assesed regardless of the
                %frameType
            case FrameObj.POLLFRAME %POLL
                obj.data = bi2de(datainput);%work on this
                    %is there anything we don't want converted here?
                    %unsure
            case FrameObj.REQFRAME %REQ
                obj.data = bi2de(); %work on this.
            %Do we need this?  Currently no data to be passed besides
            %recID and sendID.
            case FrameObj.INVALID   %INVALID
                obj.data = ''; %could be anything
                %if there is no valid frameType you should not try to
                %access tha data but data is always assesed regardless
                %of the frameType
            otherwise
                error('Not a supported frame type for data')
                % If this error occurs while using a legitimate frame
                % type please add an addiional case statement for that
                % frame type.

                % if there is no data for this frame type copy ACKFRAME
                % if there is string data copy DATAFRAME
                % A diferent type of data may require a different case
        end
    end

%dataSize
    % frameType dependent
    % returns 0 if ACK
    function value = get.dataSize(obj)
        switch obj.frameType
            case FrameObj.DATAFRAME %DATA
                %Convert from bits to bytes and subtract 1 to account
                %for the CRC
                value = (length(obj.data)/8)-1;
```

```
                case FrameObj.ACKFRAME  %ACK
                    value = 0;
                    %the ACKFRAME has no data but it must have a dataSize
                otherwise
                    error('Not a supported frame type for dataSize')
                    % If this error occurs while using a legitimate frame
                    % type please add an addiional case statement for that
                    % frame type.

                    % If there is no data for this frame type copy ACKFRAME
                    % If there is data copy DATAFRAME
                    % A diferent type of data may require a different case
            end
        end

%dCRC8
        %frameType dependent
        function value = get.dCRC8(obj)
            switch obj.frameType
                case FrameObj.DATAFRAME %DATA
                    %The last byte of obj.data is the CRC. It is seperated
                    %from the data here
                    [m, ¬] = size(obj.data);
                    value =zeros(8,1);      %Define value for speed
                    for j=1:8
                        value(j,1) = obj.data(m-8+j,1);
                    end
                case FrameObj.ACKFRAME %ACK
                    error('This is an ACK, it has no data therefor no CRC')
                    %If there is no data there should be no check
                otherwise
                    error('Not a supported frame type for CRC8')
                    % If this error occurs while using a legitimate frame
                    % type please add an addiional case statement for that
                    % frame type.

                    % If there is no data for this frame type copy ACKFRAME
```

```
                    % If there is data copy DATAFRAME
            end
        end

%header
        %all frametypes have the same length and components
        %frametype, sndID, rcvID, dataSize, hCRC8
        function value = get.header(obj)
            % Each part of the header is converted into binary arrays
            type_array  = de2bi(obj.frameType,8,'left-msb');
            rcvid_array = de2bi(obj.rcvID,8,'left-msb');
            sndid_array = de2bi(obj.sndID,8,'left-msb');
            size_array  = de2bi(obj.dataSize,8,'left-msb');

            temp_header = [type_array rcvid_array sndid_array size_array];

            crcGen = comm.CRCGenerator([8 7 6 4 2 0]);
            %Calculates the CRC and adds it to the end of header
            value =  step(crcGen, logical(temp_header'));
        end

%hCRC8
        function value = get.hCRC8(obj)
            % The last byte of obj.header is the hCRC
            [m, ¬] = size(obj.header);
            value =zeros(8,1);      %Define value for speed
            for j=1:8
                value(j,1) = obj.header(m-8+j,1);
            end
        end

%frameArray
        %frameType dependent
        %ACKFRAME --> frametype, sndID, rcvID, dataSize, hCRC8
        %DATAFRAME--> frametype, sndID, rcvID, dataSize, hCRC8, data, dCRC8
        % If we want to fill the end with zeros that could easily be done
        function value = get.frameArray(obj)
```

```
            % for data we combine the data and header in a n*1 binary array
            % the ack is only a header.
            % both header and data have a crc8 included

            switch obj.frameType
                case obj.DATAFRAME
                    value = [obj.header; obj.data];
                case obj.ACKFRAME
                    value = [obj.header];
                otherwise
                    error('Not a supported frame type for frameArray')
                    % If this error occurs while using a legitimate frame
                    % type please add an addiional case statement for that
                    % frame type.

                    % If there is no data for this frame type copy ACKFRAME
                    % If there is data copy DATAFRAME
                    %A diferent type of frame may require a different case.
            end
        end
    end
end
```