S I M U L A T I N G S D L C U S I N G P Y T H O N

FRAME STRUCTURE IN SDLC FORMAT

Calculating 16 bit CRC field using the CRC polynomial x^16 + x^12 + x^5 + 1 (`old method is used and its not polynomial division )`

NRZI ENCODING AND DECODING

BIT STUFFING

SYNCRONISATION BITS

*****Establishing communication channel between PC and SDLC card***

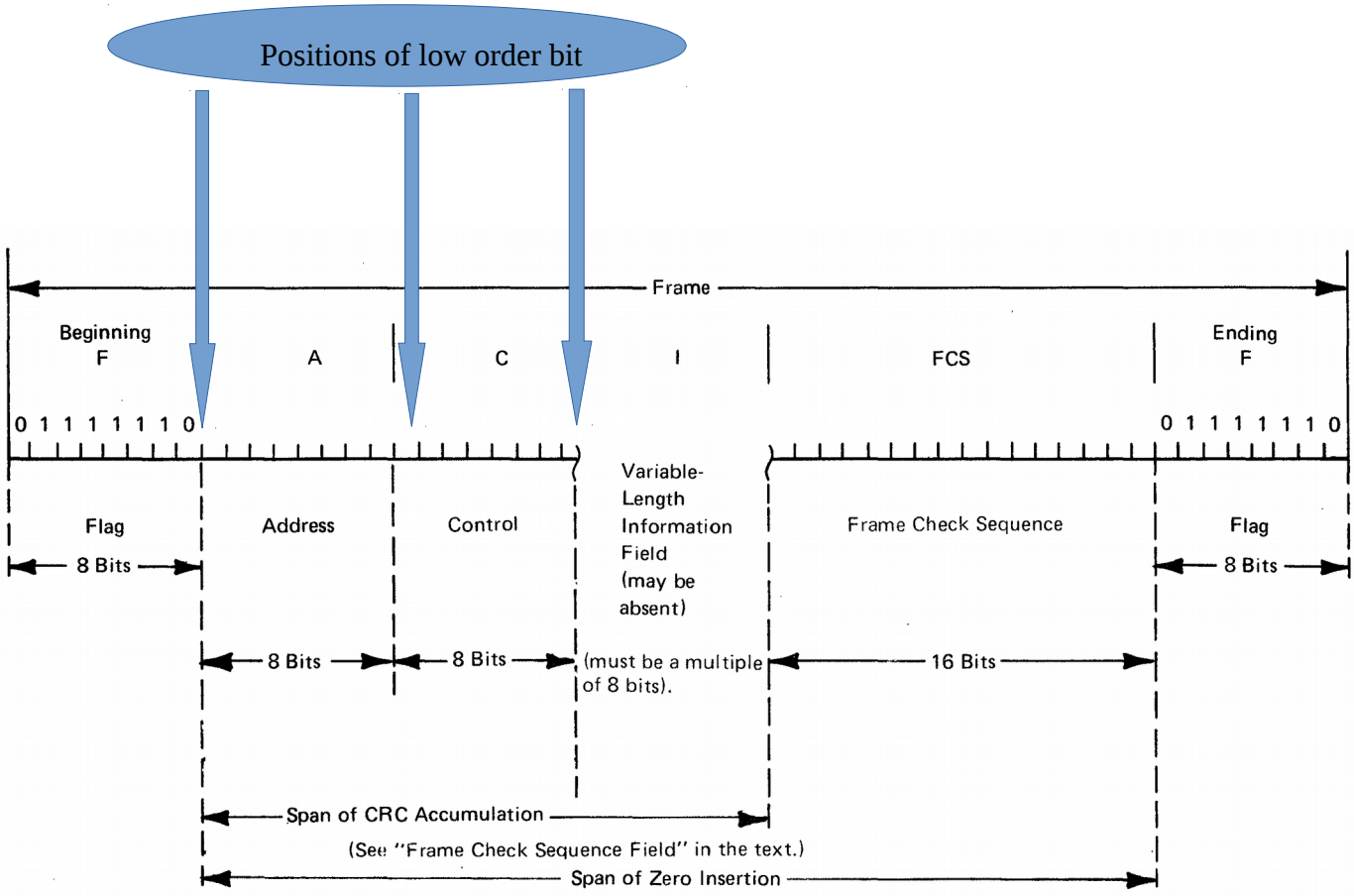
TRANSMITTER PART

RECEIVER PART

Modification required

SUBMITTED BY :-
RAJ KUMAR
P.V SAI HARSHIT
PRADEEP KUMAR

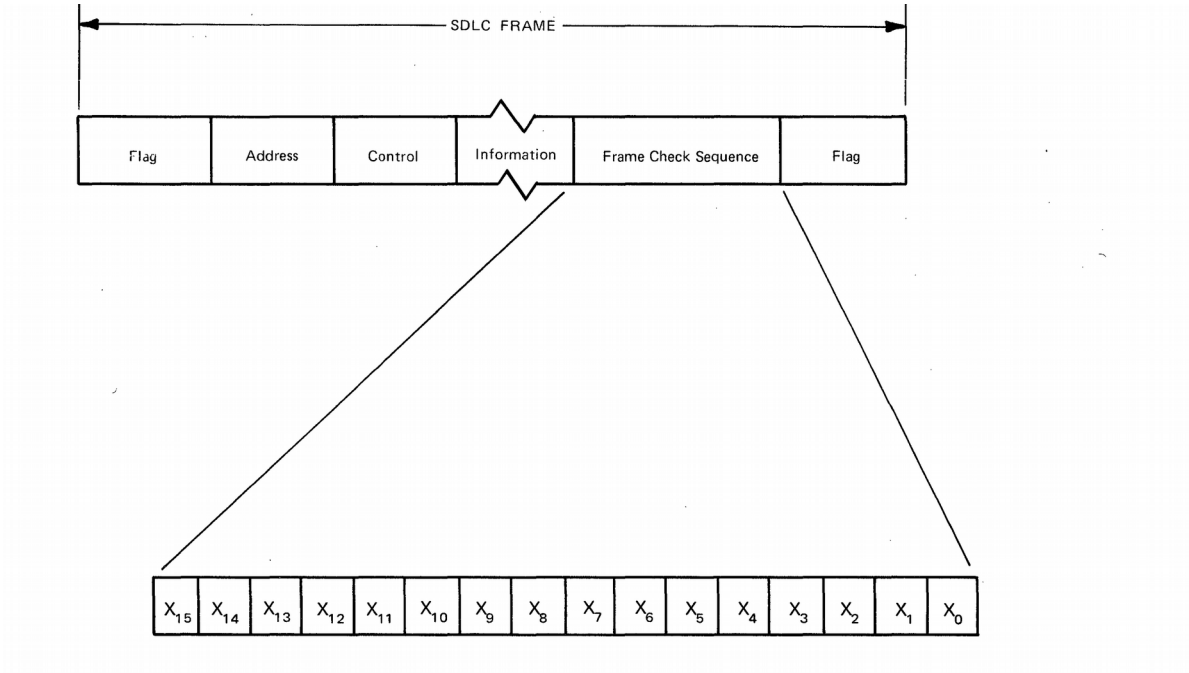
FRAME STRUCTURE IN SDLC FORMAT



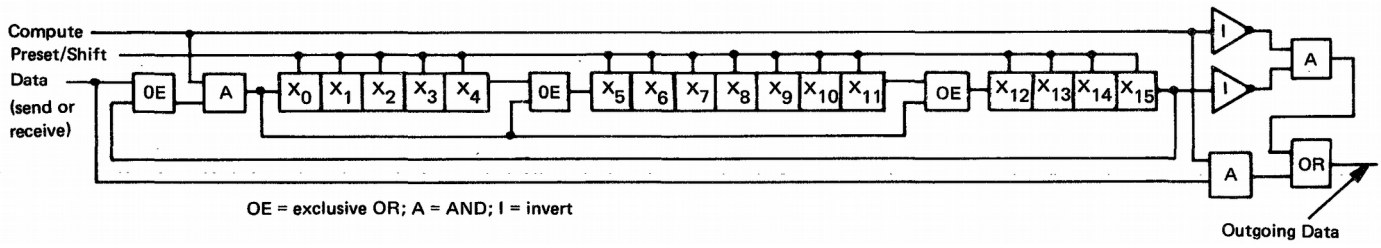
**lowest order bit should start from left most side of packet .

Example :-
For address field of 2 the eight bit address field is not 00000010 but is 01000000.

COMPUTING 16 BIT CRC FIELD USING THE CRC POLYNOMIAL x^16 + x^12 + x^5 + 1 .



PROCEDURE FOR CALCULATING CRC :-



1. Make a reamainder_bit array of length **16** and append it with all ones (array[16]) .

2.Take XOR of last element of array(array[15]) with the first bit of data bit and store it in a variable .

3. Now delete the last element of array (del array[15]) and append that value stored in variable into the 0^{th} position of that remainde_bit array (array.insert(variable , 0)) .

4.Now taking XOR of 6^{th} element of array (i.e. array[6]) with that variale calculated above .And storing it in array[5].

5. Similarly for array[12] , first XOR of array[12] with that variable and then storing it in array[12].

Repeating all above five steps for total length of data

sudo code :-

```
for i in range( len( data ) ) :
        # doing all five steps
```

The final modified remainder_bit (array of len 16) **IS NOT** our crc field of sdlc packed .

We have to invert the bit of remainder bit (i.e. 1--->0 and 0--->1).

Now this is our crc field to be sent .(NOTE to be sent only not for receive side)

FOR RECEIVING SIDE :-

Doing all five steps for the length of data .(BEFOR THIS FIRST WE HAVE TO NRZI DECODE OUR RECEIVE DATA AND THEN BIT_STUFFING AND THEN DOING ALL THESE FIVE STEPS)

And after that if we get a fixed pattern i.e. 1111000010111000 , that menas our sdlc data is correct .

(DON'T DO BIT INVERSION AFTER FIVE STEPS FOR RECEIVING SIDE)

NRZI ENCODING AND DECODING

NRZI is based on chage in voltage level .

If voltage is going high to low (in binary 1--->0) or low to high (in binary 0--->1) then that change is represented by binary 0.

If there is no change either in high state or in low state i.e. in binary 1--->1 or 0--->0 , then it is represented by 1 .

sudo code :-

```
#store all sdlc packet in a array .
store_sdlc_packet = [] ;
store_sdlc_packet  =  .read_operation  from a file or from some externel data source .

#Store first element of that array in a temp variable .

temp = store_sdlc_packet[0] ;
for i in range( len( store_sdlc_packet )
        # Del that element stored in temp varible from store_sdlc_packet

        del store_sdlc_packet[0] ;

        Now ,  if  temp  == the store_sdlc_packet[0] ;
                nrzi_encoded_data = 1 ;

        else :
                nrzi_encoded_data = 0  ;
        temp = store_sdlc_data[0] ;
```

or simple XOR ooperation we can take rather then using if conditions .

BIT STUFFING (to avoid accurence of start or end field in between the data)

Its just insertion of a zero after every five ones (** not including flags , only in address + cotrol + information + crc_remainder) .

In sending sdlc data (before nrzi encoding) , if there is any consecutive sequence of five one's then inserting a 0 after them .

In receiving sdlc data (after nrzi encoding) , if there is any consecutive secuence of five one's then remove a 0 after them .

SYNCRONISATION BITS

These are used for clock recovery / adjusting baud rate for incomming sdlc data .

In sdlc syncronisation bits are alternating sequence of 1's and 0's .

In nrzi encoded form it is consecutive zeros (the length of which is multiple of 8)

Establishing communication channel between PC and SDLC card

. USB port is used for establishing communication between PC and SDLC card.

. USB follows RS232 logic levels i.e 1 is represented by -12 volt and 0 is represented by +12 volt . So , an usb to uart having RS232 to TTL logic convertor is needed .

Normal communication between usb to sdlc at baud rate of 9600 .

USB will make an 8 bit information field in his usb packet containing 7 bit data representing an ascii character if that number is lesser then 127 and if it is grater then 127 then it will break that number using UTF8 encoding .

Information and algorithm to encode and decode utf8 in in this site .
http://home.kpn.nl/vanadovv/uni/utf8conversion.html

I wrote UTF8 decoding in python script name utf8_to_decimal.py .

UTF8 encoding can be avoided using data bit size of 5 bits per packet .

Observation from normal communication .

Data received is corrupted .
We are getting lesser number of bits then required . But getting some trace of SDLC packet format i.e end flag and some recognised form of information bits .

If we increase the baud rate from 9600 . Number of bits will increase but again data is corrupted . And this time we are not getting even trace of SDLC packet .

And sending data from PC to sdlc card also failed .

Reason for failure :-
Baud rate includes both data to be transfered and overhead (flags , address , crc) .

Now , the already communication which is happening between sdlc chips at antenna and the control room is encoded and follows a baud rate of 9600 and the other side is ready to rececive at that baud rate .

But in our usb what is happening is we first receive data at rate of 9600 bps but then the usb protocol encode that data again thus increasing its overhead , so we are

getting lesser number of bits then required .And that is the reason when we tried to receive data using normal communication (as described above) we are getting leser number of bits in our sdlc packet .

Solution for this is that we have to define our own communication and then store data and then send back it to the PC .

Or

Do sample data at rate of 9600 and then send it to our pc at rate of 115200 but only 1 bit at a time which make transmission slower then 115200 and is nearly equal to 9600 .

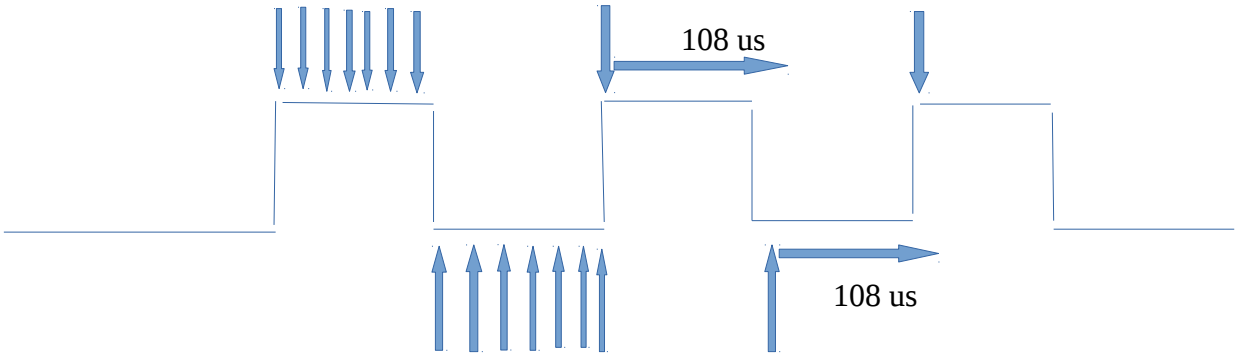
Sampling the data bits (not using normal communication)

We programmed our microcontroller to sample SDLC data on receiving side .

Steps taken :-
As SDLC also genrates some syncronisation bits ----> in nrzi sequence of 01 i.e changing levels .

First we tried to reach either rising edge or falling edge of syncronisation bits and then we added a fixed delay of 108 us to sample each bit correctly .

Microcontroller will sample the data at very high speed .



When microcontroller saw a change from logic high to low or vice versa it will found an edge , and then it will start sampling the data at a fixed delay of 108 us .

Then we send that one or zero to our PC at rate 115200 baud rate .

Using this sampling method we are getting correct nrzi encoded sdlc packet.
Limitation -----> information field shoud not exced 35 byte in an SDLC packet.

For sending to sdlc chip

We are sending data from PC to microcontroller single bit at a time .
Directly sendind data from PC to SDLC card is failed as discribed above .

Now two method can be applied ----->

1. Sending data from PC with 1500 loops of delay (around 1 us) to microcotroller and then it will correspondingly make pin high or low .

Observation in this method :-
When we try to send simply consecutive sequence of 1 and 0 the waveform is not stable i.e the pulse width is varying in between 100 us to 137 us with mostly having width of 118 us .

Using this the SDLC card is excepting 5 out of 1 transmitted signal .

2. Sending data from PC with no delay (say at high baud rate of 115200) and then first storing it in memory of microcontroller and then sending it with delay of 115 us will produce a stable pulse of width 119 us .Or we can adjust the delay to adjust the pulse width .

TRANSMITTER PART

1. Taking input ----the data , control , and then addess field in hexadecimal format and converting it into binary .

2. Calculating 16 bit CRC field using the CRC polynomial x^16 + x^12 + x^5 + 1 . **(old method is used and its not polynomial division)**

3. Zero insertion after every 5 ones in 8 bit address field + 8 bit control field +8 bit data field + 16 bit CRC field , to avoid accurence of start or end field in between the data .

4. Now making the sdlc packet from them i.e ---- start field + 8 bit address field + 8 bit control field +8 bit data field + 16 bit CRC field + end flag .

5. NRZI encoding of sdlc packet .

6.Writing this NRZI encoded data on usb port .

7. After transmitting data (writing in tx_file.txt file) , my program will try to fetch data from USB port .

8. Case ---- (a) . Its not getting any thing from RX i.e no data is received from usb port .Then , program will wait for some pre-defined duration of time (in actual that time sir will gave) . If it won't get any thing in that duration then it will again tramsmit the data and wait for ACK again , this process will happen for 3 times .If still after 3 try program wont get any RX data then it will exit(0) .

Case ----(b). It got some data from rx of usb . Then reciver part comes into play .

RECEIVER PART

1.After getting RX data , it will first decode it from nrzi , then it will start finding the start field in the data i.e. ‘01111110’ .

2.And once it found start field then it will store 8 bit address field + 8 bit control field + N bit data field + 16 bit CRC field , till end flag reaches i.e ‘01111110’ .

3. Zero removal after every five ones from 8 bit address field + 8 bit control field +8 bit data field + 16 bit CRC field.

3. Now using this 16 bit CRC it will varify if 8 bit address field + 8 bit control field +8 bit data field is correct or not .

4. If they got CRC varified then the program will give the data to the user .

MODIFICATION REQUIRED

As described in communication section ----- receiver side is able to collect data upto 35 byte of information field in a single SDLC packet , BUT for transmitted signal SDLC card is excepting 1 out of 5 packets .

The reason for transmitter failure is unstable wave form produced .

That can be corrected by first transmitting the data from PC to microcontroller and then storing that data in microcontroller memory and then sending that raw data to the sdlc card .

Two core microcontroller with reasonable ROM can be used for smooth communication . One core for receiving and other core for transmitting so that both rx and tx can run seprately(parallely) on both core .