# EE3576 Tutorials
# Communications Engineering

All answers should provide suitable diagrams and explanations. Unless the question explicitly requests a sketch of a diagram, diagrams alone shall not be considered sufficient to provide a full answer, be sure to explain the purpose of any diagram and to explain how this may be used to answer the question.

Students should review the material at: erg.abdn.ac.uk/users/gorry/eg3576/

**List of Tutorials**

## *Tutorial 1: Asynchronous Transmission and Reception*

(a) An ASCII character is transmitted using the 7-bit ASCII code with one parity bit, followed by two stop bauds.

**Start of sequence:**

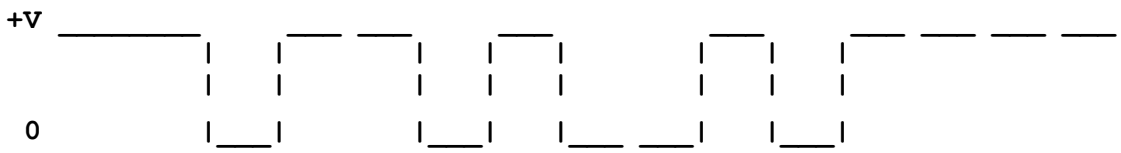

What is the decimal value of the byte sent in this sequence?

What is the ASCII character represented by this byte?

What form of parity was used?

(b) The following byte was sent using 8 bits, no parity, 2 stop bauds.

**Start of sequence:**




What is the hexadecimal value of the byte sent in this sequence?
What ASCII character represented by this byte?

(3) What is the sequence of bauds transmitted to send an ASCII character ‘j'
(hexadecimal 0x6A) at 4800 bps using no parity and two stop bits?

What period of time is required to send this character?
(A follow-up question for later in the course: what signal bandwidth is needed in kHz?)

## *Tutorial 2: Packet Transmission (Frames)*

(a) A sender generates a packet multiplex stream in which packets consist of 32 bytes of each data represented as characters. Each frame starts with the character '%'. Explain how a receiver extracts the frames from the stream of characters shown below. [5 marks]

```
FRAMEAQGP%HHNOVALUEFRAMERXFRAMDFOXKYPQTGP%EEHTHOINTERMSO
FCASHTHGPJOLWZ$GP%AAYXHOINTERCIBJKDTGSDFPOKLLYKGP%TTYFRA
MECANCASHITNICHG
```

(b) How can someone add an integrity check to detect corruption of the data? [3 marks]

(c) What is the percentage overhead (i.e. what percentage of capacity is used for bits other than frame payload data), assuming:

(i) Asynchronous transmission of slots with 2 ***stop bauds***.

(ii) The method also used 1 stop baud per 8b character, and sends 30 characters with an NMEA ***integrity check***.

[5 marks]

## *Tutorial 3: EIA-485 Transmission*

(a) A bus uses ***asynchronous character framing***, with ***differential transmission***, in which one slot is sent with 2 stop bits, no parity baud included. Sketch the sequence of signals sent on both wires, when sending the hexadecimal value 0x05. [6 marks]

(b) If the baud rate is 250 k baud, what is the maximum frame rate for 512 slot frames ? How is this rate changed if slots are of size 52B? [5 marks]

(c) Asynchronous transmission does not require a clock signal to be sent - use diagrams to show how it is possible for a receiver to work when a 250 000 baud transmit clock is 2% lower than the normal receive clock frequency? Show also that this can not work at a 10% lower baud rate. [10 marks]

(d) What is the purpose of the ***stop bauds***? [4 marks]

## *Tutorial 4: DMX Frame Transmission*

(a) The DMX multiplex may be sent on an interface that uses a 3 or 5 pin XLR connector. What are the properties of this interface that make it suited for use in an industrial environment? [4 marks]

(b) A DMX-512 control bus uses a ***balanced transmission line***, draw a diagram showing the way equipment is connected to the bus and the waveform when the bus is used to send a 100 kHz square wave. [8 marks]

(c) Explain how a receiver determines the ***start*** of a received DMX-512 frame. [8 marks]

(d) Estimate the maximum number of receivers allowed on the bus, given a nominal receiver input impedance of 12 k Ohms [5 marks]

(e) What is the effect of two 4 channel DMX fixtures being configured with a start address of 5? Explain how these process a received DMX frame to extract the values that represent each channel. [8 marks]

## *Tutorial 5: DMX Microcontroller Algorithms*

(a) A microprocessor-based DMX receiver uses software that includes the following Interrupt Service Routine. Draw a flow chart to capture and explain the operation of this algorithm. [10 marks]

```
ISR (UART_RX_vect)
{
static  uint16_t DmxCount;                  //global counter
        uint8_t  USARTstate= UCSRA;         //get USART state before data!
        uint8_t  DmxByte   = UDR;           //get USART data
        uint8_t  DmxState  = gDmxState;     //copy global to increase speed

if (USARTstate &(1<<FE))                    //check for break
   {
   UCSRA &= ~(1<<FE);                       //reset flag in USART
   DmxCount =  DmxAddress;                  //reset slots counter
                                 //(count slots before start address)
   gDmxState= BREAK;
   }
else if (DmxState == BREAK)
   {
   if (DmxByte == 0) gDmxState= STARTB;     //normal start code detected
   else              gDmxState= IDLE;
   }
else if (DmxState == STARTB)
   {
   if (--DmxCount == 0)                     //start address reached?
        {
        DmxCount= 1;                        //set up counter for
                                            // required slots
        DmxRxField[0]= DmxByte;             //get 1st DMX channel of device
        gDmxState= STARTADR;
        }
   }
else if (DmxState == STARTADR)
   {
   DmxRxField[DmxCount++]= DmxByte;         //get channel
   if (DmxCount >= sizeof(DmxRxField))      //all slots received for device?
        {
        gDmxState= IDLE;                    //wait for next break
        }
   }
 returni;
}
```

(b) The Receiver has a ***base address*** of 6 and supports four 8-bit channels. Explain how the receiver uses the DMX frame to set the output value of the 2nd channel. [4 marks]

(c) The value of a channel has an initial DMX slot value of 0, later a value of 50 and then finally a value of 100. Sketch the waveform for each of the values that is used to ***drive*** a TRIAC dimmer circuit, and the resulting voltage on the mains power line. [10 marks]

*Additional thoughts:*

(i) What is the purpose of the returni statement? What does it do?

(ii) Can you represent the algorithm as a state transition diagram? What variable is used to store the state?

(iii) Why is the code written so that the global state is copied to a local variable during the interrupt service?

## *Tutorial 6: DMX Receiver Processing*

***Example**: This example considers a DMX receiver with a start address of 6, and a field size of four slots. It traces the values of key variables after completing each interrupt service when the receiver receives a DMX frame with the following first ten data slots {11,12,13,14,15,16,17,18,19,20,...}.*

Sample data including start code: DMX data = {0,11,12,13,14,15,16,17,18,19,20,...}
DMXAddr = 6 (configured start address)
DMXRxField = 4 Bytes (given receive size).

| Variable | Note |
|---|---|
| gDMXstate | {0=Idle;1=Break;2=StartB;3=StartAddr;} |
| USART Status | {FE = Break; 0 = Data} - flag triggering ISR |
| DMXByte | Contents of receive byte from USART |
| DMXAddr | 6 (configured from DIP Switch) |
| DMXCount | Global slot counter |
| DMXRxField[0..1] | Output Array of size 4 bytes |

Here is a trace reporting key variable values after completion of the interrupt service routine.

| Interrupt causing ISR | DMXByte | gDMXstate | DMXcount | DMXRxField | | | |
|---|---|---|---|---|---|---|---|
| | | | | [0] | [1] | [2] | [3] |
| Break | N/A | 1 | 6 | ? | ? | ? | |
| Data | 0 | 2 | | | | | |
| Data | 11 | | | | | | |
| Data | | | | | | | |
| Data | | | | | | | |
| Data | | | | | | | |
| Data | | | | | | | |
| Data | | | | | | | |
| Data | | | | | | | |
| Data | | | | | | | |
| Data | | | | | | | |
| Data | | | | | | | |
| Data | ... | 0 | 4 | 16 | 17 | 18 | 19 |

## *Tutorial 7: Remote Device Management (RDM)*

(a) Remote Device Management is an extension to DMX that allows configuration of remote devices and retrieval of data from them. What changes are needed in a device's electronics to enable RDM operation? [4 marks]

(b) Explain the sequence of interactions required when a controller wishes to retrieve data from a remote device. [6 marks]

(c) In RDM, the bus may be unpowered for periods of time. Explain how a bias circuit may be used to prevent the bus floating to an arbitrary value, and calculate the values of the bias components. [15 marks]

## *Tutorial 8: Controller Area Network (CAN) Bus*

(a) A CAN bus uses a variant of differential transmission, what is the key difference to methods such as RS-485 and DMX? [5 marks]

(b) Explain how CAN uses bit stuffing to achieve redundancy. [10 marks]

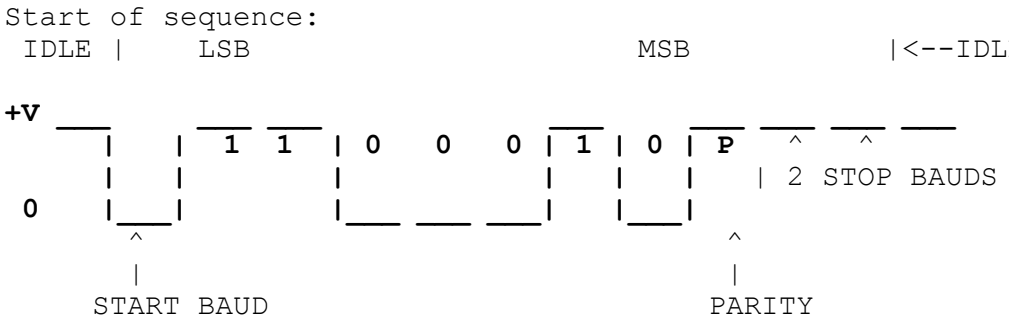
(c) Consider two nodes with two message IDs: 415 and 455, what is the sequence for the first 12 bits if each is sent individually, and what is the resulting arbitration when the two messages are sent simultaneously? [10 marks]

# ANSWERS
# ANSWERS
# ANSWERS
# ANSWERS

- 1 *Asynchronous Transmission and Reception*
- *2 Packet Transmission (Frames)*
- *3 EIA-485 Transmission*
- *4 DMX Frame Transmission*
- *5 DMX Microcontroller Algorithms*
- *6 DMX Receiver Processing*
- *7 Remote Device Management (RDM)*
- *8 Controller Area Network (CAN) Bus*

# TUTORIAL 1: *Asynchronous Transmission and Reception*

(a) An ASCII character is transmitted using the  bit ASCII code with even parity bit, followed by two stop bauds.




What is the decimal value of the byte sent in this Sequence?
What is the ASCII character represented by this byte?
What form of parity was used?

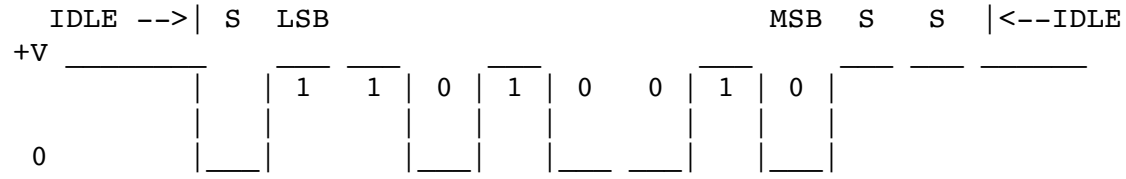
These transmissions send the LSB first and the MSB is followed by one even parity baud and two stop bauds.

The byte (sent lsb first) reads 110 0010 as seven bits.
This is binary: % 0010 0011 (0x23) decimal 35 (the MSB is set to zero). The ASCII Character '#'.

The parity baud, if included occupies the final position before the stop bauds. Since the parity baud was one, and there were an even number of 1 bauds, this byte used EVEN parity.

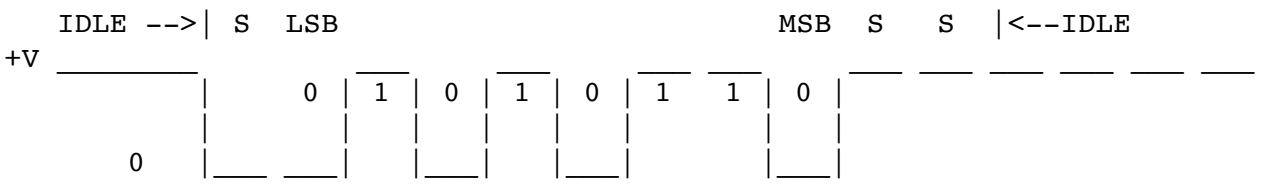
(b) The following byte was sent using 8 bits, no parity, 2 stop bits.
What is the hexadecimal value of the byte sent in this Sequence?
What ASCII character represented by this byte?

Start of sequence:




The byte (sent lsb first) reads 1101 0010. This is: % 0100 1011 (0x4B) decimal 75
Using an ASCI table, the character is at row 4 and column B, which is the letter 'K'.

(c) What is the sequence of baud transmitted to send an ASCII character 'j' (Hexadecimal 0x6A) at 4800 bps using no parity and two stop bits?




ASCII Character j is: % 0110 1010 (0x6A) decimal 106 - note this is lower case.
The byte (sent lsb first) reads 0101 0110

See the ASCII Chart at erg.abdn.ac.uk/users/gorry/eg3576/ascii.html

What period of time is required to send this character?

1 baud = 1/4800 secs; 11 bauds = 11/4800 = 2.3 mS

## TUTORIAL 2*: Packet Transmission (Frames)*

(a) A sender generates a packet multiplex stream in which packets consist of 32 bytes of each data represented as characters. Each frame starts with the character '%'. Explain how a receiver extracts the frames from the stream of characters shown below. [5 marks]

```
FRAMEAQGP%HHNOVALUEFRAMERXFRAMDFOXKYPQTGP%EEHTHOINTERMSO
FCASHTHGPJOLWZ$GP%AAYXHOINTERCIBJKDTGSDFPOKLLYKGP%TTYFRA
MECANCASHITNICHG
```

Data is sent in a series of frame. Each frame carries data, delimited by a frame alignment sequence, in this case the character %. The receiver needs to read each character in turn and skip characters until the character matches the delimiter.

(In this example, the first value carried in each of the successive 4 frames spells the word "HEAT").

```
FRAMEAQGP
%HHNOVALUEFRAMERXFRAMDFOXKYPQTGP
%EEHTHOINTERMSOFCASHTHGPJOLWZ$GP
%AAYXHOINTERCIBJKDTGSDFPOKLLYKGP
%TTYFRAMECANCASHITNICHG
```

(a) How may one add an integrity check to detect corruption of the data?

One simple way may be to add a checksum of the data. The simplest is a longitudinal parity sum (XOR of each byte) as used in the NMEA specification.

This frame has a payload of: HHNOVALUEFRAMERXFRAMDFOXKYP.

The 8b longitudinal parity value 0x51, calculate as described below.

### Longitudinal Parity

The longitudinal parity is calculated by independently summing all the 1st bits in each byte of the NMEA frame (between '$' and '*') to form the 1st bit of the parity byte. This is repeated for the next bit, by XOR summing all the 2nd bits in each byte to form the 2nd bit of the parity byte, etc, until all 8 bits have been summed.

A Microcontroller could calculate this sum for all bit positions at the same time by using a byte-wise XOR instruction.

The result is an 8 bit value. This value is represented as two hexadecimal digits. This is formed by diving the value by 16. The result of the division forms the first digit, by adding the ASCII value for '0'. The remainder of the division forms the first digit, by also adding the ASCII value for '0'. (Note that sending the value in hexadecimal avoids the problems associated with receiving a parity value that happens to be the same as a '*' or '$' - which would be interpreted by the receiver as part of a new frame.)

The longitudinal parity is sent as two hex digits in ASCII sent after the '*' character, e.g. "*51".

### Sender

At the sender, the longitudinal parity byte is calculated for the transmitted frame data. This sent after sending the '*' representing the end of the data, by following this by the two hexadecimal ASCII characters.

### Receiver

At the receiver, the longitudinal parity byte is calculated for the received frame data using exactly the same method. The calculated part value is ready when the '*' is received, representing the end of the data.

After the '*' is received, the receiver then receives the two hex characters that were calculated by the sender. These are converted to an 8 bit value. (The ASCII value of '0' is subtracted from each parity character in turn, and the transmitted parity is then the sum of 16* the first longitudinal parity value+second longitudinal parity value.).

The receiver then compares the longitudinal parity calculated at the receiver with the transmitter longitudinal parity (calculated by the sender). Any difference in the two bytes is an indication that the data was corrupted.

If the town 8b values for the calculated longitudinal parity and the transmitted parity ***match***, the data in the frame is forwarded.

If the calculated longitudinal parity and the transmitted parity ***do not match***, the data in the frame is discarded.

This is not a "strong" verification, e.g. as used to validate a credit card or sign an internet web session, but it is sufficiently robust to detect many (in fact most) transmission errors.

(c)What is the percentage overhead (i.e. what percentage of capacity is used for bits other than frame payload data), assuming:

(i)Asynchronous transmission of slots with 2 stop bauds.

Number of overhead bits/total slot size = 3/11 = 27%

(ii) The method also used 1 stop baud per 8b character, and sends 30 characters with an NMEA integrity check.

The framing overhead adds four bytes one dollar, one star and two parity digits:

… $….*XX

The efficiency is the number of information bits divided by the total number of bauds.

= (30X8)/( (30+4)*(8+2)+(4*8) )

= 240/((34*10)+32)) = 240/372 =0.65

This is 65% efficient,

The overhead is 1-efficiency, and may also be expressed as a percentage

= 1- 0.65

i.e., 35% overhead

Bonus question: What if you also include <CR> <LF> the overhead after the checksum?

Some NMEA devices send this because the strings then format nicely on a an ASCII terminal, but any data between the checksum and the next $ is ignored by a computer trying to interpret the strings.

( This is 1-(240/392) = 39% )

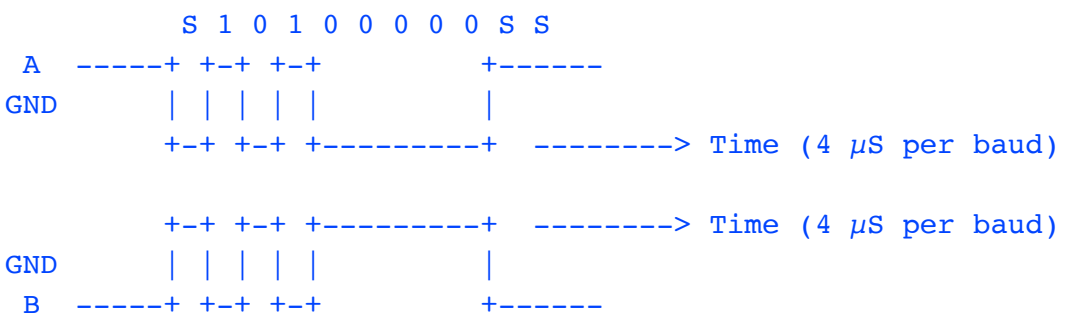
The extra 2 characters add just 4% more overhead!

## TUTORIAL 3: *EIA-485 Transmission*

(a) The DMX bus uses *asynchronous character framing*, in which one byte is sent with 2 stop bits, no parity bit included. Sketch the sequence of signals sent on both wires when sending the value 0x05. [6 marks]

In asynchronous serial communication, the electrical interface is held in the mark position between characters. Communication does not include a separate signal wire for the timing (e.g. for sending the clock signal), and no clock-encoding method is used. The clocks work totally independently at the same nominal frequency.

Instead, the start baud serves to prepare the receiving mechanism for the reception and registration of a character/slot and the stop baud serves to bring the receiving mechanism to rest in preparation for the reception of the next character/slot. The start of transmission of a character is signalled by a drop in signal level to the space level. At this point, the receiver starts its clock. After one bit time (the start bit) come 8 bits of true data followed by one or more stop bits at the mark level. The bauds are sent lsb first.




Vertical axis shows the voltage - note there are both +ve and -ve signal levels. The "stop baud" is actually a "stop period"; the stop period of the transmitter may be arbitrarily long, but must be at least 2 bauds. At the end of each character, the receiver stops briefly to wait for the next start bit. It is this difference that keeps the transmitter and receiver synchronized.

(b) If the baud rate of DMX is 250 k baud what is the maximum frame rate? [5 marks]

The baud rate is the rate of symbols per second (Note: this includes start and stop bauds).
1 baud = 1/250 000 secs = 4$\mu$S
1 slot = 8 data bits, plus 3 framing bauds = 44 $\mu$S
A DMX frames comprises the Mark, Mark after break, control slot and a maximum of 512 slots.
1 frame duration = synch+513 slots = synch +513*44 = 0.104 +22.572 mS =22.67mS
max frame rate = 1/frame duration = 44 f/sec
The rate may be lower in many practical systems (e.g. for RDM operation or SIP frames).

How is this changed for frames of size 52B ?

1 DMX frame duration of size 52B = synch+53 slots
= synch +53*44
= 0.104 +2.3 mS
=2.4mS

(The maximum frame rate is hence 1/frame duration = 417 f/sec).

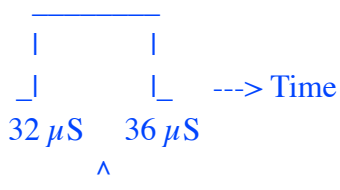
Asynchronous transmission does not require a clock signal to be sent - use diagrams to show how it is possible for a receiver to work when a 250 000 baud transmit clock is 2% lower than the normal receive clock frequency? Show also that this can not work at 10% lower. [10 marks]

The receiver uses a clock that has been pre-configured to the nominal baud rate of the sender.

The receiver must sample each "baud" , and should ideally do this in the centre of the baud-period. If the receiver baud rate is different to the sender, the clocks will slip relative to one another after each baud interval, the more bauds accumulated since the start baud, the bigger the total slip, until finally one baud will not be sampled (or will be sampled at the edge of the waveform representing the baud, where the signal is weaker.) This will result in a receive error, i.e. a baud value being misread.

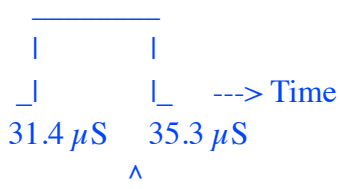
1 baud = 4 $\mu$S and therefore 1 slot is sent using 11 bauds

A sender at 25000 baud starts the last (msb) data bit at 8*4 $\mu$S after the start bit, i.e. at 32 $\mu$S.
The msb bit is completed at 9*4 $\mu$S after the start bit, i.e. at 36 $\mu$S,
A receiver at 25000 baud samples the last baud at 8.5 *4 $\mu$S = 34 $\mu$S




Sample at 34 $\mu$S, sampled at the centre of a correctly timed baud

A 2% lower sender rate baud rate results in a last bit of the slot being started:
at time 8*4*.98 $\mu$S = 31.4$\mu$S and ending at t 9*4*.98 $\mu$S = 35.3 $\mu$S




Sample at 34 $\mu$S, sampling is at the right edge of the baud

For the final baud, this results in sampling at the end of the last baud period, rather than at the centre of this baud. In theory this could be OK, but it would likely result in a degraded signal since practical line drivers limit the slew rate of the signal and therefore there is less signal energy away from the centre of the baud. The signal to noise ratio is therefore higher.

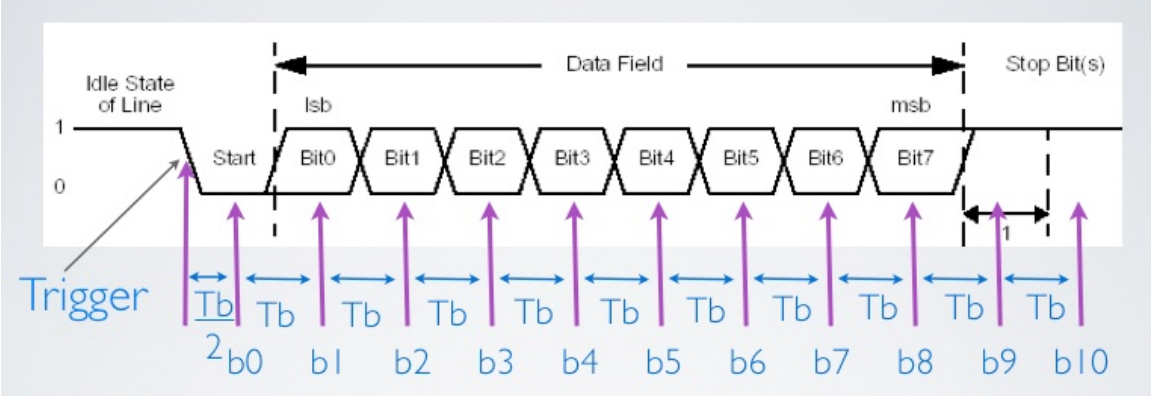
A 10% difference would be too much:
10% lower sender baud rate results in a last bit starting at 8*4*.9 $\mu$S = 28.8$\mu$S
and ending at t 9*4*.9 $\mu$S = 32.4 $\mu$S

The receiver samples after the last baud has been sent at 34 $\mu$S

What is the purpose of the stop bauds? [4 marks]

The stop bauds are  provide dto verify the end of the character frame, and hence to provide a check that the data is valid, both stop bauds must be at the same value as the idle value: in this case the mark value.

A diagram should be provided.




Tb is the duration of one baud. In the diagram, the stop bauds are b9 and b10.

A UART normally reports an invalid stop bit sequence in the status register. This could indicate:

- An incorrect receiver baud rate (i.e. not sampling in centre of each baud).
- Noise or interference resulting in corruption of the signal
- The cable had been disconnected.
- A break sequence was sent by the sender to gain the attention of the receiver. (e.g. to indicate the start of a DMX frame).

# TUTORIAL 4: *DMX Frame Transmission*

(a) The DMX multiplex may be sent on an interface that uses a 3 or 5 pin XLR connector. What are the properties of this interface that make it suited for use in an industrial environment? [4 marks]

Robust XLR connector (compared to Ethernet)
Differential twisted pair (less radiated signal, more immunity to noise)
Shielding of cable (foil, and braiding, i.e. not audio cable!)
Limited baud rate / slew rate (limits effect of noise above signal frequency)
<more are possible>

(b) An RS-485 control bus uses a ***balanced transmission line***, draw a diagram showing the way equipment is connected to the bus and the waveform when the bus is used to send a 100 kHz square wave. [8 marks]

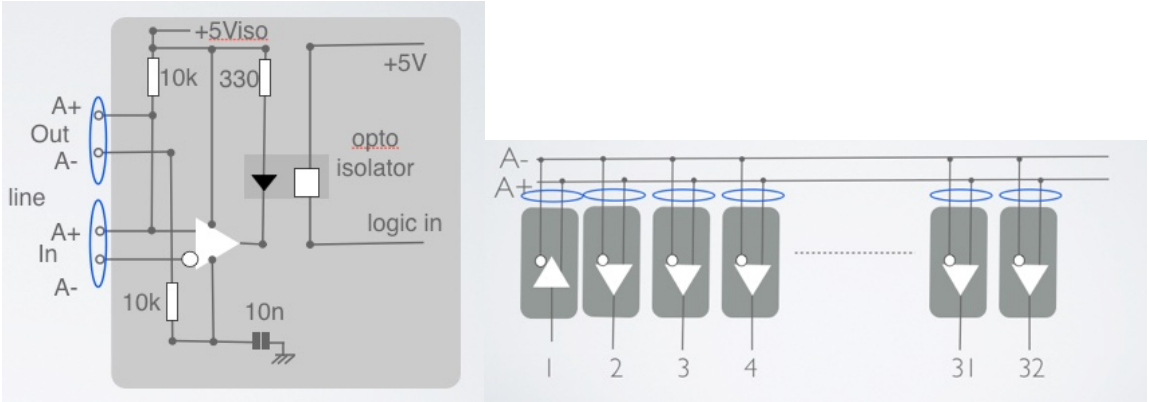
Key features:

- Balanced transmission A,B signals (B is the inverse of A) - This means no net radiated signal (i.e. A and B signals cancel out, and good immunity to external interference - any interferer changes both signals equally.
- Diagram should show vertical axis (voltage), horz axis (time, 10 ms per baud)

Equipment connected in parallel across the balanced pair.

- Diagram should show sender, could show internal termination at sender, but should show termination of cable at remote end, to match cable impedance.
- All equipment wired so that inputs/outputs directly connected - signal still passes if equipment turned off.
- Cable should only be earthed at sender (master in RDM case).
- receivers connected in parallel across the cable, with high-impedance inputs - upto 32 devices per bus length
- could show splitters and mergers to combine multiple cable segments. Mergers allow more than one input to a cable segment - splitters copy waveform to multiple segments.
-

<diagrams are needed in the answer>



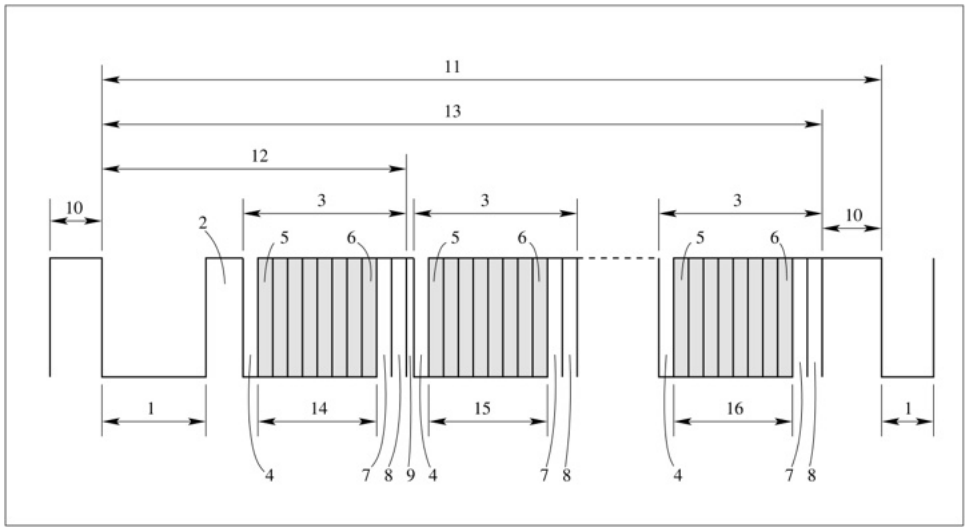
(c) Explain how a receiver determines the ***start*** of a received DMX-512 frame [8 marks]

The receiver watches the bus and waits for a break signal. This is usually recorded as an interrupt to the UART.

It then has a short time (the Mark After Break, MAB, interval) to relay itself for reception of the frame of DMX data.

Receivers then receive each slot in turn. It verifies the first slot is correctly formatted with two stop bauds and checks the value is a valid start code. The DMX Packet's startcode can be any value, but only frames with a recognised start code will be processed - all other frames will be ignored. The default ('0') startcode is used to send data. All receivers decode startcode 0, and may optionally decode other start codes.

Answer should note than any error in a slot's stop baud skips the remainder of the frame.




<diagrams or algorithm or pseudo-code needed (labelled to show 1,2>

(d) Estimate the maximum number of receivers allowed on the bus, given a nominal receiver input impedance of 12 k Ohms [5 marks]

Each receiver has an input impedance. This is 12 kOhms for EIA-485.

Thus, connecting two receivers would split the sender signal power between the two receivers. adding more receivers reduces the signal strength at each receiver.

32 parallel input impedances = 376 ohms - only slightly higher than the impedance of 120 Ohms

<method of calculation to be explained>

<An answer could also consider the bias circuit for RDM, but that was not requested >

(e) What is the effect of two 4 channel DMX fixtures being configured with a start address of 5? Explain how these process a received DMX frame to extract the values that represent each channel. [8 marks]

Explain role of the base address:

Each receiver on a DMX bus is allocated a base address that may be represented by a 9 bit binary number. The address determines the first slot that is used by the receiving device. The most common way is to use a Dual In Package (DIP) switch.

A common way for devices that require more frequent configuration is to store the base address in flash memory and use a 3 digit LED/LCD display to show the current value. The address is usually set using up and down switch inputs.

<diagrams and algorithm or pseudo-code to be probided>

Explain impact of two fixtures with same start address. Both search for the start of frame; verify the start code; and both will skip the first four data slots and extract the values sent in positions 5 ,6, 7, 8. The two fixtures with the same address will respond identically to frames sent on the control bus.

Explain how multiple slots are used:

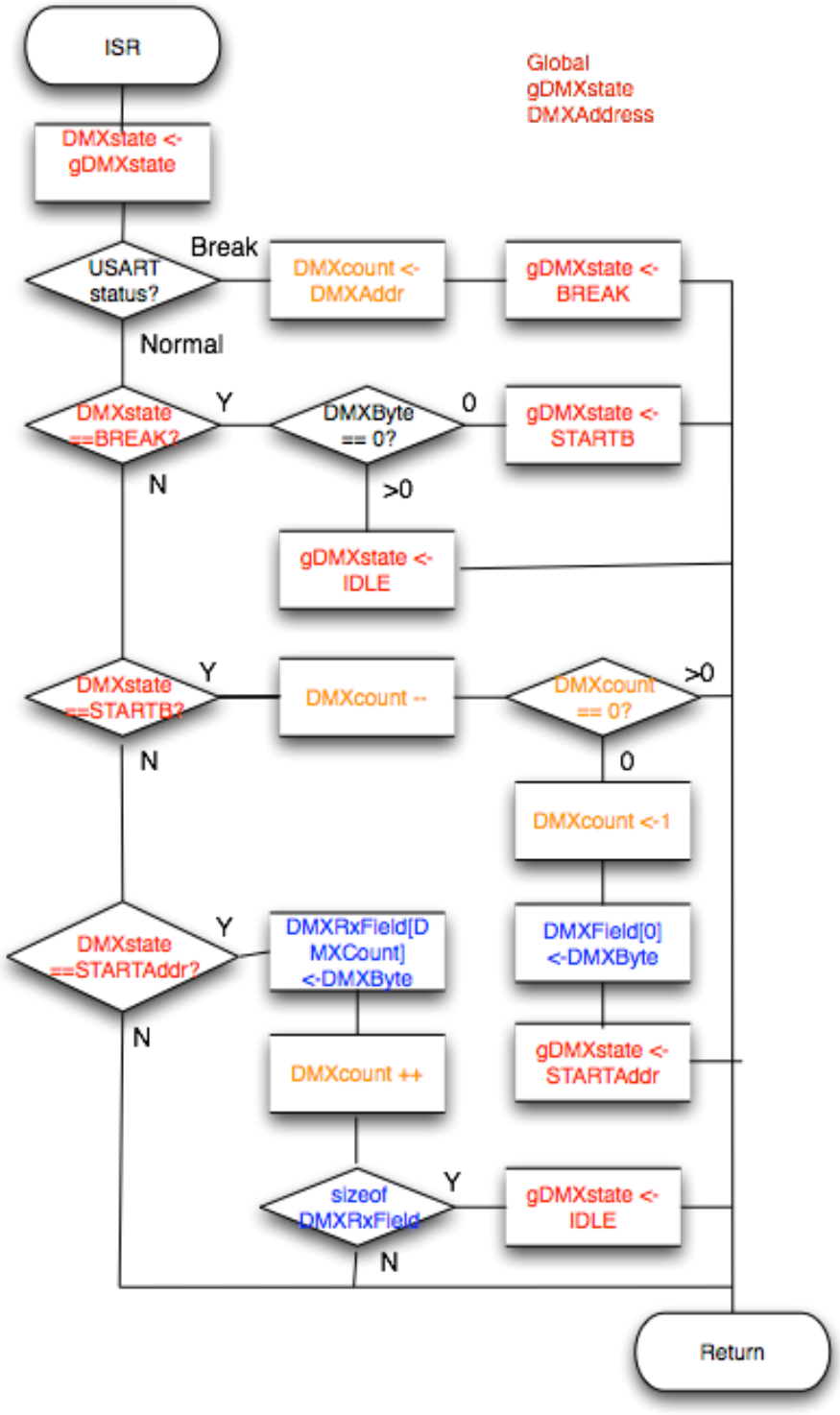
The key point is that the sender generates slots with a predefined format. The bus sends all slots without reference to any format.

The receiver is responsible for the interpretation of a contiguous set of slots starting at its own base address. This is configured to match the format for the receiver device by selecting a profile (the meaning of each slot in the contiguous set of slots). This profile is the same at both the sender and receiver.

The set of slots may be interpreted as 4 separate channels - e.g. each to drive a dimmer circuit with a value of one slot, 0-255. Alternatively, a combination of slots could be used together e.g. 5+6 could be a 16 bit value and 7+8 could represent a pair of channels with a 16 bit value, e.g. to drive the pan and tilt of a robotic arm.

(RDM provides a way for the sender to understand how the slots will be interpreted usually by discovering what device has been assigned to the base address).

## TUTORIAL 5: *DMX Microcontroller Algorithms*

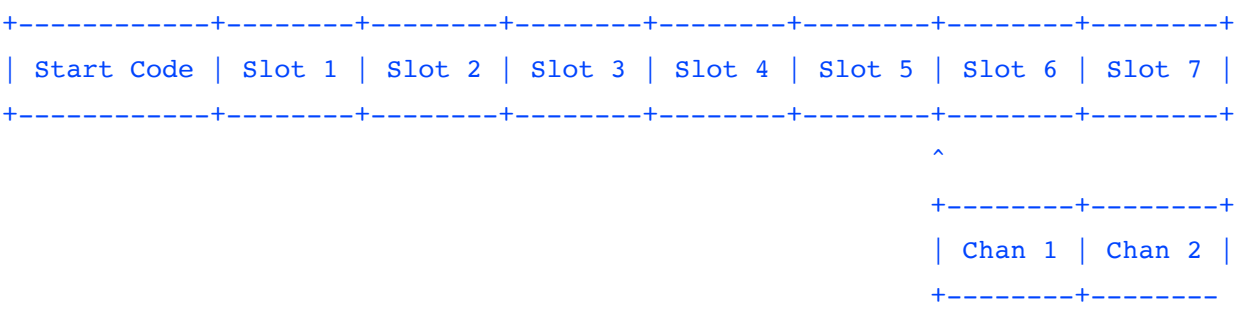



The algorithm captures a series of consecutive bytes form the DMXFrame to the array DMXRxField (also labeled DMXField above). A state variable (DMXstate) records progress through the different states in the receiver algorithm. Between interrupts the value of the state variable is stored in the global gDMXstate. The global variable DMXAddr contains the configured base address of the DMX receiver, often read from a DIP switch or supplied via the main routine (in some designs this could also be set by RDM control). The integer variable DMCcount is used to count the number of slots received. First to identify whether the base address has been received, and later to record the number of bytes in the DMXField to still be received.

Note that the flowchart assumes that reading the USART status register also takes the oldest received byte from the Receiver FIFO and makes this available in DMXByte. This means that in state “STARTB” the byte is read, but not used, effectively skipping the byte until the start address has been reached. A complete answer could refer to the DMX receiver state diagram.

(b)

Base address 6 + Channel 2 - Implies use of slot 7




Skip the start code and first 6 bytes (to the base address of the receiver). Slot 6 is the first receiver channel after the base address and slot 7 is the second receiver channel.

(c) The value of a channel has an initial DMX slot value of 0, later a value of 50 and then finally a value of 100. Sketch the waveform for each of the values that is used to ***drive*** a TRIAC dimmer circuit, and the resulting voltage on the mains power line.

Note values in the frame slots are slot values 0-255, not percentage:

0 -> TRIAC does not fire;

50/255 means the TRIAC fires at the end of each halfcycle and conducts until zero-crossing;

100/255 means it fires just after the midpoint of each half cycle and conducts until zero-crossing.

Students should sketch the firing waveform and the resulting mains waveform.

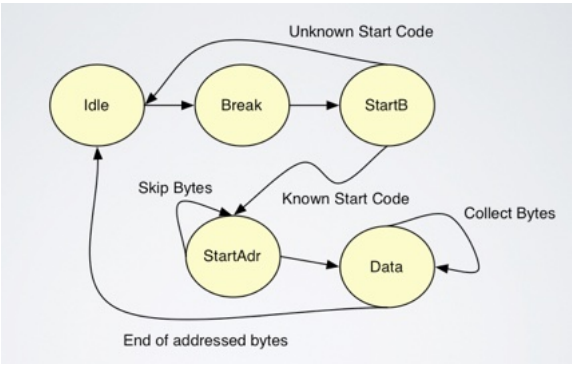
Firing of the triac needs to be synchronised relative to the end of each half mains cycle, allowing the triac to turn off each time there is no mains current.

*Additional thoughts:*

(i) What is the purpose of the returni statement? What does it do?

Returns from interrupt - restores the key registers to their value when interrupted. This re-enables interrupts, restores the programme counter (and any saved registers) - After execution, the next instruction executed follows the one that was interrupted. (This may be the main program, or possibly another lower priority interrupt service routine.)

(ii) Can you represent the algorithm as a state transition diagram? What variable is used to store the state?




The global is **gDmxState (note this diagram does not show transitions on a slot error such as missing stop bauds, which also causes the receiver to return to the Idle state and abort the frame currently being received.)**

(iii) Why is the code written so that the global state is copied to a local variable during the interrupt service?

This is a run-time optimisation. The variable is used many times - the copy to the local **DmxState** (possibly in a register) avoids the code having to access the global data area of memory each time this is checked. This improves the speed of execution.

## TUTORIAL 6: *DMX Receiver Processing*

*See worksheet for tracing DMX receiver execution.*

## TUTORIAL 7: *Remote Device Management (RDM)*

(a) Remote Device Management is an extension to DMX that allows configuration of remote devices and retrieval of data from them. What changes are needed in a device's electronics to enable RDM operation? [4 marks]

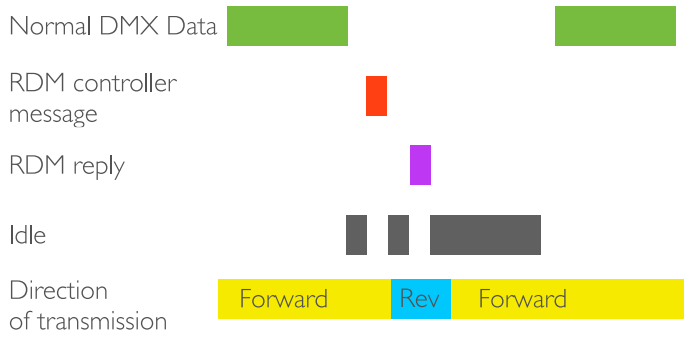
1) The line driver needs to support transmit and receive. An RDM device uses tri-state drivers to allow it to transmit and receive (actually in RDM the "third" high impedance state is never used – transceiver is either in send or receive mode).

To prevent a "data collision", the receiver hardware remains in a receive mode until explicitly set to transmit data (i.e. until the receiver is addressed by the bus master/control surface). This requires the local microcontroller to control the direction of each line driver. Any repeaters (splitters) used on the path to the receiver also need to support half-duplex operation.

2) The bus needs to be terminated at both ends. The master/control surface also needs to include a bias circuit to prevent the line "floating" to an arbitrary value when the sender changes. The bias network includes a pull-up and pull-down resistance and MUST include a DMX terminator.

(b) In RDM, the bus may be unpowered for periods of time. Explain how a bias circuit may be used to prevent the bus floating to an arbitrary value, and calculate the values of the bias components. [15 marks]

One device is assigned as the master by the person building the network – usually a control surface or PC with a RDM interface. This controls who can transmit to the bus.




The master initiates a communications request to a "slave" by addressing the unit and turning-off its own transmit hardware. The line driver of the master is in send mode.

The master then listens for a response (receive mode). The bias circuit ensures that cable noise is not seen as a signal during the time when there is no active sender.

The slave receiver recognises a control slot, and if addressed it enables its own transmitter.

Once data has been sent, slave reverts back to receive mode.

The bus master (controller) resumes control after reception from slave or a timeout. The timeout is essential for reliable operation to recover from corruption of the signal or when a receiver misses a command.

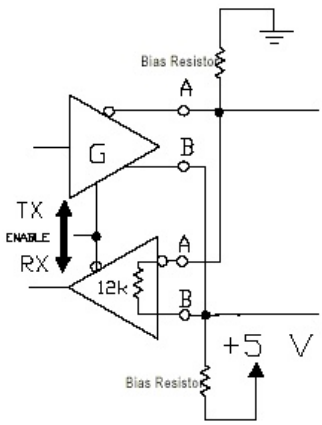
(c) In RDM, the bus may be unpowered for periods of time. Explain how a bias circuit may be used to prevent the bus floating to an arbitrary value, and calculate the values of the bias components. [15 marks]

The RDM specification says: “The command port shall provide a means to bias the termination of the data link to a value of at least 245 mV”

This value is a result of the receive driver, since the receiver thresholds is 200mV. This is set to avoid oscillation in the output when the signal varies (see the dotted output waveform below).

- The differential signal must cross upper threshold to count as a 1
- The differential signal must cross lower threshold to count as a 0

If there is no bias the lines “float”. The bias circuit therefore prevents noise from being seen as a signal, and takes the following form:




A connected via R to GND

B connected via R to +5V

Of course, only do this once for each bus!

Each RS-485 node has an input impedance of 12K.

32 nodes in parallel present load of 376 ohms.

Two 120 Ohm terminations add 60 Ohm load.

Total load is therefore 1/(0.0167+0.0026)=51.8 Ohms

The bias circuit is to maintain at least 245 mV between B A line, this needs a bias current of ~ 4.7 mA to flow through this load To create this bias from a 5V supply needs a series resistance of 1063 Ohms, subtract 51.8 Ohms from terminators, this leaves 1011 Ohms.

Placing half the resistance as a pull-up to 5V and half as a pull-down to ground gives a bias of 505 Ohms, 510 Ohms to nearest preferred value.

# TUTORIAL 8: *Controller Area Network (CAN) Bus*

(a) CAN uses a variant of differential transmission, what is the key difference to methods such as RS-485 and DMX? [5 marks]

CAN uses Open-Collector (O/C) logic, in which one of the values is signaled by no signal – i.e,, the bus is allowed to idle, driven only by the ***bus bias circuit***.

Logic 1 is said to be ***recessive***: No signal sent.

Transceiver output for CAN_L wire floats to 2.5V

Transceiver output for CAN_H wire floats to 2.5V

i.e. when sending a logic 1 is a no voltage difference measured at the receiver.

Logic 0 is said to be ***dominant***: Forces bus to a logic zero level

Transceiver output for CAN_L wire driven to 1.5V

Transceiver output for CAN_H wire driven to 3.5V

i.e. there is a 2V voltage difference measured at the receiver

If two nodes attempt to same the same value, the signal is not changed, providing they are synchonised in timing – which they should be. If the values differ, then clearly a Logic 0 dominates, since this drives the bus to the logic zero level. In this way the bus automatically overrides a '1' with a '0'. This is used in CAN bus for bus arbitration.

(b) Explain how CAN uses bit stuffing to achieve redundancy. [10 marks]

CAN is a ***synchronous*** technique, requiring a clock at the receiver to be synchronised to the transmitter clock. There are no start and stop bit as in asynchronous transmission. The receiver does this by ensuring the data contains a minimum set of transitions – just enough so that the receiver can "recover" the clock at the receiver using a digital phase locked loop.

This poses a problem – we need to allow the data bits (at the link layer) to have any value, and form a continuous sequence on the bus. However, at the physical layer we can not allow long runs of zeros or ones that would prevent receiver synchronisation.

The solution is to occasionally add extra (redundant) bits at the physical layer to ensure the ***synchronisation***. Senders and receivers therefore count the number of successive bits that are sent at the same logic level. A sender that sends 5 bits of the same polarity, then inserts (***stuffs***) an additional (***redundant***) bit before sending the next bit: The stuffing bit is one bit with the opposite polarity to the previous run of 5 bits. (The stuffing bit is not a part of the link layer message, it is only inserted to create physical layer timing)

A receiver that receives 5 bits of same polarity, deletes the following bit: The removed stuffing bit must be the opposite polarity (if it is the same polarity this is noted as an error, and the reception is aborted). The method is said to introduce ***transparency*** – ANY sequence of data may be sent at the link layer, and using the method the physical layer automatically adds/remove the stuffing to ensure the receivers always see a sequence of transitions.

Usually this results in a small overhead, but this can add up to one bit in five, introducing a maximum of 20% additional overhead.

(c) Consider two nodes with two message IDs: 415 and 455, what is the sequence for the first 12 bits if each is sent individually, and what is the resulting arbitration when the two messages are sent simultaneously? [10 marks]

Method:

The start of frame marker (an initial 0) is sent first (this is preceded by at least 3 consecutive 1's – the gap between frames). This is shown in black below.

The ID field is then 11 bits

First represent the IDs in binary, showing the least significant 11 bits: A,B below.

- Node A sends 415 (00110011111), shown in blue
- Node B sends 455 (00111000111), shown in black

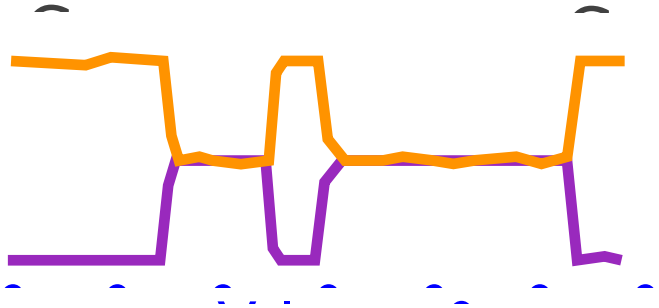
Then check bit by bit starting with the first bit on the wire whether the two bits match (this replicates the sender watching the bus).

If the A and B signals differ, then choose the'0' value (in this case A wins, and B stops sending).

Complete the rest of the transmission using the value chosen by A (B has stopped).

This is shown in increasing transmission order (first bit on the wire on the left) in the diagram below.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 0 | 0 | 0 | I | I | 0 | I | I | I | I | I | I |
| B | 0 | 0 | 0 | I | I | I | - | - | - | - | - | - |
| bus | 0 | 0 | 0 | I | I | 0 | I | I | I | I | I | I |



Representation of CAN bus signal on a scope.

The horizontal axis is time increasing left to right.

The vertical axis is voltage measured at the receiver. Note the CAN_H (3.5V) and CAN_L (1.5V) waveforms – and note their voltage as sent on the cable. At the sender the CAN_H and CAN_L signals differ by 2V for a logic 0 (dominant).

Note that for a recessive bit (logic 1) the two wires float to the midpoint (2.5V) value.

Finally note the signal is slew-rate limited and not a precise square wave – to match the transmit signal to the bandwidth of the cable.