How to Operate Serial Port Communication With The Stim Board

Michael Brodnik

Purpose:

The purpose of this paper is to help anyone who is attempting to develop code or utilize code already generated. This paper will walk you through how to communicate with the stimulation boards provided by Case Western Reserve University and allow you to output an appropriate amount of stimulation. In addition to reviewing this paper, Please refer to the Stand-alone Operation of Surface Stim Board pdf that is also located in this folder. This will provide an additional view of how to control the stim boards

Note: All pulse width parameters are converted from the bytes in usec (micro seconds)

All amplitude parameters are converted from bytes to mA (milli amps)

1. Serial port communication

   This section will explain the basics of "speaking" through a serial port. The coding I have done so far has used C++ for this purpose. The main library used is the Termios header file. Termios is a Linux system specific library for serial port communication and holds all of the necessary functions.

   1. Open()

      For this function, you set a variable (fd) equal to this function. Essentially, this will open the serial port name you call out and store this location into the variable
      Example: fd = open("/dev/ttyUSB1", O_RDWR | O_NOCITY | O_NDELAY);

   2. Cfsetispeed() & cfsetospeed()

      These functions will set the baud rate for the input and output for the serial ports. You must match the desired baud rate to that of the device you are communicating with.
      Example: cfsetispeed(&port_settings, B9600);
      cfsetospeed(&port_settings, B9600);

   3. Write()

      This function will send any numeric variable (letters and strings will be converted to their equivalent binary values) through the serial port.
      Example: write(fd, halt_rset, (sizeof(halt_rset)/sizeof(*halt_rset)));

2. Byte Strings for the Stimulation Boards

   At the beginning of your code, you must establish a schedule and then create events for that schedule. A schedule is an event organization system that allows the stim board to process new commands. An event is an action that the stim board will execute. The event can be used to

change pulse width, amplitude, and frequency. The stimulation board will continuously read the schedule and execute the events within that schedule.

The code you will be using / writing must have the following bytes within the code in order for the stim board to operate properly.

All bytes within the code will be assigned as “unsigned char”.

The first two bytes within the byte string will be 0x04 and 0x80; these are to remain static and should not be changed. 0x04 is the destination address and 0x80 is the source address. Essentially, this lets the board know where the “traffic” is coming from and leaving.

The next two bytes within the string will pertain specifically to the type of command you which to send out. The first one will be the message code and the following byte will be the size of the message. The following examples will help elaborate this methodology.

The final byte within the byte string is reserved for a checksum (ones compliment). This will allow the board to sum up all of the bytes and evaluate if the command you sent is truly valid. Later in this paper we will discuss more in detail the purpose of a checksum and how to evaluate one.

For your coding purposes, your code will have to write the byte strings in the following order.

Halt message (Optional)

Delete Schedule (Optional)

Channel Setups

Create Schedule

Create Events

Sync Message

Change event Parameters

These bytes are explained in more detail below.

1. Halt message

   Description: When a UECU module receives a Halt message, it stops all stimulus and data acquisition. The message may optionally reset the module's operating parameters to their power-on defaults.

   Halt is often sent as a high-priority broadcast message, in order to stop or initialize all modules in the UECU. This is useful when initializing and when shutting down a UECU. UECU modules receiving a Halt message will end their activities as quickly as practical.

   Events that were in progress when the Halt message is received may run to completion before the Halt is processed, but no new events will begin.

   The Halt message may also be useful when handling error messages, since it can be used to put a single module or an entire UECU system into a known state.

   Message code: 0x04
   Size: 0x01

   Example:
   unsigned char halt_rset[] = {0x04, 0x80, 0x04, 0x01, 0x01, 0x00};
   0x04 = Destination Address
   0x80 = Source Address
   0x04 = Message code
   0x01 = Size
   0x01 = Schedule number
   0x00 = Checksum placeholder

2. Delete Schedule Message

   Description: The DeleteSchedule message causes a UECU module to remove a schedule from its Scheduler and release all internal resources associated with that schedule and its events. A schedule may be removed when it is in any state, but the removal may not be immediate if the schedule is Active.

   Message: 0x12
   Size: 0x01

   Example:
   unsigned char del_sched[] = {0x04, 0x80, 0x12, 0x01, 0x01, 0x00};
   0x04 = Destination Address
   0x80 = Source Address
   0x12 = Message Code

0x01 = Size
0x01 = Schedule number
0x00 = Checksum placeholder

3. Channel Setup

Description: The Channel Setup message sets up a channel on a stimulation module. The maximum pulse amplitude and pulse width are set using this message. Then the rest of parameters are different depending on the type of module. For the Surface module each channel must have the anode and cathode specified. This allows for different configurations of bipolar and/or monopolar operation. The Interphase Delay and the Aspect Ratio are used to define the shape of the waveform.

Message: 0x12
Size: 0x01

Example:

```
unsigned char chan_set1[] = {0x04, 0x80, 0x47, 0x07, 0x00, 0x64, 0xFA, 0x00, 0x64, 0x11, 0x01, 0x00};
unsigned char chan_set2[] = {0x04, 0x80, 0x47, 0x07, 0x01, 0x64, 0xFA, 0x00, 0x64, 0x11, 0x23, 0x00};
unsigned char chan_set3[] = {0x04, 0x80, 0x47, 0x07, 0x02, 0x64, 0xFA, 0x00, 0x64, 0x11, 0x45, 0x00};
unsigned char chan_set4[] = {0x04, 0x80, 0x47, 0x07, 0x03, 0x64, 0xFA, 0x00, 0x64, 0x11, 0x67, 0x00};
```

0x04 = Destination address
0x80 = Source address
0x47 = Message code
0x07 = size
0x00 = Naming the channel
0x64 = Amplitude Limit
0xFA = Pulse Width Limit
0x00 = First byte for the Interphase Delay
0x64 = Second byte for the Interphase Delay
0x11 = Aspect Ratio (Equal catholic and anodic amplitude)
0x01 = Bipolar channel for channel 1 (0x23 refers to bipolar channel for channel 2, etc.)
0x00 = Checksum placeholder

4. Create Schedule Message

Description: The CreateSchedule message makes a new schedule. When this message is received, a UECU module allocates memory for the schedule and returns a CreateScheduleReply with a one-byte number, the schedule descriptor, that may be used to manipulate the new schedule.

Message: 0x10
Size: 0x03

Example:
```
unsigned char crt_sched[] = {0x04, 0x80, 0x10, 0x03, 0xAA, 0x00, 0x1D, 0x00};
```

0x04 = Destination address
0x80 = Source address
0x10 = Message code
0x03 = Size
0xAA = Sync Signal
0x00 = Place Holder for Refresh (repeat) rate
0x1d = Repeat every 0x001d msec
0x00 = Checksum placeholder

5. Create Event Message

Description: CreateEvent is sent to a UECU module to add a new event to a schedule. Many events are possible. See the Events.h, for a list of events and their parameters.
The source address of the CreateEvent message is used as the initial owner of the event. In other words, the module that sends the CreateEvent message will be sent all messages related to the event, including data replies and error messages. The owner can be changed to any address in the system with the ChangeEventOwner message.

Message Code: 0x15
Size: 0x06 + event parameters

Example:
```
unsigned char crt_evnt1[] = {0x04, 0x80, 0x15, 0x09, 0x01, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00}
unsigned char crt_evnt2[] = {0x04, 0x80, 0x15, 0x09, 0x01, 0x00, 0x05, 0x00, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00}
unsigned char crt_evnt3[] = {0x04, 0x80, 0x15, 0x09, 0x01, 0x00, 0x0A, 0x00, 0x03, 0x02, 0x00, 0x00, 0x00, 0x00}
unsigned char crt_evnt4[] = {0x04, 0x80, 0x15, 0x09, 0x01, 0x00, 0x15, 0x00, 0x03, 0x03, 0x00, 0x00, 0x00, 0x00}
```

0x04 = Destination Address
0x80 = Source Address
0x15 = Message code
0x09 = Size (0x06 + 0x03 for stimulus event)
0x01 = Schedule ID
0x00 = Placeholder for delays
0x00 =Spacing events 5msec apart (Note how the 6th byte is different from each event, this spaces out the commands sent out to the stim board)
0x00 = Priority (0x00 = not implemented)
0x03 = Event type (0x03 means stimulus event)
0x00 = Port channel (0x00 refers to channel 1, 0x01 refers to channel 2, 0x02 refers to channel 3, 0x03 refers to channel 4)
0x00 = Pulse width set (Change not implemented)
0x00 = Amplitude set (change not implemented)
0x00 = Zone (0x00 not implemented)
0x00 = Checksum placeholder

6. Sync Message

Description: The Sync message causes schedules to start running. It is generally broadcast to all modules in a UECU system. All schedules with the specified SyncSignal will start running when the Sync message is received. The Sync message is ignored by any modules that have no schedules using the specified SyncSignal

Message Code: 0x1B
Size: 0x01

Example:
```
unsigned char sync_msg1[] = {0x04, 0x80, 0x1B, 0x01, 0xAA, 0x00}
```

0x40 = Destination address
0x80 = Source address
0x1B = Message Code
0x01 = Size
0xAA = Specified Sync signal specified within the create schedule message
0x00 = Checksum placeholder

7. Change Event Parameters Message

Description: The Change Event Parameters message is sent to a module to change the operating parameters for an event

Message Code: 0x19
Size: 0x01 + event parameters

Example:

```
unsigned char chngevnt1[] = {0x04, 0x80, 0x19, 0x04, 0x01, 0x00, 0x3C, 0x00, 0x00};
unsigned char chngevnt2[] = {0x04, 0x80, 0x19, 0x04, 0x02, 0x00, 0x3C, 0x00, 0x00};
unsigned char chngevnt3[] = {0x04, 0x80, 0x19, 0x04, 0x03, 0x00, 0x3C, 0x00, 0x00};
unsigned char chngevnt4[] = {0x04, 0x80, 0x19, 0x04, 0x04, 0x00, 0x3C, 0x00, 0x00};
```

0x04 = Destination address
0x80 = Source address
0x19 = Message code
0x04 = size (0x01 + 0x03 for stimulus event)
0x01 = Event ID (0x01 is for channel 1, 0x02 is for channel 2, 0x03 is for channel 3, 0x04 is for channel 4)
0x00 = Pulse width parameter (sets the pulse width for the event)
0x3C = Amplitude parameter (sets the amplitude for the event)
0x00 = Placeholder for parameters
0x00 = Checksum placeholder