Dexbridge2

Dexbridge2 is Pololu Wixel code that can act as a bridge between a Dexcom G4 Transmitter and a smart phone using Bluetooth 4.0 (BLE). It requires that the Wixel be connected to a HM-10 module, using the design originally put together by Stephen Black for his DexDrip system. There is a minor hardware modification required in order to receive the bridge battery voltage, for monitoring in the app, and an extra wire that allows the HM-10 to be put to sleep and woken with the wixel low power mode. Please see the Circuit diagrams below.

The original wixel code for DexDrip was based on work by Adrien de Croy, Lorelai Lane, and others, but it has some limitations. This bridge code has the following features to address these:

- Does not receive any Dexcom packets until it has been given a Dexcom G4 Transmitter ID to filter on. This is important to ensure that it correctly locks on to the signal of the transmitter in question, and only passes that Transmitter ID's data to the phone app.
- Stores the Transmitter ID in flash, so that it survives if power fails for any reason. The app does not need to reset this in such an event.
- Sends a "beacon" to the app when it wakes up from low power mode, indicating the Transmitter ID it is filtering on. The app can either send a TXID packet if the Transmitter ID the wixel should be filtering on is incorrect, or ignore the beacon. This is necessary to ensure that when a patient changes their transmitter, the app knows when the wixel is awake so it can be sent a new TXID packet. Normally, the wixel will wake up 5 minutes after the previous good packet and stay awake until it receives a packet from the Dexcom transmitter it is configured to filter on.
- Sends a "beacon" to the app every 5 seconds if the code is newly installed and the wixel has NOT received a TXID packet. It will continue to send beacon packets at 5 second intervals until a TXID packet is received and the Dexcom transmitter ID is saved to flash. The beacon will indicate the ID to be zero, and the app, once configured with a Dexcom Transmitter ID, must send a TXID packet before the code will start accepting packets. The wixel will respond to the TXID packet with a "beacon" packet indicating the TXID that is set in flash on success.
- Accepts a Transmitter ID packet from the phone app, and saves it to flash. Note, the phone app must await either a data packet or a beacon packet before determining if the Transmitter ID is incorrect, and sending a TXID packet to the bridge.
- Sends the Bridge battery voltage as part of the data packet.
- Automatically corrects the packet "listen window" for changes in overall program cycle time. This ensures that any changes to the code does not require any further experimentation to the listen window to make it work reliably.
- Sets the HM-10 module's BLE ID to "Dexbrdg-XX", where XX is the least significant byte of the wixel serial number. This ensures that each bridge has a unique ID, making the BLE connection to the phone running the app more reliable.

Protocol

Each packet of data sent or received by the bridge is described below. Common to each packet are the first two 8bit bytes. The first byte is the length of the packet in bytes. The second is an ID for the type of packet being sent.

Data Packet

A Data packet is sent by the wixel to the phone app. It contains the relevant data sent from the Dexcom G4 Transmitter, plus the bridge battery voltage and TxID it is filtering on.

The data packet has the following structure:

Byte	Value	Data Type	Description
0	0x10	8 bit unsigned integer	Number of bytes in the packet (16)
1	0x00	8 bit unsigned integer	Code for Data Packet
2:5	Raw Signal	32 bit unsigned integer	Raw Sensor signal
6:9	Filtered Signal		Filtered Sensor signal
10	Dexcom Tx Battery Voltage	8 bit unsigned integer	The Transmitter battery voltage. Usually around 214 for a new transmitter. The app should alert if this reaches <= 207, that the transmitter requires replacement.
11	Bridge Battery Percentage	8 bit unsigned integer	The bridge battery percentage (0-100). This is calculated from the VIN voltage using a 10M/1M resistive voltage divider on P0_0. VIN of 2.8V (equivalent to 255mV input on P0_0) is 0%, as this is the lowest operating voltage of the Wixel (HM-10 is lower, as is the Battery output cut off). VIN of 4.2V (equivalent to 521mV input on P0_0) is 100%, as this is the maximum charge voltage delivered by the Adafruit charger.
12:15	Dexcom TxID	32 bit unsigned integer	Encoded Dexcom Transmitter ID that the bridge is filtering on.

Upon receiving this packet, the phone app has to process it, taking the parts of the packet it will use.

If the app determines that the Dexcom TxID is different to its own setting, it should immediately send a TXID packet back to the bridge, and ignore the packet.

If the app is happy with the Dexcom TxID sent, it should accept the packet and immediately send back an acknowledgement packet. The acknowledgement packet will immediately tell the wixel to go into low power mode.

The acknowledgement packet structure is as follows:

Byte	Value	Data Type	Description
0	0x02	8 bit unsigned integer	Number of bytes in the packet (2)
1	0xF0	8 bit unsigned integer	Code for Data Packet

Note that the wixel will otherwise go into low power mode if it does not receive an acknowledgement or TXID packet within 3 seconds of transmitting a data packet.

TXID packet

The TXID packet is sent from the phone app to the bridge to set the bridge to filter on a single Dexcom G4 transmitter ID. This is important to ensure the bridge correctly “locks” to the correct transmitter for a patient, and also to ensure the app only receives packets from the transmitter of the patient it is monitoring.

The structure of the TXID packet is as follows:

Byte	Value	Data Type	Description
0	0x06	8 bit unsigned integer	Number of bytes in the packet (6).
1	0x01	8 bit unsigned integer	Code for Data Packet
2:5	TxID	32 bit unsigned integer	Encoded 32 bit integer representing the Dexcom G4 Transmitter ID that the bridge is filtering packets on.

Beacon packet

The Beacon packet is sent from the bridge to the phone app to indicate which Dexcom G4 Transmitter ID it is filtering on. The app can use this beacon to know when the bridge is active, and if it has a different Transmitter ID to what the app is configured for, it can correct this by sending a TXID packet.

The structure of the Beacon packet is as follows:

Byte	Value	Data Type	Description
0	0x06	8 bit unsigned integer	Number of bytes in the packet (6).
1	0xF1	8 bit unsigned integer	Code for Data Packet
2:5	TxID	32 bit unsigned integer	Encoded 32 bit integer representing the Dexcom G4 Transmitter ID that the bridge should filter packets on.

Note, this packet also doubles as the acknowledgement packet for a TXID packet. When the app receives this packet it can be sure that this is the Transmitter ID value set in the wixel flash memory.

Decoding and Encoding a Transmitter ID Long Int

In order for the app to send the correct value in a TXID packet to the bridge, you need to be able to encode the text of the Transmitter ID to a long int. This is done using the following pseudo code, taken directly from the original dexbridge code. Your app will need to replicate this process in order to send the correct data.

```
char SrcNameTable[32] = { '0', '1', '2', '3', '4', '5', '6', '7',
                          '8', '9', 'A', 'B', 'C', 'D', 'E', 'F',
                          'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P',
                          'Q', 'R', 'S', 'T', 'U', 'W', 'X', 'Y' };


/* asciiToDexcomSrc - function to convert a 5 character string into
a unit32 that equals a Dexcom transmitter Source address.  The 5
character string is equivalent to the characters printed on the
transmitter, and entered into a receiver.
      Parameters:
            addr  -     a 5 character string. eg "63GEA"
      Returns:
            uint32-     a value equivalent to the incodeded Dexcom
Transmitter address.
Uses:
      getSrcValue(char)
      This function returns a value equivalent to the character for
encoding.
      See srcNameTable[]
*/
uint32 asciiToDexcomSrc(char addr[6])
{
      // prepare a uint32 variable for our return value
      uint32 src = 0;
      // look up the first character, and shift it 20 bits left.
      src |= (getSrcValue(addr[0]) << 20);
      // look up the second character, and shift it 20 bits left.
      src |= (getSrcValue(addr[1]) << 15);
      // look up the third character, and shift it 20 bits left.
      src |= (getSrcValue(addr[2]) << 10);
      // look up the fourth character, and shift it 20 bits left.
      src |= (getSrcValue(addr[3]) << 5);
      // look up the fifth character, and shift it 20 bits left.
      src |= getSrcValue(addr[4]);
      //printf("asciiToDexcomSrc: val=%u, src=%u\r\n", val, src);
      return src;
}

/* getSrcValue   -     function to determine the encoding value of a
character in a Dexcom Transmitter ID.
Parameters:
      srcVal     -     The character to determine the value of
Returns:
      uint32     -     The encoding value of the character.
*/
uint32 getSrcValue(char srcVal)
{
      uint8 i = 0;
```

```
	for(i = 0; i < 32; i++)
	{
			if (SrcNameTable[i]==srcVal) break;
	}
	//printf("getSrcVal: %c %u\r\n",srcVal, i);
	return i & 0xFF;
}
```

Decoding a long integer transmitter ID is far simpler. You may implement a similar piece of code if you are storing the ID as a long int, but wish to display the text equivalent.

```
// convert the passed uint32 Dexcom source address into an ascii
string in the passed char addr[6] array.
void dexcom_src_to_ascii(uint32 src, char addr[6])
{
	//each src value is 5 bits long, and is converted in this way.
	addr[0] = SrcNameTable[(src >> 20) & 0x1F];	//the last
character is the src, shifted right 20 places, ANDED with 0x1F
	addr[1] = SrcNameTable[(src >> 15) & 0x1F];	//etc
	addr[2] = SrcNameTable[(src >> 10) & 0x1F];	//etc
	addr[3] = SrcNameTable[(src >> 5) & 0x1F];	//etc
	addr[4] = SrcNameTable[(src >> 0) & 0x1F];	//etc
	addr[5] = 0;	//end the string with a null character.
}
```

Note on Promiscuous mode

In this code, if the wixel is NOT sent a TXID packet, it will NOT collect packets from any Dexcom Transmitter and pass them to the smartphone app. This is a safety feature and is by design. You do not want an app displaying or storing data from anyone else’s transmitter.

However, in the part of the code that collects packets, promiscuous mode is allowed.

If you really wish to use promiscuous mode, comment out the section in main() that is clearly commented as the section that sends beacons until a TXID packet sets the transmitter ID. Then simply never send a TXID packet.

Basic flow of communications

From start up after code is loaded on the wixel, the dexbridge2 code will begin sending beacon packets at 5 second intervals on UART1 and USB (if it is connected). To break this cycle, a TXID packet must be received on either UART1 or USB (if connected).

Once the wixel has received a TXID and saved the info to flash, it will begin scanning for packets from that transmitter.

When the wixel receives a packet, it will send a data packet on UART1 or USB (if connected).

The receiving app must process the packet, and if the TXID sent in the packet is valid, it may send back a data ACK packet, which will immediately send the wixel to sleep for a period of time until before the next packet is due.

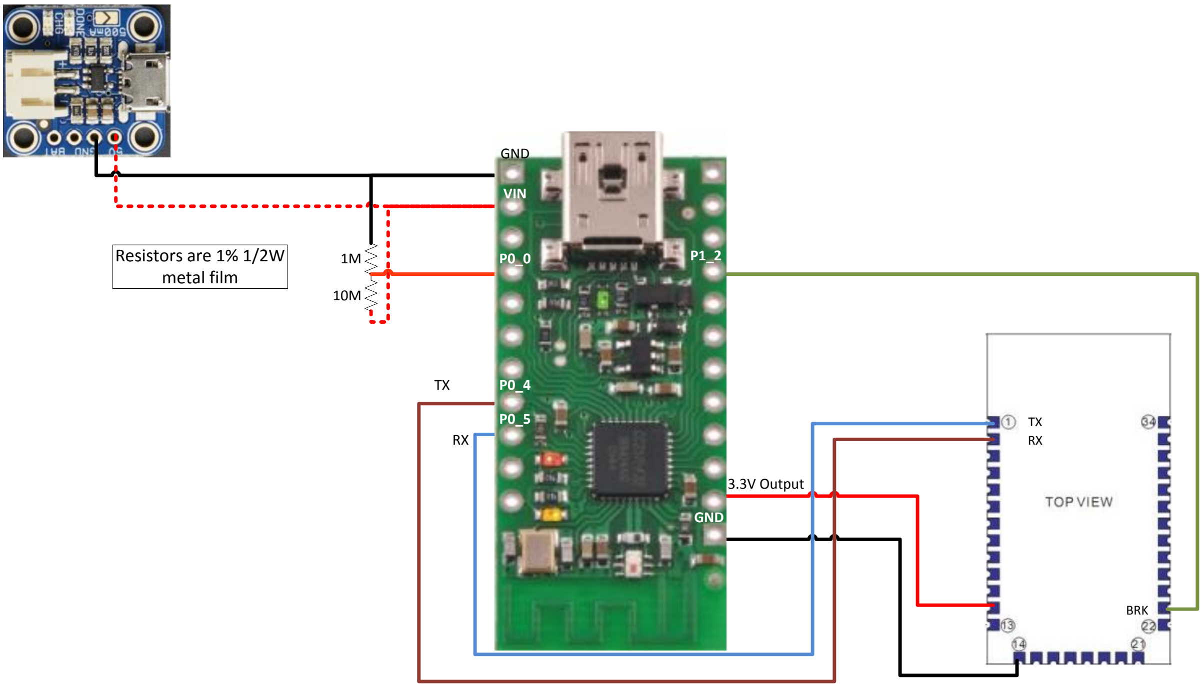
If the TXID in the data packet is incorrect, the app must send back a TXID packet to the wixel to set it to the correct ID.

On waking up, the wixel will send a beacon packet. The app must process this packet, and ignore the beacon packet if all is good, or send a TXID packet if the beacon contains the wrong ID. This ensures that when a patient changes their transmitter ID in the app, the wixel can be updated as soon as it wakes, and before it receives a packet from the new transmitter. If no beacon was sent when the wixel woke, the wixel would simply loop indefinitely until it received a packet from a transmitter that was no longer functioning.

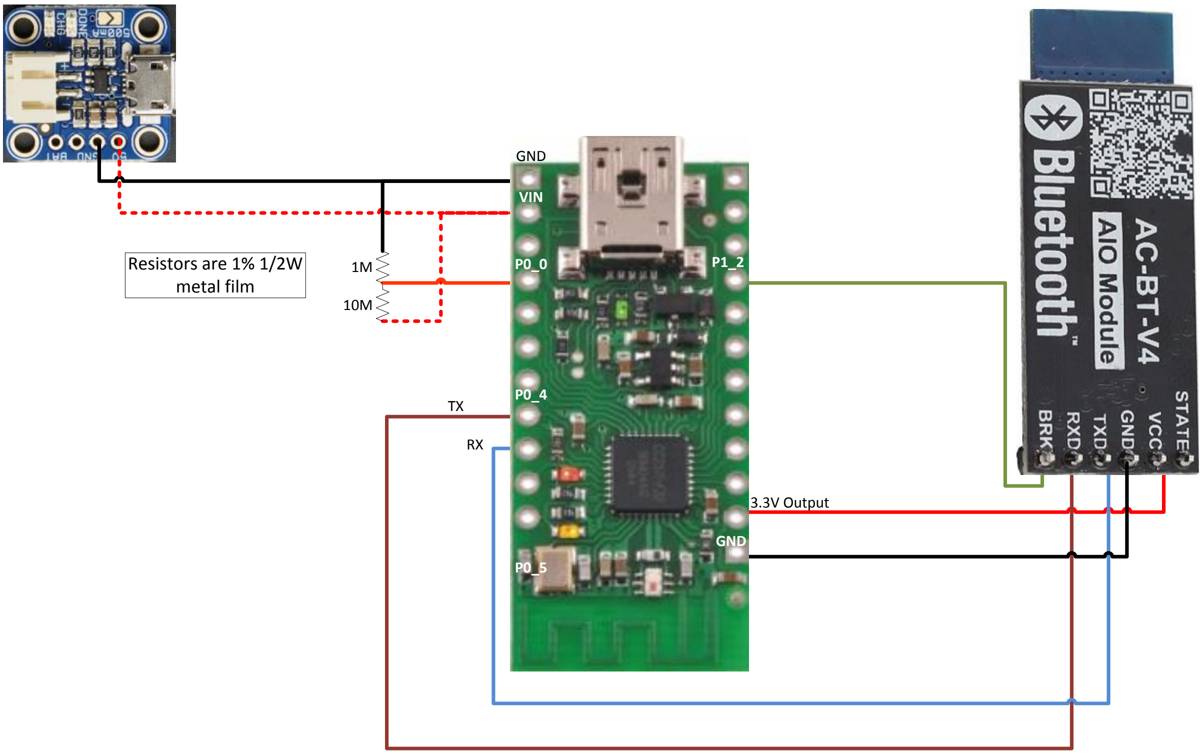
Circuit diagrams

Note, currently these circuits have **NOT** been tested.

Bare HM-10 connected to Wixel and Adafruit charger board.



HM-10 on support board, with Wixel and Adafruit charger board.



Updating the HM-1x module firmware

In order to ensure consistent operation of Dexbridge and the HM-1x modules, it is recommended that once you assemble the hardware, you immediately update the firmware of the HM-1x module you are using. This is necessary, because they are often delivered with one of two SoC devices (cc2540 or cc2541), and various levels of firmware depending on the source.

To rectify this, upgrade to at least level V534. This is the level that the wixel code for Dexbridge has been tested against.

Prerequisites

Firstly, you will need to either obtain the compiled usb_serial.wxl file (planning to be part of the release of Dexbridge), or compile it using the wixel-sdk. Connect your wixel to a computer via USB, and use the Wixel Configuration Utility to load this onto your wixel.

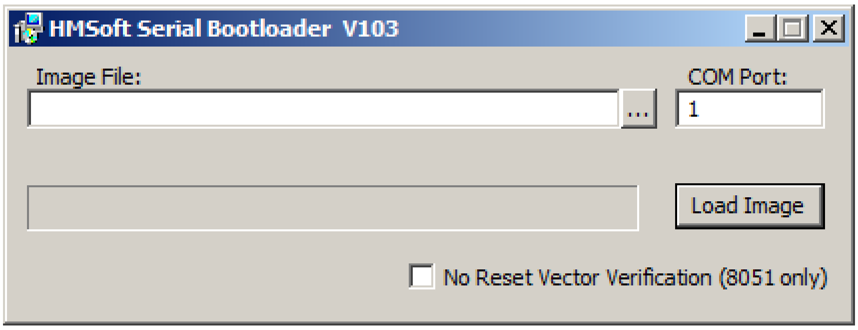
Secondly, you will need a terminal program. There are two options here. Either use a standard serial terminal program (PuTTY, Hyperterminal, etc) or an Arduino terminal program. The Ardiuno terminal program is probably the preferred option, but a standard serial terminal program can be used, it is just slightly more fiddly.
<u>When using a standard terminal program, like PuTTY, you will need to **PASTE** the command into the terminal window.</u> This is because the HM-1x modules DO NOT accept a CR or LF character. They only detect a delay at the end of the command string and try to execute it. If you try to type the command into the terminal program, it will ignore you as you cannot type quickly enough to get the whole command in.

Thirdly, go to http://www.jnhuamao.cn/download_rom_en.asp?id=# and download the V534 firmware zip file that is correct for the SoC on your HM-1X device. Use a magnifying glass to read the model number on your module. It should be either CC2540 or CC2541. If you are going to be making a few of these bridges, download both. You never know which one you are going to get. Unzip the contents into a folder. You will have a readme.txt file, a HMSoft.bin file (the firmware), and a HMSoft.exe file (the firmware updating utility).

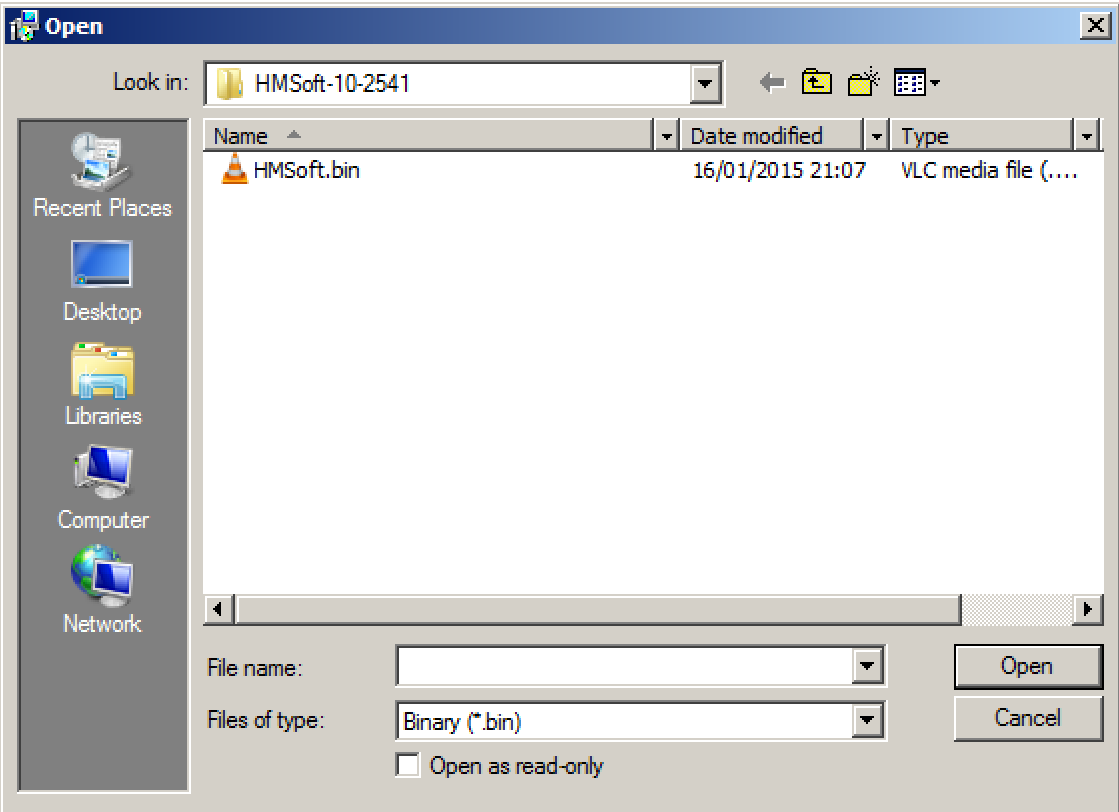
Updating steps.

1. Ensure you have the wixel attached to a PC via USB.
2. Ensure you know what COM port it is represented as. You can use the Wixel Configuration Utility to find this out.
3. Ensure you have installed the usb_serial.wxl program on the wixel.
4. Open your terminal program.
5. Send “AT” to the wixel. The response should be “OK”.
6. Send “AT+SBLUP” to the wixel. It will respond with “OK+SBLUP”. It is now ready and waiting to accept the firmware update.

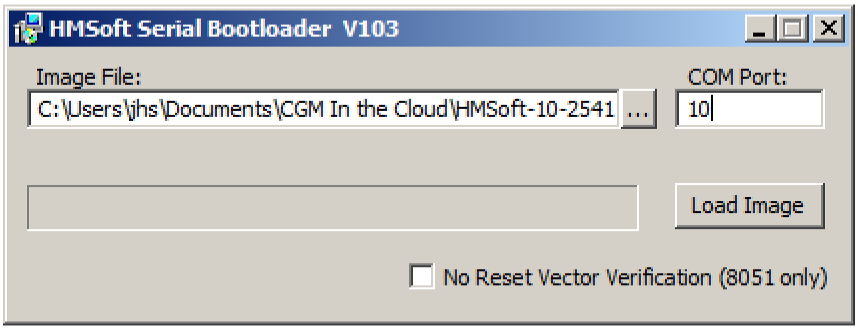
7. Open the HMSoft.exe file. It will open a window like this.



8. Click on the button to the right of the “Image File:” text box to navigate to the image file. Below is an example.



9. Enter the COM port number. For example, my wixel in this case is on COM 10, so my window will look like this.



10. Click the “Load Image” button. The progress bar will show the progress. The process takes a few minutes to complete. Once it has finished, the HM-1x module will be updated to the required level. Close the firmware update (HMSoft Serial Bootloader) window.

11. To verify the Firmware level of the module, open your terminal program once again. Send the command “AT+VERR?”. You will see a response like “OK:HMSoft v534”
12. That is it. You can now load the dexbridge2.wxl file using the Wixel Configuration Utility, and the bridge is ready for use.