XBridge2

XBridge2 is firmware code for a Pololu Wixel, and additional circuitry, that can make the wixel 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 BLE module, using the design originally put together by Stephen Black for his XDrip system as a minimum. There is a minor hardware modification required in order to receive the bridge battery voltage for monitoring in the app, and a modification to the wiring that allows the HM-1x to be powered down with the wixel low power mode, and powered up when the wixel wakes. Please see the Circuit diagrams below.

NOTE: This wixel app can be used with a classic xDrip-wixel wiring, but you will not have the full battery saving features the code can provide. The xBridge2 code will automatically detect if you are using a classic xDrip-wixel circuit, and automatically detect your HM-1x module baud rate.

Acknowledgements

The original code, engineering and decoding of the Dexcom data packet was done by Adrien de Croy, whithout whom this project would not be possible.

The inspiration to continue came from Lorelei Lane, Jason Calabrese, and others of the Nightscout community and foundation.

The original XDrip circuit upon which this work is based was engineered by Stephen Black, who also created the XDrip app.

The battery monitoring circuit was designed by Chris Botelho.

The method of powering down and powering up the HM-10 module was engineered by Gabor Szakacs.

Also, I wish to acknowledge the work of “lumos” and “slasi” of the Pololu Wixel forum for their work in Sleeping the wixel and radio correctly. I have made copious use of “slasi”’s radio_mac library, and used some of the code in “lumos”’ sleep library in the xBridge wixel code.

Overview

As the name suggests, this device acts purely as a bridge between the Dexcom G4 Transmitter and a Bluetooth Low Energy capable device, such as a mobile phone or tablet. It retransmits the data obtained from a Dexcom G4 transmitter packet. It DOES NOT ALTER OR RECALCULATE that data. It performs no math or translation of the raw data in the packet to an estimated Blood Glucose value.

It should be noted that only three pieces of data are retransmitted from the Dexcom G4. These are the two 32bit values representing blood glucose data, and an 8 bit value representing the battery condition in the G4 transmitter.

It adds information about the XBridge battery, as a percentage of remaining capacity, and the Dexcom G4 Transmitter ID that it is filtering on.

<u>This device and software, and any companion software developed for use with it, MUST NOT be used for making therapeutic decisions about diabetes management. Always use your standard Blood Glucose devices and management strategies. This is for informational and experimental use only.</u>

The original wixel code for XDrip 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. IMPLEMENTED (v2)
- 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. IMPLEMENTED (v1)
- 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. IMPLEMENTED (v2)
- 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. IMPLEMENTED (v2)
- 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. IMPLEMENTED (v1)
- Sends the Bridge battery capacity percentage, determined from the battery voltage, as part of the data packet. IMPLEMENTED (v2)
- 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. NOT YET IMPLEMENTED
- Sets the HM-10 module's BLE ID to "XBridgeXX", where XX is the least significant byte of the wixel serial number. This ensures that each bridge has a reasonably unique ID, making the BLE connection to the phone running the app more reliable. IMPLEMENTED (v2)
- Will not sleep if the wixel is connected to a PC USB port. This allows experimentation or re-scanning for the BLE device by an ap, if for any reason it is lost or the app is replaced, while the bridge has been programmed with a transmitter

ID. It also allows for packet capture by a PC for analysis of performance. IMPLEMENTED (v2.1)

Version History

1	Experimental and Proof of concept. Used to refine the Packet Capture code, develop flash storage of the Transmitter ID, and general operation of the wixel as a bridge device. Text only protocol and commands.
2	Consumer ready version. All packets are retransmitted as binary packets. Implemented: • Wixel Power Mode 2 sleeping • Wixel not sleeping on USB connected to PC. • Battery monitoring • HM-10 module power down • Protocol Functional Level 1

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.

Note that packets sent from the bridge to a phone app will include a last byte representing the Protocol Functional Level that the wixel firmware is programmed to. This was requested by Stephen Black, and makes sense. If more protocol functionality is added in future, any app using the bridge will need to know how to deal with various versions of firmware/protocol level in the wixel.

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	0x11	8 bit unsigned integer	Number of bytes in the packet (17)
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 10k/2k7 resistive voltage

			divider on P0_0. VIN of 3.2V (equivalent to 865mV input on P0_0) is 0%, as this is the lowest operating voltage of the battery. VIN of 4V (equivalent to 1081mV input on P0_0) is 100%, as this is the maximum voltage delivered the battery stably returns to once power is removed from the battery.
12:15	Dexcom TxID	32 bit unsigned integer	Encoded Dexcom Transmitter ID that the bridge is filtering on.
16	XBridge Protocol Level	8 bit unsigned integer	XBridge protocol level. Indicates the protocol level of xBridge. Note, that currently this will be 0x01.

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

This packet does not include indication of the XBridge protocol functional level, as it is from app to bridge.

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.

Both the Data Packet and Acknowledgement packet are part of Protocol Functional Level 1 (0x01)

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.

Note: as this packet is sent from the app to the phone, it does not include a XBridge protocol level byte.

The TXID packet is part of Protocol Functional Level 1, although the app does not send this byte tag to the bridge device.

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.
6	XBridge Protocol Level	8 bit unsigned integer	XBridge protocol level. Indicates the protocol level of xBridge. Note, that currently this will be 0x01.

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.

The Beacon packet is part of Protocol Functional Level 1.

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 xBridge 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 15 bits left.
      src |= (getSrcValue(addr[1]) << 15);
      // look up the third character, and shift it 10 bits left.
      src |= (getSrcValue(addr[2]) << 10);
      // look up the fourth character, and shift it 50 bits left.
      src |= (getSrcValue(addr[3]) << 5);
      // look up the fifth character
      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 code operations

The Wixel code uses the following steps when it wakes up or is powered on.

- Initialises all the base IO and Wixel systems.
- Waits for 30 seconds, processing IO. This allows you to connect a serial terminal to it via USB, so you can see what is going on.
- Loads the internal Flash stored flags register to determine if it has set up the HM-1x module. It only does this the first time it is run after the XBridge code is loaded. No point doing it too often.
- If the flag shows that the HM-1x is NOT configured, it will configure it.
- Loads the Battery Capacity limits from Flash. If these are not set up, it sets them to default values.
- Loads the TXID value from Flash.
- Powers up the HM-1x module.
- If the TXID value is NOT set, it goes into a loop, waiting for a BLE connection, and then sends Beacon packets until it gets a TXID packet. It then stores all the settings to flash, and moves on.
- Goes into a loop where it listens for Dexcom Transmitter packet.
- If a Dexcom Transmitter packet is not received in 320 seconds (5 minutes 20 seconds), it sends another beacon packet and tries again. The Wixel will NOT sleep until it has captured a packet.
- Once it has captured a packet, it waits for a BLE connection to be established between the HM-1x and the phone. It then sends the Data packet.

- If an ACK packet is not received in 10 seconds, it resends the Data packet.
- If an ACK is received, it saves all the settings back to flash, turns off the HM-1x module, sets all its IO to low to save power, and goes to sleep for around 255 seconds.
- Upon waking from sleep, it repowers the HM-1x, restarts it's radio, and begins waiting for packets again.
- At any point in time, a packet from the app or a packet or command from the USB can be received. It is processed immediately, and the program returns to where it last was.

Basic flow of communications

From start up after code is loaded on the wixel, the xBridge2 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 Packet and saved the info to flash, it will begin scanning for Dexcom packets from that transmitter.

When the wixel receives a Dexcom data 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.

The wixel will send a Beacon Packet, if it has not received a Data Packet within 5 minutes and 30 seconds of waking. 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 possible after 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.

The reason that we have had to send beacons this way is for two reasons. First, the BLE implementation in Android does NOT cleanly detect a closed connection, and retry connection at a fast enough rate to guarantee that it will connect immediately the HM-1x module is powered on by the Wixel. Otherwise, we would send the beacon as soon as the Wixel wakes and powers the HM-10 on. Any delay between wake and beginning packet capture could prevent a dexcom packet being missed.

Secondly, we do not want to interrupt the packet capture, so it is best to set a limit on packet capture, stop and send the beacon, and then start packet capture again. So, allowing packet capture to run a little longer than the anticipated 5 minutes means we are less likely to miss a packet capture.

Parts List

The minimum parts required to make a working xBridge circuit are:

1x Pololu Wixel

1x Adafruit Micro Lipo Charger with Micro USB Jack (PRODUCT ID: 1904) or equivalent.

1x 10k 1% 1/2W metal film resistor.

1x 27k 1% 1/2W metal film resistor

1x HM-1X (HM-10 or HM-11) based BLE module.

1x 3.7V Lithium Polymer (LiPo) battery. The capacity is your choice, but recommend 1200mAh for 5 days or more continuous operation. This should come with a JST connecter already fitted, which plugs into the JST socket on the charger board.

As a guideline to how long you can expect a particular battery to last, use the formula below. **Note this is theoretical only, do not expect this full figure to be achieved. This formula is for comparison of battery capacities only, and only applies to xBridge2 circuit and wixel firmware.**

Days Life = (Battery Capacity mAh/5)/24

So for a 1200mAh battery, Days Life = (1200/7)/24 = 7.14

Optionally you can include:

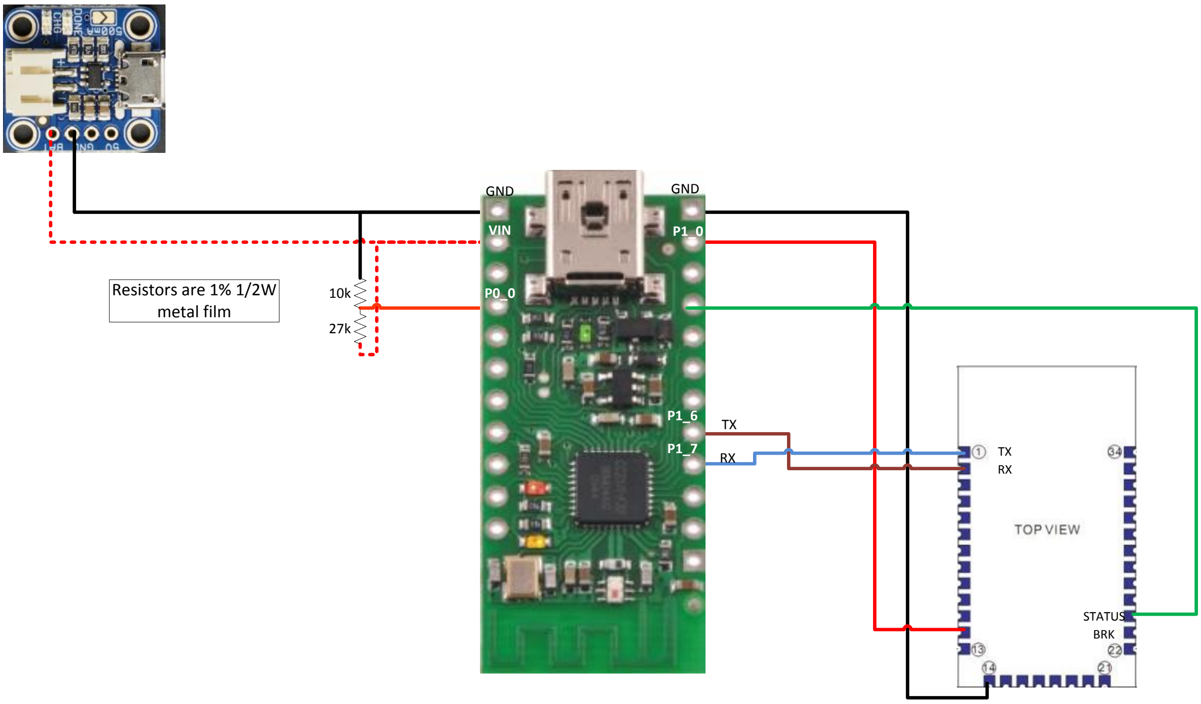
20mm Reverse/Forward Micro USB Qi Universal receiver Module (Adafruit Product IDs 2114, or 2116. This plugs into the charger board micro USB jack.

Switch in the Battery Positive lead to turn OFF the bridge.

JST Plug with wire, and Socket. Solder the socket to GND and VIN (Make sure the polarity is correct, Red battery wire goes to VIN, Black goes to GND), and solder the JST Plug and Wire to the output pins of the charger board. You can then easily disconnect the wixel from the battery and charger board when not using the bridge.

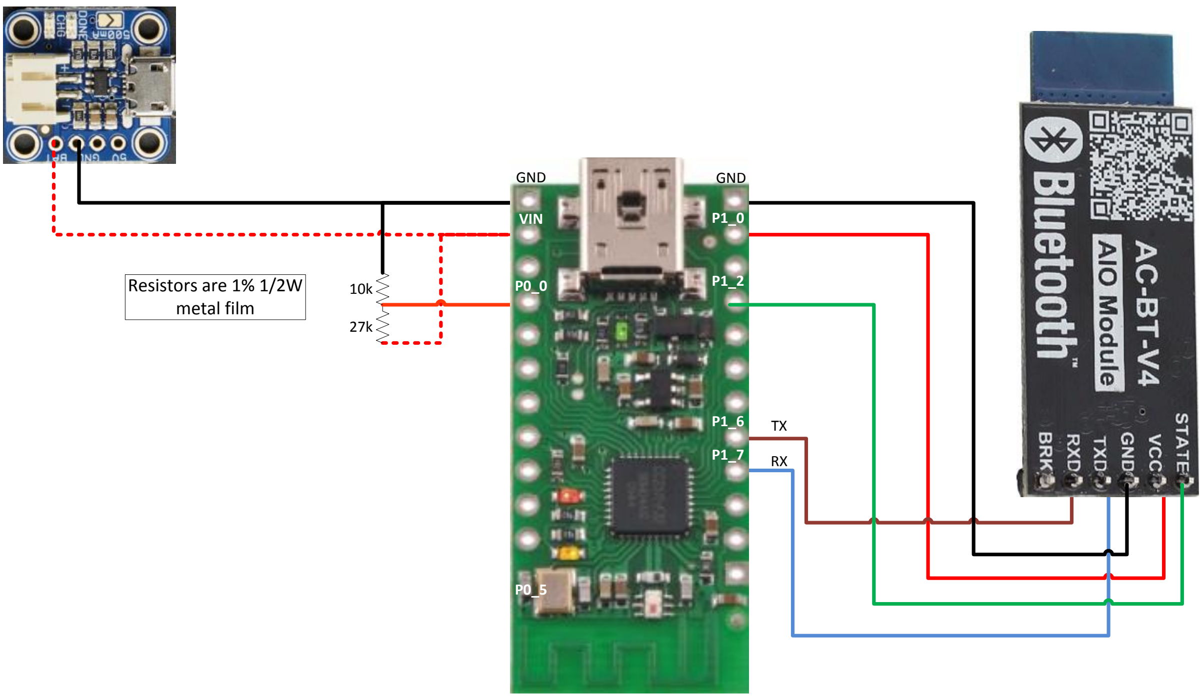
Circuit diagrams

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

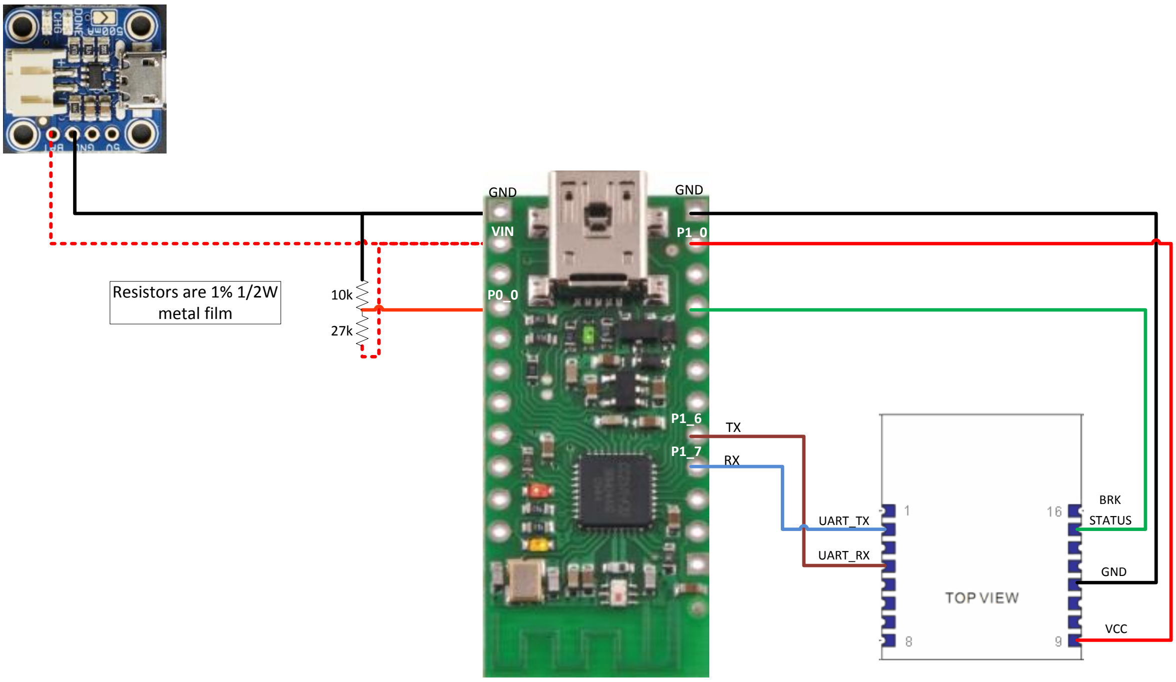


Note: be VERY careful not to overhead the solder pads on the HM-10 module. They are rather fragile. Also ensure that your wires do not place undue pressure on these.

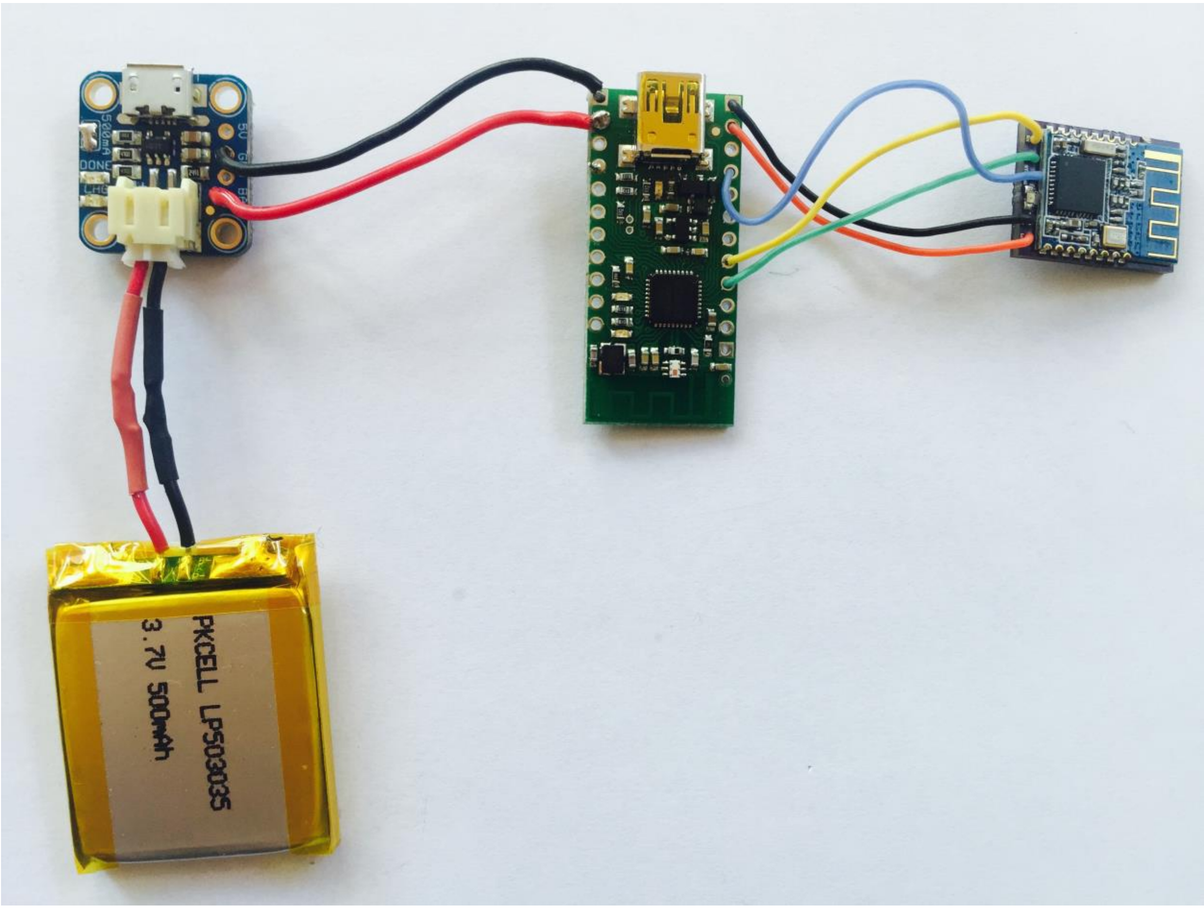
HM-10 on support board, with Wixel and Adafruit charger board.



Bare HM-11 board



Chris Bothello's HM-11 board (Thanks to Noah Terracalls)



Options for Charging

Note that it is **DANGEROUS** to allow a LiPo battery to drop below it's minimum voltage for any length of time. They do include cut out circuitry to prevent this, but they still continue to discharge over time. Always keep your LiPo charged.

The most obvious way to charge the bridge battery is to connect the Adafruit charger board to a micro USB cable and plug it in to a USB port. Once the Red LED goes out and the Green one comes on, the battery is charged.

However, this is not always convenient, especially if you want a somewhat waterproof bridge. You need to open the case, and connect things.

Something that has been used quite a bit is a Qi charger pad. Qi is a Near Field Wireless charging standard used in Mobile phones etc. Adafruit (and their resellers) have a number of these receiver pads, but you will want to get either of the models that have micro USB connectors on them. You plug these in to the Adafruit charger board, and place the charger receiver pad flat on the inside of one side of the bridge box you are using. You can then place it on a Qi transmitter charger pad (available in lots of places for $20-$50 USD), which commonly connects to a USB power supply of some sort.

Once placed on the Qi transmitter charger pad, the Red charger LED should light in the bridge. You may need to move it around a bit to make sure the centre of the receiver pad is over the centre of the transmitter pad. And that really is the only issue with the wireless charging, that it can be easily knocked and the charging mistakenly interrupted.

The choice is entirely yours.

Updating the HM-1x module firmware

NOTE WELL: Seed Studio (www.seedstudio.com) and some other suppliers are distributing HM-11 modules that are NOT genuine, and therefore this procedure will not work with those modules. A genuine HM-11 module will respond to an AT+NAME? command with "OK+NAMEHMSoft", and will also advertise itself on BLE as "HMSoft". If your module does NOT respond with this name, or advertise itself with this name, it is not genuine. However, it may still function just fine with xBridge2 wixel app, as the app will automatically detect the baud rate of the module and use it.

If you wish to be sure you have a genuine HM-11 module, I recommend ordering from https://www.fasttech.com/product/1740900-hm-11-bluetooth-v4-0-transceiver-ble-module. I have found this supplier to be reliable and responsive.

In order to ensure consistent operation of XBridge 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 XBridge has been tested against. Note: as long as your module responds to an AT command with OK, it will work fine with xBridge2.

Prerequisites

Firstly, you will need to either obtain the compiled usb_serial.wxl file (can be found at https://github.com/jstevensog/wixel-sdk/blob/master/apps/usb_serial/usb_serial.wxl), or compile it using the wixel-sdk located at https://github.com/jstevensog/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.

Note: Set the terminal parameters to be 9600 baud, 8 bits, 1 stop bit, no parity.

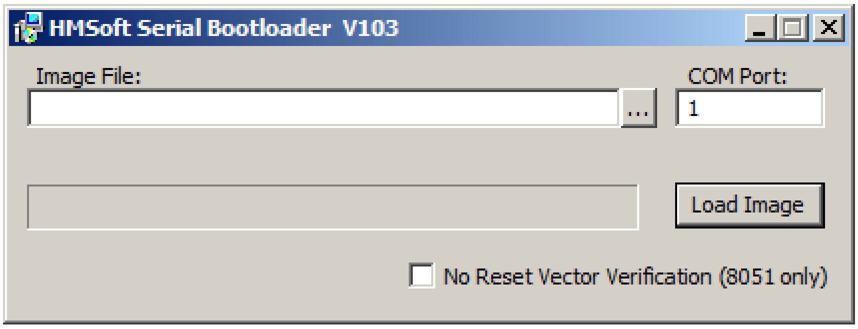
Some people have reported receiving HM-1x modules that use a different baud rate (115200). However, the Jinan Huamao documentation states that the default baud rate for the HM-10 and HM-11 modules is 9600 baud. As pointed out above, the module you receive may not have the latest firmware on it, and may not have been reset post Quality Assurance testing and being delivered to you. Once you have loaded the latest firmware on to your module, it SHOULD then be communicating at 9600 baud. If it is still not communicating at this baud rate, you will need to issue the command “AT+BAUD0” followed by “AT+RESET”, and confirming that the module is now communicating at 9600 baud, prior to loading the xBridge2 app onto the wixel.

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).

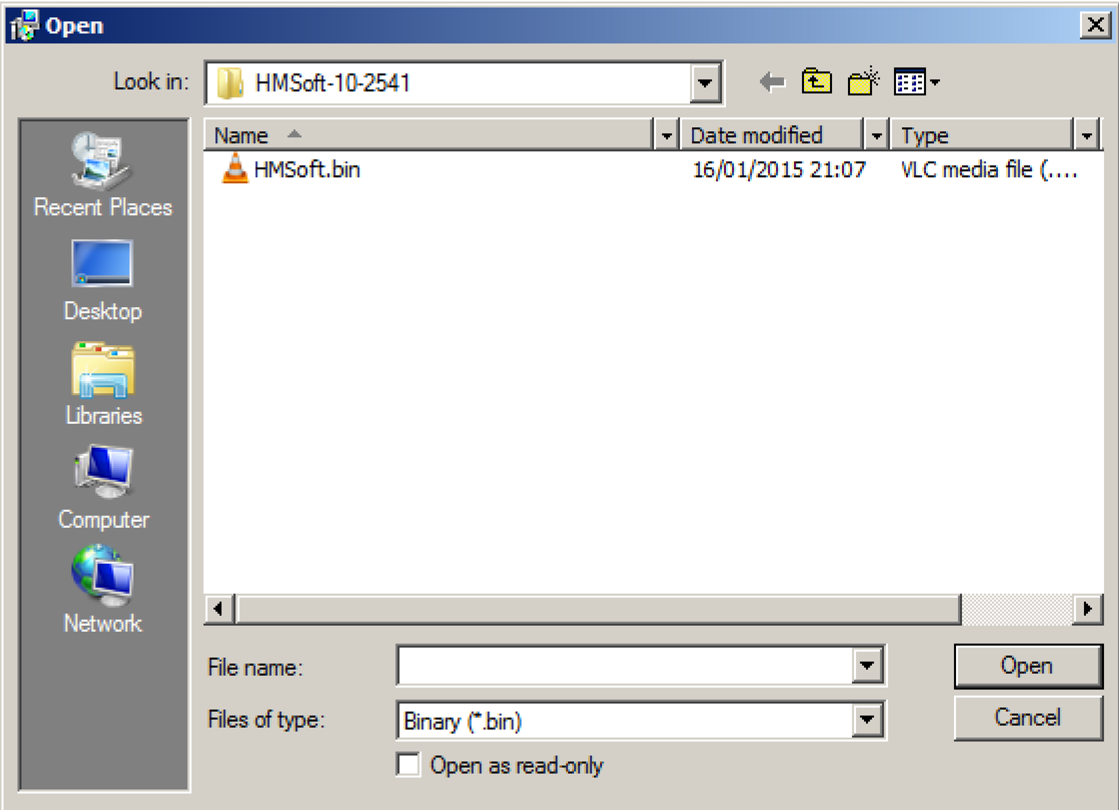
Updating steps.

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. **<u>SHUT DOWN YOUR TERMINAL PROGRAM NOW!!!!!</u>**

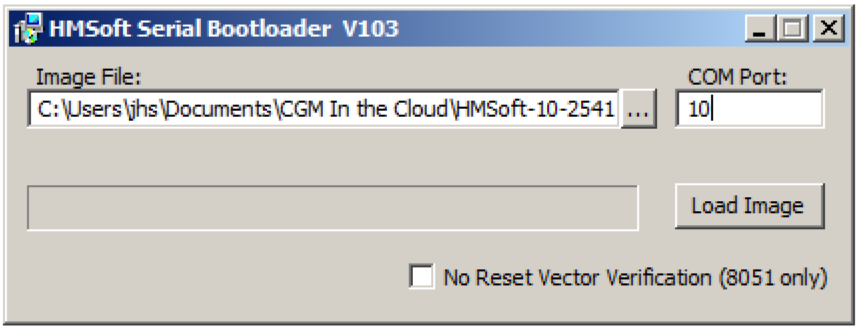
8. Open the HMSoft.exe file. It will open a window like this.



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



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



11. 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.

12. 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”
13. That is it. You can now load the xBridge2.wxl file using the Wixel Configuration Utility, and the bridge is ready for use.

Modifications to use xBridge app with the xDrip-wixel circuit.

The xBridge2 firmware now automatically detects if the circuit is wired as a “classic” xDrip wixel or an xBridge when it is powered on. It will automatically configure itself to use the original Chris Bothello voltage divider resistors if it detects a classic circuit, so it can pass the battery capacity on to xDrip.

If you do run xBridge2 on a classic circuit, you MUST select xBridge as your Data Collection Method in xDrip, otherwise it will not work.

The xDrip app can be configured to NOT display the bridge battery and pass it on to the Pebble watch face if you do not have the voltage divider resistors configured, or if you power the bridge from some other method (like directly wired to a phone as some have done).

Building the xBridge wixel program.

To build the xBridge wixel program, please use the Wixel SDK located at https://github.com/jstevensog/wixel-sdk. This SDK has both “lumos” sleep library and “slasi” modified radio_mac library. Although the code does NOT use the sleep library, it does use the radio_mac library.

Please see the instructions provided by Pololu to build the program if you are not familiar with how to do it.

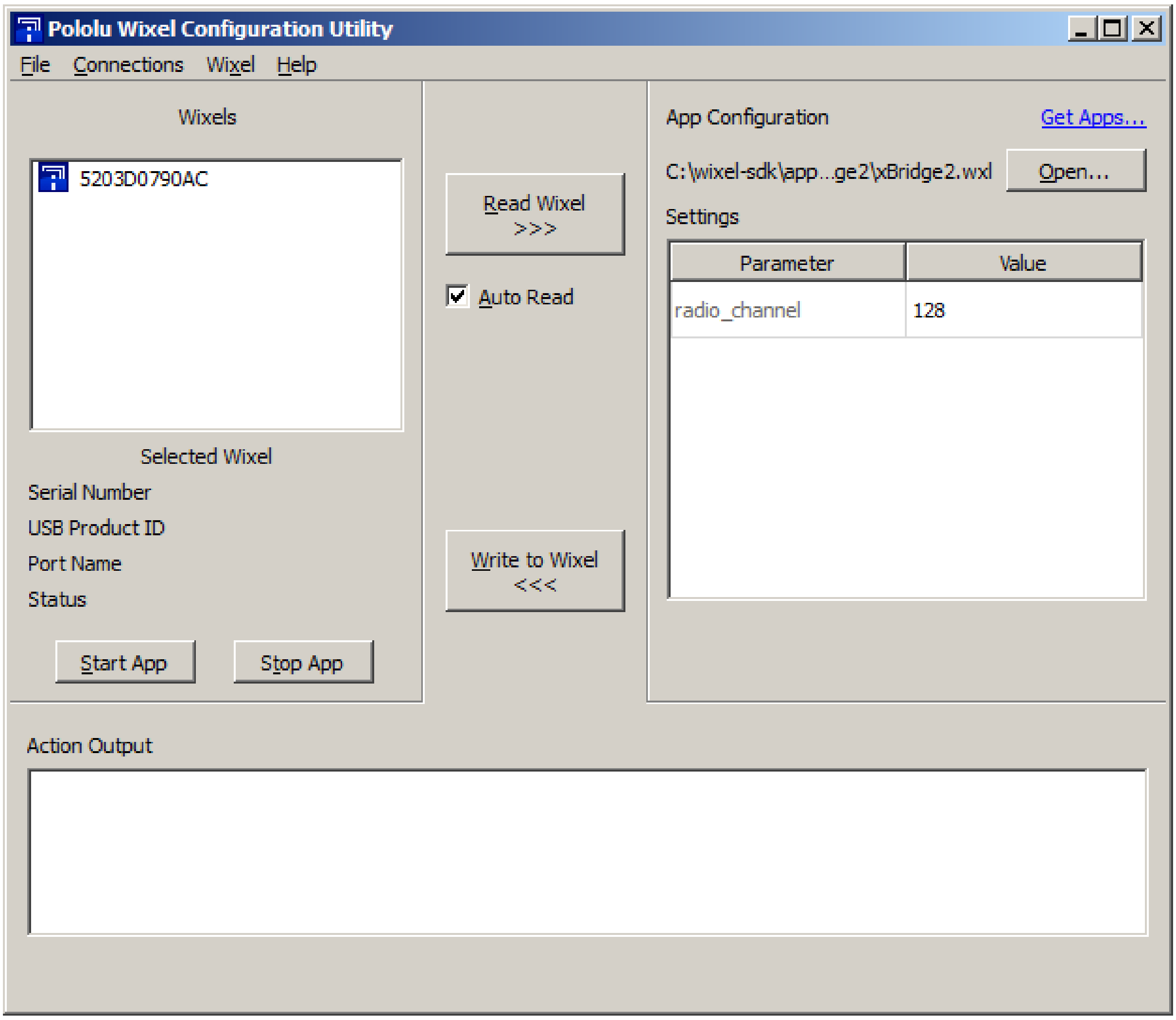
Or, you can obtain the xBridge2.wxl file from that repository in apps/xBridge2 (https://github.com/jstevensog/wixel-sdk/blob/master/apps/xBridge2/xBridge2.wxl), and simply load that onto the Wixel using the Wixel Configuration Utility.

Installing the xBridge app on the Wixel

Before you install the xBridge app on the Wixel **make sure ALL phones or other BLE “master” devices within the vicinity are TURNED OFF (or at least Bluetooth is).** Failure to do this will prevent the HM-1x module from being correctly configured IF another device has established a connection to it before the xBridge app has started and attempted to configure it.

The xBridge app currently waits for 30 seconds before starting so that, if you so desire, you can connect to it using a USB cable and serial terminal to watch the process of boot up.

Open the Wixel configuration utility, then open the xBridge2 app. Connect your bridge to your computer via USB. It should appear in the Wixel Configuration utility like this:



Click the Write to Wixel button, and the app will be loaded onto the wixel.

Wait for at least 30 seconds, until you see the Green LED on the wixel, and at least one blink of the Red LED. Or you can use a serial terminal to watch the boot process.

You should then be able to turn on your phone's Bluetooth and scan for your xBridge using xDrip. You are looking for a name like "xBridgeXXYY", where XX and YY are alpha numeric characters. Once you locate this device name, select it and a connection to your xBridge will be established.

If you see only a device called HMSoft, first try power cycling the wixel. If this does not work, there is an issue with the serial connection between the Wixel and the HM-1x. See the trouble shooting section.

The flashing (very fast and dim, every 10 seconds) Red LED is showing you that the wixel is trying to obtain a Transmitter ID. By default xDrip has this set to "00000". You need to configure xDrip correctly to work with xBridge.

In xDrip settings, select "xBridge Wixel" as the data source (the default is "xDrip Wixel"). Once you do this, a new setting will appear named "Dexcom Transmitter ID". Select this setting and enter your Dexcom Transmitter ID. Remember to enter any letters (A-Z) as UPPER case.

There will also be a setting appear that allows you to Display (or Not Display) the bridge battery in xDrip. The default is to display this.

You should now be able to start your sensor, allow xDrip to capture at least two packets, and do a double calibration to start using the system.

Serial Terminal Commands

xBridge2 supports three commands you can issue on a serial terminal connected to the wixel via USB.

S - Pressing the S key will provide you with a status display. This is useful in troubleshooting to let you know how xBridge2 thinks it is configured.

D – Pressing the D key will toggle (turn on or off the debug output) on the USB serial line. You will not need to do this unless you are experimenting with using an xBridge2 with a serial/USB program or device, that may be being confused by the debug output. Of course, if you are following the protocol description, this should not be necessary. However, if you are using a PERL SerialIO library, that expects either a line end or time delay parameter, you cannot effectively do a byte-by-byte capture of the stream and look for the identifying bytes for each packet type. In this case, this command may be of use.

B – Pressing the B key will toggle “sleeping” (powering down) of the HM-1x module on and off. It will display a message saying if it is on or off. This will assist those with phones or Android versions that do not reliably reconnect to the BLE device once it is powered on. If you experience lots of Bluetooth issues, or packet loss, you can use this command to prevent the HM-1x module from being powered down, and this should then resolve the issue. **NOTE: This will impact your bridge battery life.**

Problem Solving

Obviously, the first thing to check is the wiring between the component boards. Make sure all of this is correct before you go any further.

Problems with updating the HM-1x firmware

These modules are fairly robust and can put up with a lot. If you for some reason happen to power down the module during firmware update, after you had entered the AT+SBLUP command, do not be alarmed. It won’t have completed the update process. Simply repower he module, and send the firmware file again. It will complete the process.

The most common problem with this, is establishing the connection with the module using usb_serial app on the wixel. It has been reported (though I have not seen it) that some modules are being delivered fixed to a different baud rate from that specified by the manufacturer as the default. You can determine what the baud rate is by setting your terminal program to different baud rates and issuing an “AT” on each until you get “OK” in response. Once you flash the new firmware onto the module, it should use the default 9600 baud rate, which is also used by xBridge.

BLE Name stays at HMSoft, instead of xBridgeXXYY.

If, when you scan for your xBridge in xDrip (or any BLE scanner) you see the name HMSoft, this generally means one of two things:

1. You had a BLE connection established when you loaded the App onto the wixel. Disconnect your BLE module (even better, turn off all Bluetooth on any nearby phones), and re-flash the wixel code.
2. You did not verify that the HM-1x module is communicating at 9600 baud. Either you did not update the firmware, or you simply didn't check that this was the case. Re-load the usb_serial app onto the wixel, and determine the current baud rate by cycling from 115200 down. Once you determine the baud rate, reflash the new firmware onto the HM-1x module, or at the very least send the command "AT+BAUD0", followed by "AT+RESET".

How do I know if it is working?

Well, before you connect it to XDrip or any other app, the wixel will wake. If you are not connected to it via a USB cable, with a serial terminal program, and for at least 30 seconds initially, you will not notice anything

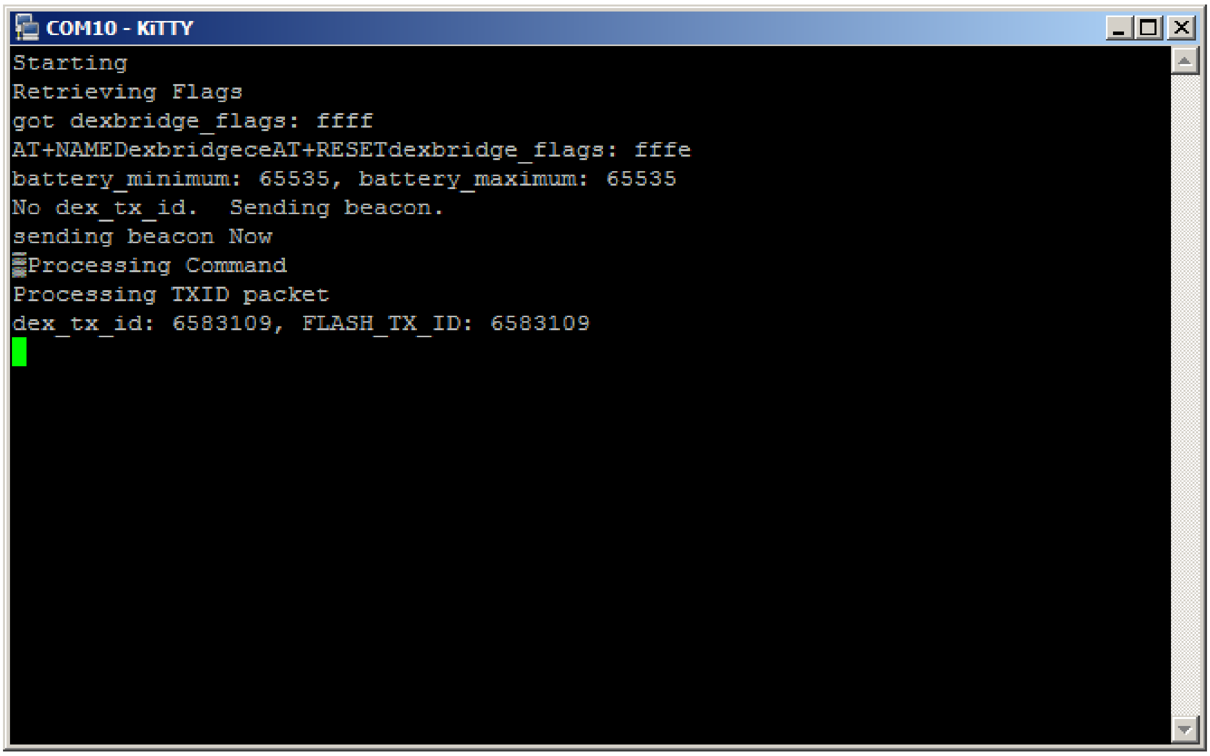
After 30 seconds, you should see the GREEN LED light up if you are connected to USB and have a serial terminal program open to the port.

Because the code only turns on the Yellow LED when a BLE connection is made under normal operation, it is sometimes difficult to know what it is up to. Here are the steps to go through.

1. When you first load the xBridge2.wxl file onto the wixel and you are connected via USB, the GREEN LED will light after 30 seconds.
2. Use a BLE utility, or XDrip, to scan for a BT device. You should find a device with a name like XBridgeXX. If you see that, the wixel has woken and correctly configured the HM-10 module, and should be ready to go.
3. If instead you see only a device called HMSoft, the wixel has not been able to correctly communicate with the HM-10 module. Check your wiring. And also check that you have loaded the xBridge2.wxl file on the wixel.
4. If you establish a BLE connection to the bridge, the green LED on the HM-1x module will go on, and the YELLOW LED on the Wixel will go on shortly after. This shows that the wixel is aware of the connection.
5. If you have not yet entered your Transmitter ID in xDrip, you will notice a very brief and dim blink of the RED LED every 10 seconds. This is showing you that the wixel is telling xDrip it doesn't have a Transmitter ID to work with yet.
6. If you would like to see more of what is going on, connect the wixel with a mini USB cable to a PC or laptop. After a time (up to 5 minutes if it was sleeping when you connected), the Green LED will light. Connect a terminal program to the device. The wixel WILL SLEEP while connected to the terminal program, but only to a mode that allows the USB to remain open, if it has received a Dexcom Transmitter Packet, sent it, and received an ACK packet from the app.
7. If you would like to see the communications it is sending, you can open a serial terminal and connect to the wixel. Use 9600, 8, 1, no parity as the settings of the terminal. Note, that XBridge can send output that is NOT in text, so on occasion you

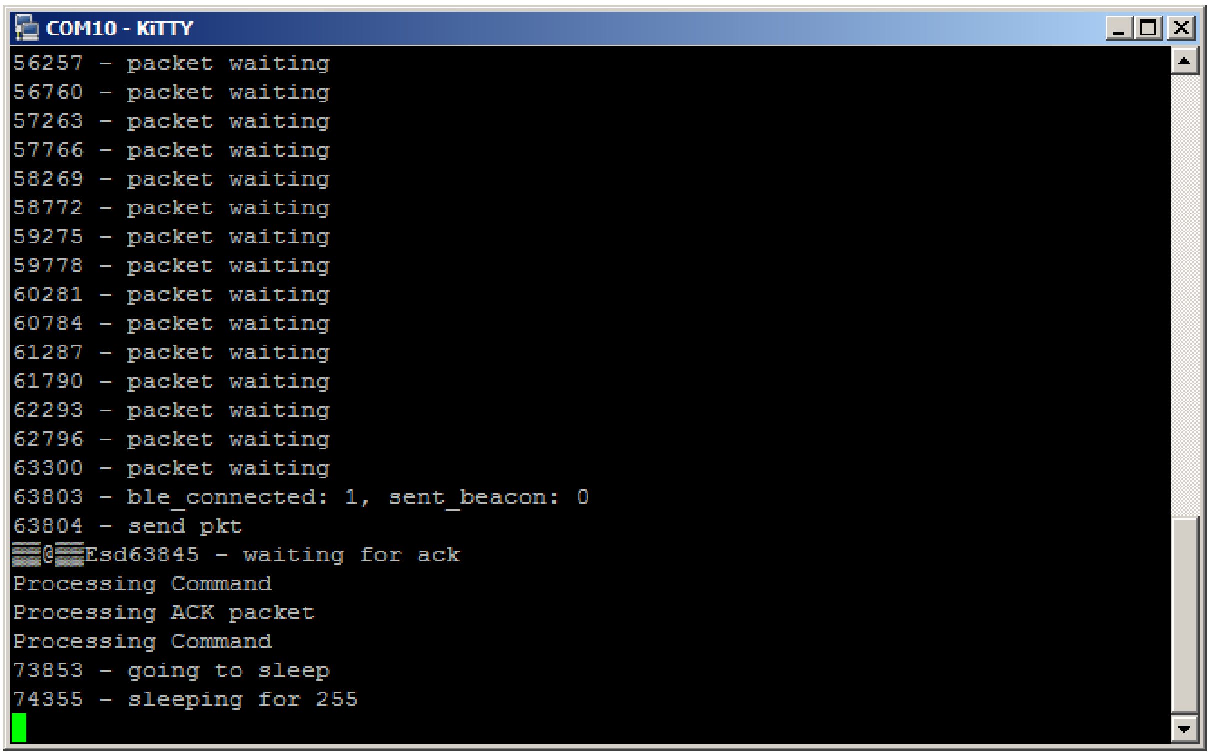
will see silly characters. But you will also see a lot of diagnostic messages telling you what is going on. You can also send an “s” or an “S” from the terminal and you will get a Status message, detailing the contents of the Flash storage variables, and the variables that hold these values in the program. See below for details of the types of messages.

Example Output on USB Serial Port



The screen shot above shows a boot up after the firmware is loaded. Note, that this wixel was already joined to an xDrip app. As you can see, when it first started up, all the xBridge_flags were set to 1 (ffff). After the HM-1x was set and configured, the flags are now set to fffe. Also, the battery_minimum and battery_maximum settings are not yet set (showing 65535).

After it has configured the HM-1x it sends a beacon packet until it gets a TXID packet in response. Once it has received the TXID, it saves all the settings to flash, and begins scanning the radio channels for a packet.



In this next screenshot, you can see it has got a packet from the Dexcom transmitter. The wixel waits until it has a BLE connection to the phone, then it sends a beacon, then the DATA packet, and waits for an ACK packet.

Once it has this, it saves the settings to flash (mo message above) and then goes to sleep.

It is relatively common for the current xDrip app to take a while to establish a connection. Hence you can see all the “packet waiting” messages here. The wixel will stay awake for two minutes after a packet capture, waiting for a BLE connection. On some occasions, this will not happen, the wixel will once again go to sleep, and this will appear as a missed packet in xDrip.

At any time when the wixel is awake, you can press the “S” key on the keyboard to get a status display. See below.

COM10 - KiTTY

```
60784 - packet waiting
61287 - packet waiting
61790 - packet waiting
62293 - packet waiting
62796 - packet waiting
63300 - packet waiting
63803 - ble_connected: 1, sent_beacon: 0
63804 - send pkt
▒▒@▒▒Esd63845 - waiting for ack
Processing Command
Processing ACK packet
Processing Command
73853 - going to sleep
74355 - sleeping for 255
75376 - awake!
75377 - ble on
Processing Command
Processing Status Command
dex_tx_id: 6583109, FLASH_TX_ID: 6583109
dexbridge_flags: fffe, DEXBRIDGE_FLAGS: fffe
battery_maximum: 1814, FLASH_BATTERY_MAX: 1814
battery_minimum: 1416, FLASH_BATTERY_MIN: 1416
battery_capacity: 70
```

Here you can see that the wixel has woken up, and turned on the HM-1x (ble on). It has then received a command (me pressing the S key), and is processing it. You can see the battery minimum and maximum are at the default values, and battery capacity has been calculated at 70%.

Functional Testing

The following tests are carried out on every iteration of the XBridge2 code to ensure it is operating as expected. If you are modifying this code, please perform these tests to ensure you code complies with these minimum functions.

This section on testing assumes you are familiar with Android Studio and the use of the Android Debug logs (ADB).

Testing Pre-requisites

In order to conduct these tests, you will require the following:

- Android Phone with USB debugging enabled.
- Android Studio
- A copy of the XDrip app repository, that has XBridge support.
- A XBridge unit as per the circuit diagrams in this document.

Test 1 – Broadcast of Beacon Packet and Setting of TXID.

1. Ensure that XDrip is running and you are viewing the ADB logs.
2. Reload the xBridge2.wxl file onto the wixel module. Note, do NOT just start and stop the wixel program, flash the wixel with the program to clear the Flash storage page used by XBridge2.
3. Observe the ADB log. You should see the following sequence of events;

a. BLE GATT connection series of messages.
b. Messages indicating Receipt of a Beacon Packet.
c. Message saying the Beacon Packet is Wrong.
d. Message “sending message”.
e. Receipt of another Beacon Packet, which is the acknowledgement to the previous one.

4. The wixel will now listen and wait for a Data Packet. There is no ADB log entry showing this listening phase, but there will be log entries when a Data Packet is found.

Test 2 – Reception of Dexcom Data, and initiating sleep.

1. Observe the ADB logs until a message appears showing the reception of a Data Packet. Note, if you are using Android Studio, you can filter on the DexCollectionService to more easily find these.
2. Once this message is received, it will be followed by a “sending message” entry. This is the Data Packet ACK message, and will put the wixel to sleep.
3. A series of messages showing a disconnection of the BLE GATT, and reconnection to the BLE device. This occurs even though the HM-1x module is asleep.

Test 3 – Waking of the wixel and reception of Dexcom Data.

1. Observe the ADB logs until a series of messages appears showing the BLE GATT connection taking place. Note, whilst you would assume this happens at very regular intervals, this is not always the case. It does not mean that the wixel has not woken as expected, nor that it has missed a packet of data. The BLE implementation does not automatically detect when a paired device becomes available again.
2. At some time after the BLE connection is established, the ADB logs will show reception of a Data Packet as per Test 2 above. Depending on timing, it may also show reception of a Beacon Packet, which will only be dealt with if the TXID is NOT what the app is expecting.
3. The wixel will return to sleep as per Test 2 above.

Usage Notes

There are a few things to keep in mind when using xBridge code, and many of these will apply to using the xDrip-wixel code as well.

1. Charge the bridge when battery capacity falls to 40% or lower. Because we are using a simple battery voltage measure to determine capacity, and LiPo batteries deplete rapidly as they approach minimum voltage lever, it is best to charge it around this point rather than wait. While you may get a few more hours out of the battery at this level, it certainly won’t last a too long.
2. Keep the bridge close to you. While BLE has a bit better range and reliability than the proprietary 2.4GHz protocol used by the Dexcom transmitter, you will still need to keep the bridge within the 20’ / 6m radius of it.
3. Currently the bridge will occasionally miss packets. I am still researching why this is the case. If you use a Qi pad to charge your bridge, this will occur more frequently, likely due to the electro-magnetic field interfering with the Wixel receiver.