On the Design and Usage of globalCALCnet

Christopher Mitchell

Abstract

CALCnet$^{2.2}$ introduced extensible, robust, and simple multi-calculator networking to the world of TI graphing calculators, but is limited to local area networks. It allows people in the same room or building to play calculator games together, share data and files, or chat. However, on its own it does not permit connection of calculators across a city or across the world, a role filled by globalCALCnet (gCn). Using inexpensive commodity hardware attached to any computer running Windows, Mac OS X, or Linux, gCn connects CALCnet networks despite their geographical separation. In addition, globalCALCnet allows calculators to access and use internet resources such as chat rooms, program downloads, and more. This document discusses the procedure for setting up a globalCALCnet client and provides a brief overview of constructing the different types of gCn bridge. It also details the protocols used to connect gCn clients and hubs across the Internet.

✦

1 INTRODUCTION

CALCnet$^{2.2}$ offers a robust, asynchronous end-to-end transport protocol capable of dynamic, stateless network topology reconfiguration, error detection and recovery, and full network features such as arbitrary simultaneous disjoint communication. It supports one-to-one directed transmissions and one-to-many broadcasts to support as wide a set of applications as possible, and is targeted at utilities, chat programs, file-transfer programs, conference applications, and especially multiplayer and massively-multiplayer gaming on TI graphing calculators. As of this publication, the CALCnet$^{2.2}$ protocol has been written into Doors CS 7.1 (and up), a shell and GUI developed for calculators by the author. It can be used on TI-83+, TI-83+ Silver Edition, TI-84+, and TI-84+ Silver Edition graphing calculators, as well as via the TI-84+ emulator on the TI-nspire calculator. This document presents first the motivation and existing alternatives, then the frame-level, byte-level, and bit-level protocol as well as the steps and caveats in implementing CALCnet in new programs, and finishes with a survey of performance, branding, and future work. The latest information on CALCnet and Doors CS can be found at http://www.cemetech.net; Doors CS-specific downloads and information, including its SDK, are at http://dcs.cemetech.net. In order to use globalCALCnet , calculators must be able to use CALCnet$^{2.2}$, and thus must have Doors CS installed.

1.1 A Brief Introduction to gCn

globalCALCnet is a method of transparently connecting multiple CALCnet$^{2.2}$ networks together over the internet. Since a CALCnet network can be from one to one trillion (technically, 16^5) calculators, gCn can work as simply as connecting two individual calculators as if they were right next to each other, or at the opposite extreme link together many networks, each consisting of many calculators, into a single virtual hub. Such a system is necessarily quite complex under the hood, but the components exposed to users are straightforward and easy-to-use. Each globalCALCnet network consists of a handful of different components:

- **CALCnet$^{2.2}$ Network:** One or more calculators linked together with a CALCnet network, requiring only the calculators and a handful of unit-to-unit cables. Almost no electrical engineering knowledge or skill is required to construct a CALCnet network.
- **AVR-based microcontroller board with FTDI or USB:** Early prototypes used the widely-popular Arduino Duemilanove board, offering a 16MHz AVR with 32KB of EEPROM and 2KB of SRAM. This board pretends to be a calculator participating in the network, but passes incoming frames up to a connected PC via serial/USB, and injects frames into the network from the PC. Later prototypes used a raw Atmega AVR bit-banging USB using teh V-USB library, a far cheaper alternative. Future work involves allowing TI-84+/SE calculators to access gCn vai their built-in miniUSB ports.
- **C++ gCnClient Application:** The gCnClient application connects both to the local Arduino and to a remote gCn metahub (virtual hub aggregator). The gcnclient is responsible for moving 1-to-1 or directed frames and broadcast frames from the Arduino to the remote server and vice versa.
- **Python gCnHub server daemon** The gcnHub, a metahub of virtual hubs, acts as one or more virtual hubs linking together groups of remote gcnclient applications. There is currently a single gCnHub server, but if gCn becomes more popular, it would be easy to distribute this program so that users could set up their own servers.

Frames (the CALCnet equivalent of packets) travel from one CALCnet2.2 network, through the connected Arduino, to the gcnclient and through the internet to a gcnhub, which routes the packet as necessary to another gcnclient, which hands off the frame to its Arduino and therefore its connected CALCnet$^{2.2}$ network. A diagrammatic view of a globalCALCnet system can be seen in Figure 1. Testing has confirmed functionality over 12 miles and a few hundred miles via the Internet.

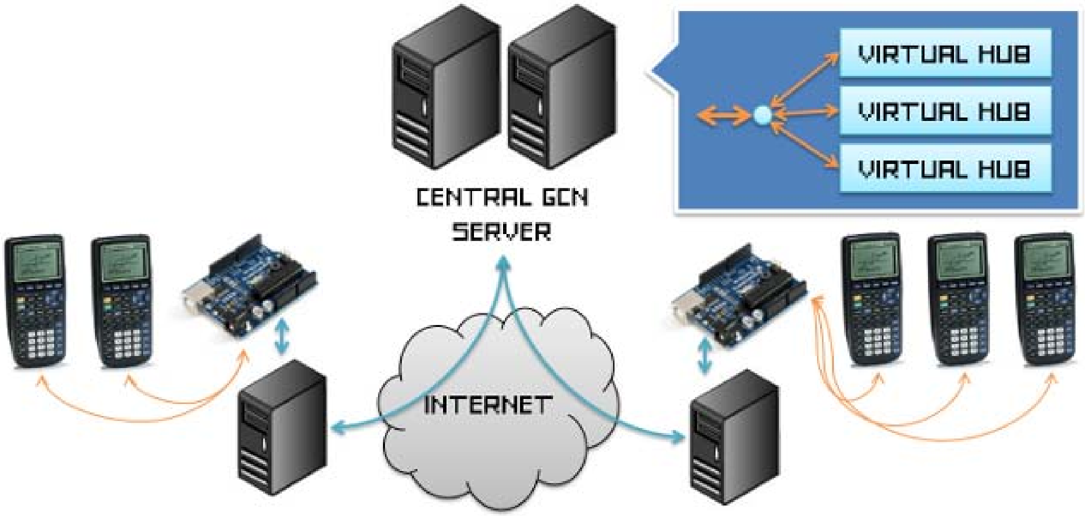


Fig. 1. Overview of globalCALCnet architecture. Data travels from calculators, to an Arduino board for translation, through the internet via gCnClient applications connected to a gCn metahub, to other Arduinos and hence calculators.

1.2 The gCn Transport Protocol

globalCALCnet is carried over IP networks including local-area networks and the Internet via the TCP protocol. Since TCP is a stream-oriented protocol, gCn contains its own chunking protocol to delineate the boundaries of messages traveling between gCn clients and metahubs. There are currently four types of messages used within gCn , as enumerated below. Each message is prefixed with a three-byte header, containing a two-byte little-endian size word and a one-byte type character. The size word is in the form LSB, MSB, such that $05,$00 indicates a message with a five byte payload (not including the three-byte header). Therefore, a message with a size byte $05,$00 consists of eight bytes in total: three of the header, and five of the payload. The one-byte type is a character such as 'c', 'f', 'j', or 'b'.

- **Join (j):** A gCn client must send a join ('j') message before any other messages, including calculator (c) messages. After the three byte header, the payload consists of a pair of length-prefixed strings. The first string is the hub name to which to join (for example, 'IRCHub' or 'netpong'), the -n argument to the gCn client application; the second string is the local client name (for example, 'KermClient'), the -l argument to the client appllication. Each string is prefixed by a one-byte length, and followed by the ASCII string. Each string is not zero-terminated.
- **Calculator (c):** A calculator ('c') message announces that a new calculator is present at a particular gCn client. This message always contains a fixed ten-byte payload, containing the ASCII representation of the five-byte, ten-nibble address of the new calculator. For example, a calculator with ID `$04, $00, $79, $51, $C3` would be announced to a gCn metahub with the message `$0A,$00,'c','04007951C3'`, a thirteen-byte total consisting of the three-byte header and the ten-byte payload.
- **Broadcast (b):** A broadcast ('b') message is sent from one calculator to every other calculator on the network, but carries no delivery guarantees in the context of CALCnet$^{2.2}$. The payload of each broadcast message is written directly to the Arduino on each of the receivers, and consists of at least sixteen bytes. The first two bytes of each payload are always 255, 137 in decimal, a standard header to ensure the Arduino knows a new frame is beginning. Next are five bytes of the receiver address (for broadcast frames, always `$00,$00,$00,$00,$00`), five bytes of the sender address, the two-byte little-endian size word of the broadcast frame's internal data payload, at least one byte of data, and the concluding byte 42.
- **Directed frame (f):** A directed frame ('f') message is sent from one calculator to one other calculator. If both calculators are on the same local CALCnet$^{2.2}$ network, the Arduino sees the frame, but does not forward it to the CALCnet client for retransmission to the CALCnet metahub. If the recipient is not on the local network, the Arduino will pretend to be the remote calculator, acknowledge the frame and check its checksum, then deliver the frame to the gCn client, which will give it to the gCn metahub for routing to its destination. A directed frame has the same payload format as a broadcast frame, with the exception that the recipient address is not (and can never be) `$00,$00,$00,$00,$00`.

2 BUILDING AN ARDUINO-BASED GCN BRIDGE

A bridge functions to connect a CALCnet 2.2 network of calculators to a computer, allowing the calculators on the network to communicate with other remote calculators around the world and to interact with internet services. The bridge must be fast enough to speak the CALCnet 2.2 protocol and eavesdrop on any frames sent over the Cn2.2 network, relatively inexpensive, and include a popular interface for connecting to computers. A decade ago this might have been RS232/serial or parallel, but sadly, these two legacy connections are found on a tiny minority of current computers. USB is therefore the obvious solution, but requires a microprocessor to speak Cn2.2, since USB and buffering might preclude proper CALCnet timing. Therefore, current gCn bridges use a popular microprocessor board known as the Arduino, available for about $25-$30. An overview photograph of a gCn bridge connected to a two-calculator CALCnet hub can be seen in Figure 2, while a closeup of the Arduino and status LED circuit (formatted as a riser board plugging into the Arduino) is included as Figure 3.

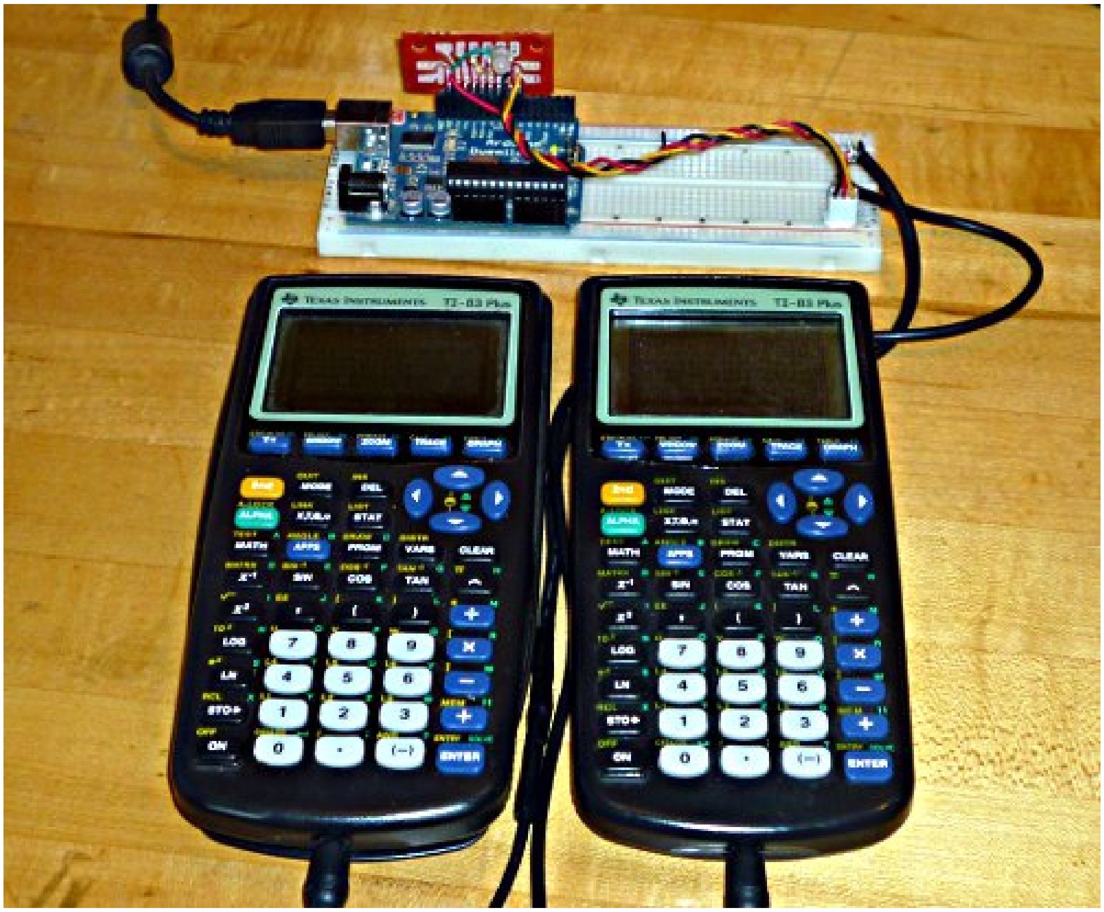


Fig. 2. View of a sample gCn bridge, including Arduino, status LED and resistors, and connection to CALCnet$^{2.2}$ hub.

The full parts list for a gCn bridge need only include an Arduino and its USB cable (over which both data and power are transferred) and plugs or sockets necessary to connect one or more calculators. If you also wish to be able to see the state of the Arduino, a single RGB LED with three 330ohm resistors is recommended; for additional LEDs showing the state of the two data lines in CALCnet, two single-color LEDs, two PNP transistors, and four more resistors are needed. Although a complete gCn bridge is marginally more complex to build than a simple CALCnet2.2 hub, the simple version requires no more soldering or electronics skill. The maximum cost for a bridge is currently around $35, although plans are in motion to try programming an AVR to perform native USB HID communication, which would bring the cost down to about $10 per simple bridge.

The schematic for a suggested globalCALCnet bridge is provided in Figure 4. The Arduino and the three connections to Clock, Data, and Ground are the only vital pieces. Please note that Clock is the tip or red wire, and Data is the ring or white wire. If you wish to have a colorful indication of the state of the gCn bridge, especially to tell if the network is frozen or experiencing some type of failure, the RGB status LED circuit is highly recommended. You may also wish to be able to see the states of the individual clock and data lines, as provided by the line status LED circuit.

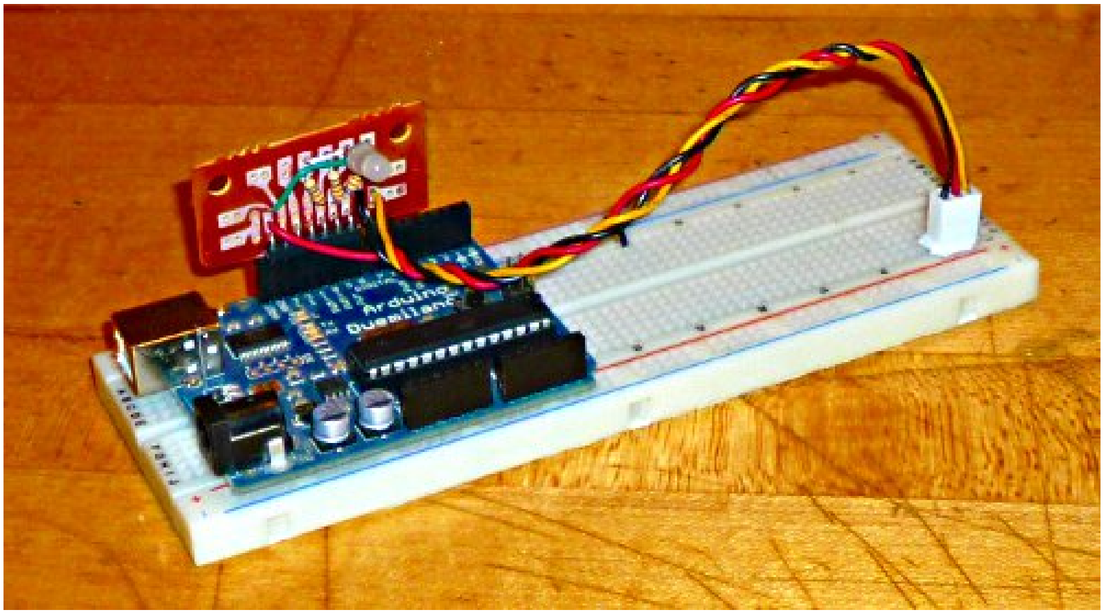


Fig. 3. View of a sample gCn bridge, focusing on Arduino and status LED and resistors.

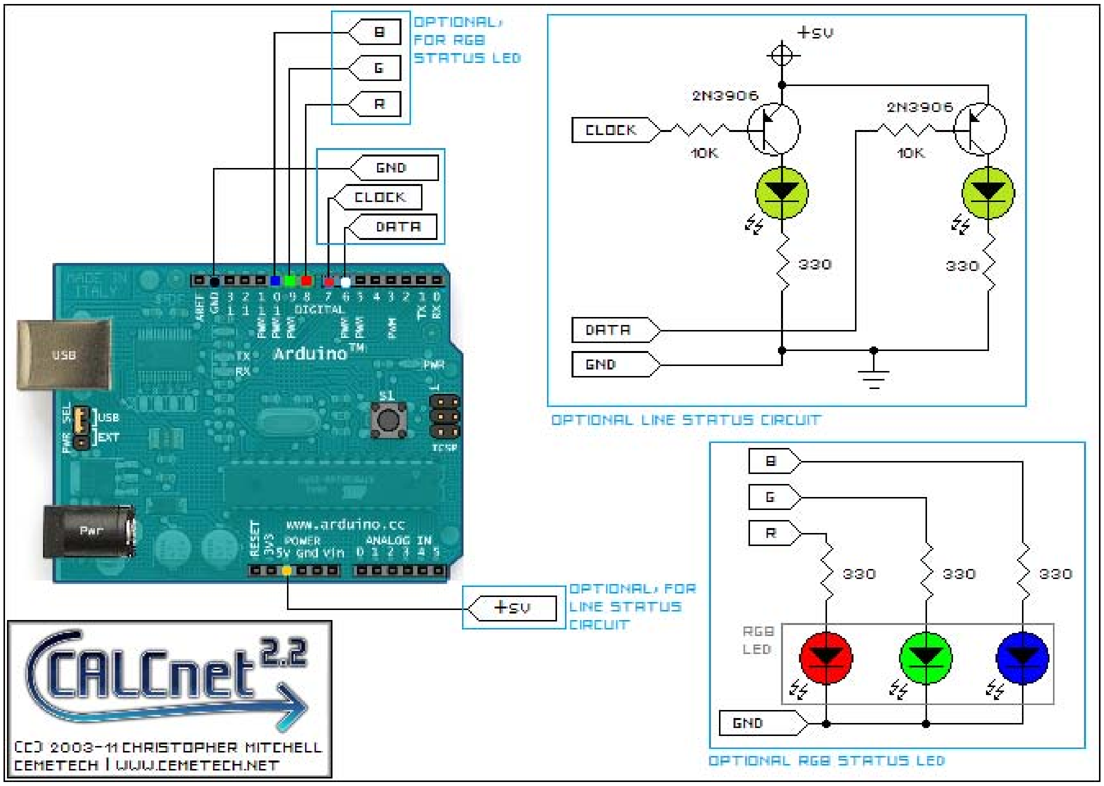


Fig. 4. Schematic of a sample Arduino gCn bridge, including required Arduino connections and optional status and line status LED sections.

Your Arduino may have an Atmega168 or an Atmega328 microcontroller: the former has 1KB of RAM and 16KB of programming space, while the latter has 2KB of ARM and 32KB of programming space. Firmware builds for the gCn bridge are available for both the Atmega168 (gcnbridge2.168.hex) and for the Atmega328 (gcnbridge2.328.hex); the former can handle a maximum of 40 calculators on the local network, while the latter can handle a maximum of 100 calculators. These limits may be a factor in choosing your Arduino board; the Diecemila always has an Atmega168, Duemilanoves come in both Atmega168 and Atmega328 flavors, Nanos always use the Atmega168, and the Uno always uses an Atmega328.

2.1 Constructing a gCn Bridge

The simplest form of a gCn bridge involves just an Arduino and either half of a calculator link cable, or a 2.5mm stereo female jack. The digital pins 6 and 7 of the Arduino, as well as one of the ground pins (ideally the one nearest the digital pins) should be connected. The CALCnet$^{2.2}$ clock line, which is the tip of the plug on a 2.5mm plug (or the red wire of a unit-to-unit cable) connects to digital pin 7 of the Arduino. The CALCnet data line, which is the ring or middle contact of the plug on a 2.5mm plug (or the white wire of a unit-to-unit cable) connects to digital pin 6 of the Arduino. Finally, the base or ground, the copper wire in the cable, connects to the ground of the Arduino. Then the USB cable can be connected to a computer, and the instructions in Sections 4 and 5 can be followed to complete the setup.

If you wish to add an RGB status LED, which both helps with debugging and troubleshooting any problems you may have, and also looks cool in general, you will need a common-cathode RGB LED and three 330Ω, 1/4-Watt resistors. Connect one resistor each to digital pins 8, 9, and 10 of the Arduino, then connect the resistor from pin 8 to the red leg of the LED, 9 to the green leg, and 10 to the blue leg. Finally, connect the common cathode (ground) leg of the LED to one of the grounds on the Arduino. To make this easier for yourself, I recommend either soldering together a circuit on protoboard or perfboard, or using a solderless breadboard to construct the circuit.

For a more permanent bridge/hub, I recommend putting the Arduino in an enclosure along with any additional circuitry that you need, plug connections to the link cables or 2.5mm female jacks that form your CALCnet$^{2.2}$ hub. Future editions of this technical report may add diagrams and photographs of such designs if users build them.

3 BUILDING A "$10 BRIDGE" OR USBHID GCN BRIDGE

The Arduino is a relatively expensive board for the components it includes, so many users requested a cheaper alternative that would not monopolize a reusable board like the Arduino. A good solution is to use a bare Atmega328 AVR microcontroller from Atmel, for which drivers to bit-bang USB have been written (http://www.obdev.at/products/vusb/index.html). Including the microcontroller, the total parts cost is around USD$10, at the labor expense of more complex construction than using an Arduino. In order to build a USBHID bridge, some basic soldering skills are necessary, unless you build and keep the circuit on a solderless breadboard. Besides the following parts, you also need either an AVR programmer, or an Arduino board you can borrow for the one-time task of burning the USBHID firmware to the AVR.

- (1) Atmega 328p microcontroller, preferrably pre-loaded with the Arduino bootloader
- (1) 16MHz crystal
- (2) 22pF capacitors for crystal loading
- (1) 4.7μF capacitor for power noise damping
- (2) silicon diodes with 0.7 volt drops for power regulation
- (2) 68Ω resistors for USB current limiting

- (1) USB female Type B connector
- (1) 1.5KΩ resistor to identify the gCn Bridge as a low-speed USB device
- (3) 330Ω resistors for LED current limiting
- (1) common-cathode RGB LED
- (1) 10KΩ reset line pull-up resistor
- (1) momentary reset button
- (1) 28-pin DIP IC socket for the AVR
- (1) perfboard or solderless breadboard of your choice

The schematic for building the circuit can be seen in Figure 5. Because of the many possible types of board that can be used for this project, no board layouts are given. If you happen to create a custom PCB for the project, please contact the author so that it may be included in this documentation. A completed example of the USBHID bridge is provided in Figure 6. Notice the four-pin connector on the USBHID bridge (the tan-colored perfboard) connected to a solderless breadboard that is being used as a temporary CALCnet$^{2.2}$ hub. For more information about CALCnet , be sure to examine the documentation at http://www.cemetech.net/projects/item.php?id=33.

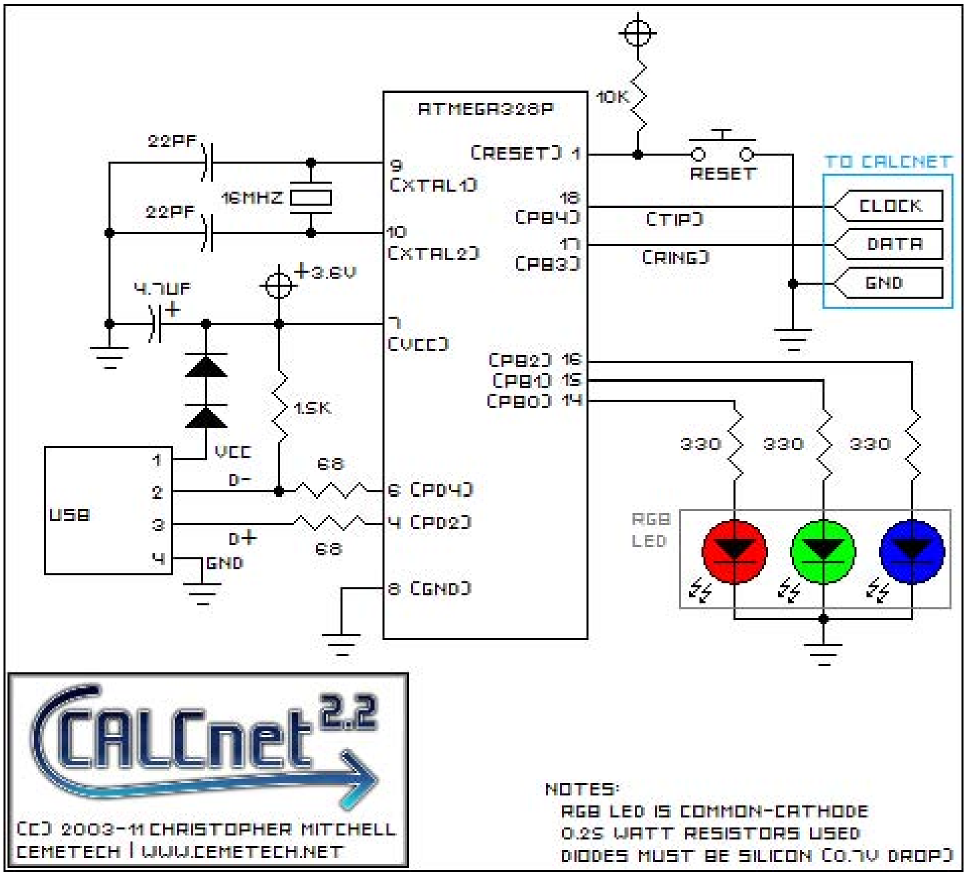


Fig. 5. Schematic of a USBHID gCn bridge.

3.1 Detailed USBHID gCn Bridge Photos

Please see Figure 3.1 for four photos of the USBHID Bridge (the ”$10 Bridge”) from the front (Figure 7(a)), the back (Figure 7(b)), the top (Figure 7(c)), and the bottom (Figure 7(d)).

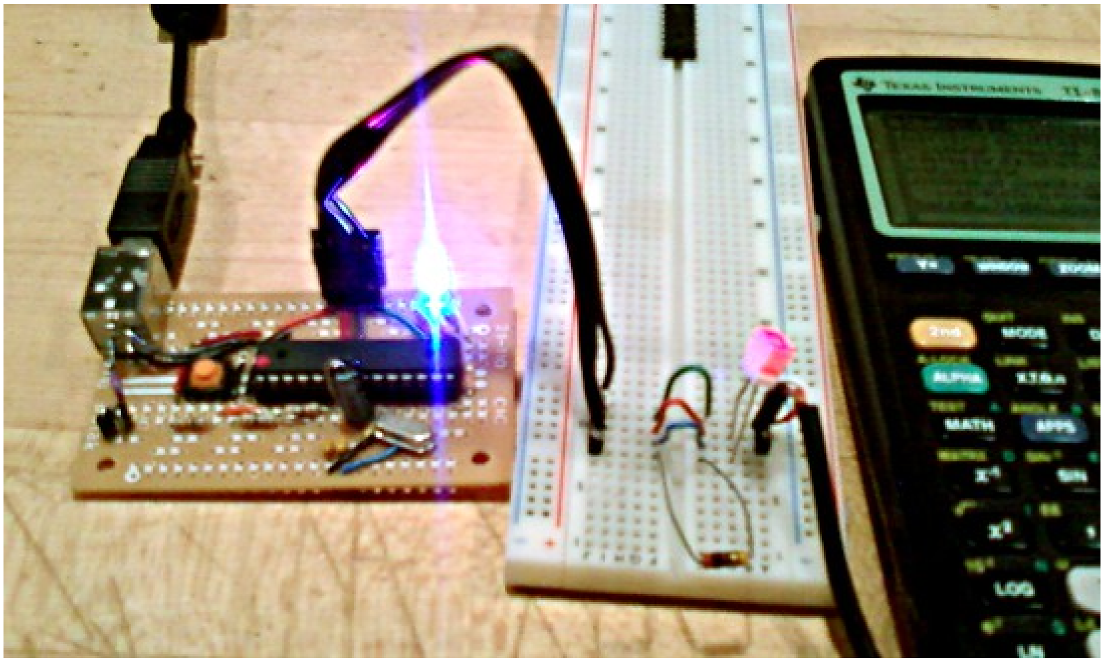
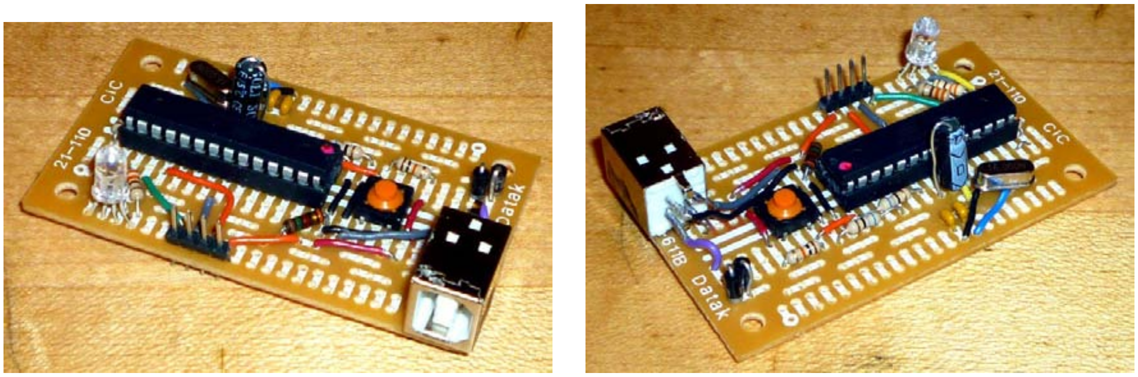


Fig. 6. View of a sample USBHID gCn bridge.

3.2 How the USBHID gCn Bridge Works

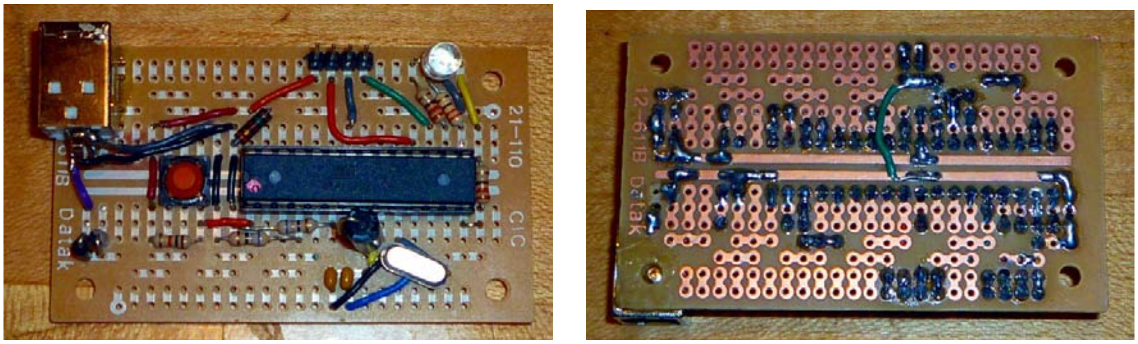
The USBHID bridge identifies to the host computer as a USB HID device, the category that includes such devices as mice, keyboards, and joysticks. Such devices have a mechanism to get and set device feature information, appropriately called Feature Reports; the USBHID bridge abuses this capability for data transfer. The USBHID bridge uses three types of feature reports, enumaterated below:

- **Report ID 1 (Status):** This two-byte report is used to poll the state of the bridge and protect against data loss. The first byte is a bitmask to indicate the presence of different types of pending data on the bridge. Starting from the least significant bit, bit 0 indicates whether a broadcast or frame is pending for transmission from the bridge to the host. Bit 1 indicates if a bridge-reset message is pending. Bit 2 indicates if a broadcast or frame of at most 32 bytes is pending for transmission from the bridge to the CALCnet$^{2.2}$ network. Finally, bit 3 indicates if a broadcast or frame of more than 32 bytes is pending for transmission from teh bridge to the CALCnet network. THe second byte provides the 8-bit ID of the last frame successfully received from the host by the bridge. The host uses this to ascertain whether a frame was lost and needs to be retransmitted to the bridge. Due to the nature of synchronous USB HID communication, no bridge to host frame ID is necessary to ensure correctness under normal operating conditions.
- **Report ID 2 (Short Frame):** Containing up to 32 bytes of data, this type of report is used in both directions to move data between the host and the bridge. In the bridge to host direction, the first byte is a `0x42` to signal the start of a valid data segment, then the remainder of the data is the sender, receiver, frame size, and frame payload. In the host to bridge direction, the format is slightly more complex. The first two bytes are a little-endian data segment size (including the two-byte size word itself). The next byte is this frame's ID, which will be transmitted when a Report ID 1 is requested from the bridge. The remainder of the data is the receiver, sender, frame size, and frame payload.
- **Report ID 3 (Full Frame):** Containing between 33 and 271 bytes of data, this type of report has identical usage and formatting to Report ID 2, differing only in the fact that it can hold



(a) Front of USBHID Bridge PCB, including CALCnet$^{2.2}$ header, RGB LED, and USB port.

(b) Back of USBHID Bridge PCB, including power regulation diodes, reset button, and crystal.



(c) Top view of USBHID Bridge PCB

(d) Bottom view of USBHID Bridge PCB

Fig. 7. Detailed photos of the (a) front, (b) back, (c) top, and (d) bottom of the USBHID Bridge PCB.

the payload and metadata of a maximum-sized 255-byte CALCnet$^{2.2}$ frame.

4 PROGRAMMING A GCN BRIDGE

A gCn bridge, which may be an Arduino and any related hardware from the previous section that you may have chosen to include, or a raw Atmega AVR μcontroller with its associated hardware, must be programmed before it can be used to connect a gCnClient (next section) and a CALCnet$^{2.2}$ network. Please note that if you built a USBHID Bridge (informally, the $10 Bridge), you must have an AVR programmer of some sort. If you have a friend with an Arduino, you can use the Arduino once to program your AVR chip, then use it on its own. Before you can program your Arduino (or AVR using an Arduino board), you may install the Arduino suite from http://www.arduino.cc/en/Main/Software, most notably so that you will have the `avrdude` software and related configuration files that are included. If you are familiar with AVR programming using avrdude or WinAVR, you may certainly use that instead. For the Arduino-based bridge, the file you will need to write to the Arduino is `gcnbridge2_Arduino.328.hex` or `gcnbridge2_Arduino.168.hex`, both of which are included with gCnClient releases and may be in the same zip file as this document. If not, visit http://www.cemetech.net to download it. The `.328` version is for any board with an Atmega 328p AVR, while the `.168` version is for any board with a 168 AVR. To build a USBHID Bridge, use `gcnbridge2_USBHID.328.hex` or `gcnbridge2_USBHID.168.hex`. The following instructions assume that you have a Windows, Mac OS X, or Linux computer with the Arduino suite installed. They also assume that you're using an Arduino board with either an Atmega168 or Atmega328 microcontroller, if not as the bridge, then at least as the AVR programmer. If you have something else, for example an

Arduino Pro (which has an Atmega644p microcontroller) or another programmer, please feel free to contact the author. The following instructions will guide you through the programming of your Arduino as a gCn bridge.

4.1 Programming a gCn Bridge From Windows

Open a command prompt and navigate to the folder where gcnbridge2.hex is stored. Make sure that you know the name of your serial port; it should be `COM3`, `COM11`, or another `COM#`. Please be VERY CAREFUL when performing the following step; it is intended for the Arduino Duemilanove (or other Atmega328 board) **ONLY**. If you have a different board, you may need a different command line. Specifically, if you have something with an Atmega168, use `"gcnbridge2_Arduino.168.hex"` instead of `"gcnbridge2_Arduino.328.hex"`, and the argument `-patmega328p` should instead be `-patmega168`.

```
"C:\Program Files (x86)\Arduino\hardware/tools/avr/bin/avrdude" -C"C:\
   Program Files (x86)\Arduino\hardware/tools/avr/etc/avrdude.conf" -
   patmega328p -cstk500v1 -P\\.\COM11 -b57600 -D -Uflash:w:"
   gcnbridge2_Arduino.328.hex":i
```

Be sure to replace `COM11` with the appropriate serial port name. If you are running a 32-bit Windows version, each `Program Files (x86)` will instead simply be `Program Files`. When `avrdude` finishes, you should see some kind of success message. If it fails, make sure that your board is properly connected and that the correct serial port has been specified.

If you are creating a USBHID / $10 Bridge instead of an Arduino bridge, use `gcnbridge2_USBHID.328.hex` or `gcnbridge2_USBHID.168.hex` instead, as appropriate.

4.2 Programming a gCn Bridge From Mac OS X

Open Terminal and navigate to the folder where gcnbridge2.hex is stored. Make sure that you know the name of your serial port; it should be something like `\dev\cu.usbserial######` or `\dev\cu.usbmodem######`. Please be VERY CAREFUL when performing the following step; it is intended for the Arduino Duemilanove (or other Atmega328 board) **ONLY**. If you have a different board, you may need a different command line. Specifically, if you have something with an Atmega168, use `"gcnbridge2_Arduino.168.hex"` instead of `"gcnbridge2_Arduino.328.hex"`, and the argument `-patmega328p` should instead be `-patmega168`.

```
/Applications/Arduino.app/Contents/Resources/Java/hardware/tools/avr/
   bin/avrdude -C/Applications/Arduino.app/Contents/Resources/Java/
   hardware/tools/avr/etc/avrdude.conf -patmega328p -cstk500v1 -P/dev/
   tty.usbserial-A9007VjP -b57600 -D -Uflash:w:"gcnbridge2.328.hex":i
```

Be sure to replace `/dev/tty.usbserial-A9007VjP` with the appropriate serial port name. When `avrdude` finishes, you should see some kind of success message. If it fails, make sure that your board is properly connected and that the correct serial port has been specified.

If you are creating a USBHID / $10 Bridge instead of an Arduino bridge, use `gcnbridge2_USBHID.328.hex` or `gcnbridge2_USBHID.168.hex` instead, as appropriate.

4.3 Programming a gCn Bridge From Linux

Please follow the Mac OS X instructions, but use a serial port such as `/dev/ttyUSB0` instead of `\dev\cu.usbserial######` or `\dev\cu.usbmodem######`. You will also need to change the path to your `avrdude` binary and `avrdude.conf` file.

5 USING THE GCNCLIENT SOFTWARE

The gCnClient software is provided for Windows (`gcnclient.exe`), Mac OS X (`gcnclient_macos`) and Linux (`gcnclient_ubuntu`). If you need the source code to compile the gcnclient software for your own operating system, please contact the author. Each of these gCnClient builds can be run from the command line of the user's preferred operating system. For Windows, this can be accessed via Command Prompt; most Linux and Mac OS versions have a Terminal. There is one optional argument to the gCnClient, `-v` (verbose), as well as five mandatory arguments specified below:

- `-n <hubname>`: Specifies the name of the virtual hub to connect to. All the calculators on one virtual hub will be able to communicate, and conversely, calculators on different virtual hubs cannot communicate. For example, a connection to the IRC channel #cemetech on EfNet is provided from the virtual hub `IRCHub`. Note that virtual hubs' names are case-sensitive.
- `-l <localname>`: Specifies the name of the local gCnClient. This may be something like "KermClient" to indicate the name of the user hosting the client, or it may be something like "MyschoolBridge" to elucidate the location or function of the bridge.
- `-s <server IP or hostname>`: The IP address or hostname of the gCn metahub to which to connect. The gCnHub metahub software may be distributed in the future; for now, contact Cemetech or the author for the IP address of the current metahub.
- `-p <port>`: This specifies the port on which the server is running the gCnHub software. It is normally `4295`.
- `-d <device type>`: This specifies the type of device connected; if omitted, it defaults to `arduino`. The valid options are `a` or `arduino`, which both specify an Arduino bridge device, `u` or `usb` or `usbhid`, all of which specify a USBHID / $10 bridge device, or soon, `d` or `direct`, which indicate a direct USB connection to a TI-84+/SE calculator.
- `-c <serial port>`: On Windows, this is usually something like `COM3` or `COM11`. On Linux, such as Ubuntu, it is `\dev\ttyUSB0` or something similar. On Mac OS, it is `\dev\cu.usbserial######` or `\dev\cu.usbmodem######`. If you are using a device other than an Arduino as your gCn bridge, such as the USBHID bridge or the direct USB solution, you should omit the `-c` argument.

You may wish to make a batch file (Windows) or a shell script (Mac OS, Linux) to faciliatate running the gCnClient software. Please consult the requisite documentation or the users on Cemetech on how to do this.

5.1 Mac OS X USBHID Bridge Notes

If you are using the Mac OS X version of the gCnClient software, and you have built a USBHID ("$10") Bridge, there are a few extra steps you need to perform before you can use gCnClient. First, download and install LibUSB 0.1 from http://www.ellert.se/twain-sane/. Next, unzip the `mac_gCnusbhid.kext.zip` file included with the gCnClient and install the `gcnhidshield.kext` kernel extension. It must be owned by user `root`, group `wheel`, and permissioned `0700`. You can then (and **only** then) plug in and use your USBHID Bridge.

If you are using Windows, or are using Mac OS X with an Arduino, you do not need to follow these extra steps. If you are using Linux, especially Ubuntu, read the following section.

5.2 Linux USBHID Bridge Notes

If you are using the Linux version of the gCnClient software, and you have built a USBHID Bridge, you must install the libusb 0.1 package before you can use your Bridge. You can run `sudo apt-get install libusb-0.1-4` (or the appropriate version number for your distribution. The other caveat under Linux is that you must run the `gcnclient_ubuntu` binary as the superuser using sudo due to unfortunate limitations in libusb.

5.3 Understanding the gCn Status RGB LED

To faciliatate using a gCn bridge/hub, the RGB status LED on the Arduino and UBSHID Bridge designs flashes a variety of colors and patterns to indicate normal activity as well as modes of failure. The color codes are enumerated below:

- **Red flash:** Frame or broadcast frame written from computer to CALCnet (⬛ 🟥 ⬛).
- **Blue flash:** Frame or broadcast frame written from CALCnet to computer (⬛ 🟦 ⬛).
- **Yellow, white, yellow sequence:** New calculator detected on CALCnet (🟨 ⬜ 🟨).
- **Alternating red and green:** Network transmission is jammed. If this persists, press the Arduino's reset button (🟥 🟩 🟥 🟩 . . .).
- **Solid red:** A short is present in the CALCnet2.2 wiring connected to the gCn bridge(🟥 🟥 🟥).

6 TROUBLESHOOTING

globalCALCnet is a complex system involving five different pieces of software just to get a piece of data from a calculator to the gCn metahub: in order, the on-calculator program using CALCnet$^{2.2}$, Doors CS, the Arduino running the gcnbridge2 firmware, the C++ gCn client, and the Python gCn metahub. Needless to say, unless all five pieces are working harmoniously, it's easy for glitches and hiccups to occur. Collected below are some common problems and the easy methods to solve them.

- **The gCn client is reporting an error such as "Error Binding to Socket"**
  Check that you are specifying a valid IP/hostname and port number. The port number is usually 4295, and a gCn metahub is often running at gcnhub.cemetech.net, but please consult the Cemetech forum at http://www.cemetech.net/forum if this these do not work. It's also possible that the metahub is offline or frozen, in which case you should also post on the forum.
- **I connected to a virtual hub, but I can't see my friends' calculators!**
  Make sure that you and your friends are on the same virtual hub; virtual hub names are case-sensitive (for example, IRCHub connects the CALCnet Chat! program to #cemetech on EfNet, while IRChub does not). Calculators on different virtual hubs cannot see each other.
- **Nothing is happening on my calculators.**
  There are many reasons this could be happening. Maybe your wiring is incorrect. It's much easier to troubleshoot your gCn bridge if an RGB status LED is connected (see Section 5.3 for color codes). If the LED only flashes red, is steady red, or oscillates red and green continually, there is probably a wiring fault in your bridge. If it was working properly, then suddenly jammed, especially if the status LED is oscillating red/green, press the reset button on the Arduino board. You should also check that the gCn client software is running, and that the gCn bridge is properly powered and connected.
- **I built a USBHID Bridge, and it isn't working.**
  Did you check the circuit? Did you include the two voltage-dropping diodes? Did you flash the AVR microcontroller with the requisite firmware? If you're on a Mac, did you install libusb 0.1 and the gcnclient.kext kernel extension? If you're on Linux, are you running as root? Is libusb 0.1 installed? If you answered yes to everything, and you have no idea what to try next, visit Cemetech and the members and staff would be happy to help you out.
- **Something else is broken, or the steps suggested here didn't fix the problem.**
  Please stop by the Cemetech forum at http://www.cemetech.net/forum or visit the IRC channel #cemetech on EfNet to chat with Cemetech staff and users (and the author, Christopher Mitchell) to help you troubleshoot your issues.

7 CONCLUSION

The different pieces of software related to this document are still under development, so please continue checking Cemetech (http://www.cemetech.net/) for new updates and releases. It should go without saying that the author and Cemetech disclaim any responsibility for damage to your calculators, your Arduino, your components, your computer, your internet connection, and yourself and your belongings if you use anything described in this document or published about CALCnet$^{2.2}$ and/or globalCALCnet . It has all been thoroughly tested, but as it is enthusiast-built software, it may have glitches or bugs. If you need help understanding anything described in this document, have suggestions, comments, complements, or criticisms, or want to learn more about CALCnet or gCn , please visit the Cemetech forum at http://www.cemetech.net/forum or join #cemetech on irc.efnet.net.

8 ACKNOWLEDGMENTS

The author would like to thank the members of Cemetech for their feedback, advise, and stress-testing assistance over the past eight years. Thanks especially go to Shaun ”Merthsoft” McFall for his moral support and hub-building experiences as well as his programming on CALCnet Chat! and his gCn debugging, Thomas ”Elfprince13” Dickerson for encouragement over the years and his help with the Mac OS port of the gCnClient, and to the other members of Cemetech and the calculator programming and hacking community. Thanks to Chrystina Montuori Sorrentino for moral support and Prof. F. Fontaine and the μLab research laboratory of the Cooper Union and Prof. J. Li and the Systems department of NYU for their respective indirect moral support.