Xbox One Controller Monitor Rev 2 (2018)

Overview

The 2018 model Xbox One Controller monitor provides all the functions of the original 2013 model plus many more! This document will describe how the system works, the functions of the system and how best to use it.

Theory of Operation

Timing Source

The Xbox One controller uses a pair of hall effect sensors to detect the proximity of the magnets on the analog triggers. For whatever reason these sensors are not supplied with constant power but rather are pulsed 125 times a second.

This pulse edge is used as a timing source for the new Controller Monitor giving us a default of 125 samples per second. This allows us to sync our samples to the controller logic itself. Faster sampling rates are achieved by using a secondary interrupt to further multiple this by 2X and 4X, giving additional sample rates of 250Hz and 500Hz.

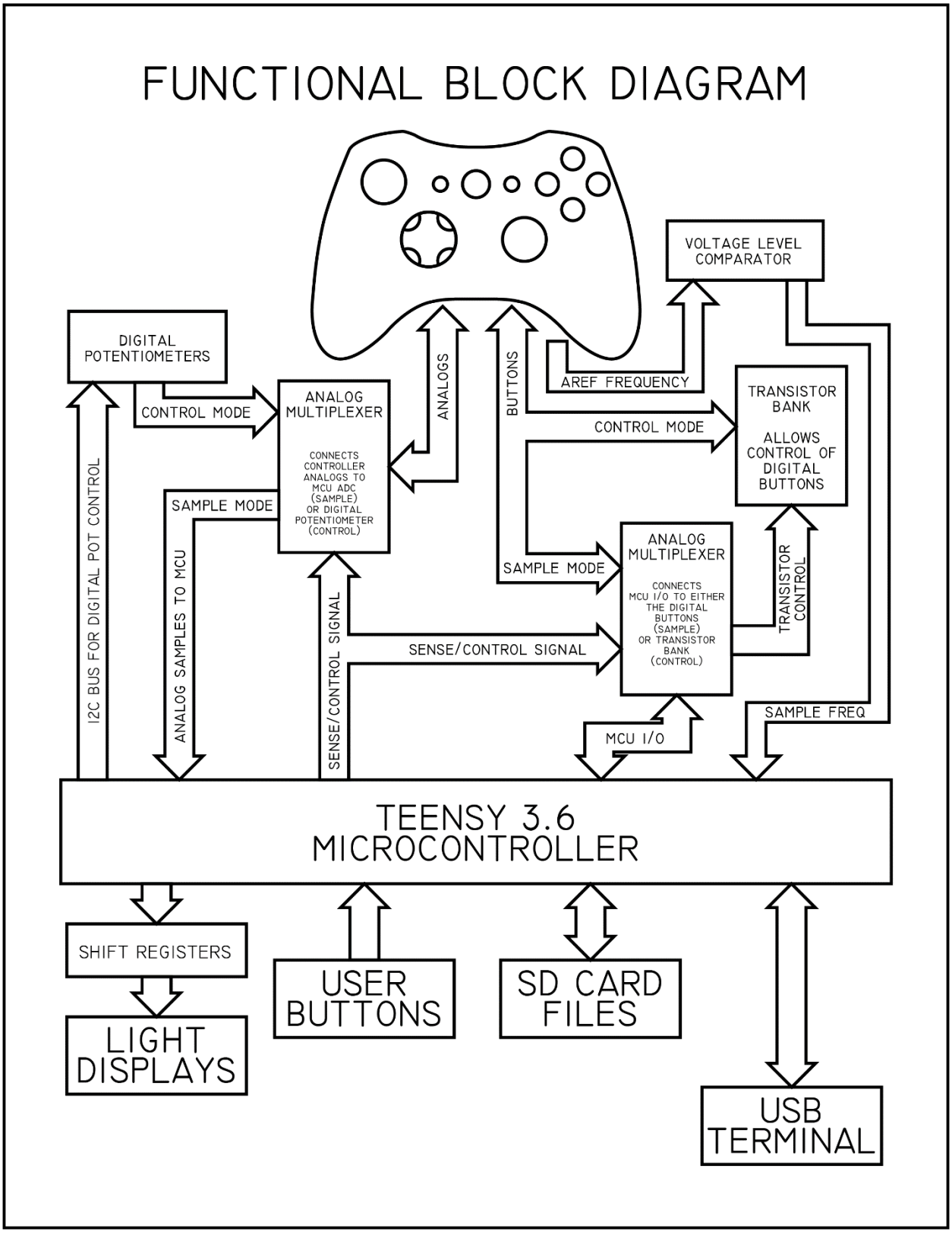
Sampling the Controller

The Xbox One controller is modified in the same way it has been since the 2013 model. All the signals are brought out to a header which connects to the Controller Monitor via an 8 foot long ribbon cable. Inside the Monitor these signals are connected to a Teensy 3.6 microcontroller (MCU) using its digital logic and ADC (analog to digital converter) ports. (see block diagram below)

In default mode the MCU turns these signals into visual LED representations in the same manner the previous models did. In addition, the samples are stored to a data array in RAM where they can be played back or saved to an external device. This RAM can also be loaded from an external source.

Playing Back to Controller

This new Controller Monitor is loaded with far more circuitry than the previous model allowing it to switch modes from Sampling to Playback. In Playback mode the Monitor can “virtually” press buttons on the controller and even affect the values sensed by the analog sticks and triggers. This not only allows you to “record once, play back many” but you can set up repetitive tasks and run playback from CSV files.


FUNCTIONAL BLOCK DIAGRAM
VOLTAGE LEVEL COMPARATOR
DIGITAL POTENTIOMETERS
CONTROL MODE
ANALOG MULTIPLEXER
CONNECTS CONTROLLER ANALOGS TO MCU ADC (SAMPLE) OR DIGITAL POTENTIOMETER (CONTROL)
ANALOGS
BUTTONS
AREF FREQUENCY
TRANSISTOR BANK
ALLOWS CONTROL OF DIGITAL BUTTONS
CONTROL MODE
SAMPLE MODE
ANALOG MULTIPLEXER
CONNECTS MCU I/O TO EITHER THE DIGITAL BUTTONS (SAMPLE) OR TRANSISTOR BANK (CONTROL)
SAMPLE MODE
TRANSISTOR CONTROL
I2C BUS FOR DIGITAL POT CONTROL
ANALOG SAMPLES TO MCU
SENSE/CONTROL SIGNAL
SENSE/CONTROL SIGNAL
MCU I/O
SAMPLE FREQ
TEENSY 3.6 MICROCONTROLLER
SHIFT REGISTERS
LIGHT DISPLAYS
USER BUTTONS
SD CARD FILES
USB TERMINAL

Packing List

You package will contain the following:

- Modified Xbox One controller with female data port along bottom.
- 20 wire ribbon cable connector with different plugs on both ends.
- Controller Monitor module.
- 3D printed angled stand.
- Suction cup Light Sensor with ribbon cable and plug attached.
- Universal 7.5V wall wart power supply with accessory bag of regional plug adapters.

Setting Up

Follow these steps to get the monitor up and running:

1. Plug the small pitch male end connector of the ribbon cable into the data port of the controller. Align the silver markings to each other to assure polarity. Take care not to bend the fragile pins.
2. Plug the other end of this cable into the rear left side of the Controller Monitor. This end is keyed and can only be inserted one way.
3. Install batteries and sync your controller either to a game console or a PC. The controller will not output valid data unless it has been connected to a host system.
4. Ensure the micro SD card is fully inserted in the back of the unit. The system will not boot properly without it.
5. Plug in the power supply and connect the barrel plug to the right upper side of the Controller Monitor. The monitor should turn on. You will hear the LED driver cooling fan turn on. Do not power up Controller Monitor without controller attached!
6. If all steps are followed properly the 4 digit LED counter on the Controller Monitor should start counting up seconds.
7. Plug a micro USB cable into the right hand side of the unit and attach the other end to a PC. Open a terminal program (baud doesn't matter) and type HP. You should get a list of commands back, confirming the connection is OK.
8. (Optional) Plug the 4 wire Light Sensor into the back of the unit. This slot is keyed and can only be inserted one way. The Light sensor has a suction cup allowing it to be stuck onto screens (give it a lick to make it stick) The area around the sensor has foam padding to help block external light.
9. Use a small flat head screwdriver to adjust the Brightness Control on the back of the unit. You can make the LED's very bright to keep them visible during high framerate video recording.

Now that your Controller Monitor is up and running you can use the Serial Terminal to perform a multitude of commands. The following sections describe all the available functions.

Serial Port Command List

The Monitor can be controlled with a variety of 2-character commands sent over the Serial Port (also called a UART or USB port)

These commands can be sent by a human using a terminal (such as PuTTY or the one built into the Arduino environment) or a PC program for automated control. For best results set your terminal to add Carriage Return and Line Feed (CR+LF) after every command. The baud rate doesn't matter as this is a USB connection. This port will be referred to as UART henceforth. When a command is said to "print something" that means over the serial port connection and into your terminal.

1X

Command type: Settings change

Sets the sampling rate to the default frequency of 125Hz. This is the base sampling rate of the XB1 controller itself. No multiplying will be applied. This setting allows 192 seconds of recording.

2X

Command type: Settings change

Sets the sample rate to 250Hz. This setting allows 90 seconds of recording.

4X

Command type: Settings change

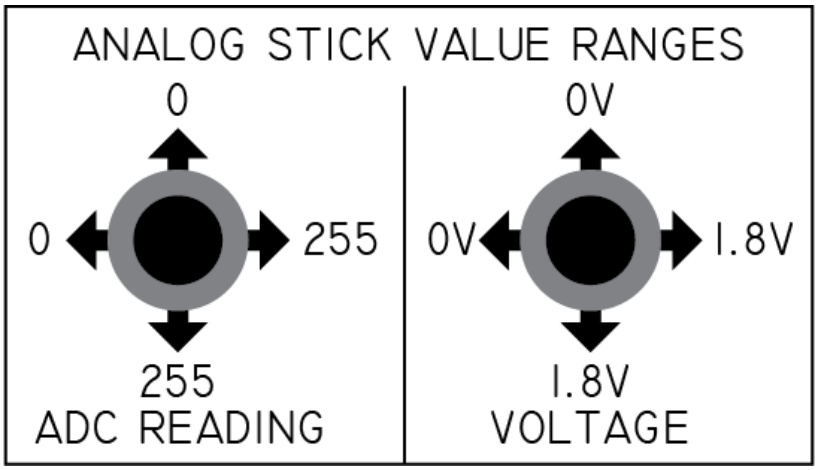
Sets the sampling rate to 500Hz. This setting allows 45 seconds of recording.

AV

Command type: Adjustments

Allows you to set default ranges for the analog sticks and triggers. This is necessary because every controller's analog range is slightly different.

The analog values are sampled with a 10-bit ADC giving a range of 0-1023. This is divided by 4 to fit inside a single byte and to also match the range of the digital potentiometers for playback.



After entering this command you'll be asked to press A without touching any analogs to provide a baseline value. Then, one-by-one, move the sticks and triggers as commanded pressing A to set each value. Try to keep the sticks centered and pressed as far as possible when setting values.

The captured values will be stored to EEPROM so you only need to do this once when pairing a controller to a monitor. Controllers/monitors sold as sets will paired before shipping.

AP

Command type: Get value

Prints out the current analog stick and trigger ranges. You can use this to get a good idea of what the value ranges in the CSV file mean. Note that they will never fully reach 0 or 255.

DS

Command type: Settings change

(Default) Sets the LED display to show seconds.

- Default Mode: Shows a seconds counter that increments until rolled past 9999 or cleared by other modes.
- While capturing: Shows how many seconds of record time are left in RAM.
- During playback: Shows how many seconds of playback have elapsed.

DF

Command type: Settings change

Sets the LED display to show frames. Frames increment at the specified sample frequency so 125Hz = 125 frames in a second.

- Default Mode: Shows a frame counter that increments until rolled past 9999 or cleared by other modes.
- While capturing: Shows how many seconds (not frames) of record time are left in RAM.
- During playback: Shows how many frames of playback have elapsed. If playback exceeds 9999 frames in decimal mode the display will roll over to 0. When displaying in hex mode a rollover is not possible because the system holds less then 65535 (0xFFFF) frames in RAM.

DH

Command type: Settings change

Sets the LED to display frame number in hexadecimal. In addition, the Frame # written to the CSV will also be in hex. Being a 4 digit display the max is now 65,535 samples (0xFFFF) which is more frame samples than the RAM can hold anyway so you never have to worry about the counter rolling over and becoming inaccurate. Combine this with high-speed camera footage for a perfect sync of your game, replay and CSV data.

DD

Command type: Settings change

(Default) Sets the LED to display frame number in decimal. Frame # written to the CSV will also be in decimal. Values past 9999 will rollover to 0.

HP

Command type: Immediate.

Help. Shows all the available standard commands.

ST

Command type: Immediate.

Status printout. Shows all the current settings.

Time of Flight Detection

T0 … T7

Command type: Settings change

(Default 7) Selects which Light Sensor (from 0-7) will be used for Time of Flight detection. The number also refers to the bit number when logging Light Sensor Values to a CSV file.

TB

Command type: Immediate.

Starts a Time Of Flight sample.

Ensure the sensor you selected in the previous command is placed on the LCD over a square that will transition from WHITE to BLACK upon button press in your game engine.

Tap any digital button and the system will measure and print how long in microseconds it takes for the square to change, along with the current average. Tap the button again to do another sample, or type TX to exit.

The system enters the tightest possible loop for accurate timing purposes. If the system hangs tap USER to abort out of the loop.

You can also use this with Event Programming playback mode. The system will execute the requested event then measure the time it takes for the light sensor to change from white to black. The result will be printed to the serial terminal. As stated above the monitor will be in a closed loop until the transition occurs.

Please note this has changed since the 2013 version of this device which looked for a black-to-white transition. The photoresistor can change faster when detecting white to black, thus the change.

TX

Command type: Immediate.

Exits out of Time Of Flight mode. Event Playback – if running – will continue as normal just without time of flight reporting.

TR

Command type: Immediate.

Resets the Time of Flight average to zero.

File Handling and Formats

FD

Command type: Immediate.

Prints disk directory. Folders (if any) will be listed but they are not currently supported when selecting filenames. All file access is root directory only.

FL

Command type: User input required.

Selects file to load into playback RAM. After typing this command you'll be asked for a 8.3 filename. The system expects a full 8 characters, period and extension (either CSV or RAW).

The first 12 characters you enter into the serial port will be used as the filename and anything else (carriage return etc) will be truncated off. Filename will then be checked for validity and automatically converted to upper-case.

If the file exists on the disk it will then be loaded into playback RAM. RAW format is quite fast loading while CSV is slower since it requires parsing. Any text flags present on the CSV will be ignored when read back into playback RAM.

Once the playback RAM has been loaded you can type PB to playback the captured data, for more information see the Data Logging Controls section.

FS

Command type: User input required, settings change.

Sets the name of the filename to be created (or overwritten) when exporting captured playback RAM to SD card. After typing this command you'll be asked for a 8.3 filename, which will be checked in the same way described above under the FL command. If the file already exists, you'll be notified that it will be overwritten.

Nothing is exported by executing this command, you're simply choosing what filename to use when something *is* exported. Playback RAM is either exported automatically after capture or manually by using the FM command.

When using sequential file playback use this command to select the first file to be loaded.

FI

Command type: Settings change.

Enables automatic filename numbering when saving files to SD card.

Format: XXXXX000.CSV, XXXXX001, CSV, etc.

Filename characters 6-8 will be overwritten when this is enabled. When this feature is enabled auto-incrementing will not take place until after the next iteration. For example, let's say the current filename as set by FS command is XXXXXAAA.CSV. You run FI command and now filename is XXXXX000.CSV. After you capture and export data to SD the number will be incremented. Upon the next capture/export it will be XXXXX001.CSV and so forth. If the value exceeds 999 it will rollover back to 0. This number is decimal only.

FN

Command type: Settings change.

Disables automatic filename numbering when saving files to SD card. The current number itself will not be cleared until you do so manually with the F0 command.

F0 (f zero)

Command type: Settings change.

Resets the automatic file number to zero.

FR

Command type: Format change to RAW

Changes the export format of playback RAM to raw data format (binary). The extension of the current filename (as set by FS command) will be changed to RAW automatically.

- When RAW is exported to an SD card a straight, headerless binary file will be written, 10 bytes per sample frame. The resulting file size in bytes will always be a multiple of 10.
- When RAW is exported over UART you'll need a control program on the other end to read in the data as it will not be readable on a standard ASCII terminal. As with the SD export it will be a simple and complete dump of playback RAM.

FC

Command type: Format change to CSV

(Default) Changes the export format of playback RAM to CSV file (comma separated values). The extension of the current filename (as set by FS command) will be changed to CSV automatically.

- When CSV is exported to SD card the first row of the CSV file will be labels for each of the columns, followed by one row per sample and finally an End-Of-File row. Check the **CSV File Format** section for more information.
- When CSV is exported over UART the first row labels are skipped and each line is one sample with visible commas. The final line contains the same End-Of-File information as the SD card saved version.

FU

Command type: Export target change to UART

This command sets the playback RAM export target to be the UART. This is where playback RAM will be dumped after a capture (if autosaving is enabled with the LA command) or when a dump is manually triggered (with the FM command). The format in which is it dumped is set by the FR and FC commands.

FF

Command type: Export target change to SD card file

(Default) This command sets the playback RAM export target to be the SD card. This is where playback RAM will be dumped after a capture (if autosaving is enabled with the LA command) or when a dump is manually triggered (with the FM command). The format in which is it dumped is set by the FR and FC commands.

The file will be saved under the currently set filename (set using the FS command) and the automatic numbering will be increased every time data is saved (if enabled using FI command).

FM

Command type: Immediate.

Exports the content of the playback RAM to the currently selected target device of either the SD card or UART USB port.

This command can be useful for transferring SD card files to PC, especially if you're using an automated serial port program:

1. Use FL command to load a file from SD to RAM.
2. FR, FU commands to select raw data (for speed) and UART as the output.
3. FM command then dumps the data directly over serial port so your control program can do whatever it needs to with it.

Please note commands FR, FC, FU and FF only change export settings. They do not trigger a save/export. To do this you must use the FM command or set the system to automatic export after capture with the LA command.

Usage of these commands allows you to transfer data to and from a host PC to the Controller Monitor's SD card without ever removing the card itself.

A mode to use the SD card as a mass storage device (over the USB connection) may by possible but isn't in the code at this time.

Input Capture Controls

LG

Command type: Immediate.

Sets pointer to zero position and starts capturing controller to playback RAM. The LED display will show how many seconds of recording time you have left in RAM (based off current sampling rate)

Tap the USER button or enter LG command again to stop capture. Running out of RAM also stops the capture. If autosaving is enabled (LA command) then the playback RAM will immediately be saved to the target device (SD card or UART) once capture stops.

LA

Command type: Settings change.

(Default) Enables the automatic saving of playback RAM (to selected device SD or UART) once capture is complete. In SD card mode it uses the currently selected filename (as set by FS command) and auto numbering settings.

LM

Command type: Settings change.

Disables the automatic saving of playback RAM after capture. To manually save to selected device use the FM command.

PB

Command type: Immediate.

Switches system into “Control Mode” and begins using the playback RAM back to drive the controller. The controller’s internal frequency is still used as the trigger point for data frame advance to ensure the data is posted at the correct time for the controller processor to handle it.

To stop playback before reaching end of file type PB again or tap the USER button.

P0...P9

Command type: Settings change.

Sets the system to "Looped Playback Mode" with the 0 – 9 character indicating how many seconds of delay between loops. Loops can repeat what is in Playback RAM (default) or load files sequentially from SD card (see PS command below).

To use, send a P0 – P9 command to select the delay between loops then start playback with PB command. Once the playback finishes system will wait X number of seconds then start over. This gives your monitoring equipment/game engine enough time to reset its tests to try again.

This looped playback feature also works with Event Programming mode. Enable with the P0-P9 command then type EB to begin event playback with loops.

PX

Command type: Immediate.

Stops automatic playback. Any currently running playback RAM will continue until it reaches end of data or is manually stopped.

PS

Command type: Settings change.

Sets automatic playback loops to use sequential file loading. This allows you to cycle through multiple files instead of just repeating what is stored in playback RAM. To use:

- Create a series of sample files with sequential numbers i.e. tests000.csv tests001.csv etc. You can use CSV or RAW, the latter will load faster.
- Select the starting file to load with the FL or /FLxxxxxxxx.xx command. That file will be loaded into RAM from SD card. Filename must have a 3 digit number in characters 6-8. It does not need to start with 000.
- Type PS to enable sequential playback. Type P0-P9 to select repeat delay.
- Type PB to begin playback. The first file (already in RAM) is played back.
- Once playback is complete the next sequential filename is loaded into RAM.
- After the delay this RAM is played back and the next file is loaded. If no more files are found system loops back to original file you indicated with the FL command and repeats from there.
- Type PX to stop looped playback when done.

PR

Command type: Settings change.

(Default) Sets automatic playback loops to use playback RAM. This will repeat whatever is in RAM over and over again, instead of loading sequential files from SD card.

Event Programming Mode

The monitor can also be manually programmed to play back certain events. This uses a simple programming language similar to BASIC and is entered using the serial terminal. This mode allows direct, precisely timed commands without having to create or modify a CSV file.

EP

Command type: Immediate.

Enters programming mode. You will stay in this mode until you enter the EXIT command. Up to 64 events can be programmed. The final event must always be the "END" command. On system boot event memory is filled with 64 "End" events ensuring any programming entered will always terminate cleanly. Event playback mode will always run at 125Hz, the same frequency as the controller. More information and syntax below. See below for programming details.

EB

Command type: Immediate.

Begins playback of events. The LED display will show an "E" plus the event number. As each event occurs the number increases. The current event number is also printed to the UART. If you've enabled Looped Playback with the P0-P9 commands the events will repeat upon reaching the end.

EE

Command type: Immediate.

Ends playback of events before reaching final END command. If auto-looping has been enabled (with the P0-P9 commands) the events will still play back after X seconds. You can disable auto looping with the PX command.

Event Programming Syntax

Event programming has a BASIC-like syntax and works as follows:

(event number)(space)(event code)(space)(duration in frames)

Example:

1 BAP 125 (press enter on terminal)
2 BAR 250 (press enter on terminal)
3 END 10 (press enter on terminal)

Result:

Event 1 = **B**utton **A** **P**ressed then wait 125 frames.
Event 2 = **B**utton **A** **R**eleased then wait 250 frames.
Event 3 = **END** playback (the time entered here is irrelevant and optional)

Commands must always be in sequence ie 1,2,3,4… and must start with the number 1. You can go back and re-enter previous commands (like in BASIC) such as the example below:

1 BAP 125
2 BAR 250
3 END 10
1 BAP 250 - *Wait, I wanted it event 1 to last 2 seconds instead!*

On system boot all 64 commands are filled with the END command. Event #64 must also always be left as an END command. Manually entering Event #64 is not allowed.

You can combine event playback with Time of Flight detection with the TB command. After each event the monitor will wait for the white-to-black transition to happen onscreen, report the time and continue. Please note that the system will halt until it sees the transition and *then* start counting the frames until the next event. The master system timer does not run while in the tight “wait for light transition” loop.

You can also automatically loop your events using the P0…P9 commands.

List of Event Programming Codes

These are the opcodes that can be programmed per event. They are all 3 letter uppercase codes. Analog commands (such as sticks or triggers) will instantly apply full-range motion when on.

Analog sticks:

- LAU – Left analog stick up
- LAD – Left analog stick down
- LAL – Left analog stick left
- LAR – Left analog stick right
- LAC – Left analog stick center (stick released)
- RAU – Right analog stick up
- RAD – Right analog stick down
- RAL – Right analog stick left
- RAR – Right analog stick right
- RAC – Right analog stick center (stick released)

Please note the analog stick can only be moved in one direction at a time, that is you can't move the left stick up and left.

Triggers:

- LTP – Left trigger pulled
- LTR – Left trigger released
- RTP – Right trigger pulled
- RTR – Right trigger released

Face buttons:

- BAP – Button A pressed
- BAR – Button A released
- BBP – Button B pressed
- BBR – Button B released
- BXP – Button X pressed
- BXR – Button X released
- BYP – Button Y pressed
- BYR – Button Y released

D-pad:

- DUP – D-pad up pressed
- DUR – D-pad up released
- DDP – D-pad down pressed
- DDR – D-pad down released
- DLP – D-pad left pressed

- DLR – D-pad left released
- DRP – D-pad right pressed
- DRR – D-pad right released

Shoulder buttons / L3 / R3:

- LBP – Left bumper pressed
- LBR – Left bumper released
- RBP – Right bumper pressed
- RBR – Right bumper released
- L3P – L3 pressed
- L3R – L3 released
- R3P – R3 pressed
- R3R – R3 released

List of Event Programming Reserved Words

Use these commands while typing in Events.

- **NEW** – clears event memory (fills it with END commands)
- **LIST** – lists events from 1 to the first END command found. You can also type LIST X to see a specific line.
- **CODE** – prints the list of the above opcodes for your reference.
- **EXIT** – exits programming mode.

UB

Command type: Immediate.

Starts executing User Code. Will run whatever is in function userBegin() and then the following:

- Every sample/control cycle: function userSampled() is called. Note this function is not called during programmed event playback.
- Every kernel cycle: (appx 50kHz) function userPolled() is called.

See User Code section for more information.

UE

Command type: Immediate.

Runs whatever is in function userEnd() and then disables User Code.

Direct UART File Commands

Your debugger program running on the UART port can also send low-level commands to the monitor. This gives you more options for autonomous logging and control.

Direct UART commands are always prefaced with a forward slash symbol "/".

/FLxxxxxxxx.RAW or .CSV

Loads filename xxxxxxxx.RAW/CSV from SD card into Playback RAM. This works much like the FL command except the monitor does not prompt you for a filename.

If the file exists it is loaded from the SD card into RAM (and parsed if it is a CSV file). It will not playback automatically once loaded.

/FSxxxxxxxx.RAW or CSV

Sets the filename that will be used when logging data. The type of file RAW or CSV will be set by the FR and FC commands. Default is CSV (slower).

If autonumbering is enabled characters 6-8 of the filename will be overwritten.

This command is also used for selecting a starting filename for sequential SD card playback.

/UR

Tells the monitor that you are going to send raw data over the UART port to fill up the playback RAM.

Once the command has been acknowledged "READY TO RECEIVE RAW" send 10 bytes at a time as described in the Data Logging Formats section. You must also send the End of File indicator (also described in that section) when you're done sending data. The maximum number of samples is 24,000 (240,000 bytes) and the final sample must be the end of file indicator.

/UC

Tells the monitor that you are going to send CSV data over the UART port to fill up the playback RAM.

Once the command has been acknowledged "READY TO RECEIVE CSV" send the CSV file one line at a time as described in the Data Logging Formats section. As with RAW, you must also send the End of File indicator (also described in that section) when you're done sending data. The maximum number of CSV lines is 24,000 and the final line must be the end of file indicator.

/PXXnnn

Sets the system to Control mode and manually sends a value to one of the digital potentiometers.

XX values: (which potentiometer)

- LX – left analog X (left or right)
- LY – left analog Y (up or down)
- LT – left analog trigger.
- RX – right analog X (left or right)
- RY – right analog Y (left or right)
- RT – right analog trigger.
- (Anything other than above) – Exit manual pot control.

NNN value: (what value)

- A 3 digit number from 000 to 255. You must include 3 digits includes leading zeros.
- See the AV command section for analog range examples.

/Cn

Changes the speed of the cooling fan. N is a character from 2-7. The duty cycle set will be N * 1000 out of a possible 8000 (higher the number faster the fan).

This value is saved to EEPROM when set and loaded on system boot. Default is 3000.

/AXXXnnn

This command tweaks the ranges of the digital potentiometers that control analog signals during playback.

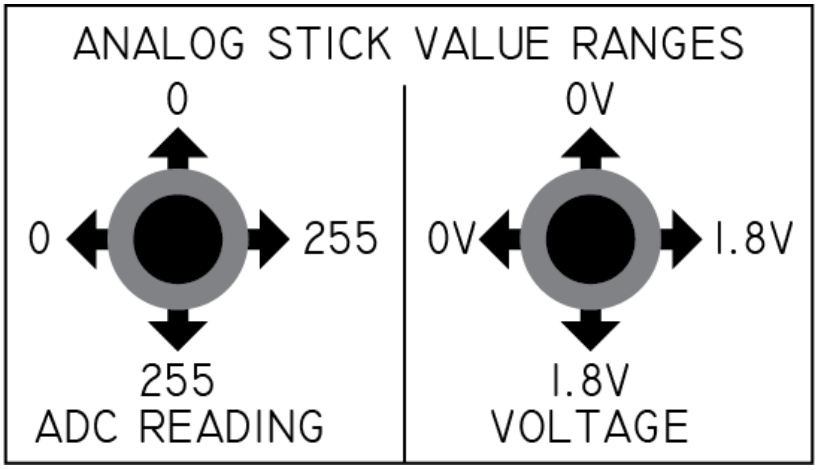
The digital pots have a range of 0-255 (with roughly 128 being centered) but using this full range results in inaccurate playback since they are wired in parallel with the existing analog sticks of the controller.

Therefore, slightly truncated settings are used such as 29 low and 235 high. These are set when built and stored in EEPROM, but this command allows you to tweak them. On playback these ranges are used to control the analog sticks with the digital potentiometers.

/AA will print the existing values for your reference, example below:

Current digital potentiometer stick ranges:
LXL=20 LXR=244
LYU=30 LYD=230
RXL=30 RXR=230
RYU=18 RYD=236
LTL=0 LTH=12
RTL=0 RTH=12

Left and up movements bring the stick closer to 0 volts and therefore use lower numbers. Right and down movements bring the stick closer to 1.8 volts and therefore use higher numbers.



Note the usage of “upper” and “lower” below refers to the numbers used, ie, pushing analog stick left results in a lower number than pushing it right. The analog triggers work in a similar way, the lower number “pulls” the trigger while the higher number is the released state.

Breakdown of the command /AXXXNNN:

XXX:

- LXL – set left X lower range (left) A lower number pushes further left.
- LXR – set left X upper range (right) A higher number pushes further right.
- LYU – set left Y lower range (up) A lower number pushes further up.
- LYD – set left Y upper range (down) A higher number pushes further down.
- RXL – set right X lower range (left) A lower number pushes further left.
- RXR – set right X upper range (right) A higher number pushes further right.
- RYU – set right Y lower range (up) A lower number pushes further up.
- RYD – set right Y upper range (down) A higher number pushes further down.
- LTL – set left analog trigger lower range (pulled). A lower number pulls the trigger more.
- LTH – set left analog trigger higher range (released). A higher number releases the trigger.
- RTL – set right analog trigger lower range (pulled). A lower number pulls the trigger more.
- RTH – set rightanalog trigger higher range (released). A higher number releases the trigger.

NNN:

- The value to set. Must be 3 characters 000-255 including leading zeros (010, 001, etc). Be sure to use the /AA command to get the current values for reference.

Usage example: You capture the movement of a player moving left, then right back to the original position. On playback you notice the player moves further left than they did originally during capture. This means the LXL value is too low – the virtual stick is being pushed too far.

To remedy this, slowly increase the LXL value to limit the range of the "virtual" stick movement.

Another example: You capture a player aiming with the right analog stick. They aim down, then back up to center. On playback you notice the down movement didn't go as far down as it did during capture. This means the RYD value is too low – it's not pushing the virtual stick down enough.

To remedy this, slowly increase the RYD value until the movement matches.

Keep in mind drastic changes to either end of a single axis (like LXL and LXR) can affect the other end as you're changing the ranges used by the digital pots. It's best to experiment with small 1 or 2 tick changes. Sometimes you may need to tweak both ends, ie, LXL020 and LXR230 gets changed to LXL025 and LXR235 (a shift of 5 to the right).

If you notice "jitter" on the analog triggers when they are in a released state (notice in the Game Controller – Properties Windows dialog) you can increase the LTH and RTH values to "settle" them down. If the analog triggers aren't being "pulled" far enough lower the LTL and RTL values (0 being the limit).

Data Logging Formats

The sample data is stored in microcontroller RAM in a large, byte-aligned variable array with a size of 240,000 bytes. This consumes most of the microcontroller's RAM though some is left over for the stack and future use.

Each sample is an array of 10 bytes stored in RAM.

[byte 0][byte 1][byte 2][byte 3][byte 4][byte 5][byte 6][byte 7][byte 8][byte 9]

The usage of each byte is as follows:

- Byte offset 0 – Left analog X value.
- Byte offset 1 – Left analog Y value.
- Byte offset 2 – Right analog X value.
- Byte offset 3 – Right analog Y value.
- Byte offset 4 – Left analog trigger value
- Byte offset 5 – Right analog trigger value
- Byte offset 6 – High byte of digital buttons (top 4 bits unused)
- Byte offset 7 – Low byte of digital buttons
- Byte offset 8 – Control byte (camera controls, flags)
- Byte offset 9 – Value of light bar

The final sample in RAM stores the End of File indicator as well as the Frequency settings used during sampling. Playback is stopped when this byte pattern is found.

- Byte offset 0 = 255 (EOF indicator)
- Byte offset 1 = 255 (EOF indicator)
- Byte offset 2 = Sampling Frequency (1 = 125HZ, 2 = 250HZ, 4 = 500Hz)
- Byte offset 3 = 0
- Byte offset 4 = 0
- Byte offset 5 = 0
- Byte offset 6 = 0x0F
- Byte offset 7 = 0xFF
- Byte offset 8 = 0
- Byte offset 9 = 0

Only the first 3 bytes are used for EOF detection, the rest are reserved for future use. On the final sample the buttons are set to all high (not pressed) so no flags will be visible on the final line of the CSV.

When a file is loaded from SD/UART into RAM the end of file entry sets the monitor to the Playback Frequency as specified in byte offset 2. This ensures the file is played back at the same speed it was recorded. If you desire to playback at a different rate (giving a speed up or down effect) you can manually change the value after the load using the 1X 2X 4X commands.

When saving/loading Playback RAM in RAW format the data is directly copied. No conversion takes place. If data has been created externally it must contain the End of File entry as described above.

Playback Data in CSV Format

The playback RAM can be exported / imported in CSV format (via SD card or UART) to allow human-readable viewing & tweaking. This is the default when booting up the system.

The CSV file format is based off the default "Save as CSV" setting from Microsoft Excel. When looked at with a hex editor the data for a single row should be as such:

0,0,0,0,0,0,0,0,0,0,0,0[/CR][/NL]

The values are stored as ASCII numerals and there is a comma after each value. The final item in a row does not have a comma after it and is instead terminated with carriage return and new line (0x0D and 0x0A).

The data presented in a CSV format is the same as what's stored in RAM (or exported as a RAW file) but with some formatting changes to make it easier to read. The first row of data are text labels to define each of the columns, which are as such:

- **Column A– Event** #: Which event this is. Starts with event #1. This value isn't important internally as the microcontroller simply uses a pointer in RAM. It's presented here as a human readable guide. Please note the RAM pointer internally is zero-aligned.
- **Column B – Left analog X value.** The values listed for analog sensors are how the microcontroller reads them, it isn't necessarily relevant to game code. These are the values used to drive the digital potentiometers when the monitor "takes over" the controller. See AV command for range information.
- **Column C – Left Analog Y value.**
- **Column D – L3 click status.**
- **Column E – Right Analog X value.**
- **Column F – Right Analog Y value**
- **Column G – R3 click status.**
- **Column H – Left Analog Trigger value**
- **Column I – Left Bumper status.**
- **Column J – Right Analog Trigger value**
- **Column K – Right Bumper status**
- **Columns L-S – Status of face buttons and D-pad**
- **Column T – Camera control flags, if any.**
- **Column U – Light sensor value.**
- **Column V – Text flags.**

For the digital button "status" a 0 means not pressed and a 1 is pressed. Please note this is inverse of the actual electrical state but makes more sense for human-readable text.

End of File Indicators

The last row in the CSV is the "End of File" indicator. The data is as follows:

- **Column A– Event #** (final event)
- **Column B – 255** (end of file flag)
- **Column C – 255** (end of file flag)
- **Column D – Sampling Frequency** (1, 2 or 4)

The rest of the columns in the End of File row do not matter and are currently ignored by the CSV parser.

Camera Control Flag

Placing a 1 in this column will send a high pulse to a camera attached to the Controller Monitor. The pulse width will be 1000/rate of playback. Example 1000/250Hz = 4 ms. Check your camera's documentation for minimum required pulse width. To widen the pulse simple place the Camera Control Flag in a few consecutive frame lines.

Usage example:

1. Record events to a CSV file.
2. Load CSV file into Excel and find your event using the Text Flags (see below).
3. Modify the CSV file to place Camera Flags just before the desired event occurs, depending on how long it takes your camera to get up to speed.
4. Load modified CSV file into Playback RAM and repeat the test.
5. Your high-speed camera starts recording just before specified event occurs.

Light Sensor Data

The Controller Monitor can sample up to 8 bits of light sensor data. The standard kit comes with a single light sensor attached to light sensor bit 7 (MSB). White light = logical 0 black = logical 1. View the schematic here.

You can program your game engine to change squares on the display from white to black to pass data along to the controller monitor.

While sampling the controller the light sensor byte is written to this slot in memory. When the data is played back the light sensor byte is re-written every time. This allows you to see if anything changes with each playback iteration.

Light Sensor usage example:

- Record events by sampling controller.
- Modify the CSV file as desired, taking note of the initial light sensor values.
- Load CSV file back into RAM and playback. The light sensor value will be overwritten during playback.
- Dump the newly modified RAM over UART to save these new light sensor values.
- Automatically make tweaks to game engine timing.
- Playback CSV file again, getting new values from light sensors once again.
- Dump RAM over UART and compare light sensor values to previous test, repeat.

Text Flags

All of the controller events are readable as number changes in the CSV file, either analog values or Booleans. To make these changes easier to spot text flags are added to the end of each row. This makes skimming the CSV data and looking for events easier.

List of text flags:

- **Left trigger** – was pulled
- **Right trigger** – was pulled
- **Face buttons** – A B X or Y was pressed
- **D-pad** – any direction was pressed
- **Shoulder buttons** – either shoulder button pressed
- **L3/R3** – either one was pressed.

These are generalized to save space but will point you to which CSV rows have the changes you're looking for (easier to spot than zeros or ones!).

Please note these text flags are ignored when loading CSV files back into playback RAM. Consider them "code comments". They are automatically created based off sample data when a CSV file is saved or exported.

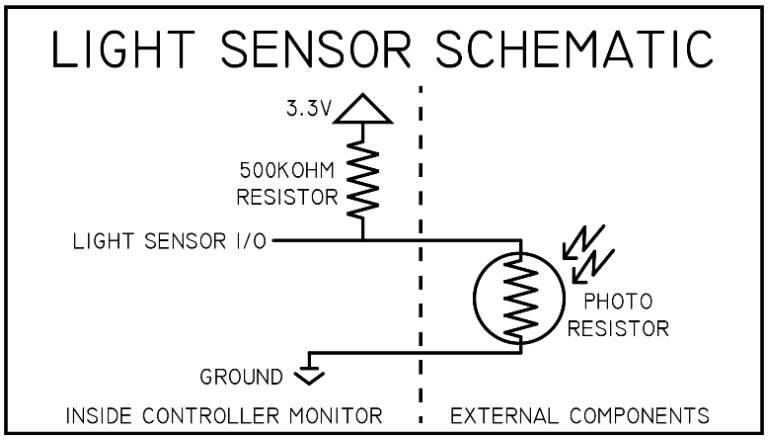
Modifying CSV Format

The CSV files created by sampling the controller can be easily loaded and re-saved using Microsoft Excel. Be sure to use the proper format (as described above) when modifying or creating new data so the monitor's CSV Import Parser ("parser") works correctly. Consider these guidelines when changing anything:

- Do not add formulas to the CSV file.
- The first row (column labels) is ignored by the parser or more specifically the first line in the file ending with /CR/NL is ignored by the parser. If you omit the first row labels this means your first sample will be ignored, so leave the column labels for best results. You can always "freeze" that row in Excel to make browsing easier.
- Column A, the Frame Number, is also ignored by the parser. During import each line is considered a sample and an internal array pointer counts up from 0 in steps of 10.
- Analog values must range from 0-255. Never use more than 3 digits or negative numbers for any analog value. See AV commands for range examples.
- Digital buttons must be a 0 or 1.
- Column T, the Camera Value is an integer in the CSV but on playback it is analyzed bitwise.
    - 0b00000001 is the signal to start camera. The other 7 bits are for future use.
    - For now you can easily do this as 1 to start camera, 0 to not stop camera.
    - If for instance future use has the MSB as a command 0b10000000 you'd need to combine this and convert to integer. Meaning 0b10000001 would be entered as 129 in the CSV.
- Like camera value column U the Light Sensor Value is a binary number written as integer.
    - If no sensors are attached this value will read 255 (0xFF or 0b11111111).
    - If sensors are attached their bits will read 0 for white and 1 for black.
    - For instance if you're just using the single sensor on bit 7 (default) the reading will be 255 for black (bit 7 = 1) and 127 for white (bit 7 = 0)
    - You could use 8 sensors and 8 light squares to pass entire bytes to the controller monitor if desired.
- After the parser has loaded the Light Sensor Value anything else on the row is ignored until a carriage return-new line is found. This includes any flags in column V. You can use column V or higher to write notes or other data.
- The final entry must have the End of File indicators as listed here.

Light Sensor Schematic

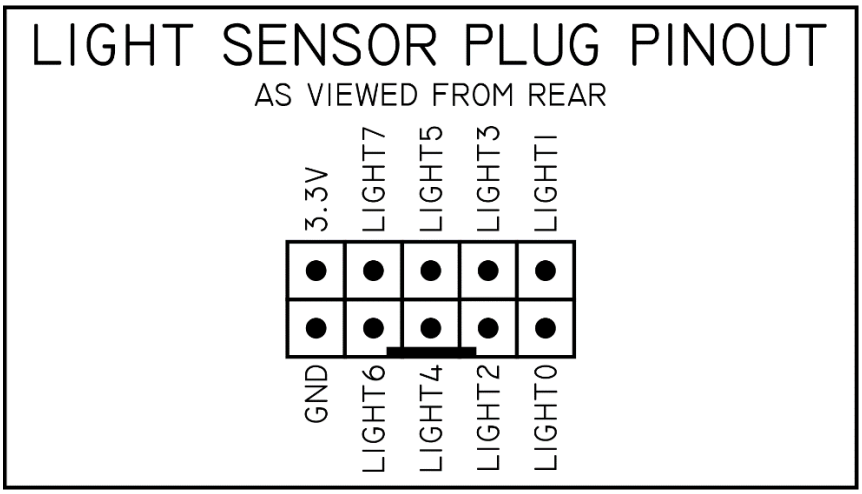
The light sensor uses a simple voltage divider to change the state of an I/O port based off light. The 8 possible light sensors are attached to single port in the MCU allowing fast reads.



- If light saturates the photoresister current flows through it bringing the voltage closer to ground. This gives us a state of 0 for when light is present.
- If light is not present the photoresister is in a state of dark resistance and the voltage moves closer to 3.3V giving us a state of 1.
- The photoresister responds faster to a white-to-black transition than vice-versa.

Photodiodes are much faster than photoresistors but they were not used because their high speed could actually be triggered by the oscillations of a monitor's backlight. Thus the slower performance of a photoresister was more appropriate. It's still well within the framerates of modern games.

If you wish to add more sensors use the drawing below as a connection guide:



Coding Environment

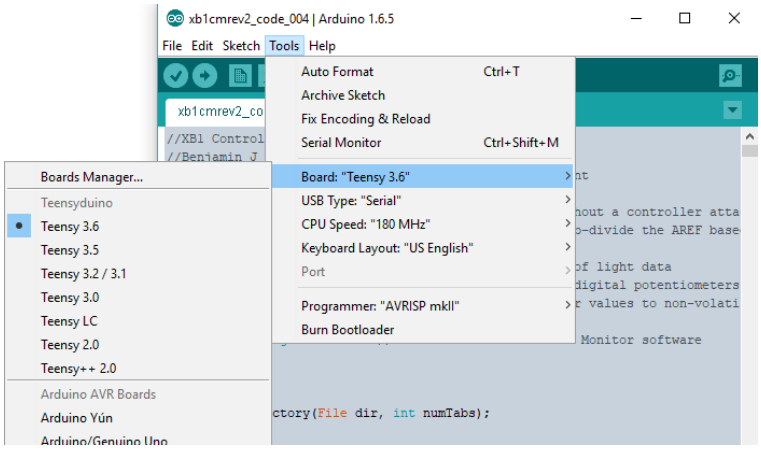
The 2018 Controller Monitor is based around the Teensy 3.6 USB Development Board. For more information visit https://www.pjrc.com/store/teensy36.html

The current source code for the monitor can be found here:

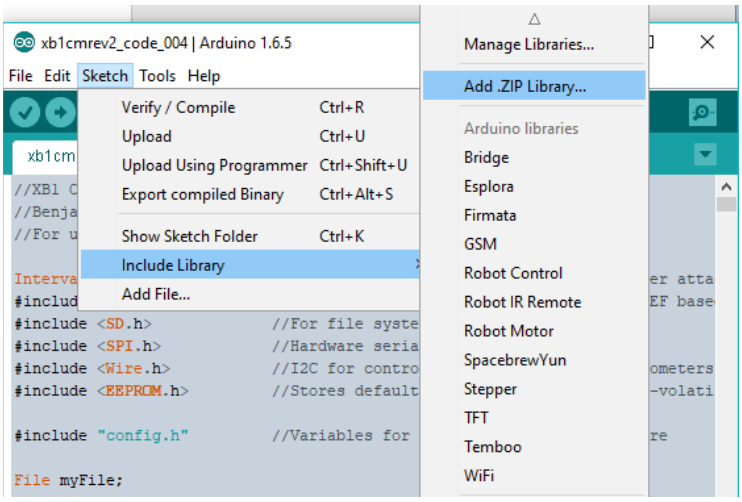
- https://www.benheck.com/Downloads/xb1cm2018_support.zip
- https://github.com/benheck/xb1cm2018

This board is programmed with the Arduino environment. To use a Teensy board software must added onto the Arduino install. To setup a development environment for the Teensy follow these steps:

- Visit https://www.pjrc.com/teensy/td_download.html to download the Teensy software add-on installer.
- Download a version of Arduino that is compatible with the Teensy 3.6 add-on software. According to the above website it's currently supported on Arduino versions 1.0.6 and 1.6.5-r5 and 1.8.1 and 1.8.5 and 1.8.6 and 1.8.7 and 1.8.8. I used 1.6.5-r5 for development.
- Unzip/install Arduino to your computer. Ensure it's working, then close it.
- Run the Teensy installer and point it to the Arduino directory. It will install software and drivers.
- Once Teensy is installed re-open Arduino. Clicking on Tools should give you the following Teensy options:



- Select Board: "Teensy 3.6", USB Type: "Serial", CPU Speed: "180MHz".
- The source code linked above will also contain a ZIP file called TeensyDelay-master.zip. This needs to be installed before the code will compile.



- Click Sketch -> Include Library -> Add .ZIP library then select the TeensyDelay-master.zip file from the source code.
- Load xb1cmrev2_code_xxx.ino. Tabs should appear for the ino file and a config.h file.
- Click the check box to verify code (compile without upload). If it compiles OK you've installed everything correctly!
- Turn on the Controller Monitor and plug in a USB port. This port may not show up right away under Tools -> Ports.
- Click the right arrow button (compile and upload). Arduino should find which port your Teensy is on and upload the code. A separate upload window will appear showing the transfer. If it can't restart the Teensy automatically tap the PROGRAM button on the back of the unit. Programming the unit does not erase any EEPROM settings.
- After programming the Teensy should now appear under Tools -> Ports. To double check click the Magnifier icon in the upper right (open serial terminal) and you should get a connection. Type HP to see if it responds to commands.

User code

There are (4) functions built into the main kernel loop intended for custom code. They are as follows:

- void userBegin() {} //Called when User Code is activated via command terminal
- void userPolled() {} //Called every kernel cycle (appx 50kHz)
- void userSampled() {} //Called every controller frame cycle (depends on selected frequency)
- void userEnd() {} //Called when User Code mode is disabled.

Putting code into these functions is the easiest and safest way to modify the code. These functions are called in other parts of the kernel at the most opportune times as to not affect the sampling/playback features.

While most of the RAM is consumed for data samples the actual program flash is quite large (1MB) and mostly empty. The only real concern when adding code is to not affect the sample loops but again, the main kernel runs so much faster than the controller sampling rate this isn't a huge concern.

The main thing to watch out for is using Serial.print commands in rapid succession. They do consume a decent amount of time and if used every kernel cycle can indeed slow down the system. If you choose to stream data over the serial port use a counter-timer to limit it to around 100 samples per second.