Arduino Strobe Controller

Written By: Steven Stuber

What is it?

The Arduino Strobe Controller is a box that outputs a differential signal on a DMX XLR cable to drive a set of daisy-chained LED strobe panels. The input to this box is the USB cable that receives messages sent by the Windows, Linux or Mac side command line application. The application deals with handshaking, transferring data and error-checking between itself and the Arduino. By supplying a port, frequency and width, a strobe signal can be generated between the ranges of 0.5Hz and 10kHz at any desired duty cycle. The main algorithm used to generate the signal is performed solely in assembly and is accurate to within one operation, taking 1/16th of a microsecond.

How to use it

The strobe controller is very simple to use. An example is shown below.

```
Ex1. ArduinoStrobeController.exe –d COM4 –f NTSC –w 6600
```

This statement contains a –d command that expects the next string to be the device location. On Linux or Mac, the location is in the form /dev/tty…or similar, and on Windows it is COM1, COM2…etc. The –d command is completely optional, but does improve the speed of the command taking effect. This is because, if not supplied, the program has to search through all connected COM ports until it finds the Arduino. This can take upwards of two to six seconds. Another use for –d is to supply the port as "find"; this will cause the regular program flow to be bypassed, and the application will instead simply return the string of the port name on which the Arduino was found. This same string can then be used for future calls with "–d xxxxx", where xxxxx is the string found using "-d find".

The –f command specifies frequency; in this case we see the string NTSC. NTSC encodes 29.97Hz, which corresponds to the frame rate of the cameras. This –f command needs to be supplied unless either the Off or On command is used. More strings like NTSC are listed further down.

The –w command specifies width in microseconds, width being the amount of time the LEDs are on for during each period. The width can also be specified as a percent, enabling duty cycles to be converted automatically for convenience to the user. The –w command is also required unless the Off or On command has been given.

Ex2. ArduinoStrobeController.exe on

This example shows how little we actually need to supply to the program in order to achieve results. The On command will automatically generate the –f and –w commands so that the Arduino will be on 100% of the time. This works the same way for the Off command.

The special built in frequency values are as follows: (NTSC = 29.97003)

NTSC	NTSCx2	NTSCx4
NTSCx8	NTSCx16	NTSC/2
NTSC/4	NTSC/8	NTSC/16

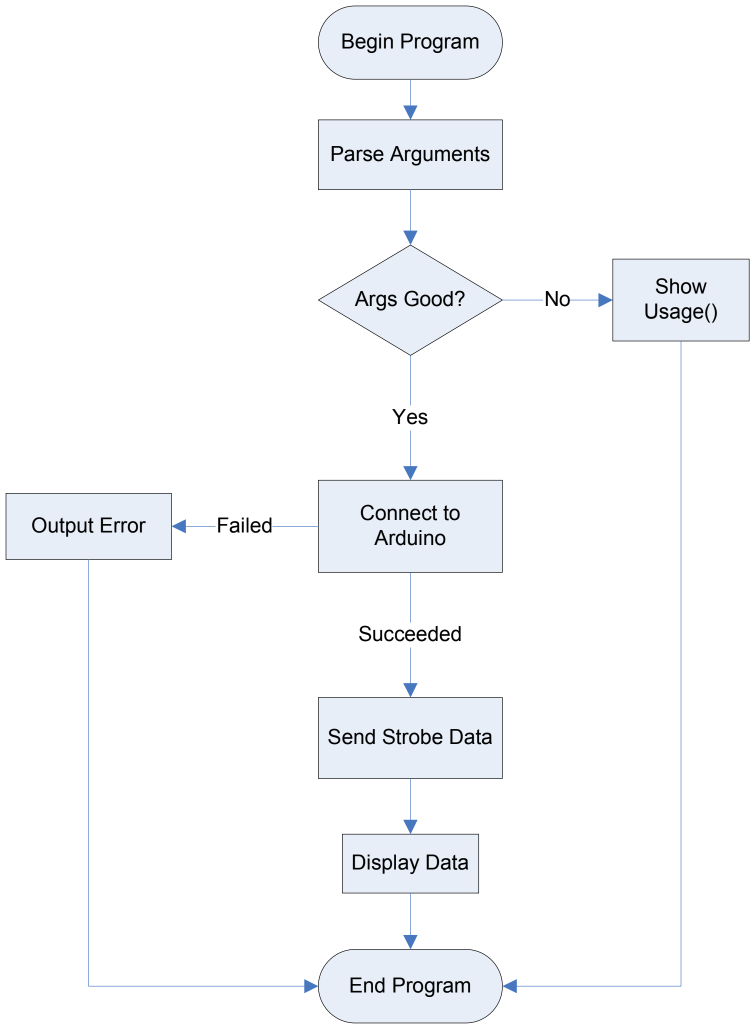
How Communication Works

The Arduino waits for a message from the computer before initiating any strobe patterns. In this initial state, the LEDs are off. The computer sends a “hello” message until the Arduino responds with a “ready” message. At this point, the computer tells the Arduino the desired frequency and width. After several copies of this have been sent, the Arduino compares them and decides to continue — that is, if they are all the same and no errors have been introduced due to noise or bad cabling — or it retries until a clean signal is received.

The Arduino then becomes unresponsive and strobes until reset. This requires us to toggle the DTR line to reset the Arduino if we want to specify new parameters. By resetting the Arduino, we see the LEDs come full-on for a short pulse and then off again. This is a result of the way the Arduino resets. Eliminating this flash, if desired, would require switching the position of the two signal pins inside the box (however may introduce some other flash due to transience). This would change a digitalwrite or fastwrite HIGH to turn the LED off, and a write LOW to turn the LEDs on. A change in all instances of these functions would be required in the Arduino side code.

Appendix

Computer Side Flow Chart

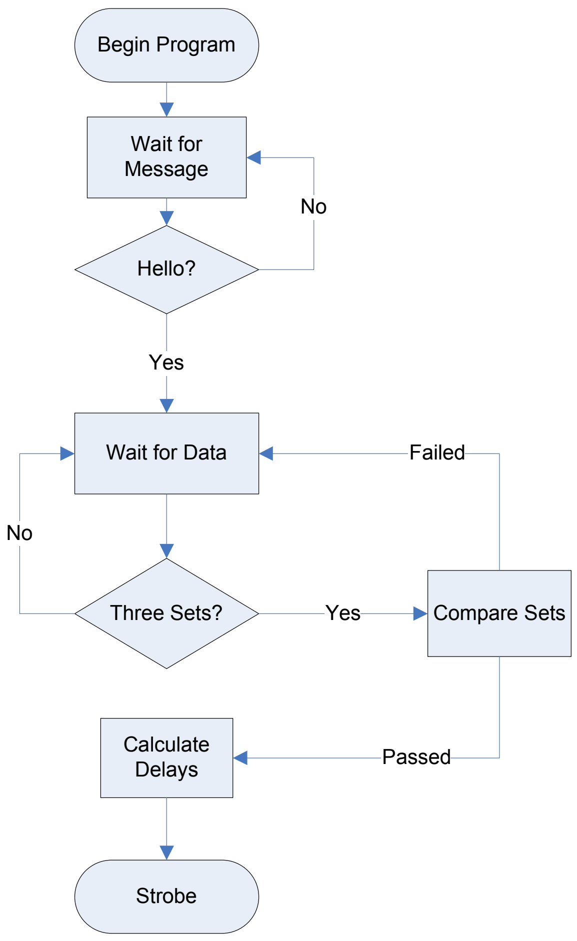


Computer Side Code Notes

The computer side application begins by parsing the arguments that were supplied. If bad or not enough arguments were found, we show the usage function and end the program. Once good arguments have been received, we attempt to connect to the Arduino. Hello commands are sent to the port specified or, if no port was specified, we will slowly attempt to connect to and say hello on every available port that could be an Arduino. If no response is found, we send an error message to the screen and warn the user that the Arduino may not be plugged in. Once connected to the Arduino, we formulate the data to be sent by padding it with characters in the format: "ssXXXXccYYYYff", where XXXX refers to frequency and

YYYY refers to width in cycles. “ss” means we are starting a set of data, “cc” means we are changing to the next item in the set, and “ff” means we have finished the set of data. The frequency could be 23.5 or 0.785 and is specified in Hz. The frequency can be specified up to a precision of around 4-5 decimal places as this is then converted to a period in cycles. The period can be specified up to a precision of 6.25E-8 seconds. The width is sent in terms of the number of cycles of a clock at 16MHz; therefore, no fractional values will affect the final result (they will be truncated). The computer will keep sending data sets until it reaches a limit and fails. Usually the first three sets will succeed and be sent to the Arduino, but sometimes more will fail, and this would indicate a faulty USB cable, or possibly noise or some other interference or driver issue. Three sets are required so that the Arduino can make a comparison to ensure they are all equal before continuing and initiating the strobe sequence.

Arduino Side Flow Chart



Arduino Side Code Notes

The Arduino program starts with a loop waiting for initial contact. Once contact is made, it expects data sets to be sent to it in the form “ssXXXXccYYYYff,” where XXXX and YYYY are frequency and width respectively. It expects groups of three data sets that are then compared. If they match exactly, the data set passes and they can move on to preparation for strobing. If the data sets fail, it simply waits for more data sets until a pass is achieved.

Upon passing a set of data, the Arduino calculates several loop values. These are 8-bit loop values that will be used in the assembly language code block inserted into the C++ code in the final loop entered. This loop uses a few sets of assembly loops to delay the amount of time needed until the exact number of cycles is met.

A special section of code in the assembly was added to allow a partial operation to be added for further accuracy. This was achieved by undertaking an operation only every certain number of cycles, or by doing one less operation every certain number of cycles. This was an attempt to get close to the actual frame rate of the cameras, but it was later found out that tuning to exactly the right frequency would be impossible due to heat and frequency drift on the crystal oscillators. The current code contains this loop, and it may be removed if desired (although it is small enough to be considered harmless). To remove this loop, you would need to modify some of the cycle offsets in the calculation functions.

Arduino Details

The Arduino used for this application is an Arduino Mega using the ATmega1280 and running at 16MHz. The compiler must be set to this processor type or the upload will not be successful. The Arduino compiler on which this application was developed was version 0015.

Circuitry Details

On top of the Arduino is a circuit board that was salvaged from another project. Pin headers were soldered onto it to enable it to be pressed into the Arduino (so as to essentially sit on top). This allows the Arduino to be reused easily by simply unplugging the top layer. The circuit board on top also has a differential bus and a connection to the DMX XLR port. The circuit used is almost identical to a DMX shield that can be seen here. The only difference is the presence of many pin headers. If the connections are followed past these pin headers, it is identical with the exception of my connection of the output to digital 4.

Parts

We used the SWITCHCRAFT E3FSC, a high-quality DMX XLR female jack, and the TEXAS INSTRUMENTS SN65176BP, the RS485-compatible differential bus. This differential bus converts a logic signal from a microcontroller (0–5 volts for logic 0 and 1) to a differential voltage signal; logic 1 is where one pin is +V volts and the other is –V, and logic 0 is where the same first pin is –V and the next is +V volts. In our case, V is half of our old logic 1 signal: 2.5 volts.

Com Port Details

Some things I have noticed about the COM port are as follows.

> Usually, several messages are lost immediately after resetting of the Arduino. This is dealt with by sending approximately 20 commands to the Arduino and expecting three or four of them to fail before any are received. If I detect that ten or more have failed before a response is received, I consider the Arduino unresponsive and toggle the DTR line to reset it manually.

Camera Details

The cameras are running on some type of processor that most likely uses a crystal oscillator. Because of this, the timing it generates is subject to a slight drift. This drift can cause the images to see a strobe light in slightly different positions, and can also cause the boundary or beginning of the strobe to drift upward or downward. This drift is due to heat, and therefore perfect synchronization cannot be achieved between the cameras without using feedback. Because of this, I have tuned my strobe to be close to the frame rate of two cameras; this should ensure that it is close enough to the other camera's average frame rate to render the issue inconsequential.

Frequency Modification

Due to the above, a correction factor is divided to the frequency before the strobing cycles are calculated. This will ensure that any frequency specified will be very accurate. If a new Arduino is used, a new constant needs to be found. This constant can be found through trial and error by simply looking at the drift of the strobe in the image through a camera and modifying the factor (perfectly tuned would have no drift). Tuning should be done for a frequency of 29.97003Hz as that is the frequency used most often (also, it is impossible to tune at any other frequency other than the cameras frame rate).

Troubleshooting

If a problem arises that deals with communication there are several lines in the software that may be tweaked to try and fix it.

In strobeSerial.h:

Lines 36 and 37 – deals with serial port reads and if a lot of time outs occur then increasing these values may help fix that problem.

Line 35 if a lot of data sets fail when attempting to set the strobe frequency and width, increase MAX_TRIES or find a new USB cable.

In StrobeController.pde

The setLoopVariables() function contains basically all of the delays used for the strobing. If the strobe on time or off time is wrong this is the place where you could fix it. Do not attempt to make changes if you do not fully understand what this function is doing. This will not affect connectivity problems between the Arduino and the computer.