MooresCloud Holiday
ATmega328 Firmware Specification
DRAFT 22nd July 2013

Commands are accepted on the serial interface, out-of-band from the SPI data flow.

Configuration is 57600, 8N1.

To keep the code and processing overhead low, they use a postfix notation where the arguments precede the command.

Arguments are separated by commas. Commands do not need to be followed by a CR, LF, etc. CR or LF will be ignored, as will any control codes like BS. If echo is on, input is echoed (as typed), followed by CRLF.

Generally if no arguments are provided, the setting will not be changed. Commands will respond with the new or current setting. Responses are followed by CRLF.

Command	Argument 1	Argument 2	Description
?			Returns version e.g. “HolidayDuino01”
E	0 or 1		Disable or Enable echo – default 0E
L	0 to 300		Set length of LED string – default 50L
M	0 or 1		Scaling mode 0=none, 1=stretch – default 0M
P	0 to N		Play predefined pattern 0=stop – default 0P
W	0 to 65535		Set watchdog timer in seconds 0=none – default 0W
D	0 to N		Set default startup pattern 0=none – default 0D
A	0 to 3		Read analog input A0 to A3 on expansion header
B			Read button state as 6 bit value – bits 0-2 are current state, bits 4-6 are state captured at boot
[			Lock I2C bus – Arduino becomes I2C slave, allowing Linux to access I2C EEPROM
]			Release I2C bus – Arduino allowed as master

Setting LED length to 0 will disable the SPI interface & handshaking on the ATmega, potentially allowing connection of WS2801, LPD8806, or similar LED strips to the ATmega ICSP/SPI header (with appropriate changes on Linux side of SPI interface).

Watchdog timer is kicked by any SPI, I2C, or serial data (command or Linux console).

Might want to add checksum setting for SPI packet verification.

Still need to define UX of buttons when ATmega is in control - immediately after boot and/or after watchdog timeout – possibly different – especially if we want a hold down button during boot setup mode feature.

We might want an initial boot (console activity) time limit – e.g. if Linux console is in reboot loop, or if uSD card faulty/not present (imx outputs hex error code every X seconds).

SPI Handshaking – Linux to Arduino:

- Linux checks ACK/BUSY from Arduino (GPIO 23 is high) – if not, abort!
- Linux asserts SS (GPIO 19 made low)
- Linux waits for ACK/BUSY from Arduino (GPIO 23 goes low)
- Send bytes using MOSI & pulsing SCK high – 3 x #LED bytes in GRB order (to match WS2812 pixels), MSB first. Bitbanging is approx 3MHz, so we include an approx 18us inter byte delay to allow Arduino interrupt routine to process byte. If we can get hardware SPI working, we could try dropping to 500kHz, and avoid spinning CPU.
- Linux de-asserts SS (GPIO 19 made high)
- Arduino de-asserts ACK/BUSY (GPIO 23 goes high) immediately if incomplete frame received, otherwise only after frame delivered to LED pixels

Test firmware:

- Verify serial comms to imx
- Verify I2C EEPROM access
- Verify A2D test voltages
- Drive output LED
- Controlled by imx or need separate host connection?