Wayne Johnson
CSCI5991-001
ID #4348421
joh09024@umn.edu
Independent Study
6/20/2017

Report on Project mLauch Version 3.0

Introduction

Did you ever play with model rockets as a kid? It's a fun hobby that lead to many different areas of Science, Technology, Engineering, and Math (STEM). Building and flying a rocket can involve kinematics (the branch of physics involving motion), aerodynamics, chemistry (as in rocket fuel), even electronics and software engineering as well as many manufacturing technologies. This project is based in the areas of electronics and software development.

A typical model rocket is shown in Figure 1. The rocket has a motor that uses black powder (AKA gun powder) or on larger rockets, Ammonium Perchlorate Composite Propellent (APCP), the same rocket propellent the Space Shuttle used. It also uses a parachute or streamer to help recover the rocket and fins to keep it stable during flight. When the rocket flies to it's highest altitude, a small ejection charge in the end of the motor pops off the nose cone and deploys the parachute.

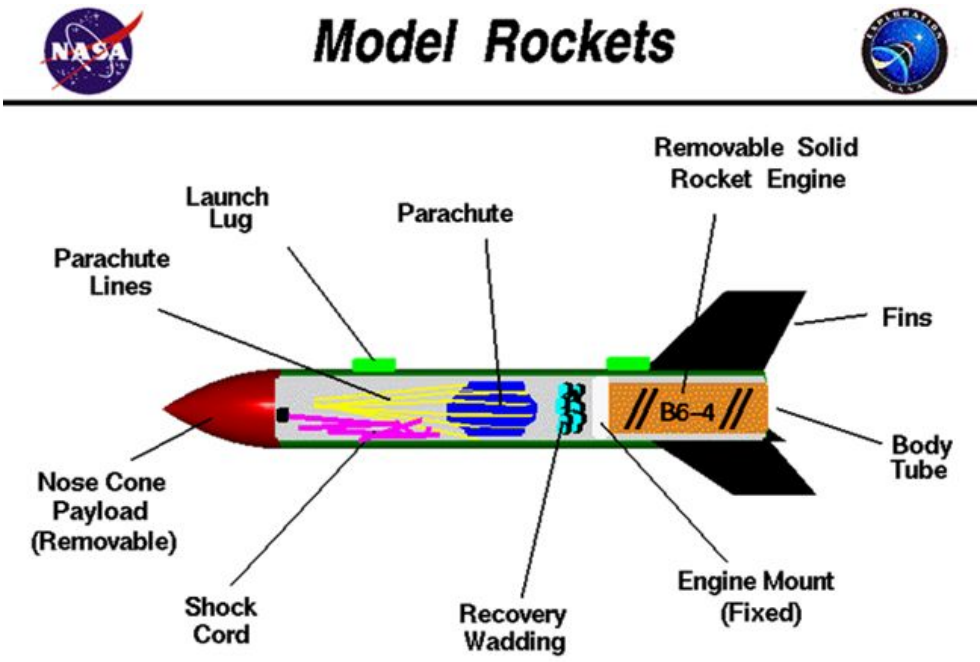


Figure 1 - Typical Model Rocket

Model rocketry has also grown up with the times. In years past, flying a “Big Bertha” on a class “C” engine was just about the peak of power in the sport. Nowadays, model rockets have developed into high power rocketry. High power rockets as large at 29 feet tall being powered by class “M” class engine have become quite common to the adult high power rocket hobby. To put things in perspective, each letter in the engine classification system is twice as large as the previous. This can lead to an “M” engine that produces as much as 2500 pounds of thrust, and weighing in (just the motor) at 5302.00 grams (12 pounds for the metrically challenged).

As an example of one such rocket launch: Locomotive Breath (a reference to a Jethro Tull song). This rocket was built out of a 16 gauge aluminum pipe and 18 gauge aluminum fins, a single use “P” rocket engine, a pair of parachutes, and a significant amount of on-board electronics such as altimeters and radio tracking equipment. The preflight modeling indicated that it would reach 17,000 feet, while reaching mach 2.3. Figure 2 is picture at launch, the rocket in flight, and what remained after a mid flight failure (as around Mach 1). A post-mortem indicated that one of the fins sheared off in mid flight.

Figure 2 - Locomotive Breath

As you can see, high power rocketry can be hazardous. It is the goal of Tripoli Rocketry Association, as well as the National Association of Rocketry (NAR) to keep the sport safe, as well as fun. Both organizations represent over 10,000 model and high power rocketry enthusiasts around the globe.

You can not simply walk up to a rocket and flick your BIC under the engine. Rocket engines are ignited electronically, by running current through a thin nichrome wire installed in the engine core. The heated wire ignites a small amount of pyrogen, which in turn ignites the rocket fuel. Figure 3 depicts a simple launch system.

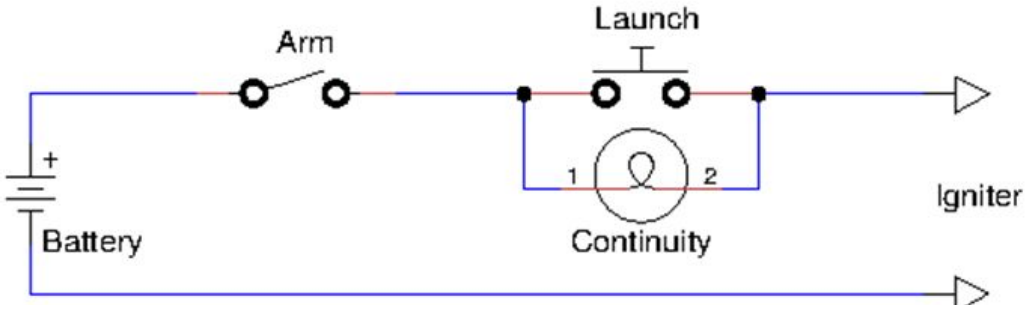


Figure 3 A Simple Launch System

Because of these dramatic increases in scale in high power rocketry, new safety procedures have been developed. It's quite common for an class M motor based rocket to be launched 1/4 mile (or more) away from the nearest human or structure. Many different schemes have been used to perform this. Everything from running a 1/4 mile long wire out to the rocket, to using radio control airplane parts. One person used a garage door remote control as a means of

closing a relay at the launch pad, to light their rocket. There are been many other schemes, some quite a bit more problematic and unsafe as well. As you can see, there is room for improvement. The mLaunch project was launched to create a safe, dependable, and low cost solution to this problem.

History of mLaunch

The first version of mLaunch (1.0) was developed in 2004 to augment and eventually replace the launch control system used by Tripoli Minnesota, a local high power rocketry club. For the previous 15 years, Tripoli Minnesota had been using a wired relay based launch controller that included a control panel with toggle switches, a long and heavy wire cable, and remote relay boxes for pads located farther away. A high level overview of this system can be seen in Figure 4.

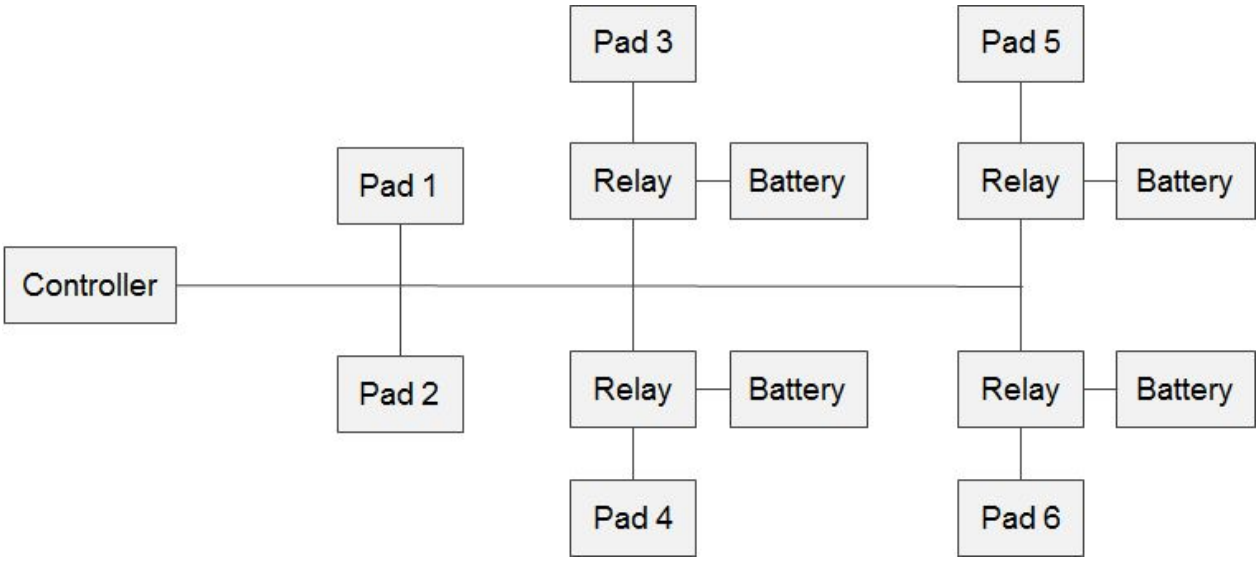


Figure 4 - Tripoli Wired Launch System

The wired launch system presented several problems. Besides being big and cumbersome to set up, having a 40 lb spool of wire, it also had many points of failure, batteries being low, connections being bad, relay contacts getting worn. It also had limited ways of diagnosing an issue. Many times club members would be seen probing around the pads with a volt meter trying to identify just why their rocket wouldn't go.

mLaunch 1

At this point, two new technologies were making it to market. Cheap and easy to program single chip microcomputers, and a new Maxstream XBee wireless module based on the Zigbee mesh network protocol. These two technologies were put together to create a wireless launch system. At the pad end, a XBee transceiver (AKA Modem), an Atmel ATMega8

System-on-a-Chip, push-button switches and LEDs to check for igniter continuity, and a pair of relays to actually launch the rocket.

At the Launch Controller end, is another XBee transceiver and ATMega8 as well as push buttons, and LEDs to show status of the pads. See Figure 5.

Figure 5 - Version 1 pad box.

As a first attempt, this system worked quite well and remained in production for over 7 years. It's still around and servers as a backup system. Unfortunately the system was built using home made printed circuit boards (many of which had to be patched due to etching flaws, as well as a "rats nest" of wired interconnections between PC board and relays, LEDs, push-buttons, and banana jacks (for power and pad connections). With the desire to make the system easier to manufacture, version 2 was planned.

mLaunch 2

mLaunch2 brought many new production class enhancements. It was a first attempt at surface mount technology, used professionally made PCBs, Digi (which bought out MaxStream) XBee Series 2 modules, and a Atmel Dragon JTAG & IDE programmer/debugger. Many factors combined to doom this version, the XBee Series 2 modules were buggy, the new SMT technology had a large learning curve, and the ATMega8 chip was cramped for RAM space leading to stack corruption and some very frustrating and confusing software failures. The project got shelved.

mLaunch 3

Knowing the end-of-life was approaching for the version 1 system, a fresh start was made on a new version 3 in 2012. This new design included more SMT parts, a bigger Atmel Mega32 Series CPU, A new PCB Supplier (the earlier one went out of business), a new Atmel ICE/JTAG3 professional class programmer & IDE.

This version was completed, and programmed, and put into production in the spring of 2014.

mLaunch 3.0 Requirements

Some of the requirements for mLaunch 3.0 were (and still are):

- Easy to make, single board construction
- Minimal circuit connections, no rats nests or wiring harness
- More reliable components
- Dual automotive grade relays which are field replaceable
- 20 amp automotive class circuit breaker instead of fuses to protect from igniter (and other) short circuits.
- Reverse voltage protection & static “surge” protection
- More detailed remote statistics such as actual ignitor resistance and pad end battery voltage via a 8x20 LCD to display
- Dual circuit per pad box (to reduce costs)
- 8 Pad support
- CPU powers up in inert state preventing accidental relay closures when started.
- Ultra bright tri-color (red/green/yellow) LEDs, 180-450 lumen, visible in direct sunlight (most of the time).

ATMega32A CPU Specification Highlights

- Up to 16MIPS Throughput at 16MHz
- 32Kbytes of In-System Self-programmable Flash Program Memory
- 1024Bytes EEPROM
- 2Kbytes Internal SRAM
- Two 8-bit Timer/Counters, One 16-bit Timer/Counter
- 32 Programmable I/O Lines
- 8-channel, 10-bit ADC
- Programmable Serial USART
- $5.88 MSRP

XBee-Pro™ ZB S2B Specification Highlights

- 2.4 GHz Transceiver (Transmitter/Receiver)
- Extended-range ZigBee, Through-hole, Mesh Networking
- 2 miles / 3.2 km Range (YMMV, outdoors, with a parabolic antenna, and the wind at your back). Expect 1/4 mile.
- $28.00 MSRP
- 250 Kbps Data Rate

Mesh Networking

Unlike the MaxStream XBee Series 1, the Digi XBee Series 2 supports true mesh networking. This means that if a node in the network receives a message for some other node, it will pass that message on to it's nearest neighbor that is nearer to the final destination. Using a mesh network in mLaunch allows us to place launch pads out of the direct range of the launch controller, but as long as there are one or more nodes (i.e. pads) connected between the controller and the remote pads, command and status messages can still be sent. This is represented in Figure 6

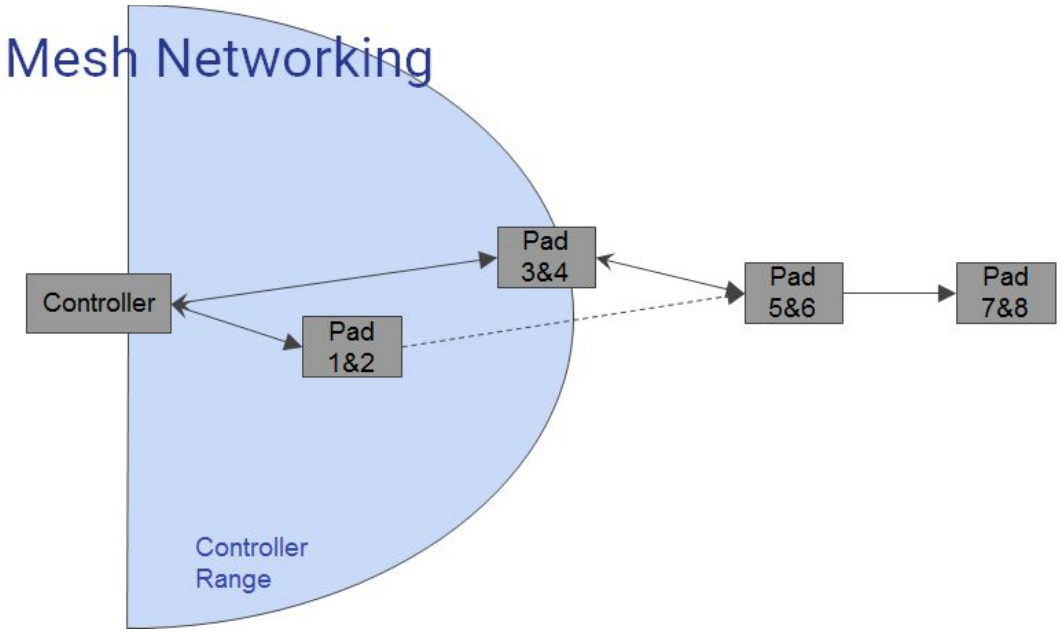
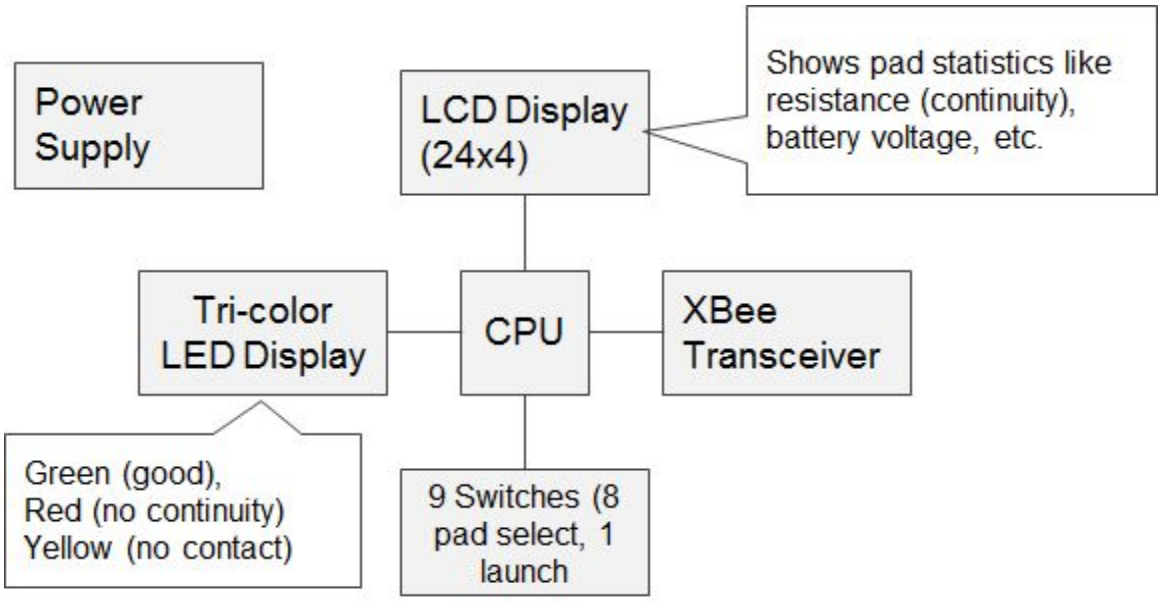


Figure 6 - mLaunch Mesh network

Block Diagrams

The mLaunch system consists of two types of modules. One Launch Controller box, and up to 4 pad boxes.

Launch Controller



Specific schematics for the Launch Controller are available at https://github.com/wdtj/mlaunch/blob/master/controller30/controller30r3.pdf

Power Supply

The mLaunch Launch Controller power supply is pretty standard. It accepts 6-18 volts input and provides both regulated 3.3 and regulated 5 volt outputs. One interesting note is that the input to the power supply contains a bridge diode. A bridge diode is normally used to rectify AC inputs into DC. Using in here allows the Launch Controller to be mis-connected to a battery and it will still work. Also included in the input circuit is an 18v varistor to protect the circuit from static discharge along the power input path.

CPU

The mLaunch launch controller CPU is an Atmel ATMega32A microprocessor on a chip. The CPU runs off the 5 volt power supply and uses a 16Mhz crystal for a CPU clock. Although the ATMega32A has a built in capacitive clock circuit, it is not accurate enough to control the serial communications connection with the XBee module at 9600 baud. In addition, the ATMega32 is connected to a 10 pin 1mm pitch header. This is used for programming the CPU as well as debugging. Details on the ATMega CPU architecture and programming are available in the ATmel supplied ATMega32 documentation at https://github.com/wdtj/mlaunch/blob/master/atmega32A.pdf.

XBee Transceiver

The XBee transceiver is the heart of the system communications. These modules are pre-programmed with either XBee coordinator or router software. There can be only one coordinator in the XBee network, and this device coordinates the mesh network. The XBee uses a 3.3 volt power supply. In order to translate between the CPU's 5 volt signals and the XBee 3.3 volt signals a simple voltage shifter is used. By default the XBee operate at 9600. The module is also preprogrammed to run in API command mode (vs AT command mode) to make it easier to communicate with the CPU. Details on the XBee communication commands and other details are available in the Digi supplied XBee user's guide at https://github.com/wdtj/mlaunch/blob/master/XBeePro2.pdf. Connection to the XBee transceiver is via the CPU's USART on the RXD and TXD pins.

LCD Display

The LCD display is a LCM-S02004DSR from Lumex. Pinout and electrical specs are here:

https://github.com/wdtj/mlaunch/blob/master/LCM-S02004DSR.pdf
It uses an industry standard LCD controller HD44780U. You can find this spec at https://github.com/wdtj/mlaunch/blob/master/HD44780.pdf The LCD requires 5 volt power and 5 volt data signals, but since the CPU is already running at 5 volts this is not an issue. The LCD uses 8 parallel lines to send data/address information to/from the LCD display, along with 3 control lines. The CPU port PA 8 bit port provides the basic parallel data, and the CPU's PD2, PD3, and PD4 lines provide control signals. Basically a command or data byte is written on the PA port and the E pin is strobed to send data to the LCD. Consult the HD44780 document for more specific details.

Switches

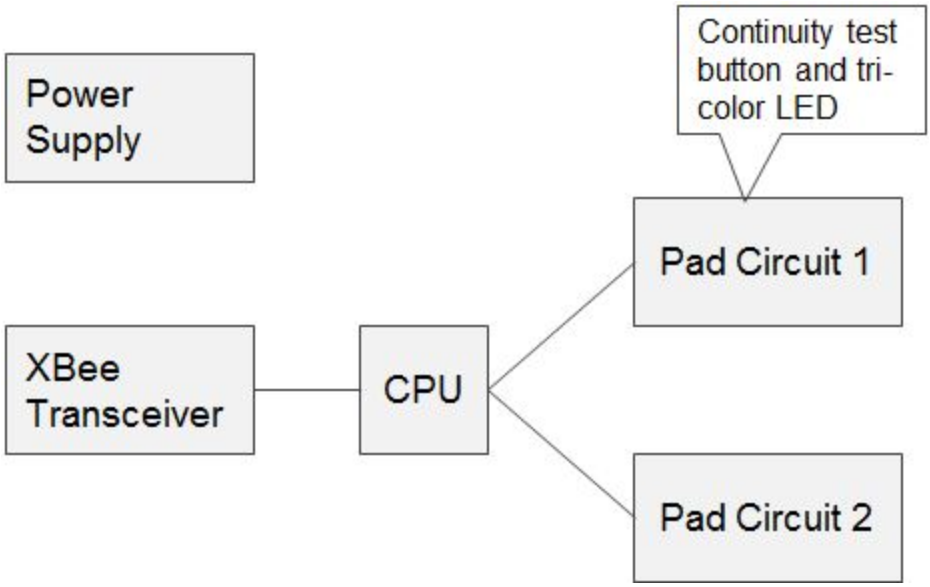
9 pushbutton switches are connected directly to input pins of the CPU. These are pulled high (a logic 1 state) by internal (to the CPU) resistors, and pressing one of the buttons causes the input to be grounded (pulled low) to present a 0. Software needs to "debounce" these switches because the mechanical contacts in the switch can vibrate on contact producing an intermittent signal for a few milliseconds. The software, when detecting a switch that is opened or closed

will need to check again a few milliseconds later to verify it's still in that state. The pad enable switches use the CPU's PB port (8 pins) and the PC7 pin for the launch button.

LEDS

The LEDs have 3 different LED circuits in one package. Red, green, and yellow. This amounts to 24 different circuits required to drive all LEDs, more than are available on the CPU. To get around this, there are 3 TI TLC5917 LED drivers in the circuit. This driver is basically 3 8 bit shift registers, wired in series. To turn on or off an LED, the appropriate bit is output into the input (SDI) of the first driver register and the driver clock signal (CLK) is strobed. Then the next bit, etc. until all 24 bits have been written. All the red LEDs are written in the first byte, all yellow on the second byte, and all green in the third byte of the 24 bits. Specifications for the TLC5917 driver can be found at https://github.com/wdtj/mlaunch/blob/master/tlc5917.pdf. The LED driver is connected to the CPU's PD5 (CLK), PD6 (LE), PD7 (SDI), and PC0 (SDO) pins.

MLaunch Pad



Specific schematics for the Pad is available at
https://github.com/wdtj/mlaunch/blob/master/pad30/pad30r4.pdf

Each pad modules is capable of managing 2 pads and thus has 4 relays (2 per pad), 2 continuity test buttons, and 2 tri-color LEDs as well as circuitry to measure the voltages across the ignitor and a circuit to measure battery voltage.

Power Supply

The mLaunch Pad power supply is also pretty standard. It accepts 6-18 volts input and provides regulated 3.3 volt output. Again the power supply contains a bridge diode to allow the battery inputs to be mis-connected and also it has an 18v varistor to protect the circuit from

static discharge along the power input path. One interesting problem in development, the diode bridge can allow the voltage sensing circuit for the pads ignition to be either above or below the ground voltage of the CPU and it's integrated ADC. This issue is handled by a set of resistors in that circuit that translate the voltages from the possible +18 to -18 volts to the ADC's required 0-2.55 volts.

CPU

The mLaunch pad CPU is again an Atmel ATMega32A microprocessor on a chip. The CPU runs off the 3.3 volt power supply and it uses a 16Mhz crystal for a CPU clock. Everything else is the same as for the launch controller. The CPU circuit also contains a resistor bridge that allows the CPUs ADC to measure the battery voltage.

XBee Transceiver

The pad transceivers are preprogrammed as router nodes. Since both the XBee and CPU uses a 3.3 volt power supply no translation is needed. Connection to the XBee transceiver is via the CPU's USART on the RXD and TXD pins.

Pad Circuit

The pad circuit (x2) contains a ULN2003A darlington transistor driver, which provides enough current to the CPU signal to drive the relay coils. These also protect the CPU from high reverse EMF pulses that are generated as the relays turn off. A network of resistors translate the possible +18 volt to -18 volt continuity signal from the pad into a 0-2.55 volt signal for the CPUs ADC.

In addition, each pad circuit contains a pushbutton switch to test continuity and a tri-color LED to display the results. The pad controls use the following pin assignments:

Function	Pad 1 CPU Pin	Pad 2 CPU Pin
Continuity Detect ADC Input	PA0/ADC0	PA1/ADC1
BATT+ ADC Input	PA2/ADC2	PA2/ADC2
Launch Relay	PA3	PA6
Enable Relay	PA4	PA7
Red LED	PB0	PB4
Green LED	PB1	PB5
Yellow LED	PB2	PB6
Continuity Switch	PB3	PB7

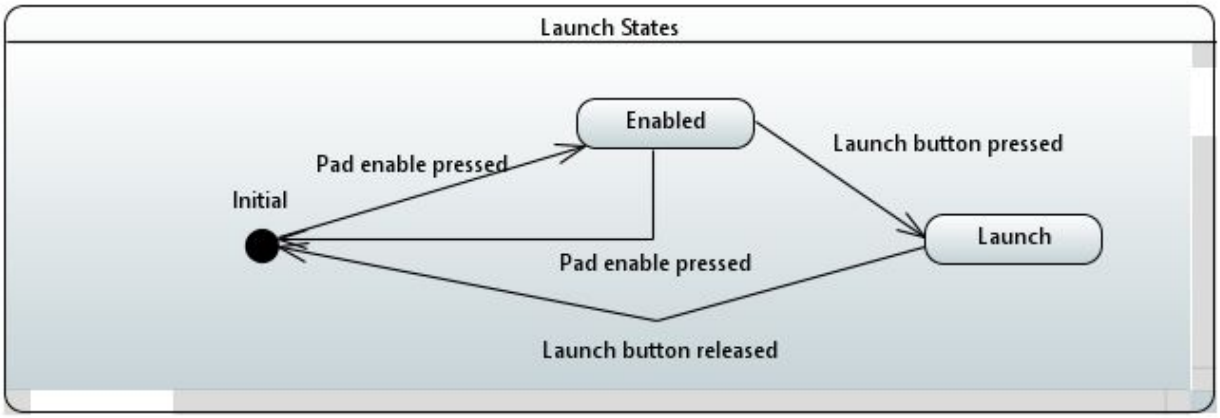
Functional Overview

In order to launch a rocket, first the rocket owner places the rocket on the pad, and connects the pad's ignition leads to the ignitor in the rocket engine. The owner can then press the pad's continuity button which will light up green (good continuity) or red (red continuity). There is also a yellow light that indicates low battery voltage.

When the Launch Control Officer (LCO) gets ready to launch the rocket, they press and release a button on the launch controller corresponding to the appropriate pad. The pad's first relay closes in order to place a small continuity test current through the ignitor. Both the pad and launch controller LEDs light with the continuity status. The LCO then presses the launch button which closed the second relay at the pad. This sends the full battery current through the igniter. When the LCO releases the launch button, the pad is then disabled.
If, before pressing the launch button, the LCO presses the pad enable button a second time, the pad is disabled and returns to it's initial state..

If, before pressing the launch button, the LCO presses the a second (or more) pad enable button those pads are enabled and their respective LEDs light with their statuses. Then by pressing the launch all the enabled pads are launched.

If, when pressing the pad enable button the first time it's held down for more than a second, the pad statistics (battery voltage and continuity resistance) are displayed on the LCD display.



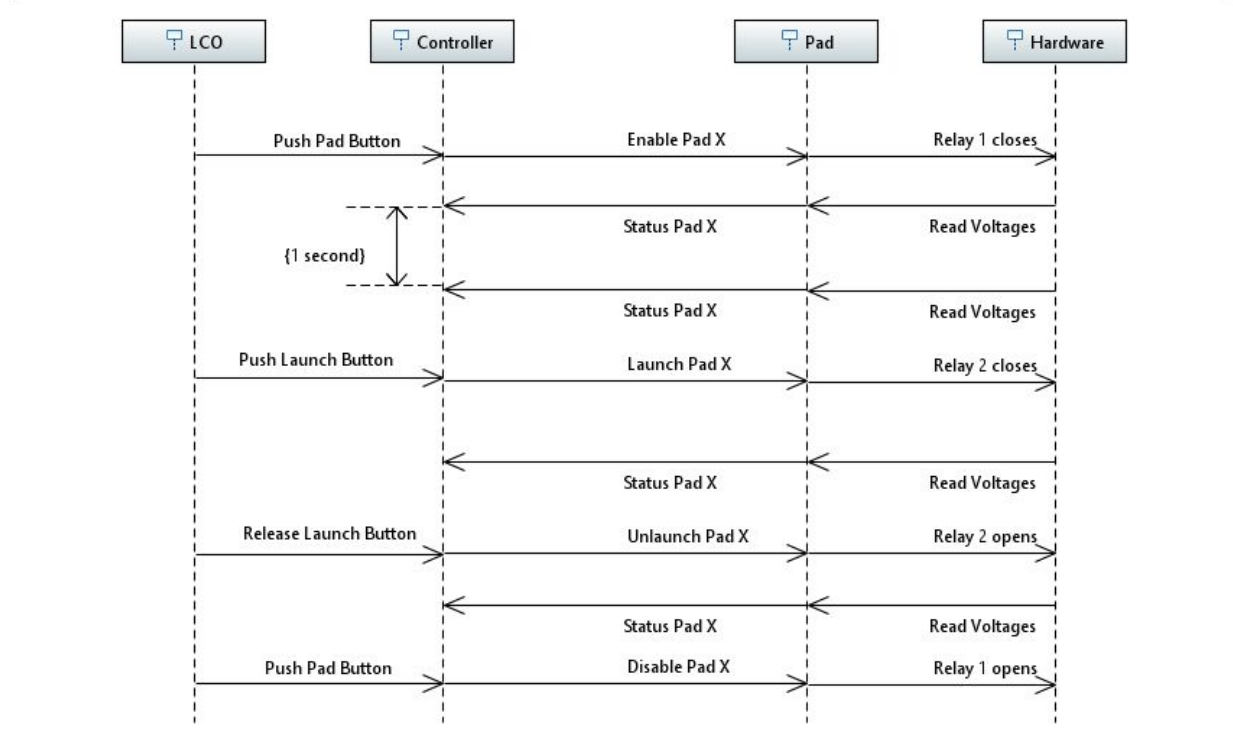
Application Level Message Packets

There are relatively few application level packets exchanged. They are kept short to reduce requirements on bandwidth. They do not have any CRC or other error checking/correcting data as these functions are done in the XBee itself. These packets include:

Enable - Enable the continuity checking relay and causes the pad to begin to send back status packets.	Ex where x is a pad number from 1-8
Launch - Closes the launch relay	Lx where x is a pad number from 1-8
Status - Returned by the pad with current information	Sxnn...n where x is a pad number from 1-8, and nn...n is pad status information
Unlaunch - Opens the launch relay	Ux where x is a pad number from 1-8
Disable - Opens the continuity relay and return to normal state.	Dx where x is a pad number from 1-8
Identify - Caused the specified pad to begin flashing it's green LEDS to identify it for configuration.	I
Assign - Assigns a pad number to that pad. Used for configuration.	Ax where x is a pad number from 1-8

After an Enable command is sent, a watchdog timer is set that will reset the pad if no other request is sent within 10 seconds. This timer is restarted when any packet is received for that pad. To ensure a pad does not reset itself too early, the controller repeats the enable command every second. After a pad receives an Enable message, it begins sending back Status messages every second until the Disable message is received or the watchdog timer resets the pad.

The watchdog timer is used to safe any and all pads in the event the Launch Controller goes away.



Network Level Commands

XBee Commands

The XBee transceivers use many different control command packets. Please consult the XBee User's guide for details. An example the NJ XBee command (from the XBee user's guide) has the format:

	Frame Fields		Offset	Example	Description
API Packet	Start Delimiter		0	0x7E	
	Length		MSB 1	0x00	Number of bytes between the length and the checksum
			LSB 2	0x04	
	Frame-specific Data	Frame Type	3	0x08	
		Frame ID	4	0x52 (R)	Identifies the UART data frame for the host to correlate with a subsequent ACK (acknowledgement). If set to 0, no response is sent.
		AT Command	5	0x4E (N)	Command Name - Two ASCII characters that identify the AT Command.
			6	0x4A (J)	
		Parameter Value (optional)			If present, indicates the requested parameter value to set the given register. If no characters present, register is queried.
	Checksum		7	0x0D	0xFF - the 8 bit sum of bytes from offset 3 to this byte.

Among the XBee commands mLaunch uses are:

NI	Node Identifier. Set/read a string identifier naming the node.
NJ	Node Join Time. Set/Read the time that a Coordinator/Router allows nodes to join.
NR	Network Reset. Reset network layer parameters on all modules within a network.
ND	Node Discover. Discovers and reports all RF modules found.
NJ	Node Join Time. Set/Read the time that a Coordinator/Router allows nodes to join.
JV	Channel Verification.
VR	Firmware Version. Read firmware version of the module.
JN	Join Notification. Set / read the join notification setting. If enabled, the module will transmit a broadcast node identification packet on power up and when joining.

When the system if first brought up, the Launch Controller initializes all it's hardware and sends a command to the XBee module to discover it's network (all the pads). Over the next 5 seconds each powered up pad responds with it's name and a network address. The XBee devices randomly staggers the responses to prevent everyone from answering the discovery broadcast at once. As each pad is identified, the Launch Controller LEDs for that pad set are turned green. When the 5 second wait is done, the controller turns off all the LEDs and is ready for business.

During operation, if any pad joins the network, the Launch Controller will receive a Discovery notification from the XBee device and add that pad to it's operational list.

XBee Data

In addition to commands sent to the XBee transceiver, application data is sent. Data is wrapped in a XBee API data packet. These data packets can then be marked to be sent to a specific node using a XBee Explicit Addressing ZigBee Command Frame, or can be sent to all devices. All communications are via explicit addressing

Software Design

The ATMega system are very minimal. They do not contain an operating system, therefore all software is written on “bare metal” and in machine code. To this end, the majority of the software is asynchronous in nature, operating on a set of Finite State Machines (FSM). When a packet is to be sent to the XBee transceiver, the message is queued up and the USART’s interrupt system then controls when the next character is to be sent. There is also a FSM managing pad controls, switch closure detection (debouncing), and the communication protocol to the XBee.

Modules

Software is broken into the following “projects” or modules:
Utilities
Pad
Controller

Utilities

The Utilities module contains common code and drivers including the following:

ADC Driver	Driver for the ATMega internal Analog to Digital Converter
LED Driver	For use with the TI TLC5917 LED drivers
OLED/LCD Driver	For use with the Lumenx (and other) LCD module
Timer0/Timer1 Driver	Driver for the ATMega internal timers
UART Driver	Driver for the ATMega internal USART
XBee Diver	Driver for the XBee module (connects through the UART Driver)

Pad

Hardware Initialization	Sets ATMega Input/Output configuration, initializes required drivers
Pad module	Main
Launch FSM	FSM to manage launch status, processes Launch Controller commands
Link FSM	Manages XBee initialization and communications
Status FSM	Maintains Pad status and sends status information

Switch FSM	Debounces switches

Controller

Hardware Initialization	Sets ATMega Input/Output configuration, initializes required drivers
Controller module	Main
Link FSM	Manages XBee initialization and communications
Switch FSM	Debounces switches

To do

Additional items for future releases include these features.

- Encryption (feature of the Xbee module)
- Voltage profile at launch. Did ignition cause the battery/igniter voltage to drop?
- Software configuration safety.
- Network routing information and field strengths on LCD.