Compressed-Air Spud Cannon and Performance Assessment Wrap-Up

James A Crawford

Synopsis

Labor Day 2017 found me shooting potatoes into my chronograph rather than cooling my heels at the beach. Design and performance details for the completion of this project follow.

1 Introduction

Part I of this paper[1] provided some background and theory to my efforts with potato cannons. This part II provides all of the final construction details along with measured potato cannon performance.

2 Electronic Design

An overview with rationale for the electronic design was provided in Part I of this project. The final schematics are provided in §7. Although the details differ from the concept described in Figure 10 of Part I, the general concepts remain unchanged. A 10 MHz clock is used rather than 1 MHz, for instance, and more discrete analog circuitry was used in large part because I already had most of the parts available.

The final hardware schematics are provided in §7. The Arduino Uno software is provided in §6. Every I/O, both analog as well as digital, of the Uno is used in this design. This is in large part due to the fairly large number of I/O required by the SN74V8154 counter. I could have stepped up to using the Arduino Mega I have on hand, but did not want to tear into the assembly it is already integrated into.

A picture of the fully assembled main chronograph circuit board is shown in Figure 1. The laser detector diode is mounted on the back-side of the circuit boards. Two such boards are used in the chronograph, but the second board is only populated with power supply related circuitry, laser (transmit) diode bias circuitry, and the laser detector circuitry– no digital parts are populated on the second board.

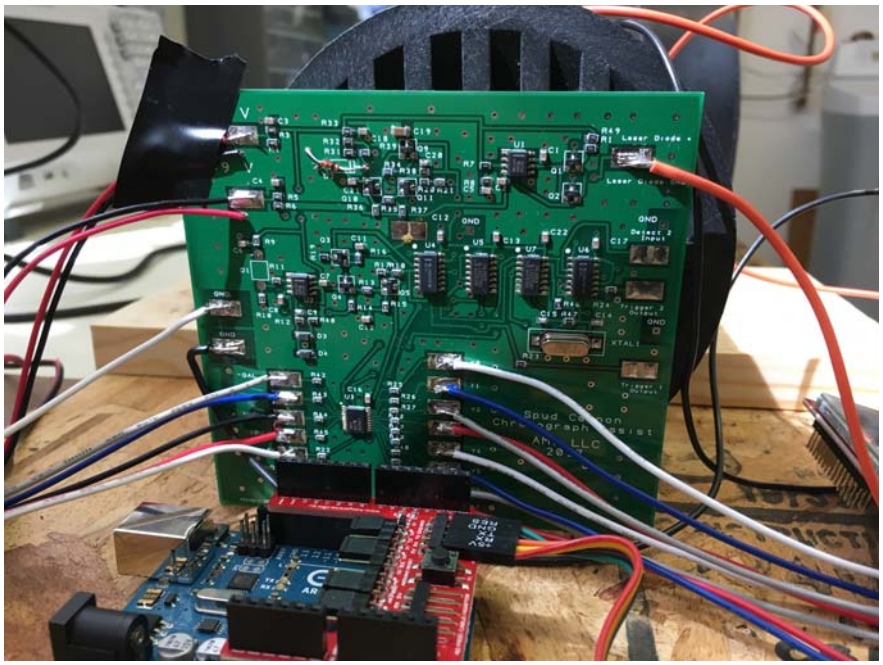

Figure 1 Fully assembled main circuit board with Arduino Uno used for control below. The SN74V8154 entails 12 different I/O signals which took up a major portion of the I/O pins available on the Arduino Uno.

[1] U24146 Potato Cannon and Chronograph, V1.0.

3 Mechanical Details

My chronometer necessarily involves some optics and the need to secure them. I opted to use materials already on hand. Use of the Fresnel lenses as described in Part I made it possible to largely avoid any precision focusing issues.

The optical assemblies are made of ABS drainage pipe parts, secured on a small wooden bed. The master and slave units are shown together in Figure 2. The only electrical connections from the 2nd Time Gate assembly back to the main chronograph circuit board are ground and the detection signal.

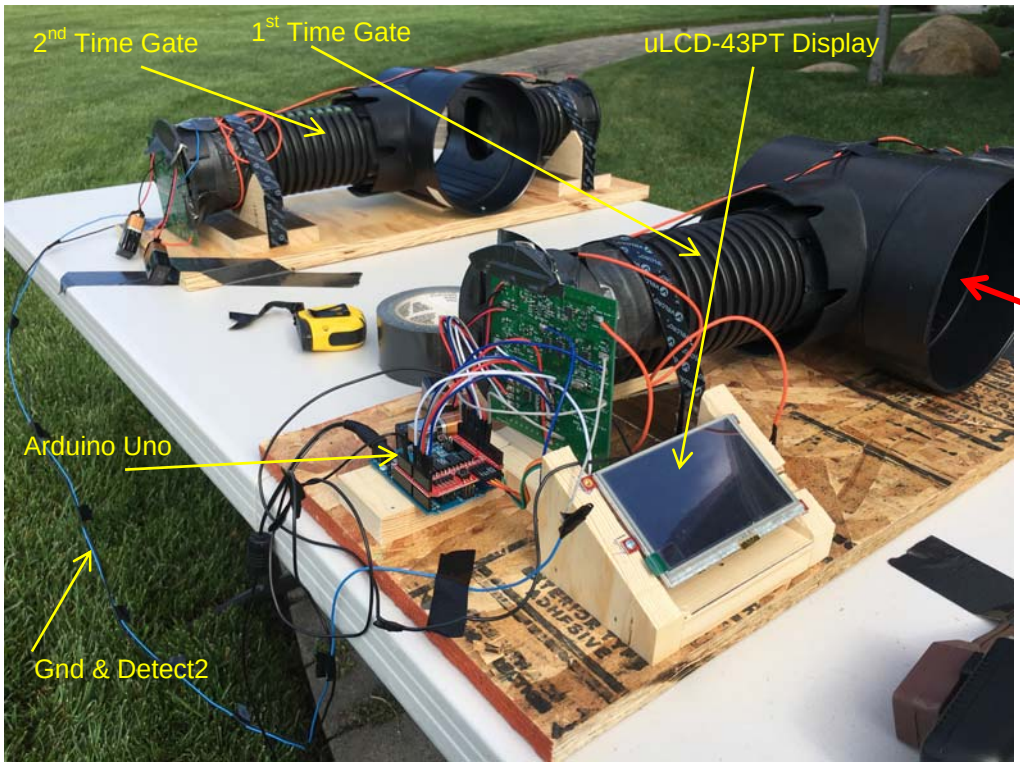

Figure 2 Chronograph with master and slave time-gates. A shot potato enters through the right-side time-gate as shown, triggering the first time-gate and proceeds through the second time-gate.

Each of the four Fresnel lenses are attached to the inner-ends of the ABS piping using duct-tape as shown in Figure 3. Since the light-emitting transmit diodes both emit a diverging cross pattern, a slit has been purposely created with the duct tape in order to effectively mask out the horizontal laser light pattern thereby only leaving the vertical pattern. Once the pipe section has been connected into the larger ABS receptacle, each lens is very secure. Note that all of the Fresnel lenses are recessed back away from the main potato fly-path thereby helping to minimize the amount of *potato juice* that ends up on the surface of the lens.

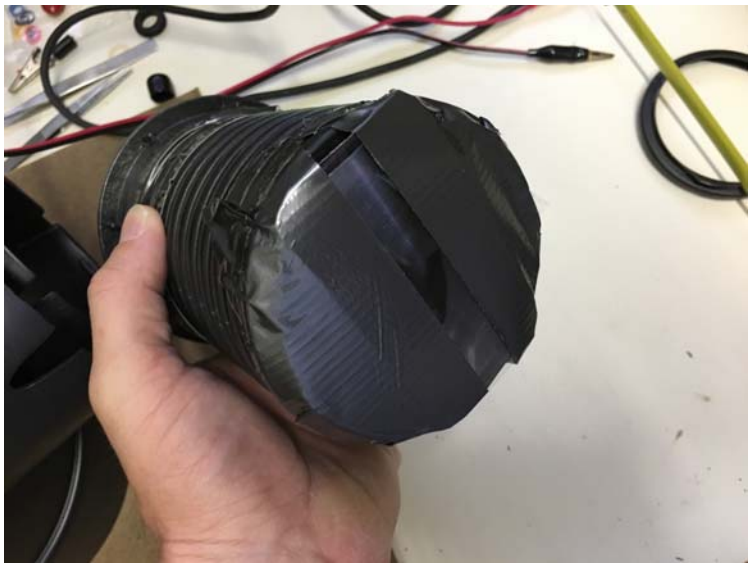

Figure 3 Attachment of plastic Fresnel lens to the inside-end of one of the ABS pipe sections. Clean-up is made far easier if the smooth-side of the Fresnel lens is facing outward rather than inward toward the potato flight-path.

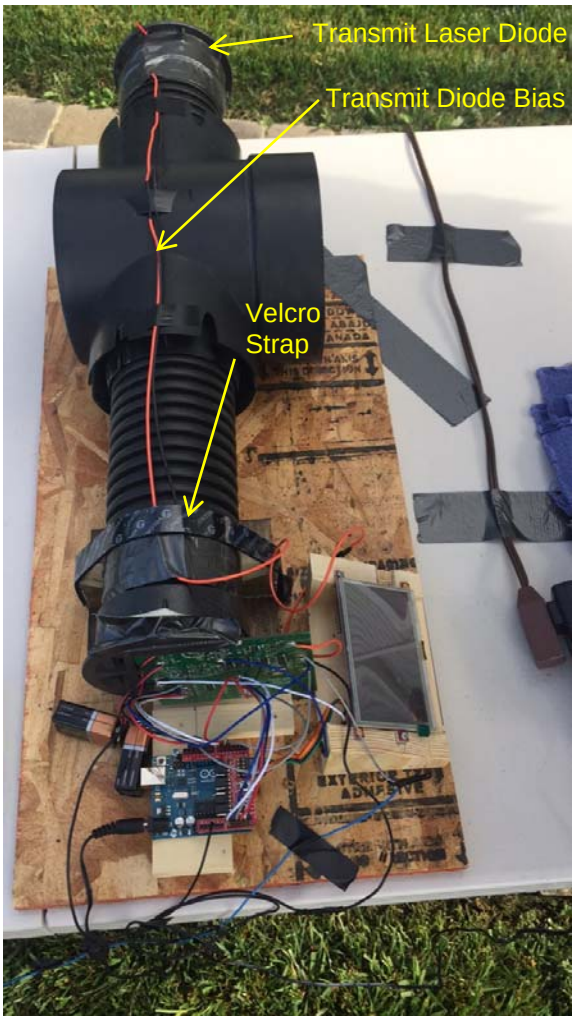

Figure 4 Top-view of the master

4 The Results

4.1 Indoor Preliminary Testing

A reasonable amount of testing was done indoors using the non-potato projectiles shown in Figure 5. The indoor measurements turned out to be nicely reliable and consistent. Outdoor testing turned out to be a bit more challenging, however.

Figure 5 Make-shift projectiles for initial testing. The PVC pipe section was later used to shape the potato projectiles. The projectile on the far right consists of a small rectangular piece of wood wrapped first with aluminum foil and then at the bottom with duct tape.

4.2 Testing Outdoors

I set up a shooting range across my front yard as shown in Figure 6. In the firing direction, the nearest neighbor is over a couple of small hills and roughly 2500+ yards away. I used a short section of 2" PVC pipe (same diameter as used for the barrel of the potato cannon) to cut down each potato for proper diameter as shown in Figure 7 and Figure 8. Each cannon shot entailed the details described in Figure 9.

Figure 6 Outdoor shooting range set up with good visibility. Projectile ranges can reach out to 1,000 yards so firearms safety should be the rule of the day.

Figure 7 Cutting down potatoes is a messy job

Figure 8 Ready to fly spuds

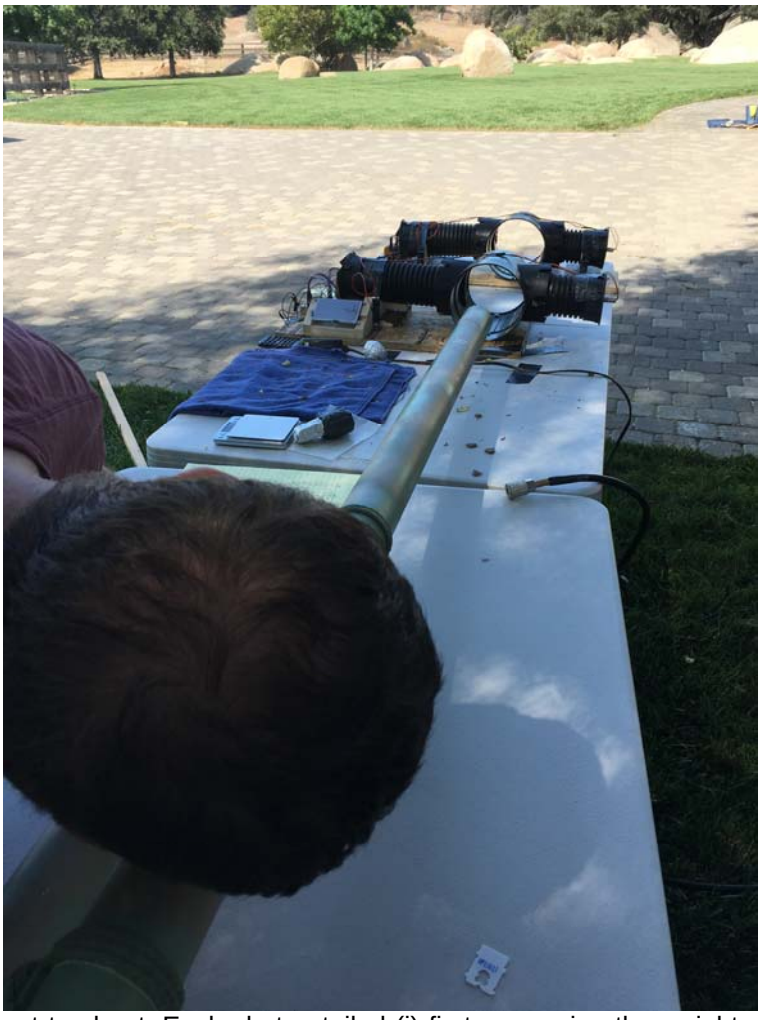

Figure 9 About to shoot. Each shot entailed (i) first measuring the weight of the potato, (ii) muzzle-loading one of the cut-down potato projectiles and ramming it down the barrel, (iii) filling the cannon's compressed air chamber from my air compressor, (iv) setting the electronic's *arming control*, and finally (v) carefully aiming the cannon through the chronograph assembly and firing. This location is different than the starting location shown in Figure 6 because of problems with too much ambient light.

5 Data Results & Conclusions

All of the data results presented here were taken using 100 PSI or 120 PSI cannon pressure. I had originally planned to take measurements at several pressure values, but several difficulties led to somewhat abbreviated data-gathering sessions.

Scatter diagrams of muzzle velocity and muzzle momentum are shown in Figure 10 and Figure 11 respectively.

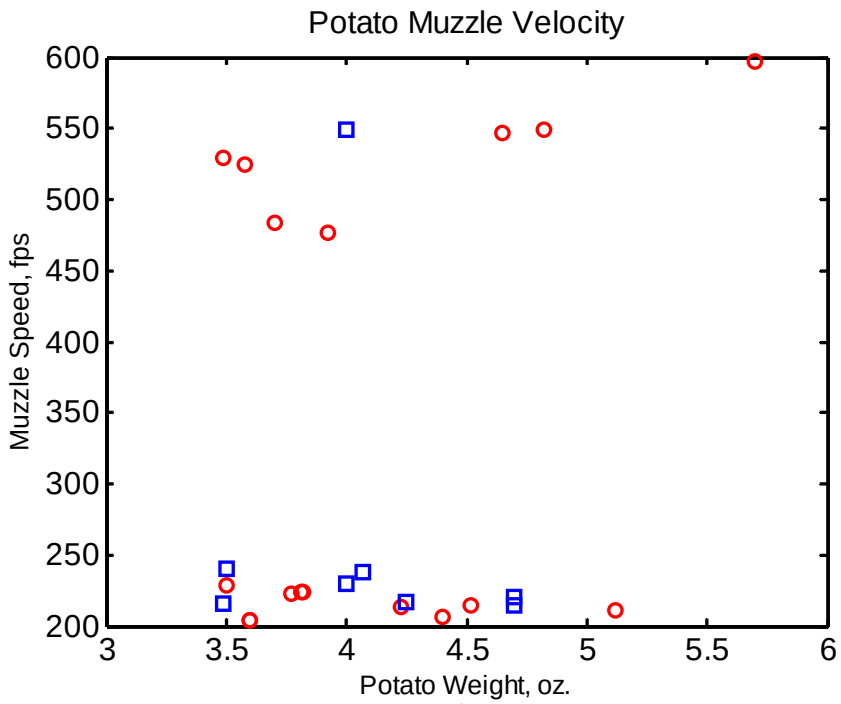

Figure 10 Measured potato muzzle velocity[2] versus potato weight. Round red markers are from the first data collection effort whereas the square blue markers are from the second.

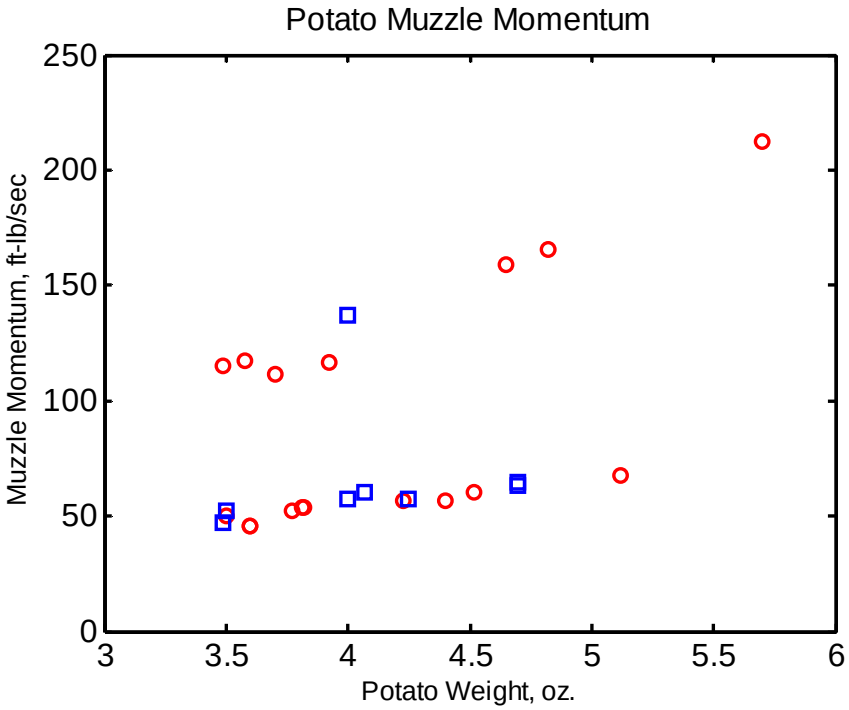

Figure 11 Potato muzzle momentum versus potato weight. Round red markers are from the first data collection effort whereas the square blue markers are from the second.

[2] From u24755_spud_data_analysis.m.

5.1 Observations

- There appear to be two major performance groupings, one group averaging about 225 fps, and the second group averaging in the 500 fps range.
    - Based upon the theory discussed in Part I, it seems rather dubious that potato velocities are reaching 500 fps.
    - A much more likely scenario is that the instrumentation is still getting fooled by potato junk being thrown out of the barrel at very high velocity versus capturing the potato in flight. Since the second time-gate is not enabled until after the first time-gate has been triggered, if the first time-gate is properly triggered by the potato, the potato "junk" has already whizzed past the 2nd time-gate and proper triggering probably also occurs at the 2nd time-gate as well. Otherwise, there could likely be 4 or more data-groupings in the observed data.
- Maximum muzzle velocity observed was an impressive 600 fps, but this is now believed to be the potato junk being shot out of the barrel rather than a potato.
- Rather difficult to correlate between potato attributes and resultant muzzle velocity
    - Too much variability from potato to potato
    - Just because a potato is cut one way or another does not automatically mean that it gets situated in the barrel the same way each time.

5.2 Lessons Learned

- Although indoor testing was very consistent, outdoor testing was more difficult.
    - Indoor testing did not use any potatoes
    - Bright ambient sunlight turned out to be a major difficulty outdoors, but was not immediately recognized. 2nd dataset was taken after sundown.
- Should have gone to the Arduino Mega with more I/O in order to observe all of the diagnostics available from the chronograph-assist circuit board.
- Ambient light (bright sun light) proved to be difficult to deal with.
    - Could probably use some amount of wavelength-based filtering thereby only dealing with energy near the laser diode's emission wavelength but that could have ruled out visible-light lasers
- Need more stability in the optical assemblies
    - Outdoor testing put more strain on the mechanical elements, but ample diagnostics were lacking to make sure that things were operating as needed.
- Even though Fresnel lenses were recessed from the main potato flight path, potato moisture affected the performance of the lenses. Needed diagnostics to make this assessment easier.
- Optical detection thresholds should probably have been made programmable.
- More preliminary testing with non-potato objects would have been helpful, transitioning to real potatoes before going outside. Didn't want potato gunk all over my garage though.

5.3 Conclusions

- Probably much more insightful to use repeatable projectiles rather than potatoes, at least if a real root understanding of the main theory elements is desired.
- Could serve as a very good science project for additional investigation:
    - Aerodynamics of the projectile
        - Best projectile shape versus air pressure used and barrel length
    - Investigation into linear and nonlinear air flow through the air-trigger valve
    - Tradeoffs between muzzle/projectile friction for tighter fit, versus energy losses due to the same friction
- When it is all said and done, high speed camera captures would go a long was in mitigating questions and ambiguities. I would recommend going that route over enhancing the chronograph further since potato velocities are rather benign compared to say a bullet.

Purposely left blank.

6 Appendix: Arduino Software

```
//
//  Use 4-D systems front panel plus an Arduino Uno board
//  to arm and query Potato Cannon Chronograph
//
//
//  James A. Crawford
//  27 August 2017
//
//
//
//  AR-PC talk with Rx= D0, Tx= D1
//  AR resets the Genie display using D4 (jumper on shield)
//  AR-Genie communicate via Rx= D2, Tx= D3   (the SoftwareSerial ports since Uno only has one UART)
//
//  Extremely Important !!
//       Running with too high a baud rate to the Genie display actually brings display updates
//     to an absolute crawl !
//     Using 9600 baud for the D0,D1 serial AND the same for the D2,D3 serial works quite well
//
//  Discouraged to use String objects; rather use char arrays to store strings
//
#include <genieArduino.h>
#include <SoftwareSerial.h>

Genie genie;
#define RESETLINE 4  // Change this if you are not using an Arduino Adaptor Shield Version 2 (see code below)

static const int Rx2=2;
static const int Tx2=3;

static long int _count_MSBs;
static long int _count_LSBs;
static bool _trigger1;
static bool _trigger2;


SoftwareSerial DisplaySerial(Rx2, Tx2);

void setup()
{
  pinMode(RESETLINE, OUTPUT);
  digitalWrite(RESETLINE,0);
  delay(300);
  digitalWrite(RESETLINE,1 );
  delay(3500);                  //let the display start up after the reset (This is important)

  pinMode(Rx2,INPUT);   // Needed in order to talk to LCD
  pinMode(Tx2,OUTPUT);
```



```
  delay(100);

  //
  //  Set up Arduino's GPIO
  //
  //  GPIO's 2, 3, 4 already used
  //
  //  Set Up Outputs
  //
  #define resetCounterBar  7
  #define galBar           8
  #define gauBar           9
  #define gblBar          10
  #define gbuBar          11
  //
  pinMode( resetCounterBar, OUTPUT );
  pinMode( galBar, OUTPUT );
  pinMode( gauBar, OUTPUT );
  pinMode( gblBar, OUTPUT );
  pinMode( gbuBar, OUTPUT );

  //
  //  Set Up Inputs
  //
  #define detect1  12
  #define detect2  13
  //
  pinMode( detect1, INPUT );
  pinMode( detect2, INPUT );

  #define dataY0    14
  #define dataY1    15
  #define dataY2    16
  #define dataY3    17
  #define dataY4    18
  #define dataY5    19
  #define dataY6     5
  #define dataY7     6
  //
  pinMode( dataY0, INPUT );
  pinMode( dataY1, INPUT );
  pinMode( dataY2, INPUT );
  pinMode( dataY3, INPUT );
  pinMode( dataY4, INPUT );
  pinMode( dataY5, INPUT );
  pinMode( dataY6, INPUT );
  pinMode( dataY7, INPUT );

  DisplaySerial.begin(9600);  // Set speed for Display softwareserial port. Apparently, only
9600 and 115200 available

  Serial.begin(115200);
  Serial.flush();
  Serial.println("Serial Port to Arduino Being Initialized\n");
  Serial.println(GENIE_VERSION);
```

```
  // NOTE, the genieBegin
  //
  delay(100);

  genie.Begin(DisplaySerial);   // Use software serial port for talking to the LCD display

  genie.AttachEventHandler(myGenieEventHandler); // Attach the user function Event Handler for processing events

  // Set the brightness/Contrast of the Display - (Not needed but illustrates how)
  // Most Displays, 1 = Display ON, 0 = Display OFF. See below for exceptions and for DIABLO16 displays.
  // For uLCD-43, uLCD-220RD, uLCD-70DT, and uLCD-35DT, use 0-15 for Brightness Control, where 0 = Display OFF, though to 15 = Max Brightness ON.
  //
  //  Check contrast control
  //
  genie.WriteContrast(1);
  delay(100);
  genie.WriteContrast(10);

  //
  //  Initialize items on Genie display
  //
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x00, 0);  // Count_MSBs
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x01, 0);  // Count_LSBs
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x02, 0);  // flightTime

  genie.WriteObject(GENIE_OBJ_USER_LED, 0x00, 0 );   // trigger1
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x01, 0 );   // trigger2
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x02, 0 );   // detect1
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x03, 0 );   // detect2

  genie.WriteObject(GENIE_OBJ_4DBUTTON, 0x00, 0);   // Reset

  _count_MSBs= 0;
  _count_LSBs= 0;
  _trigger1= false;
  _trigger2= false;

  digitalWrite( resetCounterBar, 0x00 );
  delay( 20 );
  digitalWrite( resetCounterBar, 0x01 );
  delay( 20 );

  //
  //  Try some other initializations
  //
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x02, 1 );
  delay( 2000 );
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x02, 0 );
  delay( 2000 );
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x02, 1 );
  delay( 2000 );
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x02, 0 );
  delay( 2000 );
```

```
  genie.WriteObject(GENIE_OBJ_USER_LED, 0x02, 1 );
  //delay( 2000 );
  //genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x01, "123456" );
  delay( 2000 );
  genie.WriteObject(GENIE_OBJ_LED_DIGITS, 0x01, 123 );

}
//================================================================

void myGenieEventHandler(void)
{
  //------------------------------------------------------------------------------------------------------
  //
  //  Needed programming guidance came from U24333 Visi-Geni Connecting 4D Display to Arduino Host.pdf
  //
  //------------------------------------------------------------------------------------------------------
  genieFrame Event;
  genie.DequeueEvent(&Event); // Remove the next queued event from the buffer, and process it below

  //If the cmd received is from a Reported Event (Events triggered from the Events tab of Workshop4 objects)
  //
  // The only objects on the LCD screen which report back are all trackbars and two toggle switches.
  // Need to figure out the object type first, and then the details to follow.
  // the commands (reports always correspond to a changed value). Only need the trackbar index
  // and the data values.
  //
  int  objIndex;
  int  dvalue;
  int  dvalue_pwm;
  int  dvalue_lcd;

  const bool Debug= false;

  Serial.print( "Command " );
  Serial.println( Event.reportObject.cmd );
  Serial.print( "Object " );
  Serial.println( Event.reportObject.object );
  Serial.print( "Index " );
  Serial.println( Event.reportObject.index );
  Serial.print( "Data MSB = " );
  Serial.println( Event.reportObject.data_msb );
  Serial.print( "Data LSB = " );
  Serial.println( Event.reportObject.data_lsb );

  objIndex= Event.reportObject.index;
  dvalue= Event.reportObject.data_lsb + 256*Event.reportObject.data_msb;

  switch( Event.reportObject.object )
  {
    case GENIE_OBJ_4DBUTTON:
      switch( objIndex )
```

```
        {
          case 30:  // Reset button hit. Issue reset strobe
            Serial.println( "Reset Chronograph Counter" );
            digitalWrite( resetCounterBar, 0x00 );
            delay(20);
            digitalWrite( resetCounterBar, 0x01 );
            //
            genie.WriteObject(GENIE_OBJ_USER_LED, 0x02, 0 );   // detect1
            genie.WriteObject(GENIE_OBJ_USER_LED, 0x03, 0 );   // detect2
            break;
        }
        default:
        break;
    }
}
//================================================================


void loop()
{
  static long waitPeriod = millis();

  static int counterA_lsbs;
  static int counterA_msbs;
  static int counterB_lsbs;
  static int counterB_msbs;

  bool _detect1;
  bool _detect2;

  genie.DoEvents(); // This calls the library each loop to process the queued responses
from the display

  if (millis() >= waitPeriod)
  {
    waitPeriod = millis() + 150; // rerun this code to update the display
  }

  _detect1= digitalRead( detect1 );
  _detect2= digitalRead( detect2 );

  if( _detect1 )
  {
    Serial.println( "Detect 1 TRUE" );
    genie.WriteObject( GENIE_OBJ_USER_LED, 0x02, 1 );
  }
  if( _detect2 )
  {
    Serial.println( "Detect 2 TRUE" );
    genie.WriteObject( GENIE_OBJ_USER_LED, 0x03, 1 );
  }

  if( _detect1 && _detect2 )
  {
    //
    //  Read back the counter value
```

```
    //
    //  Fetch counter A LSBs
    //
    digitalWrite( galBar, false );
    digitalWrite( gauBar, true );
    digitalWrite( gblBar, true );
    digitalWrite( gbuBar, true );
    counterA_lsbs= readCounter();
    Serial.print( "counterA_lsbs= " );
    Serial.println( counterA_lsbs );
    //
  }

}
//=================================================================

int readCounter(void)
{
  int readValue_digital;
  int readValue_analog;

  int threshold = 64;

  int readByte;

  readByte= 0;

  readByte= digitalRead( dataY7 );
  readByte= 2*readByte + digitalRead( dataY6 );

  readValue_analog= analogRead( dataY5 );
  readByte= 2*readByte + (readValue_analog > threshold)? 1 : 0;

  readValue_analog= analogRead( dataY4 );
  readByte= 2*readByte + (readValue_analog > threshold)? 1 : 0;

  readValue_analog= analogRead( dataY3 );
  readByte= 2*readByte + (readValue_analog > threshold)? 1 : 0;

  readValue_analog= analogRead( dataY2 );
  readByte= 2*readByte + (readValue_analog > threshold)? 1 : 0;

  readValue_analog= analogRead( dataY1 );
  readByte= 2*readByte + (readValue_analog > threshold)? 1 : 0;

  readValue_analog= analogRead( dataY0 );
  readByte= 2*readByte + (readValue_analog > threshold)? 1 : 0;

  return( readByte );

}
//=================================================================
```

7 Appendix: Schematics

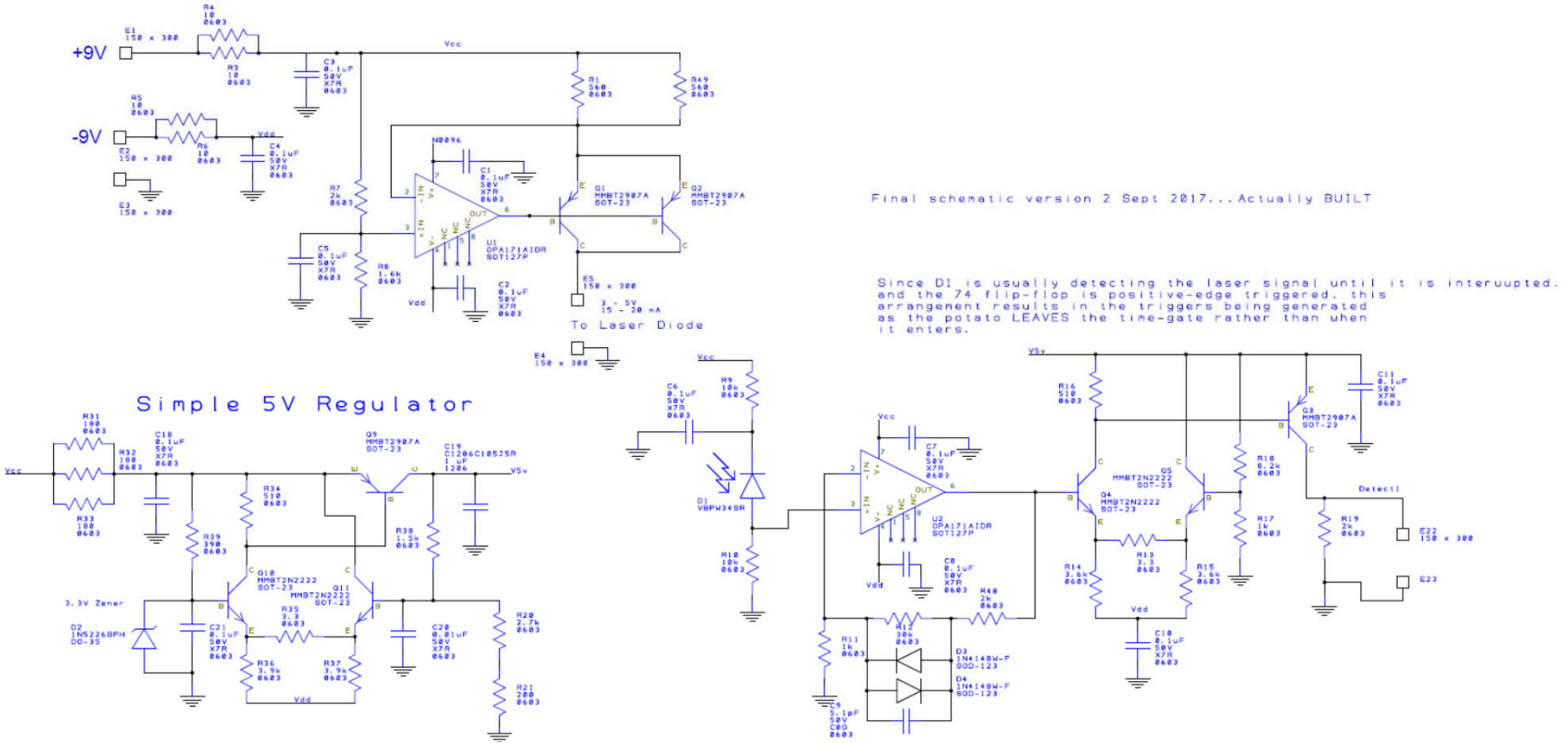

Figure 12 Schematics page 1

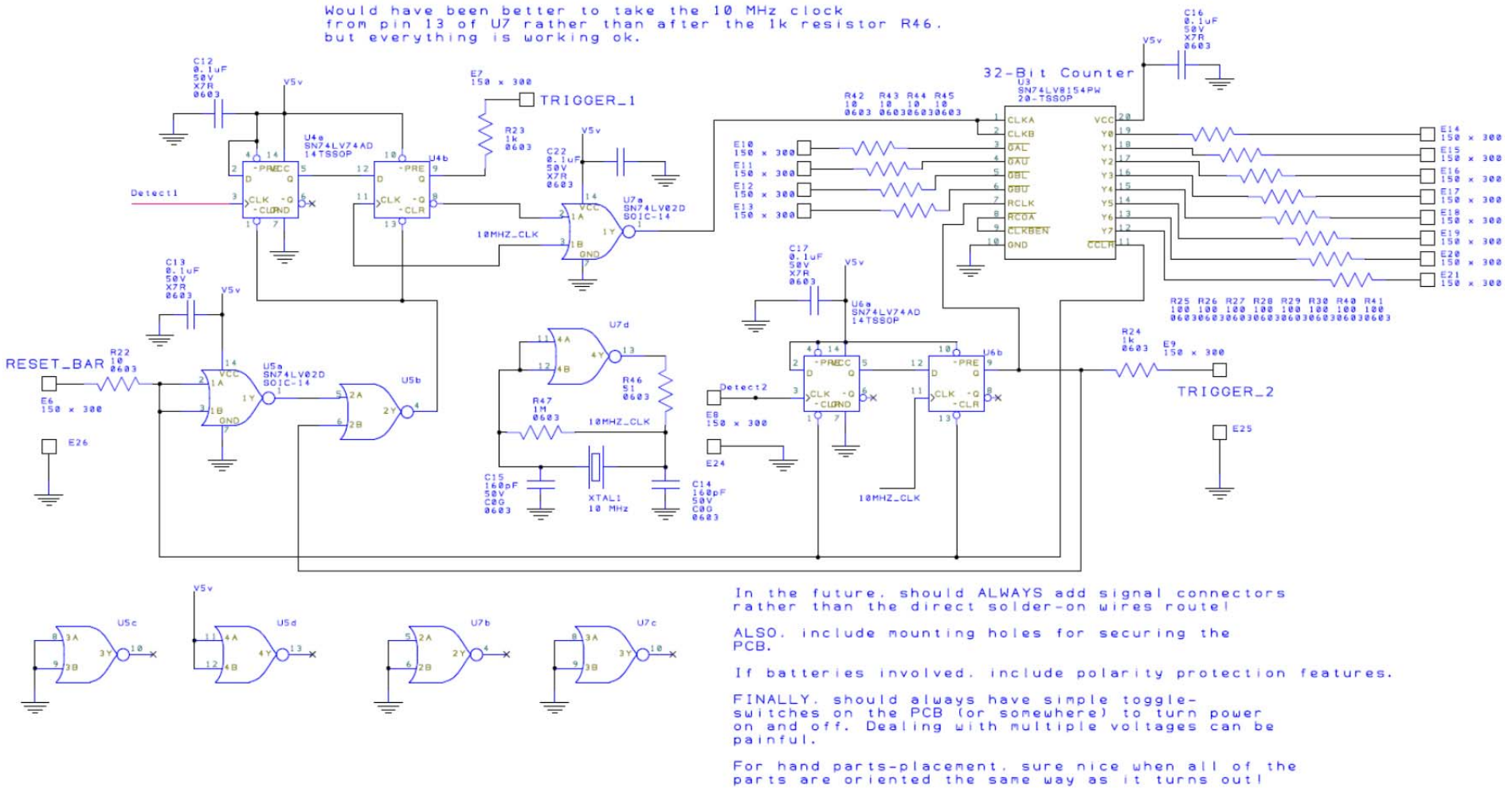

Figure 13 Schematics page 2