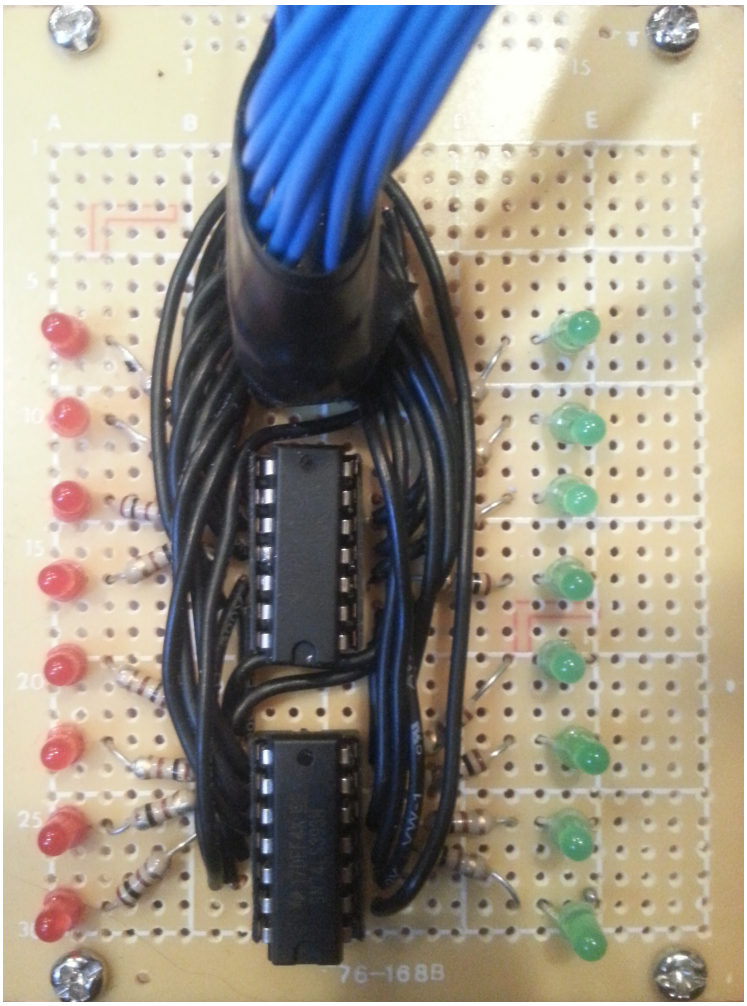
USER MANUAL
16 Bit Counter with Binary Output for PHY 445

Benjamin Brandwin



Contents

1 Introduction

1.1 Definitions

SD . . . . . Secure Digital - a non-volatile memory card format for use in portable devices, such as mobile phones, digital cameras, GPS navigation devices, and tablet computers.[1]

TTL . . . . . Transistor-transistor logic - A widespread digital logic standard using transistors and resistors. Uses positive 5V logic.

NIM . . . . . Nuclear Instrumentation Module - Logic used in nuclear and particle physics. Uses fast negative logic signals.

SPI . . . . . Serial Peripheral Interface - a synchronous serial data link standard, named by Motorola, that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines.[2]

MOSI . . . . . Master Output to slave for SPI

MISO. . . . . Slave Output to master for SPI

CLK. . . . . Clock output from master to slave

CS. . . . . Chip select, or slave select pin

1.2 Overview

This manual outlines the operation of a dual SN74LS393 - 16 bit counter with binary output, and how to interface it with modern data collection techniques. This is done using an Arduino Mega 2560 microcontroller, a Seeed-Studio SD card shield, and a Makershed Protoshield. The 16 bit counter is designed to receive a TTL logic standard negative triggered pulse. When the clearing input of the counter is grounded the counting condition is active. The program loaded into the Arduino Mega 2560 determines the counting interval. At the end of each interval, data is written to a file on an SD card, followed by a positive square pulse to the clearing input. The data is written to the file as [count interval number (1, 2, 3 etc. . .)] [number of counts

during interval] [time (ms)]. Each value is deliminated by a space. The file name and the count interval are user defined and must be changed directly in the code.

NOTICE: Although the data is written to a file on the SD card, it DOES NOT SAVE AUTOMATICALLY! Data must be saved manually by pressing the button on the Makershed Protoshield. It is important to read this entire manual to ensure you don't lose your data!

1.3 NIM to TTL Conversion

Before this counter can be used the NIM standard negative logic used in the coincidence measurements must be converted to a TTL standard positive logic. This is done quite easily using a gate and delay generator. It is important to decrease the TTL output pulse width from this device to it's minimum. It's possible for counts to overlap for high rates and so the pulse width must be made as small as possible. Keep in mind that the SN74LS393 counters are actually able to count faster(35MHz) then the Ortec 871(25MHz) typically used. The TTL output of the gate and delay generator must be split, with one end terminated with a 50Ω terminator, and the other end output to positive and negative aligator clip. The negative clip must be attached to the common ground of the counter and the positive clip must be attached to the count input.

2 Construction

2.1 Overview

This counter is built using two SN74LS393 dual 4 bit counters chained together to make a single cascading 16 bit counter. This is done by connecting Q_4 output of the lower digit counter to the A input of the succeeding higher digit counter. The counting outputs are each attached to LED's through 100 Ohm resistors, and to male header pins for output. The clearing inputs of each counter are chained together and attached to a male header, and the A input of the lowest digit counter is also attached to a male header. Male headers are also used for the $V^+(5V)$ and $V^-(0V)$ inputs.

2.1.1 Circuit Diagram

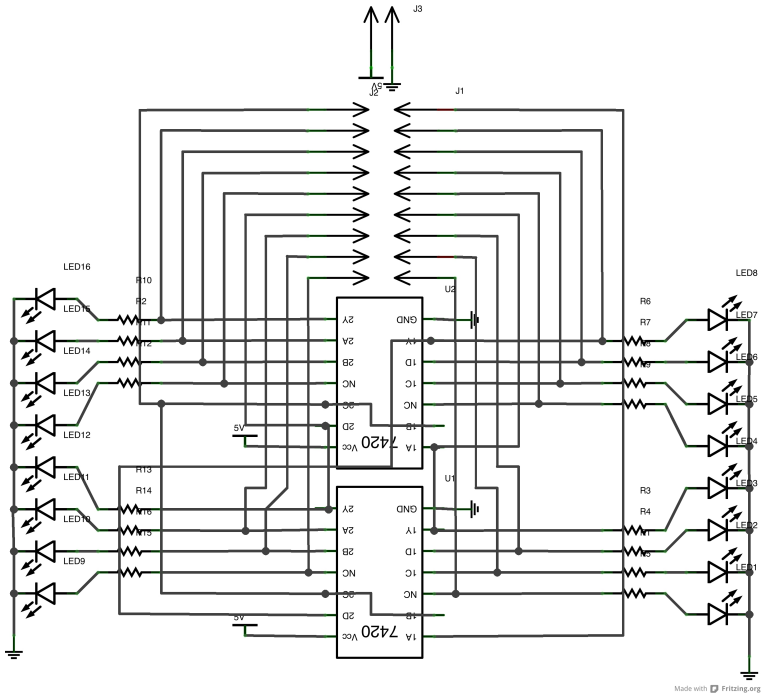


Figure 1: Counter Schematic (please refer to SN74LS393 data sheet for pin descriptions))

2.2 External Connections

The 16 bit binary number is output to a connector designed specifically to correspond to the appropriate pins on the Arduino Mega 2560's 'A' and 'C' port registers. The least significant bit is labeled with white tape and corresponds to pin 22 on the Arduino. The connector has 2 rows of 8 pins each and each successive bit alternates rows. In other words, one row contains only even bits, while the other row contains only odd bits. The counter also has 4 additional inputs. The unlabeled input is the clear, the black input is $V^{+}(5V)$, the white input is ground(0V), and the green input is for the count. The wires are taped together for their safety but all of the wires are female and removable (revealing the male headers soldered to the board), and one can easily adapt the available connections for other micro-controllers.

3 Interfacing with Arduino Mega 2560

NOTICE: In order to use the Arduino Mega 2560 the Arduino software must be downloaded from arduino.cc. The systems it is available for directly include Windows, Mac and Linux, however arduino.cc also has a downloadable source code for the advanced user. Downloading this software will install a directory on your system named Arduino. The SD card supplied contains a folder of the same name. First you must open this folder. Select the entire contents and copy them to the Arduino directory on your system. When you open the Arduino directory on your system, it should look exactly the same as opening the Arduino folder on the SD card. Also, the Arduino software is by default set to the basic Arduino board. In order to use the Arduino Mega 2560 you must open the software and navigate to Tools > Board > Arduino Mega 2560 or Mega ADK. This will place a checkmark next to the appropriate board name.

3.1 Overview

The Arduino Mega 2560 is an open source platform which is programmed using C++. This section will only be covering whats needs to be known for the operation of the 16 bit counter. Full documentation on Arduino can be found at arduino.cc. The Arduino Mega 2560 comes equipped with optionally designated port registers. These port registers allow the Arduino to read binary values from multiple pins simultaneously. The 16 bit counter will be attached to port registers A (pin22-29) and C (pin37 - 30). The 16 pin counter connector has been specifically designed to correspond to the appropriate pins on Arduino. The pin labeled in white corresponds to the least significant bit; pin 22 on the Arduino.

3.2 Programing Environment

The code used to collect data can be found directly in the Arduino environment by navigating to File > Sketchbook > Data_Logger_with_binary_output as long as the contents of the Arduino file on the SD card have been copied properly. Line 37 of the code contains the name of the file to be written to. This must changed manually by the user and can contain no spaces but may contain underscores and numbers. Line 39 of the code contains the length of the counting interval in milliseconds. This can be left alone or adjusted as desired by the user. The code is further explained in the comments. The status of the code while running can be viewed using the serial monitor

found under Tools > Serial Monitor. If any errors occur, they appear in this window. If the serial monitor fails to update after uploading a code, then it must be closed and re-opened.

3.2.1 Code

PLEASE NOTE: This code will be a lot easier to read in the Arduino programming environment.

```
const int chipSelect = 53;
/*

 * SD card attached to SPI bus as follows:
 ** MOSI - pin 51 (interconnected to pin 11)
 ** MISO - pin 50 (interconnected to pin 12)
 ** CLK - pin 52 (interconnected to pin 13)
 ** CS - pin 53 (leave unconnected)

 This code was created 3/20/13
 by Benjamin Brandwin

This code is designed to work with an arduino mega 2560 micro-controller and a 16
bit binary counter. The counter has been built to output two 8 bit binary numbers
to the arduino mega's 'A' and 'C' port registers. The counter counts pulses from
an external source (in our case gamma-gamma coincidences) and the arduino counts
time intervals. Once the desired (user selected) time interval has been reached, the
arduino records the current value at the port registers and recods the count and the
time in milliseconds to a text file on an external SD memory card. When the external
output is removed, the test Pin should advance the count by one. All together there
are 4 + 16 connections.  6 make up the control scheme, and 16 for the binary input.
The remaining 6 are: 5V, Gnd, Test Signal, Clear, Save, and vibration feedback.

 */

#include <SdFat.h>
SdFat sd;
SdFile myFile;
```

```
const int testPin = 42;  //test signal
const int clearPin = 43;  //clear signal
const int switchPin = 7; //save button
const int beep = 6; // beep feedback
const char* file_name = "dat1.txt"; // enter desired file name "HERE.txt"

long interval = 1000; // counting interval in milliseconds
unsigned long previousMillis = 0; // holds previous time a count was saved

unsigned long currentMillis = 0; //holds current time value.


int lastSwitchState = LOW;  // switching varibale for saving


long time; // holds current time of save
int minute; // holds integer minute value
int second; //holds integer second value left over after minute
int mils; // hols integer millisecond value left over after seconds

long ones; //holds first binary number
long Two56; // holds second binary number (mulitiplied by 256)
long num; // holds total binary number

int i = 0; // counting interval

void setup() {

   DDRA = B00000000; // Initialize port A (pins 22 - 29) as inputs
   DDRC = B00000000; // Initialze port C (pins 37 - 30) as inputs

   // Initialize pins as inputs or outputs
   pinMode(clearPin, OUTPUT);
   pinMode(beep, OUTPUT);
   pinMode(testPin, OUTPUT);
   pinMode(switchPin, INPUT);

   // Set pins, initial conditions including initial count clear

   digitalWrite(beep, LOW);
```

```
   digitalWrite(switchPin,HIGH);

   // Start the serial moniter at 115200 baud
   Serial.begin(115200);

  // Start sd card. return error if failed
  if (!sd.begin(chipSelect, SPI_FULL_SPEED)) sd.initErrorHalt();

  // open the file for write at end like the Native SD library
  if (!myFile.open(file_name, O_RDWR | O_CREAT | O_AT_END)) {
    sd.errorHalt("opening for write failed");
  }
  else {
    Serial.println("Open Sucesful");
  }

    digitalWrite(clearPin, HIGH);
   digitalWrite(clearPin, LOW);
}

void loop() {

  // this is a test signal sent to the counter through testPin (pin 42).
   digitalWrite(testPin, LOW);
   digitalWrite(testPin, HIGH);
    delay(1); /* adjust this delay higher or lower to test the counter for the
proper counting sequence. Values of 100 - 1000 will allow a visual test
the first 8 bits and values of 1 - 5 will allow a visual test of the next 8 bits */



  int Switch = digitalRead(switchPin);

  // if the save switch is pressed do the following:

  if (Switch != lastSwitchState) {
    if (Switch == LOW) {

      // close the file to save
```

```
      if (!myFile.close()) {
        Serial.println("Save Failed");
      }
        else {
      Serial.println("Save Succesful");

      // beep once for succesful save (useful if serial moniter is not available)
      digitalWrite(beep, HIGH);
      delay(100);
      digitalWrite(beep, LOW);


        }

        // reopen the file
      if (!myFile.open(file_name, O_RDWR | O_CREAT | O_AT_END)) {
    sd.errorHalt("opening for write failed");
  }
   else {




/* --------   if the file reopens succsfully print in that the Save was succesful----
 -------------       and the time of the save           -----------------------------
      */


      myFile.print("SAVE SUCCESFUL  ");
      Serial.print("OPEN SUCCESFUL  ");
      time = millis();
      minute = time/60000; // integer value will cut off remainder
      second = (time % 60000)/1000; /* modulus will take the integer remainder for
      second */
      mils = (time % 1000); /* modulus again will take the integer remainder for
      miliseconds */
      myFile.print(minute);
      Serial.print(minute);
      myFile.print(":");
      Serial.print(":");
```

```
    if (second < 10 ) {
      myFile.print("0");
      Serial.print("0");
    }
    myFile.print(second);
    Serial.print(second);
    myFile.print(":");
    Serial.print(":");
    if (mils < 10) {
      myFile.print("00");
      Serial.print("00");
    }
    else if (mils < 100 && mils >=10) {
      myFile.print("0");
      Serial.print("0");
    }
    myFile.println(mils);
    Serial.println(mils);

    myFile.close(); // reclose the file to save the timestamp
    if (!myFile.open(file_name, O_RDWR | O_CREAT | O_AT_END)) {
  sd.errorHalt("opening for write failed");
    }
    else {
      delay(50);
    digitalWrite(beep, HIGH); // beep a second time for a succesful reopening
    delay(100);
    digitalWrite(beep, LOW);
    }

}
digitalWrite(clearPin, HIGH); // clear the counter to begin a new count cycle
  digitalWrite(clearPin, LOW);
previousMillis = millis(); // reset the count cycle
  }

}

// Reset the switch state
lastSwitchState = Switch;
```

```
   ones = PINA; // save the first 8 bits
   Two56 = PINC; // save the second 8 bits
   Two56 = Two56*256; // multiply the second 8 bits by 256
   num = ones + Two56; // total count


  currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = millis();



    digitalWrite(clearPin, HIGH); // clear the counter to begin a new count cycle
    digitalWrite(clearPin, LOW);

   Serial.print(i); // print interval numnber to Serial Moniter
   Serial.print(" ");
   Serial.print(num); //print count to serial moniter
   Serial.print(" ");
   Serial.println(currentMillis); //print time in ms to serial moniter

   // repeat print to SD card file
   myFile.print(i);
   myFile.print(" ");
  myFile.print(num);
  myFile.print(" ");
  myFile.println(currentMillis);

  i = i + 1; //advance count interval by 1

  }
}
```

3.3 SeeedStudio SD Card Shield

The data is collected and written to an SD card using a 3rd party shield for Arduino made by SeeedStudio. Full documentation on the SeeedStu-

dio SD Card Shied can be found at SeeedStudio.com. In order to use the shield, the contents of the Arduino folder on the provided SD card must be copied exactly to the Arduino directory on your computer. This contains the libraries necessary to run the device. Data is written to the SD card via SPI (Serial Peripheral Interface) on pins 50(MISO), 51(MOSI), 52(CLK), 53(CS), however, the SD card shield is wired for the basic Arduino board which uses pins 12(MISO), 11(MOSI), 13(CLK), 4(CS). The chip select pin is easily changed within the code and can therefore be left unconnected, however the other pins must be connected to their corresponding types via jumper wires; 50− >12, 51− >11, 52− >13.

3.4 MakerShed Protshield

The MakerShed Protoshield can be found at makershed.com and is a basic prototyping shield. It contains some basic components such as a potentiometer, a user definable button, and a breadboard for small circuits. You'll notice by now that each shield extends the pins of the Arduino upwards. The protoshield must be on top. It is used to provide a manual save button, as well as audible feedback of a successful file save. A buzzer is attached to pin 6 and ground. The female header labeled btn1 is attached to pin 7, which is held high through a resistor. When the button labeled btn1, in the corner of the protoshield, is pressed, it will save the file. This must only be done once at the end of your data collection but be warned - !!THIS MUST BE DONE TO SAVE YOUR DATA!! One beep from the buzzer confirms that the file has been saved. Two beeps from the buzzer confirms that the file successfully saved and re-opened to begin counting again. If nothing is heard, this indicates that an error may have occurred.

4 Operation

IMPORTANT: You must read this entire manual to fully understand the operation this device. DO NOT SKIP AHEAD IF YOU CARE ABOUT YOUR DATA!

The first step in operation is make all of the appropriate connections from the counter to the Arduino. the unlabeled blue wire goes to pin 43 on the Arduino, this is the clearing input, and will send out a high pulse at the end of each counting interval. When testing the counter, the green labeled wire goes to pin 42. This is the signal input and will receive a positive logic square pulse from Arduino at a frequency determined by a delay within the

code. The black labeled wire gets attached to the 5V pin, and the white labeled wire goes to Gnd. It is always important to test the device before beginning operation. When the device is working properly the LED's will begin a count. If an error has occurred some LED's may light up, but they will not count and instead remain stable. Another error can also be a short circuit within the board. If this is the case, the LED's may light up, but they will be erratic and not exhibit the proper counting sequence. If this happens, a soldering knife must be used to trace around each connection to remove any small shorts.

Once testing is successful, the counting input from the TTL logic of the gate and delay generator is attached to the ground of the Arduino, as well as the count input of counter(green labeled wire). Line 37 of the code is where you will input your file name. Make your names short but specific with NO SPACES, such as Na_200_L.txt, to indicate that this measurement was of ^{22}Na with a 200 nanosecond delay in the left detector. Remember, you MUST put a .txt at the end of your file name or it will be unreadable. Load the code with your file name using the Arduino software.

WARNING: ARDUINO MUST REMAIN IN THE STYROFOAM. IF A CODE PRESENTS A TIMEOUT ERROR, YOU MUST REMOVE THE EXTERNAL SHIELDS AND THEN RELOAD.

Once the code successfully loads, immediately open the serial monitor found under Tools (on a Mac press Shift+Command+M). The Arduino can be left alone to collect data for an extended period of time, but be warned, if the Arduino shuts off or loses power before the save switch is pressed, all of your data will be lost!! Remember, the SAVE BUTTON is located next to the GREEN LED and is labeled btn1. You will receive a beep confirmation for the save, and a second beep confirmation for the re-opening of the file. This will also show up in the serial monitor. Do not modify the code for automatic saving as this will sometimes cause an error during the run. This is really bad because all it takes is one error in re-opening a file for your code to stop recording data altogether. It's important to monitor the status of the code using the serial monitor

WARNING: DO NOT press the button next to the RED LED, it is a reset switch and you will LOSE all of YOUR DATA!

An alternative setup which was not used in this counter involves the use of an impulse relay. An impulse relay only requires a momentary impulse of electric current to either connect or disconnect a separate electric circuit. If the count from the gate and delay generator is first passed through the relay, one can utilize the 'end of interval' and 'end of dwell' outputs from the Ortec counter, attaching them to the switching inputs of the impulse relay. This setup has not been tested yet, but should effectively turn the counter into a slave to the Ortec. It's important to mention that the program must also be adjusted for such a setup. In this case the program must be designed to write the data when the end of interval pulse is applied(as opposed to a timing condition). Also careful wiring must be used to ensure that the counter is cleared at the start of each new interval.

5 Works Cited

[1] http://en.wikipedia.org/wiki/Secure$_{D}igital_{C}ard$

[2] http://en.wikipedia.org/wiki/Serial$_{P}eripheral_{I}nterface_{B}usData_{t}ransmission$