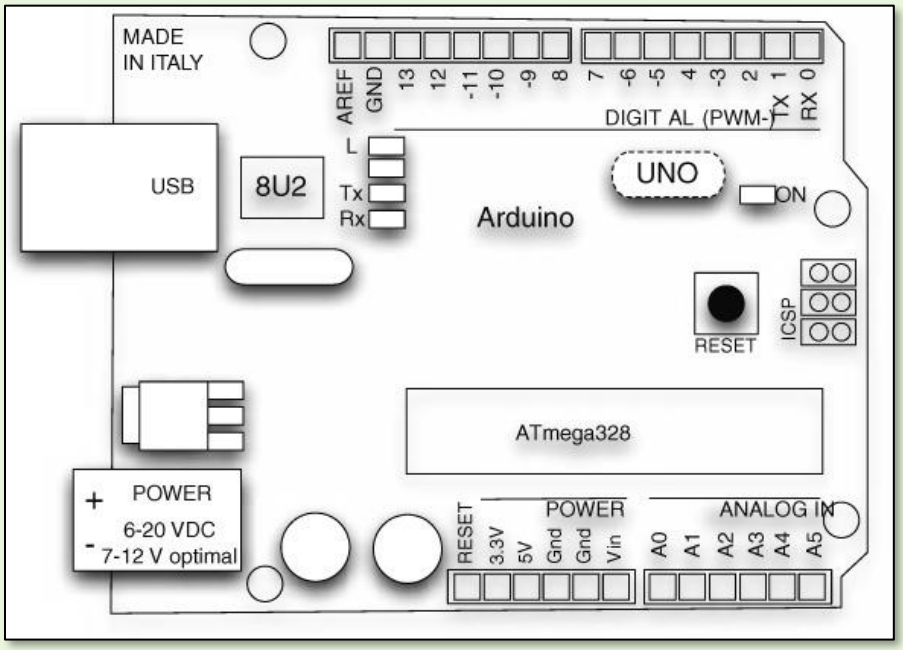
Practical PCB Design and Manufacture

Golden Arduino
BOARD 3
PCB Design



Practical PCB Design and Manufacture

Table of Contents

Practical PCB Design and Manufacture

☞ *INTRODUCTION*

In this laboratory, I documented the process and outcomes of an exercise designed to deepen my understanding of the entire prototype design flow. The main objective was to get practical experience in reading and analysing datasheets, identifying the ambiguities they contain, and using this knowledge in real-world situations. It was my responsibility to improve my designs by adding features like isolation switches, test points, and indication LEDs to make testing and assessment easier.

I built prototype circuits on a solderless breadboard, discovered unusual, non-standard components that were required for the project, and started the Power-On Reset (POR) during the first stage of the board design. This phase stressed the significance of following best practices in signal routing to prevent problems like ground bounce and cross talk, as well as techniques to lessen I/O switching noise on power rails.

A critical component provided for this lab was the commercial Red board Arduino Uno board. Although functionally adequate, meeting basic connectivity specifications, I was encouraged to conduct a thorough examination to identify potential improvements in areas such as noise control, assembly, testing, and board bring-up.

The culmination of this project involved designing a customized "Golden Arduino" board. This version was expected to not only meet the same connectivity specifications as the standard Arduino Uno but also incorporate enhanced features for improved noise control and ease of assembly and testing. Through this assignment, I systematically navigated through each of the seven steps in the board design process, applying theoretical knowledge to practical design challenges.

☞ *PLAN OF RECORD*

The plan of record for building my Board 3 is as follows:

For the board features,

1. Power Input Selection: Switch to choose power from either the power jack or USB, exclusively.
2. Voltage Regulation: 5 volts to 3.3 volts conversion via a Low Dropout (LDO) regulator.
3. Clock Configuration: Incorporates a 12 MHz clock for the CH340G and a 16 MHz clock for the ATmega328.
4. Initial Programming: Features ICSP pins for boot loading the ATmega328P.
5. Visual Indicators: An LED near the power input indicates power status.
6. Circuit Protection: Includes a TVS diode at the USB port for ESD protection and a ferrite filter on the ADC's AVCC pin to reduce noise.
7. Connectivity Enhancements: Standard Arduino headers supplemented with extra ground pins and decoupling/bypass capacitors at each IC.
8. Communication Support: Facilitates USB UART communication via the CH340G.
9. Design and Safety Standards: Test points and isolation switches improve safety and functionality, while a copper ground plane enhances circuit stability.

Practical PCB Design and Manufacture

For the proper testing and what it means to work, I have done the following implementations:

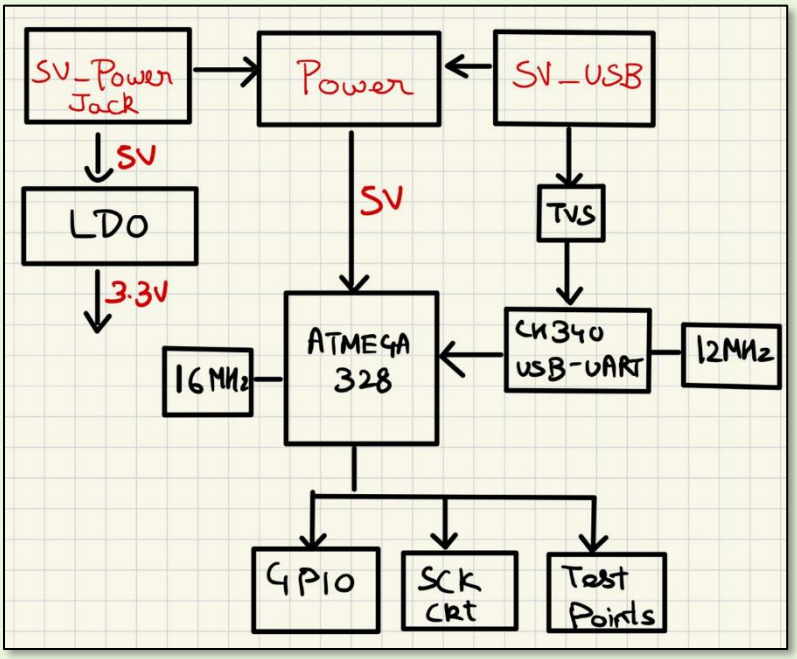
1. USB Mini support was incorporated for programming and powering the board.
2. A power plug was included to accommodate an external 5V AC to DC charger.
3. A power selector switch was added to allow selection between USB or an external 5V power supply as the source.
4. An LDO (Low-Dropout Regulator) was used to convert the 5V input to 3.3V efficiently.
5. A power isolator switch was integrated to isolate power to the microcontroller and the CH340 chip.
6. Indicator LEDs for both 5V and 3.3V were installed to provide visual feedback on power status.
7. A reset switch was strategically placed to facilitate easy resetting of the microcontroller in case of any undesirable behavior.
8. ICSP compatibility was integrated to boot load the microcontroller, offering an alternative programming method.
9. A TVS (Transient Voltage Suppressor) chip was added to protect the board from potential static voltage damage.
10. Test points were planned for critical signals including 5V input, 3.3V input, and USB connections.

For the risk management,

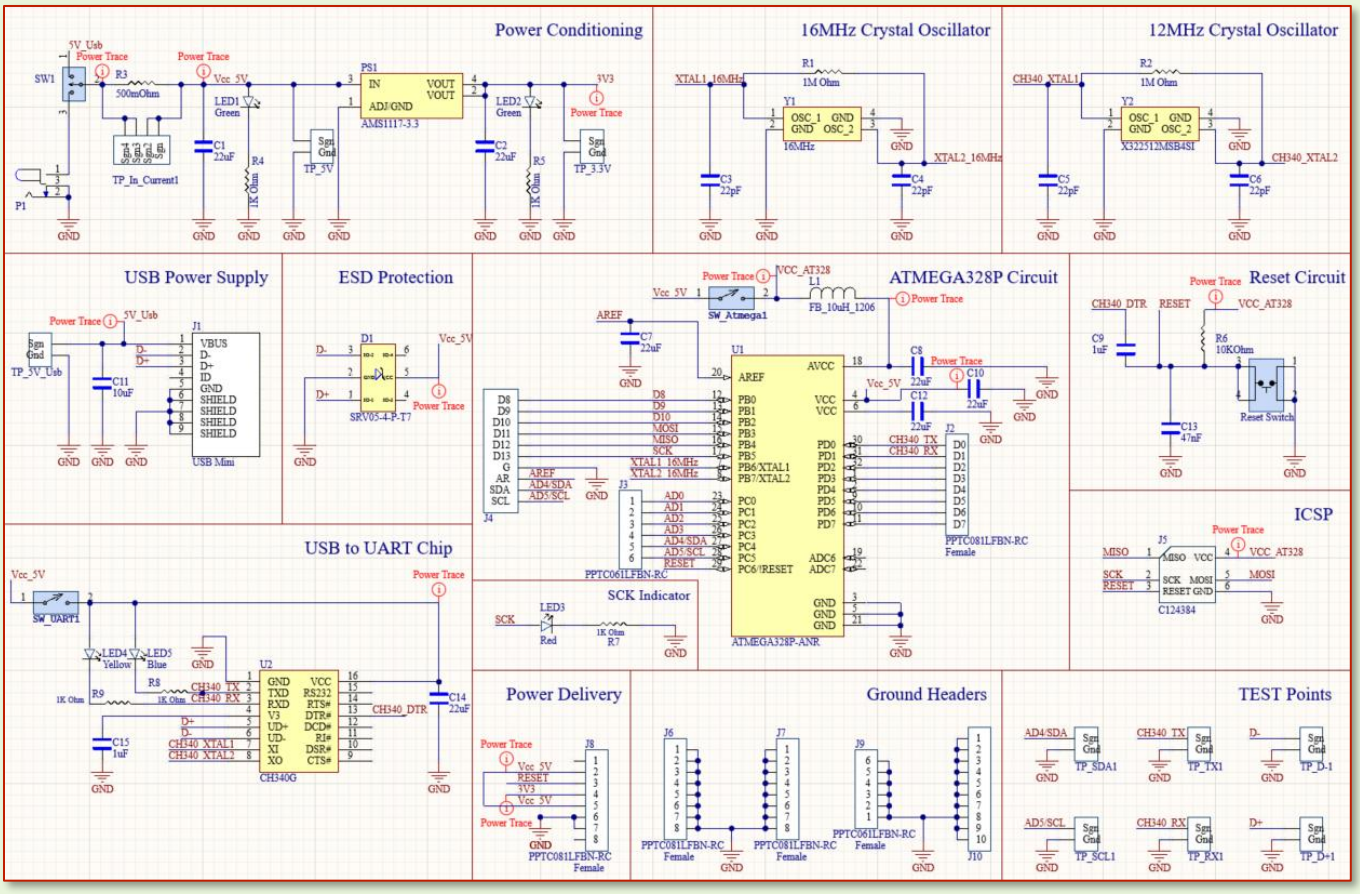
1. A naming convention was used for all components on the board, and the board itself was named and labelled.
2. Test points were added for inrush current measurement, D+, D-, TX, and RX, as well as for troubleshooting and validation purposes.
3. A filter capacitor was employed at the LDO's output to reduce voltage oscillations, and for consistency, all power lines were made 20 mills wide and signal lines 6 mills wide, as offered for free by the fabrication vendor.
4. Switches were utilized to control power to the board, and a 16 MHz clock was placed close to the ATmega328P, while a 12 MHz clock was situated near the CH340G to ensure accurate timing.
5. RX and TX line routing was designed to be approximately the same length to avoid delays, indicators were used for easier debugging, and LEDs were added to the power stage and RX-TX output to monitor transmission activity. A QR code linking to the portfolio page was also placed on the board, and board size restrictions were maintained below 3.9 in by 3.9 in.

Practical PCB Design and Manufacture

☞ ROUGH SKETCH OF SKEMATIC

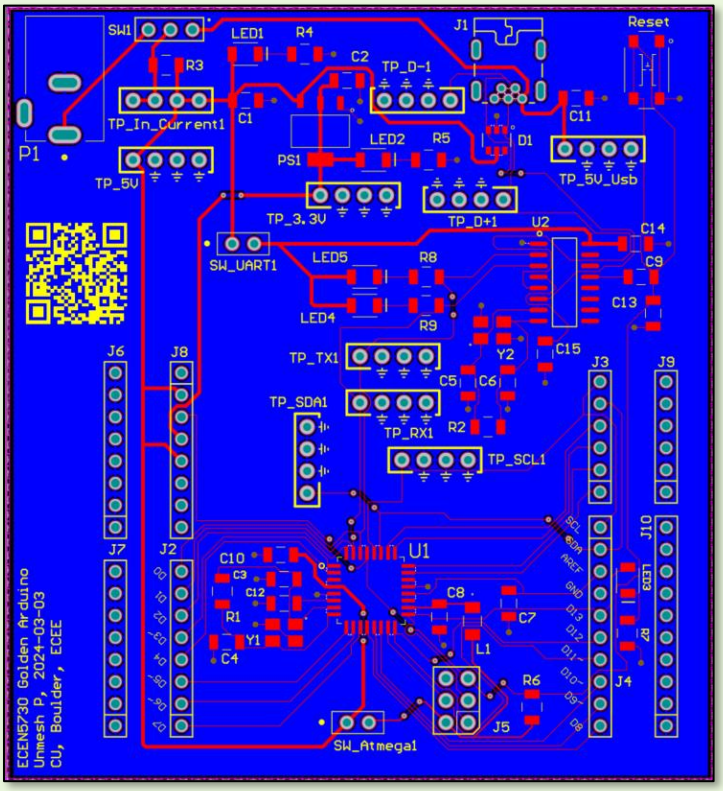


☞ SKEMATIC CAPTURE ON ALTIUM

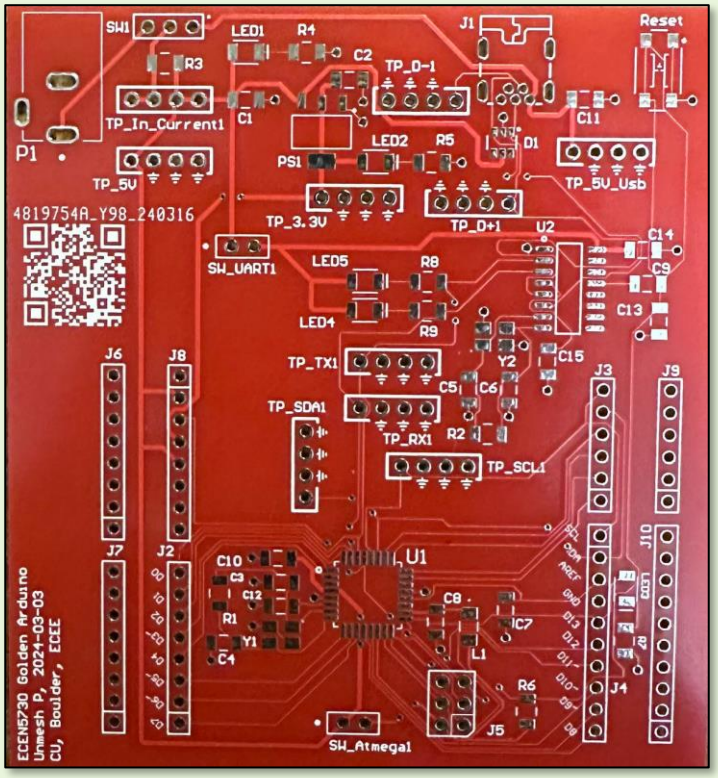


Practical PCB Design and Manufacture

☞ *BOARD LAYOUT ON ALTIUM*

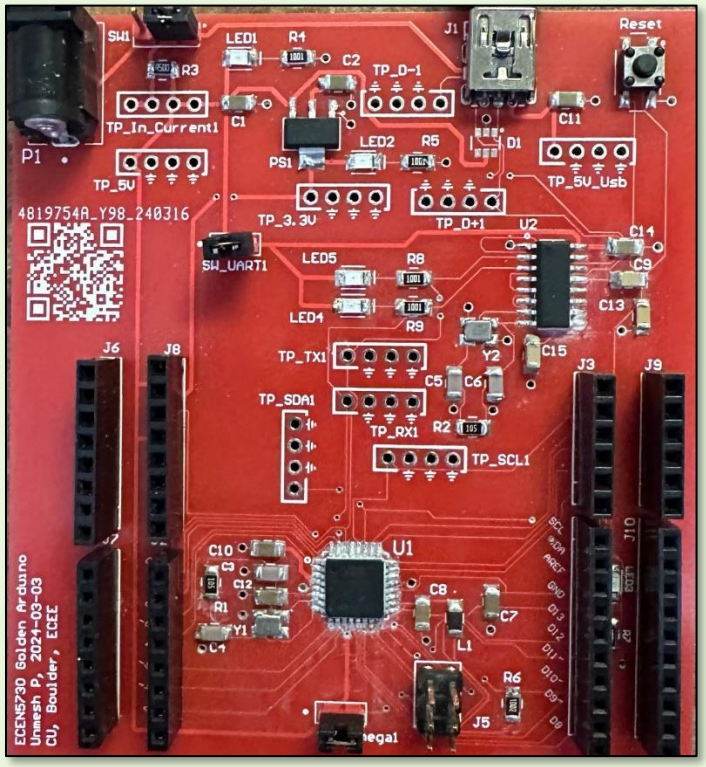


☞ *UNASSEMBLED BOARD PICTURE*

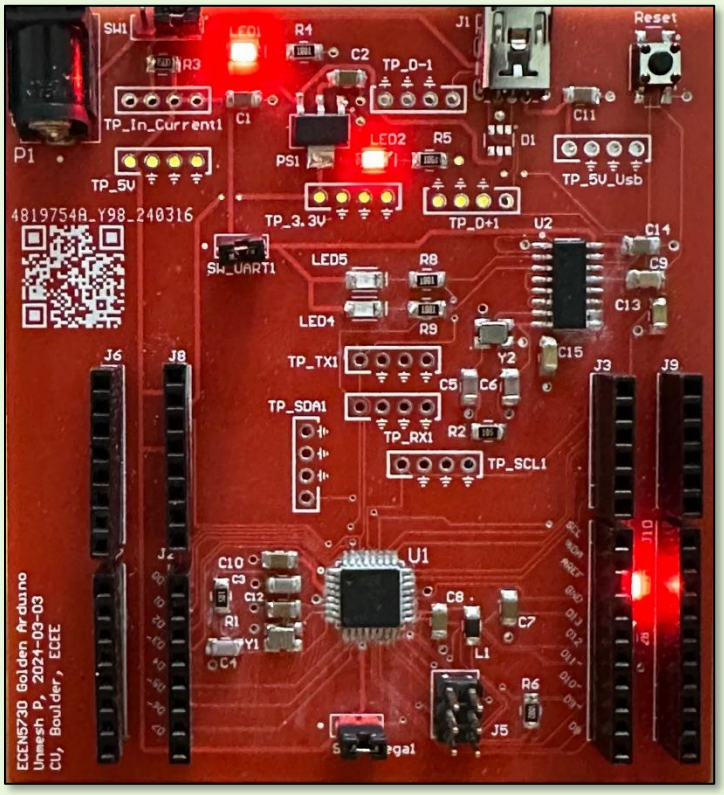


Practical PCB Design and Manufacture

☞ ASSEMBLED BOARD



☞ ASSEMBLED WORKING BOARD



Practical PCB Design and Manufacture

☞ *WHAT WORKED*

1. The LED1 turned ON and a voltage of 5V was seen at the 5V rail test point to conclude that the board was in a working state.

2. The LED2 turned ON at 3.3V which was used to confirm that the LDO had properly converted the 5V to 3.3V.

3. Each I/O pin was successfully toggled, used to light an external LED and to read a button's state, verifying microcontroller communication.

4. The UART was tested by sending and receiving data between the Arduino and a computer, confirming the integrity of data transmission.

5. A new program was uploaded to the board, and it ran without issues, ensuring the bootloader was correctly installed and functioning.

6. I2C connections and the data transfer through it was tested using a commercial Arduino as a master device and the Golden Arduino as the slave device.

The following code was run on the master device:

```
// Include Arduino Wire library for I2C
#include <Wire.h>

// Define Slave I2C Address
#define SLAVE_ADDR 9

// Define Slave answer size
#define ANSWERSIZE 5

void setup() {

  // Initialize I2C communications as Master
  Wire.begin();

  // Setup serial monitor
  Serial.begin(9600);
  Serial.println("I2C Master Demonstration");
}

void loop() {
  delay(50);
  Serial.println("Write data to slave");
```

Practical PCB Design and Manufacture

```
  // Write a charatre to the Slave
  Wire.beginTransmission(SLAVE_ADDR);
  Wire.write(0);
  Wire.endTransmission();

  Serial.println("Receive data");

  // Read response from Slave
  // Read back 5 characters
  Wire.requestFrom(SLAVE_ADDR,ANSWERSIZE);

  // Add characters to string
  String response = "";
  while (Wire.available()) {
      char b = Wire.read();
      response += b;
  }

  // Print to Serial Monitor
  Serial.println(response);
}
```

The following code was run on the slave device:

```
// Include Arduino Wire library for I2C
#include <Wire.h>

// Define Slave I2C Address
#define SLAVE_ADDR 9

// Define Slave answer size
#define ANSWERSIZE 5

// Define string with response to Master
String answer = "Hello";

void setup() {

  // Initialize I2C communications as Slave
  Wire.begin(SLAVE_ADDR);

  // Function to run when data requested from master
  Wire.onRequest(requestEvent);

  // Function to run when data received from master
  Wire.onReceive(receiveEvent);
```

Practical PCB Design and Manufacture

```
  // Setup Serial Monitor
  Serial.begin(9600);
  Serial.println("I2C Slave Demonstration");
}

void receiveEvent() {

  // Read while data received
  while (0 < Wire.available()) {
    byte x = Wire.read();
  }

  // Print to Serial Monitor
  Serial.println("Receive event");
}

void requestEvent() {

  // Setup byte variable in the correct size
  byte response[ANSWERSIZE];

  // Format answer as array
  for (byte i=0;i<ANSWERSIZE;i++) {
    response[i] = (byte)answer.charAt(i);
  }

  // Send response back to Master
  Wire.write(response,sizeof(response));

  // Print to Serial Monitor
  Serial.println("Request event");
}

void loop() {

  // Time delay in loop
  delay(50);
}
```

Practical PCB Design and Manufacture

☞ *I2C TEST*

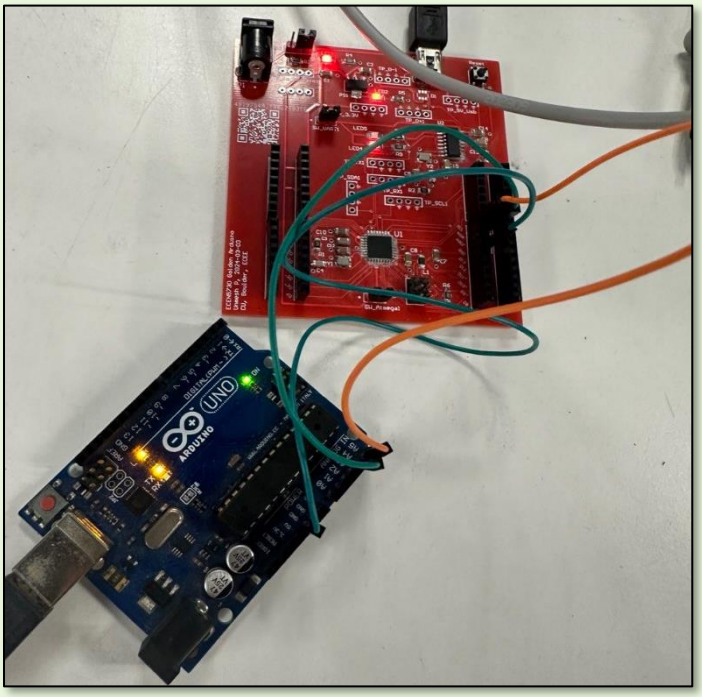


Fig 1: I2C between two Arduinos

The output I am getting on the oscilloscope is as follows:

Fig 2: I2C SCL and SDA lines

Practical PCB Design and Manufacture

☞ *SCOPE ANALYSIS*

For the Golden Arduino board, the scope shots given below are for the 5V_USB test point and the 3.3V test point respectively:

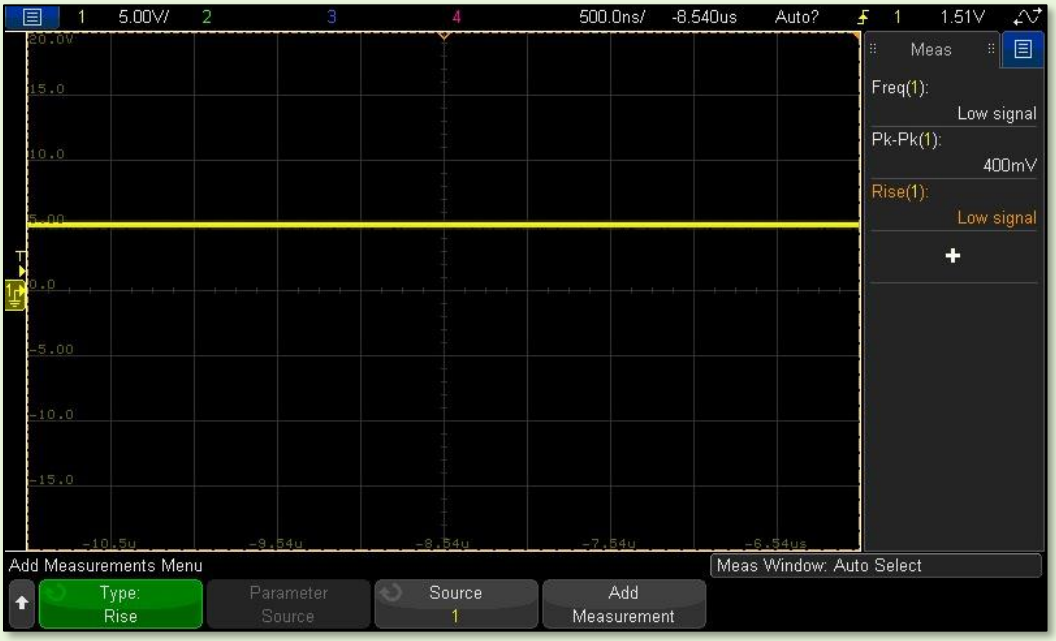


Fig 3: 5V_USB

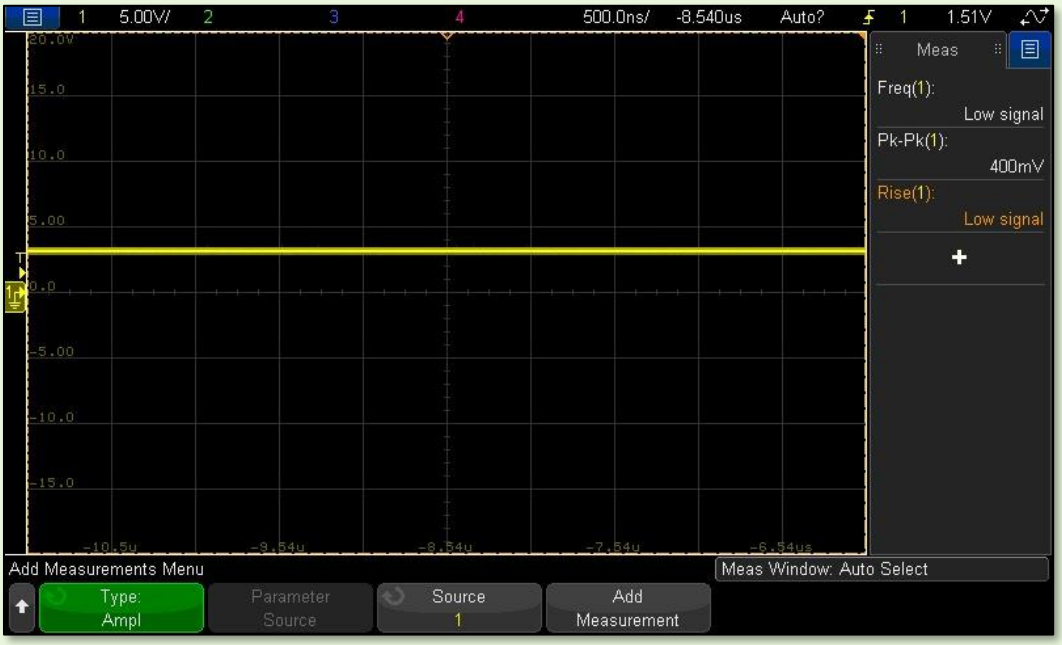


Fig 4: 3.3V LDO Output

Practical PCB Design and Manufacture

When I connected the USB cable to the Arduino, these were my readings at the D+ and D- points of the CH340 USB to UART converter on the oscilloscope.

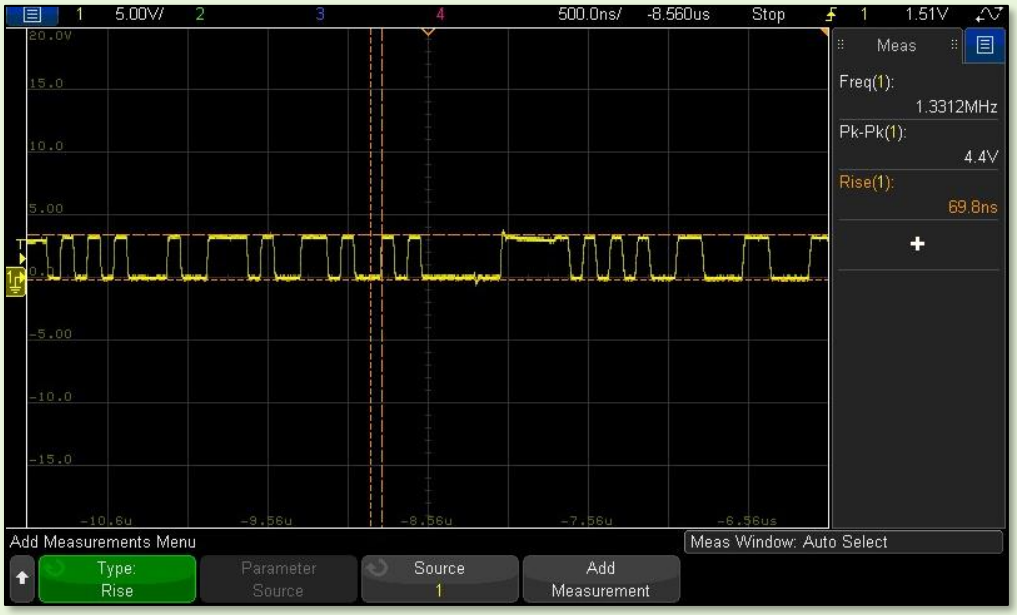


Fig 5: D+

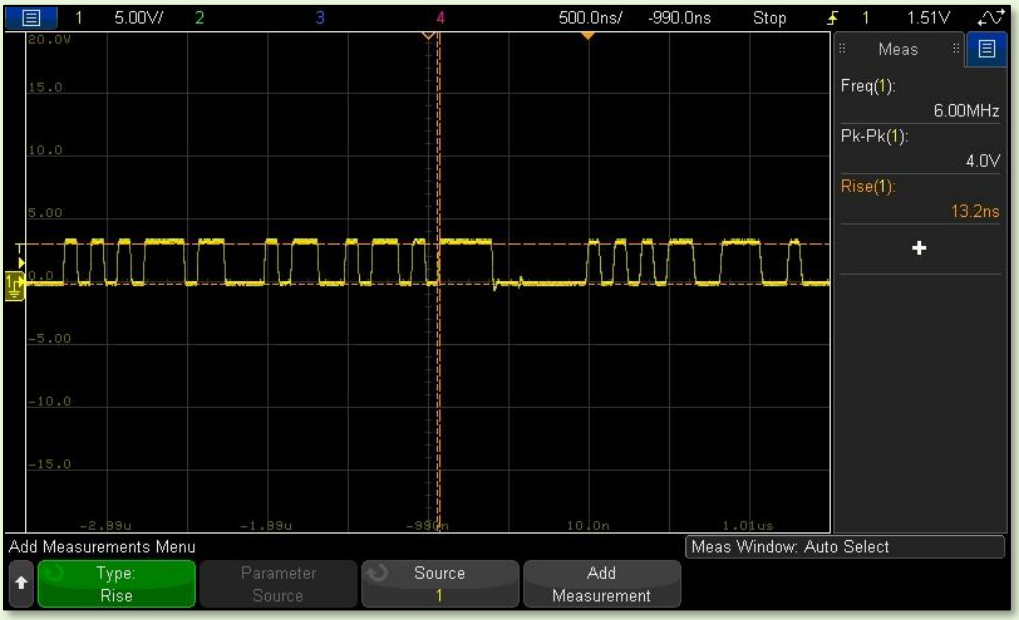


Fig 6: D-

Here, the D+ and D- lines of the CH340 are used for USB data transfer, handling differential signalling necessary for USB communication.

Practical PCB Design and Manufacture

Given below is the scope shot for the SCK indicator LED where I am running a basic Arduino blinky code with a delay of 1 sec each.

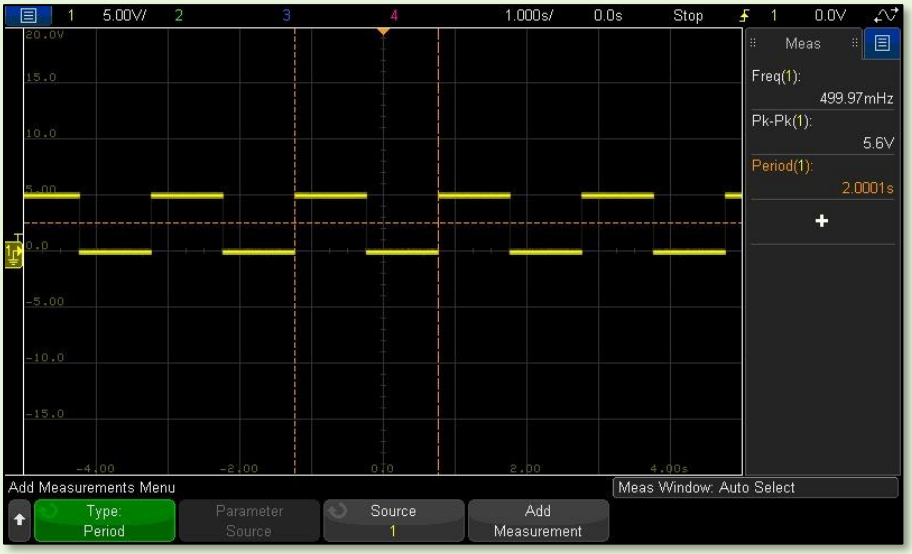


Fig 7: SCK Indicator

Thus, in the Fig 7, we can see the pulse width of 2 secs and a frequency of 500mHz. The SCK indicator on the board represents the Serial Clock line, which is essential for SPI communication synchronization.

Given below is the scope shot for the RX line of the board, where I am communicating between two Arduinos using I2C communication as shown in Fig 1.

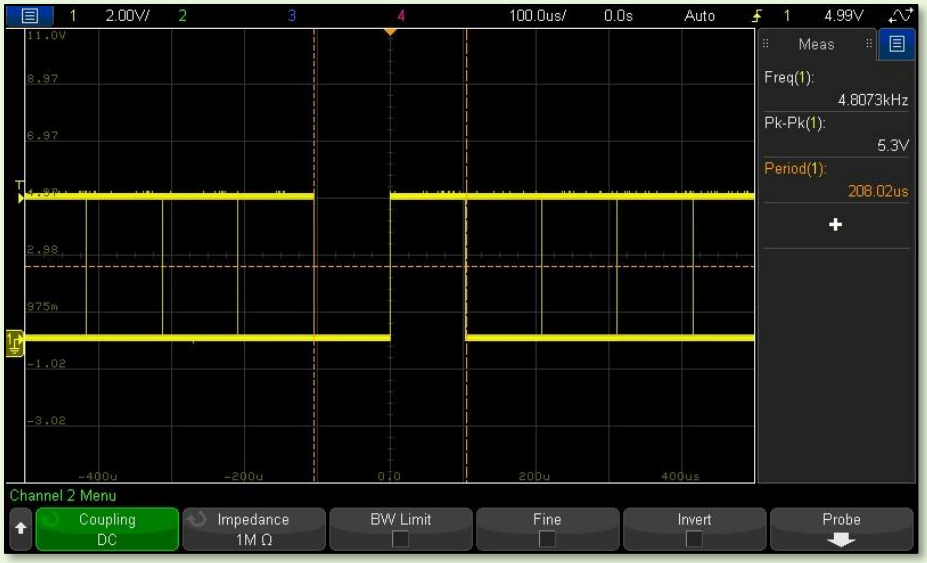


Fig 8: RX line

The RX line on the board is used for receiving serial data, crucial for UART communication between the Arduino and other devices.

Practical PCB Design and Manufacture

☞ *INRUSH CURRENT MEASUREMENT*

Inrush current refers to the initial surge of current that occurs when an Arduino Uno is first powered on. This sudden spike in current happens because the capacitors on the board are initially uncharged and behave like short circuits for a moment. As power is applied, these capacitors charge rapidly, drawing a large amount of current from the power supply. This phenomenon is brief but can be significantly higher than the normal operating current. Managing inrush current is important to prevent potential damage to the power supply and ensure the longevity and reliability of the Arduino Uno.

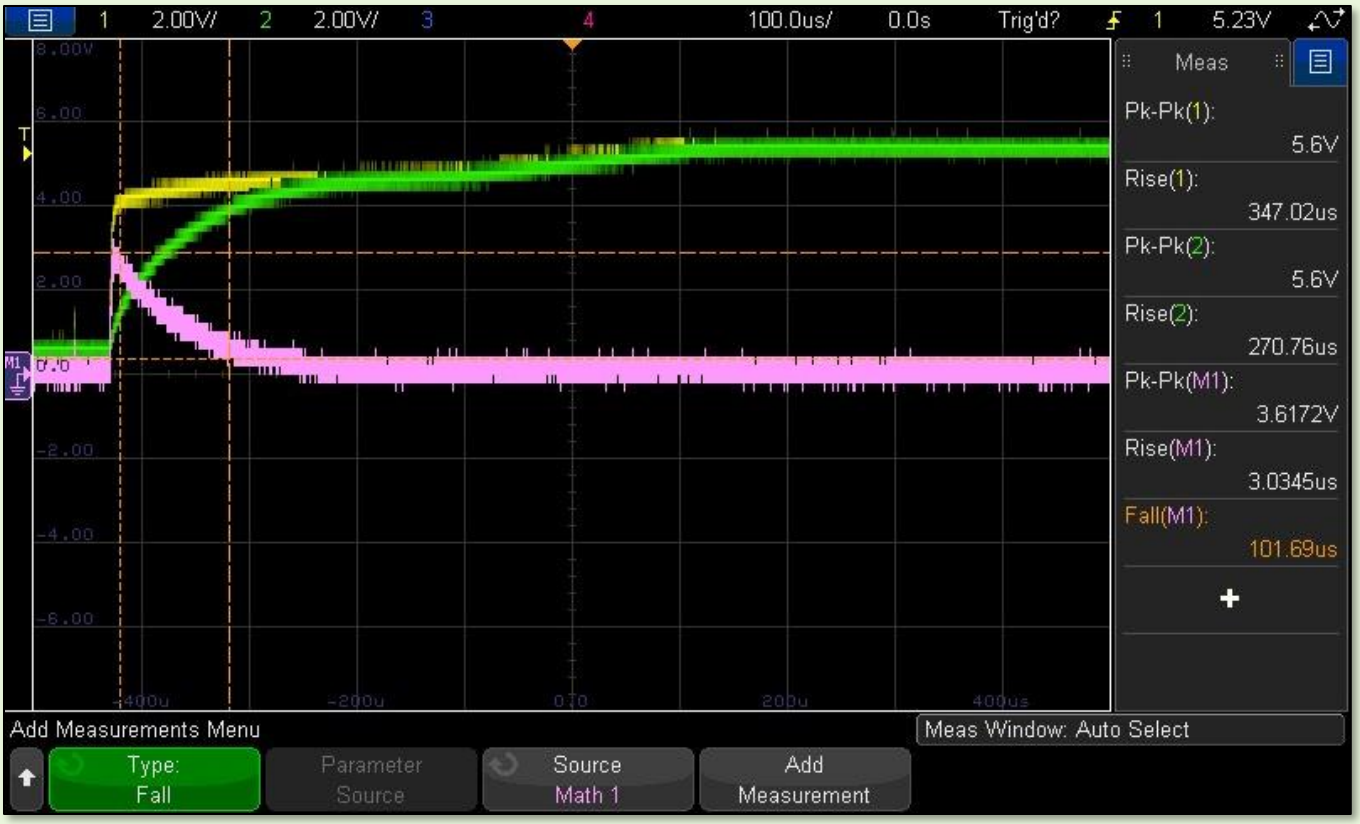


Fig 9: In Rush Current

The voltage across the sense resistor on Channel M1 during the first boot up would be measured by both Channel 1 and Channel 2, which represent the high and low sides of the current sense resistors, respectively, to provide an indication of the current drawn by the capacitors of the Golden Arduino Board. From this measurement, the inrush current value that follows was determined as follows:

I (Inrush Current) = V / R = 2.5V/0.5 Ohm = 5A.

Practical PCB Design and Manufacture

I wrote the code to activate pin 13 on the shield shown in Fig 10 below, which served as the trigger for the scope. After this, I activated D12 and observed the noise generated by its switching. Subsequently, I had D12 and D11 switch at the same time, and then D12, D11, and D10 all switched simultaneously. I used the port command to make them switch immediately after one another so that I could see everything happening on one screen at once.

The code which was written was as follows:

```
void setup()
{
  DDRB = B00111111;
  pinMode(7, OUTPUT);
  digitalWrite(7, LOW);
}

void loop()
{
  PORTB = B00111101;
  delayMicroseconds(4);
  PORTB = B00000001;
  delay(1);
  digitalWrite(7, HIGH);
  delayMicroseconds(400);
  digitalWrite(7, LOW);
  delay(10);
}
```

The provided Arduino code manipulates several digital pins with precise timing for controlling devices like LEDs or motors. In the setup() function, it configures pins 8 to 13 as outputs and sets pin 7 low. The loop() function rapidly toggles these pins: it first sets a specific pattern on pins 8 to 13, holds it for 4 microseconds, then changes the pattern, holding it for 1 millisecond. Pin 7 is toggled from low to high for 400 microseconds and then back to low, with a final delay of 10 milliseconds before the loop restarts. This pattern creates precise pulse sequences for timing-sensitive tasks.

Practical PCB Design and Manufacture

The scope shots given below are taken between the commercial Arduino and the Golden Arduino for an external shield as shown below,

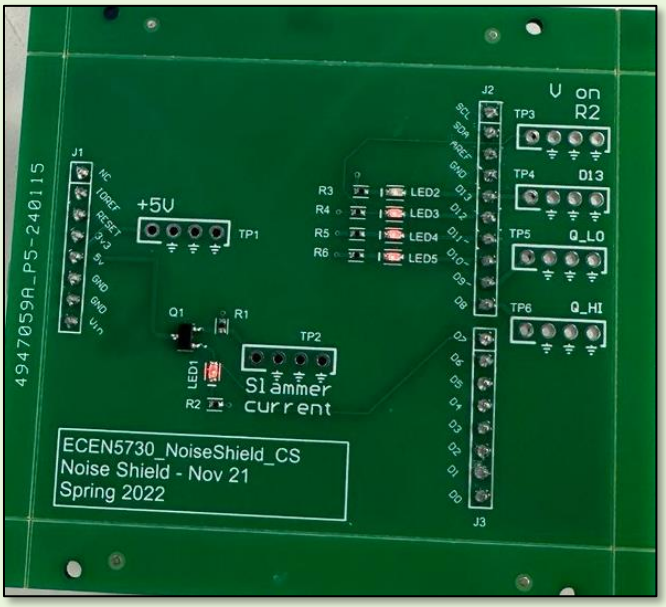


Fig 10: External Shield

So, for the commercial Arduino, the analysis across the 5V test point and the slammer circuit is as follows:

Fig 11: Slammer across 5V

Practical PCB Design and Manufacture

For the commercial Arduino vs the Golden Arduino, the analysis across the 5V test point and the slammer circuit is as follows:

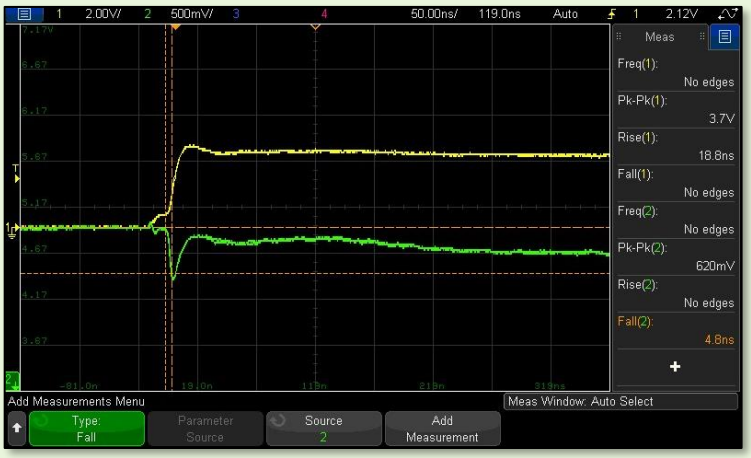


Fig 12: Commercial Arduino Rise

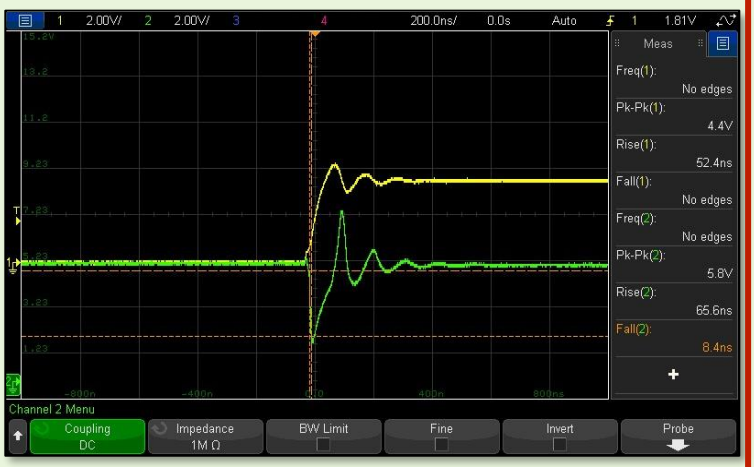


Fig 13: Golden Arduino Rise

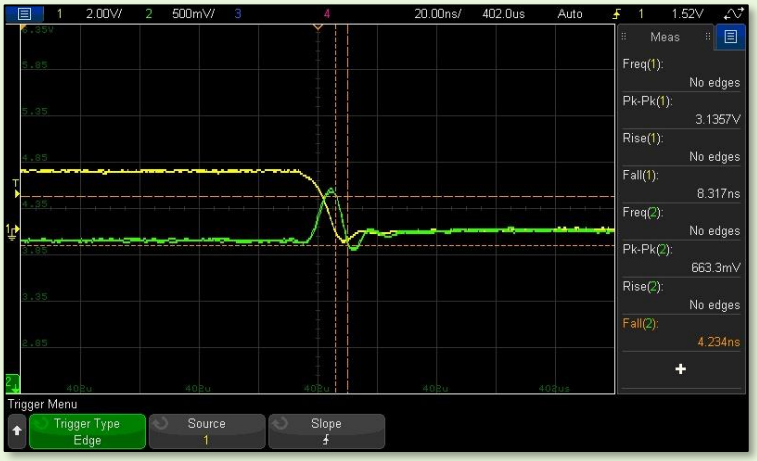


Fig 14: Commercial Arduino Fall

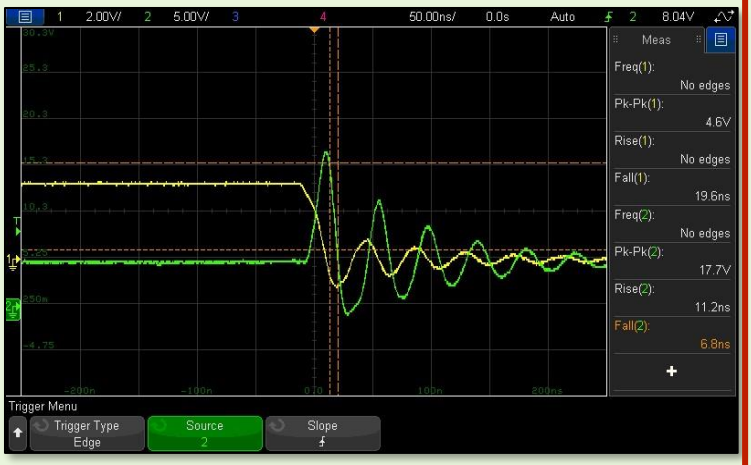


Fig 15: Golden Arduino Fall

The table below represents the rise time and fall time for noise generated across the above given test points.

Arduino	Rise Time	Fall Time
Commercial Arduino	18.8ns	4.234ns
Golden Arduino	52.4ns	6.8ns

Thus, we can see a significant increase in the rise time and the fall time across the Golden Arduino.

Practical PCB Design and Manufacture

For the commercial Arduino vs the Golden Arduino, the analysis across the D13 (i.e., across the 63Ω resistor) test point is as follows:

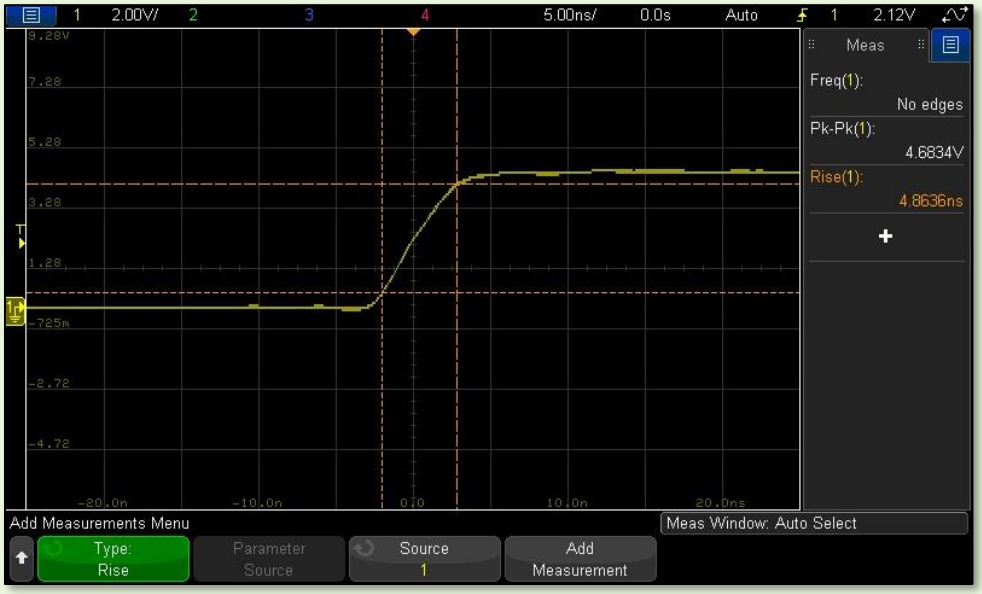


Fig 16: Commercial Arduino Rise

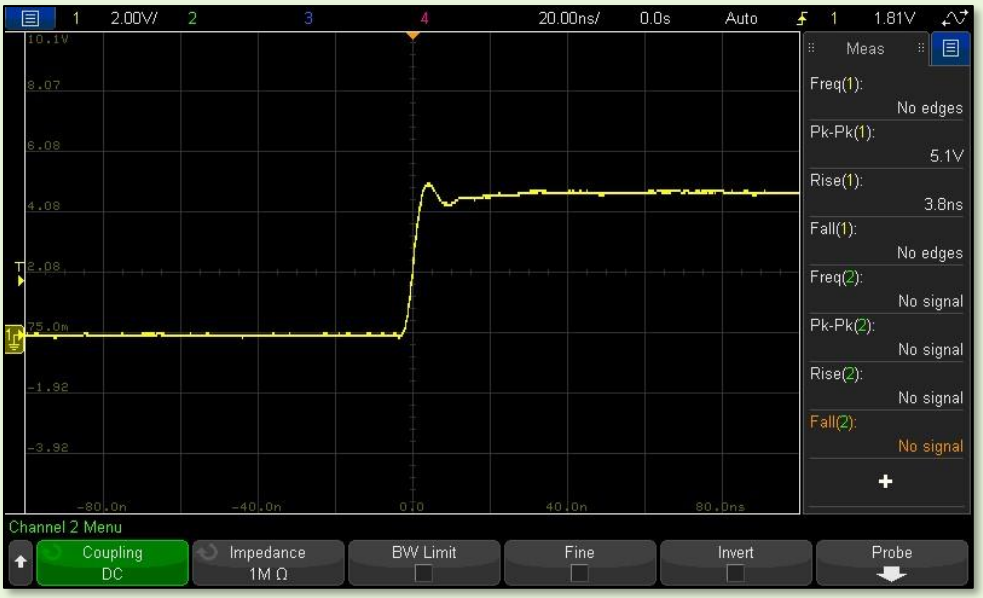


Fig 17: Golden Arduino Rise

Arduino	Rise Time
Commercial Arduino	4.86ns
Golden Arduino	3.8ns

Practical PCB Design and Manufacture

For the commercial Arduino vs the Golden Arduino, the analysis across the D13 test point and the TP3 is as follows:

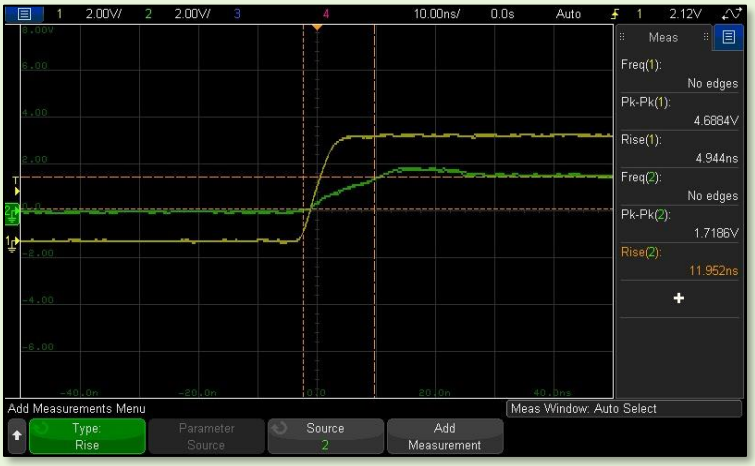


Fig 18: Commercial Arduino Rise

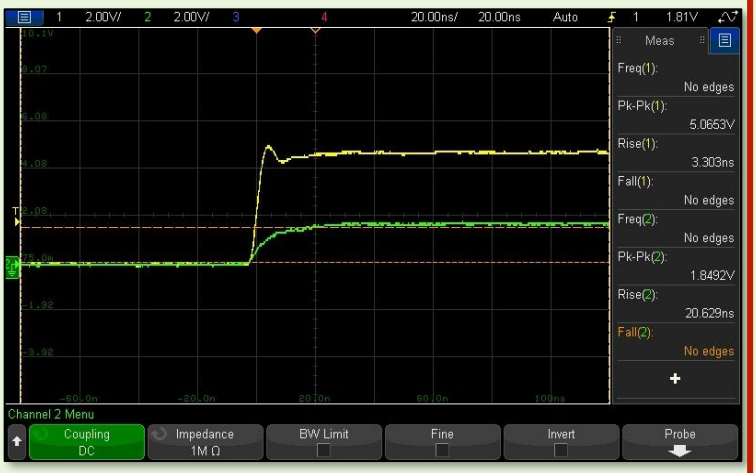


Fig 19: Golden Arduino Rise

Fig 20: Commercial Arduino Fall

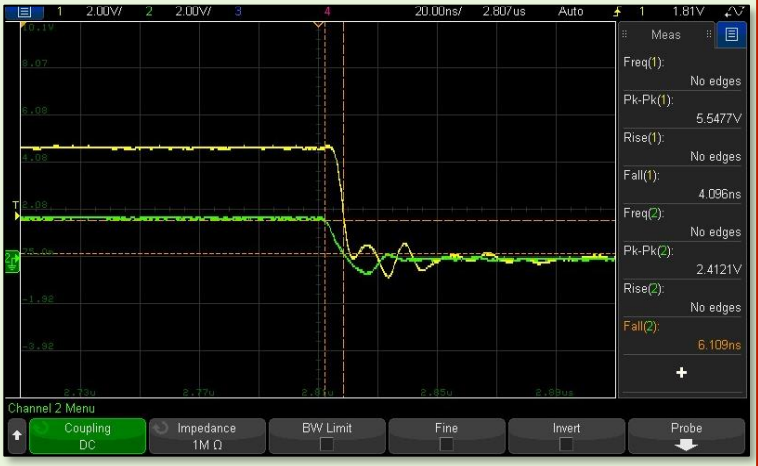


Fig 21: Golden Arduino Fall

The table below represents the rise time and fall time for noise generated across the above given test points.

Arduino	Rise Time	Fall Time
Commercial Arduino	4.944ns	4.083ns
Golden Arduino	3.303ns	4.096ns

Thus, we can see a significant change in the rise time and the fall time across the Golden Arduino.

Practical PCB Design and Manufacture

For the commercial Arduino vs the Golden Arduino, the analysis across the D13 test point and the Quiet Low-test point is as follows:

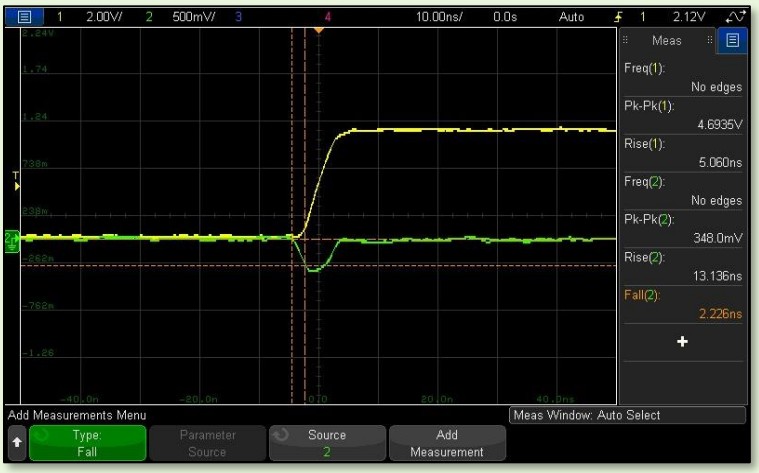


Fig 22: Commercial Arduino Rise

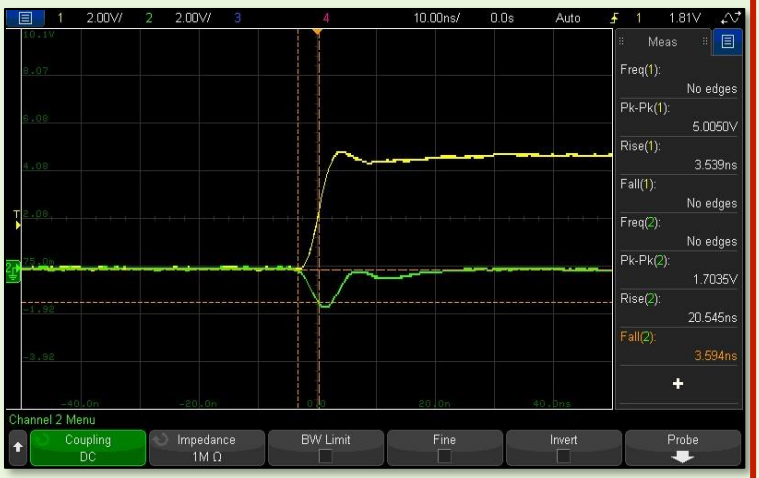


Fig 23: Golden Arduino Rise

Fig 24: Commercial Arduino Fall

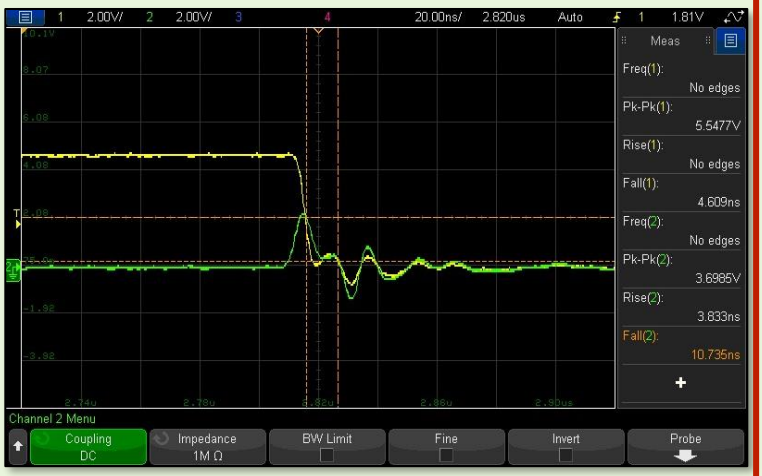


Fig 25: Golden Arduino Fall

The table below represents the rise time and fall time for noise generated across the above given test points.

Arduino	Rise Time	Fall Time
Commercial Arduino	5.06ns	5.743ns
Golden Arduino	3.539ns	4.609ns

Thus, we can see a significant change in the rise time and the fall time across the Golden Arduino.

Practical PCB Design and Manufacture

For the commercial Arduino vs the Golden Arduino, the analysis across the D13 test point and the Quiet High-test point is as follows:

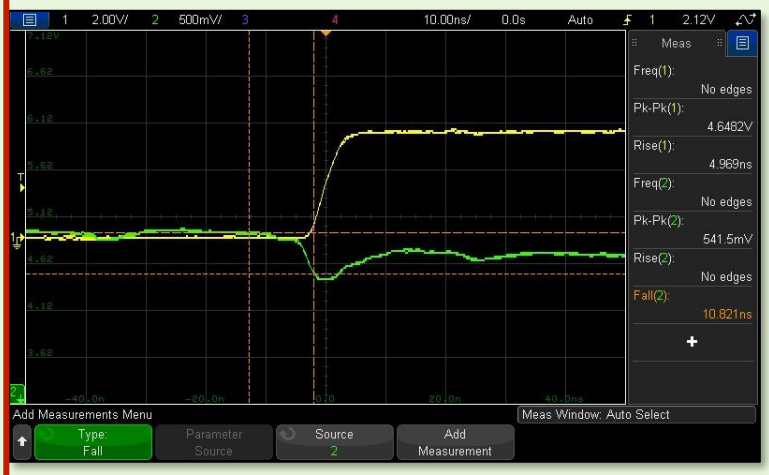


Fig 26: Commercial Arduino Rise

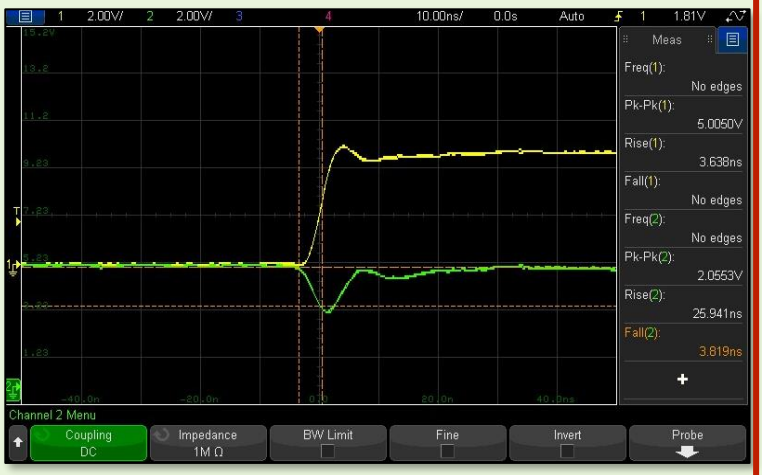


Fig 27: Golden Arduino Rise

Fig 28: Commercial Arduino Fall

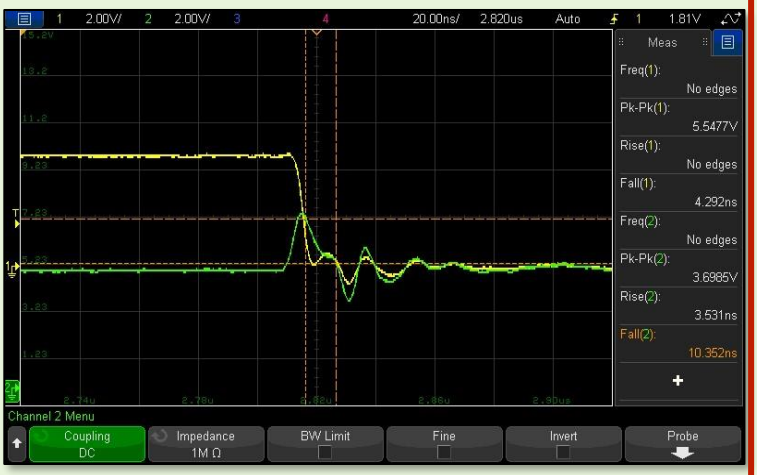


Fig 29: Golden Arduino Fall

The table below represents the rise time and fall time for noise generated across the above given test points.

Arduino	Rise Time	Fall Time
Commercial Arduino	4.969ns	5.509ns
Golden Arduino	3.638ns	4.292ns

Thus, we can see a significant change in the rise time and the fall time across the Golden Arduino.

Practical PCB Design and Manufacture

For the commercial Arduino vs the Golden Arduino, I measured the near field emissions from the circuit board using a 10x probe configured as a pick-up loop and identified the position under the board where the probe picked up the maximum RF noise.

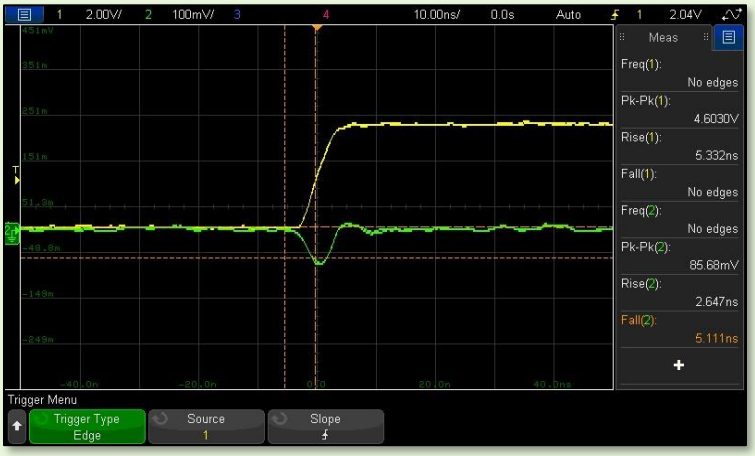


Fig 30: Commercial Arduino Rise

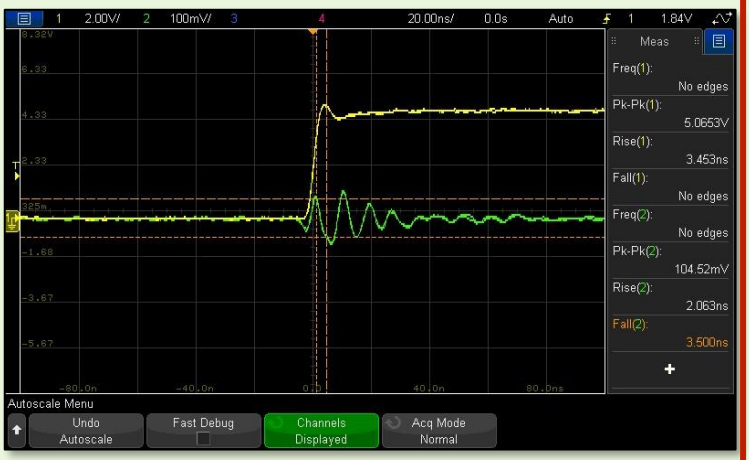


Fig 31: Golden Arduino Rise

Fig 32: Commercial Arduino Fall

Fig 33: Golden Arduino Fall

The table below represents the rise time and fall time for noise generated across the above given test points.

Arduino	Rise Time	Fall Time
Commercial Arduino	5.332ns	3.453ns
Golden Arduino	5.643ns	3.992ns

Thus, we can see a significant change in the rise time and the fall time across the Golden Arduino.

Practical PCB Design and Manufacture

☞ *BEST DESIGN PRACTICES*

I've added a lot of cutting-edge methods to my board 3, which compares successful and ineffective ways, to improve its dependability and performance:

1. **Regulated Power Supply**: Ensuring that the power supply is meticulously regulated minimizes voltage fluctuations, providing a stable current to the board.

2. **Use of Ferrite Beads:** Incorporating ferrite beads in the design can help suppress high-frequency noise in power lines, enhancing the overall electromagnetic compatibility of the board.

3. **Short and Shielded Traces:** Minimizing the length of trace routes between critical components on the PCB reduces electromagnetic interference and potential signal degradation.

4. **Consistent Ground Plane:** Incorporating a comprehensive ground plane helps in reducing ground loop effects and stabilizes the reference voltage across the board.

5. **Strategically Placed Test Points:** Positioning test points for key signals and power outputs enhances the ease of diagnostics and troubleshooting.

6. **Thoughtful Component Layout:** Spacing components adequately avoids physical and electrical overlap, thereby reducing the risk of shorts and allowing for heat dissipation.

7. **Decoupling Capacitors Near ICs:** Adding decoupling capacitors close to the microcontroller and other ICs stabilizes their power supply and smooths out transient voltage spikes.

8. **EMI Considerations:** Employing shielding techniques around critical, high-frequency components to prevent electromagnetic interference from compromising the board's performance.

9. **Thermal Management Features:** Incorporating features such as thermal vias or heatsinks, especially near components that generate significant heat, aids in maintaining optimal operating temperatures.

10. **Robust Connector Choices:** Using durable connectors suited to frequent interaction ensures long-lasting connections and enhances the overall ruggedness of the board.

Practical PCB Design and Manufacture

☞ *MISTAKES MADE*

I encountered one hard error on the board. As shown in the part of the schematic below, I found out that the orientation of my header pins did not match with those of the Arduino. So, I was not able to place my shield directly above my Arduino. **No soft errors were encountered.**

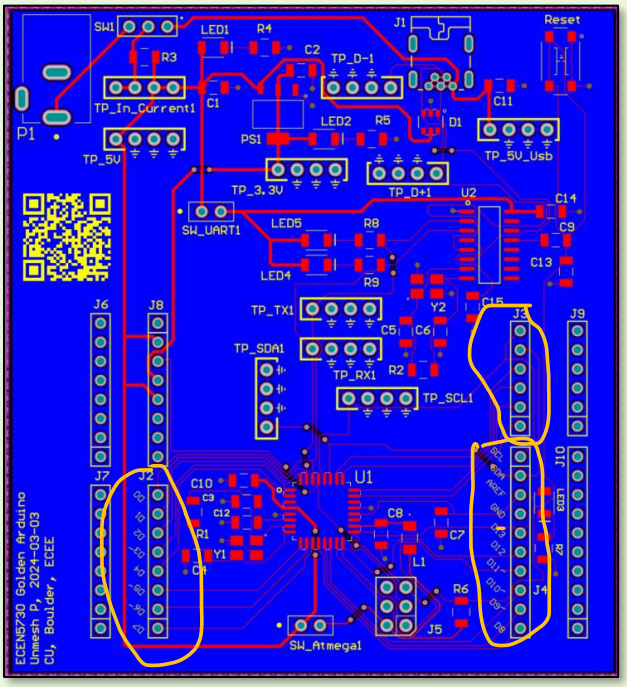


Fig 34: Wrongly placed headers

I solved this problem by connecting the shield using wires as shown in Fig 35 below. Also, I placed the shield as close as possible to the Arduino as shown in Fig 36 to reduce as much noise as possible.

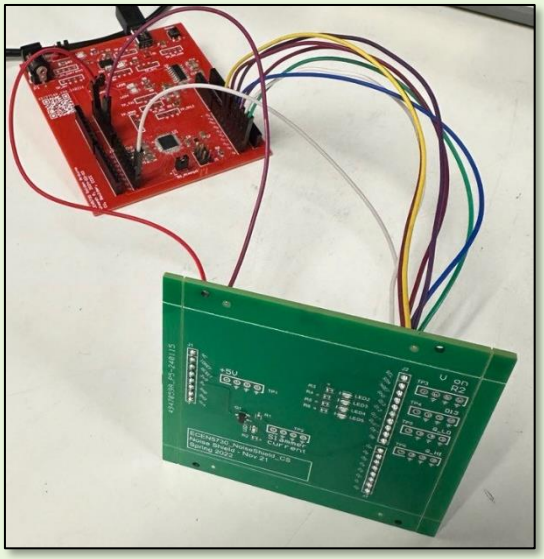


Fig 35: Wired connections

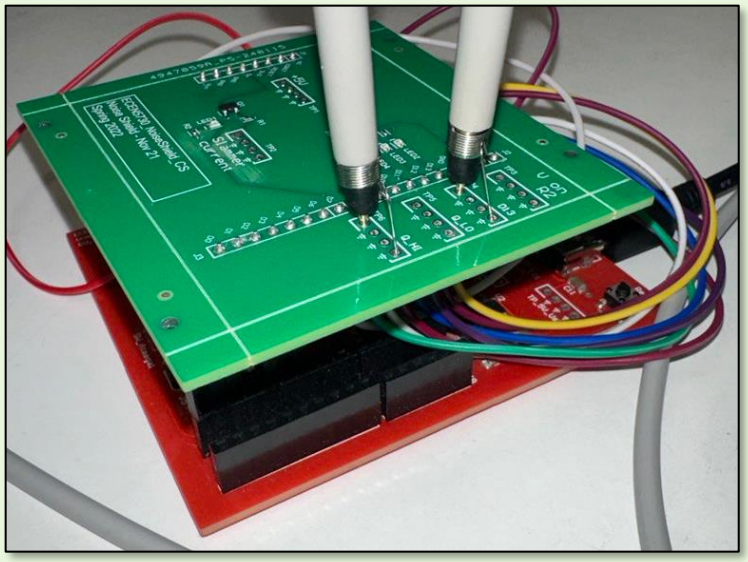


Fig 36: Shield placed near to board

Practical PCB Design and Manufacture

☞ *ANALYSIS AND EFFECTIVE LEARNINGS*

1. Understanding the value of a well-designed layout and the reasons large-scale commercial designs shouldn't be regarded as flawless was made easier thanks to this design endeavor.
2. Reviewing the design is important before submitting.
3. Simple design ideas like addressing loop inductance, placing decoupling capacitors in the right places, and using a common return ground plane can all have a significant effect on the circuit's quality.
4. When designing the PCB, keep in mind that the Near Field Emission effect can generate Far Field Emission, which can interfere with EMC regulations.
5. Ground bounce noise can be reduced adding a ground-return plane that remains under all signal traces and putting forth real effort to minimize the length of cross-under connections to the ground plane.
6. If there are any cross-under, I would minimize their number to lessen noise.
7. When soldering important components like resonators, exercise extra caution and pay attention to their footprints.

☞ *CONCLUSION*

In conclusion, the Arduino Uno's performance, dependability, and longevity are greatly improved by appropriate PCB design. A continuous return path placed below the signal traces aids in the reduction of inductive crosstalk. Signal integrity must be preserved by using the bottom layer of a two-layer PCB as a continuous ground return and by avoiding using separate wires for return pathways. It is crucial to comprehend the consequences of Near Field Emissions since they might cause problems with Far Field Emissions, which affect EMC compliance. Low-inductive decoupling capacitors placed adjacent to the IC power pins reduce switching noise, while ferrite beads are used to filter noise at sensitive IC pins. Ground vias can be positioned right beneath ICs to maximize layout efficiency and save space. The overall functionality and debugging efficiency of a board can be improved by strategically placing test points and designing wide-spaced signal routes.