Project Report

ATtiny 45 Voltmeter with ADC Oversampling

Submitted By,

Vishesh Yadav(2021UEC2600)

Electronics and Communication Engineering

Acknowledgement

I extend my deepest gratitude to Prof. Dhananjay V. Gadre, Associate Professor, ECE Division at Netaji Subhas University of Technology. His constant support and guidance are what led to the completion of this project. CEDT gave me the optimal environment to learn, research, and implement my newly acquired knowledge.
My thanks and appreciation also goes to my friends and seniors at the centre who have helped me out willingly throughout the course of this project.
I would also like to express gratitude to my family who has always supported me in my endeavours.

Contents

1. Introduction

The **ATtiny 45** is a low-power CMOS 8-bit microcontroller based on the AVR-enhanced RISC architecture. It offers an Analog to Digital Converter with a 10-bit resolution. In most cases, 10-bit resolution is sufficient, but higher accuracy is sometimes desired.

Unique signal processing technique **Oversampling and Decimation** is used to achieve higher resolution.ADC oversampling is a technique employed to enhance the resolution and accuracy of ADC measurement by sampling the input signal at a higher rate than the Nyquist frequency.

The ATtiny45 microcontroller supports oversampling in its ADC module, providing developers with the flexibility to implement higher-resolution conversions without the need for external components.

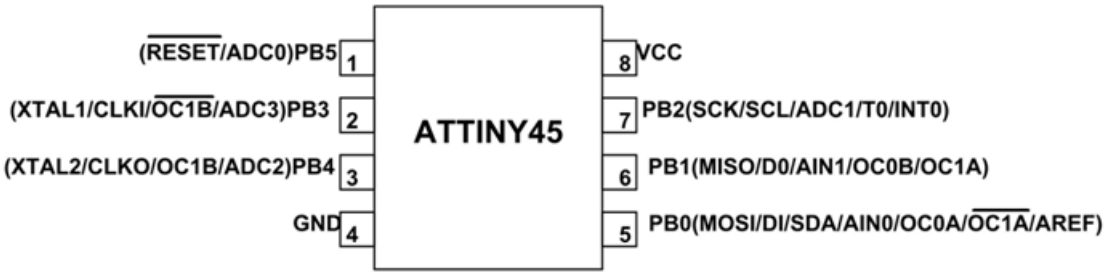
In this report, we will delve into the concept of ADC oversampling in the context of the ATtiny45 microcontroller-based voltmeter. We will explore the principles behind ADC oversampling and its advantages in terms of increased resolution and reduced noise.

2. ATtiny 45

The ATtiny45 is a microcontroller from the AVR family of microcontrollers developed by Atmel, which is now part of Microchip Technology. It is a low-power, high-performance 8-bit microcontroller that offers a compact and cost-effective solution for various embedded systems and projects.

Here are some key features and specifications of the ATtiny45:

- Architecture: The ATtiny45 is based on the AVR RISC (Reduced Instruction Set Computer) architecture, which allows for efficient and fast execution of instructions.
- Flash Memory: The ATtiny45 has 4 kilobytes (KB) of in-system programmable Flash memory, which is used to store the program code.
- RAM and EEPROM: It includes 256 bytes of SRAM (Static Random Access Memory) for storing variables and temporary data. Additionally, it has 256 bytes of non-volatile EEPROM (Electrically Erasable Programmable Read-Only Memory) for storing data that needs to be retained even when power is removed.
- GPIO Pins: The microcontroller features 6 general-purpose I/O (GPIO) pins, which can be configured as inputs or outputs to interface with external components such as sensors, actuators, or other integrated circuits.
- Low Power Consumption: The microcontroller is designed to operate at low power, making it suitable for battery-powered or energy-efficient applications. It offers multiple sleep modes and power-saving features to minimize power consumption.



2.1 Voltage Measurement

The ADC of ATtiny 45 is used to measure the voltage of an external source. The ADC offers a 10-bit resolution. You also need to select a reference voltage source for the ADC. This can either be a source voltage, an external voltage or an internal reference voltage.

Internal reference voltages are preferred over external voltage sources as a stable source is required to compare the ADC values.ATtiny offers two internal reference voltages **1.1V** and **2.56V**. 1.1V is the most stable of the two but provides a low accuracy, so in this project, I have used 2.56V as my reference voltage.

The voltage reference can be set using the 'analogReference()' function where **'INTERNAL1V1'** is used to set reference to 1.1volts and **'INTERNAL2V56'** is used to set reference to 2.56volts

To calculate the voltage from the ADC value this particular formula is used:

$$\textbf{Voltage} = (\textbf{ADCValue}/2^{resolution})\textbf{*}\mathbf{V}_{ref},$$

where ADCValue is the raw ADC value, resolution is the ADC resolution (10 bits in this case), and Vref is the reference voltage used.

2.2 Eagle Schematic

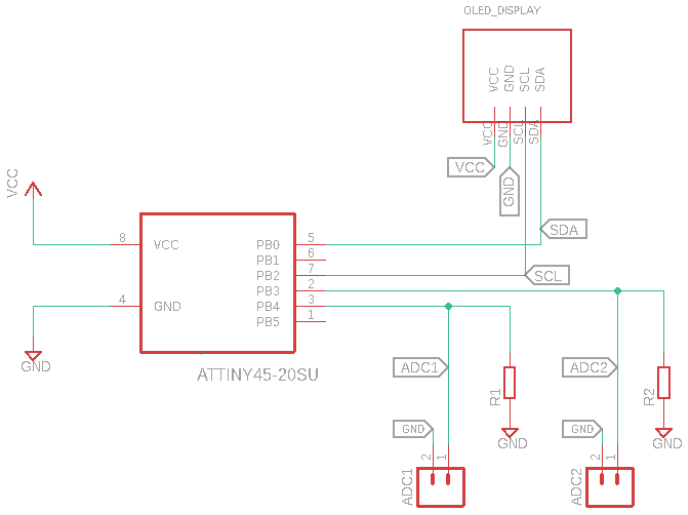


Figure 1: ATtiny45 Voltmeter

Description

The 10-bit ADC of pin PB4 and PB3 are ADC1 and ADC2 respectively which are used to take input of measuring voltage. Each ADC input is backed with a pull-down resistor which prevents ADC from capturing floating values by providing 'logic 0' when kept idle. Further, the computation is done and the result is displayed on an OLED display through I2C.

2.3 ISP Programmer

An ISP (In-System Programming) programmer is a device or tool used to program or reprogram microcontrollers or other programmable devices directly on the target hardware. It allows you to update the firmware or software of a microcontroller or integrated circuit (IC) without removing it from the circuit board.

ISP programmers typically connect to the target device using specific programming interfaces, such as JTAG (Joint Test Action Group), SWD (Serial Wire Debug), SPI (Serial Peripheral Interface), or I2C (Inter-Integrated Circuit). These interfaces provide a way to communicate with the microcontroller and transfer the programming data.

ISP programmers are commonly used in various fields, including embedded systems development, electronics manufacturing, and device repair. They offer flexibility and convenience by allowing you to program or update microcontrollers in-circuit, saving time and effort compared to removing and reinserting chips for programming.

For this project, I have used Arduino Uno as an ISP Programmer to program ATtiny45.This can be done by leveraging the ArduinoISP sketch provided in the Arduino IDE. The figure below shows the connections to be made.

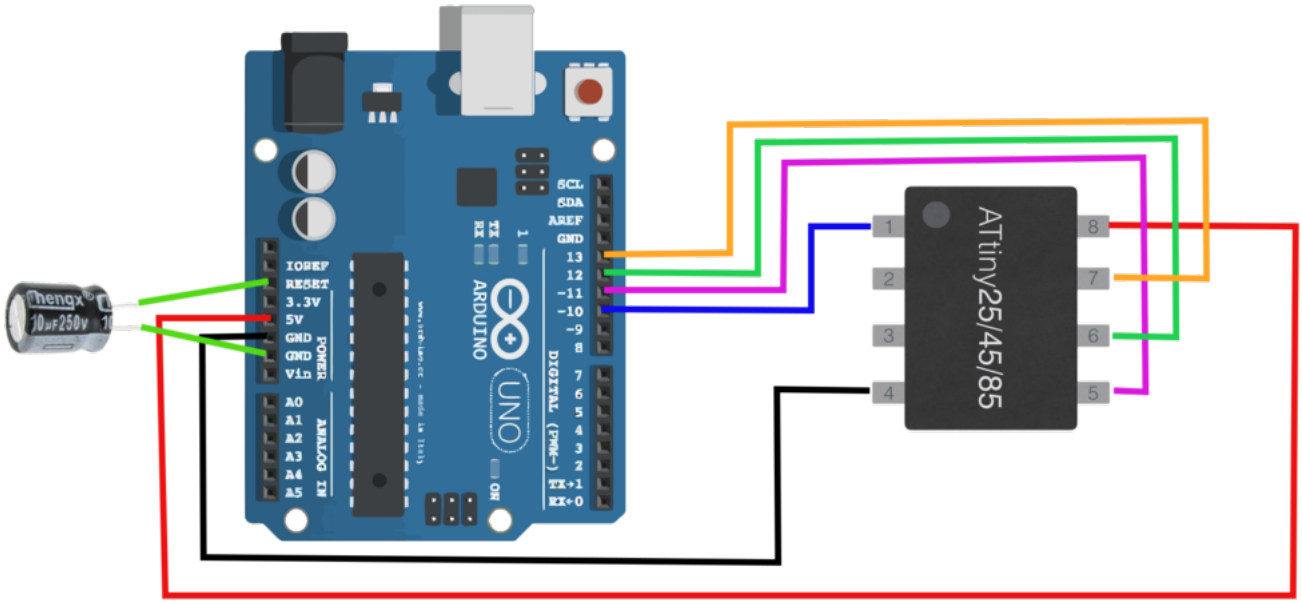


Figure 2: Arduino Uno as an ISP Programmer

3. Oversampling and Decimation

This technique requires a higher amount of samples and these can be achieved by oversampling the signal. For each additional bit of resolution, the signal must be oversampled four times. To get a better representation of the signal it is necessary to oversample this much because a larger amount of samples gives a better representation of the input signal when averaged.

On the other hand, decimation is the process of reducing the sampling rate of a signal by discarding some samples while preserving the essential characteristics of the signal. Decimation is often used after oversampling to restore the signal to its original sampling rate or to reduce computational requirements in certain applications.

$\mathrm{F}_{oversampling} = 4^n * F_{nyquist}$

3.1 Basic Code for Oversampling

```
float Oversampling(uint8_t x) {
  byte NOS = 2;\\Number of Samples (NOS)
  long readingSum = 0;
  for (uint8_t i = 0; i < NOS; i++) { //Loop for Averaging
    long inner = 0;
  for (uint16_t j = 0; j < 4^n ; j++) { //Oversampling loop
      inner += analogRead(x);
      delayMicroseconds(150);
    }
    long reading = ((inner) >> 2); // Decimation
    readingSum += reading;
  }
  float average = readingSum / NOS;
  return average;
}
```

Description

The oversampling function contains two loops, loop 1 is for averaging which is defined to take an average of two samples and the inner loop is the oversampling loop where 'n' defines the number of desired bits.

4. Calibration of ADC

4.1 Need for Calibration

The total error of the actual ADC comes from more than just quantization error, known as gain error and offset error.

For most applications, the ADC needs no calibration when used in the single-ended conversion. However, when using differential conversion the situation changes, especially with high-gain settings. Minor process variations are scaled. Therefore calibration in ADC is required.

4.2 Offset Error

The offset error, also known as zero-scale error or DC offset error, refers to the deviation from zero output when the input signal is at or near zero. In other words, it represents an error in the ADC's ability to accurately represent zero input as a zero output. The offset error is typically specified in LSB (Least Significant Bit) or in volts.

4.3 Gain Error

The gain error refers to the deviation from the ideal gain or slope of the transfer function of the ADC. It represents an error in the ADC's ability to accurately amplify and scale the input signal. The gain error is usually expressed as a percentage or in LSB or volts.

To compensate for the gain error, you can use calibration techniques or adjust the output of the ADC by applying a correction factor to the measured values.

Both offset error and gain error contribute to the overall accuracy of an ADC. They can affect the linearity, precision, and resolution of the converted digital output. It's important to consider these errors when designing or using ADCs in applications that require high accuracy.

The calibration of the ATtiny-based voltmeter was done with respect to $6\frac{1}{2}$ digit DMM(Digital Multi Meter) and then comparing it with the linear 'Y=X' line. The best-fit line for calibration was found using the Linear Regression Model to overcome the ADC gain and offset error.

5. Oversampling 10-bit to 12-bit

5.1 Code

```
float Oversample(uint8_t x) {
  byte NSO = 2;\\Number of Samples(NOS)
  long readingSum = 0;
  for (uint8_t i = 0; i < NSO; i++) {//Averaging loop
    long inner = 0;
    for (uint16_t j = 0; j < 16 ; j++) {//Oversampling loop
      inner += analogRead(x);
      delayMicroseconds(150);
    }
    long reading = ((inner) >> 2);
    readingSum += reading;
  }
  float avg = readingSum / NSO;
  float vout = (avg * (0.00059514885));          //Calibration factors
  return (vout > 0) ? vout + 0.00198888 : 0;

void setup() {
  analogReference(INTERNAL2V56);
  OzOled.init();
}
void loop() {
 //OLED print commands
  OzOled.setCursorXY(1, 5);
  OzOled.printString("CH 1:");
  OzOled.setCursorXY(1, 5);
  OzOled.printNumber((float)Oversample(2), 4);
  delayMicroseconds(150);
  delay(100);
}
```

5.2 Description

The Oversampling of ADC is done to get 2 extra bits so the oversampling loop runs 4^2 times. The reading sum is further decimated by using the right shift operator in this case right shift by 2 places to reduce computational requirements in certain applications.

5.3 Calibration Curve 10-bit to 12-bit

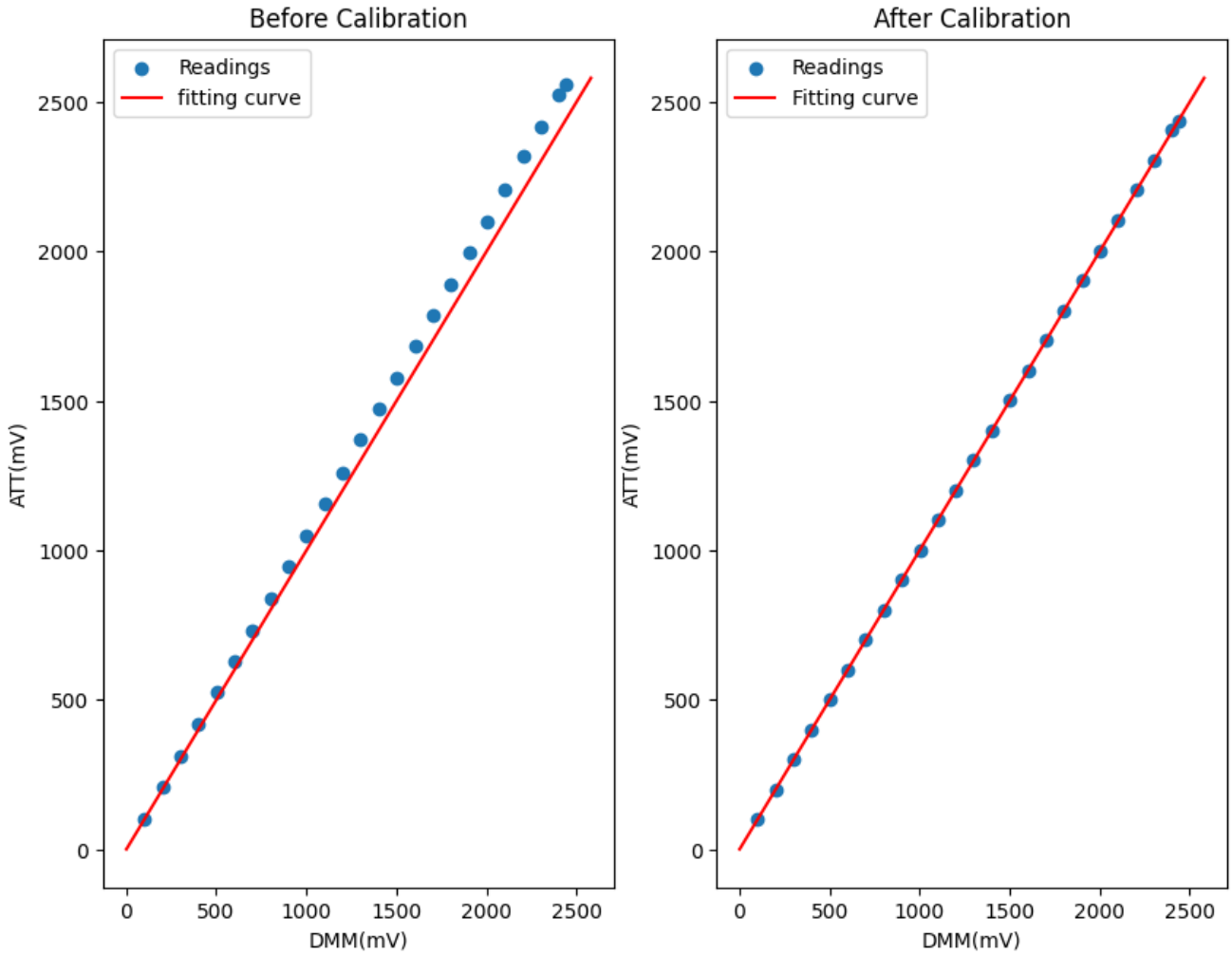


Figure 3: Before and After Calibration curves of 10-bit to 12-bit

5.4 Sampling Rate 10-bit to 12-bit

To find out the sampling rate a pulse was generated with a delay and the time interval when it was high was noted

Code for Generating pulse

```
digitalWrite(PIN,HIGH);
//Code
digitalWrite(PIN,LOW);
delay(100);
```

Sampling rate with 1MHz Clock Frequency

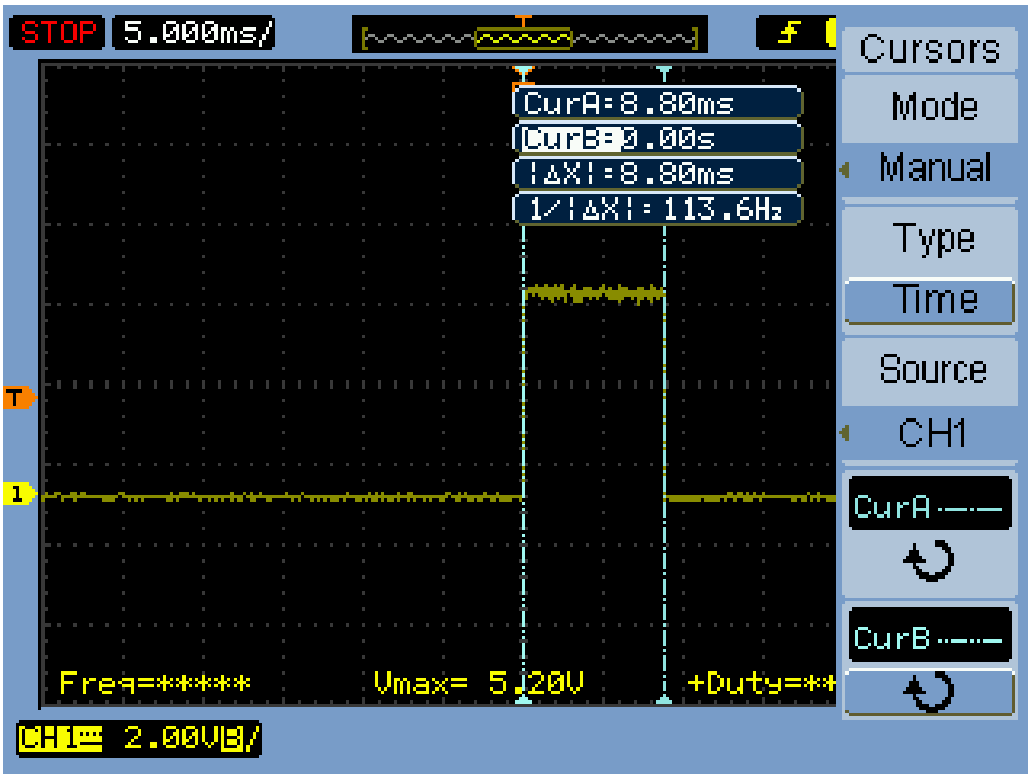


Figure 4: Pulse with 1MHz clock

Sampling rate with 8MHz Clock Frequency

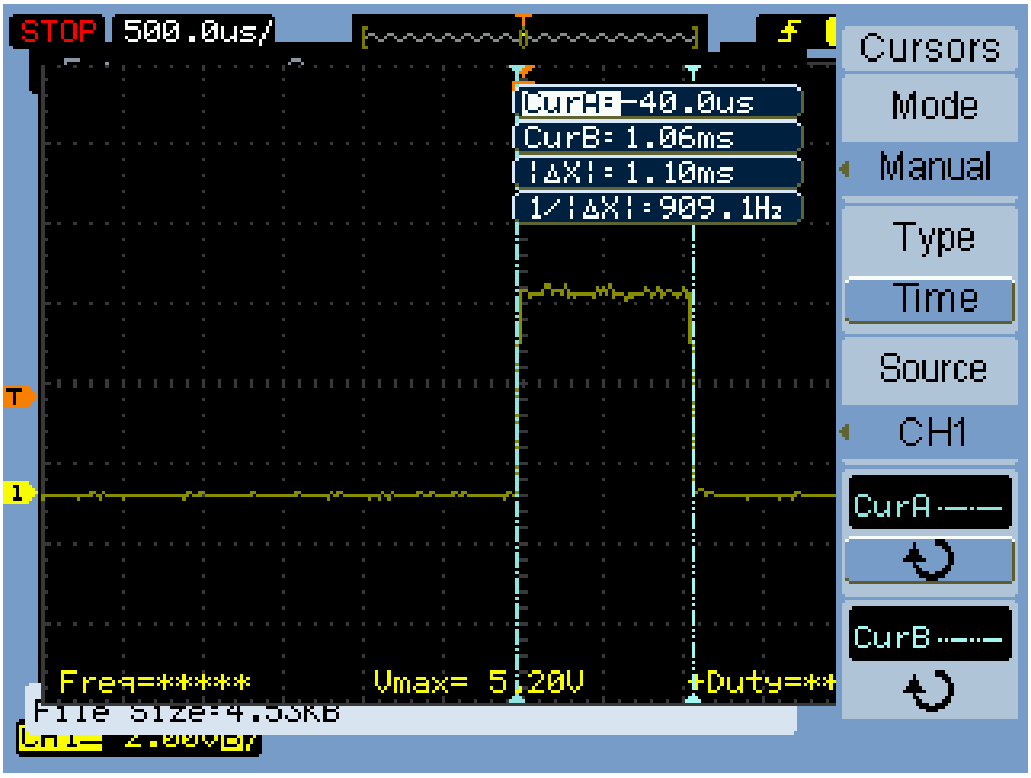


Figure 5: Pulse with 8MHz clock

Description

The High time of the first pulse when the clock frequency was 1MHz is observed to be 156ms, this depicts that the code for oversampling takes 156ms to execute and in this time period four samples are taken. As we increase the Clock frequency from 1MHz to 8MHz we see a drastic change in the performance where the code now takes four samples in 18ms.

6. Oversampling 10-bit to 16-bit

6.1 Code

```
float Oversampling(uint8_t x) {
  byte NOS = 2;\\Number of Samples (NOS)
  long readingSum = 0;
  for (uint8_t i = 0; i < NOS; i++) { //Loop for Averaging
    long inner = 0;
  for (uint16_t j = 0; j < 4096 ; j++) {  //Oversampling loop
      inner += analogRead(x);
      delayMicroseconds(150);
    }
    long reading = ((inner) >> 6); // Decimation
    readingSum += reading;
  }
  float average = readingSum / NOS;

  float vout = (average * (0.0000379371111));
    if ((vout + 0.014130) > 0.1) {
      if ((vout + 0.014130) > 2.4) {
        return vout + 0.01350;
      }
      return vout + 0.012130;
    }
return (vout > 0.001) ? vout + 0.014130 : 0;
}
void setup() {
  analogReference(INTERNAL2V56);
  OzOled.init();
}
  void loop(){
  //OLED print commands
  OzOled.setCursorXY(1, 5);
  OzOled.printString("CH 1:");
  OzOled.setCursorXY(1, 5);
  OzOled.printNumber((float)Oversampling(2), 4);
  delayMicroseconds(150);
  }
```

6.2 Description

The Oversampling of ADC is done to get 6 extra bits so the oversampling loop runs 4^6 times. The reading sum is further decimated by using the right shift operator in this case right shift by 6 places to reduce computational requirements in specific applications. In comparison to conversion to 12-bit oversampling the time taken is more as the oversampling loop runs 256 times more than the oversampling loop of 12bit.

6.3 Calibration Curve 10-bit to 16-bit

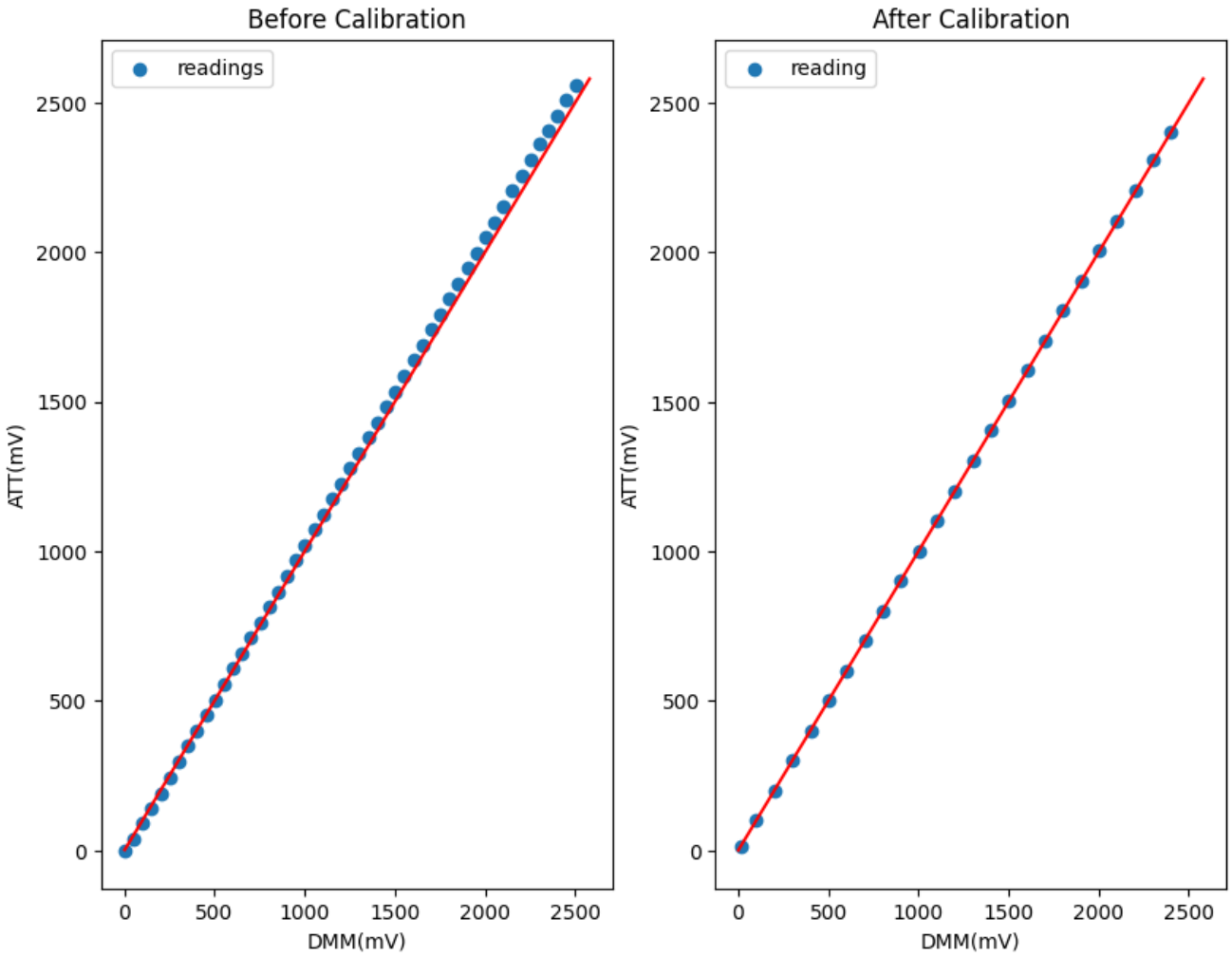


Figure 6: Before and After Calibration curves 10-bit to 16-bit

6.4 Sampling Rate 10-bit to 16-bit

Sampling rate with 1MHz Clock Frequency

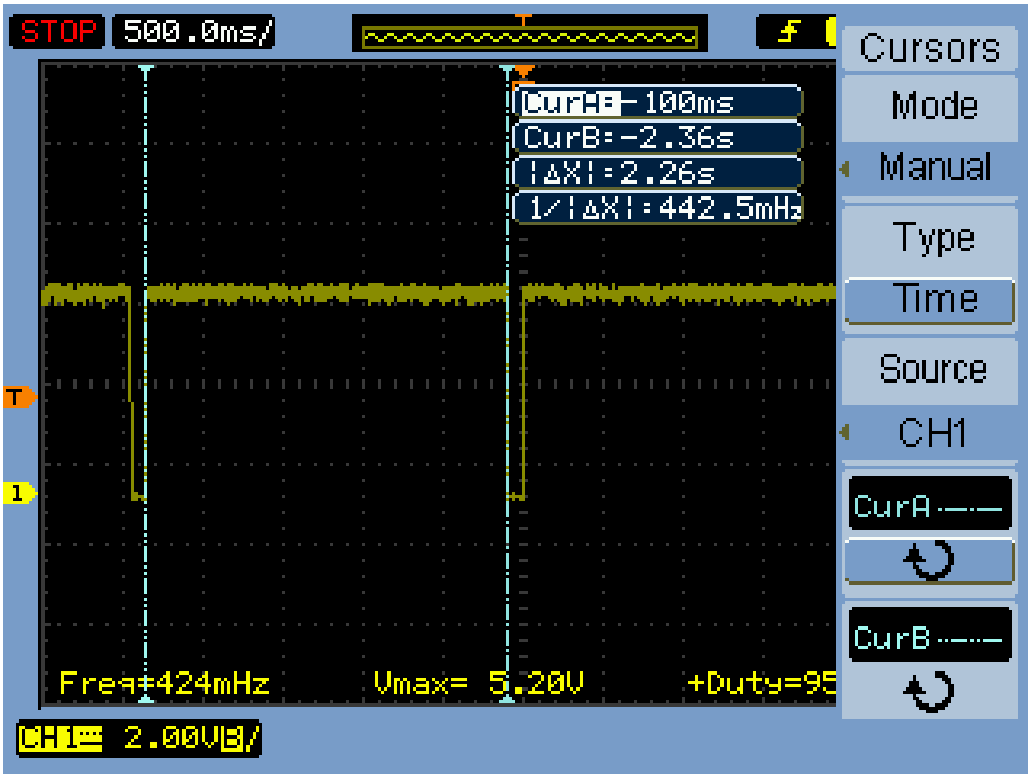


Figure 7: Pulse with 1MHz clock

Sampling rate with 8MHz Clock Frequency

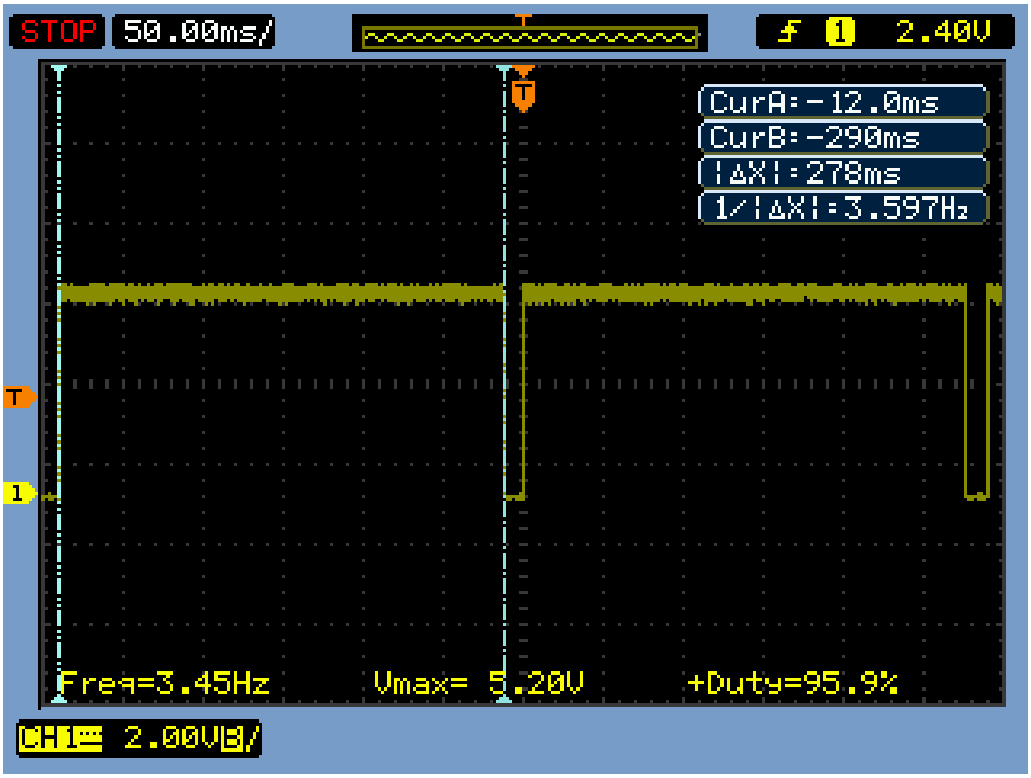


Figure 8: Pulse with 8MHz clock

Description

In 10-bit to 16-bit ADC oversampling, the oversampling loop runs 4^6 times which on its own takes a lot of time to compute so, when the clock frequency is 1MHz it takes four samples in 2040ms but when we increase the clock frequency to 8MHz the times come down to 254ms for four samples which is still a lot but nothing compared to resolution it is offering.

7. References

- ATtiny 45, Datasheet
  https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf
- ATtiny45 Programming
  https://wikifactory.com/+sdgsolutionspace/programming-attiny45
- Enhancing ADC resolution by oversampling
  https://ww1.microchip.com/downloads/en/appnotes/doc8003.pdf
- Characterization and Calibration of the ADC on an AVR
  https://ww1.microchip.com/downloads/en/Appnotes/Atmel-2559-Characterization-and-Calibration-of-the-ADC-on-an-AVR_ApplicationNote_AVR120.pdf

‘