COMP ENG 2DX3

Final Deliverable Report

Instructor: Dr. Doyle, Dr. Haddara, Dr. Athar

Muhammad Haseeb Aslam

aslamm14- 400449291

As a future member of the engineering profession, the student is responsible for performing the required work in an honest manner, without plagiarism and cheating. Submitting this work with my name and student number is a statement and understanding that this work is my own and adheres to the Academic Integrity Policy of McMaster University and the Code of Conduct of the Professional Engineers of Ontario. Submitted by [Muhammad Haseeb Aslam, 400449291]

Device Overview

A - Features

- MSP432E401Y SimpleLink Microcontroller
  - 32-bit Cortex M4F CPU
  - Operating Voltage: 2.5 – 5 V
  - 256 Kb of SRAM
  - 1024 Kb of flash memory
  - Bus speed configurable up to 120 MHz (default: 16MHz)
  - 8 UART and 10 I2C Modules
- VL531LX Time of Flight sensor
  - Maximum Distance: 4m
  - Operating Voltage: 2.6 – 3.5 V
  - Operating frequency: 50 Hz
  - I2C Interface
- 28BYJ-48 Stepper Motor
  - Supply Voltage: 5 – 12 V
  - 2048 steps/revolution
  - Full wave stepping method used
- One Active low on-board push button
  - PJ1 is used to start the movement of the motor and initiate the Tof sensor for scanning until the specified number of revolutions are completed.
- Serial Communication
  - I2C communication used between MSP432E401Y and VL531LX Tof sensor with the microcontroller as the leader and sensor as the follower.
  - UART with baud rate of 115200 used for communication between microcontroller and PC.
- Visualization
  - PySerial (Python) is used for serial communication between PC and board.
  - Open3D library is used for plotting and scan construction.

B - General Description

Operating at a default bus speed of 16 MHz, this system enables low-cost 3D visualizations for a range of applications. Upon executing the Python code, the character `'e'` is transmitted to the communication interface, which is recognized by the Keil C program as the signal to initiate the demonstration. To confirm successful communication, LED1 (PN1) blinks.

Once the on-board button (PJ1) is pressed, the stepper motor begins rotating for a predefined number of revolutions. The ToF (Time-of-Flight) sensor captures distance measurements every 11.25°, resulting in 32 readings per full revolution. After each revolution, LED4 (PF0) flashes to prompt the user to step forward approximately 600 cm—this step applies only if multiple revolutions are configured. During each sensor reading, LED2 (PN0) blinks to serve as a visual indicator.

The rotation direction is alternated after every revolution to prevent wire tangling and ensure smooth operation.

Distance data is collected via I2C and stored in an array. Instead of sending data continuously, all readings are transmitted to the PC via UART in a single batch once all revolutions are completed. The data is then processed and visualized using Python.

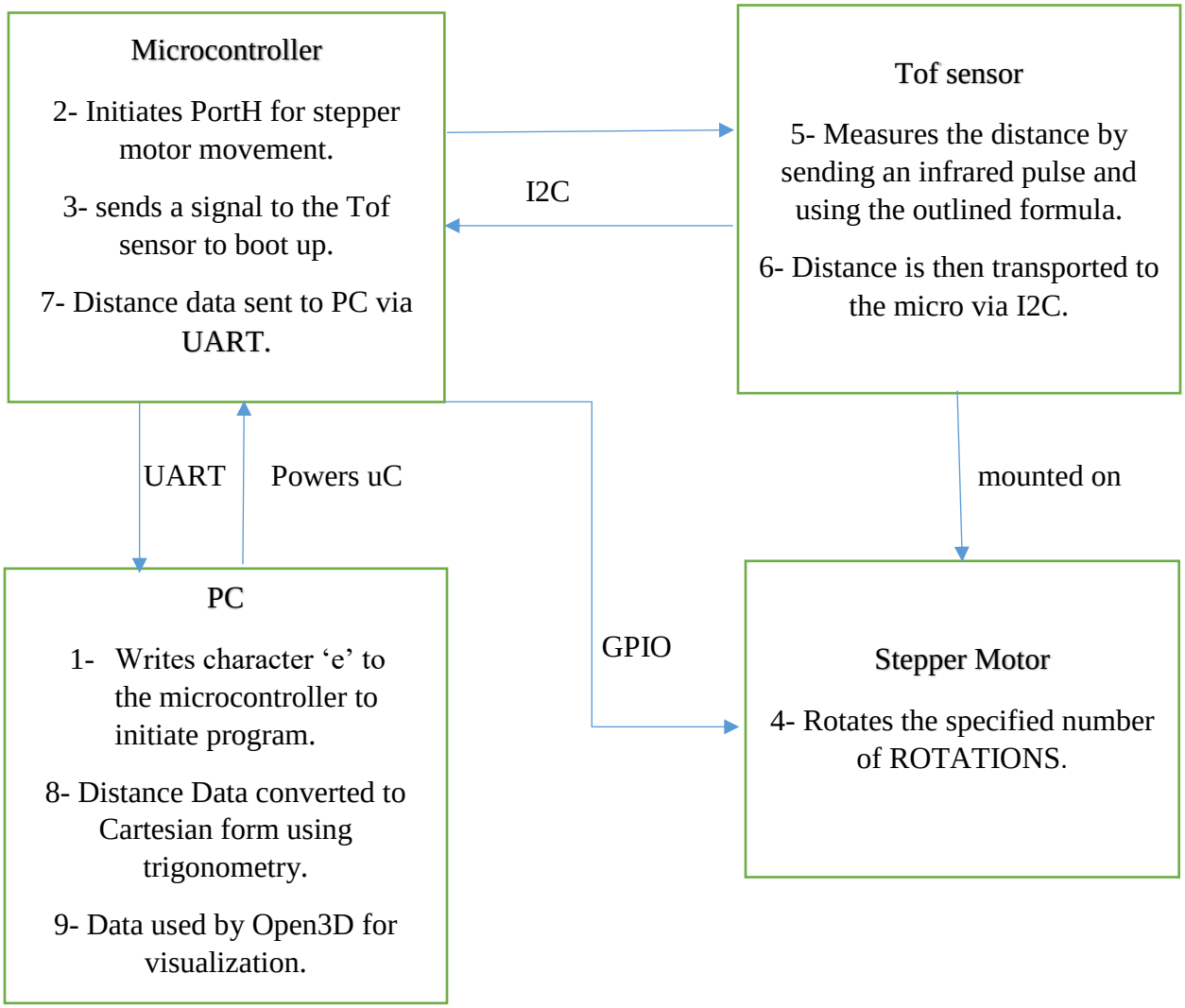
The ToF sensor operates by emitting an infrared beam from its transmitter. The beam reflects off nearby objects and returns to the receiver, allowing the system to calculate distance based on the time delay.

The formula used by the sensor is:

$$Distance = \frac{photon\ travel\ time}{2} * speed\ of\ light$$

The data received is analyzed using Python library Open3D. The distance measurements are converted into X and Y coordinates using simple trigonometry. The depth of the image is produced by specifying the z-coordinates. After each revolution, the z-value is increased by 600. This indicates the amount of distance covered by foot from the user.

C - Block Diagram



Device Characteristics Table

Device Feature	Specification
Microcontroller	
Clock Speed	16 MHz
Status LED	PN0
Measurement LED	PN1
Reference LED	PF0
Push buttons	On-board PJ1
GPIO Input Ports	PortJ
GPIO Output Ports	PortN, PortG, PortH
Serial Port	‘COM4’ (configurable)
UART Baud Rate	115200 b/s
Stepper Driver Board	
V+	5V
V-	GND
In1	PH0
In2	PH1
In3	PH2
In4	PH3
Steps/Revolution	2048
Tof Sensor	
Vin	3.3V
GND	GND
SDA	PB3
SCL	PB2
Maximum Distance	400cm
Programming	C (Keil) for board, Python for visualization
Visualization software	Open3D (Python)

Detailed Description: Part A - Distance Measurement

The VL53L1X ToF sensor emits a 940 nm pulse of infrared light. Once transmitted, the light reflects off any object in its path and returns to the sensor’s receiver. The distance to the object is then calculated using the time taken for the pulse to return and the speed of light. Formula is as follows:

$$Distance = \frac{photon\ travel\ time}{2} * speed\ of\ light$$

Following the reception of the signal, the sensor performs transduction, signal conditioning, and analog-to-digital conversion, resulting in a digital value that represents the measured distance. This digital measurement is then transmitted via the I2C protocol to the board through the SDA line.

The microcontroller and sensor operate in a leader-follower configuration. Key connections between the microcontroller and the sensor include Vin (connected to 3.3V), GND (connected to ground), SDA (connected to PB3), and SCL (connected to PB2). After powering up the sensor using the BootState function and initializing it via SensorInit, measurements are triggered with the StartRanging function. The microcontroller doesn't automatically initiate measurements; instead, it uses a polling method triggered by an active-low push button (PJ1). Pressing PJ1 toggles a variable named ScanEnable, which breaks the infinite while loop and initiates the distance measurement process. The result is then retrieved via UART using the GetDistance function.

In the Keil C environment, a predefined constant named REVOLUTIONS determines how many full rotations the stepper motor will complete. After executing the Python PySerial script and establishing secure UART communication, pressing PJ1 triggers the system—LED1 (PN1) flashes to confirm successful UART communication. The motor then begins rotating, collecting 32 total measurements per revolution, with LED2 (PN0) blinking after each measurement to indicate progress. This corresponds to a measurement taken every 11.25 degrees. Once a full revolution is completed, LED3 (PF4) blinks to prompt the user to step forward by 600 mm, a step size defined in the Python script for hallway depth scanning.

To avoid wire tangling, the C code toggles a variable called direction after every revolution to reverse the motor's rotation. The Keil code transmits several data points, including RangeStatus, Distance, Step, Depth, and SpadNum; however, for this specific application, only the Distance data is utilized.

Part B - Visualization

The Python script executed on the PC serves two main functions: it initiates the embedded C program by writing the character 'e' to the UART interface, and it also handles the Open3D-based visualization. Communication occurs through "PORT4" with a baud rate of 115200 bps. Incoming data from UART is stored in an array named distance_m.

A for loop (for i in len(measurements)) iterates through each distance value in the array and performs the following operations to calculate coordinates:

x = measurement * sin(ANGLE * (i % (TotalMeasurements / ROTATIONS)))
y = measurement * cos(ANGLE * (i % (TotalMeasurements / ROTATIONS)))

Here, ROTATIONS is predefined to match the number of revolutions specified in the Keil C code (default is 3). ANGLE is set to 11.25 degrees, corresponding to the angular resolution between each measurement. The modulus operation (i % (TotalMeasurements / ROTATIONS)) ensures that the angle resets with each new rotation, incrementing step-by-step based on the number of total measurements being processed.

The z-value is calculated using:

if i % (len(measurements) / ROTATIONS) == 0:

```
z += 600
```

Initially set to 100, the z value increases by 600 after every full revolution, reflecting the user's forward movement in the scanned environment.

Finally, the calculated x, y, and z values are appended to a variable called map, which is then used by the Open3D library to render the 3D visualization. Open3D provides dedicated functions to configure and display the visualization window. It also enables user interaction with the rendered 3D model, enhancing the scanning experience with an immersive and manipulable interface.

Application

LiDAR technology utilizes laser pulses to accurately measure distances and create detailed 3D representations of objects and terrains, establishing itself as a highly versatile remote sensing tool with wide-ranging real-world applications. One major application of LiDAR is in precision agriculture, where it produces detailed maps of crop health and soil elevation. These insights allow farmers to optimize irrigation systems, monitor plant growth patterns, and enhance yield predictions. Beyond the agricultural sector, LiDAR plays a crucial role in robotics by providing machines with high-resolution spatial awareness—essential for tasks such as warehouse automation, precision assembly, and navigation in hazardous environments.

LiDAR also has transformative uses in coastal engineering and renewable energy. In coastal studies, it aids in modeling shoreline erosion, tracking sediment displacement, and designing resilient infrastructure to combat rising sea levels. In wind energy applications, LiDAR helps analyze wind flow patterns, determine optimal turbine placement, and increase the overall efficiency of wind farms.

Instructions

Before proceeding with the scanning steps, ensure that all required software and configurations are in place. This includes installing Keil and Python (version 3.9 or above), properly configuring the microcontroller settings, and installing the necessary Python libraries—**Open3D** for visualization and **PySerial** for port communication.

1- Plug the microcontroller to your PC via the provided USB to ensure proper UART communication.
2- Determine the COM port. To do this, search and open device manager on your PC. Once open, scroll down to Ports(COM & LPT) and expand the tab. In the drop down menu, copy value of the port number in ‘XDS110 Class Application/User UART (COM number)’. You’ll need to use this Port value for your own specific use case.
3- Modify the COM port number. To do this, open the python file provided, search for the line
s = serial.Serial('COM4', 115200). Replace COM4 with your COM number identified in the previous step.

4- Please make sure the circuit wiring is properly configured by matching it with the details provided under device characteristics. Make sure each wire connects to the correct port and pin number on the microcontroller and the component (stepper motor, tof sensor).
5- Go to your keil Project. Press Translate, followed by Build and then Load.
6- You need to press the reset button on the board to flash the project onto it. This button will be located next to where the USB wire is plugged onto the board.
7- Please modify the number in ROTATIONS in both Keil and Python file according to how many revolutions you want (default is 3) the motor to do and how many data points you need. You can also edit the STEPS constant in keil if you want (the default angle is 32). If you choose to do that, also modify the ANGLE variable in Python by using the formula ANGLE = 360/STEPS. Don't forget to save the files!
8- Run the Python code, to allow UART connection and receival of data between PC and board. This writes the letter 'e' to the UART comm to be received by Keil, allowing procedure to take place. LED1 (PN1) flashed means secure communication.
9- Press the on-board button PJ1 to start rotation of the motor.
10- Move yourself in the direction you want (if you want the depth too), when the motor stops after completing one full rotation and blinks LED3 (PF4).
11- Once the total number of revolutions are complete, the 3D figure made by Open3D will automatically appear in a new window.

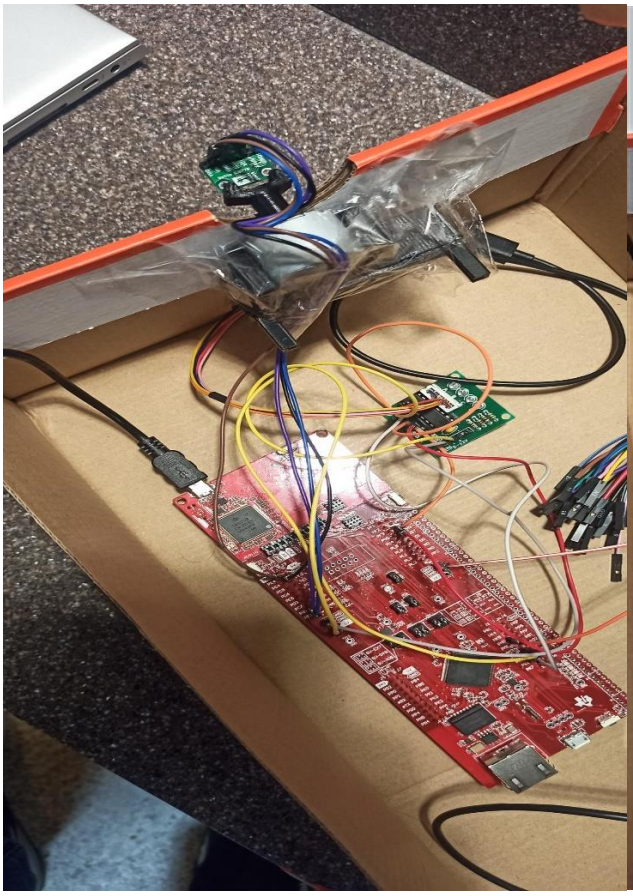


Figure 1: Assembled Physical Prototype

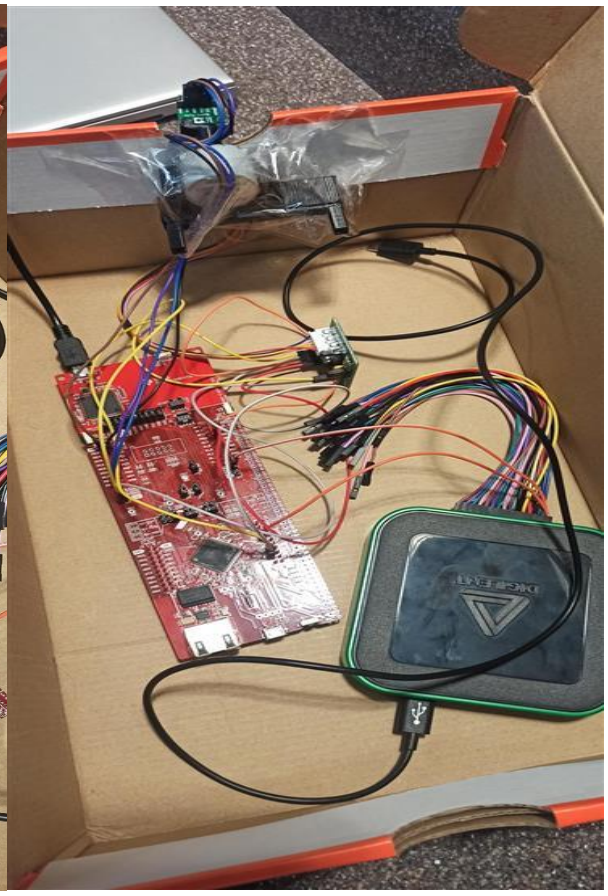


Figure 2: Full image (ignore AD3)

Expected Output

The device's performance is assessed by comparing the actual scanning location (Location B) with its corresponding 3D model. A real-world image of the hallway is presented alongside the digitally reconstructed scan generated by the system. As shown in the results, the device accurately captures the spatial characteristics of the scanned environment. A notable observation is the variation in the hallway's width—it begins wide, narrows in the middle, and then expands again toward the far end—an architectural feature that is clearly represented in the final scan. In Figures 3 and 4 below, please disregard the presence of people and boxes in the hallway, as the scan was conducted at a different time than the photograph was taken.

Real Images

Figure 3: Hallway Start

Figure 4: Hallway End

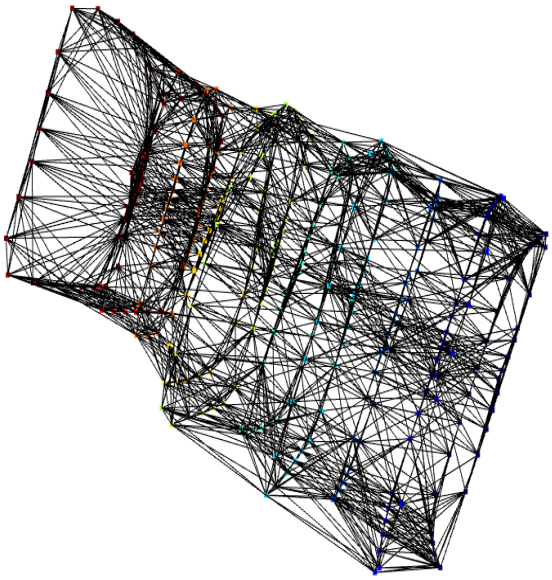


Figure 5: Side View (right side is hallway start)

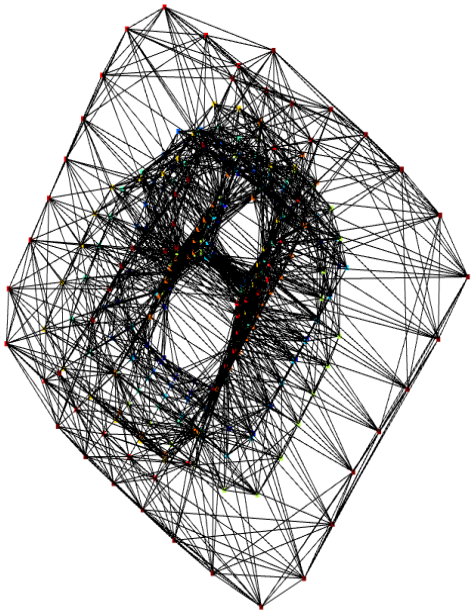


Figure 6: Tilted front view (hallway start)

Limitations

The device relies on **single-precision floating-point arithmetic**, which—unlike Python's double-precision calculations—can introduce minor computational errors. These limitations are particularly relevant during spatial mapping, where trigonometric functions (e.g., sin and cos) generate lengthy repeating decimals. To address this, the TI-EXP432E401Y employs 32×32-bit single-precision registers within its ALU-based Floating-Point Unit (FPU). While this hardware support enables efficient single-precision operations, the inherent trade-off remains reduced precision during conversions (e.g., degrees to radians) and iterative calculations.

The maximum reading from the ToF sensor is 3m, and our microcontroller reads in a 16 bit format, thus we can conclude that:

$$MAX\ QUANTIZATION\ ERROR = \frac{MAX\ Reading}{2^{number\ of\ ADC\ bits}}$$

$$MAX\ QUANTIZATION\ ERROR = \frac{4000mm}{2^{16}}$$

$$MAX\ QUANTIZATION\ ERROR = 0.06104$$

The **maximum supported data transfer** rate for my system was confirmed to be 128,000 bits per second (bps) through the following verification process: By accessing the Device Manager, navigating to Ports settings, and examining the configuration details of the XDS110 Class

Application/User UART interface. For this project's implementation, a slightly lower baud rate of 115,200 bps was selected to ensure reliable operation.

The communication protocol used to interface the ToF to the microcontroller was the I2C protocol. The ToF sensor has a maximum transmission speed of 50Hz

The speed of the motor was the most **significant factor** influencing system performance, as it was determined by the time delays between the four phases necessary for each rotation. Even if the ToF sensor took longer to initialize, the motor would rotate more frequently, resulting in longer operation times. To mitigate the impact of this factor, I utilized the most commonly used shorter time delay which was 10ms between each of the phases.

The **bus speed** was measured by using the formula:

$$SysClk = \frac{fVCO}{(PSYSDIV + 1)}$$

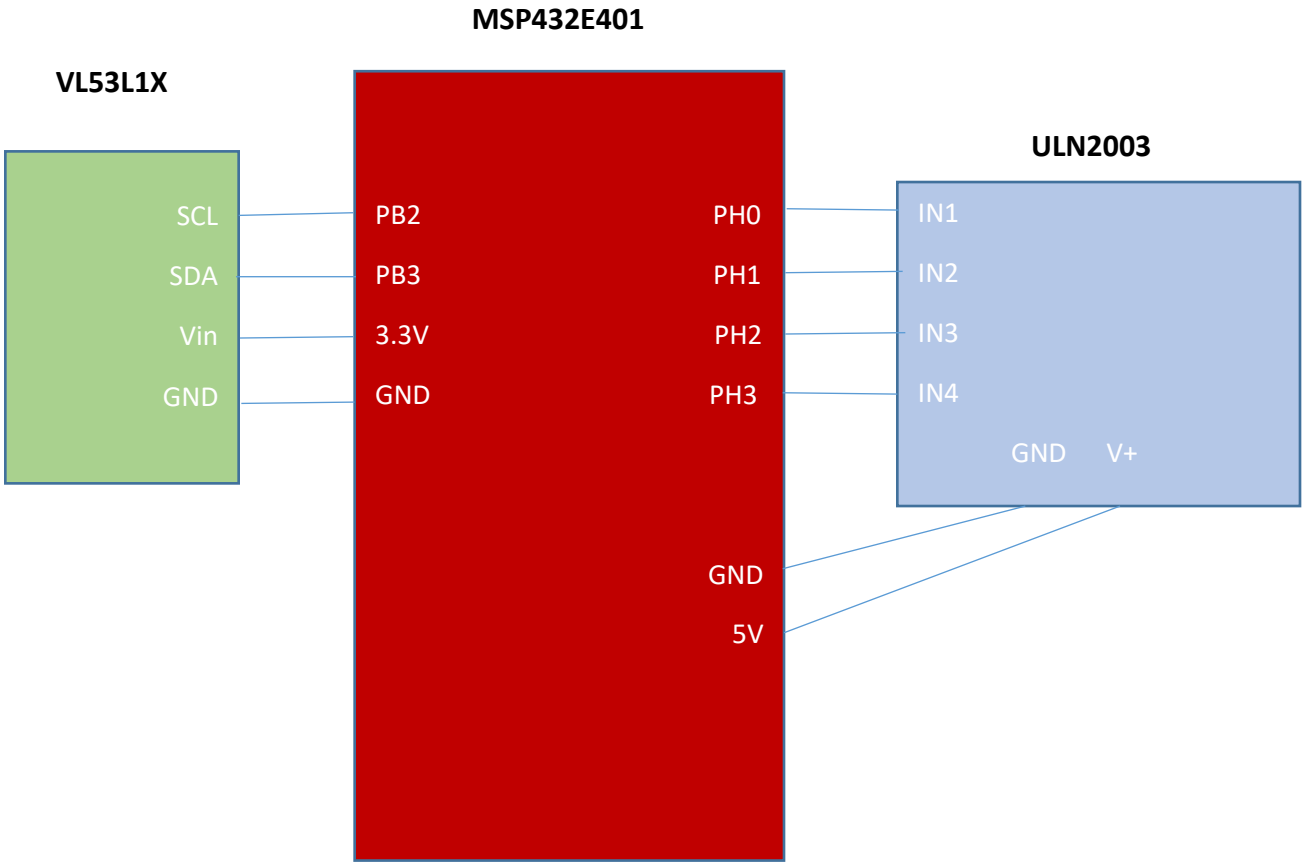
fVCO is set to 480MHz by default using the formula:

fVC0 = (fXTAL/(Q + 1)/(N + 1))*(MINT + (MFRAC/1,024)), where fXTAL is set to 25 MHz, Q to 0, N to 4, MINT to 96 and MFRAC to 0.

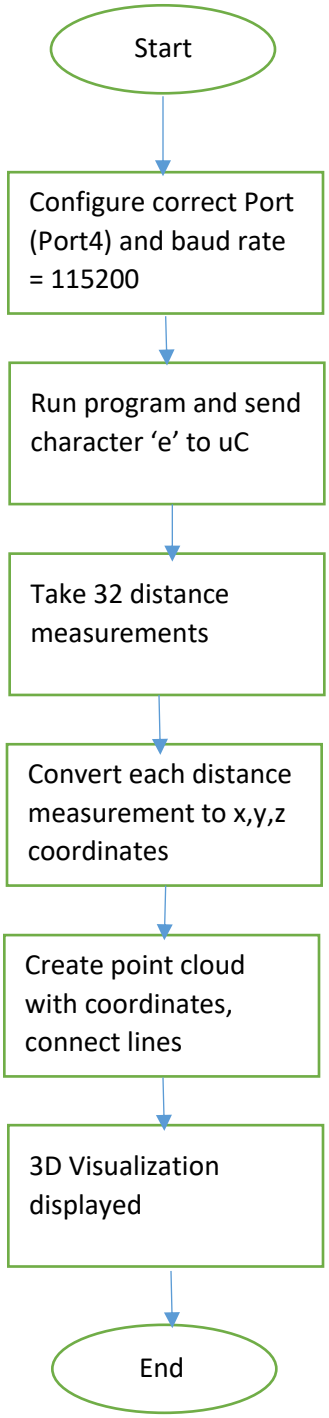
To set my bus speed to 16 MHz, I only changed the PSYSDIV from 3 to 29. After that,

$$SysClk = \frac{480M}{(29 + 1)} = 16MHz$$

Circuit Schematic



Visualization Logic Flow:



Programming Logic Flow: