

PRISM Final Report

1st James Nagler

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
naglerj@mit.edu

2nd Landon Hering

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
lando@mit.edu

Abstract—We present the design and implementation of PRISM (Projection & Rendering in Scanned Mapping), a dual-FPGA system for real-time 3D scanning and projection. The first FPGA manages point collection and processing using a VL53L4CD time-of-flight (ToF) sensor and associated hardware, while the second FPGA handles rendering and projection. Scanning is performed through an initialization phase followed by two concurrent processes: motor control and continuous ranging, with data processed and hypothetically transmitted via UART. The projection pipeline stores points in cylindrical coordinates and projects them dynamically based on the rendering camera’s azimuth.

Index Terms—Digital systems, Field programmable gate arrays, Point cloud, AXI-Stream, I2C

I. PHYSICAL CONSTRUCTION (NAGLER)

The scanning system itself consists of:

- A 200 mm diameter turntable powered by two small reduction stepper motors driven by a ULN2003 chip.
- A 14-bit magnetic encoder, the SPI-enabled AS5048A, on a custom PCB that reads the true angle through a neodymium magnet embedded under the turntable.
- A standard NEMA-17 stepper motor with threaded rod and steel guide rail (repurposed from a broken Prusa i3 MK3) to control the z-axis, powered by the Pololu TB67S128FTG stepper motor driver.
- Attached to the z-axis is a plastic piece that holds the I2C-enabled VL53L4CD ToF sensor that operates at 100Hz, and a limit switch at the bottom of the z-axis is used to zero the system at boot.
- Two FPGAs, one for the scanning hardware, and one rendering over HDMI, communicating via UART.

All circuitry is wired on a breadboard, powered by two separate PSUs (12V and 5V), and all parts for the turntable were designed in Fusion 360 and printed on a 3D printer.

II. INITIALIZATION (NAGLER)

A. Procedure

The hardware consists of three main parts that need to be initialized: the turntable, the z-axis, and the distance sensor. This is shown through an FSM in figure 1. In summary, the turntable and z-axis must move to a “zero” position. For the turntable, this is determined through the magnetic encoder and magnet. The z-axis is slightly more complicated: it needs to find its absolute zero, determined by the limit switch, but

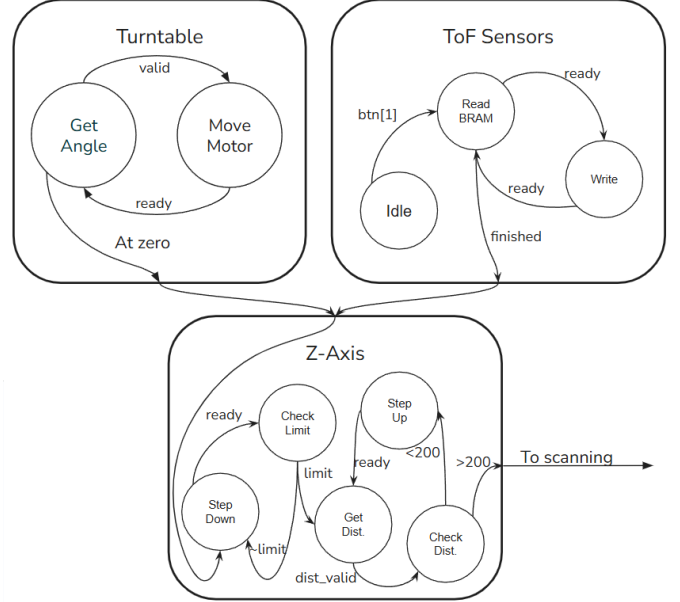


Fig. 1. Complete FSM for the initialization phase.

needs to determine the reference “zero”, or the first distance measurement that sees past the turntable. The final step of this z-initialization is due to absolute zero being below the plane of the turntable, by design. This ensures a reliable zero height.

The distance sensor requires roughly 100 commands to be sent to it before operation, as per ST Microelectronics’ specification. This can be achieved with Lab 5-7’s `camera_registers.sv`, which was cleverly renamed to `init_sensor.sv`.

B. Clocks

All clocks on the initialization and scanning FPGA run from the same 10 MHz clock. Importantly, the distance sensor, which operates over I2C, operates at its maximum of 1MHz, which is used to support the 100Hz sampling rate. The magnetic encoder operates at 100kHz (per its datasheet), and the motors can operate at a maximum stepping frequency of 1kHz. All system modules operate on 10MHz. This clock was determined by considering three main factors: UART transmission speed, I2C data speed, and the filtering stages each frame of data requires. A baud rate of 230kHz is used,

meaning (with 16x oversampling), the clock should be no lower than about 4MHz. Each frame of distance requires 5 commands to be sent over I2C, 2 clock cycles to generate a correct output, and 2 clock cycles for filtering. To be safe and add a reasonable margin for additional filtering, a 10 MHz clock was chosen to ensure the proper functionality of all devices and enough time to provide proper filtering.

C. Evaluation

Due to the relatively small number of processing steps required, and the pipelined nature of the FSM, the initialization and scanning FPGA built with a large slack. Even on a 100MHz clock, there is over 3ns of slack in the final design. Combining FSM stages would not improve performance, as discussed in subsection IV-A. The one BRAM for initialization of the firmware used 436 bytes, which cannot be reduced, since it is from the sensor's specifications. The timing requirements are specified concretely in section II-B.

III. SCANNING (NAGLER)

The scanning phase consists of two concurrent processes: motor control and scanning. Due to the slow turntable movement, this can be done asynchronously, providing a valid height and angle at its output for the scanning module. This is shown through an FSM in figure 2.

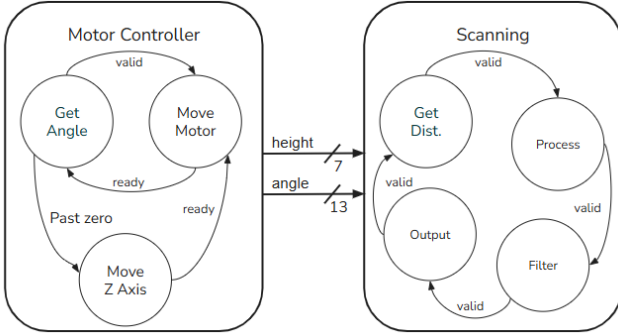


Fig. 2. FSMs for motor control and scanning.

Getting the distance from the sensor requires an I2C polling loop, along with 2 reads and a write over I2C for every data frame. This is represented as an FSM in figure 3.

The initialization, motor controller, and scanning are actually implemented in `top_level.sv` as a very large FSM, but it is easier to represent it as a series of pseudo-modules (shown with dotted lines) that are composed of true modules. The motor controller, including its relevant initialization modules, is shown in figure 4. The **zero** and **ground** wires are controlled when `btn[1]` is pressed, and **single_step** is used during the *Step Up* state from figure 1.

The entire scanning phase, including initialization with BRAM, is shown in figure 5. To reduce the complexity of using the `i2c_master.v` IP, a wrapper (shown as Read / Write Data in figure 5) was used to easily read and write commands over I2C. This allowed for the implementation of the get distance FSM (figure 3). Finally, the data must be

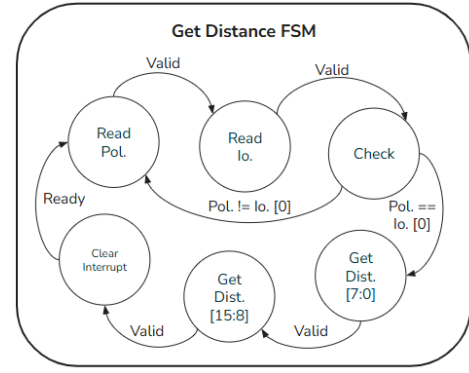


Fig. 3. FSM to get distance.

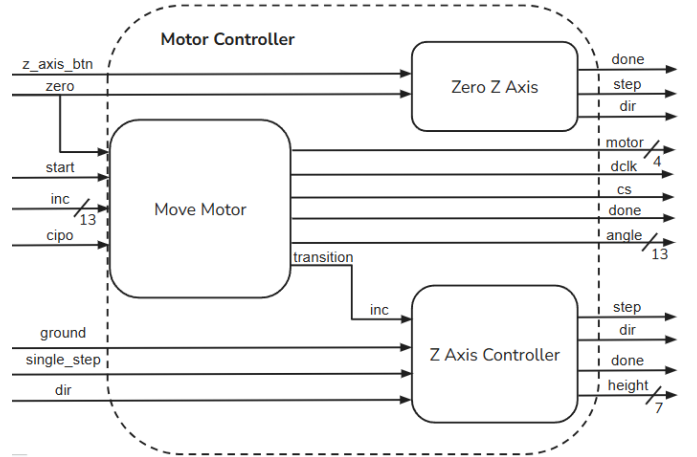


Fig. 4. Motor controller pseudo-module.

processed into a packed data frame, filtered over time, and outputted over UART, shown in figure 6.

A. Filtering

The filtering module implements a simple IIR. To reduce synthesis complexity, an α of the form $\frac{1}{2^x}$ was used, with a final value of $x = 3$ used to produce all results. However, since the distance (radius) is the only measurement being filtered, to preserve the highly accurate rotational and height data, objects that lie along a constant radius or vary smoothly in height and angle produce better results.

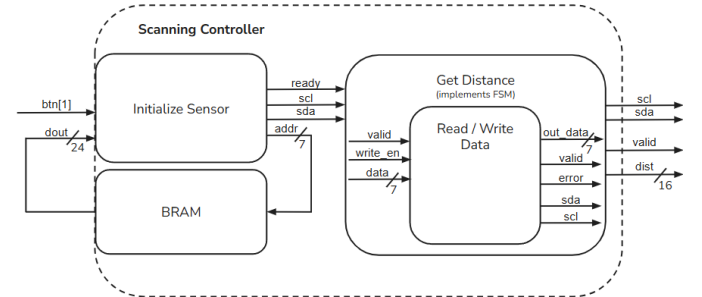


Fig. 5. Scanning pseudo-module.

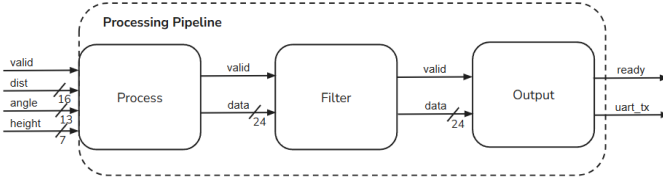


Fig. 6. Processing pseudo-module.

B. Data Packing

Z (8 bits)	R (7 bits)	Angle (9 bits)
------------	------------	----------------

Above is the full 24-bit frame. All data is stored in cylindrical coordinates to ease processing steps in the complete pipeline. The radius can only be from 0-100 mm, so 7 bits are sufficient. The Z follows similar logic. The angle has a theoretical max of 13 bits (testing showed the MSB of the 14-bit frame did not change correctly), but the upper 9 bits are taken to reduce load in the visualization section.

From the initial scans plotted from the ESP32, even with a filtering algorithm implemented on the plotting side, the distance sensor used is clearly better suited for larger distances, with a specified accuracy of 15 mm. Therefore, this hardware would produce better results if done on a larger, almost human-sized scale, with a turntable that one could stand on and a Z-axis that could reach several meters. Perhaps with more time, this could have been implemented.

IV. SCANNING EVALUATION (NAGLER)

The main limiting factor in the final refresh rate of the design was the speed of the turntable, which was unfortunately slow (17 seconds per rotation). Since 9 of the 13 angle bits were transmitted, 7,680 points were discarded, and a valid distance could only be taken at a frequency of roughly 30Hz.

A. Dimensional Accuracy

To assess the accuracy of the points, a scan of a box was taken and analyzed with two metrics: dimensional accuracy and planarity RMS. Dimensional accuracy measured the difference in the scanned points and ground truth for all 3 axes (XYZ), and the planarity RMS (Root Mean Squared Error) measured the flatness of all 6 planes of the box. All metrics are shown in Figure 7.

The worst-case dimensional accuracy was measured to be 6mm, which is significantly better than the 15mm target dimensional accuracy.

B. Roundness Evaluation

To test the accuracy of the measured radius on a cylindrical object, a cylinder was scanned and processed as a function of theta, relative to the radius. The accuracy relative to the true radius and the overall roundness were excellent, performing well under the 15mm target accuracy, shown in Figure 8.

The point clouds on both measurements used a heatmap where warmer colors represent more recent points, and cooler colors less recent points. All data was collected on the TX output from one of the FPGA PMOD pins with an ESP32, and plotted using matplotlib.

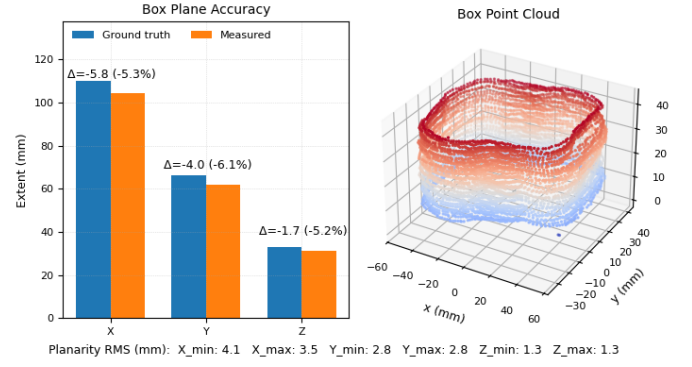


Fig. 7. Box Evaluation.

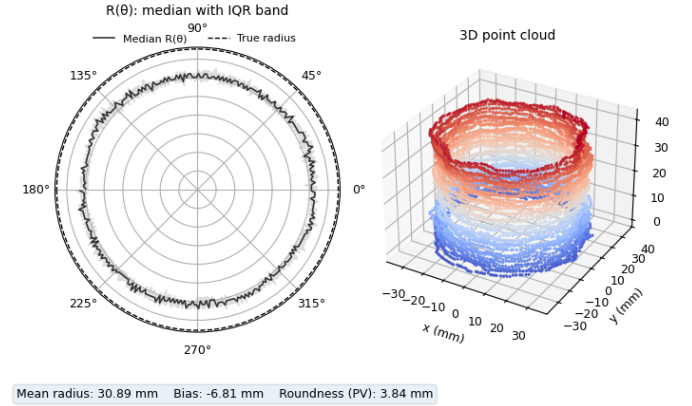


Fig. 8. Cylinder Evaluation.

C. Novelty

Though this project could be done on a traditional microcontroller, it would be incredibly difficult to achieve the complete 100Hz ranging speed of the sensor and complete integration of all hardware. The speed and control an FPGA has allows for this.

V. POINT STORAGE (HERING)

Points are then generated from the scanned points or through test generators. Incoming point data is packed into 128-bit words and transferred through a cross-domain FIFO into the memory system. This 128-bit word size enables efficient DRAM access by allowing entire point groups to be written in a single transaction rather than issuing multiple strobe-based partial writes.

The traffic generator receives these FIFO outputs and produces write requests for the DDR3 memory controller, which are then issued by the arbiter. It additionally accepts a new-frame signal; upon detecting a new frame, it sequentially reads all point entries stored in DRAM up to the max address written to. This mechanism enables repeated projection of the same points as the camera is rotated.

Points read from memory are unpacked into 24-bit points to send to the point transformation pipeline to get actual pixel

values for each point. This overall point-storage and retrieval process is illustrated in Figure 9.

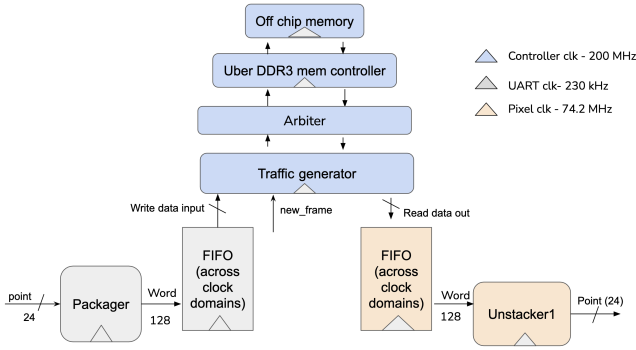


Fig. 9. Point Storage block diagram.

VI. POINT PROJECTION (HERING)

A. High Level

Points are then projected from 3d space down to 2d space based on the camera pose and focal length. Given the 200 mm turntable diameter, the camera is placed at a fixed distance of 250 mm from the turntable center to ensure the complete scan volume remains within the field of view. Thus, we change the camera position using 2 buttons on the FPGA, with the buttons changing the azimuth of the camera, allowing it to rotate along one axis around the object being scanned.

This is implemented through using a debouncer on the 2 buttons to update the camera's azimuth. Then, for each point retrieved in a frame, the point is projected and converted to a point on the 720p screen (or not valid if it is off the screen).

B. Math Modules

First, the projection module converts the camera and point positions using a sine lookup table with 512 entries each representing the sine value of each degree in increments of $\frac{360}{512}$ scaled by the focal length of 1024 (this utilizes all 512 bits of information we have encoding angular position). Multiplying these values from the lookup table by the point radius for points and camera radius for the camera gives the cartesian conversions for the sampled point and camera point.

Next, to get the actual points on the screen, the difference in these cartesian points is used, summing with certain differences, ultimately get an x divisor and y divisor.

Using the fact that only the azimuth of the camera is changing, we can calculate $\theta' = \theta - \varphi$, the new angle for each point in cylindrical coordinates based on the azimuth of the camera φ . Then, these transformed coordinates are converted to Cartesian coordinates to give new coordinates relative to the camera. This conversion utilizes a lookup table with 512 different entries representing the sine value of each degree in increments of $\frac{360}{512}$ scaled by the focal length of 1024 (this utilizes all 512 bits of information we have encoding angular position).

Next, to get the actual points on the screen, these Cartesian coordinates are then multiplied by the focal length and divided by their z value to get their position on the screen. The focal length used is 1024 to enable efficient multiplication (through a left shift), and then to not lose any precision, full 32 bit inputs are going into the dividers and lead to a 32 cycle latency in the division (takes one clock cycle for every bit using a fully pipelined divider). This math could be optimized to use a more efficient divider, but attempts to use more efficient dividers were associated with an increase in unexpected latching behavior so the slower more robust divider was used. Finally display logic checks that these are valid points based on if the outputs of the dividers lie within the screen, and then output the pixel address (a single number $a = v * 1280 + u$). The modules comprising this projection pipeline are shown in Figure 10.

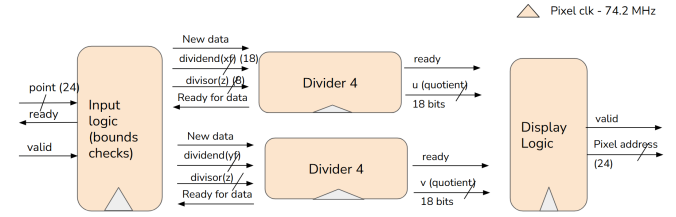


Fig. 10. Point Projection

C. Output

Final output in simulation is very accurate with it only deviating by at most 3 pixels from a simulated output using python 64 bit math. This indicates that the turntable projection met our goal of having a working camera transformation and display of points in simulation. However, achieving this level

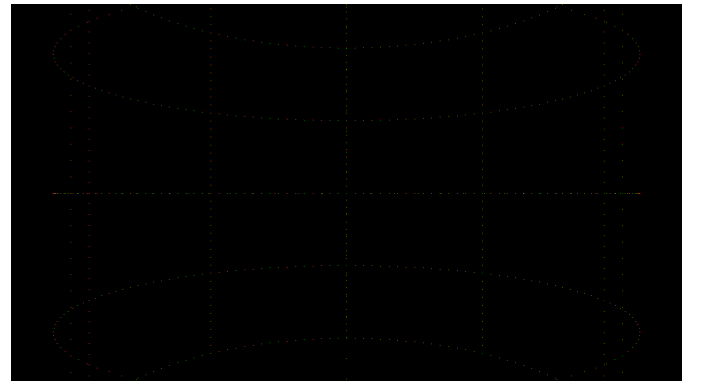


Fig. 11. Projection Cylinder Simulation. Red dots are expected output, green is real output, yellow indicates expected and real output match

of precision in hardware was much more difficult, with signed math, AXI-streaming, pipelining, clock domain crossings, and signal synchronization making the rendering much more complicated than in simulation. Ultimately, point rendering was

eventually accomplished to a reasonable degree, though with much less precision and much less clear outputs compared to the simulation.

VII. VIDEO GENERATION (HDMI)

Points are rendered to HDMI through a dual frame buffer, with points being written through random-access strobe writes to one buffer (the front buffer) and being read sequentially from the back buffer. The traffic generator and arbiter are responsible for swapping buffers on every frame, clearing the new front buffer after a swap, generating write masks, and issuing all DRAM read/write requests associated with framebuffer access. The arbiter gives priority to HDMI reads to ensure that all pixels are properly rasterized, then giving priority to frame buffer writes and finally to the point generator inputs. This dual-buffer architecture is shown in Figure 12. Finally,

- Axi-stream is a very powerful framework for connecting separate modules together, but introduces many signals that are important to get right.
- Debugging through use of on-device hardware like LEDs and the seven-segments display are essential to identifying bugs and incorrect signal assignments.

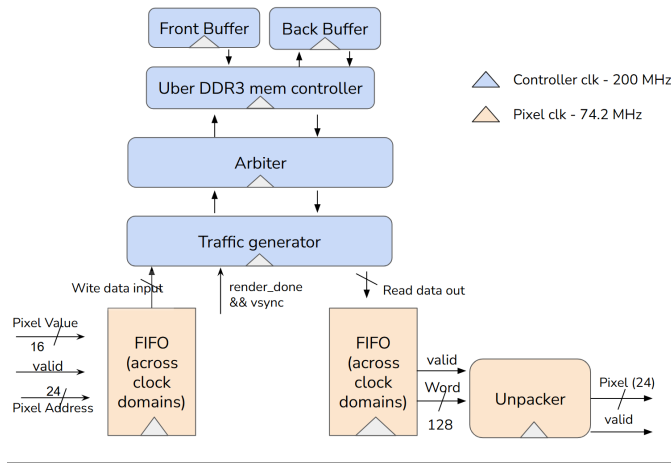


Fig. 12. Dual Frame Buffer block diagram.

after framebuffer lookup, the output pixel data is rasterized onto the HDMI display, utilizing an HDMI signal generator and TMDS encoders and serializers for each color channel. This HDMI output failed to meet our goal of having a stable HDMI, with this output consistently flickering.

VIII. RETROSPECTIVE

- I2C is difficult. Creating a read/write wrapper was pretty much necessary to make this project work.
- Building off of the previous point, picking a sensor that requires very little setup is crucial to making FPGA land easier to deal with. The original sensor, the VL53L5CX, would have been miserable to work with.
- This project would have been better suited for much larger scale objects (human-size), but this would have required much more time creating the physical turntable.
- Utilizing DRAM from multiple sources brings unique challenges, with a memory arbiter needing to take responsibility for which memory requests get priority over others.