SkyLocator

1st Masarah Ahmadhussein
EECS Dept.
Massachusetts Institute of Technology
Cambridge, MA
masarah@mit.edu

2nd Vanessa Gonzalez
EECS Dept.
Massachusetts Institute of Technology
Cambridge, MA
vgonzale@mit.edu

3rd Lejla Skelic
EECS Dept.
Massachusetts Institute of Technology
Cambridge, MA
lejla@mit.edu

***Abstract*—We present the design for SkyLocator, a VR headset system that simulates gazing at the stars during the night in an environment without light pollution. The display updates in real time based on the user's head orientation. The system is implemented using a Nexys 4 DDR FPGA board, SD card, an external computer, and an Inertial Measurement Unit. We implement the system and discuss its performance as well as possible improvements.**

***Index Terms*—Digital Systems, Field Programmable Gate Arrays, Virtual Reality, ESP32, Inertial Measurement Unit, SD Card, Astronomical Projections, Universal Asynchronous Receiver/Transmitter**

I. Definitions

- **User horizon:** a hemisphere of stars visible to the user
- **User frame:** the field of view of the user (i.e. what is displayed on the screen)
- **Vernal Equinox:** The equinox on the Earth when the subsolar point appears to leave the Southern Hemisphere and cross the celestial equator, heading northward as seen from Earth. [5]
- **Celestial Coordinate:** It is a star coordinate system where the origin is defined to be the vernal equinox. The system plane circle is defined to be the equator, and the vertical axis points to the north direction. It has two coordinates, declination (DEC) and right ascension (RA). These values' precision is in minutes in our system. DEC is expressed in (0, 360) degrees and (0, 60) minutes. Similarly, RA is expressed in (0, 180) degrees and (0, 59) minutes. By definition, RA is expressed using values (-90, 90) degrees; however, we decided to shift this value by 90 degrees to avoid the difficulties of storing and working with signed numbers.
- **Observer Coordinate:** It is a star coordinate of which the circle plane is defined by the user horizon, and the vertical axis is defined by the absolute point above the observer vertically, which is referred to as the zenith. It has two coordinates, altitude, and azimuth. The origin is the observer horizon for altitude and the north heading for azimuth.
- **Altitude/Azimuth:** The horizontal coordinate system is a celestial coordinate system that uses the observer's local horizon as the fundamental plane to define two angles: altitude and azimuth. [6]
- **Star color:** In astronomy, stellar classification [2] is the classification of stars based on their spectral characteristics. Most stars are classified under the Morgan–Keenan (MK) system using the letters O, B, A, F, G, K, and M, a sequence from the hottest (O type) to the coolest (M type).

II. High-Level System Description

The system, depicted in Fig. 1, roughly consists of 3 parts:

[User] The user component collects the time and location information from the user. This information is passed to the ESP32 and used by the code on the server to calculate vernal equinox altitude and azimuth. The coordinates are adapted to the star coordinate system we are using and returned to the FPGA.

[Memory] The memory consists of SD card-stored star data that is accessed by the system in the beginning to create local memories that will be used primarily by the display modules.

[Graphics] For each pixel in the frame, local memory is matched at a location to determine whether or not there should be a star displayed at that location and what color it should be.

[IMU] The ESP32 is connected to the IMU which gives us the angles (yaw, pitch, roll) at which the user is looking so we can properly adapt the display. This information is sent to the FPGA periodically through a UART connection.

III. User

[Welcome Screen]The seven-segment display is used as the input screen. At the begining of each input state, there is a letter on the most significant digit to indicate the type input the user should enter. The user inputs their location and time using the buttons. The user utilizes the left and right buttons to toggle between the digits, and up and down digits to increment or decrement the number or change the sign of longitude or latitidue. The Interface limit the user ability to enter undefined input by checking the relavent maximum and minimum for each digit. For example, the The program will accept an accuracy of four digits to accept hour/angle:minute. Once an input is done and verified to be valid, the user flips the first four switches wiith the correct pattern to transition to the following input. Once all inputs have been entered, the user can turn to switch 0 ON to exit the home page and start the loading page.

[Loading Screen]During the loading page, the user input will be sent to an external computer through UART. The

vernal equinox coordinates, which is the reference angle for the star database, are computed through astropy library [2]. An approximation method has been investigated in python to find the coordinates of the vernal equinox that was possible to implement in the FPGA with the help of CORDIC trigonometry modules. However, the approximation function for the Greenwich sidereal time and earth rotation angle functions failed to output a sufficient level of accuracy due to possible outdated astronomical assumptions(check theta.py). astropy is a sophisticated astronomy python library that can be used to compute the coordinate of the vernal equinox more accurately.

The digits of the user input, in total 21 digits, are transmitted to the FPGA as 8 bits chunks from LSb. Each packet will contain a start bit, 8 bits of information, and a stop bit. The ESP32 expects a specific order of the digits sent and it parses accordingly.

data	**long 180°59'**	**lat 360°59'**	**time 24 h 59 m**	**parity**
bits	15	15	11	1

After computing the coordinates, the server will send the result to ESP32 which will send the info to the FPGA in 20 digits, where each digit is sent as a byte packet. Data will be sent and received in 9600 baud.

data	**altitude 360°99'**	**azimuth 360°99'**	**buffer**	**parity**
bits	15	15	1	1

[Star Screen] The user can choose between using the IMU to indicate the head direction or using the buttons to move the user's perspective through `frame_buffer`. This feature allows for the program to be used independently of the headset and ease the debugging of visuals.

The FPGA will take the latest value stored in the IMU register from the ESP32 messages. Three angles need to be determined to describe the user's head tilt, pitch, roll, and yaw. Pitch and roll angles are relative to the horizon, which can be determined using acceleration data. Yaw angle is relative to the north direction. Since the IMU gives the vector of the north direction, trigonometry was involved in the ESP32 to find the compass direction of the north. The measurement units are mapped to the desired range; the acceleration is mapped to an angle.

In every frame, the last stored angle from the ESP32 or the stored user's angle based on button input is used to output the frame bounds. The frame is defined to have a 60' height and 100' width.

In implementing the `frame_bound`, a major focus was to make it parameterize so the system could easily work through different screens. Defining the parameter at the beginning of the program which could change based on the screen dimension, the desired zooming ratio, and the desired speed of moving through the map using the switches.

The `frame_bound` takes the user angle and defined it as the middle point in the horizon. It adds an offset based on the screen size and ratio around that point to create the frame's upper, lower, right and left bounds. It mods the input to make all the bounds valid in the `start_memory`. Also, it limits the user's up and down movement based on the normal declination limits and combines the vernal equinox to it.

IV. Memory

The two memories of the system are the external star data database on the SD card and the internal/local star memory that are populated with stars in the user horizon.

The database and the memory modules have been created and integrated into `top_level` state machine. The memory structure was conveniently adapted for usage and scalability.

A. SD card

The data on the SD card was stored in a special format manually. The data contains the 300 brightest stars in the universe. [4] Each star's data occupies 5 bytes in the memory. The data stored in those 5 bytes (40 bits) as follows:

6 bits	**9 bits**	**6 bits**	**8 bits**	**6 bits**	**3 bits**	**2 bits**
zeros buffer	**RA degrees**	**RA minutes**	**DEC degrees**	**DEC minutes**	**spect-rum**	**bright-ness**

Each star contains 34 bits of data, so we pad the data with 6 bits of zeros to align it on the byte boundary for easier reading from the SD card, post-processing, and locally storing the data.

We decided to include information on the color of the star for display. We used the Morgan–Keenan (MK) system with the letters O, B, A, F, G, K, and M There are subclasses and further details that could be included; however, we decided to focus on the most dominant color. We encoded the spectral information as follows:

MK	**Color**	**Encoding**
O	**Blue-violet**	**0**
B	**Blue-white**	**1**
A	**White**	**2**
F	**Yellow-white**	**3**
G	**Yellow**	**4**
K	**Orange**	**5**
M	**Red-orange**	**6**

Finally, we decided to encode the star brightness [3]. Ideally, our project would include varying star radii that would be based on the stars' brightness. Within our 300 stars, the brightness was in the interval [-1.46, 3.54]. These values are on a reverse logarithmic scale, and thus we decided to settle on the following encoding:

Brightness interval	**Encoding**
(-inf, 0]	**0 (brightest)**
(0, 1.4]	**1**
(1.4, 2.4]	**2**
(2.4, +inf]	**3 (least bright)**

A module has been implemented that reads the data in 5-byte chunks as bytes are being read from the SD card. Each of these chunks is then stored in the local memory.

B. Local memories

Local memory consists of two parts: star memory and one-hot memory. These two memories will be useful for the graphics part of the system. Before the start of the VR simulation, we populate the two memories with the stars from the universe. The memories are implemented to be scalable. Currently, they store only 300 stars; however, they can store up to 64,800 stars. We decided to go with a lower number of stars given the small dimensions and low resolution of our VR headset display.

Both memories are indexed by DEC (range 0-360) and RA (range 0-180). One-hot memory at [DEC][RA] contains 1 if there exists a visible star at that (DEC, RA) coordinate visible to the user; otherwise, it contains a 0. This design choice was made with several possible user frame implementations in mind. The user frame could have either been calculated as the function of the `frame_bounds`, `hcount`, and `vcount` (Fig. 1) or the one-hot memory could have been used to store the information about the stars that do and do not belong to the user horizon (at the user's location) until the user resets their location. Star memory at [DEC][RA] contains the 34 bits of data of a star that is at the coordinate (DEC, RA).

Since we have 300 stars, most of the local memories will be empty. However, this system can easily be scaled up to many stars; it can accommodate 64,800 stars.

After user's time and location inputs, the one-hot memory will be populated with zeros. Then, the stars will arrive in 5 byte chunks. Each of these chunks contains 34 bits of star data. Based on the star data (star's DEC and RA location), we populate the star memory as well as the one-hot memory.

V. UART Communication

To establish communication between the ESP32 and the FPGA, the SPI communication protocol has been investigated since it is preferable to ease parsing and provide flexibility with the message length. However, no functional libraries were found. Thus, UART has been implemented due to its simplicity of having RX (receiving line) and TX (transmission line).

[FPGA side] Two pins in the Pmod header have been defined as RX and TX to provide a communication port with the ESP32. This has been implemented by creating a UART module that can send a byte and/or receive a byte during communication. For both functions, a finite state machine has been defined. For receiving, the IDLE state will wait for the RX line to drop down which indicates the start bit. After, it will enter receiving state where it will parse the 8 bits and verify the stop bit before signaling the arrival of a new packet to read. For sending, it will wait in IDLE until it sees a valid input signal then it starts sending from LSb the data register content. An order of information has been specified in advance between the FPGA and ESP32 to create the state machine.

[ESP32 side] Due to the difficult calibration of the accelerometer and the magnetometer of the IMU, an ESP32 was necessary for measuring the angles of rotation that were needed to update the user's screen. Furthermore, the ESP32 was used to calculate the user horizon from the input provided by the user and sent from the FPGA. For this purpose, a POST request was sent to the server. The server returns the vernal equinox altitude and azimuth in the system altitude and azimuth coordinate bounds. Then, the ESP32 begins sending the IMU-measured angles to the FPGA. These angles are used to update the screen.

VI. Graphics

The graphics of this system converts data for each star from the memory into a viewable format where users can see where each star would be positioned in the night sky if they were to look up at the location that they gave in to the user input. This display consists of three major components: reference lines that can be used to determine if stars are drawn in the correct position, a single and split-screen display of stars, and a moving frame that allows users to adjust the stars that they can see at their location by using IMU angular input or by pressing the btnr, btnl, btnu, and btnd buttons on the FPGA. The reference lines that are drawn to the screen will always remain stationary, but the stars that are drawn to the screen move with a shifting `frame_bound`.

The graphics of this system were also scaled to have a resolution of 500 x 760 pixels, as this dimension would allow us to fit two copies of our display (one for each eye) on the screen that we use for our headset. This would allow us to leave some space between each of the displays when the system was using a split-screen display of the stars, as the overall resolution of our screen was 1024 x 768 pixels. We initially intended to use a smaller screen with dimensions of 800 by 480 pixels that would better fit in our headset, but it did not arrive in time. Due to this, we were able to use these larger screen dimensions to display more stars at a time. However, the screen did not work as well in the headset as we would have hoped due to its significantly larger size.

A. Reference Lines- Right Ascension (RA) and Declination (DEC) Angles

We display stars to the user by plotting them on a grid with their position being dictated by the star's RA and DEC angles. The RA angle values will serve as the x-axis and the DEC angle values will serve as the y-axis. In order to make testing star position on the grid easier in the future as well as the ensure that the user would have a visible reference for where a star could be located in the sky, we decided that it would be best to draw a grid on our display screen to represent the RA and DEC angles as described above.

This was accomplished within the one of our modules, where we were able to draw lines to the screen that created a grid of approximately 50 by 76 degrees where a line was drawn for each degree. In my calculations, we assumed that each star would be represented as a 9x9 square, so each possible coordinate in our grid was a 9 by 9-pixel square. Each reference line was drawn through the center of each of these coordinate squares, as this would make it easier for users to tell a placed star's exact RA and DEC angles. The actual

numerical value that each line represented depended on the `frame_bound`.

While these lines were very helpful for the purpose of debugging and helping a user to visually orient stars, they could be a bit of an eyesore when we just wanted to view a complete star display. Due to this, we decided to make it possible for the user to toggle the display of the reference lines using switch 12 if they did not desire to see the lines. This toggle works both for the single-screen display and the split-screen display.

B. Single Screen and Split Screen Star Display

The visual display of our graphics resulted in stars from the memory being drawn onto the screen. Whether or not a star is drawn to the screen will ultimately depend on whether or not its RA and DEC angle values fall within the given `frame_bounds`.

We also implemented a toggle for a split screen that would allow users to decide if they would like graphics to only be displayed on the left side of the screen or on both the left and right sides of the screen. We need the same display to be visible in each eye of the user if we wanted our display to work properly for a headset, so we gave the user the option to toggle between the two display options using switch 11. The calculations used to determine the position of a pixel in each of these displays are the same for both the single and split screen displays, with the split screen display allowing the same RA and DEC angle grid to be drawn on the right half of the screen that was already drawn on the left for the single screen display.

In order to determine when a star is being drawn to a specific pixel on the display, we created a separate module. The basic purpose of this module is to take in `hcount`, `vcount`, and `frame_bounds` and return the star coordinates of the star that would be drawn at that pixel if a star exists at that coordinate. The module accomplishes this goal by performing a calculation that determines which 9-pixel bucket the current pixel fell into on the screen both horizontally and vertically. Using `frame_bounds` to determine the lower and upper boundaries of the screen, this calculation will allow us to determine the RA and DEC angles of the given pixel within the frame bound. This module with then return the RA and DEC angle of the current pixel to `top_level`, where this information can be used to obtain the information about the star at the calculated RA and DEC angles from the `one_hot_cache` and the `star_cache`. Then one of our display modules is able to determine if the current pixel belongs to a visible star using the value obtained from the one hot memory as well as determine how it should be drawn, based on color information also stored in the value from the star memory.

C. Moving Stars in Shifting Frame

As we expect users to want to look around the sky at their given location, we determined that stars should be capable of moving across the screen when the direction that a user is looking in with the headset changes. This implementation exists in a form that is independent of the IMU and in a form that relies on the input from the IMU, with the user being able to toggle between both options using switch 10. When not using the IMU, the user can adjust the frame bounds from their current position by using the btnr, btnl, btnu, and btnd buttons on the FPGA to adjust their frame to the right, left, up, and down. When receiving input from the IMU, the `frame_bounds` module with accept the angle values received by the IMU and use them to determine where the new frame bound should lie.

With this functionality, the display of stars can move from left to right across the display when the user pressed the buttons on the FPGA. Our system also accounts for potential wrapping issues that may occur when a frame bound passes from 360 degrees back to 0 degrees. This will allow users to continuously move their frame bound from left to right, causing all of the stars in the display the shift depending on which button was pressed.

VII. Physical Construction

The VR headset consists of:

- Cardboard headset.
- Nexys 4 DDR FPGA board with a 2GB SD card that stores the star DB
- LCD screen that is inserted in the headset. The screen is split into two identical images to account for its closeness to the eyes. The intended VR headset thin display was, unfortunately, not obtained in time for testing.
- Adafruit 9-DOF IMU Breakout (L3GD20H + LSM303) is used since it contains LSM303 which includes a magnetometer and an accelerometer. 9-axis IMU LM303 attached to the headset that collects the head tilt information to update the star display.
- ESP32S2 microcontroller for communicating with the server to get some astronomical data and receive IMU data.

VIII. Evaluation

[BRAM Usage]

- Star memory 360x180x34 bits
- One-hot cache 360x180 bits

Total: 3.8%

[Timing Requirements]

- Our system consisted of 65MHz VGA signal, 25MHz SD card communication, and 9600 baud was used for ESP32-FPGA UART communication. The systems were synchronized by carefully organizing the subsystems that worked on a slower clock.
- We did have a negative WNS, so we failed to meet timing requirements in some portions of our system, which we will describe further in the next section.

[Slack] After building our project, all of the WNS values produced in our timing report summary were approximately -3.5. This is not a good thing, as it means that there is negative slack in our system that is causing us to not reach

timing requirements. Our best guess as to where this negative slack may be coming from is accessing the star and one hot memory to obtain the information used the plot the stars on our visual display. We attempted to circumvent this issue by pipelining our `hcount`, `vcount`, and memory values, but later realized that the logic that we did to display the stars were contained within an `always_comb` block, which may have prevented this pipelining from making the difference that we hope for. Despite this timing issue, our system still worked as expected and did not seem to impact our performance in a very detrimental way, but this would definitely be something that we would have looked into more if we had more time.

[ESp32-FPGA Communication] ESP32 and FPGA UART communication was tested using multiple ESP32 units, oscilloscopes and wave generators, as well as a direct connection between ESP32 and the FPGA in combination with a Serial Monitor to evaluate the UART implementations. While the ESP32 seems to have met the requirements, the FPGA state machine has not passed all of the tests and thus it was not integrated with the overall system. The FPGA UART communication seems to be working well on its own but not as a state machine communicating with the ESP32. Further inspection and testing are needed for these modules to be integrated into the whole system so that the VR headset can be fully functioning.

[Star Placement] One method that we used to visually evaluate if stars were placed in the correct position on the display was through the use of reference lines, as we mentioned earlier. In our design, each line represented 1 degree of either the RA angle or DEC angle. These visual lines allowed us to determine if our design for the display was successful or if an error was being made when calculating where stars should be placed. If we noticed a discrepancy in our display using this method, we would know where the code was failing and what needed to be investigated in order to fix it.

[Use Cases and Project Checklist Deliverables] We did reach all of our minimum goals; however, we were not able to meet our ideal goals. The main bottleneck to the system was the ESP32-FPGA UART communication state machine. In the given timeframe, we were unable to debug the communication. The problem seems to be coming from the FPGA side; however, we were not able to identify the exact source of the problem.

Despite this, we were able to reach one of our stretch goals:

- drawing stars to the display using the color of the star that is stored in the star memory.

Another stretch goal we could have met was building the headset; however, the parts (most importantly, the display) have not arrived in time for us to assemble a feasible headset (the screen we were provided with was too big for the cardboard case).

[Potential Future Improvements and Expansions]

These are some of the ways that we believe our project could be expanded in the future:

- Implementation of a star memory access that would allow for roll rotation.
- Implementation of `frame_bounds` that would tilt together with the user horizon.
- Implementation of a variable grid size to display stars. This would allow us to alter the visual width and height of the display and not limit our potential values for `frame_bounds`.
- Implementation of variable star sizes using the information that is stored in star memory. While the dimensions of the grid could remain constant, we could use the brightness of each star to determine what percentage of its 9x9 pixel square it would take up in the grid.

IX. Contributions

Masarah was responsible for understanding the calculations needed to translate the celestial coordinate to the observant (user) coordinates and the possible approximations that can be implemented in the FPGA. She communicated with Prof. Michael Person, a professor at MIT and a research scientist and director of Wallace Astrophysical Observatory, for guidance and explanation of the methods for different star coordinate systems and to verify the scientific accuracy of the implementation. She researched IMU applications for virtual reality and the methods to find the absolute head orientation. She worked on implementing the user interface, the IMU implementation, and calibration on the ESP32, SPI, UART and UART FSM modules on the FPGA. Also, she worked on writing the python code for the server side using astropy library.

Vanessa was responsible for designing and implementing the graphics that were used to display the system. She worked on the mathematical concept used to convert pixel position to RA and DEC angle values, obtaining the correct values from memory to properly plot stars in the display, and creating a module that would allow for both a stationary and moving display.

Lejla was responsible for designing and implementing the memory parts of the system, `top_level`, and the ESP32 UART communication; worked on the mathematical and conceptual design, system design and data representation, as well as system integration; worked on designing, testing, and implementing the ESP32-FPGA communication system, most notably the ESP32-side communication as well as the FPGA-side UART communication.

Source Code:

- Title: SkyLocator
- Date: 14-12-2022
- Code version: 1.0
- Availability: https://github.mit.edu/lejla/6205_F22_Final_Project_Lejla_Masarah_Vanessa

References

[1] Astropy Collaboration et al., "Astropy: A community Python package for astronomy," vol. 558, p. eA33, Oct. 2013, doi: 10.1051/0004-6361/201322068.

[2] "Stellar classification," Wikipedia, 18-Nov-2022. [Online]. Available: $en.wikipedia.org/wiki/Stellar\_classification$. [Accessed: 23-Nov-2022].

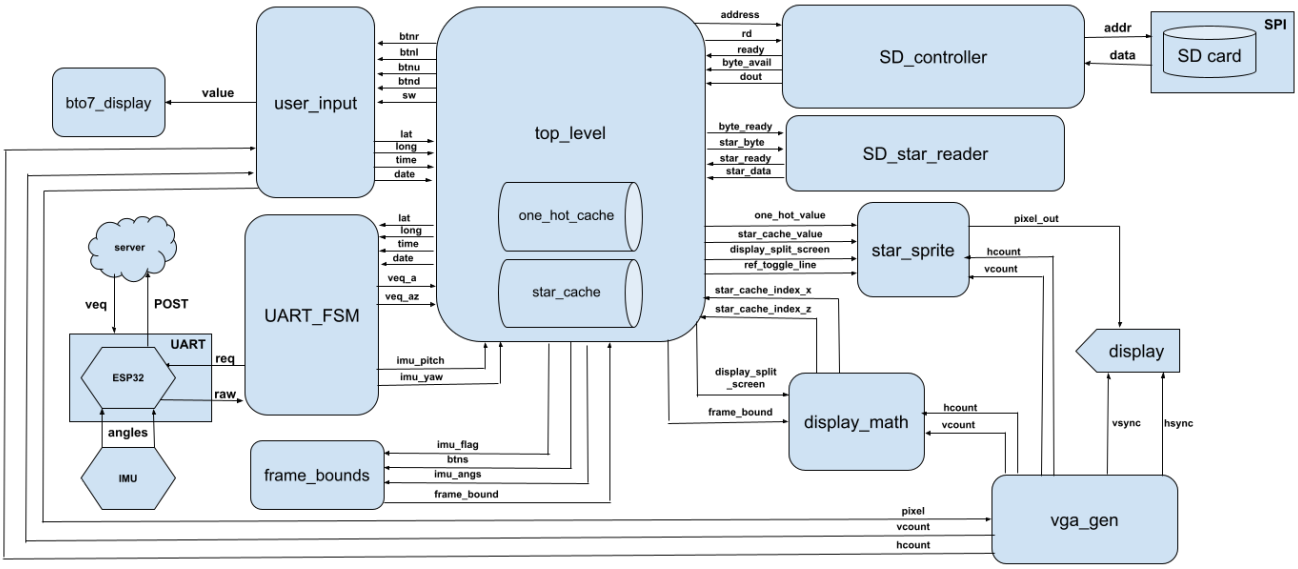

Fig. 1: Project block diagram.

[3] "Apparent magnitude," Wikipedia, 18-Nov-2022. [Online]. Available: $en.wikipedia.org/wiki/Apparent\_magnitude$. [Accessed: 23-Nov-2022].

[4] D. Hoffleit and W. Warren Jr , "The Brightest Stars Database," The brightest stars. [Online]. Available: $atlasoftheuniverse.com/stars.html$. [Accessed: 23-Nov-2022].

[5] "March equinox," Wikipedia, 14-Dec-2022. [Online]. Availale: $https://en.wikipedia.org/wiki/March_equinox$ [Accessed: 14-Dec-2022].

[6] "Horizontal coordinate system," Wikipedia, 14-Dec-2022. [Online]. Availale: $https://en.wikipedia.org/wiki/Horizontal_coordinate_system$ [Accessed: 14-Dec-2022].