FPGA-Based Grid navigating Robot with Direction Commands over Bluetooth

Dhayadharsh S.M
Electronics and Commmunication Engineering
Amrita Vishwa Vidyapeetham
Coimbatore, India
cb.en.u4ece22147@cb.students.amrita.edu

Sreegayathri K.K
Electronics and Communication Engineering
Amrita Vishwa Vidyapeetham
Coimbatore, India
cb.en.u4ece22150@cb.students.amrita.edu

Shiyam Ganesh.M
Electronics and Communication Engineering
Amrita Vishwa Vidyapeetham
Coimbatore, India
cb.en.u4ece22152@cb.students.amrita.edu

***Abstract*— This paper presents an FPGA-Based Real-Time Remote Exploration-based Rover Control system built from scratch. The architecture employs BASYS3 acting as the control unit providing processing based on data obtained from rover. To demonstrate such a setup, a maze exploration and solving application is chosen. This demo includes computing floodvalues, finding shortest path, transmission, rover movement and it sending wall data which is dynamically updated and stored in on-chip Block RAM (BRAM) for new computation and recommand. An HC-SR04 ultrasonic sensor is used to detect walls during navigation, with data transmitted wirelessly to the FPGA via an ESP32 module. The ESP32 also handles Bluetooth-based control and motor actuation through a TB66FNG motor driver. The system supports real-time updates to the maze, allowing for adaptive and efficient pathfinding. This hybrid approach enables reliable hardware-level decision-making while maintaining flexibility through wireless communication.**

Keywords*— *FPGA, Vivado, VHDL, flood-fill algorithm, BRAM, ESP32, HC-SR04, TB66FNG, autonomous rover, Bluetooth control, maze-solving

I. Introduction

In scenarios like extra-terrestrial exploration, where a spacecraft controller manages a rover navigating unknown terrain, it is crucial to design the system with clear functional roles. This project demonstrates such a setup by implementing a hardware-accelerated flood-fill algorithm in VHDL on a BASYS3 FPGA using Xilinx Vivado. The FPGA computes the shortest path and controls a rover that autonomously explores a maze and sends back wall data. Maze wall information is stored and updated in BRAM for quick access. An HC-SR04 ultrasonic sensor connected to an ESP32 detects obstacles in real-time and relays wall data to the FPGA. The ESP32 also controls the TB66FNG motor driver for rover movement and receives movement instructions from the FPGA. This hybrid FPGA–microcontroller system, built from scratch, uses a state-based approach and UART communication to coordinate sensing, decision-making, and actuation. It demonstrates a modular, scalable, and responsive architecture ideal for dynamic environments.

II. Objectives

The main objective of this project is to build a complete control system for a rover (like a Micromouse) from scratch. This means designing everything from the scratch, including how different parts of the system talk to each other, how data is sent and received, and how each part works individually and together. The system is tested using a maze-solving task because it involves both fixed logic and real-time decisions based on the environment. For example, the rover needs to detect walls while moving and adjust its path on the go. The system is designed using a state-based approach, where each part—like sensors, motors, and decision-making—is handled step-by-step in an organized way. This makes the whole system more reliable and robust.

A. System Implementation Goals

- Design and develop a flood-fill maze solving algorithm using VHDL.
- Implement the algorithm on BASYS3 FPGA platform. Integrate on-chip Block RAM (BRAM) to dynamically store and update wall information.
- Interface an ESP32 module with the FPGA to receive wall data in real time.
- Use the HC-SR04 ultrasonic sensor on the ESP32 for detecting walls. Enable movement control of the micromouse via TB66FNG motor driver.
- Ensure the design supports real-time path computation and maze exploration.

B. Functional Requirements

- The flood-fill algorithm must compute the shortest path based on explored regions of the maze.
- Wall data is dynamically received and used to update maze state in BRAM.
- The system must send wall detection alerts from FPGA to ESP32 via Wi-Fi. The movement logic must respond to the shortest path output in hardware.
- Support for 8-bit command input for movement .
- Bluetooth-based control and monitoring through an external ESP32.
- Efficient real-time memory access and logic operations in the VHDL design.

III. Hardware Setup

The hardware implementation of the rover integrates multiple modules to achieve real-time wall detection, path planning, and movement control.

A. Module flow description

The main module connects many submodules like FloodFill, Shortest Path Processor, Wall Update(which also contains receiver), New UART Transmitter, BRAM modules and a simple module for Seven Segment control.

The overall system uses two clocks **clka, clkb**. As BRAM read operations have one cycle latency, there was a decision to be made between including additional wait states or an idea of **running the BRAM in twice the frequency of algorithm** clock. The latter idea was implemented with clka being the system clock with 100 MHz and clkb being the algorithm clock of frequency 50MHz.

Each of the different modules are explained below:

1. Flood Fill Module :
   - Solves the maze using flood-fill algorithm.
   - Writes distances from goal to all cells.
   - Uses a circular queue (Understood the actual use of it here).
   - Interfaces with BRAM to read/write distance values.
   - Starts on signal start, ends with signal done.

2. ShortestPathProcessors :

Start from Initial Cell:
   - Initial position: (x, y) (user-provided).
   - Next positions are derived from the number of blocks covered by the rover along the shortest route (next_x, next_y).

Explore Four Neighboring Cells:
   - Check all four directions: North, East, South, West.
   - Compare their values to find the minimum.

One-Hot Encoding for Minimum Direction:
   - Store the direction(s) with the least value as a binary one-hot code.
   - Example: 1000 for North, 0100 for East, etc.
   - If multiple directions have the same minimum value:
   - "Or" their bits → e.g., North + East = 1100.
   - This is stored as variable min_dir.

Decision Based on min-dir Bit Count:
- Only 1 bit set (1 direction):
  - Choose that direction.
  - Add the cell to the queue.
- More than 1 bit set (multiple equal directions):
  - Apply preference logic to resolve.

Preference Logic for Multiple Options
- Preference 1 : Prefer unexplored directions
  - If [min-dir AND Exploration-bits(refer BRAM1 structure)] == 0:
  - All the Min Directions are unexplored, so no choice can taken.
  - If [min-dir AND Exploration-bits(refer BRAM1 structure)] == min_dir:
  - The multiple Min Directions are all explored, so no choice can taken.
  - Else unexplored direction is [min-dir AND Exploration-bits].
  - This was possible only due to the data format designed specifically.
- Preference 2: Prefer path opposite to arrival. For this had to store prev_dir.
- Preference 3: Choose in the order NEWS.
- The data is stored in BRAM2, which will be used by UART_TX module for sending data to rover.

3. Wall Update :

i. Runs in Multiple States:

This entity (function/module) runs across different stages:
   - Starts in the Shortest Path Processor (SPP).
   - Ends in the Wall Update stage.

ii. Wall Update Receives Direction Info:
   - It contains a wall-update RX (receiver).
   - After SPP computes the direction, this unit:
     - Converts the direction into coordinates.
     - Stores those coordinates in a queue.

iii. Waits for Input from Rover:

Once the direction is stored in the queue, it waits for sensor input from the robot/rover (e.g., wall detection).

iv. Wall Info Reception & Storage:
   - On receiving data from the rover:
     - It stores wall presence info into the grid cell corresponding to the coordinates in the queue.
     - It then updates neighbouring cells accordingly.

v. Omni-directional Wall Representation:
   - For example:
     If cell [0][1] has a wall in the South,
     Then cell [0][2] must have a wall in the North.
   - This ensures consistency in wall representation between adjacent cells.

vi. Wall Update Loop Continues Until:
   - One of the following conditions is met:
     - A wall pattern like 1111 1111 is received (indicating all walls around a cell).
     - The last updated cell becomes the next (x, y) of SPP.
     - A specific value like 1010 1010 is reached, indicating goal achieved.

4. New UART Transmitter:
   - Sends direction and position data over UART (tx) stored in BRAM2.
   - Uses trigger to start transmission.
   - Also outputs when transmission is done (done_c_sig).

5. BRAM Blocks :

   There are two instances of BRAM:
   - BRAM1 for maze solving
   - BRAM2 for storing TX data of shortest path
   - Refer Data format for more information

   This is the only module running directly on system clock

6. Key Signals and States

- Flood Fill Control

start, done, ena_ff, wea_ff, etc. are used to manage Flood Fill execution.

- Wall Update Control

ena_wu, wea_wu, dina_wu, etc. handle BRAM writes when wall data is received.

- Shortest Path Control

start_sp, tx_ready, directions, and related BRAM signals are for shortest path processing.

- Maze Cell Position

c_x, c_y → Current cell
next_x, next_y → Next move after update
goal_x, goal_y → Static goal cell (hardcoded as 2,2)

- Goal Checking

reached_goal_in, destination_reached help determine when the robot has completed the maze.

1. *States Flow*

The state flow is explained with TABLE 1 which tells what each state takes places. All those states belong to the module, and they form a state machine that coordinates the other submodules like FloodFill, ShortestPathProcessors, Wall_Update, and UART TX. The Fig. 1 explains the state flow.

State	What it does
IDLE	Waits for switch to start everything.
WRITE_CELL	Loads initial maze data into BRAM.
PRE_WRITE_CELL	Waits or prepares next write cycle.
START_FLOODFILL	Triggers the FloodFill module.
WAIT_FLOODFILL	Waits for flood fill to finish (done = '1').
READ_BRAM	Reads the updated BRAM for verification or next steps.
SHORT_PATH	Starts shortest path processor module and Wall_update module.
UART_NW	Prepares to send data
TEMPO	Shortest path is transmitted
WALL UPDATE	Runs the Wall_Update module with received wall data.
DONE_C	Finishing state; may reset or wait for next trigger.

TABLE 1

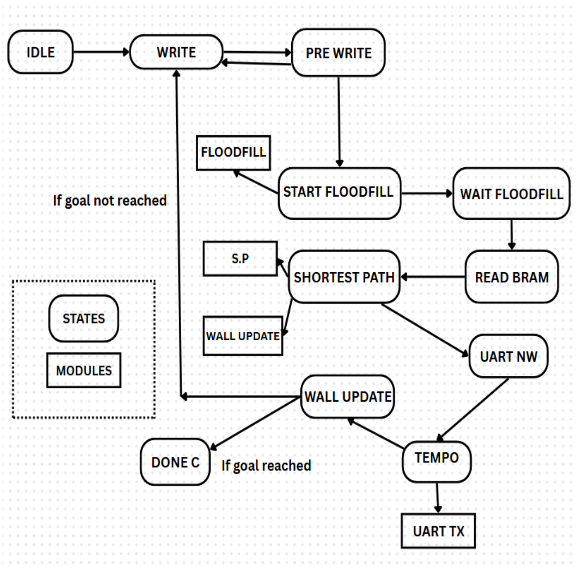


Fig. 1

B. *Data Format*

1. BRAM 1
   - Size: 16 bits per cell
   - Wall Info (NEWS):MSB 4 bits N (North), E (East), W (West), S (South)
   - Previous Visited Direction (NEWS): 4 bits Stores the direction from which the cell was entered.
   - Flood Fill Value :LSB 8 bits Used in pathfinding to store cost/distance values.
   - This storage pattern enabled us to byte wise write data reducing the need to copy and change only the required bits.

2. BRAM 2
   - Size: 16 bits
   - Though only 4 bits are actually needed, 16 bits are used in reusing the same BRAM used for maze storage.

3. TX (Transmit) Data – 8 bits

The MSB four bits determine what type of data is sent:
   - Direction: 0000 → one-hot encoding for NEWS
   - Start Signal: 0000
   - Path-End Indicator: 1111 → signifies end of path transmission.
   - FPGA sends direction and control signals to bot
4. RX (Receive) Data – 8 bits
   - Wall Info: 0000 → direction-based (NEWS format)
   - The LSB contains the wall info of the particular cell (1100 means wall in N & E)
   - Obstacle : MSB 1111
   - Goal Reached Signal: MSB 1010
   - Bot sends wall info, obstacle, or goal status to FPGA

5. Baud Rate = 115200
   Uart communication

C. *ESP32 Microcontroller with rover*

The ESP32 acts as a communication bridge and motor controller. It receives movement commands from the FPGA in binary format. It also:

- Sends wall detection signals from the HC-SR04 sensor to the FPGA for updating maze values.
- Controls the TB66FNG motor driver to move the rover accordingly.
- Handles Bluetooth communication for remote control and path updates from a phone or another ESP32.

1. FPGA to ESP32: Direction Transmission

- FPGA sends direction commands to ESP32 via Bluetooth.
- Directions are encoded in 1-byte format like:
  - 0x08 → North
  - 0x04 → East
  - 0x02 → West
  - 0x01 → South
- ESP extracts direction from the last 4 bits of received data.

2. ESP32 Controls Motors Using PWM
   - ESP32 uses PWM to control motor speed and direction.
   - Uses ledcAttachChannel() for PWM setup.
   - Separate PWM logic is written for each movement direction (N, E, W, S).
3. Wall Detection via Ultrasonic Sensors
   - Wall detection threshold is manually set (e.g., 7 cm).
   - Prevents collision while turning.
   - Threshold is based on bot size and safe turning space.
4. Bluetooth Data Format
   - FPGA sends a sequence of directions (e.g., 0x04, 0x08, 0x02, ...).
   - 0xFF indicates the end of the path or that the goal is reached.
   - ESP begins executing the received directions.
5. Absolute Direction Mapping
   - Directions are mapped to global directions (N, E, W, S).
   - Not based on bot's relative heading (like left/right).
   - Ensures consistent movement regardless of bot's orientation.
6. Fixed Movement Using Encoder Ticks

   Motors produce fixed number of encoder pulses per revolution.ESP uses interrupt pins to count pulses and measure movement.Allows bot to move exactly 1 cell at a time.
7. Calibration
   - Wheel diameter is measured → distance per wheel revolution is calculated.
   - Number of ticks per unit distance is fixed.
   - Turning ticks for left/right (90°) are also calibrated.
8. Turning Angle Calculation
   - Arc length for turning: $\pi \times$ wheelbase $\times$ angle / 360
   - Ticks needed for turning: arc length $\times$ PPR / wheel circumference
   - No need for an MPU to achieve precise turns.
9. Sending Wall Data to FPGA
   - ESP sends wall presence data back to FPGA:
     - Format: 0000xxxx → each bit = one sensor's detection.
     - Example: 00000001 = front wall 00000101 = front + left walls
   - FPGA updates cell wall data accordingly.
10. Bot Checks Walls While Moving
    - While moving in a direction:
      - If wall is detected → Send stop signal (0xFF) → Wait for new directions.
      - If no wall → Move forward
11. Shortest Path Loop
    - FPGA calculates and transmits shortest path.
    - Bot keeps:
      - Following direction
      - Checking walls
      - Sending updates
      - Receiving new directions
    - Process continues until the bot reaches the goal (0xFF).

D. HC-SR04 Ultrasonic Sensor

- This sensor is used for real-time wall detection in the maze.
- Mounted on the front, right and left of the rover.
- Continuously measures distance to obstacles.
- If a wall is detected within a certain threshold (e.g., <10 cm), the ESP32 encodes this as wall information and sends it to the FPGA to update the BRAM.

E. TB66FNG Motor Driver

The TB66FNG is a dual H-Bridge motor driver IC used to control two DC motors independently. It receives control signals from the ESP32 to actuate the rover's motors.

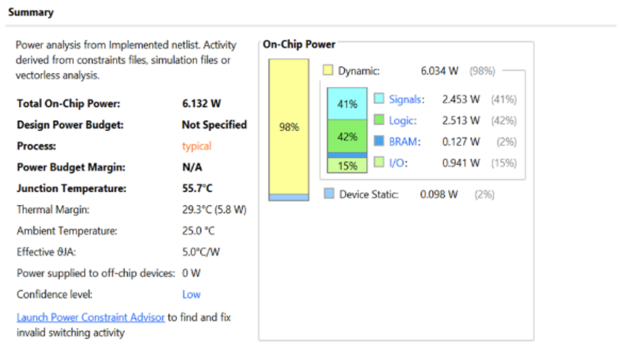
- Controls two DC motors with forward/reverse control with encoders.
- Supports PWM (Pulse Width Modulation) for speed adjustment.
- Handles up to 15V motor supply voltage and provides ~1A current per channel.
- Includes built-in thermal shutdown and overcurrent protection.

F. Challenges faced and lessons learned

Challenge	**Solution**
Implementation of sequential algorithms in hardware	Effective use of FSM. Handling of operations and assignment systematically.
Transitioning from C-based (sequential) to HDL (parallel) coding	Practiced concurrent thinking; designed parallel modules to understand timing, clocking, and signal flow.
Signal type array not being inout (needed for maze storing and reading)	Using BRAM IP
Two cycle reading latency of BRAM.	Ran the BRAM clock twice the algorithm clock's frequency
Need to copy, change and write back in memory for partial changes.	Use of proper structuring of BRAM to use the byte wise write feature
BRAM read/write access and handling memory collisions	Used signal control and arbitration logic to avoid unintentional concurrent access.

Post-implementation struggles (getting simulated correctly but implemented model lacking full structure)	Proper assignment of variables in all possible branches of the states.
Triggering issues across multiple modules	Various triggers like 1 – cycle trigger, multicycle trigger, edge-triggers based signal duration
Multiple drivers for BRAM causing conflicts	Designed a mux-based system to manage input control and ensure only one active driver at a time and proper FSM structure.

IV. RESULTS AND DISCUSSIONS



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

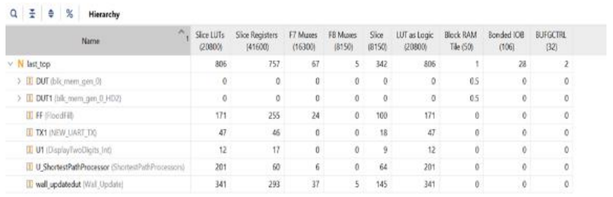
Total On-Chip Power:	**6.132 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**55.7°C**
Thermal Margin:	29.3°C (5.8 W)
Ambient Temperature:	25.0 °C
Effective θJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	6.034 W	(98%)
Signals:	2.453 W	(41%)
Logic:	2.513 W	(42%)
BRAM:	0.127 W	(2%)
I/O:	0.941 W	(15%)
Device Static:	0.098 W	(2%)

Figure 3-Power Analysis Summary



Hierarchy

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
last_top	806	757	67	5	342	806	1	28	2
DUT (blk_mem_gen_0)	0	0	0	0	0	0	0.5	0	0
DUT1 (blk_mem_gen_0_HD2)	0	0	0	0	0	0	0.5	0	0
FF (FloodFill)	171	255	24	0	100	171	0	0	0
TX1 (NEW_UART_TX)	47	46	0	0	18	47	0	0	0
U1 (DisplayTwoDigits_Int)	12	17	0	0	9	12	0	0	0
U_ShortestPathProcessor (ShortestPathProcessor)	201	60	6	0	64	201	0	0	0
wall_updatedut (Wall_Update)	341	293	37	5	145	341	0	0	0

Figure 4-Resource Utilization Report

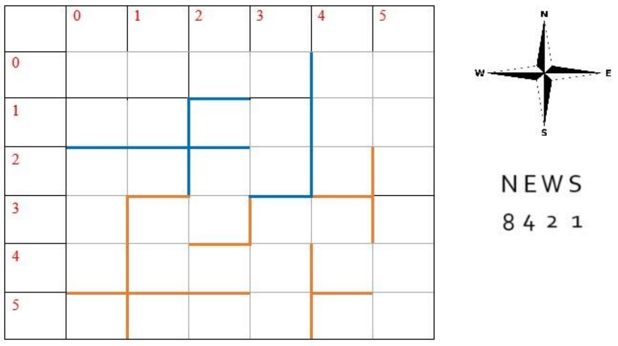


Figure 5- Sample maze

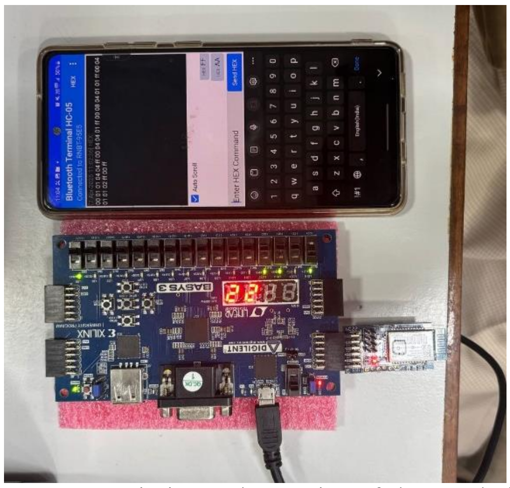


Figure 7 Transmission and reception of short path demo

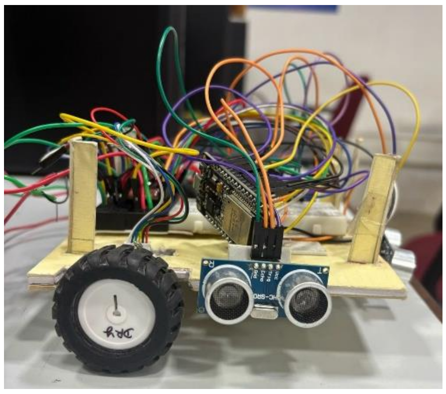


Figure 8- Maze Bot

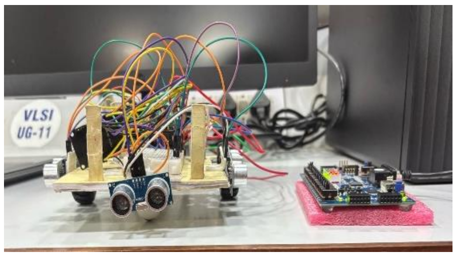


Figure 9 FPGA interfaced with bot using Bluetooth

V. CONCLUSION

The project was a great exercise in a true electronic system level design. This demonstration helped to understand a variety of strategies that can be used to make the electronic system implementable. This project also exposed the struggles faced by verification team in order to debug such an highly modular interconnected system. To reiterate, the entire system right from its individual states, modules, data formats of storage , reception and transmission were hand crafted for this specific rover based exploration application. The project was implemented and the results have been successfully recorded for future reference

TABLE I.

RESOURCE UTILIZATION

Entity	LUTs Used	Registers Used	Special Features
TOP	806	757	BRAM(2)and all modules together
Floodfill	483	255	Calculated cell values
Wall Update	550	293	Obstacles detection and reception
Shortest Path	641	60	Transmission

REFERENCES

[1] Nunes, U. et al. (2004). *A Micromouse Implementation Using a Modified Flood-Fill Algorithm and Wall Detection with Sensors*. *IEEE Transactions on Education*, 47(3), 430–435.

[2] IEEE Micromouse Technical Committee. (2019). *Rules and Design Guidelines for Micromouse Competition*.