The Fall Guardian

Abstract

Falls are a significant health risk, especially for children and elderly individuals, which often lead to injuries and require immediate assistance. *Fall Guardian* is a dual FPGA based fall detection and response system designed to provide real-time monitoring and emergency communication. The child FPGA is equipped with an accelerometer to detect falls, an OLED display to provide visual feedback, and a keypad for user confirmation. The parent FPGA monitors the child device and responds to help requests via Bluetooth Low Energy (BLE). The system effectively detects falls, confirms help requests, and ensures rapid response through reliable communication between the child and parent devices.

1. Introduction

Falls are one of the leading causes of injury among vulnerable populations such as children and the elderly. According to the World Health Organization (WHO), falls are the second leading cause of unintentional injury deaths worldwide, with over 37 million falls each year requiring medical attention [1]. Timely assistance after a fall can significantly reduce the severity of injuries and improve recovery outcomes.

To address this issue, we designed *Fall Guardian*, a dual FPGA based system that detects falls in real-time using an accelerometer and enables immediate communication through BLE. The system consists of a child FPGA that detects falls and sends alerts, and a parent FPGA that monitors the child's status and responds to help requests. The use of an OLED display for feedback and a keypad for user input ensures that the system remains user friendly and responsive.

2. Related Works

Existing fall detection systems often rely on machine learning algorithms and smartphone-based accelerometer data to identify sudden changes in motion [2][3]. While these solutions can provide high accuracy, they are typically complex and require significant computational resources, making them challenging to implement on low-cost embedded systems.

Our project, developed within a short three-week timeframe, takes a more straightforward approach. It combines FPGA-based processing with direct threshold-based detection using an accelerometer. The system verifies user status through a keypad and communicates via BLE to ensure a quick response.

As our fall detection mechanism relies only on an accelerometer, it was challenging to define specific thresholds for detecting a fall. We had to fine-tune the accelerometer data through trial and error to help us achieve consistent detection while minimizing false positives. We also

implemented a user confirmation step through the keypad and OLED display, ensuring that help requests are only sent when truly needed. This feature enhances the system's reliability and prevents unnecessary emergency responses.
While the project lacks the complexity of commercial products, it demonstrates that a simple FPGA-based design can still effectively handle fall detection and real-time communication through BLE.

3. Implementation

3.1 Architecture

As mentioned earlier, this project consisted of two FPGAs. One labeled as "child" (the one that detects the falls) and the other labeled as "parent" (the one that monitors if the child fell). The child FPGA is connected to 4 peripheral modules, which are Pmod BLE, Pmod KYPD, Pmod ACL2, and Pmod OLEDrgb. On the other hand, the parent FPGA is connected to only 3 peripheral modules, which are Pmod BLE, Pmod KYPD, and Pmod OLEDrgb. The ACL2 Pmod is not used here because fall detection is not used for parents, only for children. Figure 1 and 2 below show a high level block diagram of the child and parent FPGAs, respectively.

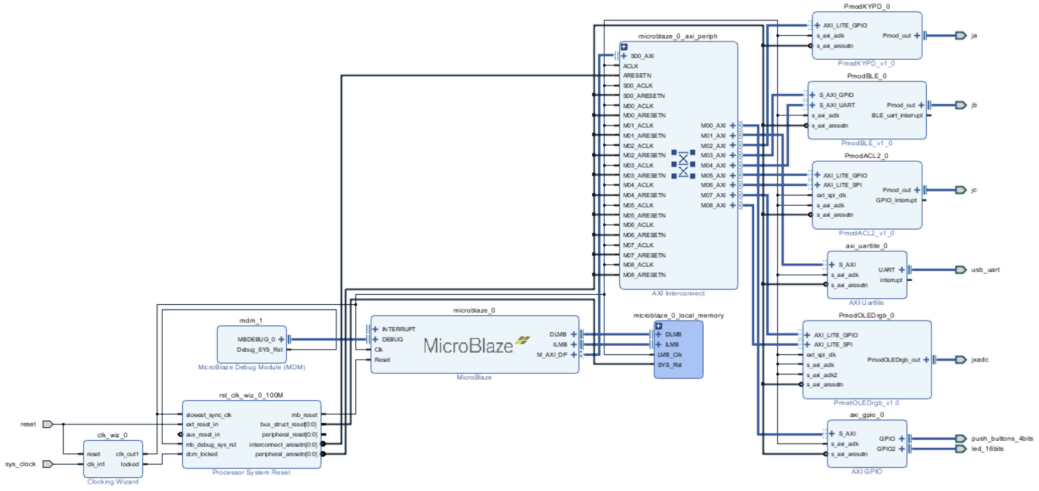

Figure 1: Block diagram for child.

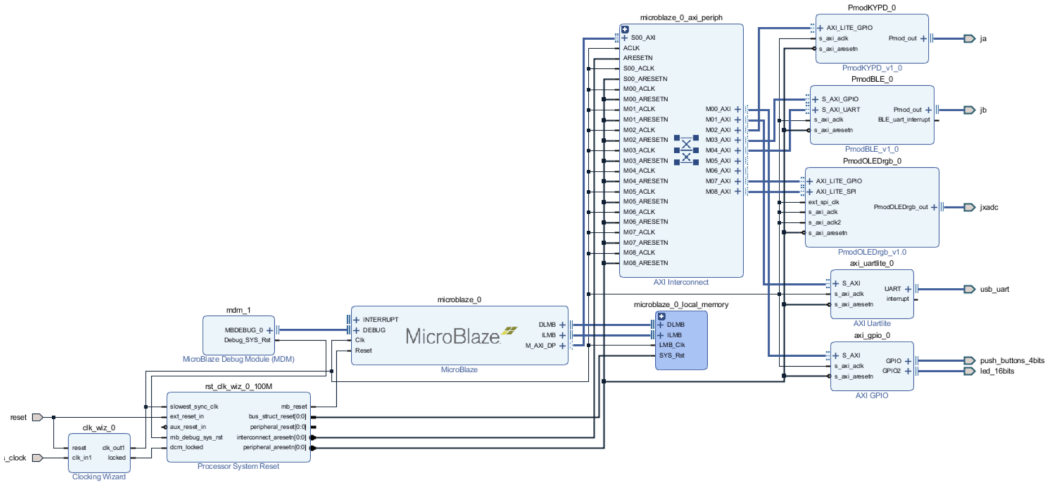

Figure 2: Block diagram for parent.

3.2 Development Details

This project was developed using C completely. This is done through the MicroBlaze soft processor on the FPGA. In order to reach the C software development kit for the FPGA, the following steps were taken:

- The high level block diagrams in figure 1 & 2 were designed using the Vivado Block design editor.
- Block diagrams were connected to the right connectors (JA, JB, JC, etc) and validated.
- Synthesis and bitstream generation.
- HDL wrapper created.
- Hardware exported and SDK launched.
- FPGA programming.
- Create a new application project and add source files.

After the above steps, C code was ready to be executed and tested on the FPGAs. For debugging, the Putty terminal was used in combination with various xil_printf statements to make sure that all state transitions were accurate and data received was as expected.

3.3 State Machines

This contains two state machines, one for the child FPGA and one for the parent FPGA. Figure 3 shows the child state machine. The first 4 states are related to the connection and address input part for the BLE functionality and interacting with the BLE. The last 3 states include a FALL DETECTION STATE where the device constantly checks the ACL2 numbers to check if they are exceeding the falling threshold. The FALL CONFIRM STATE is where the child confirms that they need assistance. This is where the BLE would send the help request message if needed. Lastly, the HELP CONFIRM STATE is where the child receives that the parent responded to their help request.

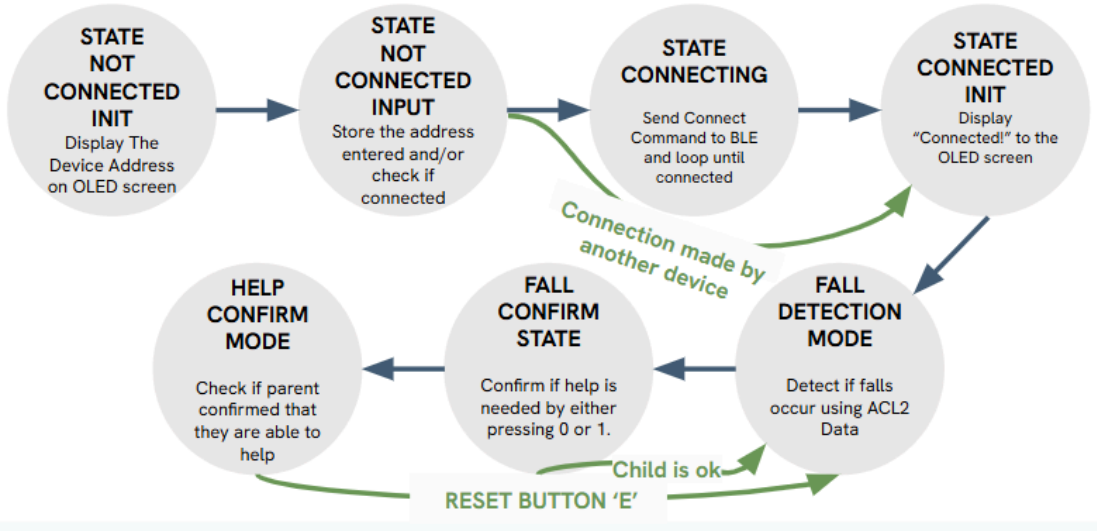

Figure 3: Child State Machine

Similarly for the parent, Figure 4 shows the state machine used for the parent FPGA. After the 4 connecting states, the device joins the FALL MONITOR STATE where it is detecting if any device is sending a help request. Once a help request is received, the device enters the HELP SEND MODE state, where it sends a confirmation message informing the recipient that they are able to assist.

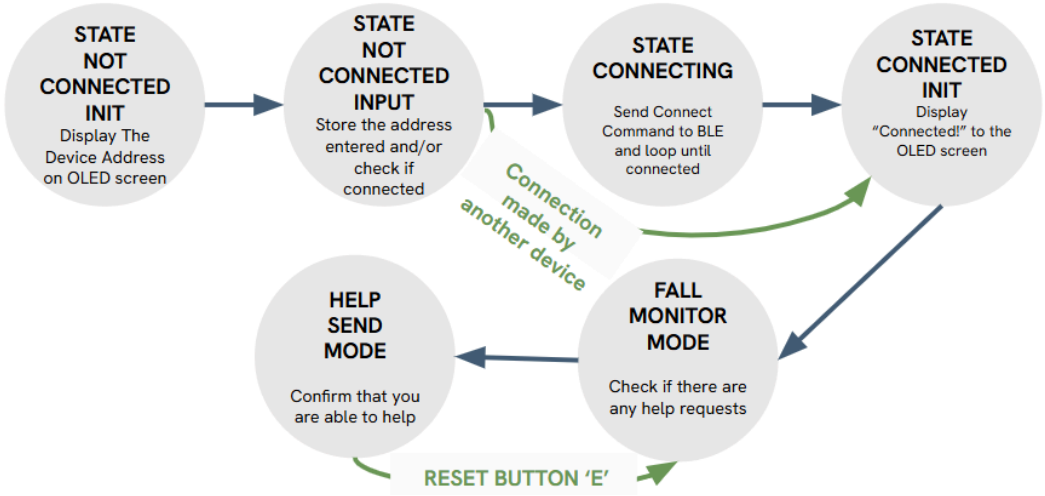

Figure 4: Parent State Machine

3.4 Software

As mentioned earlier, this project was programmed using C. To interact with the attached Pmods shown in Figure 5, multiple functions were used from the Vivado Library [4]. For the BLE Pmod, functions were also used from the BLE_interface.c file provided by the BLE_OLEDrgb project from previous years [5].

Figure 5: Used Pmods for this project.

Some of the functions used in the development of this project include:

- From PmodBLE.h: BLE_RecvData, BLE_IsConnected, BLE_SendData.
- From PmodBLE_interface.h: PmodBLE_Initialize, PmodBLE_GetDeviceAddress, PmodBLE_ConnectTo, PmodBLE_SendMessage, PmodBLE_ReceiveMessage.
- From PmodKYPD.h: KYPD_begin, KYPD_loadKeyTable, KYPD_getKeyStates, KYPD_getKeyPressed.
- From PmodACL2.h: ACL2_begin, ACL2_reset, ACL2_configure, ACL2_getStatus ACL2_DataToG, ACL2_end
- From Pmod OLEDregb.h: OLEDrgb_begin, OLEDrgb_DefUserChar, OLEDrgb_GetCursor, OLEDrgb_Clear, OLEDrgb_PutString, OLEDrgb_SetCursor.

4. Results

The results of this project are shown using the OLEDrgb screen. Additionally, for debugging, serial communication shown through Putty was used (Figure 6).

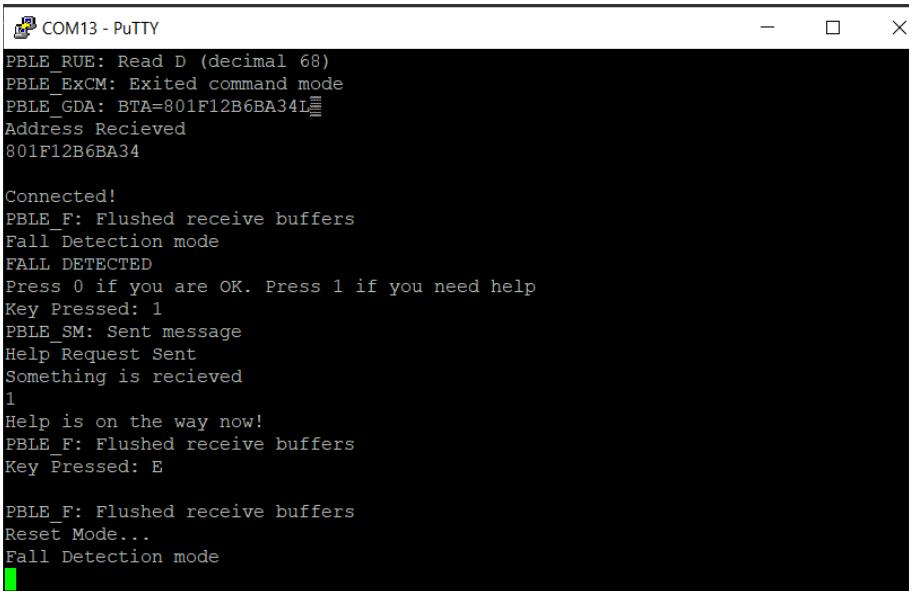

Figure 6: Debugging through serial communication.

Initializing the BLE connection can happen both ways. Assuming a scenario where the child connects to the parent, the setup would start by displaying the address of each device on the OLEDrgb screen (Figure 7).

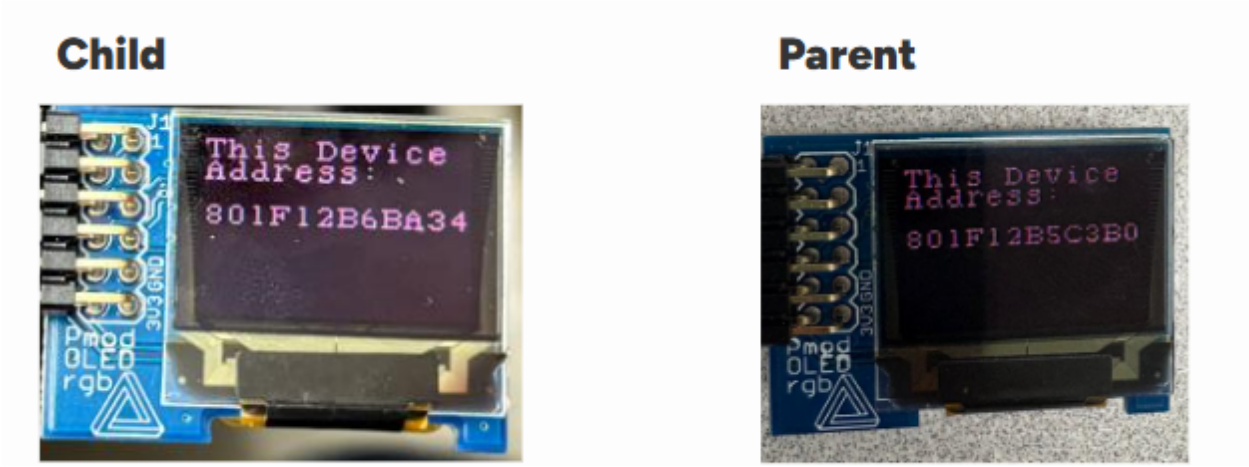

Figure 7: Child and Parent OLEDrgb screen responses.

After entering the address of the parent using the child KYPD, the child would enter a connecting state (Figure 8).

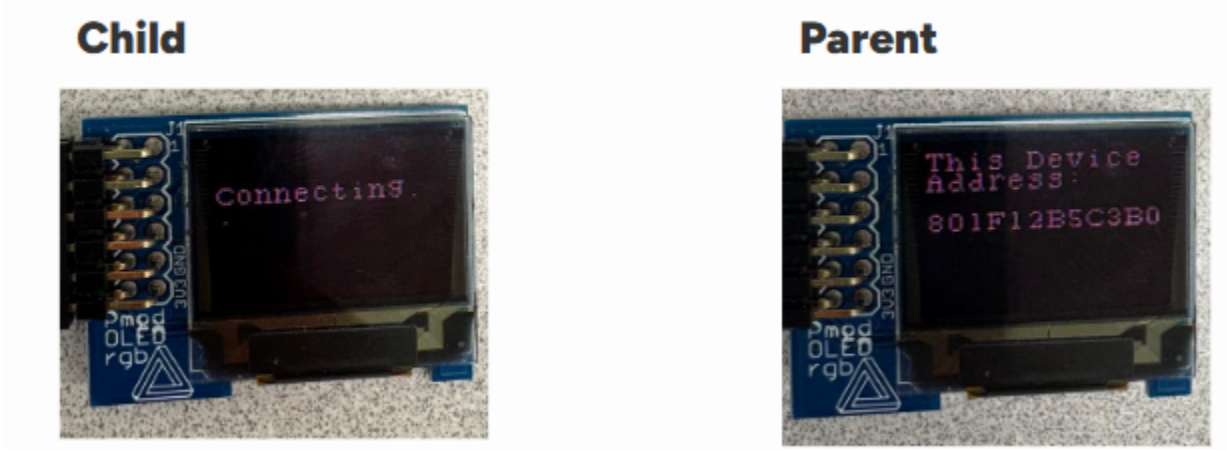

Figure 8: Child and Parent OLEDrgb screen responses.

Once the child reaches the connected state, the parent would also enter the same state as well (Figure 9).

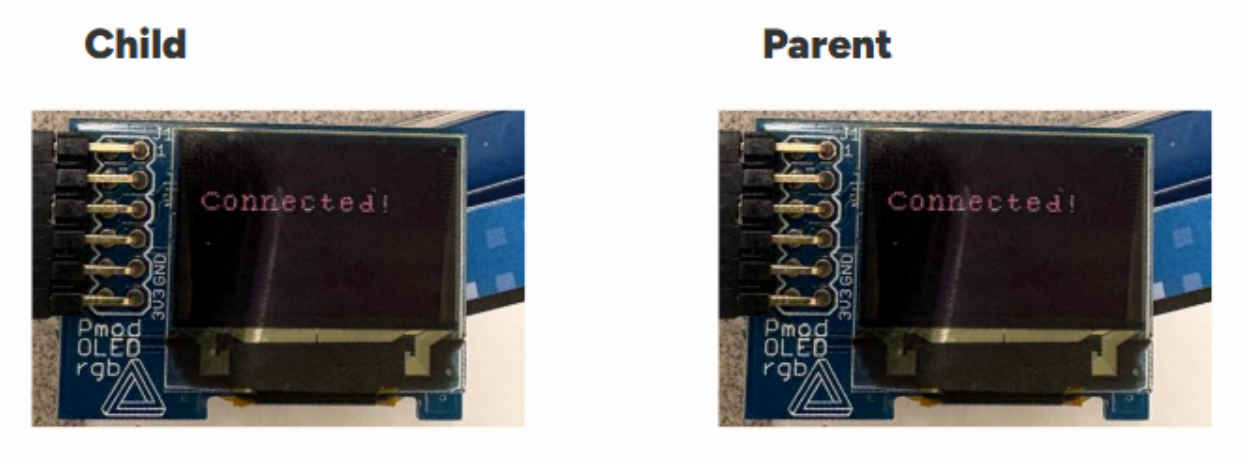

Figure 9: Child and Parent OLEDrgb screen responses.

The child now enters the FALL DETECTION STATE. On the other hand, the parent enters the FALL MONITORING STATE (Figure 10).

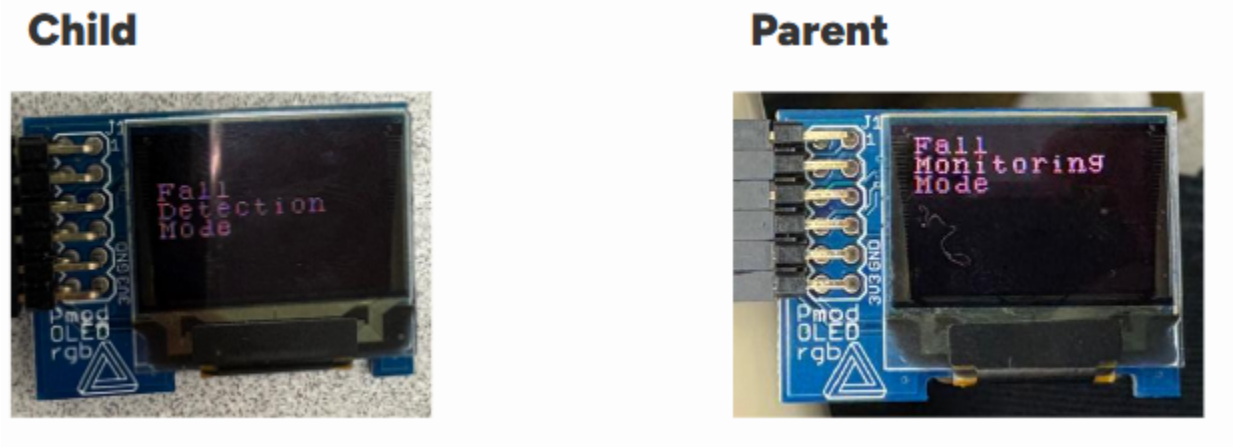

Figure 10: Child and Parent OLEDrgb screen responses.

Once the child starts accelerating downwards with a z component lower than -0.3 and an x and y component higher than 0.2, a fall is detected and shown on the screen (Figure 11).

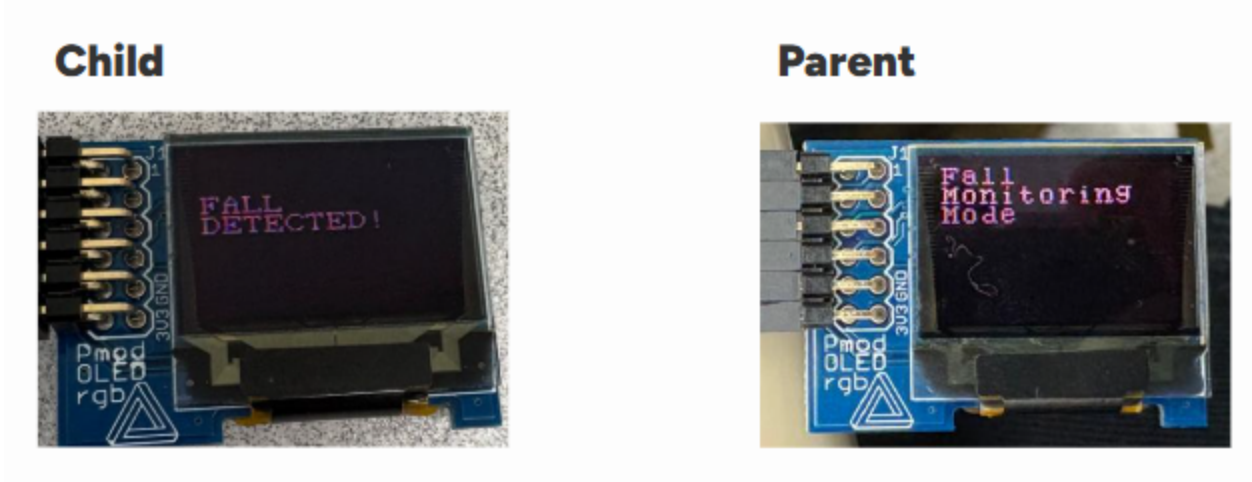

Figure 11: Child and Parent OLEDrgb screen responses.

After a fall is detected, the child enters the FALL CONFIRM STATE where they confirm whether they need help or not using the KYPD (Figure 12).

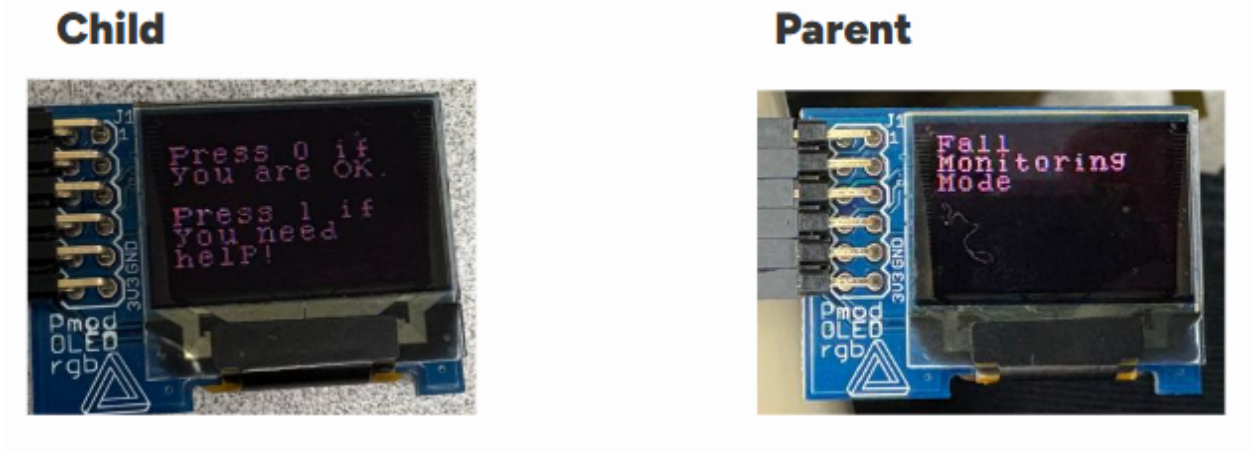

Figure 12: Child and Parent OLEDrgb screen responses.

If the child presses ‘0’, then the device just goes back to the fall detection state without sending any help request. Figure 13 below shows the message shown (“Glad you are OK!”).

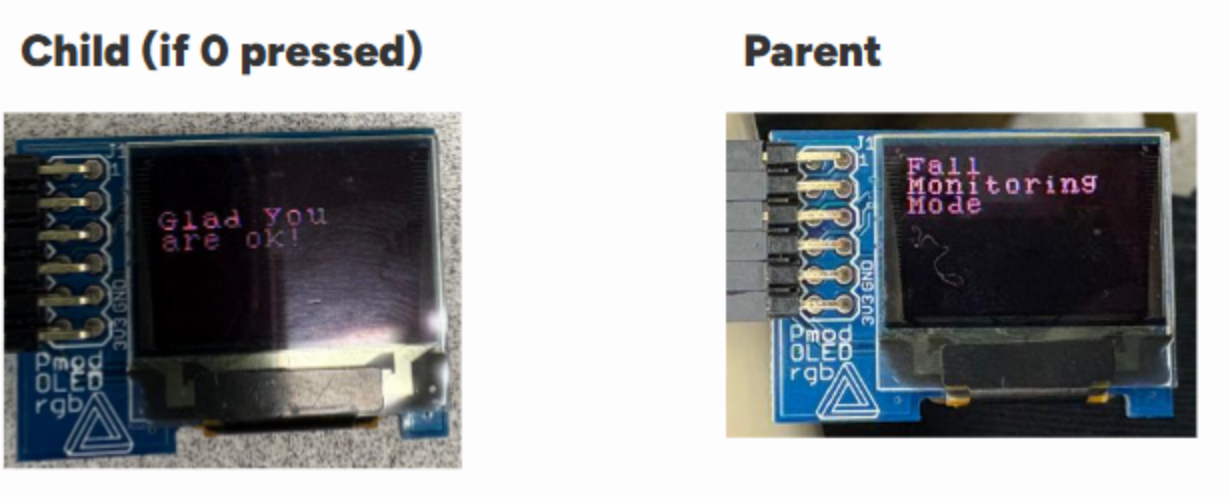

Figure 13: Child and Parent OLEDrgb screen responses.

However, if the device presses 1 (help request confirmed!), the parent would immediately receive the request and is asked to confirm if they are able to provide help (Figure 14).

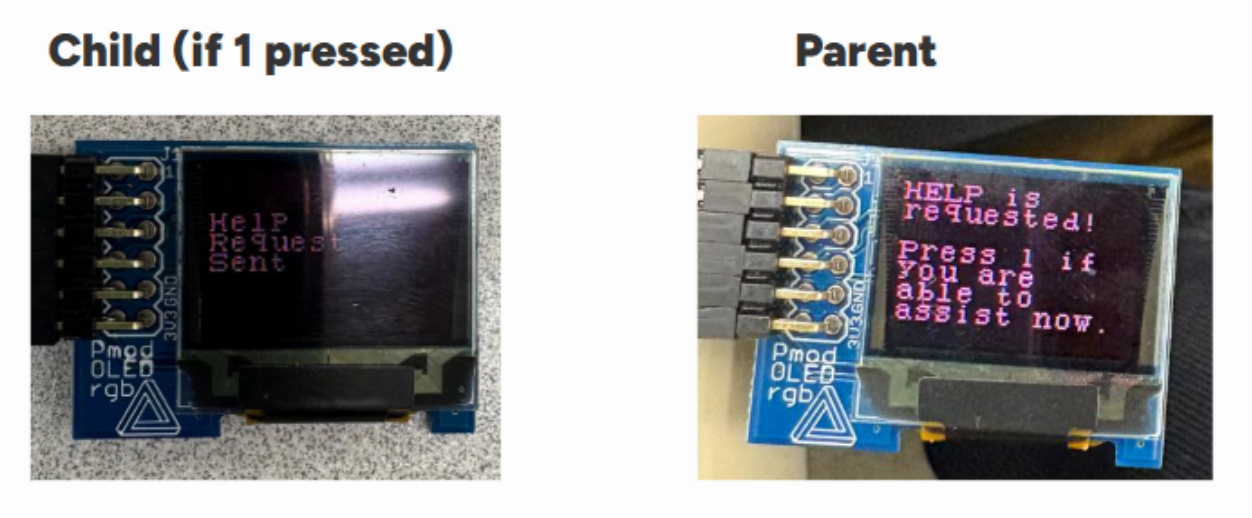

Figure 14: Child and Parent OLEDrgb screen responses.

Once the parent confirms that they are able to assist, the child is reassured that help is on the way. The parent then moves back to the fall monitoring state (Figure 15).

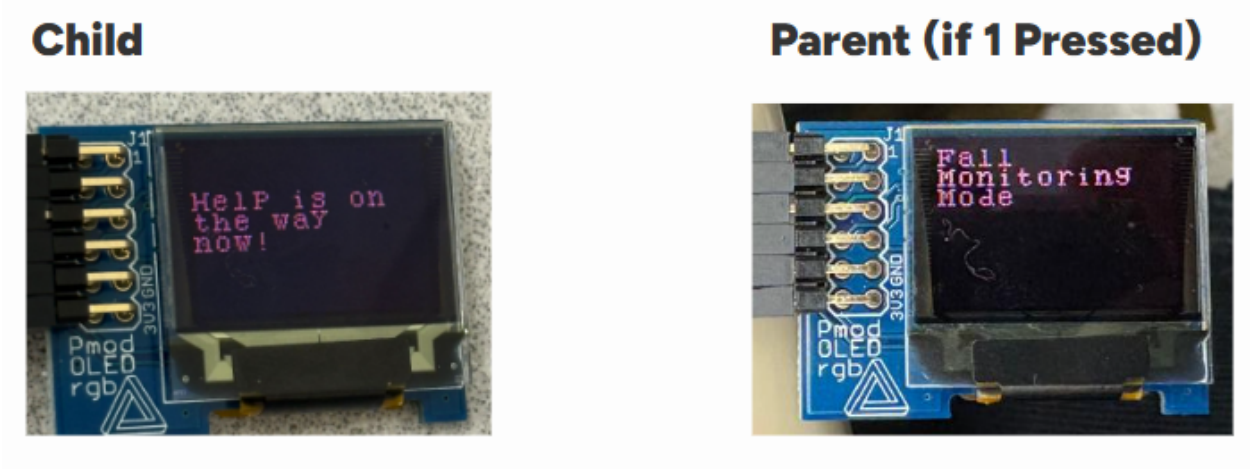

Figure 15: Child and Parent OLEDrgb screen responses.

Lastly, in order to reset the child (this works on the parent as well), all you need to do is to press ‘E’ on the KYPD and it will move to the fall detection state (Figure 16 & 17).

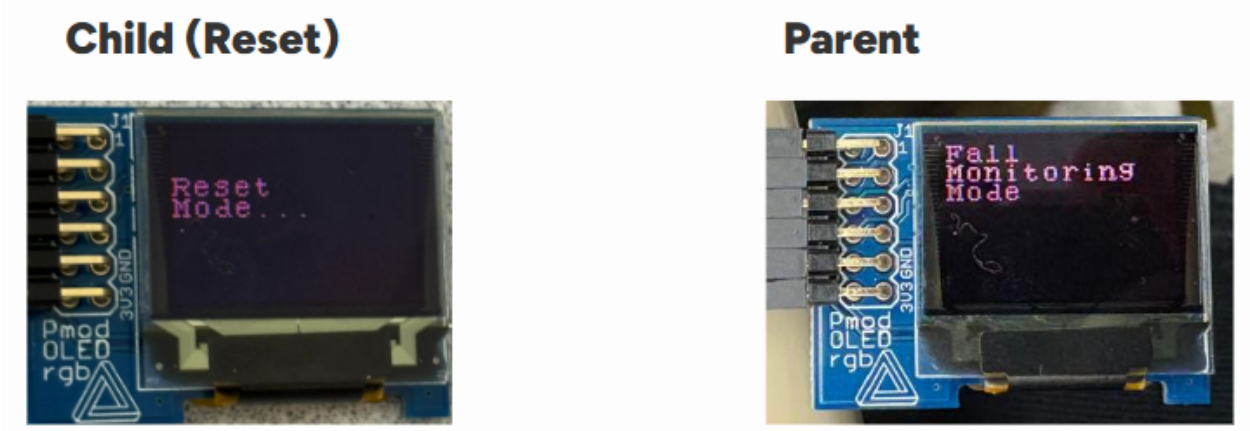

Figure 16: Child and Parent OLEDrgb screen responses.

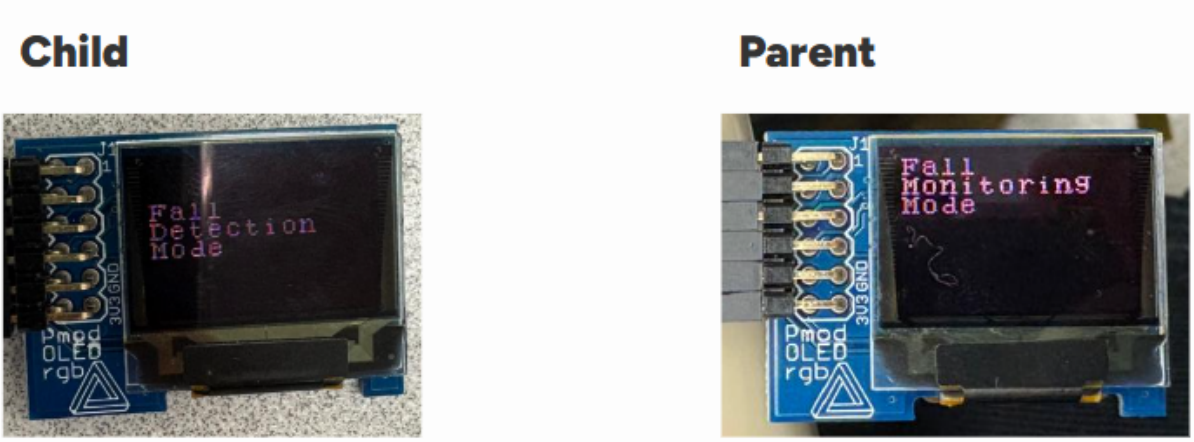

Figure 17: Child and Parent OLEDrgb screen responses.

5. Conclusion

Fall Guardian successfully demonstrates a simple yet effective FPGA-based fall detection and response system. By combining an accelerometer for motion detection, a keypad for user input, an OLED display for feedback, and BLE for communication, the system provides a reliable solution for real-time fall monitoring. The dual-FPGA setup allowed seamless communication between the child and parent devices, ensuring that help requests are processed efficiently.

The project's strength lies in its straightforward design and real-time performance. Despite the limited development timeframe, the system achieved consistent fall detection and effective user confirmation, minimizing false alarms. The use of BLE ensured quick and reliable communication, while the OLED and keypad interface enhanced user interaction.

Future improvements could include adding additional sensors for more accurate motion tracking, extending communication range using wifi or cellular, and replacing the keypad with a touchscreen to improve user experience and reduce device size. Overall, *Fall Guardian* provides a solid foundation for further development in fall detection and emergency response systems.

6. References

[1] World Health Organization (2021, April 26). Falls. https://www.who.int/news-room/fact-sheets/detail/falls#:~:text=Falls%20are%20the%20second%20leading,greatest%20number%20of%20fatal%20falls.

[2] Frontiers (2022, July 13). Wearable Sensor Systems for Fall Risk Assessment: A Review. https://www.frontiersin.org/journals/digital-health/articles/10.3389/fdgth.2022.921506/full?utm_source=chatgpt.com.

[3] B. Saha, S. Islam, ectal. BlockTheFall: Wearable Device-based Fall Detection Framework Powered by Machine Learning and Blockchain for Elderly Care. *2023, July 10.*

[4] Digilent. (n.d.). *Digilent/Vivado-Library*. GitHub. https://github.com/Digilent/vivado-library/tree/master

[5] Previous Students. (n.d.). *BLE_interface.c* In *BLE_OLEDrgb.zip* (Published project code). BruinLearn.