Integrating RISC-V Processor with an FPGA

Manoj Kumar G.Sivakumar
Faculty of Engineering
Multimedia University (MMU)
Cyberjaya, Malaysia

***Abstract*— This paper presents the successful integration of the NEORV32 RISC-V processor core onto an FPGA platform, specifically the Intel Altera De2i-150 board. The integration process involved setting up a comprehensive development environment, configuring the core, and uploading it to the FPGA. The processor's performance was validated through a practical application involving a holonomic X-drive mobile robot, which demonstrated the processor's capabilities in handling complex computational tasks such as real-time processing of encoder data, inverse kinematics, and PID control.**

Keywords—RISC-V, FPGA, SoC, Computer architecture, holonomic drive, VHDL

I. INTRODUCTION

The integration of processors on Field-Programmable Gate Arrays (FPGAs) has been a significant area of research and development in the field of electronics and computer engineering [1]. This project explores the integration of the RISC-V processor on an FPGA.

Traditional Instruction Set Architectures (ISAs) were frequently designed to prevent illegal use, which limited their use and adaptability [2]. RISC-V, an open-source ISA, provides a more adaptive and collaborative environment. Despite its benefits, there are significant obstacles and limitations in understanding the implementation approaches for combining RISC-V processors with FPGA technology.

The RISC-V architecture, known for its simplicity and modularity, has acquired significant attention in both academic and industrial circles. This project explores the implementation of the NEORV32, an open-source RISC-V core, onto an FPGA platform. The objective was to integrate the core, verify its functionality, and demonstrate its practical application through the control of a holonomic X-drive mobile robot. This integration serves as a foundational step for further research and educational use.

II. METHODOLOGY

A. Board Choice

The FPGA board selected for this project was the Intel Altera De2i-150 development board featuring a Cyclone IV FPGA chip. This choice of board is suitable because it supports both VHDL and Verilog language as well as a well-documented user manual and user-friendly interface IDE (Quartus II).

Fig. 1. Intel Altera De2i-150 development board

Key features of the board include

- FPGA: Cyclone IV EP4CGX150DF31 device
- Logic elements: 149,760 Les
- Programmer: On-board USB Blaster circuitry
- Header Pins: 40-pin expansion ports
- Clock: 50Mhz external oscillator

B. RISC-V Processor Choice

The NEORV32 processor was chosen due to its rich in peripherals and memory interface which offers a full-scale microcontroller-like SoC.

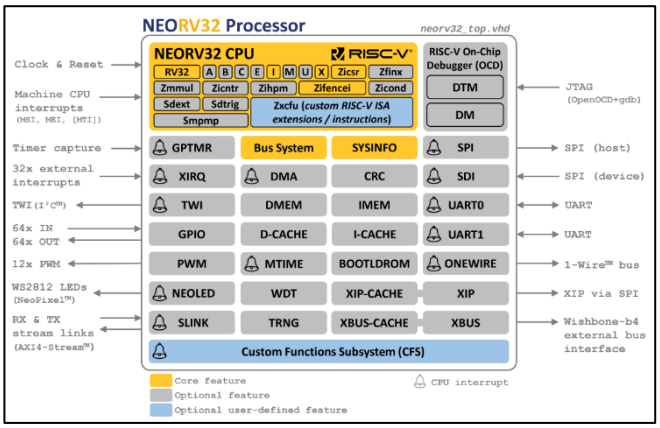

Fig. 2. SoC Block diagram of NEORV32 processor

The NEORV32 processor written in VHDL which is suitable with the chosen FPGA Board. The CPU core is a 32-bit little-endian RISC-V core [3] and consist of many configurable ISA extensions as well as data memories and caches which optionally further increase the performance of the core. Peripherals included as part of the SoC are UART, PWM, GPIO and etc for communication and I/O tasks.

C. Development Environment Setup

The first step involved was setting up an appropriate development environment. Ubuntu was installed on Windows using the Windows Subsystem for Linux (WSL) [4]. This setup is necessary for software such as GCC compiler, make file and other dependencies required for compiling the C-Code application using the RISC-V GNU Toolchain which only supported in Linux environment [5].

A pre-built RISC-V toolchain by snotling was used in this project which saves up a lot of resources as well as time, instead of building it from scratch which will take up a lot of disk space, hours to compile and need to include the libraries and external dependencies separately.

D. Uploading RISC-V Core into FPGA

The steps to configure and upload and the NEORV32 RISC-V core can be represented in the figure below:

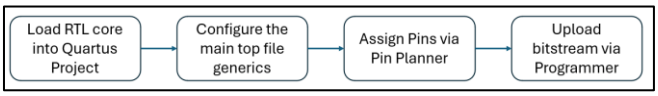

Figure 3: Summary on Uploading RISC-V Core

The first step is to load all the RTL core into Quartus Project which holds all the description of the RISC-V hardware design. The next step is to configure the main top files generics. In this step, it is important to configure the generics with specific parameters like the clock frequency used which must be followed according to the board's specifications and enable intended ISA extensions.

The subsequent step is to assign the pins via the Quartus Pin Planner. The pins can be assigned accordingly by following the FPGA board user manual [6] on the FPGA chip pins that are hard wired to the specific components for example the clock source and the buttons for the reset. The I/O standard of the pins must be selected as well to 3.3V LVCMOS which able to interface with variety of microcontrollers and sensors via UART and GPIO.

Finally, the bitstream can be uploaded via Quartus Programmer tool. The compiled core is loaded successfully into the FPGA.

III. Application: Holonomic X-Drive Mobile Robot

In the second part of this project, the focus shift to application. A 4-wheel holonomic X-Drive mobile robot is developed using the RISC-V core as the main processor. This application was chosen due to the high computational demands such as PID control and inverse kinematics.

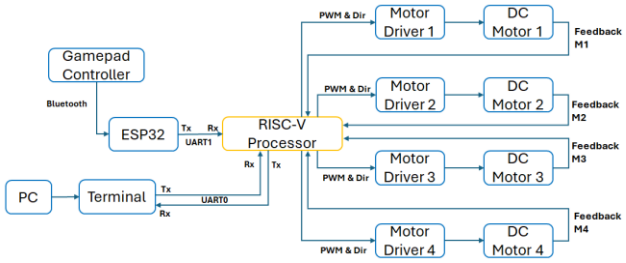

Fig. 4. shows overview of the application

The development of the 4-wheel holonomic X-drive consist of several key components:

1. RISC-V Processor: Serving as the central processing unit which will handle all the complex computations required for the motion control and kinematics
2. Feedback Mechanism: Each motor will be attached to an encoder module which will be used as feedback to the PID controller to perform the robot movements
3. ESP32 Module: The ESP32 module will facilitates the wireless communication with the DualShock 5 Controller via Bluetooth. The gamepad data will be transmitted to the main core via UART

A. *Quadrature Encoder Implementation using VHDL*

A quadrature encoder is used to measure the position and speed of a rotating object. As the encoder rotates it will generate 2 square wave signals, output A and output B which are 90 degrees out of phase. There will be 2 scenarios which are output A leading output B which indicates the rotation of the shaft connected to the encoder rotating clockwise and vice versa when output B leading output A [7].

The quadrature encoder logic will be implemented in hardware level using VHDL and be synthesized together with the core. This implementation is possible by using the Custom Function Subsystem (CFS) using the Zxcfu ISA extension. The subsystem can operate independently of the CPU which provides true parallel processing capabilities and offloads some of the resource needed.

The processed values will be pass to memory read/write registers which can be accessed by the CPU in the C-code.

```
//NEORV32_CFS->REG[x], x = 0, 1, 2 & 3,
//reads the register that contains the encoder count for each motor
//Error = Setpoint(Reference) - Actual
M1_Error = M1_Coords - NEORV32_CFS->REG[0];
M2_Error = M2_Coords - NEORV32_CFS->REG[1];
M3_Error = M3_Coords - NEORV32_CFS->REG[2];
M4_Error = M4_Coords - NEORV32_CFS->REG[3];
```

Fig. 5. shows the code snippet of reading encoder pulse

B. *Inverse Kinematics Derivation for Holonomic Drivetrain*

Deriving the inverse kinematics from scratch has several benefits especially in terms of optimization for real-time control and gives a deep understanding of the underlying mathematics and mechanics which will be valuable for troubleshooting.

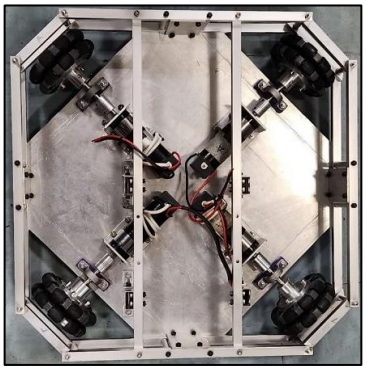

Fig. 6. shows the drivetrain base used for testing

Several considerations must be done before deriving the equations for the inverse kinematics which includes:

1. The separation angle between the wheels must be 45 degrees apart
2. The wheels must be rotating the same direction if voltage supplied to the motor

The kinematics can be obtained by simulating the robot direction respect to the joystick position of the gamepad controller.

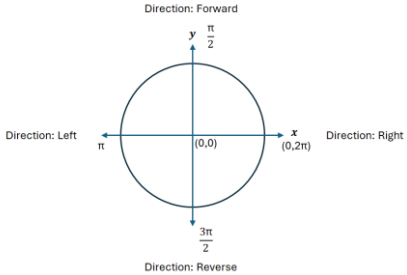

Fig. 7. shows the drivetrain base used for testing

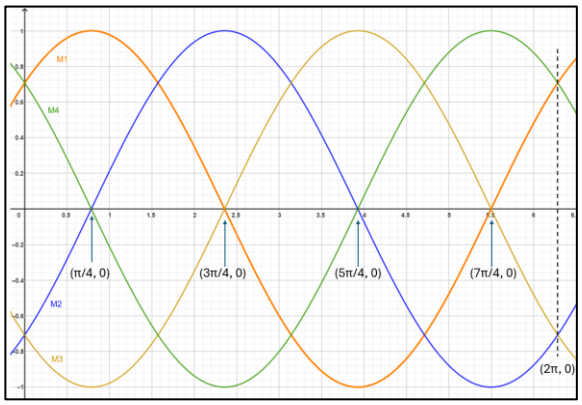

Fig. 8. Output waveform based on predicted motion

Based on the behaviour graph which describes the wheel rotation based on the gamepad joystick position, 4 equations can be formed for each motor respectively:

$$M_1 = \sin\left(\theta + \frac{\pi}{4}\right) \quad (1)$$

$$M_2 = -\sin\left(\theta + \frac{\pi}{4}\right) \quad (2)$$

$$M_3 = -\cos\left(\theta + \frac{\pi}{4}\right) \quad (3)$$

$$M_4 = \cos\left(\theta + \frac{\pi}{4}\right) \quad (4)$$

C. *Interfacing Dualshock 5 Controller using ESP32*

For interfacing the DualShock 5 controller to the main controller, and ESP32 will be utilized as the Bluetooth module to make the wireless connection.

Since the ESP32 is used to obtain the data, there will be 2 parts in the program, which are the transmission program and receiver program.

• Transmission: Input from DS5 controller will be read by ESP32 and send to RISC-V core

• Receiver: RISC-V core will receive the data from the EPS32, and the processing will be done

Considering there are many buttons on the gamepad, an 8-byte data array will be initialized. Using bit manipulation, each byte of data will be process accordingly of the bits based on the buttons available on the gamepad.

```
gamepad_data[0] = 0x01;  //unchanged
gamepad_data[1] = Lx_abs; //byte1 -> LX
gamepad_data[2] = Ly_abs; //byte2 -> LY
gamepad_data[3] = Rx_abs; //byte3 -> Rx
gamepad_data[4] = Ry_abs; //byte4 -> Ry
gamepad_data[5] = key_B5; //byte5 -> DPAD
gamepad_data[6] = key_B6; //byte6 -> other buttons
gamepad_data[7] = key_B7; //byte7 -> checksum + touchpad + L3, R3

Serial.write(gamepad_data, sizeof(gamepad_data)); //Transmit 8bytes to main core
```

Fig. 9. Data bytes that will be sent to the main processor

D. *PID Controller*

The Proportional-Integral-Derivative (PID) controller implemented in the firmware is used for feedback loop to reach a desired setpoint by minimizing the error provided by the actual measured values [8].

The PID controller implemented in the software will use the encoder pulse to create the close loop control. With this configuration the position of the robot can be determined by setting the required pulse needed for all motors to achieve the position control.

E. *Error Checking Mechanisms*

There will be 2 error checking mechanisms implemented in the firmware which are timeout mechanism and error handling.

A timeout mechanism is developed to prevent the UART buffer from waiting indefinitely which eventually will result in slow response. The timeout mechanism uses CPU clock cycles to count the elapsed time and reset the data buffer, if necessary, when the timeout period occurred without receiving any data.

While for error handling, it is used to verify the integrity of the data packets received from the UART. This implementation is possible by adding a checksum in the first byte of the data packets which will be used for double checking if the checksum is matched with the one from the transmitter side. A comparison is performed and if the checksum received does not align, it will be terminated and it checks for new data packets.

IV. PERFORMANCE TUNING

Since the application intended to be developed requires high resources and heavily dependent on the CPU performance, it is essential to optimise the core for maximum performance. The performance can be improved by enabling some of the dedicated hardware accelerators from the RISC-V CPU extensions, which are C, M and Zfinx extensions. Besides the extensions, the CPU performance can be further improved by enabling FAST_MUL_EN and FAST_SHIFT_EN as well as cache memory and data memory where it reduces the memory access latency [3].

A. *Performance Difference*

The performance can be evaluated using CoreMark, which is a benchmarking tool to test the overall processor performance and functionality.

```
NEORV32: Processor running at 50000000 Hz
NEORV32: Executing coremark (2000 iterations). This may take s

2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 9157466 k
Total time (secs): 183
Iterations/Sec   : 10
Iterations       : 2000
Compiler version : GCC13.2.0
Compiler flags   : -march=rv32i_zicsr_zifencei -mabi=ilp32 -Os
Memory location  : STATIC
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x4983
Correct operation validated. See README.md for run and reporti

NEORV32: Hardware Performance Monitors (low words only)
 > Active clock cycles          : 567531428
 > Retired instructions         : 1698378811
no HPMs available
```

Fig. 10. CoreMark data before performance tuning

```
NEORV32: Processor running at 50000000 Hz
NEORV32: Executing coremark (2000 iterations). This may take some time...

2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 6172660 k
Total time (secs): 123
Iterations/Sec   : 16
Iterations       : 2000
Compiler version : GCC13.2.0
Compiler flags   : -march=rv32i_zicsr_zifencei -mabi=ilp32 -Os -Wall -ffunction-sections
Memory location  : STATIC
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x4983
Correct operation validated. See README.md for run and reporting rules.

NEORV32: Hardware Performance Monitors (low words only)
 > Active clock cycles          : 1877693585
 > Retired instructions         : 1698378811
no HPMs available
```

Fig. 11. CoreMark data after performance tuning

Based on the comparison, initially the iterations/sec was 10 and it is increased to 16. The performance difference has increased by 62.5% which will help the overall performance of the system

V. RESULTS

A. *Overview*

The implementation successfully demonstrated the RISC-V processor's capabilities to handle complex computational tasks. The holonomic drive mobile robot performed as expected, hence validating the processor's performance and efficiency. The processor can switch between manual movement using the DualShock 5 gamepad and the pre-defined movement using encoders where the robot is able to move to setpoints based on the gamepad button clicks without any freezing or errors. The program can interchange between the 2 modes smoothly without any issues using a defined button on the gamepad.

B. *Manual Movement*

Based on the implementation of the DualShock 5 controller, the processor is able successfully receive the data bytes from the ESP32 via UART.

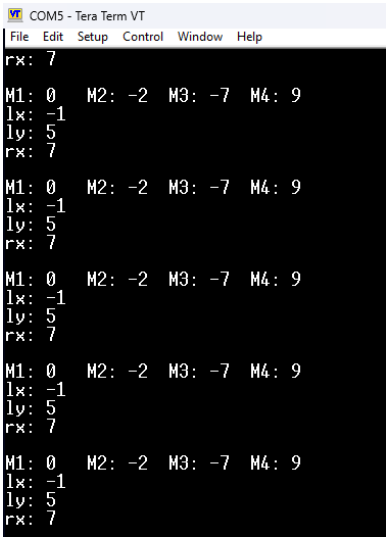

Fig. 12. Results from the gamepad controller

The implemented timeout mechanism and error handling works as expected, all the data received by the controller is valid and receiving smoothly without any distortion.

Expected Wheel Rotation for Omnidirectional X-Drive

Direction	M1	M2	M3	M4
Forward	CW	CCW	CCW	CW
Backward	CCW	CW	CW	CCW
Left	CCW	CCW	CW	CW
Right	CW	CW	CCW	CCW
Diagonal Forward Left	0	CCW	0	CW
Diagonal Forward Right	CW	0	CCW	0
Diagonal Backwards Left	CCW	0	CW	0
Diagonal Backwards Right	0	CW	0	CCW

Fig. 13. Expected wheel rotation for X-drive drivetrain

All the presented data aligned to the expected wheel rotation for the X-drive drivetrain. The mobile robot able to move in all directions without turning its axis, which makes it a true holonomic robot. This shows that the core able to compute trigonometric functions efficiently.

C. *Pre-defined Movement using Encoder*

The encoder data from the subsystem can retrieve successfully and pass it to the PID controller. The target encoder count will be stored in a 2D array for each destination and the required counts needed to be moved

```
int numberOfDestination = 5;
int destinations[5][4] =
{
  {0, 0, 0, 0},
  {-20000, -20000, 20000, 20000},
  {-40000, 0, 40000, 0},
  {-20000, 20000, 20000, -20000},
  {0, 0, 0, 0}
};
int currentDestinationIndex = 0;
```

Fig. 14. Pre-defined setpoints for each motor encoder

The firmware also enables to cycle back and forth from the robot current location to the next or previous setpoint since the data of the setpoint is stored in an array.

```
if(gamepad.state.btn.btnR1 && !btnR1Check)
{
    currentDestinationIndex++;

    if(currentDestinationIndex >= numberOfDestination)
    {
      currentDestinationIndex = 0;
    }
}
btnR1Check = gamepad.state.btn.btnR1;

if(gamepad.state.btn.btnL1 && !btnL1Check)
{
    currentDestinationIndex--;

    if(currentDestinationIndex < 0)
    {
      currentDestinationIndex = numberOfDestination - 1;
    }
}
btnL1Check = gamepad.state.btn.btnL1;
```

Fig. 15. Results from the PID controller

The core can compute all 4 motors' respective PID control as well as generating and applying the control signal to the motor drivers without any issues or latency.

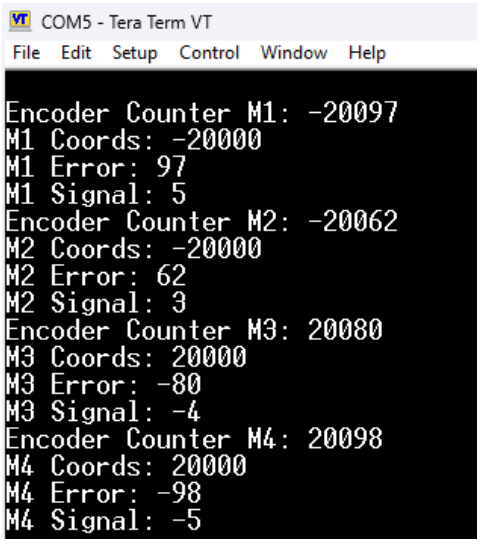

Fig. 16. Results from the PID controller

VI. Conclusion

The integration of the RISC-V processor into the FPGA platform using the Intel Altera DE2-115 board proved to be successful. The processor was able to handle the computational demands of a complex application, demonstrating its potential for educational and research purposes. Future work could expand peripheral support and incorporate advanced features such as neural networks for more sophisticated tasks.

References

[1] R. Höller, D. Haselberger, D. Ballek, P. Rössler, M. Krapfenbauer and M. Linauer, "Open-Source RISC-V Processor IP Cores for FPGAs — Overview and Evaluation," *2019 8th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 2019, pp. 1-6, doi: 10.1109/MECO.2019.8760205.

[2] Andrew Shell Waterman, “Design of the RISC-V Instruction Set Architecture”, 2016

[3] Stephen Nolting, “The NEORV32 RISC-V PROCESSOR”. [Online].
Available: https://stnolting.github.io/neorv32/ [Accessed May 29, 2024]

[4] Craigloewen, “How to install Linux on Windows with WSL”. [Online]
Available:https://learn.microsoft.com/en-us/windows/wsl/install [Accessed April 9, 2024]

[5] Stephen Nolting, “Prebuilt RISC-V GCC Toolchain for Linux”. [Online].
Available: https://github.com/stnolting/riscv-gcc-prebuilt [Accessed: April 9, 2024]

[6] De2i-150 FPGA Getting Started Guide, Manual, Intel Altera, 2016. [Online]
Available:
https://www.secs.oakland.edu/~llamocca/Tutorials/Emb_Intel/Documentation/DE2i-150_FPGA_User_Manual.pdf
[Accessed: June 25, 2024]

[7] S. Sarkar and K. Sivayazi, "A New Decoding Logic for Quadrature Encoder Interfacing for Software Implementation on FPGA," *2018 3rd International Conference for Convergence in Technology (I2CT)*, Pune, India, 2018, pp. 1-4, doi: 10.1109/I2CT.2018.8529379.

[8] G. Găşpăresc, "PID control of a DC motor using Labview Interface for Embedded Platforms," *2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC)*, Timisoara, Romania, 2016, pp. 145-148, doi: 10.1109/ISETC.2016.7781078.