LiDAR Pong

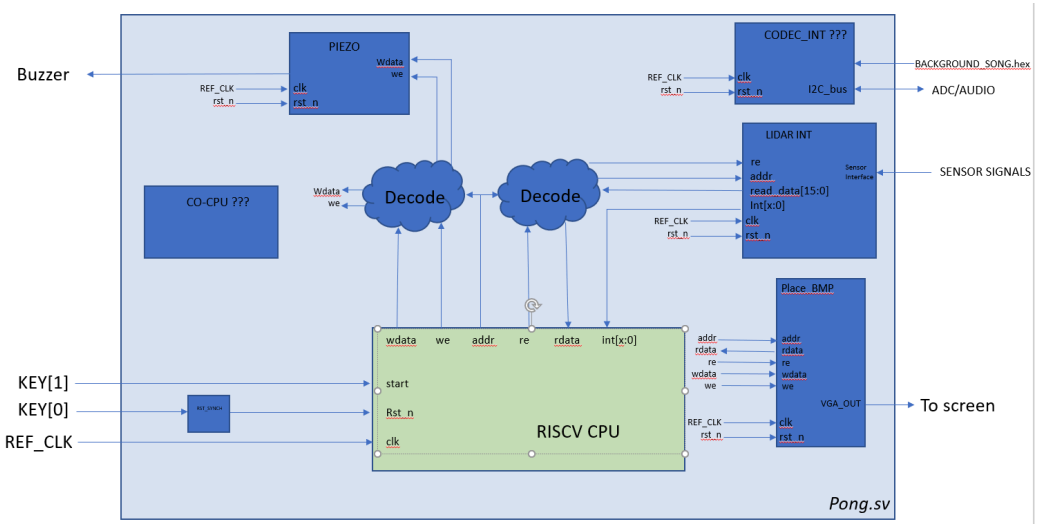
TeamNotSure

Noah Kurszewski, Noah DeGroot, Brian Huang, Jonathan Yang, Ryuki Koda

University of Wisconsin-Madison

Project Final Report

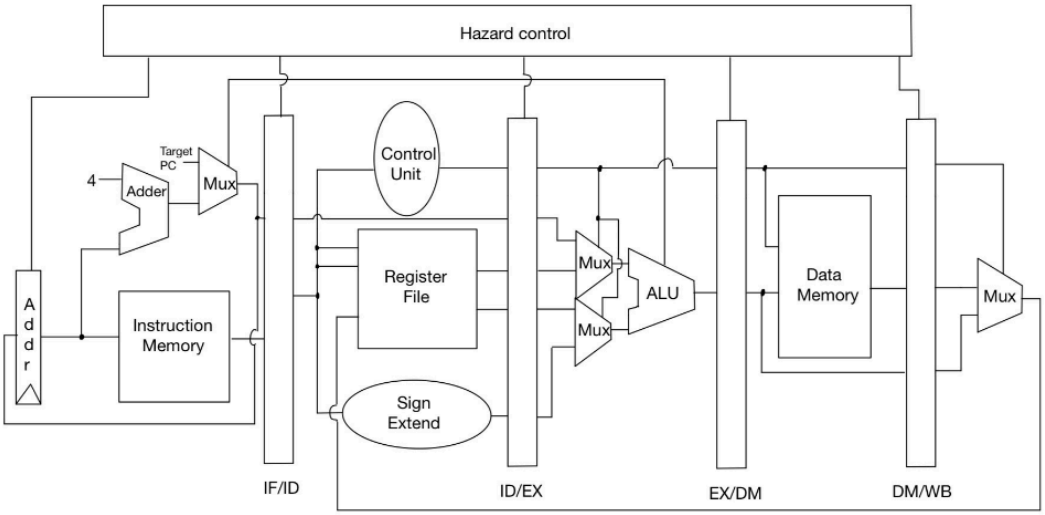
1. Repeat of ISA Table
2. Hardware Block diagrams
2.1. Top Level Block Diagrams



Description

2.2. CPU

This is a high-level block diagram of our RISC V-32 IM processor. The original 16-bit pipeline logic was mostly preserved, but we extended it to 32-bit and added multiplication and division. For detailed interface specifications, please refer to the following sections.



2.2.1. PC Interface

Signal	Width	Direction	Description
Int_occurred	1	In	Indicates an interrupt occurred and should jump to the interrupt vector provided **int_vec. PC** should be stored to context save copy
Int_vec	16	In	From the interrupt controller. Specifies ISR location to jump to
Flow_change_ID_EX	1	In	Indicates PC will be updated due to branch or jump (from ID unit)
Rti_ID_EX	1	In	Return from interrupt has occurred. Restore context saved PC
Stall_IM_ID	1	In	Asserted if stalling pipe due to load/use or MOVC related stall
Dst_ID_EX	16	In	Branch target address from ALU
Pc	16	Out	Forms address to instruction memory
Pc_ID_EX	16	Out	Piped version needed in EX stage for computing new branch target
PC_EX_DM	16	Out	Piped to EX_DM for JAL instruction store in R15

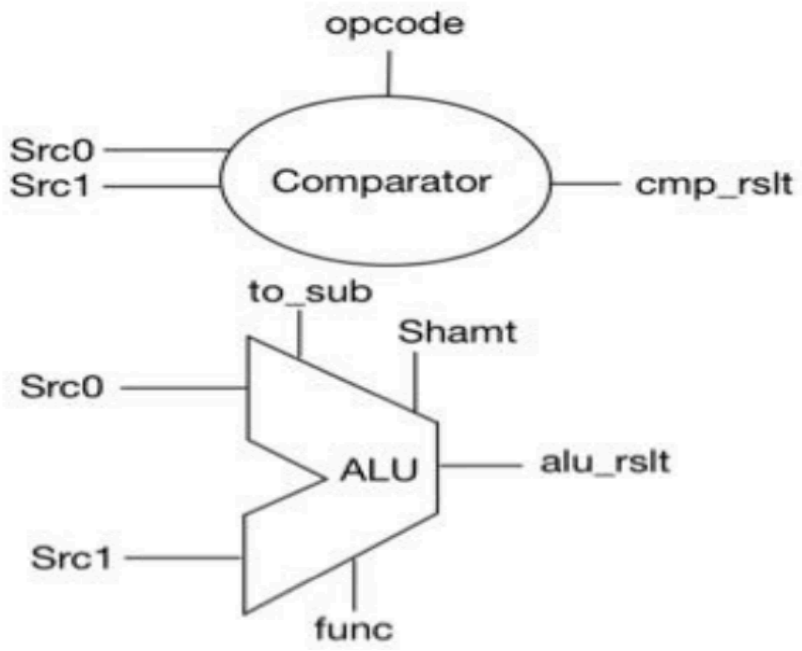
2.2.2. PC Registers

Signal	Width	Description
Pc	32	Described in table above
Pc_ID_EX	32	Described in table above
PC_EX_DM	32	Described in table above
Pc_saved	32	Context store of next PC saved/restored during interrupts/RTI
Flow_change_last_stalled	1	A MOVC followed by a branch can cause a scenario where flow change and stall can happen simultaneously. In which case we need to update PC with newPC_stalled next
newPC_stalled	32	Capture of target address if stalled.

2.2.3. ALU Interface

This is a high-level block diagram of our ALU. We extended it to 32-bit and added multiplication and division. Since RISC V has no flags, we designed a comparator to do the comparison for instructions that need comparison results.

For detailed interface specifications, please refer to the following sections.



Signal	Width	Direction	Description
Src0	32	In	Source 0 for ALU and comparator computation.
Src1	32	In	Source 1 for ALU and comparator computation.
To_sub	1	In	Indicates if the instruction will be subtraction.(1 for sub, otherwise 0)
Shamt	5	In	Shift amount for ALU
Func	4	In	ALU function opcodes, including Mul/Div.
Opcode	3	In	Comparator function opcodes used for different kinds of comparison.
Cmp_rslt	1	Out	Forms result for the comparison
Alu_rslt	32	Out	Computation result for ALU functions.

2.2.4. ALU Registers

Signal	Width	Description
Alu_rslt	32	Computation result for ALU functions.

Cmp_rslt	1	Forms result for the comparison.

2.3. BPM_Display_Block

This block controls the output of the different images dynamically to the screen. This will have a memory mapped control register and various ROMs that store the image files.

2.3.1. Register Map

Mnemonic	Address	Description
BMP_CTL	0xC008	A write to this 16-bit register starts a placement of an image or a character. Has the following bit mapping: {add_fnt,fnt_indx[5:0],2'b00,add_img,rem_img,image_index[4:0]}
BMP_XLOC	0xC009	X-location (in pixels) of upper left corner of BMP image (write to this before writing to BMP_CTL)
BMP_YLOC	0xC00A	Y-location (in pixels) of upper left corner of BMP image (write to this before writing to BMP_CTL)
BMP_STAT	0xC00B	A read from this returns {15'h0000,busy} Your code should poll for not busy before any write to BMP_CTL.
USER_KEYS	0xC00C	A read from this reg returns {14'h0000, key_2 ,key_1}. Note: this register is read to clear

2.3.2. Screen Info

The screen is 640p x 480p.

2.3.3. Image Info

The current images that are in memory are the net, ball, and paddles

Note: All the images are surrounded by a pixel of black to erase the previous image. This will only work if the images move by 1 or 2 pixels. If they move by more than that, this needs to be updated. Also, anytime the ball crosses the net, the net will have to be redrawn.

Index	Image	Width (p)	Height (p)
0	net	28	480
1	paddle	8	33
2	ball	8	8

2.3.4. Interface

Signal	Width	Direction	Description
clk	1	in	system clock

rst_n	1	in	async reset
add_fnt	1	in	set high when adding a character
fnt_index	6	in	one of 42 characters: 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ =>,()
add_img	1	in	pulse high for one clock to add image
rem_img	1	in	pulse high for one clock to remove image
image_indx	5	in	index of image in image memory (32 possible)
xloc	10	in	x location of image to register
yloc	9	in	y location of image to register
waddr	19	out	write address to videoMem
wdata	6	out	write 6-bit pixel to videoMem
we	1	out	VideoMem write enable
busy	1	out	asserted if PlaceBMP is busy and cannot accept another command

2.3.5. LiDAR

This block is designed to interface with the LiDAR sensors.

Signal	Width	Direction	Description
clk	1	in	system clock
rst_n	1	in	async reset
re	1	in	set high when reading from LiDAR
addr	16	in	Address to read from LiDAR
read_data	16	out	Data read from LiDAR
RX	16	in	UART RX line reading from LiDAR

2.3.6. Piezo

This block is designed to control different sounds made during the game

Signal	Width	Direction	Description
clk	1	in	system clock
rst_n	1	in	async reset
play_sound	1	in	set high when playing a sound
sound_indx	2	in	00 button press 01 hit wall 10 score 11 game over (win)
piezo	1	out	sound output to buzzer
piezo_n	1	out	sound output to buzzer

3. Software

3.1. Game Mechanics

3.1.1. Boundaries

The foremost game mechanic our software is responsible for is defining the boundaries of the game. There are visual boundaries that determine where visual assets can and can't be displayed, such as preventing the ball and paddle from disappearing from the game area. Working in conjunction with these visual boundaries, there are logical boundaries that align the game logic with the visuals of the game, preventing the paddles and ball from leaving the visual environment but still interacting with each other off screen. There are two types of boundaries: walls and goals. The walls are the top and bottom boundaries. They prevent the paddles from leaving the game area and are surfaces the ball can bounce off. The goals are the left and right boundaries. They trigger the incrementation of the respective player's score, the reset of the paddle and ball locations, and the start of the next match when the ball contacts them.

3.1.2. Collisions

Our software detects collisions between the ball and the goals, walls, and paddles and responds differently depending on the surface. For goals, a collision triggers the incrementation of the respective player's score, the reset of the paddle and ball locations, and the start of the next match when the ball contacts them. For walls, the ball bounces off with an angle of reflection equal to the angle of incidence and at the same speed it hits the wall. For the paddles, the ball bounces off with an angle of reflection that increases the further away from the center of the paddle the ball hits. The speed at which the ball leaves the paddle also increases the further away from the center of the paddle the ball hits.

3.1.3. Scoring & Winning

When a player scores, our software triggers a change in the respective player's score, returns the ball to the middle of the playing field, and fires the ball toward the player that did not score. When a player reaches the threshold to win (11 points), the software triggers the win screen to display. The game will continue to display the win screen until the reset button is pressed.

3.2. Hardware Interface

3.2.1. VGA

Our software triggers changes to what is displayed on the VGA display when the game is started, when a point is scored, and when a

player wins the game. The software doesn’t directly write to the display, but it triggers the hardware to change what is displayed.

3.2.2. Piezo

Our software triggers various piezo tones when the ball collides with a wall, a paddle, or a goal and triggers a song on the piezo when a win occurs. The wall and paddle collisions share the same tone, but the goal tone is different. The software doesn’t directly play the tones, but it triggers the hardware to play the tone stored in memory that corresponds to the event that occurred.

3.2.3. LiDAR

Our software reads from the LiDAR hardware block to determine the height of each player’s hand from their respective sensor. The height of the player's hand sets the height of the paddle, and the software adjusts the height of the paddle as the player moves their hand up and down.

3.2.4. Buttons

On boot, our software reads the status of KEY[1] from hardware, waiting for it to be pressed. When it is pressed, it triggers the start of the game, firing the ball towards player

4. Application Code: The Pong Game

First, when the player interacts with the LiDAR sensors, it detects the distance between the hand and the sensor, adjusting the position of the Pong paddle accordingly to the respective player. This means the greater the distance corresponds the higher the position of the paddle and vice versa. Next, the ball's movement is determined by its distance from the center of the paddle upon impact. While the ball maintains at a constant speed, hitting it closer to the center of the paddle results in a relatively horizontal rebound, while hitting it further away from the paddle causes the ball to deflect in the paddle moving direction. To determine a winner, the scoring system awards one point to the player each time the ball enters the opponent’s region behind the paddle. The game continues until a player reaches a score of 7. Lastly, although the intended design is for multiplayer competition, a single-player mode is included for the player to compete against a computer-controlled opponent.

4.1. Default Parameters

For clear coding, included in this section is a list of macros used in the C program, with their respective explanations.

4.1.1. Game Related Parameters

Below is a list of parameters related to game physics.

Parameter Name	Value	Description
BOARD_WIDTH	640	The width of the window (in pixels)
BOARD_HEIGHT	480	The height of the window (in pixels)
BALL_RADIUS	4	The ball radius
HALF_PADDLE_HEIGHT	16	½ of paddle height
HALF_PADDLE_WIDTH	4	½ of paddle width
PADDLE_SPACING	5	Horizontal distance between paddles and the edge of screen
AI_SPEED	1	The paddle speed controlled by the AI player
PADDLE_SPEED_FAST	2	The faster paddle speed controlled by human players
PADDLE_SPEED_SLOW	1	The slower paddle speed controlled by human players

4.1.2. Hardware Related Parameters

Below is a list of parameters related to hardware peripherals and memory addresses.

Parameter Name	Value	Description
THRESHOLD1	100	Paddle moves down at a faster speed (0 < d <t1) *
THRESHOLD2	200	Paddle moves down at a slower speed (t1 < d < t2)
THRESHOLD3	300	Paddle does not move (t2 < d < t3)
THRESHOLD4	400	Paddle moves down at a slower speed (t3 < d < t4)
THRESHOLD5 **	500	Paddle moves down at a faster speed (t4 < d < t5)
BMP_CTL	49160	Starts placing an image when writing to this address
BMP_XLOC	49161	X location of the image
BMP_YLOC	49162	Y location of the image
BMP_STAT	49163	A reading from this address returns if last image is done
PIEZO_REG	49165	Starts the buzzer

* d is the distance between hand and the sensor

** If the distance is too large, i.e. d > t5, it is considered hand not interacting with the sensor, and the paddle will not move.

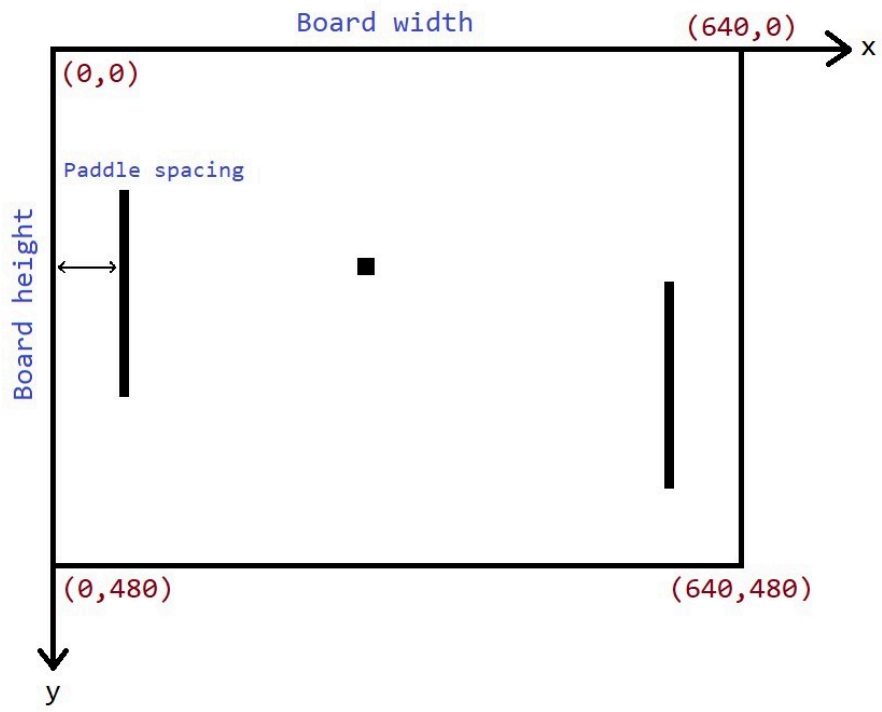
4.2. Game mechanics

The pong game core algorithm consists of the physics of the ball, paddles, and edges of the board. The board can be represented by a 2-Dimensional Cartesian coordinate, with the top-left corner placed at the origin.

The ball parameters include the center position of the ball, $(xb, yb) \in \mathbb{R}^2$, and the velocity vector of the ball, $(dx, dy) \in \mathbb{R}^2$. In order to enable arbitrary angles of ball moving direction, both position and velocity vectors are set to float type. Otherwise, the ball could only move in either horizontally or with a 45° angle. Although floating point numbers are not natively supported by our

processor, the C compiler will automatically convert floating point arithmetic into integer arithmetic during the compilation process.

The paddle parameters include only the center position, $(xp, yp) \in \mathbb{N}^2$. The types of paddle positions are intentionally set to integers for simplicity.



4.2.1. Ball Movement

Without any interaction with board edges or paddles, the ball will move in the direction of the velocity vector. Each time the ball moves, its new position vector will be updated by the formula

$$(x', y') = (x + y) + (dx, dy)$$

4.2.2. Collision with Edge

The ball will be reflected when colliding with the edge of the board. By checking the y-coordinates of the ball, the program changes the direction of the ball, while keeping the original speed. The new velocity will be $(dx', dy') = (dx, -dy)$

4.2.3. Collision with Paddle

Upon collision with the paddle, the ball bounces off with an angle of reflection that increases the further away from the center of the paddle the ball hits.

To check if the collision happened, we first check the x-coordinates of the paddle and the ball, and then verify that the y-coordinates of the ball are within half paddle length of the paddle center.

Specifically, when a collision happens, the vertical component of the new velocity vector will be calculated as follows:

- Calculating the hitting position $\Delta d = yb - yp$. Since we are provided that the ball hits the paddle, we know that $\Delta d \in [-1/2PL, 1/2PL]$, where PL stands for paddle length;
- Normalizing Δd and then multiplying it by the range of the ball speed$dy' = \Delta d/((1/2)PL)\cdot vb$, where vb is the range of the ideal ball speed, being set to 1 by default.

It remains to find the horizontal component. Ideally, we want to keep the speed of the ball constant, i.e., equal to vb This could be achieved by letting $dxideal = \sqrt{(vb^2 - dy'^2)}$

However, there are a few issues with this implementation. Firstly, calculating square roots are complicated, and thus slow down the program; secondly, when hitting position are at the edge of the paddle, the horizontal component of the new velocity approaches 0, and the ball will keep bouncing between the upper and lower edges of the board for very long time until reaching the other side of the board.

Notice that $vb = 1$, by default, we can make approximations by using Taylor expansion:$dxideal \approx dx' = 1 - 1/2dy'^2$. Since $dx' \in (1/2, 1]$, we have avoided both problems involved with square root, while also keeping the ball speed at a reasonable range.

4.2.4. AI Player

If the game operates in single player mode, the other player will be controlled by the AI player.

The algorithm for AI player is quite simple:

- If the ball moves towards the AI player, it compares the y-coordinate differences of the paddle and the ball, and moves the paddle accordingly;
- If the ball moves away from the AI player, it moves the paddle to the center of the board;
- To allow AI players to make mistakes, the AI player speed is limited to 1.

4.3. Testing

The testing process involved early testing with terminals on a Windows / Linux machine, and final testing on VGA and BMP display with our RISC-V processor on FPGA.

To enable testing while not having to create a separate file, we used compiler directives. By setting DEBUG to 1, the terminal testing mode will be enabled; Setting DEBUG to 0 results in the real application code that runs on our processor.

4.3.1. Testing with Terminal

When testing on the terminal, we used the printf function in the standard C library. A 2D char array was used to represent the current structure of the board, and we effectively "draw" the board with characters such as asterisks and ampersands.

To accommodate the terminal width and height limitations on a common terminal, we created another set of values for each macro defined in section 3.1.1. The switch between different sets of macros is automatically done by the DEBUG mode.

Since our C program is constantly printing on the terminal, it is hard to create human inputs, and subsequently we use an AI player to control both paddles. By setting a correct board height and using sleep function to wait Upon watching the result being printed, we successfully verified the correctness of the game algorithms.

4.3.2. Testing with VGA

After verifying with the terminal, we proceed to test on a screen, which is connected to the FPGA board with a VGA cable.

5. Engineering Standards

Discuss any engineering standards used in your design and how these engineering standards make the design better or more compatible with other designs

- IEEE 1364 – 2001 verilog
- IEEE 1800-2009 systemVerilog
- IEEE 754 32-bit floating point standard
- Some De-facto industry developed standards (VGA, I2S, pS/2)

6. Potential Societal Impacts of Your Design

6.1. Practical Advancements

Though the project itself does not demonstrate any advanced technologies, many of the game components are often used in the practical world. Using the

LiDAR system as the controller provides precise measurements in terms of distance, allowing an accurate perception and reflects it onto the screen. Additionally, it is not affected by environmental conditions such as lighting or surface textures. Therefore, due to its accuracy and long-range detection, they are often used in many aspects such as autonomous vehicles or object perception; and for their resistance against environmental conditions, it is often used for mapping the surroundings.

6.2. Social Interaction

Although it is a simple game, it has incorporated more hardware components, allowing more robust control for the players. Therefore, providing a better grasp on hand-eye coordination for the players. Additionally, a multiplayer system is developed in the game, which allows stress relief and entertainment for those to have fun and combat with one another.

6.3. Technological Progression

This game evokes a feeling of nostalgia back in the 70's and 80's when arcade games were just developed. Building this game helps to not only understand the struggle and the advancements of the technological world, but also to recognize the progress and efficiency that technologies have brought to the world. With modern CPU achieving greater efficiency and GPU processing with billions of transistors, it is without a doubt that technological progression is still in its evolution, witnessed by the people.

7. Final Application Demonstration

8. Contributions of Individuals

- Brian Huang
  - Implemented pipelined RISC V architecture
  - Implemented control blocks for optimization including forwarding, rf bypassing, and data memory banks
- Noah Degroot
  - Integrated hardware peripherals with FPGA board
  - Tested RISC V integration with peripherals
- Noah Kurszewski
  - Oversees progression of the entire project
  - Assisted with peripheral integration
- Ryuki Koda

    - Assisted in RISC V architecture
    - Generated unit tests and testbench using RISC V ISA.
- Zhiheng Yang
    - Implemented high level C code for the functionality of the game
    - Implemented a computer-controlled opponent for single-player mode