UNIVERSITY OF TORONTO

JOÃO MARCUS RAMOS BACALHAU

GUSTAVO MAIA FERREIRA

HEYANG WANG

ECE532 – FINAL DESIGN REPORT HOLE IN THE WALL

Toronto
2015

Summary

List of Figures

List of Tables

1 Overview

1.1 Motivation

Nowadays, almost every gaming platform has some kind of video feedback that enhances the interaction with the game. Games franchises like Just Dance, miCoach and Wii Sports are becoming more attractive because they offer not only a different interface to play but also a new form of exercise at home. Developing a game using image processing on a FPGA is a relevant task since many of the current Kinect sensors perform part of the image processing on embedded systems. This pre-processing has the advantage of removing part of the processing from the main processor of the platform.

1.2 Goals

The main idea of this project is to re-create a Japanese game show called “*Nōkabe*”, known in the West as “Avoid the Wall” or “Hole in the Wall”. In this game, a player must avoid a wall that comes towards him. The only way to do that is to fit on different shapes and sizes of holes on the wall. The difference between the game show and this project is that imaginary walls displayed on a monitor are used instead of physical walls. The player is be able to see himself and the wall coming to his direction on the monitor. To win the game, as in the television game show, he must put his image in the hole on the imaginary wall displayed on the monitor.

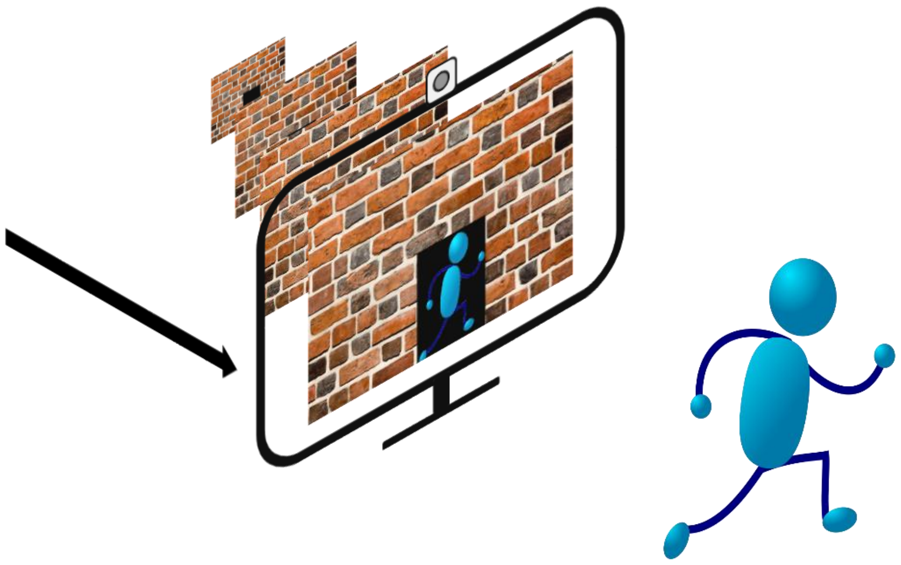

Figure 1. Illustration of the project setup and functionality.

In order to reach this objective, a Nexys 4 DDR board is used. This board contains a Xilinx Artix-7 FPGA, which is connected to OV7670 camera module and a VGA monitor. Background sound and audio effects are some features that improve the user experience. The switches and push-buttons available on the board are used to control the menu screen and the level selection of the game. Different difficulty levels can be achieved by changing the speed of the wall and/or the shape of the holes. The most important advantage of this project over the actual game is the fact that it does not have moving parts and no special environment with mechanical pieces dragging the walls is required.

1.3 Image processing background

The threshold operation is a simple way to segment as object in an image. This filter is a pixel-by-pixel operation that outputs a binary image. The operation basically writes on the same place as the pixel that is being evaluated a one if the value of the pixel is above a threshold. Figure 2 shows how this operation can be used to find the position of the person in an image.

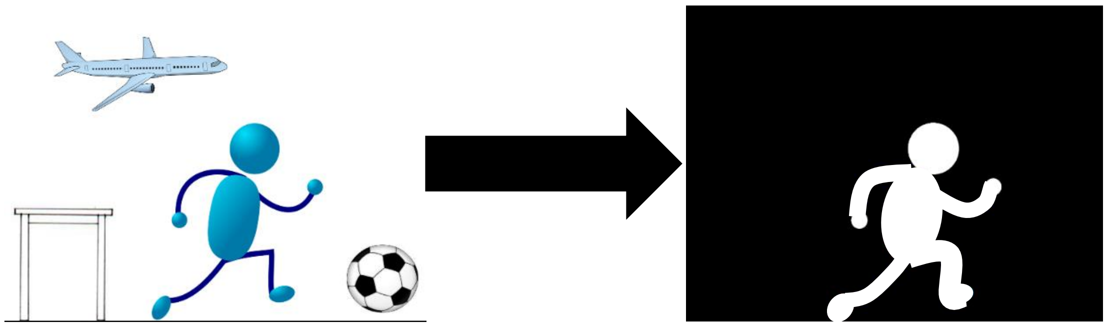

Figure 2. Colour Threshold

Another operation that is important for this project is the merging or blending of two images. This allows combining the image from the camera and the wall for the user on the monitor. As the threshold filter this is also a pixel-by-pixel operation that is weight the value from the two images. The result of this operation is shown on Figure 3.

$$Img_3 = 15 \ * \sqrt[n]{\left(\frac{Img_1}{15}\right)^n + \left(\frac{Img_2}{15}\right)^n} \ , n = 2$$

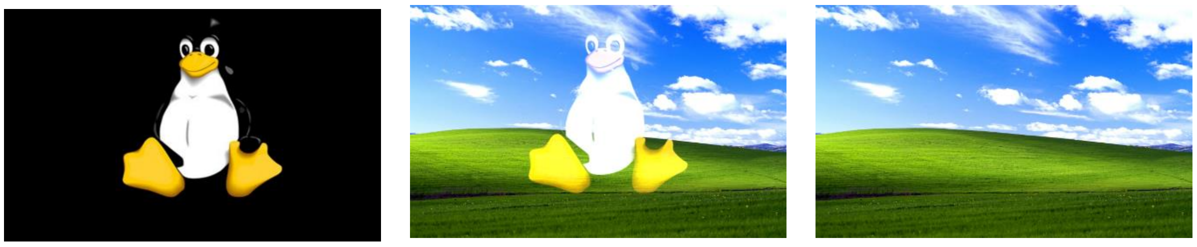
Figure 3. CEMENT Algorithm.

1.4 Block Diagram

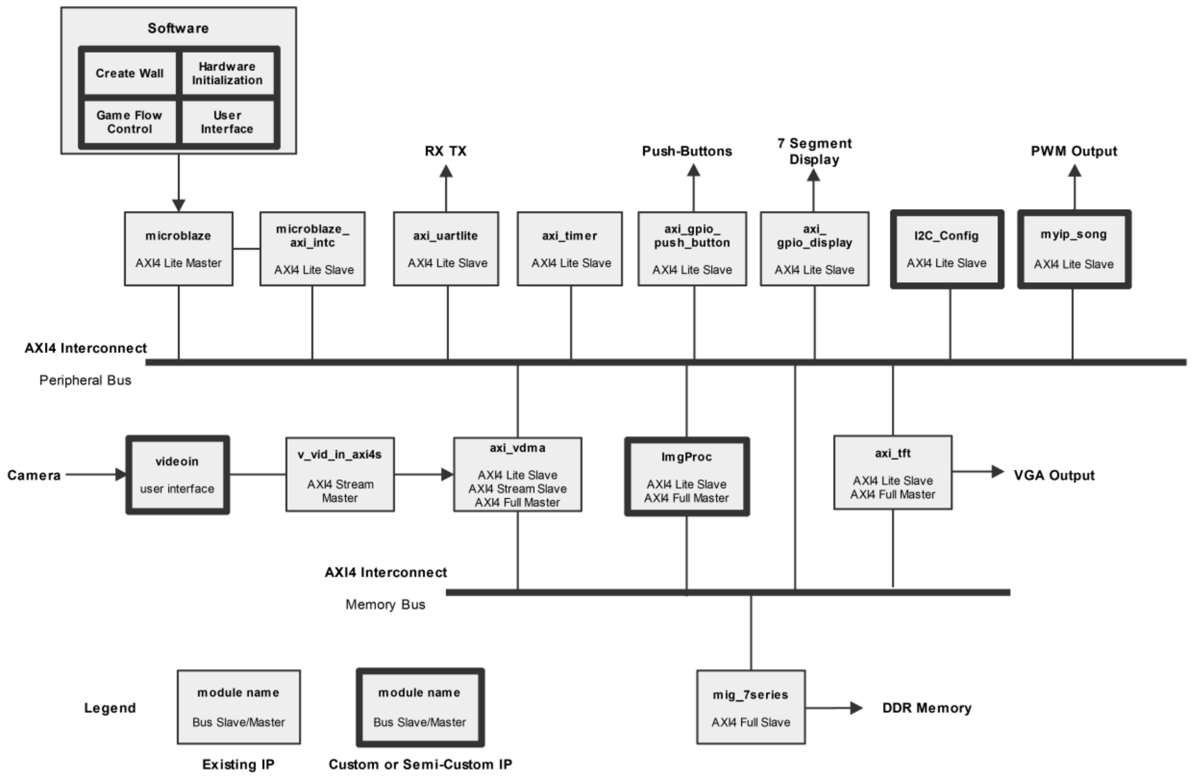

Figure 4. Block Diagram of the project.

1.5 Description of IPs

Hardware

Block	**IP**	**Description**	**Origin**
microblaze	MicroBlaze	32-bit soft processor MicroBlaze.	Xilinx
microblaze_axi_intc	AXI Interrupt Controller	Merges multiple interrupt inputs from peripherals devices to a single interrupt output to the system processor.	Xilinx
axi_uartlite	AXI Uartlite	Controller interface for asynchronous serial data transfer.	Xilinx
axi_timer	AXI Timer	32-bit timer module.	Xilinx
axi_gpio_push_button	AXI GPIO	Input interface for in-board push buttons.	Xilinx
axi_gpio_display	AXI GPIO	Output interface for in-board seven segments display.	Xilinx
I2C_Config	I2C_Config	Semi-custom IP that allows changing OV7670 camera settings via AXI4 slave interface.	Group
myip_song	myip_song	AXI IP that outputs a programmed sequence of musical notes using PWM output.	Group
videonin	videonin	Video adapter for OV7670 camera for Video In to AXI4 IP.	Group
v_vid_in_axi4s	Video In to AXI-4	Interface from a video source to the AXI4-Stream Video Protocol Interface.	Xilinx
axi_vdma	AXI Video Direct Memory Access	High-bandwidth direct memory access between memory and AXI4-Stream Video Protocol Interface.	Xilinx

ImgProc	ImgProc	AXI4 Full Master IP that reads 2 frames (4-bit images only) and perform a programmed operation.	Group
axi_tft	AXI TFT Controller	VGA display controller that reads video from memory.	Xilinx
mig_7series	Memory Interface Generator (MIG 7 Series)	Controller and Physical Layer for interfacing 7 series FPGA user designs and AXI4 interfaces to DDR2 devices.	Xilinx

Table 1. Description of Hardware IPs

Software

IP	Description	Origin
Create Wall	Generate wall images of various sizes using pixel art.	Group
Hardware Initializaition	Contains initial configuration for IPs in this design.	Group
Game Flow Control	Controls game flow: holes creation, wall update and threshold filter.	Group
User Interface	Handles user interface, push buttons and seven-segment display.	Group

Table 2. Description of Software IPs

2 Outcome

2.1 Result

The project was completed on time and with almost all the features that were proposed at the beginning. Table 3 shows all the features that were proposed.

ID	Features	Status
1	Scale the wall images to give the user the impression that the wall is coming towards him.	Complete
2	Segment the user out of the background.	Complete
3	Output Audio.	Complete
5	Songs for each level and when the player win or lose	Complete
6	Output Sound when the wall hit the player.	Not Implemented
7	Level selection using push-button	Complete
8	Display the current level on the 7 segments display	Complete
9	Generate different size of wall.	Complete
10	Generate different size of holes.	Complete
11	Generate different shapes of holes	Not Implemented
12	Different levels based on size of hole and speed of wall.	Complete
13	Video input to FPGA board with at least 30 fps.	Complete
14	Real time threshold filter wall to determine the position of the player.	Complete
15	Comparison on Hardware of the player position and the position of the hole in the wall.	Complete
16	Real time image merging to generate an transparent wall on the video	Complete
17	Hardware output a song automatically after configured once.	Complete

Table 3. Features of the project.

The figure 5 shows the actual game running. It is possible to see the camera on top of the monitor and the image of the wall blended with the video input using the CEMENT algorithm.

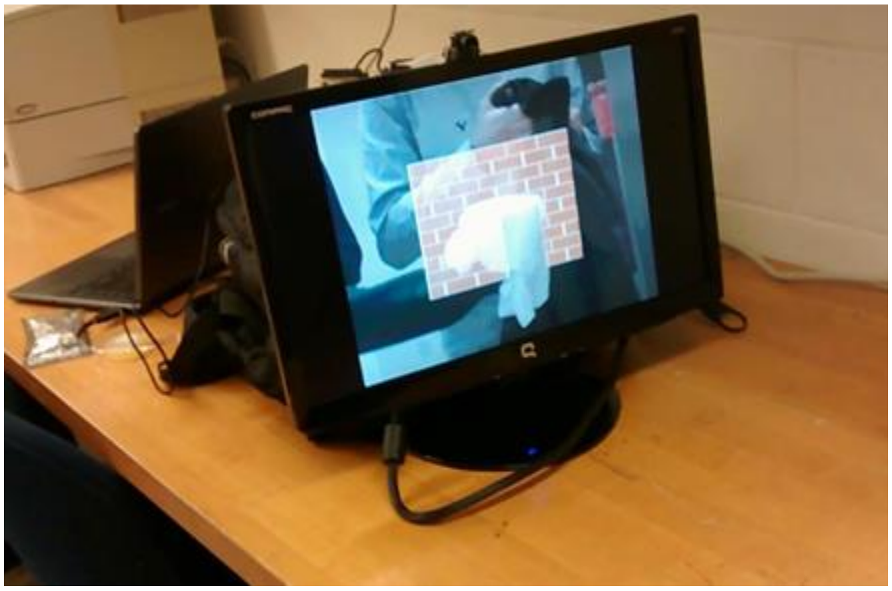

Figure 5. Photo of the wall being merged with the video input.

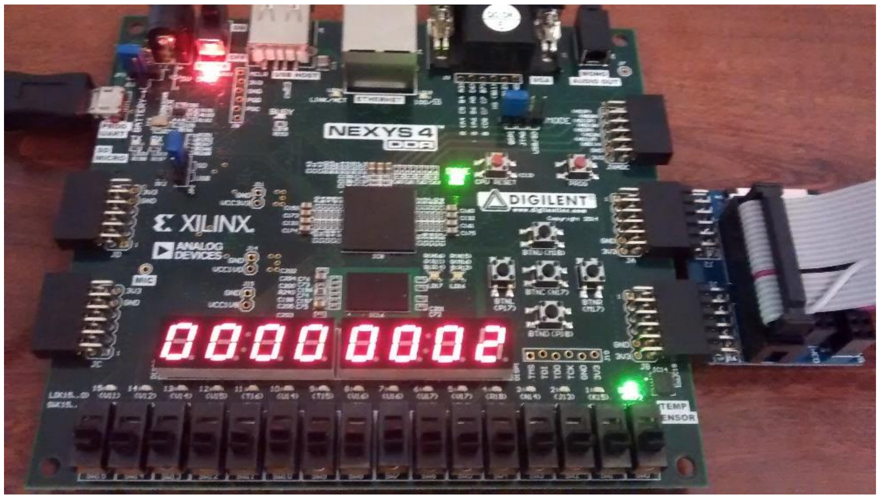

Table 4. Seven segments display with the level selection.

Figure 6 shows the 7 segments display with the number of the level being played.

At the end, the final project is very similar to what was proposed. The video data flow meets the requirements for a >30 frames per second output, with enough bandwidth and an imperceptible delay. The soundtrack and sound effects are well synchronized. The person detection could still be improved, but it is good enough to work in the setup environment. Overall, playing the game is an enjoyable experience.

2.2 Possible System Improvements

2.2.1 Enhance Colour Detection

One of the characteristics of the image processing algorithm used in this design is that it is heavily dependent on the background. To overcome this, two simple improvements are suggested. Both of them include capturing a frame of the environment without any players on it. In the first case, the number of pixels that exceed the colour threshold can be considered as an offset during the game. In the second case, that image could be used as a reference frame and the frame captured during the game only had to be compared to that.

2.2.2 New Holes and User Interface

A way to make the game more exciting and interactive is to add new hole designs, e.g. different combinations of rectangles or new shapes, and to improve the user interface, e.g. display messages using the graphics API.

2.2.3 Custom VGA Controller

The Xilinx TFT Controller has some problems that may affect this design. After launching, the image may be shifted a few pixels to the right (early VSYNC) and it is not possible to change the framerate at the output, which can be result in a bandwidth problem. This could be fixed with a custom VGA controller.

3 Description of the Blocks

3.1 Detailed Description of the hardware Blocks

3.1.1 Microblaze

Xilinx IP, v11.0. Xilinx soft processor. Configured as follows: Barrel Shifter enabled, Floating Point Unit with BASIC mode, Integer Multiplier with MUL32 (32-bit) enabled, Branch Target Cache enabled, Peripheral AXI Data Interface enabled, enable Interrupt, BRAM:128K.

Product Guide:

http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/mb_ref_guide.pdf

3.1.2 AXI Interrupt Controller

Xilinx LogiCORE IP, v4.1. Xcontac module is changed with three interrupt inputs correspondent to Uuartlite, AXI-Timer and GPIO-Pushbutton. Xintc is maintained unchanged as the original Xilinx IP, which can generate an external interrupt to feed the Microblaze processor.

Product Guide:

http://www.xilinx.com/support/documentation/ip_documentation/axi_intc/v4_1/pg099-axi-intc.pdf

3.1.3 AXI Uartlite

Xilinx LogiCORE IP, v2.0. This IP is used to output value through serial port to debug the project. Default settings.

Product Guide:

http://www.xilinx.com/support/documentation/ip_documentation/axi_uartlite/v2_0/pg142-axi-uartlite.pdf

3.1.4 AXI Timer

Xilinx LogiCORE IP, v2.0. Only one of the two counters is used in the project. Generate Mode is enabled to generate an interrupt in a time interval set by the Load Register.

Product Guide:

http://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v2_0/pg079-axi-timer.pdf

3.1.5 AXI GPIO Push Buttons

Xilinx LogiCORE IP, v2.0. A GPIO used to detect whether the user press the pushbutton. Only channel 1 is used with all 5 bits set as input. In order to generate only one Interrupt no matter how long the user press the pushbutton, axi_gpio_push_button is not connected directly to the push button pin but through a debouncer. The debouncer will count 5242879(0x4fffff) cycles of positive input from the external pin and maintain output high for 255(0xff) cycles. This setting is tested on the board and generates only one interrupt per pressing.

Product Guide:

http://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf

3.1.6 AXI GPIO Display

Xilinx LogiCORE IP, v2.0. This IP is used to get the value for 7-Segment display to show from Microblaze. Only one 32-bit wide channel is used.

Product Guide:

http://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf

3.1.7 Video In to AXI-4

Xilinx LogiCORE IP, v3.0. This IP is used as a bridge between a parallel video source to the AXI4-Stream Video Protocol Interface. At the input, only the signals vid_data, vid_active_video, vid_hsync and vid_vsync were used. The IP was customized with 1 pixel per clock, 8-bit video component width, RGBA video format and FIFO depth 1024. Its input is connected to the block videoin and its output is connected to the VDMA through an AXI4-Stream interface.

Product Guide: http://www.xilinx.com/support/documentation/ip_documentation/v_vid_in_axi4s/v3_0/pg043_v_vid_in_axi4s.pdf

3.1.8 Video Direct Memory Access (VDMA)

Xilinx LogiCORE IP, v6.2. This IP provides a direct memory access for the AXI4-Stream Video Protocol Interface. It is used to write video frames in the memory. Only Write Channel is enabled and it is connected to the output of the Video In to AXI-4. Its AXI4 Full Master Interface is connected to the memory bus. The block was customized as follows: only 1 frame buffer is used, write burst size 64 and no Genlock synchronization is needed.

The required software configuration is performed via its AXI4 Lite Slave Interface and it is contained in vdmaConfig.h.

Product Guide:

http://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf

3.1.9 AXI Thin Film Transistor (TFT) Controller

Xilinx LogiCore IP, v2.0. This IP is used as a VGA display controller. It sends a video frame from the memory to the VGA output. Burst size used is 64. The only configuration required is to set its base address register with the initial memory position. It works with a 1024x512 memory region that contains a 640x480 image.

Product Guide:

http://www.xilinx.com/support/documentation/ip_documentation/axi_tft/v2_0/pg095-axi-tft.pdf

3.1.10 Memory Interface Generator (MIG 7)

Xilinx IP. This IP is used with the Nexys4 DDR board files to implement the correct interface for the DDR2 memory in the board.

Product Guide:

http://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v1_4/ug586_7Series_MIS.pdf

3.1.11 I2C_Config

Semi-custom IP. This IP is used to allow software to configure the OV7670 Camera settings. Its AXI4 Lite Slave interface is used to get the camera settings which are written in an internal BRAM memory. The camera configuration is contained in the file cameraConfig.h. This IP implements an I2C standard communication to the OV7670 Camera to write into its registers. The following figure shows the top level diagram of this IP.

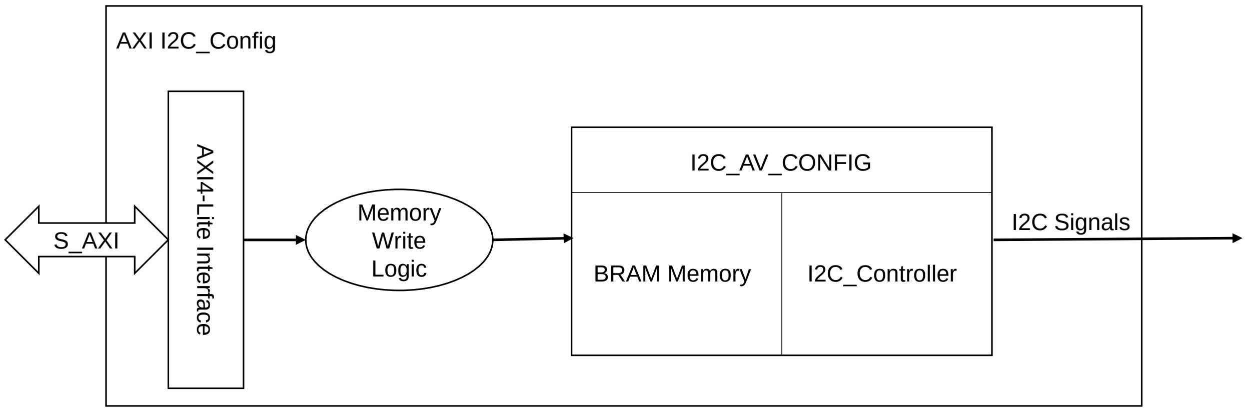

Figure 6. AXI Semi-custom camera configuration IP diagram.

This table shows the IP registers.

Address Offset	Register Name	Description
0h	WRT	Write camera configuration
04h	RST	Reset camera configuration

Table 5.Camera configuration IP registers.

3.1.12 videoin

Semi-custom IP. This IP is used concatenate two bytes of the OV7670 Camera output into a 32-bit word that is the expected input of the Video In to AXI4-Stream IP. It also creates the signal data_valid and divides the camera clock by two. The following figure shows the functional description of the IP. RGB444 colour format is used.

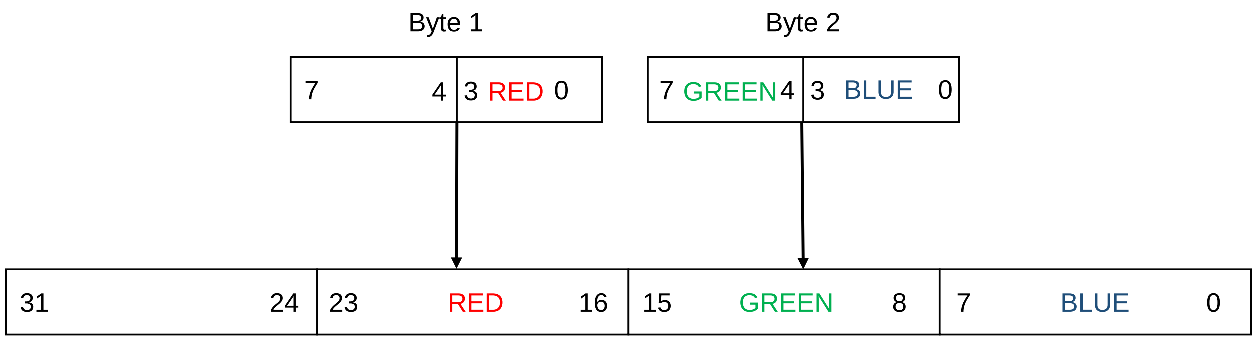

Figure 7. Functional description of the videoin IP.

3.1.13 myip_song

Custom IP. This Audio IP is a custom AXI4 Slave IP, with four configurable 32-bit registers and 3 256x32-bit BRAM. The audio IP can be configured through its AXI Slave interface to store the information of maximum 256 notes.

Register overview:

Address Offset	**Register Name**	**description**
0h	WRT	Most significant 8 bits are the address of the frequency mem, the rest 24bits give the value of the frequent stored in that address
04h	R	Most significant 8 bits are the address of the duty_cycle mem, the rest 24bits give the value of the duty_cycle stored in that address
08h	TIM	Most significant 8 bits are the address of the time mem, the least significant 16bits

		specify the upper 16bit stored in that address
0ch	CTR	BIT 0: enable audio IP BIT 1:Halt audio IP BIT[11:4]: amount of notes store in the memory

Table 6. Registers of the myip_song.

The AXI-Audio IP has 3 256x32-bit BRAM. Each store a note frequency, its duty cycle and duration of each note output. The memory can be written by writing to the four register in AXI-Audio IP's AXI interface.

To correctly configure the AXI-Audio IP, the user needs to understand how AXI-Audio IP works.

The AXI-Audio IP will output all the notes specified by the CTR register least significant 4th to 11th bit. This value does not have to be exactly the same as numbers of notes stored, it can be less or more. If it is smaller than the numbers of note stored, the IP will simply ignore the rest of notes stored and only play the amount of notes specify by the value. If this value is larger than the numbers of note stored, IP will simply output mute note for those notes that does not exist.

3.1.14 ImgProc

Custom IP. This IP is the core of the image processing in this project. It is capable of performing any pixel-by-pixel operation between two 4-bit RGB images. It is an AXI4 Full Master with burst size 64. This allows real-time image processing. The operation performed by this IP must be configured using software. All configurations used in this project are listed in the file ImgProc_config.h. The configuration is stored in an internal memory, which is used as a look-up table by this block. The functional description of this showed in the following image.

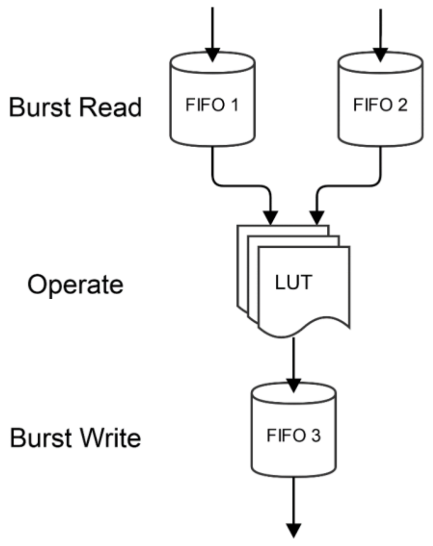

Figure 8. Functional description of the Image processing block.

The Finite State Machine of this IP implements 2 burst reads from the 2 input images, store the information in 2 FIFOs, perform the operation specified by the LUT between the two FIFOs and stores in a third FIFO, which is used to burst write back to the memory.

This custom IP was used to fully implement the CEMENT algorithm. A video frame is used as image 1, a wall image as image 2 and the look-up table is configured using a Matlab script, which is generated based in the algorithm equation. The threshold filter is partially implemented by this block. The same inputs are used. The LUT is configured to produce different outputs for three case: image 1 below colour threshold (no person), image 1 above colour threshold and image 2 is not wall (person inside hole) and image 1 above colour threshold and image 2 is wall (person hit wall). Some software processing is used to interpret this output.

This block was also used during the game flow as a hardware accelerator, replacing the processor for functions like erase memory (write all black) and copy images (block is configured as a MUX).

3.2 Detailed Description of the Software Blocks

3.2.1 Create Wall

The create wall library contains functions that create and manipulate the walls. As an example It has the create_wall that is a for loop that starts on a position of the memory and writes pixel by pixel following a logic of columns and rows that describe the bricks of the wall.

3.2.2 Hardware Initialization

Some blocks require some configuration before being used. The image size on memory and on display must be configured in the VDMA. The USB/Uart communication must be initialized. The camera settings must be passed to the camera. These configurations are performed at the beginning of the main program. The Image Processing IP and the Audio IP must also be configured. However, different songs are played and different operations are performed throughout the execution of the game. Each configuration is stored as a function that is called when required.

3.2.3 Game Flow Control

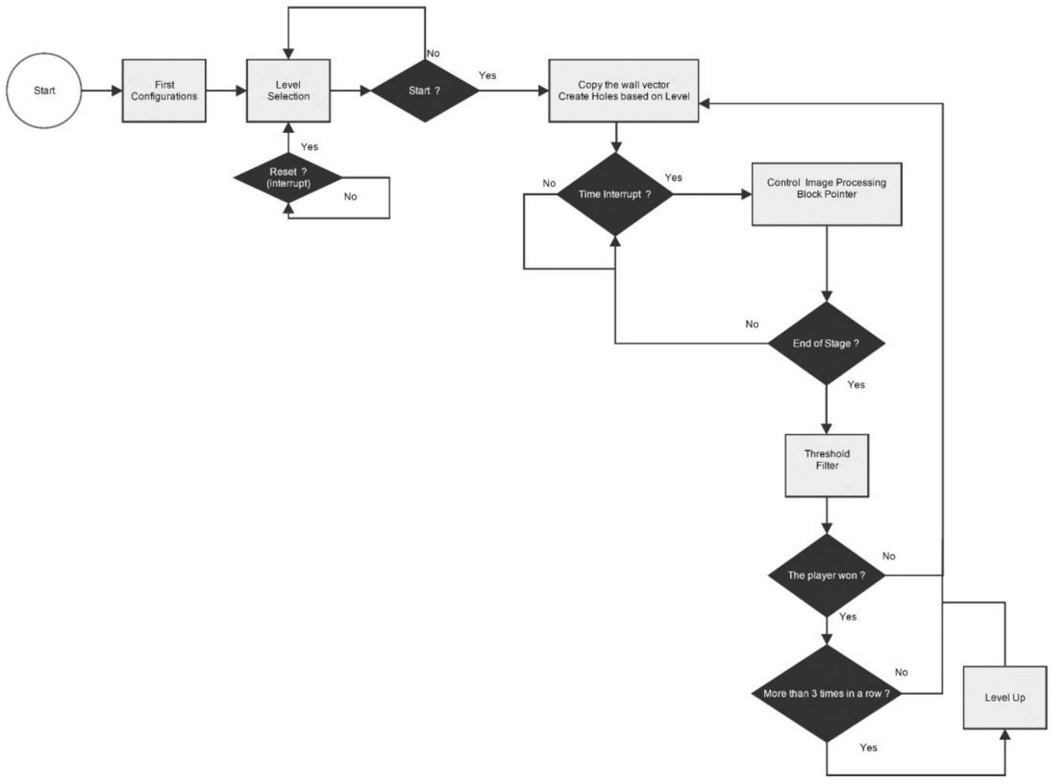

Figure 9. Flow diagram of the main control program.

3.2.4 User Interface

The game start with difficulty level 1 and user can press the central push button to reset the whole game and come to a blue screen, the blue screen act like the control terminal in PC , it awaits the player to choose difficulty level 1 through 5 by pressing up/down push button. The game will not restart until the user press the left push button.

After each round finishes, the game decides whether or not the wall hit the player by showing either a red or green screen meaning the user lose or win. Then, the game will start at the same level again if the player wins, player need to win 3 times to get to the next difficulty level if the current level is not level 5. If the player lose at any of the level, the next round will start with difficulty level 1.

4 Description of the Design Tree

Vivado project	
HoleInTheWall.v	Top-level design, instantiates block diagram and other sources.
Hole_wrapper	HDL wrapper for block diagram.
display.v	Seven segment display controller.
debounce.v	Push button debouncer.
SDK Project	
MicroblazeController.c	Main program: Contains game flow control.
audioip.h	API to the audio interface and music
cameraConfig.h	Configuration for the camera
create_wall.h	API that hold all the functions of creating wall and generating the random holes.
ImgProc_config.h	Configuration for the Image processing block and LUT of the algorithms.
lscript.ld	C linker.
platform.h	Generated Xilinx Uart configuration
vdmaConfig.h	Configuration for the VDMA Block
prototype_bsp	Generated Board Support Package files.

Table 7. Description of the design tree.

5 Project Schedule

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6
Gustavo	Study a way to simplify CEMENT expression and evaluate what is lost in doing it.	Be capable of capture video and display on the screen. Implement CEMENT on Microblaze.	Study how to implement CEMENT on hardware. Start the game control logic.	Implement CEMENT on hardware and continue with the game logic.	Finish game software.	System integration and debugging
	Study a way to simplify CEMENT expression and evaluate what is lost in doing it.	Create design video input video output controlled by MicroBlaze.	Make the OV7670 configurable by software.	Create an AXI4 Master that reads from two memory positions using burst.	Implement CEMENT algorithm in AXI4 Master IP and create driver for it.	System integration and debugging
Heyang	Learn how to implement the threshold filter on hardware.	Be capable of capture video and display on the screen. Implement threshold filter on Microblaze. Study audio output on the board	Be capable of output audio from the board.	Manipulate images on memory.	Implement center of mass algorithm on hardware. Detect if the person hit or missed the wall.	System integration and debugging
	Learn how to implement the threshold filter on hardware.	Study audio output on the board. Implement basic audio output block,	Implement system that outputs sequence of musical notes and code driver for this block	Finish audio system and create some songs. Start Time and push-button Interrupt.	Work with drivers to further improve game software and manage hole positioning.	Finish the push button interrupt API.
João Marcus	Implement the threshold filter and the CEMENT algorithm on software.	Be capable of capture video and display on the screen. Implement threshold filter on Microblaze. Study audio output on the board	Design different wall images and load them on memory. Be capable of output audio from the board.	Finish wall design. Tune filter threshold. Implement center of mass algorithm on hardware.	Implement center of mass algorithm on hardware. Detect if the person hit or missed the wall.	System integration and debugging
	Implement the threshold filter and the CEMENT algorithm on software.	Implement threshold filter on Microblaze. Implement basic audio output block.	Design different wall images and load them on memory.	Implement random function and start writing main program.	Finished main program and integrate time interrupt.	System integration and debugging