ECE532 Group Report
FPGA Super Hexagon

April 13th, 2017

Group 11
Zhiming (Charles) Chen
Yuan (Michael) Feng
Mingqi (Matthew) Hou

Table of Contents

Overview

Background and Motivation

Our project aims to implement an FPGA version of the “Super Hexagon” game.[1]

The player controls a tiny arrow in the middle of the screen. It can only rotate around the center which has a hexagonal shape. The surroundings constantly produce blocks in each of the six sides of the hexagon and approaches the center. The player needs to move the arrow to an open side (with no blocks near the center) to survive. The entire screen is constantly rotating (with varying angular speeds and/or direction). The goal of the game is to survive as long as possible. There are multiple levels of the game with different difficulties (faster blocks and/or faster rotation speed).

The main goal (and challenge) of this project is to make the block drawing algorithm entirely in hardware because software rendering is too slow for adequate screen refresh rate.

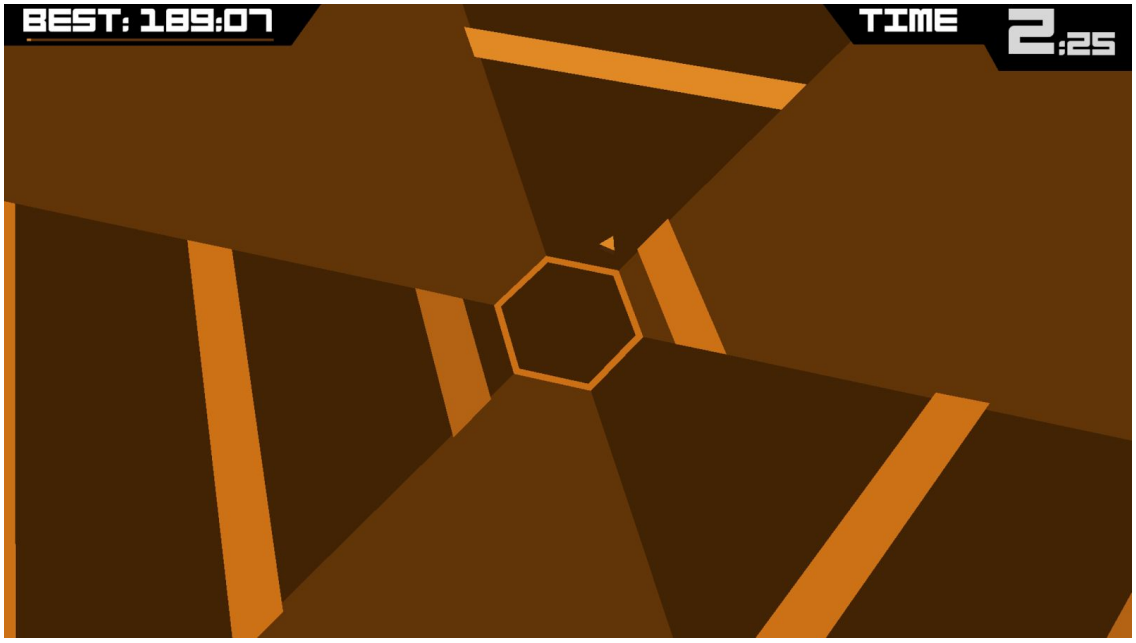

Figure 1: Screenshot of Game Play from mobile version of Super Hexagon

[1] Link to video of game play on a mobile device:
https://www.youtube.com/watch?v=5mDjFdetU28

Project Description

This project is implemented with an Xilinx Artix-7 FPGA on the Digilent Nexys 4 development board. The main components include MicroBlaze processor, on-board DDR memory, hardware render module for frame rendering, TFT module for VGA output, LCD monitor for visual display, joystick for user input, and UART (and Laptop) for debugging purposes.

Goal

The goal of the project is to design and implement a joystick controlled video game similar to “Super Hexagon”. The project aims to generate RGB graphic output at a reasonable resolution and refresh rate.

Functional Requirements

- ## Function

  The design should implement Super Hexagon Game
  - The design should have a timer.
  - The design should take input from joystick controller.
  - The design should produce graphic output.
  - Output frames should be in constant rotation.
  - Game should generate blocks approaching the center from the edges.
  - Game should terminate when the user-controlled arrow is hit by a block.

- ## Objectives
  - Display resolution should be high. Goal: 1280*720 (720P).
  - Refresh rate should be high. Goal: 30FPS.
  - Objects generated should form certain patterns. Goal: Five different patterns.
  - The game may have audio output. Goal: Output the original Super Hexagon’s background music.
  - The game may have multiple level with different difficulties (varies rotation speeds and object speeds). Goal: Three levels.

- ## Constraints
  - The design must use a Microblaze Processor.
  - Resolution must be greater than 480 * 320.
  - Refresh rate must be greater than 15 Frames per Second.
  - Output must be in RGB color space with 8-bit per channel.

Intellectual Properties (IP)

Table 1 Intellectual Properties used in the system

IP Name	Existing / Custom	Brief Description
MicroBlaze Processor	Existing IP	Serves as main control, hosts game software Generates parameters for render (block positions, rotation angle, cursor positions etc.)
Block RAM	Existing IP	Stores start/end screen pixel data
DDR Interface	Existing IP	Interface to off-chip DDR memory DDR memory holds frame buffers
TFT Interface	Existing IP	Interface to VGA pins for external monitor Monitor serves as system output
Render Module	Custom IP	Main drawing algorithms, responsible for rendering entire frames after receiving parameters from MicroBlaze processor
Timer Module	Custom IP	Keeps time based on 100 Mhz clock
Joystick Module	Custom IP	Interface to PMOD Joystick through SPI Joystick serves as user input to the system

Please see **Description of Blocks** section for more details of each IP block.

System Block Diagram

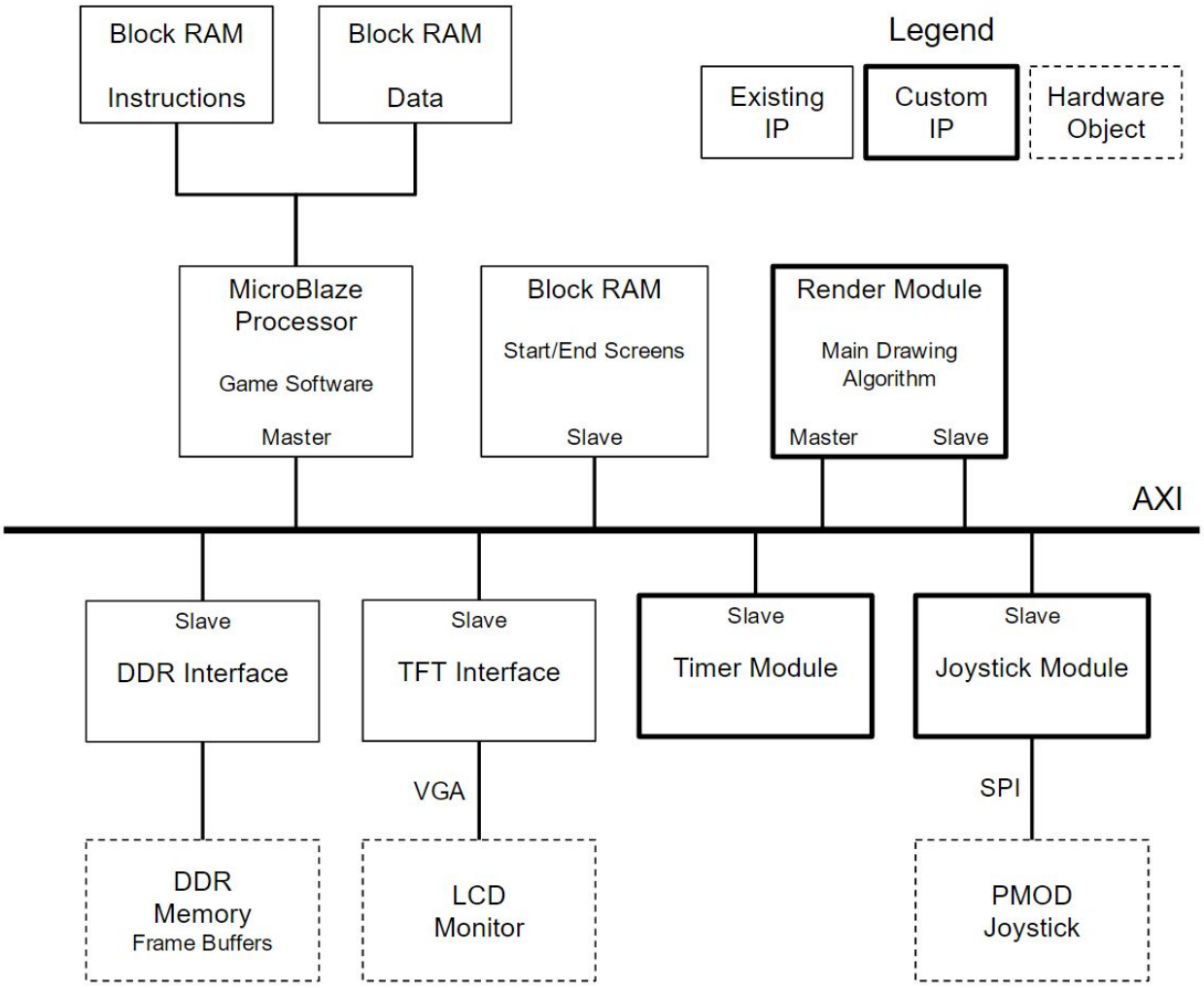

Figure 2 System Block Diagram

Outcome

Project Results

We successfully implemented the targeted design (except for audio output) and the final product is a playable Super Hexagon game running on the Nexys 4 DDR board with VGA output with 640 * 480 resolution. The audio output portion of the design was originally set as an optional component and was not implemented due to time limit.

Table 2 Objectives and Results

Objective	Goal	Result	Pass/Fail	Reason
Resolution	1280 * 720	640 * 480	Fail	Existing TFT module only supports 640 * 480 resolution
Refresh Rate	30 FPS	60 FPS	Pass	
# of Patterns	5	5	Pass	
# of Levels	3	7	Pass	
Audio	Music	None	Fail	Not enough time

Overall the system works very well and the game is fun to play (and really difficult, which is intended). For project source code, please see **Description of Design Tree** section.

Demonstration

Here is the link to a short video of the working project demonstration:
https://drive.google.com/open?id=0B6qy-kEGox7dRHkzaE9VampfczA

Potential Improvements

Some potential improvements to the system:

- The visual effect of the game can be improved
    - The corners of blocks appear weird sometimes due to integer rounding errors
    - Especially severe with near-horizontal edges (very small angle)
    - With low resolution frame, there doesn't seem to be an easy fix to this problem
    - Probably requires additional condition checks and rounding fix

- More granular global angles and block levels for each lane
    - Currently there are only 14 levels for each lane and 36 levels of global angle (10 degree per level)
    - It should be relatively easy to add more block levels for each lane
    - Global angles are more difficult to deal with because more granular angles will make near-flat edges more flat (currently the smallest possible angle is 10 degree). With more angle levels, the smallest possible angle will be even smaller and integer rounding error will be more severe.

- Higher resolution
    - Requires custom IPs for VGA interface because the existing TFT module only supports 640 * 480 resolution.
        - Can make pixel arrangement more compact
            - i.e. 12 bits per pixel instead of 32 bits
    - Higher resolution may make graphics look better, and have less integer rounding errors.
    - Higher resolution will take more space for frame buffer. Consider the size the DDR memory (128 Megabyte), this shouldn't be a bottleneck.

Project Schedule

Original Schedule (From Proposal)

Milestone #1: Demonstrate a basic working rendering module and its working testbench.

Milestone #2: Testbench Demo: Show testbench working for basic rendering and rotation modules (pattern generation can be broken down to lines, polygons, backgrounds etc., which can be tested individually).

Milestone #3: Complete Pattern Generation Module (PGM).

Milestone #4: Complete Frame Rotation Module (FRM).

Milestone #5: Mid-Project Demo: Show PGM and FRM working with other IPs including VGA and MicroBlaze processor.

Milestone #6: Complete software programming and add optional features (complex pattern, multiple level, high score etc.).

Milestone #7: Final Demo: Complete project including all testings.

Actual Schedule

Milestone #1: Figure out VGA functionality. Determine overall design structure. Determine block rendering algorithm.

Milestone #2: Build hardware control unit for render module (custom IP). Write simple software for testing. Determine rotation algorithm. Write software-implemented block rendering algorithm.

Milestone #3: Testbench Demo. Build testbenches for three components: render IP control logics; software-implemented rotation; software-implemented block rendering.

Milestone #4: Implement rotation module in hardware. Integrate rotation module into system. Draw ASM chart for block rendering algorithm. Implement SPI module (for joystick).

Milestone #5: Mid-Project Demo: Implement quadrangle drawing module in hardware (formerly referred to as “block rendering algorithm). Integrate it into system. Implement burst transactions. Integrate joystick module into system. Implement cursor drawing in hardware.

Milestone #6: Design game levels and patterns. Implement timer module. Add center hexagon drawing in hardware. Add start/end screen. Design software architecture.

Milestone #7: Integrate timer module into system. Add level selection on start/end screen. Finish writing software for game.

Notable Changes

The actual schedule looks different than the originally proposed schedule. There are multiple factors to this result:

- Testbench demo was moved from milestone 2 to milestone 3.
    - Happened to the entire class. Not in our control.
    - Gave us more time to design system architecture before writing tests.
- Rendering module and rotation module was combined for efficiency.
    - Old schedule separated these two modules.
    - Rotation can be done efficiently on block vertices so there is no reason to separate the two.
- Original schedule didn't specify work for milestone 7.
    - Parts of milestone 6 work was moved to milestone 7 in actual schedule.

Comments to project schedules

Overall, the schedules are not drastically different. By milestone 5 (mid-project demo), we have accomplished what we proposed to be done by that time. Most of the hard work was done prior to (and including) milestone 5. The last two weeks were mostly fine tunes and final touches to finish the project. This was ideal because all group members were busy with other courses during the last two weeks and efforts could not be guaranteed. By doing most of the work prior to that, we made sure this project was in good shape.

Description of Blocks

Existing IPs

MicroBlaze Processor

MicroBlaze 9.6 (Rev. 1)
Clocking Wizard 5.3 (Rev. 1)
AXI Interrupt Controller 4.1 (Rev. 7)
MicroBlaze Debug Module (MDM) 3.2 (Rev. 6)

Central control processor. It runs the game software and generates parameters used by the render module to update frames. On-screen menu and time display is done through the “xtft” library provided by Vivado SDK.

Block RAM

Block Memory Generator 8.3 (Rev. 3)
AXI BRAM Controller 4.0 (Rev. 8)

Used for storing start and end screen pixel data. Two initialization files are used (start_screen.coe, and end_screen.coe). Block Memory Generator is set to “Stand Alone” mode despite being connected to AXI BRAM Controller. This causes a critical warning during block design verification but can be safely ignored.

DDR Interface

Processor System Reset 5.0 (Rev. 9)
Memory Interface Generator (MIG 7 Series) 4.0

Interface to on-board DDR memory used for storing frame buffers. Frame buffers can be directly written by MicroBlaze processor and the Render module. Two frame buffers are used in an alternating fashion (ping-pong)

TFT Interface

AXI TFT Controller 2.0 (Rev. 13)

Interface to on-board VGA controller and external VGA pins. Natively support 640 * 480 at 60 frames per second. It constantly pulls from a preset memory location and expects pixels to be stored in 32-bit format.

Custom IPs

Render Module

This is the main module for this project. It is responsible for refreshing the entire screen. It has an AXI Slave interface connected to the MicroBlaze processor for receiving parameters. It also has an AXI Master interface connected to the DDR controller for writing output pixel data to frame buffers. Full version of the AXI Master interface is used for burst transactions.

There are 11 slave registers used in the AXI Slave interface:

Table 3 Registers for AXI Slave Interface on Render Module

Register	Address	Access	Description
slv_reg0	Base	Read/Write	Set target frame buffer base address
slv_reg1	Base + 0x4	Read Only	Color information, Also serves as start signal
slv_reg2	Base + 0x8	Read/Write	Done signal. Turns to 1 when drawing is done.
slv_reg3	Base + 0xC	Read/Write	Block information for lane 0
slv_reg4	Base + 0x10	Read/Write	Block information for lane 1
slv_reg5	Base + 0x14	Read/Write	Block information for lane 2
slv_reg6	Base + 0x18	Read/Write	Block information for lane 3
slv_reg7	Base + 0x1C	Read/Write	Block information for lane 4
slv_reg8	Base + 0x20	Read/Write	Block information for lane 5
slv_reg9	Base + 0x24	Read/Write	Global angle information
slv_reg10	Base + 0x28	Read/Write	Cursor position information
slv_reg11	Base + 0x2C	Read/Write	Not used. Reserved for future function.
slv_reg12	Base + 0x30	Read/Write	Not used. Reserved for future function.
slv_reg13	Base + 0x34	Read/Write	Not used. Reserved for future function.
slv_reg14	Base + 0x38	Read/Write	Not used. Reserved for future function.
slv_reg15	Base + 0x3C	Read/Write	Not used. Reserved for future function.

The overall workflow is as follow:

- Refresh background
- For **i** from 0 to 5
    - Draw blocks on lane **i**
- Draw cursor
- Draw center hexagon

Each block (quadrangle) is drawn based on the following block drawing algorithm:

- Calculate final angle based on lane number and global angle
- Look-up radius and trigonometric parameters for a given block based on final angle
- Calculate vertex coordinates based on radius, **sin** and **cos** of final angle
- Look-up slope and inverse slope of each edge based on **tan** and **cot** of final angle
- Calculate equation of each edge (slopes are already known, need to calculate offsets)
- For **Y** from minimum Y-coordinate of this block to maximum Y-coordinate of this block:
    - Calculate intersection X-coordinates between each edge and the current **Y**
    - Pick the two intersections that are inside the edges' actual Y range
        - The other two intersections are on line extensions, not real
    - Fill pixels from the smaller X-intersection to the larger X-intersection

The diagram below shows a specific Y during the Y-scan process

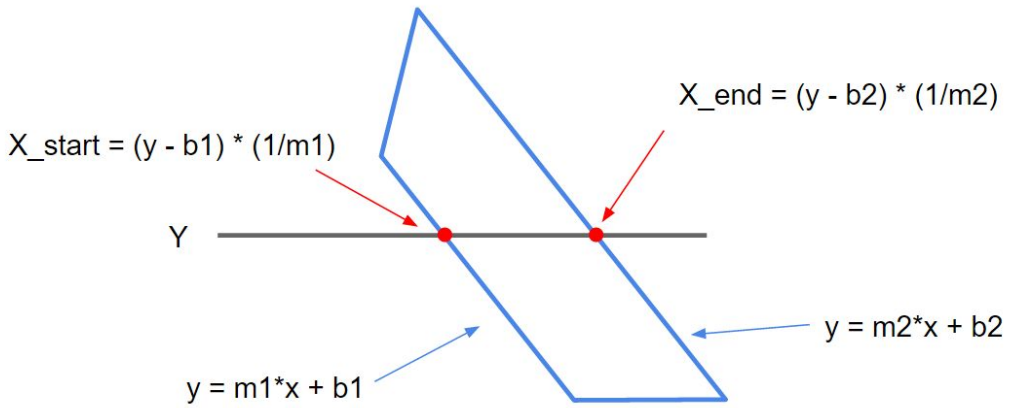

Figure 3 Block Drawing Algorithm

Timer Module

Keeps time, to be used as score for the game. Processor can read the slave registers to obtain timing information.

There are 3 slave registers used in the AXI Slave interface:

Table 4 Registers for AXI Slave Interface on Timer Module

Register	Address	Access	Description
slv_reg0	Base	Read/Write	Clock counter. From 0 to 1000000.
slv_reg1	Base + 0x4	Read/Write	Second counter. From 0 to 59.
slv_reg2	Base + 0x8	Read/Write	Minute counter. From 0 to 59

Joystick Module

Interface to PMOD 2-Axis Joystick through SPI. Processor can read the slave registers to obtain input coordinates and button response from the joystick.

There are 6 slave registers used in the AXI Slave interface:

Table 5 Registers for AXI Slave Interface on Joystick Module

Register	Address	Access	Description
slv_reg0	Base	Read/Write	Calibration
slv_reg1	Base + 0x4	Read Only	Button response.
slv_reg2	Base + 0x8	Read Only	Y upper-byte
slv_reg3	Base + 0xC	Read Only	Y lower-byte
slv_reg4	Base + 0x10	Read Only	X upper-byte
slv_reg5	Base + 0x14	Read Only	X lower-byte

Description of Design Tree

Project source files are hosted on Github:
https://github.com/Charles-CZM/G11_SuperHexagon.git

There are two main sections in the repository: **doc** and **src**
The **doc** directory contains design document, demo presentation, and video.
The **src** directory contains all source files needed to compile and run this project.

- resource
    - start_screen.bmp: Start screen drawing
    - end_screen.bmp: End screen drawing
    - bmp2coe.py: Python script used to convert bmp files to coe format.

- software
    - superhexagon.c: Main software code

- superhexagon
    - superhexagon.xpr: Vivado project file
    - super_hexagon.ipdefs/ip_repo_0: Custom IP definitions
        - hexagon_render_2.0: Render Module (main custom IP for project)
        - joystick_1.0: Joystick Module
        - timer_2.0: Timer Module
    - super_hexagon.srcs: design sources
        - constrs_1/new/super_hexagon.xdc: Constraint file
        - sources_1:
            - start_screen.coe: Start screen memory initialization file
            - end_screen.coe: End screen memory initialization file
            - bd/block_system: Block design files
            - imports/hdl: HDL wrapper file

Note: This project requires a Digilent Nexys 4 DDR development board and a PMOD 2-Axis Joystick to run.

To run this project:

1. Clone the Git repository (using “git clone”) and pull all contents.
2. Make sure Vivado 2016.2 (or higher) and SDK are properly installed on your computer, with support for Artix 7 FPGA. Only Windows platform was tested, though Linux platform should also work.
3. Open super_hexagon.xpr under src/superhexagon/.
4. Generate block design. If block design verification gives critical warnings about mismatches on block ram connection mode, ignore them. This is due to setting block

ram in “Stand Alone” mode (in order to use coe files for initialization) while connecting through AXI controller. This is not a real error.
5. Generate bitstream file. Vivado should prompt to run synthesis and implementation because they haven’t been run, click “Yes”.
6. File → Export → Export Hardware. Make sure to include bitstream file.
7. File → Launch SDK. Vivado SDK should launch.
8. Create a new application project: File → New → Application Project. Replace the main C file with the provided C file under src/software/superhexagon.c. You may need to rename it to the recognized file name by the project (e.g. helloworld.c or memorytest.c) or add it to the source file tree of the project.
9. Make sure the Nexys 4 DDR board is connected to your computer, then program the FPGA: Xilinx Tools → Program FPGA → click “Program”.
10. Compile project: Project → Build Project (or select “Build Automatically” and just save the C file. This will automatically build your project every time you save).
11. Run project: Run → Run (or ctrl + F11).

Tips and Tricks

1. **Block RAM Initialization**: Block RAMs can be initialized while connecting to AXI Block RAM Controller. After auto-connection is done, Block RAM Generator will be set to “BRAM Controller” mode which disables initialization. However, you can manually select “Stand Alone” mode and specify a initialization file. This will cause a critical warning during verification but can be safely ignored.

2. **SDK Build Error**: Sometimes when bitstream files are updated or SDK is being opened for the first time, the BSP files may not be built properly. This usually results in build failure for your project. Simply right-click on “<project name>_bsp” in the project explorer on the left and select “Re-generate BSP Sources” will solve this problem.

3. **Keep Block Design HDL Wrapper Edits**: When creating block design HDL wrappers, if you plan to make edits, make sure to select “Copy generated wrapper to allow user edits” so Vivado doesn’t overwrite your edits every time you generate block design. This is especially useful for using the built-in TFT Controller IP with Nexys 4 DDR board because of the VGA pin discrepancy (TFT Controller uses 6-bit per color while Nexys 4 DDR has only 4-bit per color. The HDL wrapper needs to be edited to avoid output mismatch).