Trever Wagenhals  
FPGA Logic Design  
Homework #6

Breakout Specification

Version 1.0 – December 1, 2018

**Revision History**

| **Version** | **Date** | **Name** | **Description** |
| --- | --- | --- | --- |
| 1.0 | 10/01/2018 | Trever Wagenhals | Initial Creation |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**System Requirements**

**Breakout Game**

VERSION: 1.0 REVISION DATE:

Approval of the System Requirements indicates an understanding of the purpose and content described in this deliverable. By signing this deliverable, each individual agrees with the content contained in this deliverable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Approver Name** | **Title** | **Signature** | **Date** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[**1** **Breakout Top Wrapper** 5](#_Toc531462435)

[**2** **VGA Controller** 6](#_Toc531462436)

[**3** **Breakout Paddle Controller** 6](#_Toc531462437)

[**4** **Breakout Controller** 6](#_Toc531462438)

[**5** **Breakout Ball Controller** 6](#_Toc531462439)

[**6** **Breakout Paddle Controller** 6](#_Toc531462440)

[**7** **Breakout Block Controller** 6](#_Toc531462441)

[**7** **FPGA Board Wrapper, Constraints, and Timing Diagram** 6](#_Toc531462442)

# 

# **1 Breakout Top Wrapper**

The breakout top wrapper contains all the instantiations of the modules for the breakout design to work. Here, the VGA controller, Breakout controller, Breakout ball controller, Breakout block controller, and Breakout paddle controller will be connected to each other. There is minimal actual logic living at this level of hierarchy. Currently, the only logic that lives here is the logic to pause the game. Technically, this logic could probably be moved into the Breakout controller module to clean up the top-level wrapper.

From the top-level wrapper, the inputs and outputs for the game to function can be easily seen. The main inputs that need to be present for the game to function are the clock, reset, move paddle directional bits, and the VGA data bits and vertical and horizontal sync strobes. There are also bits coming in to determine the ball and paddle’s speed. These bits should be hooked up to a switch and will be deciphered by the Breakout controller logic. LED bits are also present to help debug. Scoring is also output to be displayed on a 7-segment display, although no functionality has been implemented for this.

The only interesting design considerations at this point would be the generate loops around the Breakout block controllers. There are two for-loops to generate 3 rows of 8 blocks at the top of the screen, using a set stride to place the blocks directly after one another.

# **2 Breakout Package**

This package contains all the global constant signals that will be used across all the modules. The main purpose of this module is to allow the game to function identically at varying resolutions with minimal change to the code. Numerous functions are defined here to select the appropriate VGA characteristics based on the specified frequency. These functions allow for the VGA controller to know how to display the information to the screen appropriately. Past this, other variables such as the paddle’s starting location and the ball’s starting location are defined based on the set resolution, allowing the paddle to always be placed at the bottom of the screen and the ball to be placed in the center.

A few constants are also defined for different movement speeds. These movement speeds are used by the Breakout controller when determining how fast the ball and paddle should move across the screen and allow for finding a good speed for the game to be easier.

The resolution was set to 1280x1024 here because I was unable to get my monitor to work at the 640x480 resolution specified. The higher resolution actually requires tighter timing and more logic, so if its compatible at the higher resolution, it should also work at the lower resolution if the monitor being used supports it.

# **3 VGA Controller**

The VGA controller is responsible for taking in the clock and generating sync pulses for the horizontal and vertical regions. Basically, a horizontal counter and vertical counter keep track of what the current pixel within a frame you are generating based on the resolution specified. The vertical counter only increments after the horizontal counter reaches its maximum pixel count. While the horizontal and vertical counter increment, certain pixels are considered sync pixels and should generate a sync strobe to the VGA port. These pulses are put out to allow the screen to sync up with the FPGA.

The current pixel information for the horizontal and vertical axes are also output so that other controller blocks can know what the next pixel out is to determine if anything should be displayed at that pixel.

All the vertical and horizontal regions needed to generate the appropriate sync signals are specified automatically within the Breakout package based on the resolution set within the package. These values are fed into generics during the instantiation to allow for this module to function the same with changing resolutions. The clock that comes into this wrapper is expected to already be the appropriate frequency based on the set resolution.

Although there is a reset input to this module, I determined that resetting the VGA controller with the game logic caused the display to be dropped from the monitor and the monitor would need to take several seconds to sync again since the sync pulses were stopped during the reset period. Removing this allowed for a much smoother feeling reset to occur.

# **4 Breakout Paddle Controller**

The paddle controller is responsible for generating the VGA signals for when the current pixel is a paddle pixel and is responsible for determining if the ball is within the paddle to signal the ball to change directions. The color of the paddle is a generic that can be set to anything, but for our case it is just white. Whenever the current frame’s pixel being drawn is the paddle’s pixel, this VGA color will be output.

A simple state machine lives here to control when the paddle determines to update the display information. The paddle logic waits until the last graphical pixel is drawn and the blanking period has begun. Once begun, the ball’s position is looked at to see if there is a collision. If so, the ball is signaled to update its direction. On the last blanking pixel within a frame, the paddle’s direction buttons are looked at to see where the paddle is moving next. Based on the direction, the paddle’s position is updated for the next frame. Minimal logic is also needed to ensure that the paddle handles the edges of the screen correctly.

Pause game logic lives around the paddle position update logic to ensure that the paddle never moves while the game is paused. Technically, this logic should also live around the ball update logic too in the case where you may pause the game right when the ball is on the paddle, incorrectly toggling the ball’s direction each frame. This bug is minimal and known, so it wasn’t corrected.

# **5 Breakout Controller**

The Breakout controller has two mean functionalities: determining the final VGA value to display and determining the speed of the ball and paddle. The speed of the ball and paddle are determined by looking at the incoming bits tied to the switches on the Basys3 board. Based on the setting, the ball is set to move a fixed number of pixels so that it can move across the screen in the time specified. All calculations for this are done within the Breakout package.

The other functionality involves taking all the VGA sources and ORing them together to determine what the correct VGA value out is. This include the blocks, paddle, and ball all reporting the VGA value they want to display. If an object is not located at the current pixel being drawn, it will report all zeros for the VGA value. Since only one object should ever be at a location at a time, that means the output VGA value will always reflect the one object that sent a non-zero value. This approach was chosen because it doesn’t require an BRAMs like the proposed solution, which would fill up incredibly quickly even at lower VGA resolutions.

# **6 Breakout Ball Controller**

The ball controller probably has the most complicated logic as it has the more complex edge conditions to handle. Based on the directions the ball is moving, the ball speeds calculated from the Breakout controller will be used to update the ball’s position every frame. For the new positions, overflow and underflow need to be checked each time to determine if the boundary of the display has been reached. If it has, the ball must be updated to be touching the edge appropriately. Since the ball is multiple pixels wide, different pixels are looked at for each wall. Although multiple positions are used to track the ball’s position, the paddle and blocks are only aware of the ball’s center pixel to determine if a collision has occurred.

The ball must take every update strobe from the blocks and paddle to determine if an update in direction is required. It also determines if an update in direction is required if the ball is in contact with a wall.

The ball checks the collision just before the end of the blanking period, giving the other modules ample time to signal the event. If the ball touches the bottom of the screen, the game is over, and no logic is run anymore.

There are some interesting edge cases for the ball moving down and approaching the paddle. Based on the ball’s speed, it’s possible that it moves straight through the paddle’s width and doesn’t get detected. To avoid this, special conditions of advancement are looked at to determine if it should be registered as a hit or not.

# **7 Breakout Block Controller**

The block controller’s two major functionalities are to output the VGA value if the current pixel is within the block and to signal to the ball that a block has been hit before removing the block.

There is a simple state machine to wait until the end of the graphical pixels and the start of the blanking period. At this time, we determine whether the ball has hit any of the block’s walls. If so, the ball is signaled to update its direction. On the last blanking pixel, if the ball has hit the block, the state machine will go into a removed state, ensuring the logic doesn’t trigger again. If not, the block idles again for the next frame.

The ball’s movement speed is needed to calculate exactly where the block was hit since the ball may advance into a block. Each wall is looked at to determine if the block has been hit from a certain angle. It’s possible for a vertical and horizontal wall to both be considered hit for the edges of the blocks due to the implementation. This implementation is fine, except for cases where a block has no other block next to it and the ball updates both directions, making the movement look odd. To fix this, a block would have to be aware of it’s neighbors to know whether it should update in multiple directions or not.

# **8 FPGA Board Wrapper, Constraints, and Timing Diagram**

There are several levels of hierarchy above the Breakout top wrapper to help fully understand the ultimate path of all the logic. This specific design was implemented on a Basys3 board, and so the wrapper above the Breakout wrapper is the Basys3 top wrapper. This wrapper contains all the possible functionalities of the Basys3 board. This includes switches, LEDs, buttons and debounce instances, the seven-segment display, and other support not shown here such as PS2 interfacing. A lot of the switches and LEDs have minimal logic here to help with debugging the system currently. Also, the MMCM instance needed to convert the clock from Basy3’s native 100Mhz clock to the VGA clock based on the resolution is placed here. Currently, it is hard coded to generate the resolution for a 1280x1024 display.

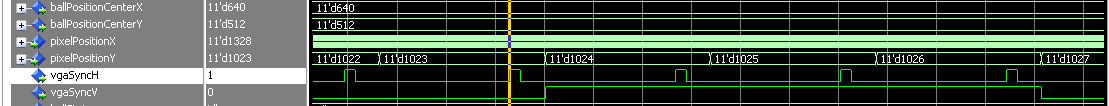
The seven-segment module and debounce modules will not be discussed as the seven segment is not expected for the project and thus is only partially implemented and the debounce module is a simple counter.

The next two levels of hierarchy are technically not needed, as all of the constraints for the inputs and outputs could be defined directly to the Basys3 top wrapper level. The benefit of the next two wrappers provide clarity on the entire system setup though for future development.

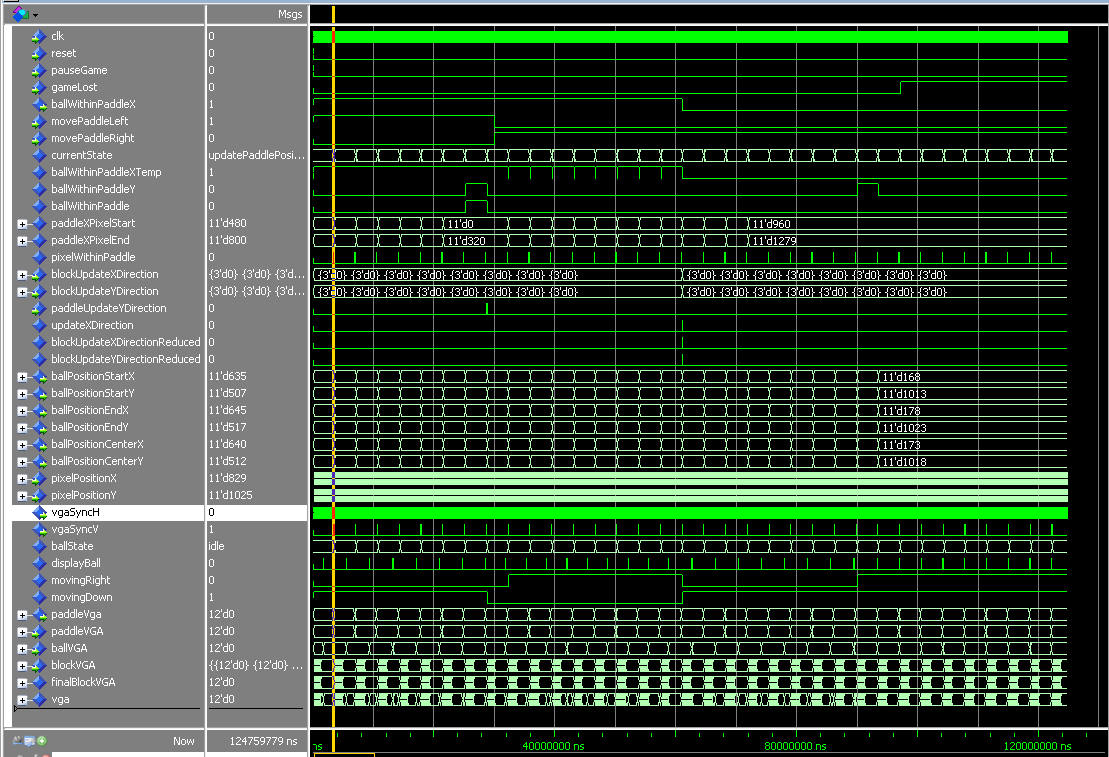
The Basys3 port wrapper is a level to handle converting all of the Basys3’s functions to their actual pin naming within the Basys3 specification. This level is important in that it allows easy tracing from the function to the pin to verify that you have actually hooked up the logic to the correct pin. Once every pin is within this file, no additional work is needed to access the other pins for future projects and unused pins can just be ignored.

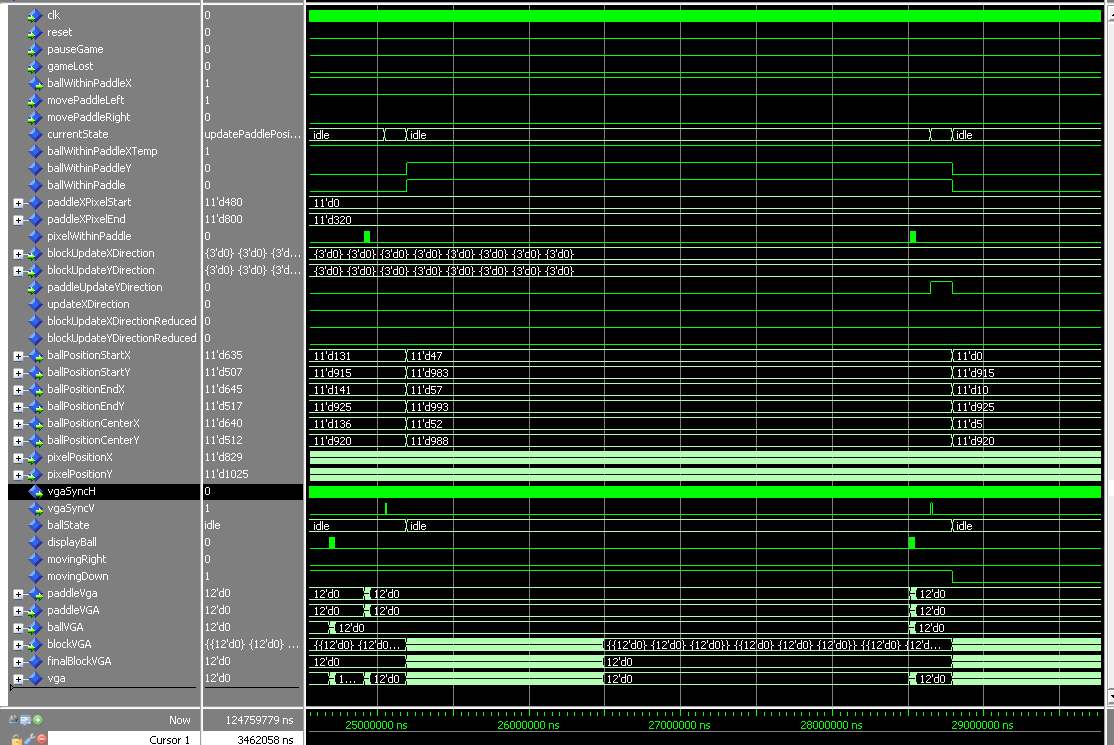
The last wrapper is the xc7a35tcpg236pkg port wrapper. This wrapper is for the actual FPGA pinout for the FPGA being used on the Basys3. This will map the actual FPGA pin names to the names assigned by Basys3 to once again allow easy tracing of pins. By doing this, this wrapper will contain every pin that needs to be specified within the constraints file. Since the pins at this level are named exactly what the FPGA manufacturer refers the pin as, the constraints file will be a simple mapping of every pin to a signal with the exact same name. This allows for the same constraints file to be used for every single project and if another board uses the same FPGA, this module can easily be leveraged as a starting point.

This figure shows the relationship between the pixel position and the horizontal and sync pulses. Due to how rapidly pixelPositionX toggles, its hard to see the exact pixels where vgaSyncH is strobing high, although you can see that it does have a period once per pixelPositionY, which is expected. There are also three pixels that strobe high for vgaSyncV, which is expected.

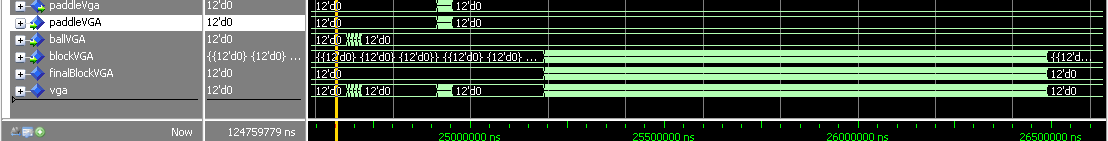


This simulation captures an entire game being played out. When the game starts, the ball advances downward to the left. The paddle is also moved to the left to contact the ball. After several frames, the ball is reported to be within the paddle, and so the ball is signaled to update its direction. Once the hit is completed, the paddle is moved to the right so that when the ball returns, it will not make contact and the game will end. After contacting the paddle, the ball then contacts the wall and advanced right and up. Eventually, the ball contacts a block and is signaled to update its direction again. The ball begins to move down and left again until it reaches the bottom of the screen and the lose condition is asserted.



This figure is a close-up look of the ball contacting the paddle. For the entire frame, the ball is within the paddle. Once the paddle advances the idle state to the blanking period, it then asserts its update direction signal for the ball to read. Before returning to idle, the balls direction is changed from moving down to up.

This figure shows the consolidation of all VGA values from the paddle, ball and blocks. As explained, only one object ever asserts their VGA value at a time and all values are ORed together to produce the final VGA value out.



# **9 Source Code and Demo**

The source code for this project can be found at <https://github.com/TreverWagenhals/TreverWagenhals/tree/master/School/FPGA%20Logic%20Design>

A demo video can be seen at   
<https://drive.google.com/open?id=1Z2Yv917-1iGQ7SAmsWvwAe9ql3Bi2h99>