Content

System Architecture

Hardware
  - Graphic Display
  - Audio Sound

Software
  - Inputs
  - Game logic
Take a Glance
Architecture

Hardware side
- vga_ball
- audio
- speaker
- screen
- Brick, Ball, Life, Score refresh

Software side
- Ball location
- Brick status
- Audio choose
- Lives
- Score refresh

Drivers
- Driver

Game Logic
- main()
- hitOnBrick()
- hitOnPaddle()
- changeSpeed()

User Input
- mouse
- keyboard

Avalon Bus
- USB
Graphic Display - Memory Budget

<table>
<thead>
<tr>
<th>Category</th>
<th>Bricks (Size, # of image, Total size)</th>
<th>Ball (Size, # of image, Total size)</th>
<th>Paddel (Size, # of image, Total size)</th>
<th>Lives (Size, # of image, Total size)</th>
<th>Number (Size, # of image, Total size)</th>
<th>Score (Size, # of image, Total size)</th>
<th>Game Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics</td>
<td><img src="image1" alt="Bricks" /></td>
<td><img src="image2" alt="Ball" /></td>
<td><img src="image3" alt="Paddel" /></td>
<td><img src="image4" alt="Lives" /></td>
<td><img src="image5" alt="Number" /></td>
<td><img src="image6" alt="Score" /></td>
<td><img src="image7" alt="Game Status" /></td>
</tr>
<tr>
<td>Size (bits)</td>
<td>64*32, 2</td>
<td>16*16, 1</td>
<td>90*20, 1</td>
<td>24*22, 1</td>
<td>20*20, 10</td>
<td>100*20, 1</td>
<td>45*45</td>
</tr>
<tr>
<td># of image</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Total size (bits)</td>
<td>98,304</td>
<td>6,144</td>
<td>43,200</td>
<td>12,672</td>
<td>96,000</td>
<td>48,000</td>
<td>48,600</td>
</tr>
</tbody>
</table>

353 Kbits used out of 4,450 Kbits of embedded memory
Graphic Display - Processing

- Use Matlab code to preprocess .png images into .mif files
- Use MegaWizard to configure single-port ROM memory blocks for every sprite
- .mif files contain 24-bit color information for each pixel, 8 bits each for R, G & B

```
1 WIDTH = 24;
2 DEPTH = 400;
3 ADDRESS_RADIX = DEC;
4 DATA_RADIX = HEX;
5 CONTENT BEGIN
6
7 0 : 000000;
8 1 : 4a4336;
9 2 : d7c7a6;
10 3 : feebc8;
11 4 : fce9c7;
12 5 : fdeac7;
13 6 : feebc8;
14 7 : feebc8;
```
Graphic Display - Architecture & Layers

Memory initialization files
brick_blue.mif
brick_red.mif
ball.mif
paddle.mif
heart.mif
score.mif
cry.mif
smile.mif
Numbers zero.mif
... nine.mif

Instatiate ROM: 1-Port Memory
brick_blue.v
brick_red.v
ball.v
paddle.v
heart.v
score.v
Numbers zero.v
... nine.v

Avalon Bus
16 bit writedata
Addr
24 bit RGB

Sprite Controller
VGA Display
VGA D-Sub
VGA_R, VGA_G, VGA_B

Lives, Score, Number (layer 1)
Ball (layer 2)

Bricks, Paddle (layer 3)
Game over, Win (layer 4)
Audio Architecture
Audio Effect Design

When ball hits on brick or wall, only in a single loop the audio_choose is set, in the next loop it goes back to 0. In our design, a loop is roughly 1.2ms, which is much shorter than the sound effects which are around 0.3s.

Solve this from hardware side:

Use a flag to mark whether the sound effect is over.

```
end else if (audio choose == 2'b01 || flag == 1'b0) begin // if hit the wall
    if (hit wall address < 11'd1815 && bg address < 17'd121593) begin
        hit wall address <= hit wall address + 1;
        bg address <= bg address + 1;
        flag <= 1'b0;
    end else if (bg address == 17'd121593) begin
        bg address <= 0;
        flag <= 1'b0;
    end else if (hit wall address == 11'd1815) begin
        hit wall address <= 0;
        flag <= 1'b1;
    end
    sample data <= (hit wall data) + (bg data);
end else if (audio choose == 2'b10 || flag2 == 1'b0) begin // if hit the brick
```
# Audio Effects Memory Budget

## Audio memory budget

<table>
<thead>
<tr>
<th></th>
<th>background music</th>
<th>hit brick</th>
<th>hit wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>time(s)</td>
<td>15.2</td>
<td>0.35</td>
<td>0.23</td>
</tr>
<tr>
<td>$f_s$(kHz)</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>memory(bit)</td>
<td>121593 * 16</td>
<td>2869 * 16</td>
<td>1815 * 16</td>
</tr>
</tbody>
</table>

**total** 2,020,432 bits
1. Audio effects includes sound effect of hitting on wall and on bricks, sampling rate is 8kHz.

2. Controlled by a audio_choose signal, sending from user space.

3. Audio effects don’t disturb the background music, just add sound effects on top of it.
USB mouse - `libusb_open_device_with_vid_pid`

```c
dev_handle = libusb_open_device_with_vid_pid(ctx, 16700, 12314); //open mouse
rr = libusb_interrupt_transfer(dev_handle, 0x81, datain, 0x0004, &size, 0);
```

The mouse will return four bytes of data.
In this project only second and the last were used.
Inputs

USB keyboard left and right

```c
if ( packet.keycode[0] == 0x50 ) /* LEFTARROW Pressed */
{
    t_speed = -0.5;
    game_start = 1;
    //printf( "LEFTARROW Pressed\n" );
}
else if ( packet.keycode[0] == 0x4f ){ /* RIGHTARROW Pressed */
    //printf( "RIGHTARROW Pressed\n" );
    t_speed = 0.5;
    game_start = 1;
}
else{
    //printf( "else\n"");
    t_speed = 0;
}
```


Core Parameters

- Ball location (x,y) (2*10 bits)
- Paddle location (10 bits)
- Brick status {brick_exists, brick_gone} (1*6*10 bits)
- Score (3*4 bits)
- Lives (2 bits)
- Game status {normal, won, lost} (2 bits)
- Audio control {normal, hit_wall, hit_brick} (2 bits)
General Game Logic

- 1. Programmable brick layout
- 2. Several Levels(difficulty) of games, 2 levels currently for faster demonstration
- 3. Press “ENTER” to start
- 4. Random initial direction, fixed absolute value of speed
- 5. 3 lives in total, shown on the top right corner
- 6. Score system: 2pts for green bricks, 1pt for blue brick
- 7. Reset after 3 lives are gone / player has passed all levels
**Bouncing Models - hit**

**Hit on brick**
Ball center falling into the blue region means the ball is hitting on the brick.

**Hit on paddle**
Similarly to determine whether the ball is hitting on the brick or not.
Bouncing Models - bounce

Hit on brick
- 1. Top, bottom \( v'_x = v_x, v'_y = -v_y \)
- 2. Right, left \( v'_x = -v_x, v'_y = v_y \)
- 3. Hit on corners, consider the corner as a circle with \( r = 0 \). The radial speed \( v_r = -v_r \), lateral speed \( v_l = v_l \).
- Do some maths, it gives

\[
\begin{align*}
v'_x &= -\cos(2\theta) \cdot v_x - \sin(2\theta) \cdot v_y \\
v'_y &= -\sin(2\theta) \cdot v_x + \cos(2\theta) \cdot v_y \\
\theta &= \tan^{-1} \frac{y_0 - y_{ball}}{x_0 - x_{ball}}
\end{align*}
\]
Thank you!