Star Wars

CSEE 4840 Embedded Systems Design

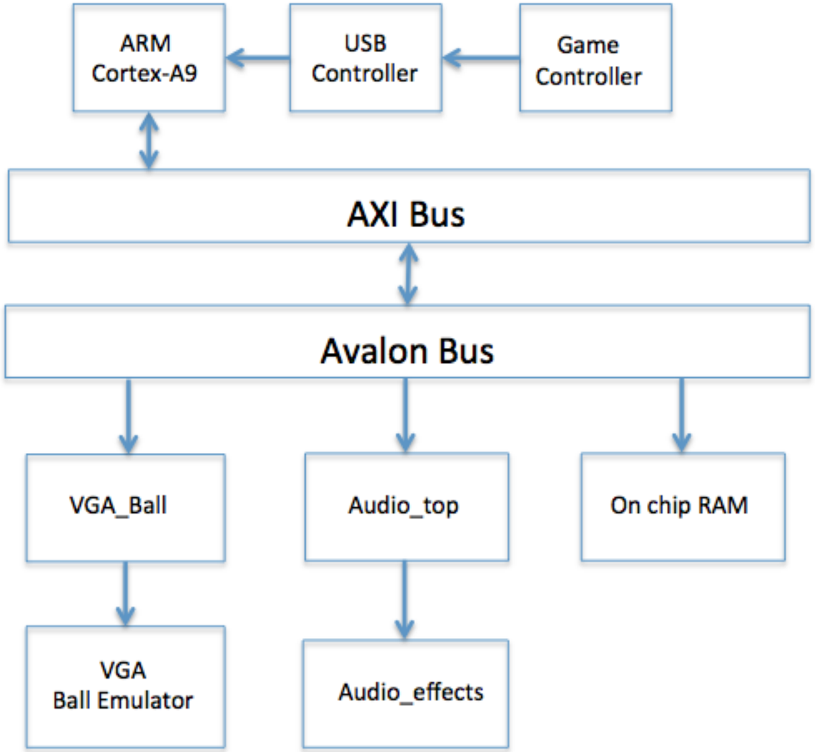
Fang Fang(ff2317)
Jiaxuan Shang(js4361)
Xiao Xiao(xx2180)
Zhenyu Zhu(zz2281)

Columbia University Spring 2014

Overview of the project

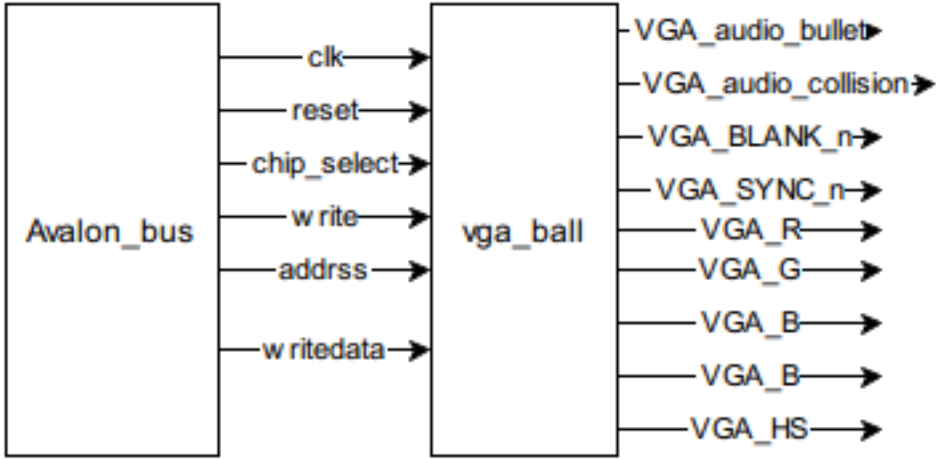
- Inspired by the classic game *Geometry Wars*
  - Various enemy flying round or chasing after the spaceship.
  - Player’s goal is to survive as long as possible and get a score as high as possible with 3 lives.
  - Bomb available to destroy all the enemies at once
- Overall 60 entities, first entity saved for spaceship, 2nd to 30th for bullets, and last 30 for enemies.
  - ID number indicating entity type
  - X, Y coordinates and direction information also contained in each unit data

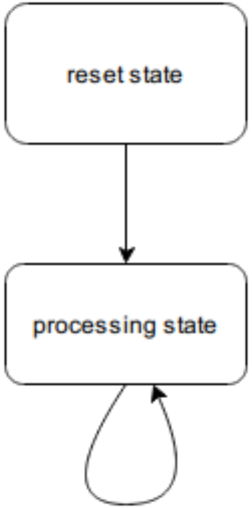
Architecture



VGA_BALL

- Module: VGA_BALL
- submodule: VGA_BALL_Emulator



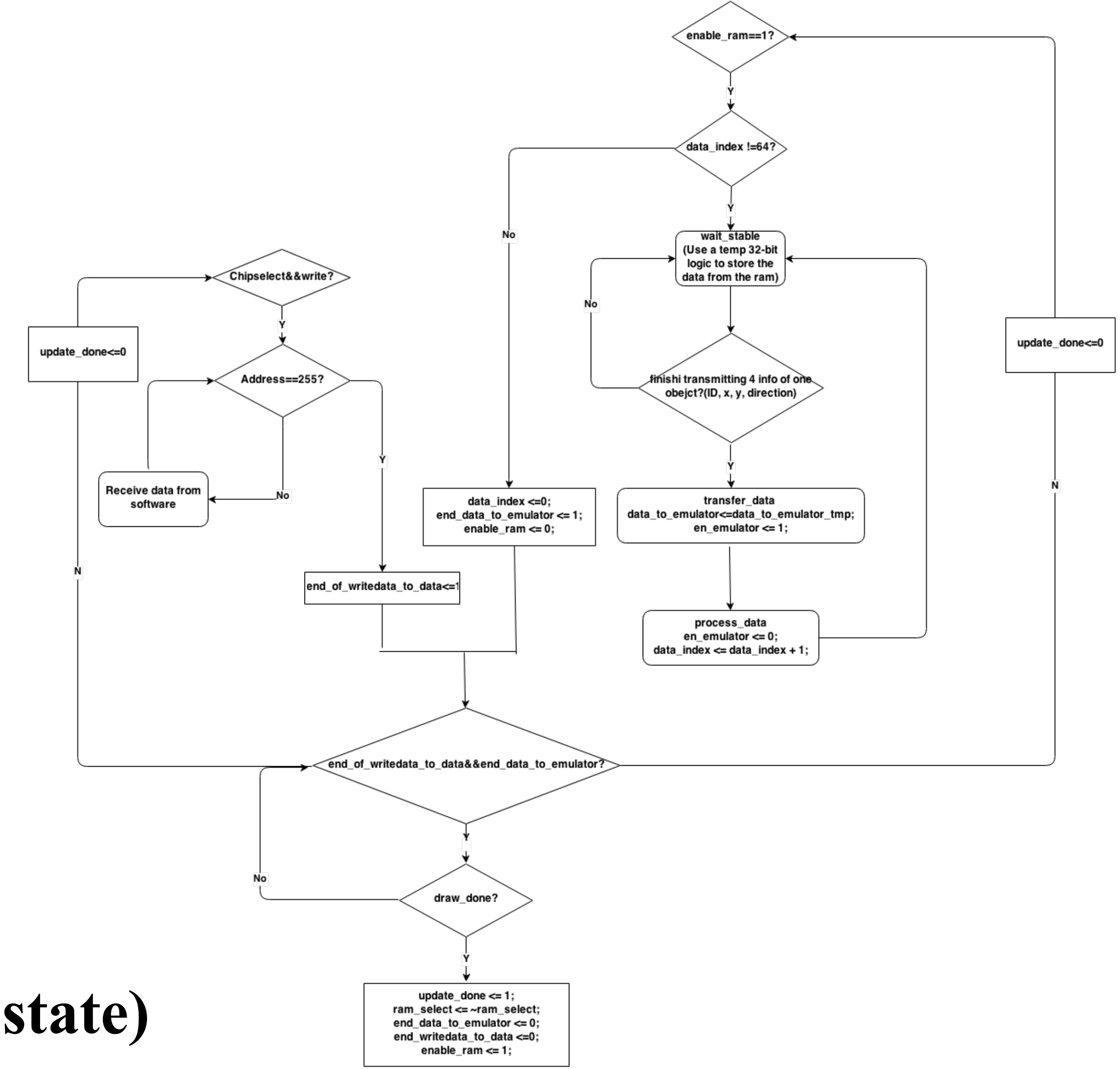


VGA_BALL

- Receives 10-bit writedata in a total of 256 (2-reg structure)
- Combines every four of them to form the information for every object (in a total of 64):

  [31:0] logic data_to_emulator: [id, x, y, direction]

- Connects to the submodule VGA_BALL_Emulator to draw the graph



enable_ram==1?

Y

data_index !=64?

Y

No

wait_stable
(Use a temp 32-bit
logic to store the
data from the ram)

No

finishi transmitting 4 info of one
obejct?(ID, x, y, direction)

Y

transfer_data
data_to_emulator<=data_to_emulator_tmp;
en_emulator <= 1;

process_data
en_emulator <= 0;
data_index <= data_index + 1;

update_done<=0

N

Chipselect&&write?

Y

update_done<=0

Address==255?

No

Y

Receive data from
software

N

data_index <=0;
end_data_to_emulator <= 1;
enable_ram <= 0;

end_of_writedata_to_data<=1

end_of_writedata_to_data&&end_data_to_emulator?

Y

No

draw_done?

Y

update_done <= 1;
ram_select <= ~ram_select;
end_data_to_emulator <= 0;
end_writedata_to_data <=0;
enable_ram <= 1;

Flow Chart (Processing state)

VGA_BALL

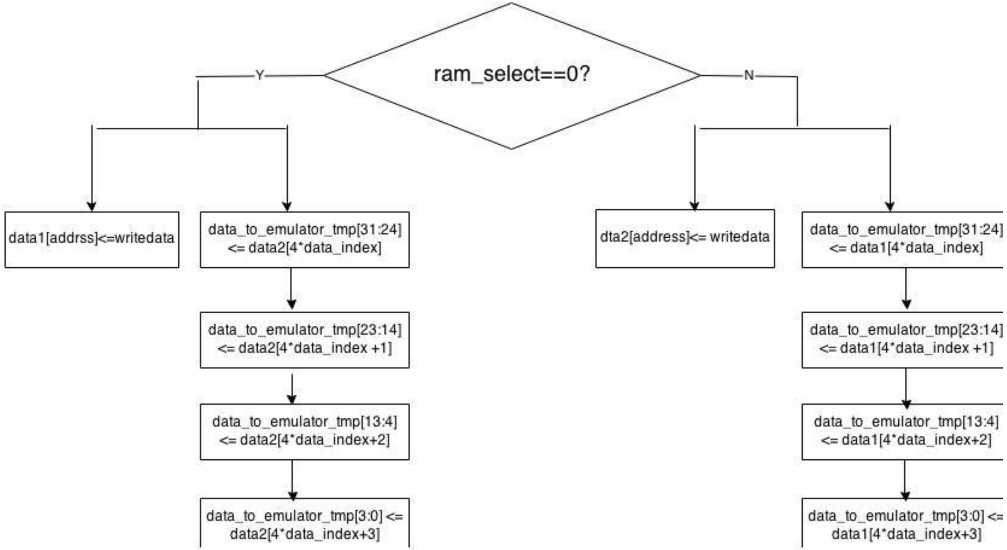
- Receives data from the software:

2-reg structure

One for transmission

One for updating

```
reg [9:0] data1 [0:255];
reg [9:0] data2 [0:255];
```



ram_select==0?

Y

data1[addrss]<=writedata

data_to_emulator_tmp[31:24] <= data2[4*data_index]

data_to_emulator_tmp[23:14] <= data2[4*data_index +1]

data_to_emulator_tmp[13:4] <= data2[4*data_index+2]

data_to_emulator_tmp[3:0] <= data2[4*data_index+3]

N

dta2[address]<= writedata

data_to_emulator_tmp[31:24] <= data1[4*data_index]

data_to_emulator_tmp[23:14] <= data1[4*data_index +1]

data_to_emulator_tmp[13:4] <= data1[4*data_index+2]

data_to_emulator_tmp[3:0] <= data1[4*data_index+3]

VGA_BALL_Emulator

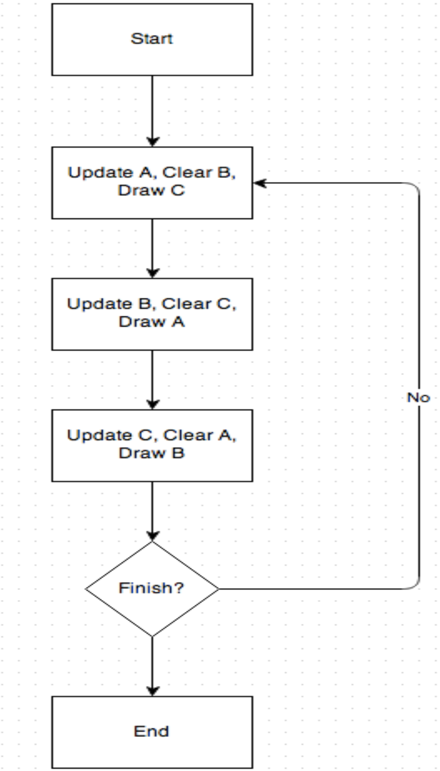
- Receives 32-bit object information and stores into 2 RAMs: One for updating, one for transmission.

- Stores the RGB value of every object into the line buffer (3 RAMs: one for updating, one for drawing, one for cleaning) according to the object information.

- Read the rom and draw the objects according to the RGB value

VGA_BALL_Emulator

Flow chart of Line Buffers


Start
Update A, Clear B, Draw C
Update B, Clear C, Draw A
Update C, Clear A, Draw B
Finish?
No
End

Audio Implementation

- I2C protocol: data is sent a bit at a time over the SDAT wire, with the separation between bits determined by clock cycles on the SCLK wire.
- I2C is a master-slave protocol. In our project, the FPGA is the master and the audio codec is the slave.

- Audio components:
  - I2C controller: control the transmission timing, configuration interface.
  - Configuration controller: determines what data to send--16-bit words. Use 19 9-bit regs to record configurations, the first 7 bits are the reg address and the last 9 bits are the register contents.

Reference: Exploring the Arrow SoCKit Part - The Audio Codec

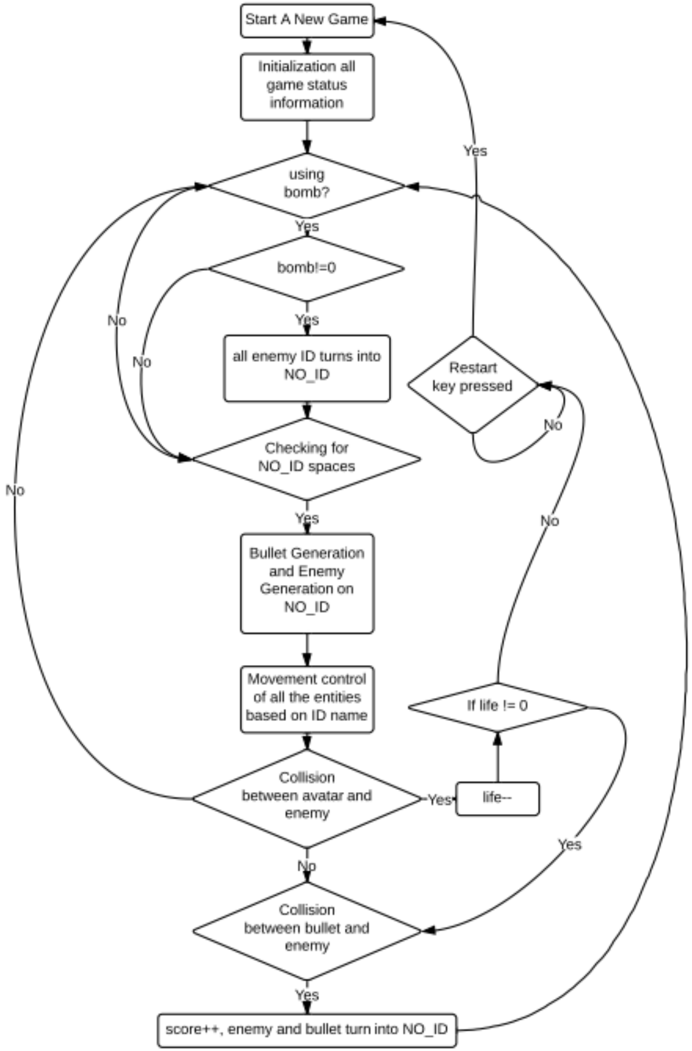
Audio Implementation contd.

- Audio components (contd.):
  - Clocks: use Cyclone V’s Phase-Locked Loops to generate master clock for audio codec. Other bit clock and LRC are generated using frequency divider.
  - Audio codec driver: the data is pushed out or read in through shift registers.
- Audio output:
  - Receive flag information from software. Control production of sound.
  - The .wav file is converted into .mif and the data is stored in ROMs.

Reference: Exploring the Arrow SoCKit Part - The Audio Codec

Software and algorithms

- Overall game logic control
  - bomb detection
  - bullet generation
  - enemy generation
  - collision detection
  - units movement control
  - score, life, bomb data collection
- Sending array messages of 256 elements to hardware containing information of 60 entities and player data information (scores, lifes, bombs, etc.)


Start A New Game
Initialization all game status information
using bomb?
Yes
bomb!=0
Yes
all enemy ID turns into NO_ID
No
No
Checking for NO_ID spaces
Yes
Bullet Generation and Enemy Generation on NO_ID
Movement control of all the entities based on ID name
Collision between avatar and enemy
No
Yes
life--
If life != 0
No
Restart key pressed
No
Yes
Yes
No
Collision between bullet and enemy
Yes
score++, enemy and bullet turn into NO_ID

Flow Chart (Software)

Experiences and Issues

- Game logic moved from hardware to software.
- Improved logic usage (34% to 17%) on the board.
- Better VGA display using sprite scheme.
- Treat the reg/ram as memory and ensure only to read/write one value from/into the memory at one clock cycle.
- After writing into the memory, the data could only be read out two cycles later. Thus the state for stabilize the data is needed.

Experiences and Issues

- Overlap:

Solution 1: Change the C code to avoid overlap (not good)

Solution 2: Use the line buffers to store the 32-bit information about the objects for each pixel (cannot solve this problem)

Solution 3: Use the line buffers to store the RGB value (currently use)