Save Edwards

Wei Wei 2313
Zhe Cao zc2237
Zeyang Yang zy2171
Ge Zhao gz2196

Overview

- Tower defense game on DE2 board
- Motivated by iPhone game *the Creeps*
- Storyline: Edwards is sleeping. Monsters are getting close to him wave after wave. We should build towers along the path to kill them and save Edwards.
- Strategies are needed to win
  - Kill monsters to earn money
  - Build proper towers on proper position on the map
  - Upgrade or sell towers when needed

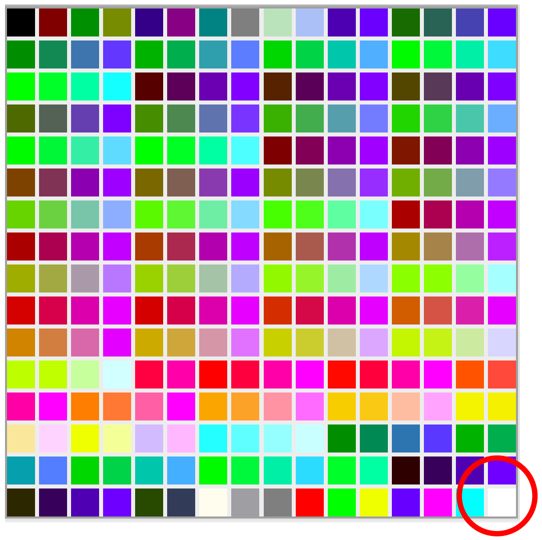
Memory

- SDRAM: CPU memory

- Data storage:
  - SRAM: store images
  - On chip RAM: store sound

Image preprocessing

- Object: generate bitmap for DE2 board
- Memory requirement: within 512 KB SRAM
- 8-bit index color, i.e. 256 colors

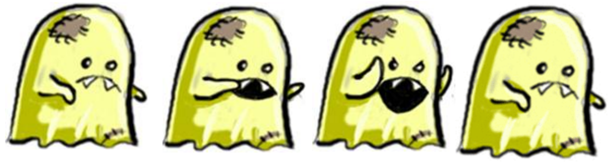


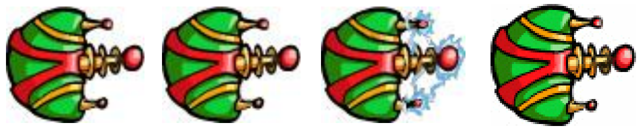
Define 0xFF
"Transparent"

Color Look Up Table (CLUT)

Image preprocessing (cont.)

- Image categories and amount
  - Background (640 x 480): 3
  - Monsters (32 x 32 or 32 x 40): 33
  - Towers (32 x 32): 159
  - Buttons (various sizes): 16
  - Numbers (8 x 16): 10
- Total image size
  - SDRAM Occupation: 1112KB
  - SRAM Occupation: 490KB

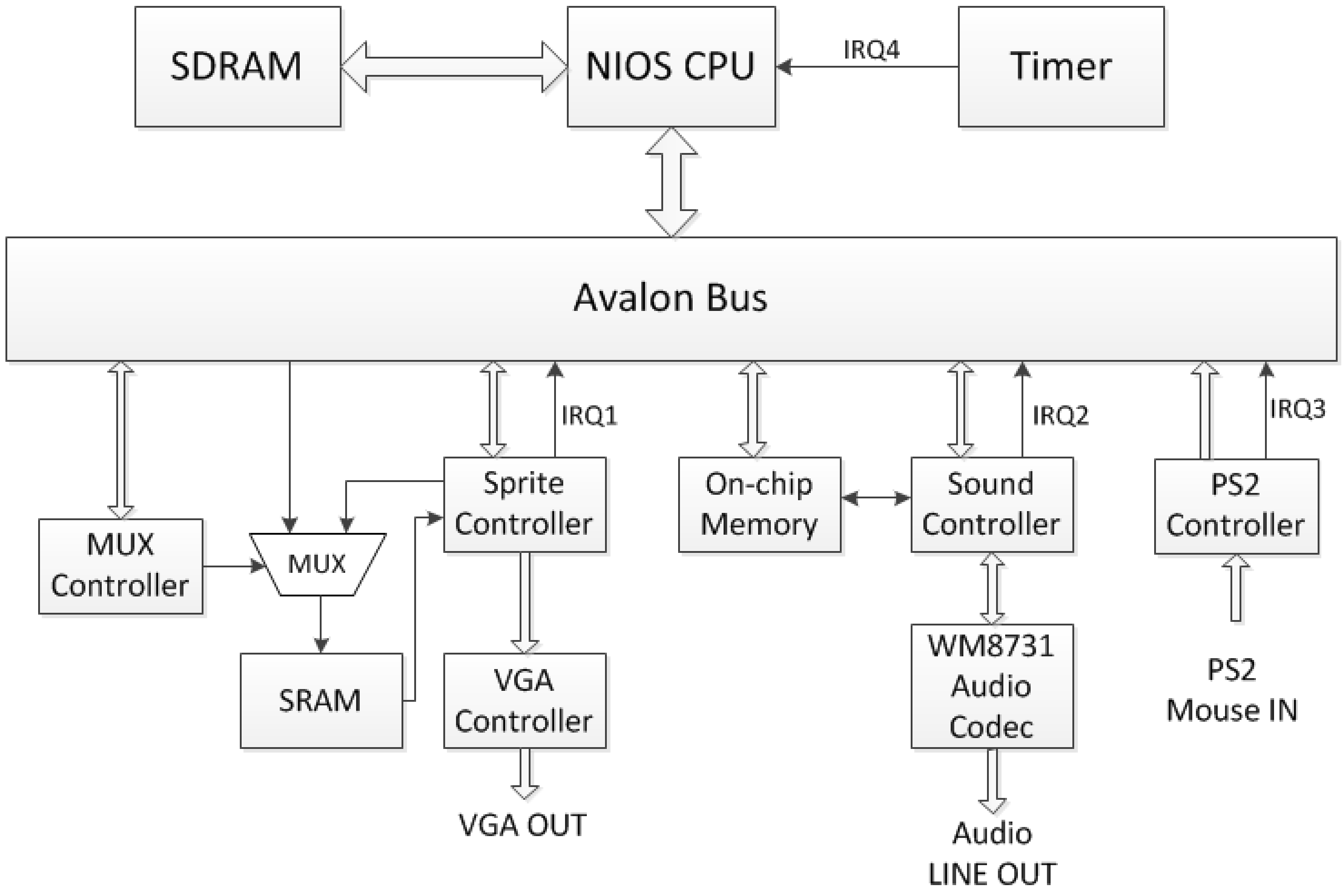




Audio preprocessing

- Sampling rate: 8KHz
- Quantization bits: 16 bits
- On chip RAM Occupation: 43KB
- Overlap algorithm considerations:
  - In real world: overlap without limits
  - In digital world: may overflow(summing) or lose quality (averaging)
  - In SaveEdwards: sum up all concurrent audios
    - carefully adjusted magnitude of audio (average below 10% maximum amplitude)
    - Realistic, overflow avoided, simple implementation, good quality using 16 bit quantization

Hardware configurations



Sprite control

- Why we choose sprite and design a specific sprite controller?
  - We have tens of objects who have their individual characteristics and motions
  - Directly code in VGA module will exponentially increase our pain when the number of the sprite increases
  - After developing this hardware platform, it is easy to add or delete sprites and to control by the software

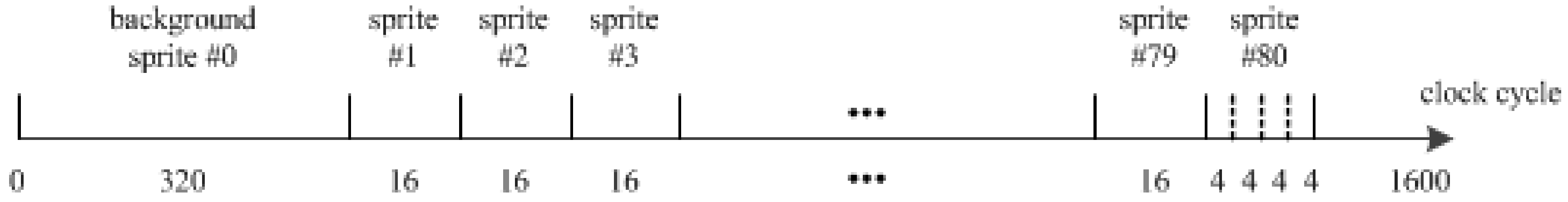
Sprite control (cont.)

- List of sprites

Sprite category	Amount
Mouse pointer	1
Selects	2
Buttons	9
Monsters	13
Glue effects	13
Health bar	13
Towers	13
Bullets	13
Numbers	11
TOTAL	88

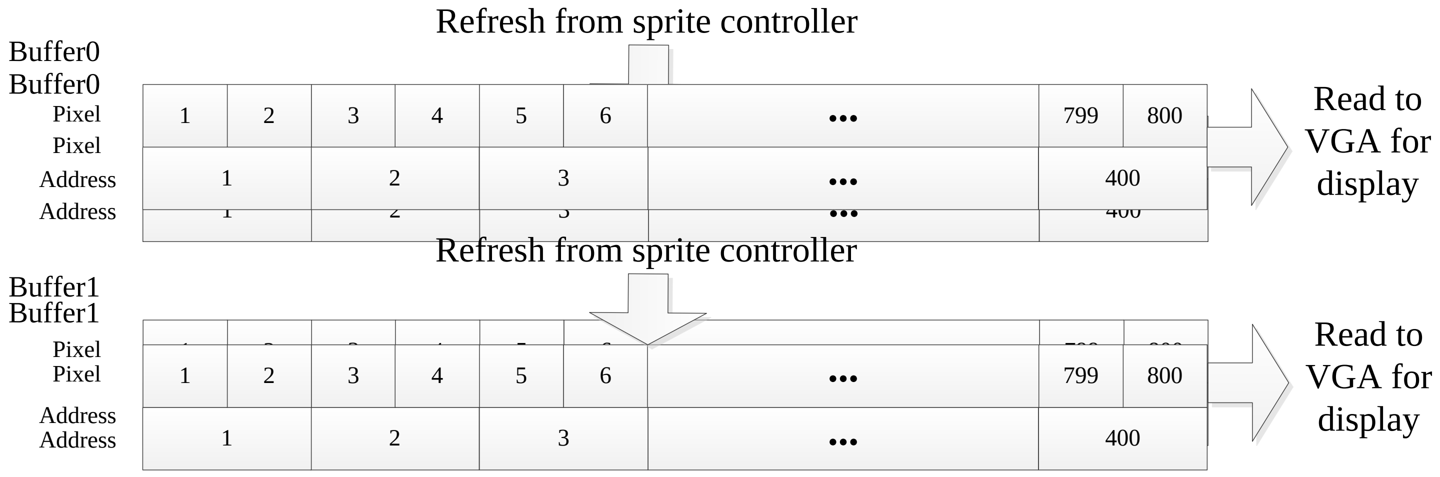
Sprite control (cont.)

- Computation in sprite controller and display on VGA “ at the same time”
  - 800 clock cycles to display one row under 25MHz frequency in VGA module
  - 1600 clock cycles to compute one row pixel data under 50MHz frequency in sprite controller



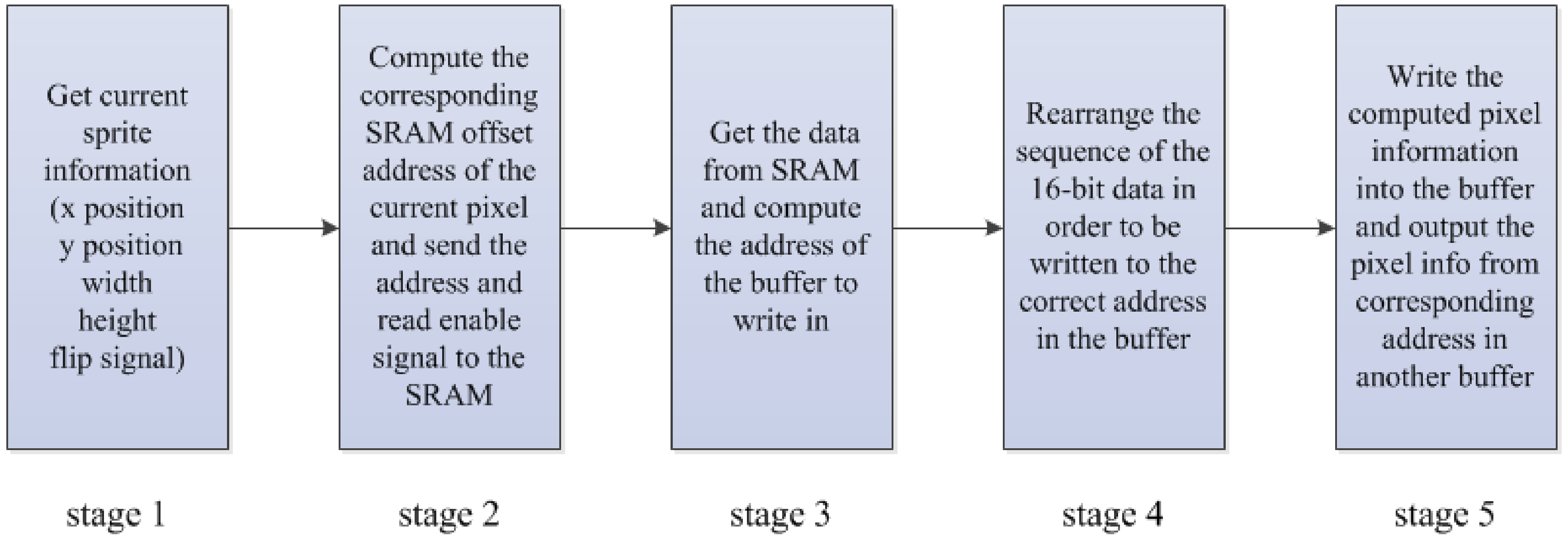
Sprite control (cont.)

- Two sets of buffers
  - Two sets of buffers and alternatively write to and read from them
  - Refreshing the pixel information of current row and displaying the pixel of previous row
  - No data contention



Sprite control (cont.)

- 5-stage pipeline


Get current sprite information (x position y position width height flip signal)
Compute the corresponding SRAM offset address of the current pixel and send the address and read enable signal to the SRAM
Get the data from SRAM and compute the address of the buffer to write in
Rearrange the sequence of the 16-bit data in order to be written to the correct address in the buffer
Write the computed pixel information into the buffer and output the pixel info from corresponding address in another buffer
stage 1
stage 2
stage 3
stage 4
stage 5

Audio control

- Why we design a specific audio controller?
  - 13 monsters and 13 towers can sound together, simply throwing all the sound data into codec will definitely mess all the things up, real time challenge!
  - Algorithm: add all sounds up
  - Similar to sprite control: each piece of sound is like one “sprite”

Audio control (cont.)

- Different from sprite controller

Sprite Control	Audio Control
Put the new data into the buffer and replace the old one -- Pipeline	Fetch the old data from the buffer and do the operation, then put the result back to the same buffer -- FSM
Software enables and disables	Software enables and hard ware disables

- Safest way to control the enable signal – software enables and hardware disables, easy to fit the time requirement
- Enable signal in software flips, the hardware considers it as an enable signal, making the communication problem easier

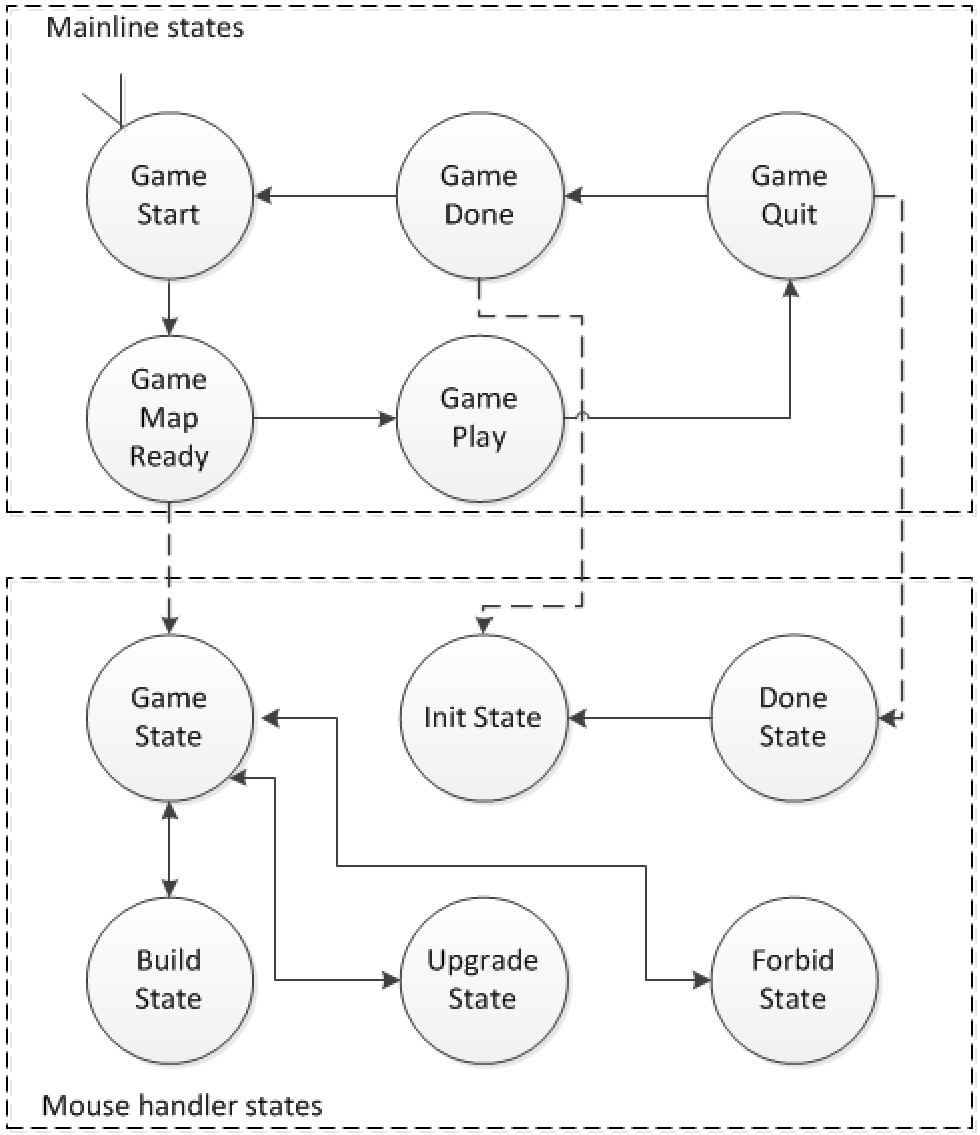
Audio control (cont.)

- FSM of the audio control
  - 28 states, first state is initial state, last state is hold state, the rest 26 states correspond to 13 towers and 13 monsters
  - Go to the next state and merge the next object's sound every time the clock cycle counter gets to 256
  - Cumulative offset pointer to indicate the offset of current data
- Audio length is controllable according to game speed

PS2 Control

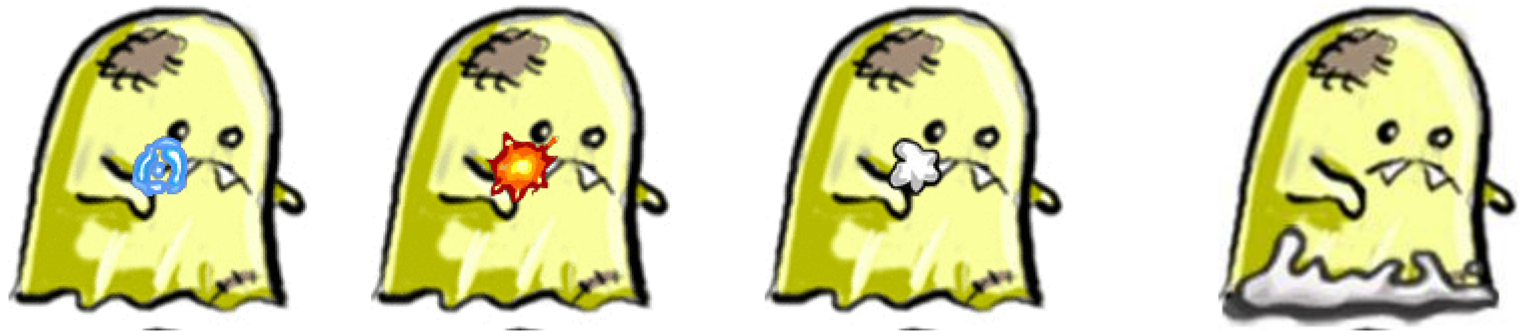
- PS2 mouse as input device
- Return status of left button, right button and middle button
- Return X, Y coordinates movement
- IRQ asserted when button clicked or released
- X & Y positions captured with frame display

Software state diagram



Gaming effects

- Monsters on the map, monsters on monsters
- Explosive effect on monsters
- Slow down effect on monsters
- Health bar above monsters



Interactive operations

- Map selection – select a map to start
- Dynamically display money, score, wave number, lives
- Play/pause buttons – play or pause
- Speed button – Normal speed and double speed
- Mute and unmute buttons
- Click on blank ground – towers to build if money is enough
- Click on obstacles, or path – forbidden sign
- Click on towers – show attack range, sell and upgrade buttons

Experiences and issues

- Incoherent in display
  - Read and write data conflict
  - Inappropriate IRQ of mouse events
- Limited sprite amount
  - The pixels of the new row to display must be computed out while displaying the previous row. The amount depends on the displaying frame rate and board clock frequency

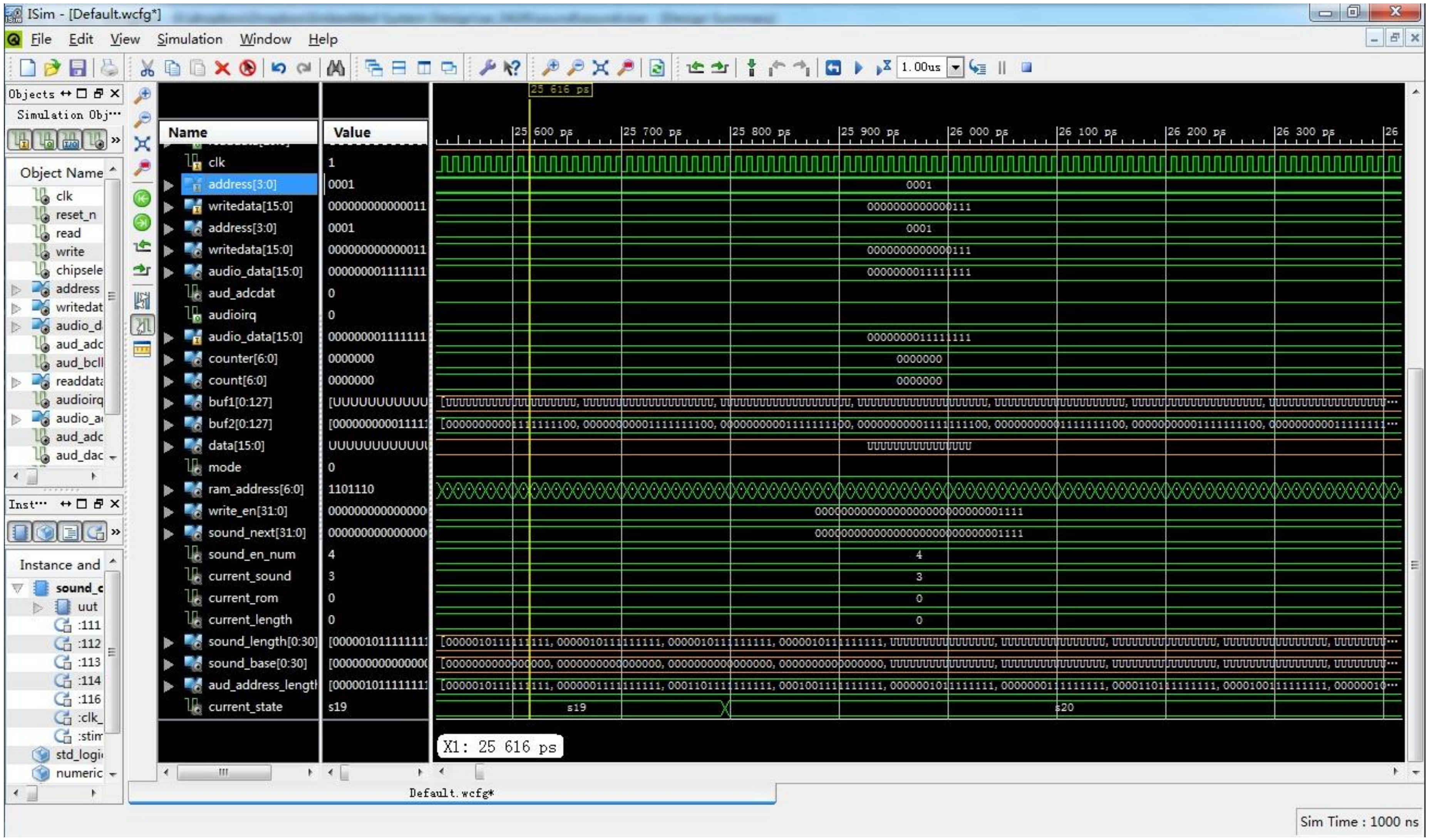
Lessons learned

- Architecture of hardware and software on FPGA board
- Resource allocation – use hardware controllers to share computation
- Scheduling optimization – remove slacked operations away from critical path

Lessons learned (cont.)

- Debug methods:
  - Write testbenches to simulate hardware entities. Use waveforms to debug.
  - Use LEDs and LCD screen on DE2 board to indicate certain wires in tested entities
  - Use console in NIOS II to debug software

Lessons learned (cont.)



```
Name                 Value
clk                  1
address[3:0]         0001
writedata[15:0]      000000000000011
address[3:0]         0001
writedata[15:0]      000000000000011
audio_data[15:0]     000000001111111
aud_adcdat           0
audioirq             0
audio_data[15:0]     000000001111111
counter[6:0]         0000000
count[6:0]           0000000
buf1[0:127]          [UUUUUUUUUUU
buf2[0:127]          [000000000011111
data[15:0]           UUUUUUUUUUUU
mode                 0
ram_address[6:0]     1101110
write_en[31:0]       000000000000000
sound_next[31:0]     000000000000000
sound_en_num         4
current_sound        3
current_rom          0
current_length       0
sound_length[0:30]   [000001011111111
sound_base[0:30]     [000000000000000
aud_address_lengt    [000001011111111
current_state        s19

X1: 25 616 ps
Sim Time : 1000 ns
```

Demos