Bubble Bobble

By Hongzheng Zhu, Qingyuan Liu, Ke Liu, Lance Chou
Overall structure:
VGA control: vga top module
VGA control: tile drawing

Tile array RAM
Dual port:
Port widths: 8bit, 8bit
Size: 9600 bit
Latency: 1 cycle

calculate tile array address:
\[ \text{actual_vcount}(10b) \times 10^2 + \text{col}(8b) \]

calculate tile image address:
\[ \text{img}_{\text{ram}} \times 16 + \text{actual_vcount \% 16} \]

tile_ram_addr (12b)

data_tile_draw (256b)

Loop logic
Loop col from 0 to 39
When finished,
set finish=1, wren_tile_draw=0

Tile image ROM
Single port
Port width: 256bit
Word count: 756
Size: 491,520 bit
Latency: 1 cycle

wren (1b)
addr_write (11b)
data_write (8b)
tile_array_addr_read (11b)

img_\text{ram} (8b)

actual_vcount (10b)
VGA control: sprite drawing

Sprite array registers
- 300 bit

**Sprite active**
For each sprite, compute $0 <= vcount - sprite_array \text{ (row)} < 32$
If yes, output
the row is to be drawn
the column where the sprite start
the img number of the sprite
Alternatively, when write == 1: use data in writedata to replace the data in the register sprite_reg_num is pointing to
Latency: 1 cycle (per sprite)

**Sprite_draw**
Once receive sprite_draw_start, compute sprite_rom_address = img_num * 16 + row_in_sprite
query the ROM for the data of each pixel on the row, and output them one by one.
Latency: 33 cycle

Control mechanism:
Loop through each sprite register, if a sprite is active from sprite_active, send signal to sprite_draw, wait until sprite_draw_finish == 1, and then process the next sprite
// def of argument for tiles
typedef struct {
    unsigned char r;
    unsigned char c;
    unsigned char n;
} vga_top_arg_t;

// def of argument for sprites
typedef struct {
    unsigned char active;
    unsigned short r;
    unsigned short c;
    unsigned char n;
    unsigned short register_n;
} vga_top_arg_s;

r 5bit  |  c 6bit  |  n 8bit  |
Total 19 bit

active 1bit | r 9bit | c 10bit | n 5bit
Total 25 bit
Transparent
Audio control
Audio control

- Damage Enemy
- Loses a Life
- Victory!

Interruption

BGM
Audio HW/SW interface

typedef struct {
    unsigned char play;
} fpga_audio_arg_t;

#define FPGA_AUDIO_BGM_STARTSTOP _IOW(FPGA_AUDIO_MAGIC, 1, fpga_audio_arg_t *)
#define FPGA_AUDIO_SET_AUDIO_ADDR _IOW(FPGA_AUDIO_MAGIC, 2, fpga_audio_arg_t *)
Controller

The controller communicates with a 8 bytes protocol via USB, mapped as below:

<table>
<thead>
<tr>
<th>Constant</th>
<th>Constant</th>
<th>Constant</th>
<th>Left/right arrow</th>
<th>Up/down arrow</th>
<th>X/Y/A/B</th>
<th>Rib/Select/Start</th>
<th>Constant</th>
</tr>
</thead>
</table>

These keys are mapped to specific interactions in the game:

- Left arrow: move left
- Right arrow: move right
- A: shoot bubble
- B: jump

```c
struct controller_output_packet {
    short updown;
    short leftright;
    uint8_t select;
    uint8_t start;
    uint8_t left_rib;
    uint8_t right_rib;
    uint8_t x;
    uint8_t y;
    uint8_t a;
    uint8_t b;
}
```
Game logic

8 levels with different maps.

Enemy generation and movement.

Attack.

Collision detection: Wall, floor, bubbles, enemy, character, reward.

Requirements to move to the next level.

Winning and defeat condition.
Demonstration