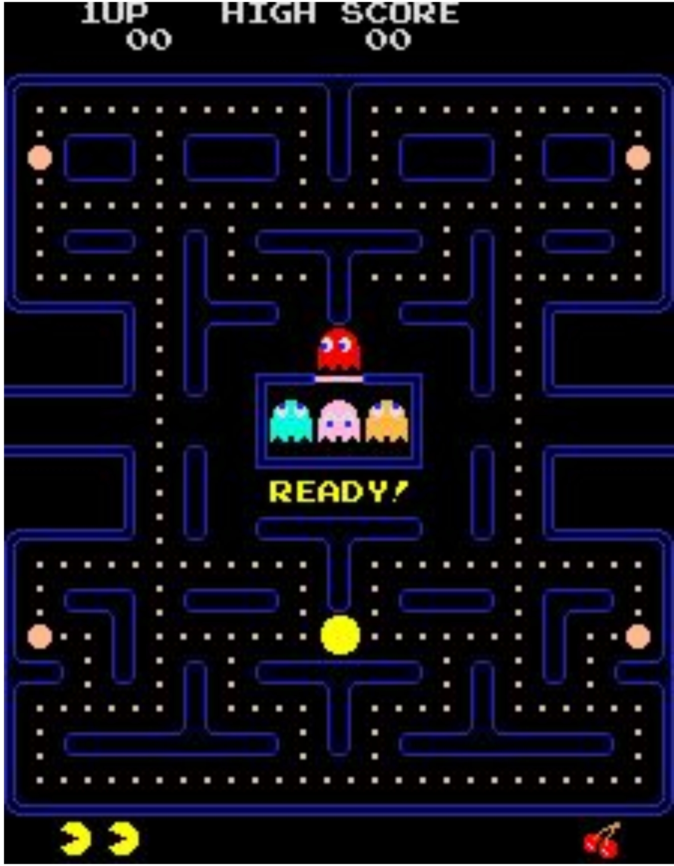
Pac-Man

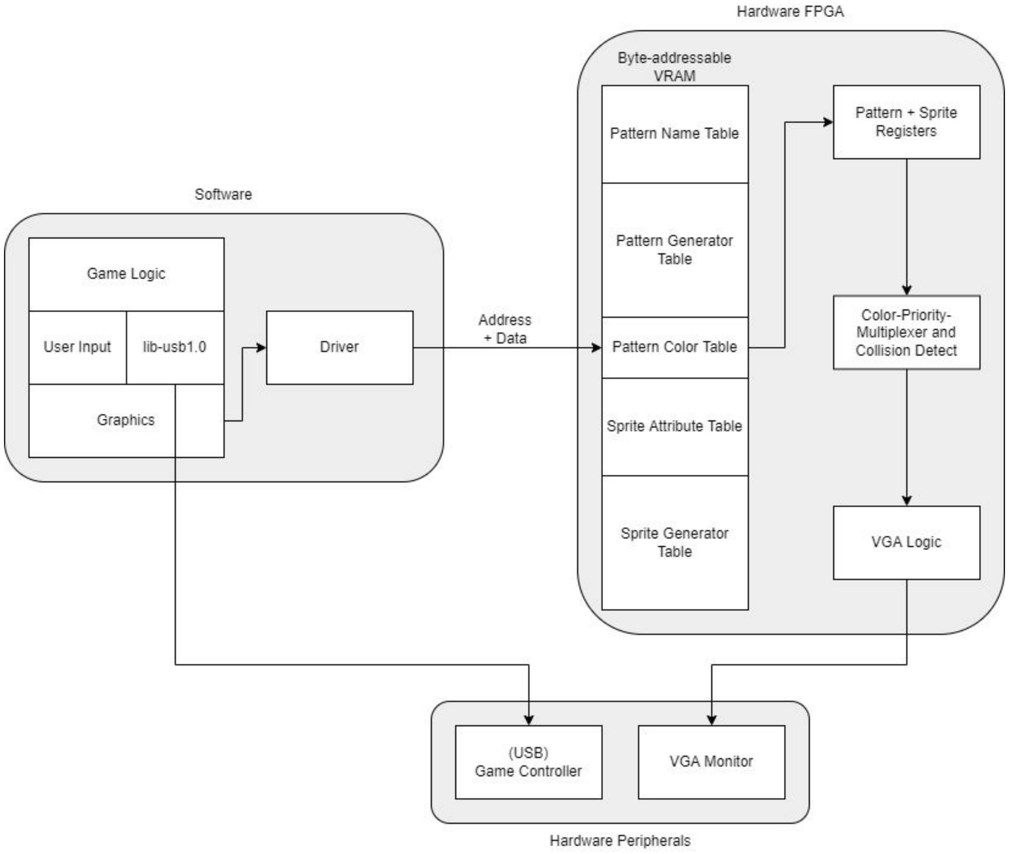
Leo Qiao, Jerry Lin

Background

- Popular game developed in the 1980s by Toru Iwatani
- Maze-based game: Eat food pellets for points and avoid contact with four ghosts
- Goal: Implement single level clone of Pac-Man



System Architecture

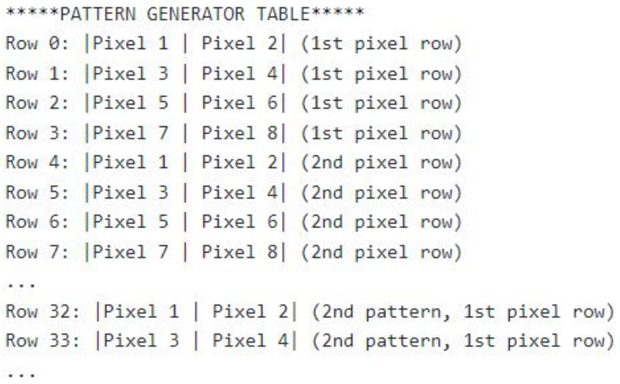


Hardware: Overview

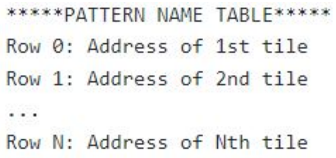
- Sprite and tile graphics based off of the TMS9918 graphics processor
- Generalizable design: Support for arbitrary graphics/games depending on SW
- All tables byte addressable: 1 byte per row
- 4 bit color code with color LUT
- 8 x 8 pixel patterns, 16 x 16 pixel sprites
- Top-level priority multiplexer handles collision

Hardware: Tiles

- Tiles used for maze wall, food pellets, and text
- Pattern Generator Table stores unique patterns
  - 32 rows/pattern, 2048 rows for 64 patterns
- Pattern Name Table stores base addresses of patterns for each tile
  - 5 LSBs of address dropped to fit in 1 byte
  - 4096 rows for 64 x 64 tiles



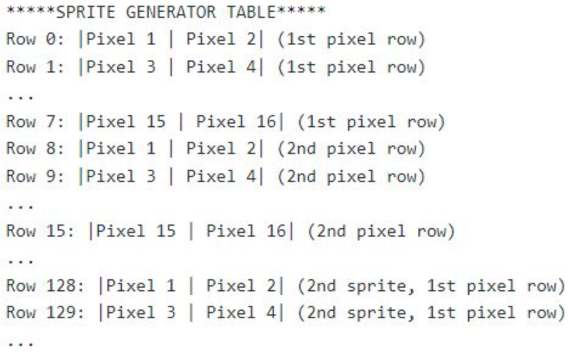
```
*****PATTERN GENERATOR TABLE*****
Row 0: |Pixel 1 | Pixel 2| (1st pixel row)
Row 1: |Pixel 3 | Pixel 4| (1st pixel row)
Row 2: |Pixel 5 | Pixel 6| (1st pixel row)
Row 3: |Pixel 7 | Pixel 8| (1st pixel row)
Row 4: |Pixel 1 | Pixel 2| (2nd pixel row)
Row 5: |Pixel 3 | Pixel 4| (2nd pixel row)
Row 6: |Pixel 5 | Pixel 6| (2nd pixel row)
Row 7: |Pixel 7 | Pixel 8| (2nd pixel row)
...
Row 32: |Pixel 1 | Pixel 2| (2nd pattern, 1st pixel row)
Row 33: |Pixel 3 | Pixel 4| (2nd pattern, 1st pixel row)
...
```



```
*****PATTERN NAME TABLE*****
Row 0: Address of 1st tile
Row 1: Address of 2nd tile
...
Row N: Address of Nth tile
```

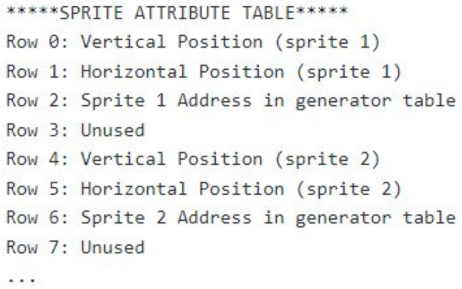
Hardware: Sprites

- Sprites used for Pac-Man and ghosts
- Sprite Generator Table stores unique sprites
    - 128 rows/sprite, 2048 rows total for 16 sprites



```
*****SPRITE GENERATOR TABLE*****
Row 0: |Pixel 1 | Pixel 2| (1st pixel row)
Row 1: |Pixel 3 | Pixel 4| (1st pixel row)
...
Row 7: |Pixel 15 | Pixel 16| (1st pixel row)
Row 8: |Pixel 1 | Pixel 2| (2nd pixel row)
Row 9: |Pixel 3 | Pixel 4| (2nd pixel row)
...
Row 15: |Pixel 15 | Pixel 16| (2nd pixel row)
...
Row 128: |Pixel 1 | Pixel 2| (2nd sprite, 1st pixel row)
Row 129: |Pixel 3 | Pixel 4| (2nd sprite, 1st pixel row)
...
```

- Sprite Attribute Table stores addresses and location of each sprite to be displayed
    - 1 byte vertical position
    - 1 byte horizontal position
    - 1 byte sprite base address
    - 32 rows total for 8 simultaneous sprites



```
*****SPRITE ATTRIBUTE TABLE*****
Row 0: Vertical Position (sprite 1)
Row 1: Horizontal Position (sprite 1)
Row 2: Sprite 1 Address in generator table
Row 3: Unused
Row 4: Vertical Position (sprite 2)
Row 5: Horizontal Position (sprite 2)
Row 6: Sprite 2 Address in generator table
Row 7: Unused
...
```

Hardware: Display

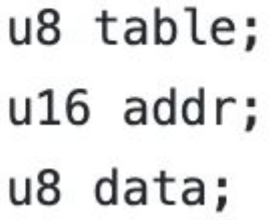
- Pattern and sprite processing occurs during VGA horizontal sync
- Pattern has one FSM, each sprite has their own FSM
  - Sprite memory accesses are non-overlapping
- Sprite pixel rows loaded into shift register: use horizontal position as down counter
- Entire pattern row loaded into shift register
- Shift register output fed into color LUT to obtain 24 bit RGB value
- Sprites have priority over patterns

HW/SW Interface

- 32 bit data packet from software:
    - Bits 0-1: Selects one of four tables
    - Bits 2-17: Address in selected table to write to
    - Bits 24-31: Data to write to table at specified address

Driver: Kernel Module

- Transform from struct to 32-bit HW command
- 3-field struct:



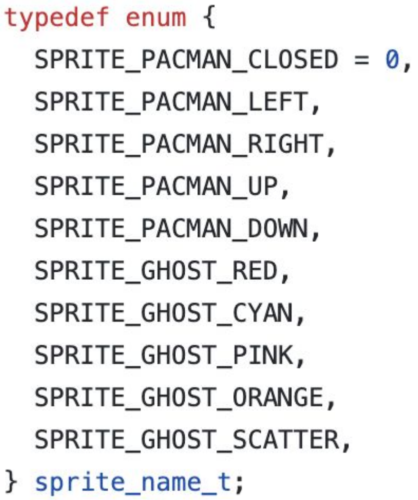
```
u8 table;
u16 addr;
u8 data;
```

Driver: User Space

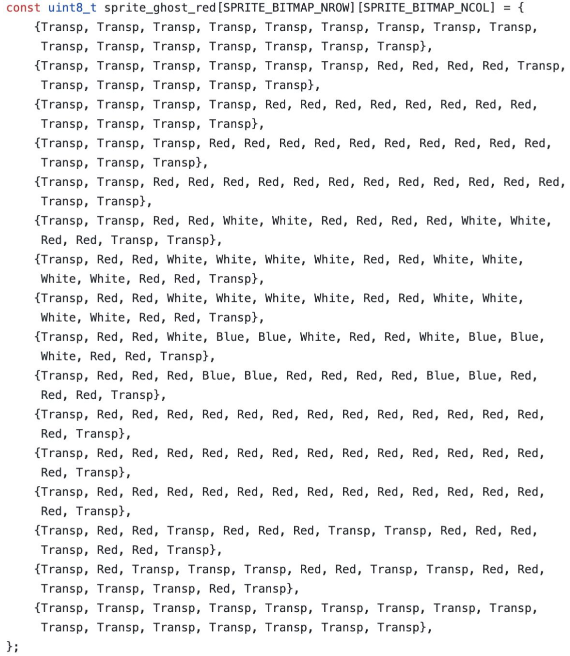
- Helper functions:
    - void set_sprite_bitmap(int i, const uint8_t *pat)
    - void set_sprite(sprite_attr_t attr)
    - void set_pattern_bitmap(int pati, const uint8_t *pat)
    - void set_pattern_at(uint8_t r, uint8_t c, uint8_t name)

Driver: Drawing Patterns & Sprites

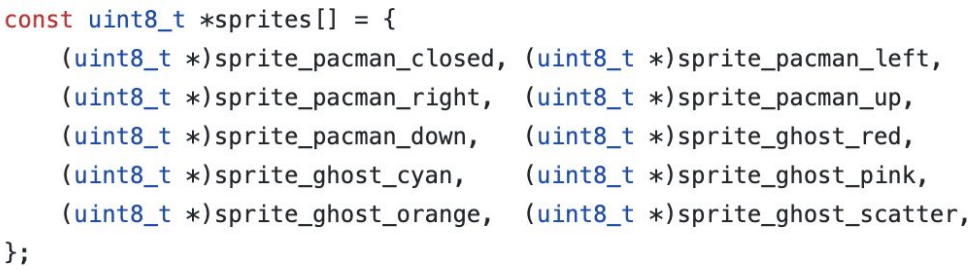
- Color macros
- Draw bitmaps with 2d-arrays
- Load as a list
- Enums -> entry number in the generator table



```c
typedef enum {
  SPRITE_PACMAN_CLOSED = 0,
  SPRITE_PACMAN_LEFT,
  SPRITE_PACMAN_RIGHT,
  SPRITE_PACMAN_UP,
  SPRITE_PACMAN_DOWN,
  SPRITE_GHOST_RED,
  SPRITE_GHOST_CYAN,
  SPRITE_GHOST_PINK,
  SPRITE_GHOST_ORANGE,
  SPRITE_GHOST_SCATTER,
} sprite_name_t;
```



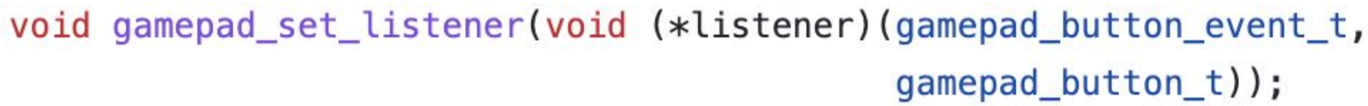
```c
const uint8_t sprite_ghost_red[SPRITE_BITMAP_NROW][SPRITE_BITMAP_NCOL] = {
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Red, Red, Red, Red, Transp,
     Transp, Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red,
     Transp, Transp, Transp, Transp},
    {Transp, Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Transp, Transp, Transp},
    {Transp, Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Transp, Transp},
    {Transp, Transp, Red, Red, White, White, Red, Red, Red, Red, White, White,
     Red, Red, Transp, Transp},
    {Transp, Red, Red, White, White, White, White, Red, Red, White, White,
     White, White, Red, Red, Transp},
    {Transp, Red, Red, White, White, White, White, Red, Red, White, White,
     White, White, Red, Red, Transp},
    {Transp, Red, Red, White, Blue, Blue, White, Red, Red, White, Blue, Blue,
     White, Red, Red, Transp},
    {Transp, Red, Red, Red, Blue, Blue, Red, Red, Red, Red, Blue, Blue, Red,
     Red, Red, Transp},
    {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Red, Transp},
    {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Red, Transp},
    {Transp, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red, Red,
     Red, Transp},
    {Transp, Red, Red, Transp, Red, Red, Red, Transp, Transp, Red, Red, Red,
     Transp, Red, Red, Transp},
    {Transp, Red, Transp, Transp, Transp, Red, Red, Transp, Transp, Red, Red,
     Transp, Transp, Transp, Red, Transp},
    {Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp, Transp,
     Transp, Transp, Transp, Transp, Transp, Transp, Transp},
};
```



```c
const uint8_t *sprites[] = {
    (uint8_t *)sprite_pacman_closed, (uint8_t *)sprite_pacman_left,
    (uint8_t *)sprite_pacman_right,  (uint8_t *)sprite_pacman_up,
    (uint8_t *)sprite_pacman_down,   (uint8_t *)sprite_ghost_red,
    (uint8_t *)sprite_ghost_cyan,    (uint8_t *)sprite_ghost_pink,
    (uint8_t *)sprite_ghost_orange,  (uint8_t *)sprite_ghost_scatter,
};
```

Software: Peripheral

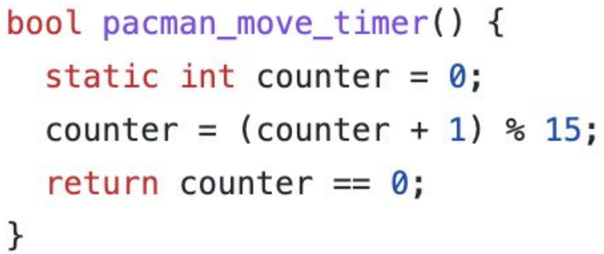
- Gamepad
- libusb-1.0
- Event listeners
  - Key-up and key-down events
  - Fires for each individual event and individual button



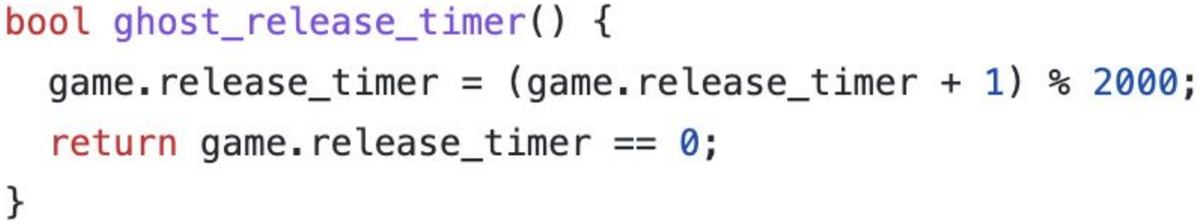
```
void gamepad_set_listener(void (*listener)(gamepad_button_event_t,
                                            gamepad_button_t));
```

Software: Game Loop

- 3 stages
  - STAGE_MENU
  - STAGE_IN_GAME
  - STAGE_END_GAME
- usleep(1000)
- Timers to give varying rates

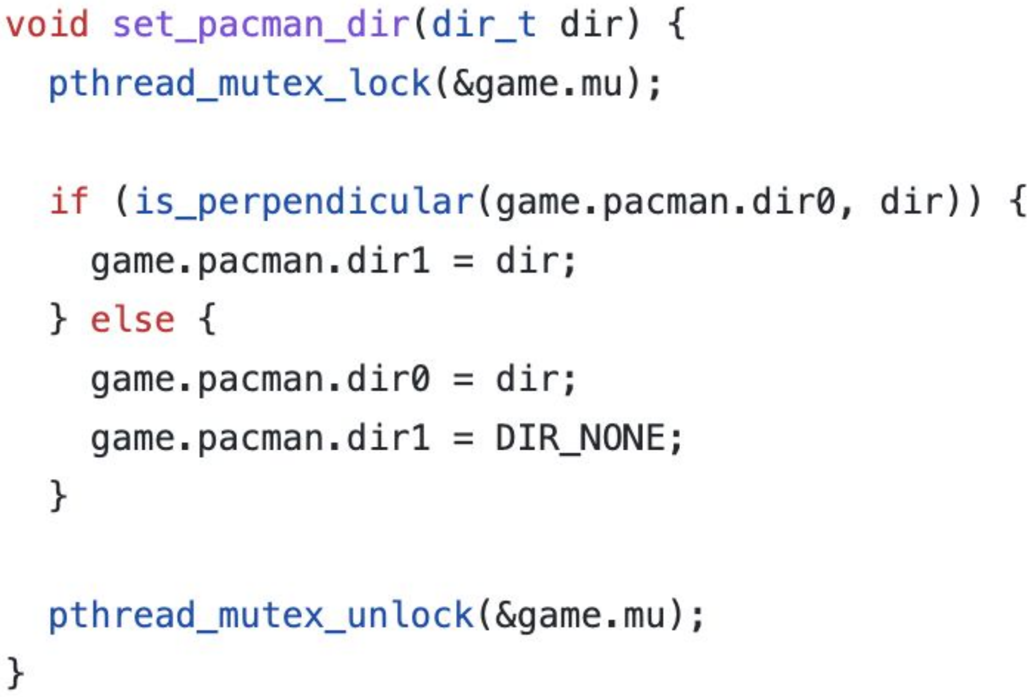


```
bool pacman_move_timer() {
  static int counter = 0;
  counter = (counter + 1) % 15;
  return counter == 0;
}
```



```
bool ghost_release_timer() {
  game.release_timer = (game.release_timer + 1) % 2000;
  return game.release_timer == 0;
}
```

Software: Pacman Movement

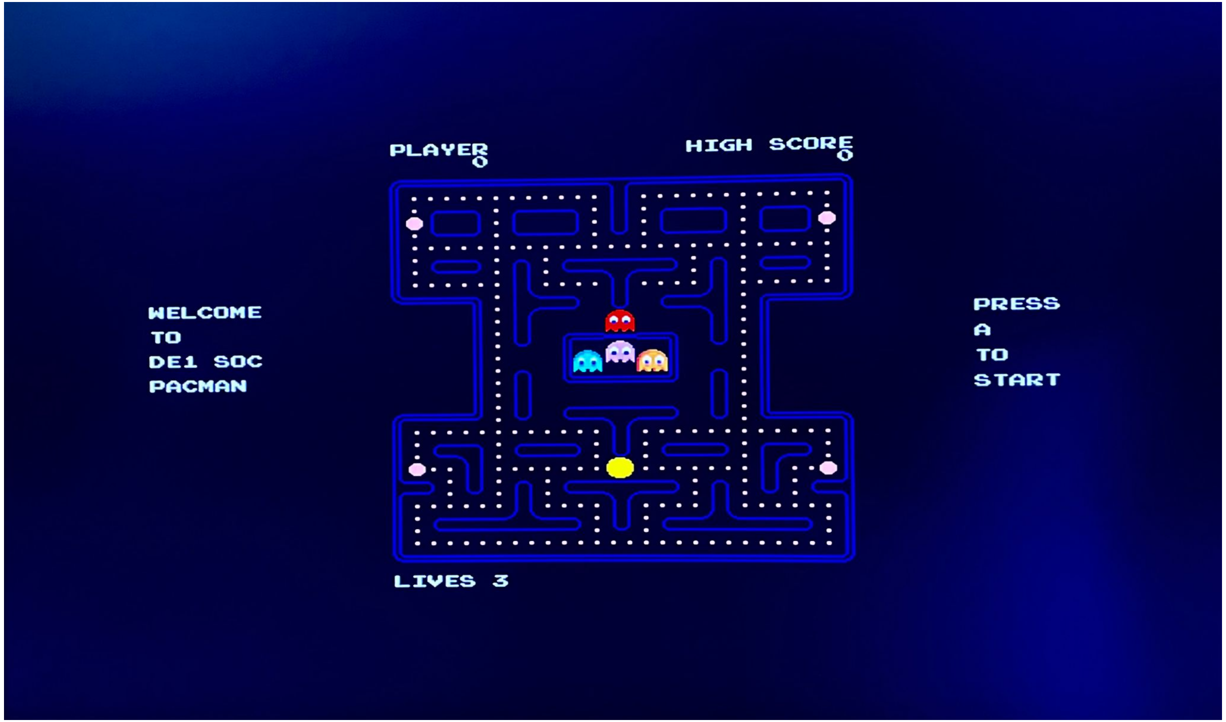


```c
void set_pacman_dir(dir_t dir) {
  pthread_mutex_lock(&game.mu);

  if (is_perpendicular(game.pacman.dir0, dir)) {
    game.pacman.dir1 = dir;
  } else {
    game.pacman.dir0 = dir;
    game.pacman.dir1 = DIR_NONE;
  }

  pthread_mutex_unlock(&game.mu);
}
```

Software: Ghosts Movement

- Modes:
  - trapped: up & down in middle cell
  - release: 2-phase move to designated start point
  - random: at each point, pick a random direction (but never backward)
  - chase:
    - run BFS for each direction, record depths of finding pacman
    - pick the direction with lowest depth
  - scatter
    - run BFS for each direction, record depths of finding pacman
    - pick the direction with highest depth

Screenshot of finished game here



Challenges, Lessons Learned

- Debugging hardware requires alternate workflows (e.g. ModelSim RTL simulation)
- Clocking and managing memory accesses
- Software/Hardware integration and troubleshooting
- Nice to have HW/SW interface early
- So many variables in game development…
  - Abstractions are important
  - Understand why OOP is popular among game devs now

Demo