

# Pac-Man Design Document

Caiwu Chen (cc4786), Tz-Jie Yu (ty2534), Emma Li (eql2002), Haoming Ma (hm3070)

CSEE4840 Spring 2025

## Contents

1. Introduction
2. A Block Diagram
3. A Description of the Algorithms
4. Resource budgets
5. Hardware/software Interface

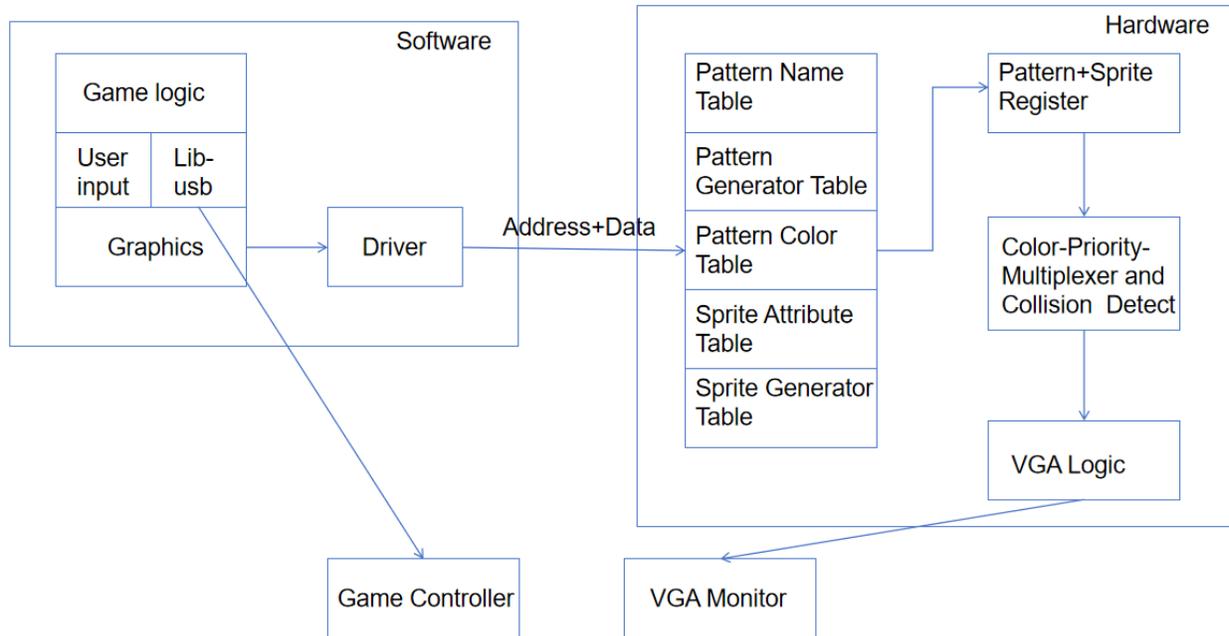
# 1 Introduction

In our project, we re-create the classic arcade game Pac-Man on an FPGA platform, DE1-SoC Board, and using the VGA monitor. Pac-Man is a game in which the player navigates a maze, aiming to consume all the pellets while avoiding being caught by roaming ghosts. Our implementation separates responsibilities between hardware and software: the FPGA is responsible for rendering the game's graphics to a VGA display, while the software controls the game logic and communicates with the hardware via a device driver. User input is provided through a USB game controller (keyboard with arrow keys), enabling real-time control of Pac-Man's movement.

## 2 System Block Diagram

The block diagram includes all major software and hardware components. Components such as the game logic, input handler, and device drivers reside in software, while hardware includes a sprite engine, tilemap renderer, audio module, and VGA controller. Communication occurs via memory-mapped registers on the Avalon Bus.





### 3 Algorithms

#### Game logic:

The game begins when the player presses the Start button. The player controls a character, the Pac-Man, that continuously moves in the direction of its mouth. Movement can be changed using the directional pad, which is the major input during gameplay.

As the character moves over a pellet, it is eaten, removed from the game board, and the player's score increases. The current score is tracked in software and updated on the display through a dedicated memory-mapped register. Score digits are rendered as sprites on the VGA display using an 8x8 pixel font. The objective is to eat all the pellets in the maze.

Ghosts are non-playable enemy sprites that move around the maze according to predefined or randomized patterns. The four ghosts in the game all use Chase mode, Ghosts will chase after the player according to their own logic. If a ghost collides with the player, the game ends.

The game ends when either

- The player eats all the pellets (win condition), or
- The player collides with a ghost (loss condition)

To restart the game after completion, the player presses the Start button again.

## Graphics rendering:

Our current design uses SystemVerilog to implement and render all visual components of the game, including the maze, player sprite, ghosts, pellets, and score digits.

Static elements such as the maze walls, pellets, and text (e.g., "SCORE") are rendered using a tile-based rendering system. The VGA display is divided into a fixed 8×8 pixel grid, and each grid cell displays a predefined tile stored in on-chip ROM. A tilemap stored in block RAM defines which tile appears at each position on screen. Each tile's appearance is determined by an indexed pattern with associated color values.

Pellet presence is tracked using a dedicated Pellet RAM, where each row of the maze corresponds to a 32-bit word. Each bit represents the presence or absence of a pellet at a particular horizontal position—bit 0 is the right-most pellet, and bit 31 is the left-most. The software updates this RAM by writing to memory-mapped registers, enabling real-time dynamic updates to the pellet layout during gameplay.

Sprites, including Pac-Man, ghosts, and score digits, are stored in on-chip Block RAM or ROM. These sprites are rendered using a custom sprite engine, which reads sprite descriptor data (e.g., x/y position, frame index, visibility) and draws active sprites during VGA scanline generation. This approach enables layering of moving sprites over a static tilemap background with minimal flicker and latency.

## Sound Generation

We include pre-sampled sound effects (e.g., pellet chomp, ghost eaten, death). These are stored in ROM and played using a PWM-based audio module that drives a connected speaker. Sounds are triggered via memory-mapped registers.

## 4 Resource Budget

One of the primary consumers of on-chip memory in our Pac-Man implementation is graphical data, including static tiles and dynamic sprites. Due to the limited amount of on-chip RAM on the DE1-SoC FPGA (less than 512KB), special attention is paid to the size and format of these resources.

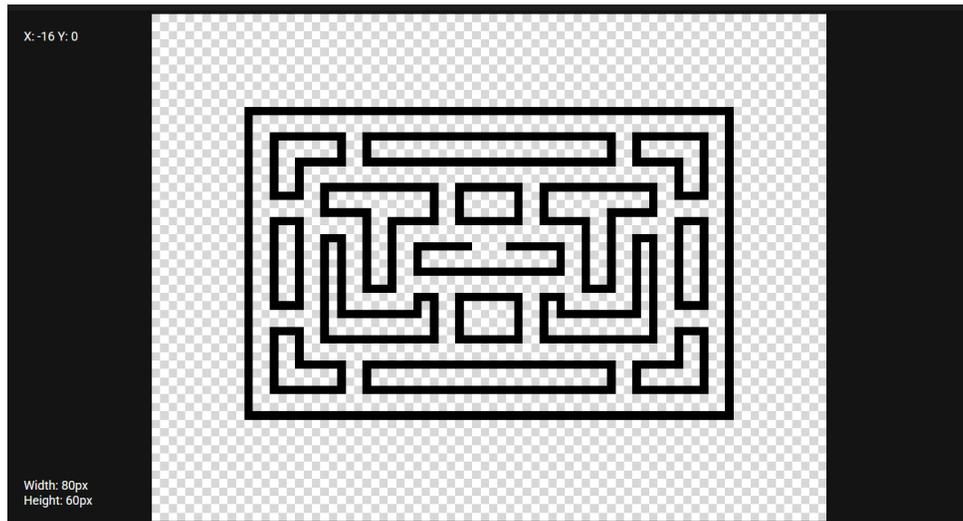
The following table provides a breakdown of estimated memory consumption for all tiles and sprites used in the game. Each image is represented as raw RGB data (8 bits per channel) and stored in Block RAM (BRAM) or ROM on the FPGA for fast access during scanline rendering.

Category	Name	Graphics	Size (bits) Width x Height x Channel x Bit-depth	# of images	Total size (bits)
Sprite	Pac-Man		16 x 16 x 3 x 8	2	12288
Sprite	Ghost		16 x 16 x 3 x 8	4 (without animation)	24576
Tile	Borders		8 x 8 x 3 x 8	6	9216
Tile	Pellets		8 x 8 x 3 x 8	1	1536
Tile	Letters		8 x 8 x 3 x 8	5	7680
Tile	Numbers		8 x 8 x 3 x 8	10	15360
Sound	Music		-	-	-
Total					

In addition to graphical assets, the display system also consumes a small portion of on-chip memory for real-time game state storage, such as the tile map layout, pellet presence, and sprite descriptors. These structures are compact but require frequent access during gameplay and must be stored in BRAM for low-latency reads. The estimated sizes are as follows:

- Tilemap RAM:  $40 \times 30$  tiles  $\times$  1 byte = 1.2 KB
- Pellet RAM: 30 rows  $\times$  4 bytes = 120 bytes
- Sprite Descriptors: 5 sprites  $\times$  8 bytes = 40 bytes

Total Bitmap Memory Usage:  $\sim$ 1.4 KB



This is a map we have designed for a long time, and we may make some changes later. Thus, the combined memory usage for both image assets and display control data remains well within the on-chip memory budget of the DE1-SoC FPGA, with sufficient headroom for additional assets or logic.

## 5 The Hardware/Software Interface

The software is responsible for maintaining the overall game logic, including player input handling, movement updates, collision detection, score tracking, and game state transitions. It receives input from the keyboard (or gamepad) and processes this information to update the positions and states of Pac-Man, ghosts, pellets, and other game elements.

These updates are communicated to the graphics hardware via a memory-mapped interface. The hardware rendering system reads this data in real-time and generates the corresponding video output signals. Specifically, it renders all visual components—including the static maze background, dynamic sprites, pellets, and text—into VGA signals, which are then displayed on an external monitor.

The address map used for communication between software and hardware is structured as follows:

Base Address	Name	Description
0x0000 - 0x04AF	Tilemap RAM	Tile index for 40 * 30 grid
0x1000 - 0x101E	Pellet RAM	30 rows * 32 bits pellet state
0x2000 - 0x2027	Sprite Descriptors	Descriptors for sprites, including position, orientation. Max 5 sprites (1 Pac-man, 4 ghosts), 8B for each.
0x3000	Score Register	Current score (16 bits)
0x4000	Control Register	Signals for start/reset/pause/display sync etc.
0x5000	Reserved Register	Reserved for future use.

Each region is aligned to 4KB (0x1000) for future-proofing and easy hardware decoding.

## Tilemap RAM

- Description: A 40 × 30 grid where each byte represents a tile index, used for static background elements such as maze walls, pellets, score labels and number files.
- Tile size: 8 × 8
- Data format:
  - Each byte: `uint8_t tile_index`
  - Example: Writing 0x0A to address 0x0022 sets the tile at row 1, column 2 to index 0x0A.
- Access: Software initializes the map at the beginning.

## Pellet RAM

- Description: Tracks presence of pellets in each row.
- Data format:
  - Each row is a 32-bit work (`uint32_t`), each bit indicates one pellet.

- Bit 0 ⇒ right most pellet.
- Access: Software clears a bit when Pac-Man eats a pellet.

## Sprite Descriptors

- Description: Each sprite has a 8-byte descriptor, controlling its position, animation frame and visibility.
- Data format:
  - Byte 0: X position
  - Byte 1: Y position
  - Byte 2: Frame (for animation)
  - Byte 3: Visibility (0 = invisible, 1 = visible)
  - Byte 4: Direction (0 = up, 1 = right, 2 = down, 3 = left)
  - Byte 5: Type (1 ~ 5 for ghosts, 0 for Pac-man)
  - Byte 6, 7 Reserved

## Score Register

- Description: Stores current player score, up to 65535
- Format: 16-bit unsigned integer
- Access: Written by software after score update

## Control Register

- Description: Provides software control signals to manage running status of hardware.
- Format:
  -

Bit	Name	Function
0	Start	Start or resume the game
1	Reset	Reset game state
2	Pause	Pause the game
3	VBLANK_ACK	Acknowledge frame sync
4	Game_Over	Game over flag