EE 314: Digital Circuits Laboratory
Term Project Description
FPGA-Based 2-Player Fighting Game

Objectives

This project involves the design and implementation of a minimalistic yet engaging fighting game for two players, built entirely in hardware using Verilog HDL and deployed on an FPGA development board.

The game is inspired by Footsies, an open-source fighting game developed by HiFight - detailed information about the game mechanics and design can be found on the official website [1]. To develop a deeper understanding of game mechanics and terminology, it is recommended to play the original game. The game includes helpful features such as frame-by-frame playback and debug visualizations (e.g., hitbox overlays), which can aid in analyzing gameplay and understanding timing and interactions more precisely.

In the game, each player controls a character who can move within a 2D arena and perform attacks with the goal of reducing the opponent's health to zero. The game will run in real time on an FPGA with output on a VGA display and will feature physical input via a keypad.

The goal is to reinforce students' understanding of digital design principles through a complete system that involves real-time graphics, user input handling, finite state machines (FSMs), and interactive gameplay. This project serves as a hands-on exercise in integrating multiple digital subsystems and debugging complex behavior in hardware.

- The FPGAs will be provided by the laboratory. For details of borrowing an FPGA for your group, refer to the FPGA Borrowing document on ODTUClass.

- For player input, you will need an external keypad that is compatible with the FPGAs' peripherals (GPIO pins), which you need to purchase yourselves. If you have an issue finding a suitable keypad, you may contact the coordinator assistant.

- Early demonstrations are possible and encouraged. Please contact the coordinator assistant in advance for scheduling.

Important Dates, Deliverables & Grading

- Milestone Report: 30th of May (**10% Credits**)

- Milestone Demonstration: 31st May–1st June (**20% Credits**)

- Final Report: 24th June (**15% Credits**)

- Project Video: 24th June (**5% Credits**)

- Final Demonstration: 25th–27th June (**50% Credits**)

For the details on deliverables, please refer to the Term Project Deliverables document on ODTUClass.

1 Game Concept Overview

The game is a side-view 2D fighting experience for two human players or one player against a CPU-controlled bot. Player(s) use physical buttons on a keypad to control their respective characters. Each character can perform basic actions such as:

- Move Left / Right

- Attack

- Block (Defensive Mechanic)

Players start each round with 3 health and 3 block points. A player cannot block attacks if their block counter is zero. A player loses the game if their health reaches zero. A hit detection system calculates whether an attack successfully hits, whiffs, or is blocked. If the attack hits, the opponent's health is reduced accordingly. If an attack is blocked, the opponent's block counter is reduced.

A 1-player mod introduces a simple bot with a pseudo-randomized behavior pattern that simulates a basic opponent. The bot behaves as if it is a player who presses buttons on the keypad randomly. The bot does not need to be intelligent, but should provide a dynamic opponent. Note that just like a real player, the bot should be able to press multiple buttons at the same time as well.

2 Game Flow and System Behaviour

2.1 Game Start and Initialization

The game enters an idle state upon power-up or reset. During this time, the currently selected game mode (2-Player or Player vs Random) will be displayed on the 7-segment displays. The player can switch between game mode selections using SW[0]. The game starts when the designated Player 1 presses the confirm button (You can pick any of the buttons assigned to Player 1). A countdown like "3", "2", "1", "START" is first displayed on the screen, during which no input should be accepted. After the countdown ends, the fighting can begin.

2.2 Gameplay Loop

The core of the game is a deterministic state machine that evaluates player inputs and transitions characters between states based on rules defined for movement and attacks. **The character can only be in one state at a given point in time.** Game logic runs with a 60 Hz clock. Since the VGA refresh rate is also 60 Hz, the character state changes occur solely between consecutive frames.

Mandatory : You should also use SW[1] to switch between a 60Hz signal and a KEY on the FPGA for the clock of Game Logic. This will be used during your demonstrations, but also for you when you are debugging.

Character States:

- **IDLE:** No input detected, character is standing still.

- **MOVING FORWARD:** Horizontal position changes towards the enemy by **three pixels per frame** on the 60 Hz VGA display. Note that the players cannot cross each other.

- **MOVING BACKWARDS:** Horizontal position changes away from the enemy by **two pixels per frame** on the 60 Hz VGA display. Note that the players cannot move out of the screen boundaries.

- **ATTACK (Basic):** A three-phase attack consisting of startup, active, and recovery phases.

- **ATTACK (Directional):** A three-phase attack that happens when the attack button is pressed while moving left or right, consisting of startup, active, and recovery phases.

- **HITSTUN:** Temporary state after being hit.

- **BLOCKSTUN:** Temporary state after blocking.

Check the Table 1 on the Appendix A for attack, hitstun and blockstun durations. Also, check the Appendix A for example cases on attack, hurtbox, and hitbox interactions.

2.3 Hit Detection and Resolution

In fighting games and many other types of competitive action games, characters are not just represented visually by their sprites or 3D models — they are also defined by invisible zones known as hitboxes and hurtboxes. These are fundamental concepts in game mechanics and collision detection.

- **Hurtbox:** A hurtbox is an invisible area that defines the region of a character's body that can be damaged by an opponent's attack. In essence, it's the "vulnerable" area of a character. Hurtboxes typically follow the character closely, changing position as the character moves.

- **Hitbox:** A hitbox is the active part of an attack — the area that can inflict damage or trigger a reaction when it intersects with an opponent's hurtbox. Hitboxes are generated when a player performs an attack action. Just like hurtboxes, hitboxes are usually not static; they follow the attack animation and may only appear for a few frames (fractions of a second) during the move's active phase. It is important to note that during the recovery frames of an attack, the hitbox extended from the character is converted into a hurtbox.

- **Interaction Between Hitbox and Hurtbox:** The core mechanic revolves around the interaction between hitboxes and hurtboxes. When the hitbox of an attacking character overlaps (or "collides") with the hurtbox of the defending character, the attack is considered to have "connected" or "landed." This results in the defending character taking damage (their health reduced by 1) and entering the hitstun state. If the defending character was blocking, they would not take any damage, but decrease their block counter and enter the blockstun state instead.

- **Visual Debugging Aids:** While normally invisible during gameplay, developers often visualize hitboxes and hurtboxes in development tools or debug modes for testing and balancing purposes. In many visualizations:

  - Hurtboxes are often shown in yellow, indicating where the character can be hit.
  - Hitboxes are commonly represented in red, showing the active attack zone.

  These visual aids are crucial for debugging the design.

Blocking is done by walking backwards. Meaning, if an attack lands on an opponent walking backwards, the opponent counts as having blocked the attack instead of getting hit.

It is important to understand that a properly functioning hit detection system is fundamental to the game's logic. Without reliable hit detection, it is not feasible to implement key mechanics such as blocking, hitstun, blockstun, game-over conditions, and related game states in a coherent and consistent manner.

Therefore, the development process must follow a logical sequence. Certain components—such as attacking and hit detection—must be implemented and verified before progressing to more advanced gameplay features.

Edge case, simultaneous hits: If both characters enter the active attack phase at the same time and their hitboxes overlap, both are hit. Damage is applied simultaneously.

2.4 Game Over and Restart

When a player's health drops to zero, the system enters a "Game Over" state. A message is displayed on the screen declaring the winner. At this point:

- The seven-segment display shows the winner and the passed time.
- LEDs blink to indicate the end of the game.
- Player 1 can exit to the menu using their buttons.

3 Input and Output Specification

3.1 Input Devices

All player controls should be implemented using button inputs. Ideally, two separate keypads should be used—one for each player. Player 1 may use the onboard FPGA buttons, while Player 2 can use an external keypad as an example (any basic functional keypad is acceptable). You should implement a debouncing scheme to filter out transient fluctuations due to mechanical bouncing.

Each player must have exactly three buttons assigned to the following actions:

- Move Left
- Move Right
- Attack

No additional control inputs should be used beyond these three.

The players can press none or any combination of buttons. The players can also hold the button down, in which case the input should be continuously given to the game logic. The game should have a deterministic input handling such that the same case occurs for all button combinations. For example, holding both Left and Right simultaneously.

3.2 Output Devices

3.2.1 VGA Specification

The VGA controller operates with a 25.172 MHz clock but a 25 MHz clock should also do the trick. The clock on our FPGA boards runs at 50 MHz, so you should divide it accordingly. **You are not required to write your own VGA controller**. This website [2] has a fully functional VGA controller you can use and is a great starting point, but you can use other sources if you want. Recall that whichever source you use, you should properly reference it in the report.

Drive the VGA with:

- Resolution: 640×480 at 60Hz.
- Color Format: 8-bit (3-3-2 RGB).

Menu Screen

- Display a menu screen. The minimum expected menu screen should at least display the text "MENU", but you can get as creative as you want.

Gameplay Screen

- Display a countdown: **"3", "2", "1", "START"** before the match begins.
- Character sprites should be rendered at a resolution of **64×240 pixels.**
- The players and their actions should be displayed on the VGA screen. Some of these actions (moving and attacking) must be animated on the screen. While the others (getting hit, blockstun, hitstun, attack phases, etc.) can be shown with just color changes. You can animate these as well, as long as they are discernible.
- There should also be a count-up timer in the upper-center of the screen that displays the time that has passed in seconds since the match started.
- Your game should have a background. It can be a static color or a static/moving pattern. A pure black background is not allowed.
- Remaining health points should be visible on the screen.
- Remaining block points should be visible on the screen

Game Over Screen

- Freeze the timer and the players.

- Display the outcome of the match on the screen.

- The game over screen should be present until Player 1 presses any button. Then, the display should return to the menu screen.

3.2.2   Seven-Segment Displays

Menu Mode

- On 7-segment displays, first "1P" or "2P" should be written. SW[0] is used to toggle between these game modes, representing one-player vs random opponent and two-player modes, respectively.

- Player 1 should select the game mode written on the 7-segment display by pressing any button.

Gameplay Mode

- After the game mode is selected, the 7-segment display should show "FIGHt" and the game shall begin.

- The 7-segment display should show "FIGHt" during the entire gameplay.

Game Over Mode

- Display the result in the format **"P1–XX-"** or **"P2–XX-"**, where the XX number represents the duration of the match in seconds.

You can assume the maximum game time is 99 seconds. Note that the game can also end in a draw, even though it is very unlikely. For this case, "Eq" should be written on the 7-segment display instead of the player name, such as "Eq-XX-". The game over display should be present until Player 1 presses any button. Then, the 7-segment display should return to the menu mode.

3.2.3   LED Indicators

Menu Mode

- All LEDs remain **off**.

Gameplay Mode

- The **leftmost 3 LEDs** indicate **Player 1's health.**

- The **rightmost 3 LEDs** indicate **Player 2's health.**

- LEDs are turned off progressively to reflect damage taken.

Game Over Mode

- **All LEDs blink** simultaneously to indicate the end of the game.

References

[1] HiFight, "FOOTSIES," 2018. Accessed: 2025-05-10.

[2] V. H. Adams, "Vga driver in verilog," 2025. Accessed: 2025-05-10.

A Frame Data Information and Visuals

The directional attack has a shorter range compared to the neutral attack, but benefits from a faster startup, meaning it becomes active quicker after the input is made. Despite the difference in range and speed, both this attack and its alternative share the same on-hit and on-block frame properties. Specifically:

- On hit, the attacker recovers 1 frame later than the defender.

- On block, the attacker recovers 3 frames later than the defender.

This frame disadvantage means that after the attack connects, whether it lands or is blocked, the defender recovers sooner and can act before the attacker. As a result, if both players attempt to retaliate by pressing the attack button immediately after the interaction, the defender's follow-up move will start earlier. This can lead to the defender successfully interrupting the attacker's next move and scoring a hit.

The character on the left side of the screen is designated as Player 1, while the character on the right side is designated as Player 2. Recovery values in Table 1 are given relative to the Attacker. Hence, -1 means the Attacker recovers 1 frame later than the receiver.

Table 1: Frame Data for Attacks (Images taken from [1])

Move	Command	Frame			Recovery	
		Startup	**Active**	**Recovery**	**On Hit**	**On Block**
	Neutral + Attack	5	2	16	-1	-3
	Forward or backwards + Attack	4	3	15	-1	-3

You can see an example hurtbox/hitbox structure in Figure 4 taken from 4.

Suppose Player 1 performs an attack (which has a red hitbox) and Player 2 is standing nearby with a visible yellow hurtbox. If the attack hitbox reaches into Player 2's hurtbox during the active frames of the attack and Player 2 was not blocking, the game will register a hit, causing Player 2 to react accordingly (e.g., take damage and enter hitstun state). This example is shown in Figure 1. If Player 2 blocks the attack, their block counter will decrease, and Player 2 will enter the blockstun state. This example is shown in Figure 2.

Suppose Player 1 performs an attack (which has a red hitbox) and Player 2 is standing nearby with a visible yellow hurtbox. If the attack hitbox does not reach into Player 2's hurtbox during the active frames of the attack (meaning it whiffs), Player 2 can attack and hit the extended hurtbox of Player 1. This example is shown in Figure 3.
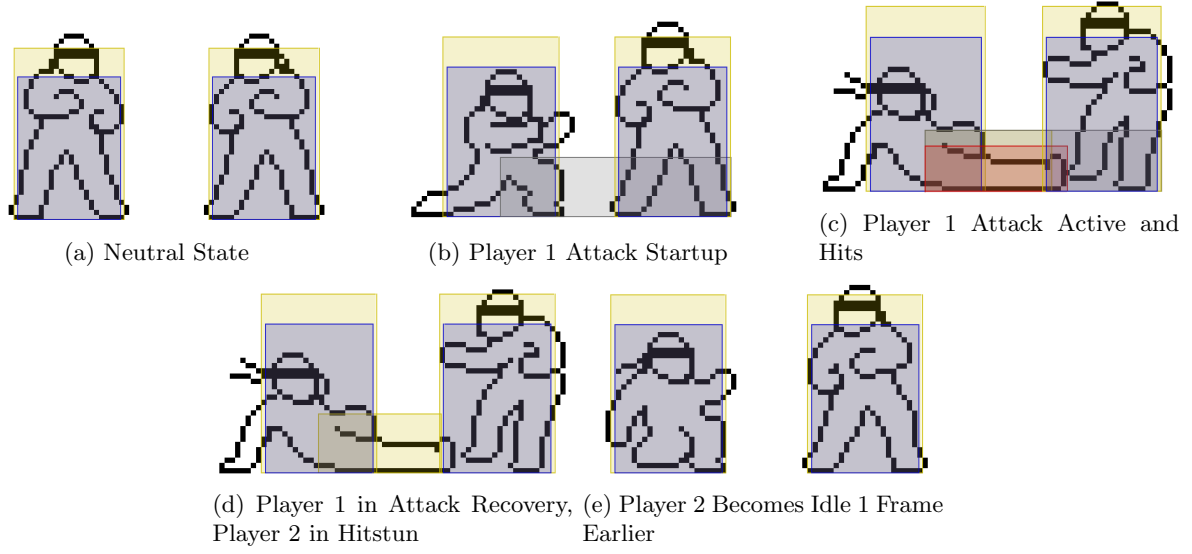
(a) Neutral State

(b) Player 1 Attack Startup

(c) Player 1 Attack Active and Hits

(d) Player 1 in Attack Recovery, Player 2 in Hitstun

(e) Player 2 Becomes Idle 1 Frame Earlier

Figure 1: How a Hit Occurs (Images taken from [1])

(a) Neutral State

(b) Player 1 Attack Startup

(c) Player 1 Attack Active, Player 2 Blocks

(d) Player 1 in Attack Recovery, Player 2 in Blockstun

(e) Player 2 Becomes Idle 3 Frames Earlier
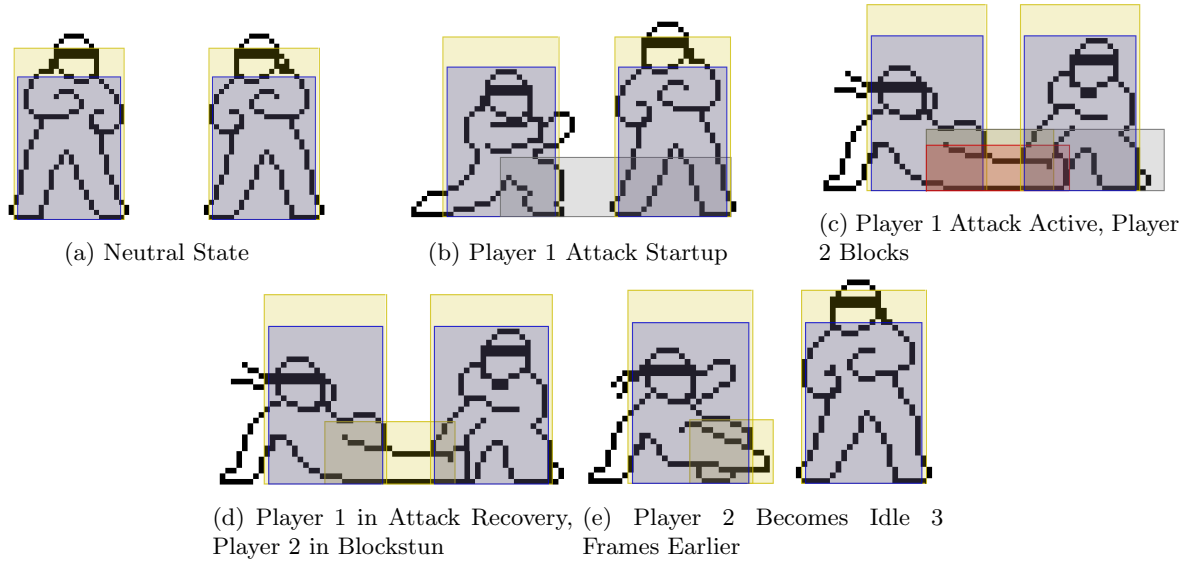
Figure 2: How a Block Occurs (Images taken from [1])

(a) Neutral State

(b) Player 1 Attack Startup

(c) Player 1 Attack Active and Whiffs

(d) Player 1 in Attack Recovery, Player 2 in Idle

(e) Player 1 in Attack Recovery, Player 2 Attack Startup
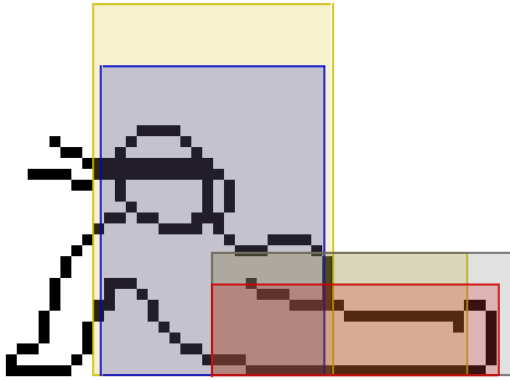
(f) Player 2 Attack Active and Hits
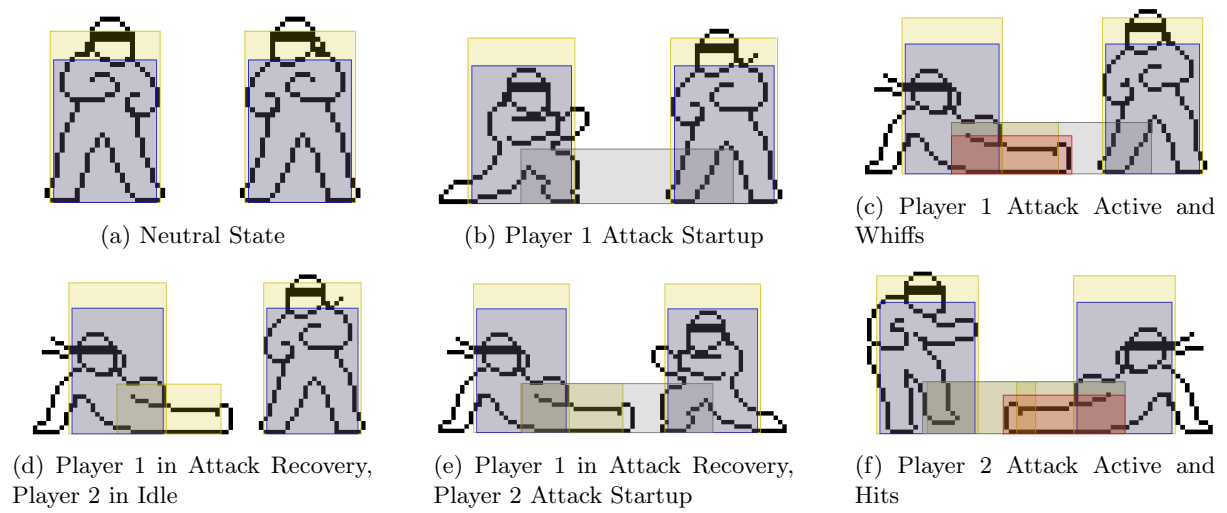
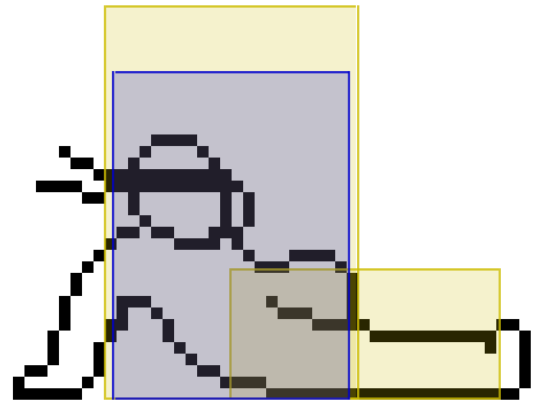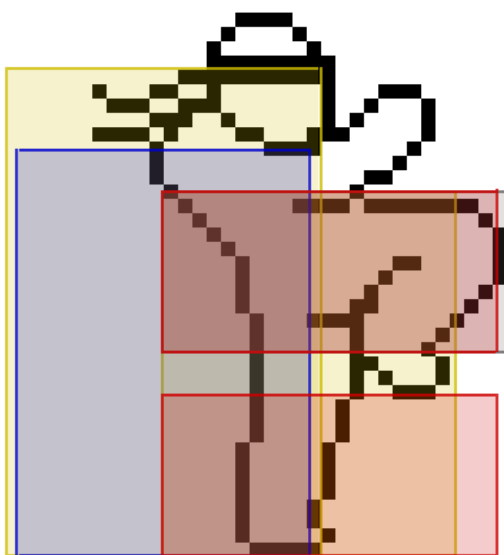Figure 3: How a Whiff and Punish Occurs (Images taken from [1])
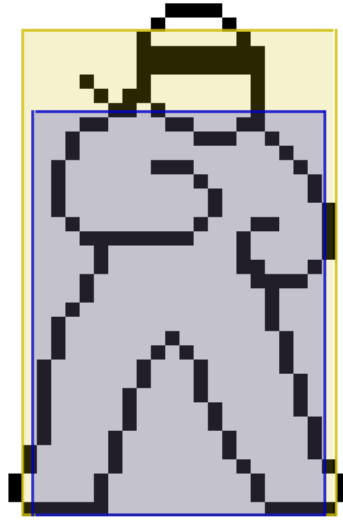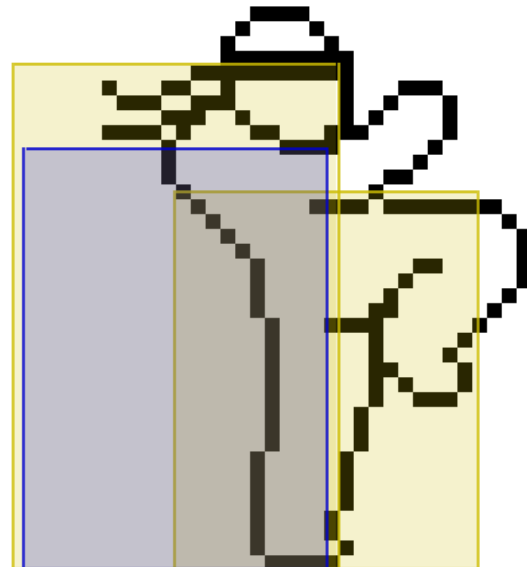
(a) Idle State Hurtbox

(b) Neutral Attack Hitbox

(c) Neutral Attack Recovery Hurtbox

(d) Directional Attack Hitbox

(e) Directional Attack Recovery Hurtbox

Figure 4: Hitbox and Hurtboxes of Attacks (Images taken from [1])

B All Other Things You Want to Add

You can add more than what is specified for the project to make the game more complex. **For such extensions of the project, there is not really any limitation; you can add more buttons, etc.**

For example, for actions:

- Jumping (By a new Up button)

- Crouching (By a new Down button)

- Combo attacks

- Command inputs (Motion/Direction Input + Attack, etc)

- ...

For example, for visuals:

- Background stages

- Stage select

- Character color scheme select

- Character animations

- Win and Lose animations

- ...

Please remember that to get credits from the project, you must fulfill the project's original criteria with its original restrictions (3 Buttons, for example). Any such extensions will not result in additional credits from the project.

C Parts List

```
DE1-SoC Board
A VGA Monitor
Keypad(s) with at least 3 buttons
```