Report

Full Demo link:

- Complexity 1:
- Complexity 2:
- Complexity 3:

Project Description:

My project, Dot Dash, is an obstacle avoidance game written for the ATmega1284p microcontroller, where the player controls a dot on an LED matrix using a joystick to avoid walls that are coming towards them. The LED Matrix is driven by two shift registers, and the joystick is connected to the atmega's ADC. Concurrent SynchSMs control the game logic in software as well as driving the two hardware complexities.

User Guide:

Rules:

The player can move the dot left or right, and the dots come at them from above. If the player has their dot in the same spot as a wall, they lose. Their dot is frozen, the grid vanishes, and their score becomes zero.

The player's score increases if they survive to the end of a level, and the walls speed up. If the player reaches a score of 3, they win the game (indicated by the score lights being fully lit up and the walls disappearing).

Controls:

- ***Joystick:*** The main peripheral the player uses to control the game is a joystick. The joystick measures the X-axis movement and moves the player's dot accordingly.
- ***Reset Button:*** The player can also reset the game at anytime by pressing a button. This resets the game to its initial state, wiping everything.

Special Considerations

- The joystick can technically read the Y-axis as well, however this is not strictly necessary for gameplay as the player only needs to move left or right. The SynchSM controlling player movement therefore only moves the player horizontally, though it has the capability to be modified to move vertically and the game would still run (there is not much room for vertical movement in the level design however).
- The joystick has a deadzone to allow for no movement and minimize unwanted inputs. My particular joystick was somewhat skewed towards the positive X-axis (moving to the right), so I adjusted to that by slightly skewing the deadzone to require more +X movement before registering a “move-right” input.

Explanation of Components/Relevant Technologies:

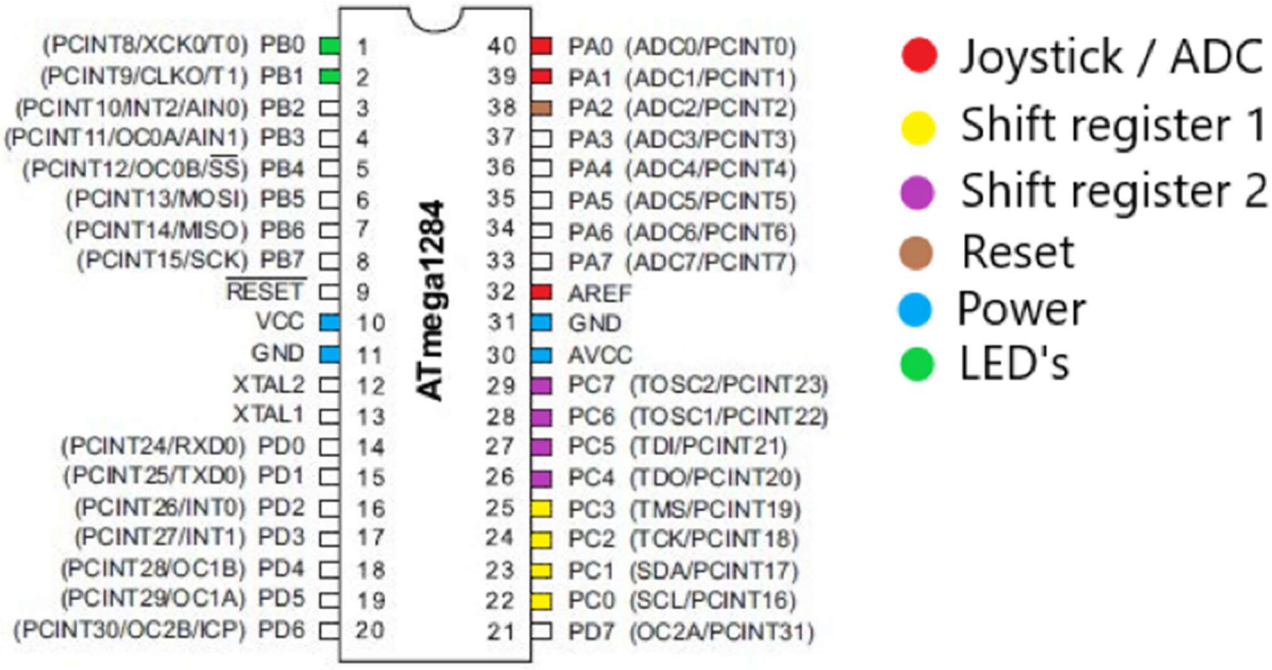
- **ATmega1284p**: The same microcontroller we used for the labs in class.
- **74HC595 Shift Register**: Used in conjunction with the 8x8 LED Matrix to allow the microcontroller to drive the matrix using 8 pins output. One shift register controls the columns of the matrix, the other controls the rows.
    - **Time Multiplexed Output**: Because the SR is configured as Serial-In-Parallel-Out (SIPO), I used time-multiplexed output to send in data in series to the shift registers to allow them to drive the matrix.
- **8x8 LED Matrix:** Used as the display of the system. Driven by two shift registers, as described above.
    - **Raster Scanning**: Because the matrix needs to control 64 LED’s using only 16 pins, simply powering multiple rows and columns will light more than the desired LED’s. To mitigate this, I implemented a form of raster scanning to drive the matrix, lighting up one row at a time and quickly alternating between the required ones per frame.

    - NOTE: I had to decide on a period for how often to alternate between rows, and settled on a 10ms period per line. This gave me the best balance of minimizing ghosting while still making it look presentable on camera.
- **Joystick**: Main control peripheral used by the user.
    - **ADC**: Used to drive the joystick implemented with the help of the ATmega1284's built in ADC functionality.
- **LED's:** Used as displays to show the user's score, win/lose status, etc.

Source File Description:

- **main.c**: Contains all the state machine code and game logic.
- **Matrix.h:** A header file written by me containing an abstracted interface to drive the matrix.
- **JoyADC.h**: Also written by me, this drives the joystick and includes the relevant code needed to activate the ADC (not written by me) needed to do so.
- **timer.h**: Timer interface provided to us from the class Google drive. (NOT WRITTEN BY ME)
- **simAVRHeader.h:** AVR header provided when creating a project for the class. (NOT WRITTEN BY ME).

Component Visualization:



Technologies Learned:

- **AVR Studio 6**
- **ATmega1284p**
- **Shift Registers/Time-multiplexed output**
- **LED Matrices/Raster scanning**
- **Joystick (Analogue-Digital-Conversion)**