Mark Mchedlishvili
Eric Mendoza-Conner

Tetris

Abstract

The circuit is a simulation of the game Tetris. Pieces are generated pseudo-randomly and fall at a constant rate. The user may move one piece at a time horizontally or rotate it. When the piece lands on either the bottom row or another piece, it is locked in place and the user controls a newly generated piece at the top of the board. When a row is filled with pieces, it is cleared and all pieces above it fall down one cell.

Design Narrative

The four push buttons are attached to 555 timers in a debounce configuration such that pressing the button will output a 18ms high voltage pulse. Two of the buttons control the horizontal movement of the tetris piece, while the other two control the rotation of the piece. Each is connected to various set/reset pins of four flip flops such that at most one flip flop will have a high voltage output at any given time. On the rising edge of an astable timer, the value of the flip-flops will increment or decrement a corresponding counter, indicating a change in horizontal position or rotation. Between the rising and falling edge, various memory management processes taking place, including collision detection. An astable timer connected to a counter controls the vertical position of the piece. A linear feedback shift register circuit generates a new random piece after a falling piece lands. The horizontal and vertical positions, the rotation, and the type of piece are used as inputs for two GAL22V10 programmable chips. These chips use the inputs to output a grid row by row containing the falling piece to an SRAM chip through tri-state buffers. Each value in this SRAM chip is compared to the value in another

SRAM chip which contains the static piece grid. If collision is detected, a high output signal is sent to check whether this was due to an illegal move (rotating/moving in a location where it is impossible to do so) or due to falling. If it is an illegal move, the last action performed by the user (as recorded by the flip flops) is undone by rotating or moving the piece in the opposite direction. Otherwise, each value of within each SRAM chip is connected through an OR gate and written to the static piece through hex buffers, thereby "adding" the falling piece to the static piece grid. Simultaneously, a new falling piece is generated and positioned at the top of the grid. The falling edge of the input clock resets the flip flops, after checks have been made that the user's input does not cause illegal collision with the grid. After checking for collision, each row of the static piece grid is checked whether it is full. If a full row is detected, a signal is sent to a programmable chip which cycles through each row above the full row. The values of the row are stored, and then written to the row below it in order to move all the pieces down. Eight octal flip flops store all values of the grid which is displayed to the user. Each flip flop uses inputs of both SRAM chips through OR gates in order to write column values and a demultiplexer selects the chip containing the corresponding row being written to. An 8x8 array of LEDs is soldered onto a perfboard and connected to the corresponding values of the octal flip flops.

Programmable Chip

For our project we implemented the use of a PLD (Programmable Logic Device), specifically the GAL 22V10 chip. PLDs are chips that can be connected to a computer and programmed via various languages, the most popular being the language CUPL that, on Windows computers, is called WinCupl and is by the Atmel company. Each pin in the chip is

assigned either an output or an input value. Various logical programming can be done using the syntax and tools described in the WinCupl user's manual.

We implemented the GAL 22V10 chip by using it with the most complicated part of our project: outputting values of the Tetris grid to the SRAM chips and rotating the Tetris pieces. The chip takes inputs for the row and column location of a piece, the rotation state of that piece, and the type of Tetris piece that the chip was addressing, then returns in a sequence the address of each row and each of the 8 cells on that rows value as 8 individual bits. There are a significant amount of conditions to consider when outputting the Tetris grid through the chip, and we ran into problems trying to address all those conditions while satisfying the limited capacity of the functions the chip can do. Additionally, we needed to address the issue of having more outputs than a GAL 22V10 can output (3 bits for the row number and 8 for the cell is more than the 10 bits allotted for output). In order to address this problem, we decided to use more than one programmable chip.

Inventory

Component	Number	Quantity	Functionality
Push Button	N/A	4	Movement Rotation
Timer	555	7	Button Debounce Input storage clock Falling piece Programmable Array
D Flip-flop	4013	4	Stores user input
Quad 2-input OR	4071	5	Combine falling piece with static piece grid Input circuit Collision
Quad 2-input XOR	4070	1	Random generator
Hex Buffer	4503	5	RAM I/O
SRAM 2kx8	6116LP-3	2	Storage of falling piece grid Storage of static piece grid
Quad 2-input AND	4081	5	Input circuit Collision Full row detection
Dual 4-input AND	4082	1	Collision
Dual 4-input OR	4072	2	Input circuit
Dual 4-input NOR	4002	1	Fall collision
Octal D Flip-flop	74LS273	10	Buffer to LED output Line clear memory management Random generator
Hex Inverter	4069	1	Write enable
4-bit counter	4516	2	Rotation and movement
8-bit binary decoder	4514	1	Output
Programmable Array	GAL22v10	3	Memory management Piece rotation
LED	N/A	65	Output Grid Pause
Breadboard	N/A	8	N/A

Issues Encountered

- Rotation of Tetris piece made it impossible to generate grid outputs in a feasible manner using only logic gates
- Soldering of LEDS onto perfboard caused several bad contacts due to small amount of space between LEDs
- Input/output and memory management of SRAM chip to store grid had to be managed nearly instantaneously using timers and programmable chips
- Programmable chip program was unable to compile

Reason for Failure

Our project failed to work due to overestimation of the capabilities of the programmable chip and WinCUPL. The number of possible Tetris piece inputs resulted in the creation of a WinCUPL program containing several hundreds of thousand lines of code. However, the logical gate requirements for the GAL22V10 chip were met. The program compiled for an unreasonably long amount of time before WinCUPL itself crashed. A test program containing an eighth of our required code compiled successfully in several seconds. It seems that there exists a limit as to the length of a CUPL program which the compiler can process. Adding code to the program increased compile time nearly exponentially until the result was an extremely long compile time during which WinCUPL crashes.

Logic Diagrams

Logic Diagrams (cont.)

Schematics

Collision
GAL22V10
GAL22V10

Collision
HM6116LP-3
HM6116LP-3
GAL22V10

Schematics (cont.)

Board Diagram