THE COOPER UNION
FOR THE ADVANCEMENT OF SCIENCE AND ART

A Practical Method for the Implementation of Cascading Tetrominoes with Digital Logic

Arnold Wey, EE '18

-&-

Camilo Andrés Gaitán Parra, CE '18

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Engineering

May 13, 2015

Professor Jared Harwayne-Gidansky
Thesis Advisor

TO DO

Define stuff, like Tetrominoes Introduction Pictures (Big, Labeled, Functional Blocks) ?State Diagrams? Labelling of Pictures PRNG? RNG? Investigate before using label Make note of Abbreviations (Remove them)

Abstract

The objective of the project is a digital logic port of the 1984 game Tetris, which involves the manipulation of a falling shape in a vertical board. Key features of gameplay are the random generation of falling Tetrominoes, the ability to move the falling shape (henceforth referred to as "Mino") left, right, and rotate it. Minos are written into the Field upon collision, and full lines are cleared to allow space for continued gameplay.

The project demonstrates the basic functionality of Tetris. Additional safeguards could be implemented but weren't defined in the project specs, such as wall collision detection, increased falling speed as the game progresses, and the implementation of a scoring system.

Acknowledgements

We cannot express enough thanks to the Electrical Engineering Faculty for their continued support and encouragement: Professor Fred Fontaine, Electrical Engineering Chair; Professor Toby Cumberbatch; Professor Sam Keene; Professor Carl Sable; and Professor Stuart Kirtman, along with Dino Melendez for being eternal fountains of knowledge and enlightenment. We offer our sincere appreciation for the learning opportunities and facilities provided by The Cooper Union.

Our completion of this project could not have been accomplished without the advice of the Electrical Engineering upperclassmen: Chris Curro, Howie Chen, Stephen Leone, Justin Alexander, Neema Aggarwal, and Venkat Kuruturi. Their help with WinCUPL, oscilloscopes, .bin file manipulation, and debugging was essential.

To Yuliya Koshkina thank you for allowing me time away from you to insert my wires into other holes. [Insert Romantic Shit, Camilo]

Thanks to our parents as well, Gwodonq Wey, Bihyueh Chen Wey, Orlando Fajarda, ý Rosemary Parra. The opportunities you gave us through your own sacrifices are boundless, along with our appreciation for everything you still do.

Finally, to our sarcastic, tyrannical, tough-love professor, Jared Harwayne-Gidansky: our deepest gratitude. Your encouragement when the times got rough are much appreciated and duly noted. It was a great comfort and relief to know that you care for our well-being and learning experience.

You have our heartfelt thanks.

Contents

1 Introduction

Tetris is a classic game which has seen implementation in almost every video game console and operating system, as well as on devices such as graphing calculators, mobile phones, portable media players, PDAs, Network music players and even as an Easter egg on non-media products like oscilloscopes. However, to our knowledge there has not been a successful implementation on Tetris in digital logic. Our objective is to port Tetris to digital logic, which involves the manipulation of a falling shape in a vertical board. Key features of gameplay are the random generation of falling Tetrominoes, the ability to move the falling shape (henceforth referred to as "Mino") left, right, and rotate it. Minos are written into the Field upon collision, and full lines are cleared to allow space for continued gameplay.

The project demonstrates the basic functionality of Tetris. Additional safeguards could be implemented but weren't defined in the project specs, such as wall collision detection, increased falling speed as the game progresses, and the implementation of a scoring system.

2 Design Considerations

2.1 Overview

Tetrominoes are hard coded into an EEPROM, which is addressed by counters controlled by buttons. Random shapes are generated by a 3-bit PRNG.

The Field is rendered row-by-row from the bottom up, while processing each row and checking for a full row, an empty row, and selecting either the current row or the next row to write into RAM. The bottom row of the mino, RowBottom, corresponds to its location in the board and is compared with the current row being processed. This comparison, carried out by Overlap, determines whether to enable the output of the corresponding row from the EEPROM. RowBottom is maintained by a 3-bit down counter.

The output is displayed by sinking current on 1 row of the LED matrix at a time, while sourcing power to the appropriate "x" ordinates. This happens at a high frequency; the frame looks whole via persistence of vision.

2.2 Trimming

These solutions were devised through an iterative process which slowly decreased the number of boards required to implement Tetris. The original design required over 30 boards, which has slowly been trimmed down to around 14.

Tetris has been traditionally implemented on a 10x20 board, which was an awkward number of inputs to MUX, and required an extra chip for every part of the circuit that used MUXs, as well as 3 4-bit shift registers instead of 2.

This was compounded by the original implementation to display the board, where the Tetromino would be loaded into a RAM separate from the RAM storing the FIELD. The outputs from the two RAMs would be time- domain MUXed to individual rows of LEDs.

Then, we considered loading each shape into 4 8-bit Universal Shift Registers (8 4-bit SRs), which could be manipulated and fed into rotation matrices in order to manipulate the output. While building, we realized that hard-coding the possible states of each row containing a Tetromino could fit inside the EEPROM, completely eliminating the need for the 4 shift registers storing each row that would handle shifting left and right, not to mention save the cost of Universal Shift Registers, which are quite costly.
This brought the total board count to around 18, but required writing 1024 unique rows

The need to manually write 1024 bytes prompted an investigation in automating the process. There's no simple way to handle rotation, so $4\,rows * 8\,shapes * 4\,orientations = 128\,rows$ were written manually into input.txt. "shapeTest.c" prints out all the possible offsets of each shape's orientations, read from input.txt. The stdout from shapeTest is redirected into Swag.txt. The debugging output is manually cleaned up, and used as input to bintext2bin.
bintext2bin.c converts input from a text file containing 8bit rows encoded as ASCII 1's and 0's into a bin file with equivalent information. (output.bin)
This bin file was loaded into the memory buffer at even addresses, for pinning convenience. Additionally, this allows the LSB on the EEPROM to be used as a logical disable.

Asserting 1 row at a time from the EEPROM and the RAM allows us to do away with another set of MUXs, and performing a bit-wise OR to control the display.

After cutting down the number of boards required to control the shape, we needed to reexamine the logic handling the RAM.

In order to access and manipulate information within the RAM, they must be stored and displayed on Shift Registers. In addition the RAM needs to be addressed to both read and write. This sequential nature made the 4017-Decade Counter an obvious choice.

A lot of sequential logic processing requires enabling different circuits at the same frame in a timeline, depending on different conditions. This is accomplished through the use of "flags," which are flip flops which store a certain condition (Full Row, Empty Row, etc.)

The use of flags greatly simplified shifting different rows and selecting between inputs to MUXs in general. For example, the outputs from Shift_Register_Two could go to an 8 input NOR, which is HIGH when Shift_Register_Two is empty. When this Flip Flop is high,the Shift-Down MUX will select the output from Shift_Register_One, which stores the "next row" in the RAM.

Writing back into the RAM requires having read and write information asserted on the same bus. This requires the use of a Tri-State enabled MUX.

A few more boards were saved by implementing Truth Table 1 on page 17 into a GAL chip, Overlap.

That Truth Table simplifies into Truth Table **??** on page **??**, and results in Equation /eqref: This had few enough product terms to fit onto a GAL16v8, and upon securing approval, we programmed the chip using WinCUPL and the ChipMaster 6000 graciously provided to us by the Cooper Union Electrical Engineering Department.

The Overlap.PLD source code, followed by snippets of the test vector file, can be found on page 6.

3 Final Implementation

*Note: the following discussion will include C-Style pointer notation, where *address refers to the value stored at the given address, and &value refers to the address where value resides.*

The current "State" is stored in a shift register at the start of each row processing cycle. This state represents the address of the RAM which will be written into. 1st, the shift registers are clocked in such a way as to store *State and *(State+1) into two shift registers. The exact order of clocking is included on page 20.

[WIP]

4 Equations

4.1 Logic Equations

4.1.1 Overlap

$$\begin{aligned} Overlap = &(B_2 + B_3 + B_4 + !A_2) * (B_2 + B_3 + !A_2 + !A_4) * (B_2 + B_3 + !A_2 + !A_3) * \\ &(B_2 + B_4 + !A_2 + !A_3) * (B_2 + !A_2 + !A_3 + !A_4) * (!B_1 + A_1) * (!B_2 + !B_4 + !A_2 + A_3 + A_4) * \\ &(B_1 + !A_1 + !A_2) * (!B_2 + A_1 + A_2) * (B_1 + B_2 + !A_1) * (B_2 + !B_3 + A_2 + A_3) * \\ &(!B_2 + B_3 + B_4 + A_2) * (!B_2 + A_2 + !A_3 + !A_4) * (!B_2 + !B_3 + !A_2 + A_3) * \\ &(B_2 + !B_3 + !B_4 + A_2 + A_4) * (B_2 + !B_4 + A_2 + A_3 + A_4) * (!B_2 + B_3 + A_2 + !A_3) * \\ &(!B_2 + B_3 + A_2 + !A_4) * (!B_2 + B_4 + A_2 + !A_3) * (!B_2 + !B_3 + !B_4 + !A_2 + A_4) * \end{aligned}$$

4.1.2 4 Bit Select

$$Y_n = (A_n \,\&\, !ADD) \; + \; (B_n \,\&\, ADD)$$

4.1.3 8AndOr

$$AND = \sum_{i=0}^{7} I_i$$

$$OR = \prod_{i=0}^{7} I_i$$

4.2 Calculations

$$\begin{aligned} Min\,Frequency_{Persistance\,Of\,Vision} &\equiv & 50\,Hz \\ 50\frac{Hz}{row} * 16rows &= & 800\,Hz \\ F_{min\,Timer} * \frac{1}{10} &= & 800Hz \\ F_{timer} &> & 8kHz \\ \frac{1.44}{C(R_a + 2R_b)} &> & 8\,kHz \end{aligned}$$

Decade Timer Frequency Calculations

$$\begin{aligned} \frac{1.44}{C(R_a + 2R_b)} &= & Frequency \\ R_a &= & 550\,\Omega \\ R_b &= & 810\,\Omega \\ C &= & .015\,\mu F \\ Frequency &= & 442\,kHz \end{aligned}$$

Note: The 555 is producing $\sim 38.5\,kHz$*, as measured by an oscilloscope.*

Row Falling Frequency Calculations

$$\begin{aligned} Frequency_{fall} &= & \frac{1}{3}\,Hz \\ \frac{1.44}{C(R_a + 2R_b)} &= & Frequency \\ R_a &= & 470\,k\Omega \\ R_b &= & 470\,k\Omega \\ C &= & 3.3\,F \\ Frequency &= & 0.310\,Hz \end{aligned}$$

Note: The 555 is producing $\sim 0.33\,kHz$*, as measured by an oscilloscope.*

5 Sample Code

5.1 WinCUPL

Overlap.PLD

```
Name    Overlap;
Partno  01;
Date    4/24/2015;
Rev     01;
Designer    Arnold Wey;
Company     CU Later;
Assembly    None;
Location    None;
Device  g16v8;

/**Inputs**/
Pin 1 = A1;
Pin 2 = A2;
Pin 3 = A3;
Pin 4 = A4;
Pin 5 = B1;
Pin 6 = B2;
Pin 7 = B3;
Pin 8 = B4;

/**Outputs**/
Pin 15 = I1;
Pin 16 = I2;
Pin 17 = I3;
Pin 18 = I4;
Pin 14 = Overlap;
Pin 13 = NotOverlap;

O0 = B2  # B3   # B4   # !A2 ;
O1 = B2  # B3   # !A2  # !A4 ;
O2 = B2  # B3   # !A2  # !A3 ;
O3 = B2  # B4   # !A2  # !A3 ;
O4 = B2  # !A2  # !A3  # !A4 ;
O5 = !B1 # A1  ;
O6 = B1  # !A1  # !A2 ;
O7 = !B2 # A1   # A2  ;
O8 = B1  # B2   # !A1 ;
O9 = B2  # !B3  # A2   # A3  ;
O10 = !B2 # B3  # B4   # A2  ;
O11 = !B2 # A2  # !A3  # !A4 ;
O12 = !B2 # !B3 # !A2  # A3  ;
O13 = B2 # !B3  # !B4  # A2   # A4  ;
O14 = B2 # !B4  # A2   # A3   # A4  ;
O15 = !B2 # B3  # A2   # !A3 ;
O16 = !B2 # B3  # A2   # !A4 ;
O17 = !B2 # B4  # A2   # !A3 ;
O18 = !B2 # !B3 # !B4  # !A2  # A4  ;
O19 = !B2 # !B4 # !A2  # A3   # A4  ;

/*Combining Terms*/
I1 = [O0, O1, O2, O3, O4, O10,O15]:&;
I2 = [O5, O6, O7, O8, O9]:&;
I3 = [O11,O13]:&;
```

```
I4 = [O16,O17,O18,O19]:&;

/*Final Terms*/
Overlap = [I1, I2, I3, I4,O14,O12]:&;
NotOverlap = !Overlap;
```

Overlap.SI

```
Name       Overlap;
PartNo     01;
Date       4/24/2015;
Revision   01;
Designer   Arnold Wey;
Company    CU Later;
Assembly   None;
Location   None;
Device     g16v8;

ORDER: B1, B2, B3, B4, A1, A2, A3, A4, Overlap, NotOverlap;

VECTORS:
00000000HL
00000001HL
00000010HL
00000011HL
00000100LH
00000101LH
00000110LH
00000111LH
00001000LH
   ...
```

8AndOr.PLD

```
Name 8AndOr;
Partno 01;
Date 4/15/2015;
Rev 01;
Designer Arnold Wey;
Company CU Later;
Assembly None;
Location None;
Device g16v8;
/**Inputs**/

Pin 5 = I7;
Pin 6 = I6;
Pin 7 = I5;
Pin 8 = I4;
Pin 9 = I3;
Pin 11 = I0;
Pin 12 = I1;
Pin 13 = I2;

/**Outputs**/
Pin 14 = O4;
Pin 15 = O5;
Pin 16 = O6;
Pin 17 = O7;
Pin 18 = OR;
Pin 19 = AND;
AND = [I7, I6, I5, I4, I3, I0, I1, I2]:&;
OR = [I7, I6, I5, I4, I3, I0, I1, I2]:#;
O4 = I4;
O5 = I5;
O6 = I6;
O7 = I7;
```

8AndNor.SI

```
Name     8AndNor;
PartNo   01;
Date     4/15/2015;
<Revision></Revision>;
Designer Arnold Wey;
Company CU Later;
Assembly None;
Location None;
Device   g16v8;


ORDER: I7, I6, I5, I4, I3, I0, I1, I2, AND, OR;


VECTORS:
00000000LL
00000001LH
00000010LH
00000011LH
00000100LH
00000101LH
00000110LH
00000111LH
00001000LH
00001001LH
00001010LH
00001011LH
00001100LH
   ...
```

4BSel.PLD

```
Name 4bSel;
Partno 01;
Date 4/15/2015;
Rev 01;
Designer Arnold Wey;
Company CU Later;
Assembly None;
Location None;
Device g22v10;
/**Inputs**/
Pin 1 = ADD;

Pin 4 = A0;
Pin 5 = A1;
Pin 6 = A2;
Pin 7 = A3;
Pin 8 = B0;
Pin 9 = B1;
Pin 10 = B2;
Pin 11 = B3;
/**Outputs**/

Pin 18 = Y3;
Pin 19 = Y2;
Pin 20 = Y1;
Pin 21 = Y0;

Y0 = A0 & !ADD;
APPEND Y0 = B0 & ADD;
Y1 = A1 & !ADD;
APPEND Y1 = B1 & ADD;
Y2 = A2 & !ADD;
APPEND Y2 = B2 & ADD;
Y3 = A3 & !ADD;
APPEND Y3 = B3 & ADD;
```

4BSel.SI

```
Name     4bSel;
PartNo   01;
Date     4/15/2015;
<Revision></Revision>;
Designer Arnold Wey;
Company CU Later;
Assembly None;
Location None;
Device   g22v10;


ORDER: A0, ADD, B0, A1, B1, A2, B2, A3, B3, Y0, Y1, Y2, Y3;


VECTORS:
000000000LLLL
000000001LLLL
000000010LLLH
000000011LLLH
000000100LLLL
000000101LLLL
000000110LLLH
000000111LLLH
000001000LLHL
000001001LLHL
000001010LLHH
000001011LLHH
000001100LLHL
000001101LLHL
000001110LLHH
000001111LLHH
```

5.2 regEx

Bill Of Materials RegEx

```
"([0-9 a-z A-Z \-]{0,30})";"(SO|DIL)([a-z 0-9 A-Z \/ ]{0,50})";"([0-9 a-z A-Z \- \/ \,
    \_]{0,1000})";
```

5.3 C Source

skewGen.c

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

#include "tetris.h"
#include "BoolStr.h"

#define BUFSIZE 256

main() {
    char   buf[BUFSIZE];
    int uc;
    int i, len, linenum, numwritten, numblank;
    FILE   *fpin, *fpout;
    const char *infile, *outfile;

    infile = "input.txt"; // Name of the input (text) file
    outfile = "output.txt"; // Name out the output (text) file

    // Open in text mode
    if( (fpin = fopen(infile, "r")) == NULL ){
        printf("Cannot open input file '%s'\n", infile);
        exit(1);
    }

    if( (fpout = fopen(outfile, "wb")) == NULL ){ // Open in binary mode
        printf("Cannot open output file '%s'\n", outfile);
        exit(1);
    }

    shape swag[7];
    swag[0].name = 'I';
    swag[1].name = 'J';
    swag[2].name = 'L';
    swag[3].name = 'Z';
    swag[4].name = 'S';
    swag[5].name = 'T';
    swag[6].name = 'O';

    for (int i = 0; i < 7; ++i){
        swag[i].orientation = 4;
        swag[i].row = 4;
        swag[i].col = 8;
    }

    linenum = 0;
    numwritten = 0;
    numblank = 0;
    int sha = 0;
    int ori = 0;
    int row = 0;

    while( fgets(buf, BUFSIZE, fpin) != NULL ){
        linenum++;
```

```
    // If last char is not newline then may not have full line
    len = strlen(buf);
    if( buf[len - 1] != '\n' ){
        printf("Did not read full line at LINE #%d ('%s')\n",
            linenum, buf);
        exit(1);
    }
    buf[len - 1] = '\0';  // Get rid of newline
    len--;         // Adjust length

    // Skip blank lines
    if( len == 0 ){
        printf("Skipping blank line at LINE #%d\n", linenum);
        numblank++;
        continue;
    }

    // Make sure number of chars in line is correct
    if( len != 8 ){
        printf("Line wrong length at LINE #%d ('%s')\n",
            linenum, buf);
        exit(1);
    }

    //convert buf into array of bools
    for (int i = 0; i < len; ++i){
        if (buf[i] == '0')
            swag[sha].up[ori][row][i] = 0;
        else if (buf[i] == '1')
            swag[sha].up[ori][row][i] = 1;
        else
            printf("Invalid arguement at LINE #%d ('%s')\n", linenum, buf );
    }

    //Stick buf into array as array of bools in the right order
    row++;
    if (row > 3){
        row = 0;
        ori++;
    }
    if (ori >3){
        ori = 0;
        sha ++;
    }
    numwritten++;
}

// Close files
fclose(fpout);
fclose(fpin);

printf("\nDone.\n");

for (int sh = 0; sh < 7; ++sh){
    p3bs(swag[sh].up);
    for (int i = 0; i < 7; ++i){
        s3bs(swag[sh].up);
        p3bs(swag[sh].up);
    }
```

```
    }
    printf("number of rows written : %d\n" , numwritten);
}
```

bintext2binary.c, Courtesy of Stuart Kirtman

```
// Program to convert lines of "binary" chars in text file to
// bytes in a binary file.
//
// SEK 4/13/2015
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFSIZE 256

int main()
{
    char   buf[BUFSIZE];
    int uc;
    int i, len, linenum, numwritten, numblank;
    FILE   *fpin, *fpout;
    const char *infile, *outfile;

    // Change these as needed
    infile = "Swag.txt";  // Name of the input (text) file
    outfile = "output.bin"; // Name out the output (binary) file

    if( (fpin = fopen(infile, "r")) == NULL ) // Open in text mode
    {
        printf("Cannot open input file '%s'\n", infile);
        exit(1);
    }

    if( (fpout = fopen(outfile, "wb")) == NULL ) // Open in binary mode
    {
        printf("Cannot open output file '%s'\n", outfile);
        exit(1);
    }

    linenum = 0;
    numwritten = 0;
    numblank = 0;

    // Process each line in the input file
    while( fgets(buf, BUFSIZE, fpin) != NULL )
    {
        linenum++;

        // If last char is not newline then may not have full line
        len = strlen(buf);
        if( buf[len - 1] != '\n' )
        {
            printf("Did not read full line at LINE #%d ('%s')\n",
                linenum, buf);
            exit(1);
```

```
        }
        buf[len - 1] = '\0';  // Get rid of newline
        len--;         // Adjust length

        if( len == 0 ) // Skip blank lines
        {
            printf("Skipping blank line at LINE #%d\n",
                linenum);
            numblank++;
            continue;
        }

        if( len != 8 ) // Make sure number of chars in line is correct
        {
            printf("Line wrong length at LINE #%d ('%s')\n",
                linenum, buf);
            exit(1);
        }

        uc = 0;
        for(i=0; i < len; i++)
        {
            uc = uc << 1;
            if( buf[i] == '0' ) // 0 already shifted in
                ;
            else if( buf[i] == '1' )
                uc |= 1;   // Make lsb 1
            else   // Error if char is not a '0' or a '1'
            {
                printf("Invalid character at LINE #%d ('%s')\n",
                    linenum, buf);
                exit(1);
            }
        }

        fputc(uc, fpout); // Write byte to (binary) output file
        numwritten++;

        // Inform user of progress
        printf("Line #%d: 'Writing '%s' as '%02X'\n", linenum, buf, uc);
    }

    // Close files
    fclose(fpout);
    fclose(fpin);

    printf("\n----------------------\n");
    printf("\nDone.\n");
    printf("Number of bytes written to '%s': %d\n", outfile, numwritten);
    printf("Number of blank lines: %d\n", numblank);

    exit(0);
}
```

6 Tables

Table 1: Overlap Unminimized, Abbreviated

B1	B2	B3	B4	A1	A2	A3	A4	Overlap	!Overlap
0	0	0	0	0	0	0	0	H	L
0	0	0	0	0	0	0	1	H	L
0	0	0	0	0	0	1	0	H	L
0	0	0	0	0	0	1	1	H	L
0	0	0	0	0	1	0	0	L	H
0	0	0	0	0	1	0	1	L	H
0	0	0	0	0	1	1	0	L	H
0	0	0	0	0	1	1	1	L	H
0	0	0	0	1	0	0	0	L	H
0	0	0	0	1	0	0	1	L	H
0	0	0	0	1	0	1	0	L	H
0	0	0	0	1	0	1	1	L	H
0	0	0	0	1	1	0	0	L	H
0	0	0	0	1	1	0	1	L	H
0	0	0	0	1	1	1	0	L	H
0	0	0	0	1	1	1	1	L	H
0	0	0	1	0	0	0	0	L	H
0	0	0	1	0	0	0	1	H	L
0	0	0	1	0	0	1	0	H	L
0	0	0	1	0	0	1	1	H	L
0	0	0	1	0	1	0	0	H	L
0	0	0	1	0	1	0	1	L	H
0	0	0	1	0	1	1	0	L	H
...	...	...	...	...	...	...	...	...	...

Table 2: Overlap Minimized

A1	A2	A3	A4	B1	B2	B3	B4	Overlap
1	1	0	0	1	1	X	X	1
0	1	0	0	0	1	X	X	1
1	0	0	0	1	0	X	X	1
0	0	0	0	0	0	X	X	1
1	1	0	X	1	1	1	X	1
1	1	X	0	1	1	1	X	1
0	1	X	0	0	1	1	X	1
1	0	0	X	1	0	1	X	1
1	0	X	0	1	0	1	X	1
0	0	0	X	0	0	1	X	1
0	0	X	0	0	0	1	X	1
1	0	1	X	1	1	0	X	1
0	0	1	X	0	1	0	X	1
1	1	0	X	1	1	X	1	1
1	0	0	X	1	0	X	1	1
0	0	0	X	0	0	X	1	1
1	1	X	X	1	1	1	1	1
1	0	X	X	1	0	1	1	1
0	X	1	1	0	1	X	0	1
0	1	1	0	1	0	0	X	1
0	X	1	1	0	0	1	1	1
1	0	1	1	1	1	X	0	1
1	0	X	1	1	1	0	0	1
0	0	X	1	0	1	0	0	1
0	1	0	1	1	0	0	0	1
0	1	0	X	0	1	1	X	1
0	1	X	1	0	1	0	1	1

Table 3: 4BSel Unminimized, Abbreviated:

A0	ADD	B0	A1	B1	A2	B2	A3	B3	Y0	Y1	Y2	Y3
0	0	0	0	0	0	0	0	0	L	L	L	L
0	0	0	0	0	0	0	0	1	L	L	L	L
0	0	0	0	0	0	0	1	0	L	L	L	H
0	0	0	0	0	0	0	1	1	L	L	L	H
0	0	0	0	0	0	1	0	0	L	L	L	L
0	0	0	0	0	0	1	0	1	L	L	L	L
0	0	0	0	0	0	1	1	0	L	L	L	H
0	0	0	0	0	0	1	1	1	L	L	L	H
0	0	0	0	0	1	0	0	0	L	L	H	L
0	0	0	0	0	1	0	0	1	L	L	H	L
0	0	0	0	0	1	0	1	0	L	L	H	H
...	...	...	...	...	...	...	...	...	...	...	...	...

Table 4: 8 Input And & Or, Abbreviated:

I7	I6	I5	I4	I3	I0	I1	I2	AND	OR
0	0	0	0	0	0	0	0	L	L
0	0	0	0	0	0	0	1	L	H
0	0	0	0	0	0	1	0	L	H
0	0	0	0	0	0	1	1	L	H
0	0	0	0	0	1	0	0	L	H
0	0	0	0	0	1	0	1	L	H
...	...	...	...	...	...	...	...	...	...
1	1	1	1	1	1	1	0	L	H
1	1	1	1	1	1	1	1	H	H

Table 5: Bill Of Materials

Qty	Value	Description
1	4001D	Quad 2-input NOR
1	4002D	4-input NOR
1	4017D	COUNTER/DIVIDER
5	4027D	Dual JK FLIP FLOP
3	4029D	Binary/decimal up/down COUNTER
8	4035D	4-bit parallel in/out SHIFT REGISTER
2	4048D	Expandable 8-input GATE
10	4053D	Triple 2-channel ANALOG MULTIPLEXER
1	4069D	Hex INVERTER
4	4071D	Quad 2-input OR
9	4081D	Quad 2-input AND
3	4520N	Dual binary up COUNTER
2	ATF16V8BS	CMOS PLD
1	CY62256LL-SNC	256K (32K x 8) CMOS-Static RAM
3	LM555N	TIMER
1	2816	MEMORY

Table 6: Decade Counter Sequence

Step	Action
1	SR_{State}
2	SR_1
3	SR_2
4	$RAMAddress$
5	SR_1
6	SR_{Return}
7	Set $FF_{R/W}$
8	Reset $FF_{R/W}$
9	Row_{Mino}
10	Set $FF_{Collision}$

7 Figures

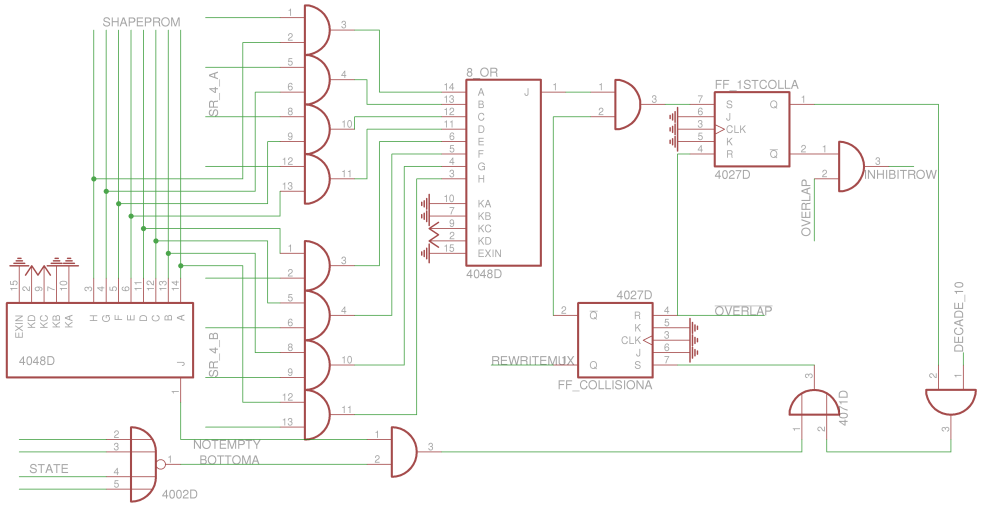

Figure 1: Collision Logic

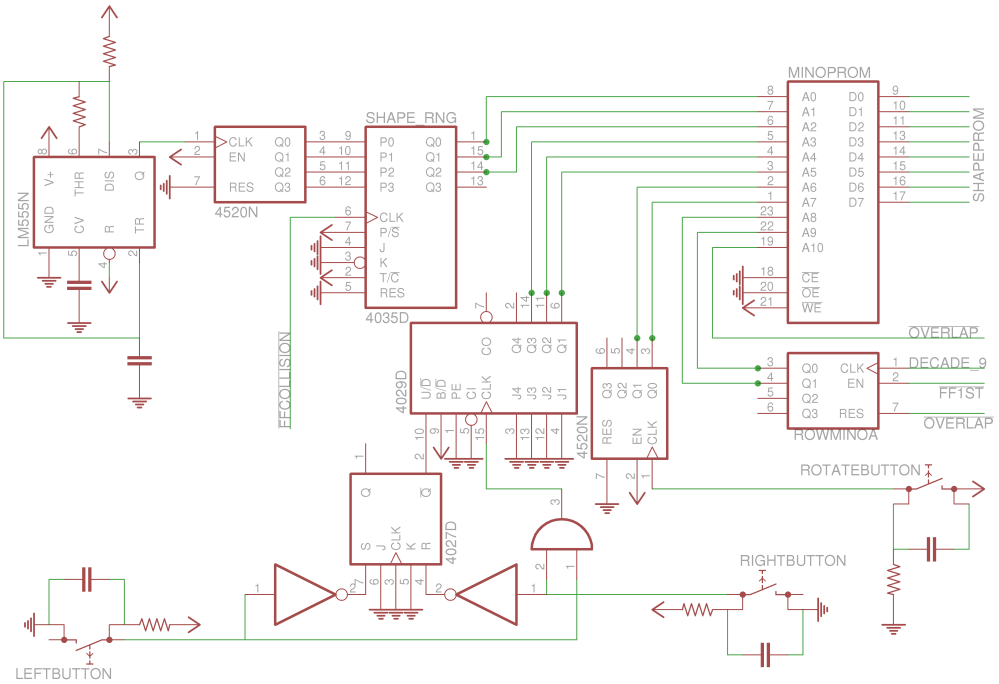

Figure 2: Mino Control Logic

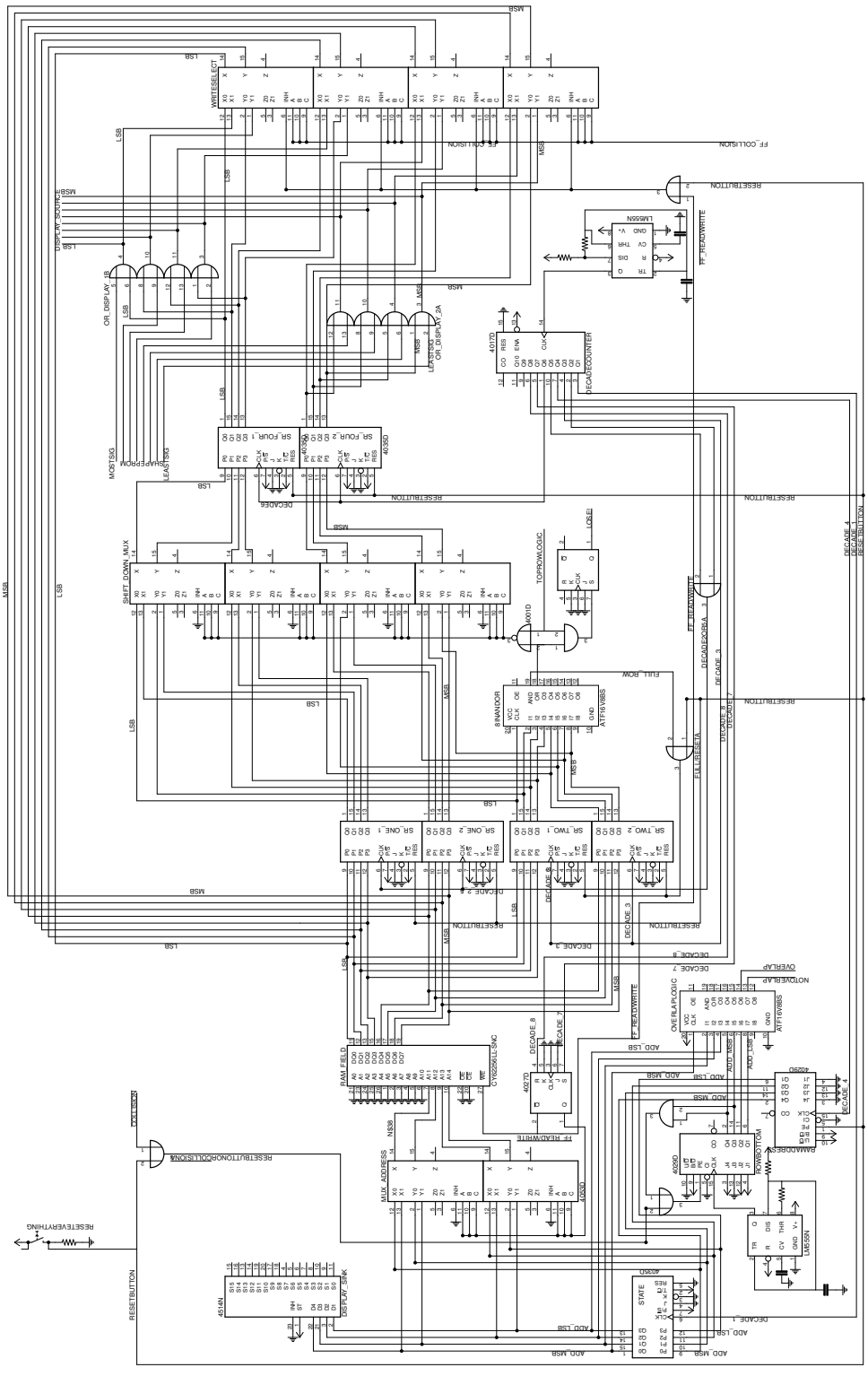

Figure 3: Display Logic

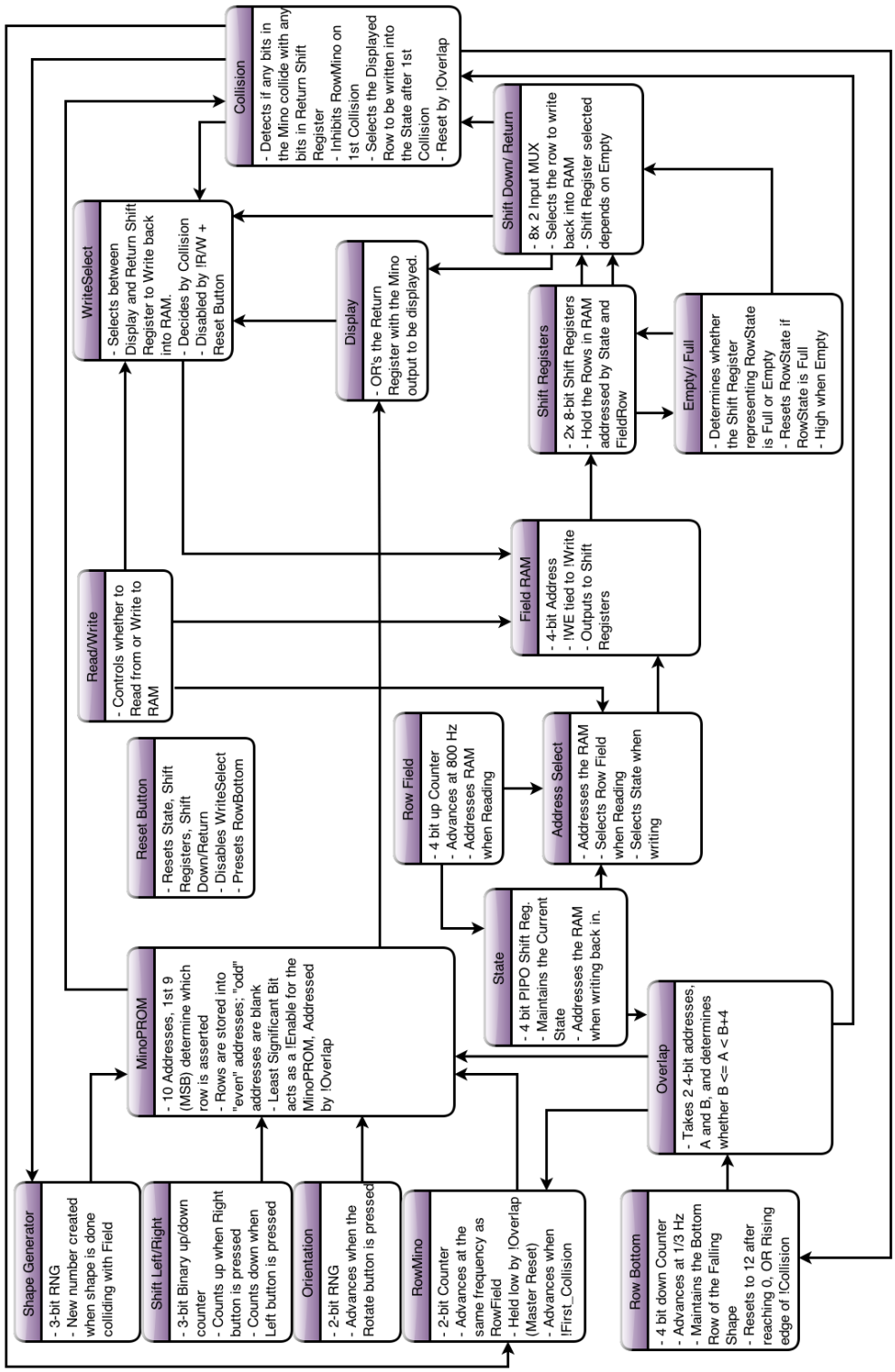

Figure 4: Functional Block Diagram