Tufts University

School of Engineering
Department of Electrical and Computer Engineering

ES4 - Introduction to Digital Circuits
Spring 2012

Final Project
Assignment 3

Names: Michael Abboud
William Stone
Sam Zeckendorf
Jesse Zhang

Turned in: 04/30/12

Table of Contents

I. Project Specifications

The goal of this project is to construct the "Angry Birds" game. The user will use two buttons to control the "player," an orange LED in the bottom row of an 8x8 LED grid and attempt to dodge incoming projectiles. The "projectiles," represented by green LEDs, will move down the rest of the grid one row per clock cycle. The data for the incoming projectiles will be stored on an EPROM, which will be loaded and interpreted at runtime. Before the game begins, the LED grid will display the player's position and an initial configuration of non-moving projectiles. The game will start after a third button is pressed, and the projectiles will start to move downwards towards the player's LED. Collision detection will involve comparing the position of the player's LED with the state of the LED directly above it. If no projectile-player collision is detected, the score (displayed on two 7-segment displays) will increase by 1. If a collision between the player and a projectile is detected, then the game will end, and the memory will reset. The projectiles will freeze in place. The score will remain on the displays until the player begins a new game.

II. Inputs

The game will have three physical inputs: two user controls and one game control. The two user controls will move the player one LED position to the left or the right, and the game control will start or reset the game, depending on the current game state. In addition, another input will be the EPROM unit, which will contain the data for the projectiles. The EPROM will be programmed beforehand and read in data for the projectiles when the game starts.

III. Outputs

The outputs of this game will consist of three displays: the 8x8 LED grid and the two 7-segment displays for the score. The LED grid will display the projectiles and the player, and the two 7-segment displays will show the score (up to a maximum of 99).

IV. Units

This circuitry for Angry Birds consists of six major units:

1. The user input unit will be the three buttons connected to the game. The left/right buttons will pulse once each time they are pressed. When the reset button is pressed, the counters will go to zero and the display will go to its initial state.
2. The master clock from a function generator will feed into a 14-bit ripple counter. The three least significant bits of the counter will be used as a fast clock (for refreshing the rows). The four most significant bits of the counter will be used as a slow clock (to pulse the counters and change board states).

3. The score keeper unit will increment the score once per 4-bit slow counter pulse depending on the current game state. The score keeper updates the score display consisting of the two 7-segment displays.
4. The row generator unit will load and generate rows of projectiles using an EPROM. The EPROM's row generator will output the current state using a 3-bit fast counter refreshes the output to the display. A 4-bit slow counter will select the board state to output.
5. The collision detector will store the 8-bit state of the bottom row of the game in an octal D flip-flop and check for collisions on the refresh of the character row. On detecting collisions, this unit signals the game counter to flash the score and the row generator to stop updating.
6. The game display unit will display the position of the projectiles and the player on the 8x8 LED array. This unit will receive data from the EPROM unit and the character row generator and interpret it for display.

A complete schematic for the Angry Birds high level design can be found in the Appendix (Figure 1).

Table 1. Inputs/outputs for each major component

Component	**Inputs**	**Outputs**
Row generator/Memory	1. 4-bit level select 2. 3-bit row refresh 3. 1-bit collision boolean	1. 8-bit row information to memory 2. 8-bit row information to collision detector
Character row generator	1. 1-bit pulse from left button input 2. 1-bit pulse from right button input	1. 8-bit decoded player position to LED display 2. 3-bit encoded player position to collision detector
Collision detector	1. 8-bit row information from memory 1. 8-bit row information from character row generator	1. 1-bit game over information to memory 1. 1-bit game over information to score keeper
Score keeper	1. 1-bit hold information from collision detector 2. 1-bit clock signal from function generator	2. 8-bit BCD score to BCD to 7-segment display converter
LED Display	1. 8-bit row information from memory 2. 8-bit decoded row information from character row generator	1. LEDs illuminate

V. User’s Manual

Button 1: Start the game. Alternatively, if the game is already running, this button will reset the game.
Button 2: Shifts the player one LED to the left. Holding the button will still result in one LED shift left.
Button 3: Shifts the player one LED to the right. Holding the button will still result in one LED shift right.
Instructions: Dodge the oncoming green projectiles! Control the red LED at the bottom of the grid using the two control buttons. If a projectile moves into the current position of the red LED, the game is over and the player LED turns orange. One point is earned for every row of projectiles dodged.

VI. Components

- 8x8 LED Display
- Buttons
- MB-108 Breadboard
- 330 Ω Resistor
- 10 μF Capacitor
- 2764 EPROM
- 74138 Decoder/Demultiplexer
- 74151 8x1 Multiplexer
- 74193 Synchronous 4-Bit Binary Counter with Dual Clock
- 14020 14-Bit Binary Counter
- 74221 Dual Non-Retriggerable Monostable Multivibrator with Reset
- 74273 Octal D Flip-Flop with Common Clock and Reset
- 7408 Quad 2-Input AND gate
- 7402 Quad 2-Input NOR gate
- 7432 Quad 2-Input OR gate
- 7404 Hex Inverter
- 7448 Device BCD to 7-Segment Display Converter

VII. Circuit Logic

Controlling the player: The player will be controlled through the use of two input buttons that move the player left and right on the bottom row of the display. This is achieved through the use of the three least significant bits of a 4-bit up/down binary counter to store the player position and a 3x8 decoder to decode the position into display data for the 8x8 LED array. The counter will increment when it receives a pulse from the “right” button, and will decrement when it receives a pulse from the “left” button. This is achieved through use of a multivibrator to turn inputs from the buttons into square pulses as clock inputs to the counter.

The block and wiring diagrams for the character row generator can be found in the Appendix (Figures 2, 4). The debouncing circuit for the left and right button inputs is given in Figure 3.

Detecting collisions: The collision detector will detect when inputs from the character row generator and the row generator collide. This is achieved through the use of an 8x1 multiplexer. The input lines of the multiplexer are outputs from an octal flip flop storing the bottom row of the display, and the select lines are the encoded 3-bit position of the player. The resulting output of the multiplexer is a boolean value where logic high represents a collision occurrence and logic low represents no collision.

The logic and wiring diagrams for the collision detector can be found in the Appendix (Figures 5, 6).

Memory: The memory will be contained within an EPROM unit. Three of the address lines (A_0 - A_2) cycle through the row information of the current board state, constantly refreshing the board display. Four of the address lines select which of the board states to cycle through. The three address lines for refreshing the board display will be fed by the outputs of a 3-bit fast clock, and the four address lines for selecting board states will be fed by a 4-bit slow clock. The output of this unit is an 8-bit row information value sent to the collision detector unit and to the 8x8 LED array.

The wiring diagrams for the EPROM is given in Figure 7. A VHDL simulation of 10 frames of the states of the 6^{th} and 7^{th} rows can be found in the Appendix.

Updating the score: Using two four-bit synchronous binary counters, the "One's" and "Ten's" radices of the score can each be represented as a BCD values. Each binary counter can be converted into a decade counter using the reset pin and an AND gate, and the clock of the “Ten’s” counter is wired to the reset of the “One’s” counter. The logic for the decade counter is as follows:

Table 6. Characteristic table for the decade counter. X represents don't-care conditions

Present State				**Next State**				**FF Inputs**			
A	B	C	D	A(t+1)	B(t+1)	C(t+1)	D(t+1)	A	B	C	D
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	***0***	***0***	***1***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***
1	0	1	0	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

As is visible from the characteristic table, at present state $(9)_{10} = (1001)_2$ or $(1001)_{BCD}$, the next state should loop back to $(0)_{10} = (0000)_2 = (0000)_{BCD}$. As such, the table is populated with don't-care conditions after this value occurs. Resetting the 74193 4-bit synchronous counter is achieved by connecting pin 12 to the value of A*C $(1010)_2$. This is because AC is only true at $(1010)_2$ and beyond, and every value greater than $(1010)_2$ is a don't-care condition. These BCD values can be inputted directly to the 7448 BCD to Seven Segment Display converter chip, which in turn connects directly to the Seven Segment Display.

If the game has detected a collision, the clock input to the score will stop pulsing. Instead, using the 4-bit slow clock, the BCD to Seven Segment Display converter will cause the displays to flash the score, alerting the player of the game-over state.

The block, logic, and wiring diagrams for the score keeper can be found in the Appendix (Figures 8, 9, 10). The VHDL simulation of the mod-10 counter used can also be found in the Appendix.

Resetting the game: The start/reset button will pulse a voltage high to the master reset inputs at the row generator, the display, and the score counter. They all will be connected to the reset button, so upon pulsing the reset button, the game restarts at the initial values.

Appendix

Figure 1. High-level design for Angry Birds

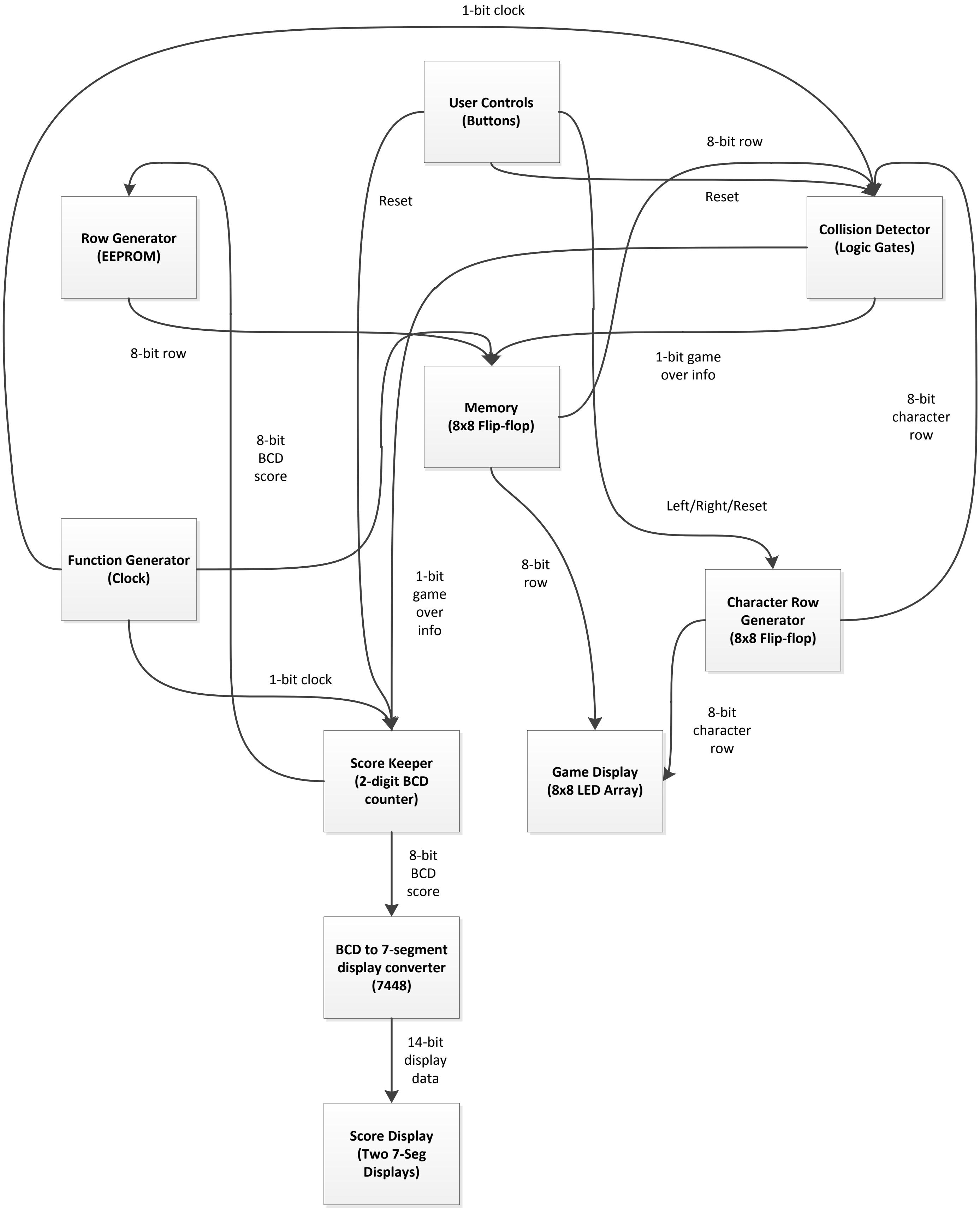

Figure 2. Character row generator block diagram

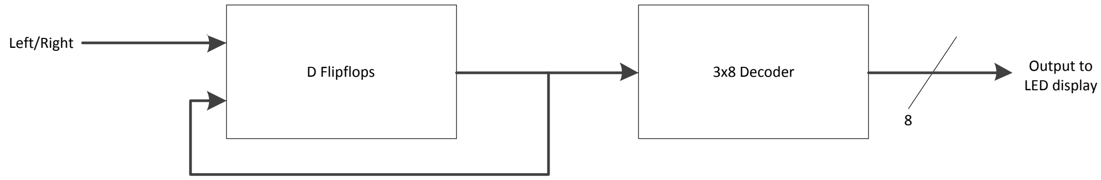

Figure 3. Character row generator logic diagram

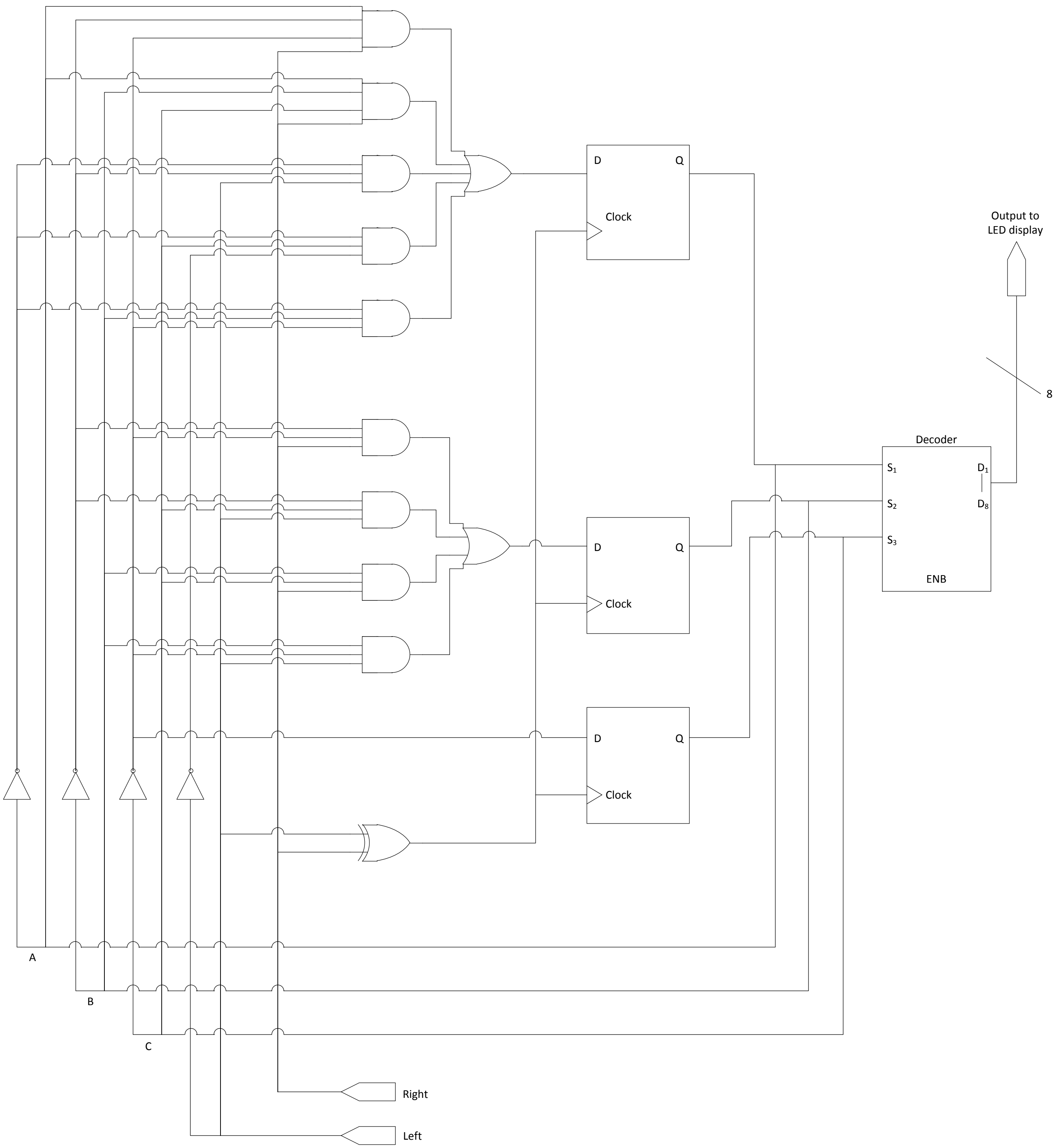

Figure 4. Collision detector block diagram

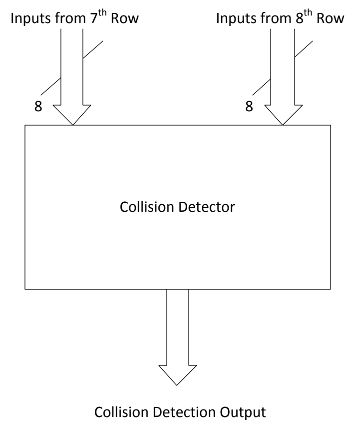

Figure 5. Collision detector logic diagram

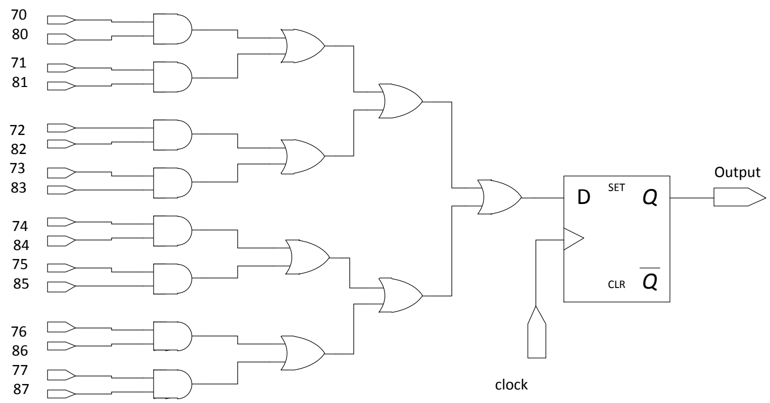

Figure 6. Collision detector wiring diagram

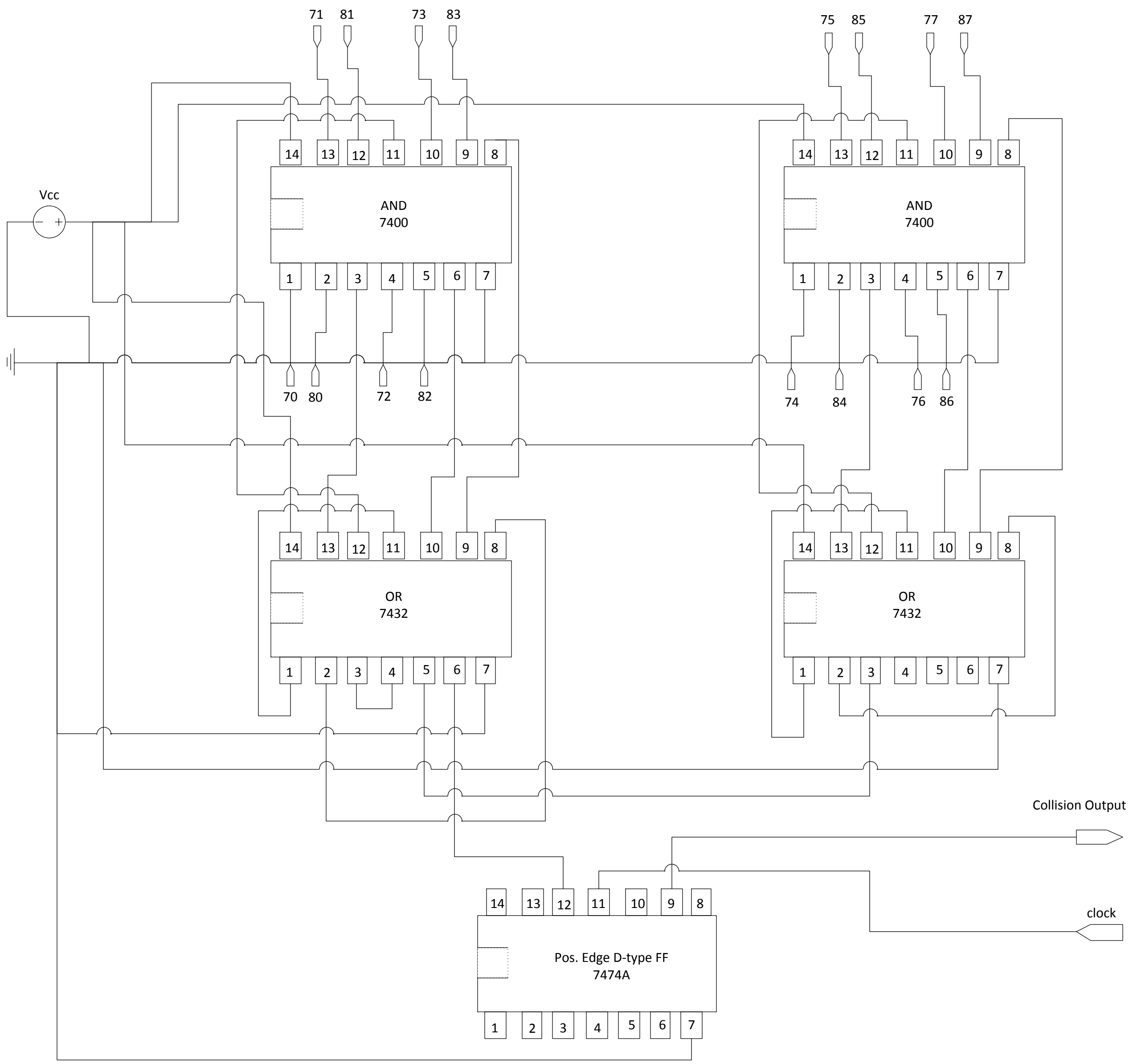

Figure 7. Row generator block diagram

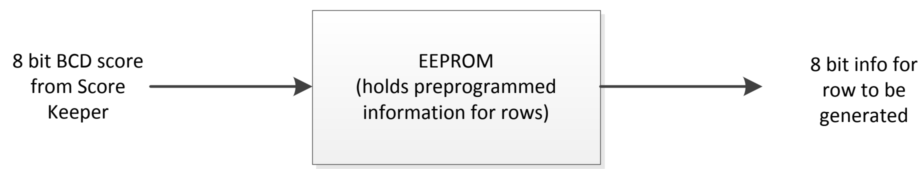

Figure 8. Memory block diagram

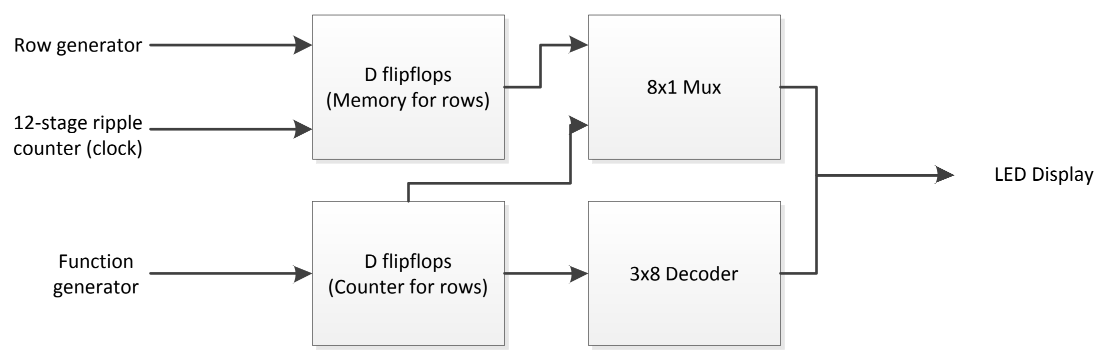

Figure 9. Memory logic diagram

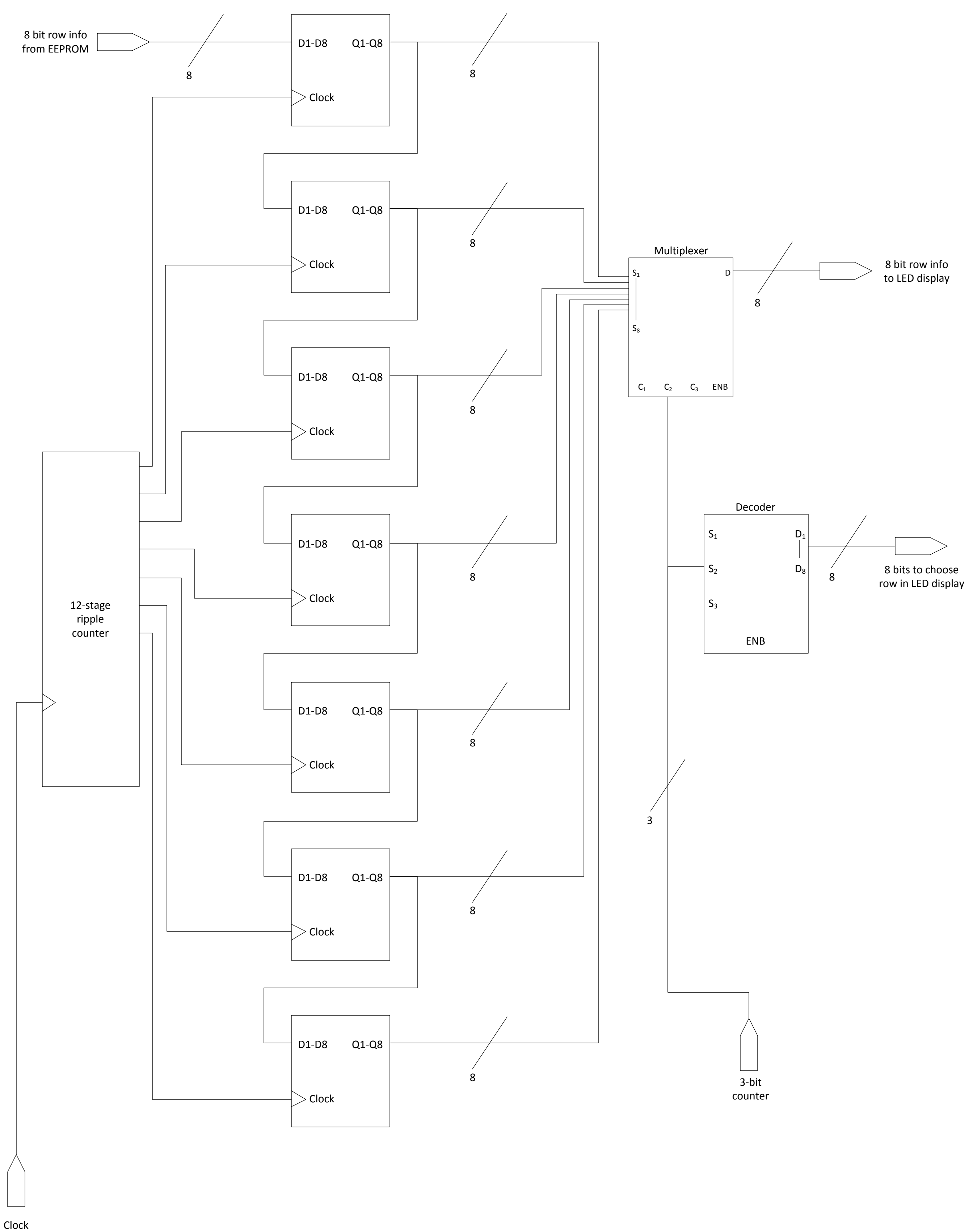

Figure 10. Memory wiring diagram

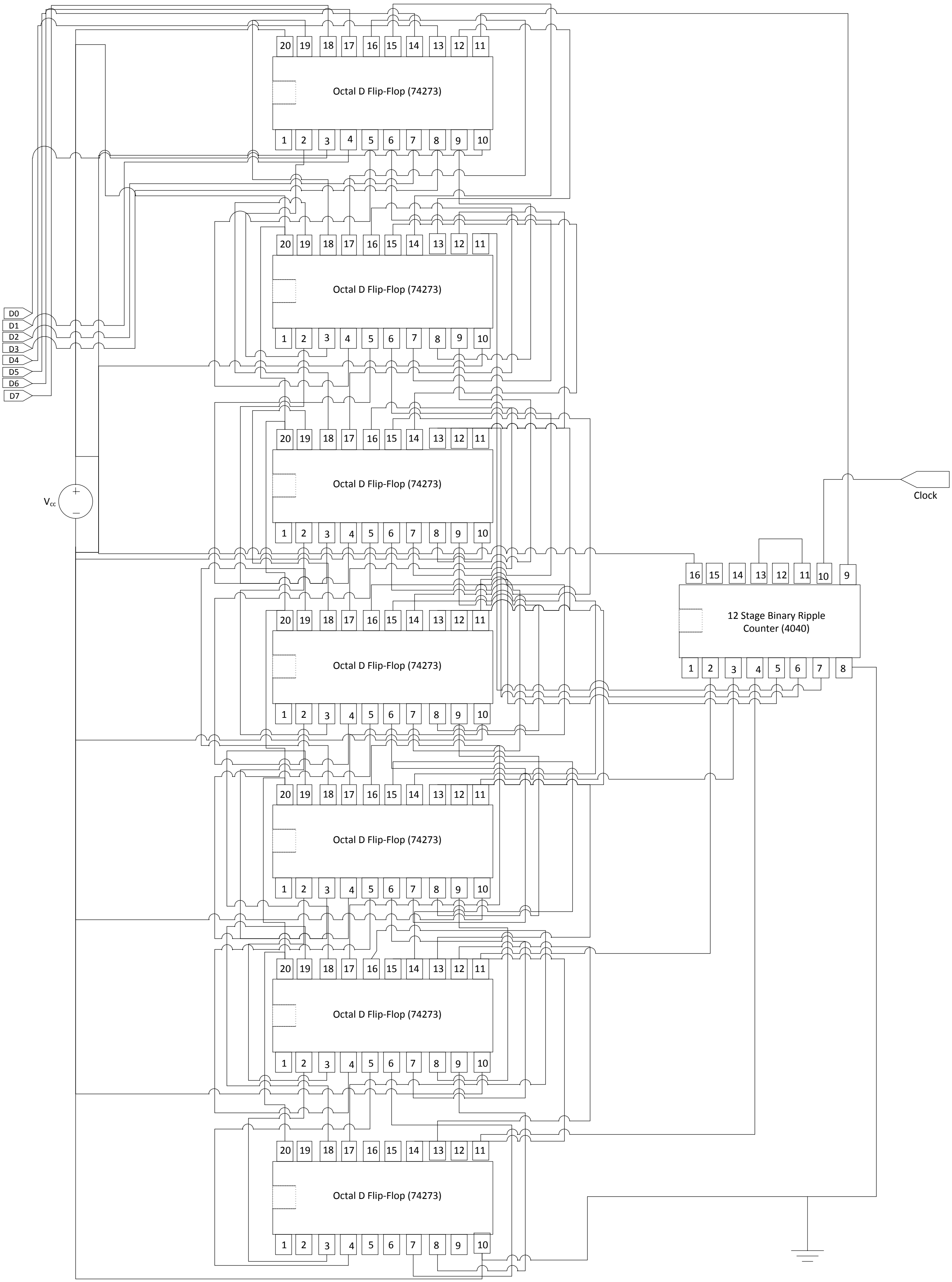

Figure 11. Score Keeper block diagram

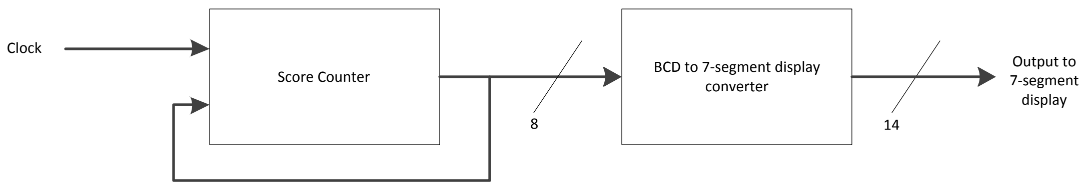

Figure 12. 4-bit binary counter logic diagram

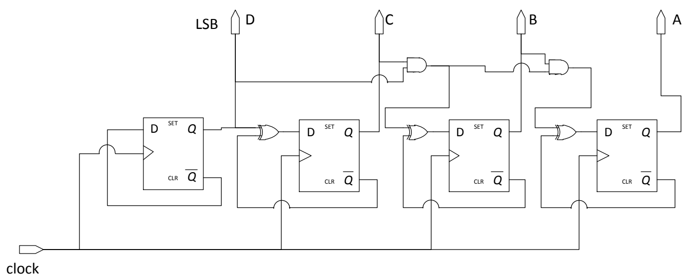

Figure 13. Score Keeper wiring diagram