Electronic RGB Tic-Tac-Toe Game Using Dragon12

Sedim Baffrow, Andrew Kashat, Yousif Fatohi, Peter Younan

ECE 3720 Microprocessors

Oakland University

Rochester, MI, USA

baffrow@oakland.edu, andrewkashat2@oakland.edu, yfatohi@oakland.edu, pyounan@oakland.edu

***Abstract*— This project proposes an electronic depiction of the classic game of Tic-Tac-Toe while using the Dragon12 HCS12 microcontroller board. The system incorporates red and blue RGB LED colors in place of the conventional X's and O's on a 3×3 grid. The system consists of two analog joysticks, which the players use to control around the play grid and place their position by holding down the button of the joystick. The project incorporates the use of nine RGB LEDs, the Dragon12 RGB LED, the buzzer sound system in the Dragon12 microprocessor computer controller system, and the 16×2 built-in LCD display screen. This report provides information regarding hardware, electrical design, software architecture, game logic, user interface behavior, implementation challenges, and potential improvements. Overall, the project was successful, resulting in a fully functional, interactive, and standalone embedded gaming platform.**

***Index Terms*—HCS12, Dragon12, microcontroller, joystick, RGB LED, LCD, embedded systems, Tic-Tac**

I. INTRODUCTION

Traditional games like Tic Tac Toe can be utilized in implementing embedded systems concepts like digital I/O, analog-to-digital conversion, timing functions, and state-machine design. For this project, the Dragon12 HCS12 microcontroller system will be utilized in the display of the interactive game of Tic Tac Toe between the two human participants. The role of each participant would be marked in the display system through the use of red and blue colored LEDs placed in the grid displays. The joysticks would be utilized by the players in selecting positions.

The system has multiple ways of giving feedback. The Dragon12 built-in RGB LED indicates turns during the game process and illuminates the winning player's color after winning. The buzzer gives out a sound each time the game has a winner. The LCD displays the turn constantly and also the score history of the red and blue players. The system indicates the end of the game in case of a tie by flashing all the LEDs, and after about five seconds, the system clears the entire board and starts a new game while maintaining the winning record display in the LCD screen.

The project aims to accomplish the following objectives: (1) implementing the connection of the RGB LED game board and joysticks to the Dragon12 board, (2) implementing the functionality of reading the joysticks and debouncing using the ATD module in the Dragon12 board, (3) implementing the Tic Tac Toe game logic in C language, and (4) providing audio and visual indication functionality through the buzzer, RGB LED, and LCD available in the Dragon12 board.

II. HARDWARE AND ELECTRICAL COMPONENTS

The hardware part of the design revolves around the Dragon12-Light board, featured with the HCS12 microcontroller. Other components are also connected to this board in designing the Tic Tac Toe game interface. The key components are highlighted below:

- Nine common-cathode RGB LEDs in 3×3 formation to display the game board.
- Two analog joysticks providing X, Y, and pushbutton inputs.
- Dragon12 onboard RGB LED (PP4 and PP5 with common cathode on PM2).
- Dragon12 buzzer connected to PT5.
- 16×2 character LCD interfaced through Port K in 4-bit mode.
- Resistors and jumper wires to connect the RGB LEDs to the I/O pins on the Dragon 12 board

The 3×3 matrix of RGB LEDs in each cell employs one red and one blue LED to display the existence of the other player. The cathodes of the LEDs are connected based on the matrix layout, and the anodes are linked through Ports A, B, and P. The joysticks are energized by the Dragon12 board, and the X and Y outputs are read through the ATD0 channels. The push buttons are read from Port H inputs enabled by the internal pull-up devices.

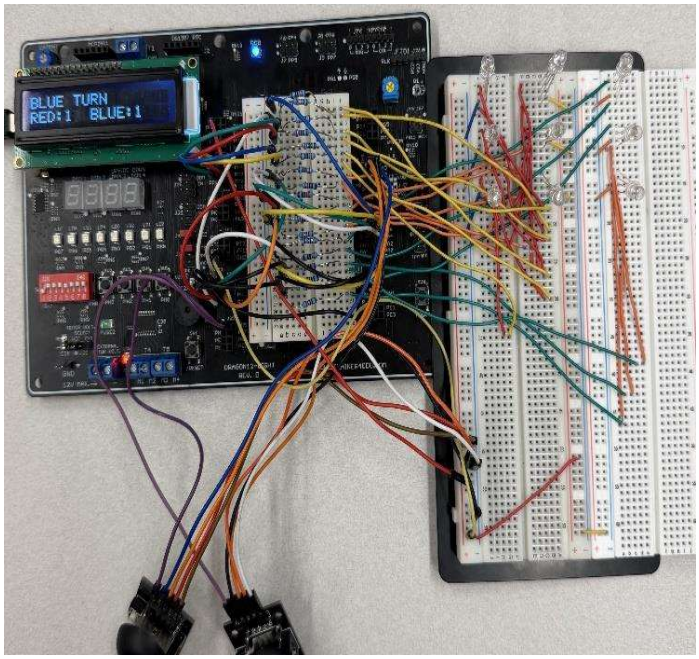

Fig. 1. Hardware Wiring Diagram of the Full System

III. SYSTEM OVERVIEW

Figure 2 shows the overall system architecture. The HCS12 microcontroller communicates the joysticks' positions via the ATD0 module and converts those positions into cursor-movement commands. The game-state machine controls the player's turns and determines if there are wins or ties. The code for the LED driver correlates the 3×3 game board array with the real-world RGB LEDs. The LCD driver displays text messages. The RGB driver and buzzer driver display color and audio information.

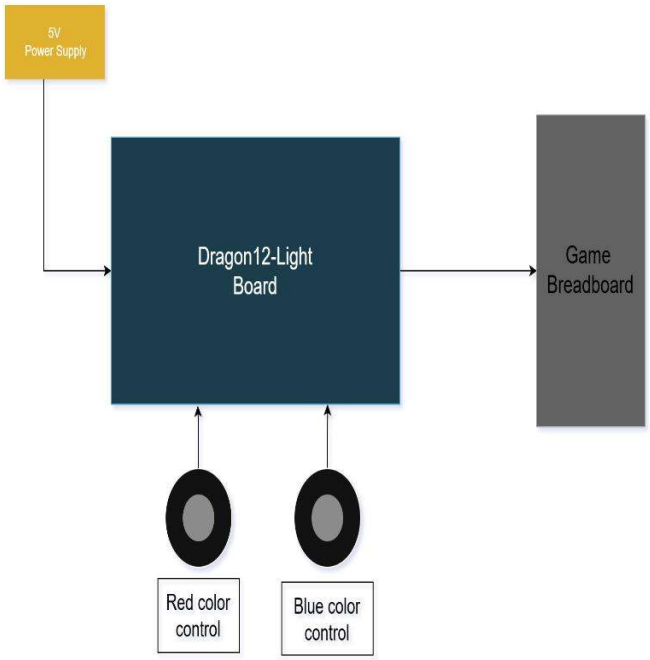

Fig. 2. System Block Diagram

IV. SOFTWARE DESIGN

The code is developed in C and divided into various logical modules like ADT initialization and read functions, LCD driver functions, RGB LED functions, functions for controlling the joysticks and debouncing the inputs from joysticks, functions responsible for game management, functions for win detection for the game, and functions responsible for managing the buzzer.

The overall representation of the game matrix in the code is done through a 3×3 matrix represented in terms of unsigned char. The matrix contains 0 in the representation of the blank position in the game matrix, 1 in the representation of red 'O ', and 2 in the representation of blue 'X' [3].

```
unsigned char board[3][3];   // 0 = empty, 1 = red, 2 = blue
```

The primary loop reads inputs from the joysticks, analyzes move events, recognizes selections based on long or short presses in the current player's turn, updates the display screen, and controls timers based on blink and game reset timeouts. To simplify timing logic and make the code more portable across platforms, the software function delay_ms() facilitated the timers in blink sequences, long pressing actions, and the post-game waiting period.

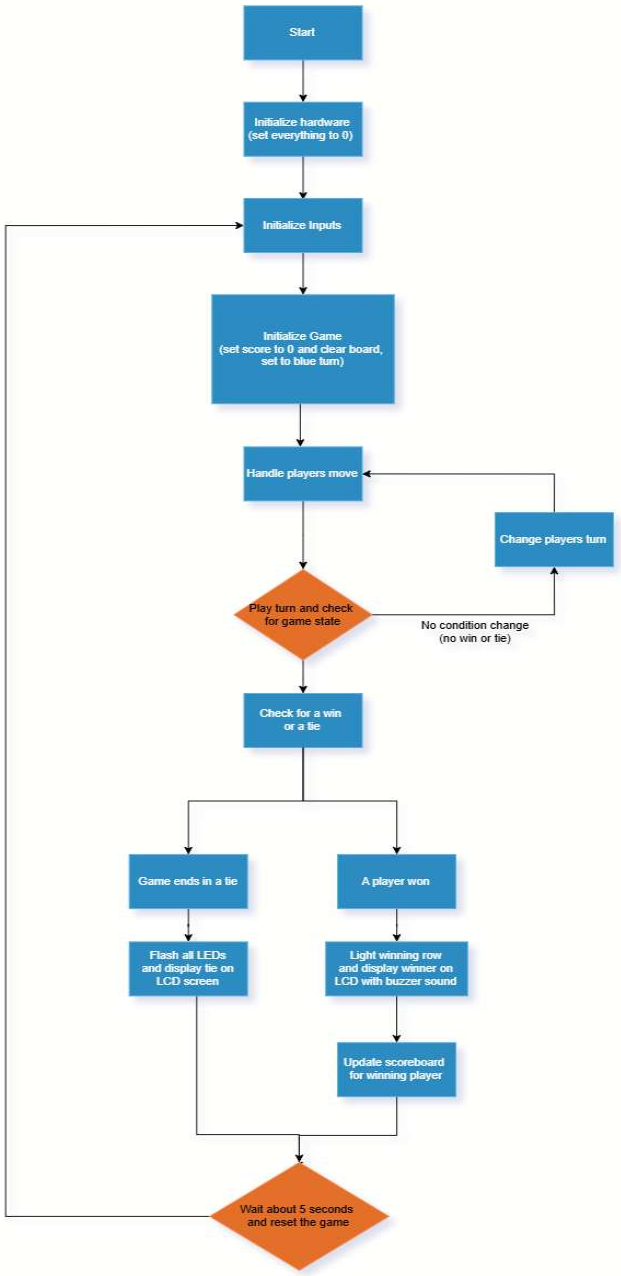

Fig. 3. Software flowchart

A. Joystick Control and Debouncing

The project makes use of two joysticks, one for each player. The project assigns ATD0 channels 3 and 2 for the X and Y inputs of the blue player's controller; the red player's controller's inputs are connected to channels 5 and 4. The values read from each axis are 8-bit values. The values of 70 and 180 are threshold values denoting left/right or up/down motion. Only if the value read from the direction pin differs from the previously read value would the project consider motion from left or right or up or down. The push buttons in the joysticks are read from PTH 4 and 5. The number of cycles in which the push button reads logic 1 defines the number of iterations of the push-button press. Once this number crosses the constant LONG_PRESS_TICKS number of cycles, the project moves toward the occupied cell, and the color of the current player stays in the selected spot. The

placedThisHold1 and placedThisHold2 flags prevent duplication of moves from push-button actions.

Table I. Joystick channels and thresholds

Joystick (Player)	Axis	ATD0 Channel	Threshold Condition	Resulting Direction
Joystick 1 (Blue)	X	ch3	xVal < 70	Left
Joystick 1 (Blue)	X	ch3	xVal > 180	Right
Joystick 1 (Blue)	Y	ch2	yVal < 70	Up
Joystick 1 (Blue)	Y	ch2	yVal > 180	Down
Joystick 2 (Red)	X	ch5	xVal < 70	Left
Joystick 2 (Red)	X	ch5	xVal > 180	Right
Joystick 2 (Red)	Y	ch4	yVal < 70	Up
Joystick 2 (Red)	Y	ch4	yVal > 180	Down

B. RGB LED Board Connection

The 3×3 game board connects in such a way that each cell connects to particular bits in Ports A, B, or P. The function set_cell_color(row, col, color) defines the cells in terms of their matrix positions. For example, in position (0,0), PA7 turns on the blue color while PB6 controls red; for position (2,2), PA3 controls blue and PB2 controls red. In some cases, bits on Port P are used instead of Port A or B [2].

Before a color is activated on any cell, the function first clears the corresponding port bits, and then activates the red or blue bit depending on the player's turn. This abstraction allows the program logic to focus on grid positions rather than low-level pin management.

Table II: LED Mapping Table

Row	Column	Red Channel (Port/Bit)	Blue Channel (Port/Bit)
1	1	PB6	PA7
1	2	PB3	PA4
1	3	PB0	PA1
2	1	PP0	PP1
2	2	PB4	PA5
2	3	PB1	PA2
3	1	PP2	PP3
3	2	PB5	PA6
3	3	PB2	PA3

C. LCD, RGB Indicator, and Buzzer

The LCD is driven in 4-bit mode from Port K. Helper functions from <helper/lcd_helper.h>—including lcd_send_nibble(), lcd_send_byte(), lcd_command(), lcd_data(), and lcd_print(), are used to manage the timing and data transfers to the display. The function lcd_show_score() prints the red and blue win counters on the second row of the screen [1]. During regular gameplay, the top row of the LCD shows either "BLUE TURN" or "RED TURN." At the end of a game, the display changes to show "BLUE WINS!", "RED WINS!", or "DRAW GAME."

The Dragon12's onboard RGB LED is controlled through pins PP4 and PP5 with a common cathode connected to PM2. The functions rgb_set_red(), rgb_set_blue(), and rgb_set_purple() update these pins to reflect the active player or winning state.

The buzzer on PT5 is driven by toggling its bit inside a short loop while a timing counter variable buzzerTicks remains greater than zero [1]. When a win is detected, start_buzzer_win() is called, which initializes buzzerTicks with a predefined value, producing a short tone automatically handled in the main loop.

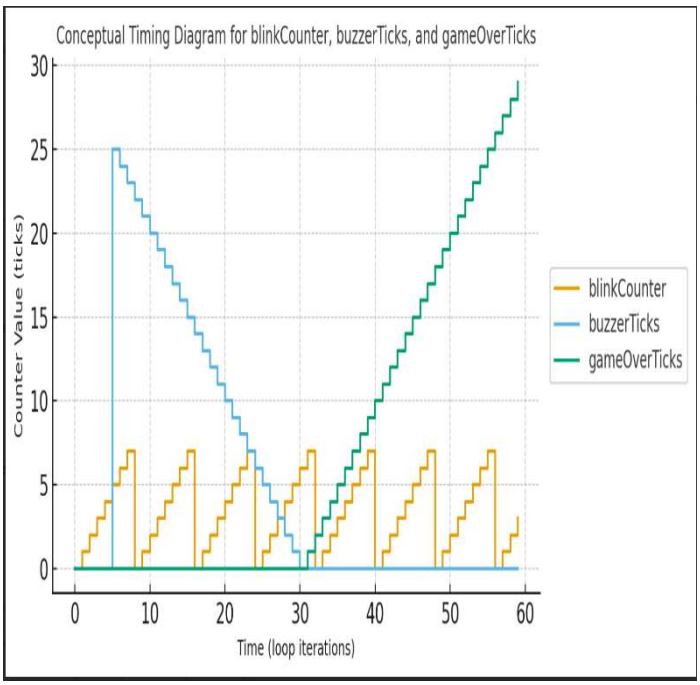

Fig. 4. Timing diagram for blinking and buzzer

V. GAME LOGIC AND STATE MACHINE

The gameplay logic is based on a few global variables. The positions are maintained in the board[3][3] matrix. The currentPlayer variable contains information about the current player. The current position of the cursor is maintained in the curRow and curCol variables. The variables gameOver and winner are used to identify if the game has ended and who wins. The redScore and blueScore variables maintain the number of wins in various games. The variable gameOverTicks postpones the auto-start of the next game.

After each valid move, the function check_winner() is called. The function searches through all rows, columns, and both diagonals for three cells with the same symbol. When a line with three symbols is found, the indices of the cells in the line are stored in the arrays winRow[3] and winCol[3] so that only cells in the winning line are able to blink. If the board is full and no winning line has been found, the function returns 3. This result is stored in the variable winner.

While the game is in progress (gameOver = 0), the cursor shows up in each empty cell as if hinting toward the next step. The cursor disappears when the game ends and a blink pattern begins in all cells in the case of a tie or in cells of the winning line if either player has won. After the announced number of ticks

defined under RESTART_TICKS in the function reset_game(), the game board erases and again starts with the Blue player.

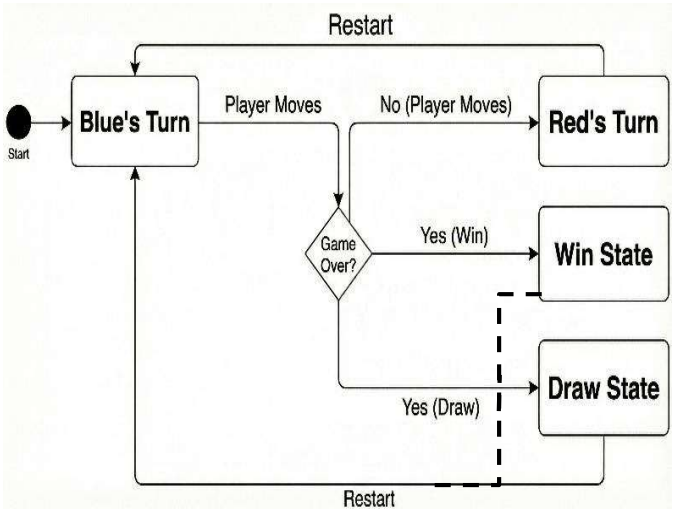

Fig. 5. Game logic State machine diagram

IV. USER INTERFACE AND FEEDBACK

The interface has been developed in such a way that the players are able to know the position by considering the LCD and RGB indications, both built into the board. During normal play, the RGB LED on the board is blue when it is the blue player's turn and red when it is the red player's turn. The LCD display indicates the current player in the first line and the scoreboard in the second line in the way of "RED:XXX BLUE:XXX", where XXX is their score. The 3×3 RGB game display indicates red or blue LEDs in the places where the moves are completed. In addition, the indication of the current position of the cursor appears in the form of a blinking LED in the current player's color.

For a winning player, there is a chime sound in the buzzer system, the LCD screen indicates that there has been a win, and the RGB LED on the board turns solid with the winning color. Also, the winning cells flash on the board while the rest stay put. For the tie situation, all nine cells on the board flash all together while the red and blue channels on the built-in RGB LED both turn on to form purple (a tie). After about five seconds pass, the board clears and the game restarts while keeping the scores.

Table III: User Interface behavior

Game Event	Board LEDs Behavior	On-board RGB LED Behavior	LCD Message	Buzzer Sound	Score Update
Blue Turn	Cursor blinks on empty cell; placed Blue LEDs shown solid	Blue ON	"BLUE TURN" on first line; scores on second line	None	No change
Red Turn	Cursor blinks on empty cell; placed Red LEDs shown solid	Red ON	"RED TURN" on first line; scores on second line	None	No change
Blue Wins	Winning Blue cells blink; others stay solid or off	Blue blinking	"BLUE WINS!"; scores displayed	Short buzzer burst	Blue score +1
Red Wins	Winning Red cells blink; others stay solid or off	Red blinking	"RED WINS!"; scores displayed	Short buzzer burst	Red score +1
Tie	All occupied cells blink; board shows no winner	Purple blinking (Red+Blue)	"DRAW GAME"; scores displayed	None	No change
New Game	Board cleared; cursor set to first empty cell	Blue ON (Blue starts)	"BLUE TURN" + updated scores	None	Scores preserved

VII. PROJECT CHALLENGES

There were a few challenges faced while working on this project. The first major challenge faced was adjusting the values of the threshold levels in the joysticks. This resulted in the cursor being too sensitive if the values were small. This would cause the cursor to lag behind in responsiveness if the values were too high. The other challenge faced in this project involves tracking the current direction of the joysticks in order not to repeat the same action in each iteration.

The next complexity involved mapping the 3×3 game board onto the pins arranged on Ports A, B, and P. There were errors in earlier code versions in which turning on one LED would incidentally affect another because the earlier state had not been cleared appropriately. The problems were remedied through the use of the set_cell_color() function and strategic clearing of the ports.

The blink functionality had to be managed. The same blinkCounter variable in the code handles the display or hiding of the cursor throughout the gameplay process and the display or hiding of the winning or tie blink after the entire game process. The logic should prevent the display of the cursor after completing the game process and prevent the blink from impacting the next gameplay process after resetting the game.

VIII. ALTERNATIVES AND IMPROVEMENTS

There are several additional improvements that could be pursued in future projects. To name one would be the addition of a computer player utilizing the minimax algorithm or heuristics that would allow one human player to compete with the computer. Also, making the gameplay more presentable, adding best of three game modes and implement the timer for each player. In addition, the graphical and sound aspects of the game could be enhanced by adding sound notifications for tied positions, invalid moves, and start of the game, and implementing PWM fading for the LED effects.

From the hardware point of view, discrete RGB LEDs could be replaced by an LED matrix driver or even an addressable RGB LED strip. The LCD interface could be enhanced by including more information regarding the number of games or moves or even replaying the winning combination from the last game.

IX. CONCLUSION

This project illustrates how a simple game can be utilized in studying various core embedded system ideas using the Dragon12 HCS12 system platform. The project incorporates analog inputs from the joysticks, digital control of the LEDs, LCD display of text information, and audio signals from the buzzer in the game. The Tic-Tac-Toe game implemented in this project correctly receives inputs from users, enforces valid moves only, identifies winning or tied positions, displays outputs appropriately, and automatically restarts the game while retaining scores.

During this process, the team acquired exposure in the areas of ATD configuration, port mapping, and state machine modeling, in addition to debouncing and interface design in microprocessor games. The final system exhibits functionality and demonstrates engagement in microprocessor games.

REFERENCES

[1] Freescale HCS12 Microcontroller Family, Dragon12-Plus2 Trainer Manual.

[2] M. Mazidi and D. Causey, HCS12 Microcontroller and Embedded Systems Using Assembly and C with CodeWarrior. Pearson/Prentice Hall, 2009.

[3] ECE 3720 Microprocessors Course Materials, Oakland University.