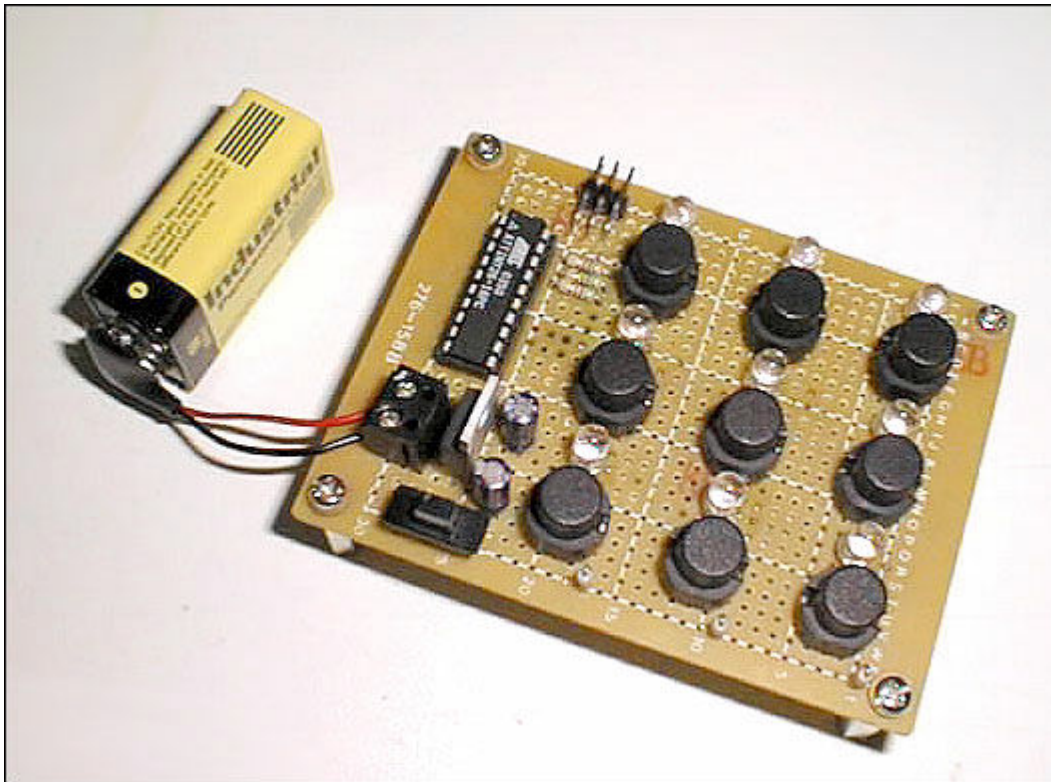


AVR A.I. TIC-TAC-TOE

A3448



Introduction

I'm a programmer, and after dealing all the time with databases and things of the kind. On my fun time I play with A.I. engine. It's amazing how the code for A.I. can look so beautiful.

Thinking about an interesting project to build. I remembered a movie I've seen when I was a kid, War Games. That movie had simply the first doable A.I. computer game shown on the big screen, the Tic-Tac-Toe. At that time, not long ago, it was a really big deal for a computer to actually play Tic-Tac-Toe.

The Tic-Tac-Toe project presented here shows the versatility of the AVR Micro Controller. Due to its optimized RISC architecture it was easy to embed in less than 2k all the code necessary to make the game, including driving the LED, and scanning the KEYPAD. The standard 1MHz internal Clock was much more than enough to do all the tasks needed.

This game consists of a 2x3x3 LED matrix, a 3x3 keypad scan, and embedded is all the logic algorithm necessary to make the game impossible to beat.

There are quite a lot of IDE tolls for the AVR. I've decided to use the AVR-GCC since it is free, portable and rock solid. I was impressed with how good was the AVR studio to debug C code since the AVR-GCC generates all the debug outputs, I could actually see the registers

changing on the fly without the need for an in circuit debugger wow!

Theory of operation.

I've divided the software in three parts:

MCU.c - Deals with internal configuration like timers, etc

IOLIB.c - Deals with keypad and display

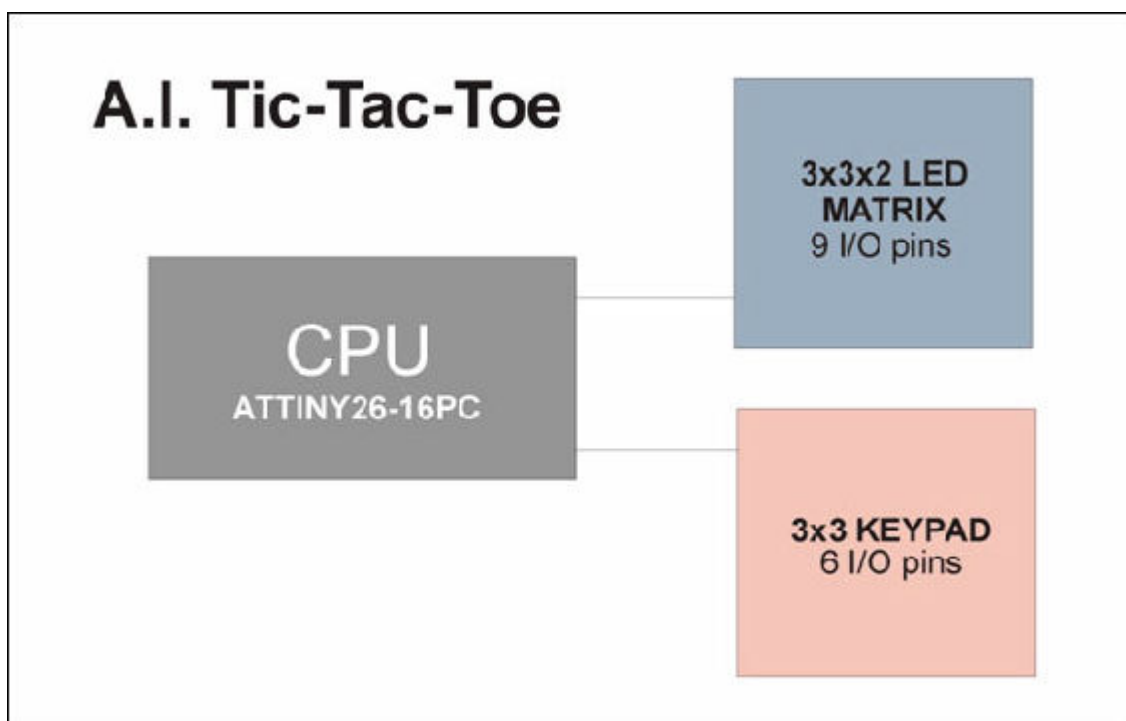
TTT.c - The compleatly independent game algorithm.

The game consists in two main loops. One the first one, the computer starts and on the second one the human player starts. The game waits unti a key is pressed to begin the game. While it ways it displays a pre-game introduction.

I've used a bi-polar LED to show the plays. Each player has a different light color, the computer has the red LED and the Human Player has the Green LED.

There is a display buffer and a fixed interruption that deals with the display update. So, the program can at any time load the display buffer and expect the display to be updated on the next refresh.

There are 16 ways for each player to win the game. Making 3 LEDs of the same color light in a line. After every move the program loads the solutions in to a solution table and checks if someone won. By the end, if the game finishes it displays the appropriate ending and loops back to the next game.



The Tic-Tac-Toe brute force algorithm

The Tic-Tac-Toe game is well known for always end in a tie. But to foreseen every single possible move and translate it into an algorithm is a very interesting task. I've searched on the web for algorithms to solve the game and the only thing I've found was a very complex DNA algorithm that was an overkill for such a simple task. So, I had no choice but to do create the algorithm by myself. Here are the magic steps of my brute force algorithm to never lose on tic-tac-toe.

Following this order.

- If the board is empty: Pick a corner.
- If You've already marked 2 positions on a line and the third one is free: Mark the third to win.
- If your opponent has already marked 2 positions on a line and the third one is free: Mark the third to defend.
- If the center is free: Mark the center.
- If the opponent has marked 2 adjacent lateral middle positions: Mark the corner in between.
- If all corners are still free (may happen if the opponent began): Mark a random corner.
- If you've already marked a corner: Mark an opposing free corner.
- If you have the center: Mark on a line to keep the opponent busy. And prevent the opponent from winning with the tree corners move.
- Try to conquer 3 corners: Mark a random free corner.
- It is going to be a tie anyway: Mark on a random free position.

Conclusion

After working a few years with the 8051 architecture. To migrate from scratch to AVR was a smooth ride. I'm always looking for the "Programming" features and certainly the AVR has many more than anything I've ever worked with. It gave me a great deal of experience experimenting with all the available IDEs. And the charm of Prototyping with the ICE instead of burning FLASH memories with programmers was unquestionable.

