THE STATIC TIC TAC TOE COMPUTER.

H. Holden, Dec. 2021.

INTRODUCTION:

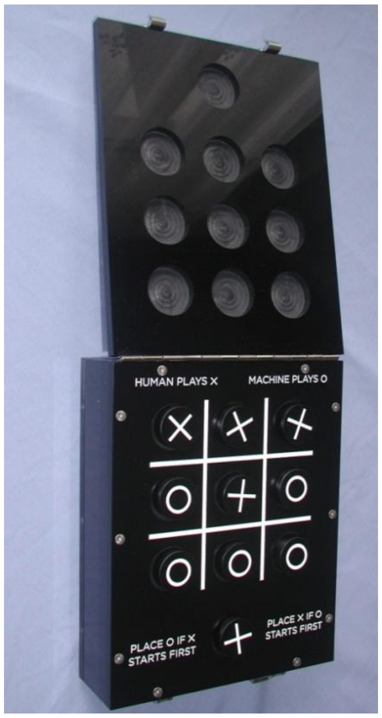
This is a type of computer whereby there is no Microprocessor, no clock, no registers or latches holding intermediate data, merely logic gates and a ROM that behaves as a complex array of gates. The computer board responds to data input bits from an Encoder board and produces four output bits use to illuminate the player board LED's. In this type of system only the "computation delay" is the propagation delay of the gates or logic devices.

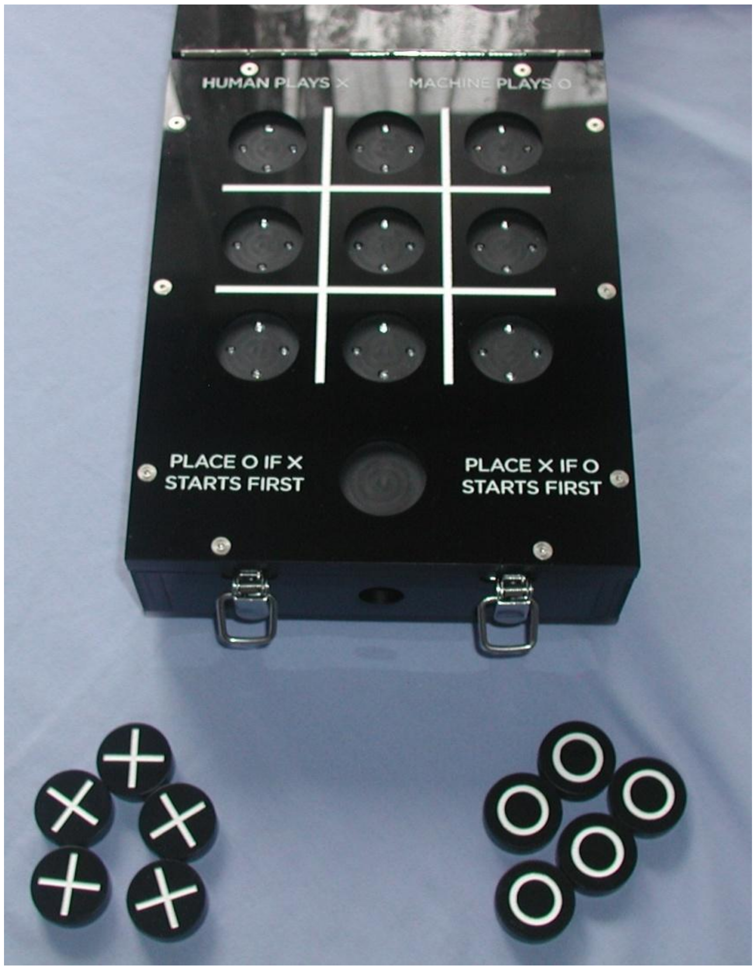
In addition, since this computer simply responds to static logic conditions it can never get confused or out of step with itself, or lock up that matter.

While speed itself is not important for this application, low current drain is. The current consumption increases with clocking frequency in clocked systems, because when Cmos IC's are clocked they draw more current. This makes a Cmos based Static Computer extremely attractive for battery operation and low power consumption, in this case the current consumption from the 9V power source varies between 75mA and 90mA.

This particular Tic Tac Toe game does contain an oscillator, but it is not used for any computation, instead to create a Random Number Generator using a "spinning wheel technique" (see below).

The player pieces shown in the photos below are 1" or close to 25mm diameter.





THE TIC TAC TOE DESIGN PROBLEM:

Before tackling this design problem I decided a good initial move would be to consider, as it is in reality, two people playing the game and what features they would exhibit during their game play experience.

Generally, since most people are democratic by nature, they would conclude that the only fair way to play the game would be to have one person start a game, then the other alternately start consecutive games. *The person who starts the game first has a significant advantage over the other player*, so this is the only way to average that bias out over a number of games.

In addition, one Human (deemed to be the one not starting first) has no knowledge of how their opponent might start the game. In other words which square the starting player might initially choose. From the perspective of the person starting second, they could perceive the game start to be randomly chosen. A predictable player might start the game the same way every time, for example starting on the central square first, in an attempt to gain maximum advantage. But that sort of predictable behaviour soon gets very boring.

When one Human “Wins” the game or beats their opponent, they likely announce it with great enthusiasm.

Finally, two Humans playing each other would be imperfect to the extent that sooner or later one or the other of them might make an error of judgement. Thus allowing the other Human, who didn’t make any errors, not just to be able prevent the other human from winning (causing a Draw), but be able to Win against or beat the other player. Typically, by taking advantage of the other Human’s ill considered moves.

Clearly then, a proper and complete Tic Tac Toe Machine would not only be unbeatable by the Human player, but it should be able to Win against the Human at every opportunity. This requires the analysis of every possible mistake the Human could make during game play.

In light of the above features I decided that the way to design the machine would be to initially create a two player board game.

The game would be in the form of a playing board, where each of the players could place a disc, with an x or an o label on it in the player area. This board then would work fine, even with no electric power available and two players could enjoy the game together as usual.

However, if one of the Human players “goes missing” and the game is powered, the Machine steps in to replace one Human player. It then becomes a “Human vs Machine” scenario, so as to satisfy the notion of a Machine that plays Tic Tac Toe against a Human.

Also, in light of the above features, of two players playing the game, the Machine must be able to perform the functions that a “Flawless Human player” would possess. Never make any mistakes, never be beaten, but also Win against its Human opponent at every opportunity when the Human makes an ill considered move.

Therefore, from now on in this text, consider the behaviour of the Machine to be that of a flawless (or unbeatable) player, intent on winning too.

The parameters cited above generated the set of basic properties which this particular machine has:

STATIC TIC TAC TOE MACHINE FUNCTIONAL PROPERTIES:

1) The Human or the Machine can start the game first. Therefore, over a number of played games, it averages out the intrinsic advantage given to the first player.

2) The Human plays pieces (discs) labelled x and the Machine plays pieces labelled o on a player board.

3) When the Machine gets to start, its start position on the player board, from the perspective of the Human opponent, is randomly selected (see Spinning Wheel technique randomizer below).

4) The Machine behaves as a flawless Human player, not only preventing the Human from winning the game under all circumstances, but also taking every opportunity to beat the Human opponent, if the human opponent makes any single slip up during game play. The machine *never* makes a slip up. So at very best, the Human cannot beat the machine, he/she can only draw with it.

5) Much like an excited Human, the Machine announces when it has won the game, with a Beep from a Piezo transducer.

6) Since the game is configured as a player board, with x and o player pieces and since the Machine only has a "brain" and not eyes and arms, the machine asks the Human player to place the Machine's o discs on the board, by lighting LED's on the board where the machine wants its o disc placed.

7) The board can sense the presence of an x or an o disc placed on the player board. The computer "knows" not only which is which, but where on the board each is. And it "knows" when it is the Machine or the Human's turn to move. When it is the Machine's turn to make a move the Machine "computes" in less than 200nS, performing an analysis of the board pattern of x's and o's and lights up the LED's on the board where it wants its o piece placed. This corresponds to the computer being enabled and the main control line in the circuitry called "/Compute" going low.

8) It is apparent that the maximum number of x player pieces that can be applied to the player board when x starts first is 5, limiting the number of o discs that can be placed to 4. And that the maximum number of o discs, when o starts first, is limited to 5 , thereby limiting the number of x discs to 4. This means that there is either an x disc, or an o disc, left over, depending on whether the Human x or the Machine o started the game.

Therefore, for physical storage of the discs on the game board, an extra space is provided for the unused disc. This space also acts as a Bipolar Electrical Switch to configure the Computer circuitry for "who starts first". This not only creates the physical storage space for the unused disc but it also avoids having to have any mechanical switches.

Therefore if x is to start the game first; An o disc is placed in the spare space, but if o is to start first; An x disc is placed in the spare space. This instruction is engraved onto the player board surface along with the fact that the Human plays the x pieces x and the machine plays the o pieces.

When the game is initially powered, with no discs at all placed anywhere on the board, all of the LED's are lit. This represents an "o start Randomizer" function. It represents the LED's on the board rapidly changing positions in terms of which spaces on the board is lit up.

If an x disc is placed in the spare space (meaning the Machine o is going to start first), a random position will be selected by the LED's for o's very first move. This is appropriate because, if the machine were a Human player, from the point of view of the other Human opponent, the initial start choice would appear random. Unless one Human player, always played the same initial move, that would be very boring, with the opponent saying "not again".

9) Since the game is configured in the form of a player board. When its lid (top hinged cover) is closed, it safely stores all the playing pieces (discs) inside it, so that they do not get lost or separated from the game.

10) The prototype game is powered from a 9V adapter, but since the current consumption is very low at around 75mA to 90mA, it can be powered from a 9V battery or battery pack.

MACHINE DESIGN:

This is divided into 3 parts:

1) Electronics Engineering & Machine Operating Theory.

2) Mechanical Engineering, Materials, Industrial Design & Artwork.

3) Supportive Data.

The supportive data, discussed in a more detail later, takes the form of 135 Charts. These cover all of the playing scenarios in the two cases of either the Human x or the Machine o starting the game first.

The charts (hand crafted) generated the data which programmed the machine's unique responses, of which there are a total of 845. The 845 responses were hand programmed into the ROM with a Hex Editor and the GQ-4x programmer.

Therefore, any pattern of play that occurs can be looked up on the charts (or checked on the actual game) to convince oneself that the machine is making the ideal response. This proves that the machine works for every possible pattern of board play and produces a Machine response which always, without fail:

1) Prevents the Human beating the Machine.

2) The Machine takes advantage of every possible Human mistake to Win against the Human.

Brief video to show machine in operation:

https://www.youtube.com/watch?v=IE9a5ZJZCgE

1) ELECTRONIC ENGINEERING & OPERATING THEORY.

I was inspired by the fact that Mr. Smith was able to build a Tic Tac Toe machine from parts from a Telephone exchange in 1958. Likely these parts were vintage at the time, most exchange spare parts then likely dated back to the 1930's.

Looking back through Electronics History, I decided it should be possible to do this game solution without a Microprocessor and simply use logic gates. I'm very fond of 74 series logic gates. The common ones I use are 74xxx or 74LSxxx types. There are also Cmos versions; the 74HCTxxx series. These perform the same logic functions with lower power consumption. So I chose these for this project. Also I used Blue LED's as they are very energy efficient.

To be able analyse the required data patterns of game play and to give the correct logic output response to the LED's on the player board, I used 27C020 UV EProm. The main reason being these have the required 18 address lines to process the player board logic using the method I have designed. This design method, among other things, involves using the Encoder board's data Parity to control the Computer's computation action depending on who starts first.

The modern version of this ROM is the AT27C020 from Microchip also coming in a PLCC package as well as DIL. The Winbond W27C020 is readily available and cheap on Ebay at less than a few dollars. However, in my workshop I had the ability to erase and re-program the 27C020 UV Eprom so I used what I had on hand. In this case, the ROM acts as a giant logic gate array, with a specific output for specific inputs. An FPGA could achieve the same result, if it was programmed correctly.

Fun with (weak) Magnets:

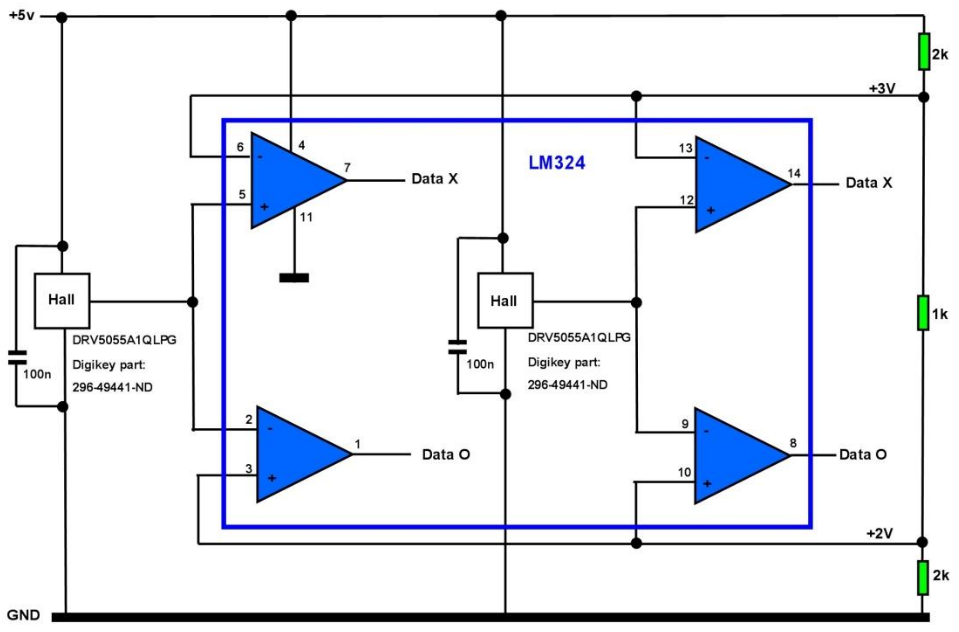
Wanting to avoid mechanical switches and not having a *u*P supported touch screen interface, I decided to use Ferrite magnets from Jaycar, embedded in "player pieces" and use Ratiometric Analog Hall sensors.

(Obviously any other technique to recognise the player pieces as being different and present or not, could be based on other properties of them, color, conductivity, electrical capacitance, they could even be RFID devices, but for this project I chose to keep it as simple as possible).

The Ferrite magnets are Jaycar part LM1616, which come in a packet of 12. Ten are used. A 14.5mm diameter 4.5mm deep hole was cut into the bottom of each 25mm diameter (10mm thick discs) and the magnets were glued into them with 24 Hr epoxy resin.

NOTE: Powerful Neodymium magnets were deliberately not used as there is far too much magnetic force and they have a habit of jumping to each other or the nearest magnetic object and magnetizing it. They are also good at accidentally erasing magnetic media. This is why weak magnets were chosen, but their flux field is above what is required by the Hall sensors.

The circuit below shows how the player Encoder Board works:



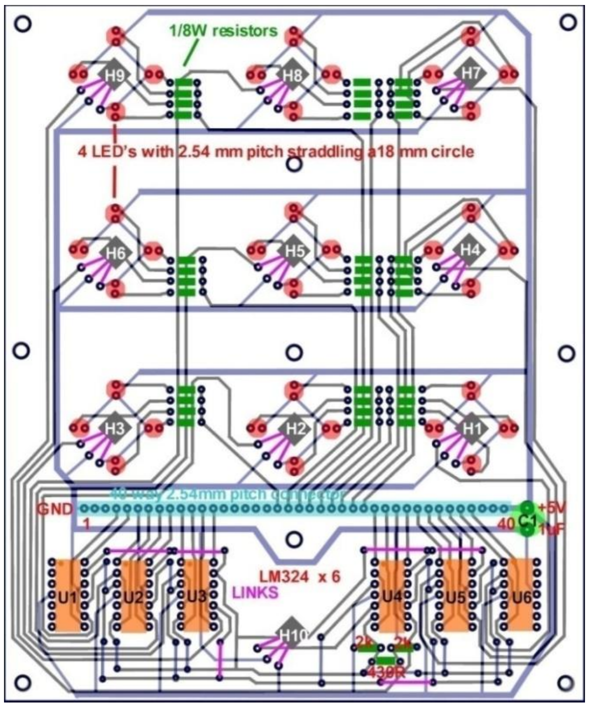
With no applied magnetic field the DC output of the Hall device sits within 70mV of 2.5V

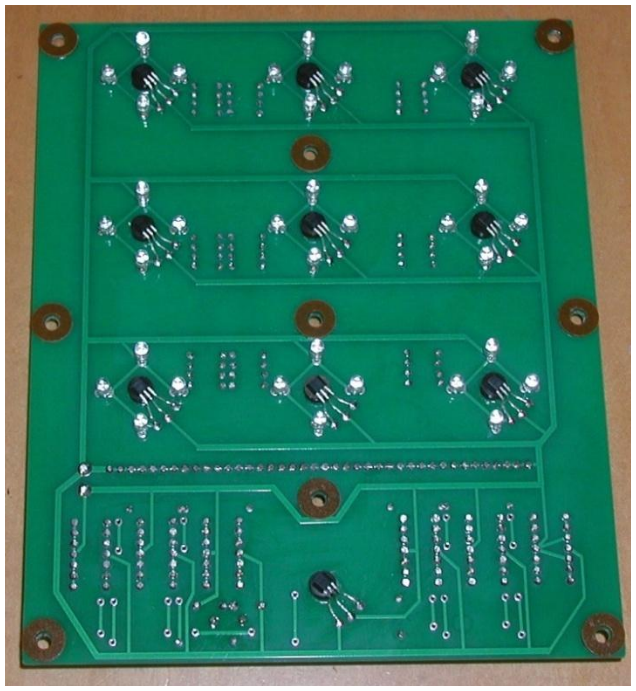
The x and o player pieces have the magnets glued inside them in a reversed direction. So placing an o disc causes the output from the Hall sensor to go low and this produces "Data o".

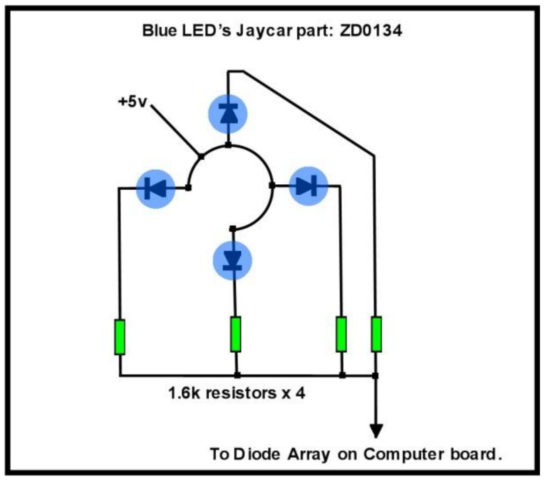
Or the Hall sensor output increases above a threshold when an x disc is placed then "Data x" is generated. As can be seen the OP Amps are being used as comparators to produce a logic 1 when an x or o player piece is placed on the board. LM324's are handy for this job because their output voltage swing, when they are powered from 5V, is a perfect match for TTL logic levels.

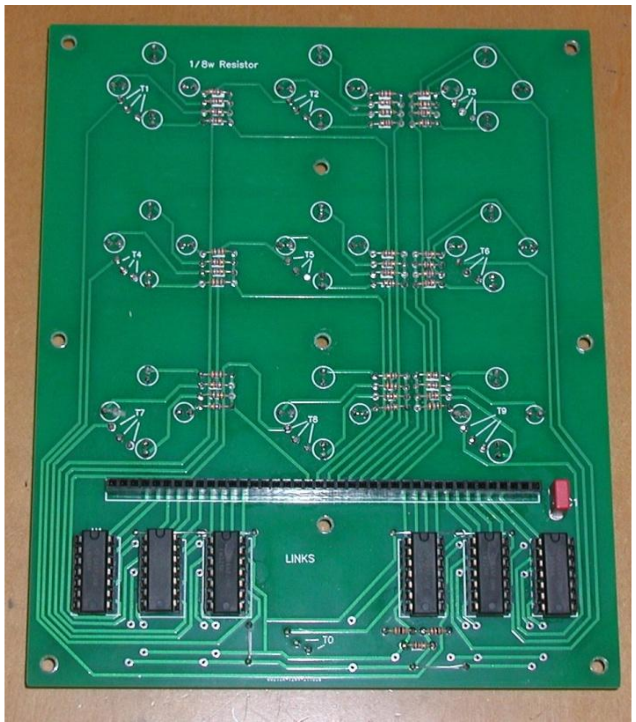
The general arrangement is shown above and this circuitry is duplicated 10 times on the encoder board. A Hall device sits under every position on the player board where a player disc can be placed during the game, so there are 9 Hall devices there. However, an extra Hall device (H10) was placed on the spare space on the board. This acts as a “change-over switch” to control the behaviour of the game, depending on which player starts first, the machine o, or the Human x.

The Hall devices, LED’s, resistors for the LED’s and the LM324’s were mounted on one PCB, called the **Encoder Board** shown below:






Blue LED's Jaycar part: ZD0134
+5v
1.6k resistors x 4
To Diode Array on Computer board.



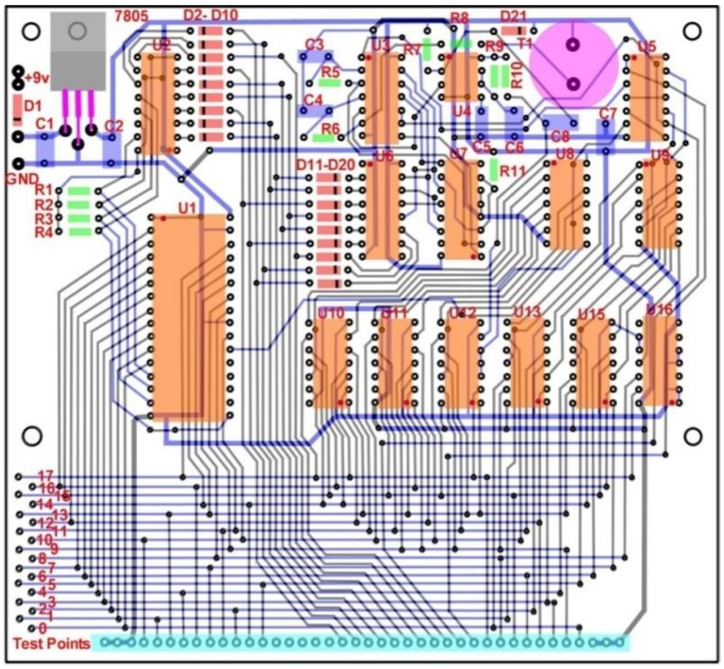
Four LED’s on each player board position are used for two reasons. One reason is symmetry and a pleasing appearance, the other is redundancy. If one LED fails (it does happen sometimes) the game can still function normally. Each LED in the group of 4 has it anode connected to +5V and each has a separate cathode resistor Each LED has around a 1mA operating current, but being Blue LED’s, they are quite bright at this low current.

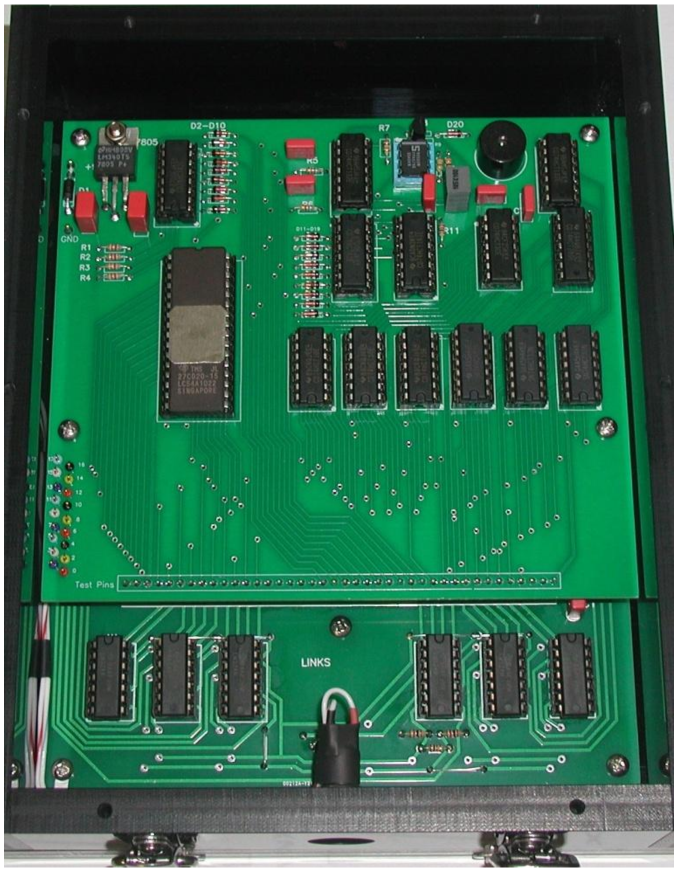
To allow for the solder pads on the bottom of the board, 0.8mm thick washers were glued to the pcb mounting holes. Some 1/8” tall Phenolic spacers were glued under the bodies of the Hall devices to make sure that they sat close to the under-surface recesses on the player board, but they are not definitely required.

The manufacturers of the Hall devices recommended bypass capacitors on their supply pins, so these were added later as surface mount ceramic types, between the solder pads. The devices appeared to work fine without them, probably because in this computer there are no switching transients on the power rails, being a static computer.

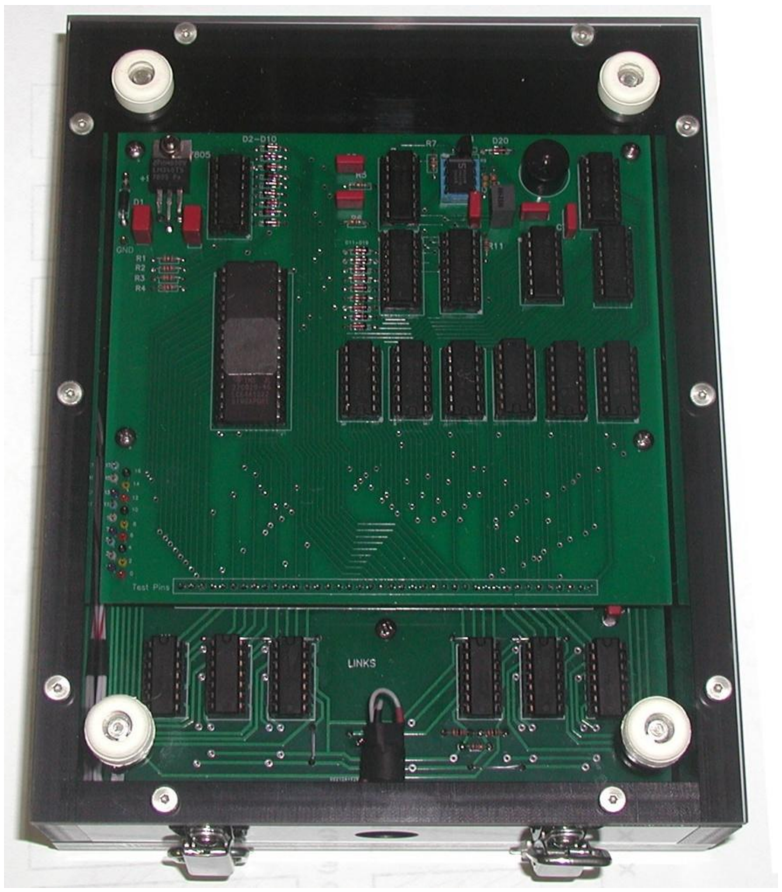
The Hall devices are numbered 1 to 10. The 40 way connector plugs onto the Static Computer pcb, this avoids a large mass of wires.

Top view of the Static Computer pcb is shown below and a photo of it mounted in the machine, and plugged directly on top of the Encoder board and mounted to it with 4 spacers to allow for the height of the 40 way connector:





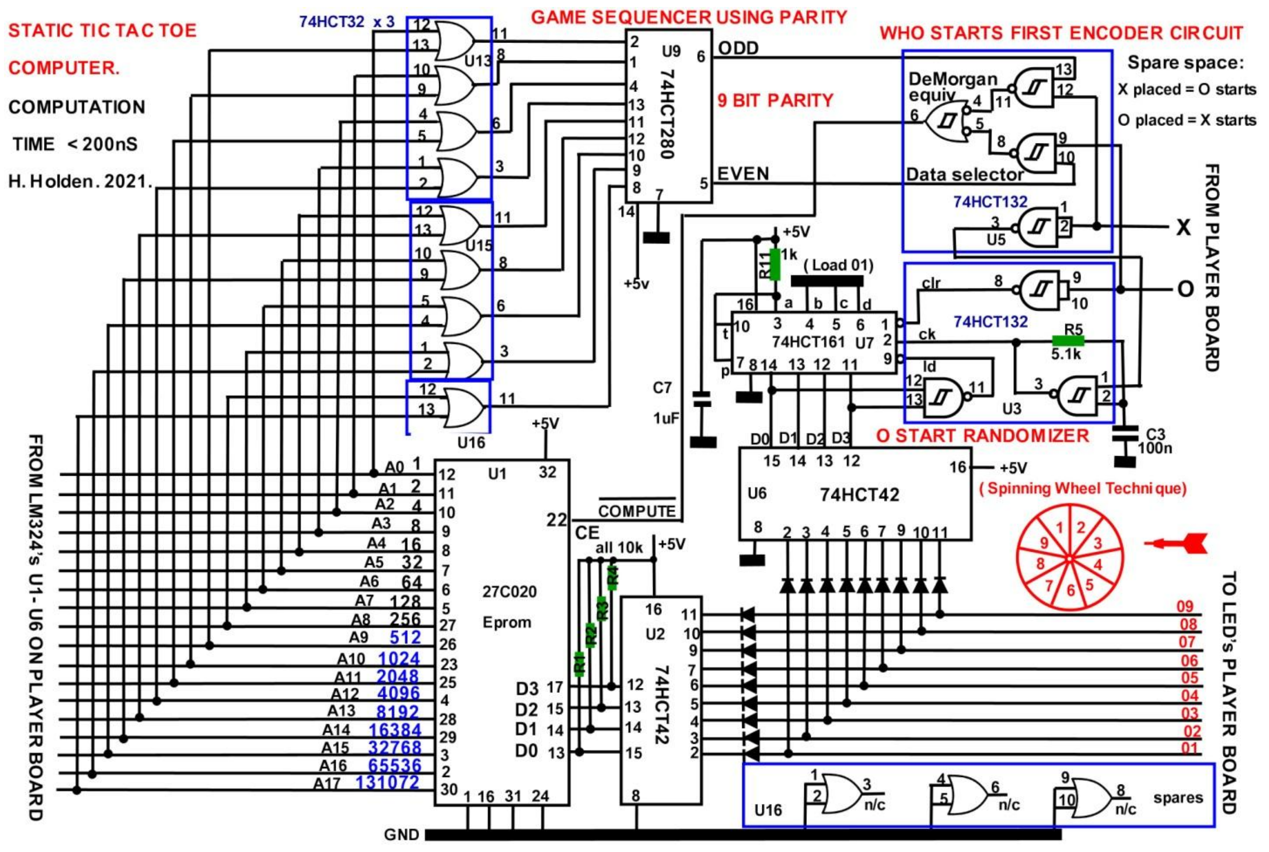
The machine was also fitted with a semi-transparent bottom plate with rubber feet. This is so the machine's electronics can be inspected to an extent, without having to remove the bottom plate:



All through hole components are used. Of note there are no Electrolytic Capacitors used in this design, only film types, this makes the circuitry very long lasting.

The Data lines A0 to A17 have physical test points on the board. These are small gold plated loops with coloured glass beads seen on the lower left side of the computer board. They are not required, but I put them there because this machine is a prototype and they were handy for confirming the Encoder board was working properly.

The Computer Board Schematic:



Operating Theory:

In general, the player board must be able to recognise any possible pattern or combination of playing Human x and Machine o player pieces on it. It must also distinguish, at each of the 9 locations, if an x is placed there, an o placed there, or no piece at all placed there.

In a sense this is a problem of “tri-state logic” rather than binary. To convert this information into a binary format, suited to evaluation by the static computer, the Human x’s and the machine o’s player pieces, for any board location, each generate a different magnitude of binary data (see table below).

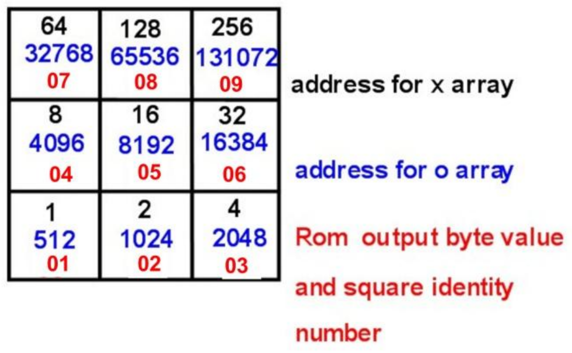
Any board pattern of x's and o's , after the binary values of x's and o's are summed, generate a unique binary address. This is why the Encoder or Player Board was designed the way it was, with the Ratiometric Hall sensors and opposite polarity magnets placed inside the player pieces.

Referring to the table below (which also appears on the Chart diagrams- see support folder) For example, if an o piece was placed in the centre on square 05, this generates a bit assigned a decimal value of 8192. Then if an x piece is placed on the board at location 07, this generates a bit assigned the decimal value 64.

Therefore, for this simple state of two pieces on the board, it is recognized as address 64 + 8192, or decimal value of 8256. That value can be used as the address to the location in the ROM where o's next move is stored, in other words the ROM then outputs a value to light the appropriate LED where the Machine o's piece is next to be placed.

No player pieces being present on the board generates decimal zero and it is associated with an output byte FF. The lower 4 bits or F from the ROM generates no output on the 74HCT42 decoder U2, as it is an invalid code, so no LED's are lit by that decoder in that state.

The table below shows the magnitude assigned to the data generated by the encoder board.



64 32768 07	128 65536 08	256 131072 09
8 4096 04	16 8192 05	32 16384 06
1 512 01	2 1024 02	4 2048 03

address for x array

address for o array

Rom output byte value and square identity number

The numbers in black show the decimal value generated on the 18 bit data line when an X piece is present on the board in the named location, 01 to 09. These labels are also the data byte value of the ROM's output. For example, if it is the machine's turn to place its o piece on the board, and the Machine wants the piece placed on square labelled 09, the ROM outputs the byte 09 (only lower 4 bits used), which causes the LED's on that square to light up.

Of note, the machine only lights the LED's under specific circumstances:

1) The game is initially powered with no player pieces present on the playing area. In this case all of the LED's are lit. This function is the "O start randomizer" for the option "Machine o starts

first”. (This also serves an LED test function, similar to the way it was once customary to light up all lamps on an instrument, briefly as a turn on test).

If an x player piece is placed on the spare space, a randomly selected square will remain lit for the first o piece to be placed corresponding to the count that U7 stopped on and the 4 bit value presented to U6. This makes the Machine o’s initial start, random.

2)The game’s electronics goes into “compute mode” when it is the machines turn to choose a square. The outputs of the ROM are enabled by the control “ /compute” on pin 22 of the ROM going low. The ROM is activated and the appropriate LED’s are lit to show where the Machine wants its o piece placed. When the /compute control line is high, the ROM’s outputs are tri-stated to open circuit and pulled up by the four 10k resistors. When the game is not in “compute mode” the 74HCT42 U2 is presented with FF (due to the pull-ups) so no LED’s are lit by this IC as it is an invalid code for the 74HCT42.

In summary: “compute mode” is entered when it is the machine’s turn to make a move and the appropriate LED’s are lit.

Noticed how the ROM outputs were assigned bytes 01 to 09, not 00 to 08. The reason for this is that the o start randomizer, 74HCT161 is a binary counter which needs to make 9 counts, 1 to 9 and cyclically reset to 1, but also have another unique state of zero, when it is reset and cannot light any LED’s. Outputs 1 to 9 of the 74HCT42 (four line to 10 line decoder IC’s U2 & U6) are used to light the LED’s. If this IC is presented with input nibble 0, its output lines 1 to 10 remain high and no LED’s are lit.

The “Spinning Wheel” randomizer:

Imagine a spinning wheel moving so fast you cannot see the numbers 1 to 9 on its segments and you throw a dart at it, or you could stop it abruptly with a brake. Some number 1 to 9 would be selected apparently in a random manner.

This is the job of U7, a 74HCT161 counter. It is clocked at around 3kHz by U3 pin 3. When count 9 is reached, the output pin 3 U11 goes low, which loads the counter with the value of 1, so on the next clock pulse the counter resets to 1. When the x piece is placed on the spare space on the board (signifying Machine o is to start first) the oscillator output U3 pin 3 is inhibited and some count between 1 and 9 remains and this lights the LED’s on that square on the board.

(Of note; one might wonder why the Count Enable pins P & T on the ‘161 IC were not used to inhibit its counting, rather than stopping the clock as was done here. The reason is that these inputs should not be toggled when the clock pulse is low and there is no temporal relationship between the moment when the clock is stopped by the Human placing the x on the spare space and that of the clock pulse happening to be high or low at that moment, they are temporally unsynchronized events).

While the particular LED's lit by the output of the randomizer, when it has stopped, remain on for the remainder of the game, the first O piece is placed over them, so they are not visible.

In the case that the Human player x starts first, an o player piece is placed on the spare space. This causes pin 8 of U3 to go low, clearing the 74HCT161 counter to zero, so no LED's are lit by the randomizer because the output corresponding to zero, from the 74HCT42 U6 is not used. More LED's are only lit after the first Human x piece is placed, to satisfy the Machines next move.

Game Sequencer using Parity:

The computer only should go into "compute mode" when it is the Machine's turn to make a move (place an o piece).

Data patterns, generated by the encoder board, just after a selected o has been placed, have no meaning to the computer, because the Human player has not placed their next x yet.

The question of: "when to let the computer compute" actually has two answers, depending on if it is the Human playing x or the Machine playing the o pieces, who gets to start the game first.

In the case of the Machine o starting first, there is initially the one o piece on the board (selected initially by the randomizer), then an x is placed by the Human. Now there are two pieces on the board, an even number. And it is time to compute for the Machine o's next move. After the Machine o's move, the total number of pieces goes to 3 which is odd and now its time **not** to compute and so on.

However, if the Human x starts first, they place their x piece, one is an odd number and in this case, now it is time to compute to determine the Machine o's move. Once o is placed the number goes even and now the computer is required not to compute, because it is x's move again. X plays again and the number goes odd, putting the computer into compute mode.

I realised to solve this problem I could use a Parity IC, with a data selector on its two outputs, to select either the odd or even outputs of the Parity IC (U9 74HCT280) to control the ROM's /compute control line. The NAND gate U5 is wired as a data selector to route the outputs of the Parity IC to the ROM, so as to control when the ROM is enabled and computation happens.

The input to the Parity IC comes from IC's U13,U15 & U6. The x and the o data are OR'd together. In this case, the result of placing an x or an o on the same board location generates the same data pattern. This data can then be used to determine if the total number of player pieces (x's or o's) on the board are odd or even.

It requires some more explanation with examples:

The two signals "X" and "O" on the upper right hand side of the computer schematic, come from the spare place on the player board. As noted this space performs two functions. One is a storage location for the piece when the game is not in use and the other to act as a bi-directional or "changeover switch" the state being determined by whether the spare x, or spare o playing piece (left over when one player starts first) is placed on it.

Example: X starts first;

The spare o piece is placed on the spare space on the board.

This makes the O signal go logic high. This feeds into pin 9 of U5 wired as data selector. (Also it resets the o start randomizer, pin 8 of U3 goes low, so all LED's go out). This has the effect of routing the EVEN data at pin 5 of the Parity IC, through to the /compute line that activates the ROM, on the ROM's pin 22.

Consider, initially there are no player pieces on the board yet in the playing area. Zero pieces are an even number for the Parity IC, so the EVEN output of the parity IC, routed to the /compute line, is logic high. Therefore no computation is performed and no board LED's are lit.

Now the Human x player starts the game by placing their piece on the board's player area. Then the EVEN output of the parity IC goes low as there is one piece on the board. This causes /compute line to go low , the ROM gets enabled and the state of the board is examined and the appropriate LED's lit for o's move. Once the o piece is played, the EVEN line pin 5 of the parity IC returns High and computation is disabled until after the Human x plays their next move and the state of parity changes again.

Example O starts first;

This is more interesting. In this case, the spare x piece was placed initially on the spare board space. As noticed previously this stops the randomiser's oscillator, leaving an LED lit for o's first move.

The “X” signal from the player board is logic high and feeds to pin 12 of the data selector U5. This routes the ODD output of the Parity IC (pin 6) to the ROM’s /compute control line. However, in this case, initially, when there are no player pieces on the board the ODD signal at pin 6 of the Parity IC is in fact low. This enables computation, however, with no pieces on the board, all the address lines are zero, this address has FF in the un-programmed state for that ROM location, but it could also be zero if programmed that way. As noted, if the 74HCT42 is presented with the 4 bit value F or 0, it will not light any board LED’s because F is invalid and output corresponding to zero from the 74HCT42 is not used. (Recall outputs 1 to 9 of the 74HCT42 were used but not output 0).

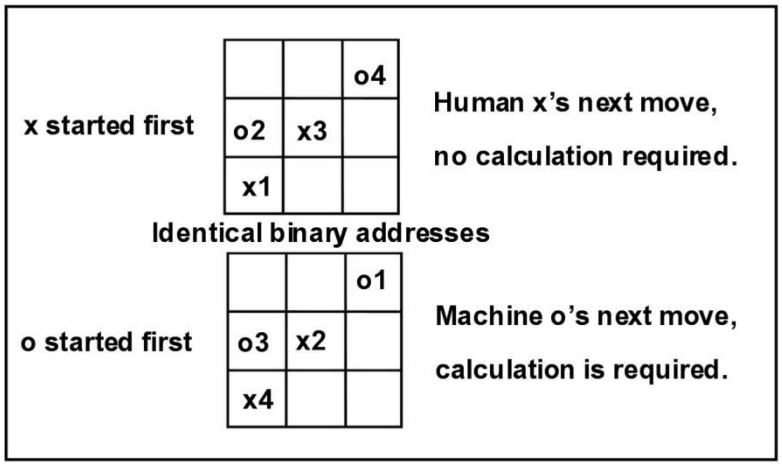
Then when the o piece is placed for o’s first move, an odd number, 1 piece is on the board and the ODD output from the parity IC then goes high, disabling computation. Player x makes their next move and then the ODD line falls low again, and so on during the game, when it is the correct time for computation to be performed.

A question could arise, better answered now:

Since the ROM is programmed only to produce valid output value (which lights one of the LED groups up, in one of the 9 squares), corresponding to some specific pattern of player pieces on the board (which is a specific generated address) why is it necessary to have the game sequencer circuitry at all ? Because invalid addresses would/could be FF or 00 in the ROM and therefore no LED’s would be lit for an invalid address ?

The answer is, you would not need the sequencer circuitry if the game was just for a “Machine o starts first case” or just for a “Human x starts first case”.

Consider the two following board patterns with 1,2,3 & 4 pieces placed on the board in that order. The same final pattern results and generates the same address for the ROM. In one case though, x had started first, in the other case, o had started first. And the cases are such that the next move, the 5th piece placed could be x’s or o’s, but that would depend on who started the game. This is why the game sequencer with the Parity IC was required, to support both playing sequences regardless of who started the game first.

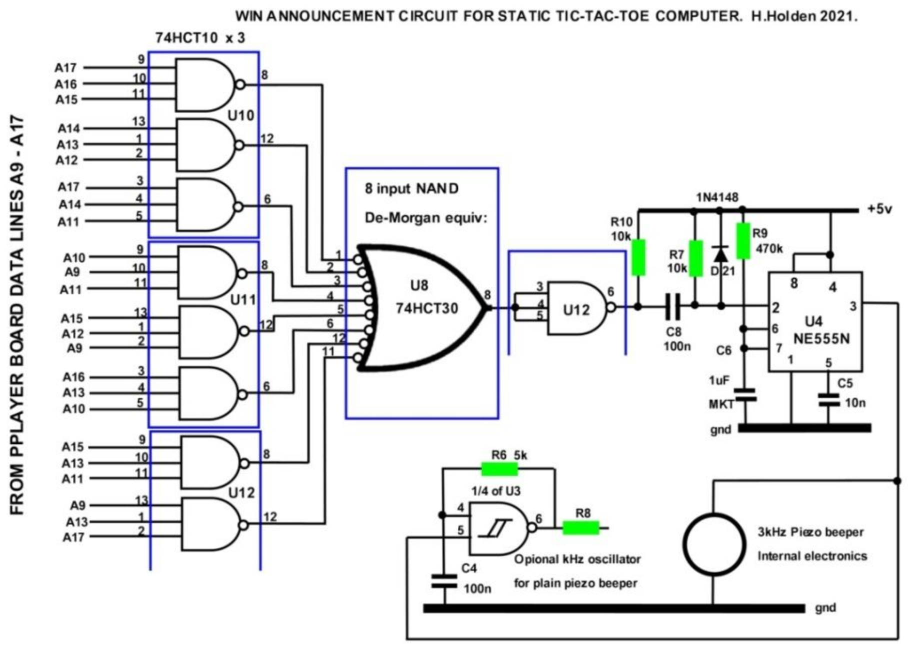


Game Sound System:

As mentioned when a human wins the game they likely say something. So the machine has been gifted with a voice of sorts. When it wins against the human player it sounds a beep.

The optional oscillator to drive the beeper was not used as I bought a beeper with in inbuilt, part 107-2397 from Element 14.

The schematic is shown below:


WIN ANNOUNCEMENT CIRCUIT FOR STATIC TIC-TAC-TOE COMPUTER. H.Holden 2021.
74HCT10 x 3
FROM PPLAYER BOARD DATA LINES A9 - A17
U10
U11
U12
8 input NAND
De-Morgan equiv:
U8
74HCT30
U12
R10 10k
R7 10k
1N4148
D21
R9 470k
+5v
C8 100n
C6
1uF MKT
gnd
U4 NE555N
C5 10n
R6 5k
1/4 of U3
R8
C4 100n
Opional kHz oscillator for plain piezo beeper
3kHz Piezo beeper Internal electronics
gnd

Of note, no beep functions are assigned to the Human player because:

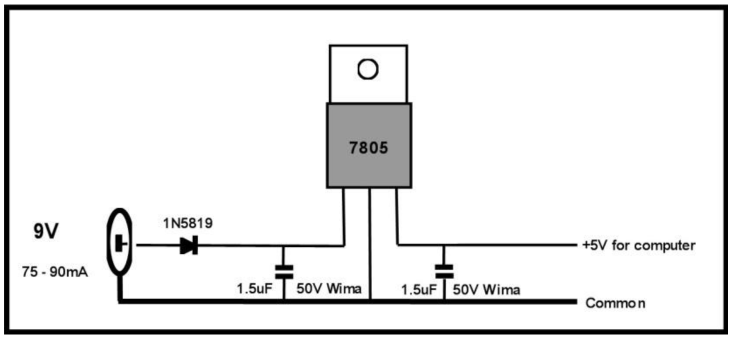
1) The human can make a sound themselves if they want. They might, especially when the machine not only prevents them winning, but it insults them further by beating them with the smallest human error or lack of attention. At that point the Human might emit an expletive.

2) Since the Human can never beat the machine, therefore the requirement for some sort of automatic announcement for the Human not required.

For the above reasons it is only required to examine the O data from the player board (not the X data) for the 8 possible configurations of O alignments that occur with a win.

Game Power Supply and Power Supply Options:

I was not exactly certain whether or not the games current consumption might exceed 100mA, so I built it with a 7805 regulator. As it turned out it, it varies in the range of 75 to 90mA. Currently I am running from a 9V 0.5A rated Jaycar plug pack. It would have been ok with a smaller 78LO5 regulator. The current briefly peaks over 100mA when the Piezo beeper sounds.



Due to the power consumption being low, the game can be powered by a 9V Alkaline cell which has about a 500mAH capacity or a 9V lithium cell with a 1200mAH capacity. However, in this design, inside the case, there is plenty of room for 6 AAA cells in a holder to significantly increase the running time.

In the case of battery operation, it would still be wise to leave a 1N5819 in circuit (to prevent reverse polarity mishaps) and also a 5V LDO regulator would be better to get the most out of the battery life.

2) MECHANICAL ENGINEERING, MATERIALS, INDUSTRIAL DESIGN & ARTWORK:

Due to the fact that the Tic Tac Toe game is such an ancient game, I could imagine people playing it hundreds of years ago with wooden blocks with x's and o's on them. Or just with a pen & paper.

In a board game format it could be made as a folding cardboard player board. The only problem with games using player pieces is that the pieces tend to get lost over time. As we know, Tic Tac Toe is now a computer game which can be played on mobile phones. So, like many simple games, it has become an "App".

I decided I wanted a compact game with a quality look to it. Possibly looking like an elegant product from the 1920's or 1930's era and made to physically last. Popular materials then were plastics such as Bakelite. These sorts of materials are harder to get nowadays, so I decided to go with a unit constructed from 10mm thick gloss Black acrylic panels with white paint filled engraved markings. Also to use a hinged lid so that the player pieces could be stored inside the game to reduce the chances of them ever getting lost.

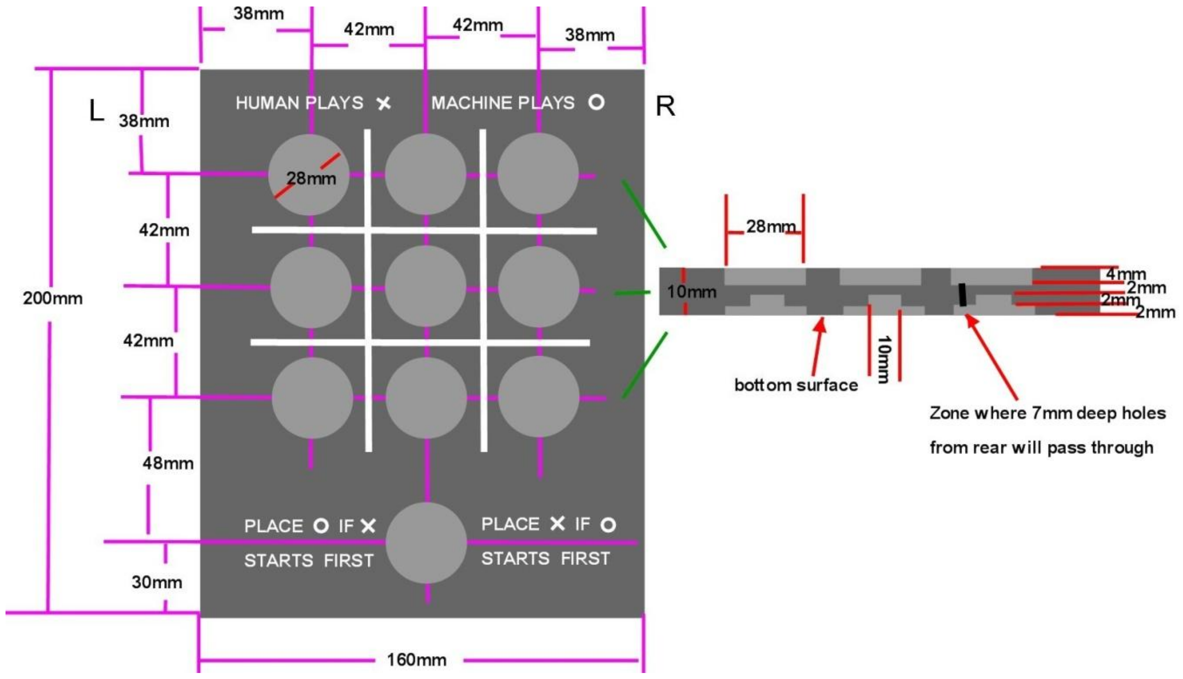
In addition, as noticed previously, I wanted the game to work without power, for two Human players. This idea is in the spirit of some video games, where you can choose to play a friend, or the machine if you are on your own with the machine and nobody else is about.

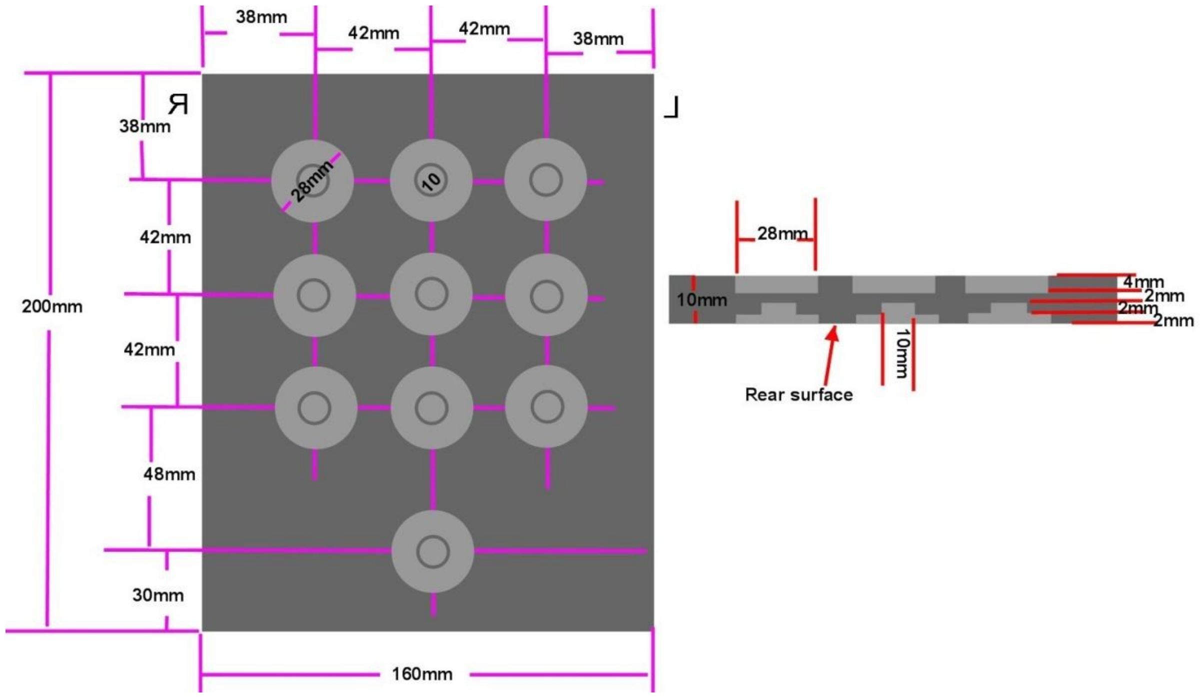
10mm thick acrylic has one advantage in that it is relatively easy to tap a coarse thread into it and a good sized screw is 4-40 UNC for this application. So I tapped long threads, approximately 15mm, into the frame to secure the top & bottom panels. For the initial machine I used a clear lightly tined 6mm thick bottom panel so the internal electronics are visible to the observer.

The unit can easily be made from any color 10mm Acrylic panels with varying color combinations. It could also be made from a number of other plastic types with variations such as Mother of Pearl or Tortoise Shell patterning.

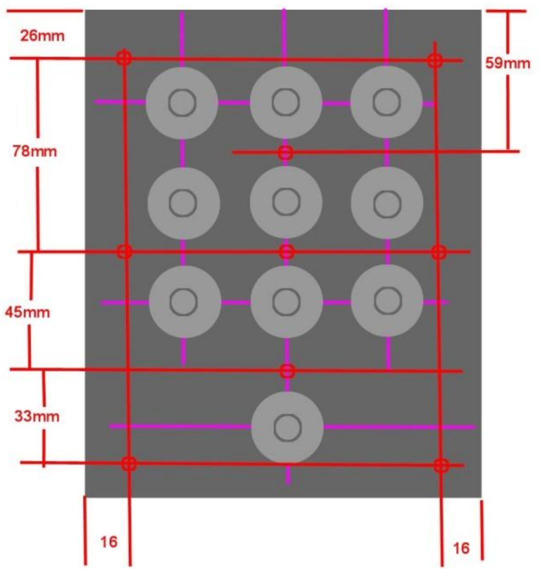
A local plastics company (Sunquest Industries) routed and engraved the acrylic panels for me. I tapped all the required holes with the 4-40 UNC threads. I specified pilot holes. Also, fitting the hinge to the lid required that I machine some 4mm diameter x 10mm long bass inserts with 2mm x 0.4 tapped holes. The reason being that small diameter fine thread pitch screws do not do very well directly into the acrylic. One option would be a pre-made threaded insert, designed for plastic, for this task.

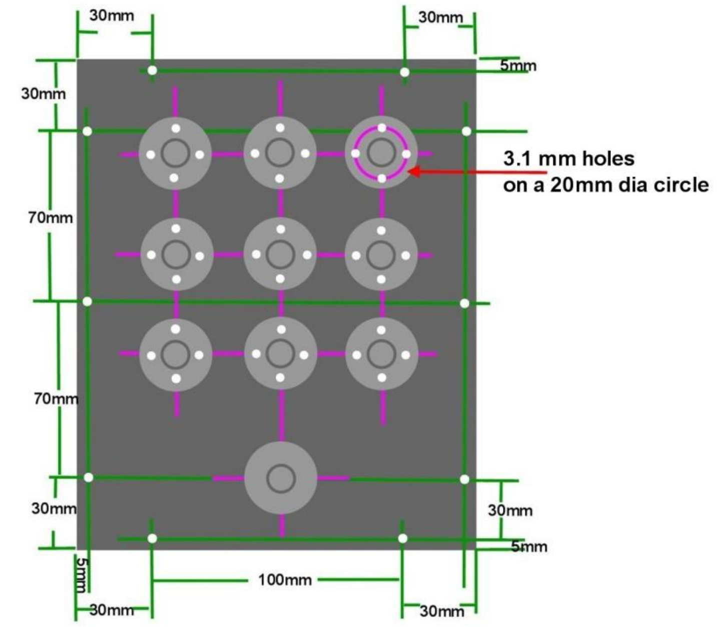
Below are the dimensions of the main player panel. The most complex panel is the top one. It has recesses for the player pieces. Also holes in these recess for the LED's and recesses on its bottom surface for the Hall devices and the LED's:



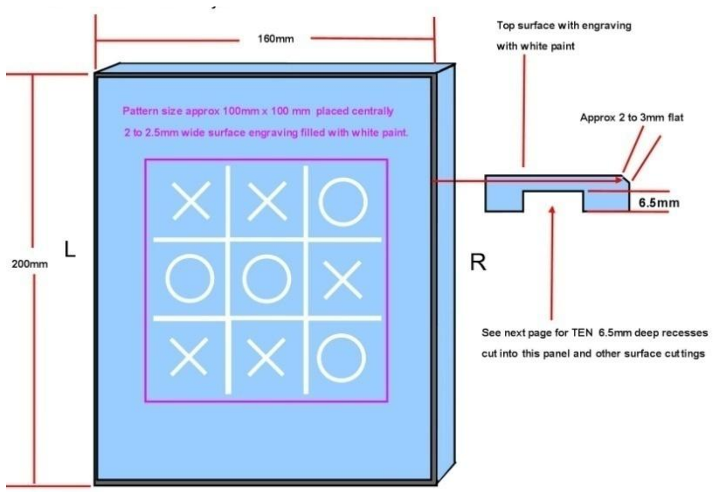


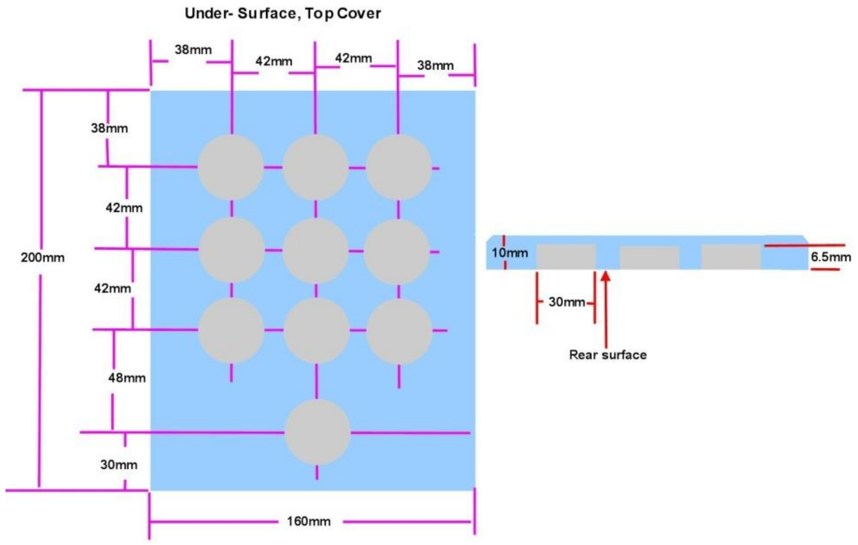
The diagram below, top left, shows the positions of the screw holes that mount the Encoder Board to the bottom of the player panel. These were drilled 8mm into the material and tapped with a 4-40 UNC thread Taper then End tap so as to receive some 4-40 UNC threaded stand offs, of the sort used on computer connectors. These are sold at Jaycar. These allow the Encoder board to be attached and for the Static Computer pcb to be fitted on top of it.



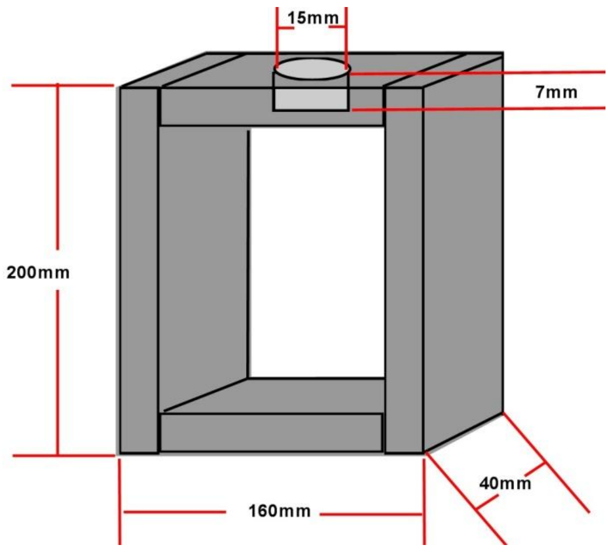


Top Cover Details:

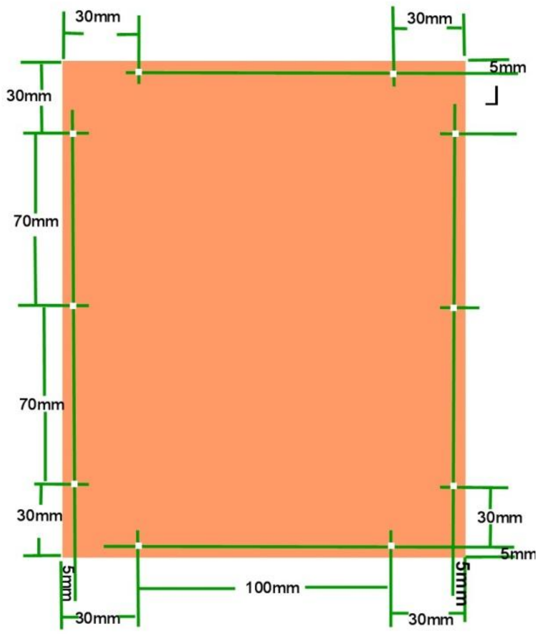

160mm
Top surface with engraving
with white paint
Pattern size approx 100mm x 100 mm placed centrally
2 to 2.5mm wide surface engraving filled with white paint.
Approx 2 to 3mm flat
6.5mm
200mm
L
R
See next page for TEN 6.5mm deep recesses
cut into this panel and other surface cuttings



The frame is simple with a recess at one end for the DC connector. The holes for this one were marked using the top and bottom plates as a template (these had pilot holes). If the machine was to be only battery operated, the ON-OFF switch would go in this recess instead.



Bottom Panel:



3) SUPPORTIVE DATA.

The question could be asked: How many machine responses would be required for;

A game where the Human x starts first and the machine o responds second ?

A game where the Machine o starts first and the Human x responds second ?

7	8	9
4	5	6
1	2	3

1) The Human x starts first case:

Ignoring issues of the boards symmetry and mirror and rotational images (which may have some equivalence if software manipulations were used): If the board has each of its squares labelled 1 to 9 and x starts first, the game beginning has 9 different starting possibilities.

Lets say x starts in square 1, then o has 8 remaining squares to choose from. The response here can be limited to taking the central square if x had not taken it initially, or for the case where x takes the center square intially; o can take the same initial corner square. The game sequence then depends very much on x's second move. O's first repsonse though could be called a "general start" because it can be sterotyped to be the center square if not initially taken by x, or in the case of x taking the central square, a specific corner square. Therefore this can limit the machine response initially to two choices.

After the initail start, the game sequence of "x starts first" can be sorted into groups of solutions of the form x1,2 and x1,3 through to x1,9. Where the first number represents x's initial move location, and the second number represents x's second move after the machine o's first response.

In the example above, if x's first move is square 1 (a corner square) there is no game sequence of x1,5, because o's first response is to take the center square, so later it is no longer an option for x.

After o’s initial response, 7 squares remain as a choice for x. This means that for each game start up sequence , x starting first, o responding and then x making its second move, there are 7 early board patterns that occur intially. At this point two x’s are on the board and one o and it is o’s turn to choose next. Analysis at this point shows that to complete the game, 9 responses are required for each of the initial 7 board patterns to allow for all of the mistakes x could make choosing a square.

Therefore with the 9 initial starting states and 7 early board patternes, 9x7 or 63 charts are required. Each of these 63 charts containing 9 data points (or machine responses) to process the game. The number of responses required by “o”(the computer) then could theoretically be in the order of (9 x 63) + 2 initial o responses = 569.

However, there is a saving grace. It turns out once the game has begun, duplicate patterns of x’s and o’s appear which had a different starting sequence. They occur early in the game where two x’s and the one o end up in the same locations, then the entire group of 9 responses are duplicated. Later on in the game, for the final moves, board pattern and data duplications also occur.

For this game, the way played here, the required number of computer responses after duplications were deleted, for the “x starts first” case is 285.

A list of the charts is shown below:

LIST OF CHARTS IN SUPPORT FOLDER:

X (Human) starts first charts. 63 Charts. Generates 285 unique machine responses.

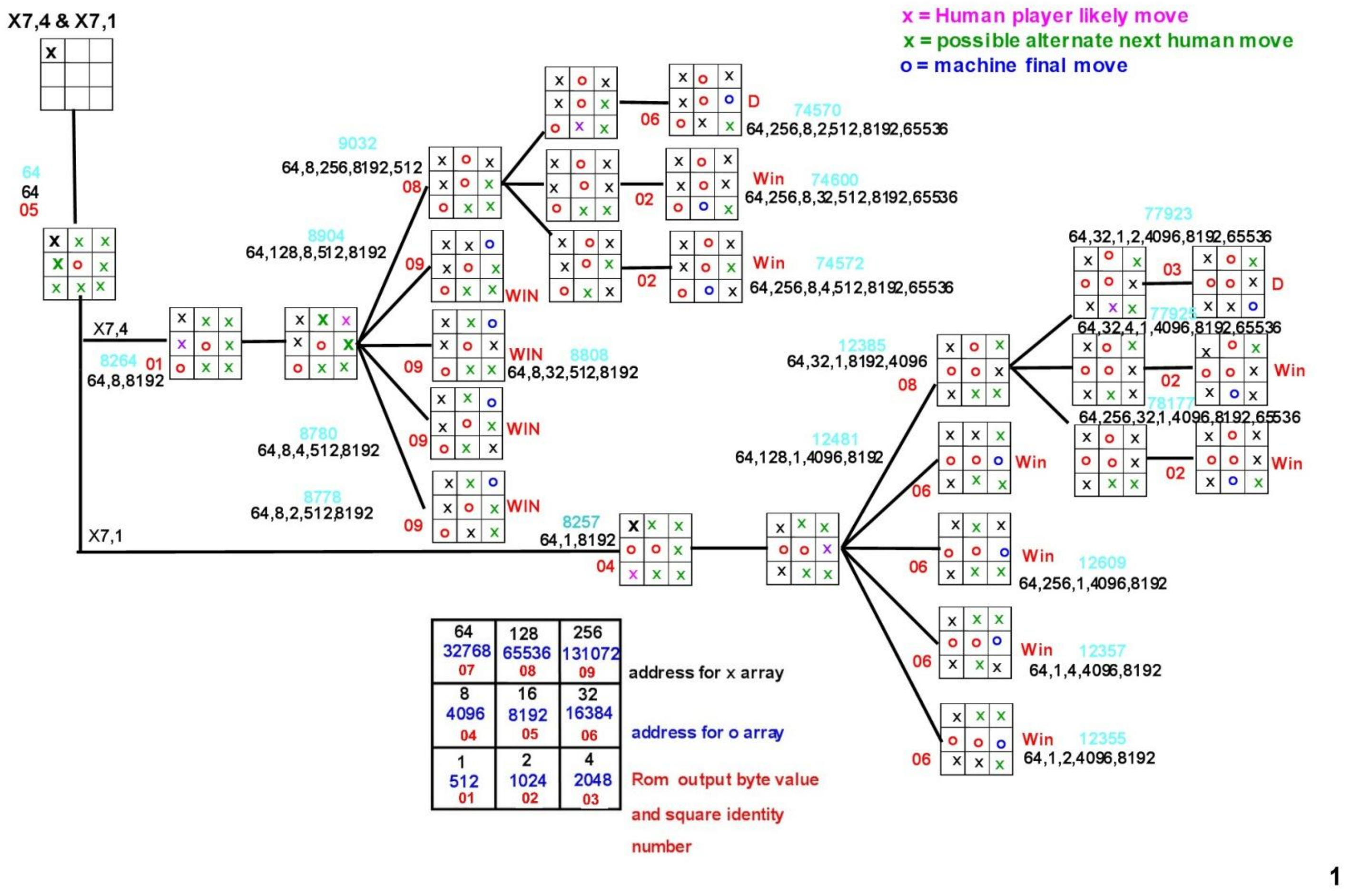
x1,2	x2,1	x3,1	x4,1	x5,1	x6,1	x7,1	x8,1	x9,1
x1,3	x2,3	x3,2	x4,2,	x5,2	x6,2	x7,2	x8,2	x9,2
x1,4	x2,4	x3,4	x4,3,	x5,3	x6,3	x7,3	x8,3	x9,3
x1,6	x2,6	x3,6	x4,6	x5,4	x6,4	x7,4	x8,4	x9,4
x1,7	x2,7	x3,7	x4,7	x5,6	x6,7	x7,6	x8,6	x9,6
x1,8	x2,8	x3,8	x4,8	x5,7	x6,8	x7,8	x8,7	x9,7
x1,9	x2,9	x3,9	x4,9	x5,8	x6,9	x7,9	x8,9	x9,8

O (Machine) starts first charts. 72 Charts. Generates 560 unique machine responses.

o1,2	o2,1	o3,1	o4,1	o5,1	o6,1	o7,1	o8,1	o9,1
o1,3	o2,3	o3,2	o4,2	o5,2	o6,2	o7,2	o8,2	o9,2
o1,4	o2,4	o3,4	o4,3	o5,3	o6,3	o7,3	o8,3	o9,3
o1,5	o2,5	o3,5	o4,5	o5,4	o6,4	o7,4	o8,4	o9,4
o1,6	o2,6	o3,6	o4,6	o5,6	o6,5	o7,5	o8,5	o9,5
o1,7	o2,7	o3,7	o4,7	o5,7	o6,7	o7,6	o8,6	o9,6
o1,8	o2,8	o3,8	o4,8	o5,8	o6,8	o7,8	o8,7	o9,7
o1,9	o2,9	o3,9	o4,9	o5,9	o6,9	o7,9	o8,9	o9,8

An example chart is shown below, one of the 63 supportive charts (of the x starts first case).

 I constructed these by hand to examine every possible human move and select an appropriate machine response. These are all provided in the support folder. Sometimes two charts are drawn on the one page as below, charts x7,4 and chart x7,1.



About these charts:

The "Win" notation on the chart refers to the machine winning the game, D refers to a draw where the human was unable to beat the machine. The green x's on the chart are merely placeholders where the Human x could play on a subsequent move.

The numbers in pale blue show the decimal address generated by the Encoder board pattern of x and o playing pieces on the board. These decimal numbers were converted into Hex numbers to program the Rom. The converted numbers are in the Word file called "Data List" I the support folder. The numbers in red, 05, 01, 04, 06 etc are the byte values programmed into the ROM at those address locations.

The red numbers 01, to 09 in the address box also refer to the location of the place on the board. So if the response is to light the LED's on square 5, the output byte from the ROM is 05.

Consider the human x starting at position 7 (in the chart example above) and making its second move onto square 4. The chart (the upper one and its pathway) is labelled x7,4. Though the human could make its next move differently as in any of the x7 series charts onto positions 1,2,3,6,8 or 9, these are all accounted for in the other x7 charts.

x's initial move generates the decimal address 64. And its o's turn so the machine computer activates and it takes the central square. Notice that this represents a "general start" to game play because it is the same start for sequence x7,1, x7,2 etc and in fact all the x starts, where the Human x does not take the board center as the initial move. Notice the x7,5 chart does not exist on the list of charts, as o took square 5 early in the game as a "general start".

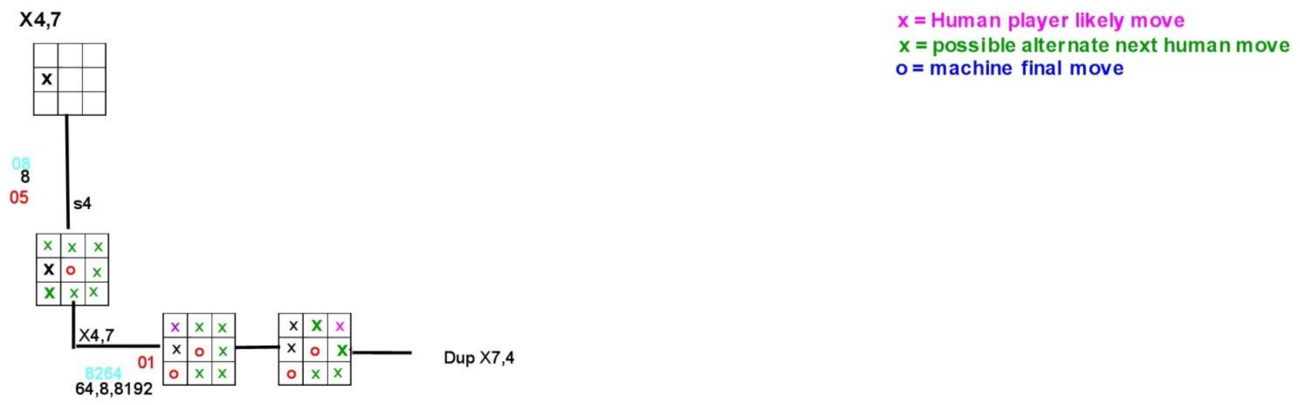
Then x plays square 4 as its second move (in this example of the sequence x7,4) and that is drawn as a purple x. This generates the address 8264 decimal and the machine, in response, takes square 1 in response to that address because an "01" is programmed at that address in the ROM.

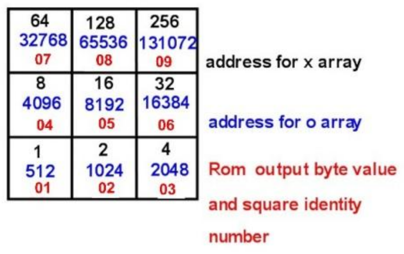
Ignoring the general start moves, there are nine responses from the sequence x7,4 as there are for the sequence x7,1.

As can be seen from the charts, there are many opportunities for the Human player x to make a mistake and only one pathway to a draw with the machine. If the Human does make a mistake the machine takes the appropriate square to Win and it sounds a beep when it does.

Most of the data points therefore are to allow for the many variations of mistakes that the Human player can make, so that the machine (which never makes an error) can take advantage of this.

Now consider the chart x4,7 and the issue of early duplicates:


X4,7
08
8
05
s4
X4,7
8264
01
64,8,8192
Dup X7,4
x = Human player likely move
x = possible alternate next human move
o = machine final move



As can be seen from the x4,7 chart above, although the game started initially differently to the chart sequence x7,4 with x's initial move at location 4, it has converged to the same as the sequence x7,4 early in the game play. Therefore there is a saving here in the total number of required machine responses.

In the "x starts first case" there were 28 chart duplicates out of 63 charts saving 252 responses, thereby requiring less total machine responses. I also found that possible duplications could be increased by settling on a similar style of game play.

Similar duplications of data appear later in game play for the final responses inside the chart, which match the results in other charts. This futher reduces the required number of machine responses. This occurs because game board patterns converge on the same result, but via different initial playing sequences.

In the attached Word document, called Data List, it lists all of the addresses that I found from the hand crafted charts (in Hex number format). Where the address is duplicated (appears for a second time or more) I have made a note of that on the list.

The charts show that every mistake made by x is accounted for.

When the Human x starts first, the second player, either Machine (or a human) playing o, is "pushed around" by the playing strategy of x. Many of the responses in this case by the Machine o are to prevent being beaten by blocking a winning Human move. The starting "player" therefore has a significant advantage. This is why, to be fair, the game has to allow either player to start it first, so as to average this bias out over a large number of games played.

2) The Machine o starts first case:

When the machine o starts first more charts (72 in this case) are required and many more unique machine responses are required.

The number of responses is a little affected by the playing strategy. Here the starting player or machine has the advantage and can largely dictate the course of the game. Even setting traps where if x makes a poor initial move, x can quickly be in a "loss" situation and be beaten by the machine.

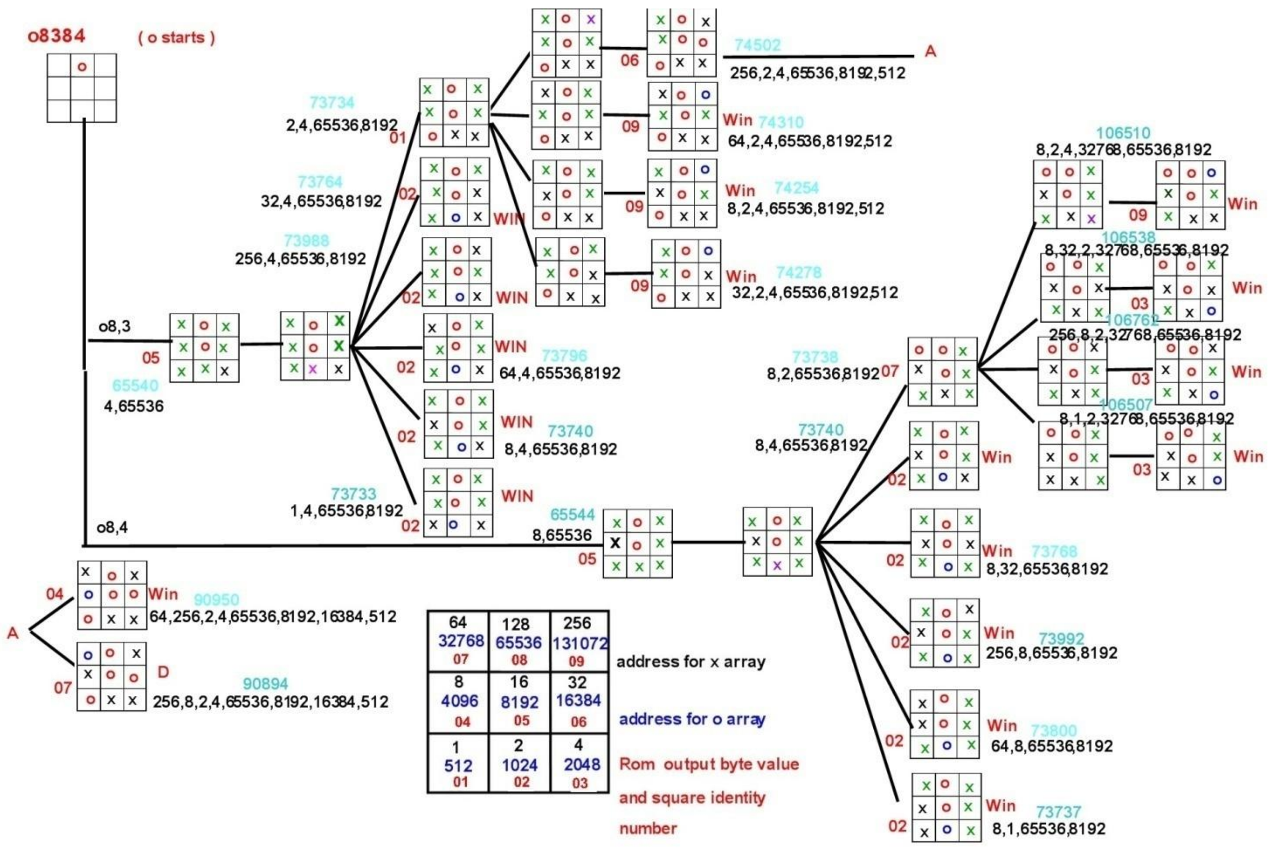
The game here has been optimized to catch the Human x out at every opprtunity when the human plays an error. Every possible mistake by the Human x has been analysed and responded to. The Human x has to be concentrating and the best he/she can hope to achieve is a draw with the machine.

Considering the Machine o starting first, it chooses a starting square and then the human player x responds. The game sequence here, on the charts, for one example below, is tabulated as o8,3 where the Machine o starts on square 8, and the 3 refers in this case to the Human x response taking square 3.

The chart below is for playing sequences o8,3 and o,8,4.

The machine o starts first on square 8 (selected by the Randomizer). As can be seen more total data points on the chart are generated than in the "x starts first case".

Looking at playing sequence o8,3 notice how there is only one correct set of responses the Human x can make to end up with a draw (shown on chart lower left hand side).



All the other patterns of play end up with the Human x losing or one Draw and the Machine mostly winning. Also, looking at the other game sequence o8,4, the Human x made a very bad move at the beginning and was destined to have the Machine win in all cases after that.

Depending on the game play, there can be either 11 or 13 required machine responses for each starting sequence of “o starts first”. Just taking an average there could be as many a 10x9x8 responses. Meaning without duplicates, there could be as many as 720 or more required responses from the machine for the “o starts first case”. Fortunately, again, due to early and late game play duplications, the required numbler is lower.

But, there are not as many whole chart duplicates as in the “x starts first case”, roughly half the number at 15 duplicates. Still, this saves over 100 required machine responses.

The total number of machine responses for “o starts first” with the game stategy used here turned out to be 560, nearly twice that for “x starts first” at 285 required responses.

Therefore the total number of unique programmed responses required to ensure both the “Human x starts first” and the “Machine o starts first” cases are supported are 845 with the game play strategy used in this design.

****