Single Chip Chess Computer

SC3

'Chess is the intellectual game *par excellence*.' - wrote Newell, Shaw and Simon in their paper published in the IBM Journal of Research and Development in 1958 which traced the development of digital computer programs that play chess. They believed that the efforts to program chess provided an indication of the then current progress in understanding and constructing complex and intelligent mechanisms.

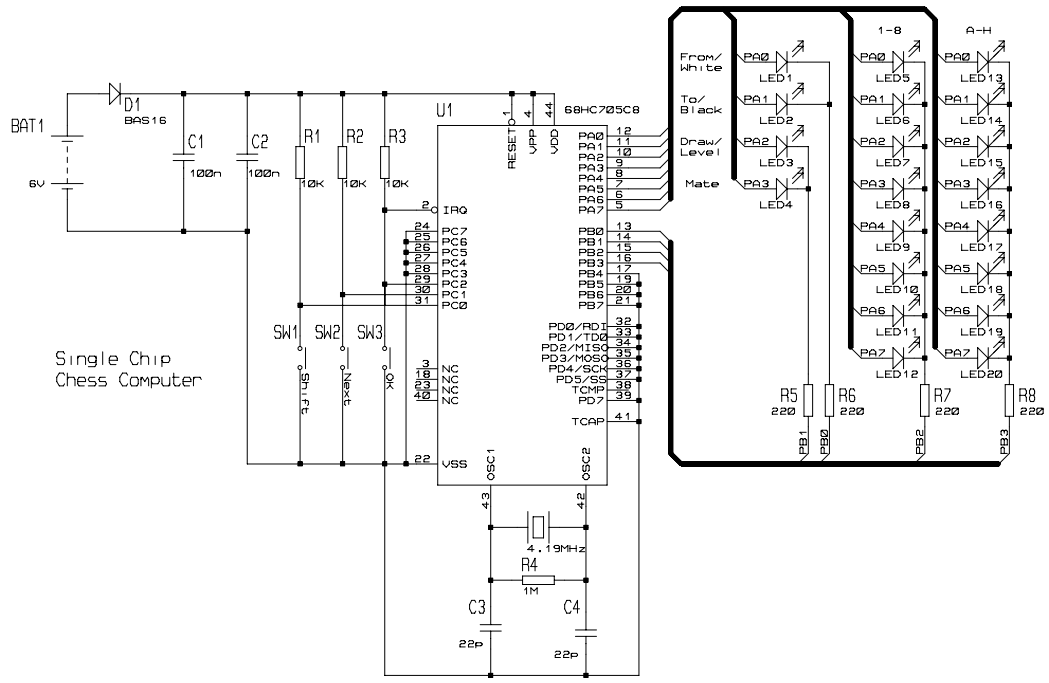
I think this probably holds true today, as just recently a computer called DEEP THOUGHT has won a chess game against a Grand Master chess player. However, it took one of the fastest parallel computers in the world and many man years of programming effort to do it.

Chess programs have probably been written for all types of digital computers ever constructed in the last 40 years, from 4-bit microcomputers to Crays and all types in between. Presented here is a chess computer based on the 68HC705C8FN microcontroller from Motorola. The 'FN' refers to the type of package the micro controller comes in which in this case is a plastic-leaded chip carrier or PLCC. It's a surface mounted component and allows the project to fit into a match box! It is not necessary to construct the project this way but it will certainly be a novelty if you do.

Of course if you decide to construct the project to fit into a match box (or something similar) then a normal chess board must be used in conjunction with it in order to play against it. Alternatively you could build it into your own chess board and thus have a fully self contained chess computer like most of those available commercially. I plan to build one with a glass top and legs so that it can double as a small coffee table. I have called it the Single Chip Chess Computer or *SC3*.

Circuit

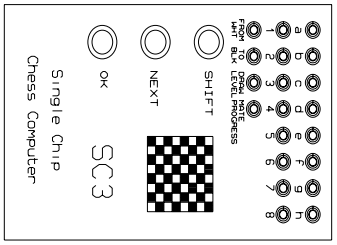
The SC3 consists of three switches for entering moves and four banks of LED's for displaying moves and status information. There is no on/off switch since the microcontroller is placed into its low power, or STOP mode whenever a switch has not been pressed longer than about 8 minutes ago. During STOP mode the on chip clock is stopped and power is only



SC3 Circuit Diagram

required to keep the static RAM from loosing its contents which is about 10μw. Only one of the LED's from each bank will be used at any one time, so only one current limiting resistor is needed per bank of LED's. The separate banks are multiplex to give the illusion of more than one LED being on. Debouncing of the switches is handled by software, and is incorporated in the same routine as the LED multiplexing. The multiplexing routine is called with parameters to indicate which LED's to multiplex. During multiplexing, the port to which the switches are connected is examined, and if a switch was already closed the routine waits until the switch becomes open. At which point a delay of several loop times is used to debounce the switch opening and the routine then loops waiting for a switch closure. When a switch being closed is sensed another delay for debouncing is initiated after which the routine returns with the identity of the switch which was pressed. A delay in the multiplexing loop determines the rate of display multiplexing which is about 7ms.

Switch SW3 is also connected to the interrupt request (IRQ) pin which enables the microcontroller to *wake up* from its low power



Front Panel

mode. Also, if you get tired of waiting for the computer to make its next move you can press this switch which interrupts the routine which is calculating the next move and forces the computer to use the best move it has found so far.

A 4.1943 MHz crystal is used in the clock circuit because they are readily available and the maximum clock frequency for this particular device is 4.2 MHz. This is divided internally by two giving a bus clock of 2.097 MHz. Capacitors C3 and C4 and resistor R4 are used to ensure reliable starting of the oscillator.

All unused inputs are tied low so as to prevent increased power consumption should these inputs otherwise stray up and down. Diode D1 is used to reduce the peak voltage of fresh batteries by about 0.7 volts and also protects the circuit from inadvertent reverse connecting of the batteries.

```
NCRDIR0  ldx   PTO      index TO position
         ldx   DIRN,x   get next board position
         bmi   NCRNDIR1 if off the board, next direction
         lda   B,x      see what's on the board here
         beq   RET_MOVE if blank, move ok
NCRNDIR1 ...
```

Listing 1

Firmware

The main building blocks of a chess computer are CPU, RAM, ROM, a timer and an appropriate input/output mechanism. Apart from the lack of processing power in the CPU a typical microcontroller makes an ideal chess computer. Depending on the algorithm used to decide the next move for the computer it has been shown clearly that a computers chess rating increases in proportion to its processing power for a predetermined period of time per move. While we can't expect a chess computer based on a microcontroller to play at the level of Grand Master a reasonable level of play can be expected while a *bad* move is not necessarily a bug in the program. It is also clear that as more time is devoted to the calculation of a move the *better* that move is likely to be.

The firmware for the SC3 consists of approximately 6000 lines of assembly code so a detailed explanation of its function cannot be undertaken here unfortunately. The program was originally written in C in order to test and debug the algorithms and then translated into 6805 assembly language. Even so using this method still required about two years of part time programming to complete. Why is the program so long? The answer, in a word, is *speed*. Due to the nature of the algorithm (described below) for generating and searching moves in order to find a good one, it is very important to make it execute as quickly as possible. The most time consuming part of the algorithm is the move generation. Because determining the next move is certainly the most visited part of the program it must execute reasonably quickly and efficiently. The board is represented by 64 memory locations with the pieces for the computer represented by positive integers and the pieces for the opponent represented by negative integers. A vacant square is represented by a zero.

With this in mind, the generation of the moves for the black and white pieces has been separated, as has the generation of capture moves as opposed to non-capture moves (also explained below), and where possible loops have been *unrolled* and code placed

in-line. A sample piece of code for generating a rooks move is shown in listing 1.

`NCRDIR0` is where program flow comes when checking for non capture moves involving rooks. `PTO` is the last destination square tried for this rook. `DIRN` is a table of board locations for moving one square *north* from the current position, with negative values indicating that a move in that direction would be off the board. If this is the case, a branch is performed to a similar piece of code which tests for moves in a different direction for this piece. If the destination square is vacant then this is a valid move in as much as ensuring that the king has not been place in check because of this move.

Both the C and the assembly versions are available on floppy disk if you are interested in understanding the code in more detail.

The SC3 has eight levels of play which correspond directly to the amount of time it spends calculating the next move. However if the score associated with the latest search is as good as the score from the previous search (one half move less deep), and of course the move chosen was the same, then the search is immediately stopped. The assumption being that it is highly likely that the best move has already been discovered to the best of the programs ability and that it is pointless wasting time searching still further until the time allotted at the level chosen has elapsed. The SC3 searches for about 11 seconds times two to the power of one less than the level of difficulty entered, unless the above condition becomes true. For example, the maximum search time on level 3 is 44 seconds.

The SC3 uses what is known as a brute force method of calculating the next move. This method appears to be the method of greatest success historically in terms of playing human opponents. Its advantage is that every single move on the board will be tried and evaluated and thus the very obviously bad moves will always be eliminated. Other methods where some *intelligent* scheme of selecting which moves to consider were not as successful because it is extremely difficult to program such *intelligence* into a machine.

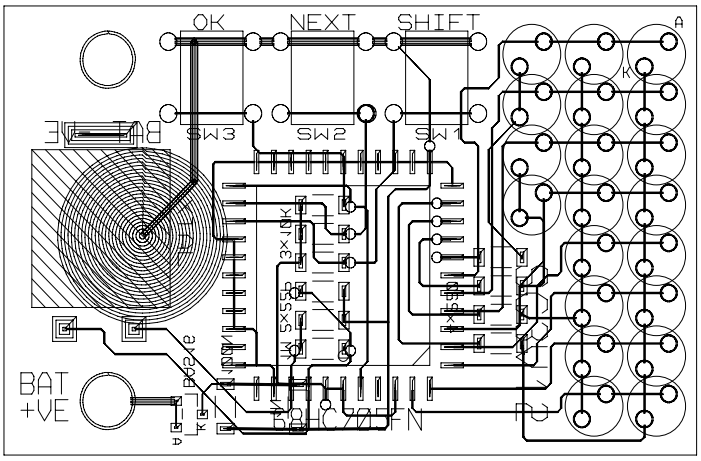


Diagram of All Layers

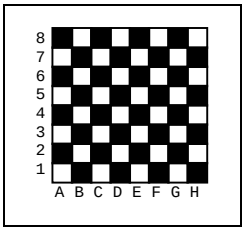
The method by which the SC3 uses to determine its moves is a derivative of a searching technique called the *minimax* algorithm. The minimax algorithm works by considering, for example, all the moves from the current board position, and for each of those moves, all the reply's to those moves, and so on, to a predetermined depth or level. At which point (a terminal position) a score is calculated based on how good the position is. In the SC3 the score is calculated by considering the material balance only. In theory it is possible, from the beginning of the game, to search all the moves until either a win, draw, or loss has been found. In practice this is impossible due to the shear magnitude of the moves that would need to be searched. For example, if on average there are about 40 moves available in any given position then searching to a depth of only 10 levels (or ply) would require the generation of 40^{10} or 10^{16} moves!

The minimax algorithm gets its name from the way it works. That is, the computer will choose the move leading to the position at the next level with the *maximum* score, knowing that where it is the opponents turn, a score equal to the *minimum* score of any of its successors is chosen.

The minimax algorithm is a *depth-first* search and by its nature the memory requirements grow linearly with the depth of the search as opposed to *breadth-first* searching, for example, where the memory requirements grow exponentially with the depth of the search. This is an important consideration because of the limited amount of RAM available on the 68HC705 for this application.

The actual algorithm used, however, is a derivative of the minimax algorithm call the *alpha-beta* search algorithm. This algorithm is based on the fact that many paths within the search tree constructed by the minimax algorithm need not

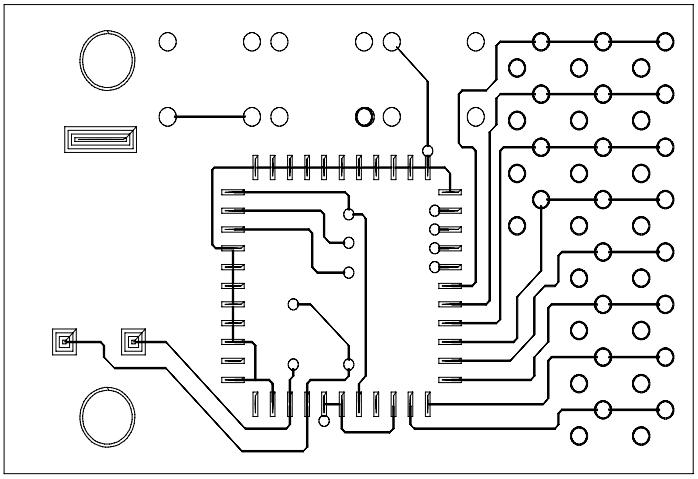
have been examined because they will have no effect on the outcome of the search. The decision whether or not to search a particular branch of the tree is based solely on the numeric value of the score calculated at the terminal nodes. It has been demonstrated that the alpha-



beta algorithm requires only six times as long to search the next level in chess, compared to the minimax algorithm where it is dependent solely on the number of moves at a given position.

Because of the nature of the alpha-beta search algorithm, the order of moves visited will impact heavily on the number of positions searched and thus the duration of the search. Most chess programs will generate all the moves at each level and then sort them in order of value defined by a comparatively quick evaluation routine. For example, a queen captured by a pawn will rate higher than a pawn captured by a bishop and so forth. Since the 68HC705 has very limited resources in terms of RAM it is impossible to generate, store and sort all the moves at each level, so a compromise is made. It turns out that it is only necessary to store enough information to determine what the *next* move will be each time the move generation routine is called. Only nine bytes of information is required at each level to uniquely identify the move made in a particular position and the same information is of course used to restore the board after the move has been searched. Using this method it's not possible to directly sort the moves to improved the search time. However it is possible to improve the search time by dividing the move *generation* into two separate sections, the first of which generates all the capture moves and the second part generates all the non-capture moves. This sometimes involves scanning the board twice but the savings in search time easily make up for this. If there is a capture move in any given board position it is highly likely that based on the properties of the alpha beta search algorithm that the second scan of the board will not be required.

As described before the score evaluated at the terminal positions of the tree is based on the material balance. Whilst this is certainly an important piece of chess knowledge it is by no means the only piece. The SC3 uses a different scoring method at the root of the tree when the alpha-beta search returns more than one move with the same score. In fact, three different position evaluation functions are used to determine the move chosen at the root for search scores of equal value. One for the opening, another for the middle and another for the end game. These positional evaluation functions use a combination of centre control and mobility - the number of moves available in the given position.



Top Layer Artwork

Play

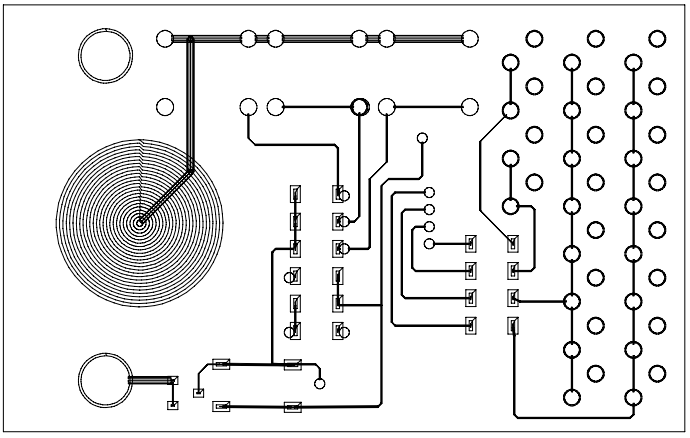
The SC3 users the common algebraic form of chess notation to identify the square *from* which the piece moves and the square *to* which the piece moves. Board co-ordinates are always referenced with respect to the human player. That is, square A1 is always the square closest to and to the left of the human player regardless of which colour they are playing.

When the SC3 is first powered up the green LED *DRAW/LEVEL* (LED 3) will be on and the *1* LED (LED 5) will be flashing. This is the prompt for you to enter the level at which you would prefer the computer to play. Pressing *SHIFT* (SW 1) will cause the *2* LED (LED 6) to flash, indicating level 2. Continuing to press *Shift* will cause the next corresponding LED to flash until the *1* LED again will be flashing. After deciding on the level you wish to play, press the OK (SW 3) button. The LED indicating which level you have chosen will stop flashing but remain alight while the *FROM/WHITE* LED (LED 1) will start flashing. This is the prompt for you to indicate

which colour pieces you wish to play. Pressing the SHIFT button at this point will cause the TO/BLACK LED (LED 2) to flash, indicating that you wish to play the black pieces. Pressing SHIFT again will cause the *FROM/WHITE* LED to flash again and so on. Having decided which colour you would like to play, press OK.

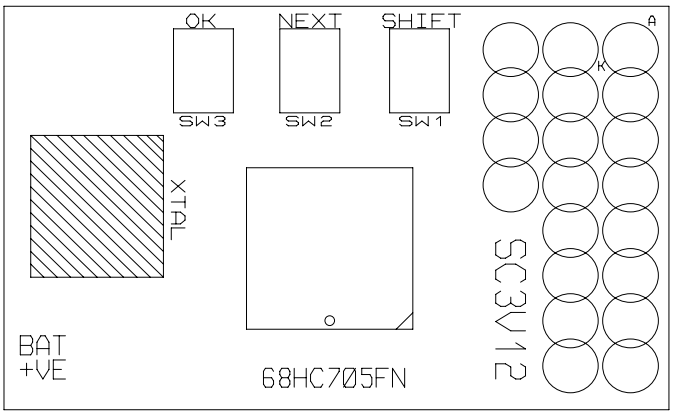
If you have decided to play the white pieces you can enter your first move now. The *A* LED (LED 13) will be flashing and the *1* and *FROM/WHITE* LED's will be lit. To enter, for example, *E2* to *E4*, press the *SHIFT* button until the *E* LED is flashing then press *NEXT* and the *E* LED will remain lit while the *1* LED will start to flash. Press the *SHIFT* button until the *4* LED is flashing. You have now entered the *from* part of the move.

To enter the *to* part of the move, *E4*, press the *NEXT* button again, and the *TO/BLACK* LED will light, and then follow the above procedure. If you make a mistake use the *SHIFT* and *NEXT* buttons to correct it. When you are satisfied that you have correctly entered your move, press *OK*. If you have not entered a legal chess move the *FROM/WHITE* and *1* LED's will light and the *A* LED will be flashing, enter your next move as indicating that the move just entered is not legal and so you must enter a legal move.



Bottom Layer Artwork

If you have entered a legal move the computer will now calculate its move. During this time the *MATE* LED will blink once ever 2 seconds to indicate that a move is being calculated. When the computer has finished calculating its move the *from* part of the move is displayed on the *A-H* and *1-8* LED's and the *FROM/WHITE* will be lit. Press the *NEXT* button to display the to part of the move and the *TO/BLACK* LED will light. Pressing *NEXT* will alternate between displaying the *from* and *to* parts of the move. Make the move on the chess board and then press OK. You are now prompted to enter your next move as before. In general, an LED flashing indicates the *group* of LED's to which the *SHIFT* button can effect.



Top Overlay

If you chose to play the black pieces the computer will be playing white and therefore will calculate and display its move first as described above.

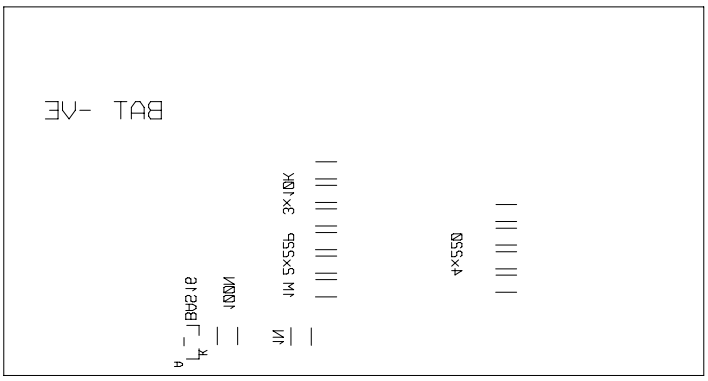
The computer may enter its low power mode while you are determining your next move. If this happens all LED's will be switched off but the state of the game is intact. To resume where you left off press the OK button and this will *wake up* the computer.

If the last move you entered is a winning move (the computer is in *check mate*) then the *MATE* LED will light. If the last move calculated by the computer is a winning move (you are in *check mate*) then the move will be displayed as usual and the *MATE* LED will light as well. If at any point during the game there are no legal moves but neither king is in check, then this is a draw and the *DRAW* LED will light. Again, pressing *OK* will begin a new game.

Construction

It's important to remember that the first component to be mounted on the top side of the board must be the 68HC705. I found it easier to solder the surface mount components on the bottom of the board first: C1-C2, R1-R5 and the diode, remembering to observe the correct orientation of the diode. Use a fine tipped soldering iron with a tip size no greater than one millimetre. Do not attempt any soldering with a larger tip as solder bridges will be inevitable.

Now the 68HC705 should be mounted on the top side of the board. Because of the 'J' shape of the leads it may be easiest to pre tin all the pads then place the 68HC705 on the board, observing correct orientation, and while holding firmly, reheat the solder so that it flows onto the pins but I have not



Bottom Overlay

PARTS LIST

Resistors
All SMT:

R1-3	10k
R4	1M
R5-8	220Ω

Capacitors
All SMT:

C1,2	100n
C3,4	22p

Semiconductors

U1	MC68HC705C8FN, preprogrammed with SC3
D1	BAS16 SMT diode
LED1-4	3mm green LED
LED5-20	3mm red LED

Miscellaneous
PC board, coded SC3V12; three pushbutton switches; two 4mm nuts and bolts; one 4.1943MHz crystal; fine tipped soldering iron; fine gauge solder; two 3 volt batteries; match box

Having soldered all the surface mount components, breath a sigh of relief and do a close inspection to ensure that there are no solder bridges. Next, since the crystal is mounted flat on the top side of the board, bend the leads at 90 degrees to its body and then bend them again at 90 degrees and cut so that there is enough length to be soldered. Ensure that the top of the crystal is clean and pre tin it with a small amount of solder. When mounting the crystal, solder the top of it to the board so as to ensure mechanical stability.

The LED's can be mounted next, ensuring correct orientation. Push them all the way down so that they are flush with the surface of the PCB. It is possible that the edges of some of the LED's may need to be filed down a little to ensure correct alignment.

Next mount the three push button switches so that the tops are level with the tops of the LED's. Some of the pins may be difficult to reach with the soldering iron from the top, assuming the pins do not go all the way through, but it is possible to reach them from underneath. That is, with the soldering iron between the top of the PCB and the bottom of the switch. Check everything again for solder bridges or misplaced components. If everything looks OK the batteries can be connected. It's a good idea to place a small piece of sticky tape over the pad closest to the large battery pad negative To make handling the batteries a little easier, it is a good idea to put some sticky tape around the edge so that they are functionally a single 6V battery. A bracket can be formed from a paper clip or reasonably stiff wire and the batteries bolted to the bottom of the board with the battery negative making contact with the board and the positive making connection with the bracket. If all is well LED 5 (the *1* LED) should be flashing and LED 3 (*DRAW/LEVEL*) should be lit. If not disconnect the battery and check everything again.

Check that all the push buttons are functioning correctly and that the appropriate LED's light up. At this point pressing the *SHIFT* button SW1 should light LED 6 and so on until LED 5 is again flashing. Press the *OK* button SW3 and green LED 1 (*FROM/WHITE*) should be lit.

If you are going to mount the SC3 in a matchbox, photocopy the page with the front panel layout, cut it out and stick it to one side of the match box. Next, drill holes for the LED's and cover the whole box with a piece of clear contact for protection.

Since there is not enough RAM to record all the moves of the game being played, and

because there is no way to take back a move once it has been entered, it is a good idea to record who was playing white, the level chosen for the game and each move as it is made, on a piece of paper. If you find yourself playing a crucial game and a mistake is made, you can always start again entering the exact same moves as before up to the point where the game was aborted.

Congratulations, set up a chess board, and good luck!

References

How to Beat Your Chess Computer, Raymond Keene and David Levy, Batsford Ltd.

Notes:

1. The holes for the LED's on the prototype PCB should be slightly enlarged to better accommodate the LED's lead size.
2. The spacing of the holes for the pushbuttons is a little too wide and should be narrowed.
3. The hole closest to the battery negative terminal associated with SW3 could be replaced with a surface mount pad to eliminate the possibility of a short circuit with the battery.