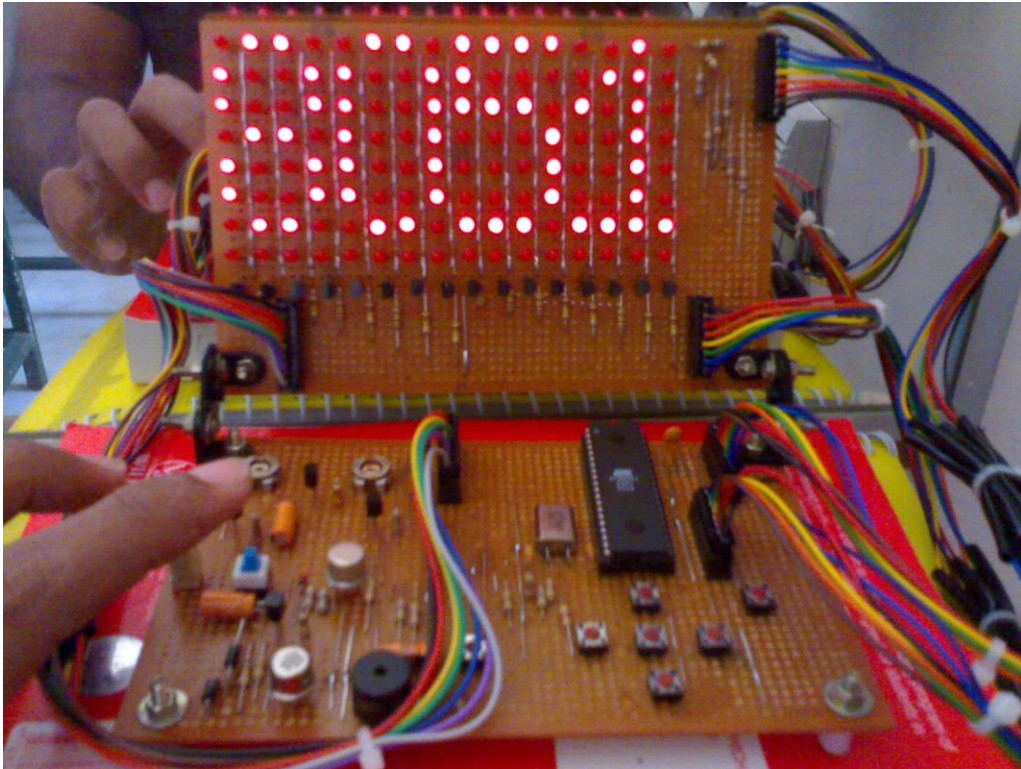


# C51Snakes

**Nokia inspired Snakes Game on 8051 platform!**



**A project by Sandeep Sasi  
([sandeepsasi.gelc@gmail.com](mailto:sandeepsasi.gelc@gmail.com))**

# **Contents**

<b>. Preface to the game .....</b>	<b>3</b>
<b>. Description of the Hardware .....</b>	<b>5</b>
<b>. Crystal .....</b>	<b>5</b>
<b>. Reset Circuitry .....</b>	<b>6</b>
<b>. Display ON / OFF delay control circuit .....</b>	<b>7</b>
<b>. 16 * 8 3mm LED Dot Matrix Display .....</b>	<b>8</b>
<b>. Keypad .....</b>	<b>8</b>
<b>. Description of the Software .....</b>	<b>10</b>
<b>. Coding Conventions .....</b>	<b>11</b>
<b>. Global Variables and Program Constants .....</b>	<b>12</b>
<b>. Display multiplexing .....</b>	<b>12</b>
<b>. System Control Functions .....</b>	<b>13</b>
<b>. Graphics .....</b>	<b>13</b>
<b>. Various modes and their associated source files .....</b>	<b>14</b>
<b>. Details of the Assembler .....</b>	<b>20</b>

## **Preface to the game**

Hello 8051 dudes,

Welcome to C51Snakes – Nokia inspired Snakes Game on 8051 platform...!

I was to do a project in embedded systems at college as a part of my academic exercise. I was in no mood to come up with something useful; but, I really wanted to do something innovative. Always, the Snakes game in Nokia handsets has fascinated in me. I spend hours playing the game. Months back, I decided to implement the game in J2ME. Unfortunately (or fortunately for now), I had to drop the project due to my tight schedule. Now, once again, I decided to undertake the venture - but this time on a more tightly constrained platform: an 8051 processor, LED dot matrix display, a simple 5way keypad and, to say the worst part, coding in assembly!

After days of sleepless nights, I finally had the game running. During the course of development, I made a number of enhancements to my original idea and now I have a game that is much evolved than the one which I had in mind at the beginning. A few highlights of the game are,

- \* As in the original game, the Snake grows in size as it snatches the eggs
- \* Eggs appear periodically on the screen at random pixels and will disappear if not snatched within a certain period of time
- \* The snake sees no boundaries at the edges of the display, will propagate in to any edge and enter from the opposite edge
- \* The Snake can be accelerated / decelerated on the fly
- \* There is also a dedicated delay settings mode to adjust the Snake's speed
- \* The game ends when the Snake collides with itself
- \* The total eggs snatched is then displayed as the game score
- \* The gaming console will hibernate if no user event occurs in 25secs!  
The game can be resumed by pressing external interrupt / master reset keys!

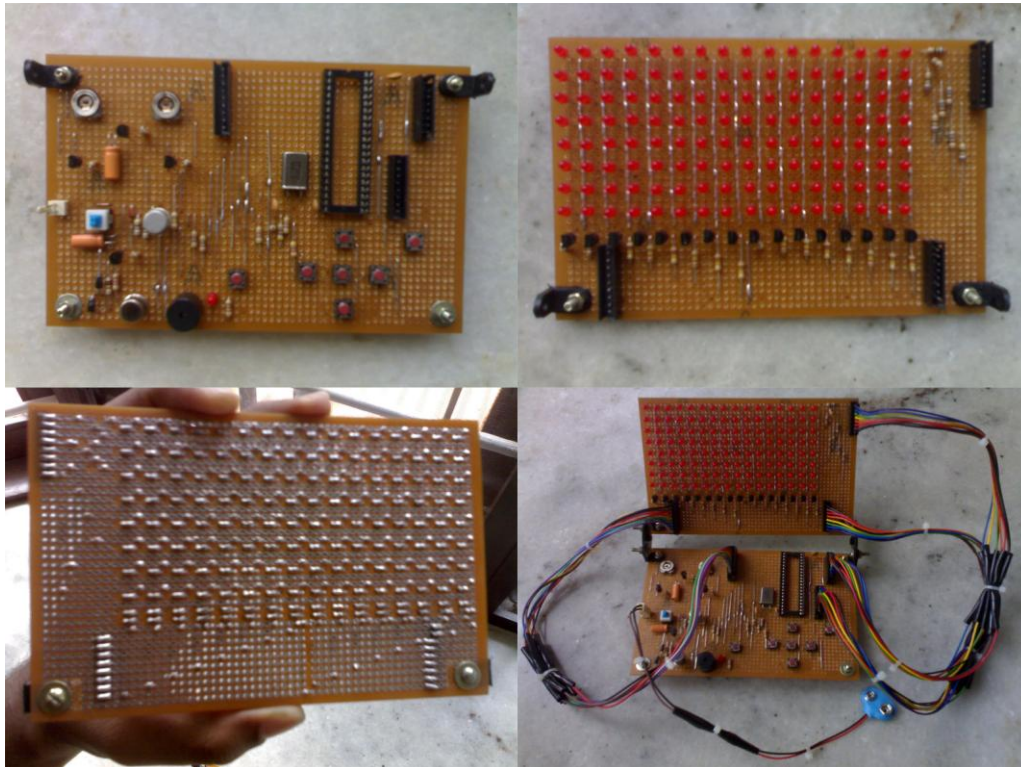
I have put in the best of my efforts to come up with a bug free code. Still, if you could find any bugs in the code or if you wish to comment on my work, please don't hesitate to mail me. I can be reached at [sandeepsasi.gelc@gmail.com](mailto:sandeepsasi.gelc@gmail.com).

So, happy gaming guys...!

Sandeep Sasi

## **Description of the Hardware**

The project is implemented using AT89S51 controller. The complete circuit diagram can be found in the directory /C51Snakes/ckt in the archive C51Snakes.zip. Here, I shall explain the important sections of the circuit that may require your attention.

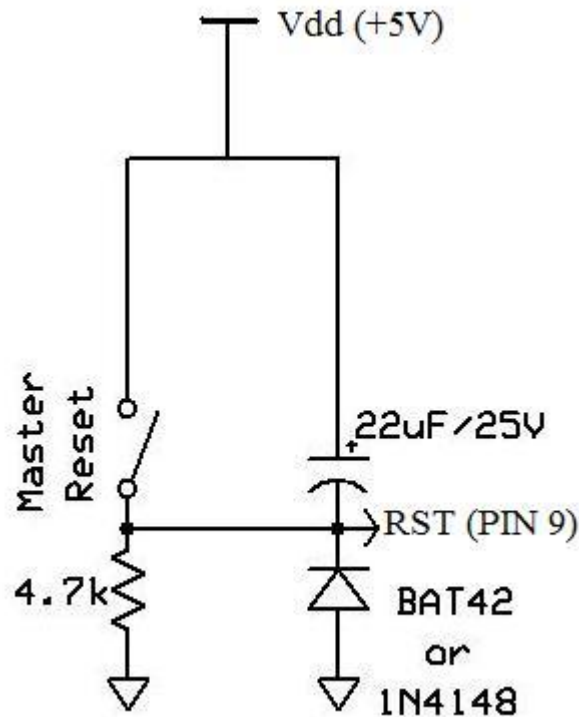


### **Crystal**

I have chosen a 27MHz crystal for the oscillator. If you wish to use a crystal of different frequency, then you will have to modify the functions ‘\_Delay10ms’ and ‘\_Delay100ms’ in the file “SystemFunction.inc”. It may be noted that these functions are heavily used by the application for timing.

It is not advisable to use a crystal of frequency lower than 20MHz lest the timing and real time response of the game should be affected.

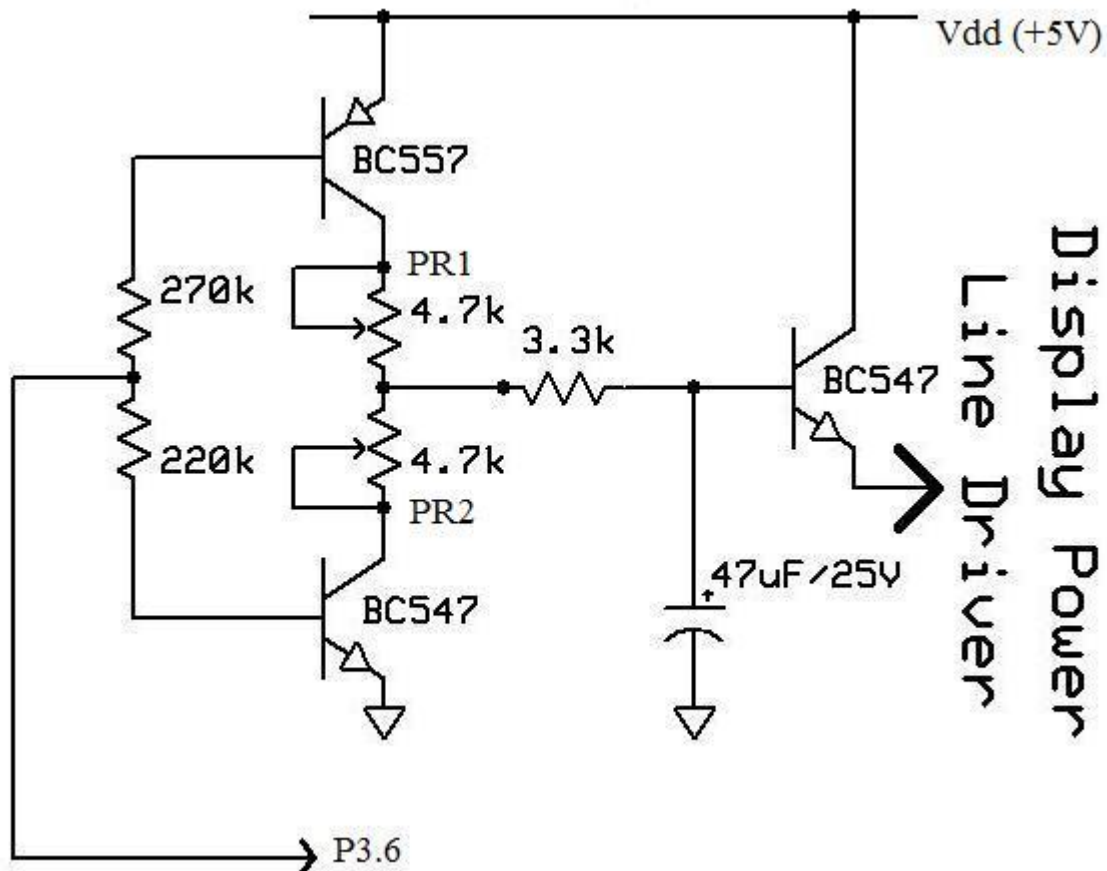
## Reset Circuitry



The  $4.7k - 22\mu F$  RC network provides a power on auto. reset feature. You may wonder why an RC network of time constant as large as  $\sim 100ms$  is used. The reset key also has an overloaded feature: wake the MCU from Power Down state (external interrupt keys may also be used for this purpose). The application is then expected to resume from the point where the Power Down occurred. However, in the first design (where I did not incorporate the RC network), the bouncing of the reset key prevented the application from resuming; Often, it would start from the beginning. But now, the RC network also offers a debounce feature. A large RC delay will guarantee that the RESET pin will be held high until even the bounce signal of the worst time period has settled. The diode provides an easy discharge path for the capacitor during a cold reset.

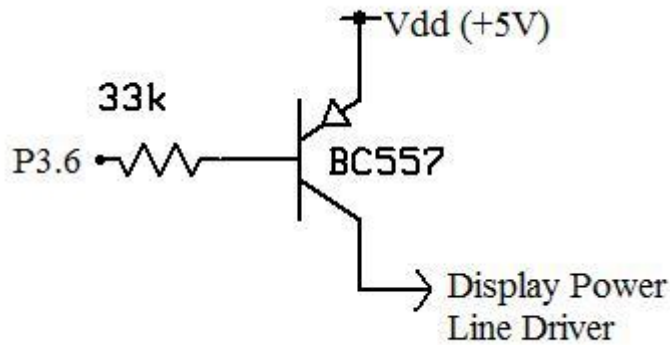
### Display ON / OFF delay control circuit

The port pin P3.6 can be used to turn on / turn off the display.



I have incorporated a delay circuit between the display ON / OFF control pin and Display Power Line Driver transistor because I felt a gradual transition of the state of the display is much more appealing to see than a sudden ON / OFF action. The ON time can be adjusted using the preset, PR1 and the OFF time using PR2. **However, if you feel that a delay circuitry is quite unnecessary, then you may replace the above circuit with the one shown below.**





### **16 \* 8 3mm LED Dot Matrix Display**

The display is driven by the ports P0, P1 and P2. The anodes of the LEDs are shorted along any column and their cathodes are shorted along any row. The LED rows 0 (bottommost) to 7 (topmost) are connected to P0 pins 0 to 7 respectively. Similarly, the LED columns 0 (leftmost) to 7 and 8 to 15 (rightmost) are driven by P1 pins 0 to 7 and P2 pins 0 to 7 respectively. All port pins are active low.

**NOTE:** The rows of LEDs in the display are directly driven by P0. A 680E current limiting resistor(CLR) is connected between a row and the corresponding pin of P0. Please don't attempt to increase the display brightness by reducing the value of the CLR. I have biased the LEDs in such a manner that the total current sunk by P0 is very close to it's max. rating (Refer electrical characteristics of AT89S51 in the datasheet).

### **Keypad**

Five way keys and a special function delay settings key is used for user input. The keypad is accessible by the pins P3.0 to P3.5.

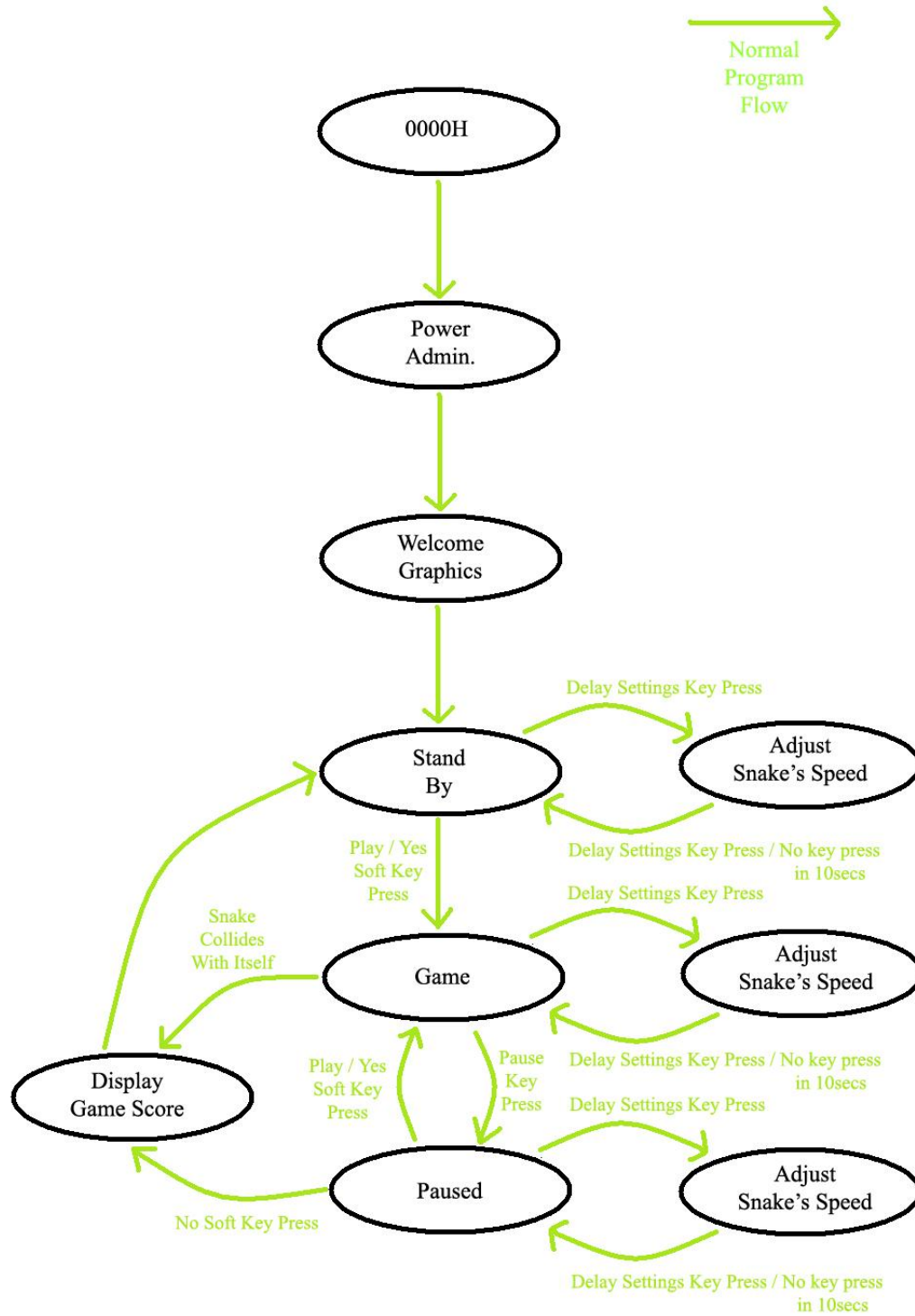
**NOTE:** The functions of the keys conveyed by their names indicated in the circuit diagram is with reference to the "Game context" of the application only. The keys can assume different functionality in various other contexts of the application ( eg. Stand By mode, Paused mode, etc). The function mapped to the keys, in various modes will be explained while covering the respective modes in the software description section in this document.



There is also a 5V power supply section and a buzzer driver circuitry. **I feel these sections of the circuit are self explanatory.** The buzzer is driven by the pin P3.7.

## Description of the Software

The logical flow of the program is given below:



Each state in the figure is a particular mode (state) of the program and will be covered shortly. First, let me explain the coding conventions I followed in this project.

## **Coding Conventions**

Group of functions that accomplish a particular task will always be moved to a separate file (.inc). Be it a file name, function name, ISR name or even labels, the identifiers clearly manifest their purpose. Each function has been documented, clearly specifying their arguments and return value(s). I have also added comments where ever necessary.

### **Function naming convention:**

The function names will always start specifying the return variable, followed by the identifier and the arguments, if any will also be contained in the name. If a function does not return any value, then the name will begin with an underscore.

Eg: `_Delay100ms`: This function will offer a delay of 100ms; No value is returned.

`_DelayABy100ms`: This function offers a delay 'A' times 100ms. So, the delay in 100ms must be passed as the argument in A.

`CGetPixel_X_Y`: This function returns the state of the pixel (X, Y) in C. X and Y are global variables.

### **Function coding convention:**

All the functions defined in page one of ROM will always push the registers they use and the variables they modify on the stack, unless, a variable or a register is explicitly specified as the function return value.

### **Register usage convention:**

Only register bank 0 has been used by the application. The registers are always used to store local variables only. Remaining three register bank locations are used by global variables.

## **Global Variables and Program Constants**

All the global variables and the program constants are defined in the source file 'ConstAndDataDefs.inc'. All the 128 bytes of available RAM has been put into use (may be I will have to move on to AT89S52, in case the next version of the game is made). The RAM partitioning and the purpose of each partition is given below.

8 bytes	00H	Working Registers
24 bytes	08H	Global Variables
16 bytes	20H	Display Buffer
32 bytes	30H	Circular Queue
48 bytes	50H	Stack

## **Display multiplexing**

The display is controlled by the ports P0, P1 and P2. Now, we have 128 LEDs and 24 MCU pins altogether to control them. So, we multiplex the display to the port pins ie flash only one of the 16 columns at a time. After flashing this column for a short period, we switch to the next column. Finally, after proceeding in this fashion, when we reach the last column, we circle around to the first column. It takes about 8ms to sweep the whole display once. So, the average display refreshing rate is about 125Hz at 27MHz crystal frequency.

	(0, 7)															(15, 7)
P0.7	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
P0.6	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
P0.5	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
P0.4	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
P0.3	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
P0.2	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
P0.1	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
P0.0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	(0, 0)															(15, 0)

P1.0 P1.1 P1.2 P1.3 P1.4 P1.5 P1.6 P1.7 P2.0 P2.1 P2.2 P2.3 P2.4 P2.5 P2.6 P2.7

**NOTE:** The pixel numbering along the Y axis contradicts with the standard adopted in computer graphics. I felt the Cartesian co-ordinate system standard is makes graphic functions much simpler to define and handle.

The application will simply dump the contents to be displayed in the display buffer (20H to 2FH). Port pins P1.0 to P2.7 are mapped to the bytes at 20H to 2FH of the RAM. Though the Intel architecture boasts that these locations are bit addressable, this facility is of little use in our application. This is because, the bit addressable instructions support only direct addressing while we are in need of indirect addressing.

The periodic multiplexing action is performed by a display multiplexing ISR (Timer 0 interrupt ISR). In fact, this is the only other task that runs in the “background”. All the functions that implement display multiplexing are defined in the source file ‘DisplayMuxing.inc’.

## **System Control Functions**

By system control, I mean functions to determine the key states, functions to give delays and functions to control the display and the buzzer. These functions are defined in the source file ‘SystemFunctions.inc’.

## **Graphics**

I am not sure whether I can use the word “Graphics” here. The only “GUI” available is a 16 \* 8 LED dot matrix display. Any way, I have segregated the graphics functions into two: those that give pixel level access to the display and those that offer more complex graphic features. The former functions are defined in the source file ‘LowLevelGfx.inc’ (low level graphics) and the

latter can be found in 'HighLevelGfx.inc' (high level graphics). Low level graphic functions are used to turn a pixel on or off and to get the state of a pixel. High level graphic functions are used to clear the screen, draw lines and print digits on the screen.

### **Various modes and their associated source files**

Once the System Power Administration block kicks off the application, the 'C51Main' block takes over the command. The flow of the application and the various modes that it enters, based on user requests, are controlled by this block which is defined in the source file 'C51Snakes.asm'. This file is also the main file to be added to the project. Like the functions of the 'Game Mode', definitions in this file are placed in page two of MCU ROM (0800H to 0FFFH). The rest of the source file definitions are placed in page one.

### **System Power Administration**

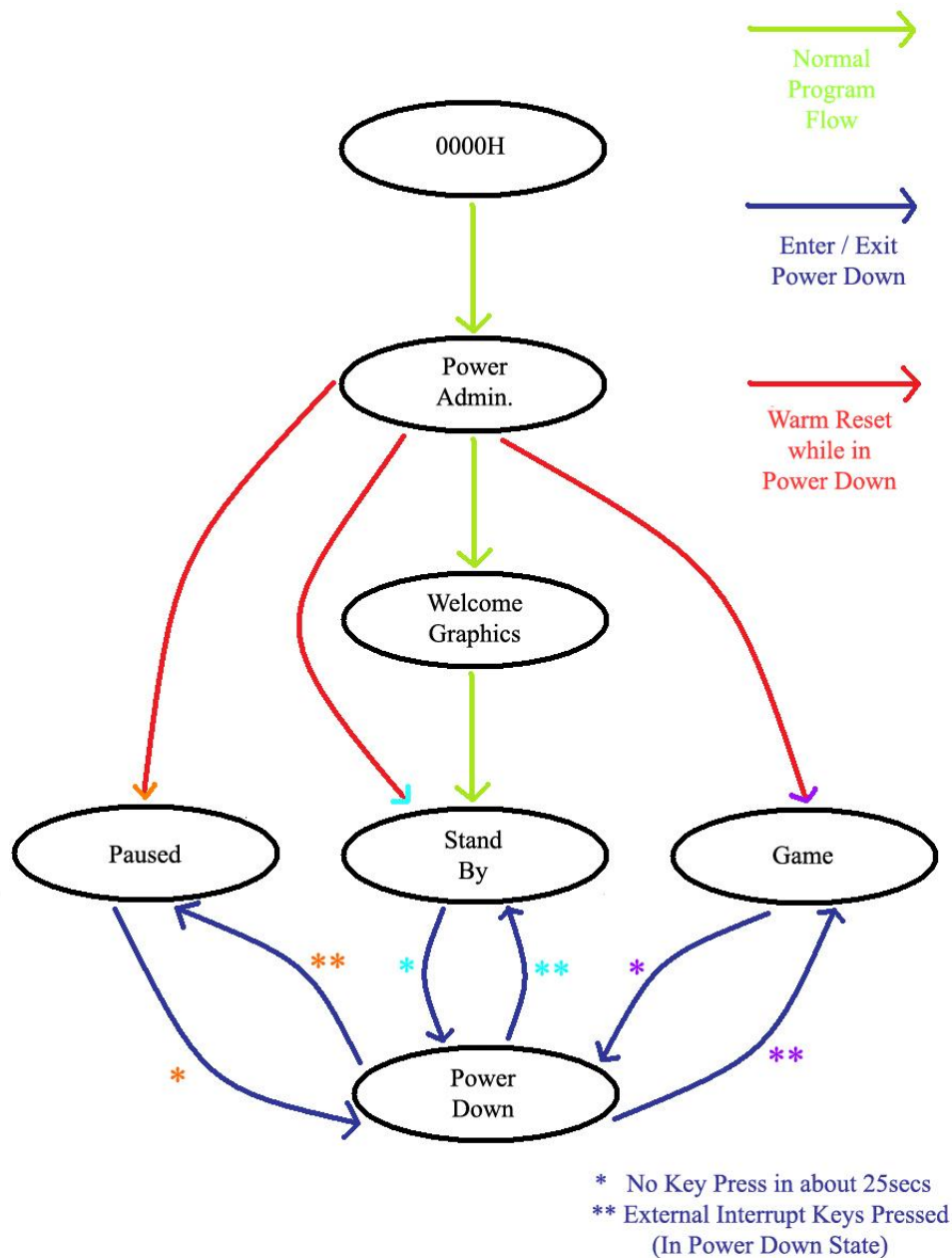
The power administration block is invoked during the application start up. This block also controls System Power Down and manages a system recovery from Power Down. The gaming console powers down if the user does not press any key in about 25secs. The application will resume (console wakes up from power down state) if the user presses external interrupt keys. This process is quite straight forward. However, what happens if the user presses the reset key, while the console is in power down state? Now, the situation gets complicated.

By the way, there are two power up modes to talk about. First, a cold power up / reset characterized by the Vdd rising from zero to maximum. This is the case when the power is switched off and again switched on. Second, a warm start up / reset, which happens when the user presses the reset key (while Vdd is held high). The RAM assumes random values during the former case while the RAM remains intact during the latter case. However, in either case, the SFRs are reset.

After a cold reset, the power administration block always initiates a normal program flow. However, after a warm reset, the block first determines whether the reset happened while the console was in power down mode. If this is the case, the power administration block will have to redirect the program flow to the point where the power down took place. Otherwise, as in the case of a cold

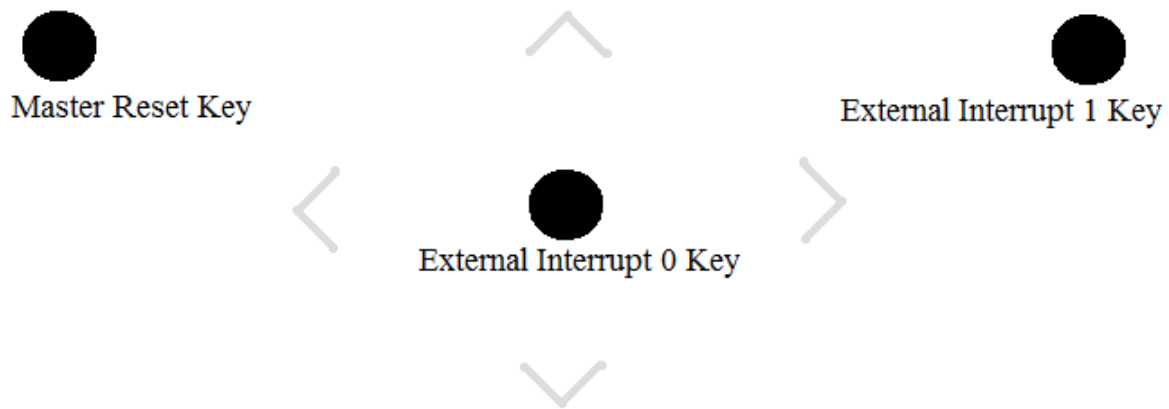
reset, a normal program flow will be initiated. In the former case, the power administration block will take care of restoring the SFRs!

The flow chart given below depicts the strategy adopted by Power Administration Block to decide the program flow during application start up:





The keys associated with the Power Administration Block are given below:



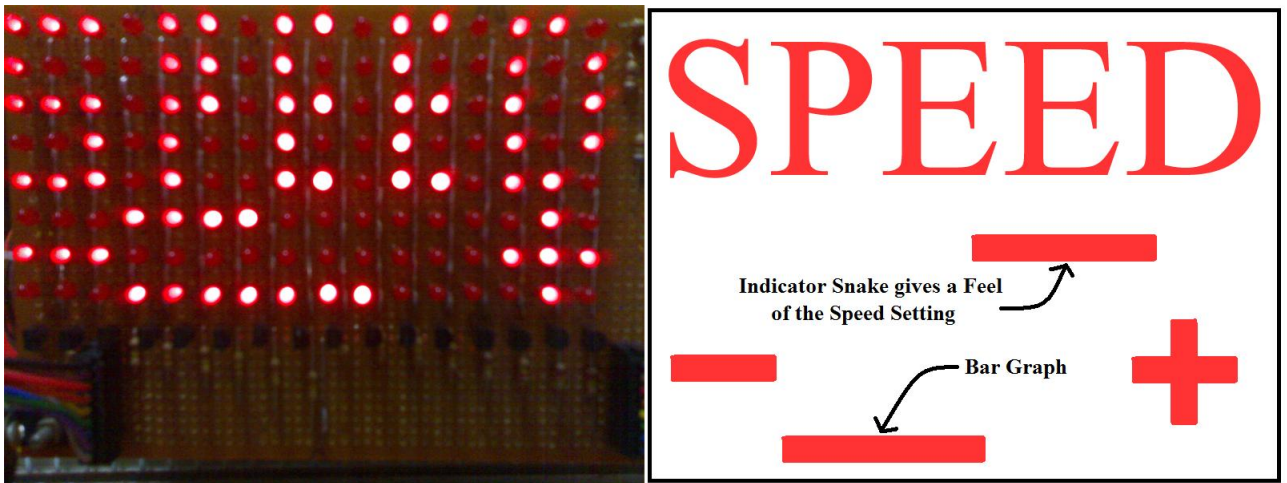
The functions relevant to the Power Administration block are defined in the source file 'SystemPower.inc'.

### Welcome Graphics

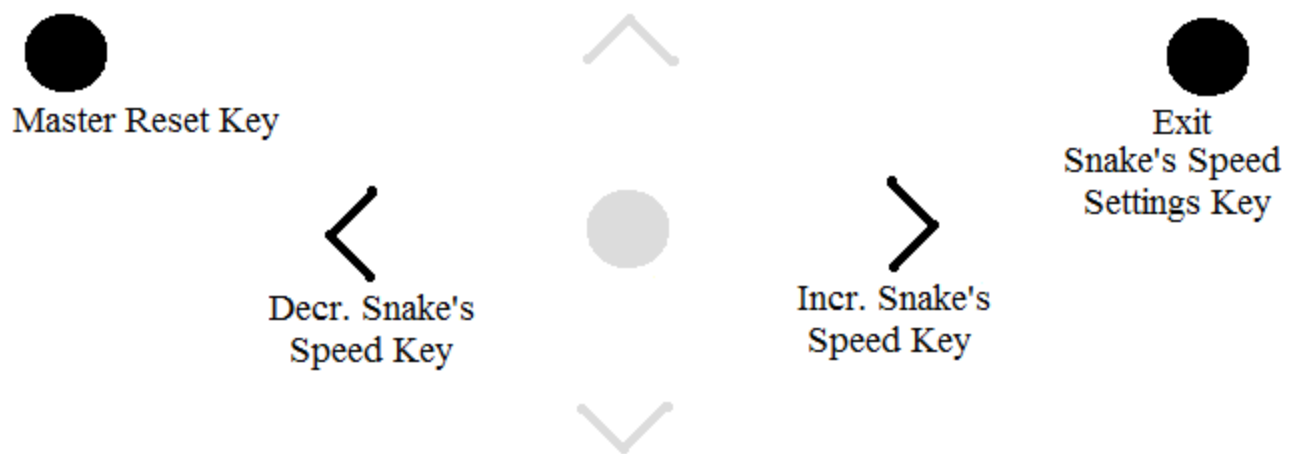
During the start up, a small animation is played on the screen which first displays the name of the platform (8051) for some time and then scrolls the name of the game, "Snakes", on the display once. All the functions relevant to welcome graphics are defined in the source file 'WelcomeGfx.inc'.

### Delay Settings Mode (Adjust Snake's Speed)

The Delay Settings Mode can be invoked from the Stand By state, Paused state or while the game is on. This mode helps the user to adjust the snake's speed. The speed level can be estimated from a bar graph as well as an indicator snake that continuously traverses the screen, which gives a better feel of the speed setting.



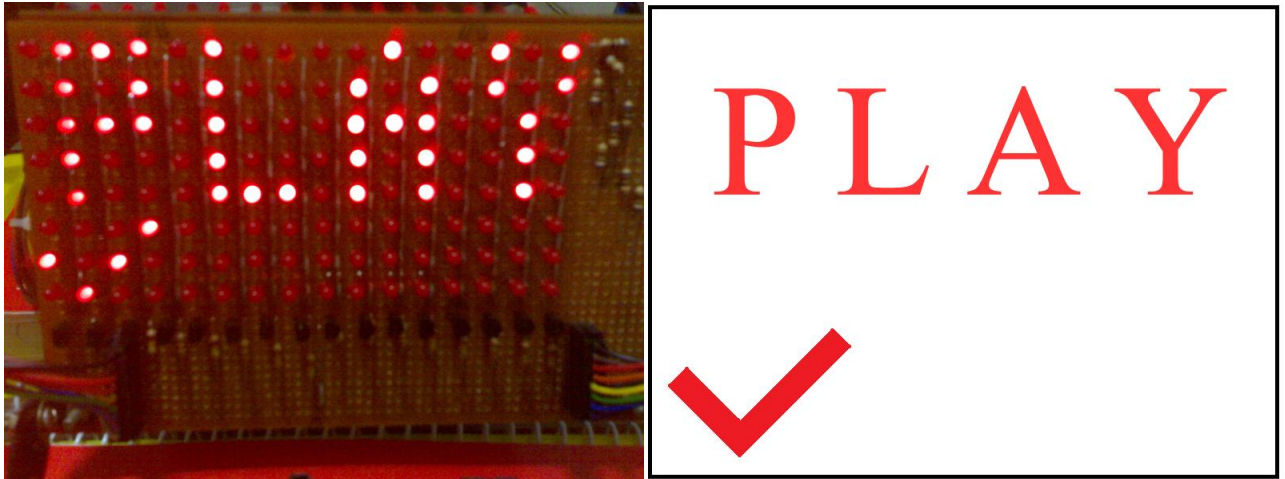
The keys associated with the Delay Settings mode are shown below:



The functions that compose the Delay Settings mode are defined in the source file 'DelaySettings.inc'.

### Stand By Mode

The application enters the stand by mode, as soon as the "Welcome Graphics" animation is displayed. The application also returns to this mode soon after the end of a game and then, begins waiting for a user signal to start the next game.



The keys associated with the Stand By mode are given below:



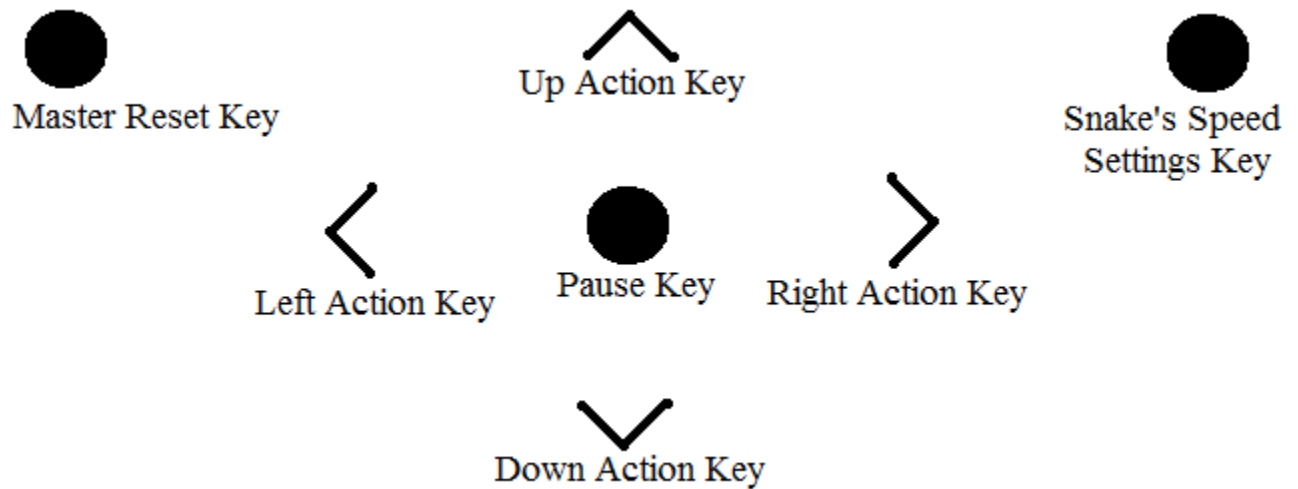
The functions related to Stand By mode are defined in the source file 'StandByState.inc'.

### The Game Mode

The entire application is centered around the game mode. By pressing the 'Play' / 'Yes' keys in the Stand By mode, the game begins. The snake runs continuously on the screen and can be controlled by the direction keys. Eggs appear periodically on the screen at random pixels and the user is expected to guide the snake to the eggs before they disappear after a certain period. It is enough to bring the snake one pixel away from the egg; the snake snatches them automatically and grows in size by one pixel. The objective of the game

(for the sake of the folks living under a rock...) is to snatch as many eggs as possible and stay alive by avoiding the collision of the head of the snake with it's own body. The game ends when a collision is detected and the number of eggs snatched is then displayed as the game score.

The keys associated with Game mode are given below:

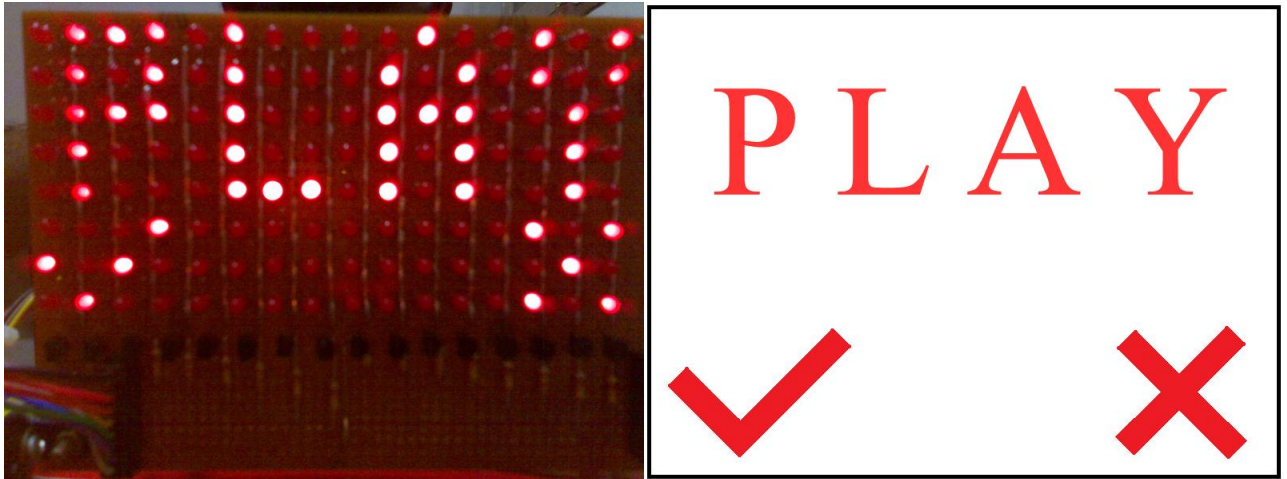


The direction keys marked Right, Up, Left and Down are the game action keys and are used to control the movement of the snake. If a direction key mapped to the current direction of movement of the snake is pressed, the snake accelerates. Similarly, if the key mapped to the opposite direction is pressed, the snake slows down. Remaining two direction keys alter the direction of movement of the snake when pressed.

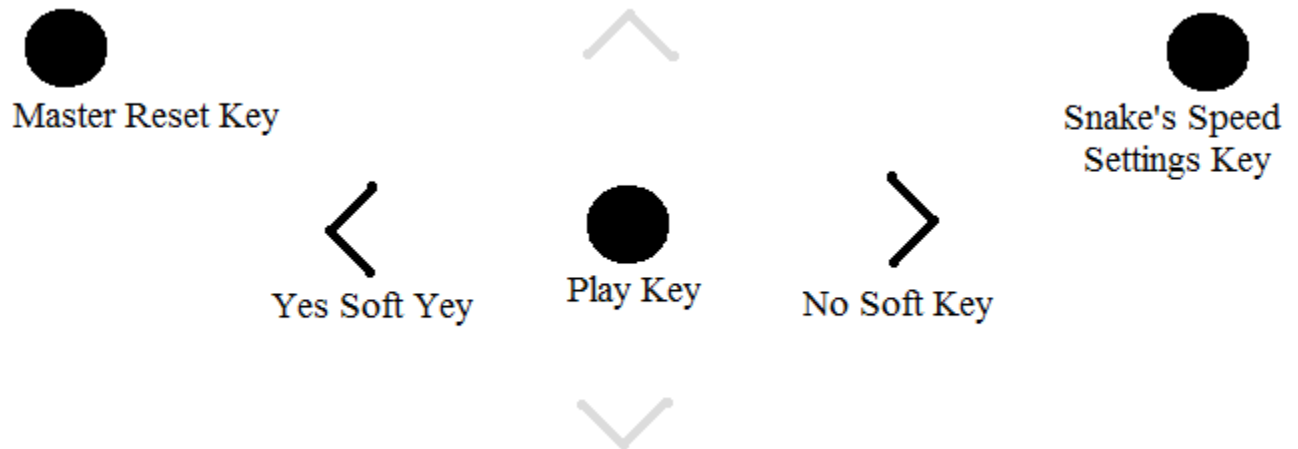
The source files 'GameFrontEnd.inc', 'GameBackEnd.inc' and 'EggHandle.inc' define the Game Mode. All the three files are placed in page two of ROM

### Paused Mode

The game enters paused mode when the user presses 'Pause' key in game mode. In this mode, the user can either choose to continue the game or quit. In the latter case, the game score is displayed and the application enters Stand By mode.



The keys associated with the Paused mode are given below:



The functions that constitute the Paused Mode are defined in the source file 'PauseState.inc'.

### **Details of the Assembler**

I used Keil uVision 3 to assemble the code. It may be noted that a full version of the assembler is required to generate the 'hex' file because the program also uses the code space beyond 2kB of the ROM. The evaluation version will assemble only the programs that fit in the first 2kB space limit.

**That's all folks...! Happy gaming...!**