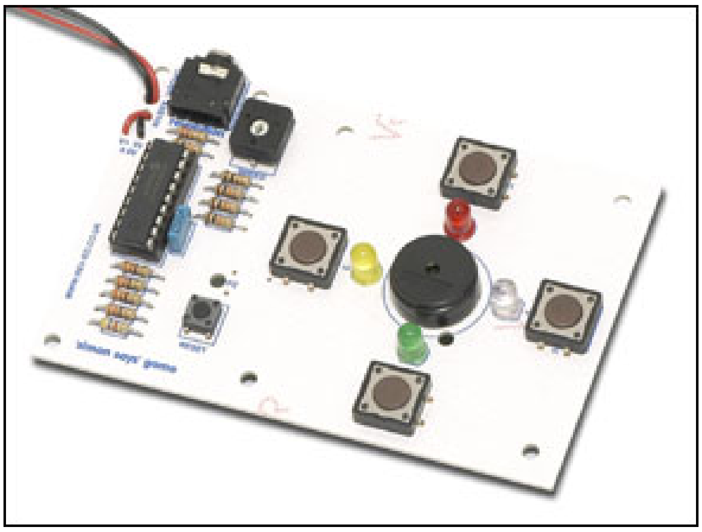
PICAXE 'Simon Says' Game

Order Codes:	
AXE106	Simon Says Game Self-Assembly Kit



Features
- 4 play switches with different colour LED indicators
- piezo sound device
- speed control preset resistor
- reprogrammable PICAXE18M microcontroller
- simple construction

Also required:
- 3x AA batteries
- soldering iron and solder
- side cutters and small cross-head screwdriver

Contents:

R1	1	4k7 carbon film 0.25W	*yellow violdet redgold*
R2-5	4	330R carbon film 0.25W	*orange orange brown gold*
R6	1	22k carbon film 0.25W	*red red orange gold*
R7-11	5	10k carbon film 0.25W	*brown black red gold*
VR1	1	100k preset resistor	
C1	1	100nF polyester capacitor	*marked 104 - not polarised*
CT1	1	stereo PICAXE connector	*ensure 'snapped' onto pcb*
LED1	1	5mm red LED	*align flat with ink image on pcb*
LED2	1	5mm yellow LED	*align flat with ink image on pcb*
LED3	1	5mm green LED	*align flat with ink image on pcb*
LED4	1	5mm blue LED	*align flat with ink image on pcb*
PZ1	1	piezo transducer	
SW5	1	miniature reset switch	*only fits one way around!*
SW1-4	4	push switch	*only fits one way around!*
IC1	1	18 pin IC socket	*use for PICAXE18M*
IC1	1	PICAXE18M microcontroller	*pin 1 faces up*
BT1	1	3xAA battery box + clip	*red wire - V+*
	1	pcb	

Assembly Instructions

1. Solder all the resistors in position. The values of the resistors are shown on the pcb, and the colour codes are given in the table on page 1.
2. Solder the PICAXE download socket CT1 is position. Make sure it clicks flat onto the PCB before soldering.
3. Solder the IC socket in position.
4. Solder the preset resistor VR1 in position.
5. Solder the rectangular polyester capacitor C1 in position. It can be used either way around.
6. Solder the reset switch in position - it will only fit one way around. Solder the four push switches in position.
7. Solder the four LEDs in position. The LED can be soldered directly to the pcb or connected via wires (not supplied). Make sure the flat on the LED aligns with the footprint on the pcb.
8. Solder the piezo sounder PZ in central PIEZO position.
9. Thread the battery clip through the PCB The red wire is connected to the V+ contact, the black wire to the 0V contact.
10. Push the PICAXE18M chip into it's socket. Make sure pin 1 faces the four resistors.
11. Insert 3AA batteries (not supplied) into the battery pack and then connect to the battery clip.
12. Program the microcontroller using the sample program given.

DO NOT USE A 9V PP3 BATTERY WITH THIS PRODUCT.
ONLY USE THE 4.5V (3xAA CELL) BATTERY BOX SUPPLIED.

**The PICAXE-18M chip must be programmed before use.
The sample program can be found in the \samples folder of the Programming Editor software (file AXE106 Simon Says.bas).**

Safety

This product is designed as an educational teaching aid. It is not a toy and should not be handled by young children due to sharp edges and small parts.
THIS PRODUCT IS NOT DESIGNED AS A TOY FOR SMALL CHILDREN.

SIMON SAYS...

Remember the 70's?

I recently enjoyed the series of programs made by the BBC called 'I love 197x'. You certainly start to realise your age when you discover that 1978 was 25 years ago! The 1978 program made reference to the cult toy of the year, 'Simon' made by MB Games, which was loved by children and loathed by parents! This was one of the very first mass produced electronic games and I remember playing it for hours with friends and relatives.

Simon

For those too young to remember 1978, the idea behind the Simon game was quite simple. It was based on the old game 'Simon Says'. The game was made up of a big round plastic case with four coloured panels – under each panel was a switch and a light bulb. You would start the game and the electronics would light up one of the four panels and sound a tone. The game was then to press the panel that lit up. Simple enough! Then Simon would repeat, lighting that panel and adding another. Now your job was to press the two panels in the correct order. The number of panels would continue to get longer until you could no longer remember the sequence, which would cause Simon to issue a harsh buzz and end the game.

As I watched the TV program it struck me that this vintage toy from 1978 could probably be reproduced with a cheap PIC microcontroller now at very low cost.

So I set myself the task of building my own 'Simon Says' game for under £5 (excluding PCB cost). At the same time I thought it would provide a perfect example of how to demonstrate how to remember sequences whilst programming, something that students regularly find difficult.

Internet Trivia

A quick search on the Web soon revealed lots of useless trivia about the game.

The first single player game was released in 1978, and then in 1979 MB released 'Super Simon' which had two sets of panels so that two people could play against each other. In 1980 Pocket Simon, a smaller version of the original game, was released. There was also a special edition Simon released with a clear casing so the electronics could be seen inside. Apparently the 'Super Simon' can also be seen in the film 'ET' on the shelf behind ET's head when he first speaks!

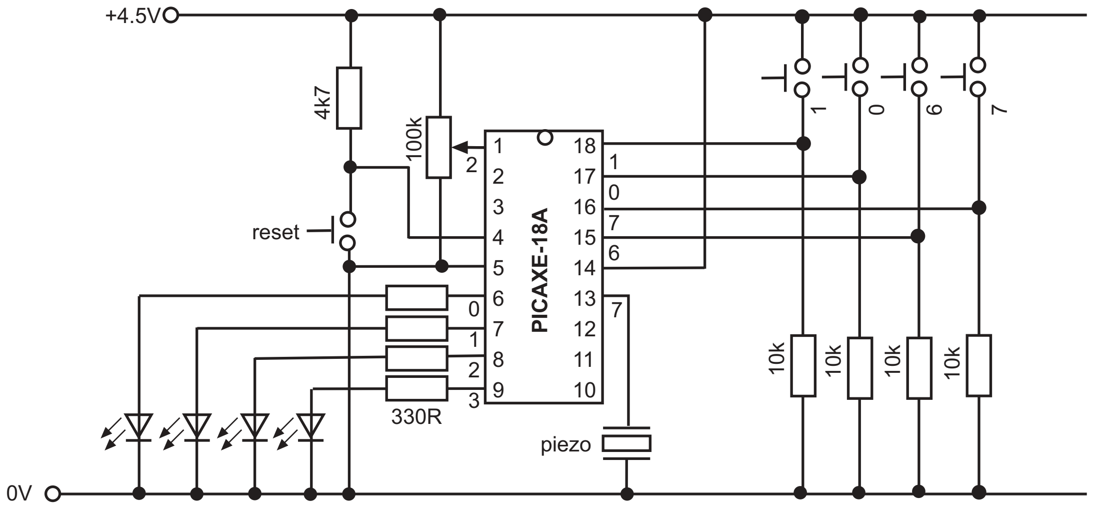
However I was more interested in how the original game worked. I discovered it needed both a 9V PP3 and 2 large D cells to make it work, presumably to power the light bulbs and speaker, but could not discover much more online.

But fortunately I then discovered that I could buy a real Super Simon, complete with box and instructions, for just £15. So five days later I was a proud owner of a vintage game, which I then, as you probably expect, completely disassembled! (Many thanks to www.retrogames.co.uk).

The Original Game

After taking off the cover, the PCB inside the Super Simon was extremely bare – 8 bulbs buffered by a couple of standard logic gates and a Texas Instrument 'microcomputer' chip. These microcomputer chips were the first 'single chip' controllers widely used in consumer products, and can be found in a wide range of early 80's equipment such as vending machines. These single chip 'microcomputers' were the predecessors of the modern PIC microcontrollers, and used in a very similar way. Many people think microcontrollers are new, when in actual fact this game was using almost identical single chip technology 25 years ago!

The New Game



The circuit for the microcontroller version of the game is shown in figure 4 (the standard PICAXE serial download circuit on pins 2,3 is not shown for clarity). The circuit is very straight forward, 4 LED outputs (I chose red yellow green and blue), a piezo-sounder and 4 push switches. As the PICAXE-18M system was used for programming I also added the PICAXE download socket to the prototype PCB (shown in Figure 5), so that the microcontroller could be re-programmed onboard via direct cable link without the need for a programmer.

The cost of the circuit was calculated at under £5 (excluding PCB)! It will also run quite happily off three small AA cells, certainly no need for the large D cells and the PP3!

Programming Introduction

The programming task for the Simon game is fairly complicated, and certainly more advanced that most GCSE projects. However it is a perfect example of how to demonstrate how to 'remember' sequences, something students generally find very complex to do.

When approaching a complicated problem like this it is essential to break the overall task down into small, manageable chunks, and then put the whole program together at the end. I identified the following tasks:

1) Wait for the player to press a switch to start the game.
2) Generate a sequence of random numbers (0 to 3 for the four LEDs). In this case I will use 100 steps (many more than 7 or 8 I can normally repeat in a game!). These numbers are stored in the microcontrollers data memory, which actually has space for up to 256 steps.
3) Get the microcontroller to play back the numbers. To do this the micro-controller must know how many steps to playback in each turn of the game. To do this I will use a variable called 'topstep' to remember how many steps to playback. If topstep = 1, one step will be played back, if topstep = 2, two steps will be played back etc.
4) When the player presses the switch, the microcontroller must light the correct LED for that switch, and then compare the switch press to see if it is the correct switch. To do this the microcontroller must also count how many switches the player has pressed, and to do this I will use another variable called 'playerstep'.
5) When the player reaches the end of the sequence, the microcontroller must acknowledge the success, add one to the value of topstep, and then repeat the process from 3) above. If the player gets the sequence wrong, a buzzer will sound and the game reset.

Program

The full program is given overleaf. The program is complex, but is provided mainly as an example of what can be achieved with microcontrollers. Full comments are given in the program, but a brief explanation is also included here.

Section 1 in the program is a loop that lights all four LEDs, generates a random number, and then waits for a switch to be pushed to start the game. By including the 'random' command within the loop, it is constantly varying and so no two games will be the same.

Section 2 use a for...next loop to store 30 random numbers in the microcontrollers memory. As the random command generates a number between 0 and 255, and we only require the numbers 0 to 3 (for the four LEDs), a simple comparison test is made to get the four desired values.

Section 3 switches all four LEDs off, and then uses a for..next loop to play back the sequence (up to the variable called topstep). The 'beep' sub-procedure in section 5 is used to light the appropriate LED and make a sound for each step (the sound is different for each LED to aid memory during the game).

Section 4 first resets the players position to 1. A test is then carried out to see if the player has done all the steps needed. If all steps have been done the 'success' section of code flashes all four LEDs, adds one more step to the topstep value, and then loops back to section 3.

If there are still steps to do, the correct target value is retrieved from memory for comparison. The program then enters a loop waiting for a switch to be pressed.

When the switch is pressed the switch is compared to the target value retrieved from memory. If the values are the same everything is correct and so the LED is lit via the ‘beep’ sub-procedure, the players position is increased by one and the program loops back for another switch push.

If the value is incorrect, the ‘fail’ section of code makes a noise and the resets the game.

Summary

Single chip controllers are not new, this game was using them 25 years ago. However electronics has changed dramatically over the last 25 years, and modern microcontrollers are much cheaper and easier to use than the original ’microcomputers’. Modern microcontrollers reduce large complex circuits down to simple clean designs, and also dramatically reduce the cost of these products. LED technology has improved, and no game would ever be manufactured now with bulbs due to cost, safety and power consumption.
.
The example program given is fairly complex, but shows the enormous capabilities of the low-cost microcontrollers. If you wish to build this circuit a PCB is available from Revolution, part AXE106 (www.picaxe.co.uk).

Figure 6 – Simon Says program

```
‘ AXE106 Simon Says Game

‘ *** Define the variables used ***

‘ Push switches on inputs 0,1,6,7
‘ Speed preset on input 2
‘ LEDs on outputs 0-3
‘ Piezo on output 7

symbol rand = b1            ‘ random number store for loading memory
symbol value = b2           ‘ switch value 0-1-2-3
symbol playerstep = b3      ‘ position of player in game
symbol freq = b4            ‘ sound variable
symbol topstep = b5         ‘ number of steps in sequence
symbol counter = b6         ‘ general purpose counter
symbol speed = b7           ‘ speed

‘ *** Section 1 **********************
‘ *** This section waits for start ***
‘ **********************************

‘ wait for any switch to be pushed
‘ with all four LEDs lit
‘ preload rand with any number by repeatedly
‘ using the random command in the loop

init:
      let pins = %00001111
      random rand
      if pin0 = 1 then preload
      if pin1 = 1 then preload
      if pin6 = 1 then preload
      if pin7 = 1 then preload
      goto init
```

```
‘ *** Section 2 ****************************
‘ *** This section loads memory for game ***
‘ ******************************************

‘ load EEPROM data memory with 100 numbers
‘ first get the random number (0 to 255)
‘ and then change to either 1,2,3 or 4
‘ and then save into data memory

preload:
      let pins = %00000000       ‘ LEDs off

      for counter = 0 to 30      ‘ for..next loop

      let value = 0
      random rand                ‘ get random number 0-255
      if rand > 180 then set0
      if rand > 120 then set1
      if rand > 60 then set2

set3: let value = value + 1      ‘1+1+1 = 3
set2: let value = value + 1      ‘1+1 = 2
set1: let value = value + 1      ‘1
set0:                            ‘0

      write counter,value        ‘ save in data memory

      next counter               ‘ next loop

‘ *** Section 3 ****************************
‘ *** This section plays back a sequence ***
‘ ******************************************

‘ switch off the LEDs and then start
‘ a game with the end counter as 1
      let pins = %00000000       ‘ LEDs off
      let topstep = 1            ‘ reset step number to 1

‘ playback the game sequence
playback:
      readadc 2,speed            ‘ read speed value

      for counter = 1 to topstep ‘ for...next loop
          read counter,value     ‘ get value
          gosub beep             ‘ make the noise
          pause 300              ‘ short delay
      next counter               ‘ loop

‘ *** Section 4 ****************************************
‘ *** This section detects the players reply sequence ***
‘ ******************************************************

‘ now the user responds
‘ reset the players position to 1
      playerstep = 1

gameloop:
‘ if playerstep is greater than topstep then all done
      if playerstep > topstep then success

‘ get the correct key value is supposed to hit
‘ from the EEPROM memory
      read playerstep,value

‘ now wait for switch to be pressed
lp:   if pin7 = 1 then pushed0
      if pin0 = 1 then pushed1
      if pin1 = 1 then pushed2
      if pin6 = 1 then pushed3
      goto lp
```

```
‘ switch pressed so check it is the correct one
‘ if it is make a beep sound and then continue
‘ else fail the game
pushed0:
      if value <> 0 then fail
      let playerstep = playerstep + 1
      gosub beep
      goto gameloop

pushed1:
      if value <> 1 then fail
      let playerstep = playerstep + 1
      gosub beep
      goto gameloop

pushed2:
      if value <> 2 then fail
      let playerstep = playerstep + 1
      gosub beep
      goto gameloop

pushed3:
      if value <> 3 then fail
      let playerstep = playerstep + 1
      gosub beep
      goto gameloop

‘ *** Failed so make noise and jump back to start ***
‘ failed so make failed noise, switch off all LEDs
‘ and go back to start
fail:
      let pins = %0000000        ‘ all LEDs off
      sound 7,(80,100)           ‘ make a noise
      sound 7,(50,100)
      goto init                  ‘ back to start

‘ *** Succeeded so add another step to sequence and loop ***

‘ success so make a success sound
‘ and then increment topstep and do another sequence
success:
      pause 100                  ‘ short delay
      let pins = %00001111       ‘ all LEDs on
      sound 7,(120,50)           ‘ success beep
      let pins = %00000000       ‘ all LEDs off
      pause 100                  ‘ short delay
      let topstep = topstep + 1  ‘ add another step
      goto playback              ‘ loop again

‘ *** Section 5 ****************
‘ *** sub light LED and beep ***
‘ ******************************

‘sub-procedure to light correct LED
‘and make a different beep sound for each LED
‘value always contains number 0,1,2 or 3.
‘add 1 and multiply by 20 to give larger difference
‘in the sound noise

beep:
      high value            ‘ switch on LED
      freq = value + 1      ‘ generate sound freq.
      freq = freq * 25
      sound 7,(freq,speed)  ‘ play sound
      low value             ‘ switch off LED
      return                ‘ return
```