Table of Contents

Welcome To Studio5 User Guide

This document outlines how to use Kic Studio5 application.

Download Circuit

A KicChip is easily connected to your Personal Computer via a cable connected to either your serial or USB socket.

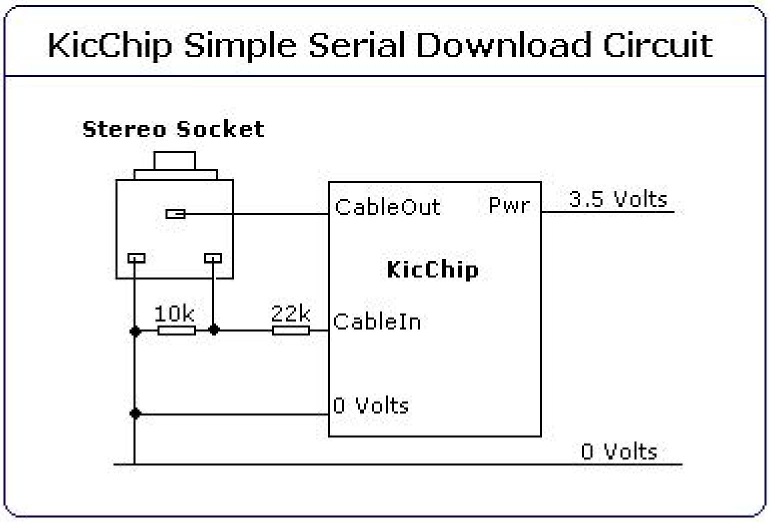
Parts:

10K Resistor

22K Resistor

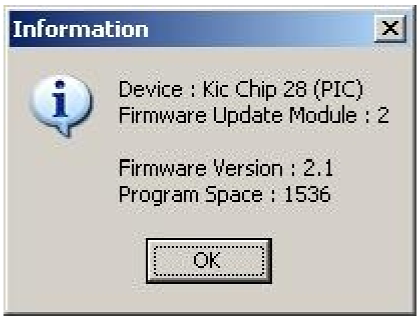
Microphone Socket

Download Cable



To confirm a connection, press "Kic Check" Button

A message window will tell you the version of the KicChip



Information

Device : Kic Chip 28 (PIC)
Firmware Update Module : 2

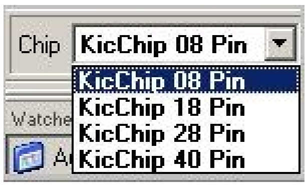
Firmware Version : 2.1
Program Space : 1536

OK

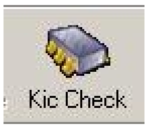
Selecting A KicChip

If you are planning on downloading any code or using the In Circuit Debug features, the KicChip you select in KicStudio should match your physical KicChip.

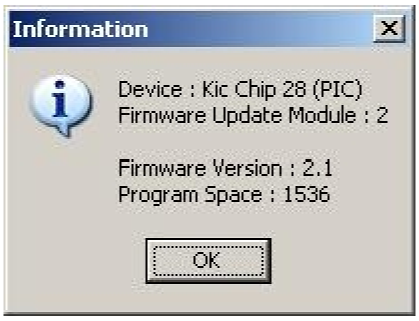
Select a KicChip from the drop down box at the top of the screen.



If you are not sure what model of KicChip you are connected to, use the KicCheck button to find out.



You are shown details of the pin size and the current firmware.



Information

Device : Kic Chip 28 (PIC)
Firmware Update Module : 2

Firmware Version : 2.1
Program Space : 1536

OK

System Configuration

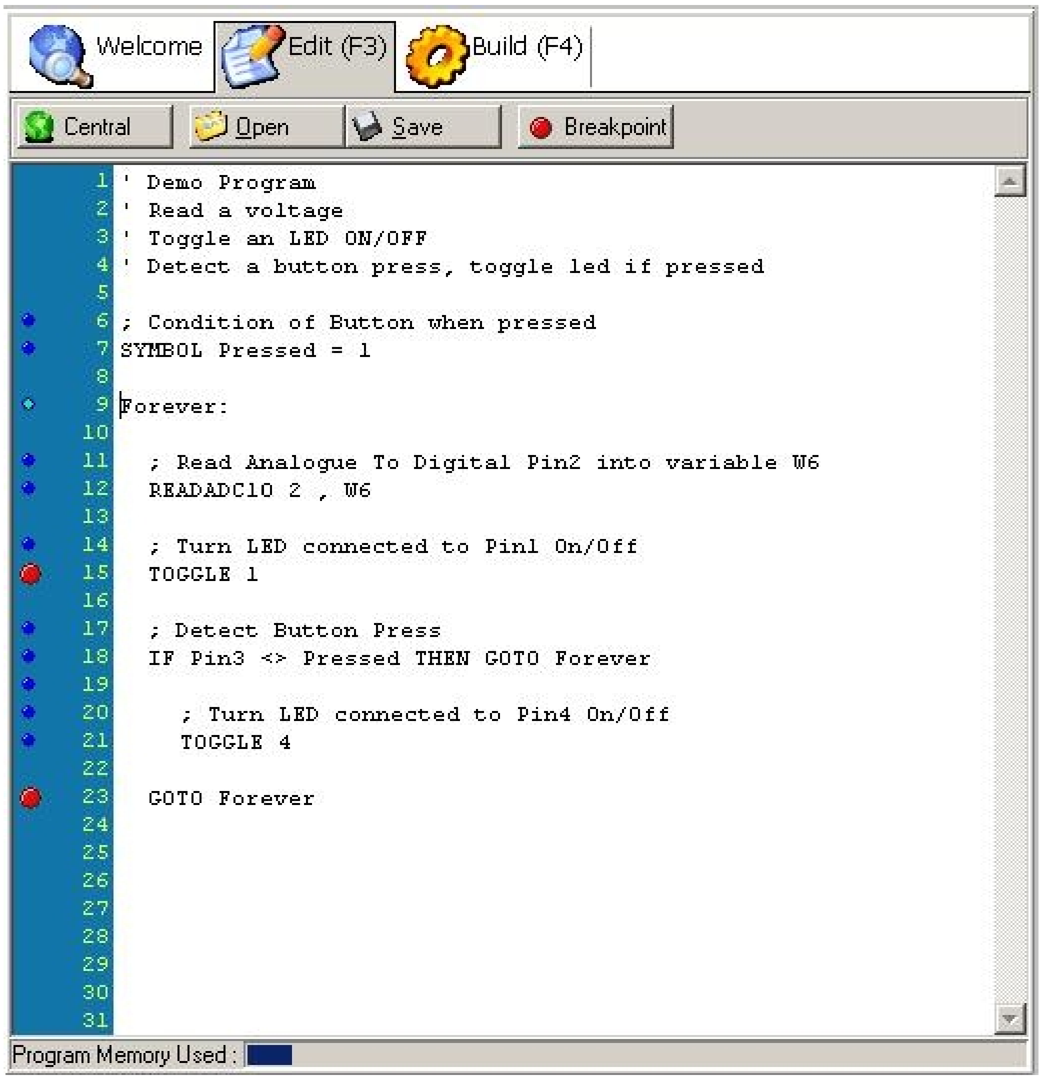
After selecting a KicChip, KicStudio does the following tasks:

1. Allocates RAM space according to the controllers specifications and updates the visual RAM window.
2. Clears all RAM locations to 0.
3. Allocates EEPROM space according to controllers specifications and updates the visual EEPROM window. Does NOT clear EEPROM locations.
4. Builds a pin module according to the controllers specifications.
5. Sets all pins to their reset state.
6. Does NOT clear previous watches, unavailable variables will have a value of 'Unavailable'.
7. Clears the symbol explorer.
8. Clears the label/breakpoint explorer.

Editing Your Program

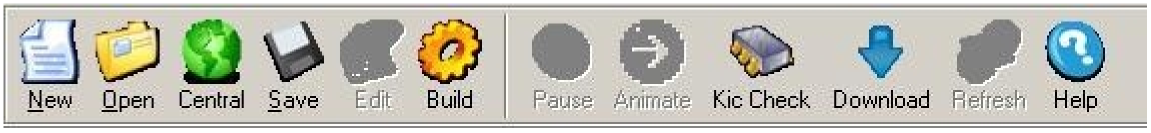
Edit Mode is the initial mode of KicStudio, or can be activated by pressing the Edit Mode Button or F3.

Edit mode must be selected in order to modify the program source code (and perform other file related tasks). When enabled, the editor has a white background and the text can be modified.



```
' Demo Program
' Read a voltage
' Toggle an LED ON/OFF
' Detect a button press, toggle led if pressed

; Condition of Button when pressed
SYMBOL Pressed = 1

Forever:

  ; Read Analogue To Digital Pin2 into variable W6
  READADC10 2 , W6

  ; Turn LED connected to Pin1 On/Off
  TOGGLE 1

  ; Detect Button Press
  IF Pin3 <> Pressed THEN GOTO Forever

    ; Turn LED connected to Pin4 On/Off
    TOGGLE 4

  GOTO Forever
```

Switching to Edit Mode causes various options to become enabled/disabled.

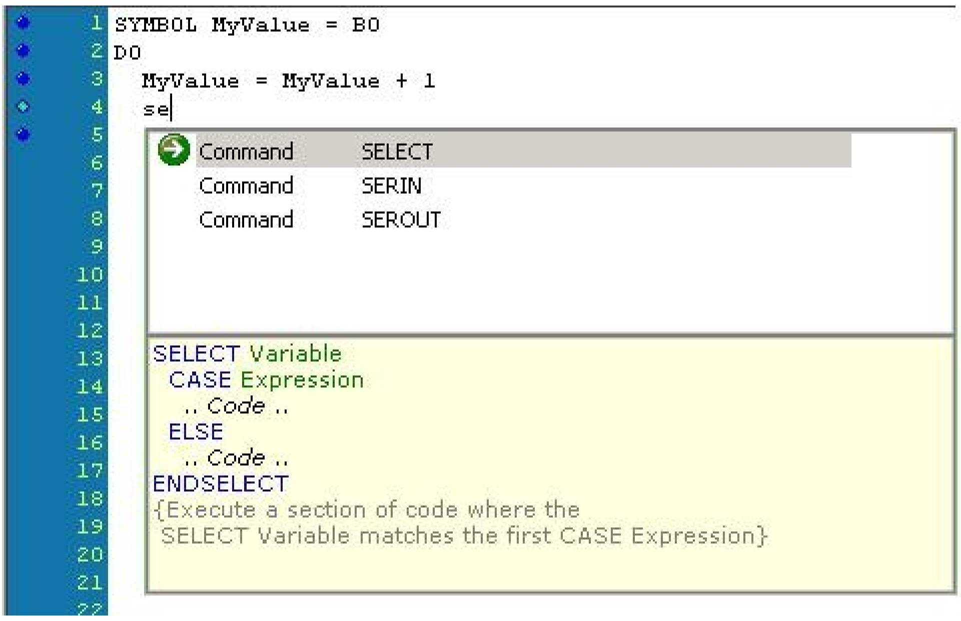


Edit mode enables the following options:

- New File
- Open File
- Select micro controller
- Clear EEPROM
- Select Build Mode
- Set animation speed

Note: The opposite of Edit Mode is Build Mode.

Auto Complete



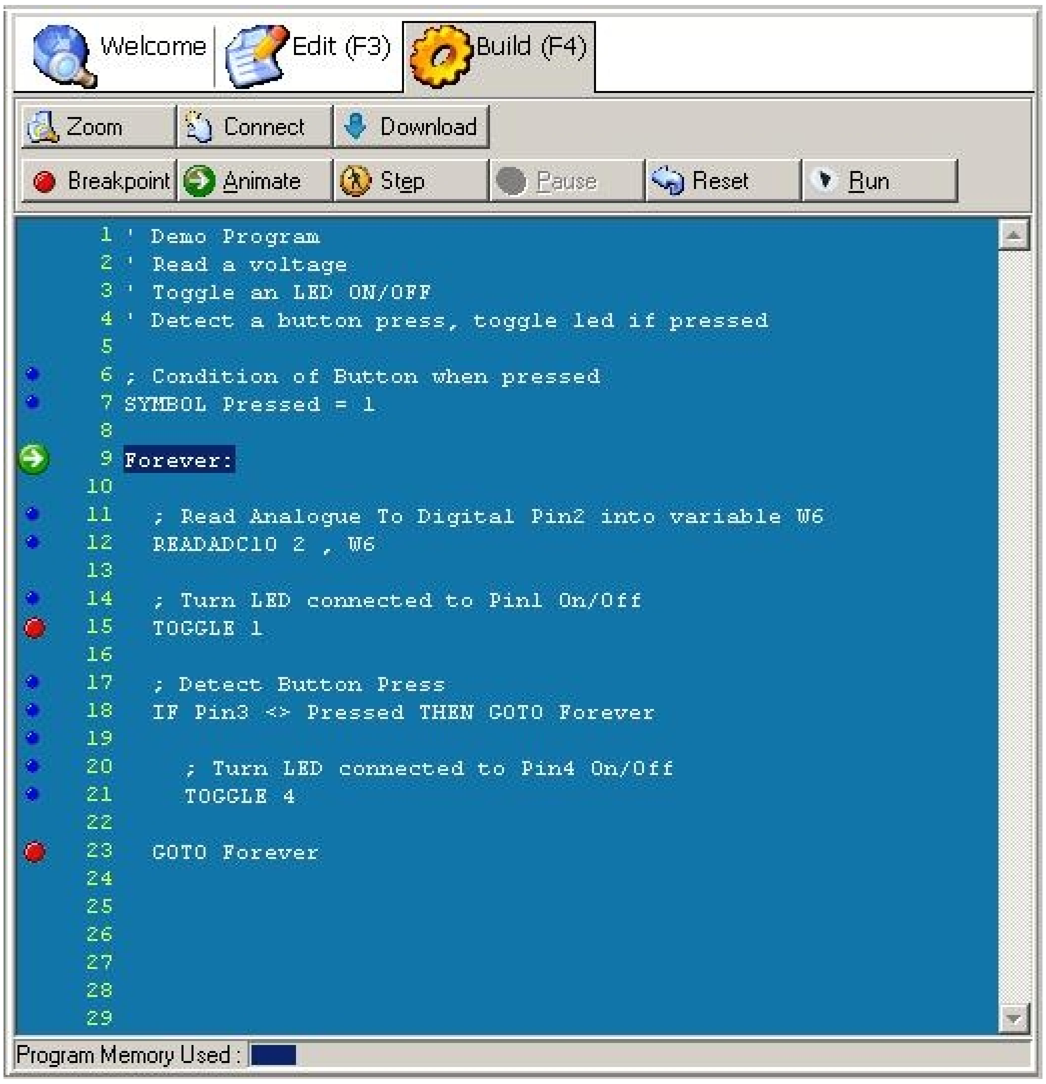
```
SYMBOL MyValue = B0
DO
  MyValue = MyValue + 1
  se
```

Command	SELECT
Command	SERIN
Command	SEROUT

```
SELECT Variable
  CASE Expression
    .. Code ..
  ELSE
    .. Code ..
ENDSELECT
{Execute a section of code where the
 SELECT Variable matches the first CASE Expression}
```

Compiling & Simulating

Select Build Mode by clicking the Build Mode button or by pressing F4.



```
' Demo Program
' Read a voltage
' Toggle an LED ON/OFF
' Detect a button press, toggle led if pressed

; Condition of Button when pressed
SYMBOL Pressed = 1

Forever:

  ; Read Analogue To Digital Pin2 into variable W6
  READADC10 2 , W6

  ; Turn LED connected to Pin1 On/Off
  TOGGLE 1

  ; Detect Button Press
  IF Pin3 <> Pressed THEN GOTO Forever

    ; Turn LED connected to Pin4 On/Off
    TOGGLE 4

  GOTO Forever
```

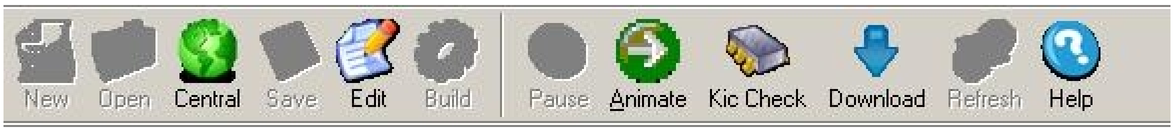
Build Mode must be selected in order to execute program source code.

Switching to Build Mode causes the following to happen:

- Simulator adds syntax errors (if any) to the message window.
- Editor becomes read only.
- EEPROM window shows cells referenced in EEPROM and DATA commands.
- Micro controller (simulator) is reset.
- If auto watch is enabled, variables in the source code are added to the Watch Window.
- Labels are added to label explorer.

- Symbols are added to symbol explorer.

Switching to Build Mode causes various options to become enabled/disabled.



Build Mode enables the following options:

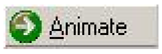
- Select Edit Mode
- Animate
- Pause
- Step
- Run
- Reset

However, these options can become enabled and disabled according to a specific task. The Pause button for example only becomes enabled after the program starts. Build Mode also disables the source code editor.

Note: The opposite of Build Mode is Edit Mode.

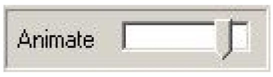
Animation

Animation can be selected in Build Mode by pressing the animate button, or F5.



Use animation to see the program source code being executed with a controlled delay between each line.

Animation speed is controlled by the animation control bar.



The main difference between running and animation is that animation highlights each line in the source code as it executes and the speed of animation can be controlled. Both running and animation currently update all screen views such as watches, pins and memory (RAM & EEPROM).

Stepping

Step can be selected in Build Mode by pressing the step button or by pressing/holding F6.

Step can be selected in Build Mode by pressing the step button or by pressing/holding F6.

Use stepping to see a single line of the source code being executed.

Stepping updates all screen views such as watches, pins and memory (Ram & Eeprom).

Use animate to step multiple lines, run to execute at a fast rate.

Running

Run can be selected in Build Mode by pressing the Run button, or F8.

Pressing Run causes the processor to execute code without updating the current line position in the editor. This reduces screen flicker and executes code much more quickly.

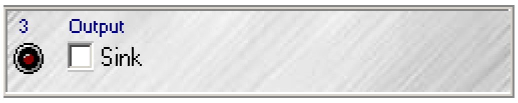
To see the source line highlighted as it is executed, use animation instead.

Both running and animation currently update all screen views such as watches, pins and memory (RAM & EEPROM).

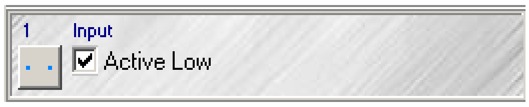
Virtual Devices

Virtual devices are created automatically after your program is compiled:

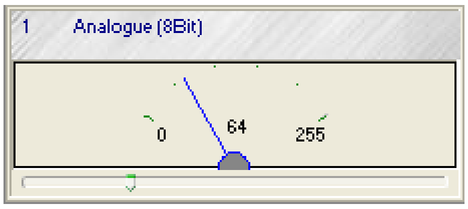
LED



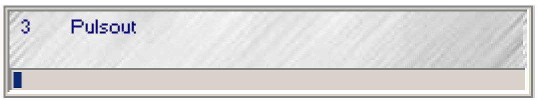
Button



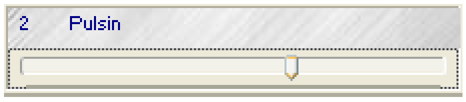
Analogue Slider



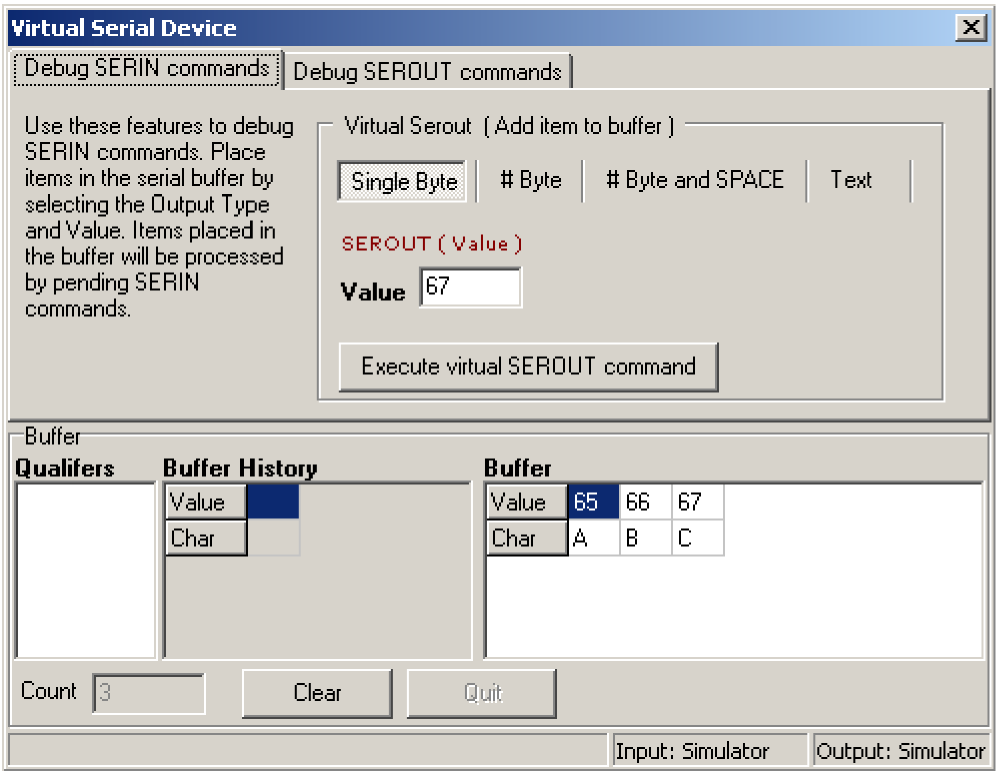
Pulsout



Pulsin



Serial


Virtual Serial Device

Debug SERIN commands | Debug SEROUT commands

Use these features to debug SERIN commands. Place items in the serial buffer by selecting the Output Type and Value. Items placed in the buffer will be processed by pending SERIN commands.

Virtual Serout (Add item to buffer)

Single Byte | # Byte | # Byte and SPACE | Text

SEROUT (Value)

Value 67

Execute virtual SEROUT command

Buffer

Qualifers

Buffer History

Value	
Char	

Buffer

Value	65	66	67
Char	A	B	C

Count 3

Clear | Quit

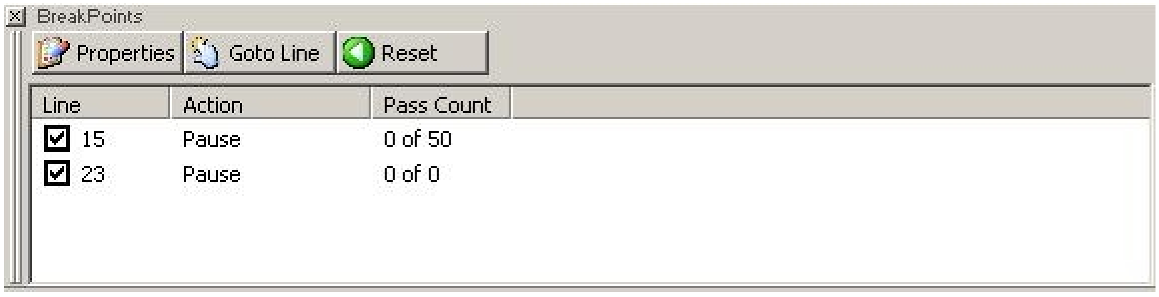
Input: Simulator | Output: Simulator

Breakpoints

Set/remove a breakpoint in the editor using the breakpoint button. Breakpoints are shown as large red dots on the breakpoint line.

Breakpoints cause program execution to stop on a line before it is executed. They affect running, stepping and animation of the code.

The Breakpoint Window contains the properties of each breakpoint in the editor.



Line	Action	Pass Count
✓ 15	Pause	0 of 50
✓ 23	Pause	0 of 0

Disable Breakpoint

Disable a breakpoint by clicking the check box in the breakpoint window, the breakpoint will still be visible as a grey circle but will not stop program execution.

Pass Count

Setting a breakpoints pass count, causes the processor to ignore the breakpoint for a fixed number of times. Set a pass count by clicking the breakpoint properties and setting the pass count number.

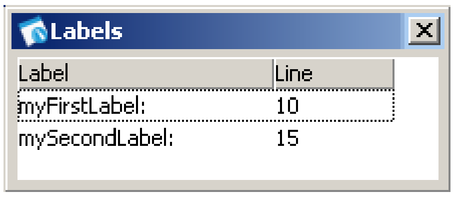
Debugger Windows

A selection of windows are available to assist the debug process.

Label Window

The Labels window displays a summary of all labels used in your BASIC program, and is a convenient method of jumping quickly to a line of code.

The window shows the label name and the line on which it is declared.

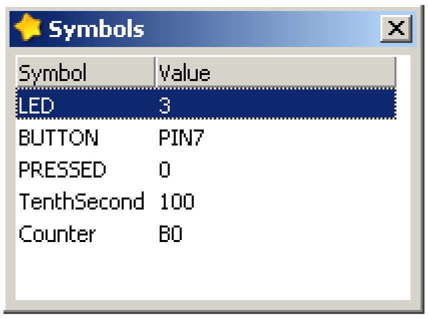


Label	Line
myFirstLabel:	10
mySecondLabel:	15

Symbol Window

The Symbol Window displays a summary of all symbols used in your BASIC program, and is a convenient method of jumping quickly to a line of code.

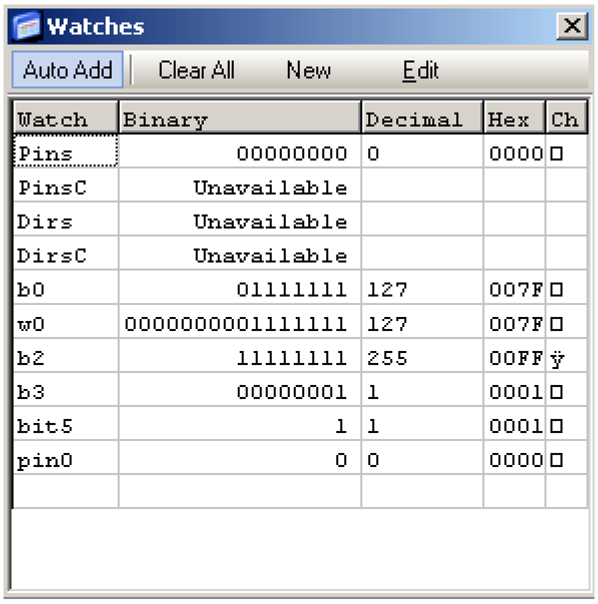
The window shows the symbol name and the line on which it is declared.



Symbol	Value
LED	3
BUTTON	PIN7
PRESSED	0
TenthSecond	100
Counter	B0

Watch Window

The watch window displays the current value of variables (including pin variables) in several common formats.



Watch	Binary	Decimal	Hex	Ch
Pins	00000000	0	0000	□
PinsC	Unavailable			
Dirs	Unavailable			
DirsC	Unavailable			
b0	01111111	127	007F	□
w0	0000000001111111	127	007F	□
b2	11111111	255	00FF	ÿ
b3	00000001	1	0001	□
bit5	1	1	0001	□
pin0	0	0	0000	□

Users can add a new watch (at any time) by clicking the new watch button. However, the use of watches slows the execution of the processor since time is taken to retreive current values and display them in various formats. Deleting all watches will speed up execution.

Variables can be modified during both edit-mode and debug-mode via the Modify Watch Window.

Modified pin related variables e.g. pins, dirs, pin1 etc are reflected in the Pin Window immediately. For example, setting "Pins" to $FF will set all pins high in the pins window. Likewise, clicking on pins in the pin window will will cause the "Pins" variable to change in the watch window. See pin window help for examples.

Watch variables are displayed in several formats:

· Binary (right aligned for easy comparisons)

· Decimal

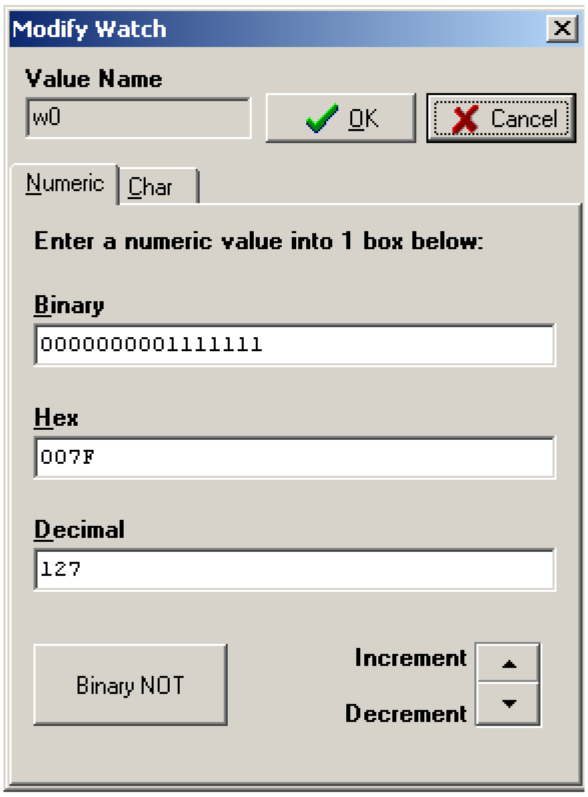
· Hex

· ASCII character

Variables that are used in a program can automatically be added by selecting the Auto Watch button. Auto watches are added when users switch over to Build Mode.

Modify A Watch Variable

The modify watch window is used to modify variables contained within the watch window.



Value Name
w0

Enter a numeric value into 1 box below:

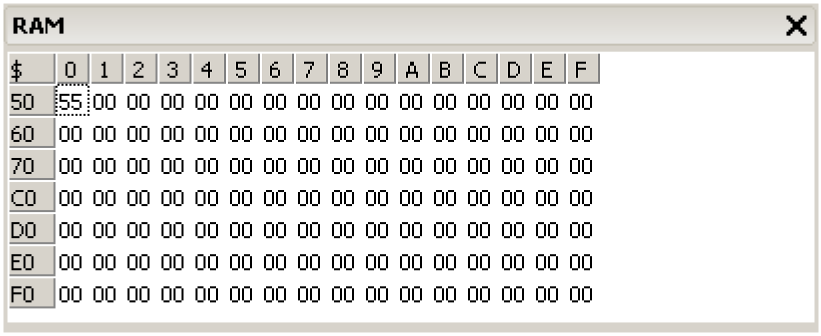
Binary
0000000001111111

Hex
007F

Decimal
127

RAM Window

The RAM window shows the availability and content a controllers RAM memory.



RAM

$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
50	55	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

RAM memory is reset when the simulator is switched to Build Mode.

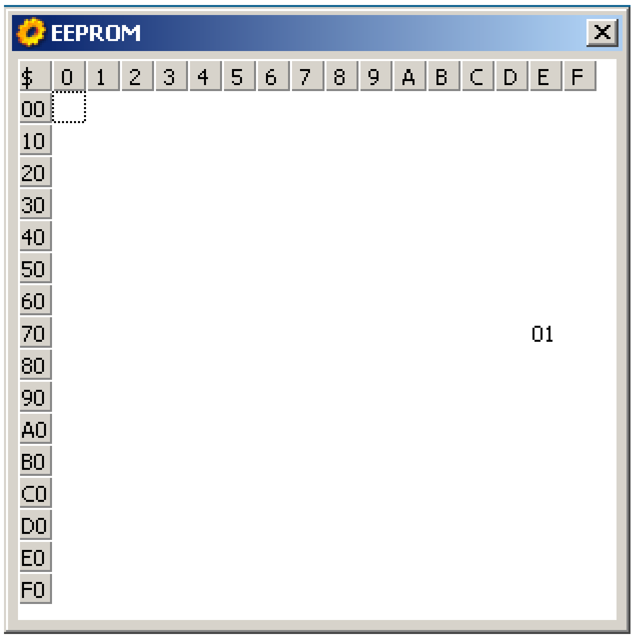
The contents of the cells can be seen changing when the program executes a POKE command.

Clicking the reset button causes RAM memory cells to be set to 0.

Selecting different controllers will show different amounts of available RAM space.

EEPROM Window

The EEPROM window shows the availability and content a controllers EEPROM memory.



$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00																
10																
20																
30																
40																
50																
60																
70															01	
80																
90																
A0																
B0																
C0																
D0																
E0																
F0																

The diagram above shows the cells as they would be prior to the first line being executed in the demo. The contents of the cells can be seen changing when the program executes a WRITE command.

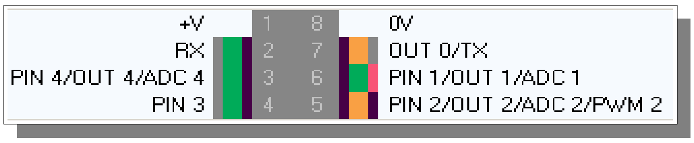
Switching to Build Mode causes the simulator to scan for DATA and EEPROM commands and updates the EEPROM window accordingly. EEPROM can only be reset by switching to Edit Mode and clicking Reset EEPROM

Pin Window

The pin window shows the functionality and status of the micro controllers pins.

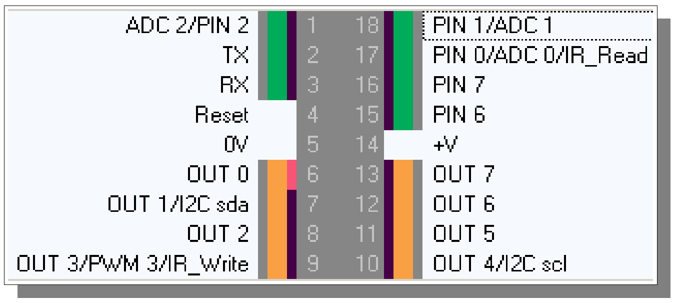
The layout of the micro controllers pins is adjusted as soon as a specific type of controller is selected:

KicChip8



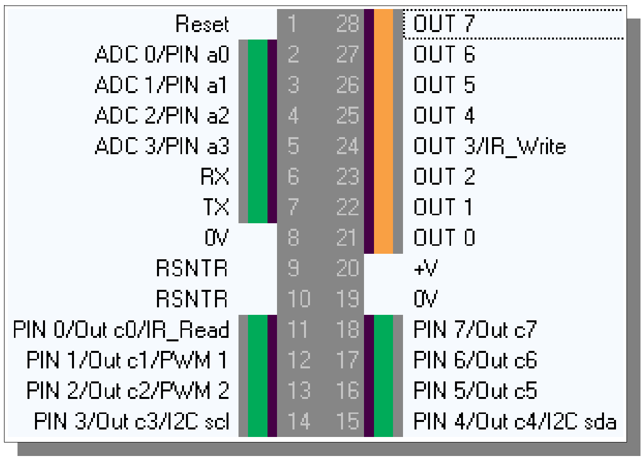
+V	1	8	0V
RX	2	7	OUT 0/TX
PIN 4/OUT 4/ADC 4	3	6	PIN 1/OUT 1/ADC 1
PIN 3	4	5	PIN 2/OUT 2/ADC 2/PWM 2

KicChip18



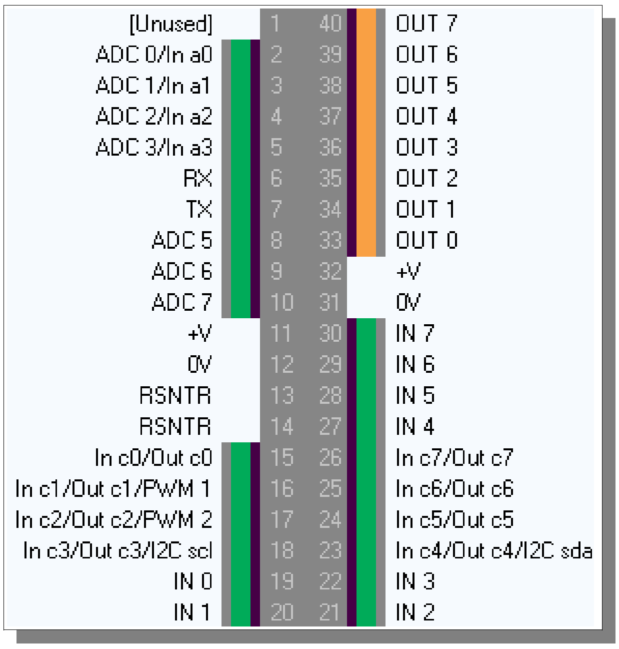
ADC 2/PIN 2	1	18	PIN 1/ADC 1
TX	2	17	PIN 0/ADC 0/IR_Read
RX	3	16	PIN 7
Reset	4	15	PIN 6
0V	5	14	+V
OUT 0	6	13	OUT 7
OUT 1/I2C sda	7	12	OUT 6
OUT 2	8	11	OUT 5
OUT 3/PWM 3/IR_Write	9	10	OUT 4/I2C scl

KicChip28



Reset	1	28	OUT 7
ADC 0/PIN a0	2	27	OUT 6
ADC 1/PIN a1	3	26	OUT 5
ADC 2/PIN a2	4	25	OUT 4
ADC 3/PIN a3	5	24	OUT 3/IR_Write
RX	6	23	OUT 2
TX	7	22	OUT 1
0V	8	21	OUT 0
RSNTR	9	20	+V
RSNTR	10	19	0V
PIN 0/Out c0/IR_Read	11	18	PIN 7/Out c7
PIN 1/Out c1/PWM 1	12	17	PIN 6/Out c6
PIN 2/Out c2/PWM 2	13	16	PIN 5/Out c5
PIN 3/Out c3/I2C scl	14	15	PIN 4/Out c4/I2C sda

KicChip40



[Unused]	1	40	OUT 7
ADC 0/In a0	2	39	OUT 6
ADC 1/In a1	3	38	OUT 5
ADC 2/In a2	4	37	OUT 4
ADC 3/In a3	5	36	OUT 3
RX	6	35	OUT 2
TX	7	34	OUT 1
ADC 5	8	33	OUT 0
ADC 6	9	32	+V
ADC 7	10	31	0V
+V	11	30	IN 7
0V	12	29	IN 6
RSNTR	13	28	IN 5
RSNTR	14	27	IN 4
In c0/Out c0	15	26	In c7/Out c7
In c1/Out c1/PWM 1	16	25	In c6/Out c6
In c2/Out c2/PWM 2	17	24	In c5/Out c5
In c3/Out c3/I2C scl	18	23	In c4/Out c4/I2C sda
IN 0	19	22	IN 3
IN 1	20	21	IN 2

The content of the pin window is updated as the program is executed (i.e. by HIGH/LOW commands). The pin window is also interactive so pins can be "Pressed" (see scenario below) during edit-mode or debug-mode.

Explanation of Pin diagram:

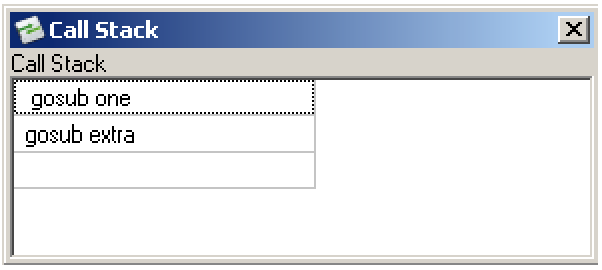
In the middle of the diagram is the body of the micro controller (grey). The body contains the leg numbers. Next to the leg numbers are the state regions (black by default). High pins are red, low pins are black Then the direction region is shown in Orange for an output pin and green for inputs. Next is the connection region (grey by default), floating pins are grey, pins connected to positive supply are shown in red and grounded pins are black.

Finally the pins possible port letters and functions are displayed, clicking this region changes the pins connection status/region (i.e. similar to pressing switches connected to a real controller).

Call Stack Window

The Call Stack Window shows the current stack of GOSUB commands.

This is useful for debugging applications that make extensive use of GOSUB and RETURN commands.



```
gosub one
gosub extra
```

Each entry in the call stack window shows the point at which an outstanding GOSUB was called.

As the simulator executes a RETURN command, the call stack is reduced (popped).

Downloading To KicChip

Code downloading can be selected by pressing the download button, or F9.

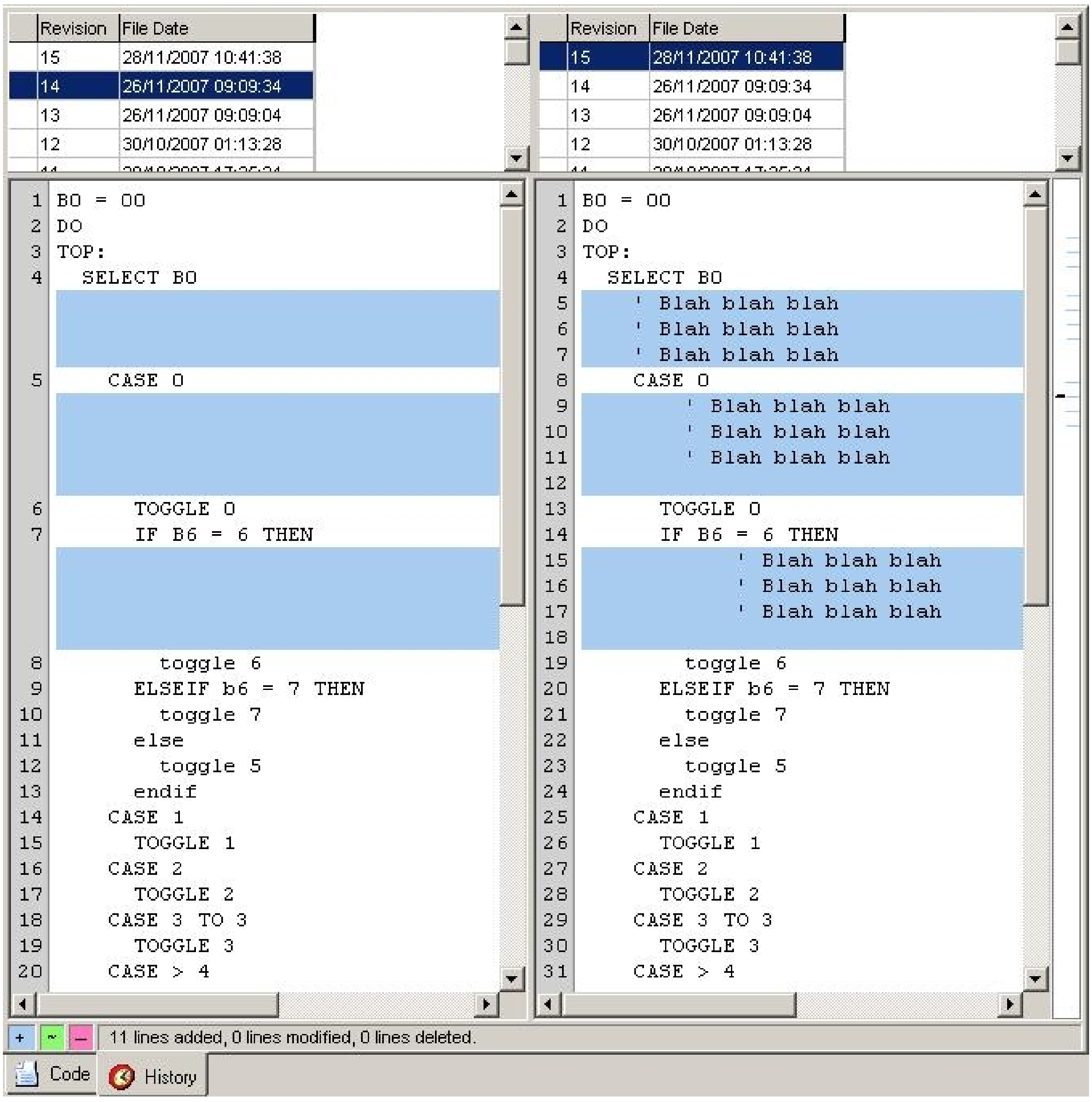
Code download puts your compiled BASIC program into the connected KicChip.

Managing Your Code History

KicStudio has a built code history viewer that allows you to track changes made to your code over time.

simply select the 2 file dates that you want to compare, the code difference window displays Modified , Added and Deleted lines:

Code History Window



Revision	File Date
15	28/11/2007 10:41:38
14	26/11/2007 09:09:34
13	26/11/2007 09:09:04
12	30/10/2007 01:13:28

Revision	File Date
15	28/11/2007 10:41:38
14	26/11/2007 09:09:34
13	26/11/2007 09:09:04
12	30/10/2007 01:13:28

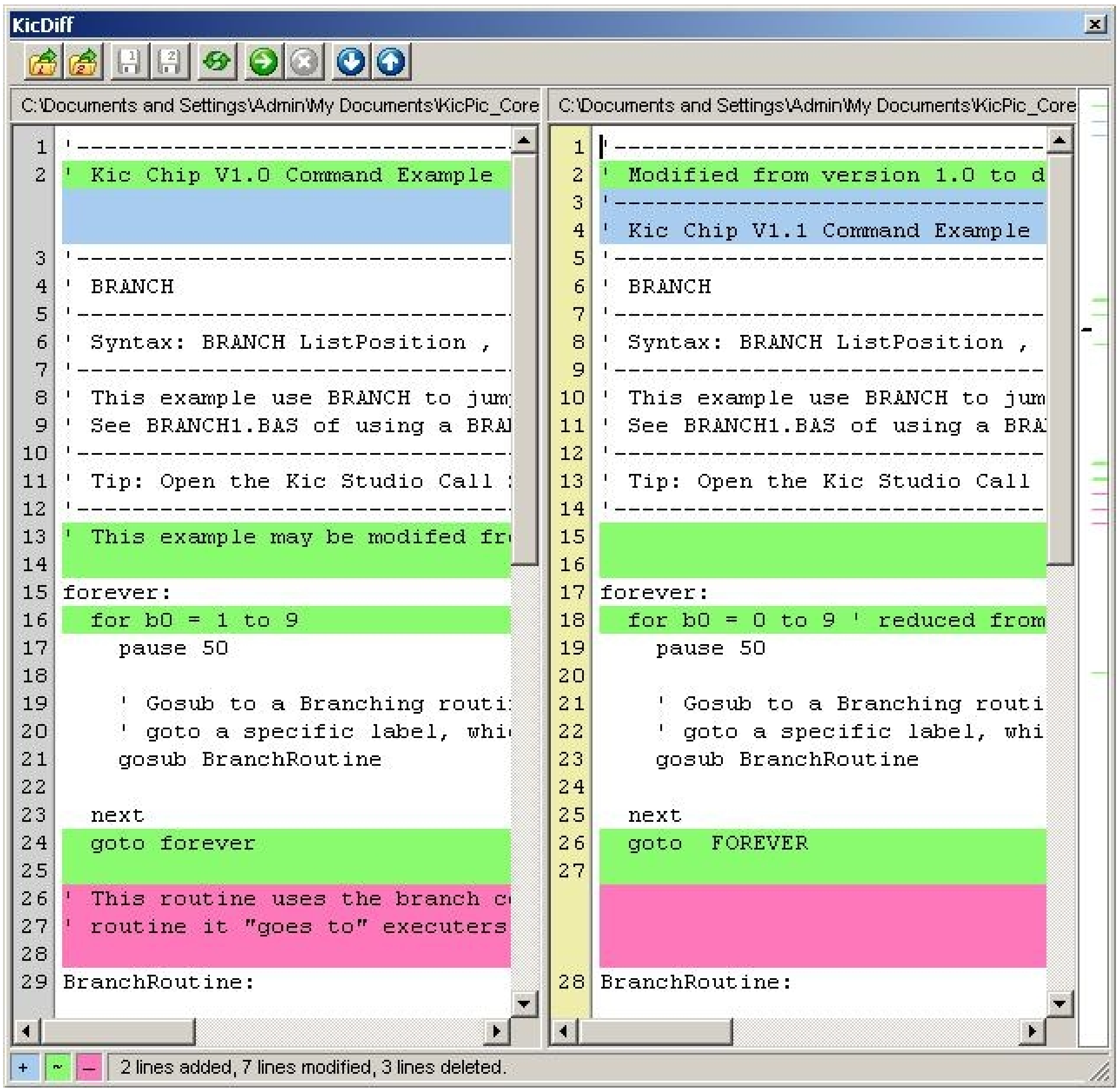
```
BO = 00
DO
TOP:
  SELECT BO



    CASE 0




      TOGGLE 0
      IF B6 = 6 THEN




        toggle 6
      ELSEIF b6 = 7 THEN
        toggle 7
      else
        toggle 5
      endif
    CASE 1
      TOGGLE 1
    CASE 2
      TOGGLE 2
    CASE 3 TO 3
      TOGGLE 3
    CASE > 4
```

```
BO = 00
DO
TOP:
  SELECT BO
     ' Blah blah blah
     ' Blah blah blah
     ' Blah blah blah
    CASE 0
         ' Blah blah blah
         ' Blah blah blah
         ' Blah blah blah

      TOGGLE 0
      IF B6 = 6 THEN
             ' Blah blah blah
             ' Blah blah blah
             ' Blah blah blah

        toggle 6
      ELSEIF b6 = 7 THEN
        toggle 7
      else
        toggle 5
      endif
    CASE 1
      TOGGLE 1
    CASE 2
      TOGGLE 2
    CASE 3 TO 3
      TOGGLE 3
    CASE > 4
```

+ ~ – 11 lines added, 0 lines modified, 0 lines deleted.

Code Difference Tool

KicStudio also ships with the Code Difference Tool, a more powerful version of the in-built code difference window.



C:\Documents and Settings\Admin\My Documents\KicPic_Core

```
'----------------------------------
' Kic Chip V1.0 Command Example

'----------------------------------
' BRANCH
'----------------------------------
' Syntax: BRANCH ListPosition ,
'----------------------------------
' This example use BRANCH to jum
' See BRANCH1.BAS of using a BRA
'----------------------------------
' Tip: Open the Kic Studio Call
'----------------------------------
' This example may be modifed fr

forever:
  for b0 = 1 to 9
    pause 50

    ' Gosub to a Branching routi
    ' goto a specific label, whi
    gosub BranchRoutine

  next
  goto forever

' This routine uses the branch c
' routine it "goes to" executers

BranchRoutine:
```

C:\Documents and Settings\Admin\My Documents\KicPic_Core

```
'----------------------------------
' Modified from version 1.0 to d
'----------------------------------
' Kic Chip V1.1 Command Example
'----------------------------------
' BRANCH
'----------------------------------
' Syntax: BRANCH ListPosition ,
'----------------------------------
' This example use BRANCH to jum
' See BRANCH1.BAS of using a BRA
'----------------------------------
' Tip: Open the Kic Studio Call
'----------------------------------


forever:
  for b0 = 0 to 9 ' reduced from
    pause 50

    ' Gosub to a Branching routi
    ' goto a specific label, whi
    gosub BranchRoutine

  next
  goto  FOREVER


BranchRoutine:
```

+ ~ – 2 lines added, 7 lines modified, 3 lines deleted.