Table of Contents

CHEAT SHEET - KICBASIC

Functions - Result in **bResult** & **wResult** variables

```
A2D       Channel  ,                    wResult
COUNT     bPinNumber, wDuration      , wResult
I2CREAD   bResult  , (Address)
LOOKDOWN  bFind    , (bData [,bData]), bResult
LOOKUP    bIndex   , (bData [,bData]), bResult
PEEK      bAddress ,                   bResult
PULSIN    bPinNumber, Edge ,           wResult
READ      bAddress,                    bResult
SERIN     bPin, Protocol,              bResult
SERIN     bPin, Protocol,              bResult [,bResults]
SERIN     bPin, Protocol,(bPrefix)     no result
SERIN     bPin, Protocol,(bPrefix),    bResult
SONYREAD  (Result held in INFRA & B13 Variable)
```

Block Statements

```
1. DO                          {code} LOOP
2. DO WHILE Logic              {code} LOOP
3. DO                          {code} LOOP UNTIL  Logic
4. FOR Variable = Start TO Final              {code} NEXT
5. FOR Variable = Start TO Final STEP Size {code} NEXT
6. IF Logic THEN {code}                    ENDIF
7. IF Logic THEN {code} [ELSE] {code} ENDIF
8. SELECT Variable
     CASE Value
       {code}
     CASE Value [, Values]
       {code}
     CASE Value TO Value
       {code}
     CASE Operator Value
       {code}
     ELSE
       {code}
   ENDSELECT
```

Procedures

```
HIGH         bPinNumber
HIGH PORTC   bPinNumber
INPUT        PinName
I2CDEVICE    bAddress   , FAST|SLOW , WORD|BYTE
I2CWRITE     bData      , (Address)
LOW          bPinNumber
LOW  PORTC   bPinNumber
OUTPUT       PinName
POKE         bAddress   , bData
PULSOUT      bPinNumber , Duration
PWMOUT       bPwmPin    , bPeriod  , DutyCycle
TOGGLE       bPinNumber
SEROUT       bPinNumber , Protocol ,(bData [,bData])
SERVO        bPinNumber , bPercent
SONYWRITE    bReceiverID, bData
SOUND        bPinNumber , (bFrequency, bDuration)
WRITE        bAddress   , bData
```

Flow & Control

```
BRANCH bIndex,        (Label [,Labels])
END
EXIT
GOSUB                 Label
GOTO                  Label
IF Logic THEN GOSUB   Label
IF Logic THEN GOTO    Label
IF Logic THEN [EXIT]
NAP
ON      bIndex GOTO   (Label [,labels])
ON      bIndex GOSUB  (Label [,labels])
PAUSE
RETURN
SLEEP
STOP
```

Variables - Modifying a variable also modifies its parts.

WORD	HIGH BYTE	LOW BYTE	BITS
W0	**B1**	**B0**	**BIT7** to **BIT0**
W1	**B3**	**B2**	**BIT15** to **BIT8**
W2	**B5**	**B4**	
W3	**B7**	**B6**	
W4	**B9**	**B8**	
W5	**B11**	**B10**	
W6	**B13/INFRA**	**B12/IRID**	
W7	**B15**	**B14**	
		DIRS	
		DIRSC	
		PINS	
		PINSC	

Bitwise Operators

aaaaaaaa op bbbbbbbb

AND	&	
OR	|	
XOR	^	
XNOR	^/	*i.e. NOT(a XOR b)*
ORNOT	|/	*i.e. a OR (NOT b)*
ANDNOT	&/	*i.e. a AND(NOT b)*

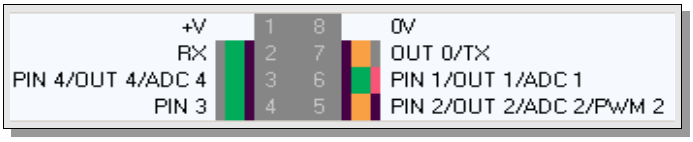
Boolean Operators

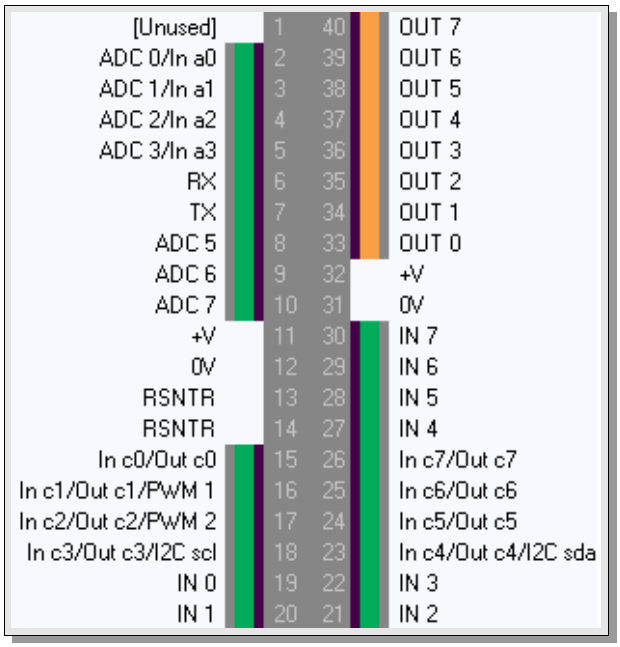
A op B

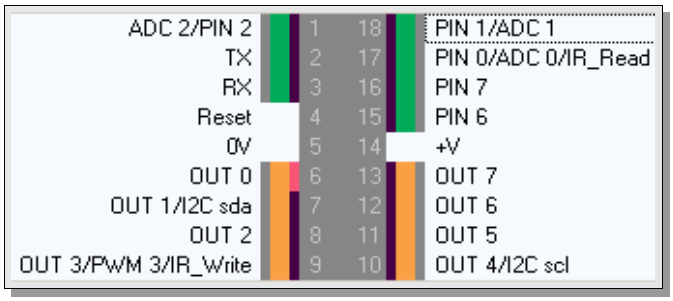
A **AND** B	Both A and B must be TRUE
A **OR** B	Either A or B can be TRUE
A **<>** B	A doesn't equal B
A **=** B	A equals B
A **>=** B	A greater or equal to B
A **>** B	A is greater than B
A **<** B	A less than B
A **<=** B	A less or equal to B

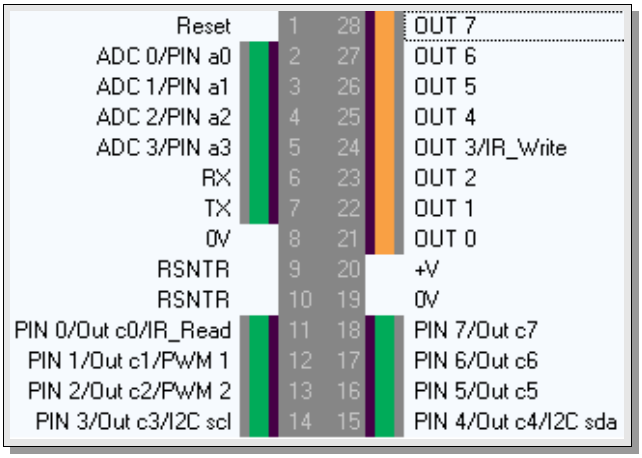
Note synergy between between bitwise and boolean ops.

PIN CONFIGURATIONS









WHITE-SPACE

What is white-space?

White space is made up of **SPACE** & **TAB** characters of the keyboard.

White space is **ignored** by the compiler.

Use white-space to Indent **blocks** of code .

Blocks of code exist within IF THEN & DO..LOOP & FOR..NEXT & SELECT..CASE statements.

Example

```
' Indenting example.
' Notice we indent after a "DO"   and "IF"
' Notice we undent after a "LOOP" and "ENDIF"
DO
    ' .. begin indenting
    IF BUTTON = PRESSED THEN
        ' .. begin more indenting
        TOGGLE LED
        PAUSE  1000
    ENDIF 'undent
LOOP ' undent
END
```

COMMENTS

What are comments?

Comments are... text to help **understand** what is happening inside a program.

Comments are... completely **ignored** by the compiler.

Valid comment start symbols

Apostrophe

The apostrophe **'** character defines the start of a comment. It can appear after a command.

```
            '   This is a comment line.
TOGGLE  0   '   This comment appears after a command.
```

REM

The keyword **REM** defines the start of a comment. It cannot appear on the same line as a command.

```
REM  This is a comment on a line all by itself.
```

VARIABLES

What are variables?

Variables are places in which we place values that we ant to change during the course of a program.

KicBasic variables (pre-defined)

Variable names are **pre-defined**, and each variable may consist of other variables of smaller sizes.

WORD Variable	**HIGH BYTE** Variable	**LOW BYTE** Variable	**BIT** Variables
W0	B1	B0	`BIT7 : BIT6 : BIT5 : BIT4 : BIT3 : BIT2 : BIT1 : BIT0`
W1	B3	B2	`BIT15: BIT14: BIT13: BIT12: BIT11: BIT10: BIT9 : BIT8`
W2	B5	B4	
W3	B7	B6	
W4	B9	B8	
W5	B11	B10	
W6	B13	B12	
W7	B15	B14	

MATHS EXPRESSIONS

What are Maths expressions?

Maths expressions allow us to perform calculations during the course of a program.

The LET command is used to assign a maths expression to a variable.

Integer Maths

All maths is integer based i.e. no fractions.

All numbers are positive i.e. umbers cannot have a value less than zero.

The largest value a number can have is 65535.

KicBasic does not support parenthesis, expressions are evaluated from left to right.

1. Numeric Operators

Integer operators act on **integers** and the result is another integer.

Expression **Operator** Expression	**Result**	**Example**
A + B	A plus B	3 + 2 = 5
A - B	A minus B	3 - 2 = 1
A * B	A times B	3 * 2 = 6
A / B	A divided by B	3 / 2 = 1
A ** B	Hi-byte result of word values A * B	$0300 ** $0200 = $06
A MAX B	Value A but limited to value of B	3 MAX 2 = 2
A MIN B	Value A but at least value B	2 MIN 3 = 3
A MOD B	Remainder of A divided by B	3 MOD 2 = 1

2. Bitwise Operators

Bitwise operators act on the **individual bits** of an integer number.

A **Operator** B		**Meaning**	**Explanation**	**Example**
A AND B	A & B	A AND B	Both A and B must = 1	A = %10101010 B = %11110000 RESULT = %10100000
A OR B	A | B	A OR B	Either A or B must = 1	A = %10101010 B = %11110000 RESULT = %11111010
A XOR B	A ^ B	A XOR B	A or B must = 1, but not both = 1	A = %10101010 B = %11110000 RESULT = %01011010
A ANDNOT B	A &/ B	A AND (NOT B)	A must = 1, B must = 0	A = %10101010 B = %11110000 RESULT = %00001010
A ORNOT B	A |/ B	A OR (NOT B)	Either A = 1 , or B = 0	A = %10101010 B = %11110000 RESULT = %10101111
A XNOR B	A ^/ B	NOT (A XOR B)	A must = B	A = %10101010 B = %11110000 RESULT = %10100101

LOGIC EXPRESSIONS

What is Logic?

Logic, also known as "Boolean Logic", is used to determine if the result of a **condition** is either **TRUE** or **FALSE**.

Boolean Operators

KicBasic does not support parenthesis, expressions are evaluated from left to right.

Expression **Operator** Expression	**Condition**	Order of **Precedence** (highest first)
A > B	A is greater than B	1
A >= B	A greater or equal to B	1
A < B	A less than B	1
A <= B	A less or equal to B	1
A <> B	A doesn't equal B	1
A = B	A equals B	1
A AND B	Both A and B must be TRUE	2
A OR B	Either A or B can be TRUE	2

The following commands use Logic
IF THEN
DO WHILE LOOP
DO LOOP UNTIL

Some operators can also be used in the SELECT..CASE Statement.

IDENTIFIERS

What are identifiers?

Identifier is a general term that applies to items that we name in a program.

We need to give identifying names to Labels and Symbols

It is VERY GOOD practice to choose an identifier name that reflects what the item *means* in real life, for example to store the value of how warm or cold an item is, we would choose an identifying name such as "Temperature".

Identifier Rules		
First character	**MUST BE**	A letter "A" to "Z" "a" to "z" or an underscore "_"
Remaining characters	**MAY BE**	A digit 0..9
Entire identifier	**MUST BE**	Unique - cannot have same name as another identifier
	MUST NOT BE	A reserved or BASIC command
KicBasic identifiers are NOT CASE SENSITIVE. This means the identifiers "myIdentifier" and "MYIDENTIFIER" are the same.		

So any of the following would be valid identifiers:

Temperature

Speed1

_Speed2

Example

```
' KicChip "Identifiers" Example

  ' Some meaningful symbol names to make our program intuitive.

  SYMBOL LED        = 3
  SYMBOL BUTTON     = PIN7
  SYMBOL PRESSED    = 0
  SYMBOL shortTime = 100

DO
  ' This code is self explanatory because we used sensible identifiers
  IF BUTTON = PRESSED THEN
    TOGGLE LED
    PAUSE  shortTime
  ENDIF
LOOP
END
```

LABELS

What are labels?

A labels defines the start of a **Subroutine**. This is a section of code that we can "jump to" and execute.

A label name must be a valid identifier, (see identifier), followed by a semi-colon character ":"

Note: The semi-colon is required when defining the label, but is not needed when referring to it later.

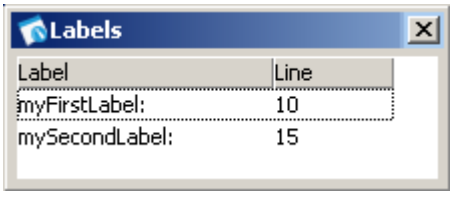
Commands that use labels

Basic Command	Recommended
GOSUB Label	**Yes**
ON bIndex GOSUB (Label [,labels])	**Yes**
BRANCH bIndex, (Label [,Labels])	**No**
ON bIndex GOTO (Label [,labels])	**No**
GOTO Label	**No**
IF Logic THEN [GOTO|GOSUB] Label	**No**

Example

```
' KicChip "LABEL" Example
  SYMBOL LED     = 3
DO
    GOSUB myFirstLabel    ' Use our label
    GOSUB mySecondLabel   ' Use another label
LOOP
myFirstLabel:             ' Here is our first label
    TOGGLE LED
    PAUSE  1000
    RETURN
mySecondLabel:            ' Here is another label
    TOGGLE LED
    PAUSE  1000
    RETURN
END
```

Simulator

The Label Window - Lists all program labels. Double-click a label to jump to it's line of code in the editor.	

SYMBOLS

What are symbols?

Symbols allow us to invent meaningful names for:

1. The pre-defined variables (W0..W7, B0..B13, B0..BIT15)
2. Devices connected to the processor
3. Universal constant values such as 100 Milliseconds being equal to 1/10 second

Symbols must respect the rules for an identifier.

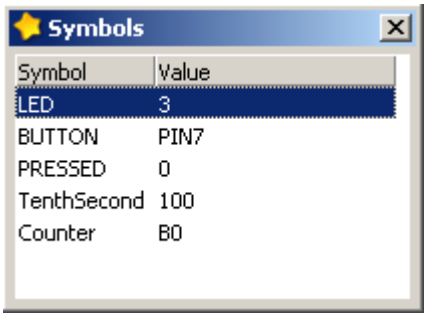
Example

```
' KicChip "SYMBOLS" Example  Some meaningful symbol names to make our program intuitive.
  SYMBOL LED         = 3
  SYMBOL BUTTON      = PIN7
  SYMBOL PRESSED     = 0
  SYMBOL TenthSecond = 100
  SYMBOL Counter     = B0
DO
  ' This code is self explanatory because we used sensible identifiers
  IF BUTTON = PRESSED THEN
    TOGGLE LED
    Counter = Counter + 1
    PAUSE  TenthSecond
  ENDIF
LOOP
END
```

Simulator

The Symbol Window displays a summary of all symbols used in your BASIC program.

Double click a symbol name to jump to its declaration in the source code.



A2D

A2D bADC, wResult

Parameters	
bADC	Specifies the **A**nalog To **D**igital **C**onverter input to use.
wResult	Stores the result of the reading (0 to 1023) - requires a word sized variable.

Summary

A2D reads the analog voltage level at one of the **A**nalog To **D**igital **C**onverter (ADC) pins and returns a 10 bit digital result (0 to 1023).

A word sized variable MUST be used to store the result or information may be lost.

Description

A2D uses the on-board hardware (ADC Module) to perform it's operations, therefore the number of available ADC pins will vary from device to device, see table below:

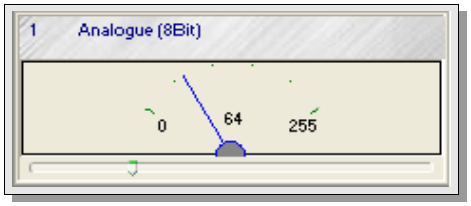
ADC Channels		
Device	Analog Channel/Pin Numbers	Multiplexed (shared) with:
Kic8	1, 2 & 4	Input/Output port
Kic18	0, 2 & 4	Input port
Kic28	0, 1, 2 & 3	InputA
Kic40	0, 1, 2 & 3	InputA
	5, 6 & 7	Dedicated (ADC Only)

Example

```
' KicChip Analog Example
' Kic18
' Devices:
  SYMBOL Potentiometer = 1   ' Use ADC1

DO
  A2D Potentiometer , W0      '  read analog value into variable w0
  DEBUG                       '  show variable w0 in debug window
LOOP
END
```

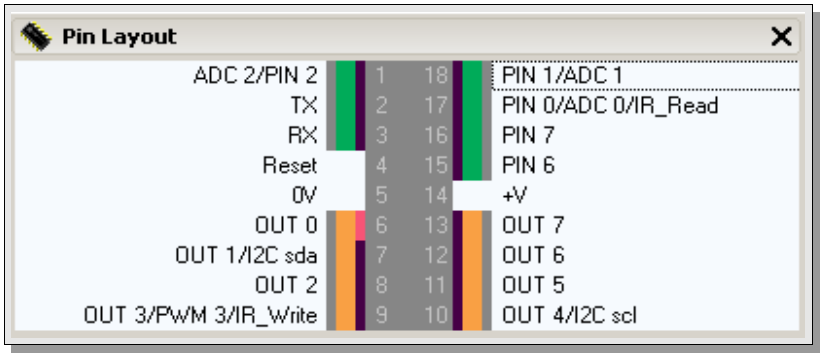
Simulator

KicStudio creates an interactive Virtual Potentiometer when this command is used.



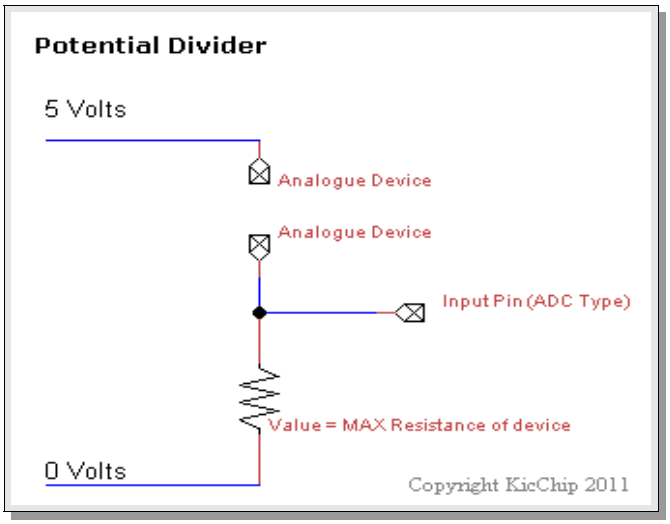
The pin window prefixes the analog pins with "**ADC**" followed by the channel number, e.g. "**ADC 1**".

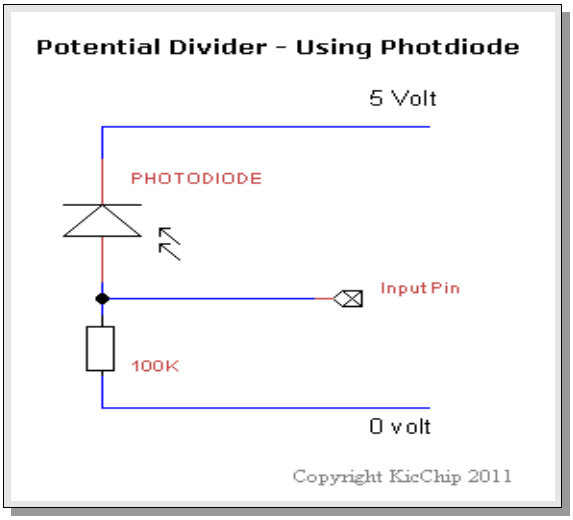
ADC inputs are shared with digital inputs and are coloured **Green** in the Pin window.



Analog Circuit Diagram (Potential Divider)

Many analog devices can be connected to a KicChip using a standard circuit layout called a potential divider.





AND (BITWISE)

```
LET Result = Value1 AND Value2
LET Result = Value1 &   Value2
```

Parameters	
Result	Byte or word sized variable or output port.
Value1	Byte or word sized value, variable, input port or constant.
Value2	Byte or word sized value, variable, input port or constant.
It is common practice to use binary-notation with bitwise commands e.g. %10101010. This makes it easier to identify individual BITS when debugging.	

Summary

Apply logical **AND** to the binary **BITS** of two integers.

The result is 1 - only if **both** values are 1

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
1	1	**1**	1	0	0	0	0	Value1
1	0	**1**	0	1	0	1	0	Value2
1	0	**1**	0	0	0	0	0	Result

See also AND , ANDNOT, OR , XOR , XNOR , ORNOT

Description

Applying logical AND is often referred to as "masking".

What is masking?

Masking is a kind of "filtering" technique that allows us to take some binary data, and modify it so that only the bits we want to *look at* are kept at their original value (masked-in) and any remaining uninteresting bits are cleared to 0 (masked-out).

This leaves a *clean looking* result, either:

1. The whole result is **zero** - in which case **none** of the bits we are measuring are set.
2. The whole result is **no-zero**, only the interesting bits are set.

Take a look at the logic table above, you can see that value1 (our mask) only allows value2 (our data) set-bits to appear in the result if value1 bit is also set.

Masking use useful when reading an input port and we only want to respond if a couple of pins are set:

Pins Mask Example

```
' KicChip "AND MASKING" Example
' Demonstrates how to "Mask Out" certain bits of a byte.

' Our mask is set so that only the 2 lowest bits are set, this means
' that when we apply the mask by using the AND command,
' only the 2 lowest bits of our source value will pass through the mask.
' Simply compile this demo and simulate,

SYMBOL mask        = %00000011    ' Our mask has only the lowest 2 bits

DO

 B0 = PINS AND MASK   'Filter out any bits that are not bit1 & bit0

 IF B0 <> 0 THEN      ' press any buttons you like
    TOGGLE 3          ' but the toggle only happens
    PAUSE  100        ' when pins0 or pin1 are set
  ENDIF

LOOP
END
```

Example

This more general example shows masking in a loop with a range of numbers:

```
' KicChip "AND (bitwise)" Example
' Demonstrates how to "Mask Out" certain bits of a byte.

' Our mask is set so that only the 2 lowest bits are set, this means
' that when we apply the mask by using the AND command,
' only the 2 lowest bits of our source value will pass through the mask.
' Simply compile this demo and simulate,
' observe values b0 & b1 in the watch window..
SYMBOL sourceValue = b0
SYMBOL mask        = %00000011    ' Our mask has only the lowest 2 bits
SYMBOL resultValue = b1

DO
 FOR sourceValue = 0 TO 254
   resultValue = sourceValue AND mask
 NEXT
LOOP
END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

ANDNOT (BITWISE)

```
LET Result = Value1 ANDNOT Value2
LET Result = Value1 &/     Value2
```

Parameters	
Result	Byte or word sized variable or output port.
Value1	Byte or word sized value, variable, input port or constant.
Value2	Byte or word sized value, variable, input port or constant.
It is common practice to use binary-notation with bitwise commands e.g. %10101010. This makes it easier to identify individual BITS when debugging.	

Summary

Apply logical **A AND (NOT B)** to the binary **BITS** of two integers.

The result is 1 - only if A is 1 and B is 0 (could be read as A > B)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
1	**1**	1	**1**	0	0	0	0	Value1
1	**0**	1	**0**	1	0	1	0	Value2
0	**1**	0	**1**	0	0	0	0	Result

See also AND , ANDNOT, OR , XOR , XNOR , ORNOT

Example

```
' KicChip "ANDNOT (bitwise)" Example
' Demonstrates how to "Mask Out" certain bits of a byte.

' Our mask is set so that only the 2 lowest bits are set, this means
' that when we apply the mask by using the ANDNOT command,
' the 2 lowest bits of our source value will NOT pass through the mask.
' Simply compile this demo and simulate,
' observe values b0 & b1 in the watch window..
SYMBOL sourceValue = b0
SYMBOL mask        = %00000011    ' Our mask has only the lowest 2 bits
SYMBOL resultValue = b1

DO
 FOR sourceValue = 0 TO 254
   resultValue = sourceValue ANDNOT mask
 NEXT
LOOP
```

```
END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

AND (CONDITION)

```
IF LogicExpA AND LogicExpB THEN
  ' Code block here
ENDIF
```

Arguments	
LogicExpA	A Boolean Logic Expression - See Logic Expressions.
LogicExpB	A Boolean Logic Expression - See Logic Expressions.

Summary

Performs logical AND on 2 expressions, the result is TRUE only if both expressions are TRUE.

Description

This command can be used to test for multiple conditions before executing an operation, it is often used in conjunction with the OR command.

See also Or Conditional Command

See also AND Bitwise Command

Example

```
' KicChip "AND (Conditional)" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.

  SYMBOL LogicB  = 1      ' Our "Test" data, change this for testing..

DO
  IF BUTTON = PRESSED AND LogicB = 1 THEN ' Button pressed AND data is 1
    TOGGLE LED                          ' Toggle LED
    PAUSE  1000                         ' Pause 1 second
  ENDIF
LOOP
END
```

BRANCH

BRANCH bIndex,(Label [,Labels])

Parameters	
bIndex	A byte variable (B0..B13)
Label , Labels	Any pre-defined variable name, or a coma separated list of label names

Not recommended

Summary

GOTO to a label – via a table lookup of label names, indexed by a byte value (bIndex).

The list is zero based - so the first label would correspond to a bIndex value of 0.

If bIndex is too large for the list of labels, then the command does nothing.

This command is NOT recommended as it uses the GOTO command and can result in untidy code.

Try to use ON GOSUB instead.

Example

```
' KicChip "BRANCH" Example
' Kic18
' Note: The same thing an be achieved using the preferred "SELECT" command

DO
  FOR B0 = 0 TO 4
    BRANCH B0 , (label1A, label1A, label1B, label1B, label1C, label1D)
    continue:
  NEXT
LOOP

label1A:
  TOGGLE 0
  GOTO continue

label1B:
  TOGGLE 1
  GOTO continue

label1C:
  TOGGLE 2
  GOTO continue

label1D:
```

```
  TOGGLE 3
  GOTO continue
END
```

Simulator

The Label Window can help with this command.

COUNT

COUNT bPinNumber, wDuration, wResult

Parameters	
bPinNumber	The input pin from which to count the pulses.
wDuration	The length of time in which to count for.
wResult	A WORD sized variable (W0..W7) to store the count result.

Summary

Count the number of pulses on an input pin during a time interval.

Description

COUNT watches the state of an input pin and counts the number of low to high transitions on that pin during a specified amount of time.

This command allows for a 16 bit result (a possible 65535 transitions).

At 8MHz the input pin is checked every 10us, so the highest pulse frequency possible is 50kHz assuming the 50% duty cycle.

Example

```
' KicChip "COUNT" Example
' KIC18

' Devices (based on KicStand):
  SYMBOL BUTTON  = PIN7    ' Button is connected to input 7 , "active low"

DO
  '   count pulses during a 3 second period
  COUNT Button , 3000, W0
  '   send result to screen
  DEBUG
LOOP
END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

DO .. LOOP

```
DO
  ' Code block here
LOOP
```

Summary

Execute commands in an **indefinite** loop.

The EXIT command can be used to exit out of a loop.

See conditional loops: DO WHILE & DO UNTIL

See Indenting Blocks Of Code

Example - Loop Forever

```
' KicChip "DO LOOP (Forever) " Example
' This loop has no "condition" so loops forever
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3


DO
  TOGGLE LED   ' show something happening
  PAUSE  1000  ' pause for 1 second
LOOP
END
```

DO WHILE LOGIC .. LOOP

```
DO WHILE Logic
  ' Code block here
LOOP
```

Arguments	
Logic	A Boolean Logic Expression - See Logic Expressions.

Summary

While a logic condition is True, execute a block of code.

The code is **not** guaranteed to execute **at least once**, since the logic test is performed first.

If you prefer to execute your code at least once, and then test to quit, use the DO UNTIL variation instead.

The loop may also be terminated via the EXIT statement.

Example

```
' KicChip "DO WHILE" Example
' While  the button is pressed, the led toggles
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is “Sourcing” power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , “active low”
  SYMBOL PRESSED = 0      ' “Pull-ups”  used, so button is active low.


DO
 DO WHILE BUTTON = PRESSED
   TOGGLE  LED
   PAUSE  1000
 LOOP
LOOP
END
```

DO LOOP .. UNTIL LOGIC

```
DO
  ' Code block here
LOOP UNTIL Logic
```

Arguments	
Logic	A Boolean Logic Expression - See Logic Expressions.

Summary

Execute a block of code UNTIL a logic test result is True.

The code is guaranteed to execute **at least once**, then the logic is tested which may terminate the loop.

If you prefer to test your logic prior to executing any code then use the DO WHILE variation instead.

The loop may also be terminated via the EXIT statement.

Example

```
' KicChip "DO UNTIL" Example
' The led toggles, until button is pressed
' Devices (based on KicStand):
  SYMBOL LED     = 3       ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7    ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0       ' "Pull-ups"  used, so button is active low.


DO
  TOGGLE LED
  PAUSE  1000
LOOP UNTIL BUTTON = PRESSED
END
```

END

END

Summary

End the running of a program.

Description

The END command stops the processor from executing any further commands.

Note that the default code provided in KicStudio automatically adds the END command to the last line of your program.

The END command is not always reached , but should always be added for good coding practice.

Example

```
' KicChip  "END" Example
' Devices (based on KicStand):
  SYMBOL MOTOR   = 3      ' Motor connected to PWM3


 ' PWM Output for 3 second


  PWMOUT MOTOR, 140, 50  ' turn on PWM
  SERVO  0 , 50          ' turn on servo
  HIGH   1               ' turn on LED
  PAUSE  3000


 ' ..PWM goes off, servos keep running


END
```

EXIT

EXIT

Summary

EXIT is used to prematurely terminate a program loop.

Applies to the following.
FOR ..... NEXT
DO WHILE ..... LOOP
DO ..... LOOP UNTIL

See also IF THEN EXIT

Description

EXIT can be used to test for special conditions that should terminate a loop.

Care should be taken however as over-use of this command can lead to untidy code.

Attempting to use the EXIT command outside of a loop will not compile and will result in a "Label not found" message.

Example

```
' KicChip "EXIT" Example
' Led toggles until button pressed
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.


DO
  TOGGLE LED                      ' show something happening
  PAUSE  500                      ' wait for 0.5sec
  IF BUTTON = PRESSED THEN EXIT   ' exit the loop if button pressed
LOOP
END
```

FOR [STEP] .. NEXT

```
FOR Variable = Start TO Final [STEP Size]
  ' Code block here
NEXT
```

Arguments	
Variable	Variable in which to store the current count value.
Start	The starting value for the variable.
Final	The end value for the variable.
Size	An (optional) increment value.

Summary

Executes a block of commands for a fixed number of times

Description

1. The **FOR** loop initialises the **Variable** argument with the **Start** argument.
2. The code block is **executed**.
3. The **NEXT** statement increments the Variable by the **Size** argument, or **1** if not specified.
4. **If** the Variable value has not exceed the **Final** argument then repeat from step2.
5. Otherwise continue with code **after** the NEXT statement.

Example - Simple Loop

```
' KicChip "FOR NEXT" Example
' Led flashes at a slower pace each time it toggles
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is “Sourcing” power from output 3


DO
  FOR B0 = 5 TO 95
    TOGGLE LED            ' Toggle the LED
    PAUSE  B0             ' Pause for the next duration
  NEXT
LOOP
END
```

Example - Step Value

```
' KicChip "FOR NEXT STEP" Example
' Led flashes at a slower pace each time it toggles
' the rate of increase is 5 times faster than the simple demo.
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is “Sourcing” power from output 3


DO
  FOR B0 = 5 TO 95 STEP 5
    TOGGLE LED            ' Toggle the LED
    PAUSE  B0             ' Pause for the next duration
  NEXT
LOOP
END
```

GOSUB

GOSUB Label

Arguments	
Label	The label to jump to.

Summary

The GOSUB command cause the KicChip to jump to a Label - i.e. **subroutine**. It is short for GO SUBROUTINE.

The code within the subroutine is executed until it reaches a RETURN statement, whereupon the KicChip continues executing the code immediately after the GOSUB statement.

Subroutines can call other subroutines (nested subroutine calls), but care must be taken not to exceed the limit for the number of nested subroutine calls.

Description

Good programming requires that users break down a program into smaller parts - known as subroutines.

Subroutines can be re-used by other programs and can save program space but avoiding repeating sections of code.

Typically a program will consist of a MAIN loop, followed by a set of subroutines.

Example

```
' KicChip "GOSUB RETURN" Example
' If button is pressed then subroutine to toggle fast is called
' else the subroutine to toggle slowly is called
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.
#74.4.The Call Stack Window|outline
DO
  IF BUTTON = PRESSED THEN
    GOSUB toggleFast        ' Execute our toggle fast subroutine
  ELSE
    GOSUB toggleSlow        ' Execute our toggle fast subroutine
  ENDIF
LOOP

toggleSlow:
  TOGGLE LED     ' Toggle the LED
  PAUSE  1000    ' Pause for 1 second
  RETURN
```

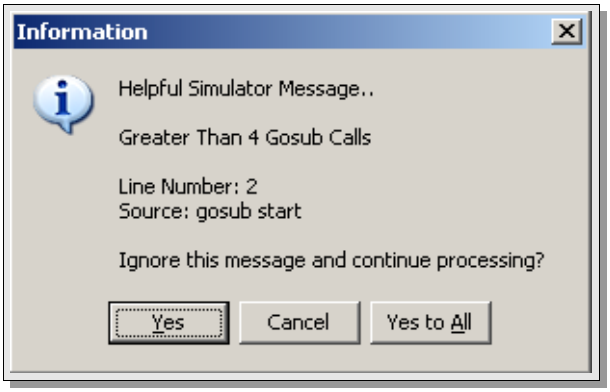
```
toggleFast:
  TOGGLE LED      ' Toggle the LED
  PAUSE  500      ' Pause for half a second
  RETURN

END
```

Simulator

The Label Window can help with this command.

The Call Stack Window can help with this command.

An error message appears if you exceed the number of nested calls to subroutines.



GOTO

GOTO Label

Arguments	
Label	The label to jump to.

Not recommended

Summary

The GOTO command causes the KicChip to begin executing code starting at the Label

Example

```
' KicChip "GOTO" Example
' LED toggles until button pressed

'*************************************************************
' To improve program style, "GOTO" should be avoided if possible!
'*************************************************************

' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.

DO
  TOGGLE LED              ' Toggle LED
  PAUSE  500              ' Pause half second

  IF BUTTON = PRESSED THEN GOTO QUIT  'break out of loop if button pressed

LOOP

QUIT:
END
```

Simulator

The Label Window can help with this command.

HIGH

HIGH bPinNumber

Parameters	
bPinNumber	An output pin number 0..7

Summary

Sets the voltage of an output pin to a "high" state.

Description

Use of the HIGH command on any output will cause that pin to switch on or go high.

The pins voltage level will be equal to approximately Vcc - 0.6v generally considered +5v.

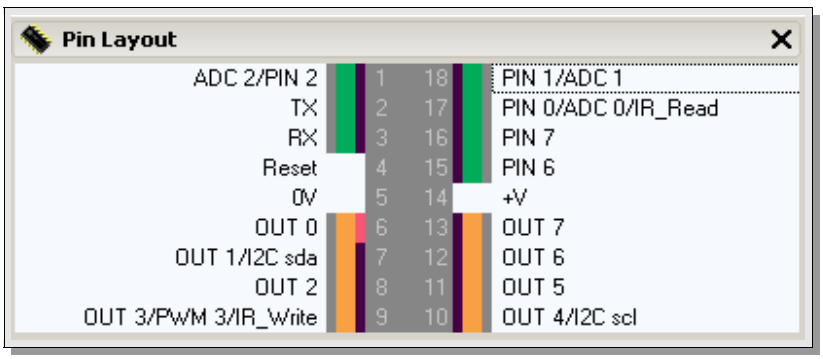
See also LOW command

Example

```
' KicChip "HIGH" "LOW" Example
' LED goes HIGH, LOW forever
' Devices (based on KicStand):
  SYMBOL LED     = 3       ' Led is "Sourcing" power from output 3


DO
  HIGH  LED       '   turn ON LED
  PAUSE 1000      '   wait 1 second
  LOW   LED       '   turn OFF LED
  PAUSE 1000      '   wait 1 second
LOOP
END
```

Simulator

This command creates a virtual a virtual LED Device and shows changes to the Pin Window.



HIGH PORTC

HIGH PORTC bPinNumber

Parameters	
bPinNumber	A PORT-C pin number 0..7

Summary

Sets the voltage of an output pin on PORT-C to a “high” state.

See also LOW PORTC Command.

Description

Use of the HIGH PORTC command on any PortC output will cause that pin to switch on or go high.

The pins voltage level will be equal to approximately Vcc-0.6v generally considered +5v.

Note that DIRSC must be set to output for each pin that uses this command.

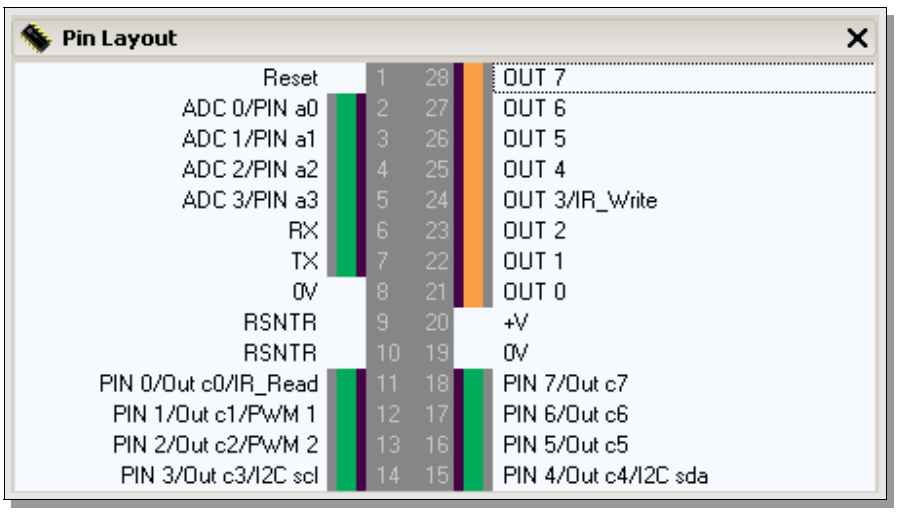
Only the Kic28 and Kic40 use this command.

Example

```
' KicChip "HIGH & LOW PORTC" Example - Select kic18 or kic40

LET DIRSC = %11111111    ' set PortC to all outputs
DO
  HIGH  PORTC 0          ' set output 0 on PortC high
  PAUSE 1000             ' wait 1 second
  LOW   PORTC 0          ' set output 0 on PortC low
  PAUSE 1000             ' wait 1 second
LOOP
END
```

Simulator

This command creates a virtual a virtual LED Device and shows changes to the Pin Window.



I2CDEVICE

I2CDEVICE bAddress, FAST|SLOW , WORD|BYTE

Parameters	
bAddress	Unique address for each device
FAST or SLOW	Check device datasheet
WORD or BYTE	Check device datasheet
Check I2C Device datasheet for correct values for each parameter.	

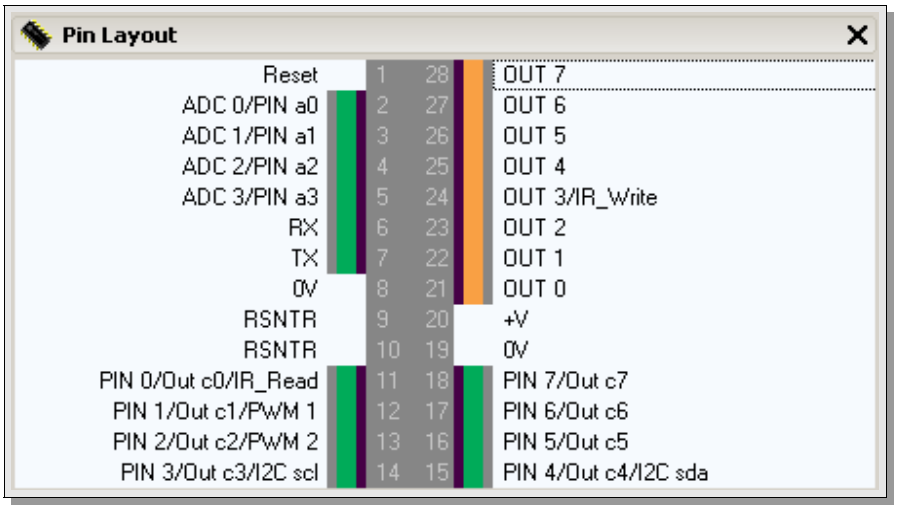
Summary

Configures the KicChip I2C interface for a particular I2C device.

Once the I2CDEVICE command has been executed, the KicChip is ready to start reading and writing to the I2C device via the I2CREAD and I2CWRITE commands.

Description

Only available for Kic28 & Kic40, Labelled as "**I2C SCL**" "**I2C SDA**" in pin window.



Example

```
' KicChip "I2C" Example - Select Kic28

' Configure i2c for external EEPROM
I2CDEVICE %10100000 , FAST , WORD

' Write some values to the external EEPROM
FOR W0 = $0000 TO $FFFF
    I2CWRITE W0,  (B0)  ' Write values to consecutive EEPROM addresses
```

```
NEXT

' Read values and show on the output pins
FOR W0 = $0000 TO $FFFF
    I2CREAD W0, (PINS) ' Read EEPROM data and show on output pins
    PAUSE  1000
NEXT
END
```

I2CREAD

I2CREAD bResult, (Address)

Parameters	
bResult	Variable name to store the result of the read.
Address	Address to read from within the device.

Summary

Read value(s) from the i2c data bus, starting from a specified address.

Note: For this command to work properly, the KicChip must be configured for a particular I2C device via the I2CDEVICE command.

See also I2CWRITE.

Example

```
' KicChip "I2C" Example - Select Kic28

' Configure i2c for external EEPROM
I2CDEVICE %10100000 , FAST , WORD

' Write some values to the external EEPROM
FOR W0 = $0000 TO $FFFF
    I2CWRITE W0,  (B0)  ' Write values to consecutive EEPROM addresses
NEXT

' Read values and show on the output pins
FOR W0 = $0000 TO $FFFF
    I2CREAD W0, (PINS) ' Read EEPROM data and show on output pins
    PAUSE  1000
NEXT
END
```

I2CWRITE

I2CWRITE bData , (Address)

Parameters	
bData	Value to send to the I2C device
Address	An address in which to store the value in the I2C Device

Summary

Write value(s) to the i2c data bus, starting from a specified address.

Note: For this command to work properly, the KicChip must be configured for a particular I2C device via the I2CDEVICE command.

Example

```
' KicChip "I2C" Example - Select Kic28

' Configure i2c for external EEPROM
I2CDEVICE %10100000 , FAST , WORD

' Write some values to the external EEPROM
FOR W0 = $0000 TO $FFFF
    I2CWRITE W0,  (B0)  ' Write values to consecutive EEPROM addresses
NEXT

' Read values and show on the output pins
FOR W0 = $0000 TO $FFFF
    I2CREAD W0, (PINS) ' Read EEPROM data and show on output pins
    PAUSE  1000
NEXT
END
```

IF LOGIC THEN .. ENDIF

```
IF Logic THEN
  ' Code block here
ENDIF
```

Arguments	
Logic	A Boolean Logic Expression - See Logic Expressions.

Summary

Execute a block of code if a logic expression is True.

See also IF THEN ELSE

Example

```
' KicBasic "IF THEN ENDIF" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3       ' Led is “Sourcing” power from output 3
  SYMBOL BUTTON  = PIN7    ' Button is connected to input 7 , “active low”
  SYMBOL PRESSED = 0       ' “Pull-ups”  used, so button is active low.


DO
  IF BUTTON = PRESSED THEN ' A logic test, if True then
    TOGGLE LED             '   perform this action
    PAUSE  1000            '   followed by this action
  ENDIF                    ' End of the test and actions
 ' .. carry on here
LOOP
END
```

IF LOGIC THEN .. ELSE ..

```
IF Logic THEN
  ' Code block here
ELSE
  ' Code block here
ENDIF
```

Arguments	
Logic	A Boolean Logic Expression - See Logic Expressions.

Summary

Execute a block of code if a logic expression is True, execute an alternate set of instructions of the logic is False.

Example

```
' KicBasic "IF THEN ELSE ENDIF" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is “Sourcing” power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , “active low”
  SYMBOL PRESSED = 0      ' “Pull-ups”  used, so button is active low.

DO
  IF BUTTON = PRESSED THEN ' A logic test, if True then
    TOGGLE LED             '   perform this action
    PAUSE  1000            '   followed by this action
  ELSE                     ' otherwise..
    TOGGLE LED             '   perform this action
    PAUSE   500            '   followed by this action
  ENDIF                    ' End of the test and actions
 ' .. carry on here
LOOP
END
```

IF LOGIC THEN EXIT

IF Logic THEN EXIT

Arguments	
Logic	A Boolean Logic Expression - See Logic Expressions.

Summary

Exit is used to prematurely terminate a program loop (See EXIT), so this command allows a Logic Test to be applied first.

Loops that can be "Exited" out of are
FOR ..... NEXT
DO WHILE ..... LOOP
DO ..... LOOP UNTIL

Example

```
' KicChip "EXIT" Example
' Led toggles until button pressed
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.


DO
  TOGGLE LED                      ' show something happening
  PAUSE  500                      ' wait for 0.5sec
  IF BUTTON = PRESSED THEN EXIT   ' exit the loop if button pressed
LOOP
END
```

IF LOGIC THEN GOSUB

IF Logic THEN GOSUB Label

Arguments	
Logic	A Boolean Logic Expression - See Logic Expressions.
Label	A program label indicating the start of a subroutine

Summary

Perform a logic test and execute a GOSUB if the result is True.

Care must be taken not to exceed the number of available nested GOSUB calls.

Description

The same thing can be achieved by using a multi-line IF THEN ENDIF structure, that way you can place a breakpoint on the GOSUB command during the simulation process.

Block Example

```
' using this structure allows 2 simulation breakpoints to be used instead of 1.
IF b0= 1 THEN     ' place breakpoint here
  GOSUB myLabel   ' and place breakpoint here
ENDIF
```

Example

```
' KicBasic "IF THEN GOSUB" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.


DO
  IF BUTTON = PRESSED THEN ' A logic test, if True then
    GOSUB toggleSlow       '   Call slow subroutine
  ELSE                     ' otherwise..
    GOSUB toggleFast       '   Call fast subroutine
  ENDIF                    ' End of the test and actions
 ' .. carry on here
LOOP


toggleSlow:
    TOGGLE LED
    PAUSE  1000
```

```
  RETURN

toggleFast:
    TOGGLE LED
    PAUSE   500
  RETURN

END
```

IF LOGIC THEN GOTO

IF Logic THEN GOTO Label

Arguments	
Logic	A Boolean Logic Expression - See Logic Expressions.
Label	A program label indicating the start of a subroutine

Summary

Perform a logic test and execute a GOTO if the result is True.

Not recommended

Description

The same thing can be achieved by using a multi-line IF THEN ENDIF structure, that way you can place a breakpoint on the GOTO command during the simulation process.

Block Example

```
' using this structure allows 2 simulation breakpoints to be used instead of 1.
IF b0= 1 THEN     ' place breakpoint here
  GOTO myLabel    ' and place breakpoint here
ENDIF
```

Example

```
' KicBasic "IF THEN GOTO" Example
' This command is NOT recommended, see ON GOSUB instead

' Devices (based on KicStand):
  SYMBOL LED     = 3       ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7    ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0       ' "Pull-ups"  used, so button is active low.

DO
  IF BUTTON = PRESSED THEN ' A logic test, if True then
    GOTO toggleSlow        '   Call slow subroutine
  ELSE                      ' otherwise..
    GOTO toggleFast        '   Call fast subroutine
  ENDIF                     ' End of the test and actions
 ' .. carry on here
 continue:
LOOP
```

```
toggleSlow:
    TOGGLE LED
    PAUSE  1000
    GOTO    continue

toggleFast:
    TOGGLE LED
    PAUSE   500
    GOTO    continue

END
```

INPUT

INPUT PinName

Parameters	
PinName	The pin **name** (e.g. PIN7) to set as an input

Summary

Sets a bi-directional pin to be an input.

See also OUTPUT & LET DIRS= Commands

Description

The INPUT command can only be used with KicChips that have bi-directional pins. e.g. Kic8

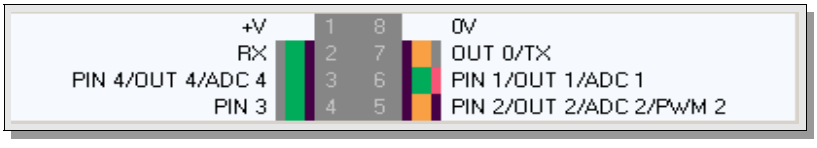
Do not confuse the INPUT command with setting a pin LOW.

Example

```
' KicChip "INPUT & OUTPUT" Example - Select Kic8

' Devices:
  SYMBOL BUTTON  = PIN1    ' connect button to pin1
  SYMBOL PRESSED =    0    ' ..make the button active low
  SYMBOL LED     =    2    ' connect LED to pin2

' Configure the pins for I/O
  INPUT  BUTTON            ' Make BUTTON an INPUT  pin
  OUTPUT LED               ' Make LED    an OUTPUT pin

DO
  IF BUTTON = PRESSED THEN    ' Test the input
    TOGGLE LED                ' toggle the output
    PAUSE 1000                ' pause for 1 second
  ENDIF
LOOP
END
```

Simulator

The Pin Window displays pin direction information , **Green** = Input , **Orange** = Output



LET

```
LET Result = Value
Result = Value Operator Value [Operator Value]
```

Parameters	
Result	Result is the name of a variable or pin
Value	A variable, constant, or mathematical expression
Operator	Any maths operator (See Maths Operators)

Summary

Assign a value to a variable or output port.

Description

The LET command allows a value to be assigned to a variable.

The value can be a simple number, variable, or a more complex mathematical expression.

Mathematical expressions are performed Left-to-Right and cannot include brackets.

Example 1

```
' KicChip "LET" Example 1
' Simulate this example in KicStudio, the value of B0 is
' incremented until it overflows to zero, whereupon KicStudio
' gives an alert message.

LET B0 = 0        ' Initialize b0
DO
  LET B0 = B0 + 1 ' Increment b0 (eventually see alert message)
LOOP
END
```

Example 2

```
' KicChip "LET" Example 2
' This example shows all the operators in action.
' Simulate the code in KicStudio and observe the values of B0 & W0
' in the watch window.
' Some operations cause alert messages to appear so that users can
' see when a value overflows its byte range.

'   addition
```

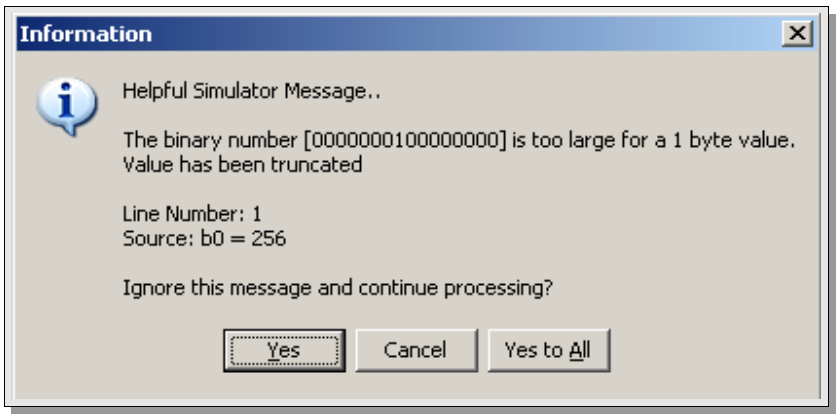
```
b0 = 0 + 1

'   subtraction
b0 = 5 - 2

'   multiply
b0 = 2 * 2

'   multiply high
w0 = $FFFF ** $FFFF

'   divide
b0 = 4 / 2
b0 = 1 / 0 '   divide by zero error

'   modulus
b0 = 5 // 4
b0 = 5 % 4

'   Max
b0 = 100 MAX 99

'   Min
b0 = 1 MIN 99

'   AND Bitwise
b0 = %11110000 AND %10101010
b0 = %11110000 &   %10101010

'   OR Bitwise
b0 = %11110000 OR %10101010
b0 = %11110000 |  %10101010

'   XOR Bitwise
b0 = %11110000 ^  %10101010

'   ANDNOT Bitwise
b0 = %11110000 &/  %10101010

'   ORNOT Bitwise (ignore message)
b0 = %11110000 |/  %10101010

'   XNOR (ignore message)
b0 = %11110000 ^/  %10101010

'   different value "types"
Let b0 = $00 + "A" - 65 + %11111111
```

```
END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

Out of range warnings are shown:



LET DIRS =

```
LET DIRS  = bFlags
LET DIRSC = bFlags
```

Parameters	
bFlags	A bit-mask. A bit set to 1 sets the pin as an output, a 0 sets the pin as an input. bFlags is generally expressed as a binary number. e.g. %10101010 so the flags can be seen.

Summary

Configure bi-directional pins as inputs or outputs

Description

The `LET DIRS=` and `LET DIRSC=` commands are used on KicChips that have bidirectional pins to set the direction for those pins.

Direction can be set as either an input (0) or an output (1) and is generally expressed as a binary "byte" where the bit on the left represents pin7 and the bit on the right represents pin0 of the port.

For instance, %11110000 would set pins 7 6 5 4 as outputs and pins 3 2 1 0 as inputs.

Pins default to inputs at power-up.

See Also INPUT & OUTPUT commands

Example LET DIRS=

```
' KicChip "LET DIRS=" Example - Select kic8
' This example configures the direction of the pins such
' that pin4 of the Kic8 is an output.

' Simulate this code and observe the color of Pin4 change
' to Orange to signify it is now an output

LET DIRS = %00010000 ' specify pin 4 as an output

DO
  TOGGLE    4    ' Toggle our specified output pin
  PAUSE  1000    ' pause for 1 second
LOOP
END
```

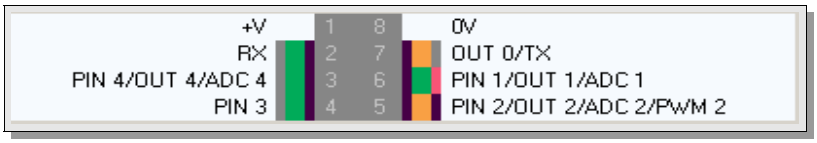
Example LET DIRSC=

```
' KicChip "LET DIRSC=" Example - Select kic28 OR kic40
' Simulate this code in KicStudio and observe PORTC pins
```

```
' Set the low 4 pins of PORTC to be outputs
  LET DIRSC = %00001111

' turn ON the lowest 4 pins of PORTC
  HIGH PORTC 0
  HIGH PORTC 1
  HIGH PORTC 2
  HIGH PORTC 3
END
```

Simulator

The Pin Window displays pin direction information , **Green** = Input , **Orange** = Output



LET PINS =

```
LET PINS  = bFlags
LET PINSC = bFlags
```

Parameters	
bFlags	A bit-mask. A bit set to 1 sets the pin high, a 0 sets the pin low. bFlags is generally expressed as a binary number. e.g. %10101010 so the flags can be seen.

Summary

Set or Clear all outputs on designated output port.

Description

The `LET PINS=` and `LET PINSC=` commands allow all outputs on a particular port to be changed simultaneously.

Output pins can be set as either HIGH or LOW and commonly expressed as a binary "byte" where the bit on the left represents out7 and the bit on the right represents out0 of the port.

For instance, %11110000 would set pins 7 6 5 4 high and pins 3 2 1 0 low.

See also HIGH , LOW , TOGGLE Commands

Example LET PINS=

```
' KicChip "LET PINS=" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3

DO
  LET PINS = %00001000    ' Turn ON output 3
  PAUSE 1000              ' wait 1 second
  LET PINS = %00000000    ' Turn OFF ALL PINS
  PAUSE 1000              ' wait 1 second
LOOP
END
```

Example LET PINSC=

```
' KicChip "LET PINSC=" Example - Select kic28 OR kic40
' Simulate this code in KicStudio and observe PORTC pins

' Set the low 4 pins of PORTC to be outputs
  LET DIRSC = %00001111

DO
  PINSC = %00001111 ' turn ON  the lowest 4 pins of PORTC
```

```
  PAUSE 1000         ' pause 1 second
  PINSC = %00000000 ' turn OFF the ALL      pins of PORTC
  PAUSE 1000         ' pause 1 second
LOOP
END
```

LOOKDOWN

LOOKDOWN bFind, (bData [,bData]), bResult

Parameters	
bFind	Variable or Constant BYTE value to search the list of data items for.
bData	A BYTE value, or list of BYTE values, to search along
bResult	A BYTE sized variable to store the POSITION of the bFind parameter in the bData list.

Summary

Finds the first position (index) of a the bFind parameter within the list of bData values, and stores this index in the bResult variable.

The list is zero based so the first data item is at position 0.

If the bFind item value cannot be located within the list of bData items then the bResult value is unchanged.

See also LOOKUP

Example

```
' KicChip "LOOKDOWN" Example
' Simulate this example in KicStudio,
' the output port toggles each pin in turn as each letter
' of the word "_HELLO" is located.

DO
  LOOKDOWN "H" , ("_HELLO") , PINS  ' =1
  LOOKDOWN "E" , ("_HELLO") , PINS  ' =2
  LOOKDOWN "L" , ("_HELLO") , PINS  ' =3
  LOOKDOWN "L" , ("_HELLO") , PINS  ' =3 (Again)
  LOOKDOWN "O" , ("_HELLO") , PINS  ' =5
LOOP
END
```

LOOKUP

LOOKUP bIndex,(bData [,bData]), bResult

Parameters	
bIndex	Variable or constant BYTE value that specifies a position in the list of bData items.
bData	A BYTE value, or list of BYTE values, to search along
bResult	A BYTE sized variable to store the Value of the located bData item

Summary

Extract a value from a list of values by referring to its position (index) in a list.

The list is zero based so the first item is at position 0.

If bIndex parameter exceeds the length of the list then the bResult value is unchanged.

See also LOOKDOWN

Example

```
' KicChip "LOOKUP" Example
' LED goes bright at 5 different levels
' Devices (based on KicStand):
  SYMBOL MOTOR   = 3      ' Motor connected to PWM3

  SYMBOL index = b0       ' pointer to next value
  SYMBOL value = b1       ' next value

DO
  FOR index = 0 TO 4

    ' Get the value pointed to by the index
    LOOKUP index , (2 , 20 , 100 , 75 , 50) , value

    ' Set the power of the motor using our value
    PWMOUT MOTOR , 140 , value
    PAUSE 1000

  NEXT
LOOP
END
```

LOW

LOW bPinNumber

Parameters	
bPinNumber	An output pin number 0..7

Summary

Sets the voltage of an output pin to a "low" (0v) state.

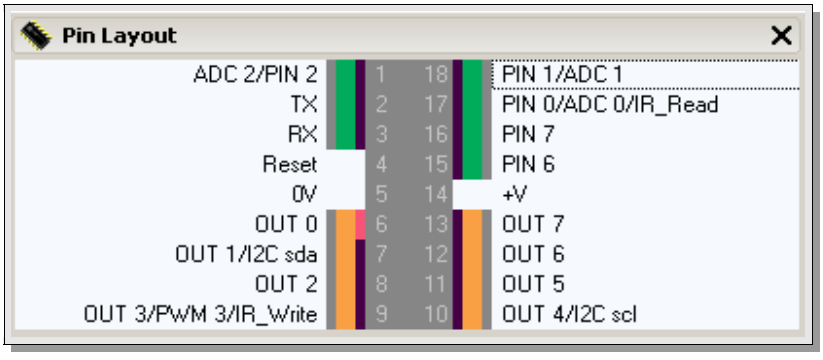
See also HIGH command

Example

```
' KicChip "HIGH" "LOW" Example
' simulate and observe
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3

DO
  HIGH     2       '   turn ON LED
  PAUSE 1000       '   wait 1 second
  LOW      2       '   turn OFF LED
  PAUSE 1000       '   wait 1 second
LOOP
END
```

Simulator

This command creates a virtual a virtual LED Device and shows changes to the Pin Window.



LOW PORTC

LOW PORTC bPinNumber

Parameters	
bPinNumber	A PORT-C output pin number 0..7

Summary

Sets the voltage of an output pin on PortC to a "low" (0v) state.

Note that DIRSC must be set to output for each pin that uses this command. Only the Kic28 and Kic40 use this command.

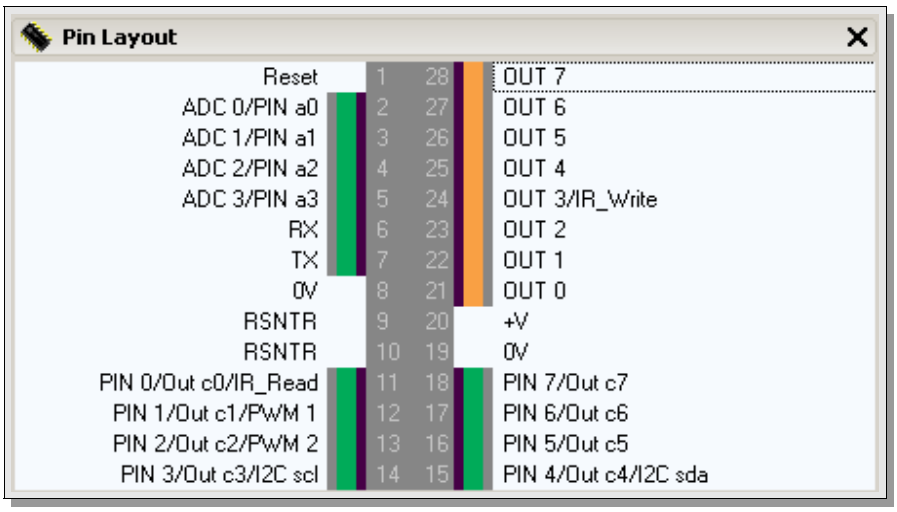
See also HIGH PORTC Command.

Example

```
' KicChip "HIGH & LOW PORTC" Example - Select kic18 or kic40

LET DIRSC = %11111111    ' set PortC to all outputs
DO
  HIGH  PORTC 0          ' set output 0 on PortC high
  PAUSE 1000             ' wait 1 second
  LOW   PORTC 0          ' set output 0 on PortC low
  PAUSE 1000             ' wait 1 second
LOOP
END
```

Simulator

This command creates a virtual a virtual LED Device and shows changes to the Pin Window.



MAX

LET Result = Value MAX MaxValue

Parameters	
Result	A variable name to store the result of the expression
Value	A variable, constant or expression
MaxValue	A variable or constant by which to limit the Value parameter

Summary

Limit the size of an expression.

Description

See also MIN command

Example

```
' KicChip "MAX" Example
' Simulate this example and notice that the maximum
' value ever assigned to variable b0 is the value 3

DO
 LET B0 = B0 + 1 MAX 3  ' increment b0 , to a maximum of 3
LOOP
END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

MIN

LET Result = Value MIN MinValue

Parameters	
Result	A variable name to store the result of the expression
Value	A variable, constant or expression
MinValue	A variable or constant by which to ensure the Result will have a minimum value

Summary

Ensure a minimum value is assigned to the result of an expression.

Description

See also MAX Command

Example

```
' KicChip "MIN" Example
' Simulate this example and notice that the minimum
' value ever assigned to variable b1 is the value 7

LET B0 = 0 ' initialize b0
LET B1 = 0 ' initialize b1

DO
 LET B0 = B0 + 1     ' Increment b0
 LET B1 = B0 MIN 7   ' Assign b0 to b1 but with a minimum of 7
LOOP
END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

MOD

LET Result = Numerator MOD Denominator

Parameters	
Result	A variable name to store the result of the expression
Numerator	A variable, constant or expression
Denominator	A variable, constant or expression

Summary

Mod returns the remainder of Numerator divided by Denominator.

So for example, the result of 7 MOD 3 is 1 because 3 divides into 7 twice (giving 6) remainder 1.

Example

```
' KicChip "MOD" Example

' The result of MOD is to give the remainder of a division.
' So   X = A MOD B
' Then X = remainder of (A/B)

DO
  B0 = 6 MOD 3  ' No remainder so b0 = 0
  B0 = 7 MOD 3  '                 b0 = 1
  B0 = 8 MOD 3  '                 b0 = 2
  B0 = 9 MOD 3  ' No remainder so b0 = 0
LOOP

END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

NAP

NAP Length

Parameters	
Length	A pre-set period of time (multiples of (2 Power Length) * 18 Milliseconds) - Values from 0..7

Length	Actual NAP time in milliseconds
0	18
1	36
2	72
3	144
4	288
5	576
6	1152
7	2304

Summary

Power down the processor for a short period of time.

Turns off PWMOUT & SERVO

See also PAUSE, SLEEP Commands

Description

Using a NAP command can save power when used instead of a PAUSE command since NAP and SLEEP actually put the processor into a low power mode.

NAP and SLEEP both use the same shut-down & start-up mechanism, with one major difference -NAP timing is subject to greater fluctuations due to external conditions such as temperature, supply voltage and manufacturing tolerances where the SLEEP command is more accurate because it is able to automatically compensate.

Example

```
' KicChip "NAP" Example
  SYMBOL SRV     = 0      ' Servo connected to output 0
  SYMBOL LED     = 1      ' Led is “Sourcing” power from output 1
  SYMBOL MOTOR   = 3      ' Motor connected to PWM3

PWMOUT MOTOR  , 140 , 50
SERVO  SRV    , 50
PAUSE 2000

DO
  TOGGLE LED   ' Toggle the LED
  NAP    7     ' power down
```

```
LOOP

END
```

ON GOSUB

ON bIndex GOSUB (Label [,labels])

Parameters	
bIndex	A byte variable who's value acts as the lookup index.
Label, Labels	List of label names

Summary

Use an index to lookup a label in a list of labels, and then perform a GOSUB to that label.

The list of label is zero based so the first label has an index value of 0.

If bIndex exceeds the number of labels available, then the command does nothing and processing simply continues from the following line.

Example

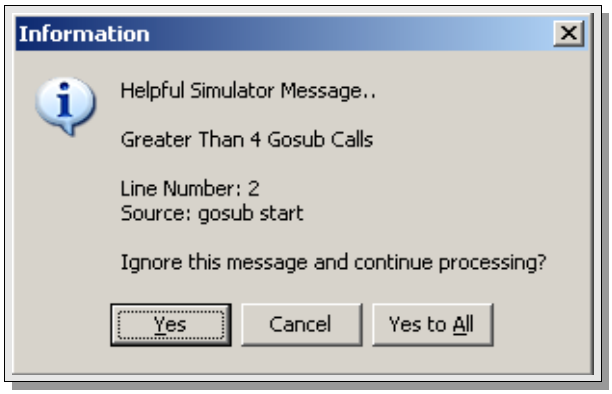
```
' KicChip "ON GOSUB" Example
' Kic18
' Note: The same thing an be achieved using the preferred "SELECT" command

DO
  FOR B0 = 0 TO 4
    ON B0 GOSUB (label1A, label1A, label1B, label1B, label1C, label1D)
  NEXT
LOOP

label1A:
  TOGGLE 0
  RETURN

label1B:
  TOGGLE 1
  RETURN

label1C:
  TOGGLE 2
  RETURN

label1D:
  TOGGLE 3
  RETURN
END
```

Simulator

The Label Window can help with this command.

The Call Stack Window can help with this command.

If the number of GOSUB calls exceeds the stack limit, the Stack Warning window is shown:



ON GOTO

ON bIndex GOTO (Label [,labels])

Parameters	
bIndex	A byte variable who's value acts as the lookup index.
Label, Labels	List of label names

Summary

Use an index to lookup a label in a list of labels, and then perform a GOTO to that label.

The list of label is zero based so the first label has an index value of 0.

If bIndex exceeds the number of labels available, then the command does nothing and processing simply continues from the following line.

Not recommended

See also ON GOSUB

Example

```
' KicChip "ON GOTO" Example
' Kic18
' Note: The same thing an be achieved using the preferred "SELECT" command
DO
  FOR B0 = 0 TO 4
    ON B0 GOTO (labelA, labelA, labelB, labelB, labelC, labelD)
    continue:
  NEXT
LOOP

labelA:
  TOGGLE 0
  GOTO continue

labelB:
  TOGGLE 1
  GOTO continue

labelC:
  TOGGLE 2
  GOTO continue

labelD:
  TOGGLE 3
```

```
GOTO continue
```

Simulator

The Label Window can help with this command.

OR (CONDITIONAL)

```
IF LogicExpA OR LogicExpB THEN
  ' Code block here
ENDIF
```

Arguments	
LogicExpA	A Boolean Logic Expression - See Logic Expressions.
LogicExpB	A Boolean Logic Expression - See Logic Expressions.

Summary

Performs logical OR on 2 expressions, the result is TRUE if either expression is TRUE.

Description

This command can be used to test for multiple conditions before executing an operation, it is often used in conjunction with the AND command.

See also AND Conditional Command

See also AND Bitwise Command

Example

```
' KicChip "OR (Conditional)" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.

  SYMBOL LogicB  = 1      ' Our "Test" data, change this for testing..

DO
  IF BUTTON = PRESSED OR LogicB = 1 THEN ' Button pressed OR data is 1
    TOGGLE LED                           ' Toggle LED
    PAUSE  1000                          ' Pause 1 second
  ENDIF
LOOP
END
```

OR (BITWISE)

```
LET Result = Value1 OR Value2
LET Result = Value1 |  Value2
```

Parameters	
Result	Byte or word sized variable or output port.
Value1	Byte or word sized value, variable, input port or constant.
Value2	Byte or word sized value, variable, input port or constant.
It is common practice to use binary-notation with bitwise commands e.g. %10101010. This makes it easier to identify individual BITS when debugging.	

Summary

Apply logical **A OR B** to the binary **BITS** of two integers.

The result is 1 - if either, or both, values are set.

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
1	**1**	**1**	**1**	**0**	0	**0**	0	Value1
1	**0**	**1**	**0**	**1**	0	**1**	0	Value2
1	**1**	**1**	**1**	**0**	0	**1**	0	Result

See also AND , ANDNOT, OR , XOR , XNOR , ORNOT

Example

```
' KicChip "OR Bitwise" Example


'   OR Bitwise
b0 = %11110000 OR %10101010
b0 = %11110000 |  %10101010


END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

ORNOT (BITWISE)

```
LET Result = Value1 ORNOT Value2
LET Result = Value1 |/    Value2
```

Parameters	
Result	Byte or word sized variable or output port.
Value1	Byte or word sized value, variable, input port or constant.
Value2	Byte or word sized value, variable, input port or constant.
It is common practice to use binary-notation with bitwise commands e.g. %10101010. This makes it easier to identify individual BITS when debugging.	

Summary

Apply logical **A OR (NOT B)** to the binary **BITS** of two integers.

The result is 1 if the first value is 1, or the second value is 0 (could be read as a >= b) .

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
1	**1**	**1**	**1**	0	**0**	0	**0**	Value1
1	**0**	**1**	**0**	1	**0**	1	**0**	Value2
1	**1**	**1**	**1**	0	**1**	0	**1**	Result

See also AND , ANDNOT, OR , XOR , XNOR , ORNOT

Example

```
' KicChip "ORNOT" Example


'   ORNOT Bitwise
b0 = %11110000 ORNOT  %10101010
b0 = %11110000 |/     %10101010


END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

OUTPUT

OUTPUT PinName

Parameters	
PinName	The pin **name** (e.g. PIN7) to set as an output

Summary

Sets a bi-directional pin to be an output.

See also INPUT. & LET DIRS= Commands

Description

The OUTPUT command can only be used with KicChips that have bi-directional pins. e.g. Kic8

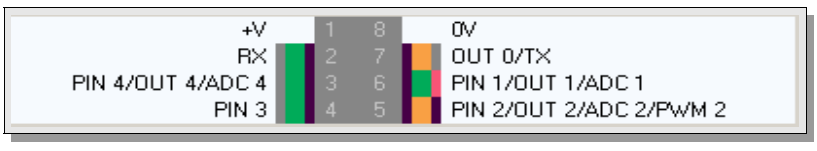
Do not confuse the OUTPUT command with setting a pin high.

Example

```
' KicChip "INPUT & OUTPUT" Example - Select Kic8

' Devices:
  SYMBOL BUTTON  = PIN1    ' connect button to pin1
  SYMBOL PRESSED =    0    ' ..make the button active low
  SYMBOL LED     =    2    ' connect LED to pin2

' Configure the pins for I/O
  INPUT  BUTTON            ' Make BUTTON an INPUT  pin
  OUTPUT LED               ' Make LED    an OUTPUT pin

DO
  IF BUTTON = PRESSED THEN    ' Test the input
    TOGGLE LED                ' toggle the output
    PAUSE  1000               ' pause for 1 second
  ENDIF
LOOP
END
```

Simulator

The Pin Window displays pin direction information , **Green** = Input , **Orange** = Output



PAUSE

PAUSE MilliSeconds

Parameters	
Milliseconds	A value or constant representing the duration in milliseconds to pause execution

Summary

Pause for a specified amount of time in milliseconds.

Description

Use of the PAUSE command puts the program into a simple delay loop for a specified amount of time.

The longest period of time allowed for a single PAUSE is just over 65 seconds (65535ms).

PAUSE ties up the processor and cannot react to normal input during the delay.

Using PAUSE will maintain the status of all outputs for the entire delay cycle.

PAUSE does not set the processor in low power mode, use NAP or SLEEP for low power usage.

See also NAP, SLEEP Commands

Example

```
' KicChip "PAUSE" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is “Sourcing” power from output 3

DO
  TOGGLE LED    ' Toggle LED
  PAUSE  1000   ' Pause 1 second
LOOP

END
```

PEEK

PEEK bAddress, bResult

Parameters	
bAddress	A byte value to indicate an address in RAM to read from
bResult	A byte variable in which to store the read result

Summary

Read a value from RAM at address bAddress into result variable bResult.

Description

The PEEK command is used to retrieve data stored in the KicChip RAM memory into a general purpose variable (b0 to b13) where it can be used to perform calculations.

The size of RAM memory varies between devices.

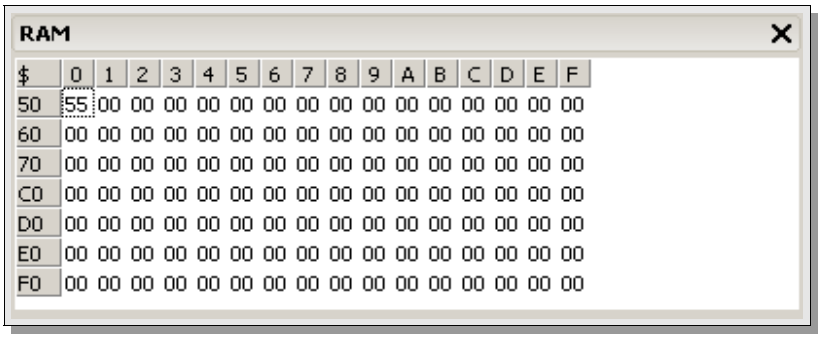
See also POKE command.

Example

```
' KicChip "PEEK & POKE" Example


LET B1 = %01010101  ' set a debug value for b1


POKE $50 , B1      ' store b1 in location $50
PEEK $50 , pins    ' read location $50 into output pins


END
```

Simulator

The RAM window displays the contents of the virtual RAM locations during simulation.



POKE

POKE bAddress, bData

Parameters	
bAddress	Value to indicate the location in RAM to write to.
bData	A BYTE sized variable or constant to store in RAM

Summary

Writes the bData value into the KicChip RAM at the location bAddress.

Description

The POKE command is used to store data directly to the processors RAM memory which frees up the general purpose variables (b0 to b13) for other calculations.

This command is also useful for resetting flag registers and other processor registers.

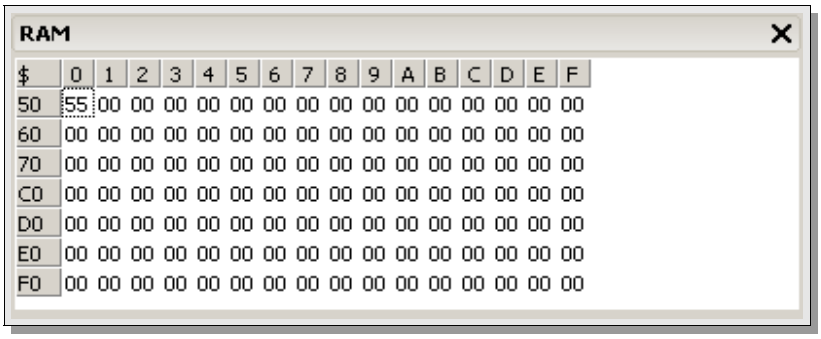
See also PEEK command

Example

```
' KicChip "PEEK & POKE" Example


LET B1 = %01010101  ' set a debug value for b1


POKE $50 , B1      ' store b1 in location $50
PEEK $50 , pins    ' read location $50 into output pins


END
```

Simulator

The RAM window displays the contents of the virtual RAM locations during simulation.



PULSIN

PULSIN bPinNumber, Edge, wResult

Parameters	
bPinNumber	The pin that is being read for a pulse
Edge	A value (**1** or **0**) to indicate if measuring a pulse from a low-to-high or high-to-low transition.
wResult	A WORD sized variable in which to store the length of the pulse

Summary

Measures the duration of an input pulse.

Description

A PULSIN command is used to measure the length of a pulse. If no pulse occurs within the specified duration, the result will be 0.

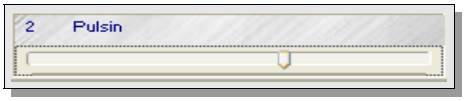
See also PULSOUT

Example

```
DO
  PULSIN 2, 1, w0   ' measure pulse on input 2
  DEBUG             ' send result to KicStudio screen
LOOP
```

Simulator

A slider is created to simulate this command.



PULSOUT

PULSOUT bPinNumber, Duration

Parameters	
bPinNumber	An output pin to send the pulse out on
Duration	The length of the pulse to send in Milliseconds.

Summary

Sends a pulse out on an output pin for a specified duration.

Description

A PULSOUT command inverts the state of the specified output pin; waits for the specified Duration; then inverts the state of the pin again; returning the bit to its original state.

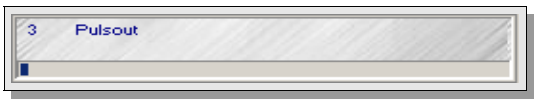
See also PULSIN

Example

```
' KicChip "PULSOUT" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is “Sourcing” power from output 3


DO
  PULSOUT 3 , 1000  ' pulse for 1 second
  PAUSE 1000        ' wait  for 1 second
LOOP
END
```

Simulator

A gauge id displayed to simulate the length of the pulse.



PWMOUT

PWMOUT PwmPin, bPeriod, DutyCycle

Parameters	
PwmPin	A valid PWM Channel
bPeriod	The PWM period value 0 to 255
DutyCycle	The PWM duty cycle value or *Mark Time* 0..1023

Summary

Provides an intermediate amount of electrical power between fully on and fully off.

Description

The PWMOUT command (Pulse Width Modulation output) allows the KicChip to generate an analog like voltage via a digital pulse stream.

PWMOUT runs continually in the background, and continues running until a Duty of 0 is sent or an END command is executed.

PWMOUT stops temporarily during NAP commands.

The calculation to determine what the average voltage will be on a PWM pin is as follows: Average Voltage = (Duty / 255) * 5v

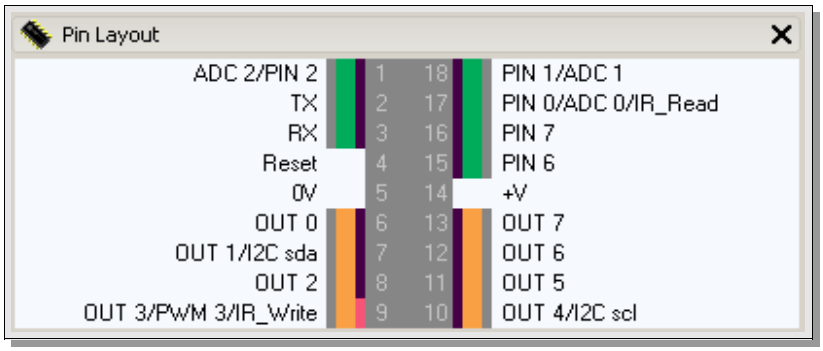
Example

```
' KicChip  "PWMOUT" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.
  SYMBOL MOTOR   = 3      ' Motor connected to PWM3
' Motor Control Values
  SYMBOL motorSlow =  2
  SYMBOL motorFast = 50


DO

 ' Increase power to motor
 FOR b0 = motorSlow to motorFast
  GOSUB PULSE
 NEXT

 ' Decrease power to motor
 FOR b0 = motorFast to motorSlow step -1
```

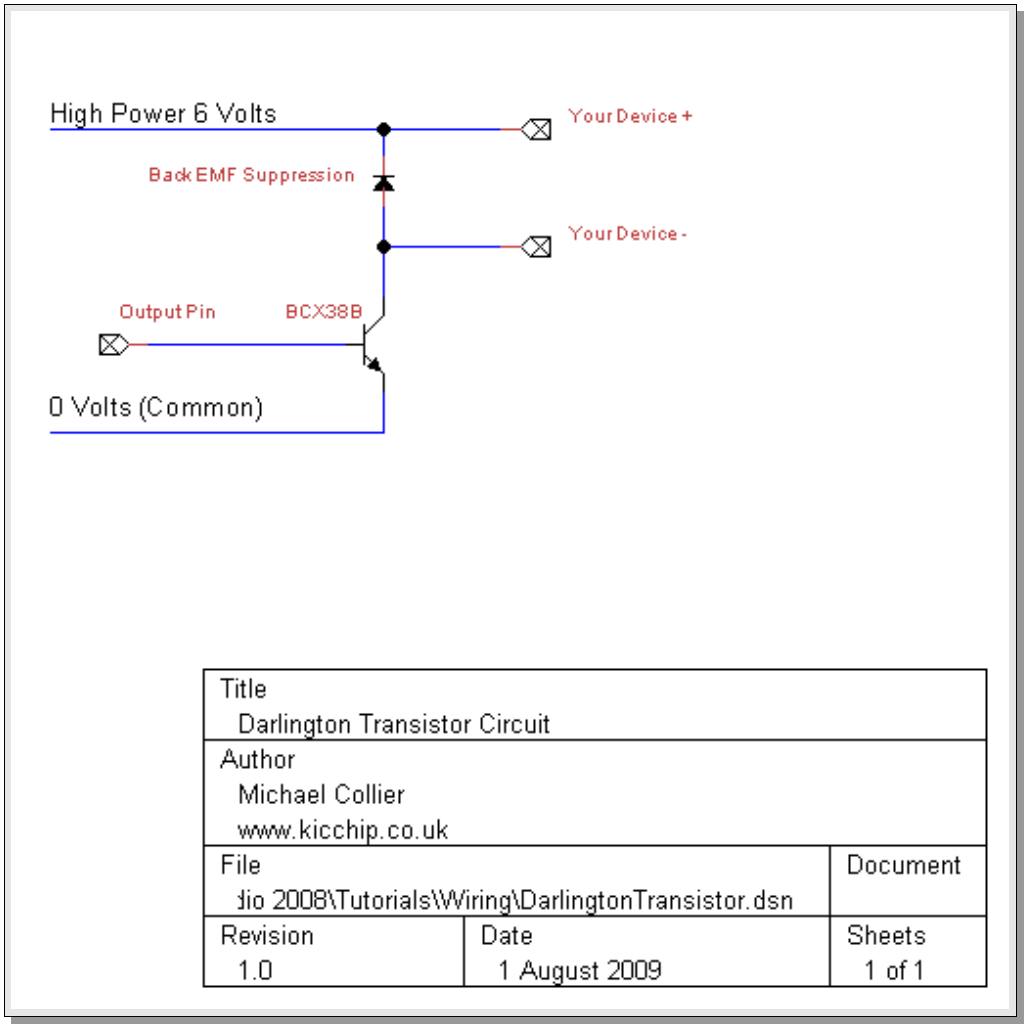
```
  GOSUB PULSE
 NEXT

LOOP

PULSE:
   PWMOUT MOTOR, 140, b0    ' Pulse motor with value in b0
   PAUSE 10
   RETURN

END
```

Simulator

Labelled "PWM" followed by channel number e.g. "PWM 3", goes red when PWMOUT applied to pin and black when PWMOUT turned off.



Circuit Diagram



Title Darlington Transistor Circuit		
Author Michael Collier www.kicchip.co.uk		
File dio 2008\Tutorials\Wiring\DarlingtonTransistor.dsn		Document
Revision 1.0	Date 1 August 2009	Sheets 1 of 1

READ

READ bAddress, bResult

Parameters	
bAddress	A byte value address to read EEPROM from.
bResult	A variable to store the result of the read.

Summary

Read a byte of data from EEPROM memory at location bAddress into into the bResult variable.

Description

The READ command allows the program to retrieve data from the processors

EEPROM. This memory is maintained even when power is removed.

EEPROM data is not cleared between debug sessions as per actual KicChip.

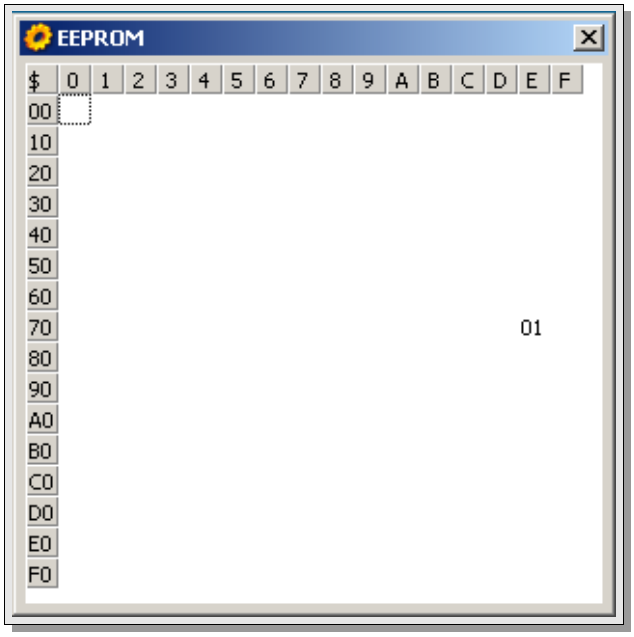
See also WRITE Command.

Example

```
' KicChip "READ/WRITE" example


FOR B0 = 0 TO 7
  WRITE 126 , B0      ' write the value of b0 into location 126
  READ  126 , PINS    ' read location 126 into the output pins
  PAUSE 1000          ' pause 1 second
NEXT
END
```

Simulator

The EEPROM Window displays EEPROM data.



RETURN

RETURN

Summary

Return from a subroutine.

Description

The RETURN command is only used to return from a GOSUB command which sets the program flow back to the line just after the GOSUB command.

The RETURN command should never be used outside of a GOSUB subroutine or the program will crash - the simulator gives a warning message if a RETURN is attempted without having performed a corresponding GOSUB command.

See also GOSUB & ON GOSUB Commands

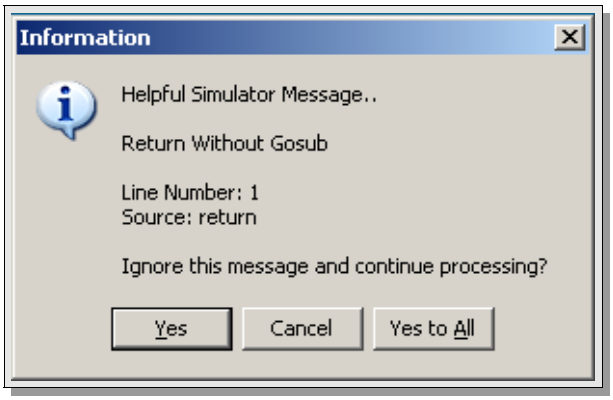
Example

```
' KicChip "GOSUB RETURN" Example
' If button is pressed then subroutine to toggle fast is called
' else the subroutine to toggle slowly is called
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is "Sourcing" power from output 3
  SYMBOL BUTTON  = PIN7   ' Button is connected to input 7 , "active low"
  SYMBOL PRESSED = 0      ' "Pull-ups"  used, so button is active low.

DO
  IF BUTTON = PRESSED THEN
    GOSUB toggleFast        ' Execute our toggle fast subroutine
  ELSE
    GOSUB toggleSlow        ' Execute our toggle fast subroutine
  ENDIF
LOOP

toggleSlow:
  TOGGLE LED     ' Toggle the LED
  PAUSE  1000    ' Pause for 1 second
  RETURN

toggleFast:
  TOGGLE LED     ' Toggle the LED
  PAUSE  500     ' Pause for half a second
  RETURN

END
```

Simulator

The Call Stack Window can help with this command.

Stack underflow causes the warning screen to appear :



SELECT CASE

```
SELECT Variable
  CASE Value
    ' Code block here
  CASE Value [, Values]
    ' Code block here
  CASE Value TO Value
    ' Code block here
  CASE Operator Value
    ' Code block here
  ELSE
    ' Code block here
ENDSELECT
```

Parameters	
Variable	The variable value to be compared within each CASE statement
Value	A value, or set of values.
Operator	A selection of the boolean operators: = Equal <> Not Equal > Greater Than < Less Than >= Greater Than or Equal To <= Less Than or Equal To

Summary

Execute sections of code determined by the results of a variable compare.

Description

SELECT CASE / CASE / ELSE / ENDSELECT is a decision making structure.

The SELECT… CASE command structure is good for performing one of several possible actions determined by the value of a single expression.

When a SELECT… CASE command is executed it will evaluate the expression and compare it to the specified conditions of each CASE.

If a CASE compare is true, the code within that CASE is executed and then program flow is continued from the ENDSELECT line.

If no CASE is true, the ELSE code is executed.

Example

```
' KicChip "SELECT CASE" Example

DO
  FOR b0 = 0 TO 4
    SELECT b0
      CASE 0, 1                ' if b0 = 0 or 1
        GOSUB labelA
      CASE 2 TO 3              ' if b0 = 2 or 3
        GOSUB labelB
      CASE >= 4                ' if b0 >= 4
        GOSUB labelC
      ELSE                     ' else any other value
        GOSUB labelD
    ENDSELECT
  NEXT
LOOP

labelA:
  TOGGLE 0
  RETURN

labelB:
  TOGGLE 1
  RETURN

labelC:
  TOGGLE 2
  RETURN

labelD:
  TOGGLE 3
  RETURN
END
```

SERIN

```
SERIN bPin, Protocol, bResult
SERIN bPin, Protocol, bResult [,bResults]
SERIN bPin, Protocol, (bPrefix)
SERIN bPin, Protocol, (bPrefix), bResult
```

Parameters	
bPin	An input pin number (**0**..**7**) or **8** if using the program cable
Protocol	Defines the protocol used for transmission See Serial Protocol Table
bResult	Variable(s) in which to store the incoming data
bPrefix	A byte value to "wait for", prior to saving the incoming data

Protocol	Description
N300	Inverted 300 Baud
N600	Inverted 600 Baud
N1200	Inverted 1200 Baud
N2400	Inverted 2400 Baud
N4800	Inverted 4800 Baud
N9600	Inverted 9600 Baud
T300	True 300 Baud
T600	True 600 Baud
T1200	True 1200 Baud
T2400	True 2400 Baud
T4800	True 4800 Baud
T9600	True 9600 Baud

Summary

Read Serial-Data in True or Inverted mode using any input pin.

Description

The SERIN command is used to receive asynchronous serial data on an input pin of the processor and store the data in variables to be manipulated by the program.

Asynchronous means that serial communications can be done without the need for a separate clock signal.

Data can be received using as little as two wires; one for data and one for ground.

Asynchronous serial communication relies on precise timing. Both the sender and receiver must be set for identical timing. This is done via the protocol which sets the baud rate. N and T directives allow for inverted or true communications, N = inverted, T = true.

SERIN works with the download cable when using pin number 8.

All processing stops until the serial data byte is received.

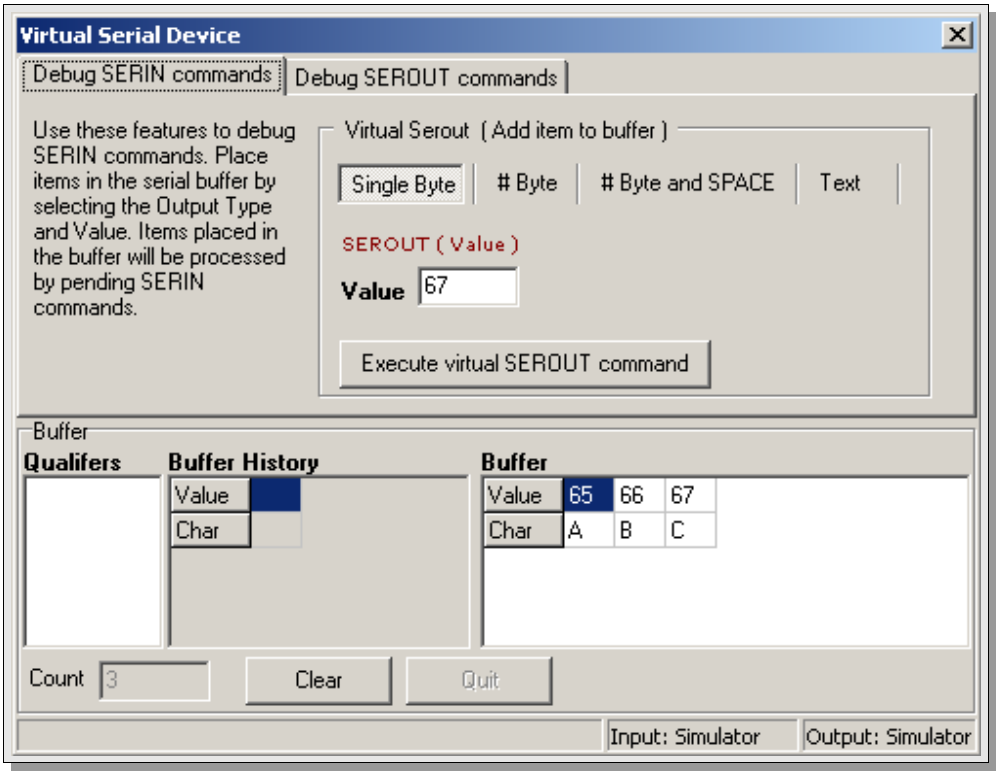
See also SEROUT

Example

```
' KicChip "SERIN" example1
' Devices:
  SYMBOL SERIAL_IN = 8     ' Use programming cable.
  SYMBOL WAIT_FOR  = 1     ' Wait for a 1 to be received first.
DO
  ' Read data from programming cable, wait for a 1 to
  ' be received before saving data in variable B0  , B1
  SERIN SERIAL_IN , N4800 , (WAIT_FOR) , B0 , B1
LOOP
END
```

Simulator

The virtual serial window allows you to push and pull data from the virtual serial stream.



SEROUT

SEROUT bPin,Protocol,(bData [,bData])

Parameters	
bPin	An output pin number (**0**..**7**) or **8** if using the program cable
Protocol	Defines the protocol used for transmission See Serial Protocol Table
bData	A byte value to send out

Protocol	Description
N300	Inverted 300 Baud
N600	Inverted 600 Baud
N1200	Inverted 1200 Baud
N2400	Inverted 2400 Baud
N4800	Inverted 4800 Baud
N9600	Inverted 9600 Baud
T300	True 300 Baud
T600	True 600 Baud
T1200	True 1200 Baud
T2400	True 2400 Baud
T4800	True 4800 Baud
T9600	True 9600 Baud

Summary

Sends out serial data on an output pin.

Description

The SEROUT command is used to send asynchronous serial data stored in variables out on an output pin of the processor. The download cable can be used if 8 is used as pin number.

Asynchronous means that serial communications can be done without the need for a separate clock signal. Data can be sent using as little as two wires; one for data and one for ground.

Asynchronous serial communication relies on precise timing. Both the sender and receiver must be set for identical timing. This is done via the protocol which sets the baud rate. N and T directives allow for inverted or true communications, N = inverted, T = true.

All processing stops until the serial data byte is received.

See also SERIN.

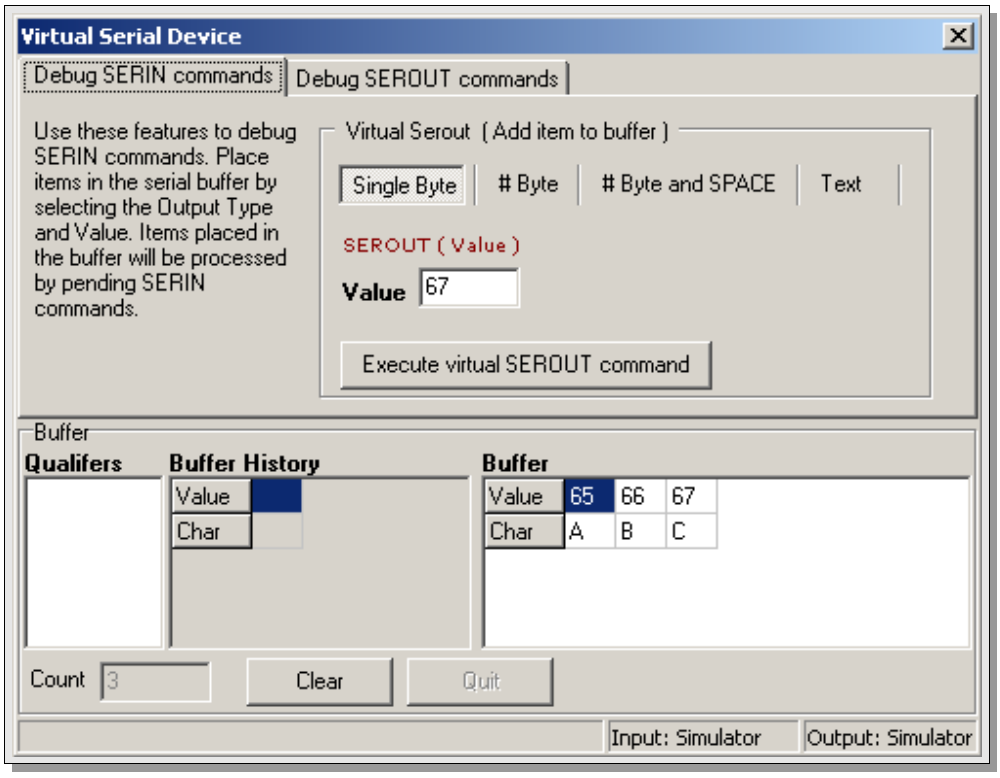
Example

```
' KicChip "SEROUT" example1
' Devices:
```

```
  SYMBOL SERIAL_OUT = 8    ' Use programming cable.
  SYMBOL PREFIX     = 1    ' Send a prefix of 1 with data
DO
  FOR B0 = 0 TO 10

    ' Send prefix and send data via cable
    SEROUT SERIAL_OUT , N4800 , (1 , B0) ' Send data
    PAUSE 50                             ' Pause in case the receiver needs
                                         ' time to process any data
  NEXT
LOOP
END
```

Simulator

The virtual serial window can be used to push and pull data from the serial stream.



SERVO

SERVO bPinNumber, bPercent

Parameters	
bPinNumber	An output pin number 0 to 7
bPercent	Servo position 1..100 Use 0 to turn off servo pulses completely.

Summary

Send control signal to a Servo device.

Description

The SERVO command sends a pulse every 20ms on a specified output pin.

The length of ON time versus OFF time determines the position the servo will move to. A bPercent value of 50 will move the servo to an approximate center position.

A bPercent of 1 will set the position to one extreme and a bPercent of 100 will set the position to the other extreme.

Any value above or below 1 to 100 will turn off the pin and the servo will no longer hold its position.

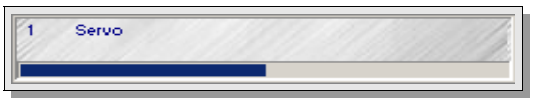
Using this command has NO impact on the speed or processing of other commands since the servo timing takes place in the background.

Example

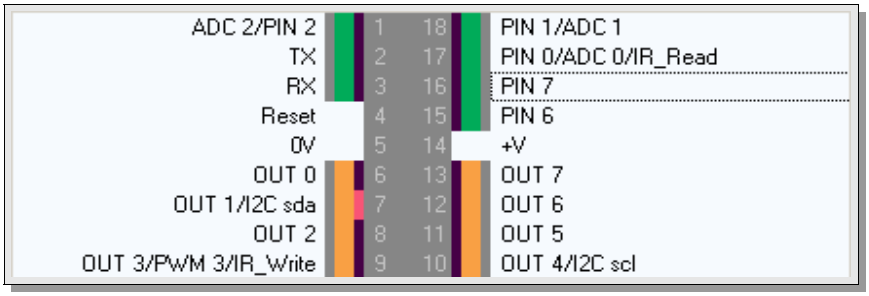
```
' KicChip "SERVO" Example
' Devices:
  SYMBOL MYSERVO     = 3      ' Connect Servo to output 3

DO

  SERVO MYSERVO ,   1    ' set servo to minimum position
  PAUSE 1000             ' Pause 1 second

  SERVO MYSERVO , 100    ' set servo to maximum position
  PAUSE 1000             ' Pause 1 second

  SERVO MYSERVO ,   0    ' set servo to OFF
  PAUSE 1000             ' Pause 1 second

LOOP

END
```

Simulator

Servo Window



Pin window indicates power has been sent via servo pin.



SLEEP

SLEEP Duration

Parameters	
Duration	A value from 0..65535 that specifies how many of multiples of 2.3 seconds to sleep for.

Summary

Sleep for some period of time.

Description

The SLEEP command is used when low power mode is desired for a period of time.

During this period of time, all timers are paused so the PWMOUT and SERVO commands will cease to function.

The length of a SLEEP period can range from 2.3 seconds to just over 18 hours.

SLEEP 1 = 2.3 seconds

SLEEP 10 = 23 seconds

Pins retain their input/output direction but output pins are interrupted every 2.3 seconds due to the design of the processor.

Example

```
' KicChip "SLEEP" Example
  SYMBOL SRV     = 0      ' Servo connected to output 0
  SYMBOL LED     = 1      ' Led is "Sourcing" power from output 1
  SYMBOL MOTOR   = 3      ' Motor connected to PWM3

PWMOUT MOTOR  , 140 , 50
SERVO  SRV    , 50
PAUSE 2000

DO
  TOGGLE LED   ' Toggle the LED
  SLEEP 1      ' power down 2.3 seconds
LOOP
END
```

SONYREAD

SONYREAD

Summary

Read a remote infra-red signal from a Sony (or compatible) remote into variable B13 (A.K.A. INFRA)

Description

The SONYREAD command waits for about 1 second for an infra-red signal transmitted from a Sony remote, another KicChip or any IR transmitter using the Sony reference code.

This command assumes there is an IR demodulator connected to the IR_READ pin.

If a valid 7-bit signal is not received within 1 second then the result will be 255.

Result data is placed in variable b13 (defined as INFRA for convenience).

Variable b12 contains the unique Sony device ID.

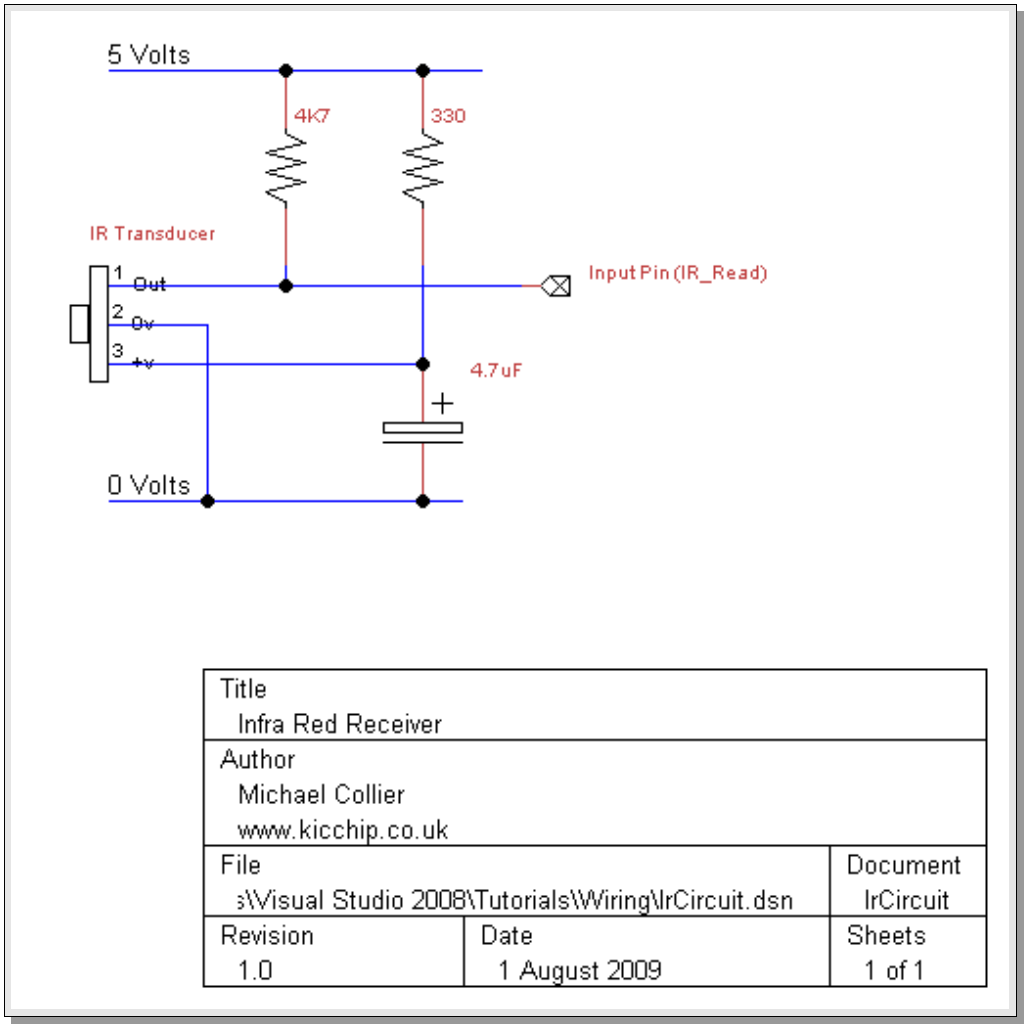
Uses pin label “IR_Read”

Se also SONYWRITE.

Example

```
' KicChip "SONYREAD" Example
' Use this with the "SONYWRITE" Example and another KicChip
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is “Sourcing” power from output 3


DO
  SONYREAD               ' read a value from IR transmitter
  IF INFRA <> 255 THEN   ' check if IR value is valid i.e. not 255
    TOGGLE LED           ' toggle an LED
    PAUSE  1000          ' Pause 1 second
  ENDIF
LOOP
END
```

Circuit Diagram



SONYWRITE

SONYWRITE bReceiverID, bData

Parameters	
bReceiverID	The ID being used by the receiving device. **For another KicChip, this MUST be 1.**
bData	The byte value to transmit

Summary

Send infra-red data (7 bits) to remote Infra-red receiver.

Uses pin label “IR_Write”

Description

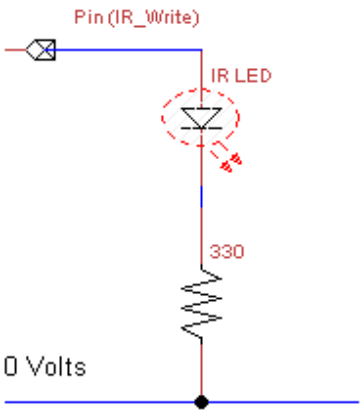
The SONYWRITE command is used to transmit 7 bits of data to another KicChip configured to receive the data - or any device designed to recognize the Sony data.

See also SONYREAD.

Example

```
' KicChip "SONYWRITE" Example
' Use this with the "SONYREAD" Example and another KicChip
' Devices (based on KicStand):
  SYMBOL BUTTON    = PIN7    ' Button is connected to input 7 , “active low”
  SYMBOL PRESSED   = 0       ' “Pull-ups”  used, so button is active low.
  SYMBOL RECEIVER  = 1       ' Receiver ID, to transmit to KicChip use 1
DO
  IF BUTTON = PRESSED THEN     ' check if transmit button is pressed
    SONYWRITE RECEIVER , 127   ' Transmit a data value of 127
  ENDIF
LOOP
END
```

Circuit Diagram



Title Infra Red Transmitter		
Author Michael Collier www.kicchip.co.uk		
File sual Studio 2008\Tutorials\Wiring\irLedCircuit.dsn		Document
Revision 1.0	Date 1 August 2009	Sheets 1 of 1

SOUND

SOUND bPinNumber,(bFrequency, bDuration)

Parameters	
bPinNumber	An output pin number 0..7 (connected to a speaker).
bFrequency	The frequency of the note 0..255
bDuration	The length of time to play the note

Summary

Sends a tone (square wave) to an output pin bPinNumber connected to a speaker.

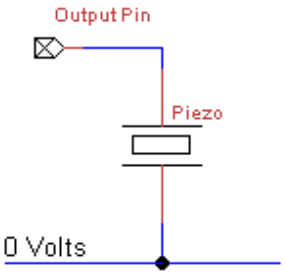
The frequency of the tone is specified by the bFrequency parameter.

The length of the tone is specified by the bDuration parameter.

Example

```
' KicChip "SOUND" Example
' Devices:
  SYMBOL SPEAKER = 3      ' Connect speaker to output 3
  SYMBOL LENGTH  = 4      ' Length of tone
DO
  FOR b0 = 90 TO 110               ' Use a loop to increase the tone
    SOUND SPEAKER , ( b0, LENGTH ) ' send a tone to the speaker
  NEXT
LOOP
END
```

Circuit Diagram



Title Piezo Speaker		
Author Michael Collier www.kicchip.co.uk		
File ual Studio 2008\Tutorials\Wiring\PiezoCircuit.dsn		Document
Revision 1.0	Date 1 August 2009	Sheets 1 of 1

STOP

STOP

Summary

Permanently stops program flow until power has been cycled.

Description

A STOP command puts the processor into a never ending loop at the end of a program.

The STOP command does not set the processor into low power mode and all timers and outputs continue running.

Example

```
' KicChip "STOP" Example
' Devices (based on KicStand):
  SYMBOL MOTOR   = 3      ' Motor connected to PWM3



   PWMOUT MOTOR, 140, 50   ' Turn on PWM
   SERVO  0 , 50           ' turn on servo
   HIGH   1                ' turn on LED
   PAUSE 3000              ' wait 3 seconds
   STOP                    ' turn of PWM
END
```

TOGGLE

TOGGLE bPinNumber

Parameters	
bPinNumber	An output pin number 0..7

Summary

Inverts the state of an output pin, i.e. if the pin is HIGH, it turns LOW.

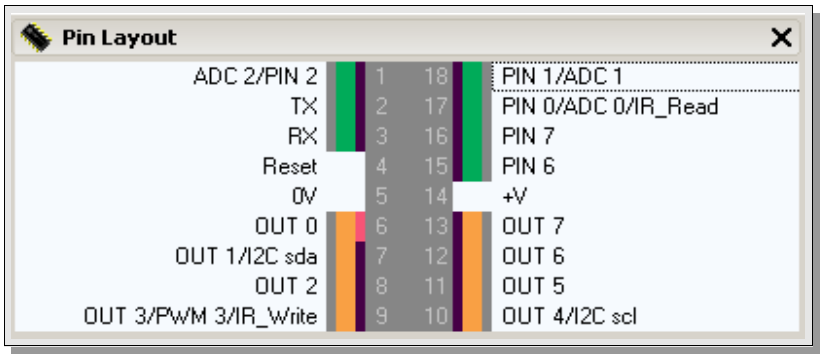
See also HIGH & LOW Commands.

Example

```
' KicChip "TOGGLE" Example
' Devices (based on KicStand):
  SYMBOL LED     = 3      ' Led is “Sourcing” power from output 3

DO
  TOGGLE LED      '   Toggle LED
  PAUSE  1000     '   wait 1 second
LOOP
END
```

Simulator

This command creates a virtual a virtual LED Device and shows changes to the Pin Window.



WRITE

WRITE bAddress, bData

Parameters	
bAddress	EEPROM Location to write to
bData	Data to store in EEPROM

Summary

Write a byte of data to EEPROM memory.

EEPROM data is not cleared between debug sessions as per actual KicChip.

See also READ Command.

Description

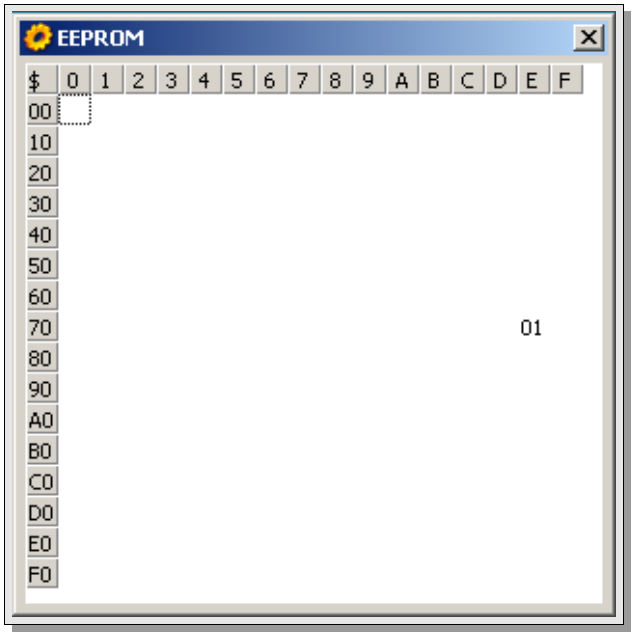
The WRITE command allows the program to store data into the processors EEPROM memory. This memory is maintained even when power is removed. **Warning: Limited number of writes 100,000**

Example

```
' KicChip "READ/WRITE" example


FOR B0 = 0 TO 7
  WRITE 126 , B0      ' write the value of b0 into location 126
  READ  126 , PINS    ' read location 126 into the output pins
  PAUSE 1000          ' pause 1 second
NEXT
END
```

Simulator

The EEPROM Window displays EEPROM data.



XNOR (BITWISE)

```
LET Result = Value1 XNOR Value2
LET Result = Value1 ^/   Value2
```

Parameters	
Result	Byte or word sized variable or output port.
Value1	Byte or word sized value, variable, input port or constant.
Value2	Byte or word sized value, variable, input port or constant.
It is common practice to use binary-notation with bitwise commands e.g. %10101010. This makes it easier to identify individual BITS when debugging.	

Summary

Apply logical **NOT (A XOR B)** to the binary **BITS** of two integers.

The result is 1 - only if both bit values are the same (could be read as a = b)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
1	1	**1**	1	0	**0**	0	**0**	Value1
1	0	**1**	0	1	**0**	1	**0**	Value2
1	0	**1**	0	0	**1**	0	**1**	Result

See also AND , ANDNOT, OR , XOR , XNOR , ORNOT

Example

```
' KicChip "XNOR" Example

'   XNOR (ignore message)
b0 = %11110000 XNOR  %10101010
b0 = %11110000 ^/    %10101010

END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

XOR (BITWISE)

```
LET Result = Value1 XOR Value2
LET Result = Value1 ^   Value2
```

Parameters	
Result	Byte or word sized variable or output port.
Value1	Byte or word sized value, variable, input port or constant.
Value2	Byte or word sized value, variable, input port or constant.
It is common practice to use binary-notation with bitwise commands e.g. %10101010. This makes it easier to identify individual BITS when debugging.	

Summary

Apply logical **A XOR B** to the binary **BITS** of two integers.

The result is 1 - only if either bit is set, but not both (could be read as a <> b)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
1	**1**	1	**1**	**0**	0	**0**	0	Value1
1	**0**	1	**0**	**1**	0	**1**	0	Value2
0	**1**	0	**1**	**1**	0	**1**	0	Result

See also AND , ANDNOT, OR , XOR , XNOR , ORNOT

Description

XOR can be used to detect a CHANGES in binary data.

Look at the table above, the result is a 1 where value1 and value2 are different.

Example

```
' KicChip "XOR" Example

'   XOR Bitwise
b0 = %11110000 XOR  %10101010
b0 = %11110000 ^    %10101010

END
```

Simulator

The Watch Window can be used to observe and modify the current value of variables during program simulation.

DEBUG WINDOWS

Summary

Several Debug Windows are available to assist in the simulation & debugging process.

The Pin Window

The Pin Window displays current pin information.

See HIGH LOW TOGGLE INPUT OUTPUT

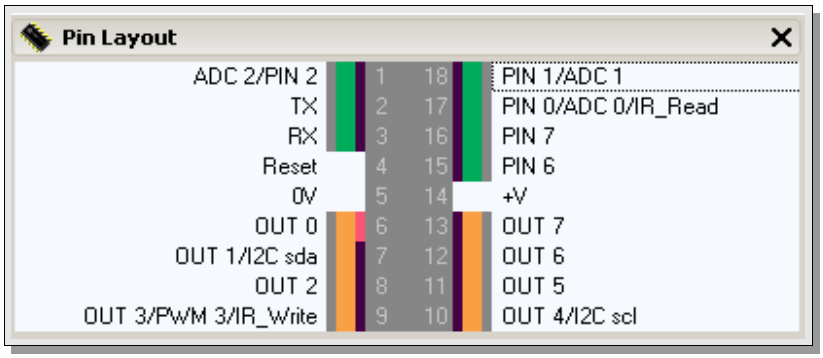
Leg Number

Output Status (Red = High , Black = Low)

Direction (Orange = Output , Green = Input)

Input Status (Red = High , Black = Low)

Alternate Uses (Analog, Infra Red, Two Wire etc.)

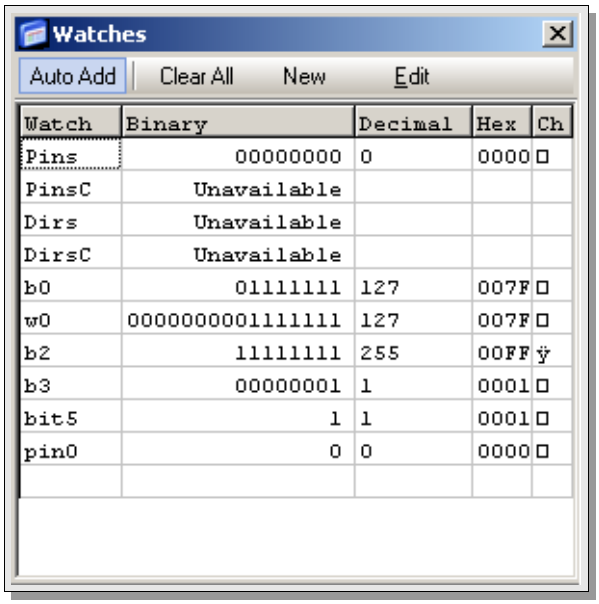


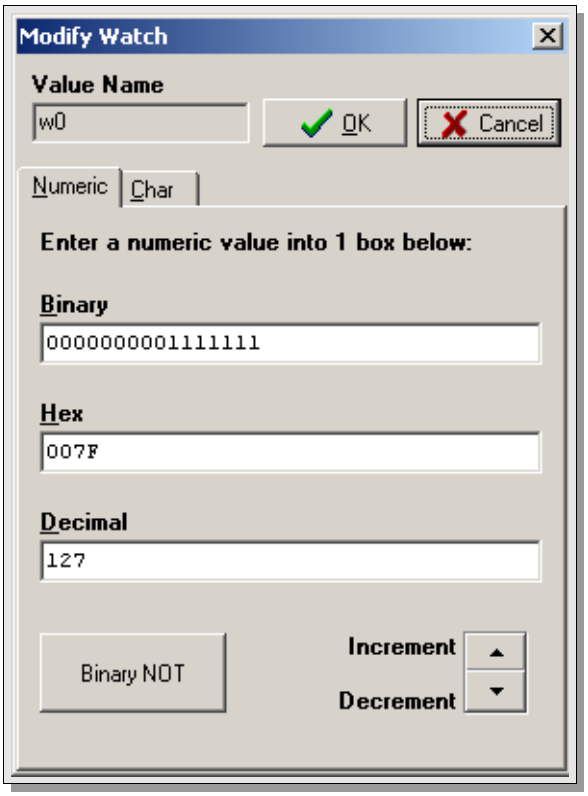
The Pin window is "Connected" to the virtual devices:

The Watch Window

View and edit variable values.

See LET command





The Call Stack Window

See GOSUB & RETURN