PropBASIC 00.00.25 Nov 9, 2009

Operators

Unary Operators:

ABS	Returns the absolute value
SGN	Returns the sign of value 1, 0, -1

Binary Operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
//	Remainder
*/	Multiply, shift 16-bits
**	Multiply, shift 32-bits
& AND	Bitwise AND
| OR	Bitwise OR
^ XOR	Bitwise XOR
&~ ANDN	Bitwise AND NOT
MIN	Minimum of two values
MAX	Maximum of two values
>> SHR	Shift right
<< SHL	Shift left

DEVICE Sets device type and parameters.

```
DEVICE deviceID, {settings{,settings}}
DEVICE P8X32A, XTAL1, PLL16X
  deviceID: only P8X32A is supported
  settings: RCSLOW, RCFAST, XINPUT, XTAL1..3, PLLX2, PLLX4, PLLX8, PLLX16
```

FREQ Sets device frequency after pll multiplier.

```
FREQ freq
FREQ 80_000_000
  * Do not use FREQ and XIN together, use one or the other
```

XIN Crystal frequency before pll multiplier

```
XIN freq
XIN 5_000_000
  * Do not use FREQ and XIN together, use one or the other
```

DATA,WDATA,LDATA Creates data values in HUB ram. DATA = BYTE, WDATA=WORD, LDATA=LONG

```
[label] DATA value1[,value2[,value3[,etc]]]
BitMask DATA 1,2,4,8,16
* Data labels MUST be on the same line as the DATA command.
```

HUB Creates HUB variables. Access via GETADDR, RDBYTE, RDWORD, RDLONG, WRBYTE, WRWORD, WRLONG

```
name HUB type [= value]
name HUB type(elements) [= value]
myVar HUB LONG = 100_000
myVars HUB LONG(8) = 0
  type: BYTE, WORD, LONG
  * Use RDBYTE, RDWORD, RDLONG to read value from HUB variables.
  * Use WRBYTE, WRWORD, RDLONG to write value to HUB variables.
```

CON Creates a named constant

```
name CON value
MyCon CON 1000
```

VAR Creates a variable. Only LONG types are supported. Arrays are supported.

name VAR LONG
name VAR LONG(*elements*)
myVar VAR LONG
myVar VAR LONG(8)

PIN Creates a pin variable. #name = pin number, @name = pin mask

name PIN *pinnumber* [*modifier*]
LED PIN 0 LOW
* modifiers: INPUT, OUTPUT, HIGH, LOW

FUNC...ENDFUNC Creates a named function. Returns 1 LONG value.

name FUNC [*minParams*[,*maxParams*]]
FUNC *name*
  ...
ENDFUNC

Calc FUNC 1

myVar = Calc 1

FUNC Calc
  __param1 = __param1 + 1

  RETURN __param1
ENDFUNC

SUB...ENDSUB Creates a named subroutine

name SUB [*minParams*[,*maxParams*]]
SUB *name*
  ...
ENDSUB

SetDAC SUB 1

SetDAC 1

SUB SetDAC
  ' code to set DAC
ENDSUB

TASK...ENDTASK Creates code that runs in a seperate cog.

name TASK

TASK *name*
  ...
ENDTASK

* Task code runs in a seperate cog.
* VAR variables are not shared between cogs.
* SUBs and FUNCs are not shared between cogs.
* HUB variables and DATA is shared between cogs.

NOP No operation. Does nothing. Uses 1 instruction.

NOP

**** Creates a single line of propeller assembly code.

\ ROR myVar,#1

ASM...ENDASM Creates a block of propeller assembly code.

```
ASM
  ...
ENDASM
```

' Creates a comment.

INPUT Makes a pin an input.

```
INPUT pinname | const
INPUT switch
INPUT 0
```

OUTPUT Makes a pin an output.

```
OUTPUT pinname | const
OUTPUT LED
OUTPUT 1
```

REVERSE Reverse pin direction (input / output)

```
REVERSE pinname | const
REVERSE sensor
REVERSE 2
```

HIGH Makes a pin an output and high.

```
HIGH pinname | const
HIGH LED
HIGH 3
```

LOW Makes a pin an output and low.

```
LOW pinname | const
LOW LED
LOW 4
```

TOGGLE Toggles pin state (high / low)

```
TOGGLE pinname | const
TOGGLE LED
TOGGLE 5
```

BRANCH Variable determines what label to jump to.

```
BRANCH var, label0, label1, label2[, label3[,etc]]
```

STR Converts a variable to ASCII characters.

```
STR array, var, digits

Ascii VAR LONG (6)
Value VAR LONG
STR ascii, value, 6
```

PROGRAM Sets program start label.

```
PROGRAM label
PROGRAM Start
```

INCLUDE Includes propeller assembly code from a seperate file.

LOAD Load PropBASIC code from a seperate file.

PAUSE Pauses for milliseconds.

```
PAUSE value
PAUSE 1000
```

PAUSEUS Pauses for microseconds. Can use fractional value.

```
PAUSEUS value
PAUSEUS 1000
PAUSEUS 4.7
```

END Ends program execution.

```
END
```

INC Adds 1 to a variable.

```
INC varname

cntr VAR LONG
INC cntr
```

DEC Subtract 1 from a variable.

```
DEC varname

cntr VAR LONG
DEC cntr
```

GOTO Jump to a label.

```
GOTO label
GOTO Start
```

GOSUB Jump to a subroutine.

```
GOSUB label

GOSUB Calc

Calc:
  ' Code
  RETURN value
* It is recommended to use named subroutines instead of GOSUB.
```

RETURN Return from a subroutine.

```
RETURN [value]
RETURN 1
```

LET Optional

IF...ELSE|ELSEIF...ENDIF

```
IF var cond value THEN label
IF var cond value THEN
  ' code
ENDIF
IF var cond value THEN
  ' code
ELSE
  ' code
ENDIF
IF var cond value THEN
  ' code
```

```
      ELSEIF var cond value THEN
        ' code
      ELSE
        ' code
      ENDIF
```

DO...LOOP

```
      DO WHILE var cond value
      LOOP

      DO
      LOOP UNTIL var cond value

      DO
      LOOP
```

FOR...TO...STEP...NEXT

```
      FOR var = startvalue TO endvalue
        ' Code
      NEXT
      FOR var = startvalue TO endvalue STEP deltavalue
        ' Code
      NEXT
```

EXIT Ends the current DO...LOOP or FOR...NEXT loop.

```
      EXIT
      IF var cond value THEN EXIT
```

RANDOM Creates a random number from a seed variable.

```
      RANDOM seedvar[, copyvar]
```

SEROUT Serial output.

```
      SEROUT pin, baud, char
```

SERIN Serial input.

```
      SERIN pin, baud, var
```

SHIFTOUT SPI output.

```
      SHIFTOUT datapin, clockpin, mode, value[\bits]
      * mode: LSBFIRST, MSBFIRST
```

SHIFTIN SPI input.

```
      SHIFTIN datapin, clockpin, mode, var[\bits]
      * mode: LSBPRE, LSBPOST, MSBPRE, MSBPOST
```

RCTIME Measures time for pin to change state (in microseconds).

```
      RCTIME pin, state, resultvar
```

PULSIN Measure incoming pulse width in microseconds

```
      PULSIN pin, state, resultVar
```

PULSOUT Create a pulse of specified width.

```
      PULSOUT pin, duration
```

DJNZ Decrease variable and jump to label if not zero.

DJNZ *var*, *label*

I2CSTART Sends an I2C start condition.

I2CSTART *SDAPin*, *SCLPin*

I2CSTOP Sends an I2C stop condition.

I2CSTOP *SDAPin*, *SCLPin*

I2CREAD Reads a byte from the I2C buss.

I2CREAD *SDAPin*, *SCLPin*, *var*[, *ackbitvalue*]

I2CWRITE Writes a byte to the I2C buss.

I2CWRITE *SDAPin*, *SCLPin*, *value*[, *ackbitvar*]

OWRESET Sends a reset on the 1-wire buss.

OWRESET *DQPin*[,*statusVar*]

OWRDBIT Reads a single bit from the 1-wire buss.

OWRDBIT *DQPin*, *var*

OWRDBYTE Reads a byte from the 1-wire buss.

OWRDBYTE *DQPin*, *var*

OWWRBIT Writes a single bit to the 1-wire buss.

OWWRBIT *DQPin*, *value*

OWWRBYTE Writes a byte to the 1-wire buss.

OWWRBYTE *DQPin*, *value*

ON...GOTO Jump to label based on value of a variable.

ON *var* GOTO *label1*, *label2* [, *label3*, [, *etc*]]
ON *var* = *value1*, *value2*, *value3* GOTO *label1*, *label2*, *label3*

ON...GOSUB Sames as ON...GOTO except does a subroutine jump.

ON *var* GOSUB *label1*, *label2* [, *label3*, [, *etc*]]
ON *var* = *value1*, *value2*, *value3* GOSUB *label1*, *label2*, *label3*

GETADDR Gets address of a HUB variable or DATA.

GETADDR *hubVar*, *var*

```
sharedValues HUB LONG(8)

valueAdr     VAR LONG
index        VAR LONG
temp         VAR LONG

GETADDR sharedValue, valueAdr
valueAdr = valueAdr + index
RDLONG valueAdr, temp
```

RDLONG Reads the value of a LONG hub variable or LDATA.

RDLONG *longhubvar*, *var*

RDWORD Reads the value of a WORD hub variable or WDATA.
RDWORD *wordhubvar*, *var*

RDBYTE Reads the value of a BYTE hub variable or DATA.
RDBYTE *bytehubvar*, *var*

WRLONG Writes a new value into a LONG hub variable.
WRLONG *longhubvar*, *value*

WRWORD Writes a new value into a WORD hub variable.
WRWORD *wordhubvar*, *value*

WRBYTE Writes a new value into a BYTE hub variable.
WRBYTE *bytehubvar*, *value*

WAITCNT Waits for the system counter to reach the target value. Then adds the delta value to the variable.
WAITCNT *target*, *delta*

WAITVID Waits for the video serializer to be able to accept new data.
WAITVID *colors*, *pixels*

WAITPEQ Waits for a pin (or set of pins) state to equal a mask value.
WAITPEQ *state*, *mask*
* INA is anded with "mask" then compared to "state".

WAITPNE Waits for a pin (or set of pins) state to NOT equal a mask value.
WAITPNE *state*, *mask*
* INA is anded with "mask" then compared to "state".

COUNTERA / **COUNTERB** Setup hardware counter parameters.
COUNTERA *mode*[, *apin* [, *bpin*[, *frqx*[, *phsx*]]]]
mode:
0 = Counter Disabled
8 = PLL Internal (Video) *
16 = PLL Single-Ended *
24 = PLL Differential *
32 = NCO/PWM Single Ended
40 = NCO/PWM Differential
48 = DUTY Single-Ended
56 = DUTY Differential
64 = POS detector
72 = POS detector with feedback
80 = POSEDGE detector
88 = POSEDGE detector with feedback
96 = NEG detector
104 = NEG detector with feedback
112 = NEGEDGE detector
120 = NEGEDGE detector with feedback
128 = LOGIC never
136 = LOGIC !A & !B
144 = LOGIC A & !B
152 = LOGIC !B
160 = LOGIC !A & B
168 = LOGIC !A
176 = LOGIC A <> B

```
184 = LOGIC !A | !B
192 = LOGIC A & B
200 = LOGIC A = B
208 = LOGIC A
216 = LOGIC A | !B
224 = LOGIC B
232 = LOGIC !A | B
240 = LOGIC A | B
248 = LOGIC always

* For PLL modes add:
  0 = VCO / 128
  1 = VCO / 64
  2 = VCO / 32
  3 = VCO / 16
  4 = VCO / 8
  5 = VCO / 4
  6 = VCO / 2
  7 = VCO / 1
```

General

Math operators can only be used when assigning values to a variable.

Math operators cannot be used in commands.

Only 1 math operator can be used per line.

Only LONG vars are supported. LONG arrays are supported.

Using a variable as an array index generates alot more code.

HUB vars can be BYTE, WORD or LONG. Arrays are supported.

HUB vars can ONLY be accessed with RDBYTE, WRBYTE, RDWORD, WRWORD, RDLONG, WRLONG commands.

Be aware that HUB vars must be address aligned by the size. So if you declare a BYTE then a LONG, there will be three wasted address location between them.

PINs, HUB vars and DATA are global to all COGs (tasks).

VARs, SUBs and FUNCs are local only to the TASK they are declared in.

All TASKS are started before the main code.

TASK code generates a seperate .spin file.

DATA must be declared before the program code. You cannot put the DATA after the program code.

Program template

```
DEVICE P8X32A, XTAL1, PLL16X
FREQ 80_000_000

' Define constants "name CON value"
' Define HUB vars "name HUB BYTE|WORD|LONG = value"
' Define DATA "label DATA value1, value2, etc"
' Define pins " name PIN value INPUT|OUTPUT|HIGH|LOW"
' Define TASKs "name TASK"
' Define vars "name VAR LONG = value"
' Define SUBs and FUNCs "name SUB minParams,maxParams"

PROGRAM START

START:
  ' Program code
END

' SUB and FUNC code
' SUB NAME
'   code
' ENDSUB

' FUNC name
'   code
'  RETURN value
' ENDFUNC


' TASK code (remember TASK have private VARs, SUBs and FUNCs)
' TASK name
'   code
' ENDTASK
```