

***DiosPro Function
Libraries***

***DiosPro Series
Volume 3***

Dios Functions

Version 3.3

Updates

Software and updated manuals can be obtained from the Kronos Robotics web site located at www.kronosrobotics.com.

Forums

A web-based discussions board is located at one of the Kronos Robotics sister sites. These forums cover everything from the Dios product line to motor controller basics and robotics. They are located at: www.mgsweb.com/forum.

Warranty

Kronos Robotics warranties its products against defects in material and workmanship for a period of 90 days. If you discover a defect Kronos Robotics will, at its option, repair, replace, or refund the item's purchase price. Simply contact Kronos Robotics for an RMA. The customer is responsible for all return shipping expenses.

Disclaimer of Liability

Kronos Robotics cannot be held responsible for any incidental, or consequential damages resulting from the use of any Kronos Robotics products.

15-Day Money-Back Guarantee

It is very important to us here at Kronos Robotics that our customers are completely satisfied. If you are not happy with any product you purchase from Kronos Robotics it may be returned for a full refund or exchange within 15 days of the invoice date.

The returned items must be in unused condition and have original packaging. There will be a restocking fee of 30% only on items that do not meet this condition. Also note that this return policy does not apply to items that you have destroyed or damaged. For example if you hook up a microcontroller backwards you may not return it.

Shipping and Handling fees are non-refundable. The customer is responsible for all return shipping expenses.

Shipping Responsibility

Kronos Robotics ships all packages Priority Air Mail via the United States Postal Service with delivery confirmation. We have found this combination gives the fastest and most reliable delivery at a very reasonable cost.

Once a package has been delivered we are no longer responsible for the package. More specifically, if someone steals the package from your doorstep we will not be held responsible and no refund will be provided.

If your package has not been delivered and sufficient time has passed please email us and we will verify delivery. If delivery has been made you will be given the confirmation number and delivery details. Thereafter you must contact your local Post Office for further information.

TradeMarks

PIC is a registered trademark of Microchip Technology, Inc. Windows is a trademark of Microsoft Corporation. 1-wire is a trademark of Dallas Semiconductor.

Contacts

email: sales@kronosrobotics.com
phone: 703 779-9752
fax: 703 779-9753
web: www.kronosrobotics.com



Contents

Library Syntax Overview	4
Function Index	5
Dios 74HC165 Input Shift Register Functions	6
Dios DS1820 Thermometer Functions	10
Dios DS1620 Thermometer Functions	12
Dios 1-Wire Functions	16
Dios 74HC595 Output Shift Register Functions	18
Dios Analog to Digital Functions	22
Dios Button Functions	26
Dios DS1302 Real Time Clock Functions	28
Dios High Speed Servo Functions	34
Dios Hardware PWM Functions	36
Dios I2c Functions	38
Dios IR Module Functions	46
Dios LVD Functions	50
Dios Math Functions	52
Dios MAX522 DAC Functions	54
Dios MicroChip Digital Potentiometer Functions	56
Dios Random Functions	58
Dios RSkeypad Functions	60
Dios String Functions	62
Dios Tone Functions	66
Dios TW523 (X10) Functions	68

How to read function syntax

Each function name will be followed by a syntax example and a syntax parameter type. Some functions can take repeating arguments. These will be indicated by the All the parameters in a function are enclosed in (). If the function has optional parameters they will be inclosed in [].

For example:

DS1820quicktemp

```
DS1820quicktemp(port,mode,addr)
DS1820quicktemp(iexp,iexp,iexp) as integer
```

```
DS1820quicktemp(port,mode,addr)
```

This syntax example is used as a reference and each parameter will be explained in the function description.

```
DS1820quicktemp(iexp,iexp,iexp) as integer
```

This syntax example shows the parameter type. It indicates the type for each parameter. Each type will be explained in detail below.

- **iexp** - Integer expression. Any variable type, literal mathematical expression. Bit and Byte extensions are supported. Registers and array elements are also supported. All will be converted to integer before used.
- **fexp** - Floating point expression. Any variable type, literal mathematical expression. Bit and Byte extensions are supported. Registers and array elements are also supported. All will be converted to floating point before used.
- **intvarb**: Integer variable accepted only. No arrays, bit or byte extensions.
- **floatvarb**: Floating point variable accepted only. No arrays, bit or byte extensions.
- **stringvarb**: String variable accepted only. No tables or literal strings.
- **label**: Valid location label.
- **string**: Quoted string as in "hello world". String variables and tables are also supported.

Notes

- All the example code contain the **DiosPro** directive. If you are using the older standard **Dios** use the Dios directive.
- The CoProc libraries have been removed. These are going to be replaced by new ZProc libraries. The old CoProc libararies are still included and will still function.

AtoDinit22	I2c_getack42	OWloadaddr17
AtoDinitlj22	I2c_getbyte42	OWprintaddr16
AtotD22	I2c_sendack43	OWread17
Button26	I2c_sendbyte42	OWreadbyte16
Buttoninit26	I2c_sendnack43	OWreadrom16
CelsiusToF14	I2c_start43	OWreset17
DS1302getbyte32	I2c_stop43	OWsendbyte16
DS1302getdate30	I2cin38	OWwrite017
DS1302getday30	I2cin238	OWwrite117
DS1302gethour31	I2cout38	PWM1duty36
DS1302getmin30	I2cout239	PWM2duty37
DS1302getmonth30	I2creadfloat40	PWMcourse36
DS1302getram31	I2creadint40	PWMinit36
DS1302getrawbyte32	I2creadstring41	PWMperiod36
DS1302getsec30	I2cwritefloat40	random58
DS1302getyear30	I2cwriteint39	randomize58
DS1302init28	I2cwritestring41	RSKeypadinit60
DS1302sendbyte32	IRinitsend47	RSKeypadread60
DS1302sendreg32	IRKeypad46	SERVO1us34
DS1302setcharger31	IRread46	SERVO2us34
DS1302setdate28	IRreadverify46	SERVO3us34
DS1302setday28	IRsendcode47	SERVO4us34
DS1302sethour29	LVDinit50	SERVOinit34
DS1302setmin29	LVDread50	STRgetword62
DS1302setmonth28	MATHabs52	STRgetwordaddr62
DS1302setram31	MATHcos52	STRinstr63
DS1302setsec29	MATHcot52	STRlen63
DS1302setyear29	MATHcsc52	STRprinttext63
DS1620init12	MATHfactor52	STRval64
DS1620readtemp12	MATHgcd52	TONE66
DS1620readTH13	MATHmod52	TONEfreqout66
DS1620readTL14	MATHradians53	TONEnote66
DS1620setTH13	MATHsec53	TW523convertfrom72
DS1620setTL13	MATHsin53	TW523convertto72
DS1820quicktemp10	MATHsqr53	TW523read69
DS1820readtemp10	MATHsqrt53	TW523readbits70
HC165init6	MATHtan53	TW523sendrawbits70
HC165read6	MATHxt53	TW523waitzerocross70
HC165readx26	MATHy2x53	TW523write68
HC595high18	MAX522init54	TW523writebits69
HC595init18	MAX522set154	TW523writeverify68
HC595low19	MAX522set254	TW53pause69
HC595write18	MCPset56		
HC595writem18	MCPshutdown56		

Dios 74HC165 Input Shift Register Functions

Using the 74HC165 input shift register we can add some input ports to our microcontoller.

The process is very easy.

- pulse the load pin
- shift-in the data

If you cascade more than one 74HC165 you can shift 8 ioports for each 74HC165 you cascade.

The example program calls the HC165init routine to setup the ioports. You then make calls to HC165read or HC165readx2 to read the input ports on the 74HC165. Once the data has been shifted in, access the individual bits on the variable that you loaded. If using the HC165read function, then only bits 0-7 need be accessed. If using the HC165readx2 function, then bits 0-15 are used.

HC165init

HC165init(dat,clk,load)
HC165init(iexp,iexp,iexp)

Description

This function sets up the IO ports needed to communicate with the 74HC165. It must be called before any of the others.

- **dat** - The IO port connected to the serial out pin on the 74HC165
- **clk** - The IO port connected to the clock pin on the 74HC165.
- **load** - The IO port connected to the load pin on the 74HC165.

HC165read

HC165read() as read
HC165read() as integer

Description

Returns the 8 input ports as a single byte. 0-255. Use Schematic 1.

HC165readx2

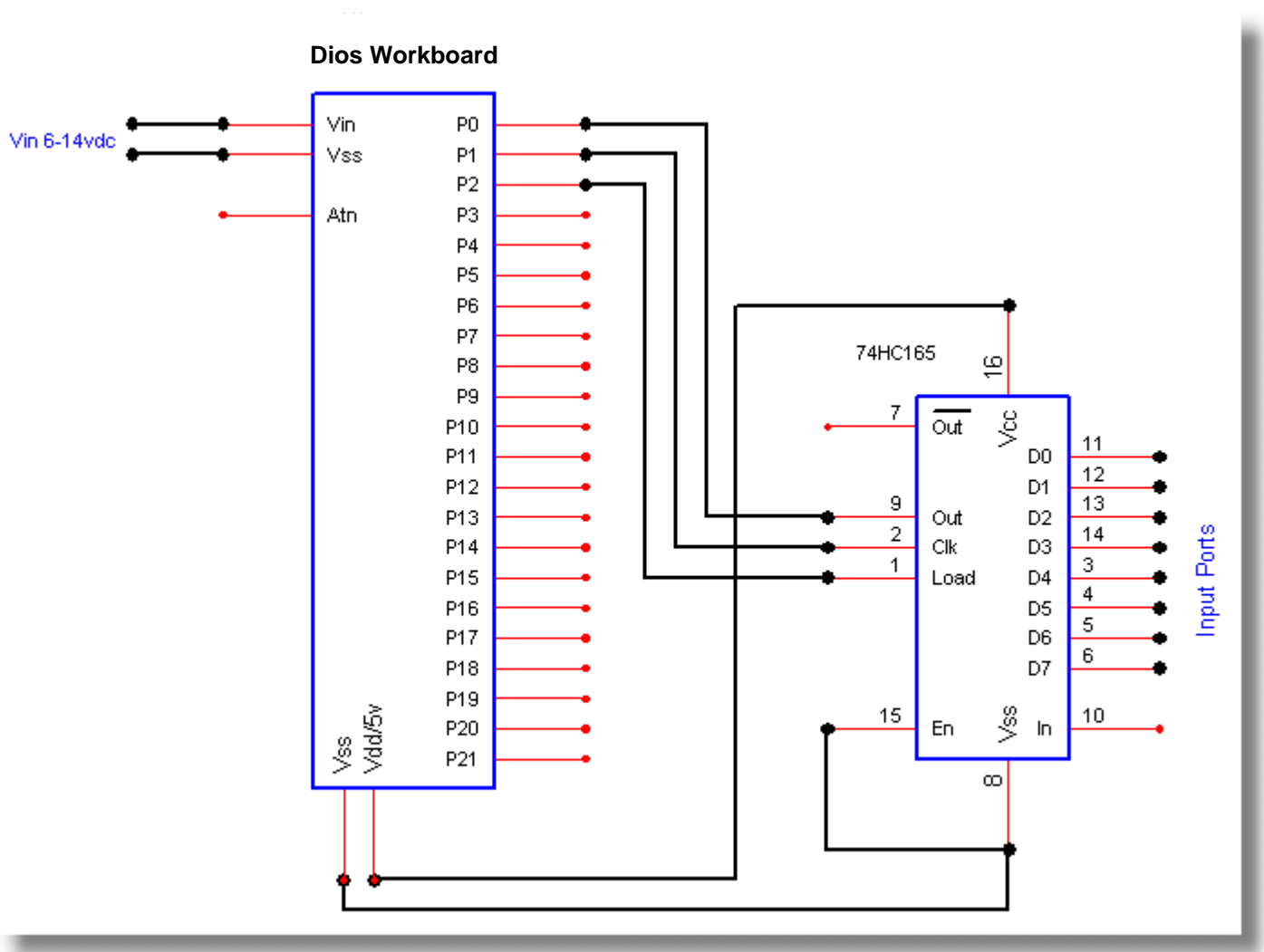
HC165readx2()
HC165readx2() as integer

Description

Returns the 16 input ports as a single word 0-65535. Use Schematic 2.

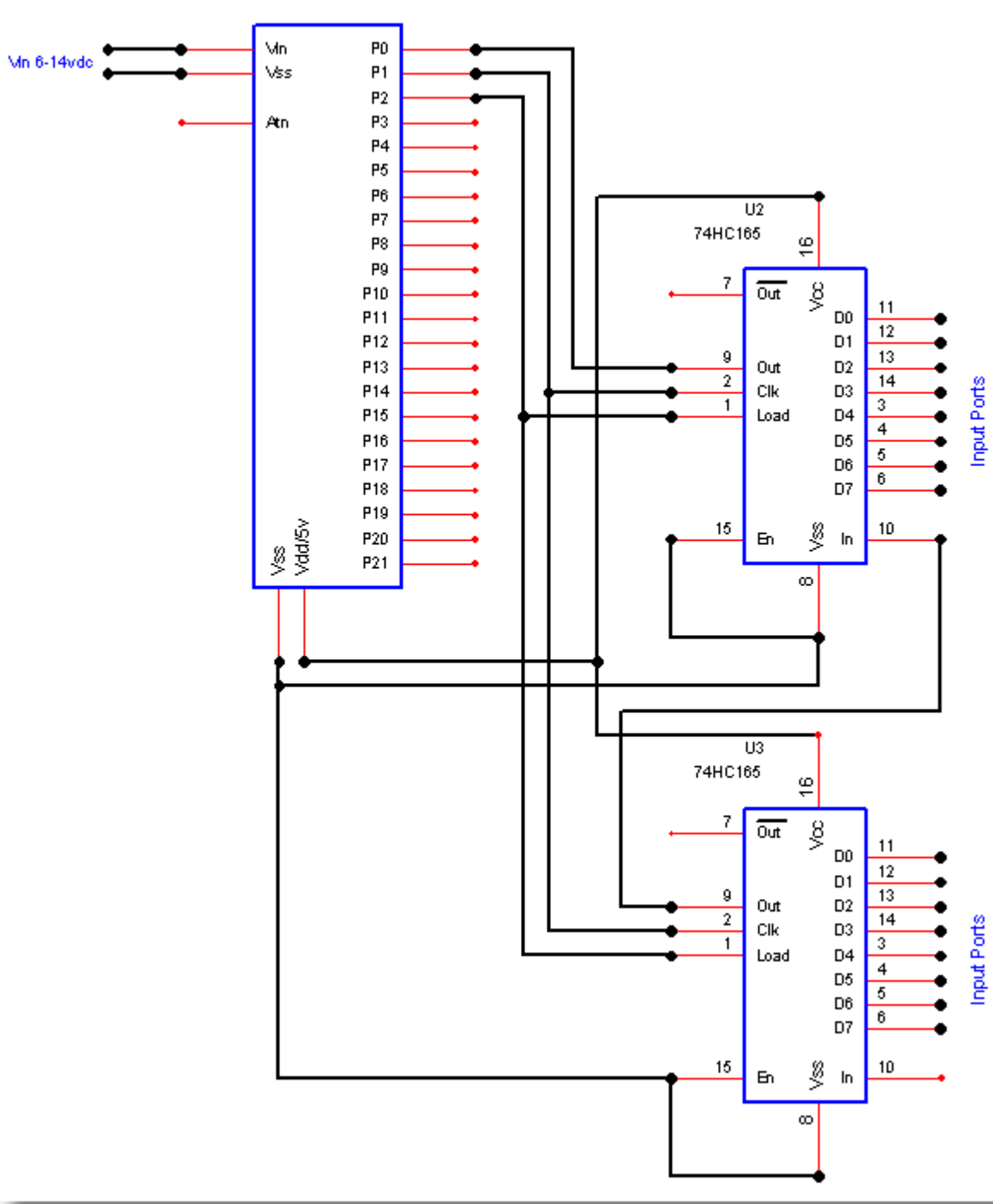
Schematic 1

This schematic shows the use of one 74HC165 to create an 8-bit input port.



Schematic 2

This schematic shows the use of two 74HC165 chained to create a 16-bit input port.



```
DiosPro
'74HC165 Output shift Register Example. 1 chip.
'See Schematic 1
func main()
  dim datin

  HC165init(0,1,2)

loop:
  datin = HC165read()
  print datin
  goto loop

endfunc

include \lib\Dios165.lib
```

```
DiosPro
'74HC165 Output shift Register Example. 2 chips.
'See Schematic 2
func main()
  dim datin

  HC165init(0,1,2)

loop:
  datin = HC165readx2()
  print datin
  goto loop

endfunc

include \lib\Dios165.lib
```

Dios DS1820 Thermometer Functions

Note: This library is not compatible with the DS18B20 chip. You must use a DS1820, DS1820S or DS18S20 chip with these libraries.

The DS1820 is a 1Wire Device that can deliver very high precision temperatures yet will only use a single IO port on your microcontroller.

These functions are used to access the DS1820 High Precision Digital Thermometer. We show both external power mode and parasite mode. While the parasite mode only requires 2 wires it is good for remote applications but has a slower conversion rate.

For Celsius to Fahrenheit conversions use the CelsiustoF function.

DS1820quicktemp

DS1820quicktemp(port,mode,addr)
DS1820quicktemp(iexp,iexp,iexp) as integer

Description

Returns a 1-byte integer value of the temperature in centigrade. The resolution is 1 degree Celsius.

- **port** - Dios IO port connected to the slave device data pin.
- **mode** - Determines parasite and Rom match.
 - 0: No parasite, No Rom Match (Single Device on Bus).
 - 1: Parasite Mode, No Rom Match (Single Device on Bus).
 - 2: No Parasite, Rom Match (You must pass string with 8 bytes of address data).
 - 3: Parasite Mode. Rom Match (You must pass string with 8 bytes of address data).

If Parasite mode is active you must tie pins 1 and 3 together and wire to Vss.

If Parasite mode is not active Vss is applied to pin 1 and Vcc to pin 3.

If Rom Match is active the passed string must match a device to read it.

If Rom Match is not active only one device can be active on the bus.

- **addr** Is an 8-byte string that contains the address of the device you wish to talk to.

DS1820readtemp

DS1820readtemp(port,mode,addr)
DS1820readtemp(iexp,iexp,iexp) as float

Description

Returns floating point value of the temperature in centigrade. The resolution is .1 degree Celsius.

- **port** - Dios IO port connected to the slave device data pin.

- **mode** - Determines parasite and Rom Match

- 0: No parasite, No Rom Match (Single Device on Bus).
- 1: Parasite Mode, No Rom Match (Single Device on Bus).
- 2: No Parasite, Rom Match (You must pass string with 8 bytes of address data).
- 3: Parasite Mode. Rom Match (You must pass string with 8 bytes of address data).

If Parasite mode is active you must tie pins 1 and 3 together and wire to Vss.
 If Parasite mode is not active Vss is applied to pin 1 and Vcc to pin 3.
 If Rom Match is active the passed string must match a device to read it.
 If Rom Match is not active only one device can be active on the bus.

- **addr** Is an 8-byte string that contains the address of the device you wish to talk to.

```

DiosPro
'DS1820 Library Example
func main()
  dim celsius as float
  dim fahrenheit as float

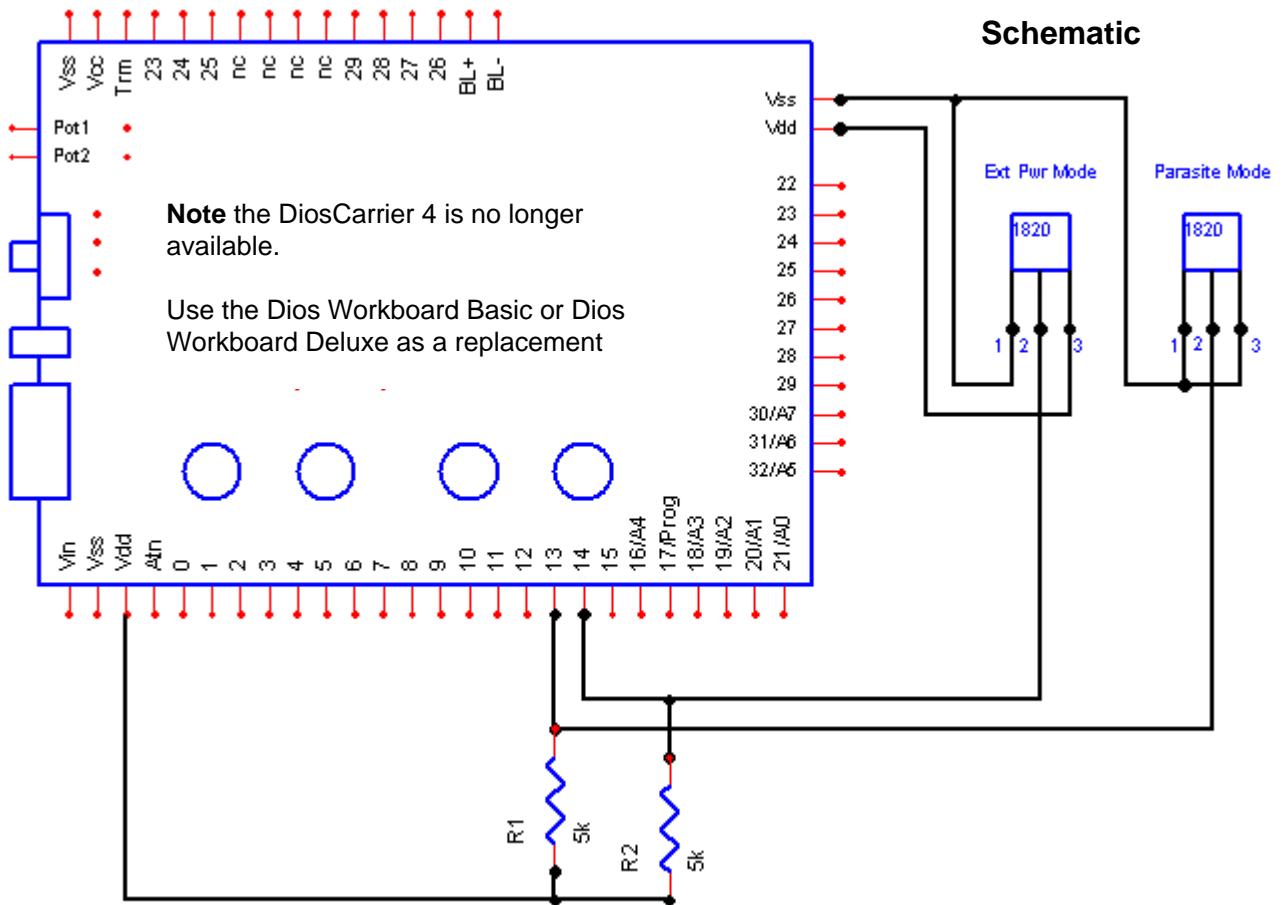
  loop:

    celsius=DS1820readtemp(13,1)
    fahrenheit = CelsiusToF(celsius)
    print {-0.2} celsius,"C ",fahrenheit,"F"

  goto loop

endfunc

include \lib\Dios1820.lib
    
```



Dios DS1620 Thermometer Functions

The DS1620 is a very accurate thermometer. It is accurate to .5 Celsius. The routines below will convert from Celsius to Fahrenheit.

Features

- Requires no external components
- Supply voltage range covers 2.7v to 5.5v
- Measures temperatures from -55°C to +125°C in .5°C increments and -67°F to +257°F in .9°F increments
- Converts temperature to to digital in 1 second (max)
- 3-Wire Interface

The following functions will make interfacing to the DS1620 extremely easy.

DS1620init

DS1620init(IO,CLK,RST)
DS1620init(iexp,iexp,iexp)

Description

This must be called before any of the other routines are called. It sets up the DS1620 for reading.

- **IO** - The Dios port connected to the DQ line of the DS1620.
- **CLK** - The Dios port connected to the CLK line of the DS1620.
- **RST** - The Dios port connected to the RST line of the DS1620.

DS1620readtemp

DS1620readtemp(IO,CLK,RST)
DS1620readtemp(iexp,iexp,iexp) as float

Description

Reads the the temperature of the DS1620. Returns a FLOAT value.

- **IO** - The Dios port connected to the DQ line of the DS1620.
- **CLK** - The Dios port connected to the CLK line of the DS1620.
- **RST** - The Dios port connected to the RST line of the DS1620.

DS1620setTH

DS1620setTH(IO,CLK,RST,TEMP)
DS1620setTH(iexp,iexp,iexp,fexp)

Description

Sets the temperature of the high trigger in the DS1620 Pin 7.

- **IO** - The Dios port connected to the DQ line of the DS1620.
- **CLK** - The Dios port connected to the CLK line of the DS1620.
- **RST** - The Dios port connected to the RST line of the DS1620.
- **TEMP** - The temperature in Celsius to trigger pin 7 on DS1620. Note that temperatures of -125c to +125c are supported

DS1620setTL

DS1620setTL(IO,CLK,RST,TEMP)
DS1620setTL(iexp,iexp,iexp,fexp)

Description

Sets the temperature of the low trigger in the DS1620 Pin 7.

- **IO** - The Dios port connected to the DQ line of the DS1620.
- **CLK** - The Dios port connected to the CLK line of the DS1620.
- **RST** - The Dios port connected to the RST line of the DS1620.
- **TEMP** - The temperature in Celsius to trigger pin 6 on DS1620. Note that temperatures of -125c to +125c are supported

DS1620readTH

DS1620readTH(IO,CLK,RST,TEMP)
DS1620readTH(iexp,iexp,iexp,fexp) as float

Description

Reads the temperature of the high trigger in the DS1620

Returns setting in Celcius

- **IO** - The Dios port connected to the DQ line of the DS1620.
- **CLK** - The Dios port connected to the CLK line of the DS1620.
- **RST** - The Dios port connected to the RST line of the DS1620.

DS1620readTL

DS1620readTL(IO,CLK,RST,TEMP)
DS1620readTL(iexp,iexp,iexp,fexp) as float

Description

Reads the temperature of the low trigger in the DS1620

Returns setting in Celcius

- **IO** - The Dios port connected to the DQ line of the DS1620.
- **CLK** - The Dios port connected to the CLK line of the DS1620.
- **RST** - The Dios port connected to the RST line of the DS1620.

CelsiustoF

CelsiustoF(celsius)
CelsiustoF(fexp) as float

Description

Returns the converted celsius data as fahrenheit. Returns a **FLOAT** value.

- **celsius** - This is a float value that represents the celcius reading.

```

DiosPro
'DS1620 Library Example
func main()

  dim celsius as float
  dim fahrenheit as float

  DS1620init(0,1,2)

again:

  celsius = DS1620readtemp(0,1,2)
  fahrenheit = CelsiustoF(celsius)
  print {0.1} celsius
  printf {0.1} fahrenheit

  pause 500
  goto again

endfunc

include \lib\Dios1620.lib
    
```

```

DiosPro
'DS1620 Library Example
func main()
  dim celsius as float
  dim fahrenheit as float

  print "Dios DS1620 Library v 2.1"

  DS1620init(0,1,2)

  DS1620setTH(0,1,2,24) 'Set High Trigger
  DS1620setTL(0,1,2,23) 'Set Low Trigger

again:
  celsius = DS1620readtemp(0,1,2)
  fahrenheit = CelsiustoF(celsius)

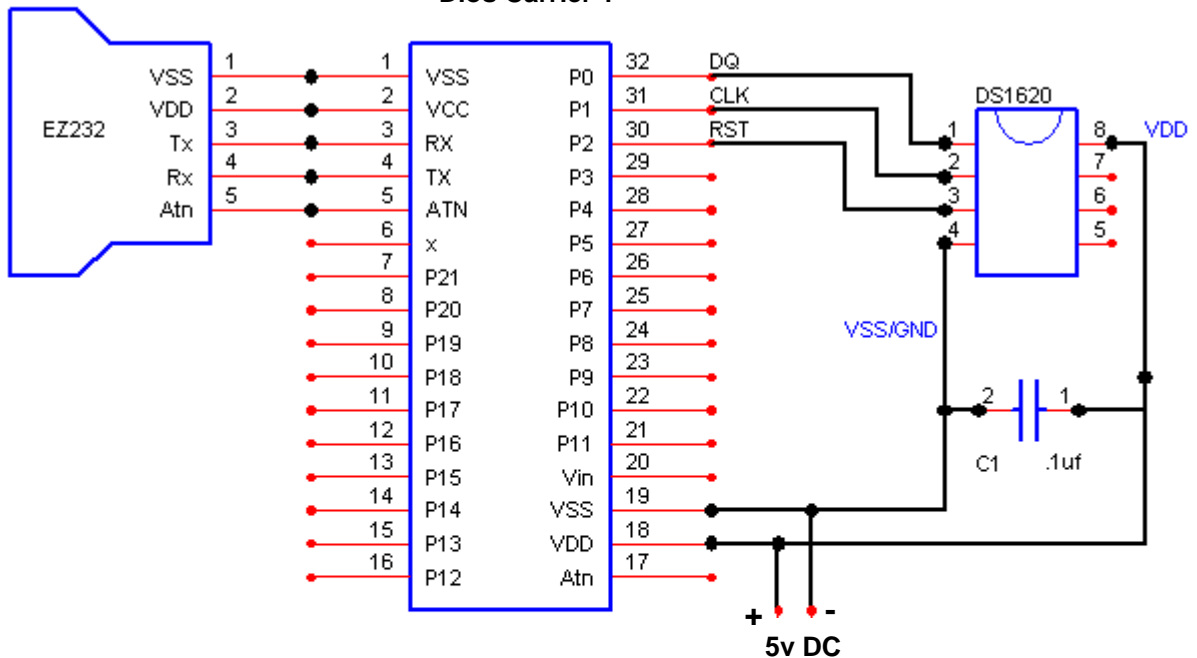
  print celsius,"c ";
  print fahrenheit,"f"

  pause 500
  goto again
endfunc

include \lib\Dios1620.lib
    
```

Schematic

Dios Carrier 1



Dios 1-Wire Functions

These functions are used to access 1Wire Slave Devices.

High Level Functions

OWreadbyte

OWreadbyte(port)
OWreadbyte(iexp) as integer

Description

Returns 1 byte of data read from the slave device.

- **port** - The Dios IO port connected to the slave device data pin.

OWsendbyte

OWsendbyte(port,byte)
OWsendbyte(iexp,iexp)

Description

Sends a byte of data to the slave device.

- **port** - Dios IO port connected to the slave device data pin.
- **byte** - The byte to write to the slave device.

OWreadrom

OWreadrom(port,stringvarb)
OWreadrom(iexp,stringvarb)

Description

Retrieves the 8-byte serial address of a single 1Wire device on the bus.

- **port** - Dios IO port connected to the slave device data pin.
- **stringvarb** - You must define this string to hold 8 bytes of data, e.g. global mystring(8) as string. We are using the mystring variable as a byte array.

OWprintaddr

OWprintaddr(stringvarb)
OWprintaddr(stringvarb)

Description

Displays the 8-byte device address in the debug window. You should call the OWreadrom function before you use this function.

- **stringvarb** - This is the string variable you used in the OWreadrom function.

OWloadaddr

OWloadaddr(b1,b2,b3,b4,b5,b6,b7,b8,stringvarb)
OWloadaddr(*iexp,iexp,iexp,iexp,iexp,iexp,iexp,iexp,stringvarb*)

Description

Loads an 8-byte (64-bit) address into a string variable. The string variable is used as a byte array.

- **stringvarb** - You must define this string to hold 8 bytes of data, e.g. global mystring(8) as string

Low Level Functions

OWreset

OWreset(port)
OWreset(*iexp*)

Description

Resets bus and returns a 1 if the presence of a bit is detected.

- **port** - Dios IO port connected to the slave device data pin.

OWwrite1

OWwrite1(port)
OWwrite1(*iexp*)

Description

Writes a 1 in a single bit time slot.

- **port** - Dios IO port connected to the slave device data pin.

OWwrite0

OWwrite0(port)
OWwrite0(*iexp*)

Description

Writes a 0 in a single bit time slot.

- **port** - Dios IO port connected to the slave device data pin.

OWread

OWread(port)
OWread(*iexp*) as integer

Description

Returns a single bit from the slave device, 0-1.

- **port** - Dios IO port connected to the slave device data pin.

Dios 74HC595 Output Shift Register Functions

If you have ever wanted to add a few more output ports to your Dios this is the cheapest and easiest way. The 74HC595 is an 8-bit shift register. It uses three IO ports, but you will gain 8 output ports. It is possible to use this device to interface to other 8-bit items such as DACS or EEPROMS. Also if you are going to display lots of data via LED's then there is nothing better. You can even tie more than one chip together to create larger registers.

We are showing hookup to the Dios Mini Ultra. However the connection to other Dios form factors are the same.

HC595init

HC595init(serin,clk,latch)
HC595init(iexp,iexp,iexp)

Description

This function must be called before any of the others.

- **serin** - The IOport connected to the serial input pin on the 74HC595.
- **clk** - The IO port connected to the clock pin on the 74HC595.
- **latch** - The IO port connected to the latch pin on the74HC595.

HC595write

HC595write(value)
HC595write(iexp)

Description

Shifts the value out to the 74HC595 register. Once shifted the new pins on the 74HC595 will be set high or low depending on the value given. Note that only the lower 8 bits in the expression will be used.

- **value** - This represents the 8 output pins on the 74HC595. So a value of 255 will set all the pins to high and a value of 0 will set them all low.

HC595writem

HC595writem(num,value1,.....)
HC595writem(iexp,iexp,.....)

Description

If you cascade more than 174HC595 you can use this command to access them.

- **num** - Used to tell the function how many 74HC595's are connected. You then provide a value for each74HC595.
- **value1-valuen** - These are the values for each of the 74HC595 chips connected.

HC595high

HC595high(port)
HC595high(iexp)

Description

This command sets an individual port on the 74HC595 to high. Valid ports are 0-7.

- **port** - The output bit to change on the 74HC595. Valid range is 0 to 7.

HC595low

HC595low(port)

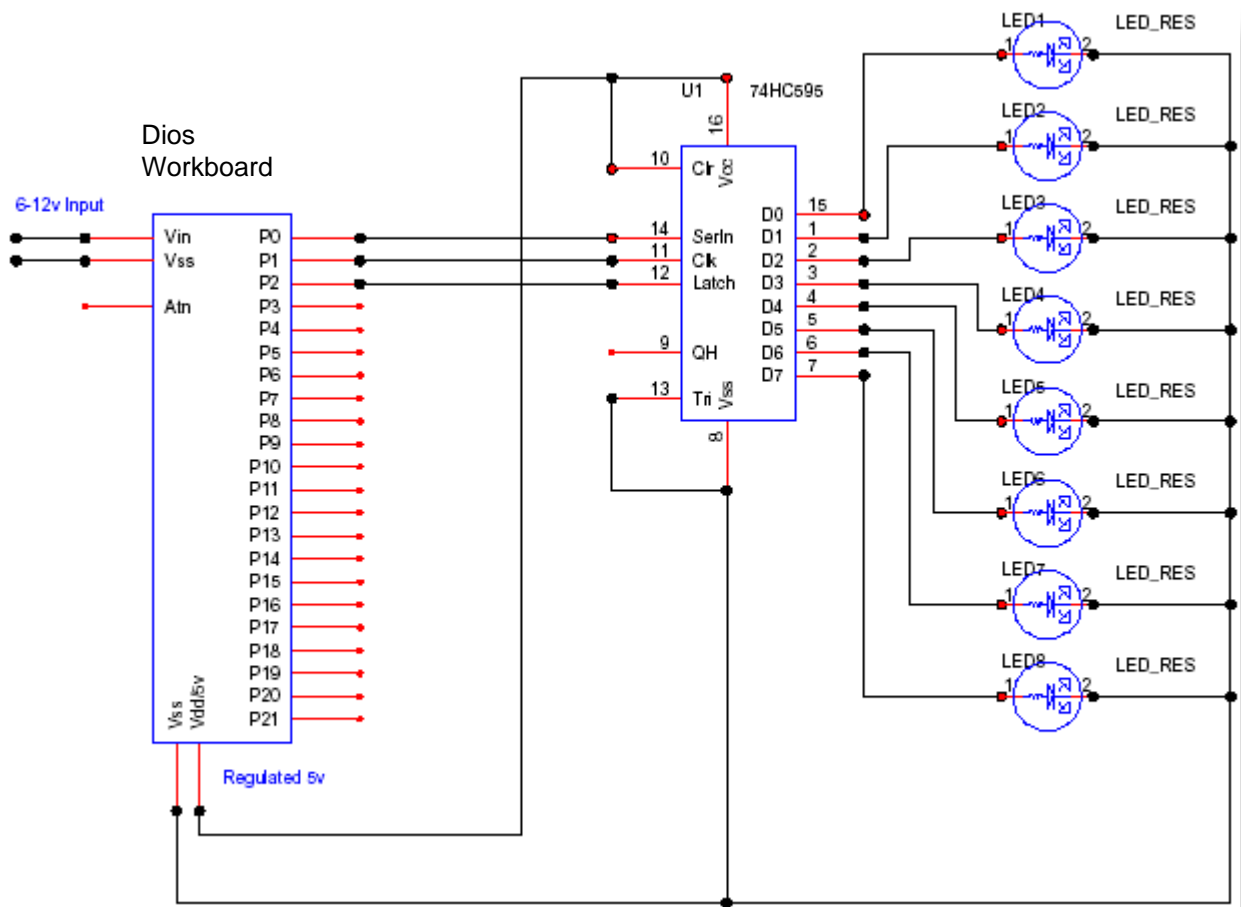
HC595low(iexp)

Description

This command sets an individual port on the 74HC595 to low. Valid ports are 0-7

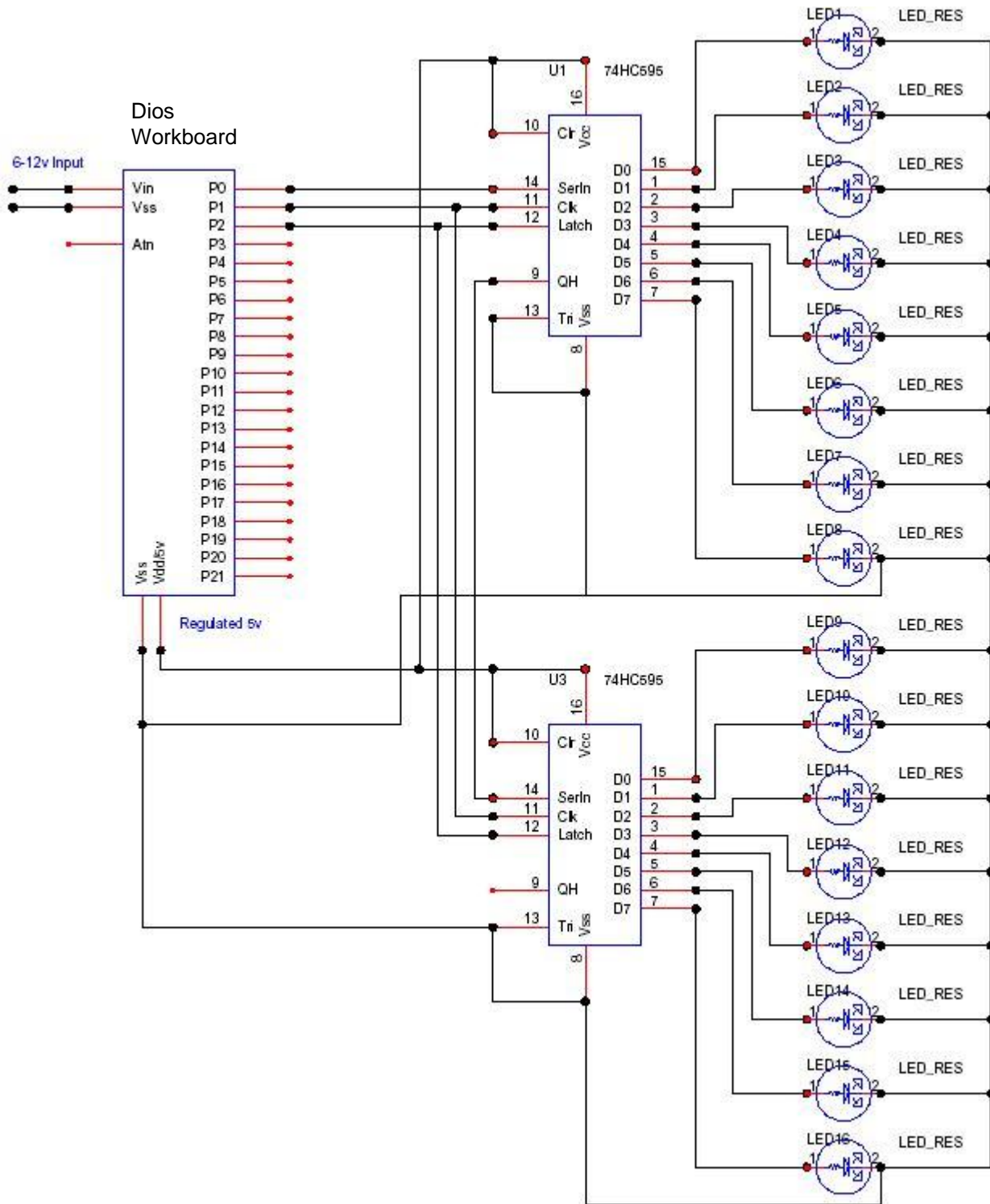
- **port** - The output bit to change on the 74HC595. Valid range is 0 to 7.

Schematic 1



In schematic 1 the Dios is connected to a single 74HC595 shift register. An LED (with internal resistor) is connected to each of the output leads of the 74HC595. This will create an 8-bit output port.

Schematic 2



In schematic 2 the Dios is connected to two 74HC595 shift registers. This creates a 16-bit output port.

```
DiosPro
'74HC595 Output Shift Register Example #1
func main()
dim x

HC595init(0,1,2)

again:

for x = 0 to 255
  HC595write(x)
  pause 10
next

  goto again

endfunc

include \lib\Dios595.lib
```

```
DiosPRo
'74HC595 Output Shift Register Example #2
func main()
dim x

HC595init(0,1,2)

again:

for x = 0 to 65535
  HC595writem(2,x.byte(0),x.byte(1))
  pause 10
next

  goto again

endfunc

include \lib\Dios595.lib
```

Dios Analog to Digital Functions

The Dios has 5 or 8 10-bit analog ports built-in depending on the chip or module used. Access to this data is quite easy. Just set up the ports you want to use and call the AtoD function. Note that you can set up an external reference; however this is quite tricky and you are much better off using the built-in 5v reference.

Note that the voltage should never exceed Vdd. If you are running the Dios at 3v then the maximum voltage for an analog to digital conversion is 3 volts. The actual speed that a port can be read is about 5000 times a second. The slower the speed the more stable the reading.

AtoDinit

AtoDinit(config)
AtoDinit(iexp)

Description

This function is used to configure the analog ports for conversion. It must be called before the AtoD function. It only needs to be called once.

- **config** - A number from 0 to 15 that determine the port numbers and reference pins to be used the conversions. Refer to the chart for valid options. See the chart for actual values.

If you select a configuration that uses analog port (IO Port 16) you must move the program pin to IO port 3.

Valid Ports

Dios 28 Pin Ports 16, 18, 19, 20, 21

Dios 40 Pin Ports 16, 18, 19, 20, 21, 30, 31, 32

DiosPro 28 Pin Ports 16, 18, 19, 20, 21, 3, 4, 5, 6, 7

DiosPro 40 Pin Ports 16, 18, 19, 20, 21, 30, 31, 32, 3, 4, 5, 6, 7

AtoDinitlj

AtoDinitlj(config)
AtoDinitlj(iexp)

Description

This function is the same as the AtoDinit but the results from all AtoD commands will be left justified. This is better for some integer math.

AtoD

AtoD(port)
AtoD(iexp) as integer

Description

This function returns the analog value of a given port.

Returns 0-1023.

- **port** - The analog port number to read, 0-7. Note that the valid numbers are determined by the config data in the `initAtoD` function.

Important

Using `AtoDinit(0)` yields the following.

AD0 = P21
AD1 = P20
AD2 = Not usable
AD3 = P18
AD4 = P16 (In order to use this analog port you must configure the Dios to use port 3 for debug)
AD5 = P32
AD6 = P31
AD7 = P30

AD8 = P5 *
AD9 = P4 *
AD10 = P6 *
AD11 = P3 *
AD12 = P7 *

* DiosPro only

The IO ports should be configured for input via the input command.
Use the AN number when using the `AtoD` function.

Note

The use of external reference is not recommended.

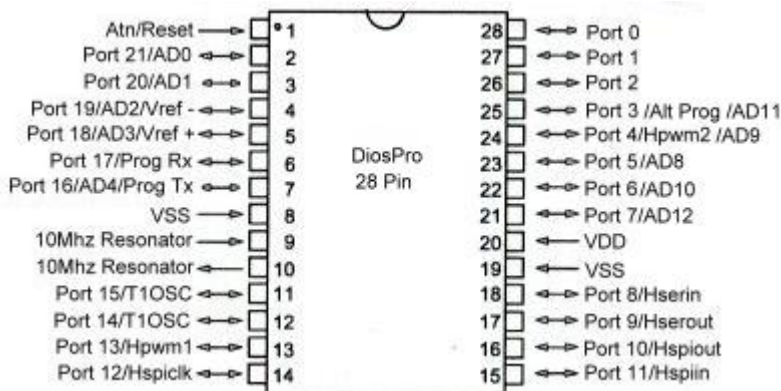
```
DiosPro
'Analog to Digital Example
func main()
  dim a,b,c

  AtoDinit(4) 'Init 3 Analog ports

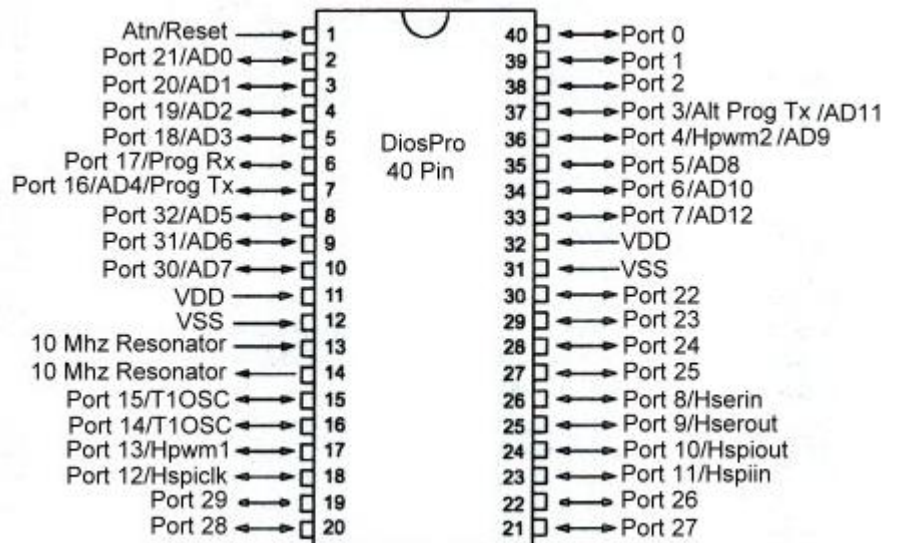
loop:
  a=AtoD(0)
  b=AtoD(1)
  c=AtoD(3)
  print a," ",b," ",c
  pause 500
  goto loop

endfunc

include \\lib\DiosAtoD.lib
```

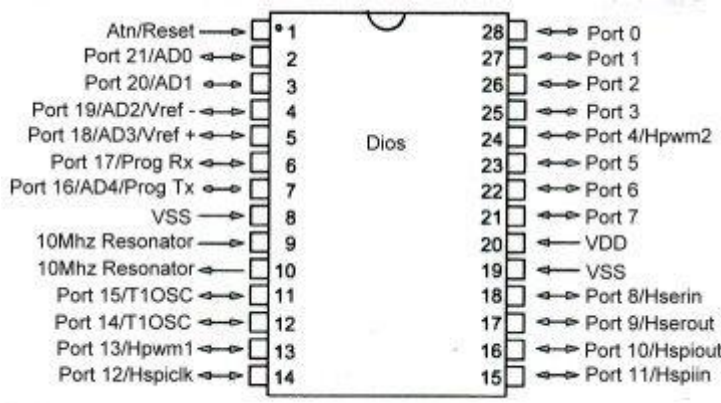


DiosPro Schematics

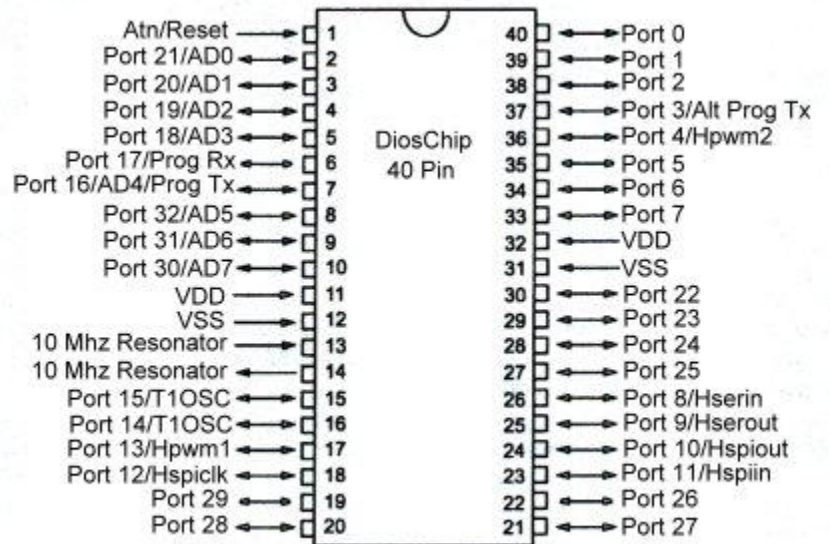


DiosPro Chart

Config Value	AD12 Port7	AD11 Port13	AD10 Port6	AD9 Port4	AD8 Port5	AD7 Port30	AD6 Port31	AD5 Port32	AD4 Port16	AD3 Port18	AD2 Port19	AD1 Port20	AD0 Port21
0	A	A	A	A	A	A	A	A	A	A	A	A	A
1	A	A	A	A	A	A	A	A	A	A	A	A	A
2	A	A	A	A	A	A	A	A	A	A	A	A	A
3	D	A	A	A	A	A	A	A	A	A	A	A	A
4	D	D	A	A	A	A	A	A	A	A	A	A	A
5	D	D	D	A	A	A	A	A	A	A	A	A	A
6	D	D	D	D	A	A	A	A	A	A	A	A	A
7	D	D	D	D	D	A	A	A	A	A	A	A	A
8	D	D	D	D	D	D	A	A	A	A	A	A	A
9	D	D	D	D	D	D	D	A	A	A	A	A	A
10	D	D	D	D	D	D	D	D	A	A	A	A	A
11	D	D	D	D	D	D	D	D	D	A	A	A	A
12	D	D	D	D	D	D	D	D	D	D	A	A	A
13	D	D	D	D	D	D	D	D	D	D	D	A	A
14	D	D	D	D	D	D	D	D	D	D	D	D	A
15	D	D	D	D	D	D	D	D	D	D	D	D	D



Dios Schematics



Dios Chart

config value	¹ AD7 Port30	¹ AD6 Port31	¹ AD5 Port32	² AD4 Port16	AD3 Port18	AD2 Port19	AD1 Port20	AD0 Port21	VREF+	VREF-
0	A	A	A	A	A	A	A	A	VDD	VSS
1	A	A	A	A	VREF+	A	A	A	AN3	VSS
2	D	D	D	A	A	A	A	A	VDD	VSS
3	D	D	D	A	VREF+	A	A	A	AN3	VSS
4	D	D	D	D	A	D	A	A	VDD	VSS
5	D	D	D	D	VREF+	D	A	A	AN3	VSS
6	D	D	D	D	D	D	D	D	—	—
8	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2
9	D	D	A	A	A	A	A	A	VDD	VSS
10	D	D	A	A	VREF+	A	A	A	AN3	VSS
11	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2
12	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2
13	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2
14	D	D	D	D	D	D	D	A	VDD	VSS
15	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2

Dios Button Functions

There are many ways to access buttons. One way is to use the following routine:

```
loop:
if ioport(5) = 1 then
  print "Button Down"

  while ioport(x)=1
  wend

  print "Button Up"
endif
goto loop
```

The following button functions will yield more features as well as the ability to do repeat rate and timeouts.

Buttoninit

Buttoninit(IOport)
Buttoninit(iexp)

Description

This function sets up the button IO port and resets the internal flags. It should be called once for each button you are going to process.

- **IOport** - The IO port the button is going to be placed on, 0-32.

Button

Button(IOport,Delay,Rate,IdleState)
Button(iexp,iexp,iexp,iexp) as integer

Description

Tests the state of the button. If the button is held down for more than the delay setting it returns a value of 1 at the speed setup by the Rate parameter.

returns a 1 if the button is on and a 0 if it is off

- **IOport** - The IO port the button is connected to, 0-32.
- **Delay** - A value between 0 and 65535 that represents the time the BUTTON function will wait while in the on state (button down) before it starts returning a value of 1 (auto rate). This value is an internal counter the will represent a time interval close to 1 millisecond.
- **Rate** - A value between 0 and 65535 that represents the number of milliseconds to wait for each auto rate event returned.

For example:

BUTTON(0,2500,50)

Will return a value of 1 every 50 milliseconds while the button is down longer than 2500 milliseconds.

- **IdleState** - An optional value that indicates the idle state of the button. If omitted an idle state of 0 is assumed.

```

DiosPro
'Button Library Example
func main()
  dim stat

  Buttoninit(0)
  Buttoninit(1)
  Buttoninit(2)

loop:

  stat = Button(4,2500,50)
  if stat = 1 then
    print "button 4 pressed"
  endif

  stat = Button(5,2500,50)
  if stat = 1 then
    print "button 5 pressed"
  endif

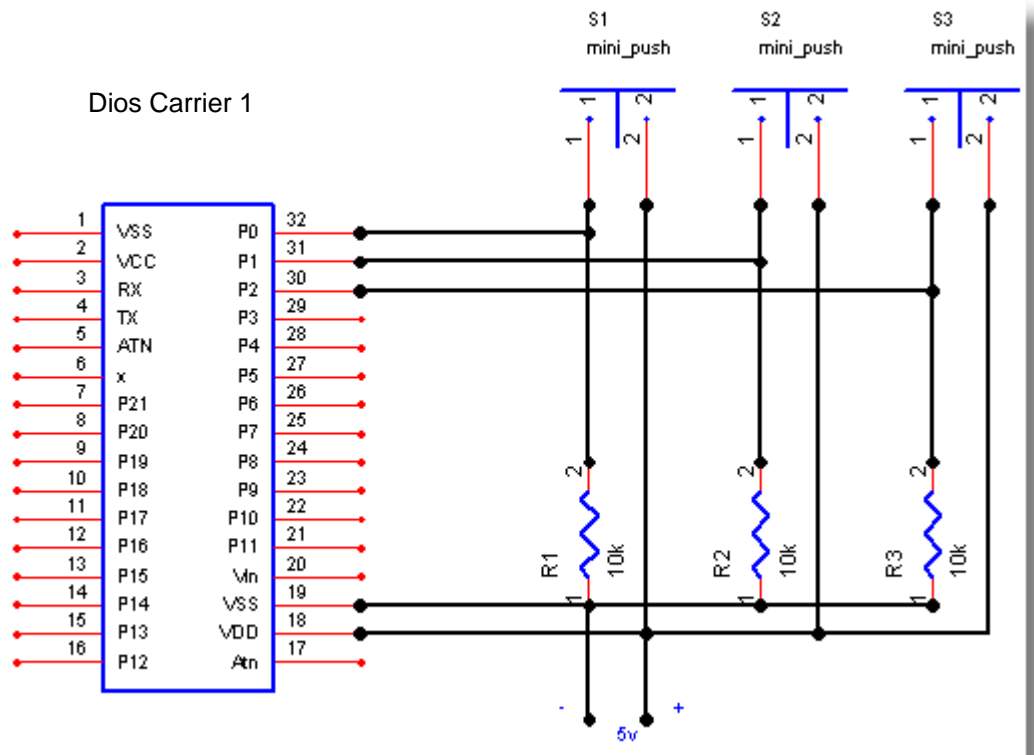
  stat = Button(6,2500,50)
  if stat = 1 then
    print "button 6 pressed"
  endif

  goto loop

endfunc

include \lib\DiosButton.lib
    
```

Schematic



Dios DS1302 Real Time Clock Functions

These functions are used for interfacing to a DS1302 Real Time Clock.

The nicad battery is optional and can be omitted. You will need a 32.768Khz xtal.

DS1302init

DS1302init(CLK,IO,RST)
DS1302init(iexp,iexp,iexp)

Description

This must be called before any of the other routines are called. It sets up the ports that are connected to the clock leads.

- **CLK** - The IO port connected to the clock pin on the DS1302
- **IO** - The IO port connected to the IO pin on the DS1302
- **RST** - The IO port connected to the reset pin on the DS1302

DS1302setdate

DS1302setdate(date)
DS1302setdate(iexp)

Description

Sets the day of the month.

- **date** - The day of the month with a valid range of 1-31.

DS1302setday

DS1302setday(day)
DS1302setday(iexp)

Description

Sets the day of the week.

- **day** - The day of the week with a valid range of 1-7.

DS1302setmonth

DS1302setmonth(month)
DS1302setmonth(iexp)

Description

Sets the month.

- **month** - The number of the current month with a valid range of 1-12.

DS1302setyear

DS1302setyear(year)
DS1302setyear(iexp)

Description

Sets the year

- **year** - The last 2 digits of the year with a valid range of 0-99.

DS1302setsec

DS1302setsec(secs)
DS1302setsec(iexp)

Description

Sets the seconds for the clock.

- **secs** - The number of seconds with a valid range of 0-59.

DS1302setmin

DS1302setmin(mins)
DS1302setmin(iexp)

Description

sets the minutes for the clock.

- **mins** - The number of minutes with a valid range of 0-59.

DS1302sethour

DS1302sethour(hours,am,md)
DS1302sethour(iexp,iexp,iexp)

Description

Sets the hours and hour mode used.

- **hours** - 1-12 for 12 hour mode. 0-23 for 24 hour mode.
- **am** - Set to 0 for AM and 1 for PM. Valid only in 12 hour mode.
- **md** - Sets mode 0 for 12 hour and 1 for 24 hour.

DS1302getdate

DS1302getdate()
DS1302getdate() as integer

Description

Returns the current day of the month with a range of 1-31.

DS1302getday

DS1302getday()
DS1302getday() as integer

Description

Returns the current day of the week with a range of 1-7.

DS1302getmonth

DS1302getmonth()
DS1302getmonth() as integer

Description

Returns the current month with a range of 1-12.

DS1302getyear

DS1302getyear()
DS1302getyear() as integer

Description

Returns the current year with a range of 0-99.

DS1302getsec

DS1302getsec()
DS1302getsec() as integer

Description

Returns the current seconds with a range of 0-59.

DS1302getmin

DS1302getmin()
DS1302getmin() as integer

Description

Returns the current minutes with a range of 0-59.

DS1302gethour

DS1302gethour(mode)
DS1302gethour(iexp) as integer

Description

If mode 0 returns hour 1-12 or 0-23 depending on mode of operation.
if mode 1 returns 0 for am or 1 for pm. Only valid if in 12 hour mode.
if mode 2 returns 0 for 12 hour mode and 1 for 24 hour mode.

DS1302setram

DS1302setram(ram,value)
DS1302setram(iexp,iexp)

Description

Writes a byte to DS1302 internal ram location.

- **Ram** - The location in the DS1302 to write to with a range of 0-30.
- **Value** - This is the value to write with a value of 0-255

DS1302getram

DS1302getram(ram)
DS1302getram(ram) as integer

Description

Reads a byte from a ram location of the clock chip. Returns a value from 0-255.

- **Ram** - The location in DS1302 Ram. The valid range is 0-30.

DS1302setcharger

DS1302setcharger(chval)
DS1302setcharger(iexp)

Description

Sets up the trickle charger.

- **chval** - Sets the charger mode.
 - Bit 0RS1
 - Bit 1RS2
 - Bit 2RS3
 - Bit 3RS4
 - RS = 00 then no Charger
 - RS = 01 then 2K resistor activated
 - RS = 10 then 4K resistor activated
 - RS = 11 then 8K resistor activated
 - DS = 01 then one diode used
 - DS = 11 then two diodes used.
 - Use 11 for the slowest charge possible.

Low level routines

DS1302sendreg

DS1302sendreg(reg,dat)
DS1302sendreg(iexp,iexp)

Description

Sends a value to DS1302 register.

- **reg** - The DS1302 register. See DS1302 Data sheet for value ranges.
- **dat** - The value for the selected register.

DS1302sendbyte

DS1302sendbyte(dat)
DS1302sendbyte(dat)

Description

Writes a byte of data to the DS1302.

- **dat** - The byte to send to the DS1302. The valid range 0-255.

DS1302getbyte

DS1302getbyte()
DS1302getbyte() as integer

Description

Gets a byte of data from DS1302. Reads in BCD format.

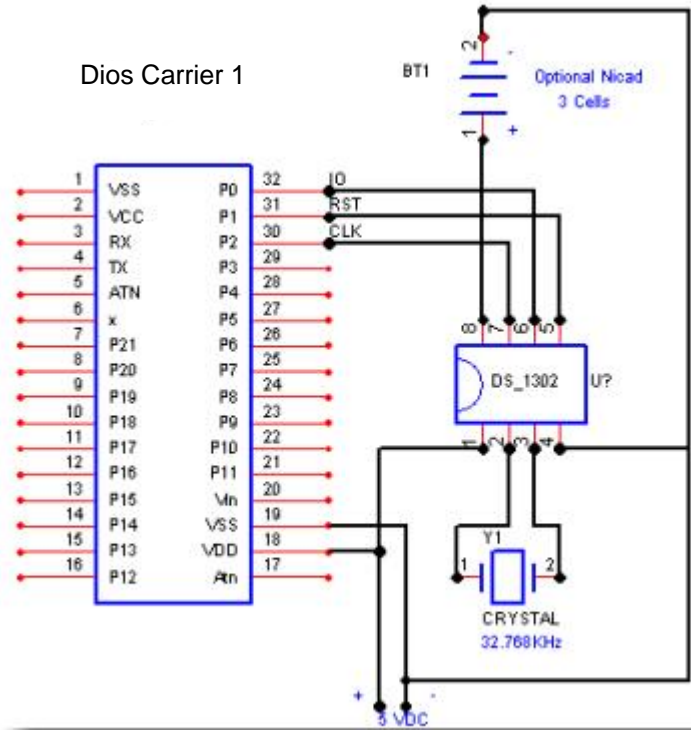
DS1302getrawbyte

DS1302getrawbyte()
DS1302getrawbyte() as integer

Description

Gets a byte in raw format. Returns a value from 0-255.

Schematic



```

DiosPro
'Example of DS1302 Clock Interface
func main()

    dim secs
    dim mins
    dim hours
    dim x

    DS1302init(2,0,1)
    decmask 3+128 'Display last 2 digits with leading 0

    'Set the Time
    DS1302setsec(55)
    DS1302setmin(59)
    DS1302sethour(23,0,1)

    again:
    secs = DS1302getsec()
    mins = DS1302getmin()
    hours = DS1302gethour(0)
    print hours,":",mins,":",secs

    pause 1000
    goto again

endfunc

include \lib\DiosDS1302.lib
    
```

Dios High Speed Servo Functions

These functions are used to setup the 1-4 servos on IO ports 0-3 for background processing.

WARNING: When connecting servos to the Dios make sure power is OFF when they are first connected.

Connect the Servo Power Jumper to VCC or VIN. If using VIN make sure its voltage does not overload your servos.

SERVOinit

SERVOinit(servocount)
SERVOinit(iexp)

Description

This function sets up the background tasks for up to 4 servos on IO ports 0-3. In order to change the servo settings use the SERVO1us, SERVO2us, SERVO3us, and SERVOus functions shown below. The Dios Servo library used the internal Timer1 resource.

- **servocount** - This is the number of servos you wish to start. A count of 0 will turn off the IRQ and timer and shutdown the servo system.

SERVO1us

SERVO2us

SERVO3us

SERVO4us

SERVOXus(value)
SERVOXus(iexp)

Description

This functions change the actual duration of the servo pulse. A setting between 0-2780 will set the timing. 0 turns the servo off.

Note: This library uses timer1 and its IRQ for background processing.

- **value** - The number of microseconds you want the servo pulse to remain high. Normally a value of 1500 will set the servo to its center position.

```
DiosPro
'High Speed Servo Example
func main()
  dim x as integer

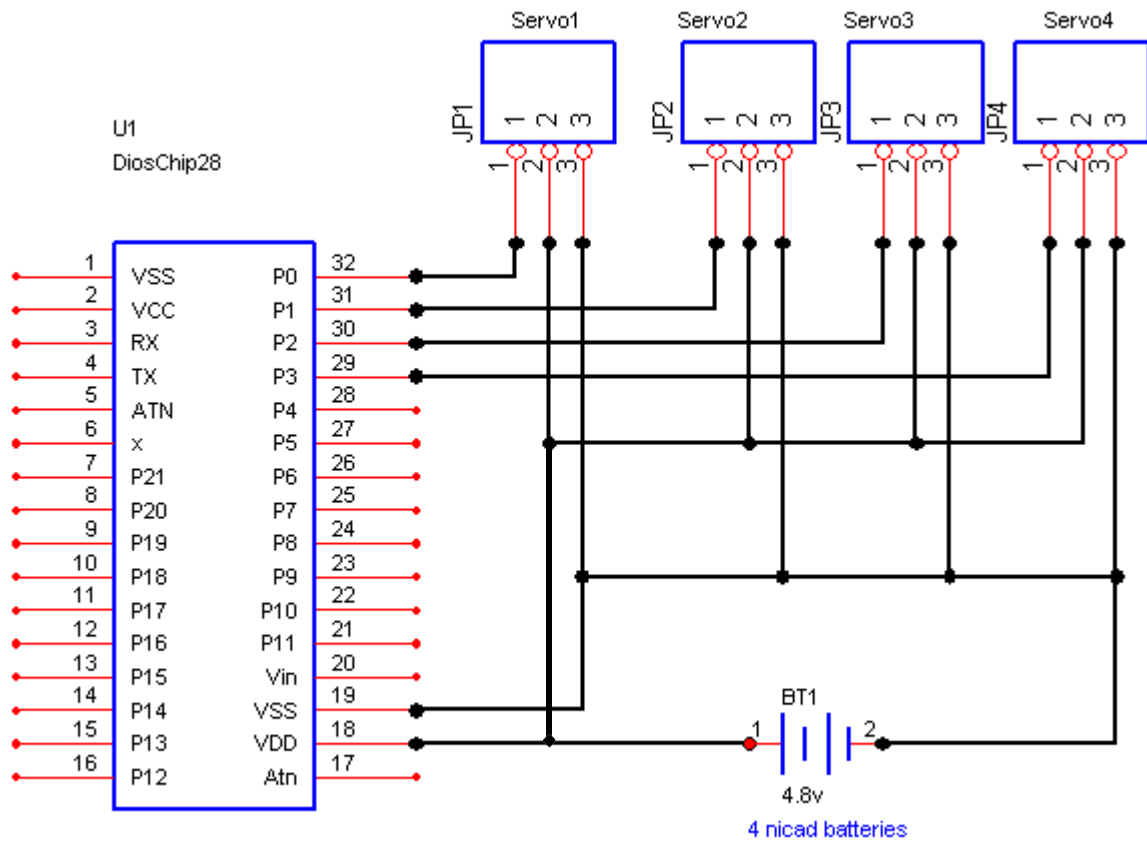
  SERVOinit(2)

loop:
  for x = 100 to 2000
    SERVO1us x
    SERVO2us x
    pauseus 500
  next

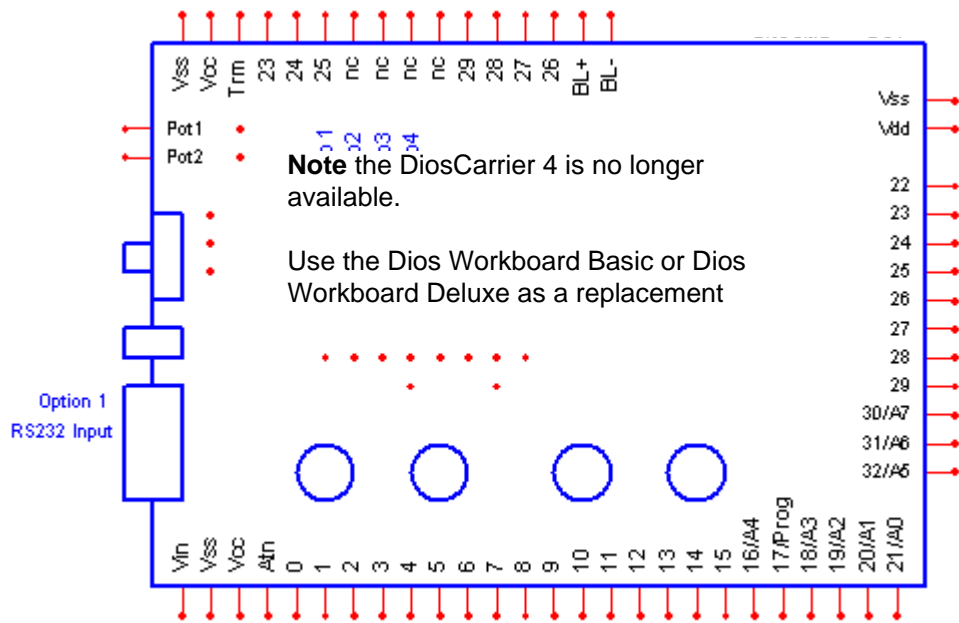
  for x = 2000 to 1000 step -1
    SERVO1us x
    SERVO2us x
    pauseus 500
  next
  goto loop
endfunc

include \lib\DiosHSSERVO.lib
```

This schematic show the Dios Carrier 1



Shows servo connections on a Dios Workboard.



Dios Hardware PWM Functions

These functions are used to setup and control the hardware PWM generators.

There are two PWM generators on the Dios. They operate in the background and require no attention once set up. Both generators share the same period generator so they must have the same period (frequency). They may have different duty cycles.

Channel 1 of the hardware PWM generator is on IO port 13.
Channel 2 of the hardware PWM generator is on IO port 4.

PWMinit

PWMinit(channels)
PWMinit(iexp)

Description

This function sets up the PWM generators. It must be called before any other PWM function.

- **channels** - Sets up 1 or 2 channels. If omitted 1 channel will be utilized. (ioport 13).

PWMperiod

PWMperiod(period)
PWMperiod(iexp)

Description

This function sets the period (frequency) of both PWM channels if enabled.

- **period** - This sets the actual period of the PWM generators. Note that the period is shared between each channel. Only a value of 0-255 is valid. After setting the period you should also set the duty cycle as well. If the duty is greater than or equal to the period the generator will not fire.

PWMcourse

PWMcourse(cperiod)
PWMcourse(iexp)

Description

This function sets up the internal multiplier to extend the range of the PWMperiod function.

- **period** - This sets the range of the PWMgenerator.
 - 0 = x1639.1Khz - 3.34Mhz
 - 1 = x49.8Khz - 834Khz
 - 2 = x12.48Khz - 209Khz

PWM1duty

PWM1duty(duty)
PWM1duty(iexp)

Description

Sets the duty cycle for channel 1.

- **duty** - This is an 8-bit value. It sets a ratio between the period setting. For example, setting a period of 100 and duty of 50 will give you a 50% duty cycle.

PWM2duty

PWM2duty(duty)
 PWM2duty(iexp)

Description

Sets the duty cycle for channel 2.

- **duty** - This is an 8-bit value. It sets a ratio between the period setting. For example, setting a period of 100 and duty of 50 will give you a 50% duty cycle.

Frequency Table

Here is a partial frequency table. Remember to set the duty cycle correctly after setting the period.

Period	Course 2	Course 1	Course 0	Period	Course 2	Course 1	Course 0
255	2.450Khz	9.802Khz	39.212Khz	50	12.301Khz	49.206Khz	196.827Khz
250	2.499Khz	9.998Khz	39.992Khz	49	12.547Khz	50.190Khz	200.761Khz
249	2.509Khz	10.038Khz	40.153Khz	40	15.302Khz	61.208Khz	244.834Khz
240	2.603Khz	10.412Khz	41.653Khz	30	20.238Khz	80.953Khz	323.813Khz
238	2.625Khz	10.5Khz	42Khz	20	29.875Khz	119.502Khz	478.010Khz
232	2.692Khz	10.770Khz	43.082Khz	10	57.035Khz	228.140Khz	912.568Khz
230	2.715Khz	10.864Khz	43.455Khz	9	62.732Khz	250.931Khz	1.003719Mhz
227	2.751Khz	11.006Khz	44.027Khz	8	69.703Khz	278.812Khz	1.115241Mhz
222	2.813Khz	11.253Khz	45.014Khz	7	78.416Khz	313.665Khz	1.254654Mhz
220	2.838Khz	11.355Khz	45.421Khz	6	89.618Khz	358.474Khz	1.433886Mhz
217	2.877Khz	11.511Khz	46.046Khz	5	104.555Khz	418.221Khz	1.672864Mhz
210	2.973Khz	11.892Khz	47.570Khz	4	125.465Khz	501.865Khz	2.007438Mhz
208	3.001Khz	12.007Khz	48.028Khz	3	156.832Khz	627.331Khz	2.509286Mhz
204	3.060Khz	12.241Khz	48.966Khz	2	209.109Khz	836.442Khz	3.345695Mhz
200	3.121Khz	12.484Khz	49.936Khz				
199	3.136Khz	12.547Khz	50.190Khz				
192	3.250Khz	13.002Khz	52.011Khz				
190	3.284Khz	13.137Khz	52.551Khz				
180	3.466Khz	13.864Khz	55.459Khz				
178	3.504Khz	14.019Khz	56.078Khz				
170	3.668Khz	14.674Khz	58.697Khz				
166	3.756Khz	15.027Khz	60.108Khz				
160	3.896Khz	15.585Khz	62.343Khz				
155	4.021Khz	16.086Khz	64.346Khz				
150	4.154Khz	16.619Khz	66.478Khz				
146	4.267Khz	17.071Khz	68.286Khz				
140	4.449Khz	17.798Khz	71.193Khz				
138	4.513Khz	18.054Khz	72.216Khz				
131	4.752Khz	19.011Khz	76.045Khz				
130	4.789Khz	19.156Khz	76.627Khz				
124	5.019Khz	20.076Khz	80.304Khz				
120	5.185Khz	20.740Khz	82.960Khz				
110	5.652Khz	22.608Khz	90.434Khz				
100	6.211Khz	24.847Khz	99.387Khz				
99	6.273Khz	25.095Khz	100.381Khz				
90	6.894Khz	27.577Khz	110.310Khz				
80	7.745Khz	30.982Khz	123.928Khz				
70	8.836Khz	35.345Khz	141.382Khz				
60	10.285Khz	41.140Khz	164.560Khz				

```

DiosPro
'PWM Example
func main()
    dim x
    PWMinit(2)

loop:
    for x = 10 to 200
        PWMperiod(x)
        PWM1duty(x.byte(0)/2)
        pause 20
    next

    for x = 200 to 10 step -1
        PWMperiod(x)
        PWM1duty(x.byte(0)/2)
        pause 20
    next

    goto loop

endfunc

include \lib\DiosHWPWM.lib
    
```

Dios I2c Functions

These functions are used to access I2c slave devices. Both low level and high level functions are provided for maximum versatility.

The hookup for each I2c device will vary slightly but will consist of a 2-wire connection from the Dios to the slave device. These two wires are the SDA line and the SCL line. In each function below the SDA and SCL lines are passed.

In the following schematic and test program we will use a 24LC64 EEprom.

High Level Functions

I2cin

I2cin(sda,scl,slaveaddr,address)
I2cin(iexp,iexp,iexp,iexp) as integer

Description

Returns a byte of data from a I2c slave device. This is for use with slave devices that require a single address byte.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **slaveaddr** - Control code for slave device. The read/write bit is set automatically for you.
- **address** - The address to read from. This is a single byte.

I2cout

I2cout(sda,scl,slaveaddr,address,value)
I2cin(iexp,iexp,iexp,iexp,iexp)

Description

Sends a byte of data to a I2c slave device. This is for use with slave devices that require a single byte address.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **slaveaddr** - Control code for slave device. The read/write bit is set automatically for you.
- **address** - The address to write to. This is a single byte.
- **value** - The byte to send to the device

I2cin2

I2cin2(sda,scl,slaveaddr,address)
I2cin2(iexp,iexp,iexp,iexp) as integer

Description

Returns a byte of data from a I2c slave device. This is for use with slave devices that require a 2 byte address.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **slaveaddr** - Control code for slave device. The read/write bit is set automatically for you.
- **address** - The address (word) to read from.

I2cout2

I2cout2(sda,scl,slaveaddr,address,value)

I2cout2(iexp,iexp,iexp,iexp)

Description

Sends a byte of data to a I2c slave device. This is for use with slave devices that require a 2 byte address.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **slaveaddr** - Control code for slave device. The read/write bit is set automatically for you.
- **address** - The address (word) to write to.
- **value** - The byte to send to the device.

Variable Level Functions

I2cwriteint

I2cwriteint(sda,scl,baseaddress,index,intvalue)

I2cwriteint(iexp,iexp,iexp,iexp)

Description

Sends an integer (16-bit) to an EEPROM. Supports 4k, 8k, 16k, 32k serial EEPROM Devices.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **baseaddress** - This is the starting point for address calculation. This will allow you to create blocks of memory for different storage types.
- **index** - This is the index of the item to be written. Look at this like a tag. It is used to calculate the raw address for storage. When you write a value using index 21 you retrieve that value using an index of 21.
- **intvalue** - This is the actual 16-bit value to write. It is an expression and can contain numbers and variables.

Example

I2cwriteint(0,1,1000,21,1435)

I2creadint

I2creadint(sda,scl,baseaddress,index)
I2creadint(iexp,iexp,iexp,iexp) as integer

Description

Returns an integer (16-bit) from an EEPROM. Supports 4k, 8k, 16k, 32k serial EEPROM Devices.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **baseaddress** - This is the starting point for address calculation. This will allow you to create blocks of memory for different storage types.
- **index** - This is the index of the item to be read. Look at this like a tag. It is used to calculate the raw address for storage. When you write a value using index 21 you retrieve that value using an index of 21.

Example

data=I2cread(0,1,1000,21)

I2cwritefloat

I2cwritefloat(sda,scl,baseaddress,index,floatvalue)
I2cwritefloat(iexp,iexp,iexp,iexp,fexp)

Description

Sends a float (32 bit) value to an EEPROM. Supports 4k, 8k, 16k, 32k serial EEPROM Devices.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **baseaddress** - This is the starting point for address calculation. This will allow you to create blocks of memory for different storage types.
- **index** - This is the index of the item to be written. Look at this like a tag. It is used to calculate the raw address for storage. When you write a value using index 21 you retrieve that value using an index of 21.
- **floatvalue** - This is the actual 32-bit float value to write. It is an expression and can contain numbers and variables.

Example

I2cwritefloat(0,1,1000,21,110.23)

I2creadfloat

I2creadfloat(sda,scl,baseaddress,index)
I2creadfloat(iexp,iexp,iexp,iexp) as float

Description

Returns a 32-bit floating point value from an EEPROM. Supports 4k, 8k, 16k, 32k serial EEPROM Devices.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **baseaddress** - This is the starting point for address calculation. This will allow you to create blocks of memory for different storage types.
- **index** - This is the index of the item to be read. Look at this like a tag. It is used to calculate the raw address for storage. When you write a value using index 21 you retrieve that value using an index of 21.

Example

```
data=I2cread(0,1,1000,21)
```

I2cwritestring

```
I2cwritestring(sda,scl,baseaddress,index,stringsize,stringvalue)  
I2cwritestring(iexp,iexp,iexp,iexp,iexp,string)
```

Description

Sends a string to an EEPROM. Supports 4k, 8k, 16k, 32k serial EEPROM Devices. You can write strings, text or tables.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **baseaddress** - This is the starting point for address calculation. This will allow you to create blocks of memory for different storage types.
- **index** - This is the index of the item to be written. Look at this like a tag. It is used to calculate the raw address for storage. When you write a value using index 21 you retrieve that value using an index of 21.
- **stringsize** - This is used for calculating the address of the item. All strings written should use the same size. Normally this is the size of your largest string + 1.
- **stringvalue** - This is the actual string to write. It may be a literal string(Quoted), table or string variable.

Example

```
I2cwritestring(0,1,1000,21,50,"Now is the time")
```

I2creadstring

```
I2creadstring(sda,scl,baseaddress,index,stringsize,stringvariable)  
I2creadstring(iexp,iexp,iexp,iexp,iexp,stringvarb)
```

Description

Populates a string variable with a string stored in EEPROM. Supports 4k, 8k, 16k, 32k serial EEPROM Devices.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **baseaddress** - This is the starting point for address calculation. This will allow you to create blocks of memory for different storage types.

- **index** - This is the index of the item to be written. Look at this like a tag. It is used to calculate the raw address for storage. When you write a value using index 21 you retrieve that value using an index of 21.
- **stringsize** - This is used for calculating the address of the item. All strings written should use the same size. Normally this is the size of your largest string + 1.
- **stringvariabl** - This is the string you want to place the retrieved data into. It must be a string variable. Make sure it is large enough to hold the data you wish to retrieve.

Example

```
I2creadstring(0,1,1000,21,50,mystring)
```

Low Level Functions

I2c_sendbyte

```
I2c_sendbyte(sda,scl,value)  
I2c_sendbyte(iexp,iexp,iexp)
```

Description

Sends a byte of data to the I2c Slave Device.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).
- **value** - The byte to send to the I2c slave device.

I2c_getbyte

```
I2c_getbyte(sda,scl)  
I2c_getbyte(iexp,iexp) as integer
```

Description

Returns a byte from the I2c Slave Device.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).

I2c_getack

```
I2c_getack(sda,scl)  
I2c_getack(iexp,iexp) as integer
```

Description

Returns the ACK bit from the I2c device. 1=NACK 0=ACK

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).

I2c_sendack

I2c_sendack(sda,scl)
I2c_sendack(iexp,iexp)

Description

Sends an acknowledge bit to the I2c Slave device.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).

I2c_sendnack

I2c_sendnack(sda,scl)
I2c_sendnack(iexp,iexp)

Description

Sends an negative acknowledge bit to the I2c Slave device.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).

I2c_start

I2c_start(sda,scl)
I2c_start(iexp,iexp)

Description

Sends a start condition to the I2c Slave device.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).

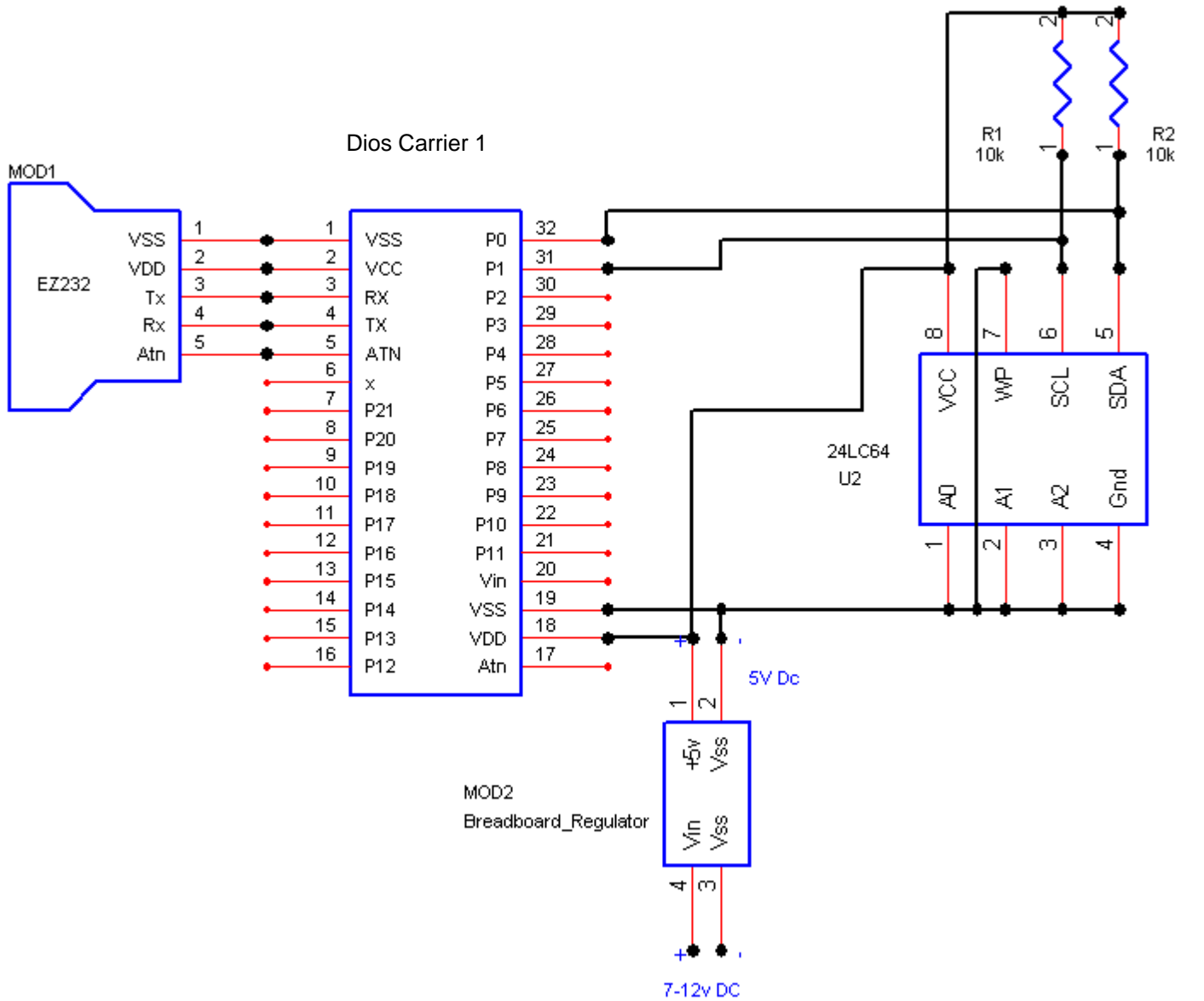
I2c_stop

I2c_stop(sda,scl)
I2c_stop(iexp,iexp)

Description

Sends a stop condition to the I2c slave device.

- **sda** - Dios IO port connected to the slave device data pin (sda).
- **scl** - Dios IO port connected to the slave device clock pin (scl).



```

DiosPro
'Simple Example of I2c Routines
'Writes data to EEprom
func main()
  dim dat,x
  const sda 0
  const scl 1

  print "Write"

  for x = 1000 to 1100
    I2cout2(sda,scl,160,x,x-1000)
    pause 1
  next

  print "Read"
  for x = 1000 to 1100
    dat=I2cin2(sda,scl,160,x)
    print x , " ",dat
  next

  print "all done"

endfunc
include \lib\DiosI2c.lib

```

```

DiosPro
'Example of varable read and write to external EEprom
func main()
  dim dat,x
  global tstring(16) as string

  const sda 0
  const scl 1
  const intblock 1000
  const stringblock 2000
  const slen 15

  'Write Integers
  I2cwriteint(sda,scl,intblock,1,12345)
  I2cwriteint(sda,scl,intblock,2,45678)

  'Write Strings
  I2cwritestring(sda,scl,stringblock,1,slen,"Blue")
  I2cwritestring(sda,scl,stringblock,2,slen,"Yellow")
  I2cwritestring(sda,scl,stringblock,3,slen,"Green")
  I2cwritestring(sda,scl,stringblock,4,slen,"Red")
  I2cwritestring(sda,scl,stringblock,5,slen,"Black")

  'Read and print integers
  dat = I2creadint(sda,scl,intblock,1)
  print "Read from pos 1 :",dat

  dat = I2creadint(sda,scl,intblock,2)
  print "Read from pos 2 :",dat

  'Read and print strings

  for x = 1 to 5
    I2creadstring(sda,scl,stringblock,x,slen,tstring)
    print "Read string from pos ",x," :",tstring
  next

endfunc

include \lib\DiosI2c.lib

```

Dios IR Module Functions

These functions are used for interfacing to a IR module using a Sony remote.

IRread

IRread(port,timeout)
IRread(iexp[,iexp]) as integer

Description

This function returns the Sony IR code + 1 from the IR module. If a timeout occurs a zero will be returned. If you need to access the raw values you can use the global variables IRcmd and IRdevice. These variables are loaded with each call.

- **port** - The port that the IR module is connected to.
- **timeout** - The time in micro seconds (approx) that the function will look for a valid IR start bit. This is an optional value and if omitted a value of 5000 will be used.

IRreadverify

IRreadverify(port,devicecode,timeout)
IRreadverify(iexp,iexp,iexp) as integer

Description

This function works the same as IRread except the device code must match before it returns a valid number. Note that it can still timeout. This function was designed for noisy situations or where multiple IR signals may be present.

- **port** - The port that the IR module is connected to.
- **devicecode** - The Sony device code that must be returned from the module.
- **timeout** - The time in micro seconds (approx) that the function will look for a valid IR start bit. This is an optional value and if omitted a value of 5000 will be used.

IRkeypad

IRkeypad(port,device,ledport)
IRkeypad(iexp,iexp[,iexp]) as integer

Description

Returns a number from the IR keypad. The number can be in the range of 0-65535. With this function you can enter keypad numbers then hit the enter on the IR remote. Once enter is hit the actual number entered is returned.

- **port** - The port that the IR module is connected to.
- **devicecode** - The Sony device code that must be returned from the module.
- **ledport** - The user feedback port. This field is optional. If included you can place a buzzer or LED to let the user know when they are pressing a key.

IRinitsend

IRinitsend()
IRinitsend()

Description

Sets up the hardware PWM on IO port 13 for 40Khz. This will be used as the carrier for the IR transmitter. This must be called prior to calling IRsendcode. It need only be called once.

IRsendcode

IRsendcode(device,cmd)
IRsendcode(iexp,iexp)

Description

This function sends the device and command code out IO port 13 in SONY format. Its a good practice to send the command several times to insure the remote device gets the code.

Note that at pause of at least 25 milliseconds must be placed in between commands or most IR modules will lock out the signal.

- **device** - This is the SONY device code. 5 bits are used so only values of 0-31 are valid.
- **cmd** - This is the SONYcommand code. 7 bits are used so only values of 0-127 are valid.

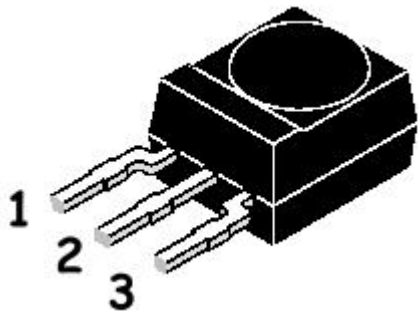
```
DiosPro
'IRsendcode Example
'Will case device to change channels
func main()

  IRinitsend()

  loop:

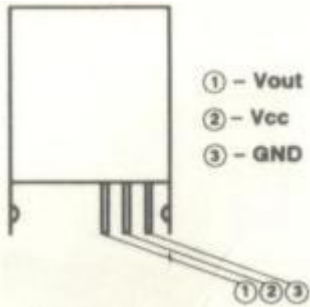
    IRsendcode(11,16)
    pause 50
    goto loop
  endfunc

include \lib\DiosIR.lib
```



The Vishay IR module has the following pin layout:

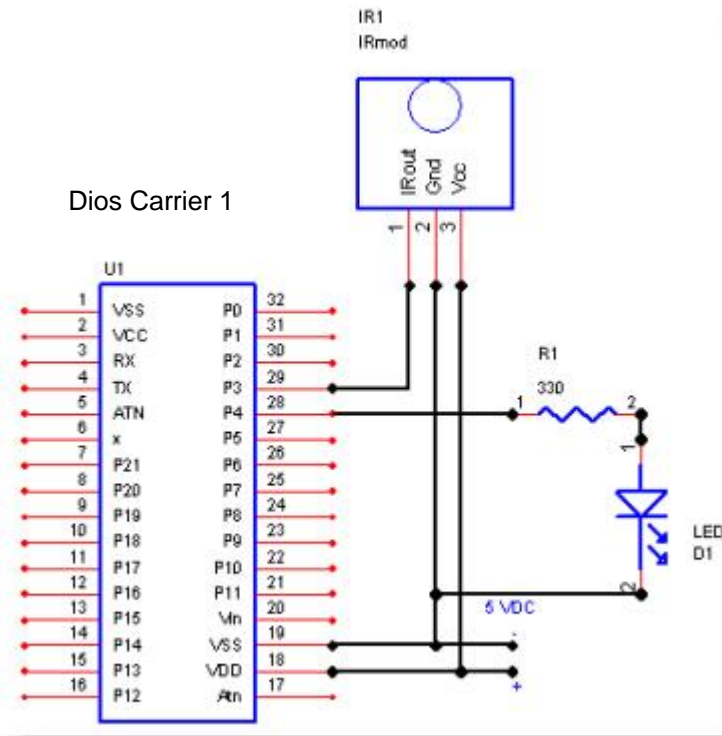
- 1 - IR out
- 2 - Gnd
- 3 - Vcc



The Sharp IR module has the following pin layout:

- 1 - IRout
- 2 - Vcc
- 3 - Gnd

Schematic



```

DiosPro
'IRread Example
func main()
  dim cmd,device

loop:
  cmd = IRreadverify(3,11,1000)
  if cmd =0 then
    goto loop
  endif

  print
  print IRdevice," ",cmd
  goto loop

endfunc

include \\lib\DiosIR.lib
    
```

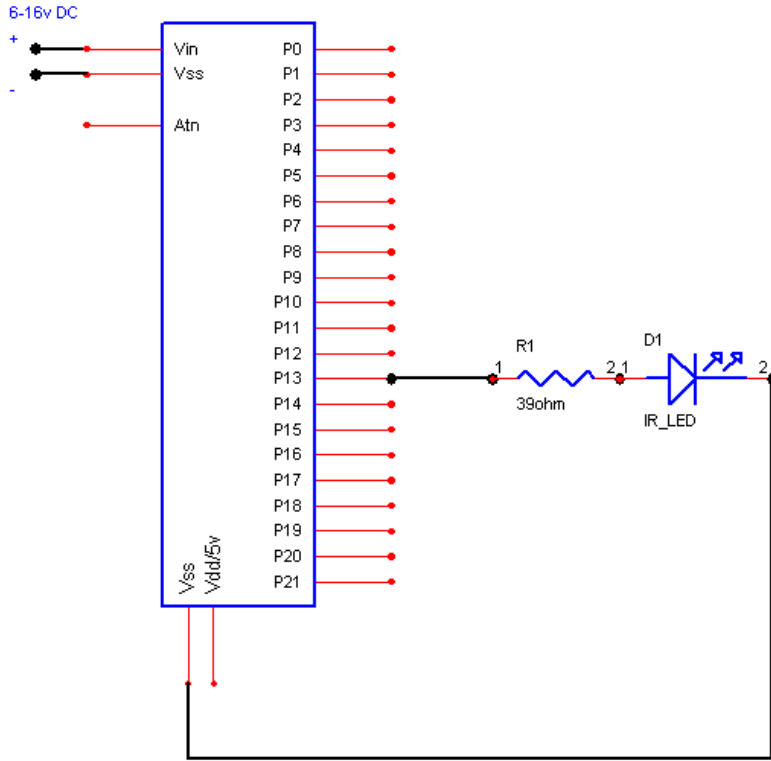
```

DiosPro
'IR Keypad Example
func main()
  dim number

loop:
  number=IRkeypad(3,11,4)
  print
  print number
  goto loop

endfunc
    
```

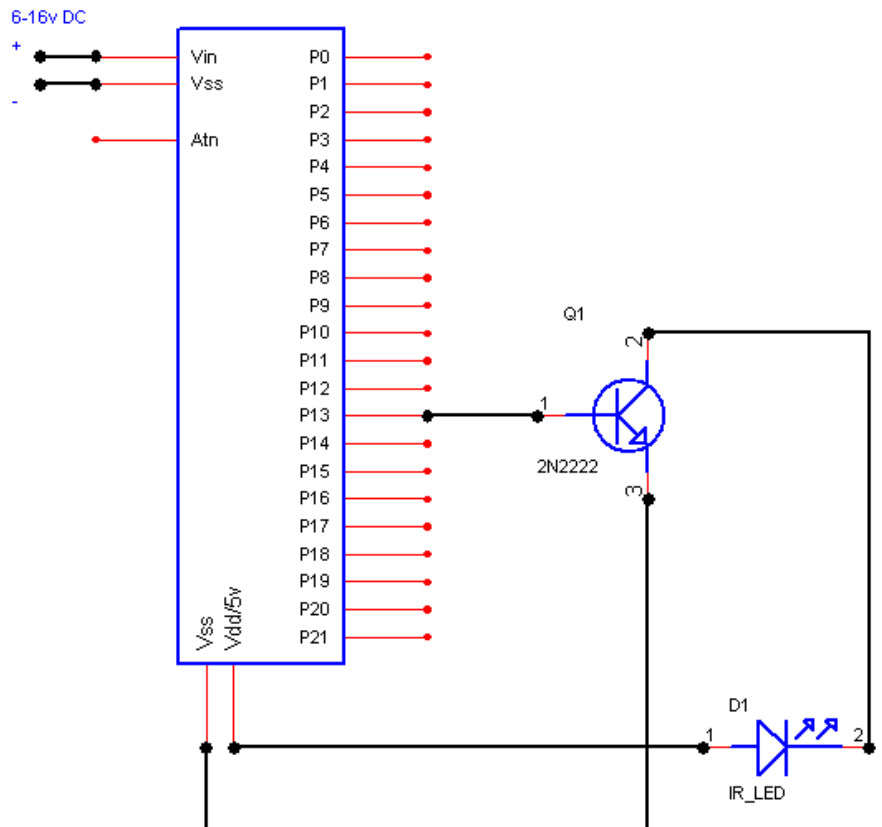
Dios Workboard



This is a simple IR transmitter circuit. It will work but the range will only work up to 10 feet or so.

Dios Workboard

In this circuit we add a small 2N2222 transistor and gain enough power to bounce the IR signal off any wall in the house.



Dios LVD Functions

These functions are used to monitor the low voltage detection circuitry built into the Dios. You can install and set up this library and forget it. When the LVD circuitry detects a low voltage situation it will fire and interrupt. At that point you can act.

The LVD circuitry is built in. Just use the Dios as you would with any other application.

LVDinit

LVDinit(level)

LVDinit(iexp)

Description

Sets the low voltage detection circuitry and interrupt.

- **level** - Number that represents the reference voltage that cause the LVD to fire.

Valid levels:

1 2.0v - 2.12v

2 2.2v - 2.33v

3 2.4v - 2.54v

4 2.5v - 2.65v

5 2.7v - 2.86v

6 2.8v - 2.97v

7 3.0v - 3.18v

8 3.3v - 3.5v

9 3.5v 3.71v

10 3.6v - 3.82v

11 3.82v - 4.03v

12 4.0v - 4.24v

13 4.2v - 4.45v

14 4.5v - 4.77v

LVDread

LVDread()

LVDread() as integer

Description

Check to see if the LVD circuitry has fired.

Returns 0 if no low voltage. Returns 1 if low voltage detected.

```
DiosPro
'LVD function Example
func main()
  dim a

  print "restart"

  LVDinit(14)

loop:
  a = LVDread()

  if a > 0 then
    print "Low Power"
    LVDinit(14)
  endif

  goto loop

endfunc

include \lib\DiosLVD.lib
```

Dios Math Functions

These are Dios Math support routines. They support both integer and floating point arguments. They all return float values. This value will be converted to integer automatically if needed.

MATHabs

MATHabs(X)

Returns the absolute value of expression X.

MATHcos

MATHcos(X)

Returns cosine of X degrees

MATHcot

MATHcot(X)

Returns cotangent of X degrees

MATHcsc

MATHcsc(X)

Returns the cosecant of X degrees.

MATHfactor

MATHfactor(X)

Returns the factorial of X.

MATHgcd

MATHgcd(X,Y)

Returns greatest common denominator of X and Y.

MATHmod

MATHmod(X)

Returns mod of X.

MATHradians

MATHradians(X)

Returns radians from X degrees

MATHsec

MATHsec(X)

Returns secant of X degrees.

MATHsin

MATHsin(X)

Returns sine of X degrees.

MATHsqr

MATHsqr(X)

Returns X squared.

MATHsqrt

MATHsqrt(X)

Returns square root of X.

MATHtan

MATHtan(X)

Returns tangent of X degrees.

MATHxrt

MATHxrt(Y,X)

Returns X root of Y.

MATHy2x

MATHy2x(Y,X)

Returns Y to the X power

Dios MAX522 DAC Functions

These functions are used to interface with a MAXM MAX522 DAC.

MAX522init

MAX522init(CS,CLK,DATA)
MAX522init(*iexp,iexp,iexp*)

Description

This function sets up the IO ports to be connected to the MAX522 DAC. It must be called before any others.

- **CS** - Dios IO port used to connect to CS (pin 1) in the MAX522 Chip.
- **CLK** - Dios IO port used to connect to CLK (pin 2) in the MAX522 Chip.
- **DATA** - Dios IO port used to connect to DATA (pin 8) in the MAX522 Chip.

MAX522set1

MAX522set1(VALUE)
MAX522set1(*iexp*)

Description

This sets the voltage level on output 1 (pin 5).

- **VALUE** - The digital value 0-255 that is converted to a voltage in the range of 0-5v. Hence each unit is equal to .0196 volts.

MAX522set2

MAX522set2(VALUE)
MAX522set2(*iexp*)

Description

This sets the voltage level on output 3 (pin 6).

- **VALUE** - The digital value 0-255 that is converted to a voltage in the range of 0-5v. Hence each unit is equal to .0196 volts.

```

DiosPro
'Creates a triangle wave
func main()
  dim x

  MAX522init(0,1,2) 'CS CLK DAT

again:
  for x = 0 to 255 step 10
    MAX522set1(x)
    pauseus 100
  next

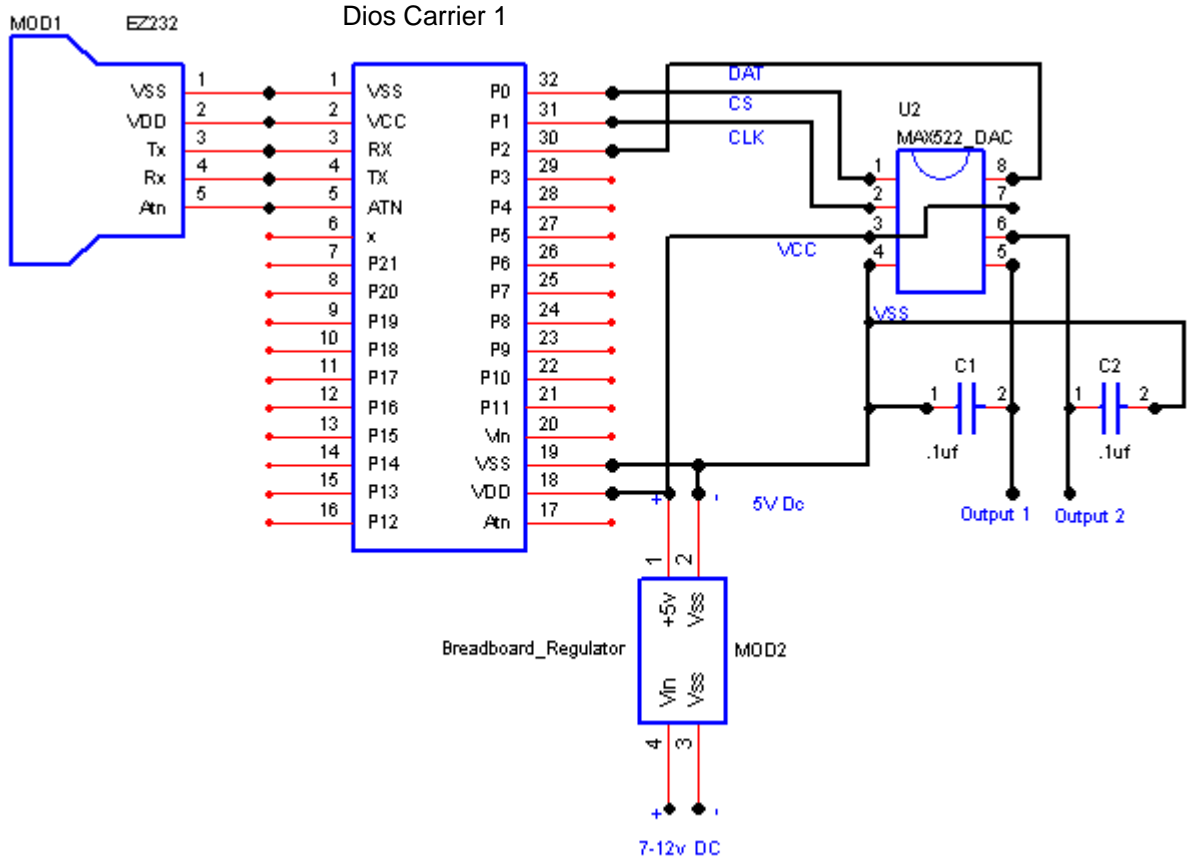
  for x = 255 to 0 step -10
    MAX522set1(x)
    pauseus 100
  next

  goto again

endfunc

include \lib\DiosMAX522.lib
    
```

Schematic



Dios MicroChip Digital Potentiometer Functions

These functions are used to interface a MicroChip Digital Potentiometer. Currently compatible with the following chips.

MCP42010 Dual 10K
MCP42050 Dual 50K
MCP42100 Dual 100K

MCP41010 Single 10K
MCP41050 Single 50K
MCP41100 Single 100K

MCPset

MCPset(cs,sck,si,unit,value)
MCPset(iexp,iexp,iexp,iexp)

Description

Sets a potentiometer to a particular value.

- **cs** - Dios IOport connected to cs (chip select) pin on the potentiometer.
- **sck** - Dios IOport connected to sck (clock) pin on the potentiometer.
- **si** - Dios IOport connected to si (serial input) pin on the potentiometer.
- **unit** - Selects the digital potentiometer to write to. 0=pot0 1=pot1 2=both. Use 0 if MCP41xxx series.
- **value** - This is the value to write. It must be in the range of 0-255. This value will set the pot to the corresponding range within its value swing.

MCPshutdown

MCPshutdown(cs,sck,si,unit)
MCPshutdown(iexp,iexp,iexp,iexp)

Description

Shuts down a potentiometer. Writing to the potentiometer will wake it up. Note that when shut down the resistors go to a high state.

- **cs** - Dios IOport connected to cs (chip select) pin on the potentiometer.
- **sck** - Dios IOport connected to sck (clock) pin on the potentiometer.
- **si** - Dios IOport connected to si (serial input) pin on the potentiometer.
- **unit** - Selects the digital pot to write to. 0=pot0 1=pot1 2=both. Use 0 if MCP41xxx series.

```

DiosPro
'Digital Pot Example
func main()
  dim x as integer

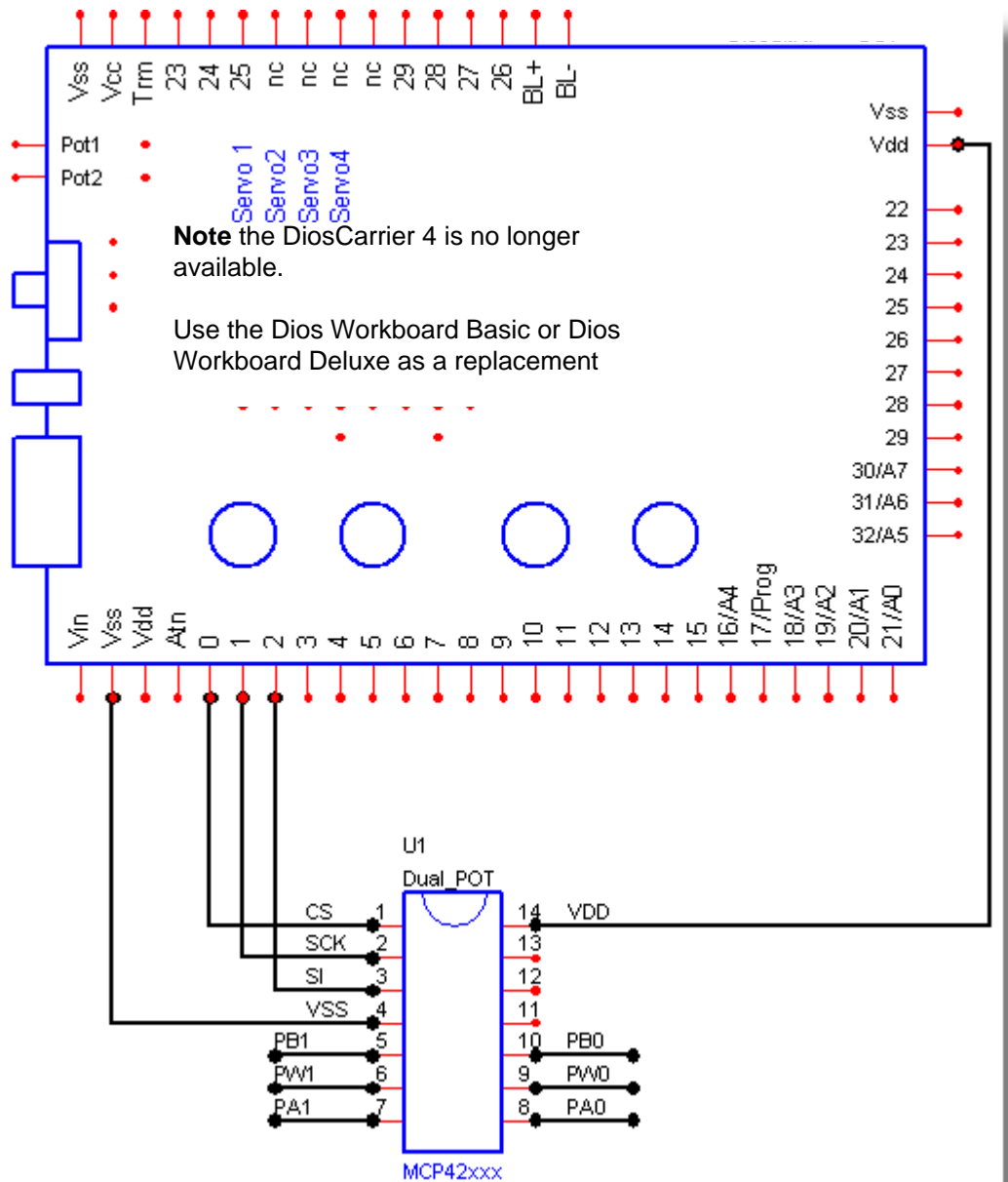
  const cs 0
  const sck 1
  const si 2

  for x = 0 to 255
    MCPset(cs,sck,si,2,x) ' Both Pots
    pause 100
  next

endfunc

include \lib\DiosMCPot.lib
    
```

Schematic



Dios Random Functions

These functions will allow you to generate pseudo random numbers. You don't always have to call the randomize function. A particular seed number will always yield the same results. If you don't power of the Dios each time you call the random function you will get a different number. Even if the Dios is reset.

randomize

randomize(seed)
randomize(iexp)

Description

Seeds the random number generator. If you don't call this routine a seed will be inserted for you.

- seed - A number between 1 and 65535.

random

random(maxval)
random([iexp]) as integer

Description

returns a number between 1 and maxval. Will return the next number in the random number sequence.

- **maxval** - The maximum number you wish to receive between 1 and 65535. If omitted, It will default to 65535. This is an optional value.

```
DiosPro
'Random Functions
func main()
  dim rnum
  dim x
  'randomize(3000) 'Give it a seed number

  for x = 1 to 100
    rnum = random(6)
    print x,": ",rnum
  next

endfunc

include \lib\DiosRandom.lib
```

Dios RSKeypad Functions

These are Dios functions for interfacing to the Radio Shack 270-215 Keypad.

RSKeypadinit

RSKeypadinit()

Description

This function must be the first keypad routine called. It sets up the ports. All ioports 0-7 are used. These are hardcoded because of the pullup resistors they contain which makes hookup very easy.

RSKeypadread

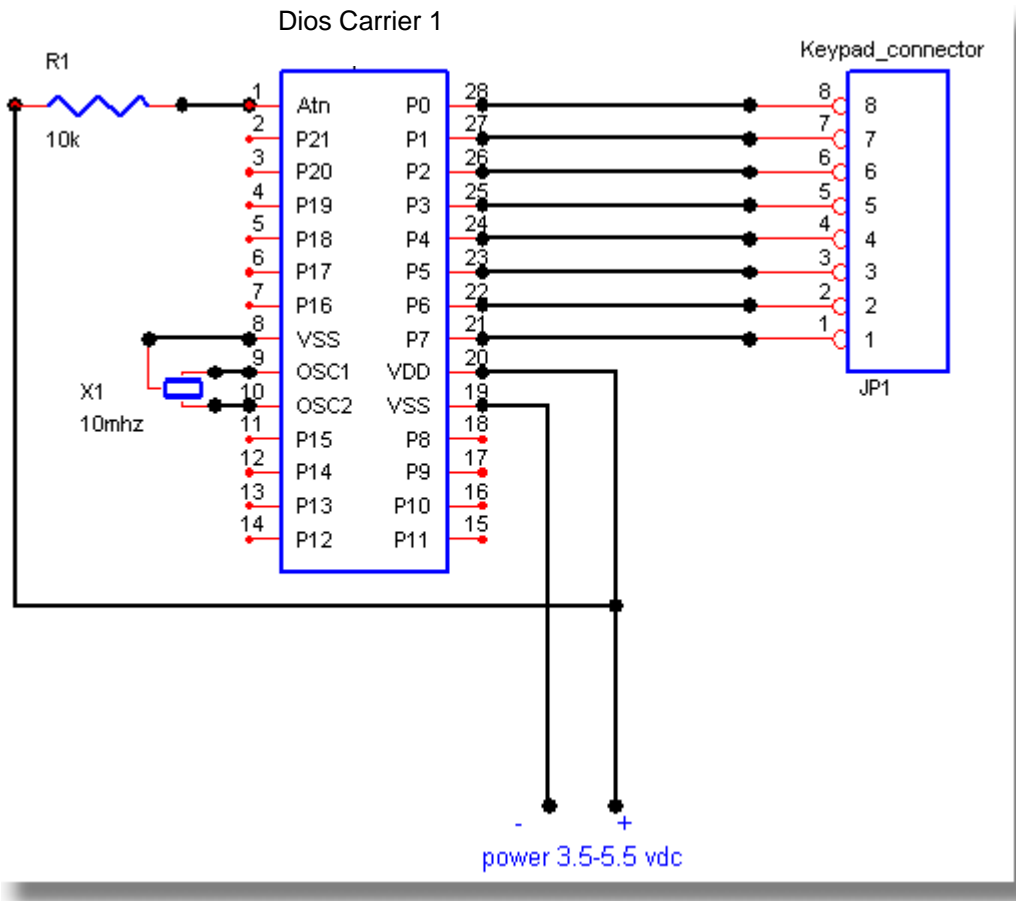
RSKeypadread()

RSKeypadread() as integer

Description

This command reads the keypad and returns the number of the key pressed. If no key is pressed 0 is returned.

Schematic



```

DiosPro
'RSkeypad Example
func main()

  dim key
  RSkeypadinit()

loop:
  key=RSkeypadread()
  if key <> 0 then
    print key
  endif
  goto loop

endfunc

include \lib\DiosRSkeypad.lib
    
```

Dios String Functions

These functions are used to display and manipulate strings. It makes extensive use of string addresses.

What is a string address?

A string address is the starting address of a string, table or passed text. It can point to any of these. If the value is between 0 and 255 then it is pointing to a string. If it is greater than 16385 then it is pointing to a table or passed text. The building Dios commands and the following library functions will automatically determine which type of address is passed.

When passing text to a function you cannot pass text greater than 255 in length. If you need to pass more, then define a table and pass its address. There is no limit to table sizes.

STRgetword

STRgetword(string,word,token,targetstring)
STRgetword(string,iexp,string,stringvarb)

Description

This function will help you divide up strings into logical pieces called words. Returns 1000 if end of string is reached.

- **string** - The string, table or text you wish to parse.
- **word** - The number of the word you wish to pull out. The numbers are 1-n.
- **token** - This is a string containing the tokens to look for in the string. You may include one or more tokens.
- **targetstring** - This is where you want to place the result. Make sure the target has enough room to hold the result.

STRgetwordaddr

STRgetwordaddr(string,word,token,terminator)
STRgetwordaddr(string,iexp,string,string)

Description

This function is used to return the address of a word found in string. It does not return a string but a address. You can use this function to pull and display words or phrases out of larger tables and strings.

- **string** - The string, table or text you wish to parse.
- **word** - The number of the word you wish to pull out. The numbers are 1-n.
- **token** - This is a string containing the tokens to look for in the string. You may include one or more tokens.
- **terminator** - This optional string argument is used to tell the parser you are at the end of a given string. By default, the character 0 (null) is a terminator, but you can add others. This will allow you to place multiple strings together for parsing.

STRprinttext

STRprinttext(string,terminator)
STRprinttext(string,string)

Description

This function will allow you to print strings, tables or text to the debug window. It will allow you to use the optional terminator to indicate the end of a string.

- **string** - The string, table or text you wish to print.
- **terminator** - This optional string argument is used to tell the parser you are at the end of a given string. By default, the character 0 (null) is a terminator, but you can add others. This will allow you to place multiple strings together for parsing.

STRinstr

STRinstr(string1,string2,terminator)
STRinstr(string,string,string) as integer

Description

This function is used to locate string2 inside string1. It returns the address if found. If not found it returns 1000.

- **string1** - The string, table or text you wish to search for.
- **string2** - The string, table or text you are looking for.
- **terminator** - This optional string argument is used to tell the parser you are at the end of a given string. By default, the character 0 (null) is a terminator, but you can add others. This will allow you to place multiple strings together for parsing.

STRlen

STRlen(string,terminator)
STRlen(string,string) as integer

Description

Returns the length of a string. The characters will be counted up to the null character or until a termination character is reached.

- **string** - The string, table or text you wish to check the length of.
- **terminator** - This optional string argument is used to tell the parser you are at the end of a given string. By default, the character 0 (null) is a terminator, but you can add others. This will allow you to place multiple strings together for parsing.

STRval

STRval(string,terminator)
STRval(string,string) as integer

Description

This function will take a number represented in string form and convert it to an actual 16-bit integer and return it.

- **string** - The string, table or text you wish to convert.
- **terminator** - This optional string argument is used to tell the parser you are at the end of a given string. By default, the character 0 (null) is a terminator, but you can add others. This will allow you to place multiple strings together for parsing.

```
DiosPro
'Example of the STRgetword
func main()
  global b(20) as string
  dim x
  dim stat

  for x = 1 to 5
    stat=STRgetword("Now is the time",x,' ',b)
    print ">",b,"< ",stat
  next

endfunc

include \lib\DiosString.lib
```

Notes

Dios Tone Functions

These functions are used to create audible tones and musical notes. Note that this library uses floating point math. The floating point module will be loaded automatically.

TONEnote

TONEnote(IOport,Notes)
 TONEnote(*iexp,string*)

Description

Plays a string of musical notes. Use a space for rest. There are 6 octaves; you start out in the 4th. Up to 255 commands can be issued.

- **IOport** - The Dios IOport you wish to play the note on.
- **Notes** - This is a string of commands for playing notes. The following commands are supported:

CDEFGAB Normal notes

cdfga Sharp notes

1 All following notes will be 1 second long

2 All following notes will be 1/2 second long

4 All following notes will be 1/4 second long

8 All following notes will be 1/8 second long

16 All following notes will be 1/16 second long

^ All following commands will move up one octave

~ All following commands will move down one octave

| This is a divider. It is ignored.

TONefreqout

TONefreqout(IOport,duration,freq1,freq2.....)
 TONefreqout(*iexp,iexp,iexp,iexp.....*)

Description

Play multiple tones one after another. Up to 20 tones are supported.

- **IOport** - The Dios IOport you wish to play the tone on.
- **duration** - This is the duration of the tone in milliseconds. 0 to 6000 is supported with frequencies up to 5,000Hz. 0 to 1000 is supported with frequencies up to 10,000Hz.
- **freq** - This the frequency of the tone. It can be a fraction. Frequencies up to 10,000Hz are supported. Note that up to 20 frequencies are supported with a single command.

TONE

TONE(IOport,duration,freq)
 TONE(*iexp,iexp,iexp*)

Description

Will play a frequency for a given duration.

- **IOport** - The Dios IOport you wish to play the tone on.
- **duration** - This is the duration of the tone in milliseconds. 0 to 6000 is supported with frequencies up to 5,000Hz. 0 to 1000 is supported with frequencies up to 10,000Hz.
- **freq** - This the frequency of the tone. It can be a fraction. Frequencies up to 10,000Hz are supported.

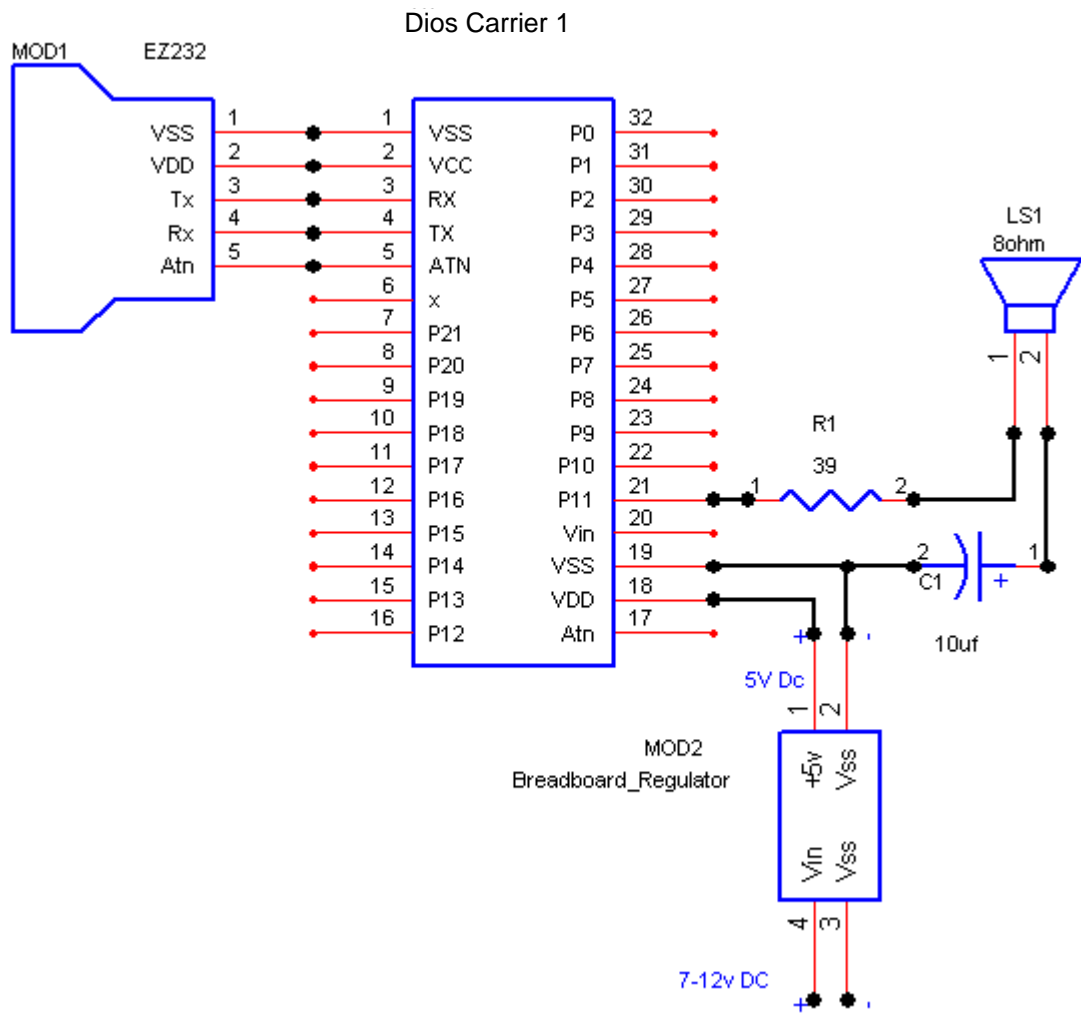
```

DiosPro
'Tone Example
func main()
  TONEnote(12,"4BAGA|BB2B4|AA2A4|B^DD ~|4BAGA|BB2B4|AABA|2G")
endfunc

include lib\DiosTone.lib

'Plays Mary had a Little Lamb
    
```

Schematic



Dios TW523 (X10) Functions

These functions are used to control X10 Devices via a TW523 interface. You can also use a PL513 interface devices as well for transmit only applications.

X10 Protocol

The X10 Basic Protocol is an 11 bit protocol consisting of a start sequence, house code and key code. Each bit is transmitted just after the zero crossing point on the AC line. All in all cases except the start bits the actual bit is inverted and retransmitted. In the case of the start bit the sequence is 1110 which even though its 4 bits really represents 2 bits in the scheme of things. In any case you dont have to worry about the actual bits in the protocol as the library takes care of this for you.

Many of the commands to the individual X10 device will require 2 11 bit sequences to be sent in order to control the device. The first set will contain the house code and device code. The next set in the sequence will contain the house code and a command code. These will need to be separated by a small delay (gap). You can use the TW523pause function for this delay.

Its important that you hold the pins high with 4.7k-10k resistors.

TW523write

```
TW523write(ZCport,DOport,house,keycode)
TW523write(iexp,iexp,iexp,iexp)
```

Description

Writes a 11 bit sequence using the house and keycode given.

- **ZCport** - The IO port connected to the Zero Crossing Detect lead on the TW523 (Line1)
- **DOport** - The IOport connected to the X10 from OEM lead on the TW523 (Line4)
- **house** - The house code of the device you wish to talk to. You may use the global constants for these values if you wish.
- **keycode** - The unit code or command code. You may use the global constants for these values if you wish.

TW523writeverify

```
TW523writeverify(ZCport,DOport,house,keycode)
TW523writeverify(iexp,iexp,iexp,iexp)
```

Description

Writes a 11 bit sequence using the house and keycode given. This function will return a 1 if the transmitted data is received back from the TW523 device. If a time out or other error is encountered (collision) a 0 will be returned. This function is not compatible with the PL513.

- **ZCport** - The IO port connected to the Zero Crossing Detect lead on the TW523 (Line1)
- **DOport** - The IOport connected to the X10 from OEM lead on the TW523 (Line4)
- **house** - The house code of the device you wish to talk to. You may use the global constants for these values if you wish.
- **keycode** - The unit code or command code. You may use the global constants for these values if you wish.

TW523read

TW523read(ZCport,DIport,house,keycode)
TW523read(iexp,iexp,iexp,iexp)

Description

Read a 11 bit sequence using the house and keycode given. The result is placed in TW523house and TW523keycode. Its up to your code to ignore or process the house and keycode information.

- **ZCport** - The IO port connected to the Zero Crossing Detect lead on the TW523 (Line1)
- **DIport** - The IOport connected to the X10 to OEM lead on the TW523 (Line3)

TW53pause

TW523pause(ZCport,DOport,bits)
TW523pause(iexp,iexp,iexp)

Description

Will read and ignore the number of zero crossings indicated. 6 bits is used for a gap between command sets.

- **ZCport** - The IO port connected to the Zero Crossing Detect lead on the TW523 (Line1)
- **DOport** - The IOport connected to the X10 from OEM lead on the TW523 (Line4)
- **bits** - The number of zero crossing states to skip.

Low Level Routines

TW523writebits

TW523writebits(ZCport,DOport,data,bits)
TW523writebits(iexp,iexp,iexp,iexp)

Description

This function writes the bits for the house and key codes. This function takes care of sending the compliment bits as well.

- **ZCport** - The IO port connected to the Zero Crossing Detect lead on the TW523 (Line1)
- **DOport** - The IOport connected to the X10 from OEM lead on the TW523 (Line4)
- **data** - The data value to send.
- **bits** - The number of bits to send.

TW523sendrawbits

TW523sendrawbits(ZCport,DOport,data,bits)
TW523sendrawbits(iexp,iexp,iexp,iexp)

Description

This function writes raw bits to the TW523. normally used to transmit start and gap bits. Compliment bits are not processed with this command.

- **ZCport** - The IO port connected to the Zero Crossing Detect lead on the TW523 (Line1)
- **DOport** - The IOport connected to the X10 from OEM lead on the TW523 (Line4)
- **data** - The data value to send.
- **bits** - The number of bits to send.

TW523readbits

TW523readbits(ZCport,Dlport,bits)
TW523readbits(iexp,iexp,iexp) as integer

Description

Reads the number of bits indicated. The compliment bits are ignored. The result is returned as an integer.

- **ZCport** - The IO port connected to the Zero Crossing Detect lead on the TW523 (Line1)
- **Dlport** - The IOport connected to the X10 to OEM lead on the TW523 (Line3)
- **bits** - The number of bits to recieve.

TW523waitzerocross

TW523waitzerocross(ZCport)
TW523waitzerocross(iexp)

Description

This function waits for the next zero crossing point on the AC signal.

- **ZCport** - The IO port connected to the Zero Crossing Detect lead on the TW523 (Line1)

Global Constants**X10 Unit Codes**

X10_1
X10_2
X10_3
X10_4
X10_5
X10_6
X10_7
X10_8
X10_9
X10_10
X10_11
X10_12
X10_13
X10_14
X10_15
X10_16

X10 Commands

X10_ON
X10_OFF
X10_ALLUNITSOFF
X10_ALLLIGHTSON
X10_DIM
X10_BRIGHT
X10_ALLLIGHTSOFF
X10_HAILREQUEST
X10_HAILACKNOWLEDGE
X10_STATUSON
X10_STATUSOFF
X10_STATUSREQUEST

X10 House Codes

X10_A
X10_B
X10_C
X10_D
X10_E
X10_F
X10_G
X10_H
X10_I
X10_J
X10_K
X10_L
X10_M
X10_N
X10_O
X10_P

Other Functions

TW523convertto

TW523convertto(value)
TW523convertfrom(iexp) as integer

Description

Converts standard codes to X10 format. IE the normalized house code of 1 (A) would be converted to 6 for use in X10 protocol.

- **value** - The normalized value to be converted to x10.

TW523convertfrom

TW523convertfrom(value)
TW523convertfrom(iexp) as integer

Description

Converts an X10 formatted code to normal. IE the X10 A code (6) to normalized (1)

- **value** - The normalized value to be converted to x10.

```
DiosPro
'TW523 Demo
func main()
  dim tmp,stat

  'Turns off device A2
  TW523write(0,2,X10_A,X10_2)
  TW53pause(0,2,6)
  TW523write(0,2,X10_A,X10_OFF)

loop:

  if ioport(4)=1 then
    print "bright"
    TW523write(0,2,X10_A,X10_BRIGHT)
  endif

  if ioport(5)=1 then
    print "dim"
    TW523write(0,2,X10_A,X10_DIM)
  endif

  if ioport(1) = 0 then
    TW523read(0,1)
    print "house=",TW523house
    print "code=",TW523keycode
  endif

  goto loop

endfunc

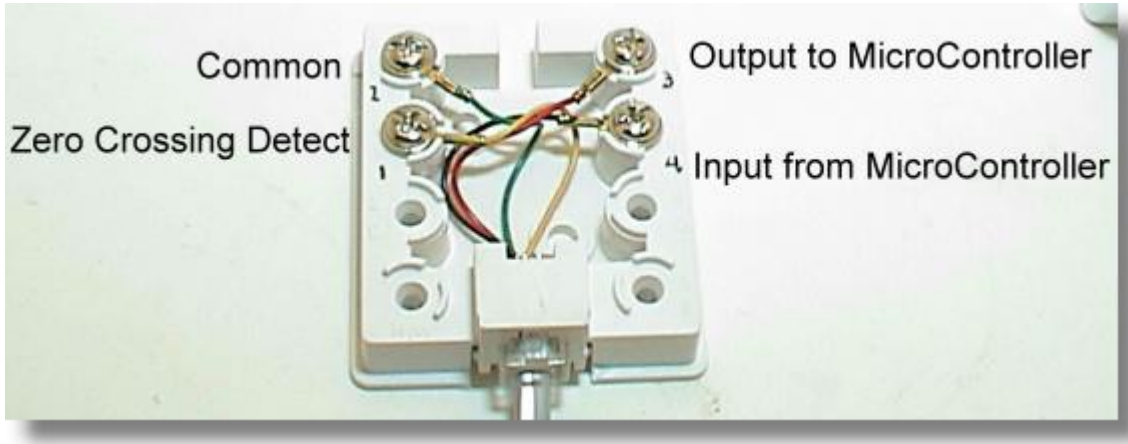
include lib\DiosTW523.lib
```

Hookup

The TW523 comes with a RJ11 telephone connector. This gives you 4 lead labeled 1-4.

- 1 - Zero crossing detect
- 2 - Common
- 3 - X-10 Output to OEM
- 4 - X-10 Input from OEM

Pick up a modular wall jack and RJ11 cable at your local home center. This will allow you to connect wires to your microcontroller with ease.



You can drop the use for the 3 resistors by using the pullupon command at the beginning of your program. Note that this only works if you are using IO ports 0-7.

You may use any Dios Board or chip to communicate with the TW523.

