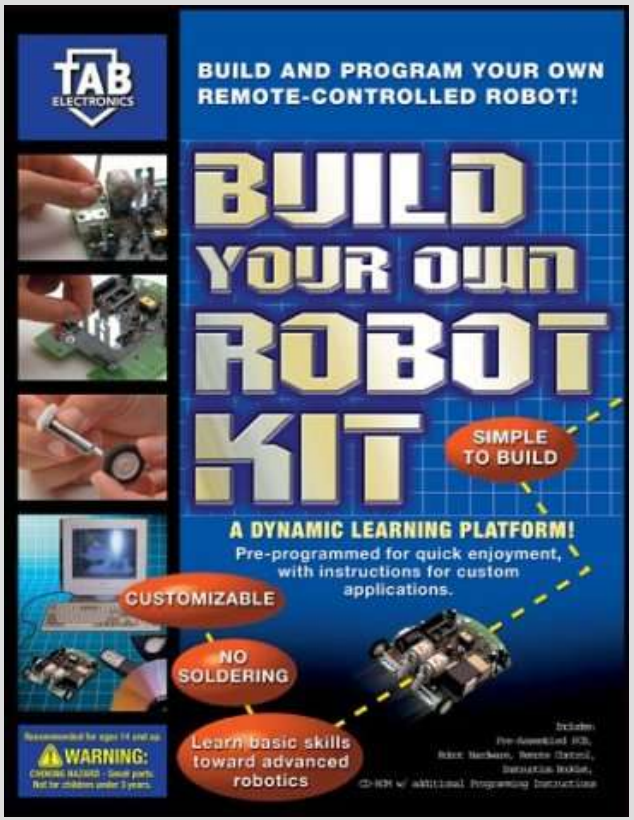
California State University, Chico
Intelligent Systems Laboratory
Chico, CA 95929-0410
http://isl.ecst.csuchico.edu



TAB Electronics' *Build Your Own Robot Kit* with a NetMedia *BasicX-24* microcontroller

B.A. Juliano (Juliano@csuChico.edu)
R.S. Renner (Renner@csuChico.edu)
January 2004

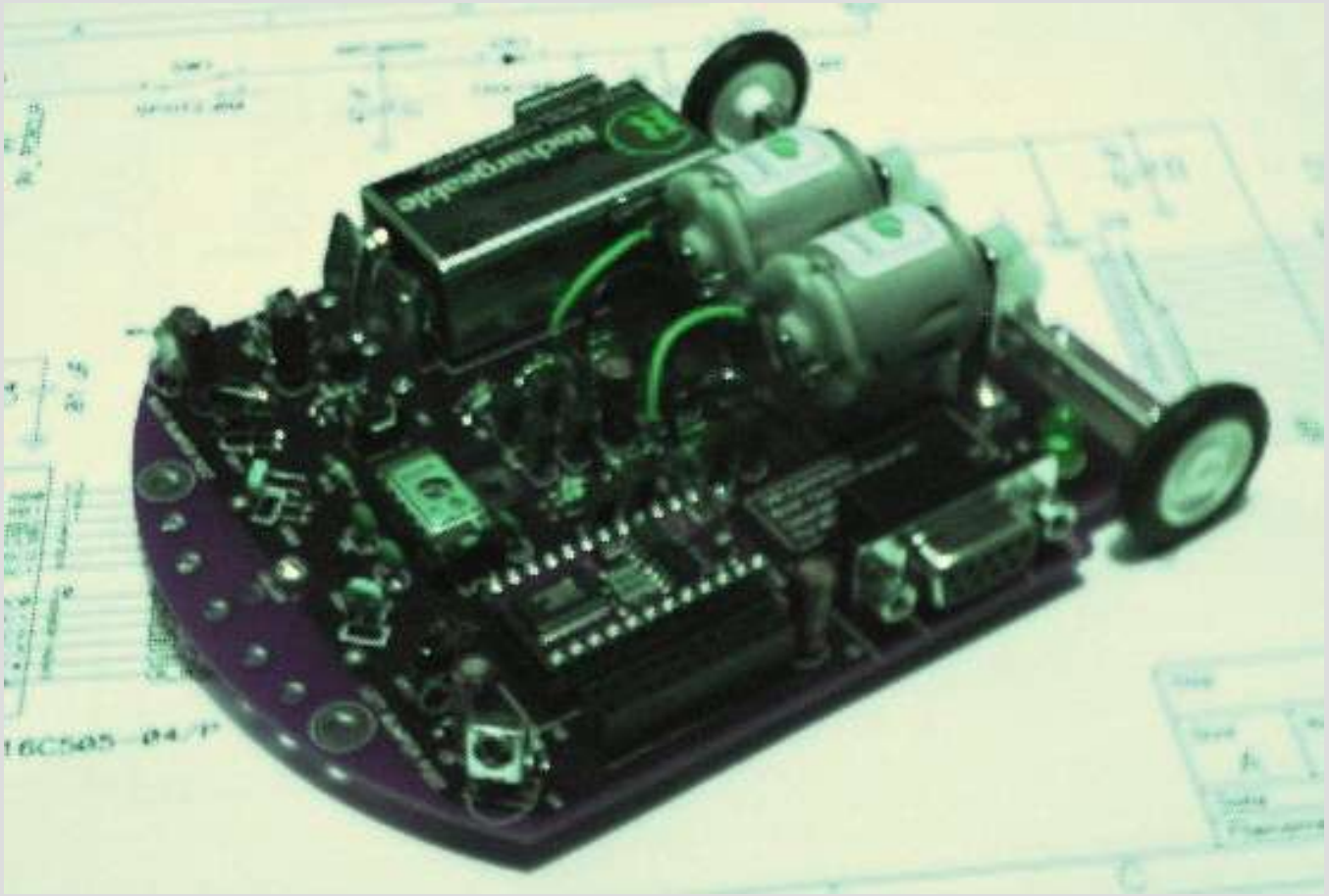
got bots?
http://isl.ecst.csuchico.edu/

TAB Electronics' *BYORK*

- **Element Products, Inc.**
  - 5155 West 123rd Place, Broomfield, CO 80020
  - Tel. 303 466-2750
  - Fax 303 466-4798
  - E-mail: sales@wirz.com
  - URL: http://wirz.com
- Element Products, Inc. is provider of two educational robot kits for the McGraw-Hill Companies. These kits are available from Barnes & Noble and Amazon.com ...

TAB Electronics' *BYORK*

- **BYORK Robot Anatomy** Body Type

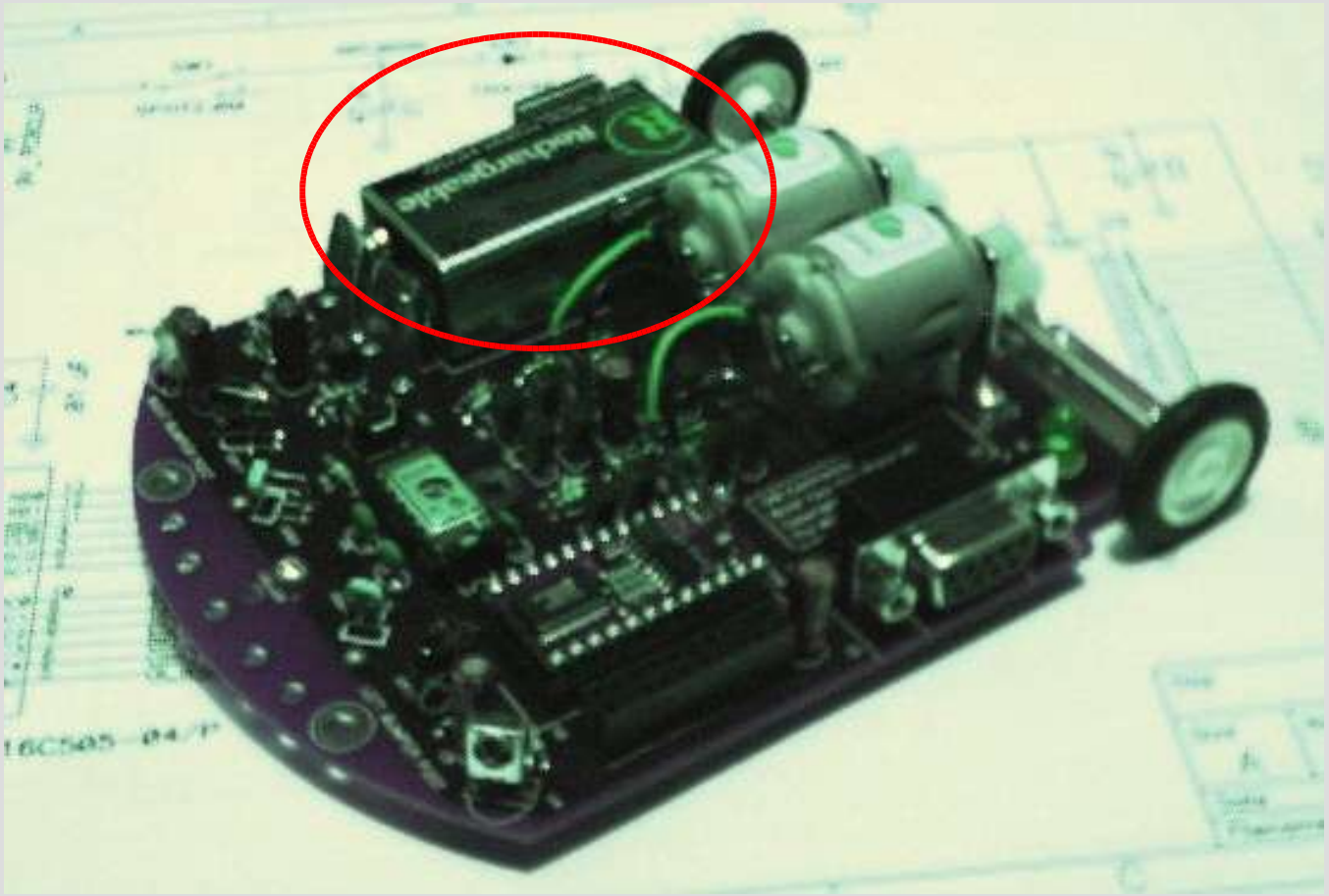


Differential drive robot, 5” long x4” wide

TAB Electronics' *BYORK*

- **BYORK Robot Anatomy**

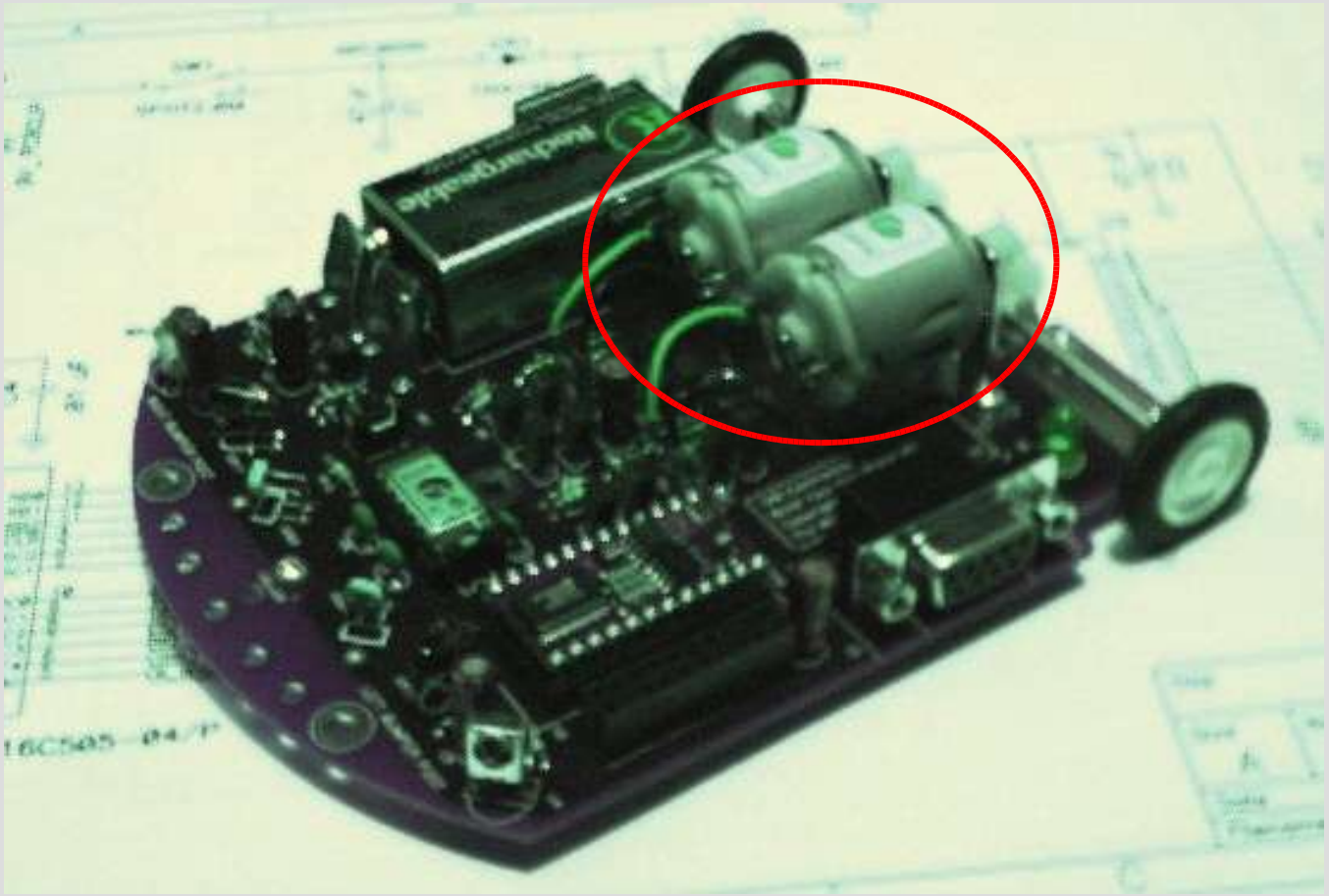
Fuel Source



Single 9-volt alkaline (20-30 min/batt)

TAB Electronics' *BYORK*

- **BYORK Robot Anatomy** Locomotion

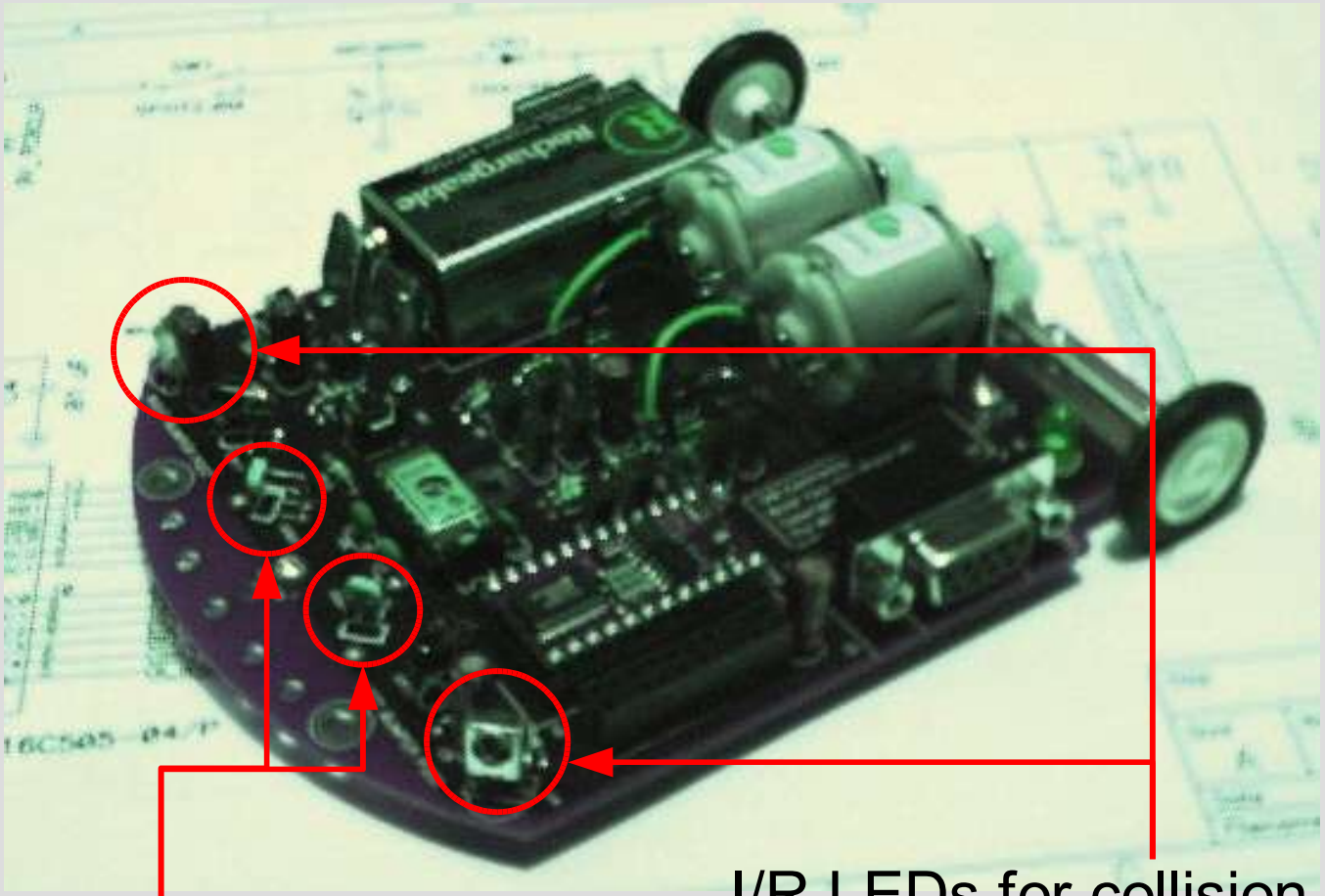


Full H-bridge for both motors

TAB Electronics' *BYORK*

- **BYORK Robot Anatomy**

Sensors



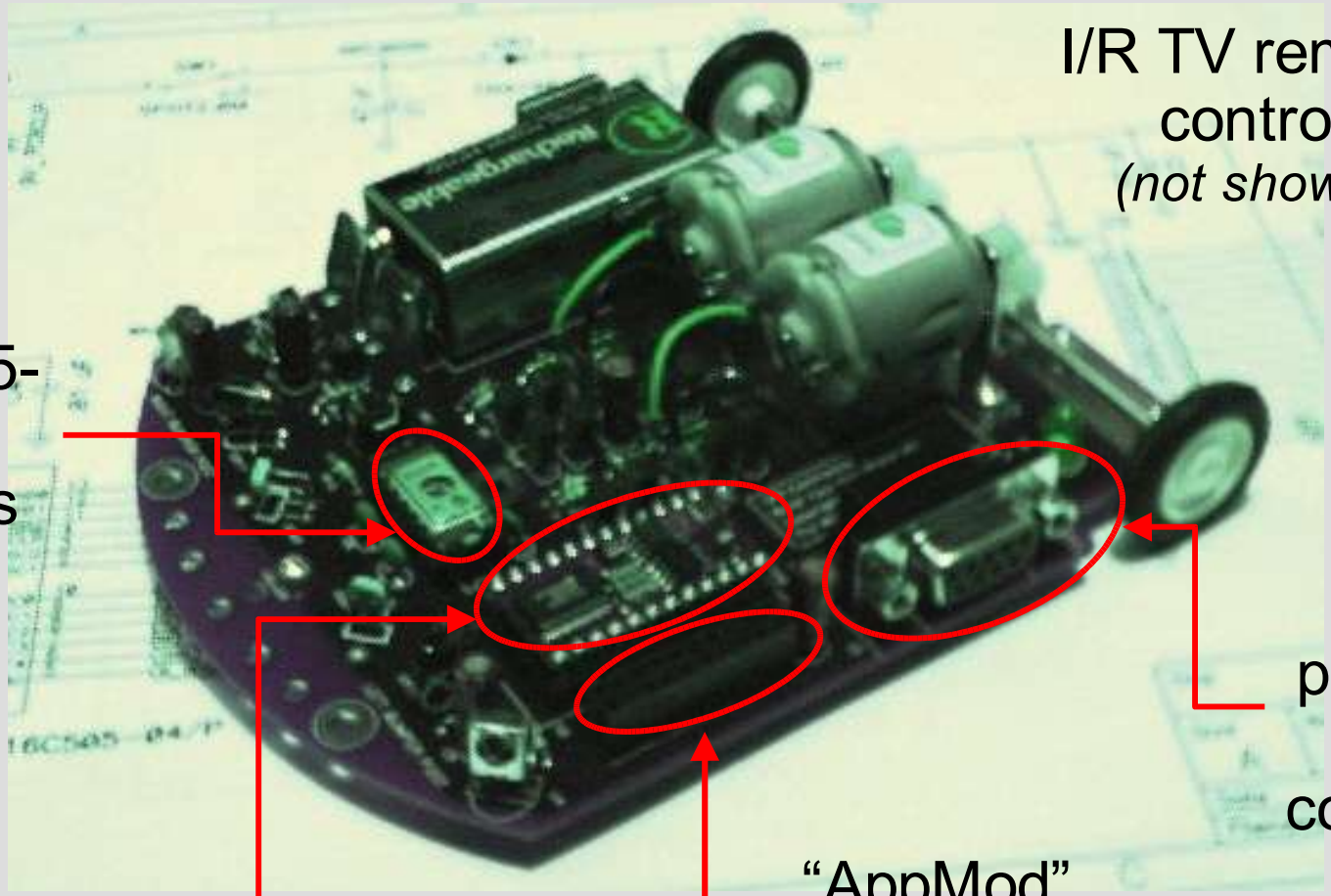
I/R LEDs for collision detection

CDS cells for light level detection

TAB Electronics' *BYORK*

- **BYORK Robot Anatomy**

Control



I/R TV remote control *(not shown)*

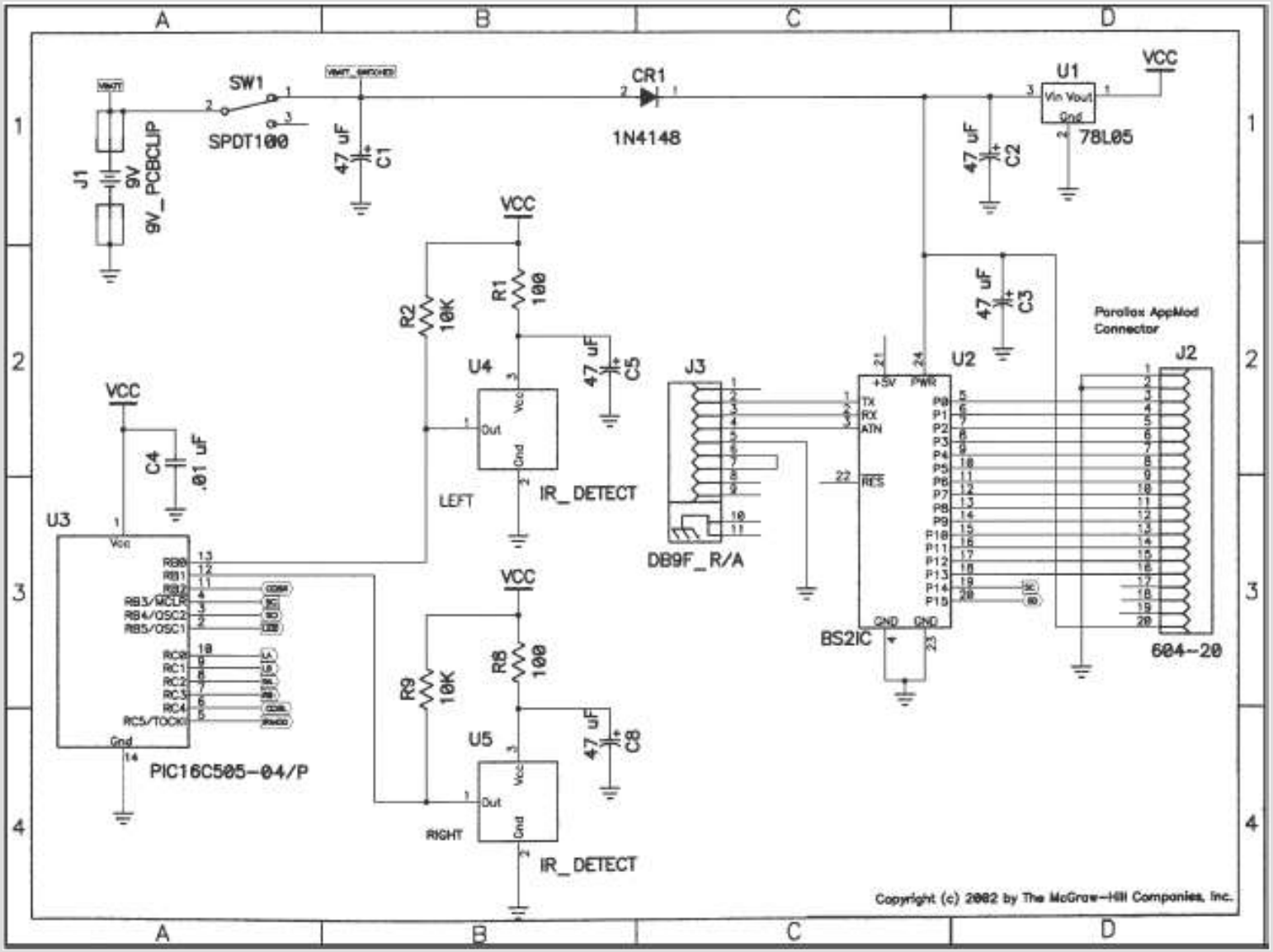
PIC16C505-controlled peripherals

BS2 program-ming connector

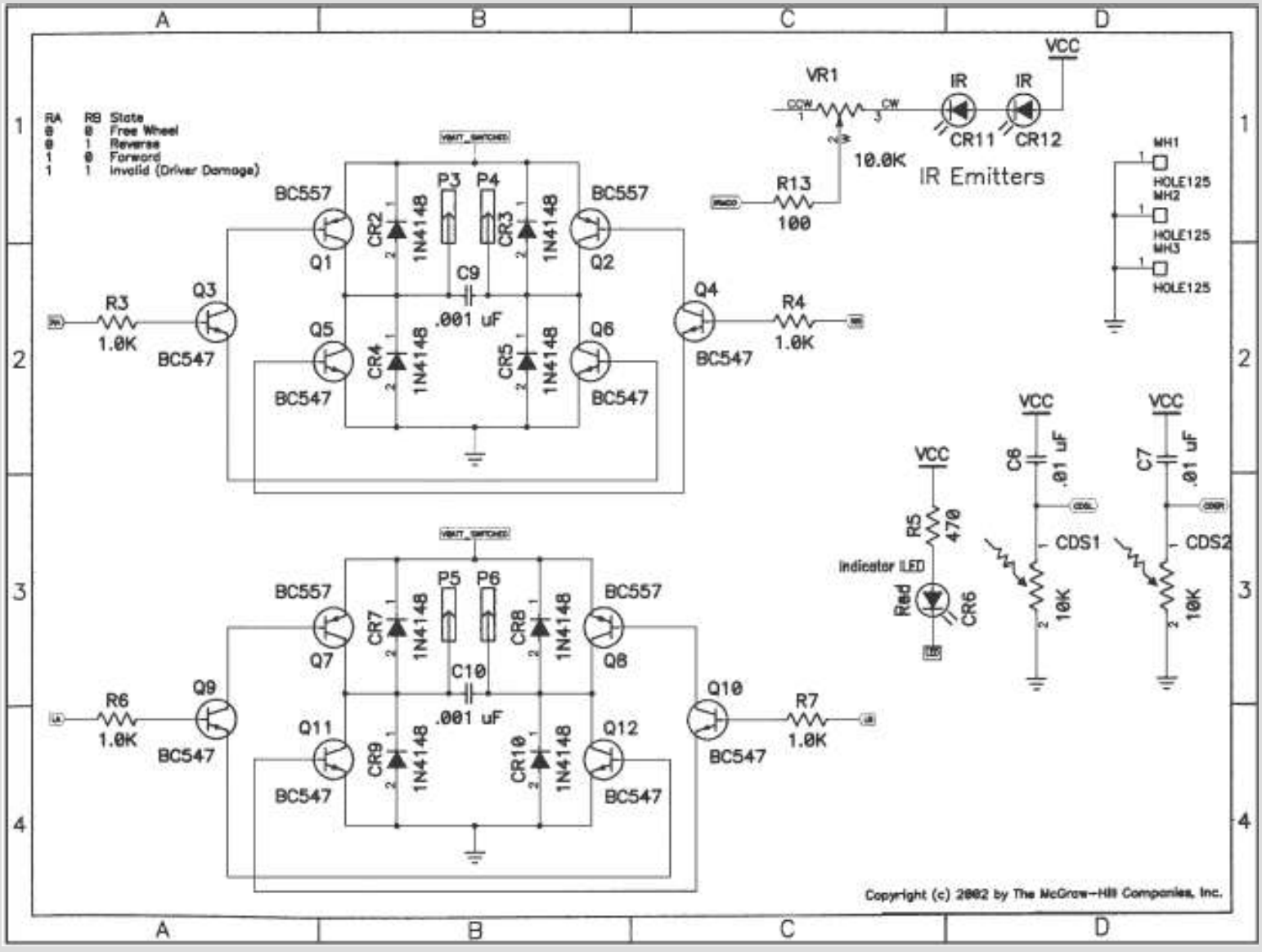
BS2 socket

"AppMod" socket

TAB Electronics' *BYORK*



TAB Electronics' *BYORK*



NetMedia BasicX-24

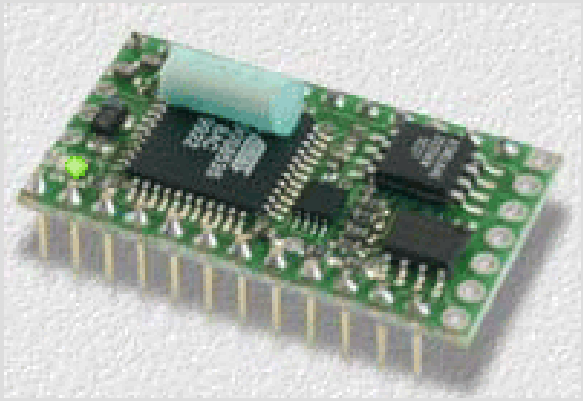
- **NetMedia Inc.**
  - 10940 N. Stallard Pl.,Tucson, Arizona 85737
  - Tel. 520-544-4567
  - Fax 520-544-0800
  - E-mail: sales@netmedia.com
  - URL: http://netmedia.com
- NetMedia Inc. is a leading manufacturer in Video Distribution and Camera equipment, embedded micro-control products, and embedded Ethernet web products.

NetMedia BasicX-24

- What is BasicX?
  - A BX-24 system combines
    - **BX-24 Hardware** – fast Atmel AT90S8535 core processor with a ROM for the BasicX OS, 400 B RAM, 32 KB EEPROM, lots of I/O devices such as timers, UARTs, ADCs, digital I/O pins, SPI peripheral bus, and more.
    - **BasicX Operating System (BOS)** – on-chip OS that provides multitasking and a high-speed BasicX execution engine.
    - **BasicX Development Environment** – true 32-bit Windows IDE.

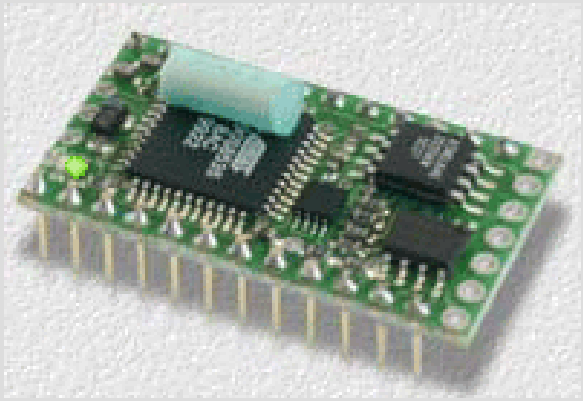
NetMedia BasicX-24

- **BasicX-24 Specifications:**



Speed	65,000 IPS
EEPROM	32K bytes
Max program length	8000+ lines
RAM	400 bytes
Available I/O pins	21 (16 standard + 2 serial only + 3 accessed outside standard dip pin area)
Analog Inputs (ADCs)	8
Serial I/O speed	1200 - 460.8K Baud
Programming interface	High speed Serial
Physical Package	24 pin DIP module

NetMedia BasicX-24

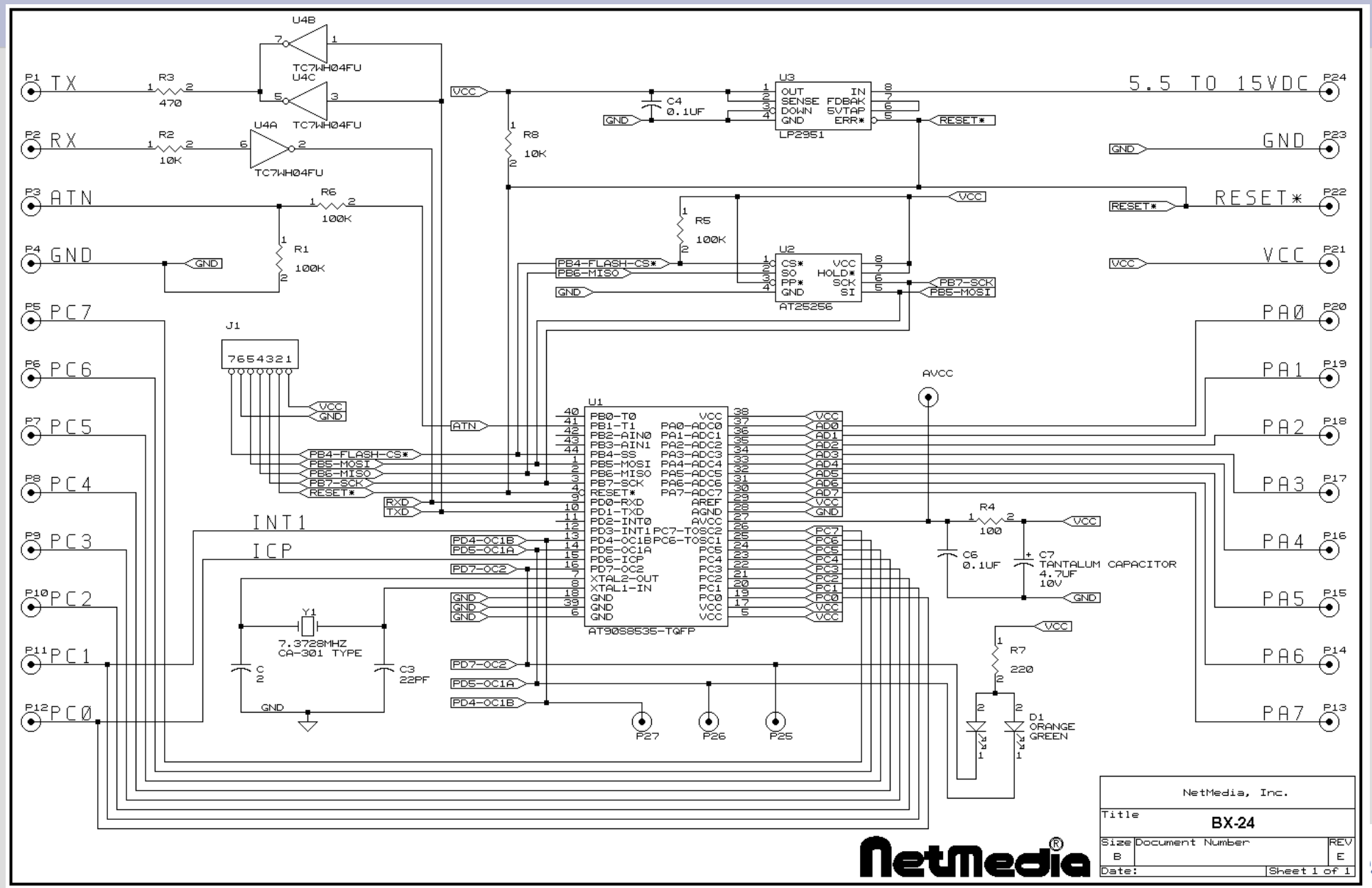


- Other features:
  - Pin-for-pin compatible with BS2 & BS2SX
  - Built-in SPI interface
  - On-chip voltage regulator
  - 2 user controllable on-chip surface-mount LEDs
  - System clock/calendar
  - Multitasking
  - Full IEEE floating point math

NetMedia BasicX-24

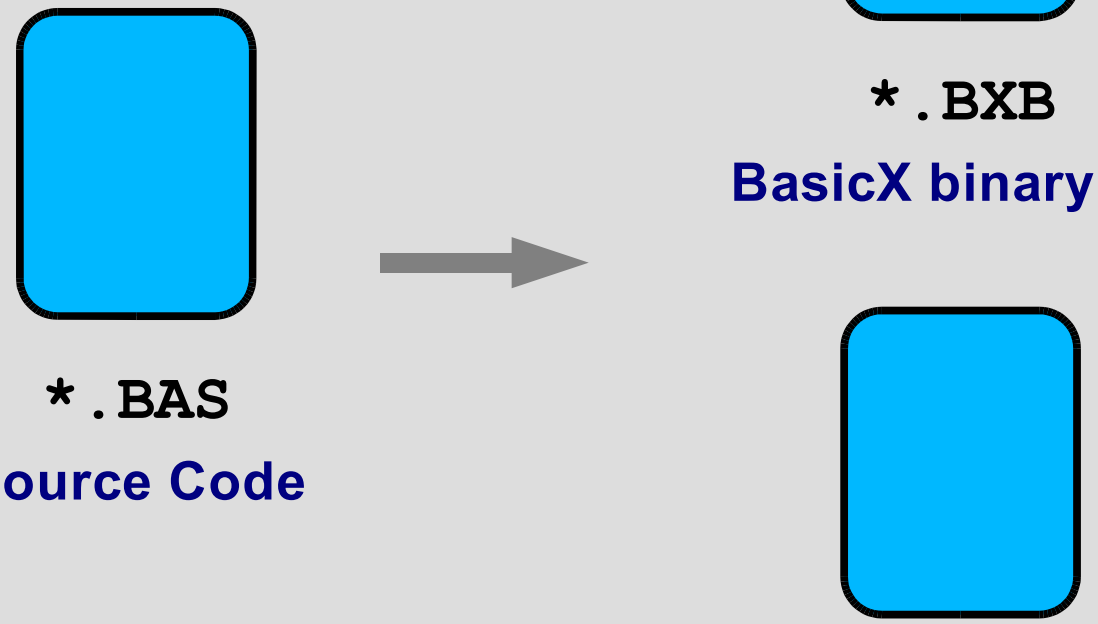
Features	BasicX-24	BS2	BS2SX
I/O Lines	16+	16	16
EEPROM	32 KB	2 KB	16 K Bank Switched
RAM	400 B	32 B	96 B
Speed (IPS)	65,000	4000	10,000
Max Prog Length	8000+	~500	~500 inst/2K Bank
Analog Inputs	8 (10 Bit ADCs)	No	No
Multitasking OS	Yes	No	No
FP Math	Yes	No	No
PC Prog Intrfc	Serial	Serial	Serial
Serial I/O	Yes	Yes	Yes
On-Chip LEDs	2 (Red & Green)	No	No
SPI Interface	Yes	No	No
On-Chip Regltr	Yes	Yes	Yes
Package	24-pin DIP	24-pin DIP	24-pin DIP

NetMedia BasicX-24



The BasicX Dev't Environment

- In a nutshell ...



The *Basic Express* Language

- General
  - Modules
    - allow one to split a program into multiple files.
    - facilitate user control of the visibility of constants, variables, and subprograms, which can be public (global) or private to a module.

    - Note: Module names are taken from filenames, which means filenames (minus extensions) must be legal Basic Identifiers.
    - Note: All BasicX identifiers must start with a letter, and all other characters must be letters, digits, or underscores. Identifiers are case-sensitive and can be up to 255 characters long.

The *Basic Express* Language

- Example Module:

```
Public  A As Integer
Private B As Single ' Module level code ends here

Public  Sub Main()
    Dim K As Integer
    A = 1
    For K = 1 to 10
        A = A + 1
    Next
    B = CSng(A)
    Call Square(B)
End Sub

Private Sub Square(X As Single)
    X = X * X
End Sub
```

The *Basic Express* Language

- General
  - Main program
    - Program starts execution with a procedure called Main, which must be a public procedure.
  - Statement format
    - The underscore character is used as a line continuation character for long statements extended between two or more lines.
  - Comment format
    - An apostrophe character is used to denote comments.

The *Basic Express* Language

- Subprograms
  - General
    - A subprogram allows you to take a group of related statements and treat them as a single unit.
    - Subprograms consist of procedures and functions.
    - The difference between a procedure and function is that a function can appear as part of an expression, but a procedure must be called in a standalone statement.

The *Basic Express* Language

- Subprograms
  - Sub procedures
    - Definition syntax:

```
[Private|Public] Sub procedure_name(arguments)
    [statements]
End Sub
```

    - Invocation syntax:

```
Call procedure_name(arguments)
```

      or

```
procedure_name arguments
```

    - You can also exit a procedure by using an `Exit Sub` statement.

The *Basic Express* Language

- Example Subprogram:

```
Private Sub GetPosition(ByRef X As Single)

    Call ReadDevice(X)
    If (X > 100.0) Then
        Exit Sub
    End If
    X = X * X

End Sub
```

The *Basic Express* Language

- Subprograms
  - Functions
    - Definition syntax:

```
[Private|Public] Function
      function_name(arguments) As type
      [statements]
End Function
```

    - Note: The function return value can be defined by assigning to the function name inside the function itself.

The *Basic Express* Language

- Example Function:

```
Public Function F(ByVal i As Integer) As Integer
    F = 2 * i ' defines the function return value
    F = F * F ' can also read the function name
End Function
```

- Example with `Exit Function` statement:

```
Function F(ByVal i As Integer) As Single
    If (i = 3) Then
        F = 92.0
        Exit Function
    End If
    F = CSng(i) + 1.0
End Function
```

The *Basic Express* Language

- Subprograms
  - Function return type
    - Functions can return non-persistent scalar types or string types.
    - Example string function:

```
Function F() As String
    F = "Hello, world"   ' F is write-only
End Function
```

    - Note: Every assignment to the function return must be immediately followed by an “Exit Function” or “End Function” statement.

The *Basic Express* Language

- Subprograms
  - Function return type
    - Note: If a function returns an UnsignedInteger or UnsignedLong object, the first statement in the function must be a Set statement.
    - Example:

```
Function F() As UnsignedInteger
    Set F = New UnsignedInteger
    [statements]
End Function
```

The *Basic Express* Language

- Subprograms
  - Parameter passing
    - Parameters can be passed to a subprogram by reference (ByRef) or by value (ByVal).
    - Pass by reference is the default.
    - Exceptions: For types String, UnsignedInteger, and UnsignedLong passed by value, these parameters are write-protected in the called subprograms for efficiency.

The *Basic Express* Language

- Subprograms
  - Parameter passing summary:

Parameter	ByRef	ByVal
Scalar variable	Yes	Yes
Array element	Yes	Yes
1D array, lower bound = 1	Yes	No
Multidimensional array	No	No
Array with lower bound not 1	No	No
Numeric expression	No	Yes
Numeric literal	No	Yes
Boolean expression	No	Yes
Boolean literal	No	Yes
Persistent variable	No	Yes

The *Basic Express* Language

- Control structures
  - The **`If-Then`** statement
    - Syntax:

```
 If (boolean_expression) Then
      [statements]
[ElseIf (boolean_expression) Then
      [statements]]
[Else
      [statements]]
 End If
```

The *Basic Express* Language

- Control structures
  - The **Do-Loop** statement
    - Syntax and variants:

```
Do
     [statements]
Loop

Do [While|Until] (boolean_expression)
     [statements]
Loop

Do
     [statements]
Loop [While|Until] (boolean_expression)
```

The *Basic Express* Language

- Control structures
  - The **`Do-Loop`** statement
    - Note: The “Exit Do” statement can be used to exit any of the Do-Loops.
    - Note: Do-Loops can be nested up to a level of ten.

The *Basic Express* Language

- Control structures
  - The **`For-Next`** statement
    - Syntax:

```
For index = beg_val To end_val [Step 1 | -1]
    [statements]
Next
```

    - Note: *`index`* must be a local variable of a discrete type.
    - Note: Loop counters cannot be changed inside the loop; loop counters are treated as if they were a constant within a loop.

The *Basic Express* Language

- Control structures
  - The **`For-Next`** statement
    - Note: The “Exit For” statement can be used to exit a For-Next loop.
    - Note: For-Next loops can be nested up to a level of ten.

The *Basic Express* Language

- Control structures
  - The **`Select-Case`** statement
    - Syntax:

```
Select Case test_expression
    Case expression_list1
        [statements]
    [Case expression_list2
        [statements]]
    [Case Else
        [statements]]
End Select
```

    - Note: *`test_expression`* must be a discrete, non-string type (boolean or discrete numeric).

The *Basic Express* Language

- Example:

```
Select Case BinNumber(Count)
    Case 1
        Call UpdateBin(1)
    Case 2
        Call UpdateBin(2)
    Case 3,4
        Call EmptyBin(1)
        Call EmptyBin(2)
    Case 5 To 7
        Call UpdateBin(7)
    Case Else
        Call UpdateBin(8)
End Select
```

The *Basic Express* Language

- Control structures
  - The **GoTo** statement
    - A GoTo branches unconditionally to a specified label.
    - Example:

```
GoTo label_name
[statements]

label_name:
    [statements]
```

    - Note: Labels must be followed by a colon.

got bots?
http://stlecst.csuchico.edu/

OpenOffice.org Source Project

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Data types

Type	Storage	Range
Boolean	8 bits	True .. False
Byte	8 bits	0 .. 255
Integer	16 bits	-32,768 .. 32,767
Long	32 bits	-2,147,483,648 .. 2,147,483,647
Single	32 bits	-3.402823E+38 .. 3.402823E+38
String	Varies	0 to 64 characters
BoundedString	Varies	0 to 64 characters

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Declarations
    - All variables must be declared before they are used.
    - In module-level code (default is private):

      ```
      [Public|Private|Dim] variable As type
      ```

    - Inside a subprogram:

      ```
      Dim variable As type
      ```

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Declarations
    - Example:

```
Public Distance As Integer       ' global
Private Temperature As Single    ' local to module

Sub ReadPin()
    Dim PinNumber As Byte        ' local to sub
    Dim S1 As String             ' variable length
    Dim S2 As String * 1         '  1-char string
    Dim S3 As String * 64        ' 64-char string
    [statements]
End Sub
```

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Constants
    - In module-level code (default is private):

```
[Public|Private] Const
                 constant_name As type = literal
```

    - Inside a subprogram:

```
Const constant_name As type = literal
```

    - Examples:

```
Const PI As Single = 3.14159
Private Const ROOMTEMP As Single = 70.0
Public  Const MAXSIZE As Byte = 20
```

got bots?
http://isl.ecst.csuchico.edu/

OpenOffice.org Source Project

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Numeric literals
    - Decimal integer examples:

```
1
-1
10
255
```

    - Decimal floating point examples:

```
1.0
-0.05
1.53E20
-978.3E-3
```

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Numeric literals
    - Hexadecimal integer examples:

```
&H3
&HFF
&H7FFF        '  32767
-&H8000&      ' -32768 (note trailing ampersand)
```

      * Trailing ampersands are required for hex numbers in range &H8000 (32,768) to &HFFFF (65,535)

    - Binary examples:

```
bx00000001 '   1
bx00001111 '  15
bx11111111 ' 255
```

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Converting data types

Function	Result	
`CBool`	`Boolean`	
`CByte`	`Byte`	} statistical rounding
`CInt`	`Integer`	
`CLng`	`Long`	
`CSng`	`Single`	
`CStr`	`String`	
`FixB`	`Byte`	} truncation
`FixI`	`Integer`	
`FixL`	`Long`	

- Note: `CBool` allows only a `Byte` type as an operand.
- Note: `FixB`, `FixI`, and `FixL` allow only floating point types as operands.

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Type declaration characters
    - For floating point numbers, the exclamation point (!) and pound sign (#) are allowable as type declaration characters, but only if they replace a trailing ".0" in floating point numeric literals. The following are equivalent:

```
12.0
12!
12#
```

    - In VB and other Basic dialects, (!) signifies single precision, and (#) signifies double precision.

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Type declaration characters
    - As indicated earlier, hexadecimal numeric literals in range 32,768 (&H8000) to 65,535 (&HFFFF) are required to have ampersand type declaration characters.
    - Note: It is illegal to append type declaration characters to variable names or to numeric literals with fractional parts.

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Arrays
    - Arrays can be declared for all data types except strings and other arrays.
    - Examples:

```
Dim I(1 To 3) As Integer, _
    J(-5 To 10, 2 To 3) As Boolean
Dim X(1 To 2, 3 To 5, 5 To 6, 1 To 2, 1 To 2, _
      1 To 2, -5 To -4) As Single
```

    - Arrays can have 1 to 8 dimensions, and both upper and lower bound of each index must be declared.
    - For parameter passing:

```
Dim I(1 To 5) As Byte   ' Can be passed
Dim J(0 To 5) As Byte   ' Can't - lower bound not 1
Dim K(1 To 2, 1 To 3) As Byte   ' Can't - not 1D
```

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Persistent variables
    - are stored in EEPROM memory; hence, they retain their values even after power is turned off.
    - must be declared at module level and are not allowed as local variables.

    - Declaration syntax:

```
[Public|Private|Dim] variable
                        As New persistent_type
```

where *persistent_type* is:

```
PersistentBoolean|PersistentByte|
  PersistentInteger|PersistentLong|
  PersistentSingle
```

The *Basic Express* Language

- Variables, Constants, and Data Types
  - Rules for Persistent variables:
    1. All persistent variables should be declared in one module.
    2. The ordering of declarations of persistent variables must match the order in which the variables are accessed (via read or write operation).
    3. All persistent variables should be private.

    - Note: These rules guarantee the ordering of persistent variables in EEPROM so that the location of each variable is the same after cycling power on and off.

The *Basic Express* Language

- Expressions
  - General
    - BasicX uses strong typing, which means binary operators must operate on equivalent types.
    - Both sides of an assignment statement must be of the same type; hence, each argument passed to a subprogram must have the correct type.

The *Basic Express* Language

- Expressions
  - Relational operators

Equality	=
Inequality	<>
Less	<
Greater	>
Less or equal	<=
Greater or equal	>=

  - Relational operators yield a Boolean type.
  - The equality and inequality operators require operands of Boolean or numeric types; all other operators require numeric types.

The *Basic Express* Language

- Expressions
  - Logical operators

```
And
Or
Not
Xor
```

    - Logical operators require operands of Boolean type or unsigned discrete types (Byte, UnsignedInteger, or UnsignedLong), and the resulting type matches that of the operands.
    - When operands are numeric types, bitwise operations are done.

The *Basic Express* Language

- Expressions
  - Arithmetic operators

Addition	`+`
Subtraction	`-`
Multiplication	`*`
Division (float)	`/`
Division (integer)	`\`
Modulus	`Mod`
Absolute value	`Abs`

    - Arithmetic operators require numeric operands.
    - Note that there are separate operator symbols for floating point and discrete operands.

The *Basic Express* Language

- Expressions
  - String operators

    Concatenation &

    - Strings can be concatenated.
    - Generally, if the destination string is larger than the resulting string, the result is left-justified and blank-filled. If the destination string is smaller, the result is truncated.

The *Basic Express* Language

- Expressions
  - Operator precedence

(Highest)	[1]	`Abs`	`Not`				
	[2]	`*`	`\`	`/`	`Mod`	`And`	
	[3]	`+`	`-`	`Or`	`Xor`		
(Lowest)	[4]	`=`	`>`	`<`	`<>`	`<=`	`>=`

The *Basic Express* Language

- Expressions
  - Assignment statements
    - Syntax:
      variable **=** *expression*
    - The types of both sides of an assignment statement must match. No implicit type conversions are done.

The *Basic Express* Language

- Unsigned Types
  - General
    - The following unsigned integer types are provided:

Type	Storage	Range
Byte	8 bits	0 .. 255
UnsignedInteger	16 bits	0 .. 65,535
UnsignedLong	32 bits	0 .. 4,294,967,295

The *Basic Express* Language

- Unsigned Types
  - **`UnsignedInteger`** and **`UnsignedLong`** are treated as classes, and are subject to the following rules:
    1. If you want to declare unsigned objects as local or module-level variables, you need to use the **`New`** keyword:

```
Dim I As New UnsignedInteger
```

       However, the **`New`** keyword is not required in subprogram parameter lists:

```
Private Sub S(ByRef I As UnsignedInteger)
```

The *Basic Express* Language

- Unsigned Types

2. Functions that use unsigned object returns must have a **Set** statement as the first line of the function.

```
Function F() As UnsignedInteger
Set F = New UnsignedInteger
    F = 65535
End Function
```

3. Unsigned objects cannot be used in **Const** statements.

3. If you pass an unsigned object by value, the object is treated as if it were write-protected within the called subprogram.

The *Basic Express* Language

- Unsigned Types
  - Type conversions

`CuInt`	Converts any discrete type to `UnsignedInteger`
`CuLng`	Converts any discrete type to `UnsignedLong`
`FixUI`	Truncates FP type, converts to `UnsignedInteger`
`FixUL`	Truncates FP type, converts to `UnsignedLong`

The *Basic Express* Language

- Unsigned Types
  - Known bugs
    1. The following arithmetic operations are not allowed for **`UnsignedLong`** types:

       `*` `\` `Mod`

    2. Portability issue – if an **`UnsignedInteger`** or **`UnsignedLong`** is used as a formal parameter, and if the object is passed by value, the actual parameter is supposed to be restricted to a single object. BasicX erroneously allows numeric literals and expressions as actual parameters.

The *Basic Express* Language

- Strict vs. Permissive Syntax Rules
  - Compiler option
    - The compiler can be configured to use either *strict* or *permissive* syntax rules.
    - This option affects how numeric literals, logical expressions, and For-Next loop counters are treated.
    - The default is to use *strict* rules.

The *Basic Express* Language

- Strict vs. Permissive Syntax Rules
  - Permissive rules
    - For-Next loop counters
      - Counters are not required to be local variables.
      - Counters are not write-protected inside loops.
      - The scope of a counter is not restricted to its loop.

The *Basic Express* Language

- Strict vs. Permissive Syntax Rules
  - Permissive rules
    - Numeric literals and logical operations
      - Signed discrete types (`Integer` and `Long`) are allowed to appear in bitwise-logical expressions.
      - A wider choice of type declaration characters that can be appended to hexadecimal numeric literals are available. You can use ampersand or percent characters, or no characters.

The *Basic Express* Language

- Strict vs. Permissive Syntax Rules
  - Known bugs
    - In permissive mode, some hexadecimal numeric literals result in incorrect values for **`UnsignedLong`** types. For example, if `X` is type **`UnsignedLong`**, the assignment `X=&HFFFFFFFF` sets `X` to 65,535 rather than the correct 4,294,967,295.
    - A workaround is to turn on strict syntax checking.

The *Basic Express* Language

- Miscellaneous statements
  - Attribute statement
    - `Attribute VB_Name` statements are ignored. All other attribute statements are illegal.
    - Example:

```
Attribute VB_Name = "MyFirstModule"
```

    - Note: In Visual Basic, module names are taken from the VB_Name attribute; BasicX derives module names directly from module filenames.

The *Basic Express* Language

- Miscellaneous statements
  - Option statement
    - `Option Explicit` requires that variables are declared before use, which is the default in BasicX. All other Option statements are illegal.
    - Syntax:

```
Option Explicit
```

The *Basic Express* Language

- Miscellaneous statements
  - With statement
    - A **`With`** statement facilitates use of shorthand identifiers for objects, which means the object name qualifier can be omitted from an object reference.
    - Currently, these statements can only be used with **`Register`** objects. No other objects are allowed in **`With`** statements.

The *Basic Express* Language

- Miscellaneous statements
  - With statement
    - A **`With`** statement can only be used inside a subprogram, and a **`With`** statement that precedes a block of code must be terminated by an **`End With`** statement at the end of the block, but before the end of the subprogram.
    - Nested **`With`** statements are not allowed.

The *Basic Express* Language

- With statement
  - Syntax:

```
With Register
     [statements]
End With
```

  - Example code:

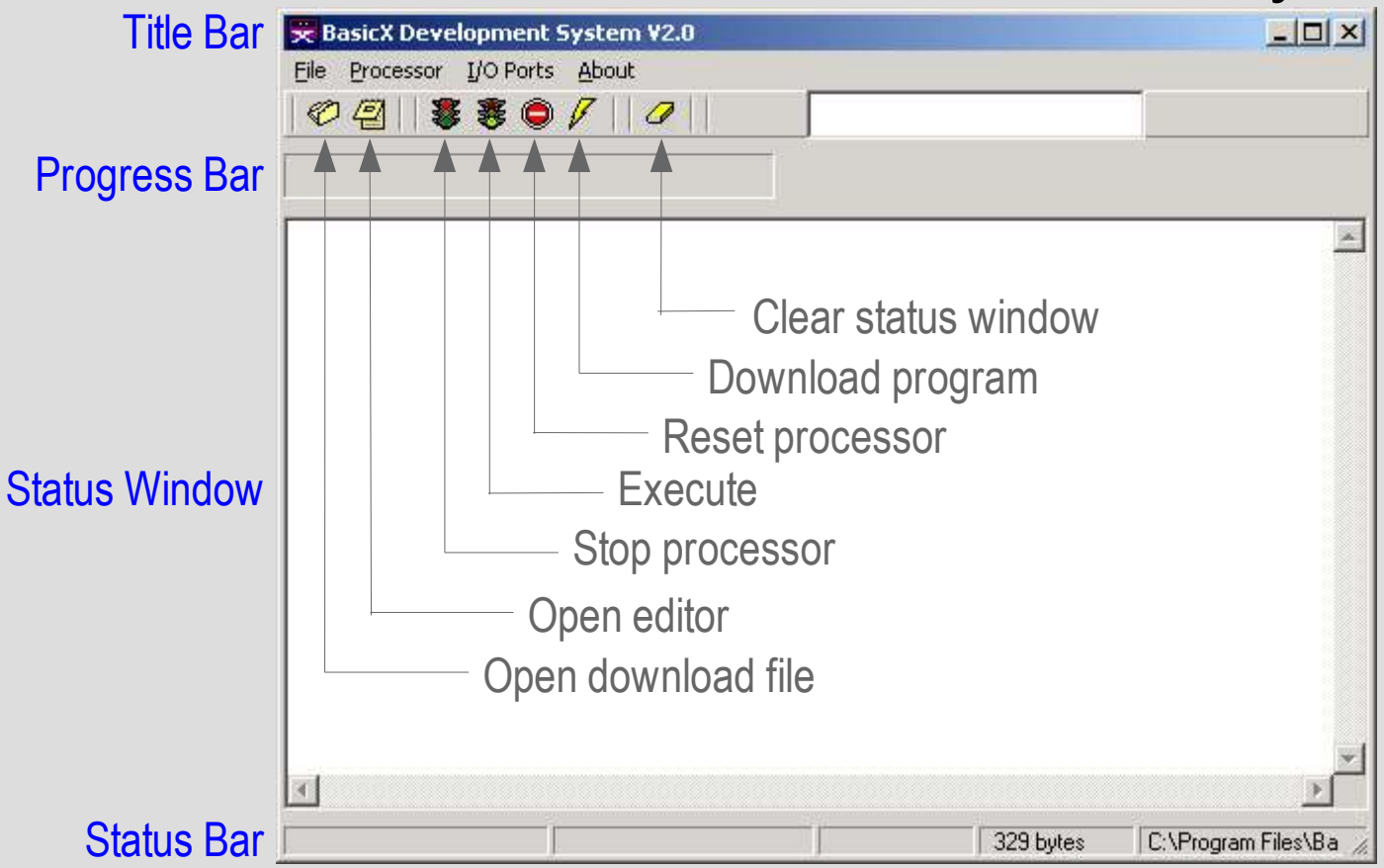
```
' The following assignments are equivalent.

Register.OCR1AH = 255

With Register
     .OCR1AH = 255
End With
```

The *Basic Express* Compiler

- **BasicX Programming Environment**
  - **BasicX Software – Complete**
    - Includes Downloader/Compiler/Editor, BasicX Documentation, Example Files, Application Notes, and ATMEL docs.
    - http://basicx.com/downloads/bx-setup-210-complete.zip
  - **BasicX Software – Program Only**
    - Includes Downloader/Compiler/Editor only.
    - http://basicx.com/downloads/bx-setup-210-program.zip
  - **BasicX Software – Documents Only**
    - Includes BasicX Documentation, Example Files, Application Notes, and ATMEL docs only.
    - http://basicx.com/downloads/bx-setup-210-docs.zip

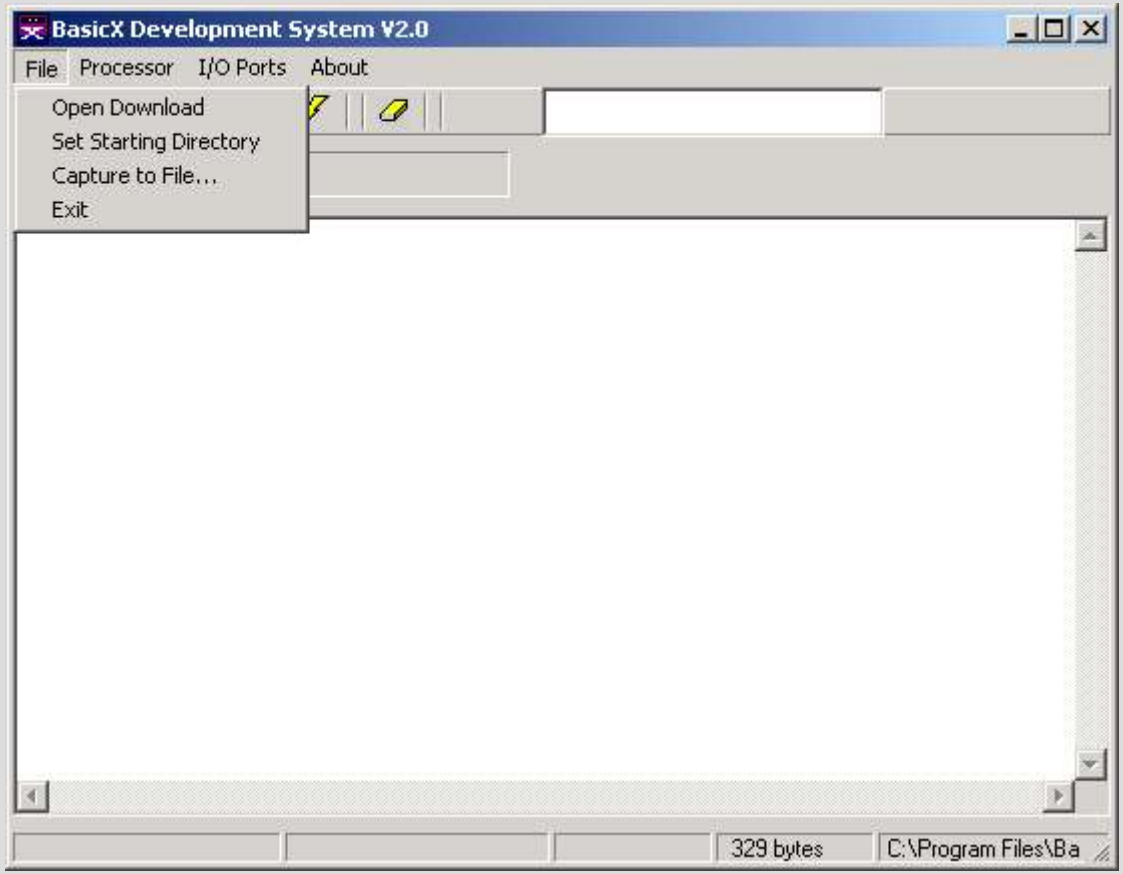
The *Basic Express* Compiler

- Downloader
  - The BasicX Downloader is where executable files are downloaded and run on the BasicX system.



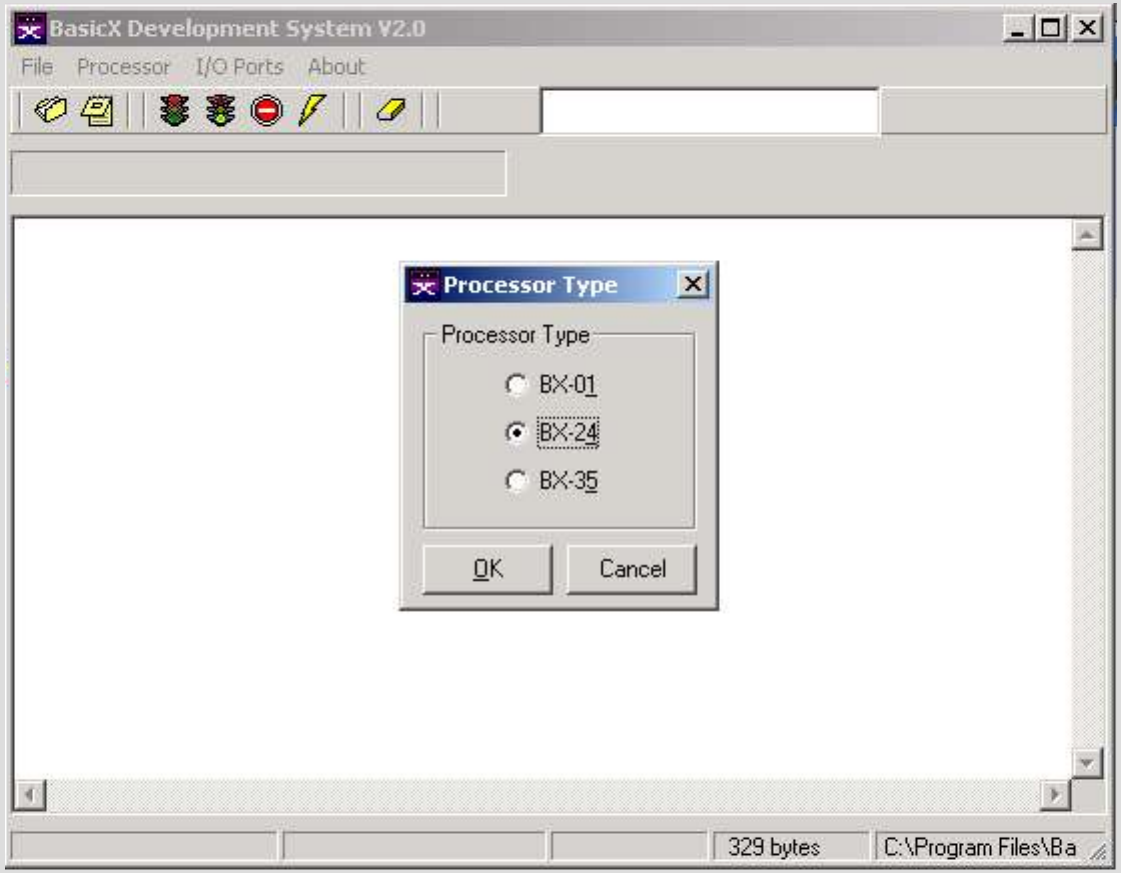
The *Basic Express* Compiler

- The **File** Menu
  - allows one to open BXB and PRF files.



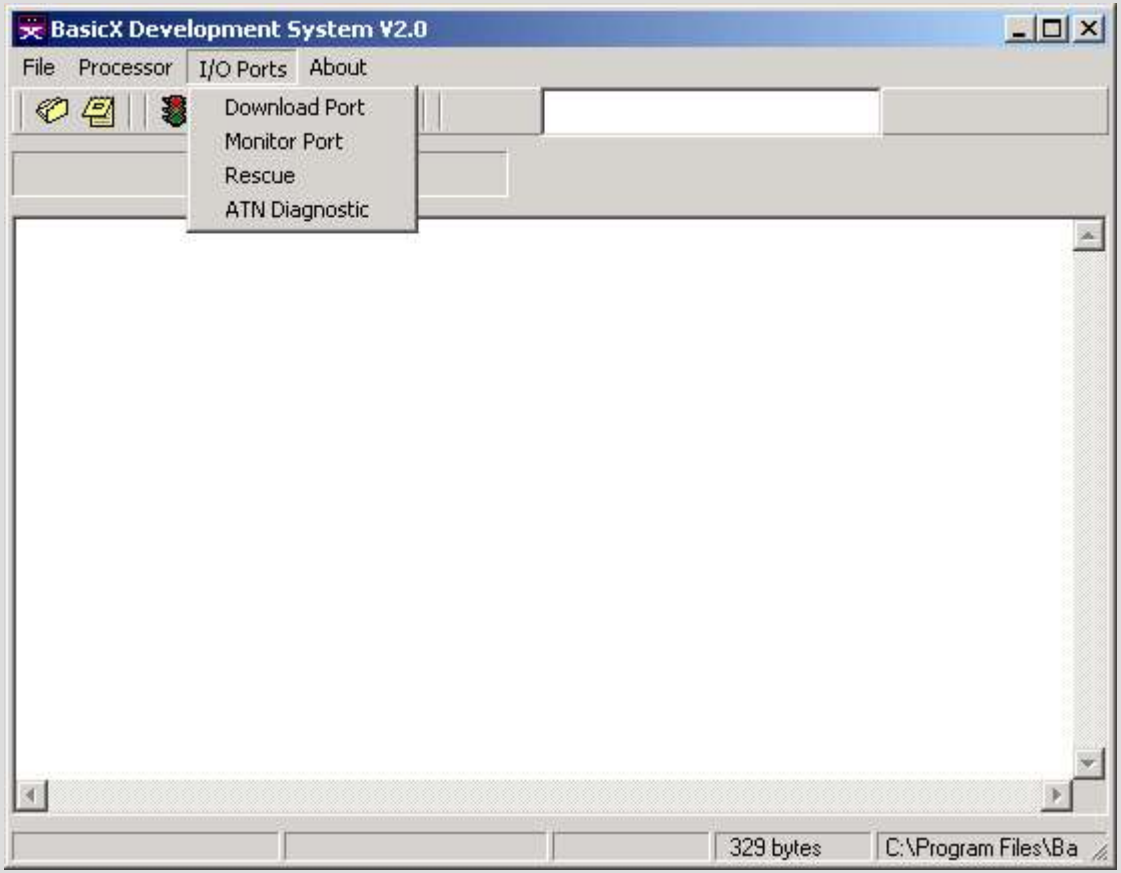
The *Basic Express* Compiler

- The **Processor** Menu
  - allows one to select which BasicX system type to use.



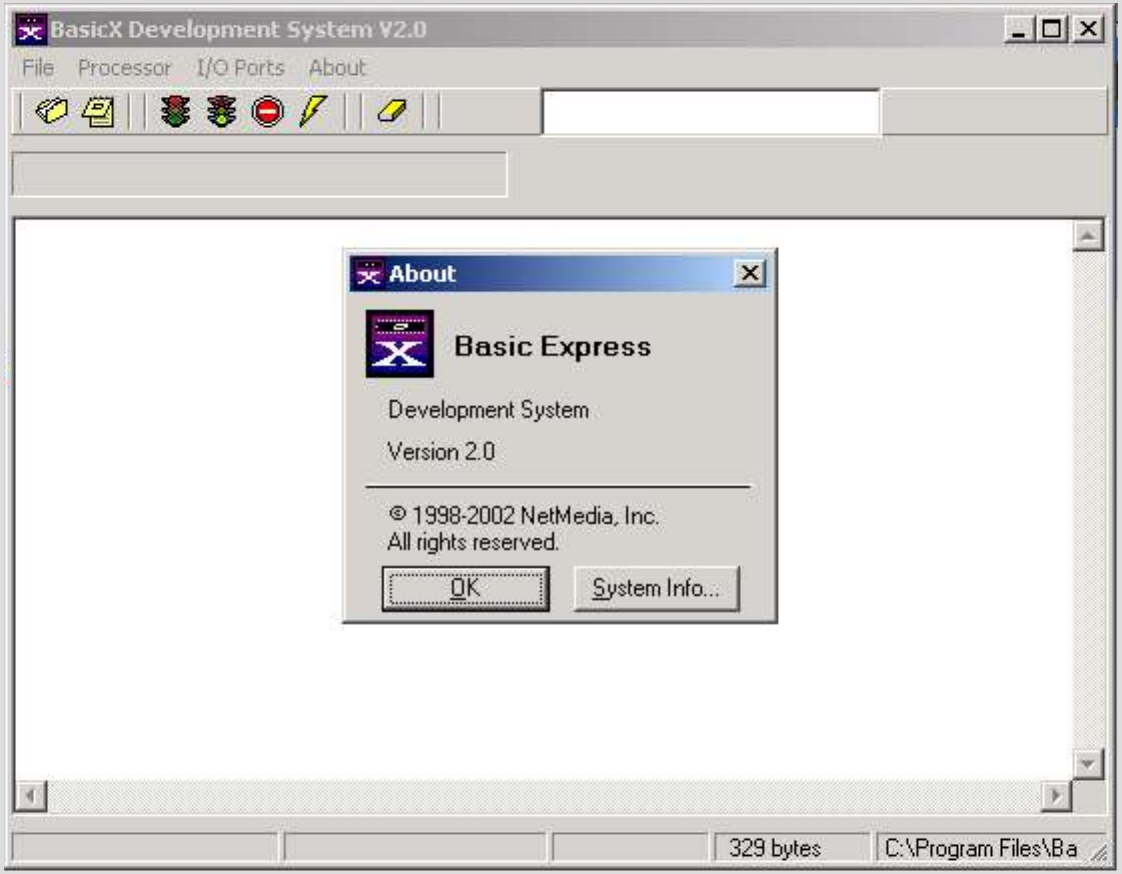
The *Basic Express* Compiler

- The **I/O Ports** Menu
  - allows one to configure communications ports.



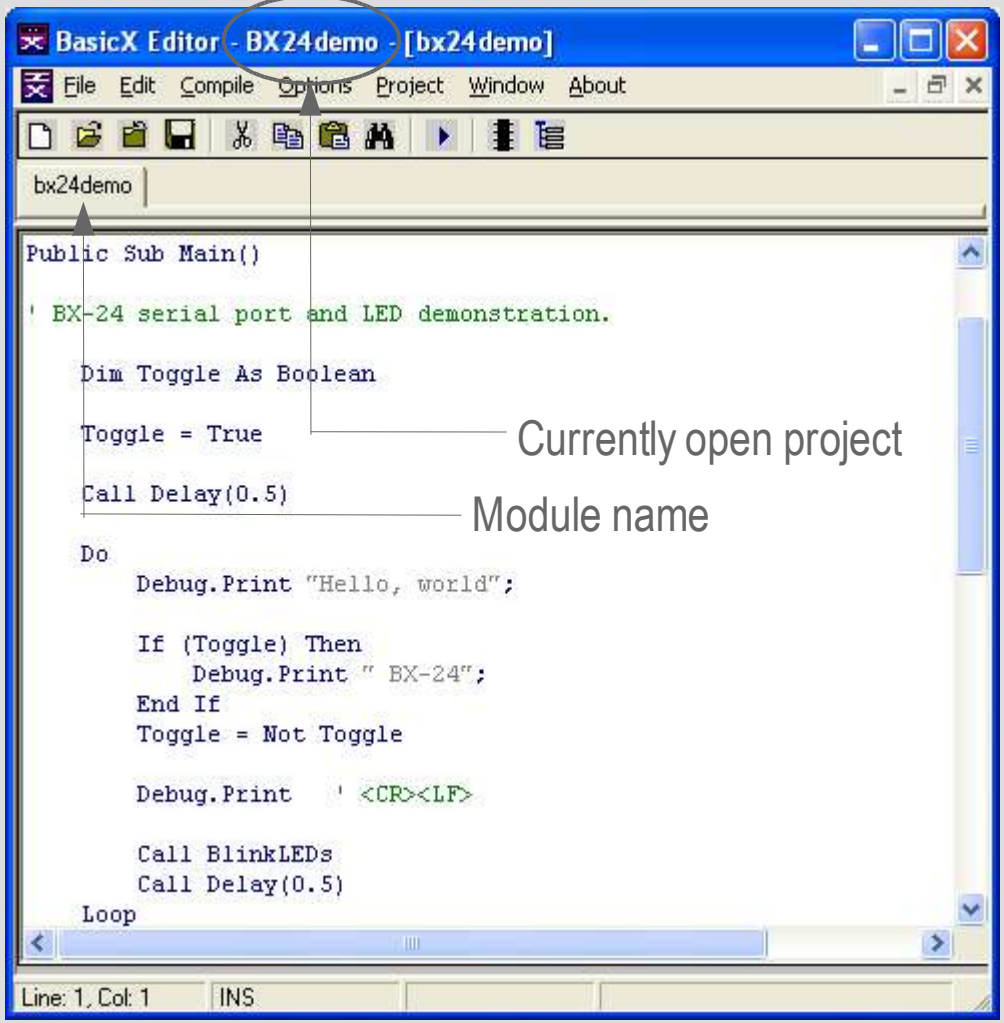
The *Basic Express* Compiler

- The **About** Menu



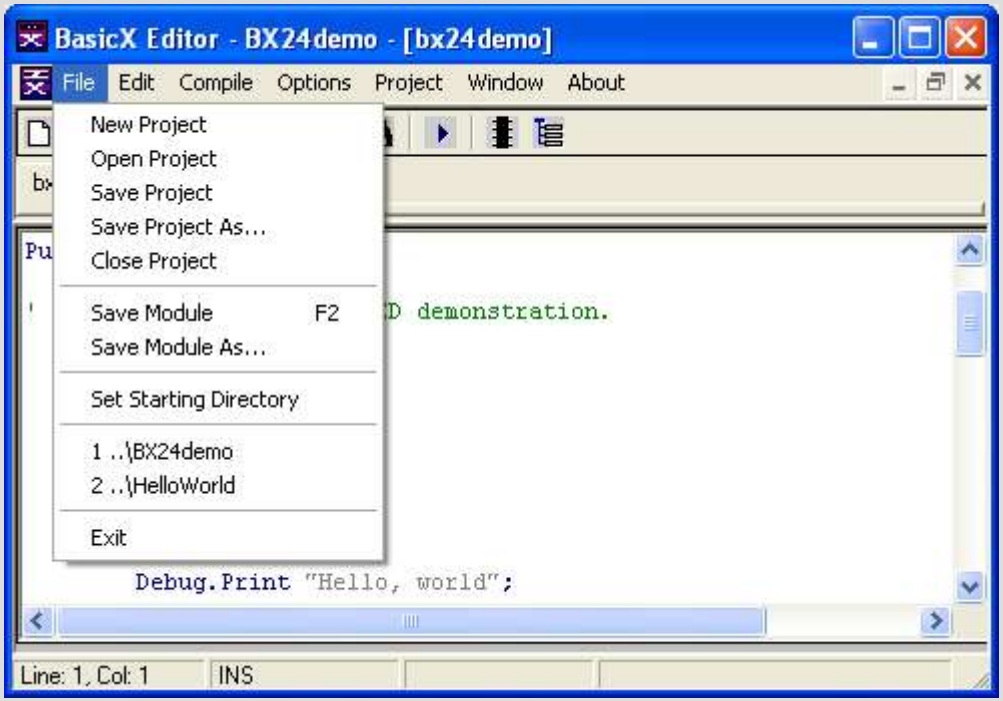
The *Basic Express* Compiler

- The BasicX Editor/Compiler



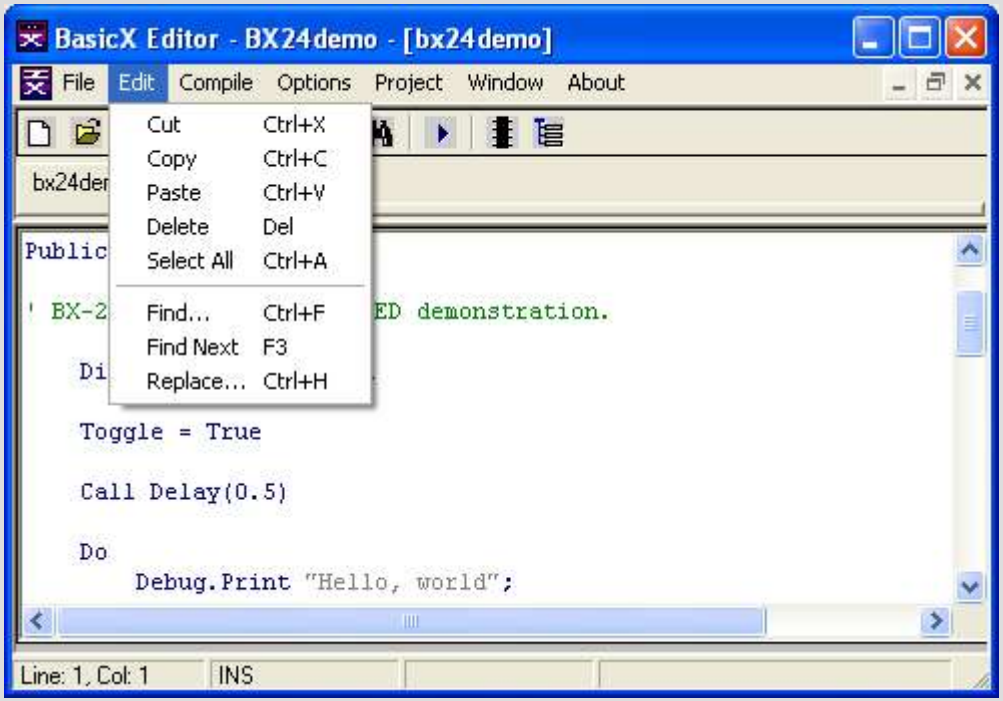
The *Basic Express* Compiler

- The BasicX Editor/Compiler
  - The **File** Menu



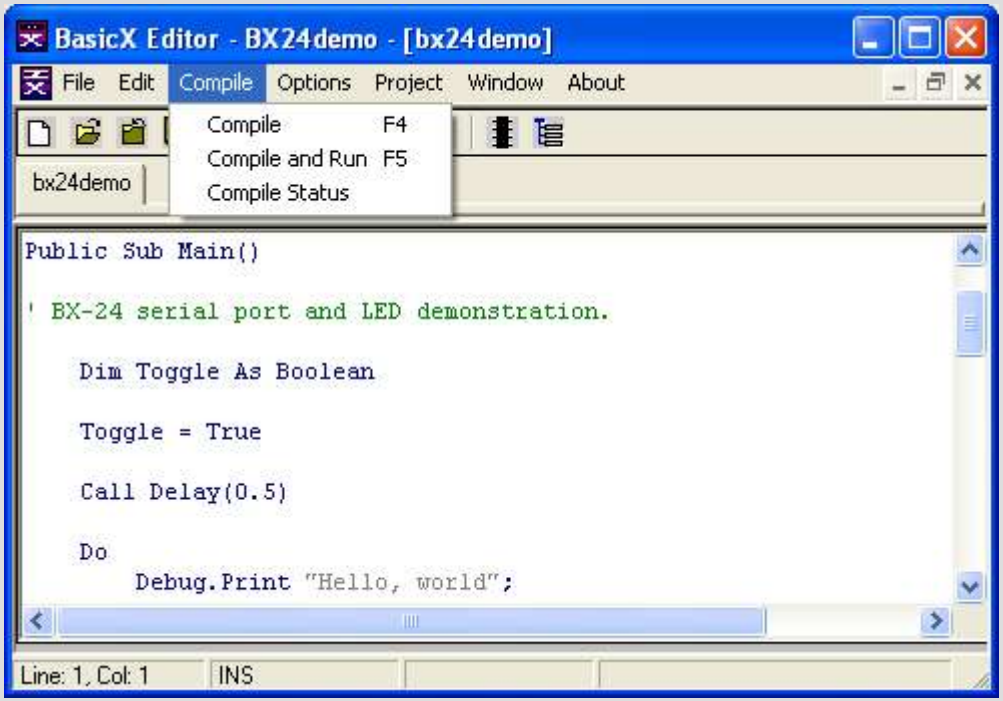
The *Basic Express* Compiler

- The BasicX Editor/Compiler
  - The **Edit** Menu



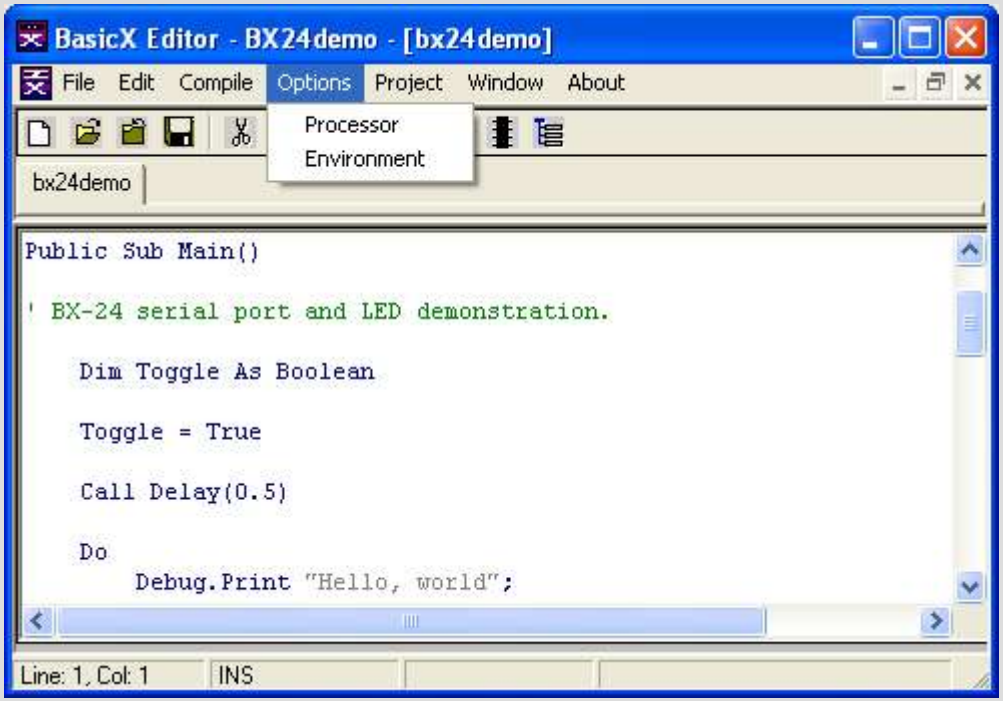
The *Basic Express* Compiler

- The BasicX Editor/Compiler
  - The **Compile** Menu



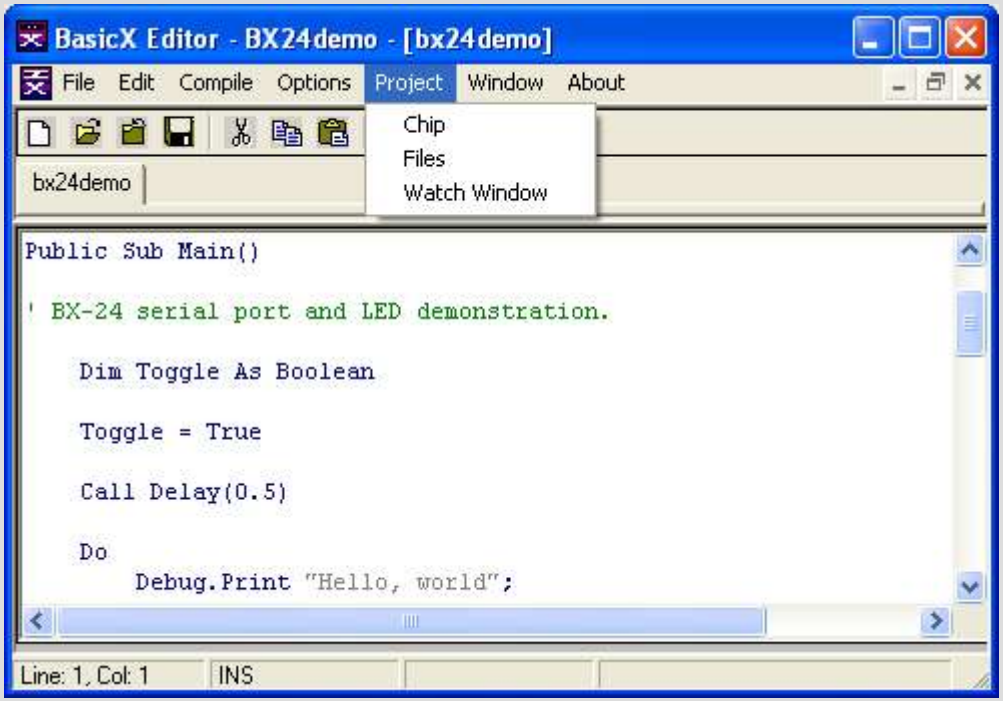
The *Basic Express* Compiler

- The BasicX Editor/Compiler
  - The **Options** Menu



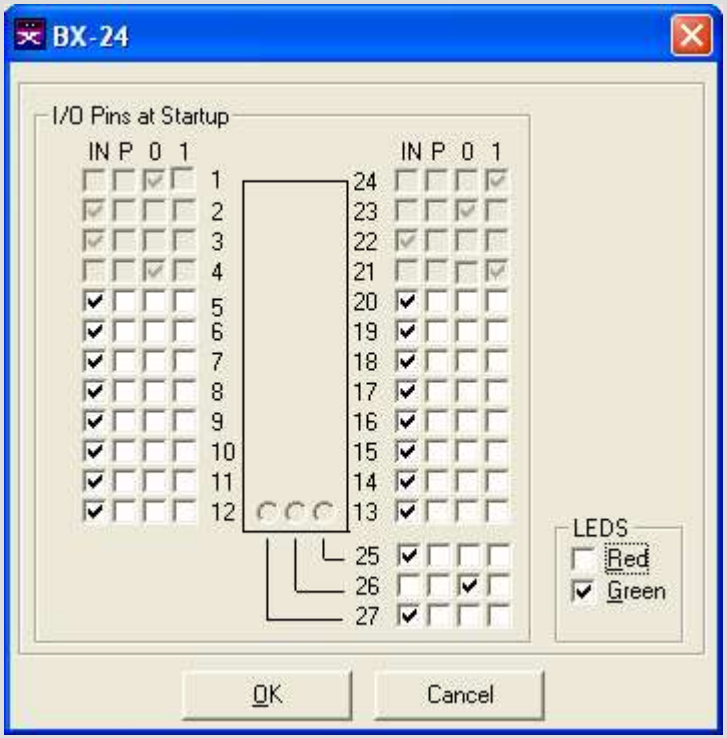
The *Basic Express* Compiler

- The BasicX Editor/Compiler
  - The **Project** Menu



The *Basic Express* Compiler

- The BasicX Editor/Compiler
  - The **Chip** Dialog Box of the **Project** Menu



Basic Express OS Reference

- Limitations on Persistent Variables:
  - Write cycle limits
    - Typically the EEPROM inside a BasicX chip is guaranteed for 100,000 write cycles; reading, however, is practically infinite.
  - Write time
    - Each byte takes approximately 4ms to write – much longer than a RAM-based variable.
  - Parameter passing
    - Persistent variables can only be passed by value.
  - Module level declarations
    - All persistent variables must be declared in module-level code.

Basic Express OS Reference

- Block Data Classes
  - Array initialization issues
    - BasicX provides the following system-defined block data classes (must be declared at module level):

      1-Dimensional array classes (byte only):

      ```
      ByteVectorData[RW]
      ```

      2-Dimensional array classes:

      ```
      [Byte|Integer|Long|Single]TableData[RW]
      ```

    - Example object declarations:

```
' B is a 1D byte array, read-only
Dim B As New ByteVectorData
' BRW is a 1D byte array, read-write
Public BRW As New ByteVectorDataRW
' S is a 2D float array, read-only
Private S As New SingleTableData
```

Basic Express OS Reference

- Block Data Classes
  - Source method
    - defines the data file from which an object gets its data; the file is read at compile time, then loaded into EEPROM at the same time the BasicX program is downloaded.
    - Example:

```
' B is a 1D byte array, read-only
  Call B.Source("ByteVector.txt")
' BRW is a 1D byte array, read-write
  Call BRW.Source("C:\Temperatures.dat")
' S is a 2D float array, read-only
  Call S.Source("CalibrationCurve.dat")
```

    - The **Source** method must be called before reading or writing the object's internal data.

Basic Express OS Reference

- Block Data Classes
  - Value property
    - 1D block data objects are treated similar to 1D arrays, where the index corresponds to the row number. Row numbering starts at 1.
    - 2D block data objects are treated similar to 2D arrays, where the first index corresponds to the column number and the second index is the row number. Column and row numbering starts at 1.
    - Note: A block data object is similar to a persistent variable in regards to write cycle limitations and the amount of time it takes to write to the object.

Basic Express OS Reference

- Block Data Classes
  - DataAddress property
    - The **DataAddress** property returns the starting EEPROM address of the object's internal data.
    - **DataAddress** is type **Long** and is read-only.
    - Example:

```
Dim T As New IntegerTableData, Addr As Long
Dim A1 As Integer, A2 As Integer
   [...]
Addr = T.DataAddress
' These two statements are equivalent
A1 = T(1,1)
Call GetEEPROM(Addr,A2,2)
' At this point A1 and A2 are equal
```

Basic Express OS Reference

- Multitasking
  - One of the most powerful features in BasicX is its ability to have multiple tasks running at the same time.
  - Multitasking programs typically need more RAM than programs with a single task.
    - Each task (other than the main program) needs its own explicit stack.
    - Each task stack is a byte array that must be located in module-level (static) code.
    - You may need to empirically determine the task stack size ...

Basic Express OS Reference

- Multitasking
  - In BasicX:
    - Tasks are timeshared on a first-come, first-served basis, except for tasks triggered by hardware interrupts.
    - Under normal conditions, tasks are switched every clock tick (the tick frequency is 512Hz).
    - A user can explicitly allow the next task to run with a `Call Sleep(0.0)` statement, which returns immediately if no other task is ready.
    - Tasks are like ordinary procedures without parameters; tasks are called with the **`CallTask`** instruction.
    - Refer to pages 13-19 of the *Basic Express Operating System Reference* for additional information ...

Basic Express OS Reference

- Semaphores
  - can be used to keep two tasks from using the same variable at the same time.
  - implementation, if written in Basic:

```
Function Semaphore(ByRef Flag As Boolean) As Boolean
  ' Is the flag available?
    If Not Flag Then
      ' Take possession of the flag
        Flag = True
      ' Tell the world we have it
        Semaphore = True
    Else
      ' Someone else has the flag
        Semaphore = False
    End If
End Function
```

Basic Express OS Reference

- Queues
  - useful as data buffers in serial communications.
  - ideal for transmitting data between tasks.
    - Queues are internally implemented as a circular buffer, and pointers for the queue are maintained within the queue itself.
    - Internal pointer overhead requires 9 bytes; hence, defining a 20 byte queue array leaves 11 bytes available for data.
    - Example:

```
Dim MyQueue(1 To 12) As Byte, B As Byte
Call OpenQueue(MyQueue,12)
Call  PutQueue(MyQueue,3,1)
Call  GetQueue(MyQueue,B,1)
```

Basic Express OS Reference

- Real Time Clock
  - The OS has a built-in Real Time Clock (RTC) that automatically keeps track of date and time.
    - A group of system calls is available to read or set the clock.
    - Example:

```
Dim Hr As Byte, Mn As Byte, Sc As Single
Call GetTime(Hr,Mn,Sc)
If (Hr = 21) And (Mn = 0) Then
    Call TurnOnIrrigation
End If
```

Basic Express System Library

- Math functions

`Abs`	Absolute value
`ACos`	Arc cosine
`ASin`	Arc sine
`Atn`	Arc tangent
`Cos`	Cosine
`Exp`	Raises *e* to a specified power
`Exp10`	Raises 10 to a specified power
`Fix`	Truncates a floating point value
`Log`	Natural logarithm
`Log10`	Logarithm base 10
`Pow`	Raises an operand to a power
`Randomize`	Sets the seed for `Rnd`
`Rnd`	Generates a random number
`Sin`	Sine
`Sqr`	Square root
`Tan`	Tangent

Basic Express System Library

- String functions

`Asc`	Returns ASCII code of a character
`Chr`	Converts a numeric value to a character
`LCase`	Converts a string to lowercase
`Len`	Returns the length of a string
`Mid`	Copies a substring
`Trim`	Trims leading and trailing blanks
`UCase`	Converts a string to uppercase

Basic Express System Library

- Memory-related functions

`BlockMove`	Copies a block of data from one RAM location to another
`FlipBits`	Generates mirror image of bit pattern
`GetBit`	Reads a single bit from a variable
`GetEEPROM`	Reads data from EEPROM
`MemAddress`	Returns the address of a variable or array
`MemAddressU`	Returns the address of a variable or array
`PersistentPeek`	Reads a byte from EEPROM
`PersistentPoke`	Writes a byte to EEPROM
`PutBit`	Writes a single bit to a variable
`PutEEPROM`	Writes data to EEPROM
`RAMPeek`	Reads a byte from RAM
`RAMPoke`	Writes a byte to RAM
`SerialNumber`	Returns the version number of a BasicX chip

Basic Express System Library

- Queue functions

`GetQueue`	Reads data from a queue
`OpenQueue`	Defines an array as a queue
`PeekQueue`	Looks at queue data without removing data
`PutQueue`	Writes data to a queue
`PutQueueStr`	Writes a string to a queue
`StatusQueue`	Determines if a queue has data available

Basic Express System Library

- Tasking functions

`CallTask`	Starts a task
`CPUSleep`	Puts processor in various low-power modes
`Delay`	Pauses task and allows other tasks to run
`DelayUntilClockTick`	Pauses task until next RTC tick
`FirstTime`	Determines whether program has been run
`LockTask`	Locks task; prevents other tasks from running
`OpenWatchDog`	Starts the watchdog timer
`ResetProcessor`	Resets and reboots the processor
`Semaphore`	Coordinates data sharing via semaphores
`Sleep`	Pauses task and allows other tasks to run
`TaskIsLocked`	Determine whether a task is locked
`UnlockTask`	Unlocks a task
`WaitForInterrupt`	Allows a task to respond to a hardware interrupt
`WatchDog`	Resets the watchdog timer

Basic Express System Library

- Type conversion functions

`CBool`	Convert **`Byte`** to **`Boolean`**
`CByte`	Convert to **`Byte`**
`CInt`	Convert to **`Integer`**
`CLng`	Convert to **`Long`**
`CSng`	Convert to floating point (**`Single`**)
`CStr`	Convert to **`String`**
`CuInt`	Convert to **`UnsignedInteger`**
`CuLng`	Convert to **`UnsignedLong`**
`FixB`	Truncate FP value, converts to **`Byte`**
`FixI`	Truncate FP value, converts to **`Integer`**
`FixL`	Truncate FP value, converts to **`Long`**
`FixUI`	Truncate FP value, converts to **`UnsignedInteger`**
`FixUL`	Truncate FP value, converts to **`UnsignedLong`**
`ValueS`	Convert to **`String`** to a float (**`Single`**) type

Basic Express System Library

- Real time clock functions

`GetDate`	Returns the date
`GetDayOfWeek`	Returns the day of week
`GetTime`	Returns the time of day
`GetTimestamp`	Returns the date and time of day
`PutDate`	Sets the date
`PutTime`	Sets the time of day
`PutTimestamp`	Sets the date, day of week, and time of day
`Timer`	Returns floating point seconds since midnight

Basic Express System Library

- Pin I/O functions

`ADCToCom1`	Streams data from ADC to serial port
`Com1ToDAC`	Streams data from serial port to DAC
`CountTransitions`	Count logic transitions on an input pin
`DACPin`	Generate pseudo-analog voltage at output pin
`FreqOut`	Generate dual sine waves on output pin
`GetADC`	Returns analog voltage
`GetPin`	Returns logic level of an input pin
`InputCapture`	Records pulse train on the input capture pin
`OutputCapture`	Sends pulse train to the output capture pin
`PlaySound`	Plays sound from sampled data in EEPROM
`PulseIn`	Measures pulse width on an input pin
`PulseOut`	Sends a pulse to an output pin
`PutDAC`	Send pseudo-analog voltage to an output pin
`PutPin`	Configure pin to 1 of 4 input or output states
`RCTime`	Measures time delay until pin transition occurs
`ShiftIn`	Shifts bits from an I/O pin into a byte variable
`ShiftOut`	Shifts bits out of a byte variable to an I/O pin

Basic Express System Library

- Communications functions

`Debug.Print`	Sends **`String`** to Com1 serial port
`DefineCom3`	Defines parameters for serial I/O on arbitrary pin
`Get1Wire`	Receives data bit using Dallas 1-Wire protocol
`OpenCom`	Opens an RS-232 serial port
`OpenSPI`	Opens SPI communications
`Put1Wire`	Transmits data bit using Dallas 1-Wire protocol
`SPICmd`	SPI communications
`X10Cmd`	Transmits X-10 data

References

- BasicX-24 microcontroller website, http://basicx.com
- *Intelligent Systems Lab* website, http://isl.ecst.csuchico.edu
- NetMedia Inc. website, http://netmedia.com
- NetMedia, 2002. *Basic Express Compiler User's Guide, v 2.0*.
- NetMedia, 2002. *Basic Express Language Reference, v 2.0*.
- NetMedia, 2002. *Basic Express Operating System Reference, v 2.0*.
- NetMedia, 2002. *Basic Express System Library, v 2.0*.
- NetMedia, 2002. *BX-24 Hardware Reference*.
- TAB Electronics, *Build Your Own Robot Kit* website, http://www.tabrobotkit.com