Physical Computing 2000
Course material

A cooperation between Smart at the Interactive Institute and Industridesign at Konstfack

Smart
Interactive Institute

Version 1.0

6 november 2000

Contents

PHYSICAL & PERCEPTUAL COMPUTING 2000 .... **5**

ABOUT THE WORKSHOP 5
Contact information *5*
Online resources *5*
Physical computing *5*
Senses and sensors *5*
Outline for the workshop *6*

CRASH COURSE IN ELECTRONICS AND PROGRAMMING **7**

SOMETHING ABOUT ELECTRICITY 7
Voltage (spänning) *7*
Current (ström) *7*
Resistance (motstånd, resistans) *8*
Ohm's law and short circuits (kortslutning) *8*
Power (Effekt) *8*
Potential and ground *9*
Table of measurement units *9*
Table of prefixes *9*
Binary and digital data *9*
Analog values *10*
D/A conversion *11*
Electronic components *11*
SOMETHING ABOUT PROGRAMMING 11
Program *11*
Programming language *11*
Source code, compilation, executable *11*
Variables *12*
Comments *13*
Subroutines *13*
Call *14*
Loops *14*
Do – Loop (Do–While, Do–Until) *14*
For – Next Loops *14*
If – then – else statement *15*
PROGRAMMING AND ELECTRONICS 15
putPin *16*
getPin *16*
pulseOut *17*
freqOut *16*
getADC *16*
Delay *16*

COMPONENTS & SYMBOLS **19**

RESISTOR 19
CAPACITOR (KONDENSATOR) 19
TRANSISTOR 19
DIODE 19
GROUND LED (LYSDIOD) 19
SENSOR & POTENTIOMETERS 20
PHOTO RESISTOR 20
THERMISTOR 20
FLEX SENSOR 20
ON/OFF SWITCH 20
BUZZER 21
SERVO MOTOR 21

THE BASICX DEVELOPMENT SYSTEM **23**

Installing the software *23*
Setting up the hardware *23*
OPENING THE ENVIRONMENT 23
THE MAIN WINDOW 24
STARTING A NEW PROJECT 24
Closing a project to start on another. *24*
Setting the download port *25*
Setting the monitor port *25*
THE EDITOR WINDOW 25

THE BX24 CHIP 27

BX24 pin configuration *27*
Connecting the BX24 .... *27*

EXAMPLE PROJECTS 29

DEMO_LED 29
DEMO_POT 29
DEMO_SERVO 30
DEMO_SPEAKER 30
DEMO_SWITCH 31
on/off switch with external pullup *31*
DEMO_BUZZER 32

LEARNING MORE 33

WHERE TO BUY STUFF 33

Physical & perceptual computing 2000

About the workshop

Physical & perceptual computing (”physcomp 2000”) is a one-week workshop. It is also a part of the larger course **Formgiving of Digital Artifacts that Stimulate Physical & Psychological Activities for Elderly**. This is the course material only for the physcomp workshop, not for the entire course.

The workshop takes place at the Interactive Institute in Stockholm, at Karlavägen 108, 5th floor. It begins on 14 November (note that this is a Tuesday!) end ends with a small presentation on Monday 20 November. You will be working in groups of 3–4 persons each.

The objective of the workshop is to make you familiar with smart technology in the form of embedded electronics and simple sensor technology, and at the same time expand your perceptual awareness and visual thinking.

Contact information

The workshop is lead by Fredrik Petersson (Smart), Cheryl Koler Akner (ID) and Konrad Tollmar (Smart).

Fredrik:	fredrik.petersson@interactiveinstitute.se	08 – 783 24 71
Konrad:	konrad.tollmar@interactiveinstitute.se	08 – 783 24 61
Cheryl:	cheryl.akner.koler@chello.se	08 – 450 41 75

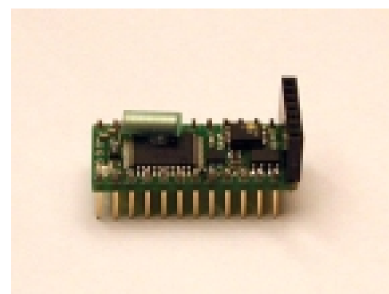
Online resources

The online documentation for the workshop can be found at http://smart.interactiveinstitute.se/konstfack

There you will also find the schedule for the workshop as well as for the entire course.

Physical computing

This workshop deals with physical computing. This means that we are not writing computer programs for desktop PCs. We will be using PCs to write the programs, but then we will put the program on a small microcontroller where the program will run. Once we’ve done that, we can take away the desktop computer and only have the microcontroller. This way we can put the computing power in everyday objects.



The BX24 Microcontroller

Senses and sensors

Throughout the workshop there is a theme we call “senses and sensors”. This is an attempt to make you think of smart technology in the context of everyday life. The smart objects are situated in the real world, and they can react to “real world” conditions, and express themselves in “real world” expressions.

The workshop will end with a “senses and sensors project” where you will use what you learn during the first three days to build something yourselves.

Outline for the workshop

The workshop will contain several short talks by various people to explain what it is all about and to inspire you to be creative.

Other than that, the course will proceed as follows:

- Day 1 – crash course in electronicks and programming. This will be a quick introduction to electronics and programming. It will not cover everything in the ”crash course” section in this written material.
- Day 2 – Light, Touch and movement. These are two parts that will contain examples and ”play time” dealing with electronics for light, touch and movement.
- Day 3 – Sound. The same as day two, but focusing on sound. On the afternoon there will be an introduction to the ”sensors and senses” project.
- Day 4 – Working with the “sensors and senses” projects. These are group projects that focus on your own work.
- Day 5 – Finishing and presenting the “sensors and senses” projects. You will also document what you have done. The presentations will be in form of a poster and web page.

Crash course in electronics and programming

Wouldn't it be cool if you could make things that ***did stuff***? Build little robotic toys for the cat to play with, build a light that changes color with the temperature, or why not an automatic vacuum cleaner.

Unfortunately, doing this almost inevitably requires learning some "tech" things, such as electronics and programming. That might sound difficult and boring – and to some extent, it might well be. There really is a lot to learn, and it is a great deal of work. But there are things you can do quite easily, and you're almost guaranteed to have fun. And that's what this workshop is about. It's going to be more of a "hands on" workshop than a theoretical course. (Except for this chapter!)

In this section – the "crash course" – we'll try to at least give you a hint of what things are and how they work. You don't have to learn everything, but this should give you enough information to get you started and give you a vocabulary so that you can ask somebody. After reading this you should at least be able to tell a Volt from a Watt and a bit from a byte.

This section assumes you have no knowledge whatsoever about this. You might want to skip ahead and return here when you start mixing things up.

Something about electricity

Voltage (spänning)

Pick up an ordinary battery. It says, for example, 4.5 V. You might know that we have 230 V (we used to have 220 V) in the electrical outlets in the wall. What does this mean?

Clearly, 220 V is more than 4.5 V. You can touch a battery without feeling anything at all, but you really should avoid sticking nails in the wall outlets. This is mainly because there is a higher **voltage** in the electric outlets than in a battery. Voltage is measured in Volt.

As you probably know, electricity doesn't spill out of the wall outlets onto the floor – even though there is a voltage across the two holes in the wall. **Nothing happens with just a voltage until you plug something in**. You can think of a voltage as a container of water – nothing happens until you open the tap and the water starts flowing.

Current (ström)

The next thing we pick up is a Light Emitting Diode (LED) – the tiny lights that you see everywhere in electronic equipment. Just holding it between your fingers will not make it light up, that much we can agree on. What if you attach one end of the LED pins to the battery? Try it – nothing should happen. The LED will not light up until you connect BOTH pins to the battery. **You must always close the circuit for the current to flow**. There must be a "loop" where the current can go from plus, through the components, and down to ground. When you are using the BX24, this will sometimes act as the final point. (So you might connect something from a pin to ground, for example.)

Also, you must turn the LED the right way: one leg is a little longer than the other one. The longer leg should go to the plus pole of the battery, and the shorter leg to the minus pole.

Current is measured in Ampere (A).

Resistance (motstånd, resistans)

The third and last of these ”basic units” is resistance (motstånd). Think of resistance as a property of a material that controls how easy it is for a current to flow. Some materials – isolators – have very high resistance. These include e.g. rubber, paper, porcelain, and air. Because air has a high resistance, it will be difficult for a current to flow through air. We can think of "no connection" as infinite resistance.

Some other materials – mainly metals – are called conductors. They have low resistance. The lower the resistance, the more current will flow. We can think of a metal wire as "zero resistance". Some other components will have something in between. There are components called resistors, which have a determined resistance, as in a "220kOhm resistor". Resistance is measured in Ohm (Ω).

Ohm's law and short circuits (kortslutning)

There is a simple equation called Ohm's law:

Voltage = Resistance * Current $U = R * I$

We can also write this as $I = U / R$.

We will not be using this to calculate actual values on our circuits, but it is good to remember "the less resistance, the more current".

In particular, consider what happens when the resistance is zero. (for example, 10 Volt divided by 2 Ohm would give you 5 Ampere, 9 Volt divided by 0 Ohm would give you infinite current…)

If you connect a wire directly from plus to ground with no resistance you create a ***short circuit*** (kortslutning). Try not to do this!

Actually, if you tried connecting a LED to a 9V battery, you would notice that it becomes hot and it might shine yellow even if it's a green LED. This is because there is too much current going through it. It might destroy the LED, but probably you will just use up the charge from the battery quite fast. What you should do is connect a resistor in series with the LED. This will reduce the current (the more resistance, the less current) and the LED will work the way it is supposed to.

Power (Effekt)

We will not talk much about this, but power is another fundamental unit in electricity. For example you might buy a 60–watt light bulb. This is a measurement of the power it will have at 230 Volt. A simple equation is

power = voltage * current $P = U * I$

Potential and ground

Potential is really pretty much the same thing as voltage. A voltage is always measured across two points: you have 9 Volts between the poles on a battery. A potential is measured at a single point.

You do not need to worry too much about this. However, it is good to know that ”ground” (jord) always has the potential 0 Volt.

Table of measurement units

This table collects the different measurement units used to measure different quantities, and also lists the Swedish translations.

Quantity	På svenska	Units
Voltage (U)	Spänning	Volt (V)
Resistance (R)	Resistans, motstånd	Ohm (Ω)
Current (I)	Ström	Ampere (A)
Power (P)	Effekt	Watt (W)
Potential (V)	Potential	Volt (V)
Ground	Jord	Potential = 0

Table of prefixes

In all sorts of measurements you use prefixes to change the size of the units. We measure things in millimeter (mm) or kilometer (km), and we can do the same thing with milliampere (mA) or kilowatt (kW). Note that there is difference between a capital M (for Mega) or a small m (for milli). You never have more than one prefix. Actually this table is not complete, there are some smaller and bigger prefixes.

Tera (T)	1 000 000 000 000	10^{12}
Giga (G)	1 000 000 000	10^{9}
Mega (M)	1 000 000	10^{6}
kilo (k)	1000	10^{3}
-	1	10^{0}
milli (m)	0,001	10^{-3}
mikro (µ)	0,000 001	10^{-6}
nano (n)	0,000 000 001	10^{-9}
piko (p)	0,000 000 000 0001	10^{-12}

Binary and digital data

You might have heard that “computers only count in ones and zeroes”. This is actually true – but a computer can still count to a hundred, or use a value like 3.141592654. How is this possible if the computer only uses ones and zeroes?

The answer is that **the computer uses combinations of ones and zeroes to represent other data**.

For example, the letter “A” could be represented by the binary digits 01000001. A particular shade of pink is represented as 111111111001100110011001. Luckily you do not need to use the numbers directly, but internally in the computer everything **is** represented as just ones and zeroes. The ones and zeroes are called **binary digits**, or **bits**. One bit can only be 0 or 1 – nothing else. If you use two bits in a row, you get 4 combinations:

00	0
01	1
10	2
11	3

If you add a third bit (the leftmost bit in this table), you get eight combinations:

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Note that the two bits to the left just repeat the pattern from the two-bit table. You can repeat this process of adding another bit. Each time you add one more bit you double the number of combinations you can make.

Number of bits	**Number of combinations**	
0	0	(2^0)
1	2	(2^1)
2	4	(2^2)
3	8	(2^3)
4	16	(2^4)
5	32	(2^5)
6	64	(2^6)
7	128	(2^7)
8	256	(2^8)

This is how the computer uses binary digits (ones and zeroes) to represent integer numbers (heltal). Using clever tricks you can also use binary digits to represent decimal numbers like 1.4142.

If you let each letter of the alphabet be represented by a value (A=65, B=66, C=67…) you can represent text as integer numbers. And like we just saw, binary numbers (0, 1) can represent integer numbers (0, 1, 2, 3…). So letters can also be represented by binary digits. This is what we call digital data – every type of information that can be stored as ones and zeroes.

Analog values

Letters are already ”countable”. There is a finite number of letters and you an just assign a code to each one. But there are other types of information, for example sound, light, colors, and temperature. If somebody asks you how hot it is outside, you might answer “10° C”. But this is really just an approximate answer. If you had a very accurate thermometer, it might read 9,045° C. The “real” temperature is an analog value.

Other questions would be “how light is it here” or “how loud is this sound”.

D/A conversion
The BX24 microcontroller has a number of inputs where you can connect a light sensor or a temperature sensor. You can then get a value on "how light it is" or "how hot it is". The BX24 is a small computer, which only handles binary data, so when it reads analog values from the pins it converts it to digital data. **Digital to analog (D/A) conversion is the process of measuring a "real world" analog value and representing it as a digital value.**

Electronic components
So far we only talked about electricity in general. In the section "components" we will look in detail at some electronic components you can use to build stuff.

Something about programming
We have already started talking about bits and bytes, which really is more about programming than about electricity. Here we turn away from the hardware to the software.

Program
A program is a set of instructions that tell a computer what to do. You can think of it like cooking instructions:

> Pour water in pot
> Heat the water until it boils
> Turn off the heat
> Put the cous-cous in the water
> Add butter
> Stir
> Wait 2 minutes

Computers are really, really stupid, so you need to be extremely precise when you ask them to do something. This is what programmers do all day.

When you let a computer follow the instructions in the program you say the program is **executing** or **running**.

Programming language
When you write a program you have to use specific rules and commands. The set of commands you can use is determined by the "programming language". Some examples of programming languages are Java, C++, Lingo and Basic. Many of these are similar, but the programs will look slightly different, and some things are easier to do in certain programming languages. When we program the BX24 we will be using a version of Basic.

Source code, compilation, executable
The programming languages usually look more or less like English. The program you write is called the "source code". In many programming languages the computer cannot understand this directly. There is something called a **compiler** that looks at the source code and translates it ("compiles it") to a code the computer can understand. The result is an **executable** program.

Variables

Programmers sometimes explain what a variable is by saying, ”it’s pretty much the same thing as in algebra”. This is true, but generally unhelpful if you never do algebra.

Let’s try using an example instead: we “program” a fictional guy named Gerald to read the temperature from a thermometer and then tell us how warm it would be if it were ten degrees warmer. The program would be something like

> Read the current temperature
> Add 10
> Tell us what it is

So if it is 12°C outside, Gerald would read this, add 10 and tell us “22”.

The catch here is the word “it” on the last row. When you write bigger programs, you might mix up which “it” you mean. So instead, we give this “it” a name. Let’s call the temperature “T”.

The program would then be something like

> T = the current temperature
> T = T + 10
> Tell us what T is

Gerald would still tell us “22”.

You can have longer variable names than just a single letter. Maybe you want to call the temperature “temp” or even “temperature”. It doesn’t matter, as long as it is a single word with no spaces in it.

Variable types – dim … as …

When you program the BX24 you must define your variables before you use them. This is done using the ”dim – as” command. There are several variable types available. What you need to think about is what the variable will be used for – will it store text (”Hello”), integer values (0, 1, 2, 3 …) or decimal values (3.145)?

The call is

```
dim variableName as variableType
```

where variableName is the name of the variable (T or temp or whatever you are using), and variableType is the type of the variable.

Variable type	**Storage**	**Range**
Boolean	8 bits	True or 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	- 3.402 832E+38 … 3.402 823 E+38
String	Varies	0 to 64 characters

You will probably mostly use Integer and Single. Whenever you want the variable to contain decimal values it should be a Single, else an Integer probably works just fine.

Some actual examples of the dim–as statement:

```
dim PinNumber as Byte
dim userName as String
dim temp as Integer
```

Comments

When you are writing programs it is a good habit to document what you do. You can do this directly in the source code.

By typing an apostrophe (’) the rest of the line will be treated as a comment. This means you can type whatever you like there, the compiler will just ignore this.

Warning – do NOT use any Swedish characters (åäö) in the comments, as this will confuse the compiler.

```
' This is an example of a comment.
```

Subroutines

A subroutine is a part of a program. If you write large programs it is always a good solution to try to divide the program in smaller parts and solve each part separately. These parts are called subroutines.

When you start writing a program for the BX24, two lines will be written automatically:

```
Sub Main()

End Sub
```

If you write a small program you will just write the rest of the code between these two lines. (Look at the examples!)

When the program starts running it will look for the “Main()” subroutine and start there.

Writing a subroutine

You probably do not need to write subroutines for your smaller programs, so this information is aimed at the more advanced programmers. You can safely skip this.

The structure for a subroutine is

```
Sub procedure_name (arguments)
      [statements]
End Sub
```

And then you call the subroutine using the call statement:

```
call procedure_name (arguments)
```

Here’s an example of how to write a subroutine:

```
Sub GetPosition(ByRef X as Single)
      ' some more code here
End Sub
```

And you would call this using

```
call GetPosition(17)
```

Call

The Call statement is used to jump to a subroutine. (See above). The program will jump to the start of the named subroutine, run it, and then return to the line after the call statement.

Loops

Sometimes you want to do things more than once. The first use of this you will see in the programming examples is when we just want a program to keep running forever using the infinite do-loop.

Do – Loop (Do–While, Do–Until)

The infinite loop looks like this:

```
do
      [statements]
loop
```

It will just repeat the statements within the loop forever. Look at the examples!

You can also have conditions on the loop:

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

or

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

An example of this would be

```
Do While (light < 50)
      call beep
      light = getLightValue()
Loop
```

For – Next Loops

Another type of loop is a loop that loops a number of times.

```
For loop_counter = start_value To end_value
      [statements]
Next
```

An example of this would be

```
Sub Main()

     dim i as Integer
     for i = 1 to 10
          debug.print "This is loop " & cStr(i)
     next

End Sub
```

This example will print "this is loop 1", "This is loop 2" … "This is loop 10"

If – then – else statement

The if – then – else statement also exists in virtually every programming language. It is used to check a condition and act differently under different conditions.

The syntax is

```
If (boolean_expression) then
      [statements]
End if
```

You can also have an "else" clause that specifies what to do if the expression is NOT true:

```
If (boolean_expression) then
      [statements]
Else
      [statements]
End If
```

Finally you can have several conditions using the Elseif command

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

An example would be

```
if (state=3) then
      call beep()
elseif (state = 2) then
      call flashLed(100)
Else
      call flashLed(50)
end
```

Programming and electronics

The beauty of working with a BX24 is that it is so easy to write programs that use the hardware you connect.

These are some more specific commands to read input from the sensors or put signals on the output pins.

Debugging

"Debugging" is the process of removing programming errors ("bugs") from the source code. The BX24 command "debug" can help you do this.

The call is

```
debug.print string
```

An example would be

```
debug.print "Hello!"
```

The trick here is to use the cStr() command which converts an integer to a string. Say you have a variable called temp, and you want to know what the value is.

```
Then you would call
      debug.print "Temp is now " & cStr(temp)
```

The ampersand (&) character tells the compiler to join the two strings.

Note that you must open the monitor port from the main window for the debugger to work.

Delay

This causes the program to pause for a determined time.

```
Call delay(interval)
```

Interval is the time to wait, as a Single, measured in seconds. Resolution is about 1.95 ms and the interval is 0.0 to 127.0 seconds.

freqOut

This will generate an analog signal that consists of two superimposed sine signals. The procedure halts all multitasking for the duration of the call.

```
Call freqOut(Pin, Freq1, freq2, Duration)
```

Arguments:

pin	output pin number (byte)
Freq1	Frequency 1, Integer, in Hertz
Freq2	Frequency 2, Integer, in Hertz
Duration	Duration of the signal. If Duration is a Single it is measured in seconds. If Duration is an Integer it is measured in milliseconds. Range is 1 ms to 2560 ms.

getADC

This performs an A/D conversion on the voltage on one of the pins 13-20.

```
Voltage = getADC(Pin)
```

It returns an Integer value from 0 to 1023. For 5V systems, units are in 5/1023 volts (about 4.89 mV)

You will use this command to read information from all the sensors. See the example demo_pot for the details on how to connect the stuff and program it.

getPin

```
F = getPin(pin)
```

This reads the state of a pin, and will return either 1 or 0 (as a byte). If the voltage on the pin is zero (the pin is grounded) you will recieve a 0; else you should get a 1.
See the example demo_switch for notes on how to use this.

pulseOut

This sends a logic high or low pulse from any available I/O pin. It halts all multitasking during the call.

```
Call PulseOut (Pin, PulseWidth, State)
```

Arguments:

Pin	The Pin number, as a byte
PulseWidth	The time interval, as a Single, measured in seconds. Range is about 1.085 µs to 71.1 ms.
State	Specifies high (1) or low(0) pulse, as a byte.

putPin

```
putPin (pin, state)
```

Where pin is the pin number (as a byte) and State is a value from 0 to 3 or one of the symbolic naes for these values:

bxOutputLow	0	Output Low (typically 0 Volts)
bxOutputHigh	1	Output High (typically 5 Volts)
bxInputTristate	2	Tristate, high impedance
bxInputPullup	3	Pull–up, on chip 120 kOhm pull–up

You will probably only use 0 or 1 directly to set the output to high or low. If you are going to use a pin as an input from a swich you might want to set it to the bxInputPullup state. The example demo_switch explains this in more detail.

Components & symbols

Here we will present some of the components you will be using. There are also pictures that show the component and how to draw the schematic symbols for the component.

Resistor

The resistor is a very basic component. They come in varying shapes and sizes, but mostly they look like a small cylinder. There are usually colored rings around the cylinder. This is actually a color code that will tell you what the resistance is. you connect a resistor wherever you want to increase the resistance. It does not matter which way you turn it, there is no ”plus” side of a resistor.

Capacitor (kondensator)

The capacitor is a component that can store some electrical charge. The size of a capacitor (the capacitance) is measured in Farad. They can be used in various types of filters and other circuits. The minus side of the capacitor is usually marked.

Transistor

There are several types of transistors. The thing they have in common is that they can work as an amplifier or as switch. We will mainly be using them as switches. This is useful when we have a component that draws too much current for the BX24 to handle. Then we connect the component to a transistor and the transistor to the BX24. (Look at the ”demo_buzzer” example)

Hardware

The transistor has three legs. These are called collector, base, and emitter. Connect the collector to plus and the emitter to the component. Connect the base to a pin on the BX24.

Software

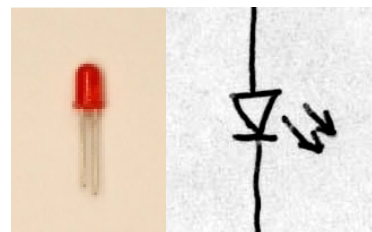
By putting ”High” on the base, the transistor will switch on and a current will flow from the collector to the emitter.

```
putPin(pin, 1)          ' Will turn the transistor on
putPin(pin, 0)          ' Will turn the transistor off
```

Diode

Ground LED (lysdiod)

The Light Emitting Diode (LED) will emit light when enough current runs through it. It has two legs, one of which is slightly longer. The longer leg is the ”plus” side, which connects to the higher voltage. The shorter leg connects to ground.



LED

Hardware:

Connect the longer leg of the LED to any pin (5-20). Connect the shorter pin of the LED to a 220 Ohm resistor. Connect the resistor to ground.

Software:

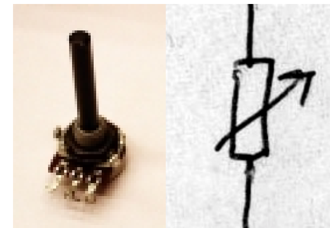
```
call pinOut(pin, 1)          ` lights the LED
call pinOut(pin, 0)          ` turns off the LED
```

Where *pin* is the number of the pin (5 to 22), so the actual call would be e.g `call pinOut(12,1)`.

Also see the example demo_led.

Sensor & potentiometers

The potentiometer is a resistor where the resistance is not a fixed value but depends on how you turn the knob, like in a volume control.



Potentiometer

Hardware

Connect the sensor from the "plus" voltage to one of the pins ON THE RIGHT SIDE. (pin 13–20). You can not use the pins on the left side (pin 5–12) for this. Connect a 220 resistor from the same pin to ground. When you turn the knob on the potentiometer the call to getADC will return different values.

Software

```
dim sensorValue as int        ' sensorValue is an integer variable

sensorValue = getADC(pin)     ' read in the current value
debug.print "The sensor value is " & cStr(sensorValue)
```

Also see the example demo_pot.

Photo resistor

The photo resistor can be connected exactly like the potentiometer. Instead of turning the knob it will measure how light it is.

Thermistor

The thermistor can also be connected like the potentiometer. It will return values that depend on the temperature. (These may not be as sensitive as the photo resistors).

Flex sensor

Again, the flex sensor can be connected exactly like the potetniometer. It will return different values depending on how much you bend it. It is more sensitive in one direction than in the other.

on/off switch

It is very simple to connect on/off switches to the BX24.

Switch

Hardware

Just connect a button from a pin to ground.

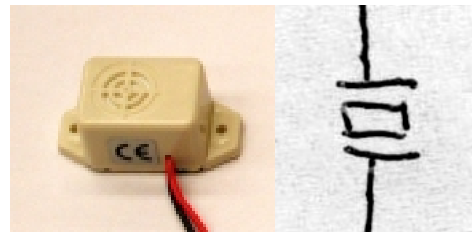
Software

```
dim state as byte
call putPin(12, bxInputPullup)
state = getPin(12)
```

Also see the example demo_switch.

Buzzer

A buzzer is a device that makes a sound (a ”beep” or ”buzz”) when a current runs through it. Some buzzers can be connected directly to a BX24 pin (like a LED), but some draw too much current and need to be connected through a transistor.



Buzzer

See the example demo_buzzer.

Servo motor

The servo motors are small motors that will swing to a given angle when you tell them to.

Hardware

The ones we are using have three cables attached. The red cable goes to plus, the black to ground, and the yellow to the BX24.

You control the servo by putting a pulse on the pin. The pulse width will tell the servo where to. The pulse width should be between 1 and 2 ms.

Software

```
call pulseOut(pin, pulseWidth, 1)
```

The BasicX Development System

The BasicX Development System Environment is the programming environment (the ”programming program”) you use to write programs for the BX24.

Here’s how it works:
1. You write a program in the editor window on your PC
2. You “compile” the program (translates it to a code the BX24 can understand)
3. You download the program from the PC to the BX24
4. You run the program on the BX24 (you can take away the PC now – the BX24 runs the program itself!)

Installing the software

This is already done on the computers you will be using in the workshop. This information is here if you want to continue playing with the BX24 yourselves.

All you need to do is download the software from www.basicx.com. (The files are under ”BasicX-24 Sofware download” in the menu. Locate a link that says ”BasicX Software for WIN95/98/NT Setup as 1 File - V1.46.6”. Save the file to disk, unzip the archive and run the setup program. A ”BasicX Environment” icon will appear on the desktop.

It seems there is no development kit for the Macintosh, only for Windows PCs.

Setting up the hardware

The hardware we are using in the workshop is primarily this:
- A Windows PC that works and has a spare serial port
- A power source that delivers about 7.5V DC. (Actually anything from 6 V to 15 V)
- The prototype board
- The serial cable from the PC to the prototype board (ask us how to do this)
- The BX24 microcontroller
- An assortment of sensors (primarily photo sensors, flex sensors and microphones), LEDs, transistors, servomotors, resistors, and capacitors.

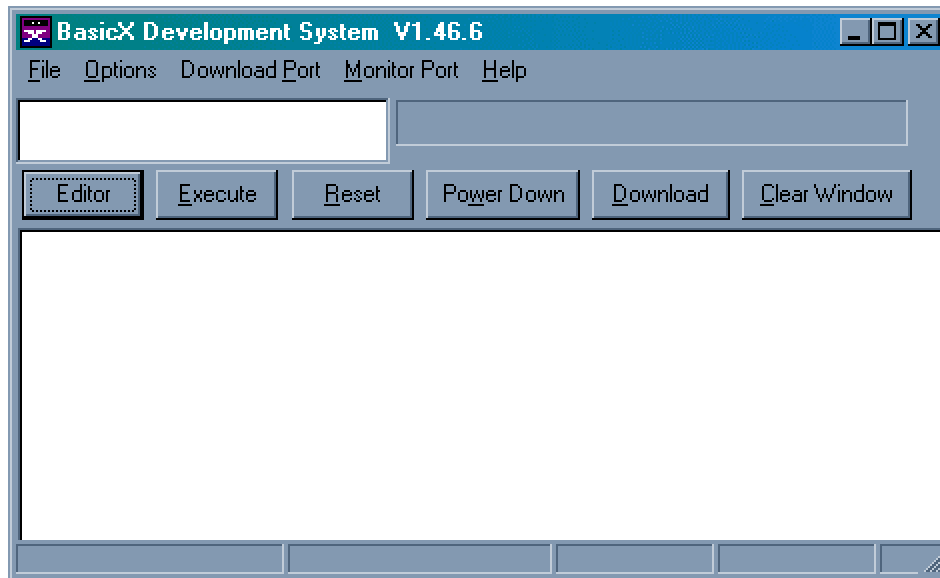
See the heading “Connecting the BX24” for details on how to connect the BX24.

Opening the environment

Double-click on the “BasicX Environment” icon.
This will open the main window.

BasicX Environment icon

The main window



The Development system main window.

From left to right, there are a number of buttons.

- Editor — opens the editor window, where you write the actual program.
- Execute — starts running a program on the BX24 after download
- Reset — stops a program that is running on the BX24
- Power Down — powers down the BX24 (I never use this)
- Download — downloads a compiled program from the PC to the BX24
- Clear Window — clear the window

Starting a new project

Every program you write will be called a "project". A project contains the program you write, and also some other files that will be automatically created. You do not need to worry about this, just remember to create a new project each time you want to start working on something else.
The easiest way to start a new project is to click the "Editor" button in the main window.

The environment will ask you to open an existing project. Instead, just type in the name of your new project. The environment will tell you this project does not exist and ask if you want to create it. Answer "yes".

The Editor window will open. This is where you write your program.

Closing a project to start on another.

When you want to start on something else, just close the editor window before you open a new project.

Setting the download port

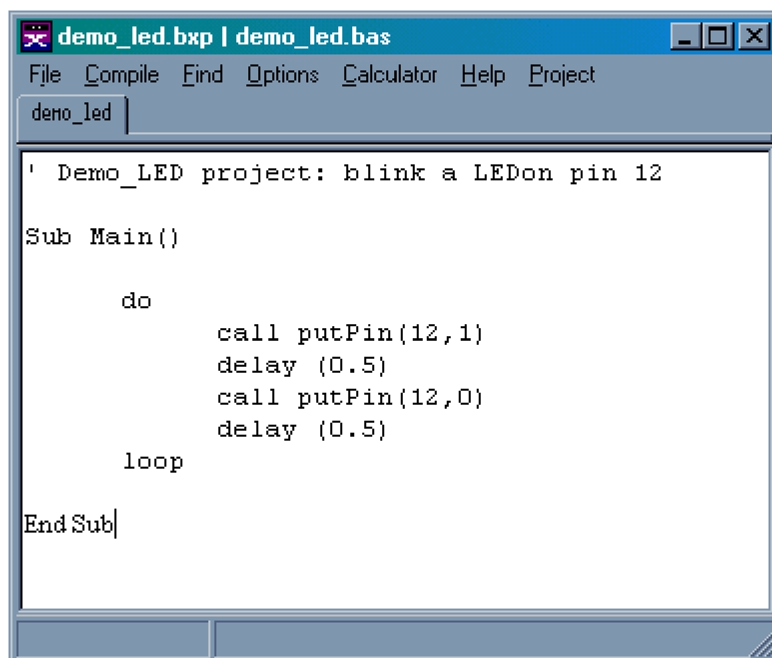
The download port is the serial port on your PC that you use to send the programs to the BX24. I am currently using COM2 – you can check to see what port the cable is attached to.

Setting the monitor port

The monitor port is the port you use to read debug information from the BX24. **If you do not set this you will not be able to see the debugging information**. (See "debugging" in the programming section).

Set the monitor port to the same port as the download port. (So again, this is COM2 for me).

The Editor window



The Editor window

There isn't that much to say. This is where you write the programs.

Under the "compile" menu you find the "compile" and "**compile and run**" commands.

These will try to compile the program and will tell you if there are any errors in it. If it can compile the program it will also try to download and run it.

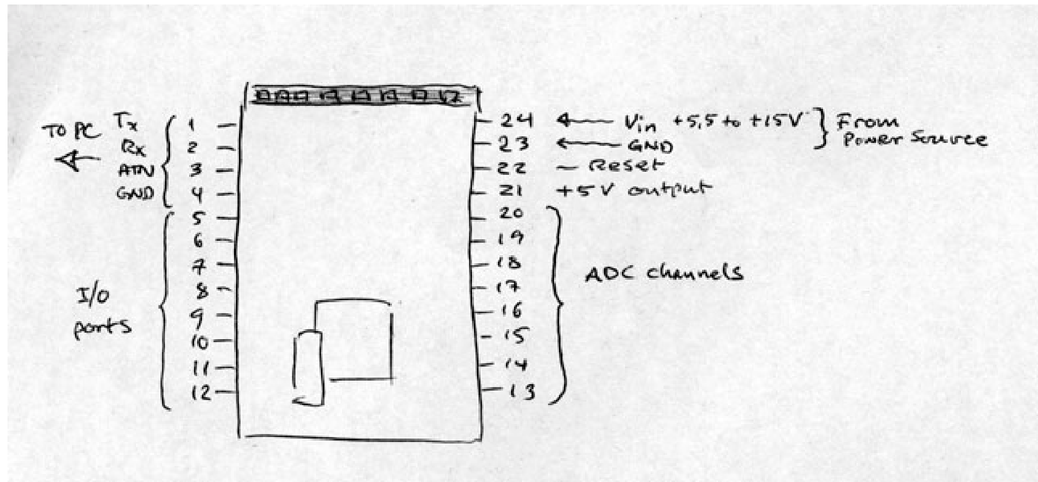
Before you can compile a program you must for some reason **set the chip preferences.** You just have to choose the "Chip" option in the "project" menu and then close the window that opens. If you like you can also select the color of the LED on the chip.

The BX24 chip

The BX24 is what we call a microcontroller. It is in fact a small computer, but without the things you usually expect in a computer – the screen, keyboard and mouse, for example.

It contains memory and an operating system. This means you can download programs to the BX24, and they can run directly on the chip.

BX24 pin configuration



BX24 pin configuration

The BX24 has 24 connectors, or ”pins”, twelve on each side.

Connecting the BX24

- Connect the topmost four pins on the right side (pin 1-4) to the serial port on your desktop PC. (We have prepared a cable for you – the yellow cable connects to the topmost pin).
- Connect the topmost pin on the left side (pin 24) to the plus side of the power source (The red cable). Set the power source to 7.5 V
- Connect the pin below (pin 23) to the ground cable (the black one) from the power source.

The remaining pins you will use to connect switches, sensors, servo motors, LEDs and other stuff.

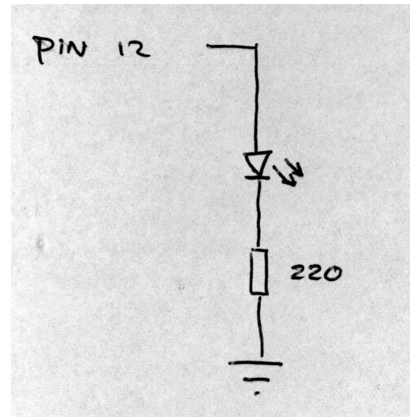
Example projects

These are some examples of very simple projects. None of them are particularly interesting really, but you can combine them to build more spectacular things.

Demo_LED

This example is really simple. It just makes a LED flash.

- Connect a LED and a 220k resistor in series from pin 12 to ground.



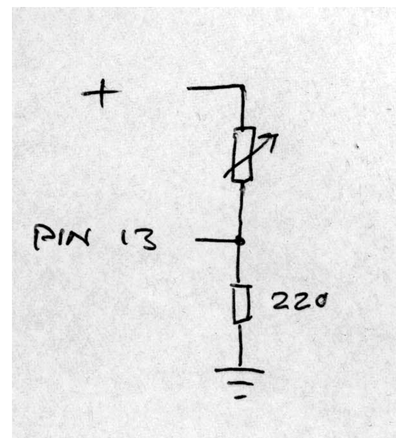
Demo_LED

```
' Demo_LED project: flash a LEDon pin 12

Sub Main()

      do
            call putPin(12,1)
            delay (0.5)
            call putPin(12,0)
            delay (0.5)
      loop

End Sub
```

Demo_pot

This example shows how to read a potentiometer value

- Connect a potentiometer from the "plus" to pin 13
- Connect a 220 ohm resistor from pin 13 to ground



Demo_pot

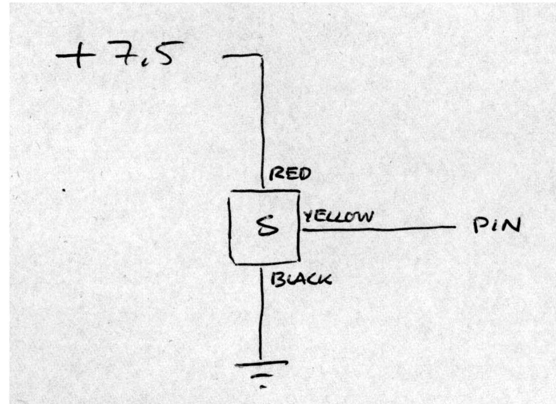
```
' Read the value of a pot on pin 13

dim potValue as integer

Sub Main()

      do

            potValue = getADC(13)
            debug.print "The pot value is " & cStr(potValue)

      loop

End Sub
```

You might get better values if you add a 220 ohm resistor in series with the potentiometer from power to pin 13.

Demo_servo

This example shows how to connect a servo motor. The program will make the servo swing from one end to the other and then back again.

- Connect the red cable from the servo to power (+7.5 V)
- Connect the black cable from the servo to ground
- Connect the yellow cable from the servo to pin 14



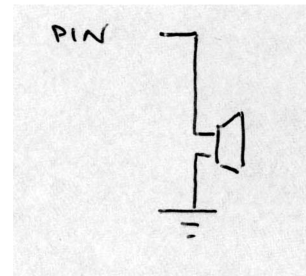
Demo_servo

```
dim position as integer
dim pulseWidth as single
const refreshrate as single = 0.02

Sub Main()
      position = 1

      do
            position = (position + 1) mod 100
            pulseWidth = 1.0 + cSng(position) / 100.0
            call pulseOut(14, (pulsewidth / 1000.0), 1)
            call delay (refreshrate)
      loop

End Sub
```

Demo_speaker

This example shows how to get sound from a speaker (It will not be a very loud sound using the speakers we have. You would need some sort of amplifier to get a louder sound.)

- Connect a speaker from pin 14 to ground



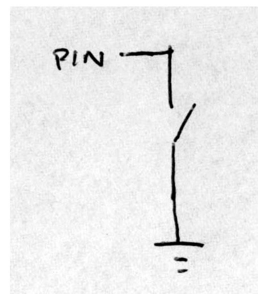
Demo_speaker

```
dim freq as integer
const duration as integer = 200
Sub Main()
      freq = 0
      do
            freq = (freq + 100) mod 10000
            call freqOut(14, freq, freq\2, duration)
            debug.print "Now playing " & cStr(freq) & " Hz"
      loop

End Sub
```

Adding a 10 uF capacitors in parallell (from pin 14 directly to ground) will produce a smoother sound. Try attaching a pair of earphones or a pair of PC speakers.

Demo_switch

There is a very simple way of connecting on/off switches and buttons to the BX24.



Demo_switch

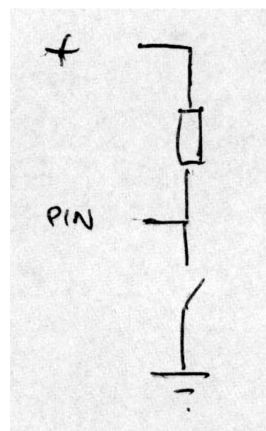
- Just connect a button from a pin to ground.

```
dim state as byte
Sub Main()
      call putPin(12, bxInputPullup)

      do
            state = getPin(12)
            debug.print "State " & cStr(state)
      loop
End Sub
```

The line call putPin(12, bxInputPullup) is important. This tells the BX to set the pin to "high" if the pin isn't grounded. This means getPin(***pin***) will return 0 when the pin is grounded, and 1 when the pin is not grounded.If you do not do this, getPin(***pin***) will return random 1's and 0's when the pin isn't grounded.

on/off switch with external pullup

If you want to understand how getPin() works it might be easier if you see this alternative solution with an "external pullup"



Switch with pullup

Hardware

Connect a 10 kOhm resistor from +5 to a pin. Also connect switch from the pin to ground.

Software

Now you can just use
`state = getPin(pin)`
as in the example above, without having to call
`putPin(pin, bxInputPullup).`

Comments

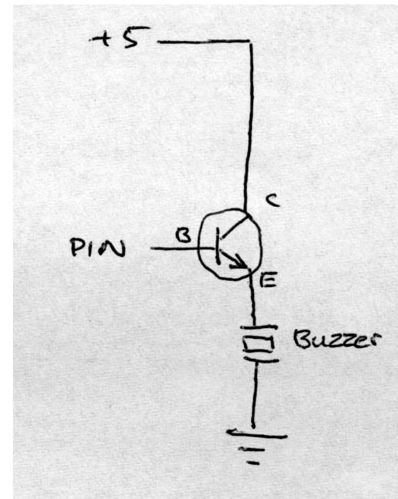
When the switch is open, it is as if it didn't exist at all.
There is no connection to ground (so no current flows), but there is a connection to +5 through the resistor. That means the potential ("the voltage") at the pin will be approximately +5V as well. (There is almost no voltage across the resistor, because there is almost no current flowing in to the BX24). A call to getPin(pin) will return 1, because the voltage is more than 0.

When the switch is closed, the pin will be grounded, and the potential at the pin will be zero. There will be a current flowing through the resistor from +5V to ground, but all the voltage will end up over the resistor. (There is no voltage from the pin to ground even though a current is flowing there, because there is almost no resistance).

Demo_buzzer

In theory, the buzzers are really easy to use: just connect it to a voltage source, and it will make a noise.
However, some of the buzzers we have draw too much power, so you will need to use a transistor to drive it.

- connect the collector leg of a transistor to power
- connect pin 12 to the base leg of the transistor
- connect the emitter leg of the transistor to the buzzer
- connect the buzzer to ground



Demo_buzzer

That buzzing sound will drive you crazy, so let's add an on/off button as well:

- connect a button between pin 11 and ground

```
dim button as byte
Sub Main()

      call putPin(11, bxInputPullup)

      do
            button = getPin(11)           ' read the button
            if (button = 1) then          ' if button up
                  call putPin(12,0)       ' silent
                  debug.print "silent"
            else                          ' else (if button down)
                  call putPin(12,1)       ' buzz
                  debug.print "buzz"
            end if
      loop
End Sub
```

Of course, if all you want to do is to make a noise when you close a switch, just connect the switch and buzzer in series from plus to ground. You don't need a microcontroller at all!

Learning more

BasicX by NetMedia
www.basicx.com
Here you can find examples and download the software development kit for the BX24. (For Windows, not for the Macintosh)

Dan O'Sullivan
http://fargo.itp.tsoa.nyu.edu/~dano/

Proceed to "Physical Computing". There's a good "Hands on guide for artists" there. The programming is for the Basic Stamp, not for the BX24. (They are similar, but not exactly the same)

Tom Igoe
http://fargo.itp.tsoa.nyu.edu/~tigoe/

Where to buy stuff
elfa, www.elfa.se
clas ohlson, www.clasohlson.se
High Tech Horizons (HTH), www.hth.com