Laboratory Exercise 11 – COMPUTING AND COMPUTER CONTROL

Part A: Basic Visual Basic

The Visual Basic environment

Basic is an example of a high-level computer language. This means that it uses words and mathematical symbols rather than machine code, which is a list of binary numbers suitable for direct action by the computer's microprocessor chip. Like other high-level languages, Basic is written as a series of commands that are either interpreted into machine code as execution proceeds, or compiled (i.e. pre-interpreted) beforehand. Basic has a very long history, and there are numerous versions and 'dialects'. Microsoft Visual Basic is relatively recent, and also incorporates pre-prepared GUI (Graphical User Interface, pronounced 'gooey') elements such as menus, windows, scroll bars and buttons. This greatly improves the user-friendliness of the programs, and makes the problems of input and output much simpler for the programmer. In addition, Visual Basic uses the modern paradigms of being 'object oriented' (the GUI elements are examples of 'objects'), and of being 'event driven' (the program waits for external stimuli, such as mouse clicks, to initiate certain activities).

The Visual Basic application provides a modern 'integrated development environment' (IDE) for preparing, editing, compiling, debugging, and running programs before they become standalone applications. It can be started using a shortcut that we have put onto the desktop (see below), or via the Windows 'Start' pop-up menu, and has controls similar to many other Windows applications. For this part of the exercise, we have written a simple Visual Basic project called **VBtrainer** for you to use. This has a place to put in various snippets of Basic code in order for you to become familiar with the language. You will work from your own copy of VBtrainer so that you can alter it and try things out independent of other students.

• Ask the lab technicians to give you an **account** on one of the PCs used for this project. Unless there are problems, you must always use the same PC and account. When you have logged on to your PC, click on the start button (bottom left of screen) then select My Computer from the menu. Under **Network Drives** you will see two drives. One is **Software3 on Teaching Samba Server**. Double click on this drive to view the folders stored on it. You will not be able to save any files onto this drive, or make any changes to the files on there, but you can make copies of them. This drive contains the folders **Trainer** and **Heater**.

• VB trainer is found inside the folder **Trainer**. Copy the Trainer folder onto the other network drive called **xyz on Teaching Samba Server** where **xyz** is your user name. This is the drive into which you must save all your work. If you do not know how to do this then ask a demonstrator. You will be able to use, alter and save any changes you make to this copy. Later on you can save differing versions (some perhaps with altered controls) under different names. At the end of each session on the PC, you must be sure to back up our copy of the project you are working on (**VB trainer** or later **Heater**) to a memory stick. Note that the standard filename extension for Visual Basic projects is **.vpb**, so double-clicking on any file with that extension will bring up the Visual Basic immediately.

• At any stage you may use the blue-grey Play button (right-facing triangle) on the toolbar to test-run the program. As on a tape or CD player, there are also buttons for Pause (||) and Stop (square). You can also start by using the Go! button of the program, and stop by using the Quit button. (Equivalent controls are also available from the menu bar, where they are called Start, Break and End.) Stopping brings you back to the editing mode. Try out these controls now.

You can learn about other facilities, especially the easy-to-use debugging system and how to add more graphical controls to the program, from the online help system or by asking the demonstrators as the exercise progresses.

As a useful tip for when you come to write your report, remember that in Windows you may, at any time, do a CTRL-‘Print Screen’ combination keystroke. This puts an image of the screen onto the clipboard, from which it may be pasted into e.g. a Word document and later trimmed or edited. Additionally, you may wish to copy (CTRL-C) the text of your programs so that you can then paste (CTRL-V) it into any other file. In this way, you can show the reader of your report what would be seen on-screen at any time, or make a copy of your program code so that it can be included in your report.

Before starting any new task or sub-experiment, copy all VBtrainer (or later Heater) files into a suitably named folder in the folder with your username, and be sure to work from that.

Basic programs

We shall now try some simple Basic instructions. The complete list, and assistance on any topic, may be reached via the Help menu, but the object of this part of the exercise is to learn and familiarise yourself first by some simple examples.

• View the VBtrainer code and look through it to find the part with the heading:

```
    Private Sub cmdGo_click()
```

The program comes to this subroutine whenever the Go! command button is clicked. Under the heading, enter the following line *exactly* as shown:

```
    picOutput.Print "Hello World!"
```

Then execute the program by pressing the Go! button. In the line you typed, `picOutput` is an ‘object’ which happens to be the big picture window on the screen, and it has an associated ‘method’ or action called `.Print` which puts text into the window. Later we will see how to get output printed on paper by referring to `Printer.Print`, in which a physical object called `Printer` has a similar method.

• Press the Quit button. Now edit the program so the command reads:

```
    picOutput.Print 2+3
```

Run the program again. Note the difference between `"2+3"`, which merely prints whatever is inside the quotation marks, and `2+3`, which evaluates the expression and thus gives us a kind of calculator. To make sure you understand this point, try changing the line to:

```
    picOutput.Print "2+3=",2+3
```

• Try other combinations and operations, including `*` and `/` for multiplication and division, and `^` to raise to a power; `^0.5` gives the square root. Basic also contains most mathematical and trigonometric functions, so try `Log(10)`, `Cos(3.1)`, `Exp(1.0)` and similar examples. Note that `Log` is actually the natural log (base *e*), and that angles are expressed in radians not degrees. A nice feature is that if you make an error in the code it will be highlighted.

Now we can generalise the process, by introducing variables `a`, `b`, and `c`. As soon as we use a variable the program allocates it a memory location to store its current value in.

• Put the following sequence into the program and run it:

```
    a = 2
    b = 3
    c = a^b + 2*(b-a)
    picOutput.Print "The value of c is ",c
    c = c+1
    picOutput.Print "The value of c is ",c
```

This shows that more complicated calculations can still be performed quite simply. But this example also illustrates a very important point about most high-level computing languages. In normal algebra `a^b + 2*(b-a) = c` is perfectly valid and means exactly the same thing as `c = a^b + 2*(b-a)`, but in Basic the first of these is *wrong*. The reason is that the = sign has a very different meaning than in algebra — it is an *instruction* to *take the value* of whatever is on the *right-hand side* and *transfer* it into the single *variable* that is named on the *left-hand side*. So in this case the value of `a^b + 2*(b-a)` is calculated, and then `c` is given that value. A more drastic illustration is the line `c = c+1`, which is nonsense in algebra — in Basic it says to add one to the value of `c` and then set `c` equal to that new value. You will get used to this different way of doing things. You might also like to think about the order in which operations are carried out. For instance, in the example above `a` is raised to the power `b`, and *not* the power `b + 2*(b–a)`.

Next we learn how to input information to the program. In the program above, changing the values of `a` and `b` requires you to change the program itself, which is not usually good practice. VBtrainer has been set up with two text boxes, so that when you run the program you can enter numbers in them before clicking Go!. Within the program these text boxes have the names `txtIP1` and `txtIP2`, so that you can write code that looks like `a = txtIP1.Text`. Note that `.Text` is now a 'property' of an object rather than a method, i.e. something it *has* rather than something it *does*. However, you should not necessarily count on Basic converting from the text to the numeric value automatically. So you should properly write `a = Val(txtIP1.Text)`.

• Alter the program so that the values of `a` and `b` are read from the text boxes rather than entered directly into the program code.

A very powerful technique in computing is to be able to do different things, depending on intermediate results. We do this using an `IF` statement. Hopefully you can guess what is going on in the following (where `...` means that some additional program statements would normally be inserted):

```
IF a>b THEN
...
ElseIF a<b THEN
...
Else
...
END IF
```

• Try to write your own program using this sort of `IF` block, reading in `a` and `b` from the text boxes. If `a` is bigger than `b` then output the result of `a-b`, if `b` is bigger than `a` then output the result of `a/b`, and if `a` equals `b` then output the square root of `a` (or `b`). Be sure to *record* the program code so that you can display it in your report, in order to allow it to be marked easily.

By now you will probably have made some mistakes, and seen that Visual Basic tries to correct you if you type anything that does not make sense. (If you have not yet made any mistakes, try typing something wrong on purpose just to see what happens!)

Visual Basic is set up to *ignore* any line whose first character is an apostrophe ('). In that case the whole line is typed in green as a **comment**. You should *use comments frequently* from now on to let others know, in English, what is going on in your programs. Equally important, comments also remind ***you*** (especially when you have not looked at the code for a while!) why you wrote what you did.

Another crucial programming concept is to do something repeatedly. This is achieved using a **loop**, as illustrated in the following example.

• Try the following in order to test an idea for generating prime numbers:

```
For i=1 to 10
c = 2^i - 1
```

```
    picOutput.Print "Is ",c," a prime number?"
    Next i
```

This goes round a set of instructions, starting with `i=1` and increasing `i` by 1 each time round until it has the value 10, when it will pass the `Next` instruction and carry on with what follows. Normally the variable changes by +1 each time the loop is performed, but we can use a `Step` instruction (e.g. `For i=1 to 10 Step 2`) to go forward or backward by a given number of integers each time.

• See if you can write a program to produce the factorial ($n!$) of a number n. These numbers get huge very quickly, so be careful.

In statistics the logarithm of a factorial turns out to be extremely useful, so much so that Stirling gave as an approximation $\ln(n!) = (n + 1/2)*\ln(n) - (n - 1)$.

• Test this out by calculating both $\ln(n!)$ and the approximation, and printing the ratio as n gets larger.

Again, *record* the code for your programs so that you can display it in your report.

We have only scratched the surface of Visual Basic. There are many more Basic statements and GUI elements. To discover them, get used to using the online help system.

Part B: Interfacing with the outside world

Our main aim in this part of the exercise is to see how to program the PC to communicate with apparatus in a simple way, and to do this we have to look at doing input and output with the world outside the computer's mouse, keyboard and screen. This is a very important aspect of the power of computers, but to understand what is going on we have to be familiar with the binary (base 2) numbers that are used.

Binary numbers

Computers communicate with other electronic devices using **binary** (i.e. base 2). For example, transistors are turned on or off, which corresponds to 'true' or 'false', or logic '1' or '0', using **b**inary dig**its** (**bits** for short). Silicon 'chips' are made up of a large number of semiconductor devices which manipulate very large numbers of these bits very quickly, for example:

- storing them, i.e. **memory**;
- adding, subtracting, or carrying out logical operations such as AND, OR;
- moving them about.

Binary digits are written in ascending powers of 2, starting at the right:

2^4 2^3 2^2 2^1 2^0
16 8 4 2 1

$$\begin{array}{rr} \text{For example, binary number } 10111 = & 1\times 16 \\ & 0\times \ \ 8 \\ & 1\times \ \ 4 \\ & 1\times \ \ 2 \\ & \underline{1\times \ \ 1} \\ \text{so in decimal it is} & \underline{23} \end{array}$$

In order to avoid writing out or displaying the long strings of bits that are a feature of binary, it is customary with computers to express the numbers in the much more compact hexadecimal notation ('hex'), which uses base 16 rather than the conventional decimal base 10 or the binary base 2. The conversion to and from binary is easy since $2^4 = 16$, so one hex digit represents four binary digits and so decimal numbers up to 15. (Too bad we don't have 16 fingers, though.)

The conversion is:

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

Two hex digits (i.e. 8 binary bits) can represent a number from 0 up to 255 (decimal; $2^8 - 1$) = FF (hexadecimal) = 1111 1111 (binary). Four hex digits (16 binary bits) can represent from 0 up to 65,535 (decimal; $2^{16} - 1$) = FFFF (hexadecimal) = 1111 1111 1111 1111 (binary).

- Example: 2B (hex) = $(2 \times 16) + 11 = 43$ (decimal). What is the binary form of this number?

An 8-bit computer is one that can handle and store 8 bits at once (8 bits is usually called a **byte**), that is, each memory location can store 8 bits (or 2 hex digits), and it can move these 8 bits on 8 parallel wires called the **data bus**. The data bus carries information within the computer and also to and from the outside world via connectors called **ports**. Later computers were 16-bit, storing and transporting two bytes at once, and nowadays they are 32-bit or 64-bit.

Input and output

It is common for computers and other electronic devices to talk to each other by transferring 8-bit bytes of information to an address or port that in practice is a socket on the outside of the computer. However, in the outside world most electrical signals are *analogue*, i.e. they can take any of a continuous range of values, while computers require *digital* information that can be represented by discrete binary integers and so is not continuous but quantised.

On the computers that are used for this exercise, we have installed electronic boards called DAQ (**D**ata **Acq**uisition) cards. These include analogue-to-digital converters (ADCs) to change an analogue signal from outside (e.g. a voltage from an external thermometer) into a byte of binary information for use in calculations within the computer. That allows *input* to the computer. For *output* from the computer we can either use digital information directly (e.g. one binary bit indicating 0 or 1 to decide whether a switch is turned off or on, possibly for a heater), or in addition the DAQ cards provide a digital-to-analogue converter (DAC) which will take a floating-point number in the range –10 to +10 from the computer and convert it to a corresponding voltage on an external wire. We will use all of these facilities.

Light and switch interface box

CAUTION: Before starting this section, make sure that the ***heater***, used later in part C, is ***NOT plugged in to the mains***. The reason will become obvious later!

Although we can use our outside connections to communicate immediately with various pieces of lab equipment, it is instructive first to use the interface boxes to become familiar with how things work. These have eight toggle switches and eight LEDs, allowing you to set the binary value of a byte, read it into the PC, and then display all of its bits. The software to do this is usually specific to a given model of DAQ card, and in VBtrainer we have hidden this complication from you in subroutines that you can call. (You are of course welcome to look at them and make copies!) To *receive* a byte of data into the PC the necessary code in your `cmdGo_click` subroutine is `Call DigIn(arg1)`, where `arg1` represents a variable containing a byte of input data read in from an external device, for example the eight switches. To *transmit* data out to an external device the code is `Call DigOut(arg2)`, where `arg2` is a variable set by your program, for example to display on the LEDs. Both `arg1` and `arg2` are positive 8-bit binary integers, i.e. their values are between 0 and 255. Note that `arg1` and `arg2` represent variables, and you can actually give them any names you like.

- Write a program that simply reads in the binary setting of the switches and displays them on the lights. Print the value on the screen as well (this is effectively a binary-to-decimal converter). *Record* the program code so that you can display it in your report, in order to allow it to be marked easily.

• Write a program to loop through all 256 possible values (0–255) of an 8-bit byte, displaying them on the lights. (Switches not needed here.) A simple loop will run too fast for you to be able see it incrementing, so you will have to slow the program down. There are various ways to do this. One is simply to insert a **delay loop** in the code to 'waste' time. For example:

```
For i=1 to 1000
Next i
```

Vary the number of passes through this loop to alter the delay. How many passes are needed to slow things down enough?

• Modify the program so that if a particular number which you specify via the statement `a = txtIP1.text` is selected on the switches, something special happens. For example, arrange the program to put up a special message when that number is input (in binary) on the switches.

Again you are reminded to *record* all code that you write for your report.

Digital-to-analogue converter

A digital-to-analogue converter (DAC) is an integrated circuit which outputs a voltage proportional to the binary number sent to it. VBtrainer includes a subroutine to do this — to generate a voltage of `arg3` volts at the interface box via the DAC all you have to do is insert in your code the statement:

```
Call AnaOut(arg3)
```

• Arrange your program so that you can type a voltage value in *volts* into an input text box and then output it using the DAC. Check that the DAC generates the correct value using a DMM connected to the green and red pair of terminals on the interface box. What range of voltages can you get?

• By changing the values of the voltage and reading the variations in the DMM, try to get some idea of the resolution of the DAC and *comment* on it.

• Next write another program that generates a pattern of output voltages one after another in a loop, and displays the output on an oscilloscope. Remember to ensure that your program has a way of ending! Start with a sine wave; this could include something that might look like:

```
pi = 3.14159
steps = 500
c = 2*pi/steps
For n = 0 to steps
a = 10*sin(n*c)
Call AnaOut(a)
Next n
```

• Use a similar technique to plot the following:

a) A squarewave.

b) A sawtooth, or a ramp and a triangle.

c) Another function of your choosing.

Part C: Temperature control by computer

The aim of this part of the experiment is to use information coming into the computer to tell it what actions to perform on signals that it puts out. Specifically, you will accurately control the temperature of a water-bath using computer software.

To allow you to do this, we have provided one further subroutine, `Call AnaIn(arg4)`. This gives the voltage coming from a thermocouple and thus measures temperature. In addition, `Call DigOut(arg2)` now takes on the specific job of digitally switching a heater on and off.

Equipment and program

- Familiarise yourself with the wiring of the equipment. As before, the DAQ card in the PC is connected to the interface box. There is a control wire from this box via a 5-pin plug to a small grey box, which also has a mains input, and an output to the heater itself. This box has a red safety light which is lit whenever mains voltage is on at the heating element. When your experiment gets under way you should see the LED flashing, with a frequency dependent on how much heat you want to deliver. The software you use is ideally ‘fail-safe’, but things can always go wrong so look at the LED from time to time in case the heater is accidentally left on. ***If there is a problem, simply unplug the grey box.*** In addition, be sure to switch off and unplug all equipment at the end of your activities for the day.

CAUTION: ***Ensure that the heating element is always immersed up to a level between the two indented marks, otherwise it will be permanently damaged***.

To measure the temperature, a chromel–alumel thermocouple is used which has a temperature sensitivity of approximately 40 μV/°C, and is therefore amplified before the ADC chip can be used. The thermocouple is plugged into the side of the interface box. The end result is an analogue input to the computer which is roughly 10 mV per °C. You will have to calibrate this more accurately later as a first step in the experiment.

• Most of the controlling program has already been written for you. It incorporates updating graphics to enable you to see what is happening continuously. ***Always*** work from ***your own copy*** of this Visual Basic project, called **Heater**, so ***copy it to the folder with your username***.

• Before inserting your own code, run the program as it is. The green Start button begins a data-taking run, and immediately turns into a red Stop button to allow you to terminate the run. Starting a new run will clear all information from the last run and set everything to default values. The Print button prints just the graphical output, and the Save button enables you to write out the data to a `.dat` file which may then be input into PhysPlot to allow more control over the graph before printing it. (Be careful with file naming in order to avoid overwriting any previous files that you still want to keep!)

It is important to understand the general mode of operation of this program. It spends 99.9% of its time within the subroutine `USER` (the only one that you are allowed to modify). Once the call is complete, the subroutine is called again and again. Every once in a while (`Dt` seconds, as set up on the controls), an interrupt is generated and a fast monitoring call is made. The counter `ncount` is updated, the variable `T1` is filled with the latest temperature, and the graph is updated with this value. Then control is returned to `USER`. Within this routine you should be able to discover that such an update has been made by checking whether `ncount` has increased, using your own internal storage variable.

If `Dt` is set low it is easier to control the temperature. Start with `Dt = 5` (seconds) when you are getting the feel of apparatus and the task. However, your final aim is to use `Dt = 30`, and show your skill in using more sophisticated control algorithms to overcome the lack of knowledge from less frequent readings. We do this because in the real world we might, for example, be using one PC to control thousands of temperatures in this way.

The aim of the experiment is to switch the heater on and off repeatedly from the controlling program so that the temperature of the water goes to and then remains as close as possible to a target temperature. The target temperature that you are actually aiming for is called `T0`, and it is set by another control within `USER`. There are also outputs, including a timer, the last input voltage (`ADC`), and the temperature (`T1`) resulting from calibration of the ADC. There are two calibration constants, `a` and `b`, for converting ADC voltage readings to temperature; linearity is assumed. You must enter these. Within the program there is a line `T1 = a*ADC + b`, where `ADC` is the voltage reading from the ADC, and you should find that `a` is roughly 100 and `b` is roughly 0.

Experiment

The aim of the first step is to find accurate values of `a` and `b`, so as to give accurate values of `T1` in °C agreeing with the alcohol-in-glass thermometer. Note that the thermocouple has a faster response time to temperature changes than the bulkier thermometer.

• Start by setting `a = 1` and `b = 0`. This means that `T1` is the actual thermocouple voltage (after amplification). By putting the thermocouple first in iced and then in very hot water, as well as two intermediate temperatures, and letting things settle down, you should be able to calibrate the ADC values against the corresponding thermometer readings. Make a graph of temperature vs. voltage using PhysPlot, and then fit a straight line — the gradient and intercept will give you `a` and `b`. Thereafter *they should be entered before each run. Be sure to record everything!*

You are now in a position to start altering the world! (Or at least the temperature of the water-bath.) As before, `Call DigOut(1)` turns on the first LED light on the interface box. However, when the grey heater control box is plugged into the interface box it *also* controls the heater. `Call DigOut(1)` turns the heater on, and `Call DigOut(0)` turns it off. Furthermore, the program always does a `Call DigOut(0)` when it closes down, to make it relatively fail-safe.

• Double-click on the heater project Heater.vbp to open the program coding. Insert your user code into the subroutine `Private Sub User()` near the end. Try `Call DigOut(1)` by putting it into a `FOR` loop which turns the heater on for half the time and off for the rest. You should see the red light on the interface box flashing in time with the safety light on the heater switch box, so this works as your own indicator.

For the rest of the experiment it is your task to set a target temperature (`T0`) to which, hopefully, `T1` converges. You should test your software thoroughly by using both a *low* and a *high* target value for each version, say 40 °C and 90 °C. You should *not*, yourself, obtain a new value of `T1` at any time but rely upon the updates every `Dt` seconds. In this way you are gradually forced to use more sophisticated algorithms on the limited information available, as with the problem in real control situations where the computer has to monitor very many devices.

• Write versions of `USER` based on the following algorithms, of increasing sophistication:

(a) With `Dt = 5` seconds, turn on the heater if the last value of `T1 < T0`, and off otherwise. Naturally there will be overshoots and hence oscillations.

(b) If `T1 < T0` turn on the heater for a fraction of time which is proportional to `|T1–T0|`, i.e. the difference between `T1` and the required temperature. Start with `Dt = 5` seconds, but when things are working well try increasing `Dt` to 30 seconds.

You are eligible to get extra marks by inventing other ways of quickly converging on `T0`. For example, use the difference in the last *two* recorded temperatures (i.e. the linear trend), as well as `|T1–T0|`, to decide how long to turn on the heater. Remember to state clearly, for all cases, what `T0` was when you write up your results. Your **report** must include a **diagram of the set-up**, **graphs**, and **printed listings of all programs** discussed.