E BLOCKS®

LEARN•DESIGN•BUILD

USB Systems

EB540-80-05

MATRIX

www.matrixtsl.com

EB540

USB Solution

Instructor Guide

Contents

About this course

Aims: The aim is to introduce the concepts involved in USB devices.

On completing this course students will have learned about:

- the relationship between USB masters, hubs and endpoints;
- the electrical principles behind USB architecture;
- the components that make up a USB device;
- the options available for USB devices;
- the addressing schemes;
- USB signals and routing;
- low power and sleep modes;
- USB device drivers;
- USB devices that do not require drivers.

What you will need:

To complete this course, students will need the following equipment:

- Flowcode software
- E-blocks including:
    - 1 Multiprogrammer (PIC - EB006)
        - with PIC18F4455 device and 4MHz crystal;
    - 1 Sensor E-Block (EB003);
    - 1 LED E-Block (EB004);
    - 1 LCD E-Block (EB005);
    - 1 Keypad E-Block (EB014);
    - 1 USB E-Block (EB055);
    - 1 RS232 E-Block (EB015);

Using this course:

This course presents students with a number of tasks listed in the exercises that follow the USB overview. All the information needed to complete these is contained in the notes.

Before starting the exercises, students should familiarise themselves with the background material.

Time:
To undertake all of the exercises will take around twelve hours.

Important note:
Information presented here is correct at the time of publication.
Please check the Matrix web site, www.matrixltd.com for the latest E-Blocks documentation.

Scheme of Work

Section	Notes for instructors	Timing (minutes)
1. Introduction to USB		
1.1 Preamble	Students familiarise themselves with the course ahead. They can use a web browser to review the differences between various versions of USB, from USB1.0 to USB 3.0.	5 -10
1.2 Key Advantages of USB	This section lists the main advantages, and drawbacks, of the USB protocol. Again, they can use the internet to familiarise themselves with other communication protocols such as Firewire and RS232.	5 - 15
1.3 Introduction to USB	This section compares the performance of USB 2.0 devices with Firewire and Serial communications links. It then gives a glossary of terms used in USB systems, including: master / slave configuration; USB power; connectors; functions; endpoints; pipes; classes device drivers addressing enumeration interface speeds noise immunity. USB is a network of attachments connected to a host computer. The attachments are either functions or hubs, and together they are known as devices. The host has a hub embedded in it called the root hub, the interface between the computer and the USB ports it houses. External devices which offer more than one function are usually combined with a hub, and are then called compound devices. Hubs may be connected to other hubs in a tiered arrangement, but logically the system appears as a linear bus. These terms may mean little until the student has had hands-on experience of the USB protocol later in the course. However, it pays to spend time on these terms now. At least students will know where to find some explanation of these terms if problems arise. No attempt has been made to tackle physical layer issues, such as the use of NRZI for signalling, and 'bit stuffing'. The terms 'J state' 'K state' and 'Single-ended zero (SE0)' are not used. Instructors wishing to expand on these issues will find the solution a fitting tool to facilitate this.	20 - 30

2. Transfer Types		
2.1 Transfer Types	The USB protocol is used across an increasingly wide variety of applications. Different situations demand different kinds of data transfer. This section outlines the differences between control, interrupt, bulk and isochronous transfers. Depending on the situation, the designer can choose to prioritise data validity, latency (delay) or bandwidth by choosing the appropriate transfer type. Once again, the students can use the internet to reinforce the ideas introduced here, or the instructor may choose to spend time supporting them. Students should be warned that, despite the name, interrupt transfers do not cause interrupts. The text explains that this transfer type is used where previously devices would use interrupts to initiate communication.	15 - 30
2.2 Transfers and Transactions	It is important that students understand and use the correct terminology in USB systems. This section distinguishes between data transfers and transactions. A data transfer may be split across several individual transactions. These transactions may occur across a number of frames. The reality is that the host will send out frames every millisecond. These frames will contain a number of transactions, sandwiched into time slots within the frame. Several transactions within a frame may be addressed to the same device. Each transaction is made up from a number of packets, known as token, data and handshake packets. Some sources refer to these as phases. Each has its own function, and as a result, contains different sets of fields. Students should study the diagrams carefully, so that they understand them. It may be profitable for them to make copies of them for their records.	15 - 30
2.3 Transactions	This section examines the four transaction types – start of frame, token, data and handshake. The purpose of each is described briefly, and their structure is outlined diagrammatically. As its name suggests, the Start-of-frame packet is found at the beginning of every frame. In other words, the host produces one of these every millisecond. One job of the Start-of-frame packet is to track frame number. One of the fields, the frame number, identifies each frame of the transaction. Devices can use this to confirm which transaction has been received, or can use it as a timing source. As it is eleven bits long, it can cope with 2^{11} (= 2048) frames. When the maximum is reached, the frame count resets.	20 - 30

2.3	Transactions continued...	The frame can contain eight microframes, when working at high speed. All eight carry the same frame number. The text goes on to outline the three types of data packet – Setup, IN and OUT. The communication 'pipes' that are set up between the host and peripheral devices are uni-directional. One task of a data packet is to define the direction of data flow. This is always taken from the viewpoint of the host. IN means flowing in to the host and so out from the peripheral. OUT means flowing out of the host, and so in to the peripheral. One implication of this, explored later, is that 'Set' requests, where the host imposes a configuration value on the peripheral device, have direction OUT, whereas 'Get' transactions, where the host requests settings from the peripheral device, have direction IN. The Setup process has its own section later. The idea of having two varieties of data packet, called Data0 and Data1, offers another form of error checking, where data is transmitted using multiple transactions. The first of these will use a Data0 packet, the second a Data 1, then a Data0, and so on. The data toggle value is specified in the PID. Both transmitting and receiving devices can monitor the data type to check for missing transactions. The receiver of data will reply with a handshake packet to indicate the status of the transfer. Hence, for an OUT transfer, the peripheral device replies by sending the handshake packet, whereas for IN communications, the host replies. Peripheral devices can reply with ACK (valid data was received,) NAK (the device is busy and did not receive the data,) or STALL (the device does not understand the transfer, or is not active.) A host can only send ACKs. If the receiver detects an error, it returns no handshake packet.	
2.4	USB packets	USB transmissions are synchronous, thanks to NRZI encoding and bit stuffing, which allow the receiver to synchronise its clock with that of the transmitter. In addition, each packet starts with a synchronising field, a series of alternating bits, to ensure that the clocks in the host and peripheral device are in synchronisation. A PID field follows. The term 'PID' has two possible meanings in the USB world. Here, it means Packet IDentifier. When referring to whole devices, it can mean Product Identifier, a 16-bit number used to identify the appropriate device driver. The Packet ID is used to identify the type of packet being sent. e.g. token, data, handshake etc. The table shows how it does this. The student text says that the four most-significant bits are the inverse of the four least significant bits. To be precise, they are the 1's complement of the four least-significant bits. It is left to the instructor to decide whether to expand on this with the students.	20 - 30

2.4 USB packets continued...	The address field contains a seven bit address, allowing 2^7 (=128) addresses. Only 127 of these are assigned as device addresses. Address 0 is reserved for the mandatory default endpoint on all devices, so that the host can send control transfers. The endpoint field identifies the endpoint (function) to which the packet is directed. Each endpoint has a number from 1 to 15, expressed as a four-bit binary number. The CRC (cyclic redundancy check) is included to check the data for errors. A mathematical operation is applied to the data at the transmitting device, and the result of that operation is sent as part of the transfer. The same mathematical operation is applied to the data at the receiver. The result is compared to that sent in the CRC. If they are the same, there is no error. If the results are different, then an error is present. The end-of-packet field indicates that the transaction is complete. The bus then goes back to its idle state.	

3. Setup		
3.1 3.1 The Setup stage	The purpose of the Setup stage is explained, as the process by which the host learns about the recently attached device, and then each of the three phases, token, data and handshake are described. The core of the transaction is the request for information. The data phase contains five fields, which occupy eight bytes. The first, bmRequestType, specifies the type of request, its direction and the recipient's address. The next, bRequest, specifies the actual request. Its contents depend on the type of request (standard, class, or vendor) identified in the bmRequestType field. The next field, wValue, occupies two bytes and contains information for the recipient from the host. The significance of the information depends on the type of request. For example, the Set_Address request will send the new address in the wValue field. Next comes another two byte field, called wIndex. Again it is used by the host to pass information to the device. This information again depends on the request. It may include an endpoint address, and interface number etc. Finally, comes another two byte field, wLength, The host will specify how many data bytes are sent. The manual then includes a series of tables describing the structure of a number of different requests. The purpose is to illustrate what goes into these requests, and how they are transmitted. It is reference material to aid later study. It is not intended that the students should in any way 'learn' these tables. The handshake phase takes place if the device receives the full transaction without detecting any errors.	20 - 30

4. Learning about USB Device Capabilities		
4.1 Descriptors 4.2 USB Device Descriptors 4.3 USB Configuration Descriptors 4.4 USB Interface Descriptors 4.5 USB Endpoint Descriptors 4.6 USB String Descriptors	During enumeration, the host learns about the capabilities offered by the device, using control transfers which request a series of descriptors. These start with the broad brush strokes of the Device Descriptor, which cover the global properties of the device. It also specifies all of the subordinate descriptors needed by the host. This is followed by one of the Configuration Descriptors, which include its power requirements, the Interface descriptors, providing information about a feature of the device, including class and protocol information, the endpoint descriptors, which specifies the maximum packet size the endpoint is capable of handling, and finally any optional descriptors. Again, the purpose is illustrative and to act as a reference. It is not intended that the students should in any way 'learn' these tables.	20 - 30

5. The Matrix USB Training Solution		
5.1 Solution Overview	This section starts with a description of the hardware provided in the Matrix kit. It includes a diagram showing the layout and connections for the USB solution.	5
5.2 Default connections and settings	Students should read this while keeping an eye on the hardware itself. They should familiarise themselves with the layout of the Multiprogrammer board, and in particular, identify the position of its various ports. They should identify and check carefully the jumper settings described here. Detailed information on all the E-Block hardware is available from the Matrix website. Students should be encouraged to read and use this information.	5 - 10
5.3 Flowcode and USB 5.3.1 USB Serial Component 5.3.2 USB HID Component 5.3.3 USB Slave Component 5.3.4 Enumeration Wait Setting	To facilitate USB connections to the host computer, Flowcode comes complete with three USB components, a serial component, a HID component and a slave component. Their properties are described. Each requires its own device driver and descriptors, and comes with its own set of component macros. Students should run Flowcode in order to examine these components, their properties and associated macros. They will need additional applications on the computer, specifically Hyperterminal (or equivalent) and Labview. They may need support in setting up and using these, in addition to the instructions that accompany the exercises.	15 - 25
5.4 USB Serial Device 5.4.1 Installing the Device Driver 5.4.2 USB Serial Device and Hyperterminal	Like all peripheral devices, a device driver is needed so that the computer can communicate with it. This driver should be generated from within Flowcode, as described in the instructions, so that the configuration matches that of the Serial Component itself. This section may be done by students only if the network grants them sufficient privileges. Otherwise, the instructor or IT Support staff may have to do it. In either case, the students should read through the notes and be aware of the processes. The Serial Component relies on the RS232 protocol for communication between the host computer and the peripheral device. The word ‘serial’ indicates that data is sent one bit at a time down the link. The RS232 protocol encodes the data and defines the electrical and timing characteristics of the signals. Though dating back to the early 1960’s, when the protocol was used to communicate between mainframe computers and dumb terminals, it is still commonly used in serial links, though it is slowly being replaced by USB connections. Hyperterminal displays messages passed to and from the host computer, and is used for that purpose here.	20 - 40

5.5 USB Slave Component 5.5.1 Installing the Device Driver 5.5.2 USB Slave and Visual Basic	The next Flowcode component is the USB Slave. This simply responds to communications from the host. The host can initiate processes such as sampling an analogue input on the microcontroller. As with the Serial Device, a driver must be generated from within Flowcode to allow the device to communicate with the host. As before, this is done after the component properties are determined in Flowcode to ensure that they match. As before, some networks may not grant sufficient privileges to students to allow them to install drivers. In that case, the task must be carried out either by the Instructor, or by IT Support staff. Nevertheless, the student should read through this section so that they are aware of the issues raised. This time Visual Basic will be used to configure the microcontroller, using the '.dll' provided on the accompanying CD-ROM.	15 - 30
5.6 USB HID Custom Descriptor Generation	Whereas the USB Serial and Slave components come with serviceable device drivers already created, the USB HID devices use a descriptor, chosen from the solution CD, or generated using software downloaded from the USB developers' website – www.usb.org/developers/hidpage/	5
5.7 PIC 18F4455 Configuration	This section includes a table showing the configuration needed by the PIC 18F4455, used on the E-Blocks Multiprogrammer board. The chip can be configured in Flowcode by using the 'Configure' tab in the 'Project Options' window which is accessed via the 'Build' menu.	5

Section	**Notes for instructors**	**Timing (minutes)**
6. The USB assignments		
	This section lists the assignments and gives a brief outline of what each covers.	5
6.1 Exercise 1 – Human Interface Device: Mouse		
6.1.1 Introduction 6.1.2 Objective 6.1.3 Target Microcontroller 6.1.4 Flowcode USB HID component 6.1.5 USB HID component settings 6.1.6 The Flowcode program in detail 6.1.7 A Generic USB Mouse 6.1.8 What to do 6.1.9 Further work	This is the first of a series of practical assignments using Flowcode to control USB devices. The aim is to set up a USB mouse, using the E-Blocks Keypad as a data source. The diagram shows which keys control which mouse functions The students are also given information about the 'Initialise HID' macro, and the settings needed to configure the USB HID device. Detailed instructions on how to build the Flowcode program are given. It is assumed that students already know how to: • load new components into a Flowcode program; • configure component properties; • insert a macro into the program, and configure it; • add a new variable to a program; • add and configure a time delay. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
6.2 Exercise 2 – Human Interface Device: Keyboard		
6.2.1 Introduction 6.2.2 Objective 6.2.3 Flowcode USB HID component 6.2.4 USB HID component settings 6.2.5 The Flowcode program in detail 6.2.6 A Generic USB Keyboard 6.2.7 What to do 6.2.8 Further work	The aim is to create a generic USB keyboard. The E-Blocks keypad will control the keyboard, and provide its input. The E-Blocks LED board will indicate when Caps Lock, Num Lock or Scroll lock are activated. The outline describes the operation of the CheckRx and the ReceiveByte component macros. It describes how the peripheral device 'knows' when the Caps Lock, Num Lock and Scroll lock keys are pressed. It details the settings needed to operate the USB HID component in this program, and detailed notes on how to construct the program. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30

6.3 Exercise 3 - Human Interface Device: Data-Logger		
6.3.1 Introduction	The aim is to use the keyboard created in the previous program in a data logging exercise. The data comes from sampling the output of the light-sensing unit built into the E-Blocks Sensors board. Data logging is controlled by the Num Lock key. The data is transferred to the Microcoft Excel spreadsheet application, which must be available on the computer. Detailed instructions on how to build the Flowcode program are provided in the ‘What to do’ section. A suitable Flowcode program is described in the ‘Solutions to Exercises’ section.	30
6.3.2 Objective		
6.3.3 USB HID component settings		
6.3.4 The Flowcode program in detail		
6.3.5 Storing the data		
6.3.6 What to do		
6.3.7 Further work		
6.4 Exercise 4 - Communications Device: USB Terminal		
6.4.1 Introduction	The aim is to exchange ASCII data between the USB solution board and the computer, via a serial connection. The data received from the computer will be displayed on the E-Blocks LCD device. The exercise instructions describe the ‘Initialise’ macro, and the role it plays. Students are shown how to configure the USB Serial component. The function of the ReadByte, SendByte, ReadString and SendString macros is described, and the significance of the return value. Detailed instructions on how to build the Flowcode program are given. A suitable Flowcode program is described in the ‘Solutions to Exercises’ section.	30
6.4.2 Objective		
6.4.3 Flowcode USB Serial component		
6.4.4 USB Serial component settings		
6.4.5 The Flowcode program in detail		
6.4.6 A Generic USB Serial Port		
6.4.7 What to do		
6.4.8 Further work		
6.5 Exercise 5 - Communications Device: USB to RS232 protocol bridge		
6.5.1 Introduction	The aim is to show how to set up a legacy RS232 COM port device. The RS232 E-Blocks board converts signals into the RS232 standard. A terminal emulation package such as HyperTerminal, running on the computer, is used to connect to the device. This is an example of a protocol bridge, where signals encoded in one protocol are converted into those of a different protocol. The instructions highlight the importance of data transmission rates in avoiding data bottlenecks. Detailed instructions on how to build the Flowcode program are given. A suitable Flowcode program is described in the ‘Solutions to Exercises’ section.	30
6.5.2 Objective		
6.5.3 USB Serial component settings		
6.5.4 The Flowcode program in detail		
6.5.5 USB to Serial Bridge		
6.5.6 What to do		
6.5.7 Further Work		

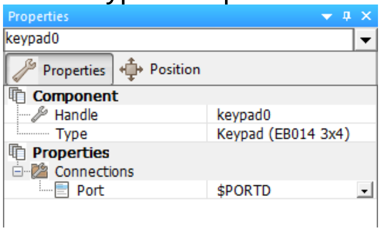
6.6 Exercise 6 - Slave Device: Basic Slave Functionality		
6.6.1 Introduction	The aim is to enable the computer to control the operation of the microcontroller on the E-Blocks Multiprogrammer board. In this way, the microcontroller can perform some of the control tasks, and relieve the computer CPU of that burden, an example of distributed processing. The microcontroller acts as a slave to the CPU. It will control the LCD display, and scan the keypad for input signals, passing relevant information to the CPU when directed to do so. The role of the macros associated with the Slave device are described, along with the configuration settings needed. The use of a custom dll and driver support files is described. In this exercise, Microsoft Visual Basic is used to configure the application. This must be available on the computer, and students should have a basic familiarity with this application. Detailed instructions on how to build the Flowcode program are given. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
6.6.2 Objective		
6.6.3 Flowcode USB Slave component		
6.6.4 USB Slave component settings		
6.6.5 The Flowcode program in detail		
6.6.6 A USB Slave Transaction		
6.6.7 Driver Support Files		
6.6.8 What to do		
6.6.9 Further Work		
6.7 Exercise 7 - Slave Device: Storage Scope		
6.7.1 Introduction	The aim is to sample an analogue signal using an Analogue-To-Digital converter, to store the results and to investigate the properties involved in these processes. The student is introduced to customising parameters within the program to optimise the transfer of data between the host and the peripheral devices. This is done by modifying the underlying C code, following comprehensive instructions. Detailed instructions on how to build the Flowcode program are given. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
6.7.2 Objective		
6.7.3 The Flowcode program in detail		
6.7.4 Component Customisation		
6.7.5 What to do		
6.7.6 Further Work		
6.8 Exercise 8 - Slave Device: Triggered Scope		
6.8.1 Introduction	The aim is to extend the functionality of the slave component created in the previous program, allowing it to control the sampling process. The sampling process now occurs on one of two ADCs, specified by the controller. As before, Visual Basic is used to create the application. Detailed instructions on how to build the Flowcode program are given. A suitable Flowcode program is described in the 'Solutions to Exercises' section.	30
6.8.2 Objective		
6.8.3 The Flowcode program in detail		
6.8.4 What to do		
6.8.5 Further Work		

7. The USB C Code Library
Students are given the location of the C code used to drive the USB components.

Solutions to Exercises

Exercise 1:

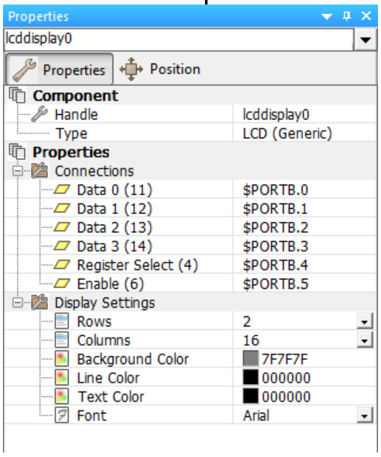
Keypad Properties



keypad0

Component	
Handle	keypad0
Type	Keypad (EB014 3x4)
Properties	
Connections	
Port	$PORTD

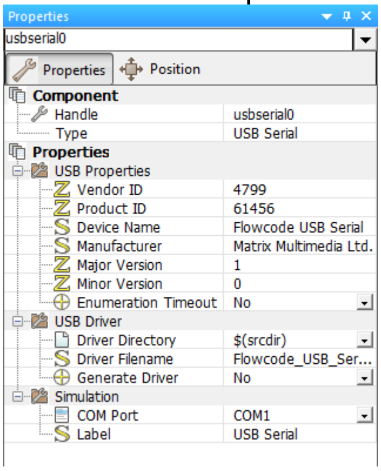
LCD Properties



lcddisplay0

Component	
Handle	lcddisplay0
Type	LCD (Generic)
Properties	
Connections	
Data 0 (11)	$PORTB.0
Data 1 (12)	$PORTB.1
Data 2 (13)	$PORTB.2
Data 3 (14)	$PORTB.3
Register Select (4)	$PORTB.4
Enable (6)	$PORTB.5
Display Settings	
Rows	2
Columns	16
Background Color	▨ 7F7F7F
Line Color	■ 000000
Text Color	■ 000000
Font	Arial

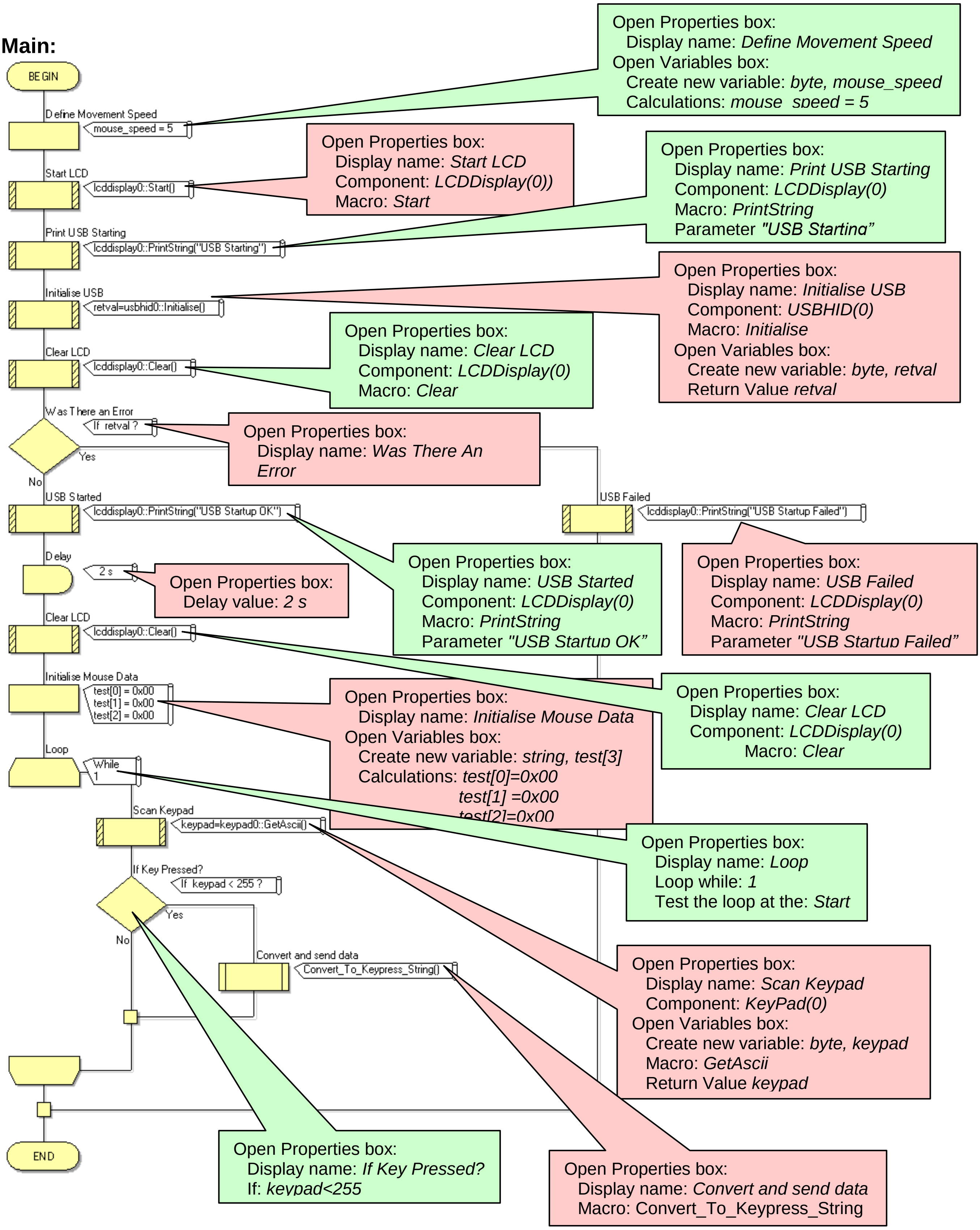
USB Serial Properties



usbserial0

Component	
Handle	usbserial0
Type	USB Serial
Properties	
USB Properties	
Vendor ID	4799
Product ID	61456
Device Name	Flowcode USB Serial
Manufacturer	Matrix Multimedia Ltd.
Major Version	1
Minor Version	0
Enumeration Timeout	No
USB Driver	
Driver Directory	$(srcdir)
Driver Filename	Flowcode_USB_Ser...
Generate Driver	No
Simulation	
COM Port	COM1
Label	USB Serial

Main:



Open Properties box:
Display name: *Define Movement Speed*
Open Variables box:
Create new variable: *byte, mouse_speed*
Calculations: *mouse_speed = 5*

Open Properties box:
Display name: *Start LCD*
Component: *LCDDisplay(0))*
Macro: *Start*

Open Properties box:
Display name: *Print USB Starting*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Starting”*

Open Properties box:
Display name: *Initialise USB*
Component: *USBHID(0)*
Macro: *Initialise*
Open Variables box:
Create new variable: *byte, retval*
Return Value *retval*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Was There An Error*

Open Properties box:
Display name: *USB Started*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Startup OK”*

Open Properties box:
Display name: *USB Failed*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Startup Failed”*

Open Properties box:
Delay value: *2 s*

Open Properties box:
Display name: *Initialise Mouse Data*
Open Variables box:
Create new variable: *string, test[3]*
Calculations: *test[0]=0x00*
test[1] =0x00
test[2]=0x00

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Loop*
Loop while: *1*
Test the loop at the: *Start*

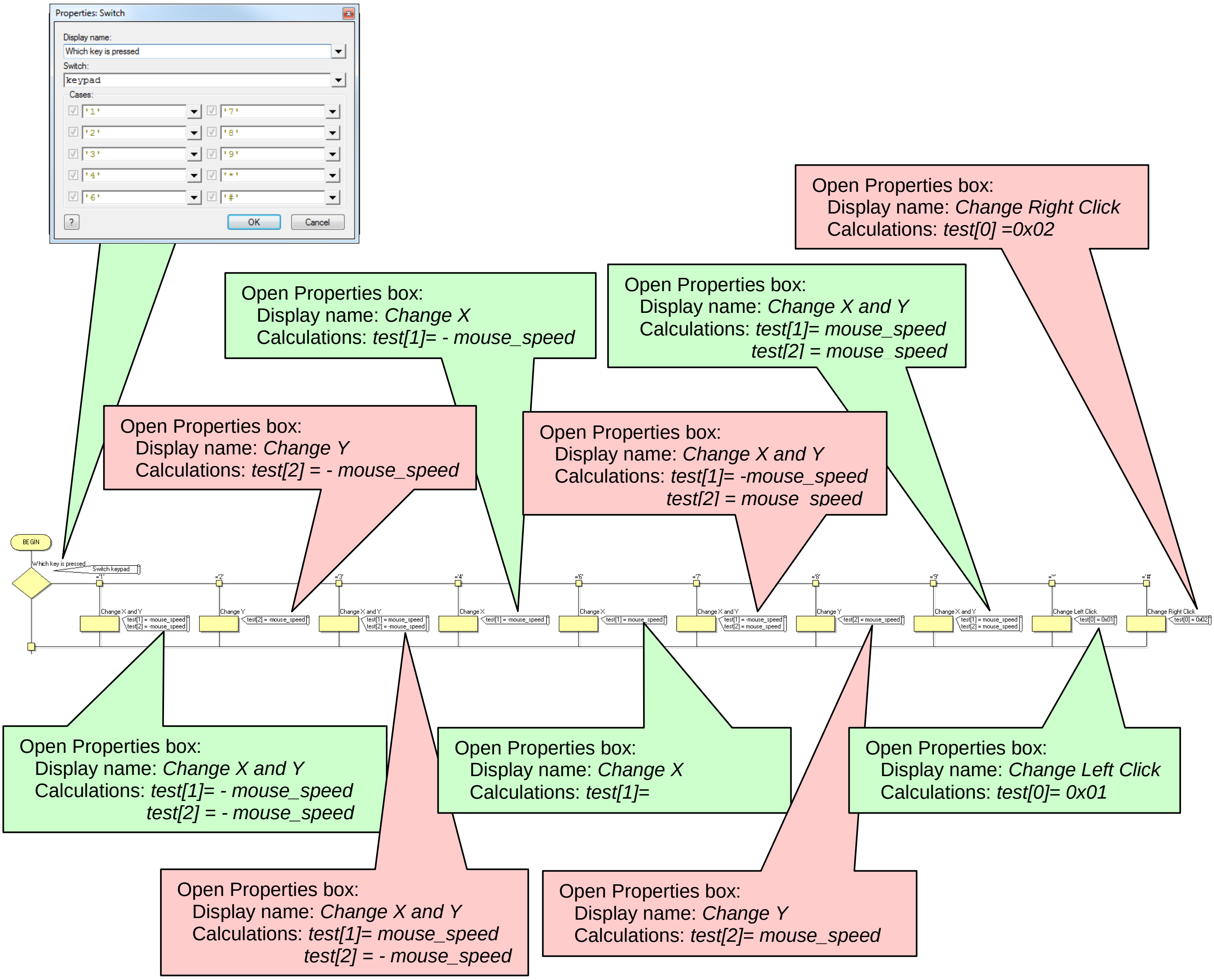
Open Properties box:
Display name: *Scan Keypad*
Component: *KeyPad(0)*
Open Variables box:
Create new variable: *byte, keypad*
Macro: *GetAscii*
Return Value *keypad*

Open Properties box:
Display name: *If Key Pressed?*
If: *keypad<255*

Open Properties box:
Display name: *Convert and send data*
Macro: Convert_To_Keypress_String

Convert_To_Keypress_String macro

1. Switch Case icon -



Properties: Switch

Display name: Which key is pressed

Switch: keypad

Cases:

☑ '1' ☑ '7'

☑ '2' ☑ '8'

☑ '3' ☑ '9'

☑ '4' ☑ '*'

☑ '6' ☑ '#'

Open Properties box:
Display name: *Change Right Click*
Calculations: *test[0] =0x02*

Open Properties box:
Display name: *Change X*
Calculations: *test[1]= - mouse_speed*

Open Properties box:
Display name: *Change X and Y*
Calculations: *test[1]= mouse_speed*
test[2] = mouse_speed

Open Properties box:
Display name: *Change Y*
Calculations: *test[2] = - mouse_speed*

Open Properties box:
Display name: *Change X and Y*
Calculations: *test[1]= -mouse_speed*
test[2] = mouse_speed

Open Properties box:
Display name: *Change X and Y*
Calculations: *test[1]= - mouse_speed*
test[2] = - mouse_speed

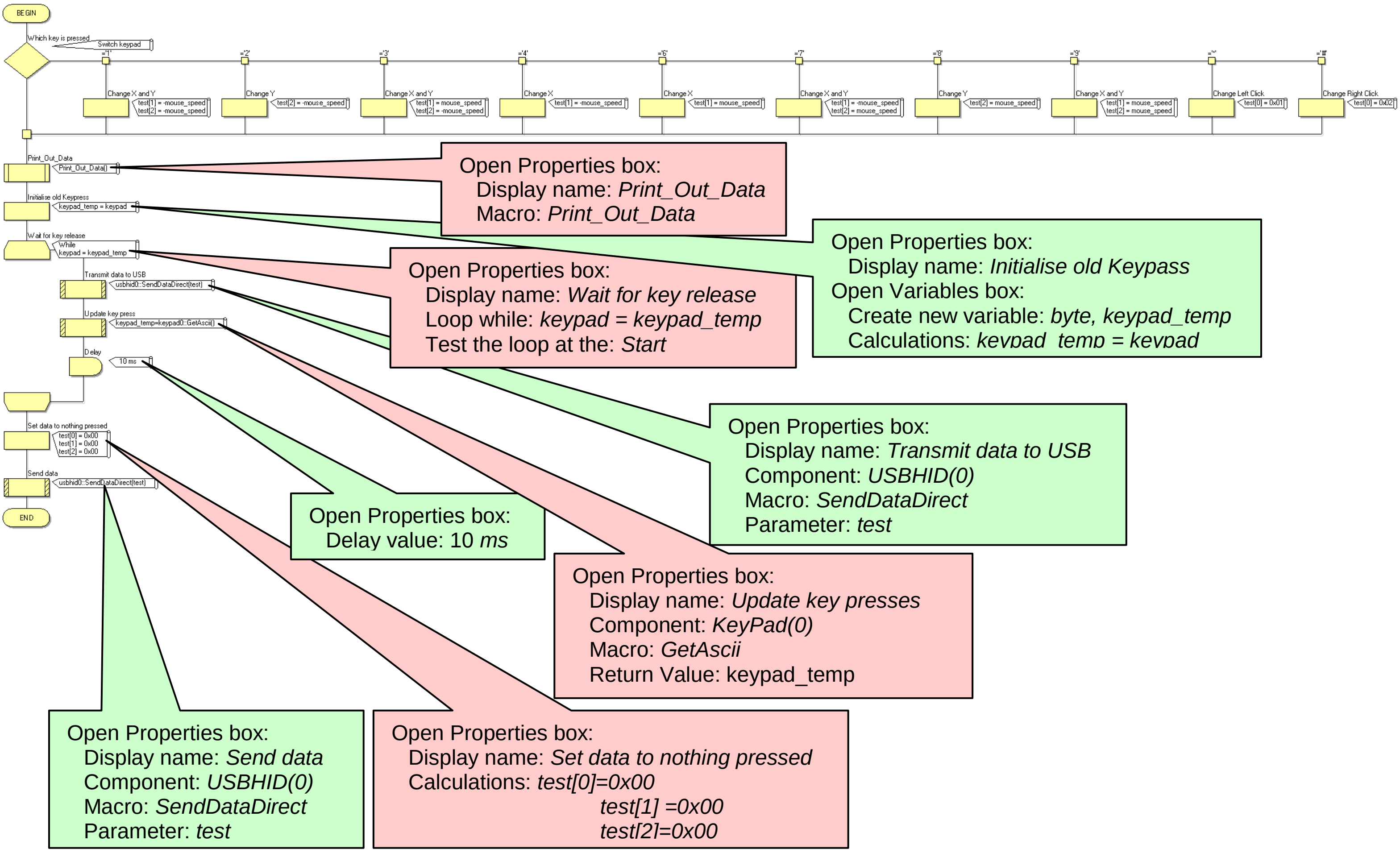
Open Properties box:
Display name: *Change X*
Calculations: *test[1]=*

Open Properties box:
Display name: *Change Left Click*
Calculations: *test[0]= 0x01*

Open Properties box:
Display name: *Change X and Y*
Calculations: *test[1]= mouse_speed*
test[2] = - mouse_speed

Open Properties box:
Display name: *Change Y*
Calculations: *test[2]= mouse_speed*

2. Other icons –



Open Properties box:
Display name: *Print_Out_Data*
Macro: *Print_Out_Data*

Open Properties box:
Display name: *Initialise old Keypass*
Open Variables box:
Create new variable: *byte, keypad_temp*
Calculations: *keypad_temp = keypad*

Open Properties box:
Display name: *Wait for key release*
Loop while: *keypad = keypad_temp*
Test the loop at the: *Start*

Open Properties box:
Display name: *Transmit data to USB*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *test*

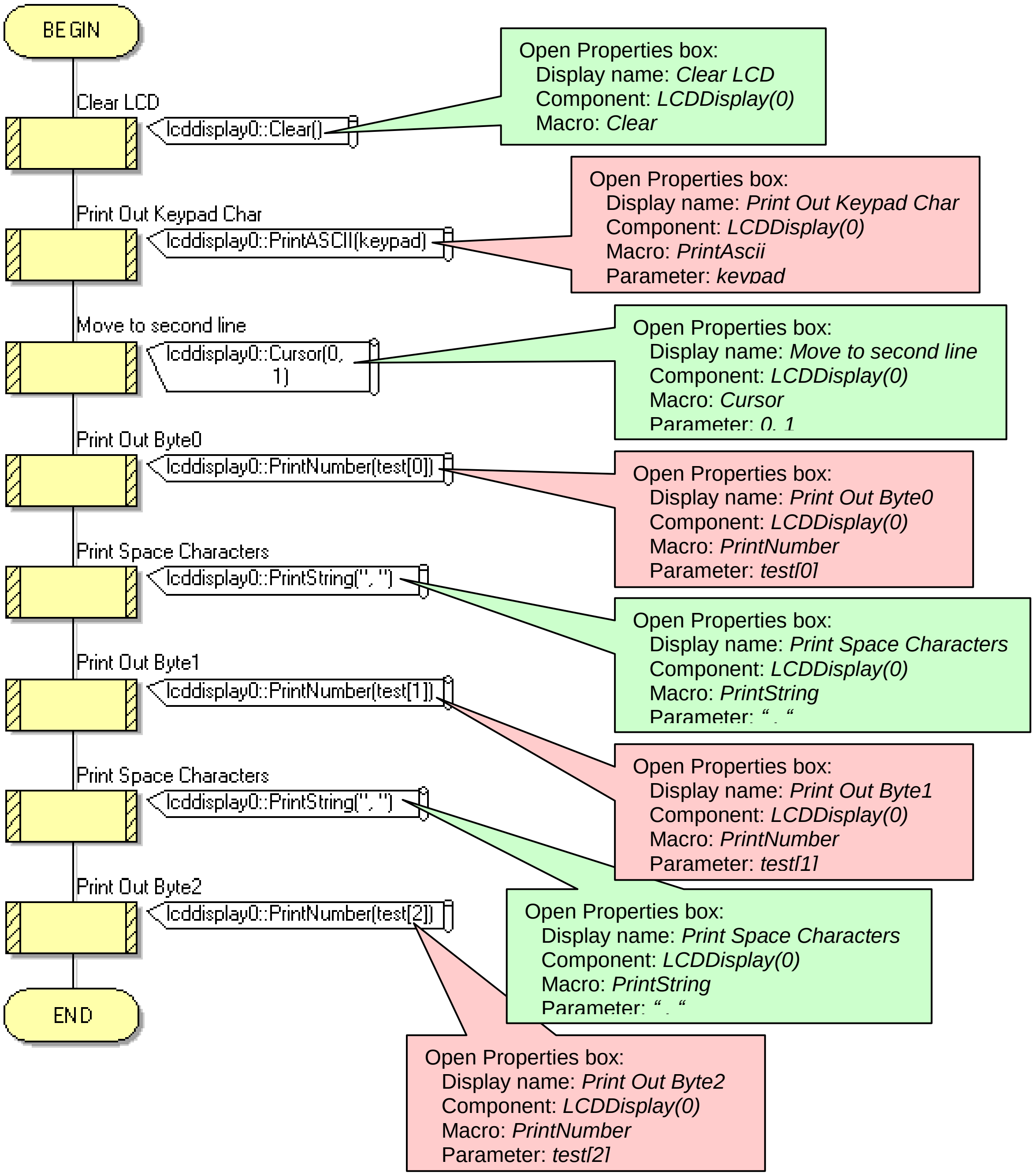
Open Properties box:
Delay value: 10 *ms*

Open Properties box:
Display name: *Update key presses*
Component: *KeyPad(0)*
Macro: *GetAscii*
Return Value: keypad_temp

Open Properties box:
Display name: *Send data*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *test*

Open Properties box:
Display name: *Set data to nothing pressed*
Calculations: *test[0]=0x00*
test[1] =0x00
test[2]=0x00

Print_Out_Data macro


BEGIN
Clear LCD
lcddisplay0::Clear()
Print Out Keypad Char
lcddisplay0::PrintASCII(keypad)
Move to second line
lcddisplay0::Cursor(0, 1)
Print Out Byte0
lcddisplay0::PrintNumber(test[0])
Print Space Characters
lcddisplay0::PrintString(", ")
Print Out Byte1
lcddisplay0::PrintNumber(test[1])
Print Space Characters
lcddisplay0::PrintString(", ")
Print Out Byte2
lcddisplay0::PrintNumber(test[2])
END

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Print Out Keypad Char*
Component: *LCDDisplay(0)*
Macro: *PrintAscii*
Parameter: *keypad*

Open Properties box:
Display name: *Move to second line*
Component: *LCDDisplay(0)*
Macro: *Cursor*
Parameter: *0, 1*

Open Properties box:
Display name: *Print Out Byte0*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[0]*

Open Properties box:
Display name: *Print Space Characters*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *“ , “*

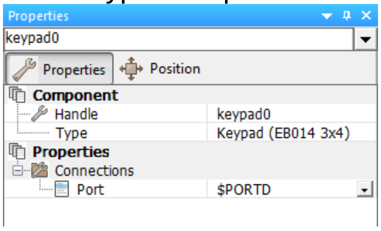
Open Properties box:
Display name: *Print Out Byte1*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[1]*

Open Properties box:
Display name: *Print Space Characters*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *“ , “*

Open Properties box:
Display name: *Print Out Byte2*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[2]*

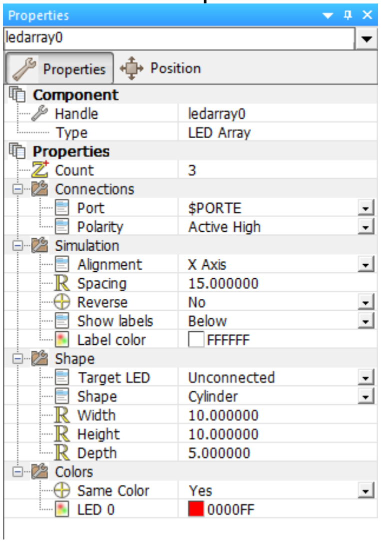
Exercise 2:

Keypad Properties



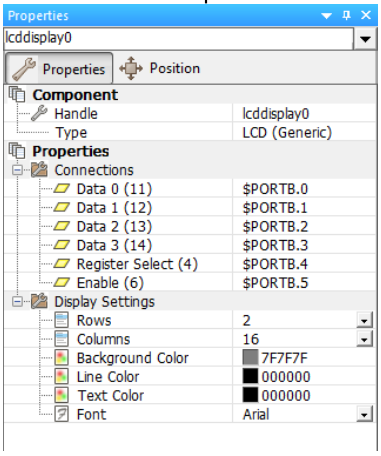
Component	
Handle	keypad0
Type	Keypad (EB014 3x4)
Properties	
Connections	
Port	$PORTD

LED Properties



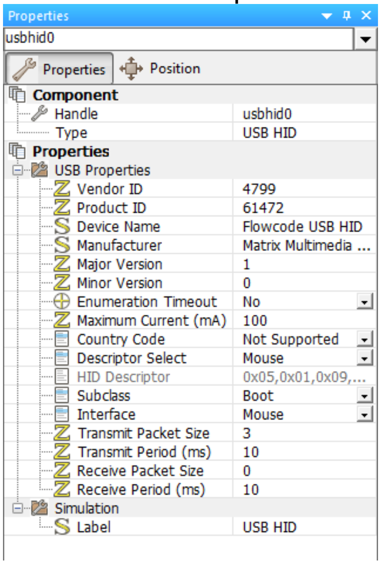
Component	
Handle	ledarray0
Type	LED Array
Properties	
Count	3
Connections	
Port	$PORTE
Polarity	Active High
Simulation	
Alignment	X Axis
Spacing	15.000000
Reverse	No
Show labels	Below
Label color	□ FFFFFF
Shape	
Target LED	Unconnected
Shape	Cylinder
Width	10.000000
Height	10.000000
Depth	5.000000
Colors	
Same Color	Yes
LED 0	🟥 0000FF

LCD Properties



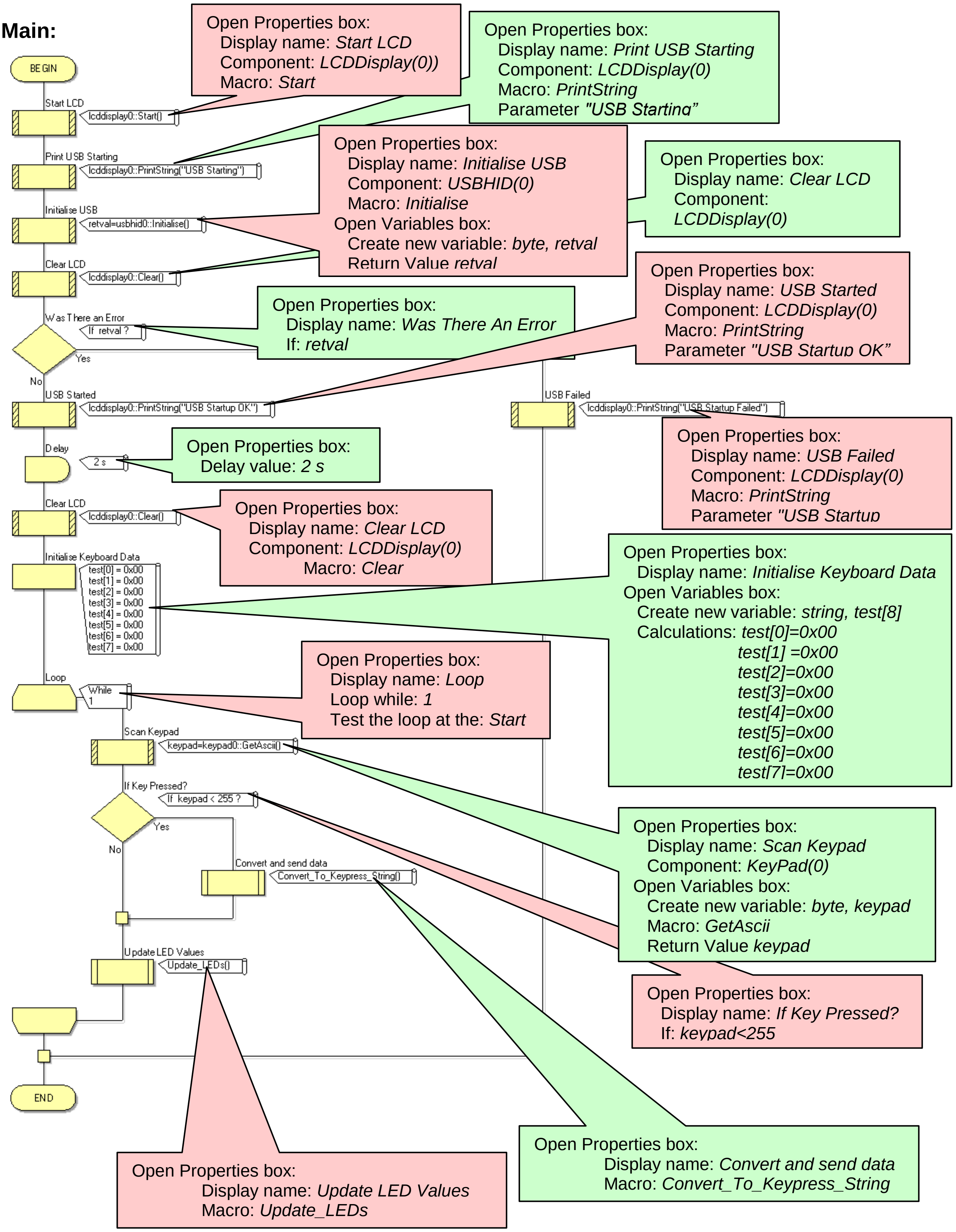
Component	
Handle	lcddisplay0
Type	LCD (Generic)
Properties	
Connections	
Data 0 (11)	$PORTB.0
Data 1 (12)	$PORTB.1
Data 2 (13)	$PORTB.2
Data 3 (14)	$PORTB.3
Register Select (4)	$PORTB.4
Enable (6)	$PORTB.5
Display Settings	
Rows	2
Columns	16
Background Color	▨ 7F7F7F
Line Color	■ 000000
Text Color	■ 000000
Font	Arial

USB HID Properties



Component	
Handle	usbhid0
Type	USB HID
Properties	
USB Properties	
Vendor ID	4799
Product ID	61472
Device Name	Flowcode USB HID
Manufacturer	Matrix Multimedia ...
Major Version	1
Minor Version	0
Enumeration Timeout	No
Maximum Current (mA)	100
Country Code	Not Supported
Descriptor Select	Mouse
HID Descriptor	0x05,0x01,0x09,...
Subclass	Boot
Interface	Mouse
Transmit Packet Size	3
Transmit Period (ms)	10
Receive Packet Size	0
Receive Period (ms)	10
Simulation	
Label	USB HID

Main:



Open Properties box:
Display name: *Start LCD*
Component: *LCDDisplay(0))*
Macro: *Start*

Open Properties box:
Display name: *Print USB Starting*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Starting"*

Open Properties box:
Display name: *Initialise USB*
Component: *USBHID(0)*
Macro: *Initialise*
Open Variables box:
Create new variable: *byte, retval*
Return Value *retval*

Open Properties box:
Display name: *Clear LCD*
Component:
LCDDisplay(0)

Open Properties box:
Display name: *USB Started*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Startup OK"*

Open Properties box:
Display name: *Was There An Error*
If: *retval*

Open Properties box:
Display name: *USB Failed*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Startup*

Open Properties box:
Delay value: *2 s*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Initialise Keyboard Data*
Open Variables box:
Create new variable: *string, test[8]*
Calculations: *test[0]=0x00*
test[1] =0x00
test[2]=0x00
test[3]=0x00
test[4]=0x00
test[5]=0x00
test[6]=0x00
test[7]=0x00

Open Properties box:
Display name: *Loop*
Loop while: *1*
Test the loop at the: *Start*

Open Properties box:
Display name: *Scan Keypad*
Component: *KeyPad(0)*
Open Variables box:
Create new variable: *byte, keypad*
Macro: *GetAscii*
Return Value *keypad*

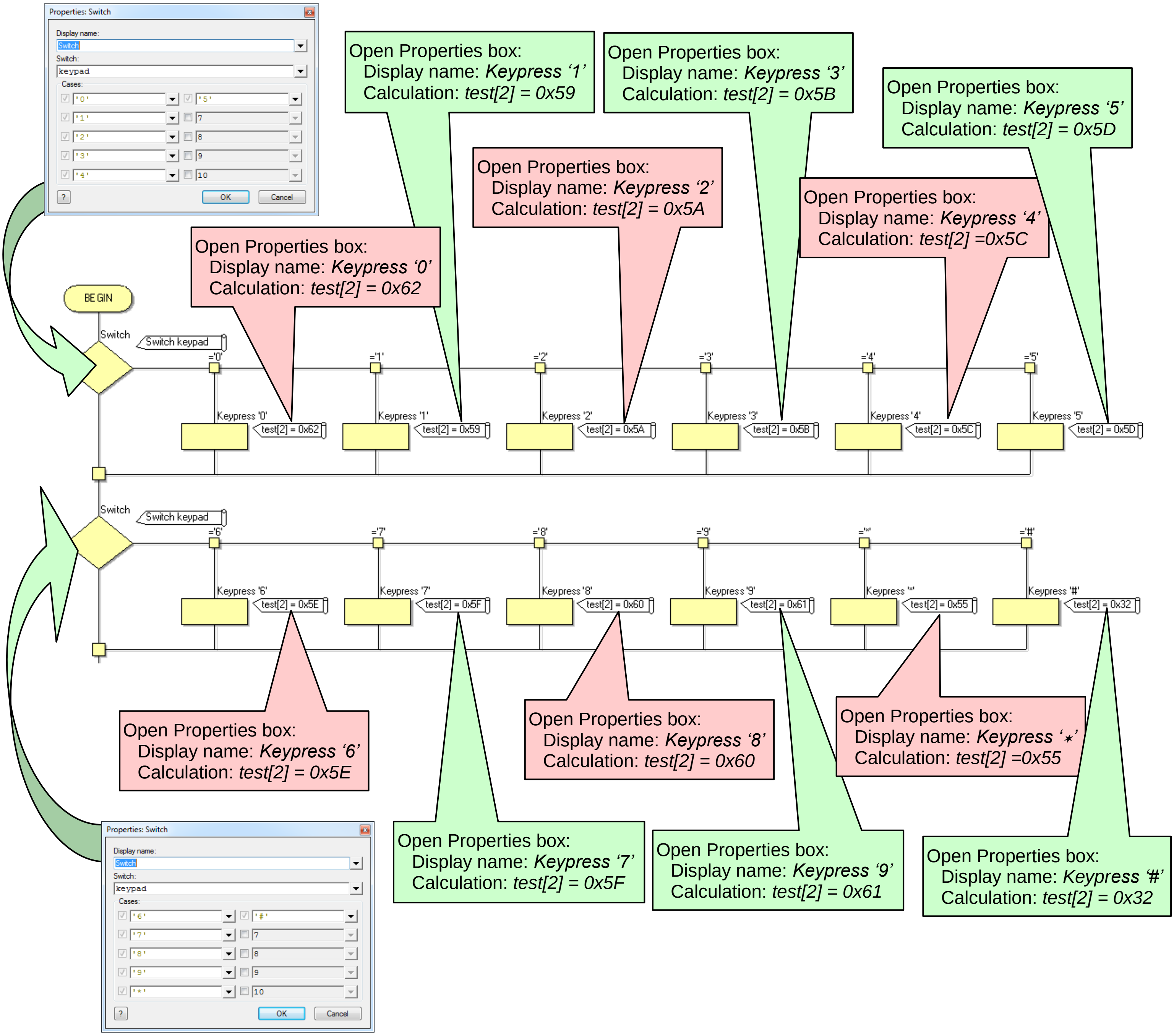
Open Properties box:
Display name: *If Key Pressed?*
If: *keypad<255*

Open Properties box:
Display name: *Convert and send data*
Macro: *Convert_To_Keypress_String*

Open Properties box:
Display name: *Update LED Values*
Macro: *Update_LEDs*

Convert_To_Keypress_String macro

1. Switch case icons -



Properties: Switch
Display name: Switch
Switch: keypad
Cases:
☑ '0' ☑ '5'
☑ '1' □ 7
☑ '2' □ 8
☑ '3' □ 9
☑ '4' □ 10

Open Properties box:
Display name: *Keypress '0'*
Calculation: *test[2] = 0x62*

Open Properties box:
Display name: *Keypress '1'*
Calculation: *test[2] = 0x59*

Open Properties box:
Display name: *Keypress '2'*
Calculation: *test[2] = 0x5A*

Open Properties box:
Display name: *Keypress '3'*
Calculation: *test[2] = 0x5B*

Open Properties box:
Display name: *Keypress '4'*
Calculation: *test[2] =0x5C*

Open Properties box:
Display name: *Keypress '5'*
Calculation: *test[2] = 0x5D*

Open Properties box:
Display name: *Keypress '6'*
Calculation: *test[2] = 0x5E*

Open Properties box:
Display name: *Keypress '7'*
Calculation: *test[2] = 0x5F*

Open Properties box:
Display name: *Keypress '8'*
Calculation: *test[2] = 0x60*

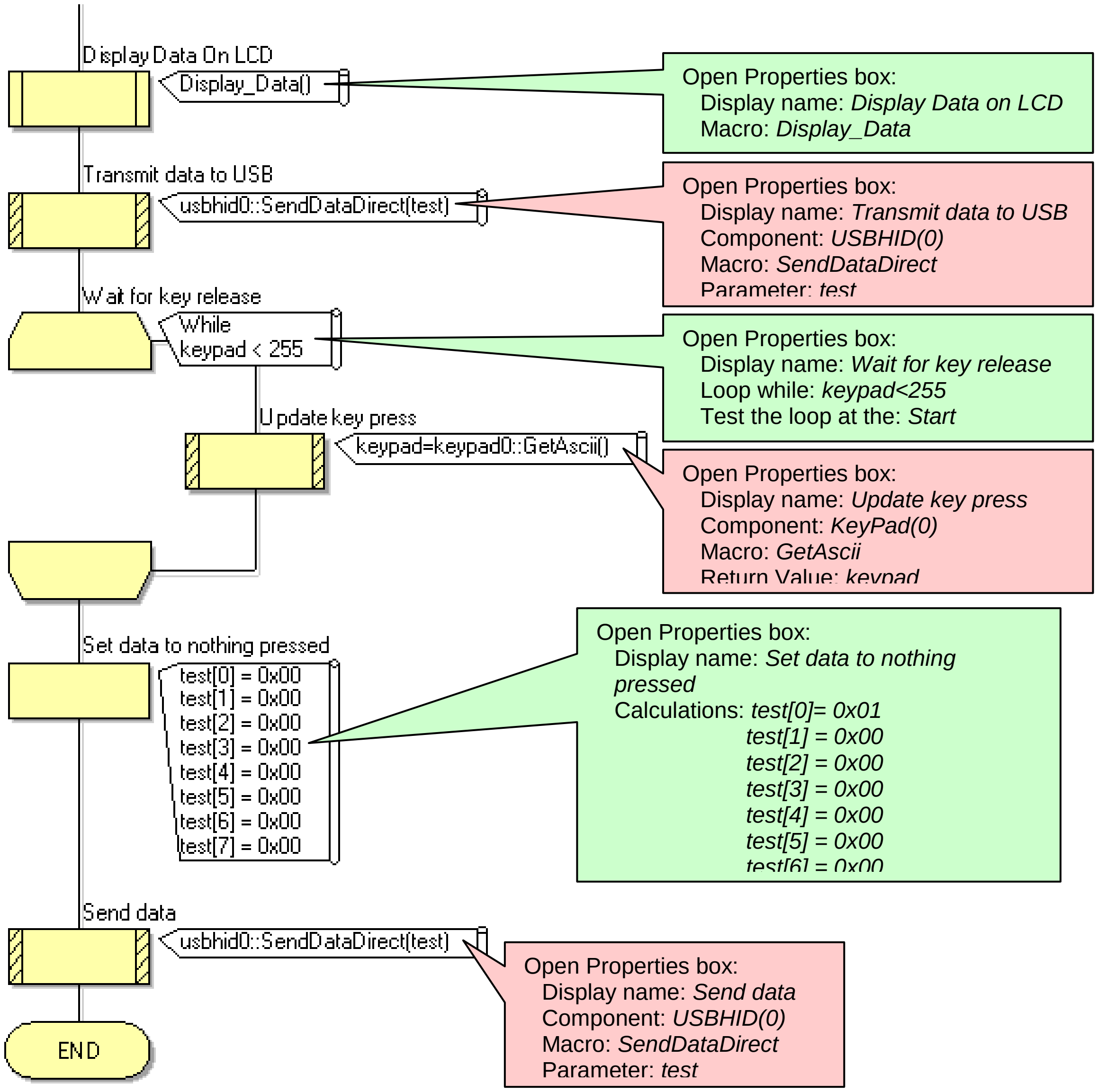
Open Properties box:
Display name: *Keypress '9'*
Calculation: *test[2] = 0x61*

Open Properties box:
Display name: *Keypress '*'*
Calculation: *test[2] =0x55*

Open Properties box:
Display name: *Keypress '#'*
Calculation: *test[2] = 0x32*

Properties: Switch
Display name: Switch
Switch: keypad
Cases:
☑ '6' ☑ '#'
☑ '7' □ 7
☑ '8' □ 8
☑ '9' □ 9
☑ '*' □ 10

2. Other icons -



Open Properties box:
Display name: *Display Data on LCD*
Macro: *Display_Data*

Open Properties box:
Display name: *Transmit data to USB*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *test*

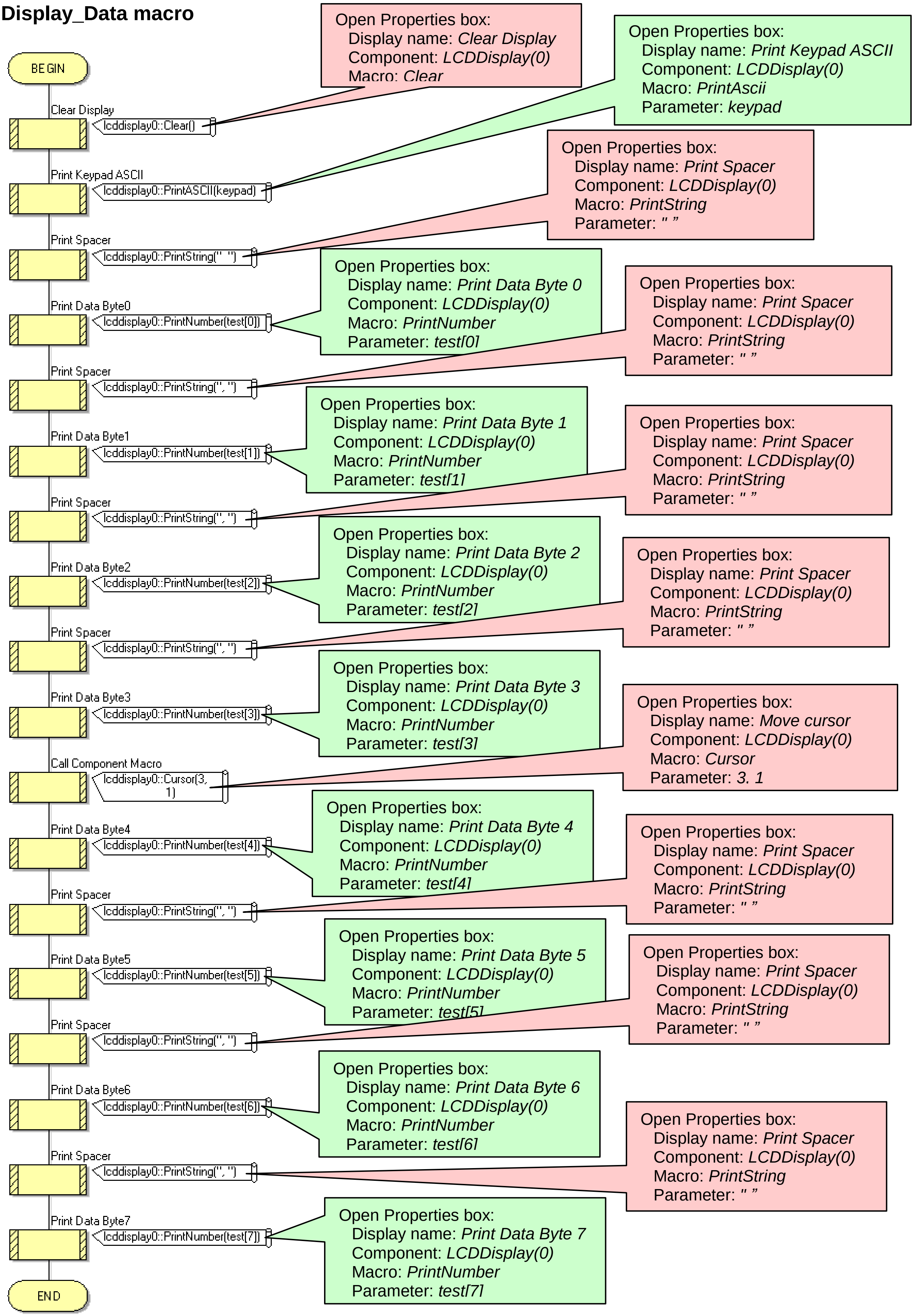
Open Properties box:
Display name: *Wait for key release*
Loop while: *keypad<255*
Test the loop at the: *Start*

Open Properties box:
Display name: *Update key press*
Component: *KeyPad(0)*
Macro: *GetAscii*
Return Value: *keypad*

Open Properties box:
Display name: *Set data to nothing pressed*
Calculations: *test[0]= 0x01*
test[1] = 0x00
test[2] = 0x00
test[3] = 0x00
test[4] = 0x00
test[5] = 0x00
test[6] = 0x00

Open Properties box:
Display name: *Send data*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *test*

Display_Data macro



BEGIN

Clear Display
lcddisplay0::Clear()

Open Properties box:
Display name: *Clear Display*
Component: *LCDDisplay(0)*
Macro: *Clear*

Print Keypad ASCII
lcddisplay0::PrintASCII(keypad)

Open Properties box:
Display name: *Print Keypad ASCII*
Component: *LCDDisplay(0)*
Macro: *PrintAscii*
Parameter: *keypad*

Print Spacer
lcddisplay0::PrintString(" ")

Open Properties box:
Display name: *Print Spacer*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *" ”*

Print Data Byte0
lcddisplay0::PrintNumber(test[0])

Open Properties box:
Display name: *Print Data Byte 0*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[0]*

Print Spacer
lcddisplay0::PrintString(", ")

Open Properties box:
Display name: *Print Spacer*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *" ”*

Print Data Byte1
lcddisplay0::PrintNumber(test[1])

Open Properties box:
Display name: *Print Data Byte 1*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[1]*

Print Spacer
lcddisplay0::PrintString(", ")

Open Properties box:
Display name: *Print Spacer*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *" ”*

Print Data Byte2
lcddisplay0::PrintNumber(test[2])

Open Properties box:
Display name: *Print Data Byte 2*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[2]*

Print Spacer
lcddisplay0::PrintString(", ")

Open Properties box:
Display name: *Print Spacer*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *" ”*

Print Data Byte3
lcddisplay0::PrintNumber(test[3])

Open Properties box:
Display name: *Print Data Byte 3*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[3]*

Call Component Macro
lcddisplay0::Cursor(3, 1)

Open Properties box:
Display name: *Move cursor*
Component: *LCDDisplay(0)*
Macro: *Cursor*
Parameter: *3. 1*

Print Data Byte4
lcddisplay0::PrintNumber(test[4])

Open Properties box:
Display name: *Print Data Byte 4*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[4]*

Print Spacer
lcddisplay0::PrintString(", ")

Open Properties box:
Display name: *Print Spacer*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *" ”*

Print Data Byte5
lcddisplay0::PrintNumber(test[5])

Open Properties box:
Display name: *Print Data Byte 5*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[5]*

Print Spacer
lcddisplay0::PrintString(", ")

Open Properties box:
Display name: *Print Spacer*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *" ”*

Print Data Byte6
lcddisplay0::PrintNumber(test[6])

Open Properties box:
Display name: *Print Data Byte 6*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[6]*

Print Spacer
lcddisplay0::PrintString(", ")

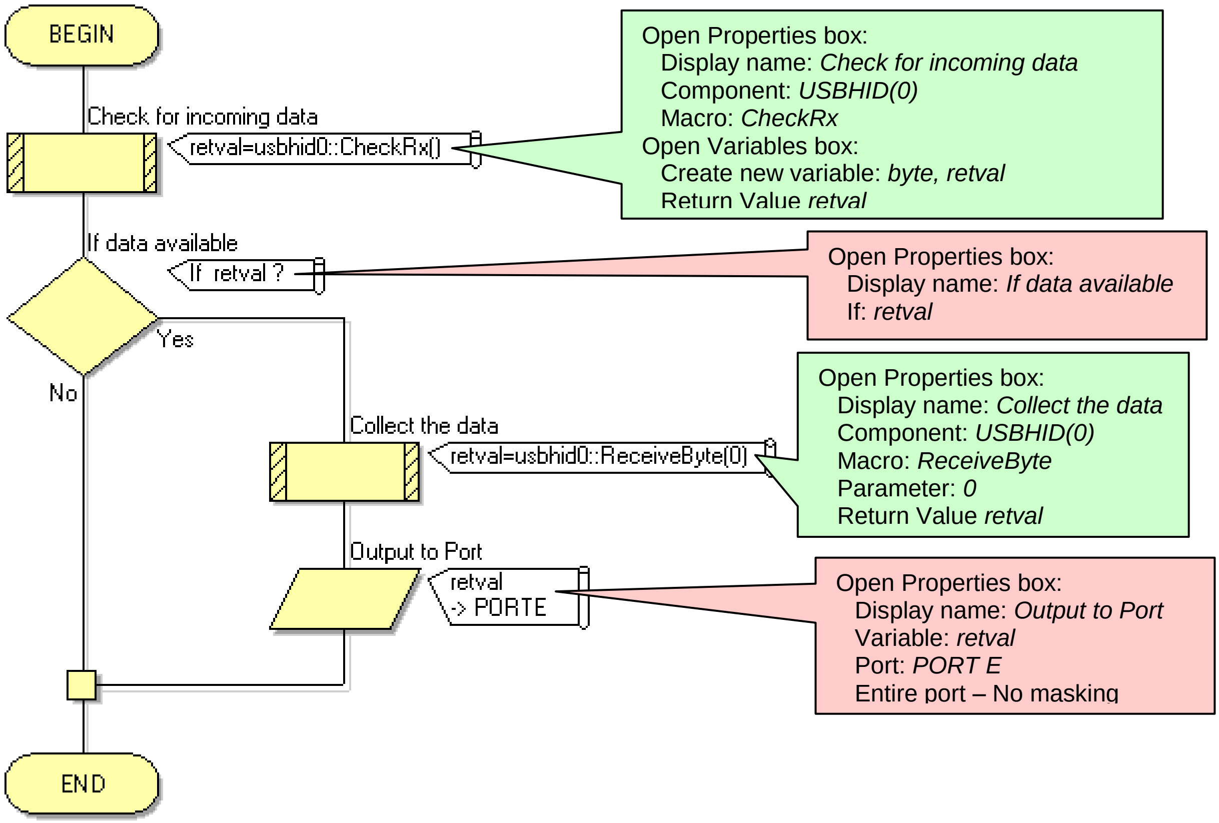
Open Properties box:
Display name: *Print Spacer*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *" ”*

Print Data Byte7
lcddisplay0::PrintNumber(test[7])

Open Properties box:
Display name: *Print Data Byte 7*
Component: *LCDDisplay(0)*
Macro: *PrintNumber*
Parameter: *test[7]*

END

Update_LEDs macro:



Open Properties box:
Display name: *Check for incoming data*
Component: *USBHID(0)*
Macro: *CheckRx*
Open Variables box:
Create new variable: *byte, retval*
Return Value *retval*

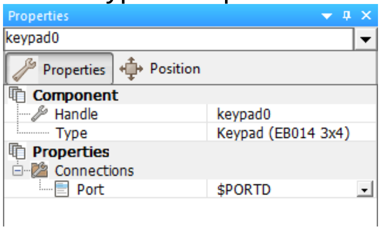
Open Properties box:
Display name: *If data available*
If: *retval*

Open Properties box:
Display name: *Collect the data*
Component: *USBHID(0)*
Macro: *ReceiveByte*
Parameter: *0*
Return Value *retval*

Open Properties box:
Display name: *Output to Port*
Variable: *retval*
Port: *PORT E*
Entire port – No masking

Exercise 3:

Keypad Properties



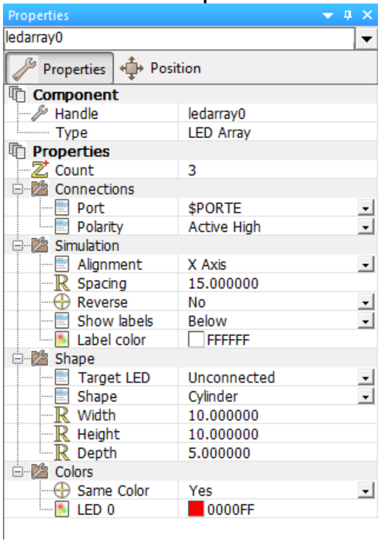
Properties

keypad0

Properties | Position

Component	
Handle	keypad0
Type	Keypad (EB014 3x4)
Properties	
Connections	
Port	$PORTD

LED Properties



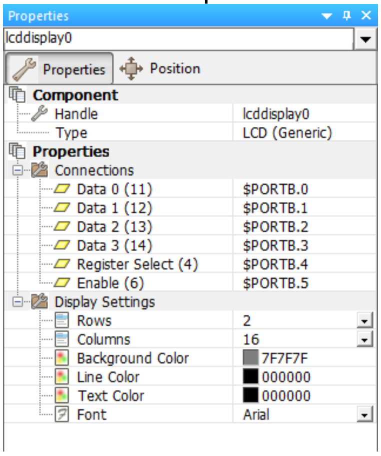
Properties

ledarray0

Properties | Position

Component	
Handle	ledarray0
Type	LED Array
Properties	
Count	3
Connections	
Port	$PORTE
Polarity	Active High
Simulation	
Alignment	X Axis
Spacing	15.000000
Reverse	No
Show labels	Below
Label color	□ FFFFFF
Shape	
Target LED	Unconnected
Shape	Cylinder
Width	10.000000
Height	10.000000
Depth	5.000000
Colors	
Same Color	Yes
LED 0	🟥 0000FF

LCD Properties



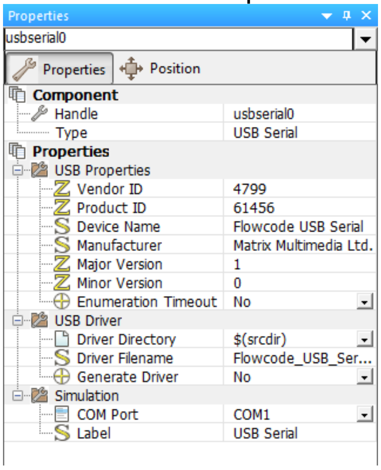
Properties

lcddisplay0

Properties | Position

Component	
Handle	lcddisplay0
Type	LCD (Generic)
Properties	
Connections	
Data 0 (11)	$PORTB.0
Data 1 (12)	$PORTB.1
Data 2 (13)	$PORTB.2
Data 3 (14)	$PORTB.3
Register Select (4)	$PORTB.4
Enable (6)	$PORTB.5
Display Settings	
Rows	2
Columns	16
Background Color	▨ 7F7F7F
Line Color	⬛ 000000
Text Color	⬛ 000000
Font	Arial

USB Serial Properties



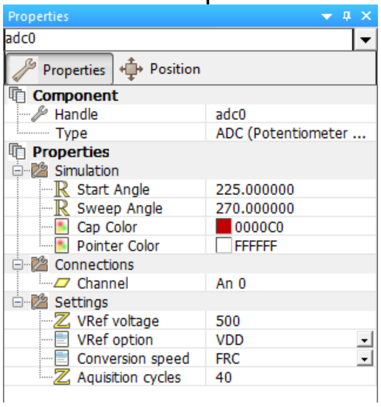
Properties

usbserial0

Properties | Position

Component	
Handle	usbserial0
Type	USB Serial
Properties	
USB Properties	
Vendor ID	4799
Product ID	61456
Device Name	Flowcode USB Serial
Manufacturer	Matrix Multimedia Ltd.
Major Version	1
Minor Version	0
Enumeration Timeout	No
USB Driver	
Driver Directory	$(srcdir)
Driver Filename	Flowcode_USB_Ser...
Generate Driver	No
Simulation	
COM Port	COM1
Label	USB Serial

ADC Properties



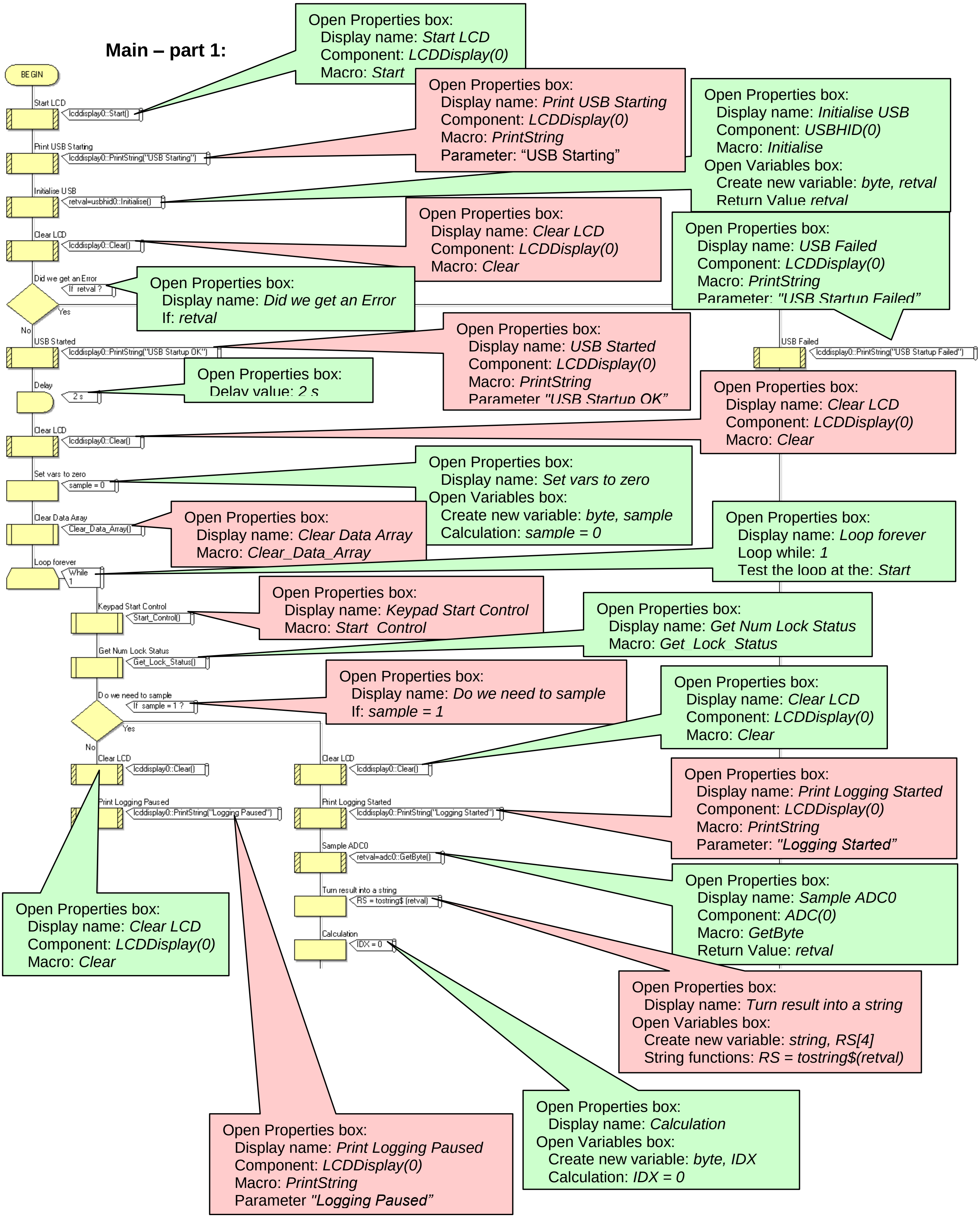
Properties

adc0

Properties | Position

Component	
Handle	adc0
Type	ADC (Potentiometer ...
Properties	
Simulation	
Start Angle	225.000000
Sweep Angle	270.000000
Cap Color	🟥 0000C0
Pointer Color	□ FFFFFF
Connections	
Channel	An 0
Settings	
VRef voltage	500
VRef option	VDD
Conversion speed	FRC
Aquisition cycles	40

Main – part 1:



Open Properties box:
Display name: *Start LCD*
Component: *LCDDisplay(0)*
Macro: *Start*

Open Properties box:
Display name: *Print USB Starting*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: "USB Starting"

Open Properties box:
Display name: *Initialise USB*
Component: *USBHID(0)*
Macro: *Initialise*
Open Variables box:
Create new variable: *byte, retval*
Return Value *retval*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *USB Failed*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB Startup Failed"*

Open Properties box:
Display name: *Did we get an Error*
If: *retval*

Open Properties box:
Display name: *USB Started*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Startup OK"*

Open Properties box:
Delay value: *2 s*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Set vars to zero*
Open Variables box:
Create new variable: *byte, sample*
Calculation: *sample = 0*

Open Properties box:
Display name: *Clear Data Array*
Macro: *Clear_Data_Array*

Open Properties box:
Display name: *Loop forever*
Loop while: *1*
Test the loop at the: *Start*

Open Properties box:
Display name: *Keypad Start Control*
Macro: *Start_Control*

Open Properties box:
Display name: *Get Num Lock Status*
Macro: *Get_Lock_Status*

Open Properties box:
Display name: *Do we need to sample*
If: *sample = 1*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Print Logging Started*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"Logging Started"*

Open Properties box:
Display name: *Sample ADC0*
Component: *ADC(0)*
Macro: *GetByte*
Return Value: *retval*

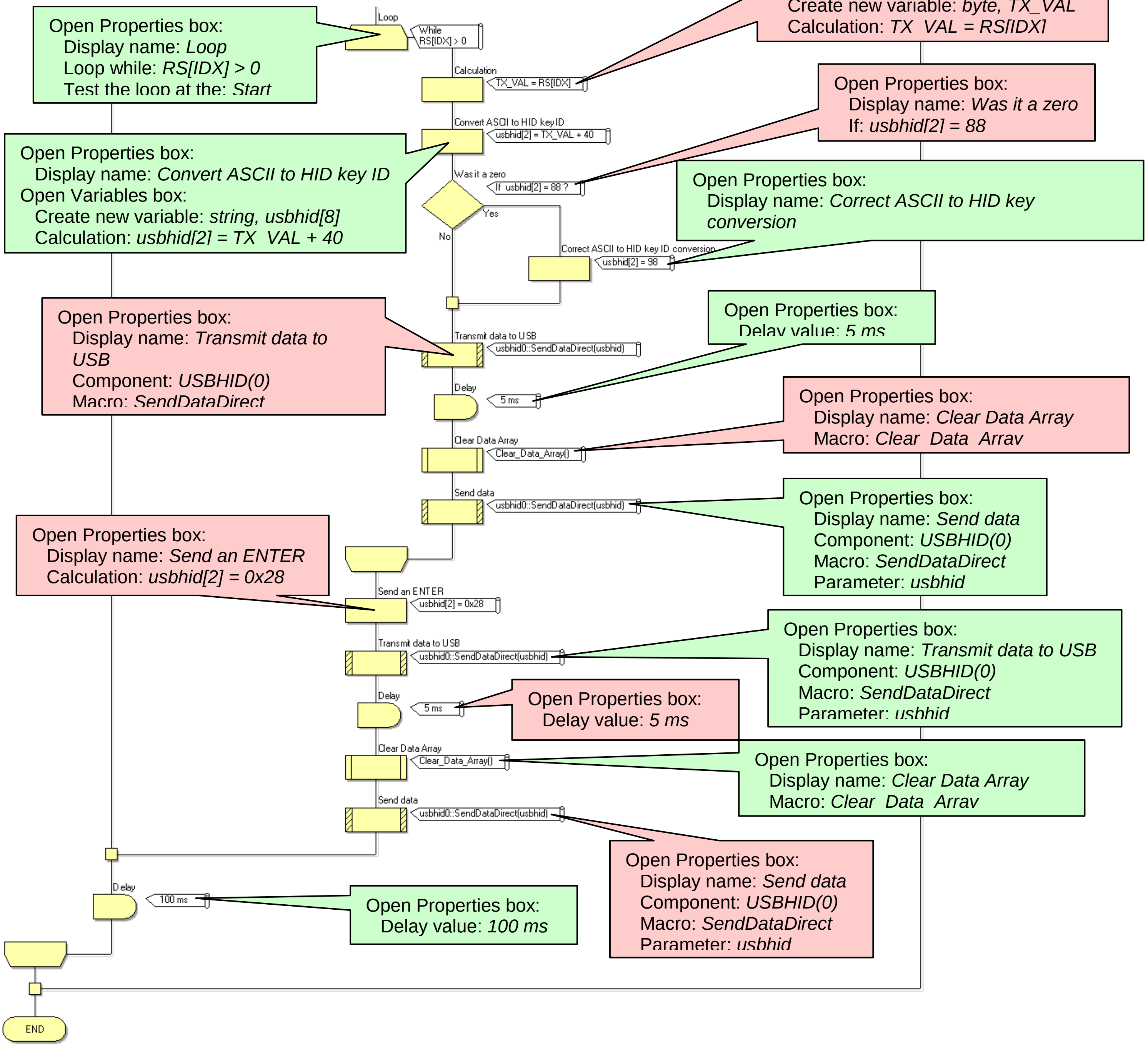
Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Turn result into a string*
Open Variables box:
Create new variable: *string, RS[4]*
String functions: *RS = tostring$(retval)*

Open Properties box:
Display name: *Calculation*
Open Variables box:
Create new variable: *byte, IDX*
Calculation: *IDX = 0*

Open Properties box:
Display name: *Print Logging Paused*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"Logging Paused"*

Main – part 2:



Open Properties box:
Display name: *Calculation*
Open Variables box:
Create new variable: *byte, TX_VAL*
Calculation: *TX VAL = RS[IDX]*

Open Properties box:
Display name: *Loop*
Loop while: *RS[IDX] > 0*
Test the loop at the: *Start*

Open Properties box:
Display name: *Was it a zero*
If: *usbhid[2] = 88*

Open Properties box:
Display name: *Convert ASCII to HID key ID*
Open Variables box:
Create new variable: *string, usbhid[8]*
Calculation: *usbhid[2] = TX VAL + 40*

Open Properties box:
Display name: *Correct ASCII to HID key conversion*

Open Properties box:
Display name: *Transmit data to USB*
Component: *USBHID(0)*
Macro: *SendDataDirect*

Open Properties box:
Delay value: *5 ms*

Open Properties box:
Display name: *Clear Data Array*
Macro: *Clear Data Array*

Open Properties box:
Display name: *Send data*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *usbhid*

Open Properties box:
Display name: *Send an ENTER*
Calculation: *usbhid[2] = 0x28*

Open Properties box:
Display name: *Transmit data to USB*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *usbhid*

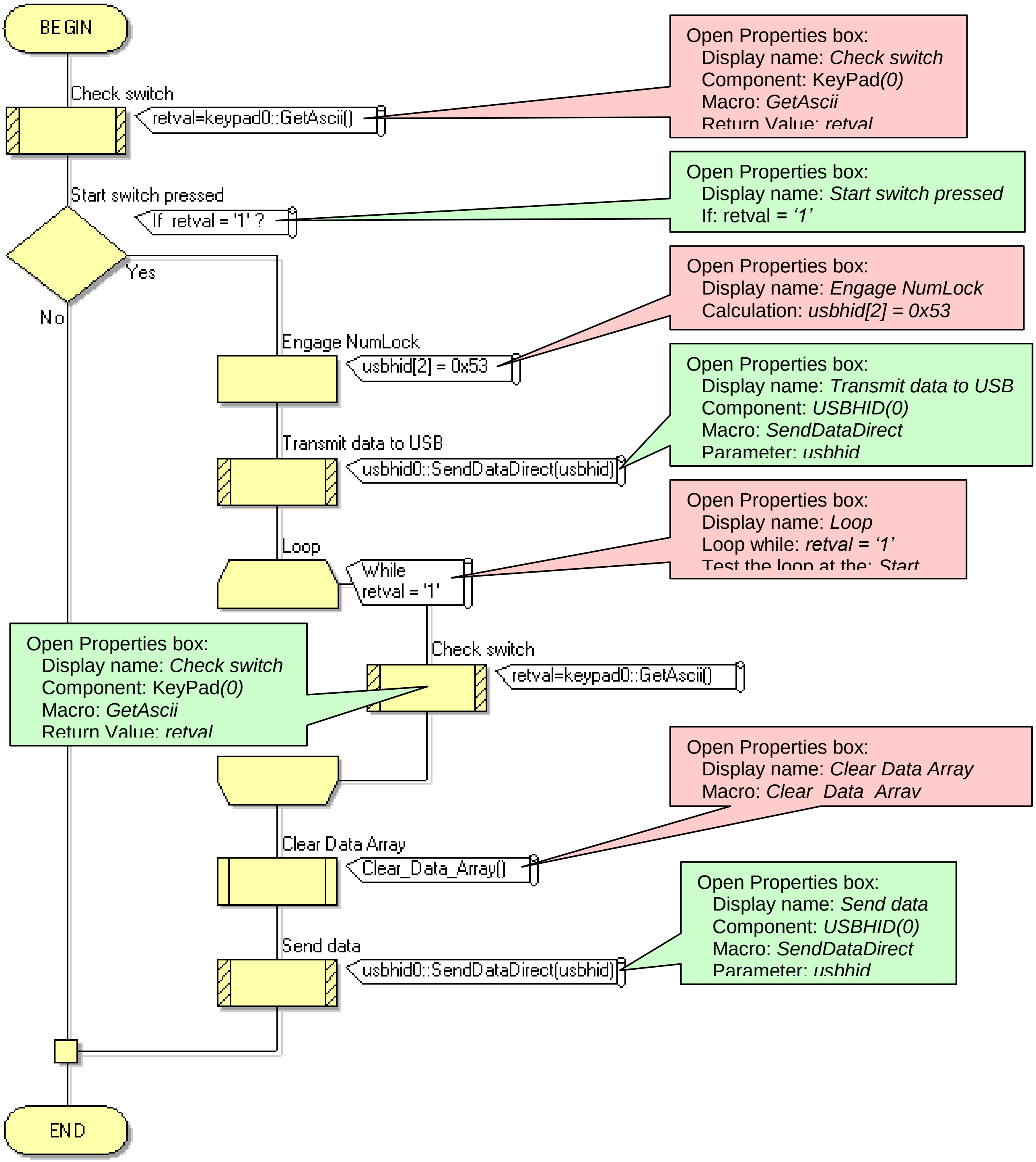
Open Properties box:
Delay value: *5 ms*

Open Properties box:
Display name: *Clear Data Array*
Macro: *Clear Data Array*

Open Properties box:
Display name: *Send data*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *usbhid*

Open Properties box:
Delay value: *100 ms*

Start_Control macro:



Open Properties box:
Display name: *Check switch*
Component: KeyPad*(0)*
Macro: *GetAscii*
Return Value: *retval*

Open Properties box:
Display name: *Start switch pressed*
If: retval = *'1'*

Open Properties box:
Display name: *Engage NumLock*
Calculation: *usbhid[2] = 0x53*

Open Properties box:
Display name: *Transmit data to USB*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *usbhid*

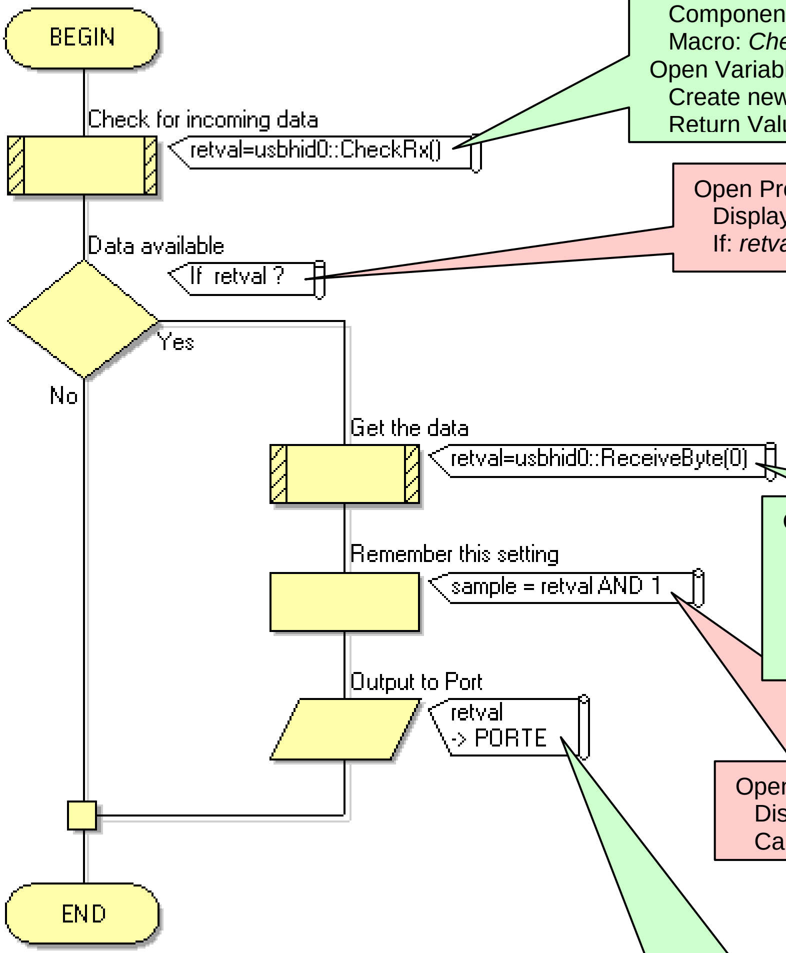
Open Properties box:
Display name: *Loop*
Loop while: ***retval* = *'1'***
Test the loop at the: *Start*

Open Properties box:
Display name: *Check switch*
Component: KeyPad*(0)*
Macro: *GetAscii*
Return Value: *retval*

Open Properties box:
Display name: *Clear Data Array*
Macro: *Clear_Data_Array*

Open Properties box:
Display name: *Send data*
Component: *USBHID(0)*
Macro: *SendDataDirect*
Parameter: *usbhid*

Get_Lock_Status macro:



Open Properties box:
  Display name: *Check for incoming data*
  Component: *USBHID(0)*
  Macro: *CheckRx*
Open Variables box:
  Create new variable: *byte, retval*
  Return Value *retval*

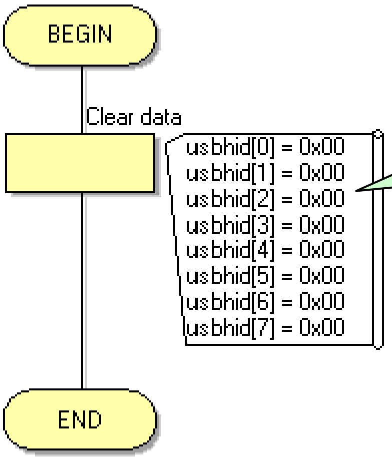
Open Properties box:
  Display name: *Data available*
  If: *retval*

Open Properties box:
  Display name: *Get the data*
  Component: *USBHID(0)*
  Macro: *ReceiveByte*
  Parameter: *0*
  Return Value *retval*

Open Properties box:
  Display name: *Remember this setting*
  Calculation: *sample = retval AND 1*

Open Properties box:
  Display name: *Output to Port*
  Variable: *retval*
  Port: *PORT E*
  Entire port – No masking

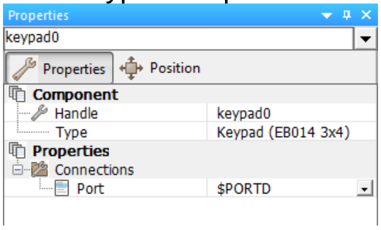
Clear_Data_Array macro:



Open Properties box:
  Display name: *Clear data*
  Calculations: *usbhid[0]= 0x00*
    usbhid[1] = 0x00
    usbhid[2] = 0x00
    usbhid[3] = 0x00
    usbhid[4] = 0x00
    usbhid[5] = 0x00
    usbhid[6] = 0x00
    usbhid[7] = 0x00

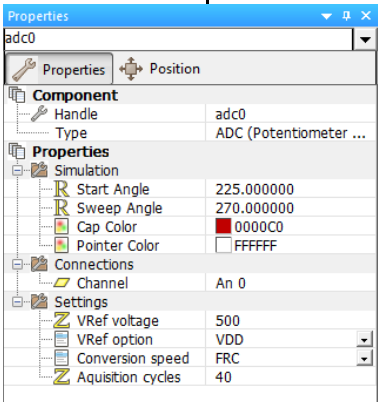
Exercise 4:

Keypad Properties



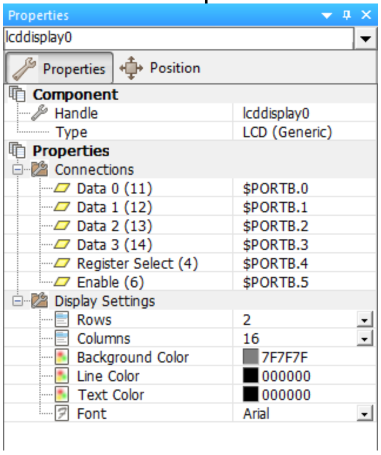
Component	
Handle	keypad0
Type	Keypad (EB014 3x4)
Properties	
Connections	
Port	$PORTD

ADC Properties



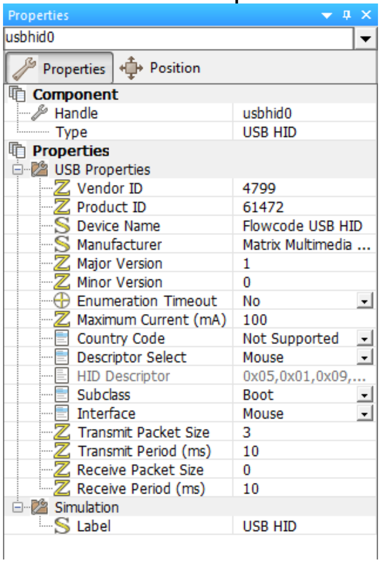
Component	
Handle	adc0
Type	ADC (Potentiometer ...
Properties	
Simulation	
Start Angle	225.000000
Sweep Angle	270.000000
Cap Color	🟥 0000C0
Pointer Color	□ FFFFFF
Connections	
Channel	An 0
Settings	
VRef voltage	500
VRef option	VDD
Conversion speed	FRC
Aquisition cycles	40

LCD Properties



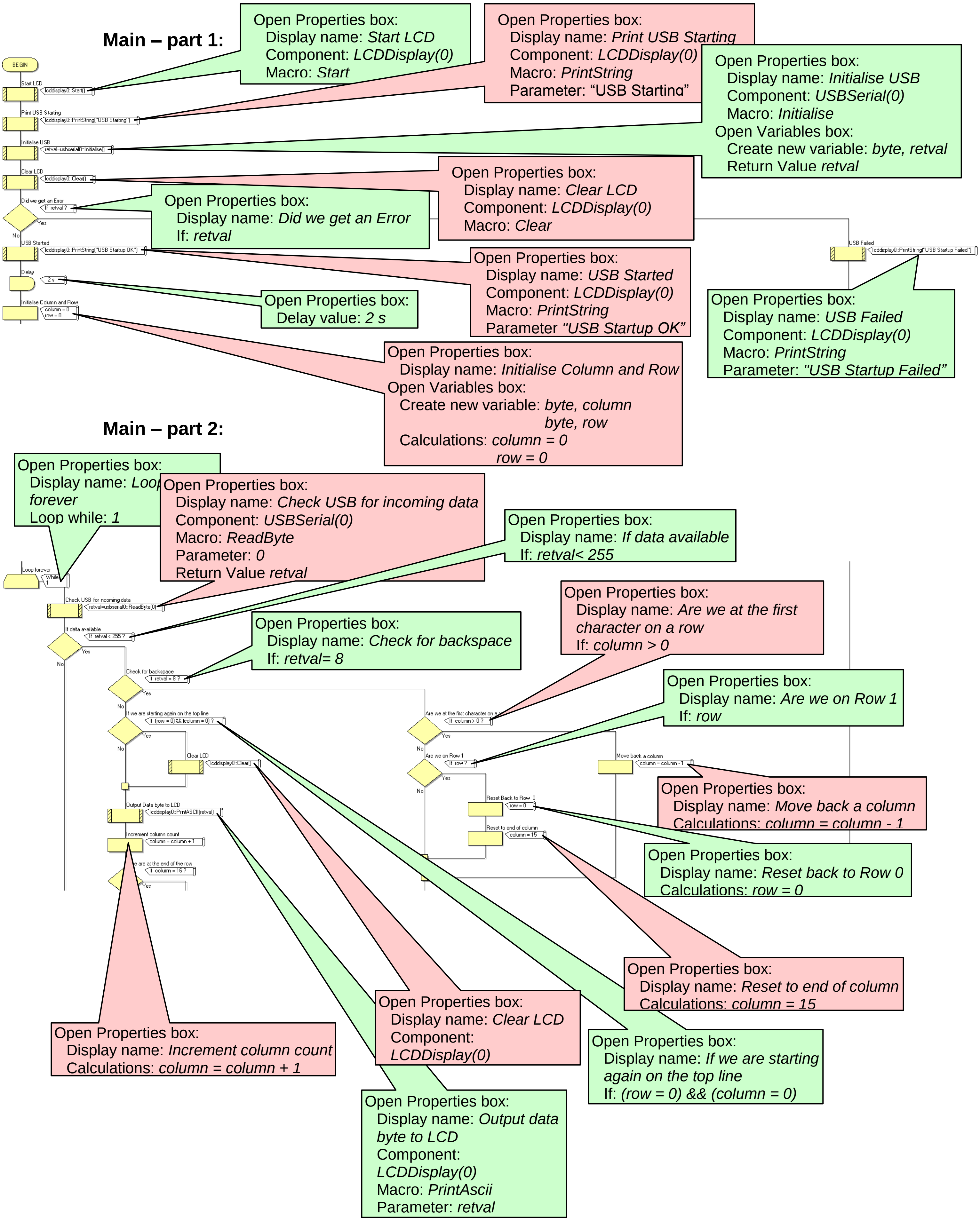
Component	
Handle	lcddisplay0
Type	LCD (Generic)
Properties	
Connections	
Data 0 (11)	$PORTB.0
Data 1 (12)	$PORTB.1
Data 2 (13)	$PORTB.2
Data 3 (14)	$PORTB.3
Register Select (4)	$PORTB.4
Enable (6)	$PORTB.5
Display Settings	
Rows	2
Columns	16
Background Color	▨ 7F7F7F
Line Color	■ 000000
Text Color	■ 000000
Font	Arial

USB HID Properties



Component	
Handle	usbhid0
Type	USB HID
Properties	
USB Properties	
Vendor ID	4799
Product ID	61472
Device Name	Flowcode USB HID
Manufacturer	Matrix Multimedia ...
Major Version	1
Minor Version	0
Enumeration Timeout	No
Maximum Current (mA)	100
Country Code	Not Supported
Descriptor Select	Mouse
HID Descriptor	0x05,0x01,0x09,...
Subclass	Boot
Interface	Mouse
Transmit Packet Size	3
Transmit Period (ms)	10
Receive Packet Size	0
Receive Period (ms)	10
Simulation	
Label	USB HID

Main – part 1:



Open Properties box:
Display name: *Start LCD*
Component: *LCDDisplay(0)*
Macro: *Start*

Open Properties box:
Display name: *Print USB Starting*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: "USB Starting"

Open Properties box:
Display name: *Initialise USB*
Component: *USBSerial(0)*
Macro: *Initialise*
Open Variables box:
Create new variable: *byte, retval*
Return Value *retval*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Did we get an Error*
If: *retval*

Open Properties box:
Display name: *USB Started*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Startup OK"*

Open Properties box:
Display name: *USB Failed*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB Startup Failed"*

Open Properties box:
Delay value: *2 s*

Open Properties box:
Display name: *Initialise Column and Row*
Open Variables box:
Create new variable: *byte, column*
byte, row
Calculations: *column = 0*
row = 0

Main – part 2:

Open Properties box:
Display name: *Loop forever*
Loop while: *1*

Open Properties box:
Display name: *Check USB for incoming data*
Component: *USBSerial(0)*
Macro: *ReadByte*
Parameter: *0*
Return Value *retval*

Open Properties box:
Display name: *If data available*
If: *retval< 255*

Open Properties box:
Display name: *Are we at the first character on a row*
If: *column > 0*

Open Properties box:
Display name: *Check for backspace*
If: *retval= 8*

Open Properties box:
Display name: *Are we on Row 1*
If: *row*

Open Properties box:
Display name: *Move back a column*
Calculations: *column = column - 1*

Open Properties box:
Display name: *Reset back to Row 0*
Calculations: *row = 0*

Open Properties box:
Display name: *Reset to end of column*
Calculations: *column = 15*

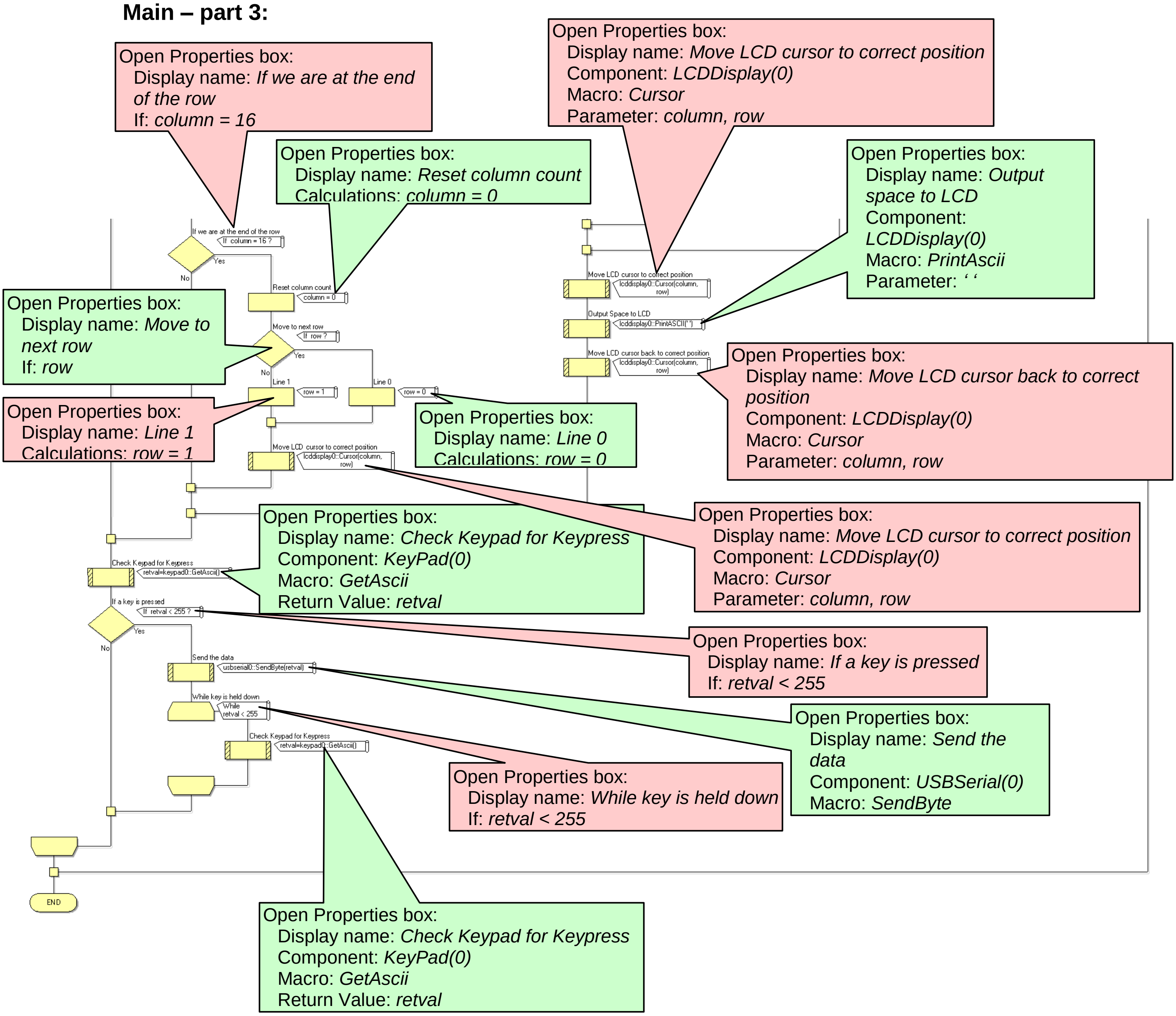
Open Properties box:
Display name: *Increment column count*
Calculations: *column = column + 1*

Open Properties box:
Display name: *Clear LCD*
Component:
LCDDisplay(0)

Open Properties box:
Display name: *If we are starting again on the top line*
If: *(row = 0) && (column = 0)*

Open Properties box:
Display name: *Output data byte to LCD*
Component:
LCDDisplay(0)
Macro: *PrintAscii*
Parameter: *retval*

Main – part 3:


If we are at the end of the row
If column = 16 ?
Yes
No
Reset column count
column = 0
Move to next row
If row ?
Yes
No
Line 1
row = 1
Line 0
row = 0
Move LCD cursor to correct position
lcddisplay0::Cursor(column, row)
Check Keypad for Keypress
retval=keypad0::GetAscii()
If a key is pressed
If retval < 255 ?
Yes
No
Send the data
usbserial0::SendByte(retval)
While key is held down
While retval < 255
Check Keypad for Keypress
retval=keypad0::GetAscii()
Move LCD cursor to correct position
lcddisplay0::Cursor(column, row)
Output Space to LCD
lcddisplay0::PrintASCII(' ')
Move LCD cursor back to correct position
lcddisplay0::Cursor(column, row)
END

Open Properties box:
Display name: *Move LCD cursor to correct position*
Component: *LCDDisplay(0)*
Macro: *Cursor*
Parameter: *column, row*

Open Properties box:
Display name: *If we are at the end of the row*
If: *column = 16*

Open Properties box:
Display name: *Reset column count*
Calculations: *column = 0*

Open Properties box:
Display name: *Output space to LCD*
Component: *LCDDisplay(0)*
Macro: *PrintAscii*
Parameter: *‘ ‘*

Open Properties box:
Display name: *Move to next row*
If: *row*

Open Properties box:
Display name: *Move LCD cursor back to correct position*
Component: *LCDDisplay(0)*
Macro: *Cursor*
Parameter: *column, row*

Open Properties box:
Display name: *Line 1*
Calculations: *row = 1*

Open Properties box:
Display name: *Line 0*
Calculations: *row = 0*

Open Properties box:
Display name: *Check Keypad for Keypress*
Component: *KeyPad(0)*
Macro: *GetAscii*
Return Value: *retval*

Open Properties box:
Display name: *Move LCD cursor to correct position*
Component: *LCDDisplay(0)*
Macro: *Cursor*
Parameter: *column, row*

Open Properties box:
Display name: *If a key is pressed*
If: *retval < 255*

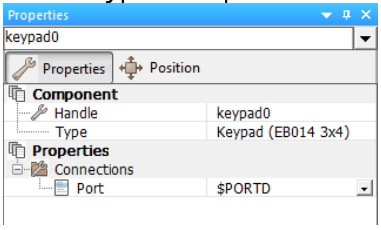
Open Properties box:
Display name: *Send the data*
Component: *USBSerial(0)*
Macro: *SendByte*

Open Properties box:
Display name: *While key is held down*
If: *retval < 255*

Open Properties box:
Display name: *Check Keypad for Keypress*
Component: *KeyPad(0)*
Macro: *GetAscii*
Return Value: *retval*

Exercise 5:

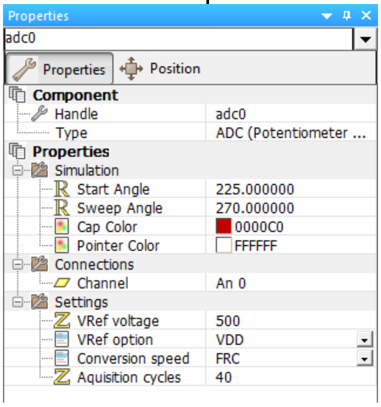
Keypad Properties



keypad0

Component	
Handle	keypad0
Type	Keypad (EB014 3x4)
Properties	
Connections	
Port	$PORTD

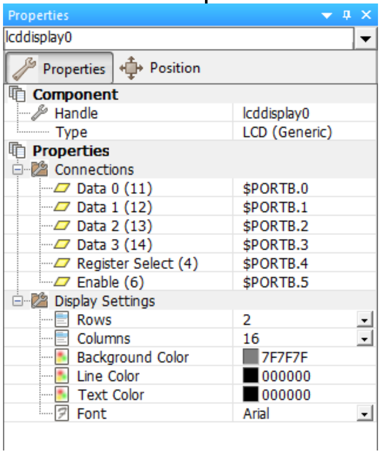
ADC Properties



adc0

Component	
Handle	adc0
Type	ADC (Potentiometer ...
Properties	
Simulation	
Start Angle	225.000000
Sweep Angle	270.000000
Cap Color	🟥 0000C0
Pointer Color	□ FFFFFF
Connections	
Channel	An 0
Settings	
VRef voltage	500
VRef option	VDD
Conversion speed	FRC
Aquisition cycles	40

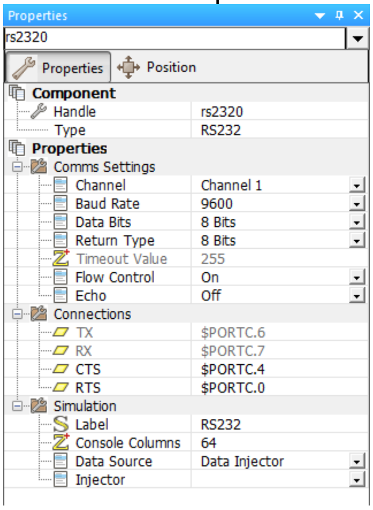
LCD Properties



lcddisplay0

Component	
Handle	lcddisplay0
Type	LCD (Generic)
Properties	
Connections	
Data 0 (11)	$PORTB.0
Data 1 (12)	$PORTB.1
Data 2 (13)	$PORTB.2
Data 3 (14)	$PORTB.3
Register Select (4)	$PORTB.4
Enable (6)	$PORTB.5
Display Settings	
Rows	2
Columns	16
Background Color	▨ 7F7F7F
Line Color	■ 000000
Text Color	■ 000000
Font	Arial

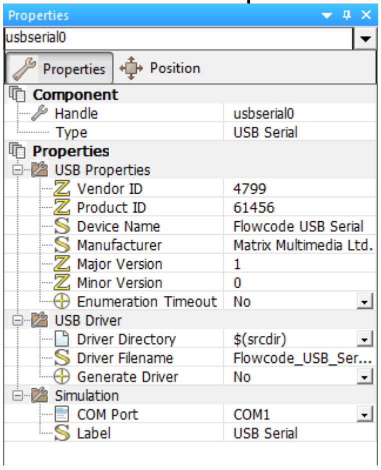
RS232 Properties



rs2320

Component	
Handle	rs2320
Type	RS232
Properties	
Comms Settings	
Channel	Channel 1
Baud Rate	9600
Data Bits	8 Bits
Return Type	8 Bits
Timeout Value	255
Flow Control	On
Echo	Off
Connections	
TX	$PORTC.6
RX	$PORTC.7
CTS	$PORTC.4
RTS	$PORTC.0
Simulation	
Label	RS232
Console Columns	64
Data Source	Data Injector
Injector	

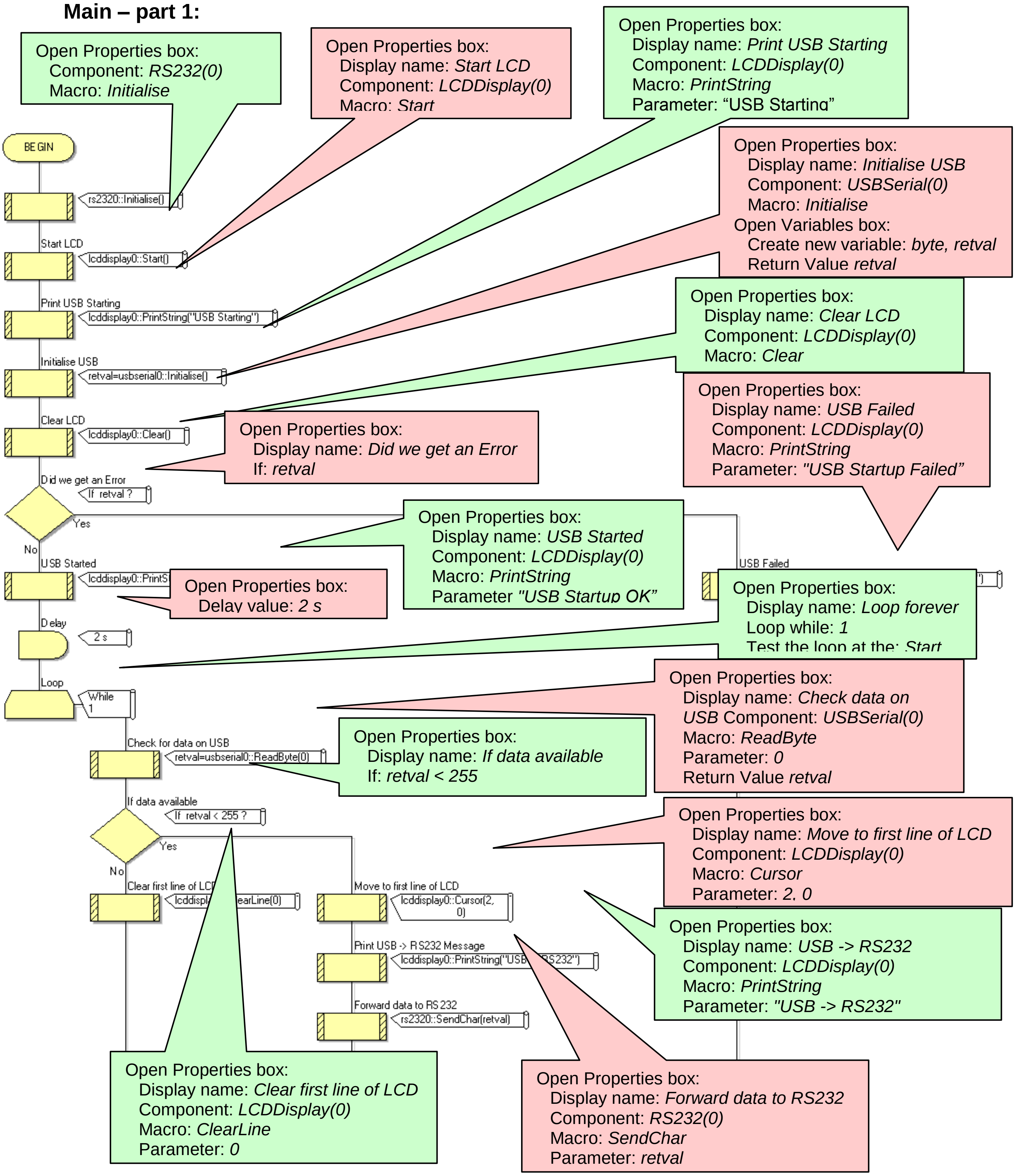
USB Serial Properties



usbserial0

Component	
Handle	usbserial0
Type	USB Serial
Properties	
USB Properties	
Vendor ID	4799
Product ID	61456
Device Name	Flowcode USB Serial
Manufacturer	Matrix Multimedia Ltd.
Major Version	1
Minor Version	0
Enumeration Timeout	No
USB Driver	
Driver Directory	$(srcdir)
Driver Filename	Flowcode_USB_Ser...
Generate Driver	No
Simulation	
COM Port	COM1
Label	USB Serial

Main – part 1:



Open Properties box:
Component: *RS232(0)*
Macro: *Initialise*

Open Properties box:
Display name: *Start LCD*
Component: *LCDDisplay(0)*
Macro: *Start*

Open Properties box:
Display name: *Print USB Starting*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: “USB Starting”

Open Properties box:
Display name: *Initialise USB*
Component: *USBSerial(0)*
Macro: *Initialise*
Open Variables box:
Create new variable: *byte, retval*
Return Value *retval*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Did we get an Error*
If: *retval*

Open Properties box:
Display name: *USB Failed*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB Startup Failed”*

Open Properties box:
Display name: *USB Started*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Startup OK”*

Open Properties box:
Delay value: *2 s*

Open Properties box:
Display name: *Loop forever*
Loop while: *1*
Test the loop at the: *Start*

Open Properties box:
Display name: *Check data on USB* Component: *USBSerial(0)*
Macro: *ReadByte*
Parameter: *0*
Return Value *retval*

Open Properties box:
Display name: *If data available*
If: *retval < 255*

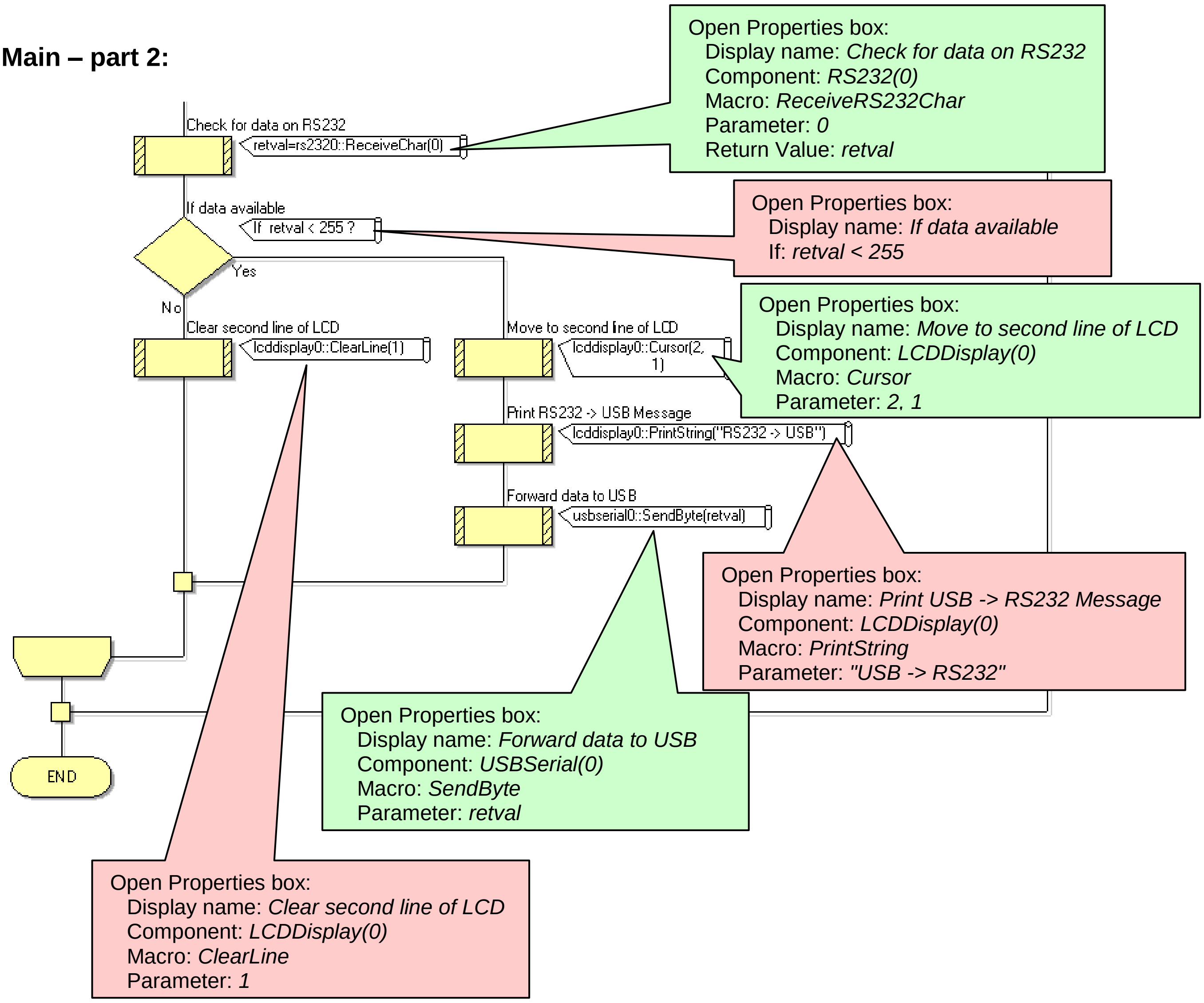
Open Properties box:
Display name: *Move to first line of LCD*
Component: *LCDDisplay(0)*
Macro: *Cursor*
Parameter: *2, 0*

Open Properties box:
Display name: *USB -> RS232*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB -> RS232"*

Open Properties box:
Display name: *Clear first line of LCD*
Component: *LCDDisplay(0)*
Macro: *ClearLine*
Parameter: *0*

Open Properties box:
Display name: *Forward data to RS232*
Component: *RS232(0)*
Macro: *SendChar*
Parameter: *retval*

Main – part 2:


Check for data on RS232
retval=rs2320::ReceiveChar(0)
If data available
If retval < 255 ?
Yes
No
Clear second line of LCD
lcddisplay0::ClearLine(1)
Move to second line of LCD
lcddisplay0::Cursor(2, 1)
Print RS232 -> USB Message
lcddisplay0::PrintString("RS232 -> USB")
Forward data to USB
usbserial0::SendByte(retval)
END

Open Properties box:
Display name: *Check for data on RS232*
Component: *RS232(0)*
Macro: *ReceiveRS232Char*
Parameter: *0*
Return Value: *retval*

Open Properties box:
Display name: *If data available*
If: *retval < 255*

Open Properties box:
Display name: *Move to second line of LCD*
Component: *LCDDisplay(0)*
Macro: *Cursor*
Parameter: *2, 1*

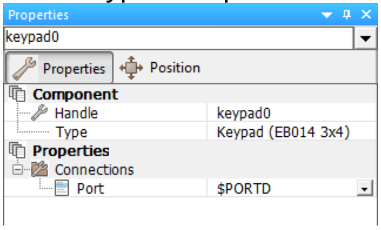
Open Properties box:
Display name: *Print USB -> RS232 Message*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB -> RS232"*

Open Properties box:
Display name: *Forward data to USB*
Component: *USBSerial(0)*
Macro: *SendByte*
Parameter: *retval*

Open Properties box:
Display name: *Clear second line of LCD*
Component: *LCDDisplay(0)*
Macro: *ClearLine*
Parameter: *1*

Exercise 6:

Keypad Properties

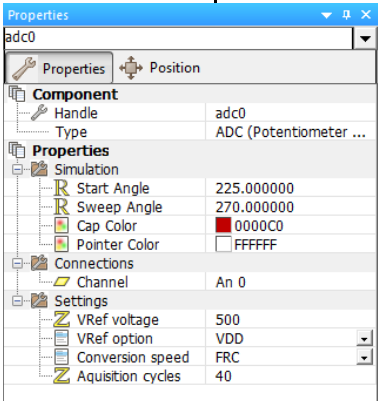


Properties

keypad0

Component	
Handle	keypad0
Type	Keypad (EB014 3x4)
Properties	
Connections	
Port	$PORTD

ADC0 Properties

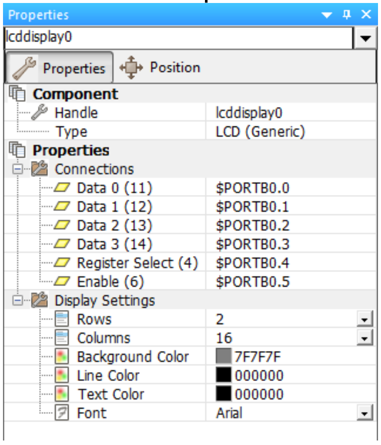


Properties

adc0

Component	
Handle	adc0
Type	ADC (Potentiometer ...
Properties	
Simulation	
Start Angle	225.000000
Sweep Angle	270.000000
Cap Color	🟥 0000C0
Pointer Color	□ FFFFFF
Connections	
Channel	An 0
Settings	
VRef voltage	500
VRef option	VDD
Conversion speed	FRC
Aquisition cycles	40

LCD Properties

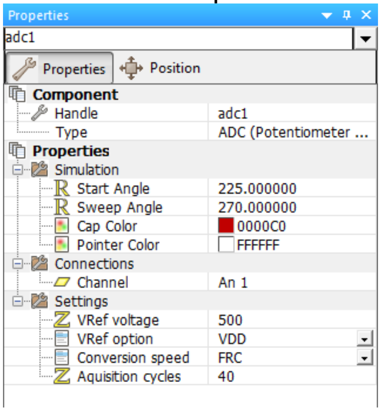


Properties

lcddisplay0

Component	
Handle	lcddisplay0
Type	LCD (Generic)
Properties	
Connections	
Data 0 (11)	$PORTB0.0
Data 1 (12)	$PORTB0.1
Data 2 (13)	$PORTB0.2
Data 3 (14)	$PORTB0.3
Register Select (4)	$PORTB0.4
Enable (6)	$PORTB0.5
Display Settings	
Rows	2
Columns	16
Background Color	▨ 7F7F7F
Line Color	■ 000000
Text Color	■ 000000
Font	Arial

ADC1 Properties

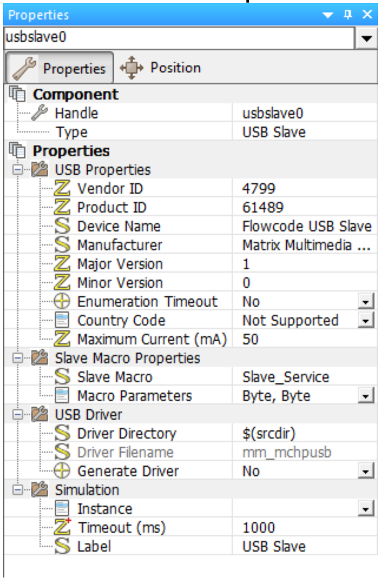


Properties

adc1

Component	
Handle	adc1
Type	ADC (Potentiometer ...
Properties	
Simulation	
Start Angle	225.000000
Sweep Angle	270.000000
Cap Color	🟥 0000C0
Pointer Color	□ FFFFFF
Connections	
Channel	An 1
Settings	
VRef voltage	500
VRef option	VDD
Conversion speed	FRC
Aquisition cycles	40

USB Slave Properties

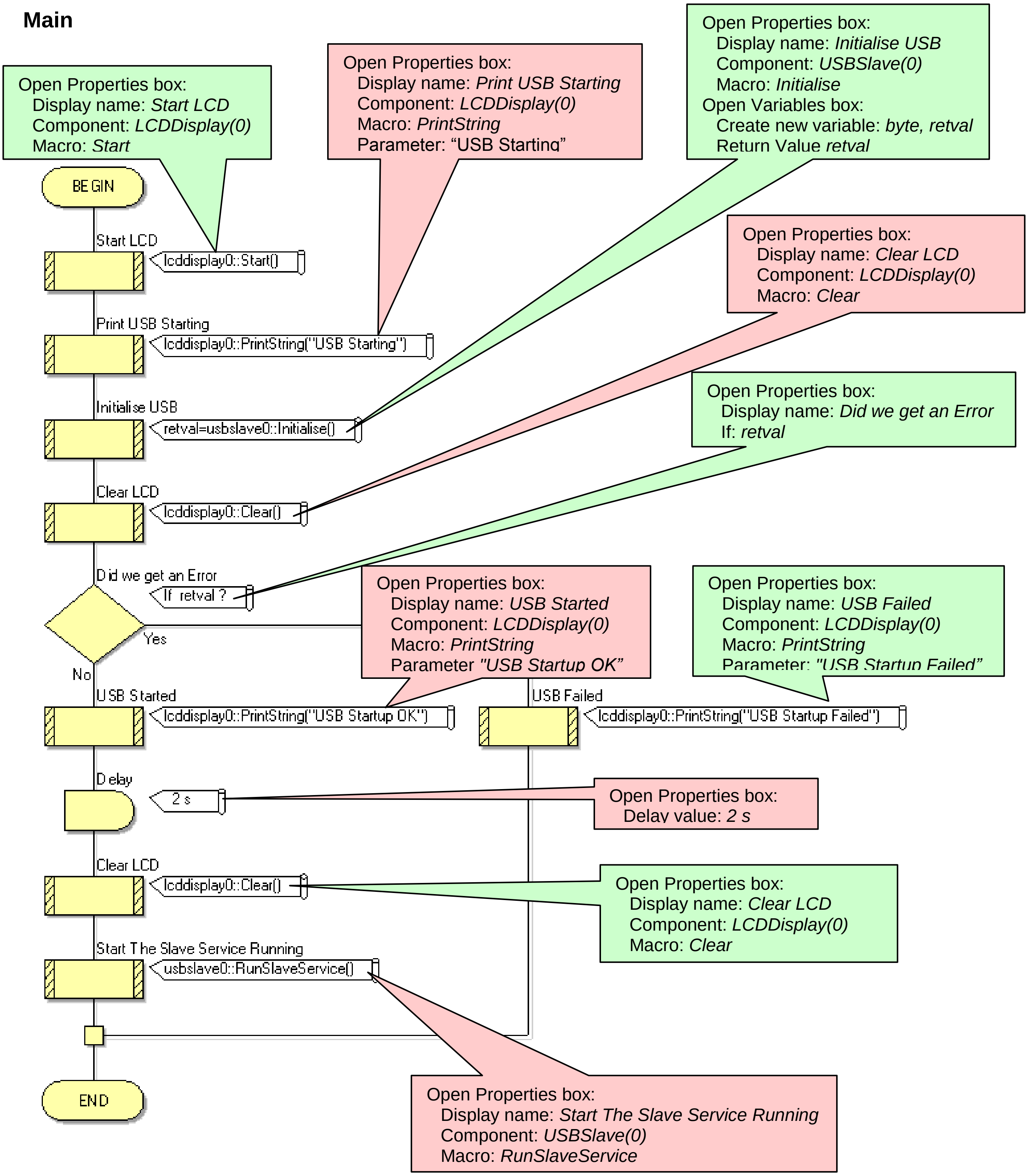


Properties

usbslave0

Component	
Handle	usbslave0
Type	USB Slave
Properties	
USB Properties	
Vendor ID	4799
Product ID	61489
Device Name	Flowcode USB Slave
Manufacturer	Matrix Multimedia ...
Major Version	1
Minor Version	0
Enumeration Timeout	No
Country Code	Not Supported
Maximum Current (mA)	50
Slave Macro Properties	
Slave Macro	Slave_Service
Macro Parameters	Byte, Byte
USB Driver	
Driver Directory	$(srcdir)
Driver Filename	mm_mchpusb
Generate Driver	No
Simulation	
Instance	
Timeout (ms)	1000
Label	USB Slave

Main


BEGIN
Start LCD
lcddisplay0::Start()
Print USB Starting
lcddisplay0::PrintString("USB Starting")
Initialise USB
retval=usbslave0::Initialise()
Clear LCD
lcddisplay0::Clear()
Did we get an Error
If retval ?
Yes
No
USB Started
lcddisplay0::PrintString("USB Startup OK")
USB Failed
lcddisplay0::PrintString("USB Startup Failed")
Delay
2 s
Clear LCD
lcddisplay0::Clear()
Start The Slave Service Running
usbslave0::RunSlaveService()
END

Open Properties box:
Display name: *Start LCD*
Component: *LCDDisplay(0)*
Macro: *Start*

Open Properties box:
Display name: *Print USB Starting*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: “USB Starting”

Open Properties box:
Display name: *Initialise USB*
Component: *USBSlave(0)*
Macro: *Initialise*
Open Variables box:
Create new variable: *byte, retval*
Return Value *retval*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Did we get an Error*
If: *retval*

Open Properties box:
Display name: *USB Started*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter *"USB Startup OK”*

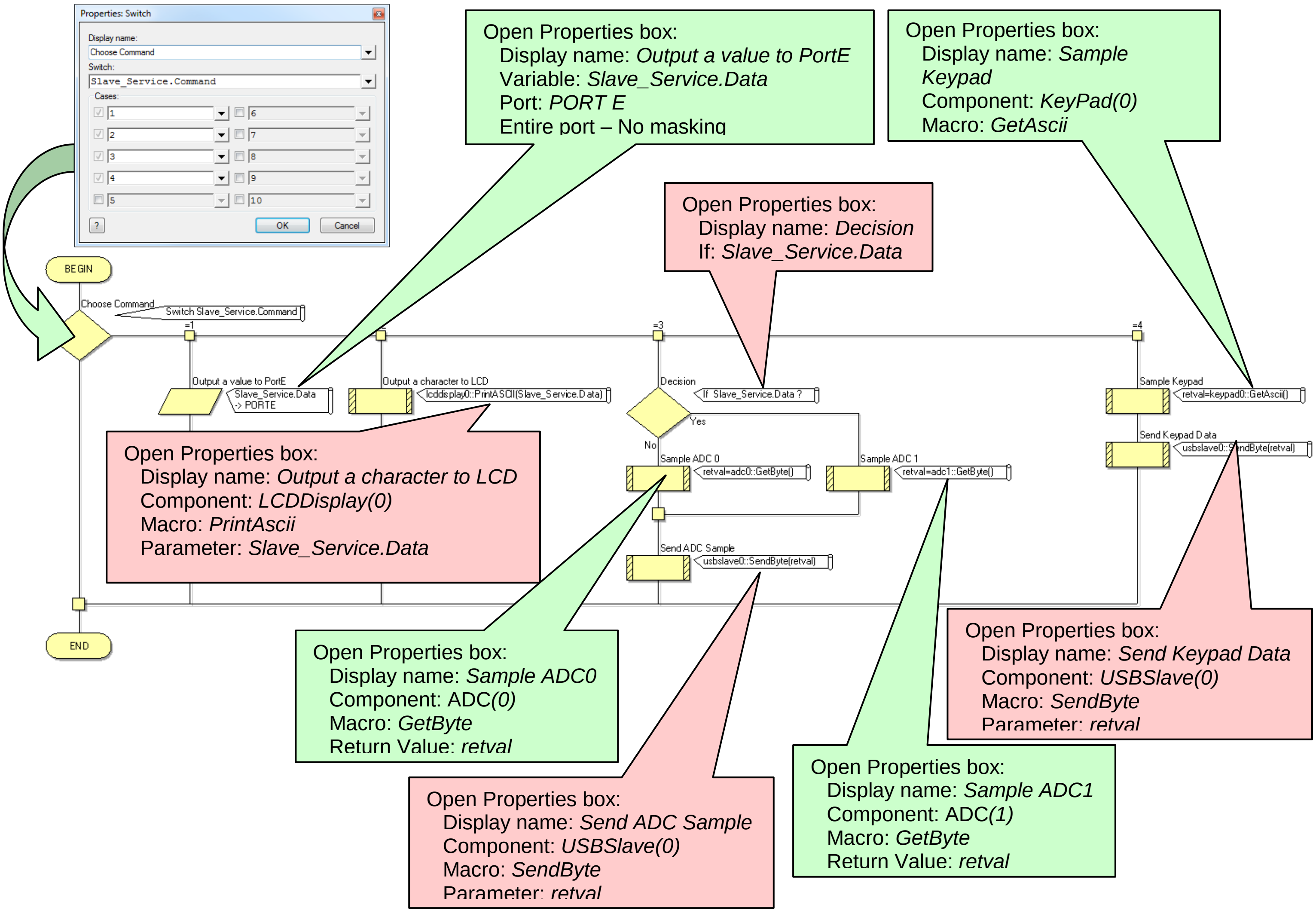
Open Properties box:
Display name: *USB Failed*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB Startup Failed”*

Open Properties box:
Delay value: *2 s*

Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Start The Slave Service Running*
Component: *USBSlave(0)*
Macro: *RunSlaveService*

Slave_Service macro



Properties: Switch

Display name: Choose Command

Switch: Slave_Service.Command

Cases:

☑ 1 ☐ 6
☑ 2 ☐ 7
☑ 3 ☐ 8
☑ 4 ☐ 9
☐ 5 ☐ 10

Open Properties box:
Display name: *Output a value to PortE*
Variable: *Slave_Service.Data*
Port: *PORT E*
Entire port – No masking

Open Properties box:
Display name: *Sample Keypad*
Component: *KeyPad(0)*
Macro: *GetAscii*

Open Properties box:
Display name: *Decision*
If: *Slave_Service.Data*

Open Properties box:
Display name: *Output a character to LCD*
Component: *LCDDisplay(0)*
Macro: *PrintAscii*
Parameter: *Slave_Service.Data*

Open Properties box:
Display name: *Send Keypad Data*
Component: *USBSlave(0)*
Macro: *SendByte*
Parameter: *retval*

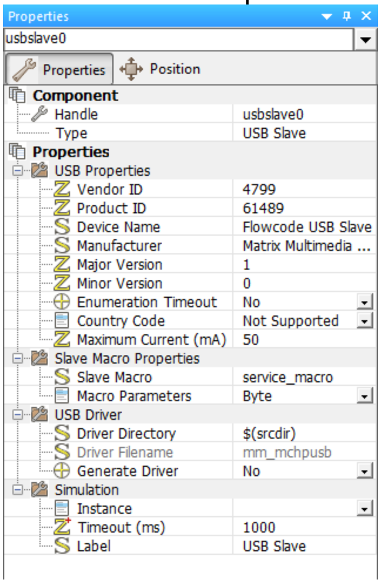
Open Properties box:
Display name: *Sample ADC0*
Component: ADC*(0)*
Macro: *GetByte*
Return Value: *retval*

Open Properties box:
Display name: *Sample ADC1*
Component: ADC*(1)*
Macro: *GetByte*
Return Value: *retval*

Open Properties box:
Display name: *Send ADC Sample*
Component: *USBSlave(0)*
Macro: *SendByte*
Parameter: *retval*

Exercise 7:

USB Slave Properties



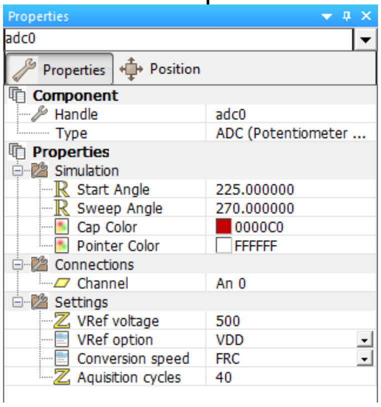
Properties

usbslave0

Properties | Position

Component	
Handle	usbslave0
Type	USB Slave
Properties	
USB Properties	
Vendor ID	4799
Product ID	61489
Device Name	Flowcode USB Slave
Manufacturer	Matrix Multimedia ...
Major Version	1
Minor Version	0
Enumeration Timeout	No
Country Code	Not Supported
Maximum Current (mA)	50
Slave Macro Properties	
Slave Macro	service_macro
Macro Parameters	Byte
USB Driver	
Driver Directory	$(srcdir)
Driver Filename	mm_mchpusb
Generate Driver	No
Simulation	
Instance	
Timeout (ms)	1000
Label	USB Slave

ADC Properties



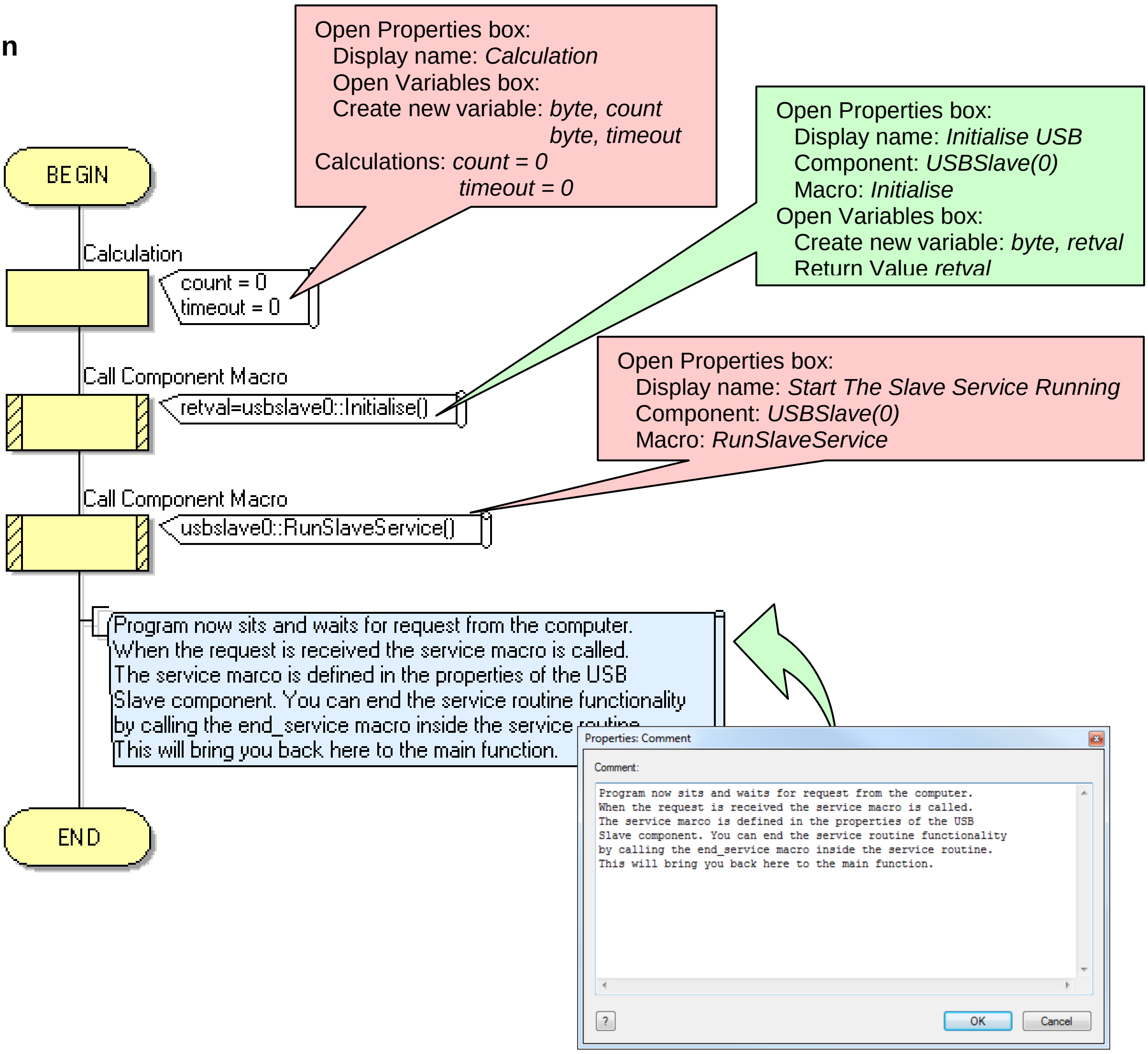
Properties

adc0

Properties | Position

Component	
Handle	adc0
Type	ADC (Potentiometer ...
Properties	
Simulation	
Start Angle	225.000000
Sweep Angle	270.000000
Cap Color	🟥 0000C0
Pointer Color	□ FFFFFF
Connections	
Channel	An 0
Settings	
VRef voltage	500
VRef option	VDD
Conversion speed	FRC
Aquisition cycles	40

Main



Open Properties box:
  Display name: *Calculation*
  Open Variables box:
  Create new variable: *byte, count*
  byte, timeout
Calculations: *count = 0*
  timeout = 0

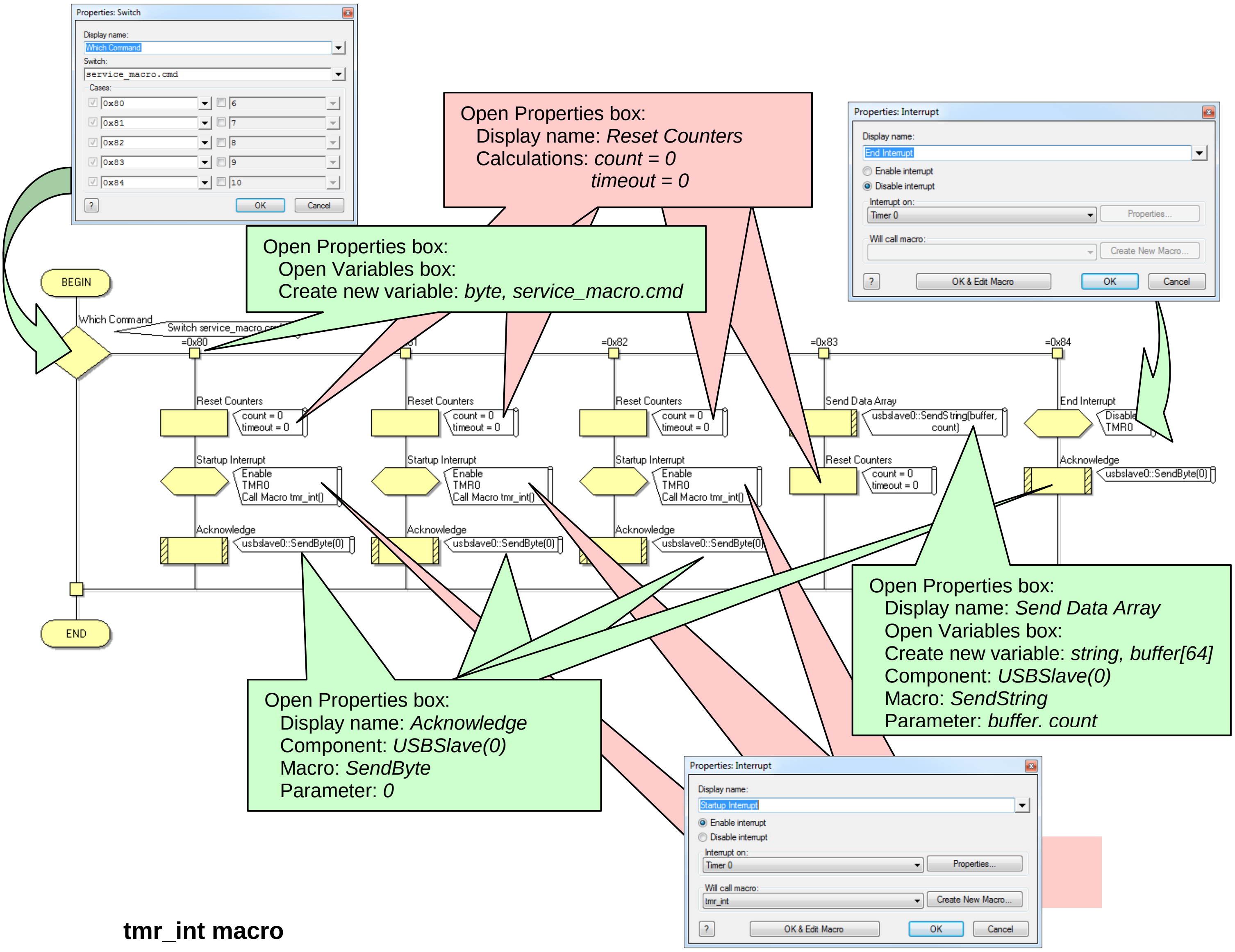
Open Properties box:
  Display name: *Initialise USB*
  Component: *USBSlave(0)*
  Macro: *Initialise*
Open Variables box:
  Create new variable: *byte, retval*
  Return Value *retval*

Open Properties box:
  Display name: *Start The Slave Service Running*
  Component: *USBSlave(0)*
  Macro: *RunSlaveService*

Program now sits and waits for request from the computer.
When the request is received the service macro is called.
The service marco is defined in the properties of the USB
Slave component. You can end the service routine functionality
by calling the end_service macro inside the service routine.
This will bring you back here to the main function.

```
Program now sits and waits for request from the computer.
When the request is received the service macro is called.
The service marco is defined in the properties of the USB
Slave component. You can end the service routine functionality
by calling the end_service macro inside the service routine.
This will bring you back here to the main function.
```

Service macro



Properties: Switch

Display name: Which Command

Switch: service_macro.cmd

Cases:
☑ 0x80 □ 6
☑ 0x81 □ 7
☑ 0x82 □ 8
☑ 0x83 □ 9
☑ 0x84 □ 10

? OK Cancel

Open Properties box:
Open Variables box:
Create new variable: *byte, service_macro.cmd*

Open Properties box:
Display name: *Reset Counters*
Calculations: *count = 0*
timeout = 0

Properties: Interrupt

Display name: End Interrupt

○ Enable interrupt
● Disable interrupt

Interrupt on: Timer 0 — Properties...

Will call macro: — Create New Macro...

? OK & Edit Macro OK Cancel

Open Properties box:
Display name: *Send Data Array*
Open Variables box:
Create new variable: *string, buffer[64]*
Component: *USBSlave(0)*
Macro: *SendString*
Parameter: *buffer. count*

Open Properties box:
Display name: *Acknowledge*
Component: *USBSlave(0)*
Macro: *SendByte*
Parameter: *0*

Properties: Interrupt

Display name: Startup Interrupt

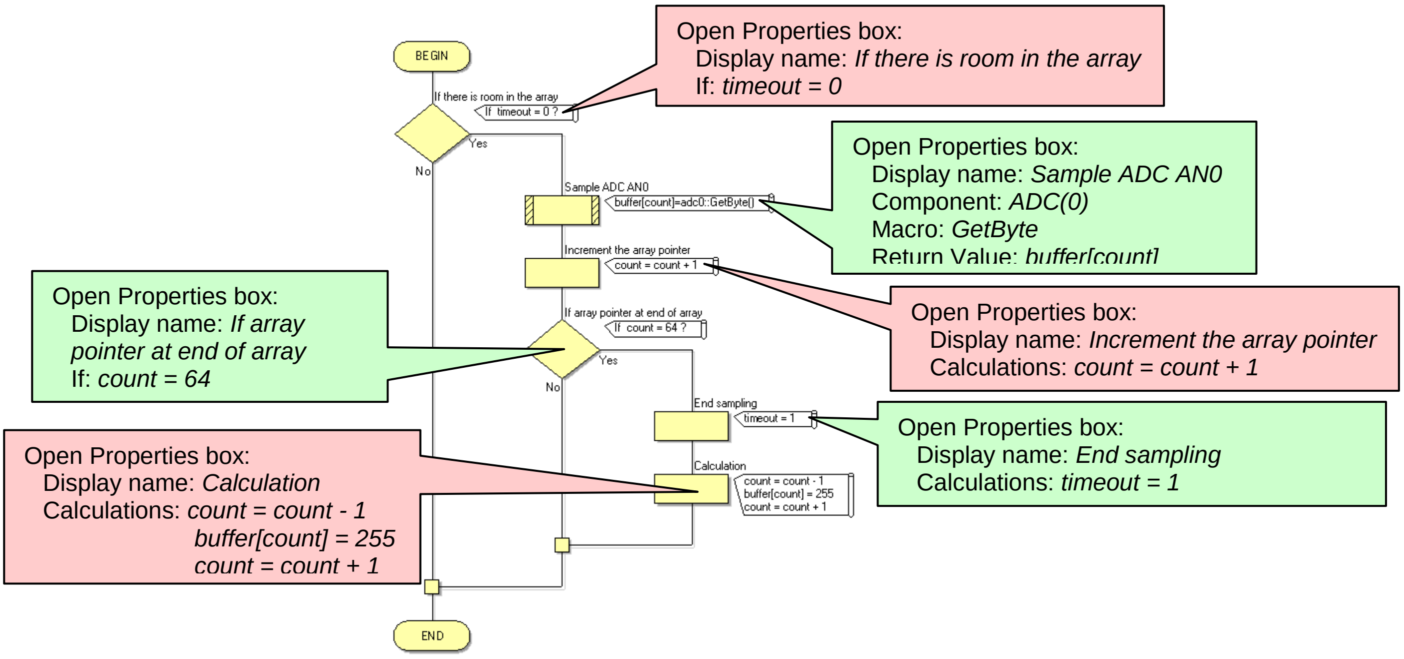
● Enable interrupt
○ Disable interrupt

Interrupt on: Timer 0 — Properties...

Will call macro: tmr_int — Create New Macro...

? OK & Edit Macro OK Cancel

tmr_int macro



Open Properties box:
Display name: *If there is room in the array*
If: *timeout = 0*

Open Properties box:
Display name: *Sample ADC AN0*
Component: *ADC(0)*
Macro: *GetByte*
Return Value: *buffer[count]*

Open Properties box:
Display name: *If array pointer at end of array*
If: *count = 64*

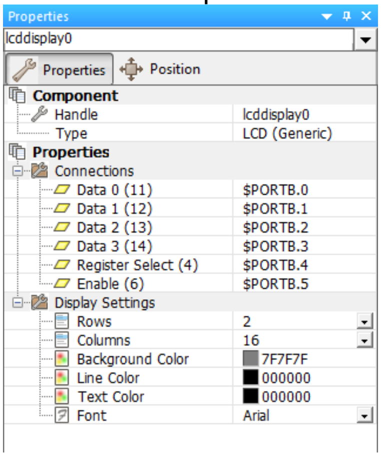
Open Properties box:
Display name: *Increment the array pointer*
Calculations: *count = count + 1*

Open Properties box:
Display name: *End sampling*
Calculations: *timeout = 1*

Open Properties box:
Display name: *Calculation*
Calculations: *count = count - 1*
buffer[count] = 255
count = count + 1

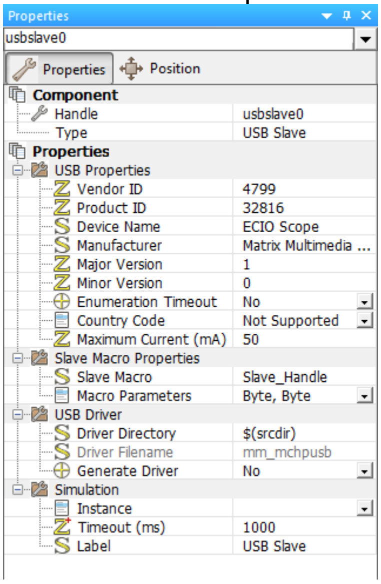
Exercise 8:

LCD Properties



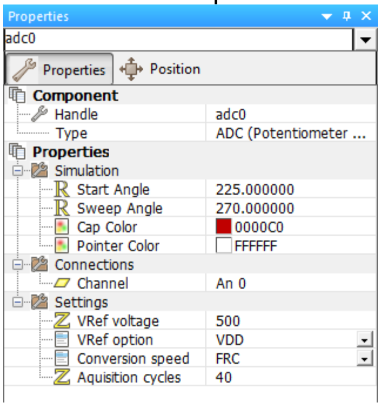
Properties	
lcddisplay0	
Properties	Position
Component	
Handle	lcddisplay0
Type	LCD (Generic)
Properties	
Connections	
Data 0 (11)	$PORTB.0
Data 1 (12)	$PORTB.1
Data 2 (13)	$PORTB.2
Data 3 (14)	$PORTB.3
Register Select (4)	$PORTB.4
Enable (6)	$PORTB.5
Display Settings	
Rows	2
Columns	16
Background Color	▨ 7F7F7F
Line Color	■ 000000
Text Color	■ 000000
Font	Arial

USB Slave Properties



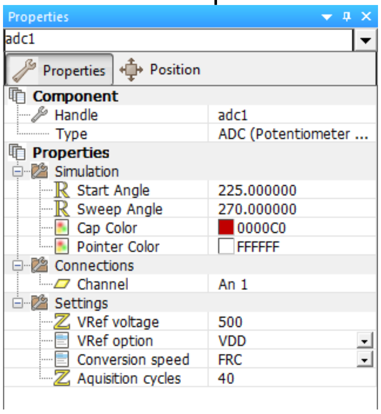
Properties	
usbslave0	
Properties	Position
Component	
Handle	usbslave0
Type	USB Slave
Properties	
USB Properties	
Vendor ID	4799
Product ID	32816
Device Name	ECIO Scope
Manufacturer	Matrix Multimedia ...
Major Version	1
Minor Version	0
Enumeration Timeout	No
Country Code	Not Supported
Maximum Current (mA)	50
Slave Macro Properties	
Slave Macro	Slave_Handle
Macro Parameters	Byte, Byte
USB Driver	
Driver Directory	$(srcdir)
Driver Filename	mm_mchpusb
Generate Driver	No
Simulation	
Instance	
Timeout (ms)	1000
Label	USB Slave

ADC0 Properties



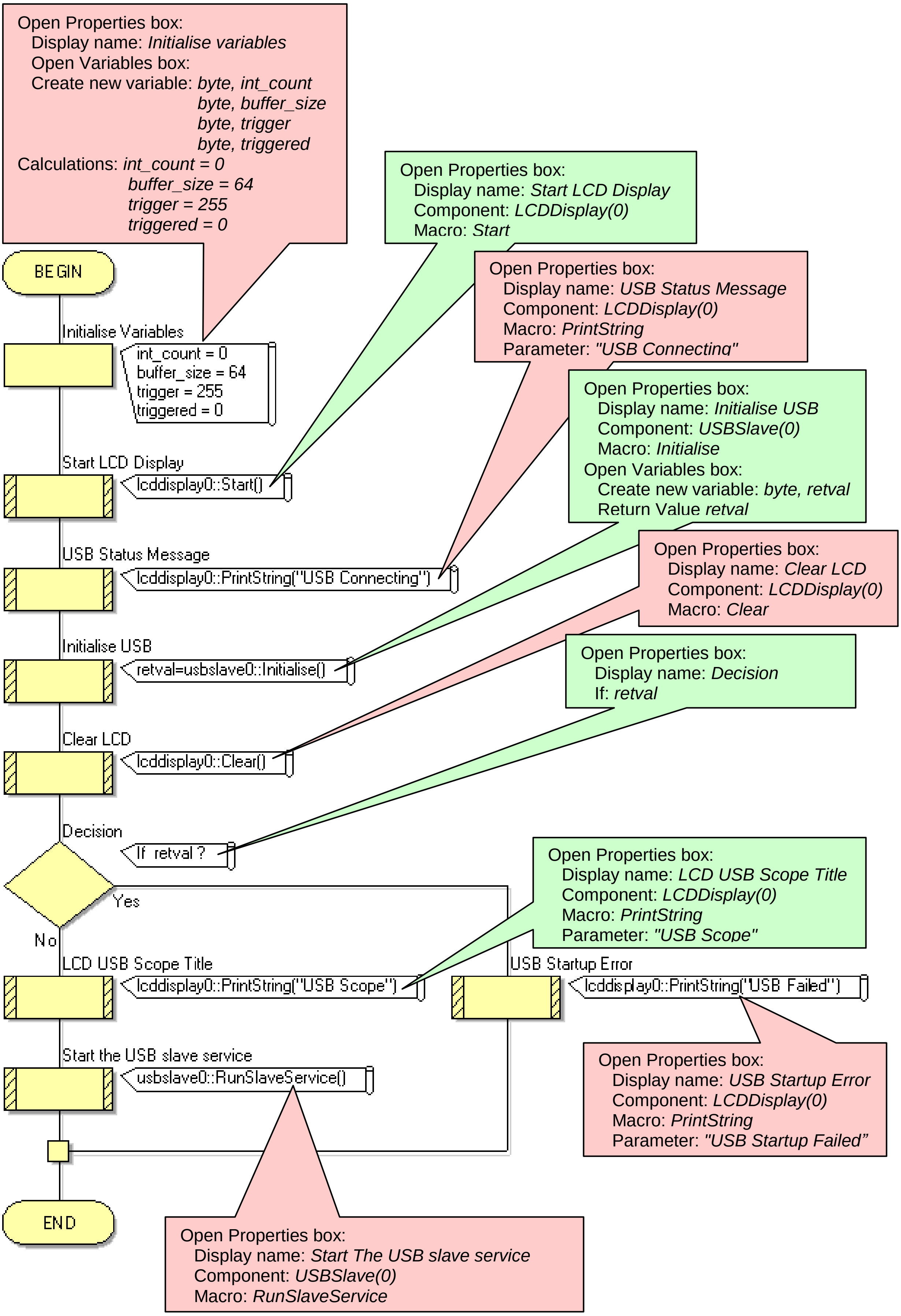
Properties	
adc0	
Properties	Position
Component	
Handle	adc0
Type	ADC (Potentiometer ...
Properties	
Simulation	
Start Angle	225.000000
Sweep Angle	270.000000
Cap Color	🟥 0000C0
Pointer Color	□ FFFFFF
Connections	
Channel	An 0
Settings	
VRef voltage	500
VRef option	VDD
Conversion speed	FRC
Aquisition cycles	40

ADC1 Properties



Properties	
adc1	
Properties	Position
Component	
Handle	adc1
Type	ADC (Potentiometer ...
Properties	
Simulation	
Start Angle	225.000000
Sweep Angle	270.000000
Cap Color	🟥 0000C0
Pointer Color	□ FFFFFF
Connections	
Channel	An 1
Settings	
VRef voltage	500
VRef option	VDD
Conversion speed	FRC
Aquisition cycles	40

Main



Open Properties box:
Display name: *Initialise variables*
Open Variables box:
Create new variable: *byte, int_count*
byte, buffer_size
byte, trigger
byte, triggered
Calculations: *int_count = 0*
buffer_size = 64
trigger = 255
triggered = 0

Open Properties box:
Display name: *Start LCD Display*
Component: *LCDDisplay(0)*
Macro: *Start*

Open Properties box:
Display name: *USB Status Message*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB Connecting"*

Open Properties box:
Display name: *Initialise USB*
Component: *USBSlave(0)*
Macro: *Initialise*
Open Variables box:
Create new variable: *byte, retval*
Return Value *retval*

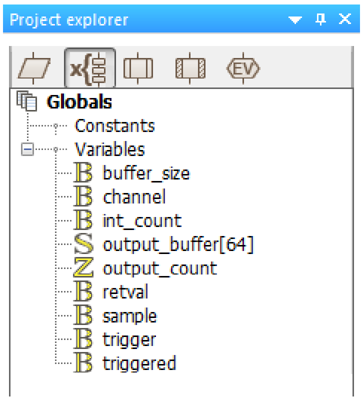
Open Properties box:
Display name: *Clear LCD*
Component: *LCDDisplay(0)*
Macro: *Clear*

Open Properties box:
Display name: *Decision*
If: *retval*

Open Properties box:
Display name: *LCD USB Scope Title*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB Scope"*

Open Properties box:
Display name: *USB Startup Error*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"USB Startup Failed"*

Open Properties box:
Display name: *Start The USB slave service*
Component: *USBSlave(0)*
Macro: *RunSlaveService*



Project explorer

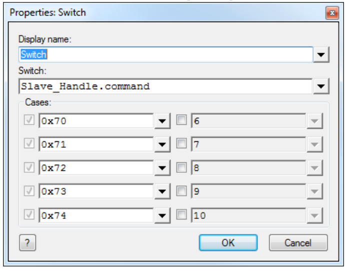
- Globals
  - Constants
  - Variables
    - buffer_size
    - channel
    - int_count
    - output_buffer[64]
    - output_count
    - retval
    - sample
    - trigger
    - triggered

Slave_Handle macro

Create the additional variables, shown in the Variable Manager.

The following are used within the macro:

Switch-case properties



Properties: Switch

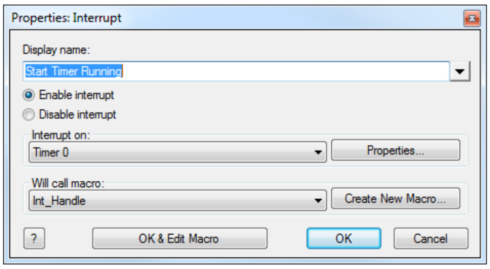
Display name: Switch

Switch: Slave_Handle.command

Cases:

☑	0x70	□	6
☑	0x71	□	7
☑	0x72	□	8
☑	0x73	□	9
☑	0x74	□	10

Enable TMR0 properties



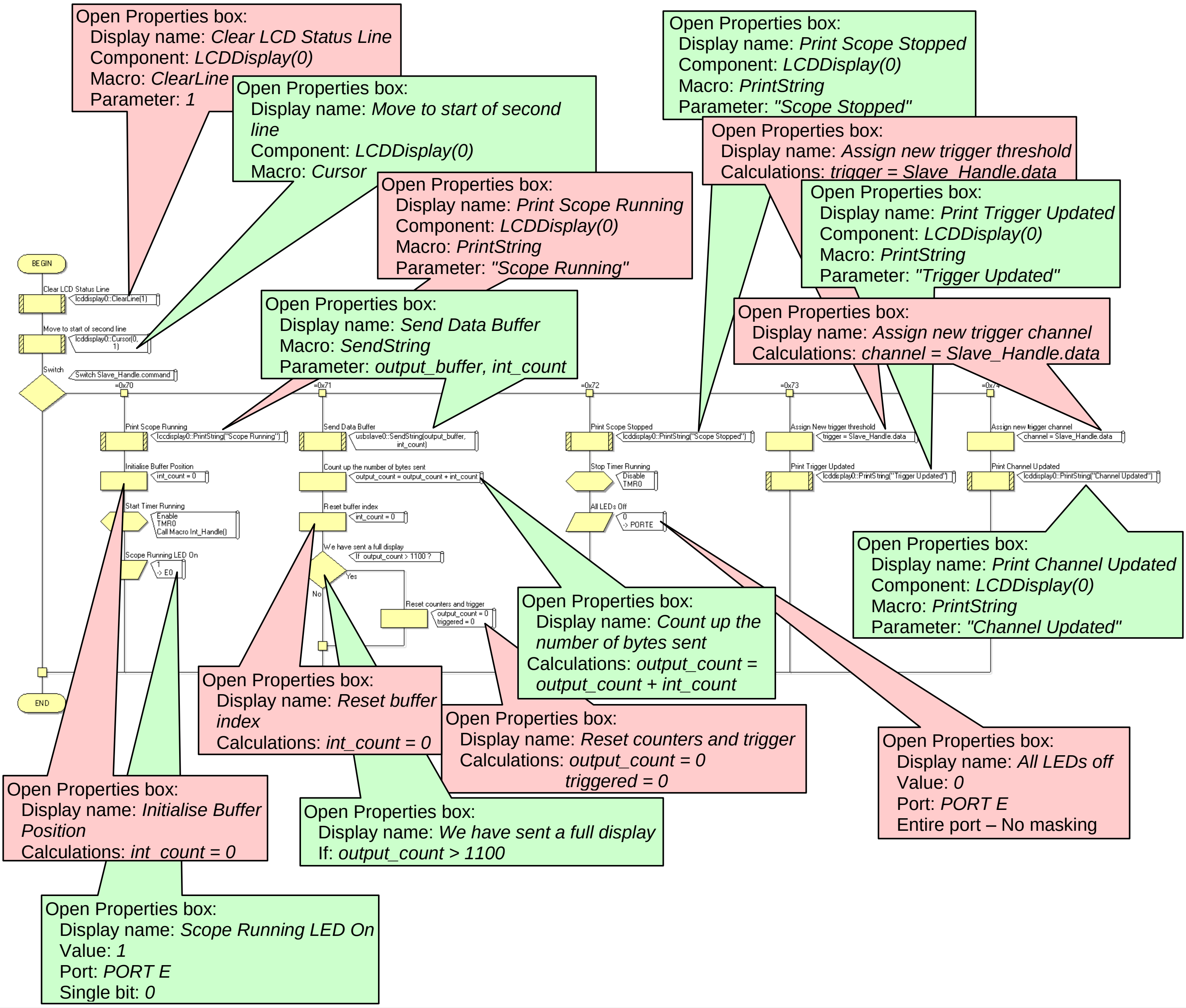
Properties: Interrupt

Display name: Start Timer Running

● Enable interrupt
○ Disable interrupt

Interrupt on: Timer 0

Will call macro: Int_Handle



BEGIN
Clear LCD Status Line
lcddisplay0::ClearLine(1)
Move to start of second line
lcddisplay0::Cursor(0, 1)
Switch
Switch Slave_Handle.command
=0x70
Print Scope Running
lcddisplay0::PrintString("Scope Running")
Initialise Buffer Position
int_count = 0
Start Timer Running
Enable TMR0 Call Macro Int_Handle()
Scope Running LED On
1 -> E0
=0x71
Send Data Buffer
usbslave0::SendString(output_buffer, int_count)
Count up the number of bytes sent
output_count = output_count + int_count
Reset buffer index
int_count = 0
We have sent a full display
If output_count > 1100 ?
Yes
No
Reset counters and trigger
output_count = 0 triggered = 0
=0x72
Print Scope Stopped
lcddisplay0::PrintString("Scope Stopped")
Stop Timer Running
Disable TMR0
All LEDs Off
0 -> PORTE
=0x73
Assign New trigger threshold
trigger = Slave_Handle.data
Print Trigger Updated
lcddisplay0::PrintString("Trigger Updated")
=0x74
Assign new trigger channel
channel = Slave_Handle.data
Print Channel Updated
lcddisplay0::PrintString("Channel Updated")
END

Open Properties box:
Display name: *Clear LCD Status Line*
Component: *LCDDisplay(0)*
Macro: *ClearLine*
Parameter: *1*

Open Properties box:
Display name: *Move to start of second line*
Component: *LCDDisplay(0)*
Macro: *Cursor*

Open Properties box:
Display name: *Print Scope Running*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"Scope Running"*

Open Properties box:
Display name: *Send Data Buffer*
Macro: *SendString*
Parameter: *output_buffer, int_count*

Open Properties box:
Display name: *Print Scope Stopped*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"Scope Stopped"*

Open Properties box:
Display name: *Assign new trigger threshold*
Calculations: *trigger = Slave_Handle.data*

Open Properties box:
Display name: *Print Trigger Updated*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"Trigger Updated"*

Open Properties box:
Display name: *Assign new trigger channel*
Calculations: *channel = Slave_Handle.data*

Open Properties box:
Display name: *Print Channel Updated*
Component: *LCDDisplay(0)*
Macro: *PrintString*
Parameter: *"Channel Updated"*

Open Properties box:
Display name: *Count up the number of bytes sent*
Calculations: *output_count = output_count + int_count*

Open Properties box:
Display name: *Reset buffer index*
Calculations: *int_count = 0*

Open Properties box:
Display name: *Reset counters and trigger*
Calculations: *output_count = 0*
triggered = 0

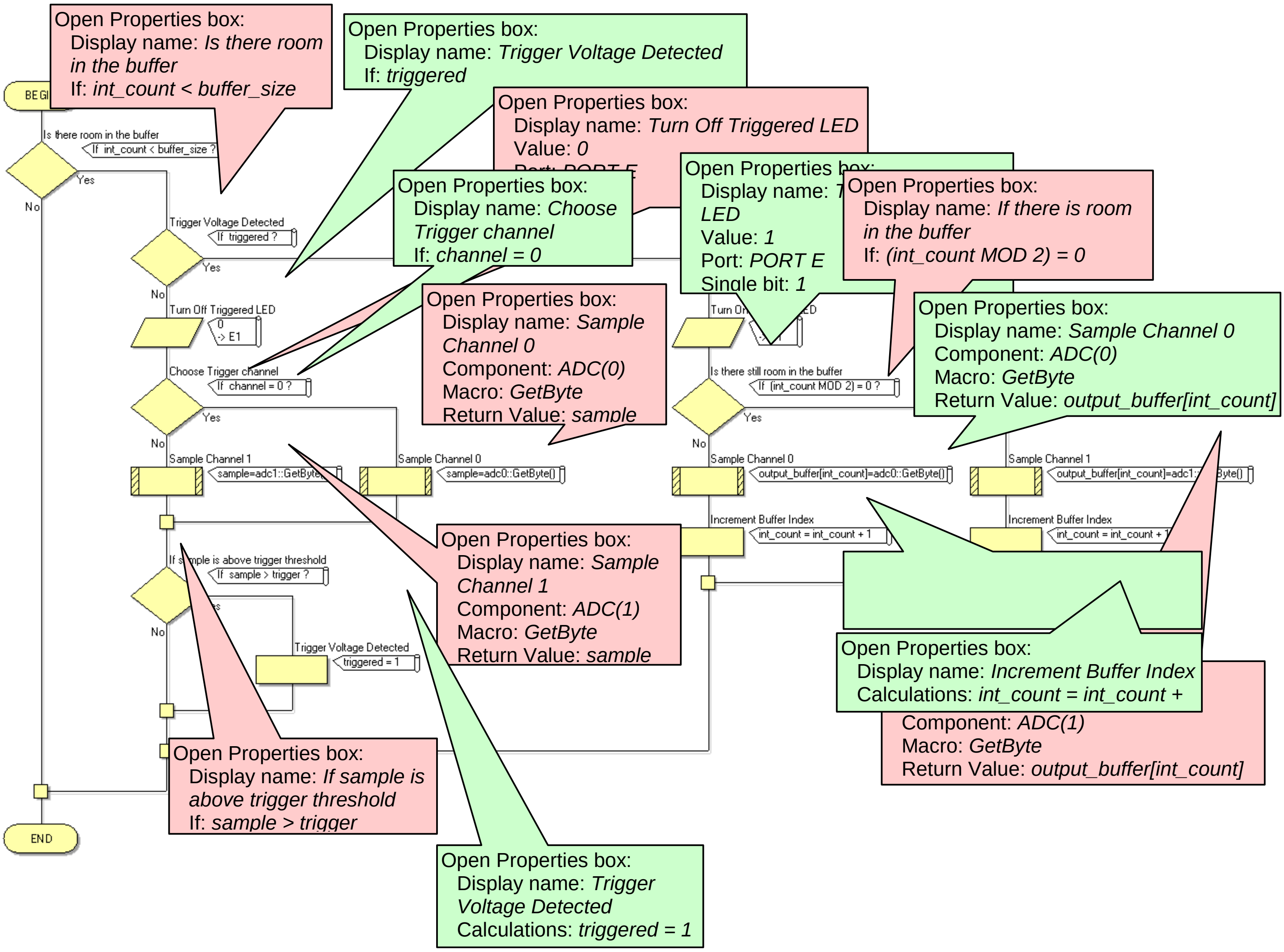
Open Properties box:
Display name: *All LEDs off*
Value: *0*
Port: *PORT E*
Entire port – No masking

Open Properties box:
Display name: *Initialise Buffer Position*
Calculations: *int_count = 0*

Open Properties box:
Display name: *We have sent a full display*
If: *output_count > 1100*

Open Properties box:
Display name: *Scope Running LED On*
Value: *1*
Port: *PORT E*
Single bit: *0*

Int_Handle macro



Open Properties box:
Display name: *Is there room in the buffer*
If: *int_count < buffer_size*

Open Properties box:
Display name: *Trigger Voltage Detected*
If: *triggered*

Open Properties box:
Display name: *Turn Off Triggered LED*
Value: *0*
Port: *PORT E*

Open Properties box:
Display name: *Choose Trigger channel*
If: *channel = 0*

Open Properties box:
Display name: *T*
LED
Value: *1*
Port: *PORT E*
Single bit: *1*

Open Properties box:
Display name: *If there is room in the buffer*
If: *(int_count MOD 2) = 0*

Open Properties box:
Display name: *Sample Channel 0*
Component: *ADC(0)*
Macro: *GetByte*
Return Value: *sample*

Open Properties box:
Display name: *Sample Channel 0*
Component: *ADC(0)*
Macro: *GetByte*
Return Value: *output_buffer[int_count]*

Open Properties box:
Display name: *Sample Channel 1*
Component: *ADC(1)*
Macro: *GetByte*
Return Value: *sample*

Open Properties box:
Display name: *Increment Buffer Index*
Calculations: *int_count = int_count +*

Component: *ADC(1)*
Macro: *GetByte*
Return Value: *output_buffer[int_count]*

Open Properties box:
Display name: *If sample is above trigger threshold*
If: *sample > trigger*

Open Properties box:
Display name: *Trigger Voltage Detected*
Calculations: *triggered = 1*

EB540

USB Solution

Student Guide

Contents

About this course

Aims:
The aim is to introduce the concepts involved in USB devices.

On completing this course you will have learned about:

- the relationship between USB masters, hubs and endpoints;
- the electrical principles behind USB architecture;
- the components that make up a USB device;
- the options available for USB devices;
- the addressing schemes;
- USB signals and routing;
- low power and sleep modes;
- USB device drivers;
- USB devices that do not require drivers.

What you will need:

To complete this course you will need the following equipment:

- Flowcode software (version 6 or higher)
- E-blocks including:
  - 1 Multiprogrammer (PIC - EB006)
    - with PIC18F4455 device and 4MHz crystal;
  - 1 Sensor E-Block (EB003);
  - 1 LED E-Block (EB004);
  - 1 LCD E-Block (EB005);
  - 1 Keypad E-Block (EB014);
  - 1 USB E-Block (EB055);
  - 1 RS232 E-Block (EB015);
  - 2 x IDC Cable (EB634)
  - Dual IDC Cable (EB635)

Using this course:

This course presents you with a number of tasks listed in the exercises that follow the text. All the information needed to complete these is contained in the notes.

Before starting the exercises, you should familiarise yourself with the background material.

Time:
To undertake all of the exercises will take around twelve hours.

Important note: Information presented here is correct at the time of publication. Please check the Matrix web site www.matrixltd.com for the latest E-Blocks documentation.

1 USB Overview

1.1 Preamble

As electronic and computer equipment becomes more and more pervasive, there is an ever-increasing demand to modernise and digitise. The universal serial bus (USB) protocol offers one way to achieve this. Most computer systems, phones, stereos and even televisions now include USB connections.

This review of USB describes the components of a typical USB system, and looks at how they communicate. It includes a glossary of the terms used in USB systems, and in the practical exercises that follow. It does not pretend to be an exhaustive primer on the subject, but aims rather to prompt the memory of the student carrying out the exercises.

1.2 Key Advantages of USB

One advantage of USB is that it can supply attached devices with electrical power so that one connection provides both a data link and power. Another advantage is that USB hubs can be used to add additional ports to the system. Other means of connecting devices, such as Firewire, MIDI or COM port protocols, are becoming either outdated, or reserved for more specialist equipment, such as digital video cameras and high throughput audio equipment.

A drawback of USB is its relatively complex structure. Most engineers can look at a serial bus and readily understand the connections responsible for transmitting data back and forth between devices. All you have to do is to toggle the signal voltage at the specified rate to achieve a communications link. With USB, communication is more complicated. For a start, there is a hierarchy to USB. Instead of linking equal peers, the system uses master / slave architecture, with the host controlling communication with the peripheral device. Connections use time splicing, (time-division multiplexing,) so that many USB end devices can talk to a single host device. This allows devices such as USB hubs to transform a single connection into multiple distinct connections, so that a number of peripheral devices can communicate with the same USB controller along the same cable.

Here are some reasons that USB has become so popular in modern day computer systems:

- capable of supplying power to peripheral devices;
- very high speed communications;
- scalable with the use of USB hub devices;
- noise immunity;
- built-in error checking and data correction;
- plug-and-play;
- versatile and highly configurable;
- low cost;
- addressable;
- physically small connectors;
- easy to embed into devices.

2 Introduction to USB

Universal Serial Bus (USB) is the name given to a specific type of high-level bus used in modern high speed digital systems. Learning how to use the USB standard can seem like a complex and daunting task, especially as the current USB 2.0 specification is over 650 pages long, (not counting a long list of associated USB standards.) Thankfully, much of this is outside the scope of this course, which focuses on creating USB peripherals.

Comparing USB with similar bus based technologies:

	USB 2.0	**Firewire**	**Serial**
Speed	12Mb/s – 480Mb/s	12.2Mb/s – 400Mb/s	1Mb/s
Architecture	Master / Slave	Peer-to-peer	Peer-to-peer
Performance	Good	Great	Excellent
Arbitration	Host controller	Device controlled	Direct connection
Devices per channel	127	16	1

2.1 Master / Slave Operation

USB is a Master / Slave system where the single Master (the host) is capable of controlling up to 127 individual Slave systems, (the peripheral devices.) Each USB device must connect directly to the host or go through a hub device, which acts to split the connection. Adding a hub onto the USB bus might incur a penalty in the form of additional latency. The hub is a USB device in itself, which communicates with the host to ensure that no collisions or errors happen between USB devices which are trying to communicate simultaneously.

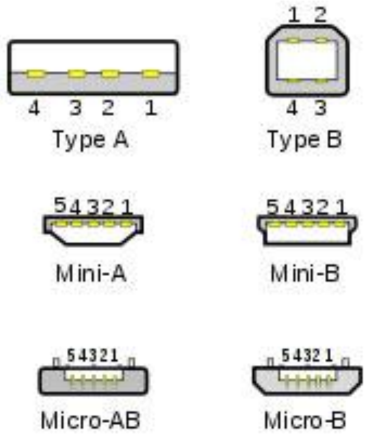
2.2 USB Power

A USB cable is made up of four shielded wires. Two carry power for peripheral devices, as +5V (nominally) and ground. The remaining two wires, named D+ and D-, carry high speed data in serial format. Current capability is limited to 500mA, though a maximum of 100mA is available during configuration (enumeration), or if the device is connected to a bus-powered hub.

The USB host can control how much current is provided for any single USB device. This helps to prevent hardware damage caused by short circuits or power hungry peripheral devices. The maximum current demand is defined during the enumeration process and is a number between 0 and 255, representing 2mA increments. For high current devices, a USB hub with its own power source can be situated between the device and the host.

2.3 Connectors

All USB *devices* have an ‘*upstream’* connection to the host and all USB *hosts* have a ‘*downstream’* connection to the device. The standard connectors are not mechanically interchangeable and therefore eliminate connection errors. The host connector is normally referred to as a type ‘A’ connector and is commonly found on all modern computer systems. The device connector comes in several formats, having different sizes and shapes. These upstream connectors are normally referred to as B for the large sized connectors, mini B for the small sized connectors and micro B for the very small sized connectors. There are also mini and micro varieties of the downstream type A connector but these are much less common.



Some connectors have five pins, not the four discussed earlier. These allow for a more flexible allocation of roles, which may be needed in smaller systems. The standard USB protocol uses a master / slave configuration. The master (usually a PC) controls the communication. The slave (peripheral device) responds to requests from the master. Sometimes, however, a device acts as a host (master) for part of the time and as a peripheral device (slave) at other times. For instance, when we send photographs direct to the internet from a digital camera via a mobile phone, which device should be the master?

The On-The-Go (OTG) supplement to the USB 2.0 specification introduces dual role devices that negotiate, and can alternate, the role of master and slave. The micro-AB connector allows a cable to be connected either way round. The cable orientation determines the initial roles. The fifth pin, the ID pin, is connected to ground inside the A plug and left floating in the B plug. The OTG device receiving the grounded ID pin takes on the host role initially, while the device with the floating ID pin defaults to peripheral. The supplement adds protocols such as HNP (Host Negotiation Protocol) to make possible this flexibility.

2.4 Functions

The word ‘function’ has a special meaning in the USB world. A function is a device which provides a particular ability to the host. Most devices offer only one function each. However, some, called compound devices, provide several functions. A video camera, for example, provides both audio data and video data. Such a device can have an embedded hub, allowing it to communicate with the host via a single USB cable.

2.5 Endpoints

Endpoints are sources or sinks of data which occur at the end of the communications channel at each USB function. A peripheral device sets up one-to-one links between each of its endpoints and the application software running on the host. All bus traffic travels to or from an endpoint.

All devices have endpoints. In practice, they are registers or buffers (blocks of memory) that store incoming and outgoing data. Each device has at least one endpoint, called endpoint 0, used for control and status communications with the host. It may have more endpoints. A USB 2.0 device can have up to 16 OUT and 16 IN endpoints. In this context, OUT always means from host to device, and IN always means from device to host.

Each is allocated an address so that the host can communicate directly with it. When the device driver in the host sends data to an endpoint, it is stored in the endpoint OUT buffer. Usually, this triggers an interrupt, which causes the device to read the data. It cannot reply directly to the host, as the communications link is controlled by the host, not the device.

Instead, it stores the reply data in the endpoint IN buffer, which the host can then request at a later time. This communication takes place using logical entities known as pipes.

2.6 Pipes

A pipe is a *logical* communication link between the software running in the host and an endpoint. More than simply a wire, each pipe has a set of parameters that define its performance, such as bandwidth, direction of data flow, and endpoint address. These are set up during enumeration, and depend on the device configuration.

All devices must support endpoint zero. This receives all control and status requests. Every device has a default control pipe to endpoint zero. This is bi-directional, and so may be considered as two pipes, one IN and one OUT, that share the same endpoint.

USB defines two types of pipes, message and stream pipes. Control transfers use bi-directional message pipes. All other transfer types use stream pipes:

- Stream Pipes have a pre-defined direction, either IN or OUT. They can be controlled by either the host or the peripheral device.
- Message Pipes are used only for control transfers, and are controlled only by the host.

2.7 Classes

Many USB devices have properties in common. Mice send information about mouse movements and button clicks; printers receive and print data, and return status information.

Some standard protocols, called Classes, have been defined to simplify configuration. Using these, hosts can use standardised device drivers rather than having to look for specific drivers from each vendor. The following table lists some of these approved classes, and the device descriptor to which they apply:

Class	**Descriptor where class is declared**
Audio	Interface
Communication	Device or interface
Human interface (HID)	Interface
Mass storage	Interface
Printer	Interface
Still image capture	Interface
Test and measurement	Interface
Video	Interface

2.8 Device Drivers

USB supports the plug'n'play standard, so that software device drivers are dynamically loaded and unloaded as devices are attached and removed from the bus. A device driver lets the computer know what functionality a device has and how to access it. When a device is plugged in, the host detects it, interrogates it and loads the appropriate driver. The end user is not required to select interrupt requests (IRQs) or specify memory addresses.

The appropriate driver is identified by two 16-bit numbers provided by the device, and unique to it. These are known as the product identifier (PID) and the vendor identifier (VID).

VID's are supplied by the USB Implementer's Forum, (USB-IF,) for a fee. The Matrix USB solution provides several PIDs, and the educational Matrix VID for use in developing educational, prototype or custom systems. For commercial systems it is better to purchase your own VID so that the hardware is permanently attributed to your company.

2.9 Addressing

The host must be able to direct data to the appropriate peripheral device. To achieve this, each endpoint is allocated a unique address during enumeration. This address is 7 bits long, allowing 128 different addresses. However, address 000 0000 is never allocated. Instead, it identifies the default endpoint 0 that a device must respond to during enumeration. This leaves 127 possible addresses for endpoints, explaining the earlier assertion that a host can control up to 127 peripheral devices.

2.10 Enumeration

Enumeration is the name of the information exchange that takes place when a USB device is connected to a host.
It includes:

- assigning an address to the device endpoint;
- reading device descriptors, (formatted blocks of information about the device and elements within it);
- determining the communication speed for the device;
- determining maximum packet size for communications;
- assigning a device driver to each endpoint;
- selecting a device configuration, which specifies features such as power requirements.

To do this, the host sends information requests in control transfers to the device endpoint 0. As only one device is enumerated at a time, only one responds to requests sent to endpoint 0, even though several devices may be attached. Where the host is a PC, the device will be listed in the Device Manager, when enumeration is completed.

2.11 Interface Speeds

The USB 2.0 specification allows for a number of different interface speeds:

- High Speed – 480Mbits/s
- Full Speed – 12Mbits/s
- Low Speed – 1.5Mbits/s

The USB host has pull-down resistors on both data lines, so that when no peripheral device is connected, both sit at logic low. This is called the reset or disconnected state.

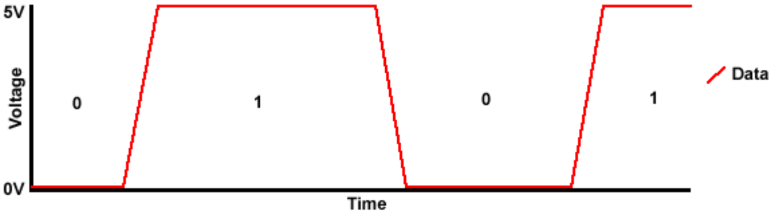
When a USB device is connected, it pulls one of the data lines high. In this way, the host, or hub, can detect when a device is plugged in. A full-speed device pulls the D+ data line high. A low-speed device pulls the D- data line high. By detecting the state of the data lines, the host determines the speed of device.

High Speed devices are a bit more complicated as they first connect as a Full Speed device and then, if the mode is supported, they disconnect the Full Speed pull-up resistor to allow them to run at the faster speed. The USB enabled PIC microcontrollers have the pull-up resistors that control the device speed built in so you do not have to replicate this circuitry.

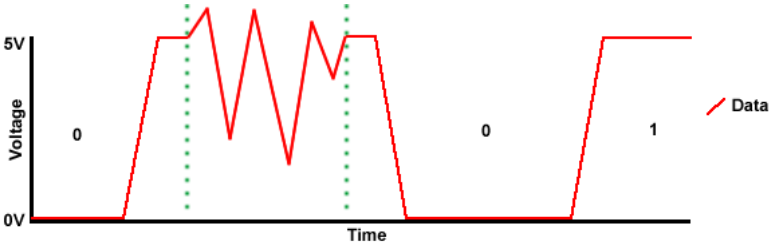
2.12 Noise Immunity

The signal wires, D+ and D-, are physically twisted around each other to limit external noise. The data on these lines is in differential format, which means that when one of the data lines is at logic 'low' then the other data line will be at logic 'high'. This increases the noise immunity.

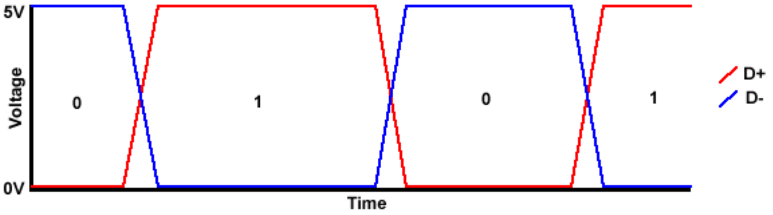
Here is an example of a standard serial data connection (non-differential format).



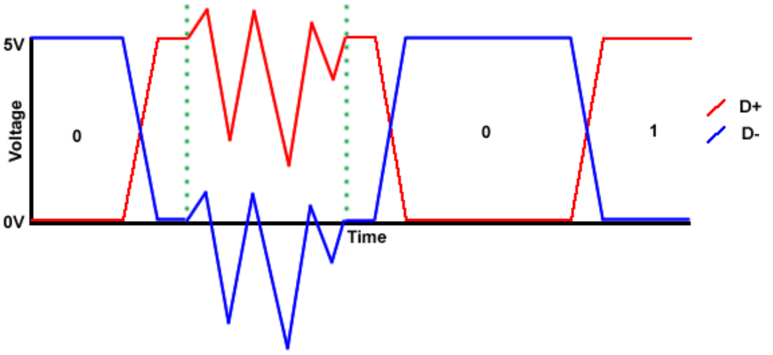
When noise is present, it becomes much harder to differentiate between the logical states.



The next example uses differential signalling. Notice the secondary signal that is the exact opposite of the original.



Now any noise affects both signals almost equally. The differential voltage between the signals has not been affected by the noise and so the data can be received correctly at the other end of the bus.



3 Data transfer

3.1 Transfer Types

USB communications takes place as a series of transfers. These can take the form of any one of four transfer types. These are: Control, Interrupt, Bulk and Isochronous transfers. Each transfer type provides the developer with trade-offs in error detection, latency and bandwidth.

Control transfers:
This is used by the host to configure the peripheral device. It enables the host to read information about the device, set the devices address and select a configuration and other settings, as described earlier. The Control transfer is key to the way USB devices auto detect, therefore every USB device must be able to support this communication mode.

Interrupt transfers:
This is used when the host or device must be checked periodically. Examples of devices that use this type include USB mice and keyboards. Despite the name, this type uses polling, not interrupts, to effect the communication. These transfers are used where an interrupt would have been used in earlier connection types.

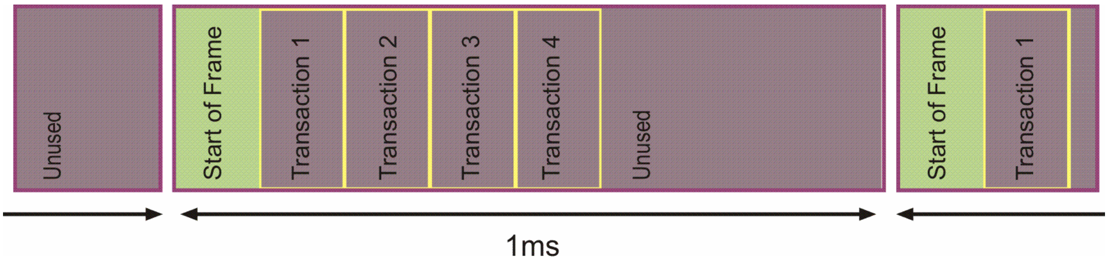
Bulk transfers:
This is intended for large blocks of data, where the rate of data transfer is not critical, but the validity of the data is important. Transfer rate depends on the system workload. This type uses ‘unused’ bandwidth after other types with specific bandwidth demands have been catered for. As a result, when the bus is busy, bulk transfers are delayed. At other times, they will proceed rapidly. Error detection, linked to retransmission where necessary, ensures that data is transmitted and received without error. Transfers of this type are used by devices such as printers and scanners, where large quantities of data are transferred.

Isochronous transfers:
This type is designed for time-sensitive information. It allows a device to reserve a predefined amount of bandwidth with guaranteed latency (delivery time.) The data is checked for errors, but when these occur, the data is dropped. There is no request for retransmission. This is ideal for systems such as Audio or Video where the user is unlikely to notice the loss of the odd data packet or frame, but would notice any irregularities in arrival time.

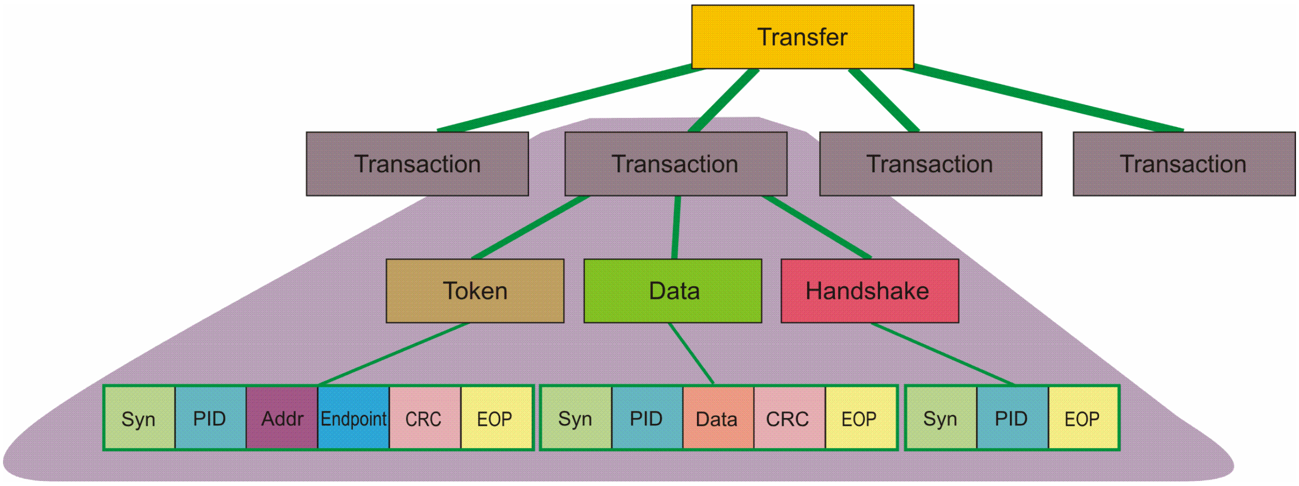
3.2 Transfers and Transactions (for USB 2 devices)

Data transfer to a device can consist of a number of separate transactions. These can occur in the same frame, or be spread across a number of frames.

A host sends out a new frame every millisecond. Each frame starts off with a Start of Frame packet, and contains a number of transactions. These can be directed to the same or to a number of devices. If directed to the same device, they can be addressed to the same endpoint, or to different endpoints within that device. Part of the frame can be unused, if there are no further transactions to send.



Each transaction must start with a token packet, and can then be followed by a data packet and a handshake packet. The fields inside each of these phases of the transaction are different, as the following diagram shows.



3.3 Transactions

USB transactions use four different types of packet:

- *Start of frame packets:*
  - indicate the start of a new frame.
- *Token packets*:
  - indicate the type of transaction to follow;
- *Data packets:*
  - contain the data payload;
- *Handshake packets:*
  - used for acknowledging data or reporting errors;

Start of frame packets: have the addition of an 11-bit frame number.

These are sent out by the host automatically every 1ms with a maximum error of plus or minus 500ns. The frame number increments with each frame, and when the maximum is reached, rolls over and starts again.

A start of frame packet contains the following fields:

Synchronisation	**Packet Identifier**	**Frame Number**	**CRC**	**End of Packet**

Token packets: come in three varieties: Setup, IN, and OUT.

The Setup packet is used to begin control transfers. It identifies the receiving endpoint and the nature of the request.
The IN packet informs the USB device that the host wishes to read information from the device.
The OUT packet informs the device that the host wishes to send information to it.

A token packet contains the following fields:

Synchronisation	**Packet Identifier**	**Address**	**Endpoint**	**CRC**	**End Of Packet**

Data packets: come in two varieties: Data0 and Data1.

These are used as part of the error-checking mechanism where the transfer involves a number of transactions. After each successful transaction, the data value, contained as part of the PID is toggled i.e.Data0 is used for the first transaction, then Data1, then Data0 and so on . Host and receiver monitor the data value as an indication that they are synchronised. Both varieties are capable of transferring up to 1023 bytes of data.

A data packet contains the following fields:

Synchronisation	**Packet Identifier**	**Data**	**CRC**	**End of Packet**

Handshake packets: come in three varieties: ACK, NAK and STALL.

The ACK acknowledges that the packet has been successfully received.
The NAK indicates that the USB device cannot send or receive data at the moment. This can also be used during an interrupt transaction where there is no data to send.
The STALL informs the host that the USB device is in a state where it needs intervention from the host.

A handshake packet contains the following fields:

Synchronisation	**Packet Identifier**	**End of Packet**

3.4 USB Packets

USB packets can contain the following fields.

- Synchronisation:
  - all packets transfers start with a sync field;
  - sync field is eight bits long;
  - synchronises the clock of the receiver with the transmitter;
  - last two bits indicate that the PID field is about to start.
- PID:
  - stands for Packet Identifier;
  - is also eight bits in size, though the four most significant bits are the inverse of bits 0 to 3, for use in error-checking;
  - identifies the type of packet that is being sent, as the following table shows:

Value	Packet type	PID name	Used in	Source
0001	Token	OUT	all	host
1001		IN	all	host
0101		SOF	start of frame	host
1101		SETUP	control	host
0011	Data	DATA0	all	both
1011		DATA1	all	both
0111		DATA2	isochronous	both
1111		MDATA	isochronous	both
0010	Handshake	ACK	all	both
1010		NAK	not isochronous	device
1110		STALL	not isochronous	device
0110		NYET	not isochronous	device

  - the two least-significant bits determine which group it falls into. This is why SOF is officially considered to be a token PID;
  - the least-significant bit is transmitted first.
- Address:
  - specifies which device the packet is designated for;
  - is seven bits long;
  - allows up to 127 devices on a single channel;
  - value 0 is used for a device which has not yet been assigned an address.
- Endpoint:
  - a four bit field;
  - destination endpoint for the packet.
- CRC:
  - cyclic redundancy check;
  - allows the data to be scanned for errors;
  - token packets have a five bit CRC;
  - data packets have a 16-bit CRC.
- End Of Packet:
  - three bit field
  - consists of two 0s and a 1 to indicate the end of a packet.

4 Setup

In the tables that follow, a standard notation is observed. For each field, the prefix to each name usually identifies the format of the data in the field:

'b' = byte (eight bits);	'w' = word (sixteen bits);
'bm' = bitmap;	'bcd' = binary-coded decimal
'i' = index;	'id' = identifier.

4.1 The Setup Stage

The Setup transaction has two purposes – to identify that a control request is taking place, and to define the type of request and the information needed. As described earlier, it consists of three phases – Token, Data and Handshake.

Every USB device must respond to setup packets on the default pipe connected to mandatory endpoint zero. They are used for the detection and configuration of the device and are responsible for assigning the device's address, requesting the device descriptor information and checking the status of an endpoint.

The Token phase:
identifies the receiver and indicates that a Setup transaction is taking place. The PID identifies the token as a Setup type. The packet includes the device and endpoint addresses.

The Data phase:
transmits the request, using the following format:

Offset	Field	Bytes	Value	Description
0	bRequestType	1	Bitmap	Transfer direction, type and recipient
1	bRequest	1	Value	Request
2	wValue	2	Value	Unicode encoded string
4	wIndex	2	Index	Index
6	wLength	2	Count	Number of bytes to transfer

In the bRequestType field:

- bit 7 defines the direction of data flow – 0 = OUT (often a SET request) and 1 = IN (a GET request.)
- bits 6 and 5 specify the type of request – 00 = standard, 01 = class, 10 = vendor, 11 = reserved.
- bits 4, 3, 2, 1 and 0 specify the recipient for the request – 00000 = device, 00001 = interface, 00010 = endpoint and 00011 = other element.

This format can be seen in action in the next three tables.
Standard device requests:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0b10000000	GetStatus (0x00)	0	0	2	Device status
0b00000000	ClearFeature (0x01)	Feature	0	0	None
0b00000000	SetFeature (0x03)	Feature	0	0	None
0b00000000	SetAddress (0x05)	Address	0	0	None
0b10000000	GetDescriptor (0x06)	Descriptor	LanguageID	Length	Descriptor and length
0b00000000	SetDescriptor (0x07)	Descriptor	LanguageID	Length	Descriptor and length
0b10000000	GetConfiguration (0x08)	0	0	1	Configuration
0b00000000	SetConfiguration (0x09)	Configuration	0	0	None

Standard interface requests:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0b10000001	GetStatus	0	Interface	2	Interface status
0b00000001	ClearFeature	Feature	Interface	0	None
0b00000001	SetFeature	Feature	Interface	0	None
0b10000001	GetInterface	0	Interface	1	Alternate interface
0b00000001	SetInterface	Alternate setting	Interface	0	None

Standard endpoint requests:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0b10000010	GetStatus	0	Endpoint	2	Endpoint status
0b00000010	ClearFeature	Feature	Endpoint	0	None
0b00000010	SetFeature	Feature	Endpoint	0	None
0b10000010	SyncFrame	0	Endpoint	2	Frame number

The features referred to in ‘SetFeature’ and ‘ClearFeature’ depend on the recipient. For devices, the two features currently specified are ‘device remote wakeup’ and ‘test mode’. No features are currently specified for interfaces, while for endpoints, the only feature is ‘endpoint halt’, which allows the host to stall and clear an endpoint.

The field ‘wValue’ is used by the host to send data to the device. Hence it is set to zero in ‘Get...’ requests. The data sent depends on the type of request. In ‘Set Feature’ requests, the ‘wValue’ field specifies the feature. In ‘Set Address’ requests, the ‘wValue’ field specifies the address, and so on.

The field ‘wIndex’ is used to specify the interface or endpoint involved in the request.

The field ‘wLength’ specifies the number of data bytes in the Data stage that follows.

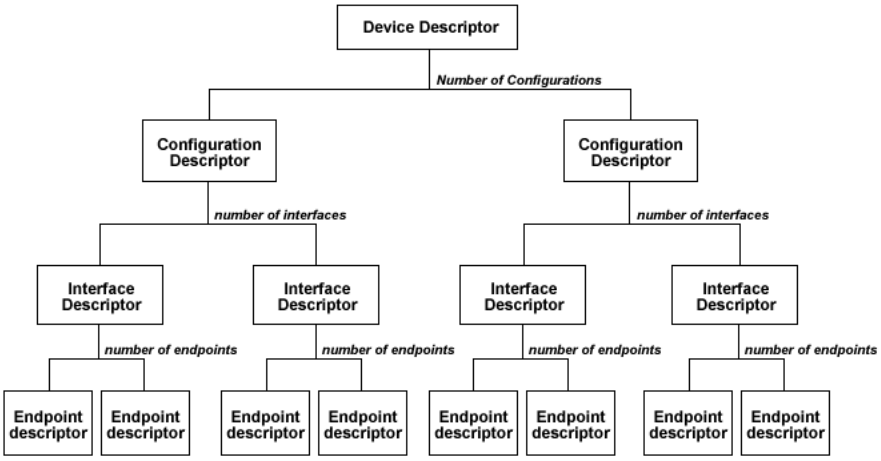
The Handshake phase:
Here, the device transmits an acknowledgement, using a PID coded as ‘ACK’, if it has received the Token and Data phases without error.

5 Learning about USB Device Capabilities

5.1 Descriptors

During enumeration, the host learns about device capabilities using control transfers. First it requests the Device Descriptor, describing features of the whole device, such as the supported USB version, maximum packet size, vendor and product identifiers and number of possible configurations. Subsequent requests concern finer and finer detail about elements of the device, as the diagram shows. In practice the complete descriptor is an extended array of data, with each descriptor following on from the last.

The USB device descriptors used in the Flowcode USB components can be found in the USB library entitled usb_config_x.c where x represents the specific Flowcode component.



Each descriptor starts with the same two fields. ‘bLength’, which tells the host how many bytes of data make up that descriptor, and ‘bDescriptorType’ which identifies the information it contains, as shown in the next table:

bDescriptorType	Descriptor Type	Description
0x01	Device	Required
0x02	Configuration	Required
0x03	String	Optional descriptive text
0x04	Interface	Required
0x05	Endpoint	Not required if the device uses only Endpoint 0
0x06	Device_qualifier	Required only when device is both full and high speed.
0x07	Other_speed_ configuration	Required only when device is both full and high speed.
0x08	Interface_power	Required only for interface-level power management
0x09	OTG	Only for On The Go USB devices
0x0A	Debug	Optional
0x0B	Interface_association	Only for composite devices

The field ‘bDescriptorType’ is one byte long.
The entries in the table are the standard descriptor types, and use only bits 0 to 3. There can be other descriptor types, using bit 4 in addition.
Bit 7 is always zero. Bits 6 and 5 identify the source of the descriptor type, as follows:

- 00 = standard;
- 01 = class-defined;
- 10 = vendor-defined;
- 11 = reserved.

5.2 USB Device Descriptors:

Each device has only one Device Descriptor, the format of which is shown below:

Offset	Field	Bytes	Value	Description
0	bLength	1	Number	Size of the descriptor in bytes
1	bDescriptorType	1	Constant	Device descriptor
2	bcdUSB	2	BCD	USB specification number
4	bDeviceClass	1	Class	Class Code
5	bDeviceSubClass	1	SubClass	Subclass Code
6	bDeviceProtocol	1	Protocol	Protocol Code
7	bMaxPacketSize	1	Number	Maximum Packet Size for endpoint 0
8	idVendor	2	ID	Vendor ID
10	idProduct	2	ID	Product ID
12	bcdDevice	2	BCD	Release number
14	iManufacturer	1	Index	Index of manufacturer string
15	iProduct	1	Index	Index of product string
16	iSerialNumber	1	Index	Index of serial number string
17	bNumConfigurations	1	Integer	Number of possible configurations

5.3 USB Configuration Descriptors

The configuration descriptor specifies a number of different properties which relate to how the device will behave whilst running in this configuration. This includes settings such as how the device is powered and the maximum power consumption. It is therefore possible to create USB devices with more than one configuration descriptor. An example of this would be to allow the device to run in a high power mode when only connected via the USB and then use an alternative low power mode when connected to a mains power source.

Offset	Field	Bytes	Value	Description
0	bLength	1	Number	Size of the descriptor in bytes
1	bDescriptorType	1	Constant	Configuration descriptor
2	wTotalLength	2	Number	Total length of the data returned
4	bNumInterfaces	1	Number	Number of interfaces
5	bConfigurationValue	1	Number	Value to select this configuration
6	iConfiguration	1	Index	Index of string descriptor
7	bmAttributes	1	Bitmap	Power configuration and wakeup control
8	bMaxPower	1	mA	Maximum power consumption

5.4 USB Interface Descriptors

The interface descriptor represents a single feature of the device. The USB serial Flowcode component uses two interface descriptors. One handles the device emulation as a COM port. The other is responsible for the pipes used to stream the serial data.

Offset	Field	Bytes	Value	Description
0	bLength	1	Number	Size of the descriptor in bytes
1	bDescriptorType	1	Constant	Interface descriptor
2	bInterfaceNumber	1	Number	Number of interface
3	bAlternateSetting	1	Number	Value used to select alternate setting
4	bNumEndpoints	1	Number	Number of endpoints used by the interface
5	bInterfaceClass	1	Class	Class code
6	bInterfaceSubClass	1	SubClass	Subclass code
7	bInterfaceProtocol	1	Protocol	Protocol code
8	iInterface	1	Index	Index of string descriptor for this interface

4.5 USB Endpoint Descriptors

The endpoint descriptor describes endpoints other than endpoint zero, which is always assumed to be a control endpoint and is configured before any descriptors are requested. The host will use the information provided in the descriptors to determine the overall bandwidth requirements of the USB device.

Offset	Field	Bytes	Value	Description
0	bLength	1	Number	Size of the descriptor in bytes
1	bDescriptorType	1	Constant	Endpoint descriptor
2	bEndpointAddress	1	Endpoint	Endpoint address
3	bmAttributes	1	Bitmap	Transfer and synchronisation type
4	wMaxPacketSize	2	Number	Maximum packet size
6	bInterval	1	Number	Interval for polling endpoint data transfers

4.6 USB String Descriptors

The string descriptor is an optional part of the descriptor chain and is used to provide human readable information such as the product name or manufacturer. The first string descriptor is used to allow support for multiple languages.

Language String Descriptor				
Offset	**Field**	**Bytes**	**Value**	**Description**
0	bLength	1	Number	Size of the descriptor in bytes
1	bDescriptorType	1	Constant	String descriptor
2	wLANGID[0]	2	Number	Supported language - 0x0409 - US English
4	wLANGID[1]	2	Number	Supported language - 0x0809 - UK English
6	wLANGID[2]	2	Number	Supported language - 0x0407 - German

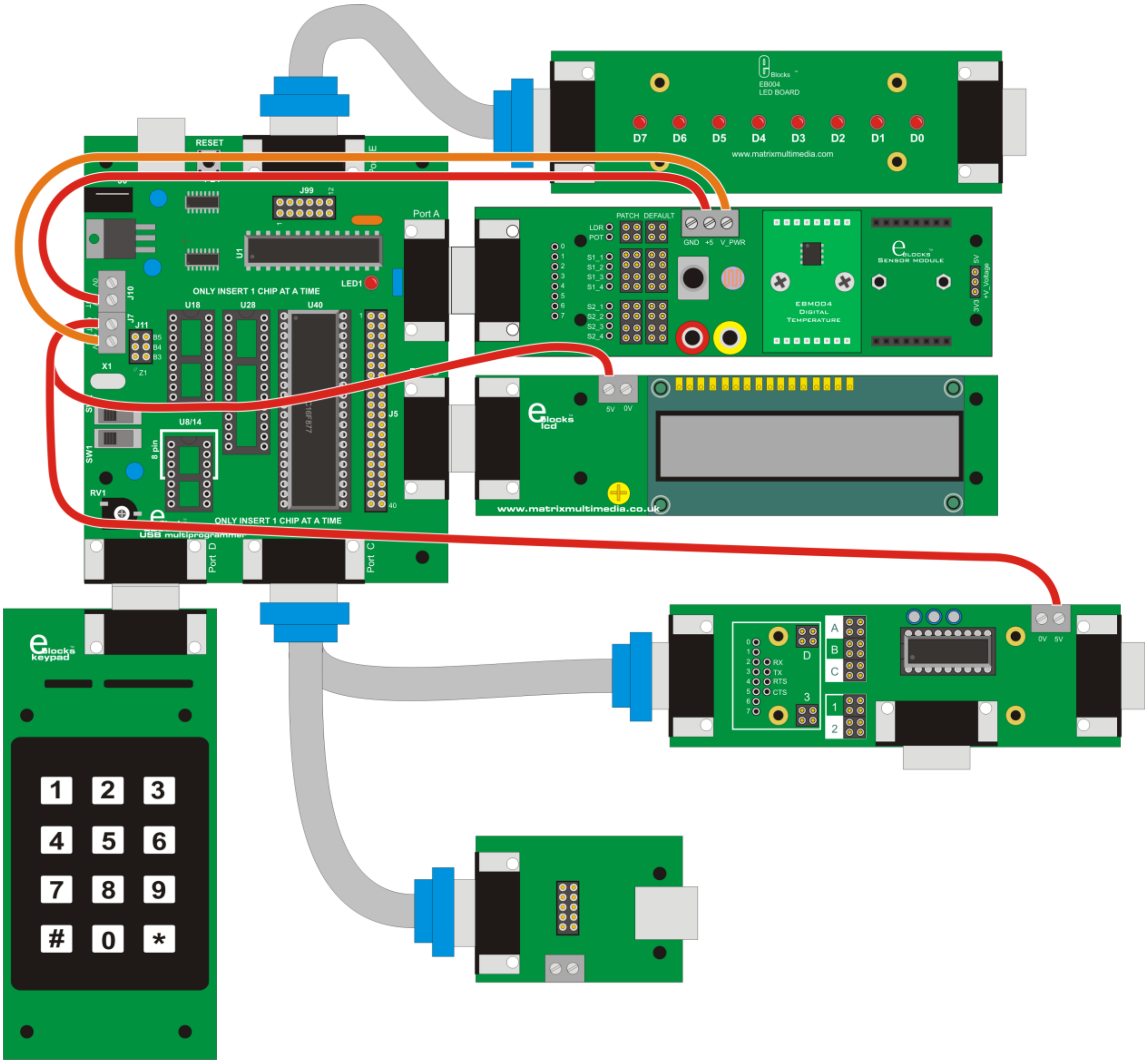
String Descriptor				
Offset	**Field**	**Bytes**	**Value**	**Description**
0	bLength	1	Number	Size of the descriptor in bytes
1	bDescriptorType	1	Constant	String descriptor
2	bString	n	Unicode	Unicode encoded string

6 The Matrix USB Training Solution

6.1 Solution Overview

The Matrix USB solution comes with everything needed for USB device development and can be used to develop a full application. The PIC18F4455 microcontroller fitted into the EB006 Multiprogrammer has a Full Speed USB hardware peripheral, which allows for easy exploration of the USB architecture. The device uses a phase locked loop to generate a stable clock speed of 48MHz from a clock source of 4MHz. PIC microcontrollers process a single assembler instruction for every four clock cycles so the clock speed of 48MHz is therefore crucial for enabling the device to run at a speed of 12 million instructions per second, the exact speed of a Full Speed USB communication.

The following diagram indicates the default layout of the E-blocks and the default jumper locations and wiring. A complete list of the jumper settings and wiring links are given on the next page.



6.2 Default connections and settings:

- EB090 – Sensors E-Block
    - Connected to Port A using IDC Cable
    - +V terminal connected to +V on EB006
    - Jumper Settings – Default
- EB004 – LEDs E-Block
    - Connected to Port E using IDC Cable
- EB005 – LCD E-Block
    - Connected to Port B
    - +V terminal connected to +V on EB006
    - Jumper Settings – Default
- EB006 – Multiprogrammer E-Block
    - PIC18F4455 fitted to 40-way socket
    - 4MHz Crystal fitted to crystal socket
    - Target Voltage Jumper J15 – 5V
    - Power Jumper J11 – PSU
    - Connections Jumper J14 – USB
    - Oscillator Jumper J18 – OSC
- EB014 – Keypad E-Block
    - Connected to Port D
- EB015 – RS232 E-Block
    - Connected to Port C using IDC splitter cable
    - +V terminal connected to +V on EB006
    - Data Jumper J4 – Position C
    - Flow Control Jumper J7 – Position 3
    - Single Wire Link – Pin 0 to Pin RTS
    - Single Wire Link – Pin 1 to Pin CTS
- EB055 – USB E-Block
    - Connected to Port C using IDC splitter cable
    - Jumper settings – Position A

7 Flowcode and USB

The Flowcode software is a powerful flowcharting language designed for microcontrollers. The PIC version, which we will be using with the 18F4455 microcontroller, comes complete with three distinct USB components i.e. three integral USB device descriptors that allow the device to connect in one of three possible modes. The components also come with driver generation tools so that you can customise your device to your requirements where required, and generate the device driver for the host without tedious manual editing.

The Flowcode USB components are as follows:

- **USB Serial –** emulates a communications port to replace legacy serial ports.
- **USB HID –** highly configurable human interface device.
- **USB Slave –** turns the microcontroller into a slave to the host.

Each component essentially creates two data endpoints to allow bidirectional communication between the embedded device and the USB host. With the underlying USB bus endpoints, device drivers, device descriptors and communications taken care of, all that remains is to create the data arrays used to transmit and receive data. Using Flowcode, we can therefore forget that the device uses USB, and treat it is a simple high speed, low-level, bi-directional serial data cable.

7.1 USB Serial Component

The simplest of the USB components, it can be treated as a normal serial RS232 connection. The USB host sees the device as a communications port, once the driver has been installed, and allows programs like Hyperterminal and Labview to create a direct communications link to the device. The component allows you to read or write single bytes or full strings, and so it can be used to create a true RS232 converter or replacement.

7.2 USB HID Component

This is similar to the serial component, in that you can use it to read and write data, but in addition, communication is now a timed event, controlled by the computer. For example, it can be used to create a simple USB keyboard. A standard HID keyboard transfers 8 bytes of information to the USB host in regular intervals of say 10 milliseconds. All your program has to do, therefore, is to update the device’s IN data buffer based on key press readings. It should also check the OUT data buffer for the status of the LEDs.

The HID component relies on a HID descriptor stored in the embedded device, which is used to enumerate the device and integrate it into the system, instead of requiring a device driver like the other two USB components. The component comes with two HID descriptors that set the device up as either a standard keyboard or as a mouse. A third custom descriptor is also available that can be adapted by the user, with the HID descriptor generation tool available from the HID page of the USB standards website.

HID descriptors can cover a wide variety of devices such as printers, speakers, touch-screen monitors etc.

7.3 USB Slave Component

The USB slave component forces the microcontroller to become a slave to the computer in that it must receive data from the USB host before trying to send data back. This is useful for creating question / answer style systems. Again Flowcode simplifies the task by allowing a configurable slave service macro. This software macro is called whenever data from the USB host is received. This allows you to process the incoming data as soon as it arrives, make decisions and respond with a timely appropriate answer.

One example of this is a command to sample an analogue channel. The acknowledgement could then contain the result of the sampling. As the data is in the form of an array, you can use the command and acknowledge functionality to transfer fairly large amounts of data.

7.4 Enumeration Wait Setting

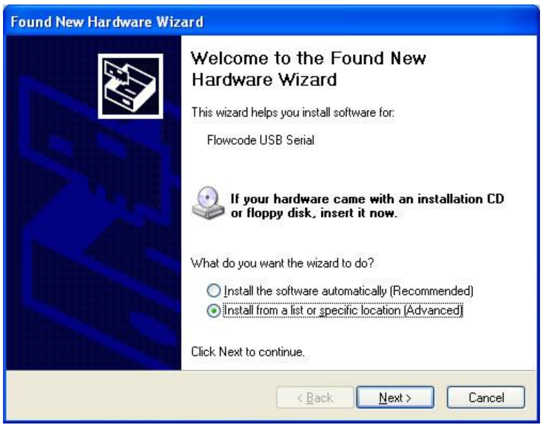
Each of the USB components has a property entitled ‘enumeration wait’. With this option un-ticked, the component will wait forever for drivers to be installed onto the PC before returning from the macro. This is handy when first getting to grips with the component, as it allows you unlimited time to install the driver correctly. When this option is ticked, a timeout delay can be specified in seconds, allowing you to use the ‘initialise’ macro to test for a USB connection in a program where you do not necessarily need a permanent USB. The ‘initialise’ macro returns ‘0’ for a successful start-up and ‘255’ for an initialisation error or connection timeout.

8 USB Serial device

8.1 Installing the Device Driver

A microcontroller running the USB Serial Flowcode component will require a device driver to allow it to work correctly with Windows. The device driver has a '.inf' file extension, and can be generated via the Properties page of the Flowcode component. This ensures that the driver matches the names and identifiers specified in the component properties. Consequently, you should always create the driver after setting up the component properties.

When you first plug in the device, you will see a screen similar to the one below. (If this does not appear, you can force a driver installation by going into Device Manager, right clicking the device and selecting 'Update driver software'. One way to enter Device Manager is to click the Start menu, select 'Run...', type in '"devmgmt.msc" and hit the Enter key)
Select the 'Install from a list or specific location,' option and then click 'Next >'.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

Flowcode USB Serial

If your hardware came with an installation CD or floppy disk, insert it now.

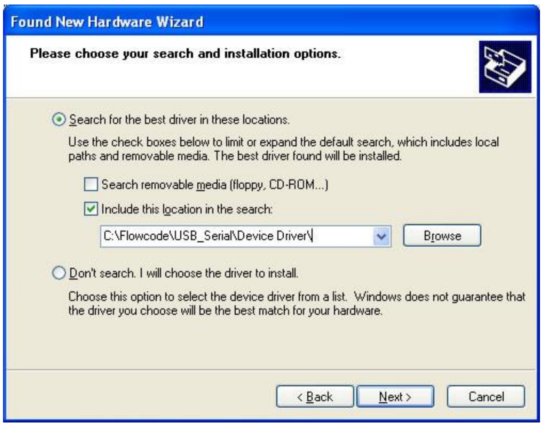
What do you want the wizard to do?

○ Install the software automatically (Recommended)

● Install from a list or specific location (Advanced)

Click Next to continue.

Now click the browse button and browse to the location on your computer where you saved the driver file to. Once you have setup the path to the file, use the 'Next >'button to continue.



Found New Hardware Wizard

Please choose your search and installation options.

● Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)

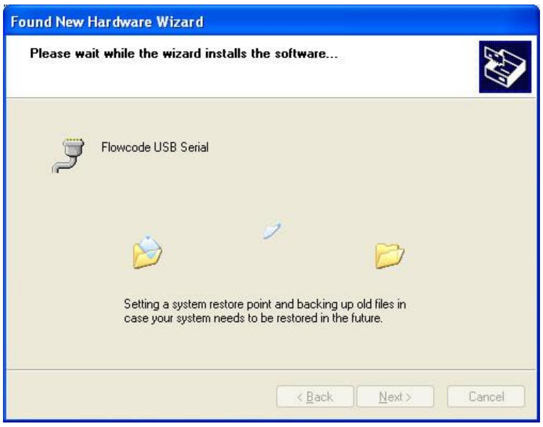
☑ Include this location in the search:

C:\Flowcode\USB_Serial\Device Driver\

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

After a short delay, the driver should start installing. If you receive a red dialogue message saying that the device has not been tested, click ‘OK’ to proceed.

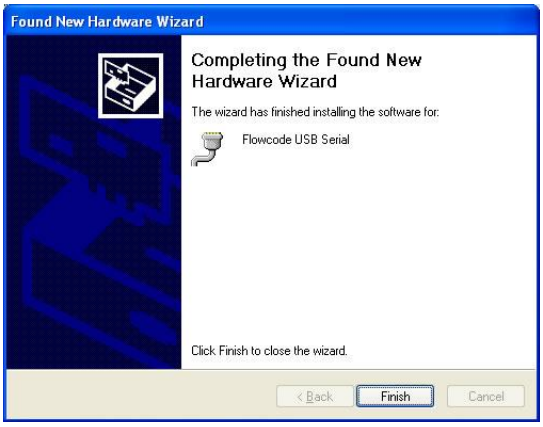

Found New Hardware Wizard

Please wait while the wizard installs the software...

Flowcode USB Serial

Setting a system restore point and backing up old files in case your system needs to be restored in the future.

Once the driver is installed, you will be presented with the following screen. Click ‘Finish’ to complete the wizard and start using your device.


Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

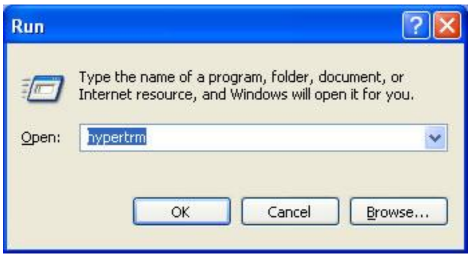
Flowcode USB Serial

Click Finish to close the wizard.

8.2 USB Serial Device and HyperTerminal

You must use terminal emulator software to communicate with the USB Serial device. Windows XP and earlier systems come with a suitable package called HyperTerminal. Newer Windows operating systems do not include the HyperTerminal software as standard but it can be downloaded from the Hilgraeve website. There is also a freeware alternative, named Realterm, available from the Sourceforge website. Software such as Visual Basic, and Labview, come can access the communications port directly.

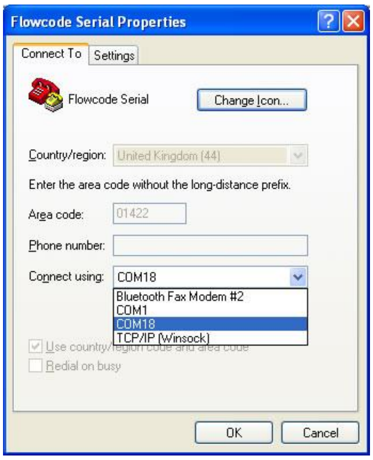
To run HyperTerminal click the Start menu, select ‘Run...’ type in “hypertrm” and hit the Enter key.


Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: hypertrm

Once you have opened HyperTerminal you must name the connection. Once you have entered a name click OK.



Flowcode Serial

Country/region: United Kingdom (44)

Enter the area code without the long-distance prefix.

Area code: 01422

Phone number:

Connect using: COM18

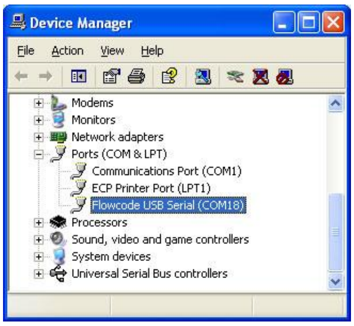
- Bluetooth Fax Modem #2
- COM1
- COM18
- TCP/IP (Winsock)

☑ Use country/region code and area code

□ Redial on busy

Next you must assign the correct port for the connection. Select this from the drop-down list and then click OK.

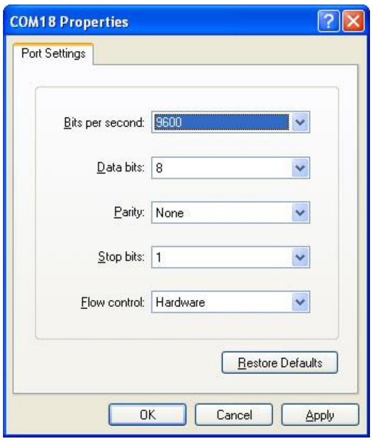
If you do not know which COM port your USB Serial device is connected to, then open the Device Manager application. The USB Serial device will appear under the Ports section once the driver has been installed. If the driver has not been installed, then please refer back to the previous section.



- Modems
- Monitors
- Network adapters
- Ports (COM & LPT)
  - Communications Port (COM1)
  - ECP Printer Port (LPT1)
  - Flowcode USB Serial (COM18)
- Processors
- Sound, video and game controllers
- System devices
- Universal Serial Bus controllers

Use the next window to configure aspects of the COM port connection, such as transfer rate. The settings can be left at the values shown above. For some applications, the transfer rate can be raised or lowered. For a USB to RS232 device, it must match the device’s throughput otherwise a data bottleneck may result, causing missed data or program instability.

Once you have configured the properties, click ‘OK’. You should now be connected to the USB device. Any characters typed into the HyperTerminal window will be forwarded to the USB device. Any data sent back will be displayed in ASCII format.



Bits per second: 9600

Data bits: 8

Parity: None

Stop bits: 1

Flow control: Hardware

9 USB Slave Device

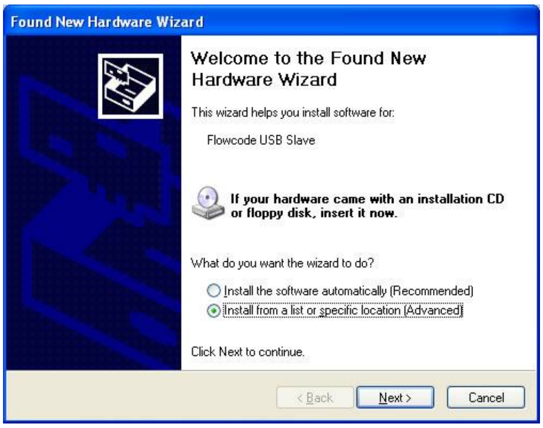
9.1 Installing the Device Driver

A microcontroller running the USB Slave Flowcode component also needs a device driver to allow it to work with Windows. As before, the driver has a '.inf' file extension. Generating it from the Properties page of the Flowcode component ensures that the driver matches the names and identifiers specified in the component properties. Therefore you should always create the driver after you have setup the component properties. The driver .inf file for the USB Slave component requires a few additional support files, provided on the solution CD and in the Slave example folders.

When you first plug in the device, you will see a screen similar to the one opposite.

(If this does not appear, you can force a driver installation by going into Device Manager, right clicking the device and selecting 'Update driver software'. One way to enter Device Manager is to click the Start menu, select 'Run...', type in '"devmgmt.msc" and hit the Enter key)

Select the 'Install from a list or specific location,' option and then click 'Next >'.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

Flowcode USB Slave

If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

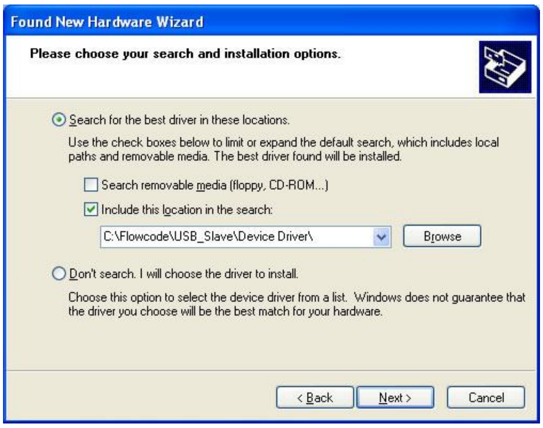
○ Install the software automatically (Recommended)

● Install from a list or specific location (Advanced)

Click Next to continue.

Now browse to the location on your computer that you saved the driver file to.

Once you have setup the path to the file, use the 'Next >'button to continue.



Found New Hardware Wizard

Please choose your search and installation options.

● Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

□ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

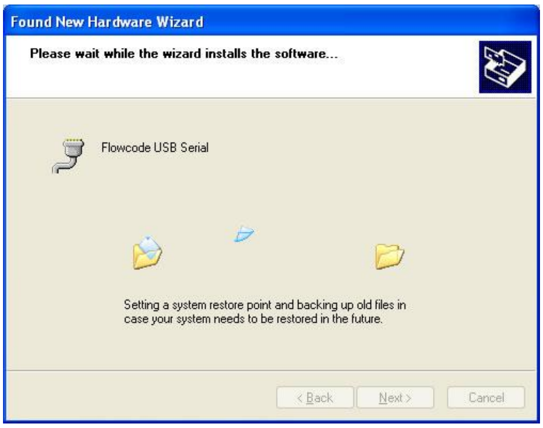
C:\Flowcode\USB_Slave\Device Driver\

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

After a short delay, the driver should start installing.

If you receive a red dialogue message saying that the device has not been tested, click 'OK' and proceed.



Found New Hardware Wizard

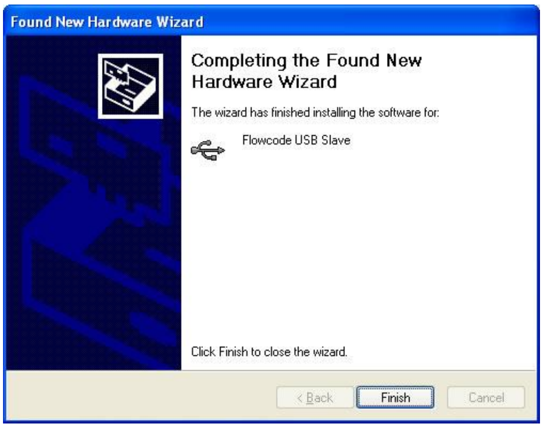
Please wait while the wizard installs the software...

Flowcode USB Serial

Setting a system restore point and backing up old files in case your system needs to be restored in the future.

Once the driver has finished installing you will see the screen shown opposite.

Click 'Finish' to complete the wizard and start using your device.



Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Flowcode USB Slave

Click Finish to close the wizard.

9.2 USB Slave and Visual Basic

To communicate with the USB Slave device you must use a DLL that has been written for that purpose. The functions included in the DLL are as follows.

ECIO_GetDLLVersion ()
Returns the DLL version number

ECIO_GetDeviceCount (Identifier)
Identifier – array containing the VID and PID of the device.
Returns the number of USB Slave devices with a matching VID and PID.

ECIO_Open (Index, Identifier)
Index – index of the device you wish to connect to.
Identifier – array containing the VID and PID of the device.
Returns the connection status: 0=Error, 1=Connection opened

ECIO_Transmit (DataOut, LenOut, ALenOut, DataIn, LenIn, ALenIn, Timeout)
DataOut – Data array to send to the device
LenOut – Number of bytes to send
ALenOut – Actual number of bytes sent
DataIn – Data array to store data received from the device
LenIn – Number of bytes to receive
ALenIn – Actual number of bytes received
Timeout – Length of time in milliseconds to wait for incoming data
Returns the communication status: 0=Error, 1=Communication successful

ECIO_Close ()
Returns the connection status: 0=Error, 1= Connection closed

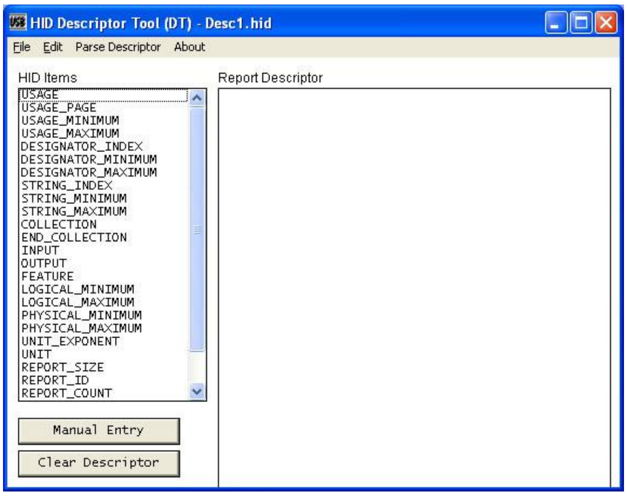
Though the functions are all named “ECIO” they will work with any device that is capable of running the USB Slave component. (The Matrix ECIO was the first device that developed for use with the component and the DLL.)

The USB identifier numbers are defined by a byte array containing the VID and PID information in the form of a string. The default USB Slave identifier string would look something like this: “vid_12bf&pid_f030”.

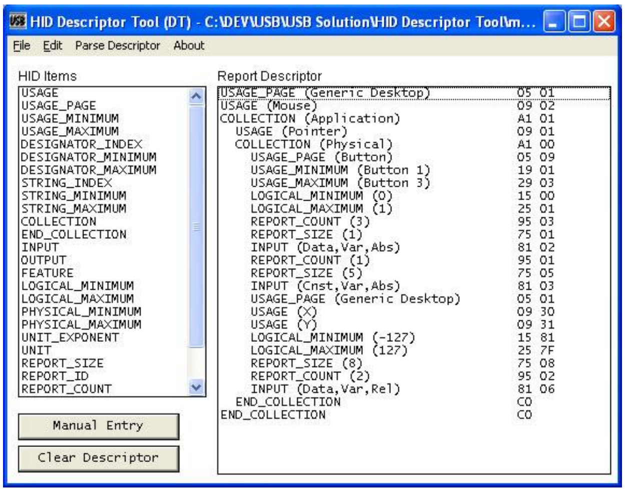
Refer to the sample VB projects in the solution when creating new applications. They could even be used to create a template, as they contain all the definitions needed to allow the DLL functions to be referenced correctly.

10 USB HID Custom Descriptor Generation

The HID descriptor replaces the driver that is required for most other USB devices. It is responsible for informing the computer what capabilities the device has and how the data will be formatted. The descriptor is fairly complicated to generate by hand so there is a descriptor tool available from the USB organisation that will generate the device descriptors for you. This tool is included on the solution CD or can be downloaded from the following page. www.usb.org/developers/hidpage/



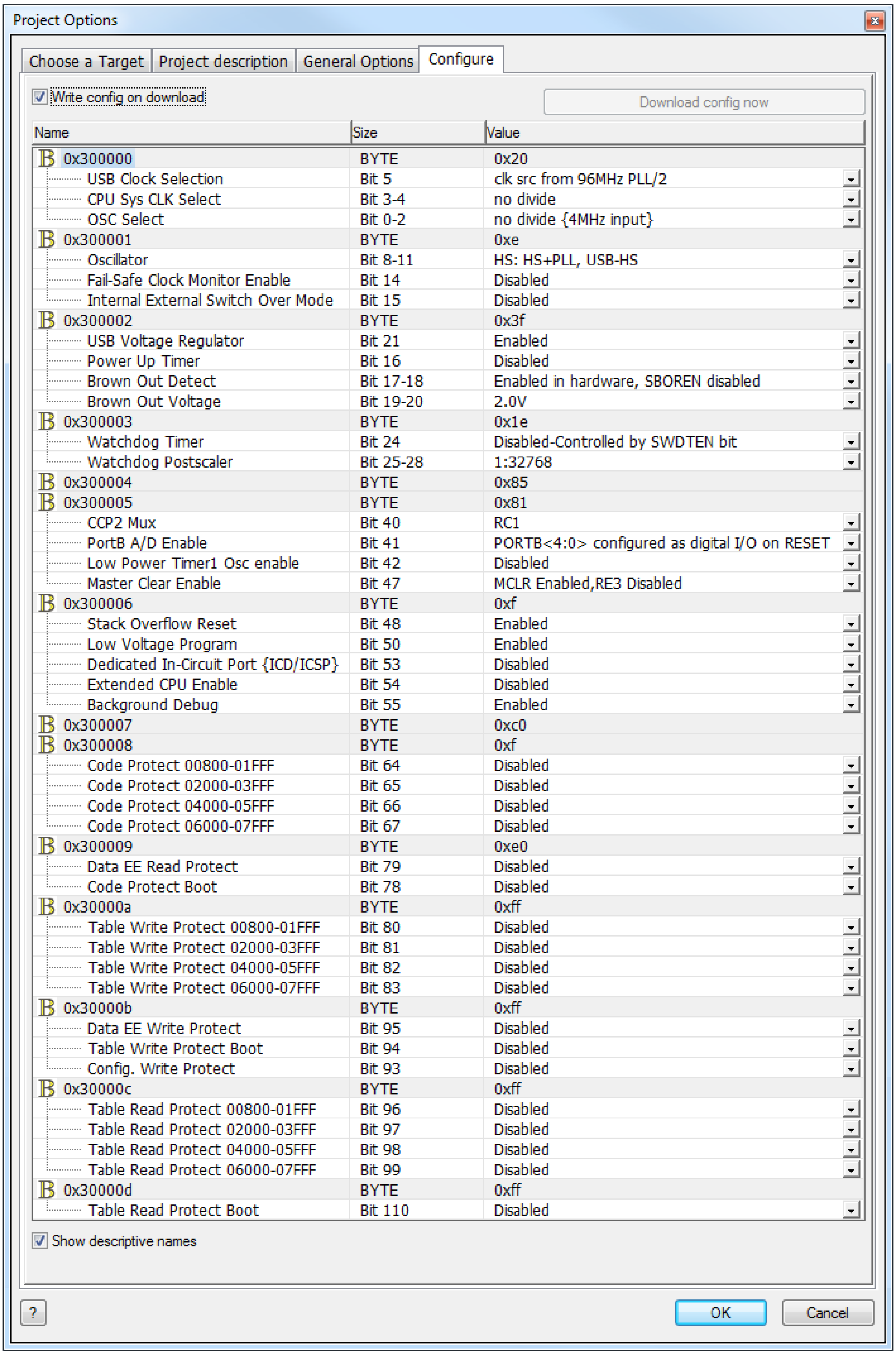
Several sample descriptors are available by clicking on ‘File’ and ‘Open...’ and browsing the descriptor tool folder. Here is the generated descriptor for a mouse as used in example 1.



```
USAGE_PAGE (Generic Desktop)              05 01
USAGE (Mouse)                             09 02
COLLECTION (Application)                  A1 01
  USAGE (Pointer)                         09 01
  COLLECTION (Physical)                   A1 00
    USAGE_PAGE (Button)                   05 09
    USAGE_MINIMUM (Button 1)              19 01
    USAGE_MAXIMUM (Button 3)              29 03
    LOGICAL_MINIMUM (0)                   15 00
    LOGICAL_MAXIMUM (1)                   25 01
    REPORT_COUNT (3)                      95 03
    REPORT_SIZE (1)                       75 01
    INPUT (Data,Var,Abs)                  81 02
    REPORT_COUNT (1)                      95 01
    REPORT_SIZE (5)                       75 05
    INPUT (Cnst,Var,Abs)                  81 03
    USAGE_PAGE (Generic Desktop)          05 01
    USAGE (X)                             09 30
    USAGE (Y)                             09 31
    LOGICAL_MINIMUM (-127)                15 81
    LOGICAL_MAXIMUM (127)                 25 7F
    REPORT_SIZE (8)                       75 08
    REPORT_COUNT (2)                      95 02
    INPUT (Data,Var,Rel)                  81 06
  END_COLLECTION                          C0
END_COLLECTION                            C0
```

11 PIC18F4455 Configuration

The PIC18F4455 microcontroller should be configured using the following settings to allow it to work correctly with the hardware in the solution. The chip can be configured in Flowcode by using the ‘Configure’ tab in the ‘Project Options’, which is accessed from the ‘Build’ menu.



☑ Write config on download

Name	Size	Value
0x300000	BYTE	0x20
USB Clock Selection	Bit 5	clk src from 96MHz PLL/2
CPU Sys CLK Select	Bit 3-4	no divide
OSC Select	Bit 0-2	no divide {4MHz input}
0x300001	BYTE	0xe
Oscillator	Bit 8-11	HS: HS+PLL, USB-HS
Fail-Safe Clock Monitor Enable	Bit 14	Disabled
Internal External Switch Over Mode	Bit 15	Disabled
0x300002	BYTE	0x3f
USB Voltage Regulator	Bit 21	Enabled
Power Up Timer	Bit 16	Disabled
Brown Out Detect	Bit 17-18	Enabled in hardware, SBOREN disabled
Brown Out Voltage	Bit 19-20	2.0V
0x300003	BYTE	0x1e
Watchdog Timer	Bit 24	Disabled-Controlled by SWDTEN bit
Watchdog Postscaler	Bit 25-28	1:32768
0x300004	BYTE	0x85
0x300005	BYTE	0x81
CCP2 Mux	Bit 40	RC1
PortB A/D Enable	Bit 41	PORTB<4:0> configured as digital I/O on RESET
Low Power Timer1 Osc enable	Bit 42	Disabled
Master Clear Enable	Bit 47	MCLR Enabled,RE3 Disabled
0x300006	BYTE	0xf
Stack Overflow Reset	Bit 48	Enabled
Low Voltage Program	Bit 50	Enabled
Dedicated In-Circuit Port {ICD/ICSP}	Bit 53	Disabled
Extended CPU Enable	Bit 54	Disabled
Background Debug	Bit 55	Enabled
0x300007	BYTE	0xc0
0x300008	BYTE	0xf
Code Protect 00800-01FFF	Bit 64	Disabled
Code Protect 02000-03FFF	Bit 65	Disabled
Code Protect 04000-05FFF	Bit 66	Disabled
Code Protect 06000-07FFF	Bit 67	Disabled
0x300009	BYTE	0xe0
Data EE Read Protect	Bit 79	Disabled
Code Protect Boot	Bit 78	Disabled
0x30000a	BYTE	0xff
Table Write Protect 00800-01FFF	Bit 80	Disabled
Table Write Protect 02000-03FFF	Bit 81	Disabled
Table Write Protect 04000-05FFF	Bit 82	Disabled
Table Write Protect 06000-07FFF	Bit 83	Disabled
0x30000b	BYTE	0xff
Data EE Write Protect	Bit 95	Disabled
Table Write Protect Boot	Bit 94	Disabled
Config. Write Protect	Bit 93	Disabled
0x30000c	BYTE	0xff
Table Read Protect 00800-01FFF	Bit 96	Disabled
Table Read Protect 02000-03FFF	Bit 97	Disabled
Table Read Protect 04000-05FFF	Bit 98	Disabled
Table Read Protect 06000-07FFF	Bit 99	Disabled
0x30000d	BYTE	0xff
Table Read Protect Boot	Bit 110	Disabled

☑ Show descriptive names

12 The USB Assignments

With drivers and other resources such as Visual Basic programs and code generation tools, you can use the examples as a guide to each component, in conjunction with this manual. Alternatively you can try to produce your own version of each program from scratch, by following the tuition provided in this manual.

Some previous knowledge will be required to carry out the full range of example programs.

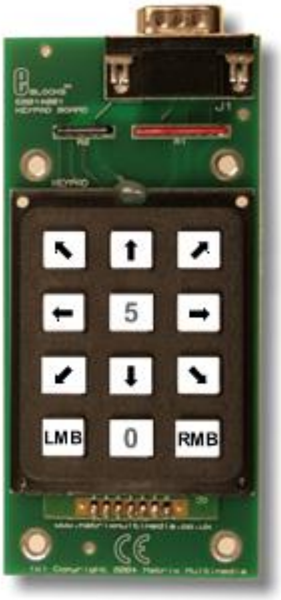
To get started with Flowcode, we recommend you follow the Flowcode course entitled ‘An Introduction to Microcontrollers’, which can be found on our website. www.matrixltd.com/

Here is an overview of the exercises covered in the solution.

- Human Interface Device: Mouse
    - Full USB Mouse using the HID Standard.
- Human Interface Device: Keyboard
    - Full USB Keypad using the HID Standard
- Human Interface Device: Data-Logger
    - Using the HID Standard to our advantage
- Communications Device: USB Terminal
    - Talking with the USB Serial device
- Communications Device: USB to RS232 converter
    - Using the USB device to convert to the serial protocol
- Slave Device: Basic Slave
    - Example of using the slave
- Slave Device: Storage Scope
    - Taking the slave example a little further
- Slave Device: Triggered Scope
    - Building more intelligence into the slave

13 Exercise 1 – Human Interface Device: Mouse

13.1 Introduction



The aim is to set up your very first USB device, a mouse simulated by the keypad E-Block. This has enough inputs to allow you to move the cursor around a screen and click the mouse buttons. The diagram shows which keys provide which mouse functions.

13.2 Objective

The objective of the exercise is to write a Flowcode program, to create a USB mouse, controlled from the keypad E-Block.

The keypad should be used as shown on the right with the numbers 1,2,3,4,6,7,8 and 9 being used to control the mouse cursor movement. The '✶' button should control the left mouse button (LMB) and the '#' button the right mouse button (RMB).

13.3 Target microcontroller

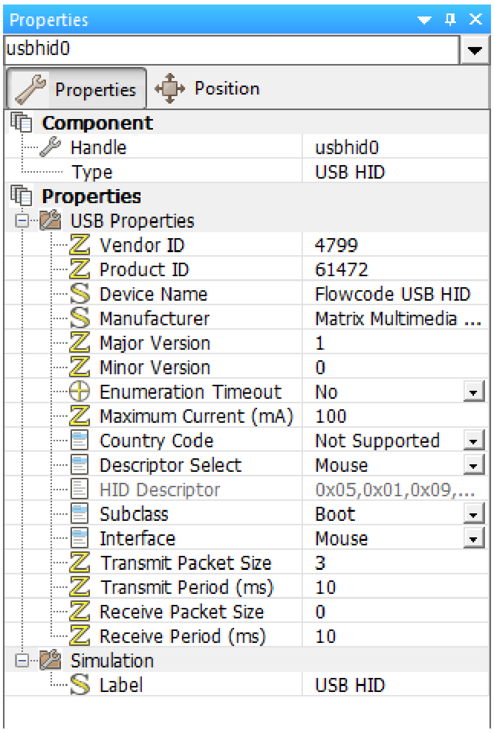
The target microcontroller is an 18F4455, which should be configured as shown in section 7 of this manual.

13.4 Flowcode USB HID component

The USB HID component has a number of associated macros. The first macro 'Initialise' sets up the USB component and then waits for the PC to install device drivers. Once this is completed, you get a return value stating the result of the initialisation. For more details please refer to the component help file.

13.5 USB HID component settings

The USB HID component properties should be configured as follows, in order to allow the mouse functionality to work.



Component	
Handle	usbhid0
Type	USB HID
Properties	
USB Properties	
Vendor ID	4799
Product ID	61472
Device Name	Flowcode USB HID
Manufacturer	Matrix Multimedia ...
Major Version	1
Minor Version	0
Enumeration Timeout	No
Maximum Current (mA)	100
Country Code	Not Supported
Descriptor Select	Mouse
HID Descriptor	0x05,0x01,0x09,...
Subclass	Boot
Interface	Mouse
Transmit Packet Size	3
Transmit Period (ms)	10
Receive Packet Size	0
Receive Period (ms)	10
Simulation	
Label	USB HID

The first half of the USB HID component properties includes the device parameters such as the identifier values and the name of the device, as well as additional information such as the manufacturer and version.

The second half of the properties configures the incoming and outgoing packet sizes, the communications period and the device country. Country code 0 stands for an international device.

The subclass and interface properties are only used in generic USB mice and keyboard devices.
The boot option makes the USB compatible with low level HID allowing you to use the device in BIOS or DOS. The descriptor select option can be set to mouse, keyboard or custom descriptor, allowing you to enter your own HID descriptions.

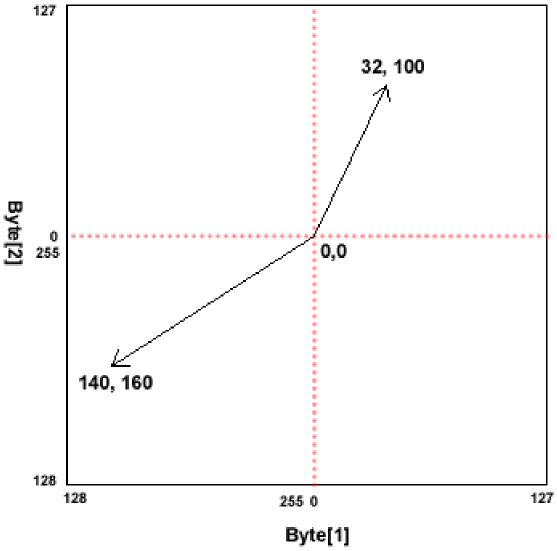
13.6 The Flowcode program in detail

The program will:

- initialise the USB device and wait for the computer to acknowledge;
- scan the keypad for a button press;
- and then:
  - convert the keypress into a valid mouse input;
  - send the data to the computer;
  - wait for the key to be released;
  - clear the mouse input.

13.7 A Generic USB Mouse

The USB mouse HID descriptor requires a 3-byte array to represent the mouse control vector. The first byte represents button presses where bit0 represents the left mouse button and bit1 represents the right mouse button. The other two bytes represent movement on the X and Y axis. The movement bytes use a signed format where any value over 127 represents a negative number. Therefore 0 = no movement, 1 to 127 represents a positive direction movement and 128 to 255 represents a negative direction movement. The value sent also defines the speed of the mouse movement so 1 would be a slow positive movement whereas 127 would be a fast positive movement. Likewise 255 would be a slow negative movement whereas 128 would be a fast negative movement. The position values represent the movement since the last sample so after a movement (e.g. when a key is released,) the control value is set back to 0. Otherwise the mouse cursor will continue to move in the current direction at the current speed.



13.8 What to do

- Create a string variable that is three bytes long to store the outgoing data.
- Add a USB HID component and configure the properties as shown above.
- Start the USB component by using a component macro to call the initialise function.
- Add a keypad component and then use a component macro to check for a key press.
- When a key press occurs use a switch case to assign the correct information to the USB outgoing data buffer.
- Send the buffer to the PC every 10 milliseconds until the key press is removed.
- Once the key press is removed clear the data buffer and resend to the computer.

13.9 Further Work

To take the example further try creating an “auto-fire mouse” that will perform a double click action by using the unused ‘5’ key.

Clicking and dragging is an important feature of an everyday mouse. Try holding the ‘✶’ key and moving the mouse cursor at the same time using the keypad. Can you explain the results?

A workaround to allow this functionality could be created by using the ‘0’ key for a toggled left button click action. For example, the first press could hold the left mouse button down and the next press brings it back up.

14 Exercise 2 – Human Interface Device: Keyboard

14.1 Introduction

The aim is to set up a generic USB keyboard. The input for the device will come from the keypad E-Block, allowing us to enter the ASCII characters 0-9, ‘✶’ and ‘#’.

14.2 Objective

The objective is to write a Flowcode program to create a USB keyboard controlled from the keypad E-Block. We will use the LED E-Block to read back the status of the toggle keys - caps lock, num lock and scroll lock.

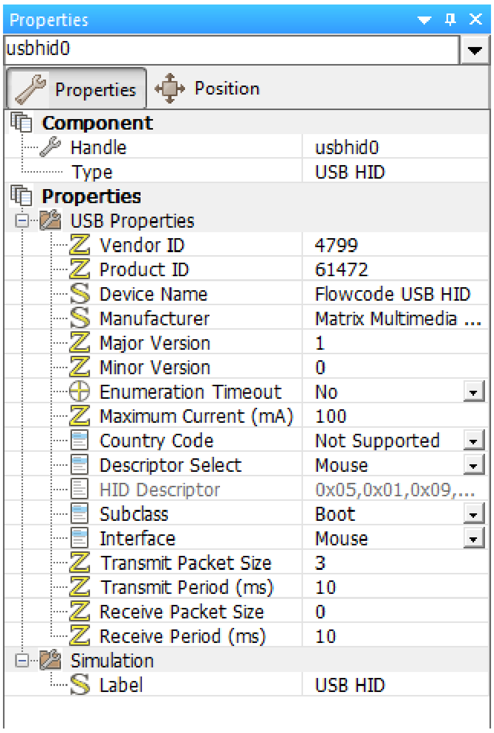
14.3 Flowcode USB HID component

Incoming data is received using the CheckRx component macro, which returns the number of bytes available and waiting to be read. A return value greater than 0 means that data is waiting. The data is read into Flowcode one byte at a time, using the ReceiveByte component macro. Addressing for this macro runs from 0 to (number of available bytes – 1).

For a standard USB keyboard, there is a single byte return value. We will use it to operate LEDs, to show operation of Num Lock (bit 0), Caps Lock (bit 1) and Scroll Lock (bit 2).

14.4 USB HID component settings

The USB HID component properties should be configured as follows, in order to allow the mouse functionality to work.



usbhid0

Component	
Handle	usbhid0
Type	USB HID
Properties	
USB Properties	
Vendor ID	4799
Product ID	61472
Device Name	Flowcode USB HID
Manufacturer	Matrix Multimedia ...
Major Version	1
Minor Version	0
Enumeration Timeout	No
Maximum Current (mA)	100
Country Code	Not Supported
Descriptor Select	Mouse
HID Descriptor	0x05,0x01,0x09,...
Subclass	Boot
Interface	Mouse
Transmit Packet Size	3
Transmit Period (ms)	10
Receive Packet Size	0
Receive Period (ms)	10
Simulation	
Label	USB HID

The first half of the USB HID component properties includes the device parameters such as the identifier values and the name of the device, as well as additional information such as the manufacturer and version.

The second half of the properties configures the incoming and outgoing packet sizes, the communications period and the device country. Country code 0 stands for an international device.

The subclass and interface properties are only used in generic USB mice and keyboard devices.
The boot option makes the USB compatible with low level HID allowing you to use the device in BIOS or DOS. The descriptor select option can be set to mouse, keyboard or custom descriptor, allowing you to enter your own HID descriptions.

14.5 The Flowcode program in detail

The program will:

- initialise the USB device and wait for the computer to acknowledge;
- check for an incoming byte;

- and when one arrives:
  - read the data byte and
  - forward the byte to the LEDs;
- scan the keypad for a button press;
- and when a button is pressed:
  - convert the keypress into a valid keyboard input;
  - send the data to the computer;
  - wait for the key to be released;
  - clear the mouse input;

14.6 A Generic USB Keyboard

The USB keyboard HID descriptor requires a 8-byte array to represent the keyboard control vector.

Bytes 0 and 1 in the array represent modifier keys - functions like Shift, Control and Alt.
The remaining six bytes are used for key press data.
This means that a maximum of up to 2 modifier keys and up to 6 data keys can be pressed at any one time.

Data Array	Data [0]	Data [1]	Data [2]	Data [3]	Data [4]	Data [5]	Data [6]	Data [7]
Key Press	Modifier1	Modifier2	Key1	Key2	Key3	Key4	Key5	Key6

Here is a list of commonly used key press data values:

Key Press	Data	Key Press	Data	Key Press	Data	Key Press	Data
A	0x04	N	0x11	Carriage Return	0x28	1	0x59
B	0x05	O	0x12	Tab	0x2B	2	0x5A
C	0x06	P	0x13	Caps Lock	0x39	3	0x5B
D	0x07	Q	0x14	Scroll Lock	0x47	4	0x5C
E	0x08	R	0x15	Num Lock	0x53	5	0x5D
F	0x09	S	0x16	/	0x54	6	0x5E
G	0x0A	T	0x17	*	0x55	7	0x5F
H	0x0B	U	0x18	-	0x56	8	0x60
I	0x0C	V	0x19	+	0x57	9	0x61
J	0x0D	W	0x1A			0	0x62
K	0x0E	X	0x1B			\	0x64
L	0x0F	Y	0x1C				
M	0x10	Z	0x1D				

And here is a list of the commonly used modifier key press values:

Key Press	Modifier
Left Control	0x01
Left Shift	0x02
Left Alt	0x04
Right Control	0x10
Right Shift	0x20
Right Alt	0x40

A full list of key press values and modifier codes can be found at.
www.win.tue.nl/~aeb/linux/kbd/scancodes-14.html

14.7 What to do

- Create a string variable that is eight bytes long to store the outgoing data.
- Add a USB HID component and configure the properties as shown above.
- Start the USB component by using a component macro to call the initialise function.
- Add a keypad component and then use a component macro to check for a key press.
- When a key press occurs use a switch case to assign the correct information to the USB outgoing data buffer.
- Send the buffer to the PC every 10 ms until the key press has been removed.
- Then clear the data buffer and resend to the computer.
- Check the incoming buffer for data, if data is available then forward to the LEDs.

14.8 Further Work

To take the example further try modifying the program so that when you press a key on the keypad you get a string of data output to the PC. This will involve sensing a key press and then performing several outgoing transactions to loop through and send each character in the string. A good use for this would be to create a shorthand keypad that has a list of common words or sentences built in so you do not have to type them manually.

15 Exercise 3 – Human Interface Device: Data-Logger

15.1 Introduction

The aim is to set up a basic data logging application using the keyboard created in the last exercise. We will sample an analogue sensor using an analogue-to-digital converter and then transfer this sample via USB.

15.2 Objective

The objective is to use the HID keyboard to send numerical data into the computer automatically. We will sample data from the light dependant resistor (LDR) embedded onto the Sensors E-Block board. The LDR is hard-wired to analogue channel 0 on pin 0 of Port A.

15.3 USB HID component settings

The USB HID component properties are configured as in the previous example.

15.4 The Flowcode program in detail

The program will:

- initialise the USB device and wait for the computer to acknowledge;
- wait for the num lock LED to be switched on;
- while the num lock LED is switched on
  - sample the voltage from the light dependant resistor;
  - convert the reading into an array of key press values;
  - send each key press in the array to the computer;
  - send a carriage return character to move the cursor to the next line;
  - reset the control vector and start the next sample.

15.5 Storing the data

A data-logger takes a large number of measurements, over a period of time, and stores them.

For this data-logger application we use Microsoft Excel to store the data. This offers simple numerical input, combined with an easy way to plot the resulting data.

To illustrate the process, select a data cell in Excel and use the numeric keypad to enter some data. Then hit the ‘Enter’ key and you will notice that the highlighted data cell moves down a row.

Therefore, we need the USB device to output each sample in the form of a decimal number. As the sample can be between 0 and 255, i.e. up to three digits long, this involves up to three emulated key presses to transmit the digits of the number.

To finish the process we send out a carriage return key press.

15.6 What to do

- Create a string variable that is eight bytes long to store the outgoing data.
- Add a USB HID component and configure the properties as in the previous example.
- Add an ADC component to your program and ensure it is connected to channel AN0.
- Start up the USB component by using a component macro to call the initialise function.
- Check for incoming data and forward it to LEDs.
- Test the Num Lock LED and if it is active do the following:
  - Use a component macro to sample the analogue channel and store the resulting value in a byte variable.
  - Convert this byte variable into a string variable using the string manipulation function “ToString”.
  - For each byte in the string, fill the IN data buffer with the appropriate data and send to the computer.
  - Delay for 10 milliseconds between each character that is sent.
  - Fill the IN data buffer with a carriage return and send to the computer.
  - Empty the IN data buffer and send to the computer so that it is aware that no keys are being pressed.

15.7 Further Work

To take the example further, try replicating the computer num lock control by using the keypad E-Block to enable or disable the num lock LED.

16 Exercise 4 – Communications Device: USB Terminal

16.1 Introduction

The aim is to create a connection to a computer using the USB Serial Flowcode component. You will create a basic terminal to send data from a computer to the LCD on the solution board. Equally, ASCII data, entered via the keypad, will be transmitted to the computer.

16.2 Objective

The objective is to write a Flowcode program to create a USB serial connection that can be used to stream data to and from the computer.

The keypad is read using the ASCII output component macro, so that the data is placed directly into the USB component macro for transmitting to the computer.

16.3 Flowcode USB Serial component

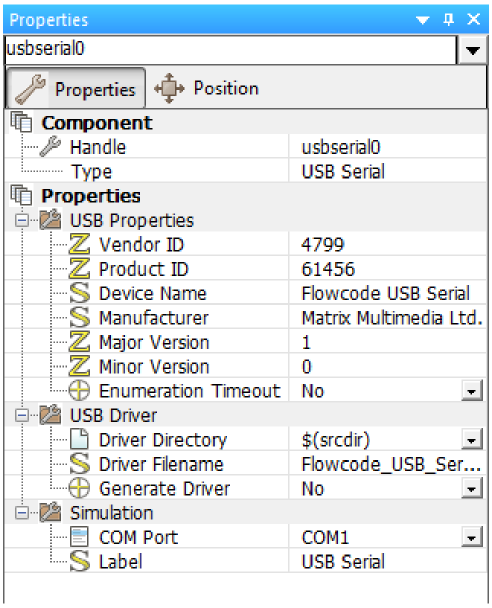
The USB Serial component has a number of associated macros. The first, called ‘Initialise’, sets up the USB component and waits for the PC to install device drivers. Once this is done, you receive a return value stating the result of the initialisation.

For more details on the functionality of the macros used by the USB_Serial component, please refer to the component help file.

To configure your computer for use with the USB serial peripheral you should refer to sections 5 and 6 of this manual.

16.4 USB Serial component settings

The USB Serial component properties should be configured as follows:



Component	
Handle	usbserial0
Type	USB Serial
Properties	
USB Properties	
Vendor ID	4799
Product ID	61456
Device Name	Flowcode USB Serial
Manufacturer	Matrix Multimedia Ltd.
Major Version	1
Minor Version	0
Enumeration Timeout	No
USB Driver	
Driver Directory	$(srcdir)
Driver Filename	Flowcode_USB_Ser...
Generate Driver	No
Simulation	
COM Port	COM1
Label	USB Serial

The USB Serial component properties allow you to configure the USB device parameters such as the identifier values and the name of the device, as well as the manufacturer and version. The Generate Driver property can be used to generate a driver file compatible with the device parameters you have configured.

16.5 The Flowcode program in detail

The program will:

- initialise the USB device and wait for the computer to send an acknowledgement;
- scan the keypad for a button press in ASCII format;
- and when one happens:
  - forward the key press to the computer;
  - wait for the key to be released;
- check for incoming data from the USB;
- and when it receives it:
  - print the data to the LCD using the Print ASCII function.

16.6 A Generic USB Serial Port

The USB Serial Flowcode component operates in a similar way to the RS232 Flowcode component. We can send data to the computer at any time by simply calling either the SendByte or SendString component macros. At the same time, we can also read data coming from the computer by using either the ReadByte or ReadString component macros. When reading incoming data in byte form a return value of 255 signifies that no new data has been received. If the value is less than 255 then we know that valid data has been received. The ReadString macro tries to read a string of a specific length and then return the entire string to Flowcode. If a timeout occurs before the data has been received, then the macro will return any data received before the timeout occurred. The data is stored in incoming and outgoing buffers, both 64 bytes in length, so that you can write and read data to and from the bus without having to worry about missing data.

16.7 What to do:

- Add a USB Serial component and configure the properties as shown above.
- Start up the USB component by using the Initialise component macro.
- Add a keypad component and then use a component macro to check for a key press.
- When a key press occurs, use a USB component macro to send the key press to the computer, and then wait for the key to be released.
- Check for incoming data from the computer.
- When any is detected, print it to the LCD using the PrintAscii component macro.

16.8 Further Work

The LCD display can only show two lines of 16 characters. Try formatting the output data so that, when you reach the end of the first line of text, you move automatically onto the second line. You could even add functionality to allow you to press the back space key on your computer to allow you to delete characters from the LCD print out.

17 Exercise 5 – Communications Device: USB to RS232 protocol bridge

17.1 Introduction

The aim is to show how to create a connection to a legacy RS232 COM port device.

Modern computer systems no longer come with a hardware COM port as standard so when a system requires a RS232 COM port connection, we have to find another way of interfacing the device. The USB Serial Flowcode component provides a means of doing just that. Moreover, it is fully compatible with the existing COM port software and hardware.

17.2 Objective

The objective is to write a Flowcode program to create a USB serial connection that can be used to stream data to and from a legacy serial device.

The RS232 E-Block can be used to convert signals from the microcontroller into +/– 12V signals used by the RS232 standard. If you are using a TTL level serial device, then you can bypass the RS232 E-Block board and connect the microcontroller's Rx and Tx pins directly to the serial device. If you do not have a serial device but have a built-in COM port on the motherboard, you can use multiple versions of HyperTerminal or RealTerm on the same computer to perform a loop-back action.

17.3 USB Serial component settings

The USB Serial component properties should be configured as shown in the previous example.

17.4 The Flowcode program in detail

The program will:

- initialise the USB device and wait for the computer to acknowledge;
- check for incoming data from the RS232;
- and when it is received:
  - forward the data to the computer via the USB link;
- check for incoming data from the USB;
- and when it is received:
  - forward the data via RS232;

17.5 USB to Serial Bridge

This method of switching data from one protocol to another is often referred to as protocol bridging.

When creating a protocol bridge, make sure that the data input rate is less than or equal to the data output rate. If a device receives data at 9600 bytes per second but only transmits at a rate of 4800 bytes per second, then the incoming data will start to pile up.

This limitation is often referred to as a bottleneck and should be avoided. A severe bottleneck can cause data going to the remote device to be missed, and this can cause entire systems to fail.

17.6 What to do

- Add a USB Serial component and configure the properties as shown in Exercise 4.
- Start up the USB component by using the Initialise component macro.
- Add a keypad component and then use a component macro to check for a key press.
- When one occurs, use a USB component macro to send the key press to the computer and then wait for the key to be released.
- Check for incoming data from the computer.
- When data is detected, print it to the LCD using the PrintAscii component macro.

17.7 Further Work

Use the LCD to print the data as it flows through the system. Data from the USB should go to the top line of the LCD and data from the RS232 should go to the bottom line.

Consider other popular protocols that could be bridged using this communications method. For example:

- SPI
- I^2C
- Bluetooth
- ZigBee
- Ethernet
- ….

18 Exercise 6 – Slave Device: Basic Slave Functionality

18.1 Introduction

The aim is to create a Flowcode program that will expose embedded functionality to the computer. This ranges from taking direct control of a peripheral to reading back the value of a variable or register in the embedded device.

The USB slave acts to hand over the control of a program to the computer while allowing intelligence to be retained inside the microcontroller. This reduces the amount of work the computer has to do, while allowing the microcontroller to hand over any low or high level functionality to the computer.

An example of this is the analogue sample. The computer sends a command, to tell the microcontroller to sample a specific analogue input channel. The microcontroller receives the request and controls the analogue to digital conversion. It retrieves the sample from the function registers and sends it to the computer. It stores, or processes, or displays it graphically, or whatever...

This method of sharing out the work of a system with intelligent peripherals is referred to as “distributed processing”.

18.2 Objective

The objective is to write a Flowcode program to create a USB slave device that can be used to perform the following:

- control the LEDs connected to Port E;
- control the LCD connected to Port B;
- sample an analogue input channel;
- control which analogue channel is sampled;
- scan the keypad for a key press.

Each of these functions uses the microcontroller to perform the low level data manipulation such as controlling the LCD or scanning the keypad array. The microcontroller passes relevant information to the computer, allowing it to perform higher level functionality on the data.

18.3 Flowcode USB Slave component

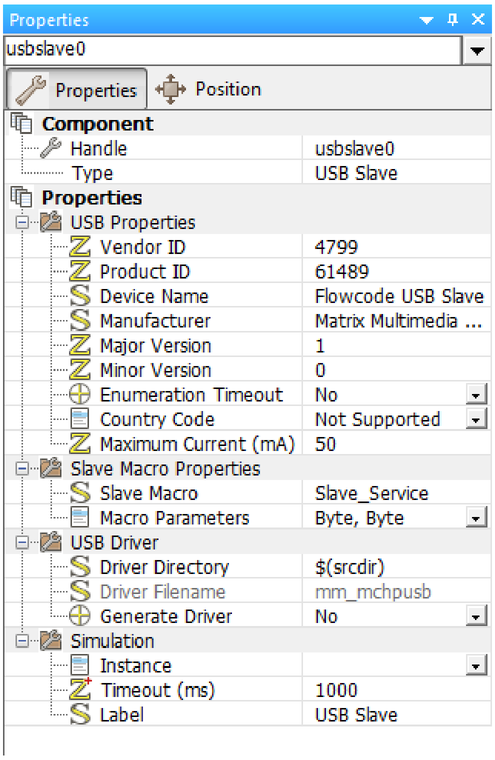
The USB Slave component has a number of associated macros, the first, entitled ‘Initialise’, sets up the USB component and then waits for the PC to install device drivers. Once this is done, it returns a value indicating the result of the initialisation. For more details on the macros used by the USB_Slave component, please refer to the component help file.

To configure your computer for use with the USB slave peripheral, refer to section 9 of this manual.

You will also require a copy of Microsoft Visual Basic installed on your computer. The Express Edition of Microsoft Visual Basic is fully compatible with the examples included in the solution and can be downloaded for free from the MSDN website.

18.4 USB Slave component settings

The USB Slave component properties should be configured as follows to allow the example Visual Basic code functionality to work correctly.



Properties

usbslave0

Component	
Handle	usbslave0
Type	USB Slave
Properties	
USB Properties	
Vendor ID	4799
Product ID	61489
Device Name	Flowcode USB Slave
Manufacturer	Matrix Multimedia ...
Major Version	1
Minor Version	0
Enumeration Timeout	No
Country Code	Not Supported
Maximum Current (mA)	50
Slave Macro Properties	
Slave Macro	Slave_Service
Macro Parameters	Byte, Byte
USB Driver	
Driver Directory	$(srcdir)
Driver Filename	mm_mchpusb
Generate Driver	No
Simulation	
Instance	
Timeout (ms)	1000
Label	USB Slave

The USB Properties section allows you to configure the USB device parameters such as identifier values and the name of the device. It also contains the properties which control the maximum current allowance for the device as well as the country code.

The Slave Macro Properties section determines which Flowcode macro you wish to use for your slave routine. The parameters should match with that of the Flowcode macro, in this case there are two ‘Byte’ parameters assigned to the macro.

Finally, the Generate Driver property can be used to generate a driver file compatible with the device parameters you have configured.

18.5 The Flowcode program in detail

The program will:

- initialise the USB device and wait for the computer to send an acknowledgement;
- start the USB slave service running;
- and when a transaction is received from the computer,:
  - run the slave service macro;
  - depending on the first input parameter, the macro will choose to:
    - output the value of the second input parameter to the LEDs;
    - output the value of the second input parameter to the LCD;
    - sample analogue channel 0 or 1 depending on the value of the second input parameter and then return the result;
    - sample the keypad and return the result.

18.6 A USB Slave DLL Transaction

The USB Slave Flowcode component makes the device sit and wait for input from the computer. When this is received, the Slave component runs a service macro. This handles the incoming transaction, in a manner similar to that used by an interrupt macro when an interrupt occurs.

The only restriction on the parameters used in the service macro is that they must correspond to the Slave Service Macro Parameters option chosen in the USB Slave Component Properties. The USB Slave component uses the macro parameters to pass the incoming data from the computer to the macro. In the example above, we use two char parameters. The first specifies the command and the second either the data to output or the analogue channel to sample.

If you need to return data from the Slave component, then you should do so within the service routine. You can return only one set of data per transaction, so if you need to transmit a large volume of data back to the computer, you must first create a string variable to store the data. You will also need to ensure that the timeout specified in the computer DLL call is enough to allow for the device to receive the command, perform a decision and then collect and return the data.

In the Visual Basic examples, this timeout is set to 500 milliseconds. If data is received within this time, the computer will immediately stop waiting and assume that the data it has received is valid and complete.

18.7 Driver Support Files

The driver file created by Flowcode is not enough on its own to allow you to install the USB Slave device. A set of driver support files are provided on the solution CD. Please copy these files to your computer and then use Flowcode to overwrite the .inf file to match the USB settings in your program. The driver support folder also contains .exe files which allow you to preinstall the USB Slave device driver onto an x86 or x64 based system.

18.8 What to do

- Add a USB Slave component and configure the properties as shown above.
- Start up the USB component with the 'Initialise component macro.
- Add a software macro and assign the macro with two byte parameters.
- Add a keypad component and two analogue components, connected to analogue channels AN0 and AN1.
- Add a switch case icon to the service macro to switch depending on the first byte parameter.
- In the switch case icon properties, tick the boxes next to options 1, 2, 3 and 4.
- For option 1, add an output icon to output the second parameter to Port E.
- For option 2, add a LCD component macro to output the second parameter to the LCD as ASCII characters.
- For option 3, sample ADC0 if the second parameter is a 0, otherwise sample ADC1. Then return the sample value, using the USB Slave SendByte component macro.
- For option 4, sample the keypad and again return the result to the computer using the USB Slave SendByte component macro.

18.9 Further Work

What other functionality could you add to the program to reduce the computer workload? Here are some examples of extra functionality you could add to the program.

- LCD Clear Command
- Output to any port
- Input from any port
- Reading a variable
- Modifying a variable
- Reading a register (requires C code)
- Modifying an internal register (requires C code)

19 Exercise 7 – Slave Device: Storage Scope

19.1 Introduction

The aim is to create a Flowcode program to enable a computer to perform a particular task. In this exercise, the task will be to sample and store an analogue voltage, using an analogue-to-digital converter component.

The computer will specify the sample rate and will start and stop the data sampling. To allow for a greater throughput of data we will customise the USB component to increase the size of the data buffer.

19.2 Objective

The objective is to write a Flowcode program to create a USB Slave device to perform the following functionality:

- change the rate of the timer interrupt;
- enable and disable the timer interrupt;
- retrieve a data array from memory;
- sample an analogue channel and store the result in data memory.

We will use component customisation to enhance the operation of the USB components.

19.3 The Flowcode program in detail

The program will:

- initialise the USB device and wait for the computer to send an acknowledgement;
- start the USB slave service running;
- and when a transaction is received from the computer:
  - run the slave service macro;
  - depending on the first input parameter, the macro will choose to:
    - 0x80: enable the Timer0 interrupt with Prescaler of 1:64;
    - 0x81: enable the Timer0 interrupt with Prescaler of 1:32;
    - 0x82: enable the Timer0 interrupt with Prescaler of 1:16;
    - 0x83: transmit the data array back to the computer;
    - 0x84: disable the timer interrupt
  - Inside the interrupt service macro the program will:
    - sample the analogue input pin
    - store the result into the data buffer
    - increment the current index of the data buffer

19.5 What to do

- Add a USB Slave component and configure the properties as shown above. This time change the Slave service macro so that it only has a single char parameter.
- Start up the USB component with the ‘Initialise component macro.
- Add a software macro, and assign it a single byte parameter.
- Add an analogue component connected to analogue channel AN0.
- Add a switch case icon to the service macro to switch depending on the byte parameter.
- In the switch case icon properties, tick the boxes next to options 1, 2, 3, 4 and 5.
- Update the switch case values to range from 0x80 to 0x84 to match the command values sent from the VB program.
- For options 0x80 to 0x82, add an interrupt icon to configure timer 0 with the prescaler settings mentioned earlier.

- For option 0x83, add a USB component macro to transmit the data buffer to the computer. Also, add a calculation icon to reset the buffer index value.
- For option 0x84, add an interrupt icon to disable the timer 0 interrupt.
- Inside the interrupt service macro, add a component macro to sample the analogue channel. Use the index variable to keep track of the current byte position in the data buffer.

19.6 Further Work

If you are using an XP based system, try changing the buffer variables and USB component definitions to allow for a transaction size of 128 bytes.

Add a second analogue component, and add a new set of commands to switch the channel of the analogue input source.

20 Exercise 8 – Slave Device: Triggered Scope

20.1 Introduction

The aim is to add extra functionality to the program created in exercise 7. This will allow the microcontroller to decide when to sample the inputs, leaving the computer free to process other tasks or applications.

20.2 Objective

The objective is to write a Flowcode program to create a USB Slave device to perform the following functionality:

- change the rate of a timer interrupt;
- enable and disable the timer interrupt;
- retrieve a data array from memory;
- monitor an analogue channel for a trigger event;
- and when the trigger event is detected:
    - sample the analogue channel;
    - store the result in data memory;
    - periodically send the data to the computer;
    - after sending 800 samples, reset the trigger event and repeat.

20.3 The Flowcode program in detail

The program is that produced in Exercise 7 with the following additions:

- when a transaction is received from the computer the program will:
    - run the slave service macro;
    - depending on the input parameter the macro will choose to:
        - 0x70: enable the Timer0 interrupt with Prescaler of 1:32.
        - 0x71: transmit the data array back to the computer.
        - 0x72: disable the timer interrupt.
        - 0x73: assign a new trigger threshold value.
        - 0x74: change the analogue channel that is being monitored.
    - Inside the interrupt service macro the program will:
        - sample the analogue input pin;
        - set a triggered variable when the analogue value is above the trigger threshold;
        - store the result into the data buffer if the triggered variable is set.

20.4 What to do

Add the trigger and channel selection code to the existing program 7. The command values in the Slave service routine must be changed to match the Visual Basic example for exercise 8. Remember that the example files are available, if you are unsure as to what is required.

20.5 Further work

Add functionality to allow for trigger events from different analogue channels. When a specific channel generates a trigger event, that channel should be sampled until the trigger event has been reset. This allows for intelligent channel switching.

Devise ways of transferring other functionality from the computer to the USB Slave component.

21 The USB C Code Library

The USB components in Flowcode are driven by means of the component C code file and the USB library. The C code file works like any other Flowcode C component control file using the functions available to Flowcode.

The USB library is accessible at the following location inside the default Flowcode directory:

C:\Program Files\ Flowcode 6\compilers\pic\boostc\include\USB

For every component, a pair of .c and .h files control the device class and a usb_config file controls the device descriptors. Further files handle the underlying generic USB functionality.