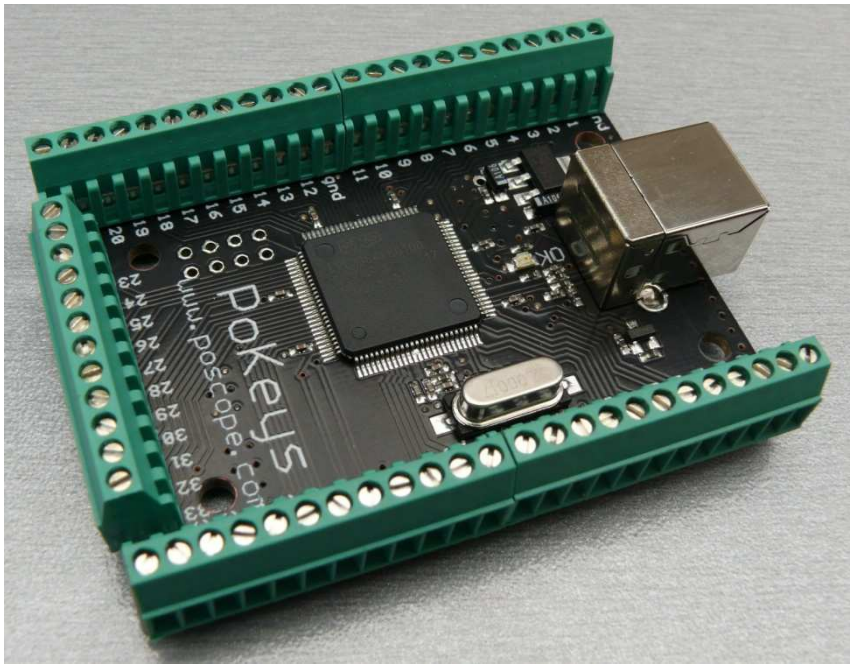
PoKeys 55

User's manual



PoKeys55 user's manual

1. Description

PoKeys55 is simple, easy-to-use USB device that combines standard keyboard and joystick simulation. PoKeys55 enables user to design specially built robust computer interface, comprising only the mechanical keys and PoKeys55 device. The device is highly adjustable and as such requires no complex knowledge on device programming.

If additional input and output capabilities are needed, the device provides 55 digital 5V tolerant inputs or outputs, 5 10-bit analog inputs and one 10-bit analog output. They are controlled via included software, which enables user either to use the highly intuitive graphical user interface or advanced console type interface. Chosen settings can be stored on device, so no special software is needed on target system.

NEW: From version 1.7 on, encoder support is available on any of the PoKeys55 inputs. User can freely choose two pins on which encoder A and B channel signals will be connected to. Configuration software allows assigning virtual keyboard keys separately for both directions.

There is also new capability to assign a special keyboard macro sequence instead of direct key mapping or encoder key mapping.

PoKeys devices are fully compatible with all new software provided.

NEW: PoKeys55 now supports matrix keyboards with the sizes of up to 8x8. User can freely adjust number of rows and columns from 1 to 8. Each key can have its own key code assigned to or mapped to a macro. High speed keyboard scanning allows even the fastest key presses to be detected. No additional external resistors or other circuitry is needed.

NEW: PoKeys55 now also supports 6 high-speed fully configurable PWM (pulse width modulation) outputs. User can freely set PWM period and PWM duties. PWM module runs at 12 MHz and allows high-speed switching.

2. Features

- Compatible with USB 1.1/2.0 HID standard
- Standard USB keyboard simulation
- Standard USB joystick simulation
- 55 digital inputs with pull-up resistors, freely mappable to virtual USB keyboard's keys
- 55 software controlled digital outputs
- 5 analog inputs (10-bit), freely mappable to any of virtual USB joystick axes
- 1 software controlled 10-bit analog output, controlled via included software
- Up to 25 encoder pair inputs
- Up to 64 256-character long keyboard macro sequences
- Intuitive and user-friendly software
- Up to 8x8 matrix keyboard support
- Up to 6 high-speed fully configurable PWM outputs support
- HD44780-based character LCD support (up to 4x20)

3. Requirements

- One available USB 1.1 or USB 2.0 port
- USB HID device driver enabled operating system (Windows 98 SE/ME/2000/XP/Vista, Linux, Mac OS)
- Included software requires Windows 2000/XP/Vista with .NET framework 2.0 installed (ONLY FOR SYSTEMS WHERE THE DEVICES WILL BE CONFIGURED, TARGET SYSTEM NEEDS NO SOFTWARE INSTALLATION FOR THE DEVICE TO OPERATE AS A STANDARD USB KEYBOARD AND JOYSTICK).

4. Installation

PoKeys55 is a USB 1.1/2.0 compliant device and as such requires no additional drivers for operation as a standard USB keyboard and joystick.

To operate the device after the device has been configured there is no software installation necessary on a target system.

To configure the device the supplied software must be installed and the requirements listed in previous section of this manual must be met.

5. Using PoKeys55 configuration application

PoKeys55 settings application is the utility for setting up the device for normal use. Upon starting the program, the main window (Figure 1) with connection dialog is displayed. If there are PoKeys55 devices detected to be attached to the system, the device selection box will be populated with all devices. To easily identify a specific device, the User ID number is appended to the default device name. To start editing device settings, click the 'Connect' button. After the current configuration is uploaded from the device, the user interface is enabled (Figure 2).

There is graphical representation for configuration of each PoKeys55 pin on left and right side of main window. To change pin function, click on pin name and change its function in central 'Pin settings' frame.

There are 5 main pin functions possible: inactive, digital input, digital output, analog input and analog output.

5.1. Inactive

Any pin can be set as inactive. Inactive pin is put in high-Z state with internal pull-up resistors enabled.

5.2. Digital input

Any one of the 55 pins can be configured as digital input by selecting 'Digital input' option box. All input pins have a weak pull-up resistor enabled and are 5V tollerant. If the pin polarity is inverted, check the 'Invert pin' box.

There are several additional possibilities for digital input pin functions.

5.2.1. Direct key mapping

Digital input set up for direct key mapping acts like a keyboard key. When there is a high state on pin (on low state when using inverted option) PoKeys55 sends a key associated with this pin. Select a keyboard key from drop-down box and check appropriate key modifiers (Shift, Ctrl, …).

Example:

Send Alt-F4: Select F4 from drop-down box and check Alt checkbox.

Send ((opening bracket): This key kombination differs from your system regional settings. As the PoKeys55 emulates a system keyboard, key associations depend on current sytem keyboard regional setting. To send an opening bracket symbol, one possible solution is to press Shift-8 (in most non-English countries) or to press Shift-9. Out of this reason there are no such secondary keys listed in drop-down box and must be entered by user as described above.

5.2.2. Keyboard macro

If there is a need for more than one key to be sent on pin activation, there is a possibility to assign a keyboard macro to a pin. Please see section Keyboard macros for more information on editing and assigning keyboard macros.

5.2.3. Encoder input

Rotational encoder switch can be used with PoKeys55 digital inputs. It is possible to connect up to 25 encoders to one PoKeys55 device. To enable encoder input, first select encoder index with numerical up-down selector, then select appropriate encoder channel. The last step is to check the box 'Encoder'.

Same as simple digital inputs, encoders can be assigned to direct key mapping or keyboard macro. This is possible for both directions (CW and CCW) separately. Simply set one mapping for channel A and another for channel B. To check for proper connection and settings, there is a special dialog that displays current encoder state. Please see section 'Displaying encoder RAW values'.

If needed, encoder inputs can be incremented or decremented 4x faster, therefore each complete step will produce increment or decrement of 4 sub-steps. Using this setting, higher precision can be obtained.

Encoder support should only be used for encoders that are hand-driven. It is not recommended to use encoders connected to driving axes of CNC milling machines.

5.2.4. Matrix keyboard column selection

Each digital input pin can be assigned as matrix keyboard column input. For additional matrix keyboard settings, see below.

5.3. Digital output

Any one of the 55 pins can be configured as digital output by selecting 'Digital output' option box. Each pin can sink or source up to 4 mA of current, with the limitation that the pins combined source or sink current does not exceed 100 mA. If the polarity of the pin is inverted, check the 'Invert pin' box.

5.3.1. Matrix keyboard row selection

Each digital output pin can be assigned as matrix keyboard row output. For additional matrix keyboard settings, see below.

5.4. Analog input

Analog input function is only available for pins 43 to 47. These analog inputs can also be freely mapped to any of the 5 joystick axes; X,Y, rotation X, rotation Y and throttle (Figure 3). To monitor current analog input value please see section Analog input values box.

5.5. Analog output

Analog output function is only available for pin 43. It is possible to set analog output voltage for this pin with 10-bit resolution. Console application or third party control application must be used to set a value for this pin!

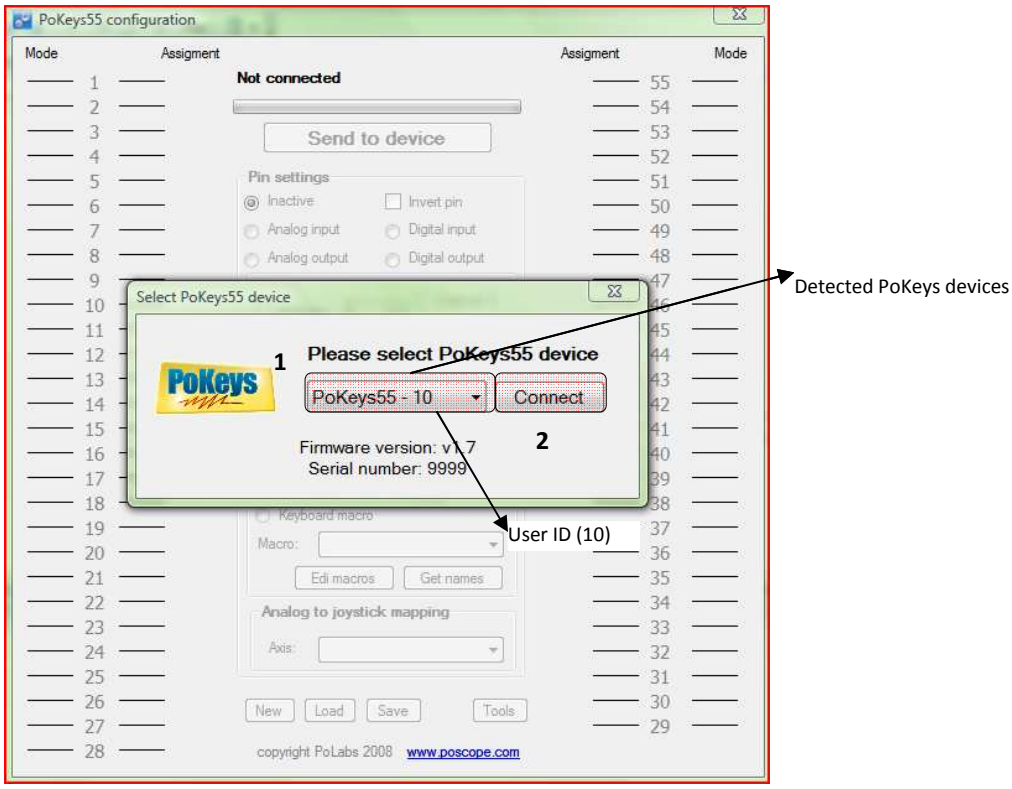


Figure 1: Main window

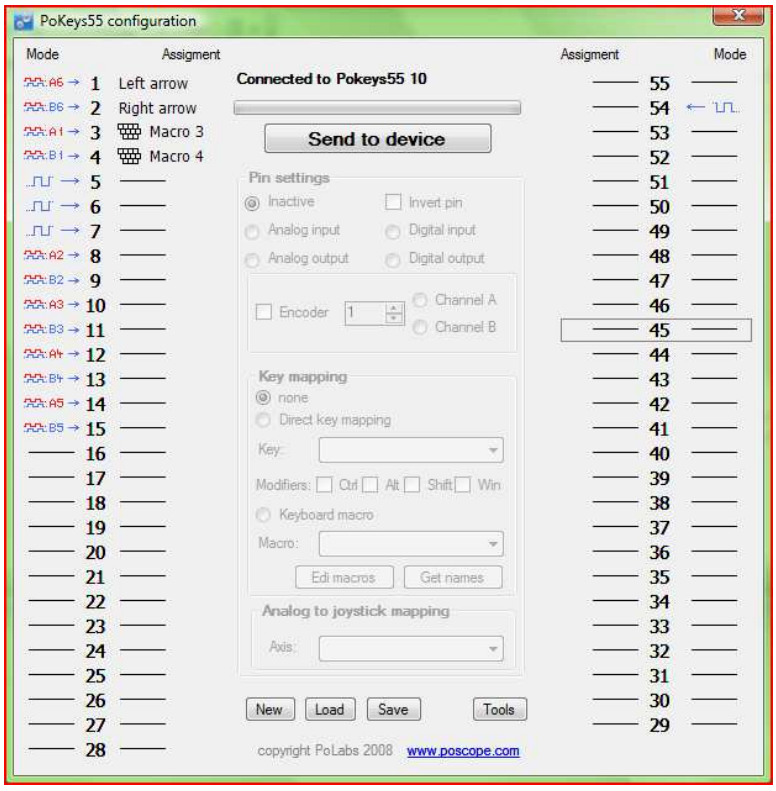


Figure 2: Connected to the PoKeys55 device with User ID 10

5.6. Keyboard macros

PoKeys55 device now supports keyboard macros – the key press combinations that can be up to 256 keys long. To define a keyboard macro, first select Keyboard macro mapping option for one of the pins. ‘Edit macros’ and ‘Get names’ command buttons become enabled. To add, change or delete macro, click the ‘Edit macros’ button. The following dialog appears

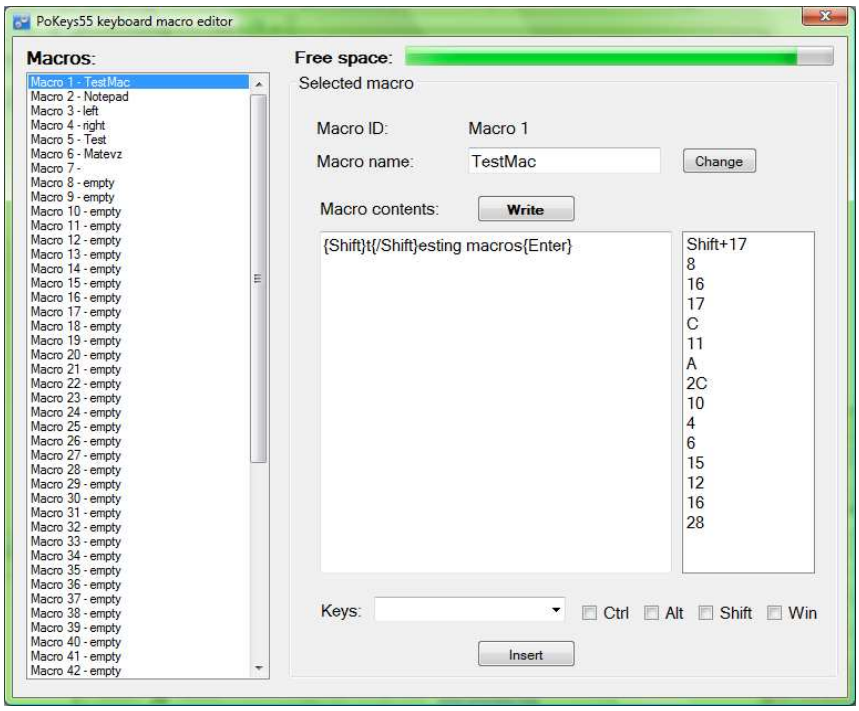


Figure 3: Macro editing dialog

First select the macro you want to edit. To change macro name, enter desired macro name (up to 7 characters long) in 'Macro name' text box and click 'Change' button. This name is used only to help user differentiate between multiple macros.

To set macro contents, simply enter text into 'Macro contents' text box. If there is an invalid character found, the text appears red. When finished, click Write to write macro to device.

List box at the right displays digital macro content.

5.7. Displaying encoder RAW values

To open encoder RAW values dialog, go to Peripherals menu and select ‘Encoder RAW values’. The following dialog below appears. It simply shows the list of all encoders and their current values.

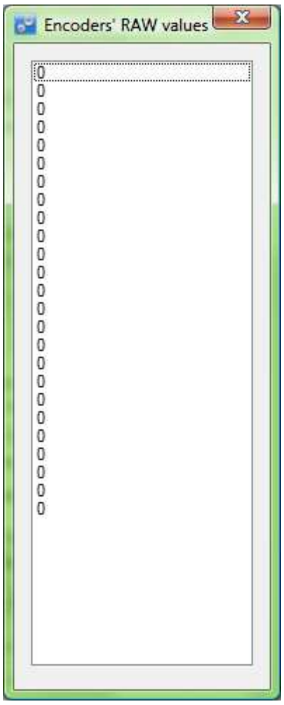


Figure 4: Encoders' RAW values

5.8. Analog inputs and outputs dialog

To open analog inputs dialog, go to Peripherals menu and select 'Analog inputs and outputs'. Dialog below appears. To enable display of analog input channel, check the appropriate check box. It is enabled only when the input is set up as analog input.

If pin 43 is set as analog output, analog value can be set for this pin.

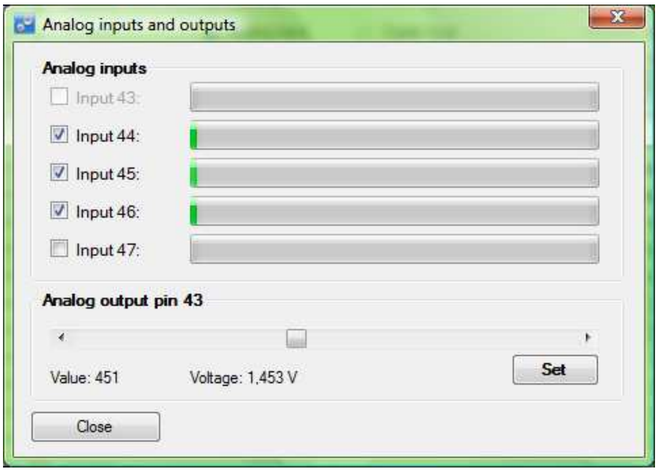


Figure 5: Analog inputs and outputs dialog

5.9. Matrix keyboard settings

To start using matrix keyboard, some insight is needed into working of a matrix keyboard. Matrix keyboard is a set of buttons, connected in a mesh. All buttons in a row share one contact, same goes for each of the buttons in the column. If a button is pressed, a key press is detected with a periodic

scanning of each of the rows and columns. PoKeys55 uses digital outputs for setting the voltage levels on rows and reads column voltage levels using digital inputs that already have internal pull-up resistors, so no external circuitry is needed.

PoKeys55 supports matrix keyboards of up to 8x8 in size, simpler 3x3, 4x3, 4x4 and others are of course fully supported.

Now, let us look how to set up a 4x3 keyboard. Open Peripherals > Matrix keyboard and set number of rows to 4 and number of columns to 3. Now click 'Enable matrix keyboard'. Matrix keyboard is schematically drawn below. Even lower, key mapping settings can be selected. Associate keys as it is needed in your application and close window. Now, select 3 pins and set them as digital inputs (column pins) and 4 pins as digital outputs (row pins). When finished, click Send to device button and start using your matrix keyboard.

Figure 6: Standard 4x3 matrix keyboard

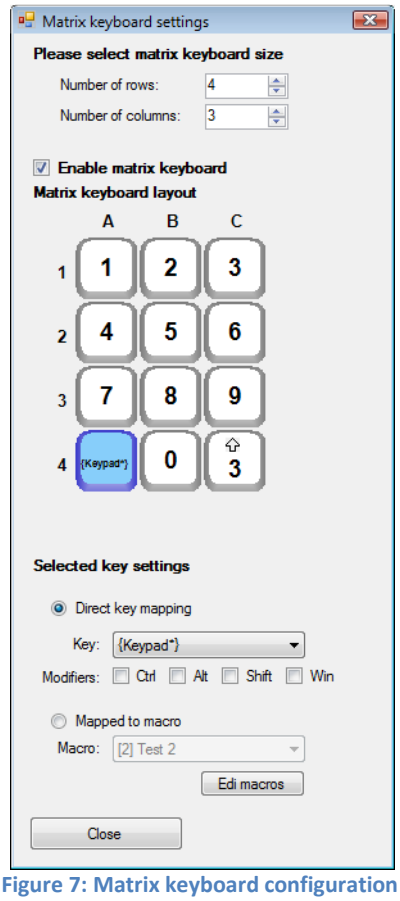


Figure 7: Matrix keyboard configuration

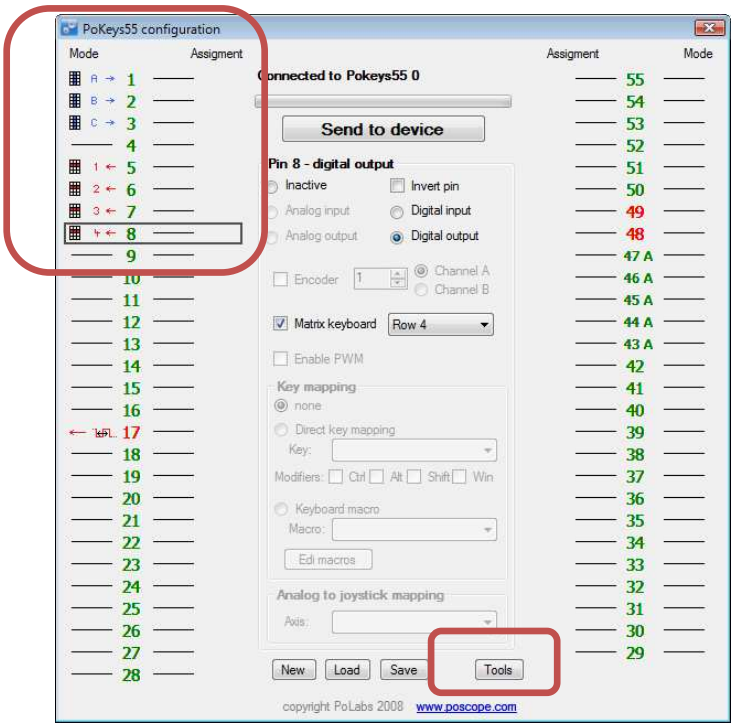


Figure 8: Assigning row and column pins

5.10. Using PWM module

PoKeys55's PWM (pulse width modulation) module can be set up via Peripherals > PWM outputs….

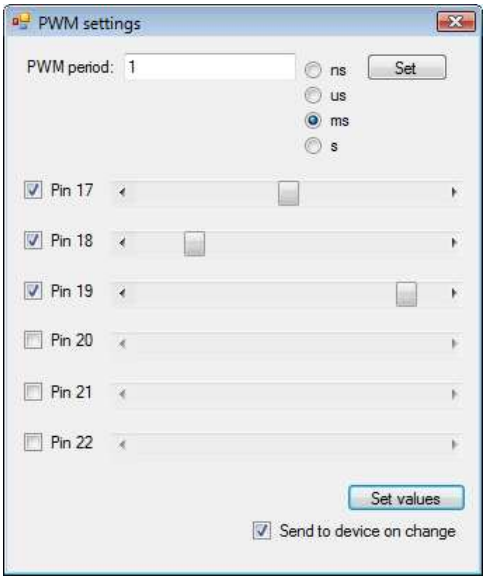


Figure 9: PWM outputs settings

In this window, user can enter PWM period and set PWM duties for each channel. Channels can be independantly enabled or disabled. After a change is made, user must click 'Set values' button or check 'Send to device on change' checkbox. Left position of a slider means 0% and right position 100% respectively.

5.11. HD44780-based character LCD

User can connect almost any widely available character LCD that is based on HD44780 or similar chipset.

Figure 10: Typical 2x16 character LCD

Usually these displays come in various sizes - 1/2/4 line with 8/16/20 characters and colors (black letters on green background, white letters on blue background ...).

These displays share standard pin-out that is listed in the table below:

Pin	Symbol	Function	PoKeys55 pin
1	**Vss**	Ground	**GND**
2	**Vdd**	Positive supply (usually 5V)[1]	**5V (usually) or 3.3V**
3	**Vo**	Contrast adjustment	**Variable resistor between GND and supply or PWM output**
4	**RS**	Instruction/data input	**Pin 29**
5	**R/W**	Read/write	**Pin 28**
6	**E**	Enable signal	**Pin 30**
7	DB0	Data bus – bit 0	Not connected
8	DB1	Data bus – bit 1	Not connected
9	DB2	Data bus – bit 2	Not connected
10	DB3	Data bus – bit 3	Not connected
11	**DB4**	Data bus – bit 4	**Pin 26**
12	**DB5**	Data bus – bit 5	**Pin 25**
13	**DB6**	Data bus – bit 6	**Pin 24**
14	**DB7**	Data bus – bit 7	**Pin 23**
15	Backlight (optional)		
16	Backlight (optional)		

LCD Display can be used to display various data. A third-party application or a script can execute all supported operations, including LCD initializing, clearing, moving cursor, setting display shifting mode, custom character defining and of course showing text. All this is available through easy-to-use PoKeysDevice DLL interface.

[1] Positive supply voltage depends on LCD used. User should find this information in datasheet of the LCD in use.

Functions of this interface can be tested through PoKeys55 settings application. Just open Peripherals > Test LCD… and dialog below will appear.

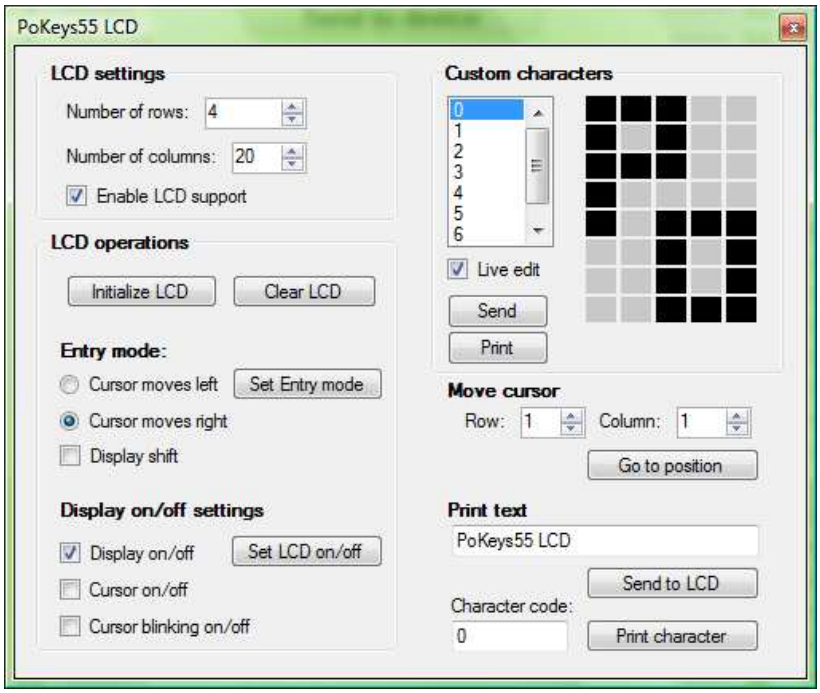


Figure 11: Character LCD testing dialog

5.11.1. LCD settings

In this part, user can set number of rows and columns in the LCD used. Support for LCD can be enabled or disabled also. When using LCD support enabled, it is advisable not to use pins designated for LCD use, unless there is a good reason for doing it.

5.11.2. LCD operations

Before user can start using the LCD, LCD module must be initialized. This is done via 'Initialize LCD' button. Button 'Clear LCD' clears LCD display and moves cursor to home position.

User can also set entry mode settings of LCD module. Cursor can be set-up to move either right (normally) or left after each character displayed. If 'Display' shift is enabled, whole display shifts with every new character displayed.

Settings are processed after user clicks button 'Set Entry mode'.

5.11.3. Display on/off settings

User can set on/off switches for whole display, cursor and cursor blinking.

Settings are processed after user clicks button 'Set LCD on/off'.

5.11.4. Custom characters

Simple interface enables to draw up to 8 custom characters. These characters can then be used on display. Selecting 'Live edit' mode will transfer the character each time a change is made to any of the pixels. Character can be previewed via button 'Print', which puts current custom character on the LCD display.

5.11.5. Move cursor

This section enabled user to move cursor to any position on the screen.

5.11.6. Print text

Sends entered text to display module. If advanced characters are needed, enter character code in lower text box and press 'Print character'.

5.12. Changing User ID number

Users can freely assign their own User ID number that represents a specific PoKeys55 device (enables distinguishing between different PoKeys55 devices in case there is more than one connected to a single host PC). To change the User ID number, go to ‘Device’ > ‘Change user ID’ menu. Simply enter any number between 0 and 255, and click the 'Change user ID' button.

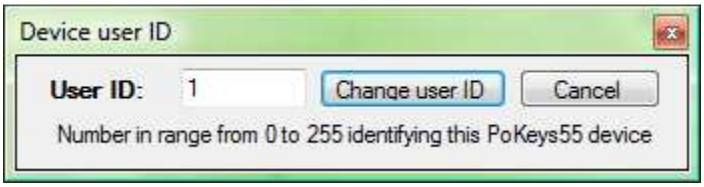


Figure 12: Device user ID dialog

5.13. Saving current configuration to file

To save the current configuration to a file, go to ‘File’ > 'Save' menu and select a new filename. To reload a saved configuration from a file, go to ‘File’ > ‘Open’ menu and select the appropriate file. To transfer new settings to the device, click on the ‘Save to device’ button.

5.14. Example: starting a program on Windows using PoKeys55 device

On a Windows operating system, users can assign a custom shortcut key to any program shortcut. Find the shortcut, then right click on it to show the context menu (Step 1). Select Properties (Step 2),

and under the Shortcut tab (Step 3), click on the 'Shortcut key' text box. Proceed by typing in a combination that you wish to assign to a particular program (Step 4). Next, open the PoKeys55 application and connect to the desired PoKeys55 device. Click on the pin that will function as a launch trigger for your application (Step 5). Under Key mapping, select the same keyboard combination that you assigned to the program shortcut (Step 6). Click on the 'Send to device' button (Step 7) to transfer settings to the device. This will activate the new shortcut.

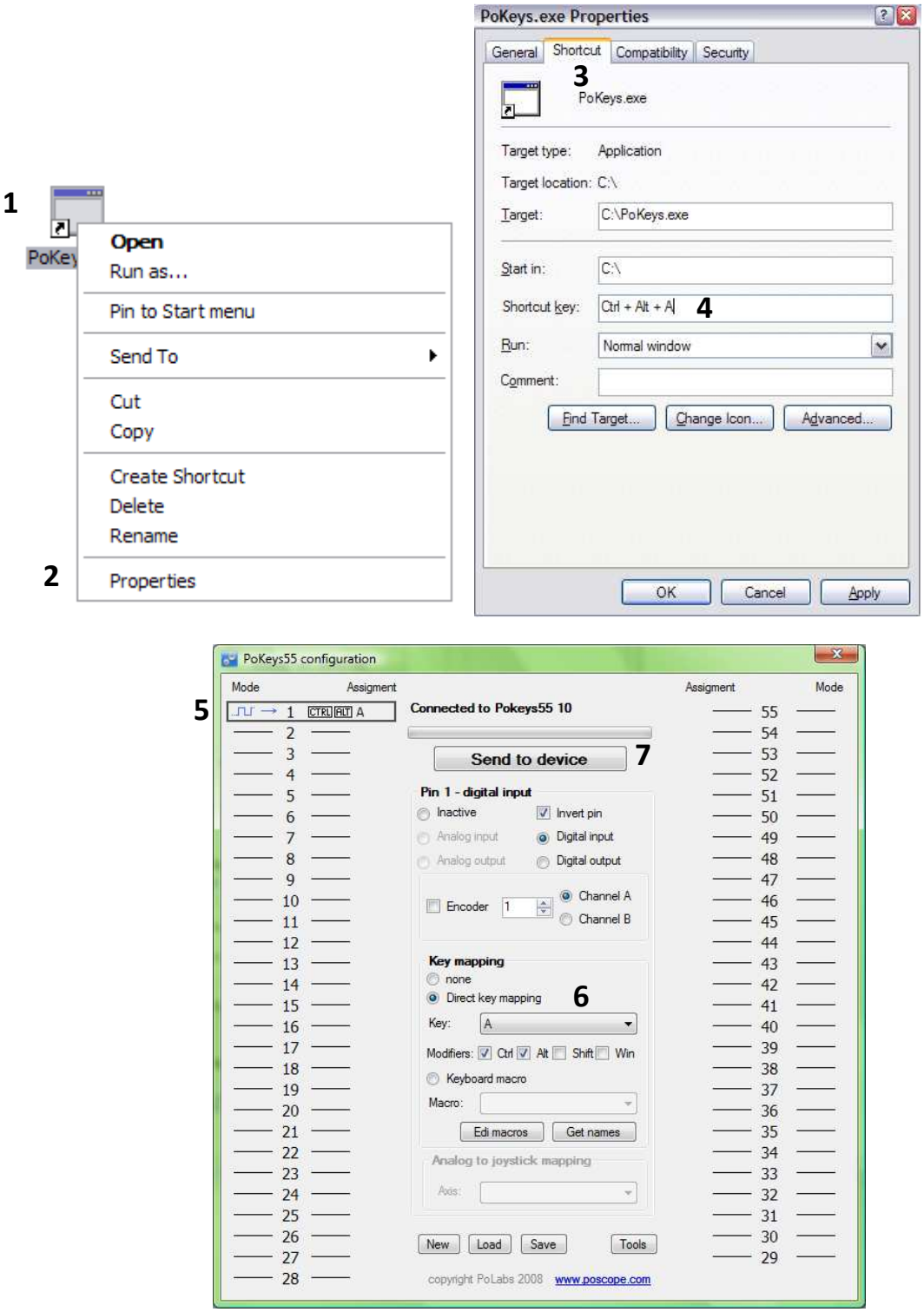


Figure 13: Setting up PoKeys55 device

6. Communicating with the device using console

In the software bundle included with the PoKeys55 device, there is also a console interface application, which enables command-line style communication with the device (figure 6). To start using the console application, go Start>Run…, type cmd and press Enter. Navigate to the folder, where PoKeysConsole.exe is located (usually C:\Program Files\PoLabs\).

6.1. Supported operations

Enumerate PoKeys55 devices

Command line: PoKeysConsole.exe -e

Description: Enumerates and prints out all the detected PoKeys55 devices with their User IDs.

Get details of the specific PoKeys55 device

Command line: PoKeysConsole.exe –d<user ID>

Description: Prints out the detailed description of the device, i.e. the device's serial number, firmware version and User ID.

Example: PoKeysConsole.exe –d1

Connect to PoKeys55 device

Command line: PoKeysConsole.exe –c<user ID>

Description: Before any operation can be executed, host software must connect to PoKeys55 device, using Connect to PoKeys55 device operation.

Example: PoKeysConsole.exe –c1

Save current configuration to flash memory

Command line: PoKeysConsole.exe –w

Description: After the settings have been changed, they need to be sent to the device. This is accomplished with via the Save configuration operation. 'Connect to PoKeys55 device' operation must be executed before this operation!

Example: PoKeysConsole.exe –c1 –w

Get current pin setting

Command line: PoKeysConsole.exe –g<pin ID>

Description: Prints out the current pin setting. If the pin ID parameter is omitted, settings for all the pins are printed out.

Example: PoKeysConsole.exe –c1 –g10

Set pin setting

Command line: PoKeysConsole.exe –s<pin ID>,<pin function>,+/-

Description: Enables the desired function on the selected pin.

Pin function	Value
Inactive	0
Digital input	2
Digital output	4
Analog input	8
Analog output	16

The last parameter is used to define polarity of digital input and output pins. It must be either + (non-inverted polarity) or - (inverted polarity).

Example: PoKeysConsole.exe –c1 –s10,2,-

Get digital input value

Command line: PoKeysConsole.exe –i<pin ID>

Description: Reads and prints out current digital input value on selected pin.

Example: PoKeysConsole.exe –c1 –i10

Set digital output value

Command line: PoKeysConsole.exe –o<pin ID>,0/1

Description: Sets the digital output to specified value.

Example: PoKeysConsole.exe –c1 –o11,1

Get analog input value

Command line: PoKeysConsole.exe –a<pin ID>

Description: Reads and prints out current analog input value on selected pin. Pin ID must be between 43 and 47, since only these pins support analog to digital conversion.

Example: PoKeysConsole.exe –c1 –a43

Set analog output value

Command line: PoKeysConsole.exe –b<pin ID>,value

Description: Sets the digital output to specified value. Value can be any number between 0 (0 V) and 1023 (3.3 V).

Example: PoKeysConsole.exe –c1 –b43,50

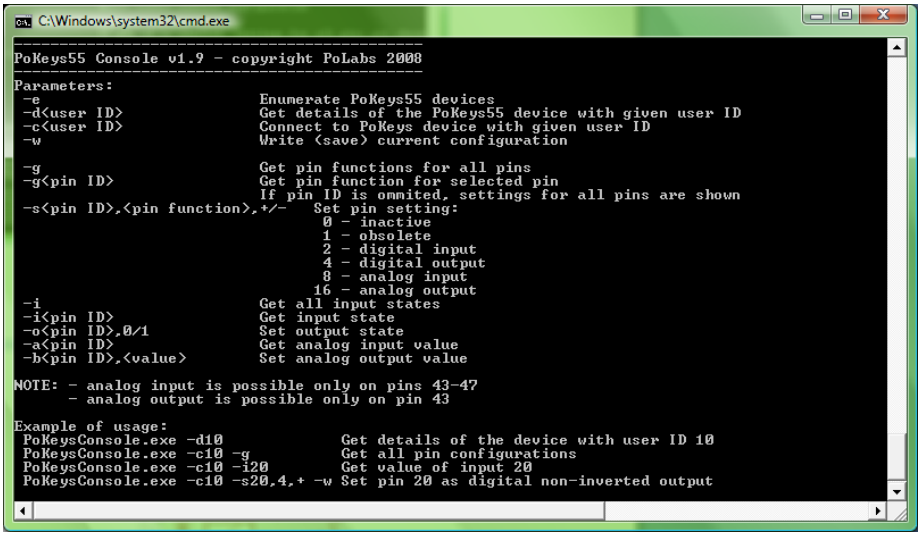
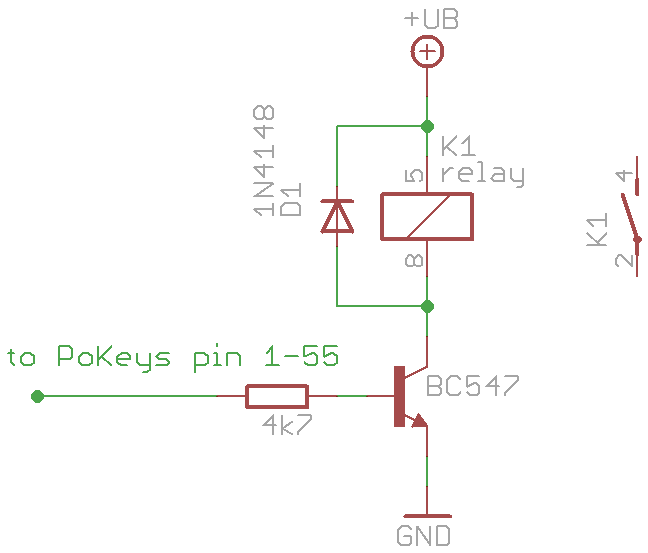


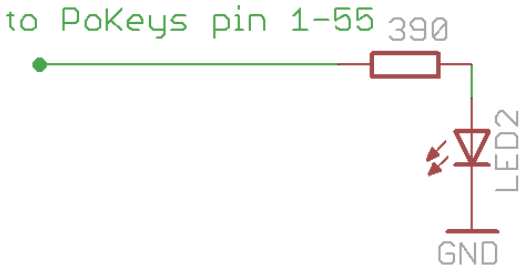
Figure 14: Console application window

7. Connecting common peripherals to PoKeys55 device

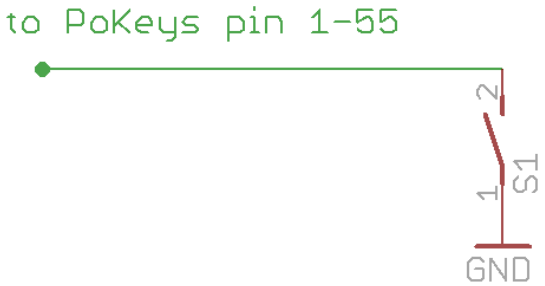
1. *Relays*



2. *LEDs*



3. *Switches*

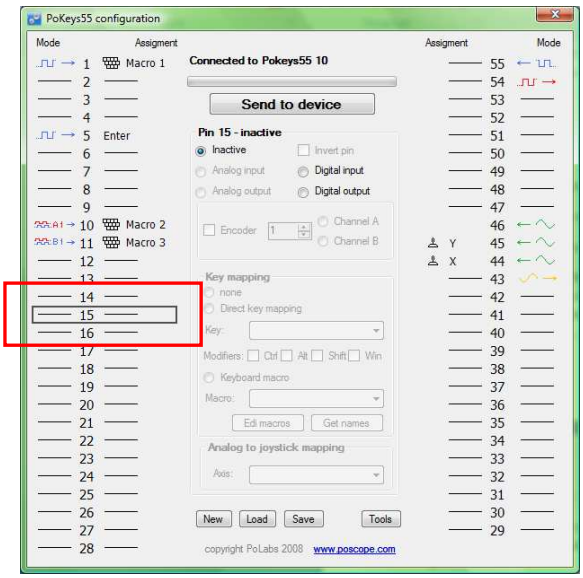


Example: Setting up key mapping

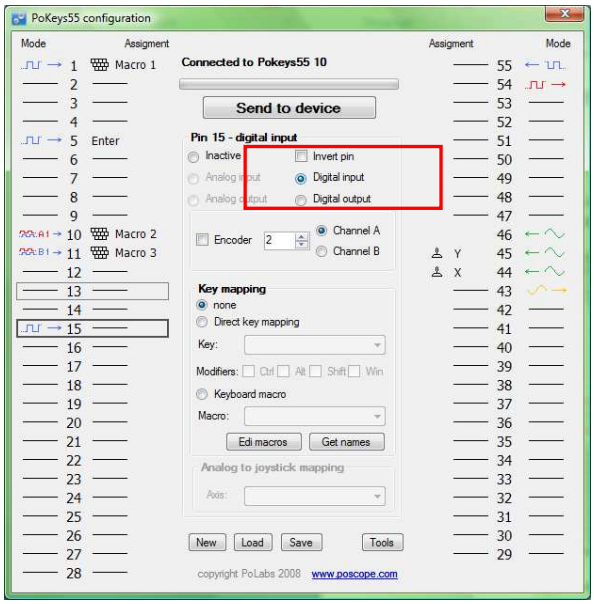
This example shows how easy is to set up a digital input pin for direct key mapping

We will set up a Shift-Escape combination for pin 15.

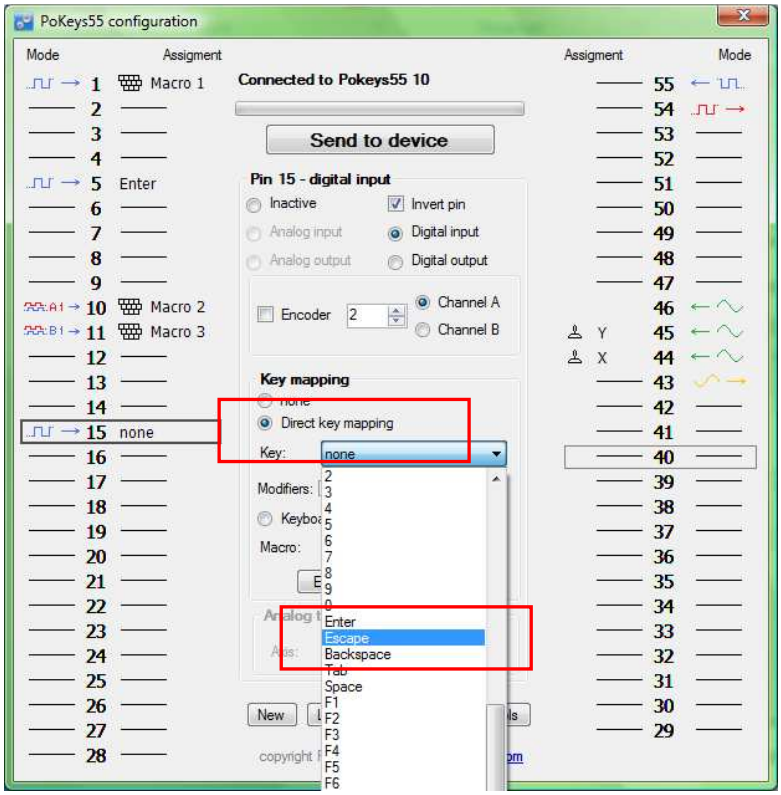
1. Connect a switch to PoKeys55 device as shown above
2. Open PoKeys55 configuration application
3. Select your PoKeys55 device from drop-down box and click ‘Connect’ button
4. Wait the application to load current configuration from PoKeys55 device
5. Click the same pin number as you connected a switch to (in this example pin 15)



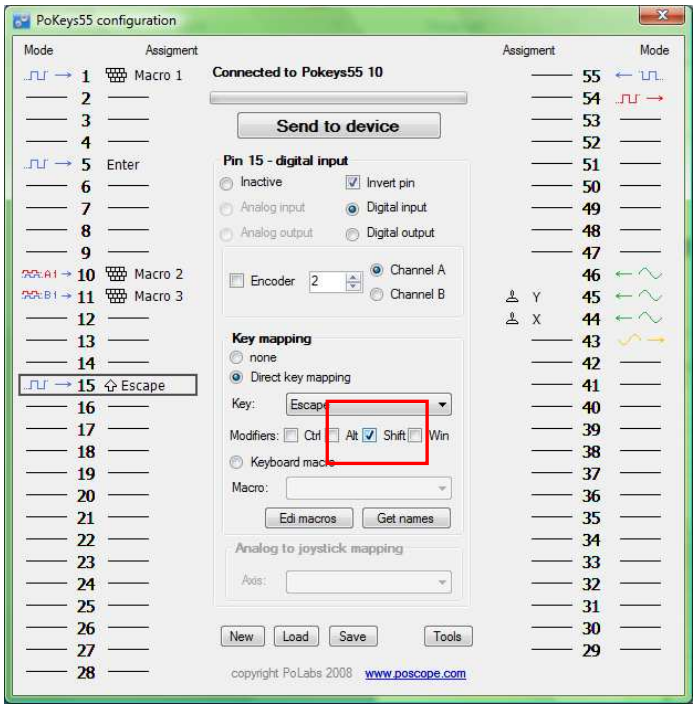
6. Set this pin as digital input



7. Select ‘Direct key mapping’ and from drop-down box select Escape

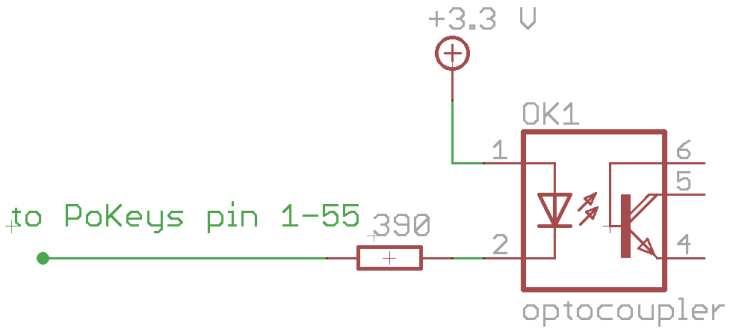


8. Click on the ‘Shift’ checkbox to enable Shift modifier

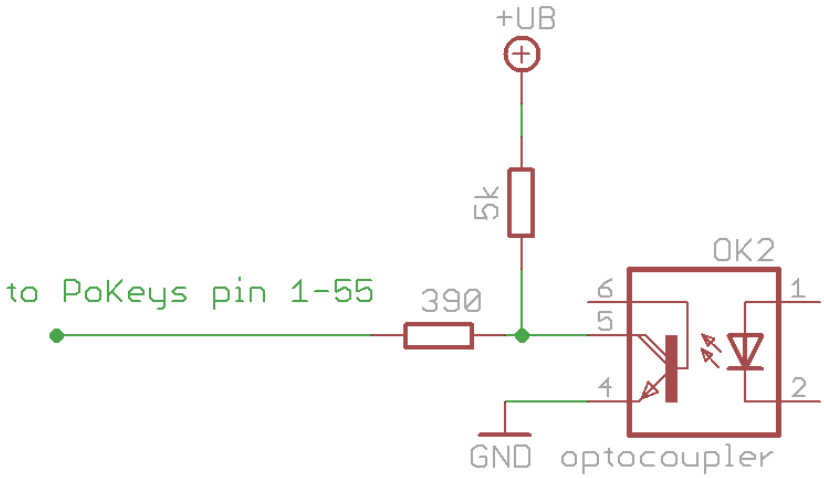


9. Send configuration to device by clicking ‘Send to device button’.

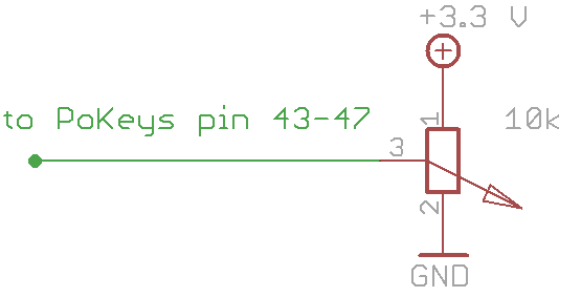
4. *Optocoupled digital output*



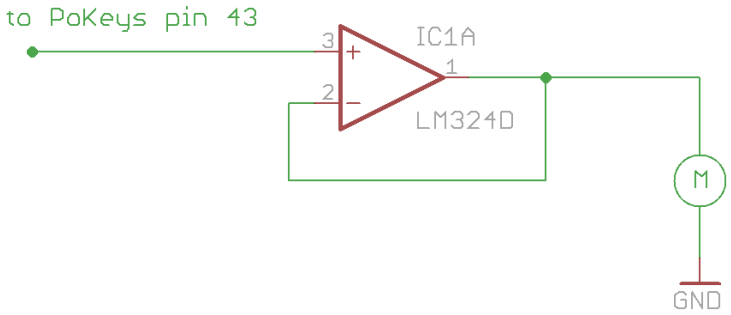
5. *Optocoupled digital input*



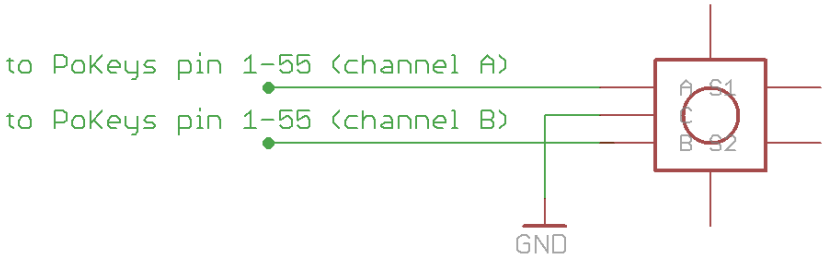
6. *Potentiometers (variable resistors)*



7. *Linear motor control*



8. *Rotational encoder switch*



8. Quick resetting the device configuration

If configuration editor cannot be used to reconfigure the device because of endless key presses from the device, simple reset procedure should be executed.

1. Disconnect PoKeys55 device from USB
2. Find pin labeled 'Reset' on the PoKeys55 device (otherwise use pin numbered 54)
3. Short this pin to ground (GND) and reconnect the PoKeys55 device to USB
4. Green light should start flashing rapidly
5. Wait approximately 5s, the light will stop flashing
6. PoKeys55 device will be reconnected with default settings

PoKeys55 configuration software is backing up current configuration state (except keyboard macro sequences) on each connection start.

These configuration files can be found in the local application folder (system folder – usually c:\Documents and settings\{username}\Local Settings\Application Data\PoKeys55\ on Windows 2000, XP or C:\Users\{username}\AppData\Local\PoKeys55\), named backup1.pkc, backup2.pkc and backup3.pkc with backup3.pkc being the oldest configuration.

9. Frequently asked questions

What software must be installed to operate the device?

On first use or when reconfiguring the device, the supplied software must be installed. There are no device drivers needed. They are already supplied with your operating system. Once the device has been configured, the settings are stored on-board. Device can then be freely used on any machine (see requirements for USB HID device driver enabled operating system) without any additional installation.

I misconfigured the device. Now the device starts pressing virtual keys before I can do anything.

If you misconfigured the device in such a way that configuration utility cannot be used to repair the configuration, see the section 'Quick resetting the device configuration'.

How do I connect switch/relay/LED/... to PoKeys55 device?

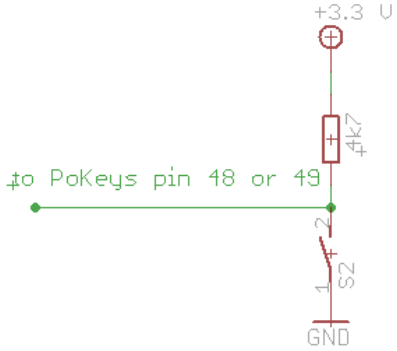
Please see the section 'Connection common peripherals to PoKeys55 device.'

I have two (or more) PoKeys55 devices connected on one system and cannot differentiate the devices to set the configurations.

It is advised that the users assign different UserID numbers to each of the device connected to a system. Please see the section '5.12 Changing User ID number'.

It appears that pins 48 and 49 are floating. What should I do?

Due to device design, pins 48 and 49 should be equipped with external 5-10 kΩ pull-up resistor as shown bellow.



I have an encoder connected to the pokeys55 and when I read the raw data, the number climbs to 255 and then loops back around. Shouldn't the number continue to increase (or decrease depending on which way I turn the shaft)?

RAW data is an 8-bit number and this is normal behavior and is named overflow. If you wish to use greater range, you should extend it in your program. Check the RAW data register periodically and calculate differences between states of the register at these periodic times in 8-bit space. Then use the difference calculated to increment or decrement your own 16-bit, 32-bit or wider register.

I have connected a switch to pin 4 and now PoKeys55 is not recognized by the computer any more.

You must have connected normally-closed switch to pin 4 and therefore connected pin 4 to ground. At boot (connecting PoKeys55 to USB) this means that PoKeys55 is entering system boot and therefore cannot be used from the computer. Please use another pin for normally-closed switches.

My problem is that when I start the program, everything looks good for about 3 to 5 seconds, then the CPU "lock up" and the only way to recover is to unplug the Pokeys55.

The problem occurs because you are writing 'dirty' code. You create object every time you need to use it in a loop, but you forget do properly dispose it. Best way to use PoKeys DLL in an application that read or writes data in a loop, is to create a global object and initialize it once at the start of application, and use its functions to read or write in a loop. This way the communication is much faster.

If you are using Visual Basic development environment, add a reference to PoKeysDevice DLL and use object browser to find proper declaration. Via object browser you can also access the list of all supported functions, which will also be used by Intelli sense in editor.

10. PoKeys55 library functions

EnumerateDevices

Enumerate the PoKeys devices and return number of found PoKeysDevices.

```
int EnumerateDevices()
```

Arguments:

none

Remarks:

This function must be called on every class initialization

ConnectToDevice

Connect to the device with the index specified.

```
bool ConnectToDevice(int deviceIndex)
```

Arguments:

deviceIndex

Index of the PoKeys device

Remarks:

Index is not UserID of the PoKeys device and therefore can change if more than one PoKeys device is used at a time. Function returns True if connection is established or False if there were errors.

DisconnectDevice

Terminate the connection with the device.

```
void DisconnectDevice()
```

Arguments:

none

Remarks:

This function should be called before class disposal or changing of the device.

GetDeviceID

Retrieve device ID data, i.e. serial number and firmware version.

```
bool GetDeviceID(ref int serialNumber, ref int firmwareVersion, ref int
pinNum)
```

Arguments:

serialNumber

Variable in which serial number will be saved to

firmwareVersion

Variable in which firmware version will be saved to

pinNum

Variable in which number of pins will be saved to

Remarks:

It is advised to use GetDeviceIDEx instead of this function. Returns False on error.

GetDeviceIDEx

Retrieve device ID data, i.e. serial number and firmware version.

```
bool GetDeviceIDEx(ref int serialNumber, ref int firmwareVersionMajor, ref
int firmwareVersionMinor)
```

Arguments:

serialNumber

Variable in which serial number will be saved to

firmwareVersionMajor

Variable in which firmware major version will be saved to

firmwareVersionMinor

Variable in which firmware minor version will be saved to

Remarks:

Returns False on error.

GetBuildDate

Retrieve firmware build date.

```
bool GetBuildDate(ref string buildDate)
```

Arguments:

buildDate

Variable in which build date will be saved to

Remarks:

Returns False on error.

GetUserID

Retrieve user ID.

```
bool GetUserID(ref byte userID)
```

Arguments:

userID

Variable in which user ID will be saved to

Remarks:

Returns False on error.

SetUserID

Set user ID.

```
bool SetUserID(byte newUserID)
```

Arguments:

newUserID

New user ID

Remarks:

It is advised that each PoKeys device on a system should have its unique user ID. Returns False on error.

SetPinData

Set pin data – pin's function and options.

```
bool SetPinData(byte pinID, byte pinFunction, byte pullUpDownResistor,
byte invertPin)
```

Arguments:

pinID

Pin ID is zero-based pin index on the device (output marked as 1 therefore has index 0)

pinFunction

pinFunction has the following structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin invert	reserved	reserved	A output	A input	D output	D input	reserved

where A stands for analog and D for digital.

pullUpDownResistor

No function at the moment

invertPin

No function at the moment, see bit 7 of pinFunction argument

Remarks:

It is advised to use function SetPinData with only 2 parameters (pinID and pinFunction). Returns False on error.

SetPinData

Set pin data – pin's function and options.

```
bool SetPinData(byte pinID, byte pinFunction)
```

Arguments:

pinID

Pin ID is zero-based pin index on the device (output marked as 1 therefore has index 0)

pinFunction

pinFunction has the following structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin invert	reserved	reserved	A output	A input	D output	D input	reserved

where A stands for analog and D for digital.

Remarks:

Returns False on error.

GetPinData

Get pin data – pin's function and options.

```
bool GetPinData(byte pinID, ref byte pinFunction, ref byte
pullUpDownResistor, ref byte invertPin, ref byte pinPossibleFunctions)
```

Arguments:

pinID

Pin ID is zero-based pin index on the device (output marked as 1 therefore has index 0)

pinFunction

pinFunction has the following structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin invert	reserved	reserved	A output	A input	D output	D input	reserved

where A stands for analog and D for digital.

pullUpDownResistor

No function at the moment

invertPin

No function at the moment, see bit 7 of pinFunction argument

pinPossibleFunctions

No function at the moment

Remarks:

It is advised to use function GetPinData with only 2 parameters (pinID and pinFunction). Returns False on error.

GetPinData

Get pin data – pin's function and options.

```
bool GetPinData(byte pinID, byte pinFunction)
```

Arguments:

pinID

Pin ID is zero-based pin index on the device (output marked as 1 therefore has index 0)

pinFunction

pinFunction has the following structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin invert	reserved	reserved	A output	A input	D output	D input	reserved

where A stands for analog and D for digital.

Remarks:

Returns False on error.

GetInput

Get digital input status.

```
bool GetInput(byte pinID, ref bool inputState)
```

Arguments:

pinID

Pin ID is zero-based pin index on the device (output marked as 1 therefore has index 0)

inputState

Variable in which input state will be saved to

Remarks:

Pin must be set as digital input for this command to function properly. Returns False on error.

SetOutput

Set digital output status.

```
bool SetOutput(byte pinID, bool outputState)
```

Arguments:

pinID

Pin ID is zero-based pin index on the device (output marked as 1 therefore has index 0)

outputState

New output state

Remarks:

Pin must be set as digital output before this operation will function properly. On device initialization, pins are NOT set as outputs even if they are configured so. Before using the pins as digital outputs SetPinFunction must be called to set the direction of the pin. This must be repeated on every startup. Returns False on error.

GetAnalogInput

Get analog input status.

```
bool GetAnalogInput(byte pinID, ref [int,byte] inputValue)
```

Arguments:

pinID

Pin ID is zero-based pin index on the device (output marked as 1 therefore has index 0)

inputValue

Variable in which analog value will be saved to. Use byte type for 8-bit analog values and int type for 10-bit analog values

Remarks:

The returned value is between 0 and 255 (8-bit resolution) or 0 and 1023 (10-bit resolution). 0 means 0 V on input, while 255 respectively 1023 means Vdd (approximately 3.3V) on input. Returns False on error.

SetAnalogOutput

Get analog input status.

```
bool SetAnalogOutputValue (byte pinID, [int,byte] value)
```

Arguments:

pinID

Pin ID is zero-based pin index on the device (output marked as 1 therefore has index 0)

value

New 8-bit or 10-bit value for analog output.

Remarks:

The value specified must be between 0 and 255 (8-bit resolution) or 0 and 1023 (10-bit resolution). 0 means 0 V on output, while 255 respectively 1023 means Vdd (approximately 3.3V) on output. Returns False on error.

SaveConfiguration

Saves current device configuration to non-volatile flash memory.

```
bool SaveConfiguration()
```

Arguments:

none

Remarks:

This function takes some time to complete. Is meantime, communication with the device is not possible. Returns False on error.

GetMatrixConfiguration

Get complete matrix keyboard configuration.

```
bool GetMatrixKeyboardConfiguration(ref byte configuration, ref byte width, ref byte height, ref byte[] row_pins, ref byte[] column_pins, ref bool[] macro_mapping, ref byte[] keycodes, ref byte[] keymodifiers);
```

Arguments:

configuration

If bit 0 is set, matrix keyboard is enabled. Other bits are reserved

width, height

Number of columns and rows of the matrix keyboard

row_pins

An array of 8 bytes, each having an index of a pin that is associated with the row. Row pins must be set as digital outputs.

column_pins

An array of 8 bytes, each having an index of a pin that is associated with the column. Column pins must be set as digital inputs.

macro_mapping

An array of 64 boolean values (see below for numbering hint). If the value is set to true, instead of key press simulation, macro is run.

keycodes

An array of 64 byte values (see below for numbering hint). If appropriate macro_mapping value is set to true, each value can contain index of a macro else it contains code of a key.

keymodifiers

An array of 64 byte values (see below for numbering hint). It contains key modifiers.

Keys indexing:

No matter what dimensions the matrix keyboard has, the following scheme is used for keys indexing. A1 is always 0, B1 1, A2 8, ... For example, if user connects a 3x3 matrix keyboard, keys have indexes: 0, 1, 2, 8, 9, 10, 16, 17, 18.

	A	B	C	D	E	F	G	H
1	0	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14	15
3	16	17	18	19	20	21	22	23
4	24	25	26	27	28	29	30	31
5	32	33	...					

Remarks:

Row pins must be set as digital outputs and column pins as digital inputs respectively. Returns False on error.

SetMatrixConfiguration

Set complete matrix keyboard configuration.

```
bool SetMatrixKeyboardConfiguration(ref byte configuration, ref byte width, ref byte height, ref byte[] row_pins, ref byte[] column_pins, ref bool[] macro_mapping, ref byte[] keycodes, ref byte[] keymodifiers);
```

Arguments:

configuration

If bit 0 is set, matrix keyboard is enabled. Other bits are reserved

width, height

Number of columns and rows of the matrix keyboard

row_pins

An array of 8 bytes, each having an index of a pin that is associated with the row.

column_pins

An array of 8 bytes, each having an index of a pin that is associated with the column.

macro_mapping

An array of 64 boolean values (see below for numbering hint). If the value is set to true, instead of key press simulation, macro is run.

keycodes

An array of 64 byte values (see below for numbering hint). If appropriate macro_mapping value is set to true, each value can contain index of a macro else it contains code of a key.

keymodifiers

An array of 64 byte values (see below for numbering hint). It contains key modifiers.

Keys indexing:

No matter what dimensions the matrix keyboard has, the following scheme is used for keys indexing. A1 is always 0, B1 1, A2 8, ... For example, if user connects a 3x3 matrix keyboard, keys have indexes: 0, 1, 2, 8, 9, 10, 16, 17, 18.

	A	B	C	D	E	F	G	H
1	0	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14	15
3	16	17	18	19	20	21	22	23
4	24	25	26	27	28	29	30	31
5	32	33	...					

Remarks:

Returns False on error.

GetPWMOutputs

Get complete PWM outputs configuration

```
bool GetPWMOutputs(ref bool[] channels, ref uint period, ref uint[] duty_values);
```

Arguments:

channels

An array of 6 boolean values, each representing one PWM channel. Channel is enabled if this value is set to true.

period

32-bit PWM period value. PWM module of a PoKeys55 device runs at 12 Mhz, so a value of 12 000 000 produces a period of 1 second.

duty_values

An array of 6 32-bit unsigned integers, each representing the value of PWM duty for each channel. Minimum value is 0, maximum value is the same as period.

Channel to pin mapping:

Channels are mapping according to this table:

Channel	PoKeys55 pin
0	22
1	21
2	20
3	19
4	18

5	17

Remarks:

When using PWM enabled pins, digital inputs and outputs are inactive. Returns False on error.

SetPWMOutputs

Set complete PWM outputs configuration

```
bool SetPWMOutputs(ref bool[] channels, ref uint period, ref uint[] duty_values);
```

Arguments:

channels

An array of 6 boolean values, each representing one PWM channel. Channel is enabled if this value is set to true.

period

32-bit PWM period value. PWM module of a PoKeys55 device runs at 12 Mhz, so a value of 12 000 000 produces a period of 1 second.

duty_values

An array of 6 32-bit unsigned integers, each representing the value of PWM duty for each channel. Minimum value is 0, maximum value is the same as period.

Channel to pin mapping:

Channels are mapping according to this table:

Channel	PoKeys55 pin
0	22
1	21
2	20
3	19
4	18
5	17

Remarks:

When using PWM enabled pins, digital inputs and outputs are inactive. Returns False on error.

LCDSetSettings

Set LCD settings.

```
bool LCDSetSettings(byte option, byte rows, byte cols)
```

Arguments:

option

Set to 1 to enable LCD support.

rows

number of rows on LCD module used.

cols

number of columns on LCD module used.

Remarks:

Enabling LCD support does NOT disable pins. Special care should be taken when simultanously using LCD designated pins for LCD display and digital inputs or outputs.

LCDInit

Initializes the LCD module.

```
bool LCDInit()
```

Arguments:

No arguments needed

Remarks:

No remarks.

LCDClear

Clears LCD display.

```
bool LCDClear()
```

Arguments:

No arguments needed

Remarks:

No remarks.

LCDGotoXY

Move cursor to position x (column), y (row)

```
bool LCDGotoXY(byte x, byte y)
```

Arguments:

x

Column

y

Row

Remarks:

x and y are 1-based addresses. Home position therefore is at 1,1.

LCDPutc

Displays one character on LCD module.

```
bool LCDPutc(char character)
```

Arguments:

character

Character code of character to be displayed

Remarks:

Character is displayed from at current position of the cursor.

LCDPrint

Displays string on LCD

```
bool LCDPrint(string LCDText)
```

Arguments:

LCDText

String containing up to 20 characters and \0 termination.

Remarks:

Text is displayed from the current position of the cursor.

LCDSetEntryMode

Set option for 'Entry mode'.

```
bool LCDSetEntryMode(byte CursorMoveDirection, byte DisplayShift)
```

Arguments:

CursorMoveDirection

0 for left-to-right movement, 1 for right-to-left movement

DisplayShift

0 for display shift off, 1 for display shift on

Remarks:

No remarks.

LCDDisplayOnOffControl

Set option for 'On/off control'.

```
bool LCDDisplayOnOffControl(byte DisplayOnOff, byte CursorOnOff, byte
CursorBlinkingOnOff)
```

Arguments:

DisplayOnOff

1 for display on.

CursorOnOff

1 for cursor display on.

CursorBlinkingOnOff

1 for cursor blinking on

Remarks:

No remarks.

LCDDefineCustomCharacter

Defines one custom character.

```
bool LCDDefineCustomCharacter(byte CharacterCode, byte[] characterData)
```

Arguments:

Character code

Character code in range from 0 to 7

characterData

8-byte array containing custom character data as specified in LCD module documentation.

Remarks:

No remarks.

11. Interfacing with PoKeys55 library – C# example

Preinitialization

C#

1. Add a reference to a PoKeysDevice_DLL.dll, located in installation folder
2. Use the class `PoKeysDevice` from the PoKeysDevice_DLL namespace

Visual Basic 6.0

1. Add a reference to a PoKeysDevice_DLL.tlb, located in installation folder
2. Use the class `PoKeysDevice_DLL.PoKeysDevice`

Class initialization

```
PoKeysDevice_DLL.PoKeysDevice cPoKeys = new PoKeysDevice_DLL.PoKeysDevice();
```

Enumerating the devices (this step must be taken even if we know exact device user ID!)

```
int iNumDevices = cPoKeys.EnumerateDevices();
```

The command returns the number of PoKeys devices detected on the system.

Getting device's serial number, user ID, firmware version and pin count:

```
int iSerialNumber = 0;
int iFirmwareVersion = 0;
int iPinNum = 0;
byte iUserID = 0;


for (int n = 0; n < iNumDevices; n++)
{
   cPoKeys.ConnectToDevice(n);
   cPoKeys.GetDeviceID(ref iSerialNumber, ref iFirmwareVersion, ref iPinNum);
   cPoKeys.GetUserID(ref iUserID);
   cPoKeys.DisconnectDevice();

   Console.WriteLine(n + ". device: Serial: " + iSerialNumber + "     Firmware: " +
iFirmwareVersion + "  User ID: " + iUserID);
}
```

Before any data can be read from or written to the device, the command ConnectToDevice must be executed. It's parameter is a device's index and not the userID! (therefore can be changed when multiple devices are connected at a time).

Reading pin configuration

```
byte iPinFunction = 0;

cPoKeys.GetPinData(0, ref iPinFunction);
```

In this example 0 (Pin 1) is used for a pin ID. Pin IDs are 0 based.

iPinFunction has the following structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Pin invert	reserved	reserved	A output	A input	D output	D input	reserved

where A stands for analog and D for digital.

Reading pin key mapping

Let us presume that pin 2 is defined as keyboard digital input with direct key mapping

```
byte iPinKey = 0;
byte iPinModifier = 0;
byte iMappingType = 0;

cPoKeys.GetPinKeyMapping(1, ref iMappingType, ref iPinKey, ref iPinModifier);
```

iPinKey is a key code as described in USB HID standard
iPinModifier is a modifier for a key (Ctrl, Alt, Shift, Win key) and can be used with these masks:

```
const byte CtrlMask = 1;
const byte ShiftMask = 2;
const byte AltMask = 4;
const byte WinMask = 8;
const byte AltGrMask = 64;
```

iMappingType has the following structure

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
reserved	reserved	reserved	reserved	reserved	macro	direct	enable

bit 0 – Enable key mapping: to enable key mapping for a specific pin, this bit must be set to 1
bit 1 – enable direct key mapping: when this bit is set to 1, pin actions are directly reflected as a keyboard key
bit 2 – enable keyboard macro mapping: when this bit is set to 1, special macro sequence is sent on pin activation

There can be only one of the bits 1 or 2 set!

Reading input value

Let us presume that pin 3 is configured as digital input.

```
bool bInputVal = false;
cPoKeys.GetInput(2, ref bInputVal);
Console.WriteLine("Input 3 is " + (bInputVal?"On":"Off"));
```

Joystick axis mapping

This is only possible on pins 43-47. If this is used on any other pin, the function will fail or be ignored.

```
cPoKeys.SetJoystickAxisMapping(42, iJoystickAxis);
```

iJoystickAxis can be used as follows:

```
0  None
1  Rx
2  Ry
3  X
4  Y
5  Throttle
```

Block read - digital

It is possible to poll 32 input pins with one request. All 55 pins can be read with two request joined in single command.

```
// Read pins 1 to 32
bool[] values_1_32 = new bool[32];

myDevice.BlockGetInput1(ref values_1_32);

// Read pins 33 to 55
bool[] values_33_55 = new bool[23];

myDevice.BlockGetInput2(ref values_33_55);

// Read all pins (1-55)
bool[] values = new bool[55];

myDevice.BlockGetInputAll55(ref values);
```

Block read - analog

It is possible to poll 4 8-bit or 3 10-bit analog inputs with one command.

```
// 8-bit mode
byte[] channels = { 42, 43, 0, 45 };
byte[] values = new byte[channels.Length];

myDevice.BlockGetAnalogInput8bit(ref channels, ref values);

byte value1 = values[0];
byte value2 = values[1];
byte value3 = values[3];

// 10-bit mode
byte[] channels = { 42, 43, 45};
int[] values = new int[channels.Length];

myDevice.BlockGetAnalogInput10bit(ref channels, ref values);

int value1 = values[0];
int value2 = values[1];
int value3 = values[2];
```

Block write - digital

It is possible to set 32 output pins with one request. All 55 pins can be set with two request joined in single command.

```
// Simple 8-bit binary counter
bool[] states = new bool[32];
```

```
for (int n = 0; n < 255; n++)
{
  for (int i = 0; i < 8; i++)
  {
    if ((n & (1 << i)) > 0) states [i] = false; else states [i] = true;
  }
  MyDevice.BlockSetOutput1(ref states); // Update pins 1 to 32
}
```

Reading encoder RAW values

RAW values from the encoder inputs can be read with following command.

```
byte iEncoderValue = 0;

cPoKeys.GetEncoderValue(1, ref iEncoderValue);
```

iEncoderValue is a value between 0 and 255.

Create new macro

This command creates macro in first free position. It returns macro index.

```
byte iMacroID = 0;
byte iMacroLen= 10;

cPoKeys.MacroCreate(iMacroLen, ref iMacroID);
```

Modify macro length

This command modifes macro length.

```
byte iMacroID = 0;
byte iMacroNewLen = 50;

cPoKeys.MacroModifyLength(iMacroID, iMacroNewLen);
```

Delete macro

This command deletes specific macro.

```
byte iMacroID = 0;

cPoKeys.MacroDelete(iMacroID);
```

Save macro configuration to flash

This command saves the current macro configuration to flash.

```
cPoKeys.MacroSaveConfiguration();
```

Change macro name

This command changes the macro name. Name property supports up to 7 characters.

```
byte iMacroID = 0;
cPoKeys.MacroSetName(iMacroID, "Macro1");
```

Set macro key

This command sets one macro key at the position iIndex. This index must be between 0 and iMacroLen - 1.

```
byte iIndex = 5;
byte iKeyCode = 10;
byte iKeyModifier = 0;

cPoKeys.MacroSetKey(iMacroID, iIndex, iKeyCode, iKeyModifier);
```

Get free space for macros

Space for saving macros is limited. To find out how much free space exists, use the following command.

```
int iFreeSpace = 0;

cPoKeys.MacroGetFreeSpace(ref iFreeSpace);
```

Get the list of macros' states

If the macro has the length of 0 it is designated as inactive. This command retrieves the list of states for all the macros. If specific macro is active, bActiveMacros has the value True.

```
bool[] bActiveMacros = new bool[64];

cPoKeys.MacroGetActiveMacros(ref bActiveMacros);
```

Display current time on LCD

```
// Initialize library
PoKeysDevice_DLL.PoKeysDevice dev = new PoKeysDevice_DLL.PoKeysDevice();

// Enumerate devices and connect to first (we have only 1 connected)
dev.EnumerateDevices();
dev.ConnectToDevice(0);

// Set settings for 4x20 LCD
dev.LCDSetSettings(1, 4, 20);
// Initialize LCD
dev.LCDInit();
// Clear LCD
dev.LCDClear();
// Move cursor to home
dev.LCDGotoXY(1, 1);
// Print Hello, world!
dev.LCDPrint("Hello, world!");

// 1s delay
System.Threading.Thread.Sleep(1000);

while (true)
{
  Application.DoEvents();
  System.Threading.Thread.Sleep(100);

  // Move cursor to home
  dev.LCDGotoXY(1, 1);
  // Print current date and time
  dev.LCDPrint(DateTime.Now.ToString());
}
```

12. Major changes from 1.x to 1.7:

To move the PoKeys55 device to a new level some major changes to interface were imminent.

Pi n function 1 was removed (this was directly key mapped pin function). Instead, key mapping functionality was added to any digital input pin. Key mapping type (none/direct/macro) can be set via changed Key mapping command as shown in above example.

12.1. Pin 13 not functioning appropriately

On PoKeys55 devices with serial numbers greater than 10133 there is a flawed connection for pin 13. Please do not use this pin.

12.2. Putting pin 4 low on startup disables PoKeys55 device from booting.

Avoid connecting switches that can be closed on startup to this pin.

13. Grant of license

The material contained in this release is licensed, not sold. PoLabs grants a license to the person who installs this software, subject to the conditions listed below.

1. Access

The licensee agrees to allow access to this software only to persons who have been informed of and agree to abide by these conditions.

2. Usage

The software in this release is for use only with PoLabs products or with data collected using PoLabs products.

3. Copyright

PoLabs claims the copyright of, and retains the rights to, all material (software, documents etc) contained in this release. You may copy and distribute the entire release in its original state, but must not copy individual items within the release other than for backup purposes.

4. Liability

PoLabs and its agents shall not be liable for any loss or damage, howsoever caused, related to the use of PoLabs equipment or software, unless excluded by statute.

5. Fitness for purpose

No two applications are the same, so PoLabs cannot guarantee that its equipment or software is suitable for a given application. It is therefore the user's responsibility to ensure that the product is suitable for the user's application.

6. Mission Critical applications

Because the software runs on a computer that may be running other software products, and may be subject to interference from these other products, this license specifically excludes usage in 'mission critical' applications, for example life support systems.

7. Viruses

This software was continuously monitored for viruses during production, however the user is responsible for virus checking the software once it is installed.

8. Support

No software is ever error-free, but if you are unsatisfied with the performance of this software, please contact our technical support staff, who will try to fix the problem within a reasonable time.

9. Upgrades

We provide upgrades, free of charge, from our web site at www.poscope.com. We reserve the right to charge for updates or replacements sent out on physical media.

10. Trademarks

Windows is a registered trademark of Microsoft Corporation. PoKeys, PoKeys55, PoScope, PoLabs and others are internationally registered trade marks.

support: www.poscope.com

14. PoKeys55 dimensions

If that picture is printed with no scaling, it can be used as a marking tool for holes.

All holes are 3mm.