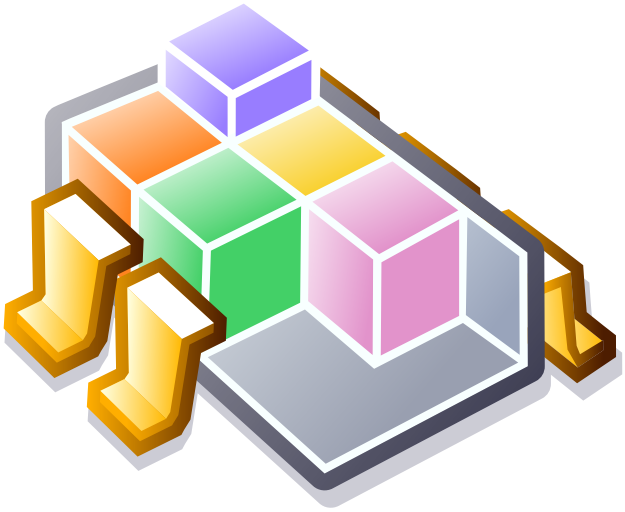
The KTechlab Handbook

David Saxton
Daniel Clarke



Contents

Abstract

KTechlab is an IDE for microcontrollers and electronics.

Chapter 1

Quick Tour

1.1 Introduction

KTechlab is an IDE for electronic circuits and microcontrollers. It can perform simulation a variety of components (logic, integrated, linear, nonlinear and reactive), simulation and debugging of PIC microcontrollers via gpsim, and comes with its own closely-linked and complementary high level languages: FlowCode and Microbe.

It has been designed to be as easy to use and unintrusive as possible; all components and FlowParts have context sensitive help, and simulating electronics is as simple as dragging components onto the work area and creating connectors that autoroute themselves between their pins. FlowCode allows users new to PICs to instantly create their own programs, while the electronic simulation allows stepping through a PIC's assembly program inside a circuit.

1.2 Documents

To get started in KTechlab, you will need to create a new document, whose type will depend on your task:

- FlowCode Document - Construct a PIC program via flowcharting.
- Circuit Document - Simulate electronics circuits and microcontrollers.
- Microbe Document - High level language for PICs, also used by FlowCode to generate assembly.
- Assembly Document - Start writing a PIC assembly program.

KTechlab uses a Document-View model, in that the Document logic is completely separate from open views of the document. This allows several views of the same file.

On creating a new document, the view is created in a separate tab. Each tab can support any number of views, tiled in any arbitrary pattern. This allows, for example, simulating a PIC program in circuit, while stepping through the program in an assembly document in the same tab.

The contents of tabs can be duplicated by dragging the tab to an empty area on the tab bar. They can be inserted into an existing tab by dragging it onto that tab.

Detailed instructions on the above documents can be found in their own respective chapters.

1.3 Drawing

In Circuit and FlowCode documents, there are several drawing tools available, including text. These are available by clicking on the paintbrush icon in the toolbar. To draw, drag the mouse to form either a shape or a line appropriate to the drawing tool in use.

When a drawing is selected, it can be resized by dragging its handles. Holding down **Shift** while dragging will snap the handle to the underlying grid. Each tool has basic options accessible from the toolbar, such as colors. There are also more advanced options found in the **Item Editor** sidebar, such as line and cap styles.

Chapter 2

PIC Programs

2.1 Manipulation

When you create a FlowCode or a Text document, you'll notice a drop down menu in the toolbar with a rocket icon. From here, you can manipulate your PIC program; changing it to different forms.

- **Convert to Microbe** - This is used only in FlowCode documents. This is explained further in chapter 4.
- **Convert to Assembly** - This can be used in four contexts. When a FlowCode document is open, it will output the FlowCode as assembly instructions. When a Microbe document is open, it will invoke the **microbe** program distributed with KTechlab to compile the program. Similarly, if a C program is open, it will attempt to compile it via SDCC. When a text document containing PIC hex is open, it will invoke **gpdasm** to disassemble the hex.
- **Convert to Hex** - This can also be used in four contexts. As with **Convert to Assembly**, this can be used with FlowCode, Microbe and C documents. It will also be enabled when an assembly document is open to assemble it via **gpasm**.
- **Upload to PIC** - This assembles the PIC program currently being edited, and uploads it using the programmer that the user has selected.

None of these actions require the current document to be saved - very useful for when a quick program is required. For non-PIC targets, the **Output** dialog invoked on clicking on one of these actions can either output the result (always text in the above three cases) to a fresh document, or to a file. If the output is saved to file, it also provides options to load the file after creation, and adding the newly created file to the open project (if one is open).

Note that you can make KTechlab always use the same view for displaying the outputted content by selecting the option under **General** settings.

2.2 Uploading

KTechlab uses third-party programmers to upload programs to PICs. A variety of common programmers come predefined. Others can be added via the **Settings** dialog.

The list of ports is obtained from scanning for serial and parallel ports that are readable and writable. Serial ports are looked for in:

- /dev/ttyS*[0..7]*
- /dev/tts/*[0..7]*
- /dev/ttyUSB*[0..7]*
- /dev/usb/tts/*[0..7]*

Parallel ports are looked for in:

- /dev/usb/parport*[0..7]*
- /dev/usb/parports/*[0..7]*

Chapter 3

Circuits

3.1 Placing components

On the left, you'll find the **Components** tab.

Dragging a component from the sidebar into the circuit will place it under the mouse cursor. Alternatively, you can double click on an item in the **Components** sidebar to repeatedly add it to the circuit. In this mode, a copy of the selected component will be placed repeatedly on mouse left-clicking until either **Esc** is pressed, or the mouse is right-clicked.

To reposition a component, left-click and drag. You'll find it snapping to the underlying grid. If you drag the component out of the right or bottom edges of the workarea, the workarea will resize itself to accommodate.

All components have a notion of orientation; 0, 90, 180 and 270 degrees. Those that aren't symmetrical about an axis can also be flipped. To rotate a selection of components, either right click and select from the **Orientation** menu, or click on the rotate buttons in the toolbar. The latter can also be accessed by pressing the **[** and **]** keys (familiar to Inkscape users). The **Item** sidebar (on the right) provides a powerful method of setting the orientation by providing previews of the components. Flipping components is also only possibly via the **Item** sidebar.

3.2 Connecting Components

There are two modes for creating connections (wires): automatic, and manual. These modes are selected via the **Connection Routing Mode** pulldown menu in the toolbar. Experiment with both - automatic routing is often better for small circuits, whereas more complex circuits may need manual routing.

In automatic mode, create a connection by dragging from either a component pin or an existing connection, and releasing the mouse over the desired pin or connection. You'll see the straight-line being drawn turn orange when a valid connection will be created on mouse release. If the line you're drawing is black, it's either because there's nothing beneath the mouse cursor, or you're attempting to connect together two items which are already connected. When flowcharting, the criteria for a valid connection are more complex - but we'll get to that later.

The best way to get a feel for manual connection routing is by experimenting with it. Click on the starting pin or connection, and then extend the proto-connector by moving the mouse away from where you clicked. To place a corner, left-click. To cancel drawing the connection, either press **Esc**, or right-click the mouse.

KTechlab tries its best to maintain the routes your connections take. However, if dragging a component results in the end points of a connection moving relative to each other, KTechlab will

be forced to redraw the connection using auto-routing. Before moving a component, you can see which connectors will have to be rerouted - as they will turn gray on clicking.

To remove an existing connection, select it by drawing a small select-rectangle over part of the connector, and hit **Del**.

3.3 Component Attributes

Most components will have editable attributes, such as the resistance for resistors. By default, you can edit simple attributes in the toolbar, when a group of the same type of components are selected. If your selection contains a mixture of different types of components (such as resistors and capacitors), then no attributes will be displayed for editing.

Some components have more advanced attributes which are not accessible via the toolbar. These are found in the **Item** sidebar on the right. The diode, for example, has a variety of behavioral characteristics that you can edit here.

There is one type of attribute that cannot be editable by either the toolbar or Item sidebar - multiline text. Double clicking on the item will bring up a dialog box where the text can be entered.

3.4 Simulation

By default, the simulation will be running when you create a new circuit. The status of the simulation is displayed in the lower right of a circuit view, and can be changed via the **Tools** menu. Firstly - a little explanation on how the Simulator works. This should allow you to make the most out of it.

When a circuit is created or modified, the affected areas are partitioned up into groups of pins and connections that can be considered independent. Each group is then simulated as a separate entity (although still interacting via the components), with the simulation provided dependent on the group's complexity. Complex groups, such as those involving nonlinear components like LEDs, are slow to simulate. Groups that contain only logic pins, of which only one controls the value on those pins, are the fastest to simulate.

The results of the simulation are provided through several graphical means.

The pins on the components will display voltage sidebars. These are colored orange for positive voltage, and blue for negative voltage. Their length depends on the voltage level, and their width on the amount of current flowing through the pin. These can be turned off in the **General** page of the **Configuration** dialog.

Hovering the mouse over a pin or connection will display a small tooltip showing the voltage and current at that point in the circuit. Several components also provide graphical feedback - for example, LEDs and voltmeters or ammeters.

Lastly, there is the oscilloscope, discussed in the next section.

3.5 Oscilloscope

The oscilloscope can record logic, voltage and current data. The logic probe is optimized for storing boolean samples, and so should be used instead of the voltage probe when measuring logic.

To collect data, create a new probe component, and attach it to an appropriate point in the circuit. You'll see the output immediately drawn in the oscilloscope. Adding more probes will squash the outputs next to each other - you can reposition these by dragging the arrows on the left of the oscilloscope view, and change their colors via the probe's attributes.

For voltage and current probes, the range of input values can be adjusted in the **Item Editor** sidebar on the right.

Zooming is controlled by a slider. The scaling is logarithmic; for every few pixels that the slider moves along, the zoom factor will be multiplied by a constant. KTechlab simulates logic to a maximum precision of 1 microsecond, and at maximum zoom level, one microsecond is represented by 8 pixels.

When the scrollbar is dragged to the end, it will remain there as new data is recorded. Otherwise, the position of the scrollbar remains fixed in time. The oscilloscope view can also be moved forwards and backwards by left-clicking and dragging the view. Due to limitations of the underlying widget system, scrolling will be very granular at maximum zoom.

Right-clicking on the oscilloscope view brings up a menu where you can control the number of times the oscilloscope view is updated. This allows for either a smoother display, or reducing CPU usage.

3.6 Subcircuits

Subcircuits offer a reusable and tidy way of using a circuit, when you're only interested in interacting with external connections to the circuit. The subcircuit is creating as an IC, with the pins acting as the interaction with the internal circuit.

First, the circuit to be used in as a template for creating a subcircuit from must be constructed. The points of interaction are defined via **External Connection** components. These must be connected up, and positioned where you want the pin to be positioned on the subcircuit IC.

Next, select the group of components and external connections to be turned into a subcircuit, and select **Create Subcircuit** from the right-click menu. You'll be offered to enter a name for the subcircuit. Once created, the name will popup in the **Components** selector under the **Subcircuits** selection. This can be treated as any normal component - with the additional option of removing it by right-clicking on the item and selecting **Remove**.

Chapter 4

FlowCode

4.1 Introduction

FlowCode allows for very quick and easy construction of a PIC program. After the user has constructed a flowchart from the program parts available, KTechlab can then convert the flowchart into a number of formats. To output hex, for example, the following chain of conversions takes place:

1. The FlowCode is converted to Microbe; a high-level language whose compiler is distributed with KTechlab.
2. The **microbe** executable then compiles the Microbe file to PIC assembly.
3. Finally, **gpasm** takes the PIC assembly file, and outputs the hex for the program.

Of course, if you don't have gputils installed - with which **gpasm** is distributed - then the last step can't be performed.

4.2 Creating a Program

Every FlowCode program needs a unique starting point - this is the place where your program will be run from on PIC startup. To define this point, open up the FlowParts sidebar on the left, and drag across the **Start** part. KTechlab will only allow you to use one of these.

You can then construct your program by using the predefined parts on the left - or insert code of your own (in assembly or Microbe format) via the **Embed** part. The flow of the program is controlled via the connections between the FlowParts - Section 3.2 offers more detail on creating connections.

FlowCode imposes limitations in addition to those of circuits on what can be connected. For example, each FlowPart can only have one output connection. Additional limitations are described in Section 4.4.

4.3 PIC Settings

When you create a new FlowCode document, you'll notice a picture of the PIC you're using in the top-left corner of the work area. This represents the initial settings of the PIC.

Each pin shown on the picture of the PIC shows the initial type of pin (input or output), and its initial state (high or low). You can change these by dragging the pin to set the type, and clicking on it to toggle its state.

The **Settings** dialog, invoked by clicking on the **Settings** button, also allows you to edit the initial pin types and states - in this case, by editing the binary values written to the PORT and TRIS registers. As well as pin settings though, the dialog allows editing of the initial values of variables in the PIC program.

At the bottom, there is a list of currently defined pin maps, as well as buttons to manipulate them. Pin maps are used to specify how a seven segment or a keypad is connected to a PIC. To use the **Seven Segment** or the **Keypad** FlowCode parts, you will need to define a pin map here first.

4.4 Nestling FlowCode

Many FlowParts, such as subroutines and loops, can contain code of their own. After creating such a container, FlowParts can be added by either dragging or dropping them into the container. The container will be highlighted to indicate that it will become the new parent of the FlowPart.

The container takes responsibility for FlowParts nestled inside. If the expand button is unclicked, all contained FlowParts will be hidden - and likewise, the contents will be shown when the expand button is clicked again. Connections cannot be made between FlowParts in different containers, and the contents of a container will be moved about along with the container.

Chapter 5

Microbe

5.1 Introduction and General Syntax

Microbe compiles programs written in the custom language for PICs, as a companion program to KTechlab. The syntax has been designed to suit a FlowCode program. The syntax for running **microbe** from the command line is:

```
microbe [options] [input.microbe] [output.asm]
```

where options are:

- `--show-source` - Puts each line of Microbe source code as a comment in the assembly output before the assembly instructions themselves for that line.
- `--no-optimize` - Prevent optimization of the instructions generated from the source. Optimization is usually safe, and so this option is mainly used for debugging.

The .microbe input file must identify the target PIC by inserting the PIC name at the top of the .microbe file; e.g. the name of a PIC16F84 is ″P16F84″.

Example 5.1 Simple complete Microbe program

```
P16F84

a = 0
repeat
{
  PORTA = a
  a = a + 1
}
until a == 5

end
```

5.1.1 Naming conventions

The following rules apply to variable names and labels:

- They can only contain alphanumerical characters [a..z][A..Z][0..9] and the underscore ″_″.
- They are case-sensitive.
- They cannot start with a number.
- They should not start with '__' (double underscore), as this is reserved for use by the compiler.

5.1.2 Bracing conventions

Curly braces, {}, indicate the start and end of a code block. They can appear anywhere before the start and after the end of the code block. Examples of acceptable code blocks:

```
statement1 {
  some code
}
```

```
statement2 {
  other code }
```

```
statement3
{
  other code
}
```

```
statement5 {
  code block
} statement6
```

5.1.3 Commenting

Commenting is similar to C. // comments out the rest of the line. /* and */ denote a multiline comment.

```
// This is a comment
x = 2
/* As is this
multiline comment */
```

5.1.4 Program Structure

The PIC id must be inserted at the top of the program. The end of the main program is denoted with 'end'. Subroutines must placed after 'end'.

5.1.5 Subroutines

A subroutine can be called from anywhere in the code. Syntax:

```
sub SubName
{
  // Code...
}
```

The subroutine is called with 'call *SubName*'.

5.2 Microbe language reference

5.2.1 if

Conditional branching. Syntax:

```
if [expression] then [statement]
```

or

```
if [expression] then
{
  [statement block]
}
```

Similarly for else:

```
else [statement]
```

or

```
else
{
  [statement block]
}
```

Example 5.2 if

```
if porta.0 is high then
{
  delay 200
}
else
{
  delay 300
}
```

5.2.2 alias

Aliases one string to another. Syntax:

```
alias [from] [to]
```

5.2.3 repeat

Executes the statement block repeatedly until expression evaluates to true. The evaluation of the expression is performed after the statement block, so the statement block will always be executed at least once. Syntax:

```
repeat
{
  [statement block]
}
until [expression]
```

5.2.4 while

Similar to repeat, this repeatedly executes the statement block. However, the expression is evaluated before execution, not after. So if the expression evaluates to false on the first pass, then the statement block will not get executed. Syntax:

```
while [expression]
{
  [statement block]
}
```

5.2.5 goto

This causes execution of the code to continue at the next statement after the label specified. Goto syntax:

```
goto [labelname]
```

Label syntax:

```
labelname:
```

It is often considered good programming practice to avoid the use of goto. Use of control statements and subroutines will result in a much more readable program.

Example 5.3 goto

```
goto MyLabel

...

[MyLabel]:
// Code will continue at this point
```

5.2.6 call

Calls a subroutine. Syntax:

```
call [SubName]
```

where *SubName* is the name of the subroutine to be called.

5.2.7 delay

This causes the code execution to stop for the given period of time. The interval is in milliseconds. Syntax:

```
delay [interval]
```

> NOTE
> At present, Microbe assumes that the PIC is operating at a frequency of 4MHz - i.e. each instruction takes 1 microsecond to execute. If this is not the case, the interval must be adjusted proportionately.

5.2.8 sevenseg

This is used to define the pin mapping for a (common cathode) seven segment display connected to the PIC. Syntax:

```
sevenseg [name] [a] [b] [c] [d] [e] [f] [g]
```

where [a]...[g] are the PIC pins to which the respective segments of the seven segment display are attached. The pins can be written either as PORTX.N or RXN.

To display a number on the seven segment, the pin mapping is treated as a write only variable.

Example 5.4 Defining and outputting to a seven segment

```
sevenseg seg1 RB0 RB1 RB2 RB3 RB4 RB5 RB6
seg1 = x + 2
```

5.2.9 keypad

This is used to define the pin mapping for a keypad connected to the PIC. Syntax:

```
keypad [name] [row 1] ... [row 4] [column 1] ... [column n]
```

where [row 1] ... [row 4] and [column 1] ... [column n] are the PIC pins to which the respective rows and columns of the keypad are attached (at the moment, the number of rows is not changeable). See Section 5.2.8 (above) for more information on pin mappings.

The columns of the keypad should be pulled down via 100k resistors to ground. The row pins must be configured as outputs and the column pins as inputs. Once the keypad has been defined, it is treated as a read only variable.

Example 5.5 Defining and reading from a keypad

```
keypad keypad1 RB0 RB1 RB2 RB3 RB4 RB5 RB6
x = keypad1
```

By default, the values returned for a keypad are:

- The value of the number if it is a numeric key (1 to 3 along top row; hexadecimal A to D down the fourth column and continuing for each extra column).
- 253 for the key in row 4, column 1.
- 254 for the key in row 4, column 3.

These values can be redefined by using the alias command, where the name of the key in row x, column y (rows and columns starting at 1), is Keypad_x_y. For example, to give the star key on a 4x3 keypad the value zero, the following alias would be used:

Example 5.6 Aliasing a keypad key to a value

```
alias Keypad_4_1 0
```

5.3 PIC I/O

5.3.1 Port Direction

The port direction is set by assigning a value to TRIS*, where * is the port letter. For example:

Example 5.7 Setting port directions

```
TRISB = b'01111001'
```

The above sets pins RB1, RB2 and RB7 on PORTB as outputs, and the other pins on PORTB as inputs. In this example, b'01111001' is a binary representation of the output type. The 1 on the right represents an output on RB0, and the 0 on the left represents an input on RB7.

5.3.2 Port I/O

The port can be treated as a variable. For example:

Example 5.8 Writing to a port

```
x = PORTA
```

The above assigns the value of PORTA to the variable x.

5.3.3 Pin I/O

Each pin on a port is obtained by prefixing the pin number by the port name; e.g. Pin 2 (starting from Pin 0) on PORTA is known as *PORTA.0*. The syntax to set a pin state is:

```
PORTX.N = STATE
```

where *STATE* can be *high* or *low*. The syntax to test a pin state is:

```
if PORTX.N is STATE then
```

Combining these examples, we have:

Example 5.9 Setting and testing pin state

```
TRISA = 0
TRISB = 255
if PORTA.3 is high then
{
  PORTB.5 = low
}
else
{
  PORTB = PORTA + 15
}
```

5.4 Variables

All variables are 8-bit unsigned integers, giving a range from 0 to 255. Microbe supports the typical unary operations (acting on one variable) and binary operations (acting on two variables) that are supported by the PIC. In addition, Microbe also supports division and multiplication.

5.4.1 Unary Operations

- *rotateleft x* - Rotates the variable x left through carry.
- *rotateright x* - Rotates the variable x right through carry.
- *increment x* - Increments the variable x. If x has a value of 255, then x wraps round to 0.
- *decrement x* - Decrements the variable x. If x has a value of 0, then x wraps round to 255.

5.4.2 Arithmetic

Supported operations:

- *Addition:* x + y
- *Subtraction:* x - y
- *Multiplication:* x * y
- *Division:* x / y
- *Binary XOR:* x XOR y
- *Binary AND:* x AND y
- *Binary OR:* x OR y

5.4.3 Comparison

Supported operations:

- *Equals:* x == y
- *Does not equal:* x != y
- *Is greater than:* x > y
- *Is less than:* x < y
- *Is greater than or equal to:* x >= y
- *Is less than or equal to:* x <= y

For example:

Example 5.10 Comparison

```
if PORTA >= 5 then
{
  ...
}
```

Chapter 6

Debugging

6.1 Starting the Debugger

Debugging support is provided for Assembly, SDCC and Microbe, when they are open as a text document. From here, stepping is controlled via the **Debug** menu. There are two methods of starting the debugger.

If the PIC program is already running in a circuit, then double-clicking on the PIC component will open up the program. For assembly PIC programs, the debugger for that text document will be linked into the PIC component. In this case, the debug menu cannot stop the PIC program - as this is owned by the PIC component.

If the assembly file is already opened, then the debugger can be run via the **Debug** menu. After compiling the program, the debugger will be ready, with the PIC program paused at the first instruction. Note that when debugging high level languages, the current execution point will not be shown if there is no line that corresponds to the first assembly instruction to be executed. In this case, clicking **Next** will bring the execution point to the first line in the program.

6.2 Controlling the Debugger

The debugger can be in one of two modes: running, and stepping. While running, the PIC program will be simulated in realtime. To allow stepping, the PIC program must be paused - either by clicking on **Interrupt** in the **Debug** menu, or clicking on the pause button on the PIC component.

In stepping mode, a green arrow in the margin of the text document indicates the next line to be executed (familiar to KDevelop users). It may be useful to turn on the icon border via the **View** menu (it can be permanently turned on via the **Editor Settings** dialog).

There are three types of stepping:

- **Step** - This executes the current instruction. The green arrow is moved onto the next line to be executed.
- **Step Over** - If the next instruction to be executed is a call, or similar, then this will "step over" the call, returning to stepping mode once the call has returned. Otherwise, stepping over an instruction behaves identically to step. To put it technically - the initial stack level is recorded, and the program execution is paused once the stack level returns to its initial level.
- **Step Out** - If the current execution is inside a call, or similar, then this will wait until the call returns. Similarly to stepping over, this is equivalent to waiting until the stack level returns to one less than the initial level, if the initial level is greater than zero.

Breakpoints allow the execution to be paused when the PIC program reaches a given instruction. To toggle a breakpoint on the line containing the cursor, either use the **Debug** menu, or click on the icon border of the text document.

The **Symbol Viewer** sidebar on the right shows the values of the Special Function Registers. To find out the value of a variable in the General Purpose Registers, you can hover your mouse over the variable name in an instruction that operates on that register. Note that the radix selection in the **Symbol Viewer** also controls how the value is displayed when hovering over a variable.

Chapter 7

FAQ

1. *KTechlab uses lots of CPU*

   There are several possible causes. Simulation of circuits that contain both reactive and non-linear components (such as capacitors and transistors) take a lot of CPU time to simulate. You can pause and resume the simulation via the **Tools** menu.

   Drawing of the work area (in particular, redrawing lots of rapidly updating voltage bars on pins) is also CPU intensive. You can reduce the refresh rate or turn off the voltage bars via the **Settings** dialog. The refresh rate of the **Oscilloscope** can also be reduced by right clicking on its display.

   Note that the next major release of KTechlab can be a lot faster in both displaying the work area and simulating reactive and nonlinear components.