For Macintosh Programmers & Developers
MacTech MAGAZINE

MICROCONTROLLER PROGRAMMING

By Tom Djajadiningrat

PIC Microcontroller Programming on MacOSX

Using a VOTI Wisp628 with JAL and XWisp

Tom Djajadiningrat works for the Designed Intelligence Group of the Faculty of Industrial Design of Eindhoven University of Technology. Recently, his wife gave him a brand-new, 3.5kg, non-Apple laptop. Admittingly, he now uses his Ti-Book much less as his new laptop seems in every way more intelligent and fun and seems to become even more so by the day. Its only disadvantage may be that it also grows even heavier every day. You can flame him on his Apple infidelity at j.p.djajadiningrat@tue.nl

ABSTRACT

This article explains how to program a Microchip PIC microcontroller on a Macintosh running OSX. The programmer hardware that is used is a Wisp628 by Van Ooijen Technische Informatica. The software consists of two parts, JAL and XWisp. JAL is an open source, high-level language that is used to generate a PIC compatible hex file. XWisp is Python-based, open source software that is used to upload the hex file from the Macintosh host via the Wisp628 programmer board to the PIC microcontroller. The article explains how to compile JAL for MacOSX, how to create a connection from the Mac via the Wisp628 programmer board to the target circuit containing the PIC, how to upload a hex file using XWisp, and how to create a simple LED flashing program.

INTRODUCTION

Software for electronic engineering purposes has been far from a stronghold for the Macintosh. Especially affordable, electronic hobbyist level software that is compatible with a Mac has traditionally been hard to find. Luckily, with the arrival of OSX this has changed somewhat. Due to OSX's UNIX underpinnings it is possible to recompile many open source software packages to run on a Mac.

In this article we show how to use two pieces of open source software for microcontroller programming on an OSX Mac. One we will recompile using the GCC compiler, the other we can run using Python. Both the GCC compiler and Python are standard components of the Apple Developer Tools for OSX. Don't worry if you have never done any of this before: this article is meant as an absolute beginner's guide. We'll take it nice and slowly, guiding you through the whole process, one step at a time.

REQUIRED HARDWARE AND SOFTWARE

This is what you need to get PIC programming on a Mac.

Hardware

- Macintosh running OSX.2 or OSX.3
- a USB-serial adapter. There are several types on the market. We tested two:
  - Keyspan USB High Speed Serial Adapter USA-19QW (Figure 1)

    costs: approx. 40-50 USD

    info: www.keyspan.com
  - GMUS-03 USB Serial Adapter (Figure 2)

    costs: approx. 22 USD

    available from: www.voti.nl
- XWisp 628 programmer board (Figure 3)

  available from: www.voti.nl.

  costs: assembled approx. 35 USD. As a kit: approx. 24 USD.
- DB9 'straight-thru' male-female serial cable (if your physical workspace is close to your USB port you may be able to do without it and connect your USB-serial adapter straight to the XWisp 628 programmer board)
- Microchip PIC-16f877 microcontroller or its slightly cheaper successor, the PIC-16f877A. These are popular PICs with high specs.

  available from: most electronics stores, including www.voti.nl.

  costs: approx. 10 USD for the 16f877 or 8.50 USD for the 16f877A
- Electronics breadboard

  available from: most electronics stores, including www.voti.nl

  costs: from approx. 11 USD

Figure 1: Keyspan USB High Speed Serial Adapter USA-19QW

Figure 2: GMUS-03 USB-serial adapter

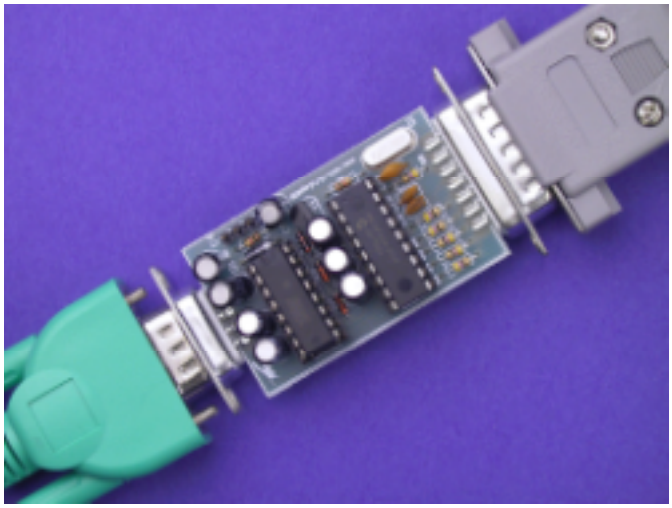

Figure 3: The XWisp 628 programmer board by VOTI. The left-hand side is connected to the USB-serial Adapter through a straight-thru DB9 male-female serial cable, the right-hand side accepts a DB15 connector leading to the target circuit.

Software

- Apple Developer Tools
  for MacOSX.2: December 2002 Developer Tools + August 2003 gcc Updater
  for MacOSX.3: Xcode Tools v1.1
  free download from: http://developer.apple.com/tools/download
- Keyspan drivers.
  Make sure that you have a version that is compatible with your version of OSX.
  free download from: http://www.keyspan.com/downloads/macosx/
- JAL, latest distribution. We used 0.4.59.
  free download from: https://sourceforge.net/projects/jal/

- XWisp

  free download from: http://www.voti.nl/xwisp/xwisp_src.tar.gz

The total costs add up to around 77-105 USD excluding shipping and handling, depending on your choices. As you may have noted, the GMUS-03 USB-serial adapter is considerably cheaper than the Keyspan. GMUS-03 Drivers for every flavour of OSX are available and the thing works admirably for what we are doing in this article. However, whilst Keyspan have certified their adapter with a great many serial devices, with the GMUS-03 your mileage may vary.

You can save some costs by buying the WISP628 programmer in kit-form, rather than ready-assembled, and by going for a cheaper model PIC.

JAL

We tackle JAL in three parts. Firstly, we prepare JAL for OSX by making a minor change to the code, then we recompile JAL, and, finally, we try out our newly created JAL executable.

Preparing JAL's source code for OSX

Assuming that you have downloaded all the required software components, we start with compiling JAL for Mac OSX. To do this you have to work with Terminal. In the Finder, start up Terminal which is located in the folder `/Applications/Utilities`. Now we have to navigate to the JAL folder. I dropped mine in `Applications`. The easiest way to change directory in Terminal is to type `cd`, followed by a space, and then drag the folder you want to navigate to from the Finder onto the Terminal window. Terminal will then add the correct, absolute path. In my case, the Terminal window says:

```
J-P-Djajadiningrats-Computer:~ Tom$cd /Applications/jal-
0.4.59
```

In your case, Terminal may say something different, depending on the version of the JAL distribution and on where you put it. Type `ENTER` to change to this directory.

From now on, we use a dollar sign to represent the Terminal prompt. For all clarity, you only need to type what follows the dollar sign, not the dollar sign itself. Also, this was the first and last time we mentioned that you need to type `ENTER` after each Terminal command.

Do a listing of the Jal-0.4.59 directory by typing:

```
$ls -F
```

This should give you the contents of the JAL directory as shown in the screenshot in Figure 4. The operand -F makes that the listing displays directories followed by a slash ('/') and executables by an asterisk ('*').

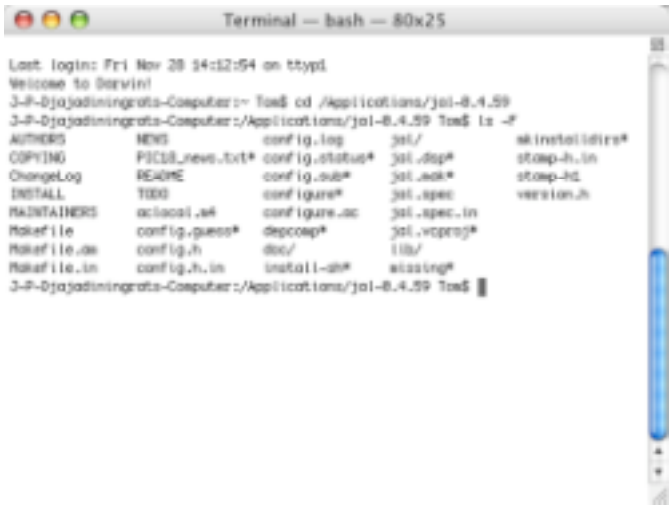

Figure 4: Navigating to and listing the jal-0.4.59 folder

Within the Jal-0.4.59 folder, we now navigate to the source code directory which is simply called `jal`:

```
$cd jal
```

To see a listing of the source code in the jal directory you can again type:

```
$ls -F
```

Before we can start compiling we have to make a small change to the file `stdhdr.h` (note that if you have a newer release than 0.4.59, this change may not be necessary anymore). Type:

```
$open stdhdr.h
```

Depending on what version of the Apple Developer Tools you are using, `stdhdr.h` opens in either ProjectBuilder or Xcode. Look for the piece of code in Listing 1a.

Listing 1a: stdhdr.h

#ifdef HAVE_MALLOC_H

```
#ifdef HAVE_MALLOC_H
#include <malloc.h>
#else
void *malloc(int);
#endif
```

Comment out these lines, C style, by adding slash-asterisk ('/*') in front and asterisk-slash behind ('*/') so that the block of code looks like Listing 1b.

Listing 1b: stdhdr.h

#ifdef HAVE_MALLOC_H

```
/*
#ifdef HAVE_MALLOC_H
#include <malloc.h>
#else
void *malloc(int);
#endif
*/
```

Don't worry too much about why you need to make these changes. Basically, on OSX we already get access to malloc by including stdlib.h, and therefore including malloc.h

gives 'already defined' errors. Close the file in ProjectBuilder/Xcode and save your changes. Now we are ready to start compiling.

Compiling JAL

Switch back to Terminal and change to the directory jal-0.4.59 which means we have to go one directory level upwards. Type:

```
$cd ..
```

The next step is to configure compilation for your Mac by launching the UNIX executable configure:

```
$./configure
```

Running it takes a little while, during which you see a whole bunch of checks rolling by. Finally, Terminal should print the message in Listing 2:

Listing 2: $./configure

Closing remarks during compilation

```
jal-0.4.59 is now configured for

  Build:                powerpc-apple-darwin7.0.0
  Host:                 powerpc-apple-darwin7.0.0
  Source directory:     .
  Installation prefix:  /usr/local
  C compiler:           gcc -g -O2
```

Now type:

```
$make
```

This takes longer, as gcc compiles JAL. To complete the installation process type:

```
$sudo make install
```

Note the sudo (superuser do) command. It allows an administrator—which presumably you are when you are on your own Mac—to run commands as superuser. Basically, becoming superuser upgrades your priviliges so that you may access files and directories which normally are protected against (accidental) misuse. In this case, we want to install into the /usr/local/bin directory which is a protected directory. After entering this command, Terminal therefore asks you for your password. Once you have given it, Terminal responds as per Listing 3.

Listing 3: $sudo make install

Response to $sudo make install

```
/bin/sh ../mkinstalldirs /usr/local/bin
  /usr/bin/install -c  jal /usr/local/bin/jal
make[1]: Nothing to be done for `install-data-am'.
```

As you can see, the JAL executable ends up in the directory `/usr/local/bin`. To convince ourselves, change to the installation directory and do a listing:

```
$cd /usr/local/bin
$ls -F
```

And there you have it: jal followed by an asterisk, indicating that this is an executable. But there is more to the Jal installation than just the executable. There are also several include files. Let's see whether we can find them.

Change to the parent directory `/usr/local` and do a listing:

```
$cd ..
$ls -F
```

In it you see a directory named `share`. Change to it and do a listing:

```
$cd share
ls -F
```

There you find a directory called `jal`. Change to it and do a listing:

```
$cd jal
$ls -F
```

Finally, there is the `lib` directory. In this directory you find a couple of dozen of include files:

```
$cd lib
$ls -F
```

I hope this gives you a feel for where your Jal installation ends up.

Taking JAL for a spin

Create a directory somewhere convenient for your JAL source files. Mine is called `src` and lives on a partition called `Data`. Now we create a JAL source file. First, using a text editor of your choosing (I used XCode), create a new empty file and type in the code in Listing 4. Don't worry about the exact meaning of the code for the moment. If you are curious, later on we will use this JAL program to flash an LED connected to pin a0 of our PIC:

Listing 4: ledflash.jal

The full ledflash.jal source file

```
include 16f877_20
include jlib
disable_a_d_functions

pin_a0_direction = output

forever loop
  pin_a0 = high
  delay_10mS(255)
  pin_a0 = low
  delay_10mS(50)
end loop
```

Save the file into your source file directory under the name ledflash.jal. The filename itself is up to you, but it is critical that you use the .jal extension.

Change to your source file directory by typing `cd`, followed by a space and then dragging the directory from the Finder onto your Terminal window:

```
$cd /Volumes/Data/src
```

Do a sanity check verifying that the file ledflash.jal is actually in there:

```
$ls
```

And now, for the moment suprême. Let's try compiling ledflash.jal using our jal executable:

```
$/usr/local/bin/jal ledflash.jal
```

As ledflash.jal is compiled, you see the lines flash by and, finally, Terminal responds as in Listing 5:

Listing 5: $/usr/local/bin/jal ledflash.jal

JAL compiling ledflash.jal

```
jal 0.4.59 (GCC 3.3)
```

```
input        files:12 lines:2244 chars:58707 (2952
lines/second)
compilation  nodes:12695 stack:53Kb heap:3928Kb
seconds:0.760
output       code:101 file:14 stack:2
OK
```

Have a look what is in our `src` directory now:

```
$ls
```

There you have it: in addition to a .jal file, there are now an assembler file (ledflash.asm) and a hex file (ledflash.hex). Drag them onto your favourite text editor to see their content:

The hex file is very compact as shown in Listing 6.

Listing 6: ledflash.hex

The ledflash.hex file resulting from compiling ledflash.jal with JAL

```
:020000040000FA
:020000000428D2
:08000800FF30A100FF30A2004F
:10001000FF30A300FF30A4000F30A5008A110A12A0
:100020004E2021108A110A125C2026148A110A120D
:100030005620FF308A110A122B2026108A110A122C
:1000400562032308A110A122B208A110A121528E2
:100050008A110A122828A7002708A8006430A900DE
:1000600001308A110A123428AA00FF30AB00290897
:10007000AD002808AC0077308A110A122B0703184C
:100080003E288A110A12AC0B3B288A110A12AD0BCA
:1000900039288A110A12AA0B372808008A110A1275
:1000A000622007309F008A110A1265288A110A12FD
:1000B000592826088500080 08A110A125F2821089D
:1000C000650008008316031308008312031308005 9
:02400E00723FFF
:00000001FF
```

The assembler file is something you are supposed to never need. The gurus use it to debug the compiler and to learn how assembler works, ie. to figure out what the PIC really does for A = B + C.

So far so good. We can write a simple .jal file and compile it using an OSX native JAL executable, resulting in a hex file suitable for a PIC microcontroller. Now how do we actually upload this hex file from the Mac to the PIC?

UPLOADING THE HEX FILE

First, we need to get our physical, serial connection between the Mac, the Wisp628 and the PIC microcontroller in order, then we look into how to use the XWisp software to upload our hex file to the PIC.

Serial connection

Plug your USB adapter into the USB port of your Mac. Assuming that you are using a Keyspan, you can do a check by running the Keyspan Serial Assistant which lives in your Applications folder. The Serial Assistant should recognize the adapter (Figure 5).

Figure 5: The Keyspan Serial Assistant acknowledges the adapter

Depending on the cable length you need between your USB port and your physical workspace, you can connect the Wisp628 board directly to the USB adapter or connect it using a 'straight-thru' DB9 serial cable,

With the Wisp628 comes a cable which has a DB15 female plug on one end and which has bare wires on the other end. Plug the DB15 female plug onto the Wisp 628 board.

Plug the 16f877 microcontroller into a breadboard

The necessary circuitry consists of two parts: a regulated 5V power supply (Figure 6) and a target circuit including the PIC (Figure 7). As these circuit diagrams may turn out to be hard to read when scaled to the width of a single MacTech column, the code archive includes PDF versions of both.

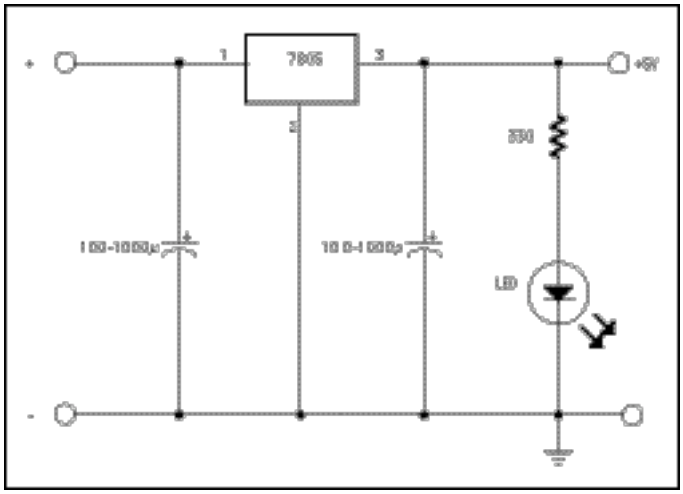

Figure 6: A regulated 5V power supply

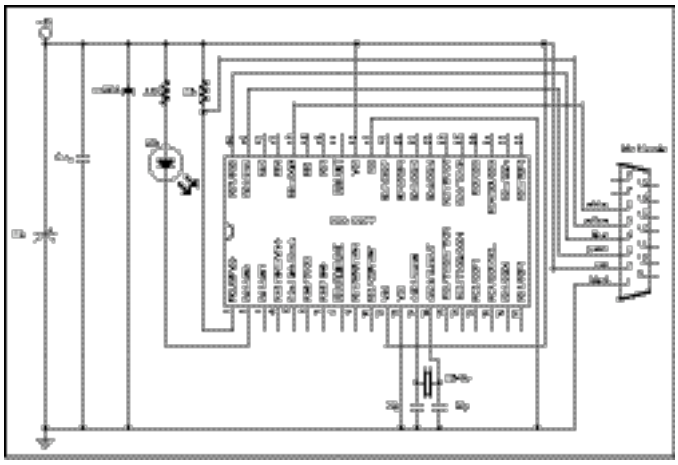

Figure 7: The target circuit with the PIC

You can build a stable 5V supply starting from a 9V battery or from a 9V DC power supply. I would recommend that you use a 9V DC power supply rather than a battery since the voltage regulator tends to eat through 9V batteries rather quickly.

Components for regulated 5V power supply:

1x 9V DC power supply or a 9V battery with a clip-on adapter

1x 7805 voltage regulator

1x green LED

2x 330Ohm resistor

2x 100-1000μF capacitor

The resistor and the LED are not strictly necessary but do make the final check very easy: the LED should light up.

Components for the target circuit with PIC:

1x 330Ohm resistor

1x 33kOhm resistor

1x red LED

1x 20MHz crystal

2x 20pF capacitors

1x 1n4004 diode

1x 22μF capacitor

1x 0.1μF capacitor

The 1n4004 is a 'fool's diode': if you accidentally reverse the polarity of your power supply it will prevent damage to your PIC. Of course, the diode has its limits. If you happen to use a supply that is capable of delivering high currents (>1A), such as an old PC power supply, the diode will blow and the PIC will be damaged after all.

All that is left now, is to connect the Wisp628 board to the PIC in the breadboard and build a small test circuit around the PIC. This is detailed for various PIC microcontrollers

on http://www.voti.nl/wisp628/n_index.html. Here we focus on the connections for a 16F877 microcontroller (Figure 8).

DB15 pin#	DB15 cable strand colour	PIC pin#	PIC pin name
1	black	12, 31	Vss (Gnd)
2	red	11, 32	Vdd (+5V)
5	yellow	1	MCLR#/Vpp
6	white	36	RB3/PGM
3	green	39	RB6/PGC
4	blue	40	RB7/PGD

Figure 8: Connections between the DB15 connector of the XWisp 628 and the target circuit with the PIC

Figure 9 shows how things were wired on my breadboard. Be aware that some breadboards, including this one, have breaks in the middle of the power rails which you need to bridge if you want to use both the left and the right half. Again, a high-res JPEG picture is included in the code archive.

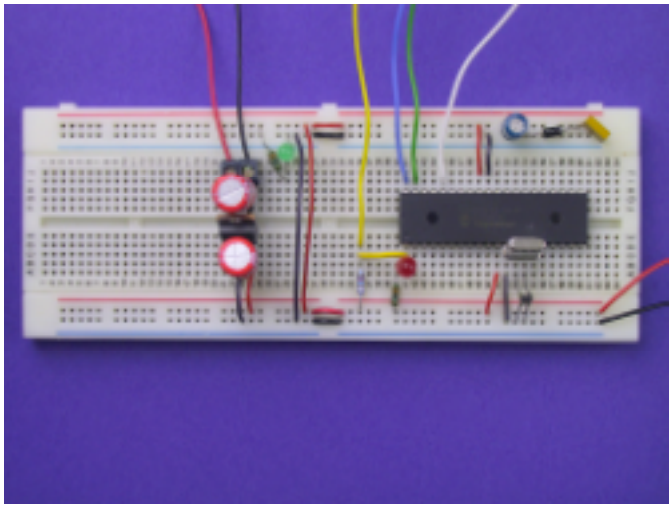

Figure 9: The 5V power supply and the target circuit with the PIC. Everything to the left of the wire bridges in the power rails forms part of the 5V power supply, everything to the right forms part of the target circuit with the PIC. The red and black lead coming in from the top-left are connected to the 9V battery/power-supply. All the other flying leads go to the Wisp 628.

Using XWisp

Now we can work on the software side of uploading the ledflash.hex file to the target circuit. For this we use the Python-based XWisp. Before we can issue the upload command we need to do three things. Firstly, we make a small modification to one XWisp source file to enable XWisp to run on OSX (Just as with the Jal source, you may find that in the latest release of XWisp, this modification may not be necessary anymore). Secondly, we copy the XWisp folder to /usr/bin. And, finally, we find the name of the serial port provided by the USB-serial adapter and its driver software.

Modifying XWisp

In the `xwisp_src` Folder that you downloaded you find six files. Drop the file xwisp.py on ProjectBuilder or Xcode to open it. Use the Find command to look for CMD_Port. In this definition, look for the line of code saying:

```
Name = self.Get_arg( Uppercase = 0 )
```

After this line, we need to add two lines:

```
self.Port = Name # these two lines are added
return # to use the port name 'as is'
```

Please note that Python code is indentation sensitive. Indents are to Python what curly brackets are to C. You need to make sure that the two added lines are on the same indentation level as the line

```
if Name == None:
```

Just to make sure, the finished version of the CMD_PORT definition is shown in the screenshot in Figure 10.

```
def CMD_PORT( self, Name = None ):
   self.Close_Bus()
   if Name == None:
      Name = self.Get_Arg( Uppercase = 0 )
   self.Port = Name # these two lines are added
   return # to use the port name 'as is'
   if (Name + '    ')[ 0 : 3 ].upper() == 'COM':
      if (int((Name + '    ')[ 3 : ]) > 9):
         self.Port = '\\\\.\\' + Name
      else:
         self.Port = Name
   else:
      N = self.Int_Value( Name )
      if N > 32:
         self.Active_Baudrate = N
      else:
         self.Port = N
```

Figure 10: a screenshot of the CMD_Port module after the necessary changes

Copying XWisp

Rename the folder named `xwisp_src Folder` to just `xwisp`. Now we need to copy the folder to /usr/bin. Unfortunately, we cannot simply use the Finder as the folder `/usr` is hidden. Therefore we again turn to Terminal. Type `sudo cp -R`, followed by a space, then drag our xwisp folder onto Terminal like we did before, and finally type `/usr/local`. In my case, the command looked like:

```
$sudo cp -R /Users/Tom/Desktop/xwisp /usr/local
```

This copied the `xwisp` folder with all its content. Now do a listing of the directory `/usr/local.`

```
$ls -F /usr/local
```

In the listing you should find the directory xwisp. If you wish to convince yourself that we have not only copied the folder but its contents too, do a listing of the contents of the xwisp directory:

```
$ls /usr/local/xwisp
```

Finding out the name of the serial port

In terminal, type:
```
$ls /dev
```

If you have the Keyspan adapter, look for a serial port named something like `tty.USA19QW181P1.1`. The name may differ with the exact type of Keyspan adapter you have got. If you use a GMUS-03, look for a port named something like `tty.usbserial0`. Note down the name somewhere as we will need it in a moment.

We are nearing yet another moment suprême. To run XWisp you need Python which luckily forms part of the Apple Developer Tools.

Change to the directory where you keep your .jal, .asm and .hex files. In my case:
```
$cd /Volumes/Data/src
```

OK, ready? We have xwisp.py living in /usr/local/xwisp, ledflash.hex living in our working directory and a serial port living in /dev. Now, on a single line, type this if you have the Keyspan adapter:
```
$python /usr/local/xwisp/xwisp.py port
/dev/tty.USA19QW181P1.1 go ledflash.hex
```

Or this if you use the GMUS-03 adapter:
```
$python /usr/local/xwisp/xwisp.py port
/dev/tty.usbserial0 go ledflash.hex
```

If you are (very) lucky, the LED connected to pin a0 of the PIC microcontroller starts flashing. No joy? No need to panic, yet. With a setup with this many components, it unlikely that you get things running the first time round. Which brings us to...

TROUBLE SHOOTING

Stuck?

Here are some things to check:

Do you manage to run XWisp?

When you try to run XWisp, Terminal should respond with printing some lines, the first of which reads:
```
XWisp 1.08, command line mode
```

If it does not, make sure that XWisp really lives where you think it lives.

Are you really reaching that serial port?

Usually, you can figure out from the error message in Terminal if you do not address the serial port correctly (If you use the Keyspan adapter you can look at its LED: it should start flickering during uploading of the hex file). If not, check the physical connection between Mac and USB-serial adapter, check the name of the serial port, and make sure that you have specified the complete path: you really need that `/dev/` in front of the port name.

Does your power supply (still) work?

Check whether the LED in your power supply circuit is on. But even if it is on, the voltage over the PIC may have dropped to unacceptable levels. The 7805 voltage regulator needs about 2V headroom to do its work: to create 5V output it needs at least 7V input. With a 9V battery you can quickly drop below this 7V level. If you find that you are going through your 9V battery a bit too quickly, you may want to replace it by a 9-12V power

adapter. A low supply voltage may lead to all kinds of weird behaviour: at 4.5V the PIC may run its program still fine, but programming it may have become impossible.

Is your target circuit correctly hooked up?

Check the wires between the Wisp628 and your target circuit.

Make sure that you really have +5V on pins 11 and 32 and ground on pins 12 and 31.

And finally... is your LED the right way round?

Well, I'm sure you wouldn't be the first... The flat side or shorter lead is the cathode and should be connected to pin a0. The other side, the anode, is connected to +5V. When you are in doubt whether the LED is connected correctly, just disconnect the PIC-side of the LED and connect that side to ground (0 Volt). The LED should light up. If it does not, it is reversed (or there is no power).

Still stuck?

On www.voti.nl you can find some highly detailed pages to help you trouble shoot.

Desperate?

There is a great Yahoo Group for Jal users:

http://groups.yahoo.com/group/jallist/

Strictly speaking this group is for Jal questions only, but you would not be the first to start a discussion about the Wisp 628.

CONCLUSIONS

So there you have it: programming a PIC microcontroller on a Mac. It takes a little time and perseverance to set it all up. But it works, have fun!

ACKNOWLEDGEMENTS

I gratefully acknowledge my colleagues Peter Peters and Joep Frens at the Designed Intelligence Group of Eindhoven University of Technology for checking the manuscript. Many thanks also to Wouter van Ooijen of Van Ooijen Technische Informatica to get all of this running, and to Daniel Saakes of the ID-StudioLab of Delft University of Technology for his Python on Mac advice.

REFERENCES

Microchip are the makers of the PICmicro microcontrollers series. Here you can find datasheets for all model PICs.

http://www.microchip.com

VOTI, Van Ooijen Technische Informatica, are the makers of the Wisp628 hardware, the XWisp software and the originators of JAL. Here you can find detailed instructions on all their products as well as a web shop.

http://www.voti.nl

Keyspan are the makers of a USB-serial DB9 adapter.

http://www.keyspan.com

A MacOSX 10.3 Panther compatible driver for the GMUS can be found here (drivers for older versions of OSX are provided on the installation CD.

http://www.ramelectronics.net/download/BF-810/OSX/

The Yahoo group 'jallist' is a forum for JAL users.

http://groups.yahoo.com/group/jallist/

Apple provides ProjectBuilder and Xcode as free downloads:

http://developer.apple.com/tools/download/

On source forge the open source community further develops Jal:

https://sourceforge.net/project/showfiles.php?group_id=71552

And finally, here you can find lots of information on PICmicros:

http://www.piclist.com