This is the Revision A verion of the Pic16F876 RoboBrick. The status of this project is that it has been replaced by the PIC876Hub10 RoboBrick.

Pic16F876 Robobrick (Revision A)

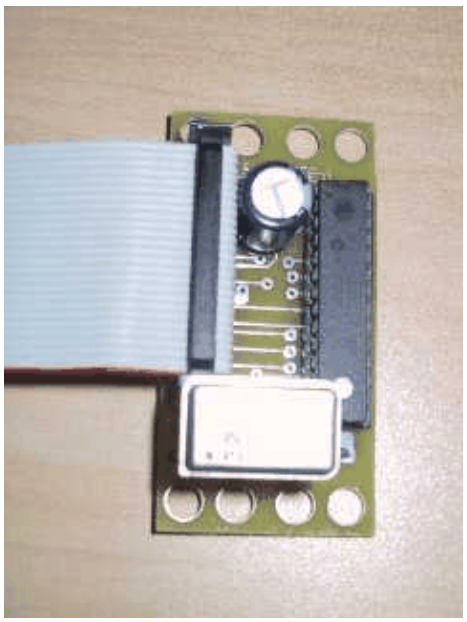
Table of Contents

This document is also available as a PDF document.

1. Introduction

The PIC16F876 RoboBrick is a master RoboBrick for controlling a robot. It is based on the PIC16F876 from MicroChip®.

A picture of the PIC16F876 RoboBrick is shown below:



One nice feature of the PIC16F876 is that it can write into its program memory without requiring any additional voltages or hardware. An initial boot loader will be placed in upper memory. This boot loader will talk to the development system via a link RoboBrick like the Tether RoboBrick. The development system will download the programs over the link and the boot loader will store the program into memory and execute it. If the program crashes, the standard watchdog timer on PIC16F876 will transfer control back to the boot loader.

2. Programming

The PIC16F876 is initially programmed with a boot loader that allows other programs to be downloaded and executed. The boot loader lives in upper memory and provides a way of downloading a program to be executed in lower memory. In addition to boot loading, there are some additional facilities for examining and setting memory for debugging purposes.

Upon startup, the boot loader selects port 0 of the hub, prints out the RoboBrick name and version, followed by a prompt as shown below:

```
RoboBrick PIC16F876-A Version 1.0
>
```

There are only five supported commands:

E<from_address>[:<to_address>]
: This command will display the contents of data memory from <from_address> to <to_address>. If <to_address> is not present, only the single byte at <from_address> is presented.

V<from_address>[:<to_address>]
: This command will view the contents of program memory from <from_address> to <to_address>. If <to_address> is not present, only the single byte at <from_address> is presented.

S<start_addresss> <byte> ...
: This command will store bytes of data into data memory starting at <start_address>.

R<execute_address>
: This command will start execution at <execute_address> by execute a `CALL <execute_address>' instruction. Thus, if the called program ever returns, it will return to the boot loader prompt.

:LLAAAACCDD...SS
: This command will take a single line from an `Intel Hex' file and store it into program memory. The format of one line of an Intel hex file is *LLAAAACCDD...SS*, where each field is described below:

  :
  : The command starts witha colon (`:').

  LL
  : *LL* is a two digit hex number that specifies how many bytes of data are being programmed. Remember, it takes two bytes to program a single word of program memory.

  AAAA
  : *AAAA* is four digit hex number that specifies the `byte address' at which to program the first byte. The byte address is word address shifted over by 1 bit.

  CC
  : *CC* is a command of either 00 or 01. 00 specifies *load* and 01 specifies *done*.

  DD ...
  : *DD* is sequence of *LL* bytes of data to be programmed. The low order program byte comes before the high order program byte.

  SS
  : *SS* is a two's complement checksum of all of the bytes that make up the command line.

  Upon receiving the first command that starts with a `:', prompts are supressed until the *done* command is received. This allows a user to `throw' a .hex file at the boot loader using a terminal emulator.

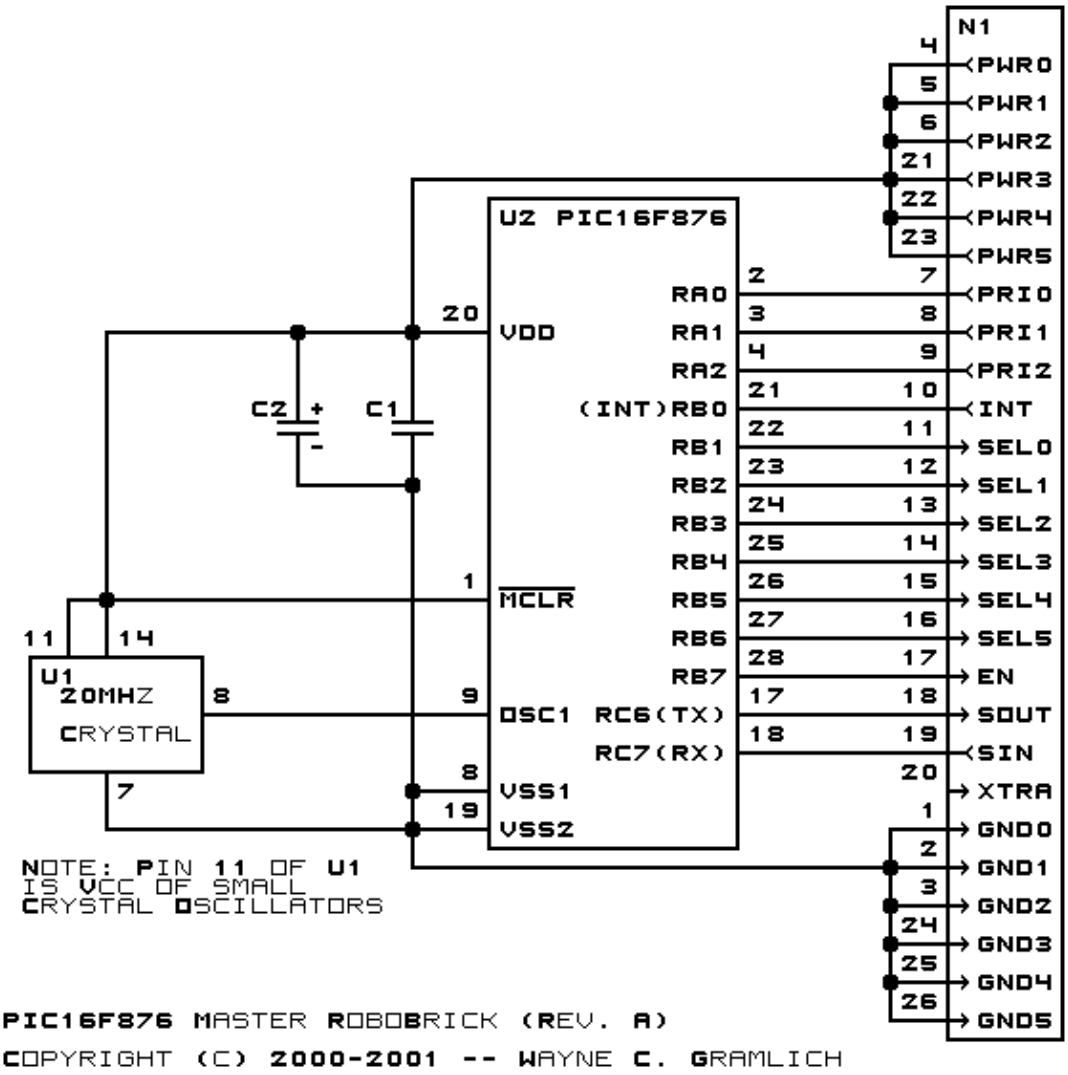
Under *no* circumstances will the boot loader overwrite either itself, the configuration word, or the first word of program memory (which is a GOTO instruction to the boot loader.)

3. Hardware

The hardware consists of a circuit schematic and a printed circuit board.

3.1 Circuit Schematic

The schematic for the Pic16F876 RoboBrick is shown below:



The parts list kept in a separate file −− pic16f876.ptl.

3.2 Printed Circuit Board

The printed circuit files are listed below:

pic16f876_back.png

The solder side layer.

pic16f876_front.png

The component side layer.

pic16f876_artwork.png

The artwork layer.

pic16f876.gbl

The RS−274X "Gerber" back (solder side) layer.

pic16f876.gtl

The RS−274X "Gerber" top (component side) layer.

pic16f876.gal

The RS−274X "Gerber" artwork layer.

pic16f876.drl

The "Excellon" NC drill file.

pic16f876.tol

The "Excellon" NC drill rack file.

4. Software

The software for the PIC16F876 will eventually become quite sophisticated. But for starters it will be little more than testing software for whole system. The files are:

pic16f876.ucl

The source code for the RoboBrick written in µ.

pic16f876.asm

The generated assembly file.

pic16f876.lst

The generated listing file.

pic16f876.hex

The generated hex file.

5. Issues

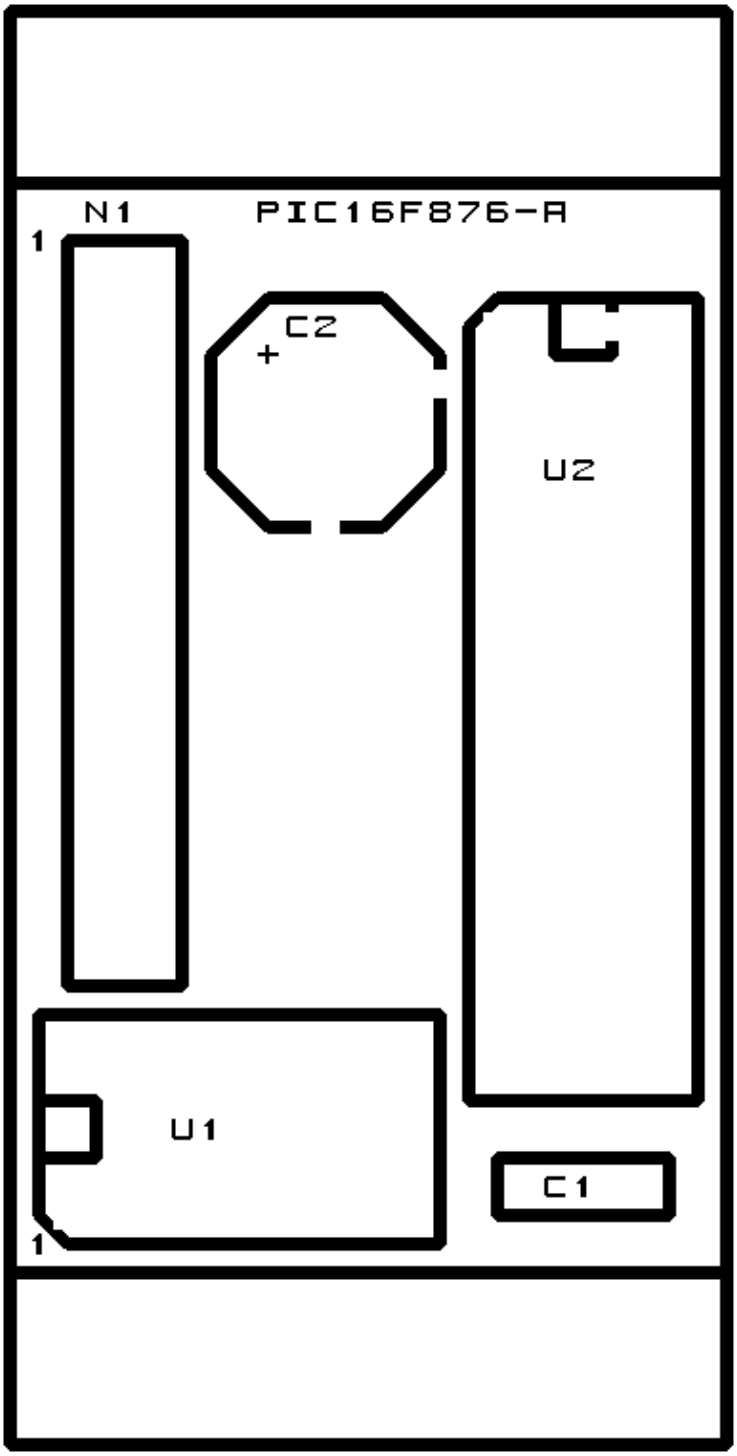
The following issues came up on the revision A version of the PIC16F876 RoboBrick:

- The 4−wire module cable crimpers are hard to find. Switch over to 6−wire modular connectors.
- Mark the slot on N1 in the artwork.
- Wire up the extra pin on on the N1 to an unused pin on the PIC16F876.
- Add a switch of some sort so that the firmware in the PIC16F876 can know whether to start with the boot loader or directly execute the downloaded program.
- Rearrange the layout so that the crystal oscillator chip does not interfere with N1.
- Think about adding a couple of LED's for signalling purposes.
- Put a 1 next to pin 1 of both the PIC and the crystal oscialltor on the component side.
- Think about making the crystal smaller.

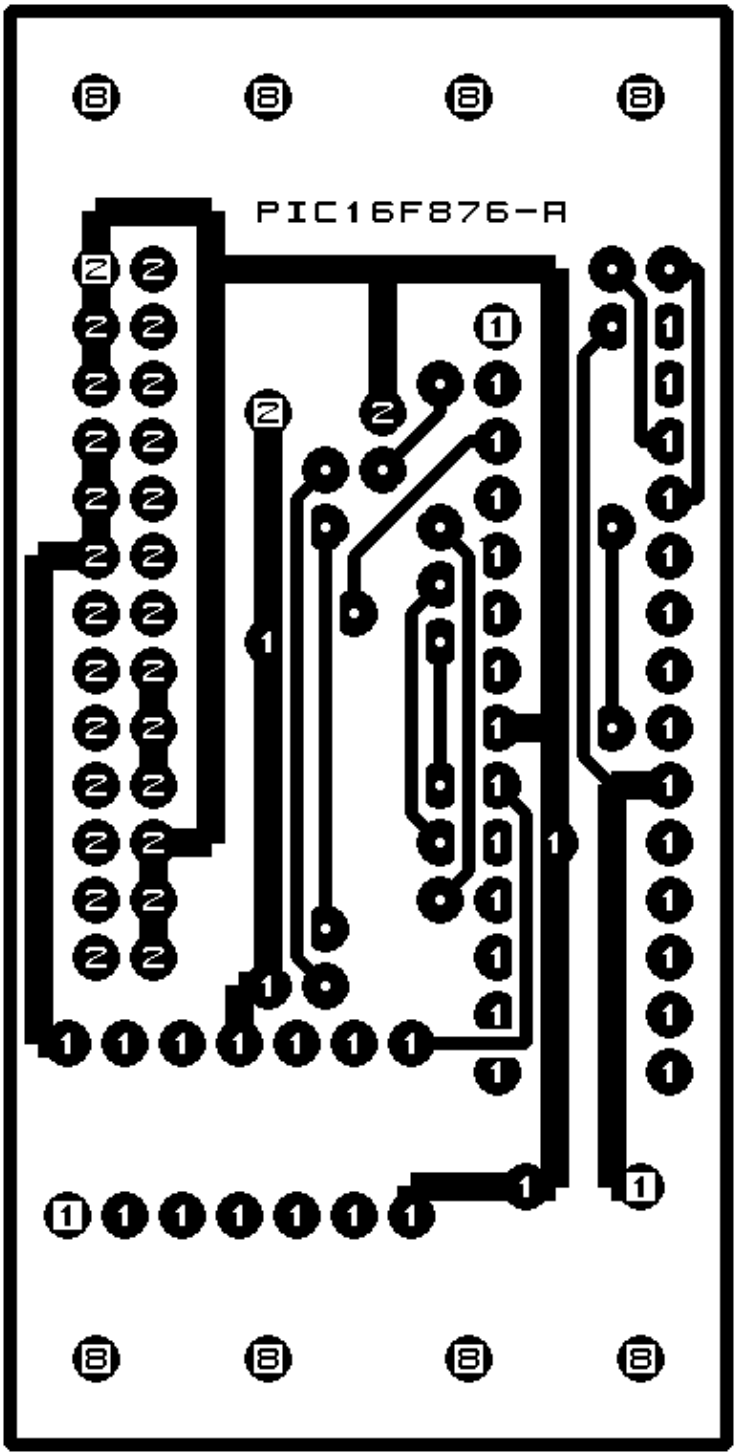
A. Appendix A: Parts List

```
# Parts list for PIC16F876 (Rev. A):
#
C1: Capacitor10pF − 10 pF Ceramic Capacitor [Jameco: 15333]
C2: Capacitor2200uF − 2200 uF 6.3V Electrolytic Capacitor [Jameco: 133145]
N1: Header2x13.Processor − 2x13 Male Header [26/80 Jameco: 117196]
U1: Oscillator20MHz − 20 MHz Crystal Oscillator [Jameco: 27932]
U2: PIC16F876.PIC16F876 −  MicroChip PIC16F876 [Digikey: PIC16F876A−20/P−ND]
```

B. Appendix B: Artwork Layer



C. Appendix C: Back (Solder Side) Layer



D. Appendix D: Front (Component Side) Layer