Affordable Comprehensive Transparent
PERSONAL MECHATRONICS LAB

Personal Mechatronics Lab DevBugger User Manual

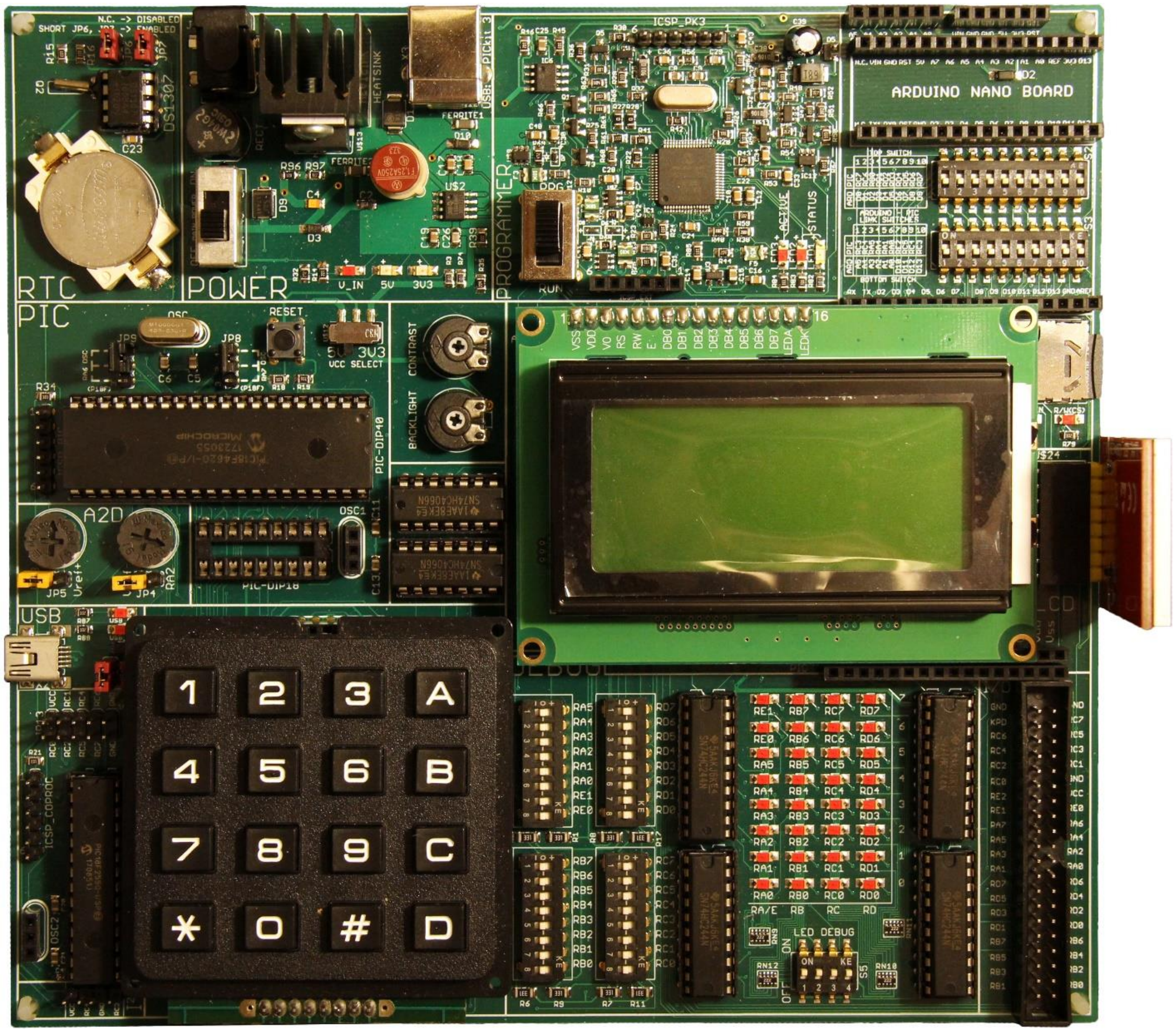


Table of Contents

ABBREVIATIONS

AC	Alternating current
COM	Communications
DC	Direct current
EEPROM	Electrically erasable programmable read-only memory
FAT	File allocation table
GLCD	Graphical LCD
GND	Ground
GPIO	General-purpose input/output
HAL	Hardware abstraction layer
HVP	High-voltage programming
I^2C or I2C	Inter-integrated circuit
IC	Integrated circuit
ICD	In-circuit debugger
ICSP	In-circuit serial programming
IDE	Integrated development environment
IPE	Integrated programming environment
I/O	Input/output
LCD	Liquid crystal display
LED	Light-emitting diode
LSB	Least significant byte
LSb	Least significant bit
MCU	Microcontroller unit
MSB	Most significant byte
MSb	Most significant bit
MSSP	Master Synchronous Serial Port
PC	Personal computer
PCB	Printed circuit board
PIC	A family of microcontrollers ("Peripheral Interface Controller") made by Microchip Technologies (hereon *Microchip*)
PLL	Phase-locked loop
QTY	Quantity
RTC	Real-time clock
SFR	Special function register
SD	Secure Digital
SPDT	Single-pole, double-throw
SPI	Serial peripheral interface
SSP	Synchronous serial port
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal serial bus
VAC	Volts AC
VDC	Volts DC

1. Introduction

1.1 Overview

The *PIC DevBugger* development board is a mobile platform for the development of robotic control solutions. This board is designed to suit the needs of students learning to develop programs for embedded systems. The form factor and features of the board allow it to be used both as an exclusive code development platform and in a final design as the main processing and control system.

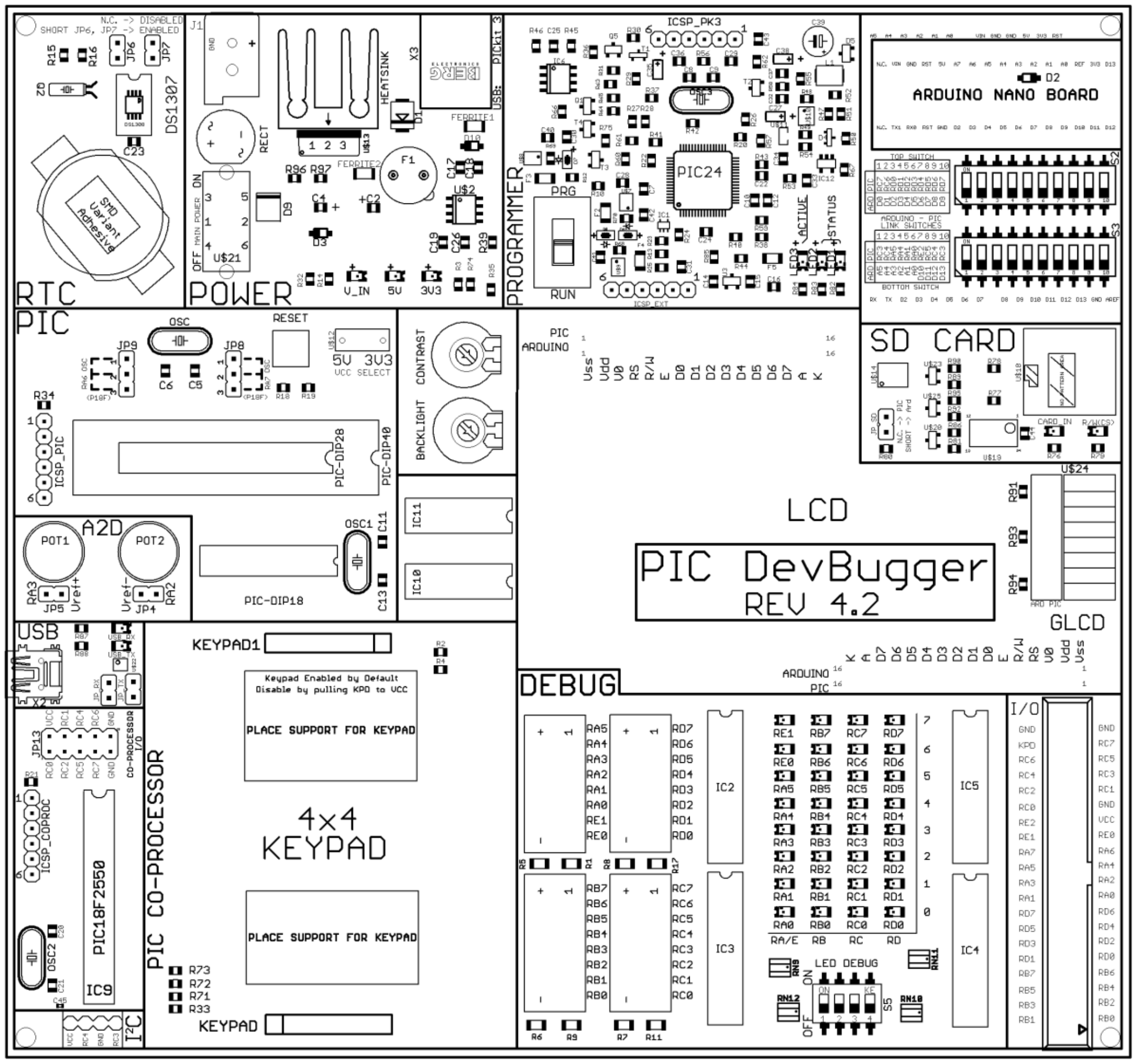


Figure 1. Simplified top-down view of the DevBugger

This document is intended to serve as an introduction to the *DevBugger*, and as comprehensive reference material to guide development. It is recommended that first-time readers scroll through this document to gain familiarity with the capabilities of the board and resources available, without feeling pressure to be immersed in the full details.

1.1 Features

- Open, modular design for learning purposes
- Supports 18-, 28-, and 40-pin devices from the PIC16 and 18 families
- PICkit 3 In-Circuit Debugger/Programmer (works with MPLAB X IDE)
- USB or DC power supply input (from 8 VDC to 45 VDC)
- Dual system voltage support for 70% of the onboard modules (5 V and 3.3 V system VCC)
- Debugging module with 32 indicator LEDs and signal-emulation switches
- Real Time Clock (RTC) peripheral with 32.768khz crystal and battery socket (RTC chip included)
- 40-pin input/output (I/O) bus with ribbon cable connector
- Interchangeable oscillator clock (10 MHz crystal included), with dedicated oscillator socket for 18-pin PIC
- Inter-integrated circuit (I^2C) bus expansion socket
- On-board adjustable analog to digital converter (ADC) voltage reference
- On-board 4x4 matrix keypad socket
- Co-processor with included keypad encoder and USB virtual communications (COM) port firmware. This chip is a full-featured PIC, whose firmware can be modified for extra memory, parallel processing, and/or I/O pin extension for the primary MCU
- Arduino Nano board interface, with individually selectable pin pass-through to PIC
- Arduino shield connector, with pin pass-through from both the PIC and Arduino
- On-board sockets for Hitachi HD44780 compatible LCDs (sizes up to 4x20 characters are supported), with contrast and backlight controls. Can be controlled by either the Arduino or primary PIC
- Graphical LCD (GLCD) sockets (1 for a PIC microcontroller, 1 for the Arduino Nano) compatible with 8-pin SPI-based GLCD controllers
- microSD card socket supporting PIC and Arduino Nano hosts
- Selectable Universal Asynchronous Receiver/Transmitter (UART) interfacing for the onboard microcontrollers

1.2 Included in the Box

The development kit includes the following items:

- PIC DevBugger development board
- CD/URL from which to download necessary software, drivers, and sample code
- USB A to B cable
- USB A to mini-B cable
- 40-pin I/O bus cable
- HD44780-controlled LCD display (4x16 characters)
- 4x4 matrix keypad
- AC-DC adaptor (120 VAC to 12 VDC, 1 A)
- ST7735R-controlled graphical LCD (128x128 pixels)

2. Operation

2.1 Operational Modes

The *DevBugger* has two modes of operation:

Programming

Used to load code onto the PIC18F4620 and debug the code execution if desired. To enter this mode, flip the PRG/RUN slide switch in the programmer module to the PRG position. In this mode, **RE3/$\overline{MCLR}$**, **RB6**, and **RB7** pins of the PIC18F4620 are disconnected from the main I/O bus and are connected instead to the programmer module. A PC application compatible with the PICkit 3 programmer, such as MPLAB X IDE or IPE, can be used to control code loading and debugging in this mode.

Execution

This is the main operational mode of the board. To enter this mode, flip the PRG/RUN slide switch of the programmer to the RUN position. In this mode, All I/O pins of the PIC18F4620 are connected to the I/O bus, and code executes freely. In this mode, the programmer module can be used as an independent PICkit 3 debugger/programmer for the onboard co-processor or external PICs. For more information, please see §3.3 Programmer.

2.2 Connecting to the PC for Programming

To load code to the PIC18F4620, the *DevBugger* must be connected to a PC:

1. Install the **MPLAB X IDE** and/or **MPLAB X IPE** distributed by Microchip
   a. The development board will be recognized as a PICkit 3 debugger/programmer by these environments when the USB port in the power module is plugged in
2. Use the included AC-DC adaptor to connect the board to power, then turn it on by sliding the main power switch to the ON position
3. Connect the *DevBugger* to the PC via the PICkit 3 USB port using the included USB A to B cable
4. Set the board to **Programming** mode (§2.1 Operational Modes)
5. Flip the power switch to the ON position (§3.1 Power)
6. The PC should detect the *DevBugger* and install the driver automatically.
7. The PIC18F4620 on the board is now ready to be programmed

2.3 Customizing Board Operation

The *DevBugger* was designed with versatility in mind. To customize the operation of the board, several configuration jumpers and switches have been included, which must be set by the user. More information about jumper settings for individual modules can be found in §3. Board Modules and Appendix B: Summary of Jumper Configurations.

A jumper is a set of 2 or more exposed pins that can be connected (shorted) using a "shunt". The example in Figure 2 below shows two configuration jumpers, **JP8** and **JP9**. If pins 1 and 2 on each jumper are connected using a shunt, then we say we have shorted pins 1 and 2. As is indicated on the silkscreen beside pins 1 and 2 ("OSC"), this configuration enables the PIC18F4620 to use the external oscillator block, at the expense of pins **RA6** and **RA7** as general-purpose input/output (GPIO).

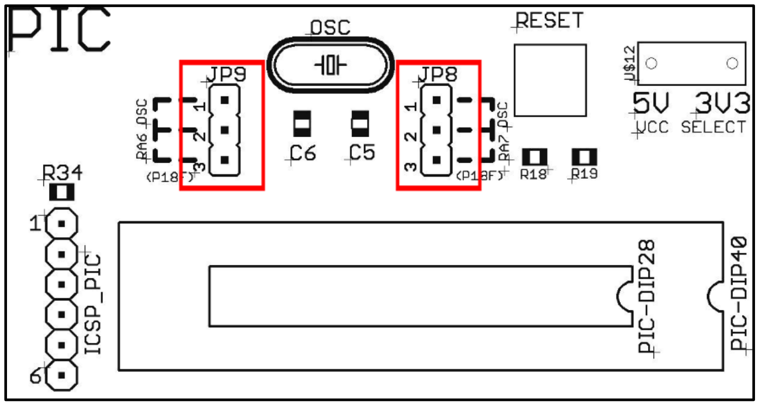


*Figure 2. Configuration jumpers for the PIC18F4620 in red. Users may choose to use pins **RA6** and **RA7** to connect to the external oscillator (10 MHz), or they may choose instead to use pins **RA6** and **RA7** as GPIO pins and use the 8 MHz internal oscillator block to control the execution of their code.*

2.4 Interfacing with External Circuits

The *DevBugger's* I/O bus connects to all pins of the PIC microcontroller when the *DevBugger* is operating in **Execution** mode. The I/O bus can provide some power at the working voltage of the board (selectable between 5V and 3.3V), but its primary purpose is for sending control signals and interfacing with other devices such as sensors. No high-power devices or circuitry (such as motors, fans, solenoids, or high-powered lights) should be powered from the VCC output on the development board—doing so may damage the board. Also, note that the keypad peripheral adds an extra pin to the bus (**KPD** pin), for on-the-fly enable/disable control, as described in §3.8 Matrix Keypad (4x4). A detailed description of the pin connections for the I/O bus is given in §3.4 I/O.

WARNING: The *DevBugger* should **not** be powered from the VCC pin on the I/O bus; no guarantee is made for its integrity in such use cases. Power should always be supplied through the DC input jack when the board is used as part of an embedded system (see §3.1 Power for details). The onboard power supply module will ensure delivery of clean DC power to all modules of the development board.

3. Board Modules

3.1 Power

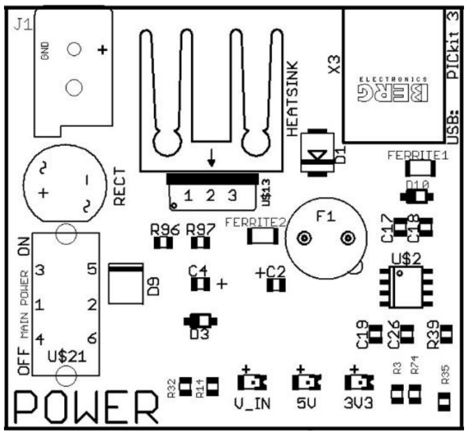


Figure 3. Power module

3.1.1 Description

The slide switch in the power module is the master ON/OFF switch for the board. This switch should be in the OFF position when connecting or disconnecting a power supply. When the switch is ON, the V_IN LED indicates the presence of input power from the DC jack or USB port.

The *DevBugger* should be powered using a DC supply between 8 V and 15 V. The input connector is a female 5.5×2.1mm jack, which is compatible with many commonplace adaptors. Wall adaptors of any polarity (center positive or center negative) may be used since the input is rectified. After rectification, the supply voltage is converted to 5 V by a LM338 linear voltage regulator, which is mounted on the large black heat sink. **The heat sink's voltage is equal to the output voltage (5 V)—it is NOT ground**. This regulator's 5 V output is routed to a 3.3 V regulator, and the VCC select switch in the PIC module is used to select which of these are used for the board modules.

The maximum current supplied by the power module is limited to 3.15 A in total, 3.3 V and 5 V rails combined. A fuse protects the power supply from operating above its current limit.

3.1.2 USB Supply

The USB power supply should only be used in conjunction with the DC power supply because USB ports cannot supply more than 500 mA of current, and the *DevBugger* does not regulate the voltage on this line. It is often closer to 4 V than 5 V, which can cause the PIC to reset "randomly" due to brown out (insufficient voltage), especially if attempting to drive external circuitry. Similar scenarios apply to other modules (especially the RTC which requires a 4.5 V input if not on the battery), and thus there are no guarantees about them working properly if powered solely by USB.

3.1.3 Tips

- If power is connected properly and the V_IN LED comes on, but no other LEDs light up, then the fuse blew and needs to be replaced

3.2 PIC

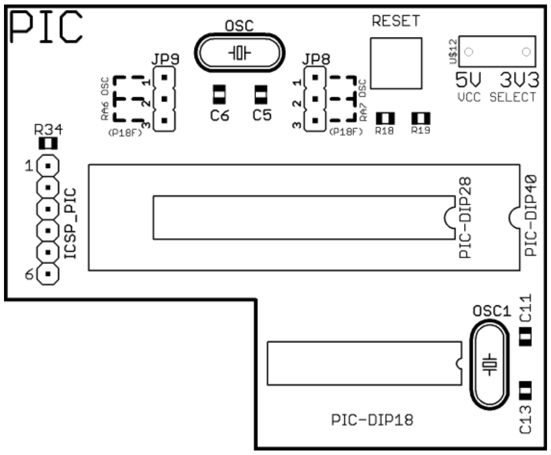


Figure 4: PIC Module

3.2.1 Description

The PIC module contains the PIC microcontroller that executes the main user program. The microcontroller included in the kit is a PIC18F4620. Throughout this document, the phrase *primary PIC* will be interchangeable with "PIC18F4620". Thorough knowledge of how this microcontroller works, as well as its capabilities and limitations, will be critical to the development of reliable programs. The datasheet is suggested as the primary source for this information. A few general points are summarized as follows:

- It has a processor, random access memory (RAM), non-volatile memory (flash and EEPROM), and interfaces with its environment through input/output (I/O) pins. It also has internal modules that perform specialized functions, for example, analog to digital conversion, and communication via I^2C. These modules are configured by writing to control registers called special function registers (SFRs)
- Instruction execution in the processor is clocked by an *oscillator*, which may be embedded in the microcontroller or may be part of external circuitry
- The PIC18F4620 (along with most microcontrollers) comes in many form factors, varying from surface-mounted versions the size of a pinky nail to the 40-pin dual in-line package (DIP) version that comes with the *DevBugger*. The DIP version was selected so that it would be easy for students to replace if it is damaged. The DIP version is mostly empty, like the device shown in Figure 5, below

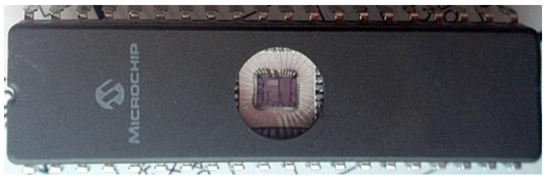


Figure 5. The small square of semiconducting material inside the window is called a "die" and contains the entirety of the circuits for the device. The die is connected to the pins on the DIP package using thin conductors barely visible to the naked eye called bonding wires.

This module of the board has several sockets for PICs of different sizes. Only one socket may be occupied at a time, otherwise bus conflicts will arise. The *DevBugger* is compatible with most PICs from the PIC16F and PIC18F families. Typically, these devices dedicate **RA6** and **RA7** for interfacing with an external oscillator, but some allow users to employ an internal oscillator and configure **RA6** and **RA7** as general purpose I/O pins. To give users of the *DevBugger* flexibility, jumpers **JP8** and **JP9** are present to select what **RA6** and **RA7** connect to. To use the external oscillator, short pins 1 and 2 on **JP8** and **JP9**; to enable **RA6** and **RA7** as I/O, short pins 2 and 3. Of course, the appropriate configurations must also be set in the user code.

A reset button is provided to pull down the **RE3/**$\overline{MCLR}$ pin, immediately causing a hard reset of the primary PIC. The precise consequences of this can be found in §4 of the PIC18F4620 datasheet, but the mains points are the program counter returns to address 0 of program memory, some registers remaining unchanged while others are set to default states, and all general-purpose input/output (GPIO) pins are set to *input* mode. This last point is important to note, because it means that **all the lines are floating while the reset button is held**, unless they are driven by external circuitry.

An in-circuit serial programming (ICSP) header is available to the left of the primary PIC so that any debugger/programmer that's ICSP-compatible can be used with it at any time. The pinout for this header is provided in Table 1, and is generic for all the ICSP headers on the *DevBugger*.

Table 1. ICSP header pinout

Pin number	Signal name
1	$\overline{MCLR}$
2	VCC
3	GND
4	PGED
5	PGEC
6	PGM

The VCC select switch allows users to select whether the system VCC is 5 V or 3.3 V. This switch influences the power supply of all the modules except power, RTC, programmer, and Arduino Nano. The *DevBugger* circuitry for all modules influenced by the switch are compatible with both supply voltages, however, care needs to be taken to ensure that the PIC and character LCD used are compatible with the selected voltage. **For the PIC18F4620, a system VCC of 5 V must be used to ensure desired operation**. The 3.3 V option exists for low-power PICs, which are a good choice for mobile applications since their power consumption is reduced. The character LCD provided in the kit must be powered at 5 V, but 3.3 V options exist in the market.

3.2.2 Tips

- The PIC18F4620 datasheet is a wonderful resource, and all users should refer to it to (at least) understand how to configure the registers for I/O and peripherals.
    - **TRISE** is not like the other data direction registers (**TRISx**) because the high bits control signals that will alter the behaviour of port D. Thus, it is suggested that

  - **TRISE<7:4>** are always cleared (i.e. 0) unless these upper bits are being purposefully set for a specific application.

- Pages 41 and 42 in the PIC18F4620 datasheet show all the possible ways resets can occur, and how the **RCON** register can be interpreted to know which type of reset occurred most recently
  - This is helpful if the device appears to be "randomly" resetting
- If PIC doesn't seem to be working properly, two good checks are: (1) make sure the PIC itself is completely socketed (a multimeter can be used to test continuity on each pin if the board is shut off), and (2) make sure the oscillator is completely socketed (again, continuity can be checked). It is not recommended to use an oscilloscope on the oscillator pins as it may stop the oscillator temporarily
- Free samples of the DIP-40 PIC18F4620 are available online from Microchip. It is good to order these early on in case the PIC ever needs to be replaced

3.3 PROGRAMMER

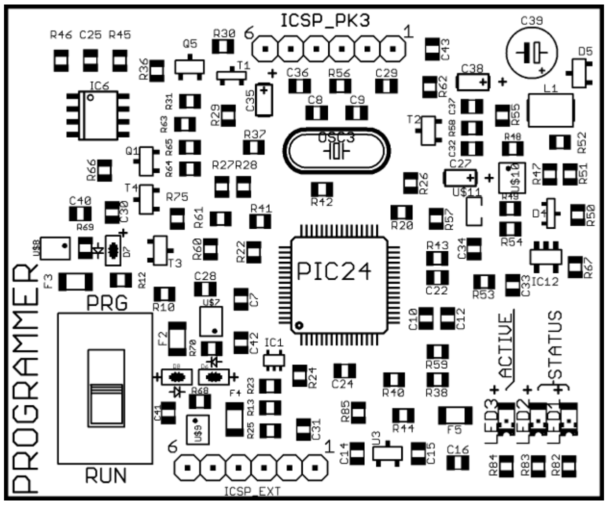


Figure 6: Programmer module

3.3.1 DESCRIPTION

The programmer module is the answer to the question, “how does code from my PC get transferred into the microcontroller’s program memory?!”. Microchip sells a device called *PICkit 3 In-circuit Debugger/Programmer* for this purpose, and the programmer module is a clone of it. In the middle of the programmer module is a PIC24 microcontroller which manages communication with MPLAB X via USB, as well as programming and debugging other PICs. Whenever you program code into the primary PIC, or start a debug session, the MPLAB X software transfers your binary executable to the PIC24 via USB. This executable is then transferred to the memory of the primary PIC through a protocol that uses its **RB6** and **RB7** pins, which is why these pins become unavailable when the PRG/RUN switch is in the PRG position.

Three LEDs indicate the status of the programmer. When the board is first powered on, all LEDs should illuminate. When the programmer finishes initialization and enters standby mode, both status LEDs will turn off leaving only the red ACTIVE LED illuminated to indicate the programmer is ready.

3.3.2 USING THE PROGRAMMER

To prepare the primary PIC for programming, the PRG/RUN switch in the programmer module must be in the PRG position, and to execute the code after it’s uploaded to the PIC, the switch must be in the RUN position. To debug code on the primary PIC, the PRG/RUN switch must be left in the PRG position for the entirety of the debug session. To program or debug external devices, such as the co-processor (see §3.7 Co-processor), the PRG/RUN switch must be in the RUN position.

3.3.3 HOW IT WORKS

The PIC24 executes the PICkit 3 firmware, which is an operating system that (1) receives commands from MPLAB software via the USB port in the power module, (2) decodes these commands, and (3) uses a scripting engine to execute them. The execution stage involves using the **PGD/PGED**, **PGC/PGEC**, and $\overline{MCLR}$ pins on the target device to erase, read, and write instructions/data. For the primary PIC, **RB6/PGD** is used to transfer of data, and **RB7/PGC** is used as a clock to synchronize the bits. The programmer operates in High Voltage Programming (HVP) mode given a supply voltage of 5 V. It incorporates an internal voltage converter that boosts the 5 V supply to the 12 V needed for HVP. HVP is necessary to generate the high erase current required to reset the flash program memory in target PIC devices. For complete details on how the PICkit 3 loads user code into target devices, please see Microchip's Flash Microcontroller Programming Specification.

3.3.3 TIPS

- If a connection error arises between the PICkit 3 and MPLAB, try closing MPLAB and then turning the *DevBugger* off and on. Additionally, ensure you are using the DC adaptor as opposed to USB if you experience connection issues (often USB cannot provide enough power)
- If MPLAB says "target not found" for the primary PIC, double-check that the PRG/RUN switch is in the PRG position
- Chapter 6 and section 2.7 from the document "PICkit 3 Programmer User Guide 2052116A" are recommended resources for understanding the problems that can arise with the PICkit 3 and how to troubleshoot them if need be
- While debugging, you can set watches on global variables and SFRs through MPLAB X

3.4 I/O

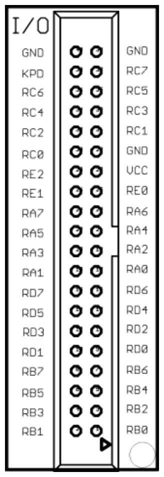


Figure 7: I/O module (top view)

3.4.1 Description

The input/output (I/O) module is intended to be the primary interface between the onboard modules and external circuitry. It features a 40-pin *bus* (collection of signals) that provides users with direct access to each I/O pin available on the PIC, as well as a special purpose pin, **KPD**, for enabling/disabling the keypad at runtime (see §3.8 Matrix Keypad (4x4)). To avoid damage to the PIC and peripherals, be careful to not present voltages above the system VCC or below 0 V to any of these pins. Additionally, many I/O pins are shared between the PIC and the other modules, as summarized in Table 2. Pins in bold can be disconnected from the module they correspond to via switches or jumpers.

Table 2. Pins used by Onboard Modules

Module	**Pins shared with PIC**
RTC	**RC3, RC4**
LCD	RD2, RD3, RD4, RD5, RD6, RD7
Keypad	**RB1, RB4, RB5, RB6, RB7**
A2D	**RA2, RA3**
Programmer	**RB6, RB7, RE3/$\overline{MCLR}$**
Arduino Nano	**RA0, RA1, RA4, RA5, RE2, RC7:3, RD7:0**
PIC	**RA6, RA7**
I^2C	RC3, RC4
Debug	**RE1:0, RA5:0, RB7:0, RC7:0, RD7:0**
SD Card	**RE2, RC3, RC4, RC5**
GLCD[1]	RD0, RD1, RC3, RC5
USB	**RC6, RC7**

[1] As long as the chip select pin is high, the other 3 pins can be used as GPIO while the GLCD is plugged in

3.5 DEBUG

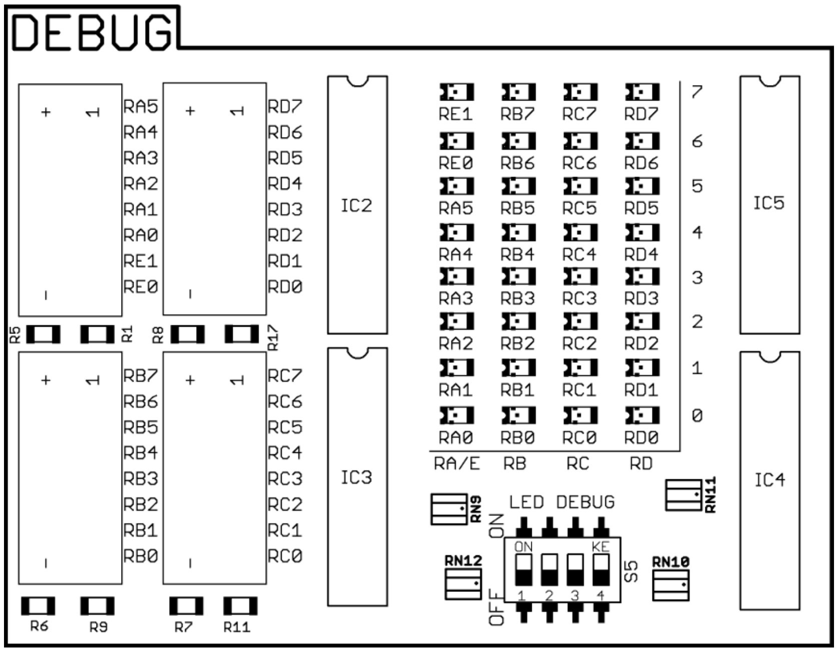


Figure 8: Debug module

3.5.1 DESCRIPTION

The debug module allows users to monitor the state of each I/O pin of the primary PIC through 32 indicator LEDs. Each column of the LED array can be enabled/disabled by toggling the switches in the “LED DEBUG” DIP switch below the array. These indicator LEDs are fully buffered, meaning they induce minimal load on the signal lines they are monitoring. However, unless each line is connected to an external signal or is driven by the I/O ports, its **indicator LED may change unpredictably since it is floating** (recall: a driving circuit is some circuit that enforces a voltage across an electrical connection—perhaps at a logical low value, logical high, or somewhere in between). It is a good design practice to not leave pins floating. Also recall from §3.2.1 Description that pins are floating while the PIC is being reset.

The *DevBugger* also comes with 32 DIP switches to simulate digital inputs to the pins of the primary PIC for debugging purposes. Each switch corresponds to a pin on the primary PIC as indicated by the adjacent labels. Each DIP switch has 3 positions as summarized in Table 3 below, and the default position is the middle, which leaves its corresponding pin **floating unless driven**. This is the perferred state if the user wishes to control the pin state using the PIC or external circutry connected to the I/O bus. When using the debug switches, **make certain the corresponding pins are set as inputs**, otherwise the indicator LEDs may give misleading results and components could heat up.

Table 3. Effect of Debug Module DIP Switch Positions on Pin State

	Position		
	Left	**Middle**	**Right**
Physical change	Pin connected to strong pull-down resistor (330 Ω)	Pin disconnected from pull-up/pull-down resistors	Pin connected to strong pull-up resistor (330 Ω)
Effect on pin state	Pulled down to ground	No effect	Pulled up to VCC

3.5.2 Tips

- The LEDs in the debug module are intended to be used to view inputs and outputs and cannot be trusted when left floating (i.e. leaving inputs undriven). This is the reason why some pin states may seem to "influence" each other
- **The switches are intended to simulate inputs, and should not be used to pull a pin up or down while it is being driven by the PIC or external circuit**
- The R/W (CS) LED in the SD card module can be used to monitor the state of the RE2 pin since it does not have a LED in the debug module, provided that the PIC is selected as the SD card host
- There are no debug LEDs for RA7–6 on the board because users tend to dedicate them to the external oscillator block; as such, external circuitry must be used if the states of these lines are to be monitored. A suggested schematic is provided in Figure 9, below. 1 LED and 220 Ω resistor are required for each line as well as a 74HC244N buffer.

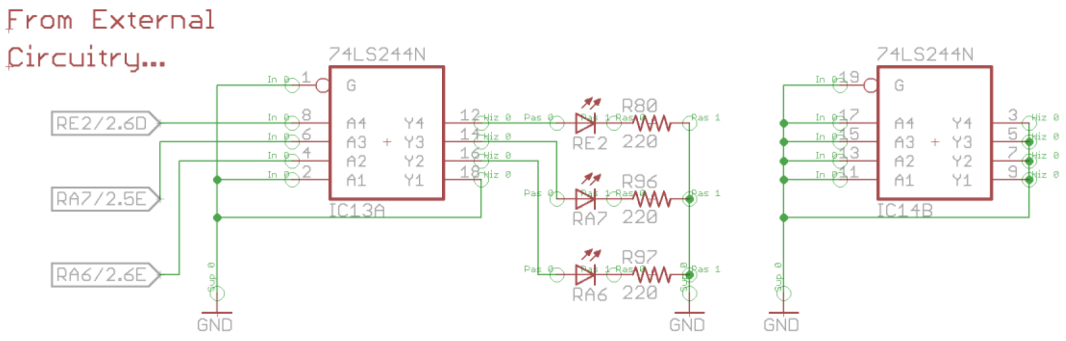


Figure 9. Suggested schematic for monitoring states of RA7, RA6, and RE2. Note that this schematic does not show VCC (the datasheet for the buffer should be consulted before constructing the circuit).

3.6 LCD

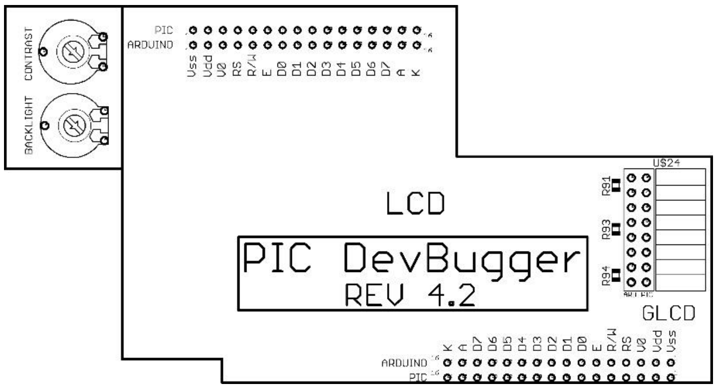

CONTRAST
BACKLIGHT
PIC
ARDUINO
Vss Vdd V0 RS R/W E D0 D1 D2 D3 D4 D5 D6 D7 A K
LCD
PIC DevBugger
REV 4.2
U$24
R91
R93
R94
GLCD
K A D7 D6 D5 D4 D3 D2 D1 D0 E R/W RS V0 Vdd Vss
ARDUINO
PIC

Figure 10. LCD module

3.6.1 Description

This module provides options for two different types of liquid crystal displays (LCDs) that can be used in user interfaces. A character LCD consists of character segments used solely for displaying text, while a graphical LCD uses an array of pixels to display arbitrary images (which may happen to be text).

The module is equipped with 2 equivalent sets of headers upon which 16-pin character LCDs can be mounted. The 2 sets have been provided to support different orientations of the LCD (up- or down-facing). The character LCD included in the kit has 4 rows, each with 16 visible characters, so we say it is 4x16. The outermost headers connect to the PIC data lines and the innermost headers connect to the Arduino Nano data lines. The *DevBugger* has been configured to use 4-bit data transfer modes in the interest of conserving I/O pins (Table 4).

Table 4. Character LCD Pins

Signal name	Pin on PIC	Pin on Arduino Nano
RS (register select)	RD2	D4
Enable	RD3	D5
D4	RD4	D6
D5	RD5	D7
D6	RD6	D8
D7	RD7	D9

This module is also equipped with a set of right-angle headers upon which an 8-pin graphical LCD can be mounted. The bottommost header connects to the PIC data lines while the topmost header connects to the Arduino Nano data lines. The GLCD included in the kit uses serial peripheral interface (SPI) for communication, which uses 4 signals as described in Table 5. These signals must not exceed 3.3 V when they reach the GLCD, or else they will damage it. To ensure safe voltages for the signals, the DevBugger incorporates level shifter circuits, which translate signals from one voltage range to another without changing their waveform in any other way.

Table 5. Graphical LCD Pins

Signal name	Pin on PIC	Pin on Arduino Nano
Serial clock (SCK)	RC3	D13
Master Output Slave Input (MOSI)	RC5	D11
Chip select ($\overline{CS}$), active low	RD0	D2
Register select (RS/A0)	RD1	D3

The character and graphical LCD sockets have independent control signal lines, as described in the tables above. **The backlight potentiometer is shared between these modules, however**. There is a position for this potentiometer just before the maximum brightness that produces an optimal setting for both the character LCD and graphical LCD. **If you see the graphical LCD flickering, you should decrease the brightness as you may be damaging it.**

3.6.2 GLCDs Supported in the Sample Code

The sample code for the PIC was developed using the GLCD included in the kit, so if you use this you should be safe. This GLCD has a ST7735R controller inside it and uses a 1.44” 128x128 pixel thin-film transistor (TFT) LCD display. Figure 11 shows a front and rear view. Also, note the series resistor on the LED line: this is important to ensure the backlight is not prematurely burned out. Version 2.1 of this display (indicated on the red PCB of the display) tends to include this resistor, so students should try to purchase this version.

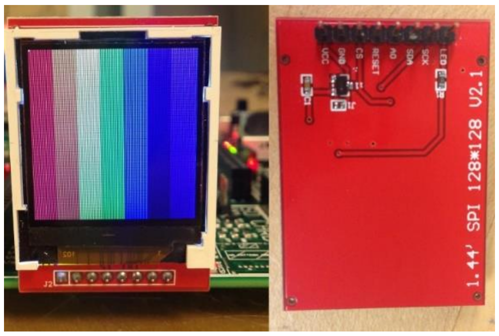
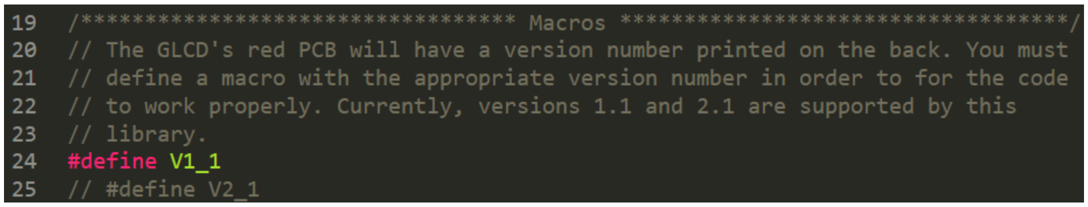


Figure 11. Front and rear view of the GLCD. Note the series resistor on the “LED” net (rightmost pin).

Versions 2.1 and 1.1 are supported by the sample code, and the only difference is a subtle change in the address boundaries of the display. As Figure 12 shows, you choose between these versions by defining a version-specific macro in GLCD_PIC.h.



```
/********************************** Macros **********************************/
// The GLCD's red PCB will have a version number printed on the back. You must
// define a macro with the appropriate version number in order to for the code
// to work properly. Currently, versions 1.1 and 2.1 are supported by this
// library.
#define V1_1
// #define V2_1
```

Figure 12. Choosing GLCD version in code (GLCD_PIC.h)

3.7 Co-processor

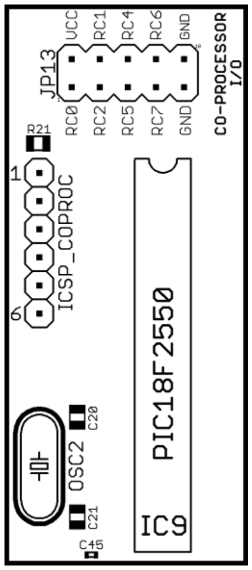


Figure 13. Co-processor module

3.7.1 Description

The co-processor module features a PIC18F2550 with its own dedicated ICSP header, oscillator socket (oscillator not included with the *DevBugger*), and I/O bus. The co-processor is just like the PIC18F4620 in that it has its own program memory, RAM, and onboard modules, except it is in a 28-pin package, features a slightly different internal architecture, and by default, runs specialized firmware to help users with development. By default, the co-processor can boot into four programs on power-up:

1. Keypad firmware that outputs keypress data to the primary PIC's lines, using **RB7:4** for data and **RB1** as the data available line. **RB1** can be configured on the primary PIC for interrupt on-change. This is referred to as "keypad encoder (port B)"
2. Firmware that runs the keypad routine explained above while simultaneously running USB firmware that facilitates PIC-PC communication via UART (virtual COM port)
3. Keypad firmware that outputs keypress data onto the UART bus. Either the primary PIC or Arduino, or both, can receive these transmissions
4. User-defined code (which may include any or all the above)

The selection of these programs depends on which key on the keypad is pressed during power-up, as summarized below in Table 6. The following sections describe each of these programs.

Table 6. Co-processor Boot Options

Action Taken	**Program loaded**
Did not press a key, or pressed one with no designated function	Keypad encoder (port B)
Pressed key 1	Keypad encoder (port B) & Virtual COM port (USB to serial converter) running in parallel
Pressed key 2	Keypad encoder (UART)
Pressed key 3	User-defined code

3.7.2 Program 1: Keypad Encoder (Port B)

The default keypad encoder program converts the 8-line matrix input from the keypad into simple-to-parse 4-bit binary data. Key 1 is encoded as 0x00, key 2 as 0x01,…, and key D as 0x0F. When a key is pressed, the co-processor latches this 4-bit data onto the lines it shares with the primary PIC's **RB7:4**, then sets the "data available" pin, **RB1** on the primary PIC. **RB4** is the least-significant bit (LSb) and **RB7** is the most-significant bit (MSb). While no key is being pressed, **RB7:4** are left floating and **RB1** is asserted low. To disable the keypad and free up **RB1** and **RB7:4** for GPIO, short the **KPD** pin on the I/O bus to VCC.

- When the keypad is disabled the co-processor changes the data direction on its pins to input (high impedance). When this happens, the bus connecting the two PICs (i.e. **RB1**, **RB7:4** on the primary PIC) will be floating unless otherwise driven to some voltage either by one of the PICs or external circuitry.

3.7.3 Program 2: Keypad Encoder (Port B) & Virtual COM Port

In this program, the keypad encoder routine explained in the previous section runs concurrently with virtual COM port firmware on the co-processor, which is used to create a port for PC communication. The functionality of the keypad encoder and COM port are independent even though they run simultaneously, and as such, the virtual COM port will be explained in this section as though is a separate program.

Important Note on the Co-processor Clock

An external oscillator in the OSC2 socket (beside the co-processor) is required for USB communication to comply to the timing tolerances in the USB specification. However, this external oscillator is *only* used to drive the USB hardware module in the PIC18F2550; all other modules in the co-processor are clocked by the internal 8 MHz oscillator (see page 24 of the PIC18F2550 datasheet for the clock diagram). Thus, since the virtual COM port feature is optional, the co-processor does not require an external oscillator unless it is running this program.

- By default, the co-processor requires a 20 MHz oscillator for the virtual COM port program (i.e. a 20 MHz oscillator can be put into the OSC2 socket and the virtual COM port will then work without any other actions)
- Users can modify the clock settings so that a different oscillator, such as 12 MHz, can be used (note that 12 MHz is the minimum):
    - In "configBits.c" and "pdusb_2550.c", change the line "#pragma config PLLDIV = 5" to the value that's correct for your oscillator
    - OSCCON may need to be changed in "system.c" and "machineConfig.c"
    - In "usart.h", CLOCK_FREQ must also be changed to match the value of _XTAL_FREQ in "configBits.c"
- Internally, the signal from the oscillator is run through several dividers and a 24x phase-locked loop (PLL) to produce a 48 MHz clock for the USB module and 24 MHz clock for the rest of the system.
- The USB module also contains a 3.3 V regulator to produce the voltage levels required for the USB data lines, and the *DevBugger* has a 220 nF capacitor to stabilize the regulator.

Two use cases for the virtual COM port are described as follows:

Use case 1: Sending data from a microcontroller on the *DevBugger* to a PC

1. User sends data to the co-processor via the UART bus
2. Co-processor receives the bytes, loading them into a buffer for USB transmission
3. Bytes transmitted via USB to PC

Use case 2: Sending data from a PC to a microcontroller on the *DevBugger*

1. PC sends data to the co-processor via a USB packet
2. Co-processor receives packet, interprets it, and loads the data fields into a UART buffer
3. Bytes transmitted via co-processor's UART TX line

A note on the software, for the curious:
The virtual COM port program uses the USB hardware inside the PIC18F2550 to implement the communications device class (CDC) of the USB protocol. Most modern-day operating systems come with CDC drivers which is one reason why this device class was selected. Furthermore, this device class allows users to communicate with the co-processor using a serial monitor program as if it were a RS-232 port (hence the term virtual serial port). On the co-processor, the software that takes care of the USB protocol is called the USB stack. The USB stack is a collection of data structures, variables, and functions organized into separate files, with each file implementing a certain detail related to the USB protocol. One file, called the hardware abstraction layer (HAL) contains the definitions for the functions directly involving the hardware. Once the hardware is "wrapped" in software interfaces, the rest of the implementation details become generic, that is, compatible with any hardware if the low-level interfaces in the HAL are implemented. **Note how the term "low-level" implies a "layering" within the software.** The other files build on top of this low-level "layer", and as the layers grow, the functions become increasingly *abstract,* which is necessary for enabling application development. Thus, the set of functionalities implementing the USB protocol is called a "stack", because of the layered nature of it. As a final note on the software details, the entire routine for the virtual COM port is interrupt-based, which allows for concurrent handling of the USB protocol, data transmission, and data reception.

3.7.3 Program 3: Keypad Encoder (UART)

When the co-processor is booted into its UART keypad encoder mode, it runs the same code to poll the keyboard for key presses but transmits the key press data via 9600/8N1 UART. The way this is achieved is as follows:

a. When no key is pressed:
   - The co-processor's UART transmitter is in the idle state (logical high)

b. When a key is pressed:
   - The co-processor sends the byte corresponding to the keypress (0x00 for '1', 0x01 for '2', 0x02 for '3', 0x03 for 'A', etc.)

- While the key is still pressed, the co-processor continuously sends the byte corresponding to the keypress. When the key is released, the co-processor's UART transmits 0xF0 to signal the end of data transmission, then returns its transmitter to the idle state

Note: When the co-processor runs this program, it sets its data direction on port A to high impedance (input) for all its pins except RA4. RA4, which is connected to **KPD**, is set high through software, and thus **RB1** and **RB7:4** of the primary PIC will be free to use as GPIO.

3.7.5 Program 4: User-Defined Code

The original working files for the firmware are available so that user-defined code can be added. Recall from §3.3 Programmer that the onboard PICkit 3 can act as an independent programmer when the PRG/RUN switch is in RUN mode and can therefore be used to program the co-processor through its ICSP header. To do this, use jumpers to connect pins 1–6 of ICSP_EXT in the programmer module to pins 1–6 of ICSP_COPROC in the co-processor module. Each pin should be connected to the one of identical number (i.e. pin 1 of the programmer's header connects to pin 1 of the co-processor's header). Note that Table 1 shows a detailed pinout of the ICSP headers.

One of the reasons users may be interested in running their own code on the co-processor is it has 7 GPIO when the USB data lines (**RC4** and **RC5** on the co-processor) are not in use, and 5 GPIO otherwise. These GPIO pins are not present anywhere else on the board.

The pins that are shared between the primary PIC and co-processor are summarized in Table 7, below.

Table 7. Connections shared between primary PIC and Co-processor

Pin on primary PIC	Pin on Co-processor	Default function
RB1	RA5	Keypad data lines
RB4	RA0	
RB5	RA1	
RB6	RA2	
RB7	RA3	
RC6 (UART TX)	RC7 (UART RX)	UART bus
RC7 (UART RX)	RC6 (UART TX)	

The connections between the two PICs' **RC6**/**RC7** can be enabled in hardware by shorting jumpers **JP_TX** and **JP_RX**. **JP_TX** is for the co-processor's UART transmitter, **RC6**, and for the primary PIC's UART receiver, **RC7**. This is discussed further in §3.15 USB.

3.7.6 Tips

- The co-processor will determine which program to load into within approximately 30 milliseconds after power is turned on. Thus, if it is desired to use a boot key, it must be pressed before and while the power is being turned on

- By default, the co-processor must be reset, or power cycled (i.e. turned off, then on) to access the boot selection code. It can also be reset by connecting its $\overline{MCLR}$ pin on the ICSP_COPROC header to ground

3.8 Matrix Keypad (4x4)

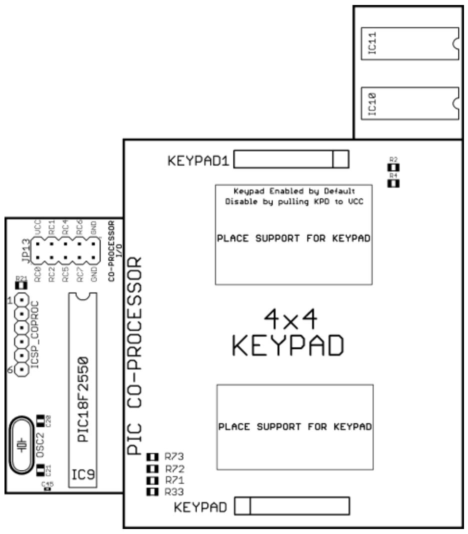


Figure 14: Keypad module with the co-processor, used to encode keypad data, visible on the left. Note the two 74HC4066N switches in the top-right corner, controlling the primary PIC's connection to the I/O bus for the keypad data pins.

3.8.1 Description

4x4 matrix keypads are commonly used in applications involving microcontrollers as a simple user interface. Two equivalent headers on the DevBugger provide sockets for the keypad in up or down orientations, to match the character LCD orientations. To handle the task of checking the keypad for key presses, the PIC18F2550 microcontroller in the co-processor module runs a keypad encoder program by default, as explained in §3.7 Co-processor.

When the co-processor is running its default keypad encoder program, the keypad module can be disabled on-the-fly through the special **KPD** pin on the I/O bus. If **KPD** is set high, the keypad will be *disabled*, and if left floating, the keypad will be *enabled*. When the keypad module is enabled, two 74HC4066N switch chips isolate the keypad data pins from outputs on the PIC I/O bus. Disconnecting the keypad data pins protects them from interference from external circuitry and increases the reliability of keypad inputs. When **KPD** is high, the co-processor will set its output pins to high impedance mode (disconnected), and **RB1** and **RB7:4** of the primary PIC will be made accessible to the user. Note that **KPD** can be controlled either by external circuitry or directly by the primary PIC by connecting it to one of the ports on the I/O bus.

When the co-processor is running the UART keypad encoder program, it sets its output pins to high impedance mode (input) and sets **KPD** high through software, thus permitting the PIC18F4620 to access the I/O bus with **RB1** and **RB7:4**.

3.8.2 Tips

- Keys should be pressed gently and only one at a time
- **KPD** should only either be left floating or driven high. Users who would like to drive it low should exercise caution because the co-processor sets **KPD** high through software when operating in UART mode

3.9 A2D

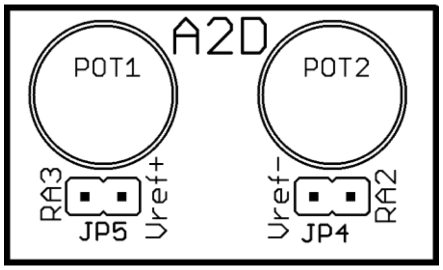


Figure 15: A2D module

3.9.1 Description

Working with embedded systems often involves acquiring data from sensors, many of which transmit signals between logical low (i.e. 0 V) and logical high. To interpret these intermediate voltage levels accurately, the PIC18F4620 uses its internal analog to digital converter (ADC) module to convert some voltage V_{IN} to a 10-bit number according to the following formula:

$$0bxxxxxxxxxx = \frac{2^{10}-1}{V_{ref+}-V_{ref-}}\left(V_{IN} - V_{ref-}\right),\ V_{IN} \in [V_{ref-}, V_{ref+}] \qquad (1)$$

By default, the lower voltage reference, V_{ref-}, is set to ground and the higher voltage reference, V_{ref+}, is set to VCC (the supply voltage for the PIC). Thus, by default, the ADC will convert input values between 0 and 5 volts to a 10-bit number with resolution equal to $\frac{5\,V}{1023} = 0.005\,V$. Formula 1, above, shows how if the voltage references are used to narrow the voltage window, then finer resolution measurements can be taken. For example, if $V_{ref-} = 2\,V$ and $V_{ref+} = 4\,V$, then the resolution is $\frac{2\,V}{1023} = 0.002\,V$, but over a smaller range.

The *DevBugger* has been equipped with two potentiometers to set the voltage reference levels for the PIC's ADC module. To enable voltage references, the appropriate registers in the PIC need to be configured (e.g. ADCON1, ADCON0, TRISA, etc.), as well as jumpers **JP4**/**JP5** as follows:

- Short **JP4** to enable V_{ref-} on **RA2**
- Short **JP5** to enable V_{ref+} on **RA3**

Note that any pin configured as a voltage reference must be dedicated to that functionality and cannot be used as I/O.

3.9.1 Limitations

Table 26-24 in the PIC18F4620 datasheet summarizes characteristics to be aware for the ADC module. The most important are outlined below:

- $V_{ref+} - V_{ref-}$ must not be less than 1.8 V
- V_{ref+} must not exceed the supply voltage of the PIC (i.e. VCC) by more than 0.3 V
- $V_{ref-} - ground$ must not be any less than 0.3 V
- $V_{ref+} \geq V_{ref-}$
- $V_{ref+} \geq ground$

3.10 Real Time Clock (RTC)

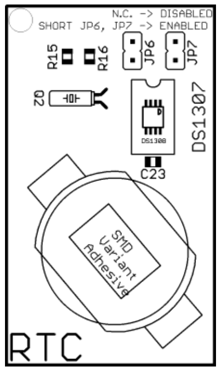


Figure 16: RTC module

3.10.1 Description

The DS1307 RTC and CR2032 coin cell battery are a means of timekeeping, even when the rest of the board is unpowered. This can be used to free the microcontroller from the task of tracking long-term durations so that it may focus on other tasks. It can also be used to record events with a time stamp. To enable access to the module, jumpers **JP6** and **JP7** must be shorted to enable interfacing through I^{2}C on pins **RC3** and **RC4** (these jumpers are already shorted by default).

Note: To ensure consistent operation, users should refrain from removing the coin battery unless a replacement is required.

3.10.2 Using the Real Time Clock

The DS1307 can be used to keep track of time in seconds, minutes, hours, days, months, and years. The chip accounts for the number of days in each month, including leap years. The hours feature allows the chip to keep time in either 12 or 24-hour format with an a.m./p.m. indicator for the 12-hour format.

The RTC communicates through the I^{2}C protocol and acts as a slave device with a 7-bit address of `0b1101000x` (where `x` is the $R/\overline{W}$ bit, which denotes if the transaction is a read (`x` = `1`) or a write (`x` = `0`) in an I^{2}C transaction). In order to use the RTC, the primary PIC's master synchronous serial port (MSSP) module must be configured for operation as a master device for I^{2}C. The master device is responsible for initiating and controlling transactions for all slave devices, including the RTC. One of the most important parameters the master configures is the frequency of the clock it will generate to synchronize the transfer of data. The DS1307 datasheet specifies that a clock frequency of 100 kHz must be used.

To write to control registers of the RTC, the following steps can be used:

1. Start the MSSP module with the desired I^{2}C configuration parameters
2. Write the slave address on the I^{2}C bus, appended with a zero to indicate a write operation
3. Write the 8-bit address of the register to begin writing data
4. The data intended to be written to the registers can be transmitted sequentially on the I^{2}C bus. The DS1307 automatically increments the register address pointer after receiving

each byte of data, so there is no need to restate each register address for successive writes

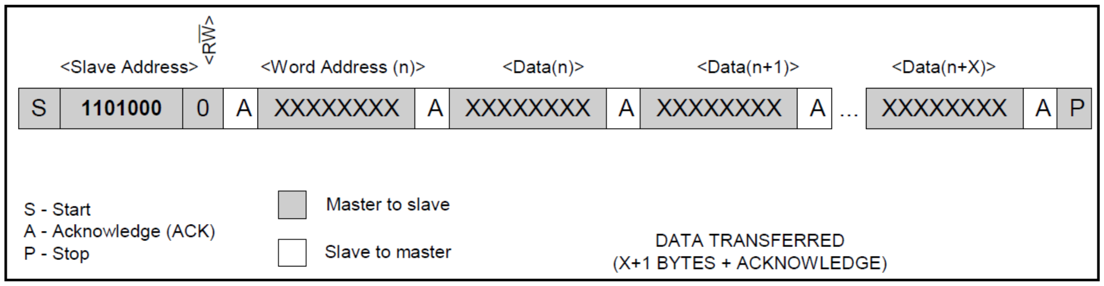


Figure 17: Dissection of a write transmission to the DS1307

Note: The RTC stores date and time values in binary coded decimal (BCD) form in its registers. For example, to set a time of 2018 December 31 Sunday 23:59:45, simply transmit the following bytes to the RTC, starting with the first element in the array:

```
{‘0x45’,’0x59’,’0x23’,’0x00’,’0x31’,’0x12’,’0x18’}
```

3.10.3 Tips

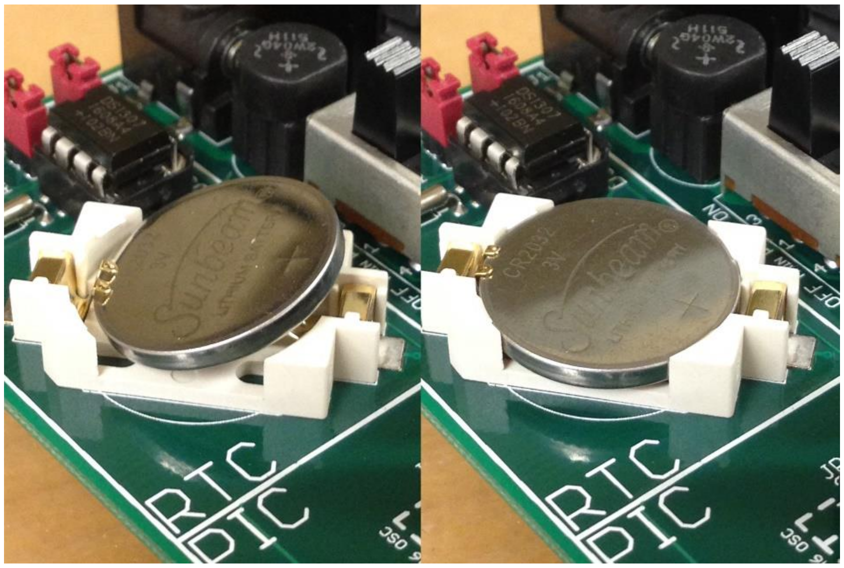


Figure 18. RTC battery replacement

- If the RTC battery requires replacement, care must be taken to ensure the battery is inserted into the holder properly as it is a fragile part. The battery must first be placed below the bent metal prong, as shown on the left-hand side of Figure 18, then the rest of the battery can then be pushed into place. To safely remove the battery, use one hand to pull the prong away from the battery and the other hand to lift it up while applying pressure down onto the holder
- If RTC issues are consistently experienced, it is suggested that users check the voltage across the battery using a voltmeter by connecting one lead on the bottom of the battery

and one lead on the top. A voltage of 3.3 V is ideal, and batteries much less than this (e.g. 1 V) should be replaced

- The RTC may not operate reliably if the *DevBugger* is powered solely by USB. The DC power supply should always be used. This is because the DS1307 switches the primary power source from the battery to VCC when VCC exceeds V_{bat} + 0.2 V, but only becomes responsive to commands when VCC exceeds 1.25 times V_{bat}. This number is 4.13 V for a 3.3 V battery, as is included with the *DevBugger*, and this is not reliably achieved with a USB power supply
- If the RTC is counting by any number other than ones (e.g. by fours or sixes), lightly touch your finger to the 32.768 kHz oscillator just to the left of the RTC while the power is on, and it should immediately count properly again until power is removed. Usually this fixes the problem forever since most people leave the battery in. If this does not remedy the situation, it is possible that the oscillator and divider circuit in the DS1307 are compromised, and it is suggested that a new DS1307 be used

3.11 I^{2}C

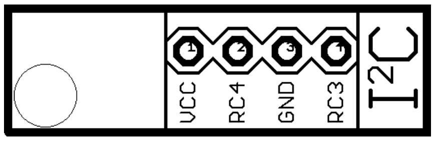


Figure 19: I2C module

3.11.1 Description

A port has been provided for users to connect devices that use the inter-integrated circuit (I^{2}C) protocol to the I^{2}C bus on the DevBugger. The port has the I^{2}C clock pin (**RC3**) the I^{2}C data pin (**RC4**), a ground reference, and a supply pin (VCC) that can be used for low-power devices.

When the DevBugger is operating in I^{2}C mode, **RC3** and **RC4** are “open-drain”. This means that the PIC can force two states for these pins: (1) logical low (i.e. 0 V), and (2) floating; it *cannot* drive the pins high. Thus, I^{2}C busses require pull-up resistors, to ensure that the lines are always high unless driven low. The DevBugger includes 10 $k\Omega$ pull-up resistors on **RC3** and **RC4** in the RTC module. Thus, they part of the I^{2}C bus when **JP6** and **JP7** are shorted. If users desired to use the I^{2}C bus without the RTC module, then pull-up resistors must be added through external circuitry. Note that only one set of pull-up resistors should be used on the I^{2}C bus.

3.12 Arduino Nano

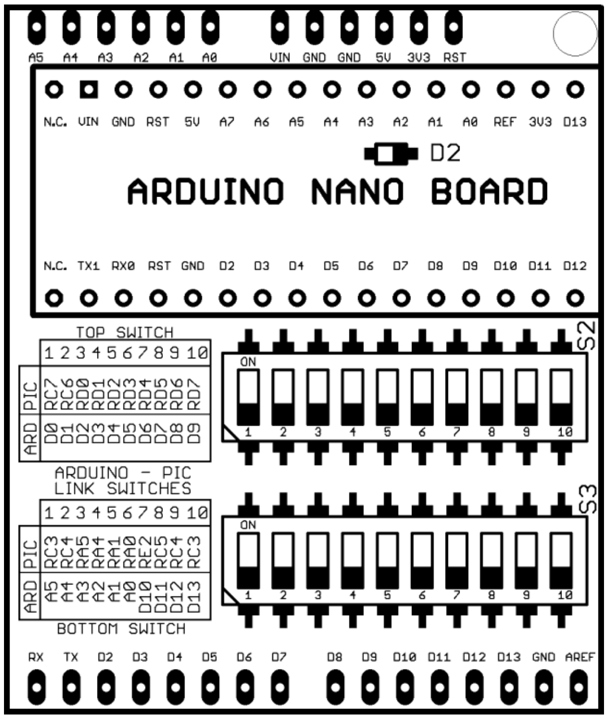


Figure 20. Arduino Nano module

3.12.1 Description

This module provides a dedicated interface for an Arduino Nano, as well as boards used to extend its functionality, called shields. Each pin for the Arduino Nano is directly connected to the corresponding pin on the shield at all time, thus, this module could be used simply as a shield adaptor.

Two switch banks, S2 and S3, are used to determine which pins from this module (if any) are connected to the pins on the primary PIC, which creates opportunities for communication between the microcontrollers. By default, all the switches are OFF, which isolates the I/O for this module from the rest of the *DevBugger*, allowing it to operate completely independently. When a switch is turned ON, it creates the connection indicated in the table to the left of the switch.

> ***Example:***
> *Suppose that the leftmost switch, switch 1, on switch bank S3 is turned ON by moving it upwards, toward the "ON" text at the top of the switch bank. The table to the left of switch bank S3 would be consulted, and the user would see that this connects Arduino pin A5 to PIC pin RC3. To undo this connection, the switch would simply have to be put in the OFF position.*

The connections between Arduino pins and PIC pins are set in the *DevBugger* PCB and thus cannot be changed without external circuitry. However, the pin connections have been designed in such a way that the PIC and Arduino are able to communicate through various busses including Serial

Peripheral Interface (SPI), Universal Asynchronous Receiver/Transmitter (UART) and inter-integrated circuit (I^2C).

Note: The ATmega chip used by the Arduino Nano has separate pins for SPI and I^2C, while the PIC18F4620 does not. **RC3** and **RC4** are used by the MSSP module, which is either configured for I2C or SPI communication at any given time. This is why **RC3** and **RC4** are duplicated in the link switches; one connection is to the Arduino Nano's I^2C pins, and the other is to the Arduino Nano's SPI pins.

A summary of the pin connections is described in Table 8, below.

Table 8. PIC-Arduino Connections

Pins		**Function(s)**	**Pins**		**Function(s)**
Arduino Nano	**PIC**		**Arduino Nano**	**PIC**	
A5	RC3	I^2C/SPI, GPIO	D0/RX0	RC7	UART, GPIO
A4	RC4		D1/TX1	RC6	
A3	RA5	GPIO	D2	RD0	GLCD, GPIO
A2	RA4		D3	RD1	
A1	RA1		D4	RD2	Character LCD, GPIO
A0	RA0		D5	RD3	
D10	RE2	SD card, GPIO	D6	RD4	
D11	RC5	SPI, GPIO	D7	RD5	
D12	RC4	I^2C/SPI, GPIO	D8	RD6	
D13	RC3		D9	RD7	

Note: The Arduino Nano can also be used to receive key press data from the co-processor via the UART bus, and it can also control the SD card, GLCD, and character LCD completely independently from the primary PIC.

3.12.3 Tips

- ➢ Ideally, code should be programmed into the Arduino while it is disconnected from the *DevBugger* (this is the most reliable option)
- ➢ Programming attempts with the Arduino socketed in the *DevBugger will* fail most times if the link switches with UART functionality are ON (PIC: **RC6**, **RC7**; Arduino: RX0/D0, TX1/D1). Ensure the UART pins on the Arduino Nano are isolated before programming. This must be done because the USB to serial converter (i.e. FTDI chip) on the bottom of the Arduino Nano PCB uses the TX and RX pins to communicate with the bootloader on the ATmega chip on the top of the Arduino Nano PCB, which is where user code executes.

- The *DevBugger* must be powered from the DC power supply to use the Arduino Nano properly
- Besides the USB module, another option for PC communication from the PIC would be to transfer data to/from the Arduino via SPI or I^2C, and then use the Arduino to transfer data to the PC. Arduinos have built-in FTDI chips—which are separate from the microcontroller on them (in fact it is on the bottom of the PCB while the Arduino's microcontroller is on the top)—which function as USB to serial converters. This functionality completely bypasses the microcontroller on the Arduino, so another option for PC communication involves loading program onto the Arduino Nano which does not use its UART, and then sending data from the PIC to the Arduino's TX pin, and receiving data from the Arduino's RX pin. In this case, the Arduino Nano is only on the board for its FTDI chip

3.13 SD Card

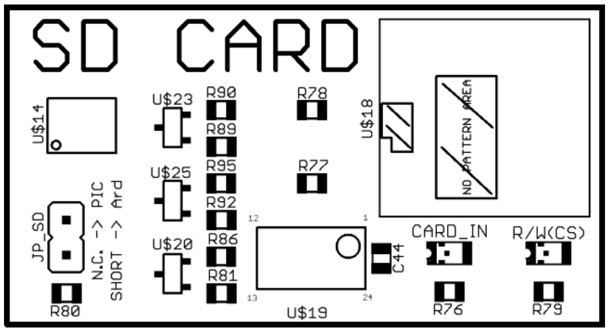


Figure 21. SD Card Module

3.13.1 Description

The board has been equipped with a microSD card socket, with a spring-loaded push-push mechanism, to provide access to a large amount of non-volatile storage that can be transferred easily between devices. To select whether the control signals will come from the PIC or Arduino Nano, the jumper **JP_SD** can be used as described in Table 9. This jumper is connected to a quad single-pole, double-throw (SPDT) analog switch IC that routes the connections between the SPI lines ($\overline{CS}$, SCK, SDA, and DAT0) and the SD card pins.

Table 9. microSD Card Jumper State

microSD Card Jumper State	**SD Card Controller**
Not connected (N.C.)	PIC18F4620
Shorted	Arduino Nano

There are four signals required to communicate with a SD card, as described in Table 10. Each of the control signals in Table 10 must be 3.3 V when it reaches the microSD card. Like the GLCD, these signals are routed through level shifter circuitry to ensure this is always the case.

Table 10. Pins Involved in microSD Card Interface

Signal name	**Pin on PIC**	**Pin on Arduino Nano**
Serial clock (SCK)	RC3	D13
Master Input Slave Output (MISO/DAT0)	RC4	D12
Master Output Slave Input (MOSI/SDA)	RC5	D11
Chip select ($\overline{CS}$), active low	RE2	D10

Two LEDs are included in the SD card module to help with debugging, one to indicate that the card is plugged in properly (CARD_IN), and the other to indicate that a transaction between the SD card and one of the microcontrollers is occurring (R/W(CS)).

3.13.2 HxD

SD cards used in consumer electronics often come formatted with a file allocation table (FAT) file system to provide a standard interface for the storage and retrieval of data. Modern operating systems support this interface, which is why SD cards can usually be browsed on a PC. If the FAT file system on a SD card is completely erased, however, then a modern operating system will not know how to display its contents. This use case is where *DevBugger* users may benefit from using the software HxD.

For users who choose to use a SD card for the storage of raw binary data (i.e. no file system), the following steps can be taken to view and edit the volume on a PC:

1. Install HxD
2. Open HxD with administrator privileges
3. Connect the microSD card to the PC. An adaptor may need to be used
4. On the top menu of HxD, click Extras → Open disk…
5. Select the “Physical disk” corresponding to the SD card, such as “Removable Disk 1”
6. Uncheck the “Open as Readonly” box if editing is desired
7. Click OK, and make note of the warning box that appears
8. The entire contents of the SD card can now be viewed and edited. Take note of the number of sectors available; the SD card sample code provided has been observed to overshoot the number of blocks by approximately 2%

3.13.3 Tips

- SD cards come in different “classes”, and higher classes generally support higher speeds. If you are finding that initialization succeeds, but reads and/or writes do not, you may wish to try decreasing the SPI clock frequency (i.e. increasing the SPI clock divider)
- The SD card will not be reset by pressing the reset button on the DevBugger. To properly reset it, it must be power cycled, or issued the software reset command sequence
- Arrays greater than one RAM bank in size (i.e. 256 bytes) must be global (i.e. defined outside a function) or declared within a function using the keyword “static”. For example, `static unsigned char myBuffer[512];`, is valid syntax for large array declaration inside a function
- One side of the **JP_SD** jumper is connected to VCC, and the other end is connected to the “IN” pin on the switch IC. When the “IN” pin voltage is at ground, the PIC is selected as the SD card controller; when it at VCC, the Arduino Nano is the controller. By default, the “IN” pin is pulled to ground by a 330 Ω resistor. Thus, the microSD card controller could be changed on-the-fly through software or circuitry if an I/O pin is connected to the side of the JP_SD jumper that has the pull-down resistor (this is not the side of the jumper *closest* to the resistor; use a multimeter in the continuity mode while the power is off to verify which side to connect to)

3.15 USB

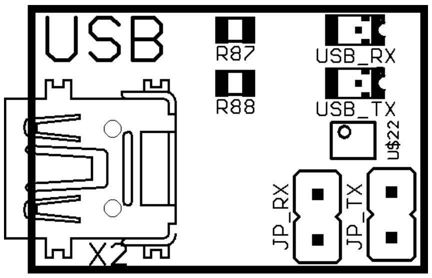


Figure 22. USB Module

3.15.1 Overview

The USB module facilitates the transfer of data between a PC and the microcontrollers on the *DevBugger*. It also serves the purpose of bridging the serial pins on the co-processor with the rest of the board and monitoring the signals on them.

The interface between the primary PIC and the co-processor is serial and uses the hardware UART module of the microcontrollers. Two jumpers, **JP_TX** and **JP_RX** select whether the primary PIC's UART lines are connected to those of the co-processor, as summarized in Table 11.

Table 11. UART Connections and Jumpers in the USB Module

	JP_RX State	**JP_TX State**
Shorted	RC6 (TX) of primary PIC connected to RC7 (RX) of Co-Processor	RC7 (RX) of primary PIC connected to RC6 (TX) of Co-Processor
Not shorted (open)	N.C.	N.C.

Debugging LEDs in the USB Module

Two LEDs are included in the module to aid with debugging. These LEDs do not monitor the state of the USB data lines, but rather, the state of the UART lines used to interface with the co-processor. As summarized in Table 12, the USB_TX LED monitors the state of the line routed through JP_TX (primary PIC: **RC6**; co-processor: **RC7**), and the USB_RX LED monitors the state of the line routed through JP_RX (primary PIC: **RC7**; co-processor: **RC6**). The LEDs monitor these lines through a dual inverting op amp, which draws very little current from the lines and thus does not interfere much with the signals. The op amp was selected to be inverting because in the UART idle state, the lines are pulled high by the internal module, and it is much easier to see bright pulses than dark pulses. To exemplify this point, **RC6** and **RC7** in the debug module monitor the actual (non-inverted) state of the UART lines, but it is nearly impossible to tell when a transmission/reception is occurring by looking at them.

Table 12. Conditions under which each LED in the USB module lights up

Transmitter	**Receiver**	**LED Indicator in USB module**
PC	PIC	USB_TX
PIC	PC	USB_RX

3.15.2 Use Cases

For a description of using the co-processor as a bidirectional USB to serial converter, please see the two use cases presented in §3.7.3 Program 2: Keypad Encoder (Port B) & Virtual COM Port.

3.15.3 PC-Side Software

First things first, the PC you communicate with needs to have the Microchip CDC driver installed. The installation file for Windows is included with the sample code package in the MCHPcdc_driver folder under Software Installers. The computers in the AER201 labs should also come with this driver.

Once the driver is installed, there are many options for transferring data between a PC and the co-processor, including python scripts (for example, pyserial) and serial monitors.

Serial monitors are PC programs that can send and receive data over serial ports (including virtual serial ports). Some look like command line terminals, and other ones have rich graphical user interfaces. For Windows users, the free software Docklight is recommended.

The following settings must be configured on the PC-side software for proper interfacing with the co-processor:

- Baud rate: 9600
- Number of data bits: 8
- Parity: No
- Stop bits: 1

Furthermore, always check that the correct COM port is being used (i.e. the COM port designated for the co-processor will change depending on how many other USB devices you have connected to your PC). Other settings should not matter.

3.15.4 Implications For UART Communication Between Microcontrollers

If you are not planning on using UART on the co-processor, the primary PIC, *AND* the Arduino Nano, you can skip this section.

As shown in Figure 23, at any given time, two microcontrollers on the *DevBugger* could be driving the RX line of the third microcontroller. This is generally not a safe practice because if two microcontrollers are driving the same line at opposite voltage levels (i.e. one at 0 V, and the other at 5 V), then they will be creating a short circuit and risk compromising their functional integrity.

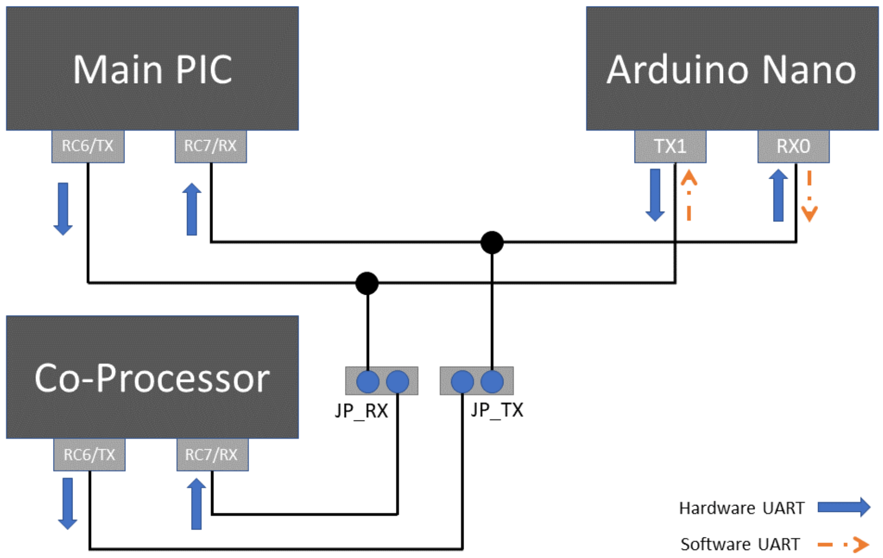


Figure 23. UART connections on the DevBugger. Hardware UART implies that the UART module on the microcontroller is used, whereas software UART means that a UART module is emulated in software

Proposed Solutions

There are a few techniques to avoid the scenario previously described:

1. Use a third signal line to indicate that the UART lines are busy. The hardware is simple, take 1 GPIO pin from each microcontroller, connect them all together, and set them to the tristate/input data direction. Tie this shared line to ground using a pull-down resistor with an experimentally verified value (likely a strong pull-down between 100 Ω and 2 $k\Omega$). Suppose a microcontroller wants access to the UART lines. In software, the protocol could be implemented as follows:
    a. Read the state of the signal line. Is it low?
        i. Yes, it is low → the UART lines are not in use. Continue to step b
        ii. No, it is high → the UART lines are in use. Do something else and try step a again after some time
    b. Since the lines are not in use, change the GPIO pin data direction to output and set the signal line high to indicate that the resource is being locked
    c. Broadcast the message desired. Should any of the other microcontrollers need to broadcast a message, they will read that the state of the signal line is high and will remain paused at step a until access to the resource is permitted
    d. Once the broadcast is complete, change the GPIO pin data direction to tristate/input to permit access to the resource (the pull-down resistor will enforce the logic 0 voltage level)
2. Use the Arduino Nano only ever as a receiver, and only ever in one of: software UART, hardware UART. With these two variables fixed, the primary PIC and Co-processor will still have full send/receive functionality, but there are two scenarios for communication with the Arduino Nano:

   a. Arduino implements software UART → Only the primary PIC can send/receive data to/from the Arduino Nano via the UART bus
   b. Arduino implements hardware UART → Only the Co-processor can send/receive data to/from the Arduino Nano via the UART bus

   Note that in case a, the Co-processor could still send data to the Arduino Nano if the primary PIC chooses to forward the data to it (i.e. act as a one-way bridge).

3. Remove the jumpers connecting the UART lines to the Co-processor. Then, the Arduino Nano and primary PIC can exchange data freely by hardware UART on the PIC's end and software UART on the Arduino Nano's end, but neither can communicate with the Co-processor via UART. The key here is that the Co-processor must not be driving lines on the UART bus if the primary PIC and Arduino Nano are communicating on it.
4. Use the Arduino-PIC link switches to disconnect the Arduino Nano from the UART bus. Then, the Co-processor and primary PIC can exchange data freely by hardware UART, but neither can communicate with the Arduino Nano via UART. I^{2}C, SPI, and GPIO can still be used for primary PIC and Arduino Nano communication, however.
5. Set the data direction registers for **RC6** and **RC7** on the primary PIC to tristate/input. Then, the Arduino Nano and co-processor can exchange data freely by hardware UART. Neither can communicate with the primary PIC via UART in this case. I^{2}C, SPI, and GPIO can still be used for primary PIC and Arduino Nano communication, and GPIO can still be used for communication between the primary PIC and co-processor.

4. Programming and Software

4.1 Overview

The onboard programmer is compatible with all MPLAB X and MPLAB programming tools as a PICkit 3 In-Circuit Debugger/Programmer. Quick start and full user guides for MPLAB X IDE and MPLAB X IPE can be found from the official Microchip website. It is suggested that the documents at the following links be saved as reference material to guide development.

- **MPLAB X IDE User's Guide:**
  http://ww1.microchip.com/downloads/en/DeviceDoc/50002027D.pdf

- **PICkit 3 In-Circuit Debugger/Programmer User's Guide For MPLAB X IDE:**
  http://ww1.microchip.com/downloads/en/DeviceDoc/52116A.pdf

- **Integrated Programming Environment (IPE) User's Guide:**
  http://ww1.microchip.com/downloads/en/DeviceDoc/50002227C.pdf

Other software suggested, to help with using the onboard peripheral modules, is as follows:

- **Docklight**, a free serial monitor that can send and receive data via a virtual COM port (url: https://docklight.de/downloads/). This may be useful for users wanting to use the USB to serial converter program on the co-processor

- **HxD**, a free hex editor that can read and edit raw data on a SD card (url: http://download.cnet.com/HxD-Hex-Editor/3000-2352_4-10891068.html). This may be useful for users who want to use a SD card for raw storage but still be able to access its contents on a PC

4.2 Using the Onboard Programmer

The two most common ways to use the programmer from within MPLAB software will be covered in this section.

4.2.1 MPLAB X IDE

MPLAB X IDE is the program best suited for development and will most likely be the only programming tool used. It contains a project explorer used to easily navigate through source files, and buttons for building, programming, and debugging. To get code onto the primary PIC, the following steps can be taken:

1. In MPLAB X IDE, ensure that in Project Properties -> Configuration the correct PIC device is selected, along with PICkit3 as the hardware tool and XC8 as the compiler
2. Click the hammer tool to build the code. It may be desirable to define keyboard shortcuts for building the code and programming the device. To do this, go to Tools -> Options -> Keymap. Search for “Program” and define keyboard shortcuts for “Make and Program Device Main Project”
3. Connect the *Development Board* to the PC
4. Flip the programming switch to PRG mode and flip the power switch to ON
5. In MPLAB X IDE, right-click the project and select the “Run” option. The green triangle next to the hammers can also be pressed, but this button will always run the project that’s selected as the *main project* (project name in bold). This, if this option is to be used, the user should always make sure that they have explicitly set the main project by right-clicking on the project and selecting the “Set as Main Project” option
6. If the project is to be debugged, the user can right-click the project and select the “Debug” option instead of “Run”
7. The IDE will sometimes caution the user to check if the device is truly a 5 V device. This is normal and is just done because the PICkit 3 may cause damage to a low voltage device if the programmer uses 5 V on its I/O pins.

Note: If the MPLAB X IDE is having difficulties with programming devices or debugging devices, simply save your work and restart the IDE. This has been observed to be successful where many other solution attempts fail.

4.2.2 MPLAB X IPE

MPLAB X IPE is suited for operations involving code that’s already compiled and linked into a HEX file. The following steps can be used to load HEX code onto a PIC using the IPE:

1. Connect the *Development Board* to the PC
2. Flip the programming switch to PRG mode and flip the power switch to ON
3. In the IPE, select the correct Family and Device option from the dropdown menu. For example, If the PIC18F4620 device is used, select “Advanced 8-bit MCUs (PIC 18), PIC18F4620”
4. The *DevBugger* programmer should be detected and will be displayed as a PICkit 3 programmer in the tool section

5. Click “Connect” to connect to the programmer
6. Wait for any firmware update (if applicable) to download and install.
7. Go to File → Import → Hex to browse for a HEX files and load it
8. Press the “Program” button to load the HEX file to the PIC device. The IPE will indicate if programming is successful or if any errors occurred
9. The IPE will sometimes caution the user to check if the device is truly a 5 V device. This is normal and is just done because the PICkit 3 may cause damage to a low voltage device if the programmer uses 5 V on its I/O pins.

Notes:

- If the Development board is disconnected at any point, it must be reconnected to the application by pressing the “Connect” button before programming
- If any errors occur or the application cannot find the programmer or the PIC device, make sure the programming switch is in PRG mode, turn off the power to reset the board, and try again
- The following is a description of the common operations that may be performed within the MPLABX IPE:
    - **Program** downloads the currently loaded HEX code into the target device
    - **Erase** performs a bulk erase of the device, effectively returning it to its factory state. In some cases, this may fix a device that appears faulty
    - **Read** is used to read the current program loaded into the target PIC device. This program can be retrieved and stored in a HEX file. It also retrieves the EEPROM data stored in the PIC. Note that the option to save the HEX file will not be available until it is enabled from advanced mode. For more detail on usage of the advanced mode, see MPLABX IPE User Manual, Section 2.5
    - **Verify** reads the contents of the PIC device and compares it against the loaded HEX file. If any differences are observed, an error is given, and the operation fails
    - **Blank Check** is used to verify that the chip is indeed “blank”; this is useful after performing an *Erase Chip* operation to verify that the operation succeeded

4.3 SAMPLE CODE

The development board package includes sample code that demonstrates functionality of the onboard modules via the primary PIC and the Arduino Nano. The main file for each sample program has comments at the top that describe what it does, and conditions that must be true prior to executing the code for it to work properly (called *preconditions*). The preconditions are not exhaustive, but they cover the main configuration elements.

4.3.1 PIC18F4620 SAMPLE CODE

Some PIC18F4620 examples in assembly are included, but because Microchip has been dropping support for assembly language programming, the newest sample code is written in C for the MPLAB XC8 compiler to ensure compatibility for the future. For an official guide on the MPLAB XC8 compiler, please refer to the links below.

- **MPLAB XC8 Getting Started Guide:**
  http://ww1.microchip.com/downloads/en/DeviceDoc/50002173A.pdf

- **MPLAB XC8 C Compiler User's Guide:**
  http://ww1.microchip.com/downloads/en/DeviceDoc/50002053G.pdf

- **MPLAB XC Compilers Download Page:**
  http://www.microchip.com/mplab/compilers

Table 13 summarizes the sample programs available for the PIC18F4620. Additionally, the co-processor firmware is made available as sample code for the PIC18F2550.

Table 13. PIC18F4620 Sample programs

Program Name	**Description**
PortTest (C & asm)	Iterates through the pins on the PIC, making the debug LEDs toggle sequentially. Note that RB6 and RB7 will not light up unless the PRG/RUN switch is in the PRG position, and RB1 will not light up unless the keypad is disabled
CharacterLCD_1	Prints "Hello world! :)" to the character LCD
CharacterLCD_2	Demonstrates more character LCD capabilities such as display shifting and moving the cursor to a specific DDRAM address
KeypadLCD_IO_Polling (C and asm)	Receives keypress data from co-processor and displays the corresponding characters on the character LCD
KeypadLCD_IO_Interrupt	Demonstrates of interrupt on change feature of RB1. The main loop changes the characters displayed on the top line of the LCD, while the interrupt handler displays key press data on the bottom line
KeypadLCD_UART	Receives keypress data from co-processor and displays the corresponding characters on the character LCD

ADC **(C & asm)**	**C version**: displays ADC readings, which can be varied using the potentiometers in the ADC module, on the character LCD **asm version**: displays RA3 readings on PORTC and RA2 readings on PORTD
RTC	Sets the real-time clock (RTC) and reads back the time once per second
PC_Demo	Sends key press data to a PC connected to the co-processor, while also displaying data received from the PC on the character LCD
GLCD	Draws rectangles, pixels, and patterns on the GLCD
SD_Init	Initializes the SD card. Information about the card is collected, during initialization such as its capacity card type, SD spec. version the card complies to, date of manufacture, etc. These fields are displayed on the character LCD
SD_IO	Demonstrates basic SD card I/O capabilities, specifically, reading/writing single/multiple blocks and erasing
PWM **(C & asm)**	**C version**: modulates the brightness of RC2 with a sine wave **asm version**: steadily ramps up the brightness of RC2
USART (asm only)	Demonstrates basic functionality of the EUSART module by incrementing a counter and transmitting its value via UART in an infinite loop. The current value of the counter is also displayed on LATD
PIC_I2C_Arduino	Demonstrates communication between the primary PIC and the Arduino Nano, via I2C. To see the results on the Arduino Nano side, open the Arduino sample program Arduino_I2C_PIC. If the PIC receives a triple-A sequence from the keypad, it changes from being a transmitter to being a receiver that displays data on the character LCD. To change it back, reset the PIC
PIC_UART_Arduino	Demonstrates arbitrary information exchange between the main PIC and the Arduino Nano. To see the results on the Arduino Nano side, use the Arduino sample program Arduino_UART_PIC
BoardTest	Tests the basic capabilities of most of the development board modules. Can be used to check the board is working properly

4.3.2 Arduino Nano Sample Code

Arduino Nano code development should occur in the Arduino Nano IDE, available at the following link:

- **Arduino IDE Download Link**: https://www.arduino.cc/en/Main/Software

To program the Arduino Nano, do the following:

1. Use a USB mini-B cable to connect the Arduino Nano to a PC. The drivers for the Arduino Nano should be located and installed automatically. The Arduino Nano will then appear as a virtual COM port to the PC
2. Open the desired program in the Arduino IDE
3. In the Tools menu, go to Port, and ensure the COM port for the Arduino Nano is selected
4. In the top-left of the Arduino IDE, the menu in Figure 24 should be visible

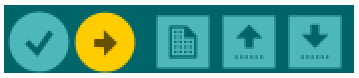


Figure 24. Arduino IDE menu with the program button in yellow

5. Click the button with the right-facing arrow, indicated in yellow in Figure 24
6. The console will display messages, and if there are no issues with building the code, the Arduino Nano will start programming. The programming button clicked in step 4 will be yellow throughout the duration of programming
7. Programming is complete when the TX and RX LEDs on top of the Arduino Nano stop flashing, and the programming button returns to blue

Table 14 summarizes the sample programs available for the Arduino Nano.

Table 14. Arduino Nano Sample programs

Program Name	**Description**
PortTest	Iterates through the pins on the Arduino Nano. If all the PIC-Arduino link switches are enabled, and the PIC18F4620 has all its pins tristated (or it is simply removed from its socket), then the debug LEDs will light up according to the Arduino Nano pins
Arduino_UART_PIC	Demonstration of Arduino-PIC communication via UART. This program is paired with the PIC18F4620 sample code "PIC_UART_Arduino". In this program, the Arduino prints arbitrary data received from the PIC to the character LCD (which must be in an Arduino socket)
Arduino_I2C_PIC	Demonstration of Arduino-PIC communication via I^{2}C. This program is paired with the PIC18F4620 sample code "PIC_I2C_Arduino". In this program, the Arduino receives keypad data from the PIC and forwards it to a PC. If 'AAA' is given via the keypad, then the PIC becomes a receiver and the Arduino Nano can send data to the PIC to be displayed on the character LCD

5. Advanced Operations

5.1 Re-imaging Firmware for the Onboard Programmer

The PICkit 3 programmer on the development board is fully programmed and ready to use out of the box. However, it has been observed in previous iterations of this development board that some users experienced firmware faults on the programmer that required the programmer firmware to be re-imaged.

To re-image the programmer, plug in an external PICkit 3 programmer to the ICSP_PK3 header on the very top of the programmer module. The pins of the ICSP header are arranged such that pin 6 is on the very left edge, and pin 1 is on the right most corner (see Table 1). When a PICkit 3 is used, the white tringle making on the PICkit 3 body indicates pin 1 of the ICSP interface. When inserted, the indicator LEDs should face toward the rest of the board. Alternatively, the ICSP_EXT header on another *DevBugger* can be used as the external programmer.

The firmware file, “DB4_PK3_24FJ256GB106_Firmware.hex”, is provided in the sample code folder in the “13_Microchip Official Firmware” folder.

To reimage the programmer, follow the below procedures:

1. Open MPLAB X Independent Programming Environment (IPE)
2. Connect the external PICKit 3 Programmer to the computer
3. Select 16-bit MCUs (PIC24) in the Family dropdown menu
4. Select PIC24FJ256GB106 in the device dropdown menu
5. The IPE should have detected the PICKit3 programmer, click connect to connect to the programmer.
6. Wait for the IPE to download and update firmware for the PICKit 3. Reconnect to the programmer if the procedure interrupts itself
7. Click File -> Import -> Hex. Load “DB4_PK3_24FJ256GB106_Firmware.hex”
8. Ensure the board being reprogrammed is powered on, and the program/run switch is in the “RUN” position.
9. Click “Program” on the main interface and wait for programming to finish.
10. The programmer firmware should now be restored to the default for the *DevBugger*. To verify the programmer firmware has been correctly loaded, detach the external programmer and cycle power on the board just programmed (i.e. turn off, then on). The 3 indicator LEDs in the programmer module should light up when the board is first powered, and after approximately 5 seconds, the 2 status LED will extinguish, leaving only the red active LED illuminated. If this behavior is observed, the programmer is properly loaded and functioning correctly.

Note: The firmware may be updated automatically when connecting the *DevBugger* programmer to MPLAB software, which is as desired.

Appendix A: List of all Removable Parts

Table 15 lists the removable parts from the *DevBugger* and their respective manufacturer (MFG) and distributor part numbers. The distributor is Digi-Key unless otherwise noted.

Table 15. List of removable parts on the DevBugger

Description	QTY	MFG P/N	Distributor P/N
RTC chip	1	DS1307	DS1307+-ND
Short jumpers (**JP6**, **JP7**, **JP_SD**)	3	QPC02SXGN-RC	S9337-ND
Tall jumpers (**JP8**, **JP9**, **JP_TX**, **JP_RX**)	4	NPC02SXON-RC	S9341-ND
Rectifier 2W04G; 4-pin, cylindrical	1	2W04G-E4/51	2W04G-E4/51GI-ND
3.15 A Schurter MSF 250 Fuse, 8.5 mm diameter, fast blow	1	0034.6019	693-0034.6019 (Mouser)
10 MHz crystal oscillator, 20 pF (primary PIC)	1	XT9S20ANA10M	73-XT49S1000-20 (Mouser)
20 MHz crystal oscillator, 30 pF (co-processor)	1	FOXSLF/200-20	559-FOXS200-20-LF (Mouser)
PIC18F4620, DIP-40 package	1	PIC18F4620	PIC18F4620-I/P-ND
PIC18F2550 DIP-28 package	1	PIC18F2550	PIC18F2550-I/SP-ND
Keypad switch ICs, DIP-14	2	SN74HC4066N	296-8329-5-ND
Debug module buffers, DIP-20	4	SN74HC244N	296-1582-5-ND
4x4 keypad	1	-	GH5004-ND
4x16 Character LCD	1	-	NHD-0216BZ-RN-YBW-ND
128x128 GLCD	1	-	LCDMG-144128 (Creatron)
RTC coin battery	1	ECR2032	B0042A9UXC (Amazon)
2x20 bus cable	1	CABLE-40-40-10CM	909-CABLE-40-40-10CM

Note:

- The part numbers included are for reference only, and the parts installed on the board might not be the exact part listed by the distributor
- Creatron claims the graphical LCD they sell has an ILI9163C controller. The ILI9163C and ST7735R are very similar controllers, and it's very likely that most code will be compatible between the two. If the code differs anywhere, it would most likely be initialization
- 20 MHz crystals that have 20 pF load capacitance have been tested and work properly

APPENDIX B: SUMMARY OF JUMPER CONFIGURATIONS

Table 16. 2-pin Jumpers

Jumper Name	Module	Effect when shorted	Effect when open
JP4	A2D	RA2 enabled as V_{ref-}	RA2 available as GPIO
JP5	A2D	RA3 enabled as V_{ref+}	RA3 available as GPIO
JP_SD	SD Card	Arduino controls SD card	PIC controls SD card
JP6	RTC	RC3 connected to RTC module	RC3 not connected to RTC module
JP7	RTC	RC4 connected to RTC module	RC4 not connected to RTC module
JP_RX	USB	Co-processor's RC7 pin is connected to the primary PIC's RC6	Co-processor's RC7 line is isolated
JP_TX	USB	Co-processor's RC6 pin is connected to the primary PIC's RC7	Co-processor's RC6 line is isolated

Table 17. 3-pin Jumpers

Jumper Name	Module	Effect when 1+2 shorted	Effect when 2+3 shorted
JP8	PIC	1 out of 2 required connections to primary oscillator is made	RA7 connected to I/O bus
JP9	PIC	1 out of 2 required connections to primary oscillator is made	RA6 connected to I/O bus