INTRODUCTION

The PIC32MX Microcontroller Unit (MCU) is a complex system-on-a-chip that is based on a M4K™ core from MIPS® Technologies. M4K™ is a state- of-the-art 32-bit, low-power, RISC processor core with the enhanced MIPS32® Release 2 Instruction Set Architecture. This section provides an overview of the CPU features and system architecture of the PIC32MX family of microcontrollers.
Key Features:

- Up to 1.5 DMIPS/MHz of performance
- Programmable prefetch cache memory to enhance execution from Flash memory
- 16-bit Instruction mode (MIPS16e) for compact code
- Vectored interrupt controller with 63 priority levels
- Programmable User and Kernel modes of operation
- Atomic bit manipulations on peripheral registers (Single cycle)
- Multiply-Divide unit with a maximum issue rate of one 32 × 16 multiply per clock
- High speed Microchip ICD port with hardware-based non-intrusive data monitoring and application data streaming functions
- EJTAG debug port allows extensive third party debug, programming and test tools support
- Instruction controlled power management modes
- Five stage piplined instruction execution
- Internal Code protection to help protect intellectual property

1. PIC32MX Architecture Overview

The PIC32MX family processors are complex systems-on-a-chip that contain many features. Included in all processors of the PIC32MX family is a high-performance RISC CPU, which can be programmed in 32-bit and 16-bit modes, and even mixed modes. The PIC32MX MCU contains a high-performance interrupt controller, DMA controller, USB controller, in-circuit debugger, high performance switching matrix for high-speed data accesses to the peripherals, on-chip data RAM memory that holds data and programs. The unique prefetch cache and prefetch buffer for the Flash memory, which hides the latency of the flash, gives zero Wait state equivalent performance.

There are two internal buses in the chip to connect all the peripherals. The main peripheral bus connects most of the peripheral units to the bus matrix through a peripheral bridge. There is also a high-speed peripheral bridge that connects the interrupt controller DMA controller, in-circuit debugger, and USB peripherals. The heart of the PIC32MX MCU is the M4K CPU core. The CPU performs operations under program control. Instructions are fetched by the CPU, decoded and executed synchronously. Instructions exist in either the Program Flash memory or Data RAM memory. Figure I show the PIC32MX Block Diagram.

The PIC32MX CPU is based on a load/store architecture and performs most operations on a set of internal registers. Specific load and store instructions are used to move data between these internal registers and the outside world.

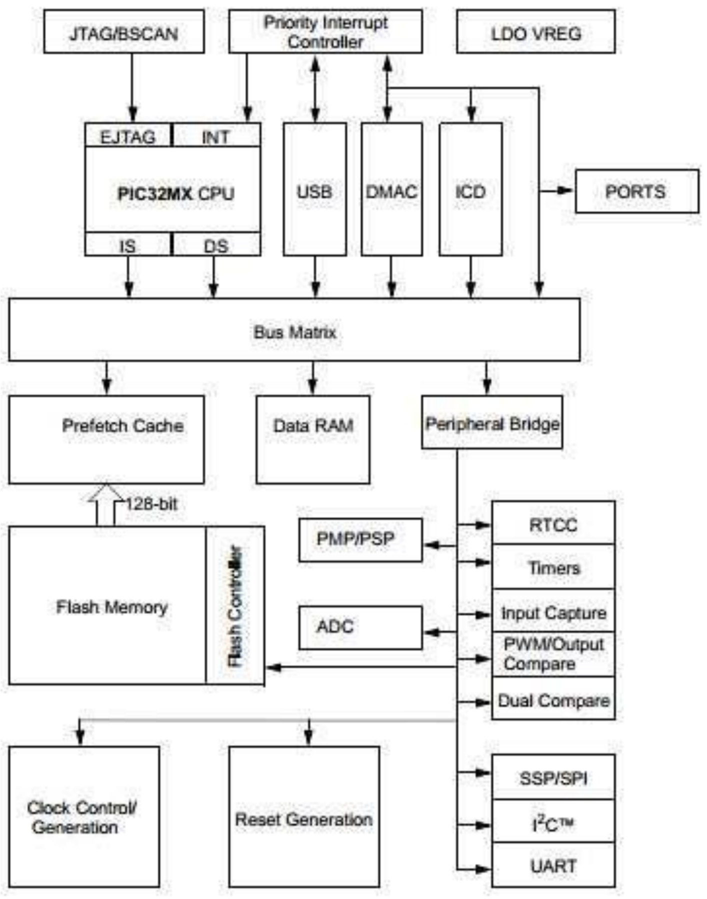


Figure I - PIC32MX MCU Block Diagram

In this lab, we will use the PIC32MX460F512L (Cerebot MX4cK).

2. Cerebot MX4cK Overview

The Cerebot MX4cK is a microcontroller development board based on the Microchip PIC32MX460F512L, a member of the 32-bit PIC32 microcontroller family. It is compatible with Digilent's line of Pmod™ peripheral modules, and is suitable for use with the Microchip MPLAB® IDE tools. The Cerebot MX4cK is also compati ble for use with the ChipKIT™ MPIDE development environment. ChipKIT and MPIDE is a PIC32 based system compatible with many existing Arduino™ code examples, reference material s and other resources.
The Cerebot MX4cK is designed to be easy to use and suitable for use by anyone from beginners to advanced users for experimenting with electronics and embedded control systems. A built in programming/debugging circuit compatible with the Microchip MPLAB® IDE is provided on the

board, so no additional hardware is required for use with MPLAB. The kit contains everything needed to start developing embedded applications using either the MPLAB® IDE or the MPIDE. The Cerebot MX4cK provides 74 I/O pins that support a number of peripheral functions, such as USB controller, UART, SPI and I2C™ ports as well as five pulse width modulated outputs and five external interrupt inputs. Fifteen of the I/O pins can be used as analog inputs in addition to their use as digital inputs and outputs.
The Cerebot MX4cK can be powered via USB, or an external power supply that may be either an AC-DC power adapter, or batteries.

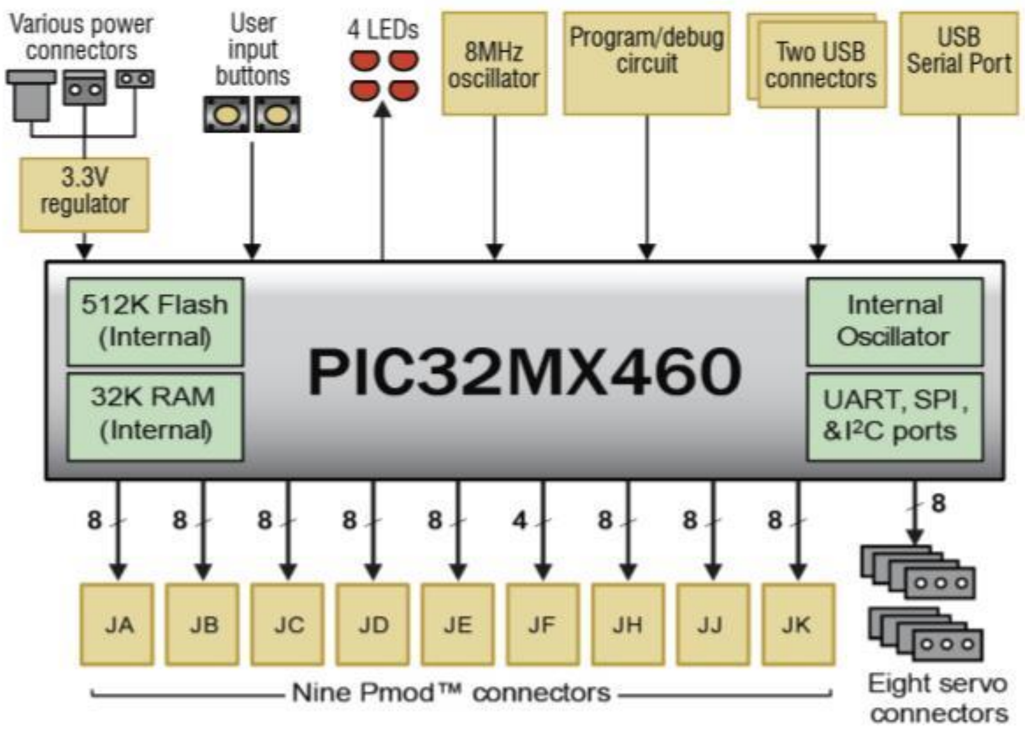


Figure II - Cerebot MX7cK Circuit Diagram

3. Functional Description

The Cerebot MX4cK is designed for embedded control and robotics control applications as well as for general microprocessor experimentation. Firmware suitable for many applications can be downloaded to the Cerebot MX4cK's programmable PIC32 microcontroller.
Features of the Cerebot MX4cK include:

- a PIC32MX460F512L microcontroller
- support for programming and debugging within the Microchip MPLAB development environment
- nine Pmod connectors for Digilent peripheral module boards • eight hobby RC servo connectors
- USB 2.0 Device, Host, and OTG support
- two push buttons

- four LEDs
- multiple power supply options, including USB powered
- ESD protection and short circuit protection for all I/O pins.

Features of the PIC32MX460F512L include:

- 512KB internal program flash memory
- 32KB internal SRAM memory
- USB 2.0 compliant full-speed On-TheGo (OTG) controller with dedicated DMA channel
- two serial peripheral interfaces (SPI)
- two UART serial interfaces
- two I2C serial interfaces
- five 16-bit timer/counters
- five timer capture inputs
- five compare/PWM outputs
- sixteen 10-bit analog inputs
- two analog comparators

The Cerebot MX4cK has a number of input/output connection options, and is specially designed to work with the Digilent Pmod™ line of peripheral modules to provide a variety of input and output functions. For more information, see www.digilentinc.com. In addition to the Pmod connectors, the board provides eight connectors for RC hobby servos, two push button switches, and four LEDs, as well as providing connections for two I^{2}C busses. A serial EEPROM and a 12-bit digital to analog converter are provided on one of the I^{2}C busses.
The Cerebot MX4cK features a flexible power supply system with a number of options for powering the board as well as powering peripheral devices connected to the board. It can be USB powered via either the debug USB port or the USB device port, or it can be powered from an external power supply or batteries.

4. Using the Cerebot MX4cK with the ChipKIT MPIDE

ChipKIT and the MPIDE is a PIC32 based hardware and software system compatible with many existing Arduino™ code examples, reference material s and other resources. The MPIDE development platform was produced by modifying the Arduino™ IDE and is fully backward compatible with the Arduino IDE. The Cerebot MX4cK board is designed to be fully compatible with the ChipKIT MPIDE system, version 20111209 or later.
The MPIDE uses a serial communications port to communicate with a boot loader running in the target board. The serial port on the MX4cK board is implemented using an FTDI FT232R USB serial converter. Before attempting to use the MPIDE to communicate with the MX4cK, the appropriate USB device driver must be installed.
The USB serial converter on the Cerebot MX4cK board uses USB connector J8, labeled UART on the board. This connector is a microUSB. Use a standard USB-A to mini-B cable (provided with the board) to connect the board to an available USB port on the PC.

In the MPIDE, use the "Tools.Board" command to sele ct the Cerebot MX4cK from the list of available boards. Use the "Tools.Serial Port" comma nd to choose the appropriate serial port from the list of available serial ports on the PC.
When the MPIDE needs to communicate with the MX4cK board, the PIC32 microcontroller is reset and starts running the boot loader. The MPIDE then establishes communications with the boot loader and downloads the program to the board.
When the MPIDE opens the serial communications connection on the PC, the DTR pin on the FT232R chip is driven low. This pin is coupled through a capacitor to the MCLR pin on the PIC32 microcontroller. Driving the MCLR line low resets the microcontroller, restarting execution with the boot loader.
Once the MPIDE has established communication with the boot loader, it transfers the user's program to the boot loader, which programs it into the flash memory in the Microcontroller.
The automatic reset action when the serial communications connection is opened can be disabled. To disable this operation, remove the shorting block from jumper JP8. The shorting block is reinstalled on JP8 to restore operation with the MPIDE.
Two red LEDs (LD7 and LD8) will blink when data is being sent or received between the Cerebot MX4cK and the PC over the serial connection.
The header connector J7 provides access to the other serial handshaking signals provided by the FT232R. Connector J7 is not loaded at the factory but can be installed by the user to access these signals.

5. Power Supply

Switch SW1, on the left side of the board is the power switch. Place this switch in the ON position to turn on board power and in the OFF position to turn off board power.
The ChipKIT Pro MX4 may be USB powered via either the USB debug port, the USB UART port, or the USB device port. Alternatively, the board may be powered via dedicated, "external", power supply connectors.
Jumper block J12 is used to select the power source used to provide power to the board. This jumper block provides the following four positions:

- USB – power is supplied by USB device connector J15 . This is used when the ChipKIT Pro MX4 is used to implement a USB bus powered device.
- EXT – Power is supplied by one of the external powe r connectors.
- DBG – Power is supplied by DEBUG USB connector J9.
- URT – Power is supplied by UART USB connector J8.

Place the shorting block in the appropriate position on J12 for the desired power source for the board.
The ChipKIT Pro MX4 is rated for external power from 3.6 to 12 volts DC. Using a voltage outside this range could damage the board and connected devices. If operating the board at a voltage greater than 5V, it is necessary to remove the shorting block on jumper JP10 to protect the USB load switch, which is limited to a maximum voltage of 5.5V. The USB load switch is used to control bus power when the ChipKIT Pro MX4 is being used to implement a USB host. When operating from any of the three USB sources, the input voltage will be 5V.

The output of power select jumper block J12 is connected to the VIN power bus. The VIN power bus supplies power to Q3, a PFET load switch, and IC9, the voltage regulator for the licensed debugger circuit. The licensed debugger circuit is powered as soon as the power switch is turned on. Power to the rest of the board is controlled by Q3. The main board power supply is enabled by bringing the gate of Q3 low. When Q3 is turned on, the unregulated power bus BRD_VU is powered.
If the licensed debugger is connected to an active USB port, it enumerates with the host computer and once it has been successfully enumerated, it turns on the main board power supply by driving the PWR_ON signal high.
If the licensed debugger is not connected to an active USB port, the PWR_ON signal is ignored and board power is turned on immediately by the power switch via transistor Q4.
The main board power supply is a switch mode voltage regulator implemented using a Microchip MCP16301 switch mode step-down regulator. This regulator provides 3.3V at up to 600 mA with approximately 96% efficiency. This powers the main board regulated power bus BRD_3V3 There are three connectors on ChipKIT Pro MX4 for connecting an external power supply: J13, J14, and J18.
The barrel connector, J13, is useful for desktop development and testing where using USB or battery power is not suitable. J13 is the connector used by the AC adapter optionally available from Digilent, or other sources. J13 is a 2.5mm x 5.5mm coaxial connector wired with the center terminal as the positive voltage.
Connector J14 is a two-pin male header that provides easy battery or battery-pack connection. Digilent has both two-cell and four-cell AA battery holders with two pin connectors available for connection to J14.
Connector J18 is a screw terminal connector for an alternative power supply connection for use with higher current battery packs, bench supplies or other power sources where use of a hard wired power supply is desirable.
Connectors J13, J14, and J18 are wired in parallel and connect to the “External Power” position on the Power Select jumper block J12. A shorting block should be placed on the “EXT” position of J12 when using this option for board power. Only one of these three power connectors should be used at a time. If multiple power supplies are connected simultaneously, damage to the board or the power supplies may occur.
When using an external power supply, ensure that the polarity is not reversed as the board is not protected from reverse polarity on the power supply and will be damaged.
The ChipKIT Pro MX4 has a second screw terminal connector, J5 that supplies power to the servo power bus, VS, to power the RC hobby servo connectors. This allows servos to be powered from a separate power supply than the one powering the electronics on the ChipKIT Pro MX4. This can be useful when using servos that draw large amounts of power.
Jumper JP1 can be used to connect the ChipKIT Pro MX4 unregulated power bus BRD_VU to the servo power bus, VS. When no shorting block is installed on JP1, the BRD_VU and VS busses are separate. When a shorting block is on JP1, the two busses are joined and the BRD_VU bus can be powered in any of the previously indicated ways, or from connector J5.
The ChipKIT Pro MX4 can provide power to any peripheral modules attached to the Pmod connectors and to I2C devices powered from the I2C daisy chain connectors, J2 and J6. Each

Pmod connector provides power pins that can be powered by either unregulated voltage, BRD_VU, or regulated voltage, BRD_3V3, by setting the voltage jumper block to the desired position. The I2C power connectors only provide regulated voltage, BRD_3V3.
The PIC32 microcontroller and on-board I/O devices operate at a supply voltage of 3.3V provided by the BRD_3V3 bus. The PIC32 microcontroller will use approximately 55mA when running at 80MHz. The other circuitry on the board will use approximately 10mA. The remaining current is available to provide power to attached Pmod and I2C devices.

6. Pmod Connectors

The ChipKIT Pro MX4 has nine connectors for connecting Digilent Pmod peripheral modules. The Pmod connectors, labeled JA–JF and JH–JK, are 2 x8 pin right-angle, female pin header connectors. Each connector has an associated power select jumper block labeled JPA–JPF and JPH–JPK.
Digilent Pmods are a line of small peripheral modules that provide various kinds of I/O interfaces. The Pmod product line includes such things as button, switch and LED modules, connector modules, LCD displays, high current output drivers, various kinds of RF interfaces, and many others.
There are two styles of Pmod connector: six-pin and twelve-pin. Both connectors use standard pin headers with 100mil spaced pins. The six-pin connectors have the pins in a 1x6 configuration, while the twelve-pin connectors use a 2x6 configuration. All of the Pmod connectors on the ChipKIT Pro MX4 are twelve pin connectors.
Six-pin Pmod connectors provide four I/O signals, ground and a switchable power connection. The twelve-pin connectors provide eight I/O signals, two power and two ground pins. The twelve-pin connectors have the signals arranged so that one twelve-pin connector is equivalent to two of the six-pin connectors. Pins 1–4 and 7–10 are the signal pins, pins 5 and 11 are the ground pins and pins 6 & 12 are the power supply pins.
The pin numbering that Digilent uses on the twelve-pin Pmod connectors is non-standard. The upper row of pins are numbered 1–6, left to right (when viewed from the top of the board), and the lower row of pins are numbered 7–12, left to right. This is in keeping with the convention that the upper and lower rows of pins can be considered to be two six-pin connectors stacked. When viewed from the end of the connector, pin 1 is the upper right pin and pin 7 is immediately below it (closer to the PCB).
Each Pmod connector has an associated power select jumper. These are used to select the power supply voltage supplied to the power supply pin on the Pmod connector. They are switchable between either the unregulated power supply, BRD_VU, or the 3.3V main board supply, BRD_3V3. Place the shorting block in the 3V3 position for regulated 3.3V and in the VU position to use the unregulated supply.
Each signal pin on the Pmod connectors is connected to an input/output pin on the PIC32 microcontroller. Each pin has a 200 ohm series resistor and an ESD protection diode. The series resistor provides short circuit protection to prevent damaging the I/O block in the microcontroller if the pin is inadvertently shorted to VDD or GND, or two outputs are shorted together. The ESD protection diode protects the I/O block from damage due to electro-static discharge.

Although ESD protection is provided between the connector pins and the microcontroller pins, ESD safe handling procedures should be followed when handling the circuit board. The pins on the microcontroller and other circuits on the board are exposed and can be damaged through ESD when handling the board.
Digilent Pmod peripheral modules can either be plugged directly into the connectors on the ChipKIT Pro MX4 or attached via cables. Digilent has a variety of Pmod interconnect cables available.
See the Pinout Tables in Appendix C, for more information about connecting peripheral modules and other devices to the ChipKIT Pro MX4. These tables describe the mapping between pins on the PIC32MX460 microcontroller and the pins on the various connectors.
The PIC32 microcontroller can source or sink a maximum of 18mA on all digital I/O pins. However, to keep the output voltage within the specified input/output voltage range (V_{OL} 0.4V, V_{OH} 2.4V) the pin current must be restricted to +7/-12mA. The maximum current that can be sourced or sunk across all I/O pins simultaneously is +/200mA. The maximum voltage that can be applied to any digital only I/O pin is 5.5V. The maximum voltage that can be applied to any analog capable I/O pin is 3.6V.
For more detailed specifications, refer to the PIC32MX3XX/4XX Family Data Sheet.

7. Push Buttons and LEDs

The ChipKIT Pro MX4 board provides two push button switches for user input and four LEDs for output. The buttons, BTN1 and BTN2 are connected to I/O pins TRCLK/RA6 and TRD3/RA7 respectively. To read the buttons, pins 6 and 7 of I/O Port A must be configured as inputs by setting the corresponding bits in the TRISA register. The button state is then obtained by reading the PORTA register. When a button is pressed, the corresponding bit will be high (‘1’). Note that the microcontroller pins used by the buttons are shared with pins 3 & 4 of Pmod connector JF. Note that the button circuitry will effectively act as a 10K ohm pull-up resistor when the button is pressed and a 20K ohm pull-down resistor when the button is not pressed. This will not interfere with most normal uses of the I/O pins if the buttons are not being used.
The four LEDs are connected to bits 10-13 of I/O Port B. LED 1 is connected to bit 10, LED 2 is connected to bit 11, and so on. These four bits are also shared with pins 1-4 of Pmod connector JK. To use the LEDs, set the desired bits as outputs by clearing the corresponding bits in the TRISB register. The state of an LED is set by writing values to the LATB register. Setting a bit to 1 will illuminate the LED and setting the bit to 0 will turn it off.
When using the MPIDE and the ChipKIT system, the buttons are accessed using digitalRead() and the LEDs using digitalWrite(). Use the following pins to access them:

- BTN1 – PIN_BTN1, pin 42, RA6
- BTN2 – PIN_BTN2, pin 43, RA7
- LD1 – PIN_LED1, pin 64, RB10
- LD2 – PIN_LED2, pin 65, RB11
- LD3 – PIN_LED3, pin 66, RB12
- LD4 – PIN_LED4, pin 67, RB13

8. UART Interface

The PIC32MX460 microcontroller provides two UART interfaces, UART1 and UART2. These UARTs can provide either a 2-wire or a 4-wire asynchronous serial interface. The 2-wire interface provides receive (RX) and transmit (TX) signals. The 4-wire interface includes request-to-send (RTS) and clear-to-send (CTS) in addition to receive and transmit.
UART1 can be accessed from Pmod connector JE and UART2 can be accessed from Pmod connector JH using the following pins:

- U1CTS JE-01
- U1TX JE-02
- U1RX JE-03
- U1RTS JE-04

- U2CTS JH-01
- U2TX JH-02
- U2RX JH-03
- U2RTS JH-04

Detailed information about the operation of the UART peripherals can be found in the PIC32 Family Reference Manual, Section 21, UART.
The USB Serial converter is connected to UART1. The MPIDE uses this to communicate with the boot loader. This can also be used for a serial communications interface between the ChipKIT Pro MX4 board and other software running on a PC. Resistors are used to decouple the USB serial interface and so UART1 can also be used via Pmod connector JE when it is not being used to communicate via the USB serial converter.
Note that when using the MPIDE software, devices connected to JE can interfere with the operation of the serial interface and prevent the MPIDE from successfully downloading sketches to the board. If this happens, disconnect the external device from JE until the sketch has been downloaded and then reconnect it.
When using the ChipKIT Pro MX4 with the MPIDE and the ChipKIT system, the UARTs are accessed using the Hardware Serial facility built into the system. UART1, connector JE, is accessed using the Serial object and UART2, connector JH, is accessed using Serial1.

Experiment #1
Introduction to MPIDE Integrated Development Environment

Objective

The aim of this tutorial is to teach the students some of the aspects of an important tool that can assist in software development. Its introduces you to the synthesis and analysis tools for producing Arduino code using the MPIDE integrated development environment (IDE) on the ChipKIT^{TM} Pro MX4 processor board.

Equipment needed

ChipKIT^{TM} Pro MX4 processor board

Overview

The target system for this Lab is the Digilent ChipKITTM Pro MX4 processor board. Conventional IDE systems consist of software that runs on the PC and special hardware that manages the microcontroller on the target system. A separate PIC microcontroller on the ChipKITTM Pro MX4 processor board provides the special hardware needed to interface with the PC that is running the MPIDE.

Connecting the ChipKITTM Pro MX4 Processor Board

Figure 1-1 is a picture of the ChipKIT^{TM} Pro MX4 processor board. Before connecting the URT cable to the PC, verify that the Board Power select jumpers shown at the upper left are positioned correctly. As shown, the processor board is powered using the URT connection. The board power switch is located in the upper left corner. Once the board power is turned on, the power indication LED should be on. It is helpful to connect the processor board with the URT cable and power it on before launching a new MPIDE project.

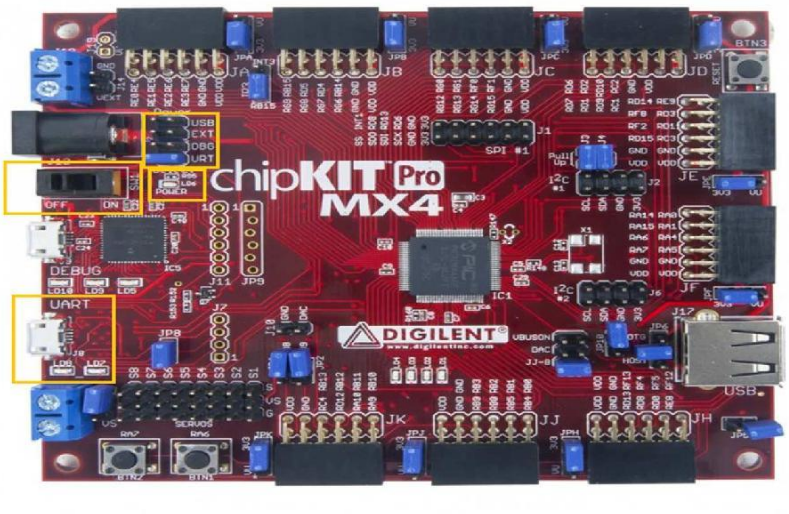


Figure 1-1 ChipKITTM Pro MX4 processor board

Launching a New MPIDE File

1. Launch the MPIDE by selecting the MPIDE icon on the PC desktop. Initially, a blank window as shown in Figure 1-2 will appear.

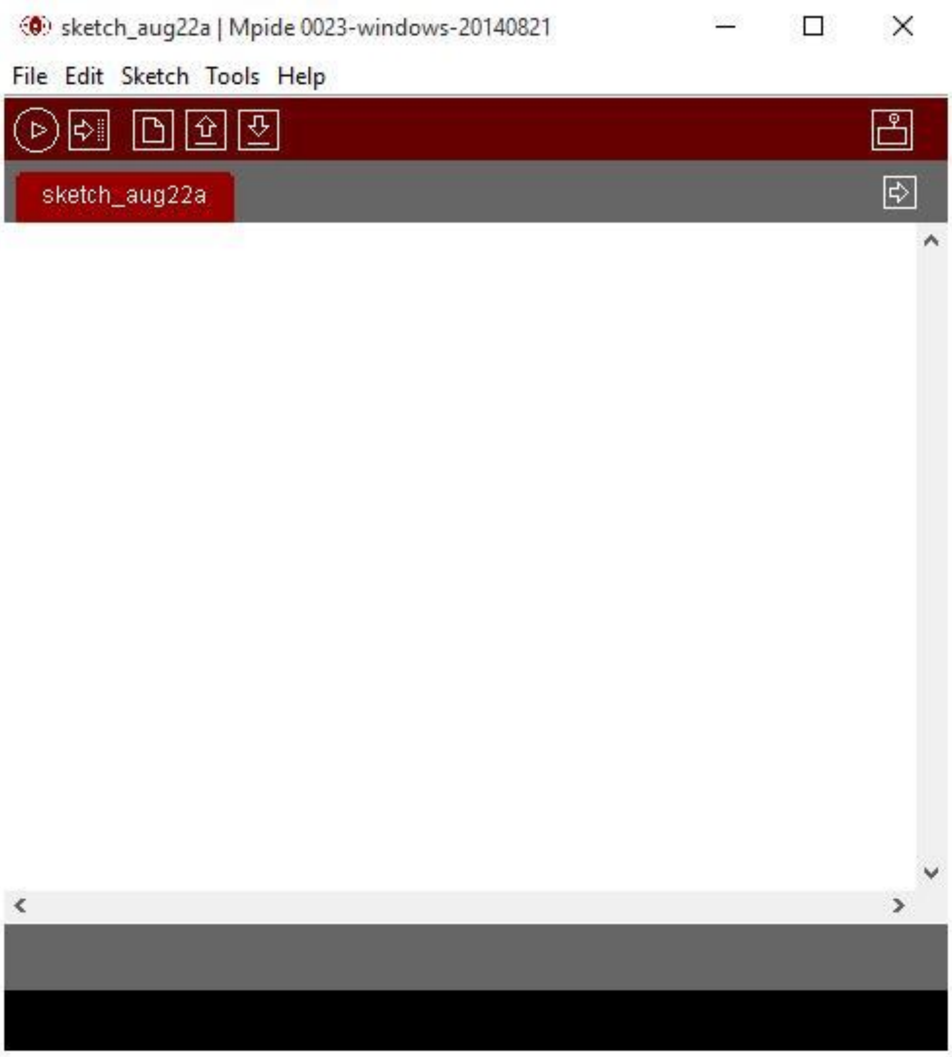
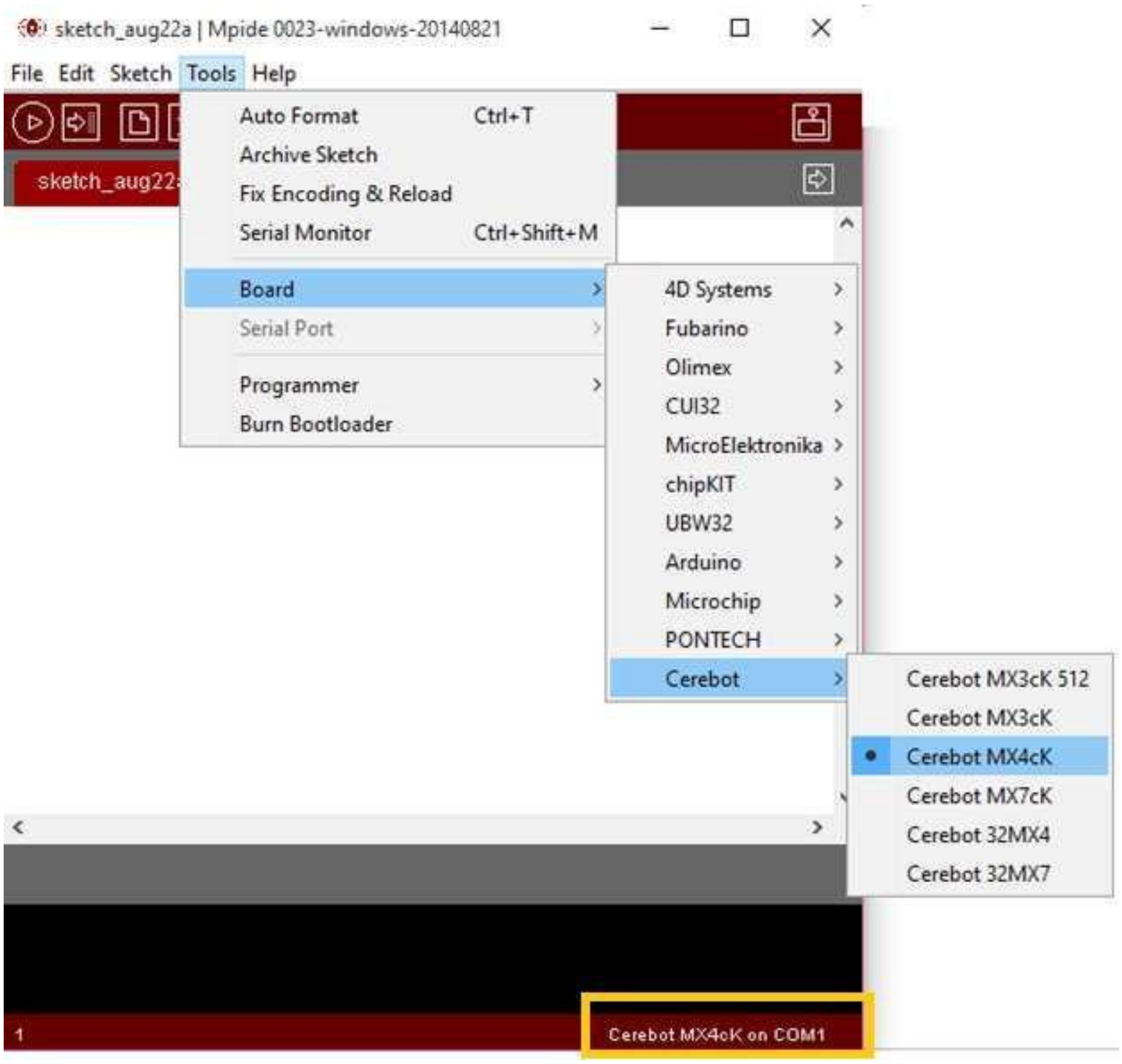

sketch_aug22a | Mpide 0023-windows-20140821
File Edit Sketch Tools Help
sketch_aug22a

Figure 1-2 MPIDE window

2. Form the Tools Menu, select the suitable Board and Serial Port as shown in Figure 1-3.



Cerebot MX4cK on COM1

Figure 1-3 Tools Menu

3. Now, the kit is correctly connected, and you can write your code.
4. Use the verify button to compile and check your code for errors as shown in Figure 1-4.

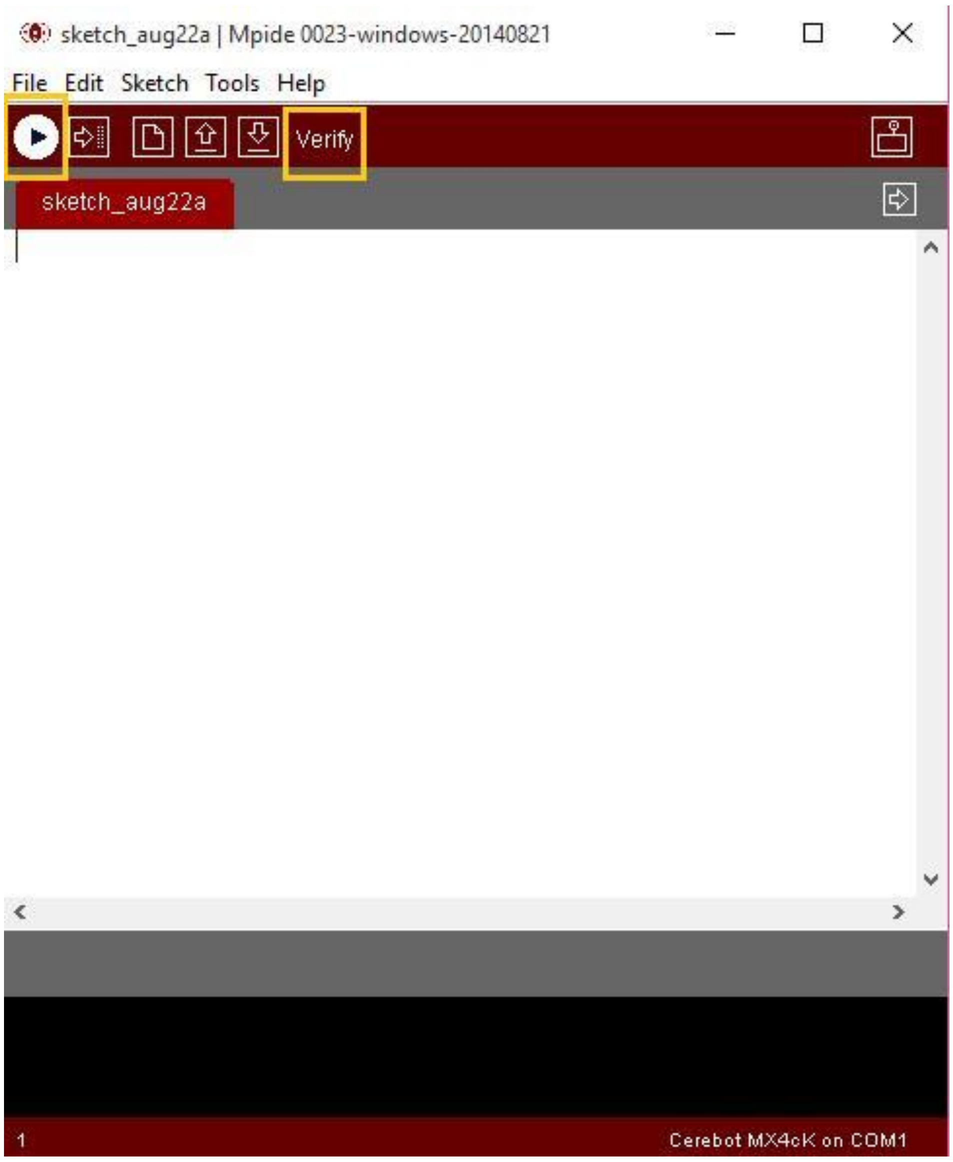

sketch_aug22a | Mpide 0023-windows-20140821
File Edit Sketch Tools Help
Verify
sketch_aug22a
1
Cerebot MX4cK on COM1

Figure 1-4 Use of the verify button

5. In case of error, or wrong instructions, we may refer to the Help Menu.
6. Once the code is correct and ready, you can use the upload button to upload it to the kit and run it (Figure 1-5).

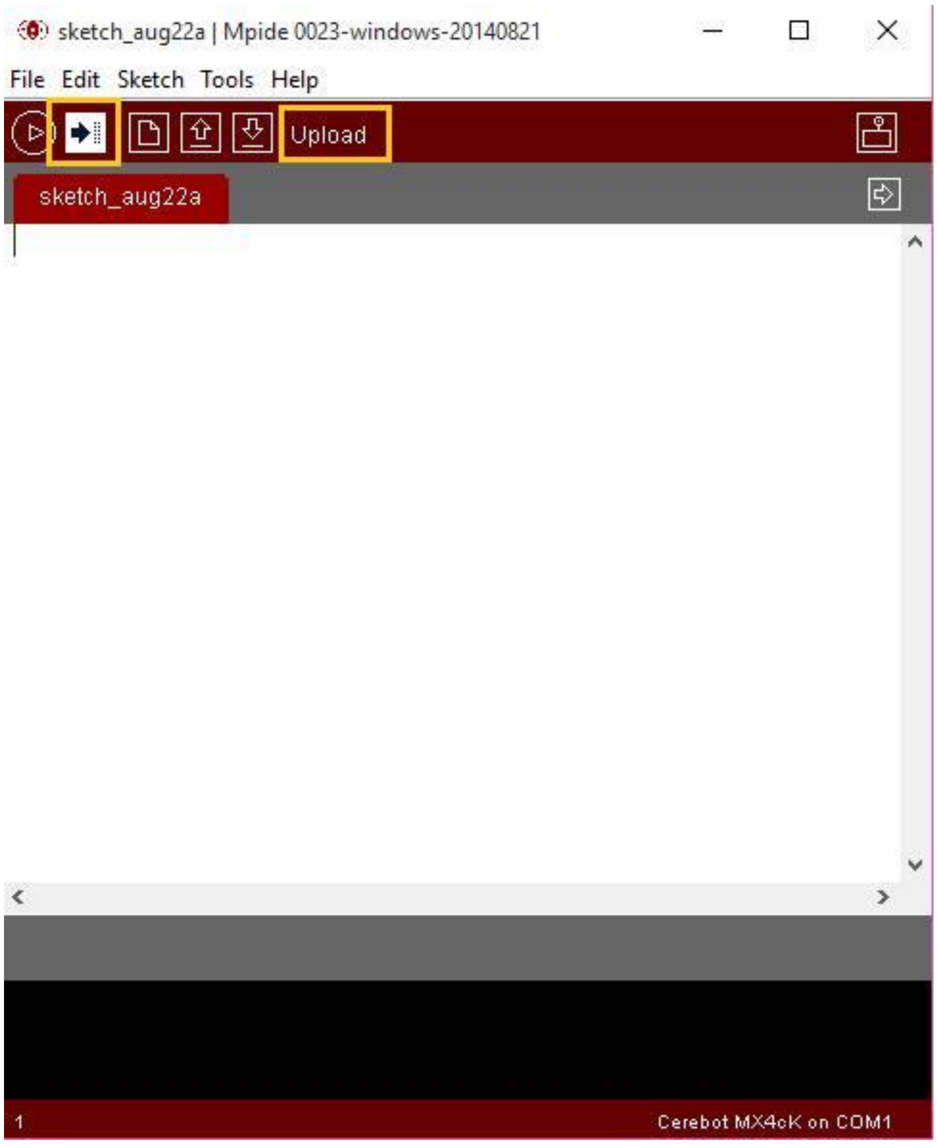

sketch_aug22a | Mpide 0023-windows-20140821
File Edit Sketch Tools Help
Upload
sketch_aug22a
1
Cerebot MX4cK on COM1

Figure 1-5 selection of the processor device

The Experiment

1- Repeat the above steps, and write a code that makes LED2 on the board flash
2- Use a delay of 500ms
Hint: you can reach LED2 by PIN_LED2

Experiment #2 Basic Input/Output

Objective

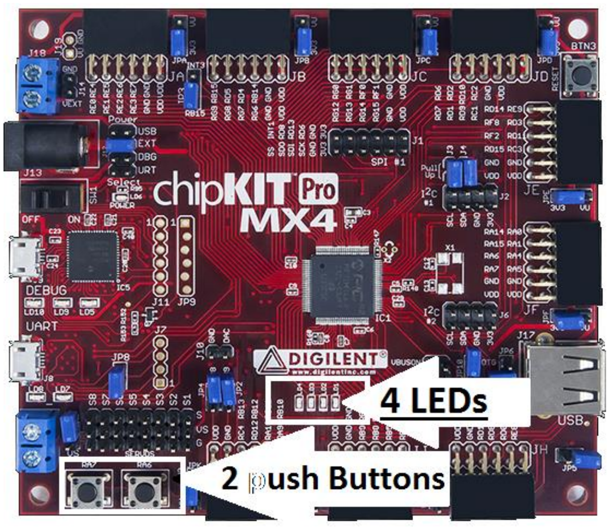
Carry out the IN/OUT digital operations (ON or OFF states) manually.

Equipment

ChipKIT^{TM} Pro MX4 processor board

The Experiment

In This experiment we will work with the basic digital Input/output using the on-kit push buttons and LEDs, see the following image



As specified in Table 2-1, we will use two buttons (BTN1 & BTN2). If BTN1 is pressed, then the LEDs will operate as a counter from 0 to 15 continuously. However, if BTN2 is pressed the LEDs will rotate one LED ON sequentially.

Note: If BTN1 is selected again (after the selection of BTN2), the counter will continue from where it stopped.

Table 2- 1 Button-controlled LEDs

Button	Exp. Function	Arduino pin #	Pin Name	Port Number
BTN1	Ascending Counter	42	PIN_BTN1	RA6
BTN2	Light rotate	43	PIN_BTN2	RA7
LED1		64	PIN_LED1	RB10
LED2		65	PIN_LED2	RB11
LED3		66	PIN_LED3	RB12
LED4		67	PIN_LED4	RB13

Experiment Hints:

The counter can be implemented by defining a variable then incremented each time.

Ex:

```
int counter=0;
.
.
.
void loop(){
.
.
.
counter++;
.
.
.
}
```

But how can we transfer the value of the counter to the LEDs.
We can do it in several ways
First, note that all the LEDs are connected to PORTB. Moreover, note that PORTB is a 16 bit port.

```
PORTB
B15  B14  B13  B12  B11  B10   B9   B8  B7   B6  B5   B4   B1  B2   B1   B0

—    —    —    —    —    —    —    —   —    —   —    —    —   —    —    —
          |    |    |     LED1
          |    |    |LED2
          |    |LED3
          |LED4
```

In order to turn the LED1 on, we need to put 1 at B10. To do so, we can simply add the value 1024 to PORTB.

Another way is by using the mod (%) operator.
Note that when counting from 0 to 15 in binary, the first bit changes every time. The second bit changes every other time, that is, when the value of the counter%2==0. The third bit changes once every fourth time, that is when the counter%4==0. The fourth bit changes once every eighth time, that is when the counter%8==0.

On last method is by using the bit wise operations. So to transfer the first bit of the counter variable to LED1, we can simply use digitalWrite(LED1, (counter&0x01)); Moreover, to transfer the second bit of the counter variable to LED2, we use digitalWrite(LED2, (counter&0x02)>>1); and so on.

Experiment #3
Keypad Handling

Objective

In this experiment, you have to write a program to handle the alphanumeric keypad that is available in the design center. In addition, you will gain knowledge on operating modules that require configuration and operation via serial UART port and serial communications in the Arduino environment.

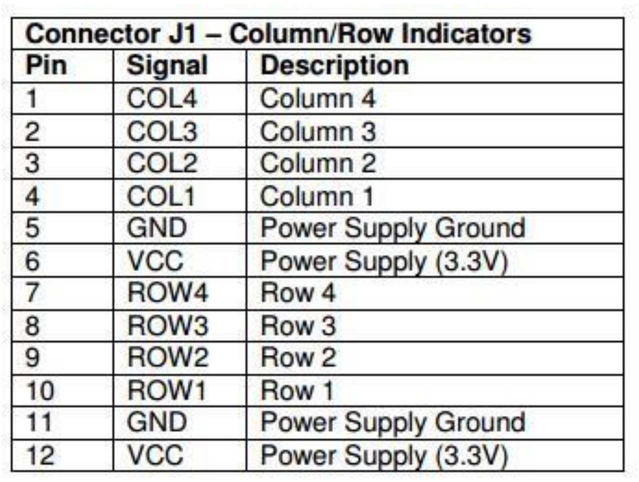
Equipment List

1. ChipKIT^{TM} Pro MX4 processor board with USB cable
2. PmodKYPD - 16-Button Keypad
3. Terminal
4. MPIDE

Overview

The PmodKYPD is an array of buttons used for input. The PmodKYPD uses a standard 12-pin Pmod header that indicates which row and which column has been pressed in the array of buttons (see Table 3-1).

Table 3-1 Connector J1 – Column/Row Indicators



Connector J1 – Column/Row Indicators		
Pin	**Signal**	**Description**
1	COL4	Column 4
2	COL3	Column 3
3	COL2	Column 2
4	COL1	Column 1
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V)
7	ROW4	Row 4
8	ROW3	Row 3
9	ROW2	Row 2
10	ROW1	Row 1
11	GND	Power Supply Ground
12	VCC	Power Supply (3.3V)

The PmodKYPD is set up as a matrix (see Figure 3-1) in which each row of buttons from left to right is tied to a row pin, and each column from top to bottom is tied to a column pin. This gives the user four-row pins and four column pins to address the button push.
To read a button's state, the column pin in which the button resides must be pulled low. This enables all of the buttons in that column. When a button in that column is pushed, the corresponding row pin will read logic low.

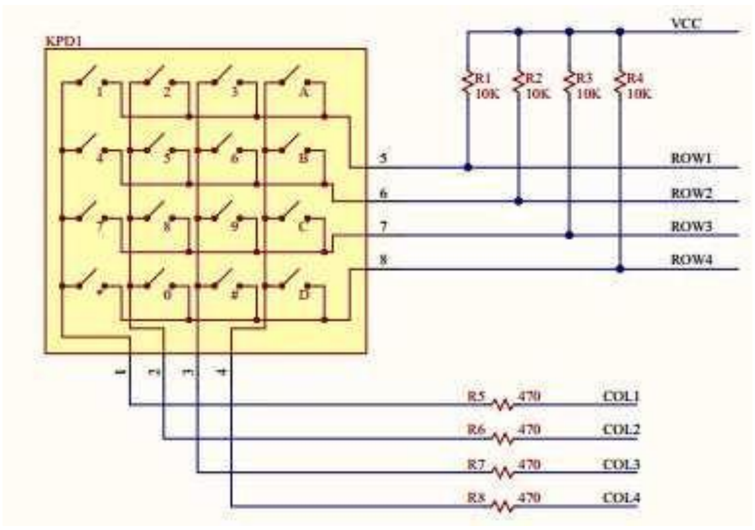


Figure 3-1 PmodKYPD schematics.

All of the buttons can be read by walking a logic 0 through each column pin (keeping the other pins at logic high) and reading the row pins. This will read the state of each button.
The Universal Asynchronous Receiver Transmitter (UART) module is one of the serial I/O modules available in PIC32MX Family devices. The UART is a full-duplex, asynchronous communication channel that communicates with peripheral devices and personal computers through protocols such as RS-232, RS-485, LIN 1.2 and IrDA®.
Although the common standard bit rates are 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, and 115200, communications are possible at any rate provided that the sender and receiver use the same rate.
The serial communication can be controlled the Serial class functions.

Experiment

This experiment will require you to implement a code to detect keystrokes from a keypad (keypad interface program).

1- Connect the PmodKYPD to connector JA (ChipKIT pin # 0 to 7)
2- Set rows to be input and the columns to be output.
3- Initialize the Serial (UART) to 9600b/s.
4- Write an Arduino program to do the following
   a) Keep checking the keypad. If any key is pressed, your program should determine the scanning code for this key.
   b) Each time a key is pressed on the keypad, the program should display the key to the Terminal (Serial).
5- Repeat the above task using the *keypad.h* library functions

Create a keypad class and use it carefully.

Hint: Do not forget to use software debouncing. (The available keys have a contact bounce <12ms)

Experiment #4
Interrupts

Objective
In this experiment, student will get knowledge of how to use external interrupts.

Equipment List
1. ChipKIT^{TM} Pro MX4 processor board with USB cable
2. Terminal
3. MPIDE
4. Push button and potentiometer resistors kit

Overview
Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. Good tasks for using an interrupt may include or monitoring user input.

1. External Interrupts
The ChipKIT^{TM} Pro MX4 microcontroller provides five external interrupt inputs. An external interrupt input can be used to generate an interrupt to the microprocessor CPU when the pin changes state. They can be programmed to interrupt on a **rising** edge or a **falling** edge on the pin. These are accessed using the attachInterrupt() and detachInterrupt() functions when using the ChipKIT MPIDE software,. The interrupt number is specified using the numbers 0-4, or preferably, the symbols EXT_INT0 through EXT_INT4.
The following gives Pmod connector position, ChipKIT pin number, and microcontroller I/O port and bit number:
- INT0 – JH-08, digital pin 53, RD0
- INT1 – JH-07, digital pin 52, RE8
- INT2 – JE-07, digital pin 36, RE9
- INT3 – JF-01, digital pin 40, RA14
- INT4 – JF-02, digital pin 41, RA15

2. attachInterrupt()
Specifies a named Interrupt Service Routine (ISR) to call when an interrupt occurs. Replaces any previous function that was attached to the interrupt.

Syntax:
attachInterrupt(interrupt, ISR, mode)

Parameters:
interrupt: the number of the interrupt (int)
ISR: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

mode: defines when the interrupt should be triggered. Four constants are predefined as valid values (in MX4ck only two is available):

- RISING: to trigger when the pin goes from low to high,
- FALLING: to trigger when the pin goes from high to low.

Experiment

1- Connect a push button and potentiometer resistors kit to the breadboard's pin corresponding to EXT_INT1.
2- Write a function to switch LED1 ON and OFF.
3- Use **attachInterrupt()** to the switch LED1 ON and OFF when the button is pressed.

Question: what about the ***void loop*** function?

Experiment #5
Analog to Digital Convertor ADC

Objective
In this experiment, you will get knowledge of how to use of on board analog to digital convertor.

Equipment List
1. ChipKIT^{TM} Pro MX4 processor board with USB cable
2. Terminal
3. MPIDE
4. Push button and potentiometer resistors kit

Overview
The PIC32MX460 microcontroller provides a **10-bit analog to digital (A/D) converter** that provides up to sixteen analog inputs. The ChipKIT Pro MX4 board provides access to 14 of them on the Pmod connectors. The converted values produced by the A/D converter will be in the range 0–1023. The analog capable I/O pins on the PIC32 ar e **not 5V tolerant**. The absolute maximum voltage rating for the analog pins is **3.6V**. (Please be careful!)
The analog inputs are accessed using the **analogRead()** function in the MPIDE software. The analog input pin number is specified using the symbols A0–A13. The digital pin numbers for the pin or the numbers 0–13 can also be used, but using the symbols A0–A13 is recommended.
The following gives the Pmod connector position, digital pin number, and microcontroller I/O port and bit number for the analog inputs (Note: We will use only analog input A4 to A7):
- A0 – JJ-01, digital pin 56, RB0
- A1 – JJ-02, digital pin 57, RB1
- A2 – JJ-03, digital pin 58, RB2
- A3 – JJ-04, digital pin 59, RB3
- A4 – JJ-07, digital pin 60, RB4
- A5 – JJ-08, digital pin 61, RB5
- A6 – JJ-09, digital pin 62, RB8
- A7 – JJ-10, digital pin 63, RB9
- A8 – JK-01, digital pin 64, RB10
- A9 – JK-02, digital pin 65, RB11
- A10 – JK-03, digital pin 66, RB12
- A11 – JK-04, digital pin 67, RB13
- A12 – JB-07, digital pin 12, RB15
- A13 – JB-10, digital pin 15, RB14.

Experiment
1- Connect a push button with a pull-up resistor and a diode (to form the external interrupt source) to the breadboard's pin corresponding to EXT_INT1.
2- Write a code to read the value of the different power/signal sources.
3- Use *attachInterrupt()* to switch between the channels.
4- Display the value on the terminal as: **Channel x = x.xxV.**

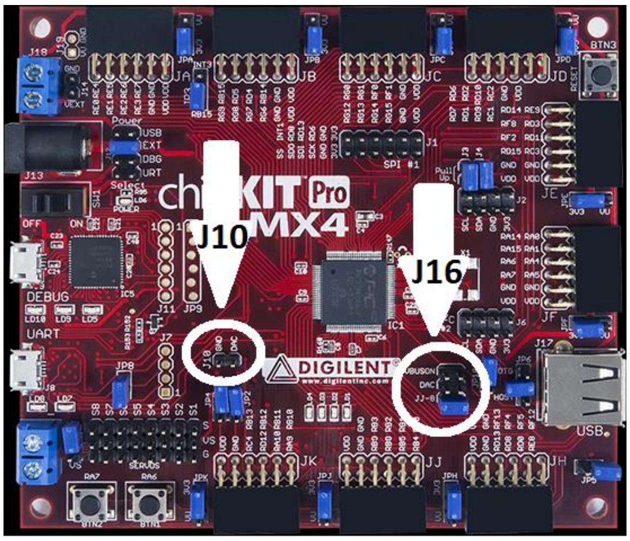
Experiment #6
Digital to Analog Convertor DAC

Objective
In this experiment, student will get knowledge of how to use on board digital to analog convertor.

Equipment List
1. ChipKITTM Pro MX4 processor board with USB cable
2. Oscilloscope
3. MPIDE

Overview
The ChipKIT Pro MX4 provides two on-board I^2C peripheral devices. These are the Microchip 24LC256 serial EEPROM and the Microchip MCP4725 Digital to Analog Converter. In the ChipKit MX4 pro there are two I^2C modules; I^2C1 and I^2C2. The Microchip 24LC256 and the Microchip MCP4725 are both connected to I^2C2. In this experiment we will work with the MCP4725 DAC.
The Microchip MCP4725 Digital to Analog Converter is a single channel, 12-bit, serial digital to analog converter that provides an analog output voltage for various uses.
The MCP4725 I^2C device address is 1100000 (0x60).
The analog output voltage of the MCP4725 is available from two places; the first one exists on J10 and the second one is controlled by J16 of the ChipKIT Pro MX4 board.
J10 is a two pin header that provides the DAC output voltage on one pin and the ground on the other pin. This header is normally used to connect the DAC output to off-board applications.
The DAC output can be connected to pin 20 of the PIC32MX460 microcontroller. This is controlled by shortening the middle two pin header of jumper block J16. If these two pin header on J16 are shortened, the DAC output will be connected to pin 20 of the PIC32MX460 microcontroller.
The PIC32MX460 has many internal analog comparators. One function of the pin 20 is that it can be connected to one input of the analog comparator #1. By allowing the DAC output to be connected to pin 20 will allow the usage of the DAC as programmable reference voltage for the comparator #1.

Pin 20 is the 20th pin of the PIC32MX460 microcontroller and is accessed via JJ-08 which is equivalent to the Arduino pin # 61 and is connected to RB05. The pin 20 has the many multiplexed functionalities (VBUSON/***C1IN+***/AN5/CN7/RB5), which can be selected by J16.



Wire Library

The wire.h library allows you to communicate with I^2C / TWI devices. I^2C devices communicate with 2 signals, called the SDA (data line) and SCL (clock line).

Basic Usage

1. **Wire.begin():** Begin using Wire in master mode, where you will initiate and control data transfers. This is the most common use when interfacing with most I2C peripheral chips.
2. **Wire.begin(address):** Begin using Wire in slave mode, where you will respond at "address" when other I2C masters chips initiate communication.

After initializing the I^2C bus, we need to communicate with the device, in our case the DAC. The data that can be send and received from the DAC can be only known from the device data sheet.

<u>NOTE: You need to download and read the MCP4725 datasheet, minimum the pages 23 to 29.</u>

Transmitting

1. **Wire.beginTransmission(address):** Start a new transmission to a device at "address". Master mode is used.
2. **Wire.write(data):** Send data. In master mode, beginTransmission must be called first.
3. **Wire.endTransmission():** In master mode, this ends the transmission and causes all buffered data to be sent.

Receiving

1. **Wire.requestFrom(address, count):** Read "count" bytes from a device at "address". Master mode is used.
2. **Wire.available():** Retuns the number of bytes available by calling receive.
3. **Wire.receive():** Receive 1 byte.

Experiment

Part 1:

1- Connect the oscilloscope channel to the DAC on board pins.
2- Write a program to generate a square wave.
3- Use the two push buttons (BTN1 and BTN2) to increment and decrement the duty cycle of the generated signal
4- Write a program to generate a saw tooth wave.

Experiment #7
Bluetooth Control

Objective

The purpose of this experiment is to familiarize the students with the Bluetooth interface (PmodBT2) and how to communicate it with the microcontroller.

Equipment List

1. ChipKITTM Pro MX4 processor board with USB cable
2. PmodBT2 - Bluetooth Interface
3. Terminal
4. MPIDE

Overview

The PmodBT2 is a powerful peripheral module employing the Roving Networks® RN-42 to create a fully integrated Bluetooth interface.
The PmodBT2 uses a standard 12-pin connection and communicates via UART (See Figure 7-1). There is a secondary SPI header on the board for updating the RN-42 firmware if needed.

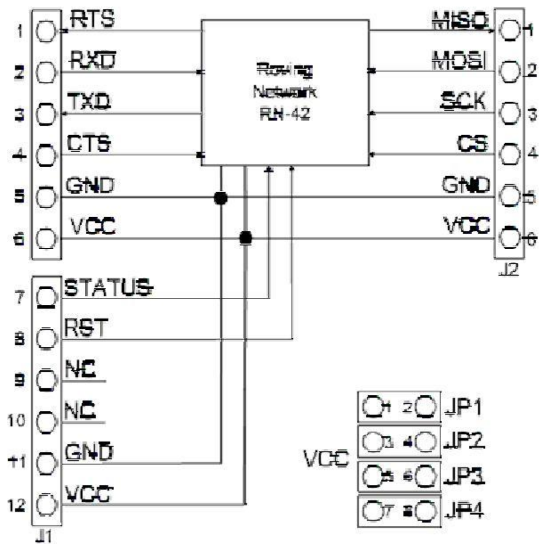


Figure 7-1 PmodBT2 Block Diagram

By default, the UART interface uses a baud rate of 115.2 kbps, 8 data bits, no parity, and a single stop bit. The startup baud rate may be customized to predefined rates or set to a specific user customized baud rate. Predefined baud rates range from 1200 to 921k.
The Bluetooth module operates in two modes: data mode (default) and command mode. While in data mode, the module operates as a data pipe. When the module receives data, it strips the Bluetooth headers and trailers and passes the user data to the UART port. When data is written to the UART port, the module constructs the Bluetooth packet and sends it out over the Bluetooth

wireless connection. Thus, the entire process of sending/receiving data to the host is transparent to the end microprocessor. See Figure 7-2.
In order to enter the command mode, the PmodBT2 must receive "$$$" to which it will respond "CMD". When in command mode, the module will respon d to a large number of commands allowing the user to customizing the module for specific applications. In order to exit command mode, send "---<cr>" (three minus signs in a row an d where <cr> stands for the carriage return character) to which the device will respond "END".

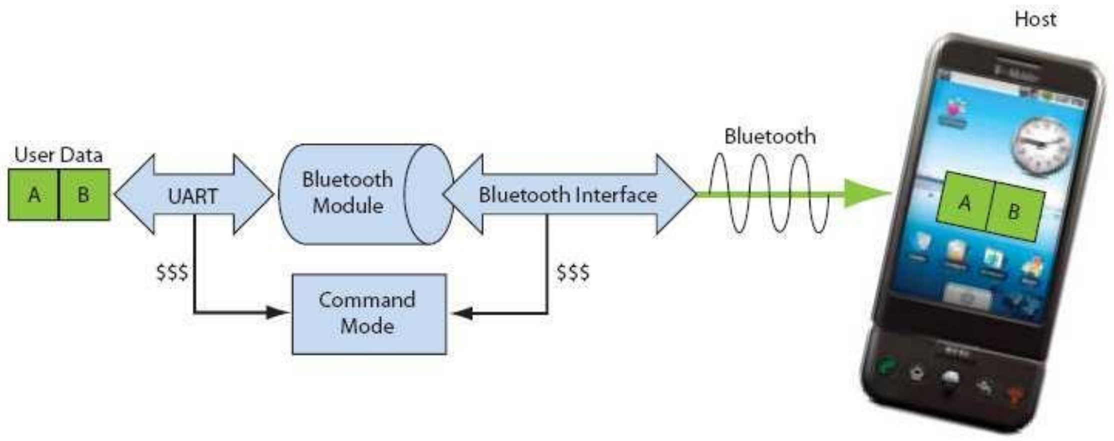


Figure 7-26 Data and Command Modes

Demo

1. Connect Pmod BT2 via UART2 (JH).
2. Connect the processor to the PC via the USB cable.
3. Install on your mobile BlueTerm app.
4. From BlueTerm app click on connect device and then select RNBT - 61E5.
5. Send "$$$" via the mobile
6. Try some command such as D, GB, GR (refer to the Bluetooth Data Module Command Reference & Advanced Information User's Guide for more information).

Serial Communication Functions

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART). It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.
You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().

Functions:

- SoftwareSerial()
- available()
- begin()
- isListening()
- overflow()

- peek()
- read()
- print()
- println()
- listen()
- write()

Experiment

This experiment will require you to implement a chat system between the mobile and the Terminal. Part 1:

1- Connect the Pmod BT2 to UART2
2- Configures the UART module at a 115.2 k data rate
3- Write a simple code to send your full name via the Bluetooth interface

Part 2:
Modify your code to implement a complete chat system between the mobile and the terminal.

Experiment #8
Servo Motor and EEPROM

Another device which is connected to I^2C2 component of the chipKIT Pro MX4, is the Microchip 24LC256 serial EEPROM. The 24LC256 is a 256Kbit (32Kbyte) serial EEPROM device which provides a non-volatile memory storage.

There are many uses for the EEPROM in an embedded system. It can be used for logging data (i.e. from sensors), for storing configurations, for storing serial numbers and/or passwords, for storing license info, and for storing (constant) messages to be sent to a screen.

The 24LC256 features

- Low-power CMOS technology:
  - Maximum write current 3 mA at 5.5V
  - Maximum read current 400 µA at 5.5V
  - Standby current 100 nA typical at 5.5V
- 2-wire serial interface bus, I2C™ compatible
- Self-timed erase/write cycle
- 64-byte Page Write mode available
- 5 ms max. write cycle time
- 1,000,000 erase/write cycles
- Data retention > 200 years

EEPROM library:

We can access the EEPROM directly using EEPROM.h library.
The EEPROM library functions:

- EEPROM.clear().
- EEPROM.read(address).
- EEPROM.write(address, value).
  - address: the location to write to, starting from 0 (*int*).
  - value: the value to write, from 0 to 255 (*byte*).

  And other functions.

Servo motor control:

The chipKIT Pro MX4 provides eight 3-pin RC hobby servo connectors, labeled S1-S8. The connectors share the I/O pins with Pmod connector JC. Individual I/O pins may be accessed through the JC connector if they're not being used to control a servo. The Servo library can be used to drive servos attached to these connectors.

RC Servos use a pulse width modulated signal, PWM, to control the servo position. The 16-bit timers in the PIC32 microcontroller have the ability to generate PWM signals using the output compare registers.

The following give the correspondence between servo connector, MPIDE digital pin number, and microcontroller

I/O Port register and bit position:

- S1 – PIN_S1, digital pin 16, RG12
- S2 – PIN_S2, digital pin 17, RG13
- S3 – PIN_S3, digital pin 18, RG14
- S4 – PIN_S4, digital pin 19, RG15
- S5 – PIN_S5, digital pin 20, RG0
- S6 – PIN_S6, digital pin 21, RG1
- S7 – PIN_S7, digital pin 22, RF0
- S8 – PIN_S8, digital pin 23, RF1

Servo Motor functions:

- Servo.attach(pin#).
- Servo.Write(angle)
  - servo: a variable of type Servo.
  - angle: the value to write to the servo, from 0 to 180
- Servo.read().
- Servo.detach().

And other functions.

Experiment:

Write a program that will read either one of two commands from the serial

a. S:angle, where angle is any integer value from 0 to 180
b. R

If you receive the first command, you should store the angle in the EEPRROM and send a message to indicating that the angle has been stored. However, if you receive the R command you should read the angle stored in the EEPROM and move the servo motor to that angle.

Experiment #9
Controlling a Stepper Motor

Objective

The purpose of this experiment is to analyze the direct driving of the stepper motor. In addition, you will gain the basic knowledge on how to control the stepper motor using your mobile phone over Bluetooth.

Equipment List

1. ChipKITTM Pro MX4 processor board with USB cable
2. PmodSTEP™
3. Stepper Motor (5V-12V, 25 Ω, unipolar or bi-polar)
4. Terminal
5. MPIDE

Overview

The stepper motor consists of two sets of field windings positioned around a permanent magnet rotor, as illustrated in Figure 8-1.

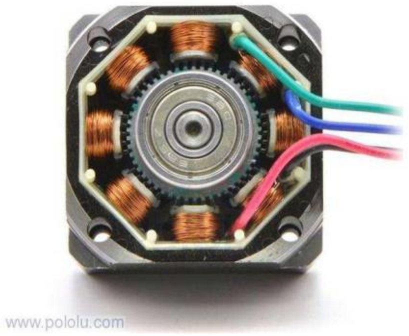


Figure 8-1 Photograph field coils and rotor of a stepper motor

The combinations of voltages applied to the four control terminals of the field windings control the magnitude and direction of the current through the windings, as illustrated in Figure 8-2. The current through the windings create an electromagnet. The motor shaft rotates to a position that minimizes the reluctance path between the field winding electromagnet's north and south poles of the rotor.

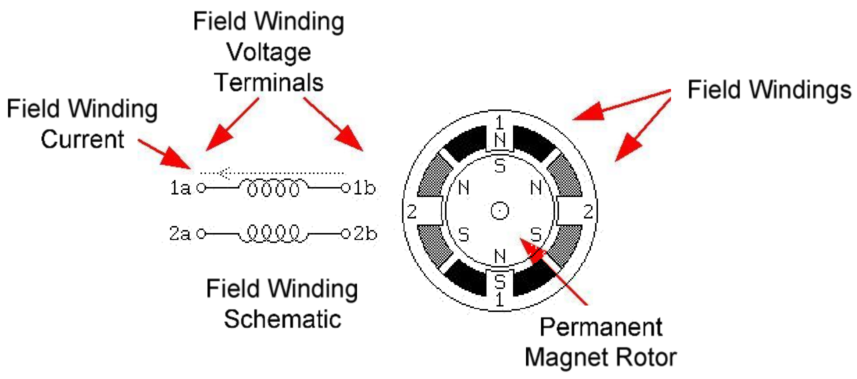


Figure 8-2 Bipolar Stepper motor diagram

Considering the combinations of voltages on the winding terminals as possible control states, there are only eight states that produce current in the field windings, as shown in Table 8-1 below. In order to move the rotator shaft from one stable position to the physically adjacent stable position, the control voltages must switch to one of four out of the eight possible combinations of voltages. The action of moving from one stable position to an adjacent stable position is referred to as either a full-step or a half-step. Half-step increments are half the angular rotation of full-steps. Repeating a sequence of full- or half-step movements at a uniform rate will cause the rotator shaft to appear to rotate at a constant speed in discrete steps.

Table 8-1 Stepper motor control codes.

Step Control		Winding Voltage			
Step Name	Hex Code	“1a”	“1b”	“2a”	“2b”
S0_5	0x0A	H	L	H	L
S1	0x08	H	L	L	L
S1_5	0x09	H	L	L	H
S2	0x01	L	L	L	H
S2_5	0x05	L	H	L	H
S3	0x04	L	H	L	L
S3_5	0x06	L	H	H	L
S0	0x02	L	L	H	L

The stepper motors used in this experiment are designed to require 200 full steps for the rotor shaft to complete one full revolution, or 1.8 degrees per step. 400 half-steps are required to make one revolution, or 0.9 degrees per half-step.

PmodSTEP

The PmodSTEP driver module, assigned designations SM1 through SM4, is used to control the voltages on the four stepper motor field windings The PmodStep can be powered either from Vcc on the microcontroller board, or alternatively, from the external power screw terminal block. (See Appendix E for PmodSTEP connection details.)

Experiment

Part 1: Write a program to

1- Connect the Stepper motor to the PmodSTEP (4 pins connection)
2- Connect the PmodSTEP to connector JC (ChipKIT pins: 20, 21, 22 and 23)
3- Rotate stepper motor on both directions: clockwise and counter clockwise.
4- Control the speed of the stepper motor using BTN1 and BTN2 (to increase and decrease)
5- Rotate a stepper motor to a predefined angle.
6- Use the Arduino Stepper Library (Stepper.h) to control the stepper motor.

Part 2:

1- Write a program to control the stepper using Terminal commands. The first step is to send a single command, formatted as follows:

L/R <angel> <speed> [new line= \n]

Where: **L** indicates left direction and **R** indicates right direction

<angle> is the angle value

<speed> the speed of the stepper motor

2- After receiving, the command the controller rotate the motor according to the command and will reply with either:

a) **OK**[new line= \n]: if the command is correct
b) **ERROR**[new line= \n]: if the command is wrong

An example as it should appear on the terminal:

```
L 90 100
OK
R 180 50
OK
L L 10 10
ERROR
C 200 80
ERROR
```

3- Modify the previous commands by adding:

a) ‘ **S**’ character to indicate single command. Commands will be:

SL/SR <angel> <speed>[new line= \n]

b) ‘ **M**’ character at the beginning of each command to program a sequence of commands to be stored and executed afterwards. Commands will be:

ML/MR <angel> <speed>[new line= \n]

c) Send “ **END**” to indicate the end of sequence and to make the c ontroller execute the sequence now.

d) After receiving the sequence of commands, the controller will reply with either:

RECEIVED[new line= \n]: if the command is correct
OK[new line= \n]: if “END” is sent
ERROR[new line= \n]: if the command is wrong

An example as it should appear on the terminal:

```
ML 90 100
RECEIVED
MR 180 50
RECEIVED
ML 45 100
RECEIVED
MR 360 50
RECEIVED
END
OK
```

Experiment #10
Using SPI and Accelerometer

Objective

In this experiment you will learn the basics of Serial Peripheral Interface (SPI). Where you will use what you learn about SPI in order to control and access readings of an accelerometer module. This module “PmodACL” is powered by “ADXL345” 3-acc ess accelerometer chip.

Equipment List

1. ChipKITTM Pro MX4 processor board with USB cable
2. PmodACL™
3. Terminal
4. MPIDE

Overview

1- Serial Peripheral Interface (SPI)

SPI is a four wire synchronous serial interface and SPI devices can operate either as master devices or as slave device. The PIC32 microcontroller labels the four SPI signals as Slave Select (SS), Serial Data Out (SDO), Serial Data In (SDI), and Serial Clock (SCK). A master device transmits SS, SDO and SCK, and receives SDI. A slave device receives SS, SDI, and SCK and transmits SDO. The SS signal is used to enable the slave device, and this signal is only significant for slave devices. A master device can use any general-purpose I/O pin to generate SS to enable the slave. An SPI transaction begins with the master device bringing SS low. When the slave sees SS go low, it becomes enabled and waits for the master to send data. The master shifts data out on SDO and simultaneously shifts data in on SDI. The slave device receives data from the master on its SDI pin and simultaneously sends data to the master on its SDO pin. Each time the master sends a byte to the slave, it simultaneously receives a byte from the slave.

The PIC32MX460 microcontroller provides two Serial Peripheral Interfaces, SPI1 and SPI2. SPI2 is accessed via Pmod connector JB and SPI1 is accessed via connector J1. Because of the way that peripheral functions are shared the pins for SPI1, the signals on J1 are shared with various Pmod connectors around the board. For this reason, when using only a single SPI port, SPI2 is the preferred port to use.

The following gives the mapping between SPI Signals and connector pins:

- SS2 JB-01
- SDO2 JB-02
- SDI2 JB-03
- SCK2 JB-04
- SS1 J1-01 (also JD-03)
- SDO1 J1-02 (also JH-08)
- SDI1 J1-03 (also JK-10)
- SCK1 J1-04 (also JD-09)

When using the Cerebot MX4cK with the MPIDE and the ChipKIT system, the SPI ports are either accessed using the standard ChipKIT SPI library or using the Digilent DSPI library. The standard SPI library supports access to a single SPI port, SPI2. This is accessed using the SPI object.

Functions:

- begin()
- setDataMode()
- transfer()
- beginTransaction()
- endTransaction()
- setBitOrder()
- setClockDivider()
- end()
- usingInterrupt()

2- PmodACL

The PmodACL is a 3-axis digital accelerometer module powered by the Analog Devices ADXL345. Features include:

- user-selectable resolution
- single-tap/double-tap detection
- activity/inactivity monitoring
- free fall detection
- SPI and I2C interfaces

Functional Description

The PmodACL uses a standard 12-pin connector and can communicate via SPI or I2C. A pull-up resistor on the ~SS line keeps the ADXL345 in I2C mode unless the host drives the line low, in which case the device will communicate via SPI.

Interface

All communications with the device must specify a register address and a flag indicating whether the communication is a read or a write. This is followed by the actual data transfer. Device configuration is performed by writing to control registers within the device. Accelerometer data is accessed by reading device registers.

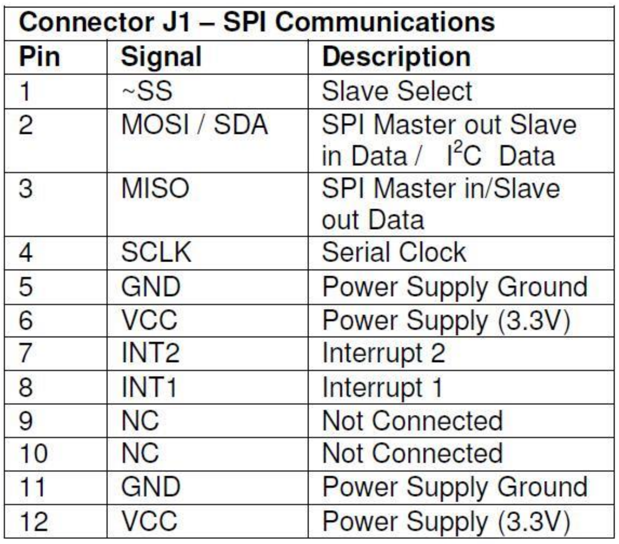
ADXL345

The ADXL345 is a complete 3-axis acceleration measurement system with a selectable measurement range of ±2 *g*, ±4 *g*, ±8 *g*, or ±16 *g* (*gravitational*). It measures both dynamic acceleration resulting from motion or shock and static acceleration, such as gravity, that allows the device to be used as a tilt sensor.

The sensor is a polysilicon surface-micromachined structure built on top of a silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against forces due to applied acceleration.

Deflection of the structure is measured using differential capacitors that consist of independent fixed plates and plates attached to the moving mass. Acceleration deflects the proof mass and unbalances the differential capacitor, resulting in a sensor output whose ampli-tude is proportional to acceleration. Phase-sensitive demodulation is used to determine the magnitude and polarity of the acceleration.

Interface Connector Signal Description



Connector J1 – SPI Communications		
Pin	**Signal**	**Description**
1	~SS	Slave Select
2	MOSI / SDA	SPI Master out Slave in Data / I^2C Data
3	MISO	SPI Master in/Slave out Data
4	SCLK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V)
7	INT2	Interrupt 2
8	INT1	Interrupt 1
9	NC	Not Connected
10	NC	Not Connected
11	GND	Power Supply Ground
12	VCC	Power Supply (3.3V)

Serial Communications

I^2C and SPI digital communications are available. In both cases, the ADXL345 operates as a slave. In SPI mode, the CS pin is controlled by the bus master. In both SPI and I2C modes of operation, data transmitted from the ADXL345 to the master device should be ignored during writes to the ADXL345.

Register Map

Address Hex	Name	Type	Reset Value	Description
0x2D	POWER_CTL	R/W	00000000	Power-saving features control
0x31	DATA_FORMAT	R/W	00000000	Data format control
0xB2	DATAX0	R	00000000	X-Axis Data 0
0xB3	DATAX1	R	00000000	X-Axis Data 1
0xB4	DATAY0	R	00000000	Y-Axis Data 0
0xB5	DATAY1	R	00000000	Y-Axis Data 1
0xB6	DATAZ0	R	00000000	Z-Axis Data 0
0xB7	DATAZ1	R	00000000	Z-Axis Data 1

1- **Register 0x2D—POWER_CTL (Read/Write)**

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

2- **Register 0x31—DATA_FORMAT (Read/Write)**
The DATA_FORMAT register controls the presentation of data to Register 0x32 through Register 0x37.

D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

Range Bits: These bits set the *g* range as described in the following table

Setting		
D1	**D0**	***g* Range**
0	0	±2 *g*
0	1	±4 *g*
1	0	±8 g
1	1	±16 *g*

3- **Register 0xB2 to Register 0xB7**
DATAX0, DATAX1, DATAY0, DATAY1, DATAZ0, DATAZ1 (Read Only)
These six bytes (Register 0xB2 to register 0xB7) are eight bits each and hold the output data for each axis.
a- Register 0xB2 and Register 0xB3 hold the output data for the x-axis b-
Register 0xB4 and Register 0xB5 hold the output data for the y-axis c-
Register 0xB6 and Register 0xB7 hold the output data for the z-axis.
The output data is twos complement, with DATAx0 as the least significant byte and DATAx1 as the most significant byte, where x represent X, Y, or Z. The DATA_FORMAT register (Address 0x31) controls the format of the data.
It is recommended that a multiple-byte read of all registers be performed to prevent a change in data between reads of sequential registers.

Experiment

In this experiment you are required do the following:

1- Connect PmodACL module through **JB** port.
2- Configure the SPI connection for the ADXL345, **SPI_MODE3**.
3- Configure the slave select control (Chip Select) to be on pin number **8**.
4- Before communication starts, the Chip Select pin needs to be set **HIGH**.
5- Communication starts with a **low** signal and ends with a **high** signal on the Chip Select pin.
6- Write to the **DATA_FORMAT** register to set the range to **±4g**.
7- Write to the **POWER_CTL** in order to **set** the **Measure bit** that will initiate the operation of the accelerometer.
8- Keep reading the accelerometer sensor values: **x**, **y**, and **z**, where each value has two registers. Read the lower and the upper bytes of the three values, and then combine those two bytes to form a 16-bit word.
9- Display the reading you got in step 6 on the serial port.
10- Do not forget to add delay after each transfer.

<u>Experiment #11</u>
<u>Controlling a DC Motor</u>

Objective

The purpose of this experiment is to learn how to generate a proportional output using the output compare resource on the PIC32MX processor so that you can implement digital-to-analog conversion with pulse width modulation (PWM). The proportional output will be used to control the speed of a DC motor.

Equipment List

1. ChipKITTM Pro MX4 processor board with USB cable
2. Digilent DC Motor
3. Digilent PmodDHB1™
4. MPIDE

Overview

The information Digital I/O works well for generating and detecting discrete events. But the real world is continuous; thus, for microprocessors to have value, they need to have the capability to input and output analog signals. A digital-to-analog converter (DAC) converts a digital, or binary, value to an analog voltage. An analog-to-digital converter (ADC) converts a voltage in a specified range to a binary value that represents the magnitude of the signal.

Pulse Width Modulation

There are many different ways to control the speed of motors but one very simple and easy way is to use <u>Pulse Width Modulation</u>. As its name suggests, pulse width modulation speed control works by driving the motor with a series of "ON-OFF" pulses and varying the duty cycle, the fraction of time that the output voltage is "ON" compared to when it is "OFF", of th e pulses while keeping the frequency constant. The power applied to the motor can be controlled by varying the width of these applied pulses and thereby varying the average DC voltage applied to the motors terminals. By changing or modulating the timing of these pulses the speed of the motor can be controlled, i.e., <u>the longer the pulse is "ON", the faster the motor will rotate and likewise, the shorter the pulse is "ON" the slower the motor will rotate.</u>
In other words, the wider the pulse width, the more average voltage applied to the motor terminals, the stronger the magnetic flux inside the armature windings and the faster the motor will rotate and this is shown below.

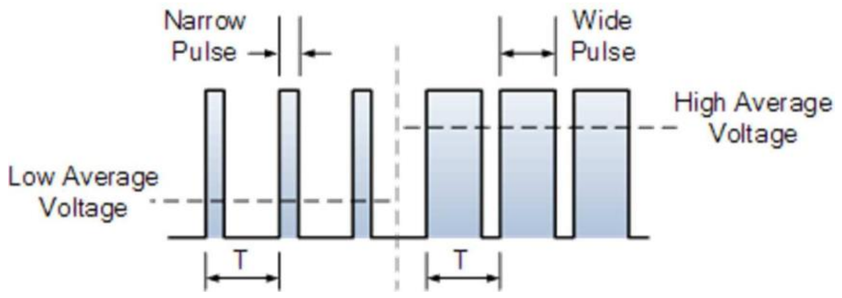


Figure 10-7 Pulse Width Modulated Waveform

The DC motor driver circuit provided by the PmodDHB1 allows control of the motor speed by connecting the Output Enable pin to the PIC32 PWM output as well as motor direction control by connection to another PIC32 I/O pin.

Experiment

The objective of this experiment is to implement an open-loop motor speed control.

1- Connect the DC motor to the PmodDHB1
2- Connect the PmodDHB1to connector JH
3- Use the pins 52, 53, 54, and 55 for the motor control
4- Pin 52 is used to control the speed while pin 53 is used to control the direction
5- Use BTN1 and BNT2 to increase and decrease the speed
6- Read the values of pins 54 and 55 and print them on the terminal
7- Reuse BTN1 to change the direction of the motor
8- Do not forget to use delay.

Experiment #12
Positioning Satellites and LCD

➢ Objective

The purpose of this experiment is to familiarize the students with the GPS Receiver Peripheral Module (PmodGPS). The PmodGPS can add satellite positioning accuracy to any embedded system.

➢ Equipment List

1. ChipKITTM Pro MX4 processor board with USB cable
2. PmodGPS™
3. LCD
4. MPIDE

➢ Overview

The Global Positioning System (GPS) is a space-based satellite navigation system that provides location and time information in all weather conditions.
There are three parts to a GPS system: a constellation of between 24 and 32 solar-powered satellites orbiting the earth in orbits at an altitude of approximately 20000 kilometers, a master control station and four control and monitoring stations (on Hawaii, Ascension Islands, Diego Garcia and Kawajale) and GPS receivers.
Each of the satellites is in an orbit that allows a receiver to detect at least four of the operational satellites. The satellites send out microwave signals to a receiver where the built-in computer uses these signals to work out your precise distance from each of the four satellites and then triangulates your exact position on the planet to the nearest few meters based on these distances. Hence, the process of measuring the distance from satellite to GPS receiver is based on timed signals.
GPS satellites synchronize operations so that these repeating signals are transmitted at the same instant. The signals, moving at the speed of light, arrive at a GPS receiver at slightly different times because some satellites are further away than others. The distance to the GPS satellites can be determined by estimating the amount of time it takes for their signals to reach the receiver. When the receiver estimates the distance to at least four GPS satellites, it can calculate its position in three dimensions.
In fact, signals from just three satellites are needed to carry out this trilateration process; the calculation of your position on earth based on your distance from three satellites. The signal from the fourth satellite is redundant and is used to confirm the results of the initial calculation.

➢ LiquidCrystal Library

This library allows an Arduino board to control LiquidCrystal displays (LCDs) based on the Hitachi HD44780 (or a compatible) chipset, which is found on most text-based LCDs. The library works with in either 4- or 8-bit mode (i.e. using 4 or 8 data lines in addition to the rs, enable, and, optionally, the rw control lines).

Function

- LiquidCrystal()
- begin()
- clear()
- home()
- setCursor()
- write()
- print()
- cursor()

LiquidCrystal()

Description

Creates a variable of type LiquidCrystal. The display can be controlled using 4 or 8 data lines. If the former, omit the pin numbers for d0 to d3 and leave those lines unconnected. The RW pin can be tied to ground instead of connected to a pin on the Arduino; if so, omit it from this function's parameters.

Syntax

```
LiquidCrystal(rs, enable, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

Parameters

`rs`: the number of the Arduino pin that is connected to the RS pin on the LCD

`rw`: the number of the Arduino pin that is connected to the RW pin on the LCD (optional)

`enable`: the number of the Arduino pin that is connected to the enable pin on the LCD

`d0, d1, d2, d3, d4, d5, d6, d7`: the numbers of the Arduino pins that are connected to the corresponding data pins on the LCD.

`d0, d1, d2, and d3`: are optional; if omitted, the LCD will be controlled using only the four data lines (d4, d5, d6, d7).

➢ PmodGPS: Functional Description and Interface

As illustrated in table 12-1, the PmodGPS uses a standard 6-pin connector and communicates via a 2-wire Universal Asynchronous Receiver/Transmitter (UART) [8]. The interface operates at a default baud rate of 9.6 kBd, 8 data bits, no parity, and with single stop bits. However, users can change the baud rate to predefined values that range from 4.8 kBd to 115.2 kBd.

The PmodGPS also has a 2-pin connector for control of the NRST pin to the module and the Radio Technical Commission for Maritime services, or RTCM pin for Differential Global Positioning System (DGPS) data using RTCM protocols.

Table 12-1- Interface Connector Signal Descriptions

Connector J1		
Pin	**Signal**	**Description**
1	3DF	3D-Fix Indicator
2	RX	Receive
3	TX	Transmit
4	1PPS	1 Pulse Per Second
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3v)
Connector J2		
Pin	**Signal**	**Description**
1	~RST	Reset (active low)
2	RTCM	DGPS data pin (contact GlobalTop for use)

The 3DF pin on J1 indicates the status of the user's positional fix. When the module has a constant fix (2D or 3D) this pin stays low, if the module is unable to get a fix then the pin will toggle every second. (See figure 12-1) LD1 also follows this same behavior pattern in order to give the user a visual representation.

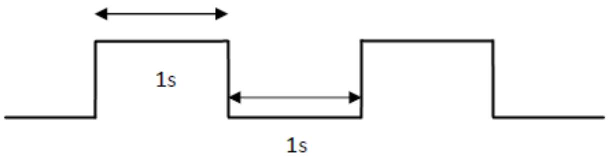


Figure 12-1- 3DF Pin output without a fix

GPS - NMEA sentence information

The PmodGPS uses sentences based on National Marine Electronics Association (NMEA) protocols for data output. Once the module connects to at least four satellites, it sends several messages containing GPS data to the microcontroller.

Each NMEA message begins with a ($) dollar sign. The next five characters are the talker ID and the arrival alarm. The PmodGPS talker ID is "GP" and the arrival alarm is the specific sentence output descriptor. Individual comma separated data fields follow these five characters. After the data fields there is an asterisk followed by a checksum. Each sentence should end with <CR><LF>.

The most important NMEA sentences include the GGA which provides the current Fix data, and the GSA which provides the Satellite status data[1].

$GPGGA - Global Positioning System Fix Data

$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65

Where:

$GPGGA	Message ID
064951.000	UTC Time (hhmmss.sss)
2307.1256	Latitude (ddmm.mmmm)
N	N/S indicator
12016.4438	Longitude (dddmm.mmmm)
E	E/W indicator
1	Position Fix Indicator
8	Satellites used
0.95	Horizontal Dilution of Precision (HDOP)
39.9	MSL Altitude
M	Units
17.8	Geoidal Separation
M	Units
	Age of Diff. Corr.
*65	Checksum

$GPGSV - Satellites in view

$GPGSV,3,1,12,29,36,029,42,21,46,314,43, 26,44,020,43,15,21,321,39*7D

$GPGSV,3,2,12,22,28,259,16,27,13,107,,09,11,130,,16,09,288,25*79
$GPGSV,3,3,12,30,08,210,33,06,08,320,22,25,02,188,26,14,01,203,21*7B

Where:

$GPGSV	Message ID
3	Number of Messages
1	Message Number
12	Satellites in View
29	Satellite ID (CH1)
36	Elevation (CH1)
029	Azimuth (CH1)
42	SNR (C/No)

[1] Other sentences may repeat some of the same information but will also supply new data, such as: RMC, VTG, MSS …etc. (For more information see NMEA Reference Manual).

….

15	Satellite ID CH(4)
21	Elevation (CH4)
321	Azimuth (CH4)
39	SNR (C/No)
*7D	Checksum

strtok() function to sentence information

To extraxt the message (string) we can use the strtok() function. This fuction parses a string into a sequence of tokens. On the first call to strtok() the string to be parsed should be specified instr. In each subsequent call that should parse the same string, str should be NULL.

➢ Experiment

Once the module is connected to UART and we got a fix position (PF is 0), several messages containing GPS data will be sent to the microcontroller. The goal here is to filter these messages into data field so that they can be accessed individually.

However, since we need to be insight of a satellite which is not possible, we will use a constant string instead

Note: There are three common formats[2]:

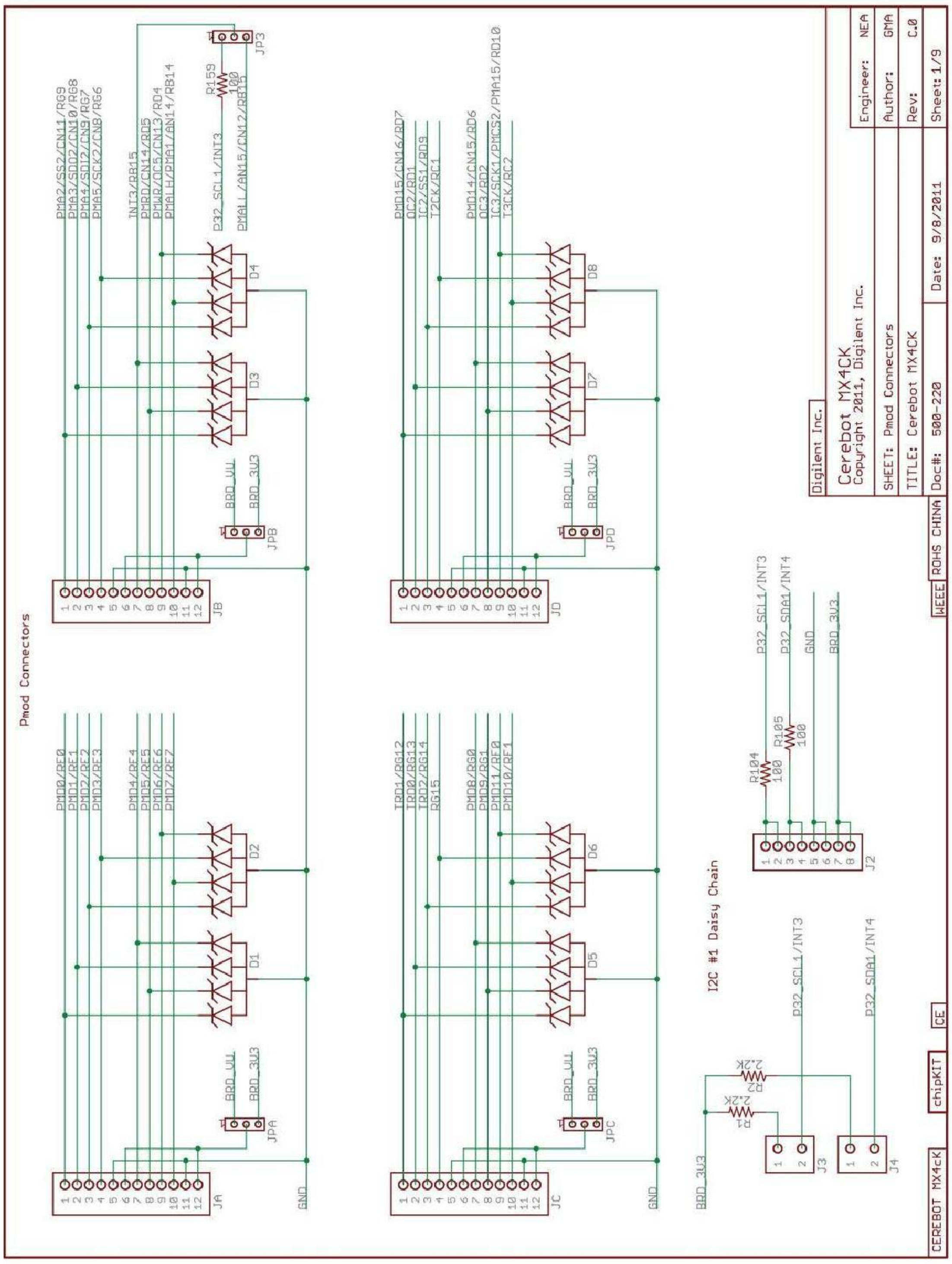
- Degrees, Minutes and Seconds (DD° MM' SS.S")
  This is the most common format used to mark maps.
- Degrees and Decimal Minutes
- Decimal Degrees

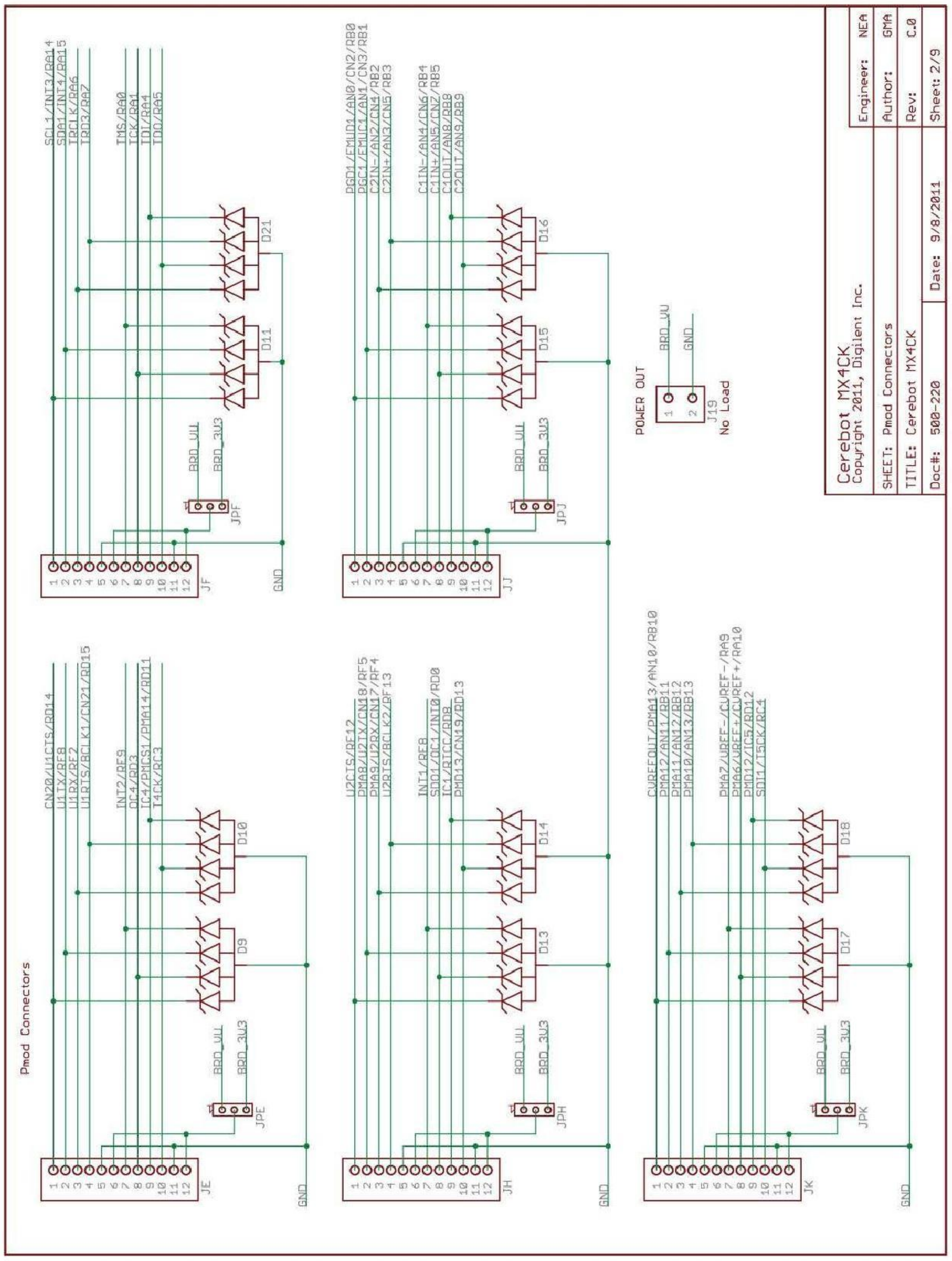
1- Connect the LCD to connector JA (r/w at pin 1, rs at pin 3, enable at pin 2, and d4 to d7 at pins 7 to 4)
2- Use the following parameters (instead of the GPS module):

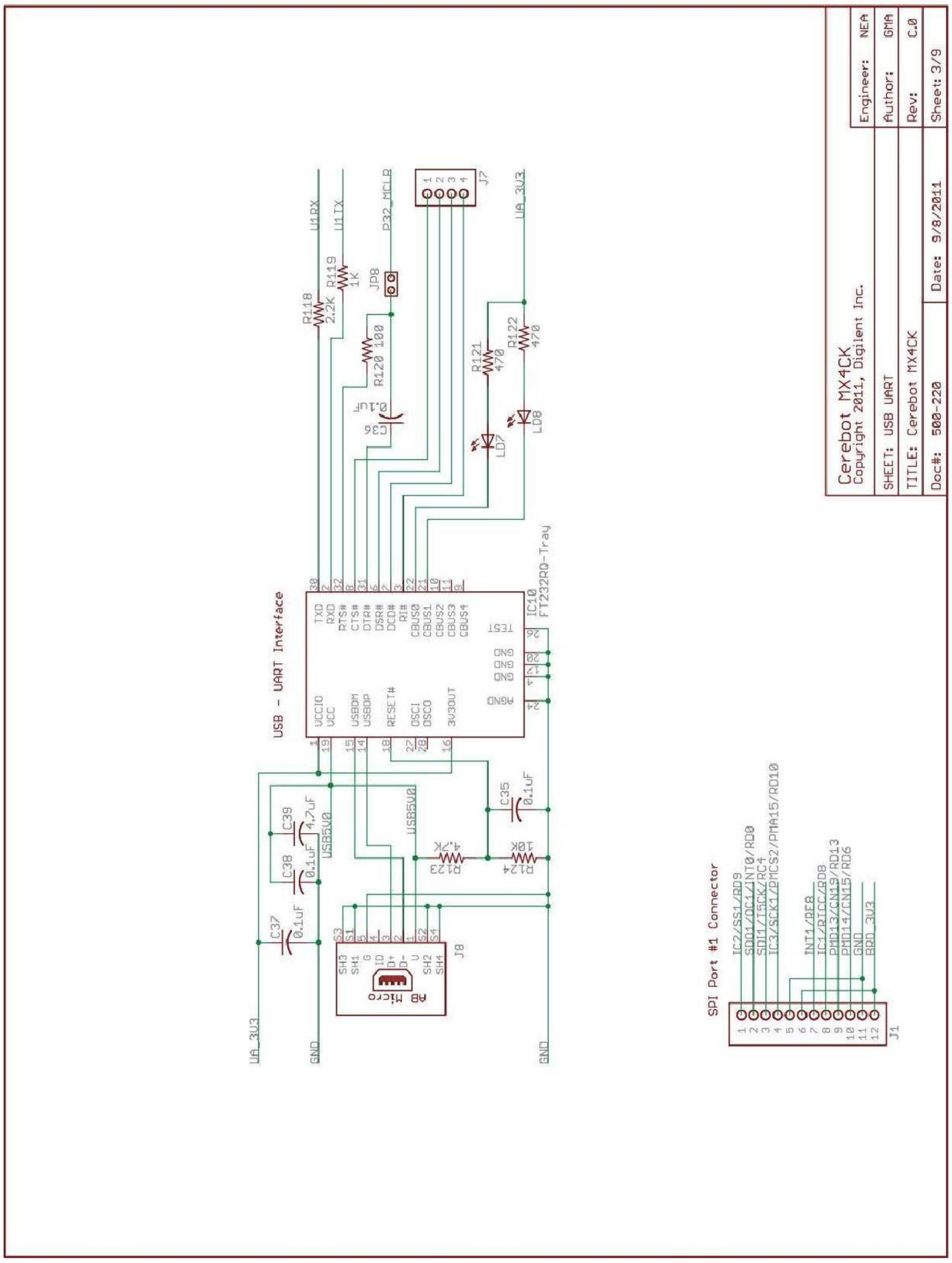
```
char *next1_token,*next2_token;
char b[]="msgID,utcTime,lat,n/s,Long,e/s,pos,sat,hdop,msl,units,gs,units,cor,checksum";
char buf[]="$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,
 *65";
```

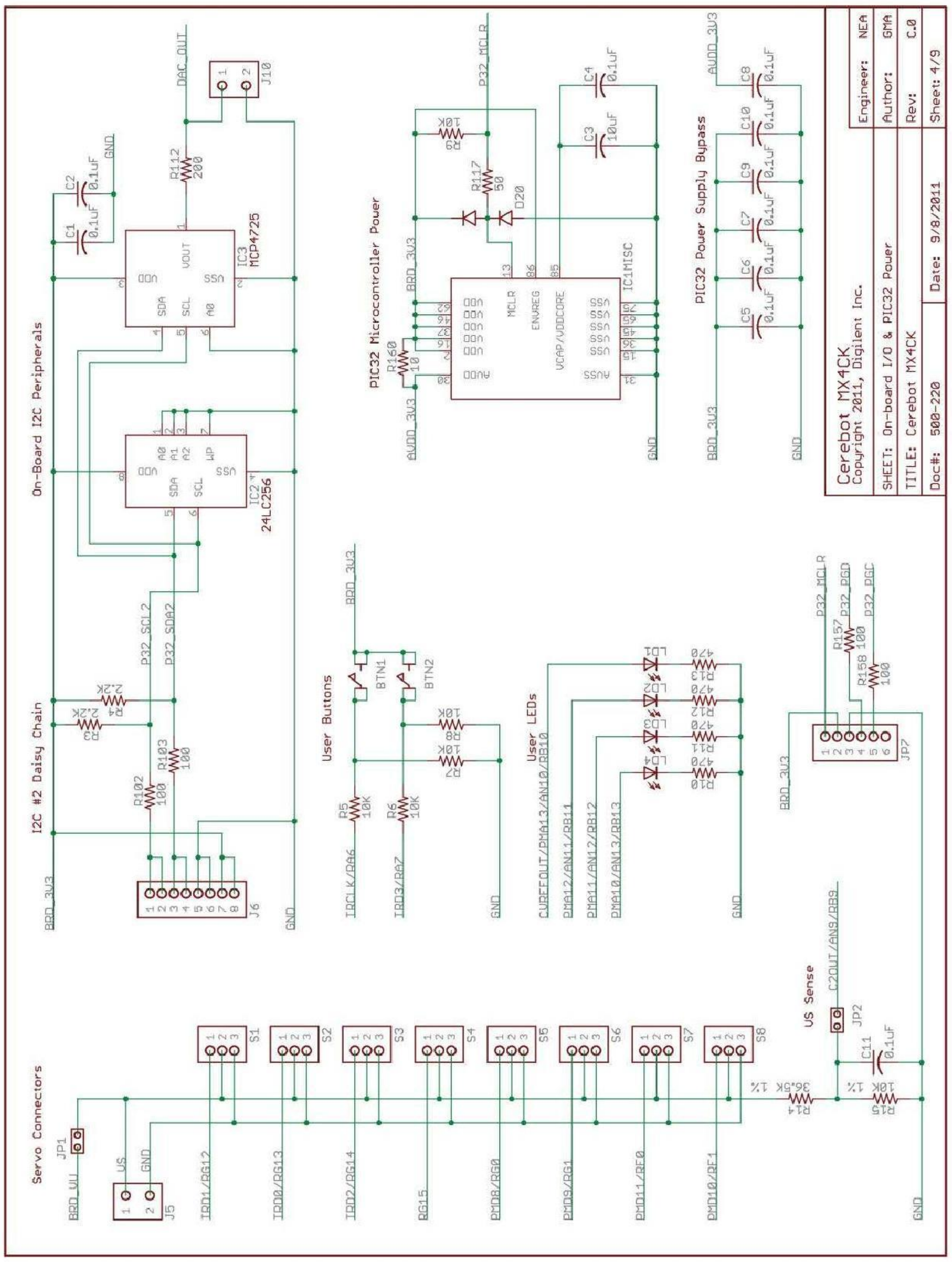
3- Use BTN1 to view each information separately on the LCD
Hence: you can print the information first on the serial port

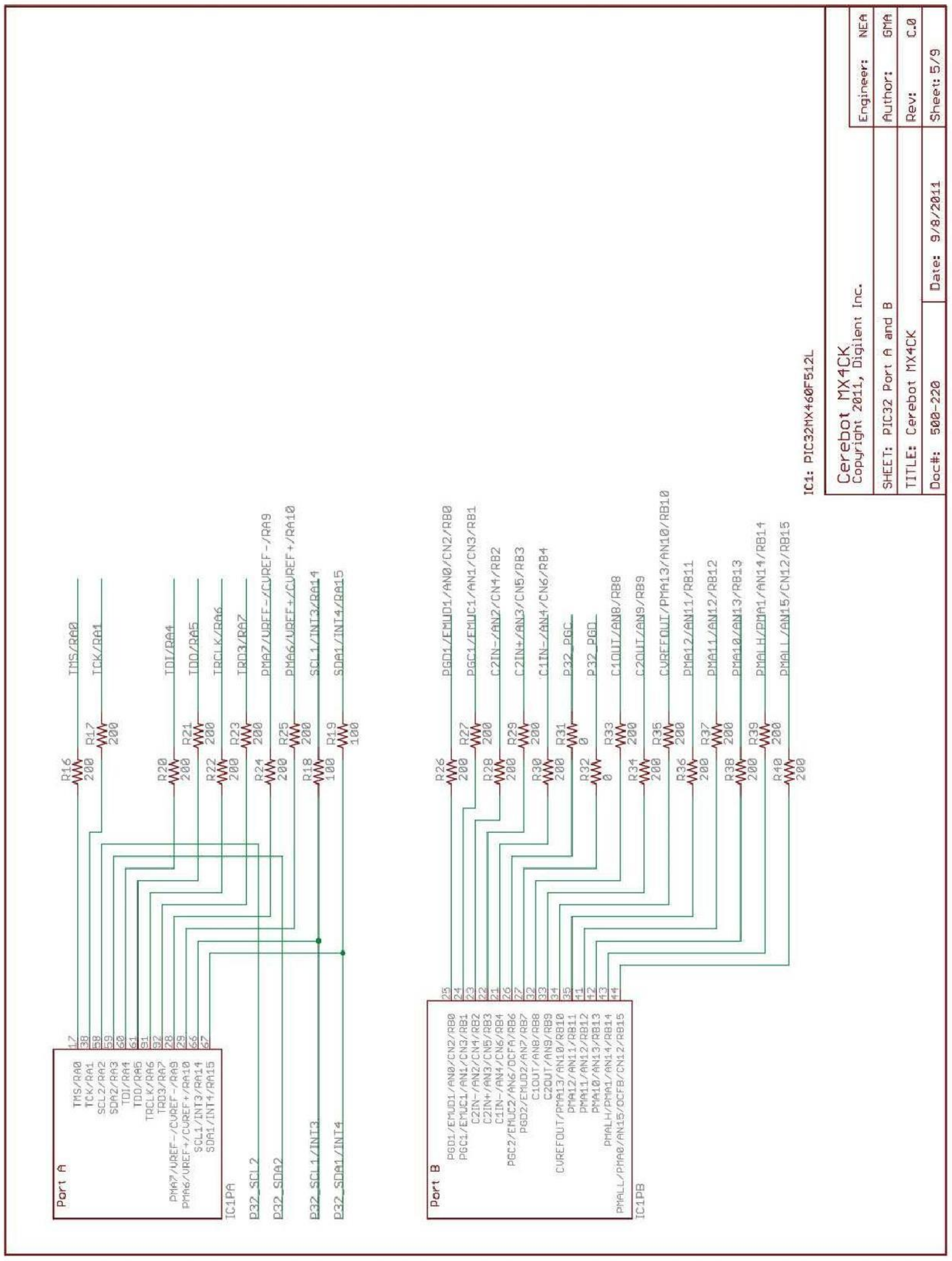
[2] http://www.sunearthtools.com/dp/tools/conversion.php

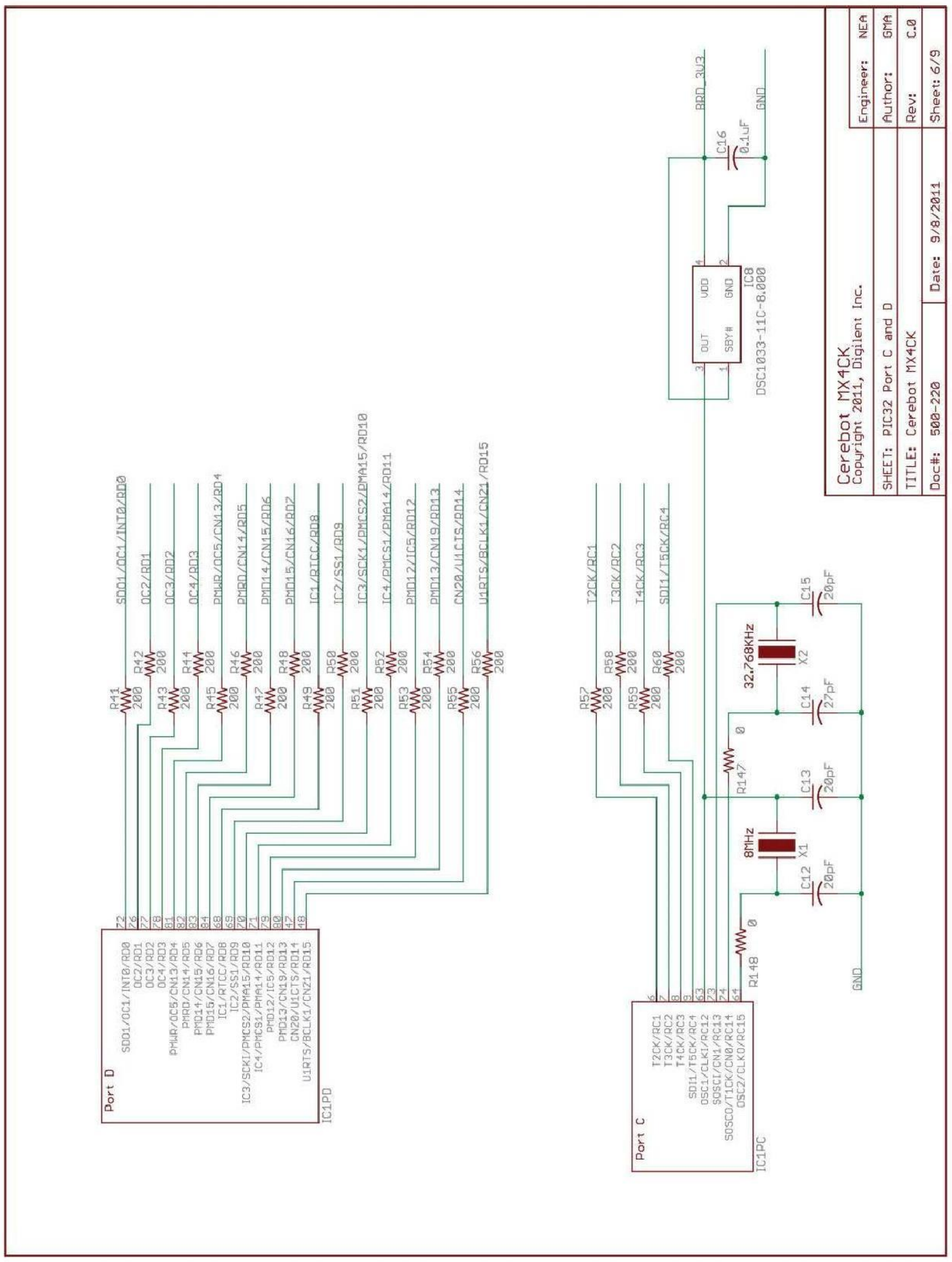

Pmod Connectors
PMD0/RF0
PMD1/RF1
PMD2/RF2
PMD3/RF3
PMD4/RF4
PMD5/RF5
PMD6/RF6
PMD7/RF7
PMA2/SS2/CN11/RG9
PMA3/SDO2/CN10/RG8
PMA4/SDI2/CN9/RG7
PMA5/SCK2/CN8/RG6
INT3/RB15
PMRD/CN14/RD5
PMWR/OC5/CN13/RD4
PMALH/PMA1/AN14/RB14
P32_SCL1/INT3
PMALL/AN15/CN12/RB15
R159
100
JP3
TRD1/RG12
TRD0/RG13
TRD2/RG14
RG15
PMD8/RG0
PMD9/RG1
PMD11/RF0
PMD10/RF1
PMD15/CN16/RD7
OC2/RD1
IC2/SS1/RD9
T2CK/RC1
PMD14/CN15/RD6
OC3/RD2
IC3/SCK1/PMCS2/PMA15/RD10
T3CK/RC2
D1
D2
D3
D4
D5
D6
D7
D8
BRD_VU
BRD_3V3
JPA
JPB
JPC
JPD
JA
JB
JC
JD
GND
I2C #1 Daisy Chain
P32_SCL1/INT3
P32_SDA1/INT4
BRD_3V3
R1
R2
2.2K
J3
J4
R104
R105
100
GND
J2
Digilent Inc.
Cerebot MX4CK
Copyright 2011, Digilent Inc.
SHEET: Pmod Connectors
TITLE: Cerebot MX4CK
Doc#: 500-220
Date: 9/8/2011
Engineer: NEA
Author: GMA
Rev: C.0
Sheet: 1/9
WEEE
ROHS CHINA
CE
chipKIT
CEREBOT MX4cK

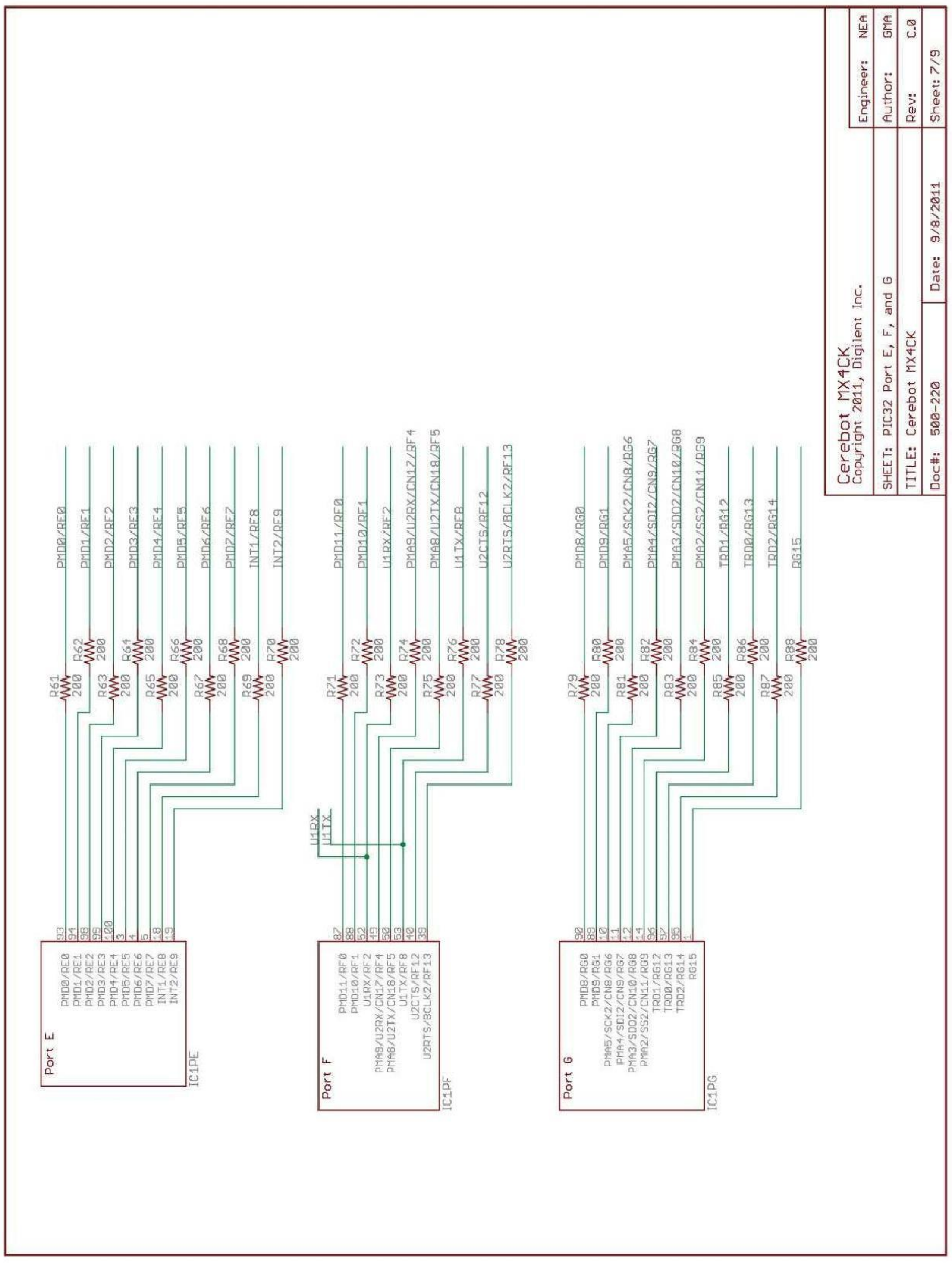

Pmod Connectors
JE
JF
JH
JJ
JK
JPE
JPF
JPH
JPJ
JPK
BRD_VU
BRD_3V3
GND
D9
D10
D11
D13
D14
D15
D16
D17
D18
D21
CN20/U1CTS/RD14
U1TX/RF8
U1RX/RF2
U1RTS/BCLK1/CN21/RD15
INT2/RF9
OC4/RD3
IC4/PMCS1/PMA14/RD11
T4CK/RC3
SCL1/INT3/RA14
SDA1/INT4/RA15
TRCLK/RA6
TRD3/RA7
TMS/RA0
TCK/RA1
TDI/RA4
TDO/RA5
U2CTS/RF12
PMA8/U2TX/CN18/RF5
PMA9/U2RX/CN17/RF4
U2RTS/BCLK2/RF13
INT1/RF8
SDO1/OC1/INT0/RD0
IC1/RTCC/RD8
PMD13/CN19/RD13
PGD1/EMUD1/AN0/CN2/RB0
PGC1/EMUC1/AN1/CN3/RB1
C2IN-/AN2/CN4/RB2
C2IN+/AN3/CN5/RB3
C1IN-/AN4/CN6/RB4
C1IN+/AN5/CN7/RB5
C1OUT/AN8/RB8
C2OUT/AN9/RB9
CVREFOUT/PMA13/AN10/RB10
PMA12/AN11/RB11
PMA11/AN12/RB12
PMA10/AN13/RB13
PMA7/VREF-/CVREF-/RA9
PMA6/VREF+/CVREF+/RA10
PMD12/IC5/RD12
SDI1/T5CK/RC4
POWER OUT
J19
No Load
Cerebot MX4CK
Copyright 2011, Digilent Inc.
SHEET: Pmod Connectors
TITLE: Cerebot MX4CK
Doc#: 500-220
Date: 9/8/2011
Engineer: NEA
Author: GMA
Rev: C.0
Sheet: 2/9

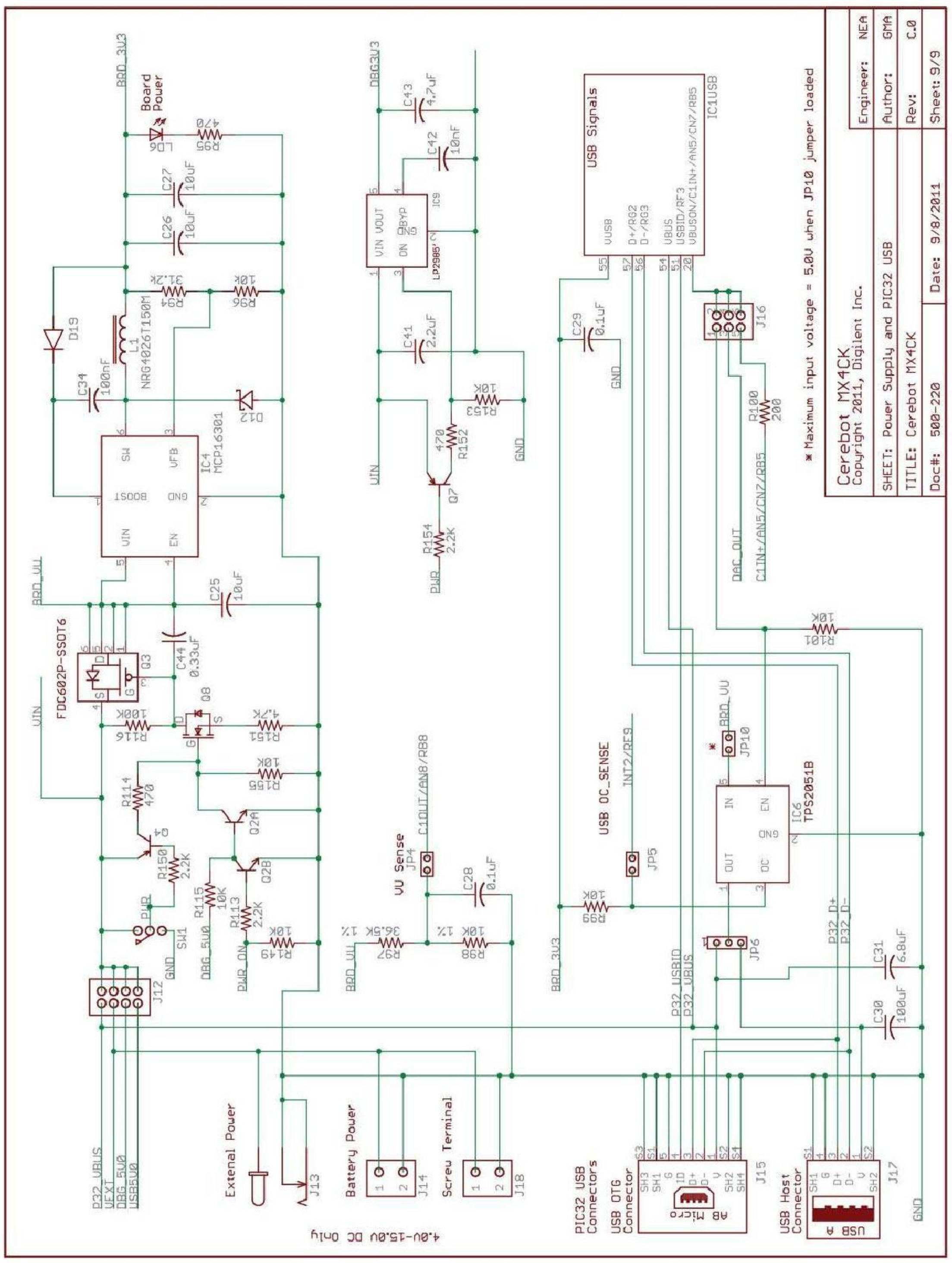

USB - UART Interface
IC10
FT232RQ-Tray
J7
J8
AB Micro
SPI Port #1 Connector
J1
Cerebot MX4CK
Copyright 2011, Digilent Inc.
SHEET: USB UART
TITLE: Cerebot MX4CK
Doc#: 500-220
Date: 9/8/2011
Engineer: NEA
Author: GMA
Rev: C.0
Sheet: 3/9


On-Board I2C Peripherals
I2C #2 Daisy Chain
PIC32 Microcontroller Power
PIC32 Power Supply Bypass
User Buttons
User LEDs
Servo Connectors
VS Sense
Cerebot MX4CK
Copyright 2011, Digilent Inc.
SHEET: On-board I/O & PIC32 Power
TITLE: Cerebot MX4CK
Doc#: 500-220
Date: 9/8/2011
Engineer: NEA
Author: GMA
Rev: C.0
Sheet: 4/9


Cerebot MX4CK
Copyright 2011, Digilent Inc.
SHEET: PIC32 Port A and B
TITLE: Cerebot MX4CK
Doc#: 500-220
Date: 9/8/2011
Engineer: NEA
Author: GMA
Rev: C.0
Sheet: 5/9
IC1: PIC32MX460F512L
Port A
IC1PA
Port B
IC1PB
P32_SCL2
P32_SDA2
P32_SCL1/INT3
P32_SDA1/INT4
TMS/RA0
TCK/RA1
TDI/RA4
TDO/RA5
TRCLK/RA6
TRD3/RA7
PMA7/VREF-/CVREF-/RA9
PMA6/VREF+/CVREF+/RA10
SCL1/INT3/RA14
SDA1/INT4/RA15
PGD1/EMUD1/AN0/CN2/RB0
PGC1/EMUC1/AN1/CN3/RB1
C2IN-/AN2/CN4/RB2
C2IN+/AN3/CN5/RB3
C1IN-/AN4/CN6/RB4
P32_PGC
P32_PGD
C1OUT/AN8/RB8
C2OUT/AN9/RB9
CVREFOUT/PMA13/AN10/RB10
PMA12/AN11/RB11
PMA11/AN12/RB12
PMA10/AN13/RB13
PMALH/PMA1/AN14/RB14
PMALL/AN15/CN12/RB15


Cerebot MX4CK
Copyright 2011, Digilent Inc.
SHEET: PIC32 Port C and D
TITLE: Cerebot MX4CK
Doc#: 500-220
Date: 9/8/2011
Engineer: NEA
Author: GMA
Rev: C.0
Sheet: 6/9
Port D
IC1PD
Port C
IC1PC
DSC1033-11C-8.000
IC8
C16 0.1uF
BRD_3V3
GND
8MHz X1
32.768KHz X2
C12 20pF
C13 20pF
C14 27pF
C15 20pF
R147 0
R148 0


Port E
IC1PE
PMD0/RE0
PMD1/RE1
PMD2/RE2
PMD3/RE3
PMD4/RE4
PMD5/RE5
PMD6/RE6
PMD7/RE7
INT1/RE8
INT2/RE9
Port F
IC1PF
PMD11/RF0
PMD10/RF1
U1RX/RF2
PMA9/U2RX/CN17/RF4
PMA8/U2TX/CN18/RF5
U1TX/RF8
U2CTS/RF12
U2RTS/BCLK2/RF13
U1RX
U1TX
Port G
IC1PG
PMD8/RG0
PMD9/RG1
PMA5/SCK2/CN8/RG6
PMA4/SDI2/CN9/RG7
PMA3/SDO2/CN10/RG8
PMA2/SS2/CN11/RG9
TRD1/RG12
TRD0/RG13
TRD2/RG14
RG15
R61 200
R62 200
R63 200
R64 200
R65 200
R66 200
R67 200
R68 200
R69 200
R70 200
R71 200
R72 200
R73 200
R74 200
R75 200
R76 200
R77 200
R78 200
R79 200
R80 200
R81 200
R82 200
R83 200
R84 200
R85 200
R86 200
R87 200
R88 200
Cerebot MX4CK
Copyright 2011, Digilent Inc.
SHEET: PIC32 Port E, F, and G
TITLE: Cerebot MX4CK
Doc#: 500-220
Date: 9/8/2011
Engineer: NEA
Author: GMA
Rev: C.0
Sheet: 7/9


Cerebot MX4CK
Copyright 2011, Digilent Inc.
SHEET: Power Supply and PIC32 USB
TITLE: Cerebot MX4CK
Doc#: 500-220
Date: 9/8/2011
Engineer: NEA
Author: GMA
Rev: C.0
Sheet: 9/9
* Maximum input voltage = 5.0V when JP10 jumper loaded
4.0V-15.0V DC Only
USB Signals
Extenal Power
Battery Power
Screw Terminal
PIC32 USB Connectors
USB OTG Connector
USB Host Connector
Board Power
VV Sense
USB OC_SENSE
FDC602P-SSOT6
MCP16301
NRG4026T150M
LP2985
TPS2051B

References

[1] Digilentinc. "ChipKIT Pro MX4 Board Reference Manua l". pp. 1-38, Retrieved from http://www.digilentinc.com/

[2] Microchip. "Section 12. I/O Ports". pp. 1-12, Retri eved from http://ww1.microchip.com/downloads/en/DeviceDoc/61120E.pdf

[3] Digilentinc. "PmodBT2™ Reference Manual". pp. 1-2, Retrieved from http://www.digilentinc.com/Data/Products/PMOD-BT2/PmodBT2_rm.pdf

[4] Digilentinc. "PmodKYPD™ Reference Manual". pp. 1-1, Retrieved from

http://www.digilentinc.com/Data/Products/PMODKYPD/PmodKYPD_rm.pdf

[5] Microchip. "Section 14. Timers". pp. 1-14, Retrieve d from http://ww1.microchip.com/downloads/en/DeviceDoc/61105F.pdf

[6] Microchip. "Section 8. Interrupts". pp. 1-8, Retrie ved from http://ww1.microchip.com/downloads/en/DeviceDoc/61108G.pdf

[7] ChipKIT^{TM} Standard Library: DSPI http://ChipKIT.net/started/learn-basics/ChipKIT-compatible-libraries/standard-library-dspi/