Chapter 2

Looking Under the Hood: Hardware

2.1 The PIC32

The Microchip PIC32 is a family of complex and powerful microcontrollers that can be purchased for less than $10 in quantities of one. This microcontroller offers many peripherals useful for mechatronics purposes, such as several channels for analog-to-digital conversion, digital I/O, synchronous and asynchronous serial communication, pulse-width modulated output, etc.

The "32" in PIC32 refers to the 32-bit wide data bus, meaning, for example, that an entire 32 bits can be accessed from RAM simultaneously. Primary advantages of 32-bit PICs over 8-bit PICs are that they have more peripherals available, offer more program memory (flash) and data memory (RAM), and have significantly more computational horsepower due to the 32-bit data bus, faster clock rates, and the fact that one instruction can be executed every cycle (including some 32-bit math operations).

Particular numbers referenced in this chapter refer to the PIC32MX795F512L chip, which is the PIC32 used on the NU32 board. (You may wish to compare the capabilities of this PIC to others on Microchip's PIC32 family website.)

This chapter is intended to provide an introduction to the PIC32 hardware. For more details, you should consult the PIC32MX5XX/6XX/7XX Family Data Sheet and the individual chapters of the PIC32 Family Reference Manual, which are available on Microchip's website and the class wiki.

2.1.1 Pin Functions and Special Function Registers (SFRs)

The PIC32MX795F512L features a max clock frequency of 80 MHz, 512K program memory (flash), and 128K data memory (RAM). It also features 16 10-bit analog-to-digital input lines (multiplexed to a single analog-to-digital converter, or ADC), many digital I/O channels, USB 2.0, Ethernet, two CAN modules, five I^2C and four SPI synchronous serial communication modules, six UARTs for RS-232 or RS-485 asynchronous serial communication, five 16-bit counter/timers (configurable to give two 32-bit timers and one 16-bit timer), five pulse-width modulation outputs, and a number of pins that can generate interrupts based on external signals, among other features.

To cram so much functionality into 100 pins, many of the pins serve multiple functions. See the PIC32's pinout diagram (Figure 2.1). As an example, pin 21 can serve as an analog input, a comparator input, a change notification input (which can generate an interrupt when an input changes state), or a digital input or output.

Table 2.1 briefly summarizes some of the pin functions. The most important functions for embedded control are indicated in **bold**.

Which function a particular pin actually serves is determined by *Special Function Registers* (SFRs). Each SFR is a 32-bit word that sits at some memory address. The values of the SFR bits, 0 or 1, control the functions of the pins as well as other functions of the PIC32.

Pin 78 in Figure 2.1 can serve as OC4 (output compare 4) or RD3 (digital I/O number 3 on port D). Let's say we want to use it as a digital output. We can set the SFRs that control this pin to disable the OC4 function and to choose the RD3 function as digital output instead of digital input. Looking at the

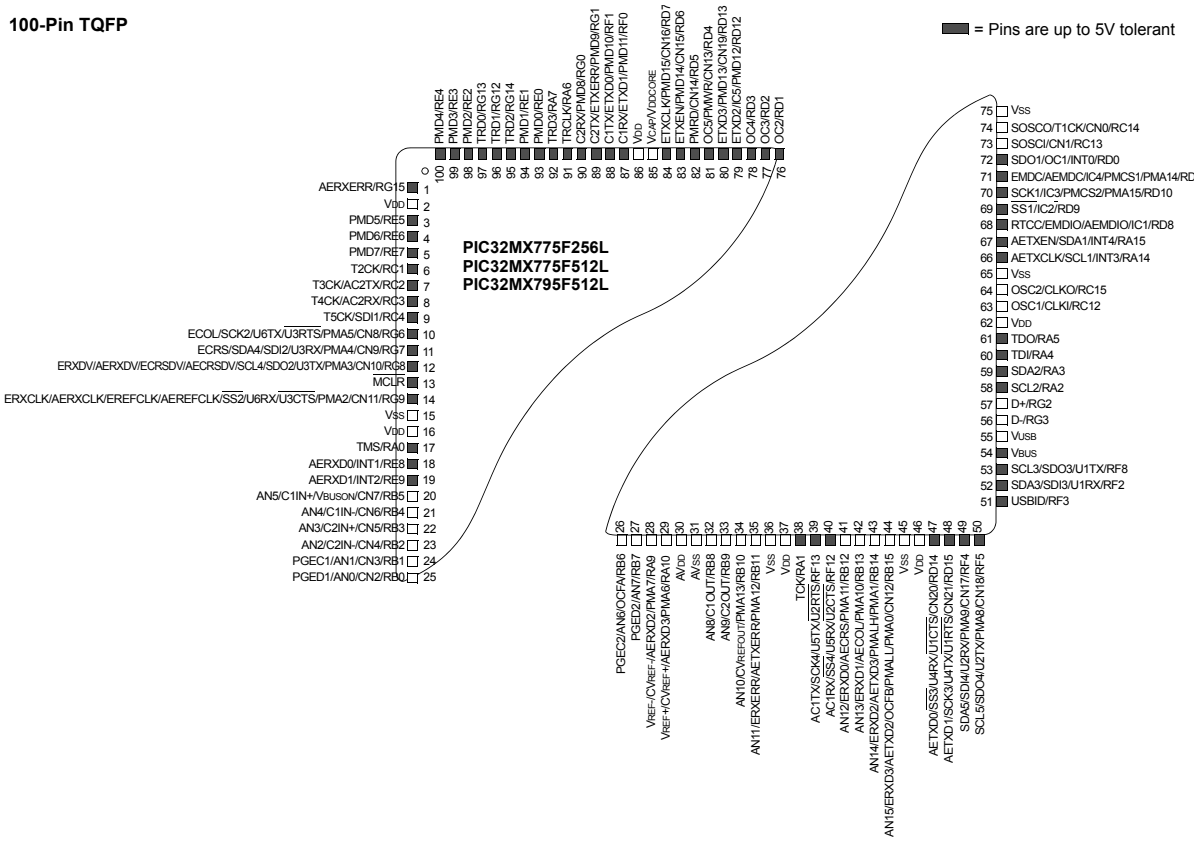


Figure 2.1: The PIC32MX5XX/6XX/7XX pinout. Shaded pins tolerate up to 5.5 V as inputs.

PIC32 Family Reference Manual section on Output Compare, we see that the 32-bit SFR named "OC4CON" determines whether OC4 is enabled or not. Specifically, for bits numbered 0 ... 31, we see that bit 15 is responsible for enabling or disabling OC4. We refer to this bit as OC4CON⟨15⟩. If it is a 0, OC4 is disabled, and if it is a 1, OC4 is enabled. So we "clear" this bit to 0. (Bits can be cleared to 0, or cleared for short, or "set" to 1, or set for short.) Now, referring to the I/O Ports section of the Reference Manual, we see that bit 3 of the SFR TRISD, i.e., TRISD⟨3⟩, should be cleared to 0 to make pin 78 a digital output.

In fact, OC4CON⟨15⟩ is cleared to zero by default on reset of the PIC32, so this step was not necessary. On the other hand, TRISD⟨3⟩ is set to 1 on reset, making pin 78 a digital input by default. (This is for safety, to make sure the PIC32 does not impose an unwanted voltage on anything it is connected to on startup.)

We will see SFRs again and again as we learn how to work with the PIC32.

2.1.2 PIC32 Architecture

Figure 2.2 is a block diagram of the architecture of the PIC32. Of course there is a CPU, program memory, and data memory. Perhaps most interesting to us, though, is the plethora of *peripherals*, which are what make microcontrollers useful for embedded control. From left to right, top to bottom, these peripherals consist of PORTA ... PORTG, which are digital I/O ports; 22 change notification pins to generate interrupts when input signals change; five 16-bit counters (which can be used as one 16-bit counter and two 32-bit counters by chaining) that can be used for a variety of counting operations, and timing operations by counting clock ticks; five pins for output pulse-width modulation pulse trains (or "output compare"); five pins for "input capture," which are used to capture timer values or trigger interrupts on rising or falling inputs; four SPI and five I^2C synchronous serial communication modules; a "parallel master port" (PMP) for parallel communication;

Pin Label	Function
ANx (x=0-15)	analog-to-digital (ADC) inputs
AVDD, AVSS	positive supply and ground reference for ADC
BCLK1, BLCK2	clocks for infrared (IrDA) comm encoding and decoding for 2 UARTs
CxIN-, CxIN+, CxOUT (x=1,2)	comparator negative and positive input and output
CLKI, CLKO	clock input and output (for particular clock modes)
CNx (x=0-21)	change notification; voltage changes on these pins can generate interrupts
CVREF-, CVREF+, CVREFOUT	comparator reference voltage low and high inputs, output
D+, D-	USB communication lines
EMUCx, EMUDx (x=1,2)	used by an in-circuit emulator (ICE)
ENVREG	enable for on-chip voltage regulator that provides 1.8 V to internal core (set to VDD to enable the regulator on the NU32 board)
ICx (x=1-5)	input capture pins for measuring frequencies and pulse widths
INTx (x=0-4)	voltage changes on these pins can generate interrupts
MCLR (overbar)	master clear reset pin, resets PIC when low
OCx (x=1-5)	output compare pins, usually used to generate pulse trains (pulse-width modulation) or individual pulses
OCFA, OCFB	fault protection for output compare pins; if a fault occurs, they can be used to make OC outputs be high impedance (neither high nor low)
OSC1, OSC2	crystal or resonator connections for different clock modes
PGCx, PGDx (x=1,2)	used with in-circuit debugger (ICD)
PMALL, PMALH	latch enable for parallel master port
PMAx (x=0-15)	parallel master port address
PMDx (x=0-15)	parallel master port data
PMENB, PMRD, PMWR	enable and read/write strobes for parallel master port
Rxy (x=A-G,y=0-15)	digital I/O pins
RTCC	real-time clock alarm output
SCLx, SDAx (x=1-5)	I^2C serial clock and data input/output for I^2C synchronous serial communication modules
SCKx, SDIx, SDOx (x=1-4)	serial clock, serial data in, out for SPI synchronous serial communication modules
$\overline{\mathbf{SS1}}, \overline{\mathbf{SS2}}$	slave select (active low) for SPI communication
TxCK (x=1-5)	input pins for counters when counting external pulses
TCK, TDI, TDO, TMS	used for JTAG debugging
TRCLK, TRDx (x=0-3)	used for instruction trace controller
UxCTS, UxRTS, UxRX, UxTX (x=1-6)	UART clear to send, request to send, receive input, and transmit output for UART modules
VDD	positive voltage supply for peripheral digital logic and I/O pins (3.3 V on NU32)
VDDCAP	capacitor filter for internal 1.8 V regulator when ENVREG enabled
VDDCORE	external 1.8 V supply when ENVREG disabled
VREF-, VREF+	can be used as negative and positive limit for ADC
VSS	ground for logic and I/O
VBUS	monitors USB bus power
VUSB	power for USB transceiver
VBUSON	output to control supply for VBUS
USBID	USB on-the-go (OTG) detect

Table 2.1: Some of the pin functions on the PIC32. Commonly used functions for embedded control are in **bold**. See Section 1 of the Data Sheet for more information.

an analog-to-digital converter (ADC) multiplexed to 16 input pins; six UARTs for asynchronous serial communication (e.g., RS-232, RS-485); a real-time clock and calendar (RTCC) that can maintain accurate year-month-day-time; and two comparators that compare which of two analog inputs has a higher voltage.

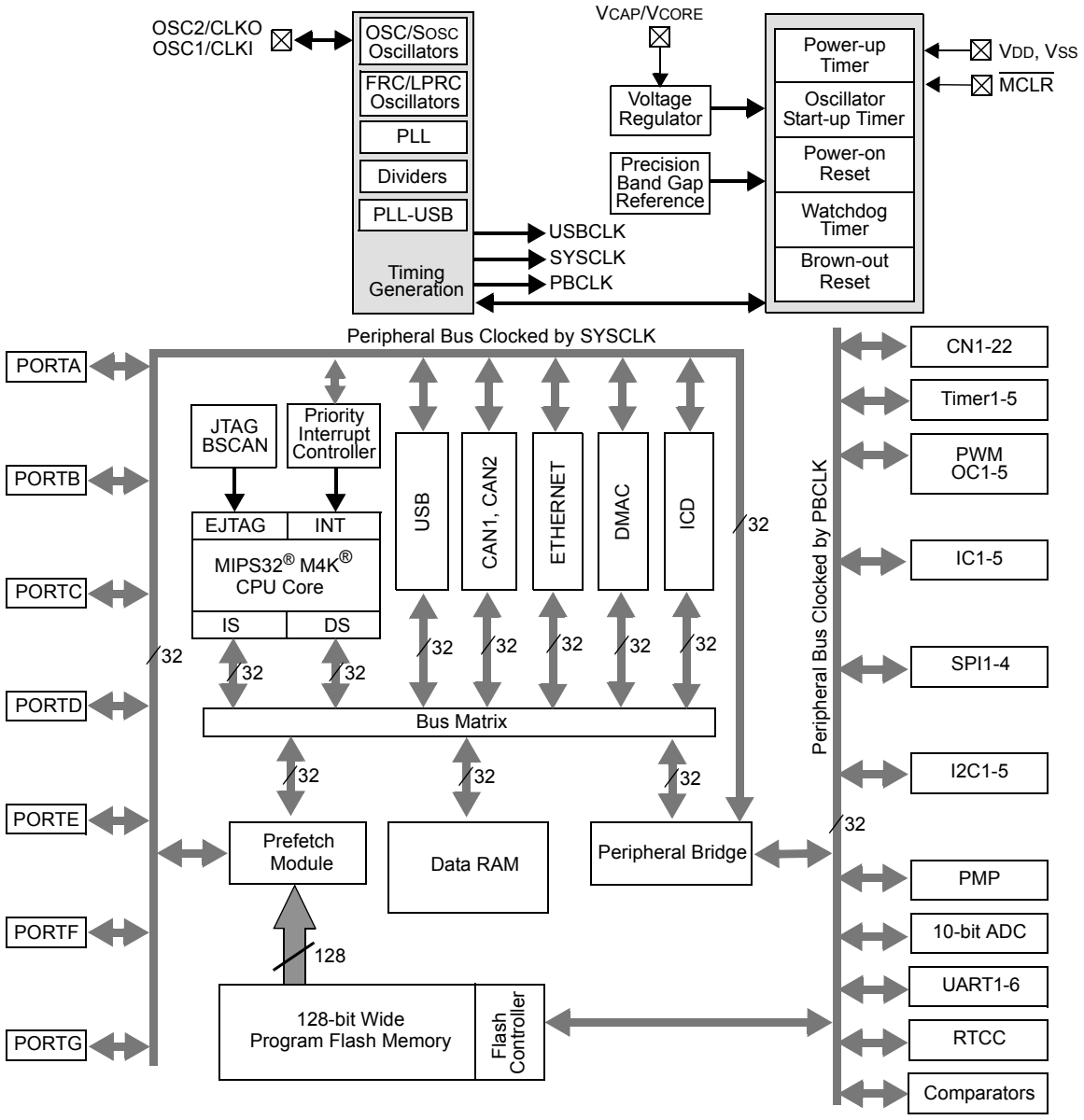


Figure 2.2: The PIC32MX5XX/6XX/7XX architecture.

Note that the peripherals are on two different buses: one is clocked by the system clock SYSCLK, and the other is clocked by the peripheral bus clock PBCLK. A third clock, USBCLK, is used for USB communication. The timing generation block that creates these clock signals and other elements of the architecture in Figure 2.2 are briefly described below.

CPU The central processing unit runs the whole show. It fetches program instructions over its "instruction side" (IS) bus, reads in data over its "data side" (DS) bus, executes the instructions, and writes out the results over the DS bus. The CPU can be clocked by SYSCLK at up to 80 MHz, meaning it can execute one instruction every 12.5 nanoseconds. The CPU is capable of multiplying a 32-bit integer by a 16-bit integer in one cycle, or a 32-bit integer by a 32-bit integer in two cycles. There is no floating point unit (FPU), so floating point math is carried out in a series of steps in software, meaning floating point operations are much

slower than integer math.

The CPU also communicates with the interrupt controller, described below.

The CPU is based on the MIPS32® M4K® microprocessor core licensed from MIPS® Technologies. The CPU operates at 1.8 V (provided by a voltage regulator internal to the PIC32, as it's used on the NU32 board).

Bus Matrix The CPU communicates with other units through the 32-bit bus matrix. Depending on the memory address specified by the CPU, the CPU can communicate with program memory (flash), data memory (RAM), or peripherals. The memory map is discussed in Section 2.1.3.

Interrupt Controller The job of the interrupt controller is to present "interrupt requests" to the CPU. An interrupt request may be generated by variety of sources, such as a changing input on a change notification pin or by the elapsing of a specified time on one of the timers. If the CPU accepts the request, it will suspend whatever it is doing and jump to an *interrupt service routine* (ISR), a function defined in the program. After completing the ISR, program control returns to where it was suspended. Interrupts are an extremely important concept in embedded control.

Memory: Program Flash and Data RAM The PIC32 has two types of memory: flash and RAM. Flash is generally more plentiful on PIC32's (e.g., 512K flash vs. 128K RAM on the PIC32MX795F512L), nonvolatile (meaning that its contents are preserved when powered off, unlike RAM), but slower to read and write than RAM. Your program is stored in flash memory and your temporary data is stored in RAM. When you power cycle your PIC32, your program is still there but your data in RAM is lost.[1]

Because flash is slow, reading a program instruction from flash may take three CPU cycles when operating at 80 MHz (see Electrical Characteristics in the Data Sheet). The job of the prefetch module (below) is to minimize or eliminate the need to wait around for program instructions to load.

Prefetch Module The job of the *prefetch (cache) module* is to fetch program instructions from flash program memory in advance of the CPU's need for them. In many cases, the prefetch module can provide one instruction per cycle, hiding the delays due to slow flash access. The module can *cache* all instructions for small program loops, so that flash memory does not have to be accessed while executing the loop, or *prefetch* instructions when executing linear code. (Note the 128-bit wide data path between the prefetch module and flash, allowing the prefetch module to run ahead of execution despite the slow load times from flash.) When the CPU asks for an instruction, the prefetch module first checks if the instruction is in its cache. If so, it provides it to the CPU immediately. If not, the (slower) load from flash memory begins.

The cache can also store constant data.

Clocks and Timing Generation There are three clocks on the PIC32: SYSCLK, PBCLK, and USBCLK. USBCLK is a 48 MHz clock used for USB communication. SYSCLK clocks the CPU at a maximum frequency of 80 MHz, adjustable all the way down to 0 Hz. Higher frequency means more calculations per second but higher power usage. PBCLK is used by a number of the peripherals, and its frequency is set to SYSCLK's frequency divided by 1, 2, 4, or 8. We might want to set PBCLK's frequency lower than SYSCLK's if we want to save power on the peripheral bus. If PBCLK's frequency is less than SYSCLK's, then programs with back-to-back peripheral operations will cause the CPU to wait cycles before issuing the second peripheral command to ensure that the first one has completed.

All clocks are derived either from an oscillator internal to the PIC32 or an external resonator or oscillator provided by the user. High-speed operation requires an external circuit, so the NU32 provides an external 8 MHz resonator as a clock source. The NU32 software sets the PIC32's configuration bits (see Section 2.1.4) to use a phase-locked loop (PLL) on the PIC32 to multiply this frequency by a factor of 10, generating a SYSCLK of 80 MHz. The PBCLK is set to the same frequency. The USBCLK is also derived from the 8 MHz resonator by a PLL multiplying the frequency by 6.

[1] It is also possible to store program instructions in RAM, and to store data in flash, but we set that aside for now.

Digital Input and Output A digital I/O pin configured as an input can be used to detect whether the input voltage is low or high. On the NU32, the PIC32 is powered by 3.3 V, so voltages close to 0 V are considered low and those close to 3.3 V are considered high. Some input pins can tolerate up to 5.5 V, while voltages over 3.3 V on other pins could damage the PIC32 (see Figure 2.1).

A digital I/O pin configured as an output can produce a voltage of 0 or 3.3 V. An output pin can also be configured as *open drain*. In this configuration, the pin is connected by an external pull-up resistor to a voltage of up to 5.5 V. This allows the pin's output transistor to either sink current to pull the voltage down to 0 V or allow it to be pulled up as high as 5.5 V. This increases the range of output voltages the pin can produce.

Counter/Timers The PIC32 has five 16-bit counters. Each can count from 0 up to $2^{16} - 1$, or any preset value less than $2^{16} - 1$ that we choose, before rolling over. Counters can be configured to count external events, such as pulses on a TxCK pin, or internal events, like clock cycles. In the latter case, we refer to the counter as a *timer*. The counter can be configured to generate an interrupt when it rolls over. This allows the execution of an ISR on exact timing intervals.

Two 16-bit counters can be configured to make a single 32-bit counter. This can be done with two different pairs of counters, giving one 16-bit counter and two 32-bit counters.

Analog Input The PIC32 has a single analog-to-digital converter (ADC), but 16 different pins can be connected to it, one at a time. This allows up to 16 analog voltage values (typically sensor inputs) to be monitored. The ADC can be programmed to continuously read in data from a sequence of input pins, or to read in a single value when requested. Input voltages must be between 0 and 3.3 V. The ADC has 10 bits of resolution, allowing it to distinguish 1024 different voltage levels.

Output Compare Output compare pins are used to generate a single pulse of specified duration, or a continuous pulse train of specified duty cycle and frequency. They work with timers to generate the precise timing. A common use of output compare pins is to generate PWM (pulse-width modulated) signals as control signals for motors. Pulse trains can also be low-pass filtered to generate approximate analog outputs. (There are no analog outputs on the PIC32.)

Input Capture A changing input on an input capture pin causes the PIC32 to store the current time measured by a timer. This allows precise measurements of input pulse widths and signal frequencies. Optionally, the input capture pin can generate an interrupt.

Change Notification A change notification pin can be used to generate an interrupt when the input voltage changes from low to high or vice-versa.

Comparators A comparator is used to compare which of two analog input voltages is larger. A comparator can generate an interrupt when one of the inputs exceeds the other.

Real-Time Clock and Calendar The RTCC module is used to maintain accurate time, day, month, and year over extended periods of time while using little power and requiring no attention from the CPU.

Parallel Master Port The PMP module is used to read data from and write data to external parallel devices with 8-bit and 16-bit data buses.

DMA Controller The Direct Memory Access controller is useful for data transfer with external devices without involving the CPU. For example, DMA can allow an external device to dump large amounts of data directly into PIC32 RAM.

SPI Serial Communication The Serial Peripheral Interface bus provides a simple method for serial communication between a master device (typically a microcontroller) and one or more slave devices. Each slave device has four communication pins: a Clock (set by the master), Data In (from the master), Data Out (to the master), and Select. The slave is selected for communication if the master holds its Select pin low. The master device controls the Clock, has a Data In and a Data Out line, and one Select line for each slave it can talk to. Communication rates can be up to tens of megabits per second.

I^2C Serial Communication The Inter-Integrated Circuit protocol I^2C (pronounced "I squared C") is a somewhat more complicated serial communication standard that allows several devices to communicate over only two shared lines. Any of the devices can be the master at any time. The maximum data rate is less than for SPI (e.g., 400 kilobits per second).

UART Serial Communication The Universal Asynchronous Receiver Transmitter module provides another method for serial communication between two devices. There is no clock line, hence "asynchronous." Each of the two devices has a Receive Data line and a Transmit Data line, and typically a Request to Send line (to ask for permission to send data) and a Clear to Send line (to indicate that the device is ready to receive data). There are different protocols for using these same lines (e.g., RS-232 and RS-485). Typical data rates are 9600 bits per second (9600 baud) up to hundreds of thousands of bits per second, but higher rates are possible.

USB The Universal Serial Bus is a popular asynchronous communication protocol. One master communicates with one or more slaves over a four-line bus: +5 V, ground, D+ and D- (differential data signals). The PIC32 implements USB 2.0 full-speed and low-speed options, and can communicate at theoretical data rates of up to several megabits per second.

CAN Controller Area Networks are heavily used in electrically noisy environments (particularly industrial and automotive environments) to allow many devices to communicate over a single two-wire bus. Data rates of up to 1 megabit per second are possible.

Ethernet The ethernet module uses an external PHY chip (physical layer protocol transceiver chip) and direct memory access (DMA) to offload from the CPU the heavy processing requirements of ethernet communication. The NU32 board does not include a PHY chip.

Watchdog Timer If the Watchdog Timer is used by your program, your program must periodically reset the timer counter. Otherwise the PIC will reset. This is a way to have the PIC restart if your program has entered an unexpected state where it doesn't pet the watchdog.

2.1.3 The Memory Map

The CPU accesses the peripherals, data, and program instructions in the same way: by specifying a memory address. The PIC32's memory addresses are 32-bits long, and each address refers to a byte in the *memory map*. This means that the memory map of the PIC32 consists of 4 GB (four gigabytes, or 2^{32} bytes). Of course most of these addresses are meaningless; there is not nearly that much flash or RAM to address.

The PIC32's memory map consists of four main components: RAM, flash, peripheral SFRs that we write to (to control the peripherals) or read from (to get sensor input, for example), and *boot flash*. Of these, we have not yet seen "boot flash." This is an extra 12 kilobytes (KB) of flash that contains program instructions that are executed immediately upon reset of the PIC32.[2] The boot flash instructions execute any time-critical PIC32 setup and then calls the program installed in program flash.

The following table illustrates the PIC32's *physical* memory map. It consists of a block of "RAMsize" bytes of RAM (128 KB for us), "flashsize" bytes of flash (512 KB for us), 1 MB for the peripheral SFRs, and 12 KB for the boot flash:

[2]The last four 32-bit words of the boot flash memory region are Device Configuration Registers. See Section 2.1.4.

Physical Memory Start Address	**Size (bytes)**	**Memory Type**
0x00000000	RAMsize (128 KB)	Data RAM
0x1D000000	flashsize (512 KB)	Program Flash
0x1F800000	1 MB	Peripheral SFRs
0x1FC00000	12 KB	Boot Flash

To make things a little more complicated, we have the option of allocating some of the RAM to hold program instructions. We can then run a program out of RAM. In this case, we have to specify how many bytes of RAM are for data and how many are for the program. Let's call these dataRAMsize and progRAMsize, where dataRAMsize + progrRAMsize = RAMsize. Then our table becomes

Physical Memory Start Address	**Size (bytes)**	**Memory Type**
0x00000000	dataRAMsize	Data RAM
0x00000000 + dataRAMsize	progRAMsize	Program RAM
0x1D000000	flashsize (512 KB)	Program Flash
0x1F800000	1 MB	Peripheral SFRs
0x1FC00000	12 KB	Boot Flash

Since we rarely allocate RAM for a program (progRAMsize = 0), let's leave that row unshaded, as less important than the other rows.

The PIC32 also gives us the option to *partition* the flash and RAM into a *kernel segment* KSEG and a *user segment* USEG. By analogy to your personal computer, the kernel address space KSEG holds the computer's operating system, while the user address space USEG holds the program that runs under the operating system. This partition is for safety: the user's program should not interfere with or compromise the operating system; it shouldn't be able to overwrite data that the operating system needs to function. USEG does not include the peripheral SFRs and boot flash, since a poorly written user program could do a lot of damage with access to these.

Since we want all our programs to have full access to all peripherals, we won't partition, and the entire memory map will be treated as the kernel segment.

Finally, you should be aware that the contents of most memory addresses can be cached by the prefetch module (see the earlier discussion of the prefetch module). On the other hand, some instructions, data, and no peripheral SFRs can be cached. For example, if port B is configured as a digital input port, the peripheral SFR PORTB contains the current input values. When your program asks for these values, we need the current values; we cannot pull them from the prefetch cache.

To represent the fact that different memory addresses have different cacheability, we conceptually partition into two kernel segments, KSEG0 and KSEG1. Memory addresses in KSEG0 have contents that can be cached, while memory addresses in KSEG1 have contents that cannot be cached. As a concrete example, say we have a program instruction sitting at physical address 0x1D000000. If it can be cached, we can specify a *virtual* memory address for this instruction by simply adding 0x80000000, resulting in a virtual address of 0x9D000000. If the instruction cannot be cached, we assign a virtual memory address by adding 0xA0000000, resulting in a virtual address of 0xBD000000. Therefore KSEG0 virtual addresses start at 0x80000000 while the KSEG1 virtual addresses start at 0xA0000000. Of course the physical addresses for KSEG0 and KSEG1 are the same. But since we don't need all 32 bits of the address to access our peripherals, flash, and RAM, we are free to use the first three bits of the address to represent other information—in this case, whether the contents of the address are cacheable. These bits are essentially a flag created in the compilation process indicating that the address contents may or may not be put into the cache.

Table 2.2 gives our final memory map, with the relationship between the physical memory addresses and the virtual memory addresses. The CPU and compiler (and any program you write or memory map you see) deal only with virtual memory addresses, and a simple *fixed mapping translation* (FMT) unit on the PIC32 provides the corresponding physical address by performing a bitwise AND:

Physical Address = Virtual Address & 0x1FFFFFFF

Physical Memory Start Address	Virtual Memory Start Address	Size (bytes)	Memory Type
0x00000000	KSEG0 cacheable: 0x80000000 KSEG1 non-cache: 0xA0000000	dataRAMsize	Kernel Data RAM
0x00000000 + dataRAMsize	KSEG0 cacheable: 0x80000000 + dataRAMsize KSEG1 non-cache: 0xA0000000 + dataRAMsize	progRAMsize	Kernel Program RAM
0x1D000000	KSEG0 cacheable: 0x9D000000 KSEG1 non-cache: 0xBD000000	flashsize	Kernel Program Flash
0x1F800000	KSEG1 non-cache: 0xBF800000	1 MB	Peripheral SFRs
0x1FC00000	KSEG0 cacheable: 0x9FC00000 KSEG1 non-cache: 0xBFC00000	12 KB	Boot Flash

Table 2.2: PIC32 physical and virtual memory map.

On the PIC32, the partitioning of the virtual and physical memory addresses is determined by values set in bus matrix SFRs with crazy names like BMXDKPBA ("bus matrix data RAM kernel program base address"). To get the memory map shown in Table 2.2, we would set BMXDKPBA equal to dataRAMsize, which must be a multiple of 2 KB but less than BMXDRMSZ, a read-only register indicating the total RAM on the PIC32. Setting BMXDKPBA tells the bus matrix that the kernel program RAM begins at that address relative to the first RAM address (at 0x00000000 in physical memory). We would then set BMXDUDBA and BMXDUPBA to be equal to BMXDRMSZ. The "U" in the first two SFRs indicate they relate to the user partition, and by setting the start address of the user data and program RAM after the end of the RAM section, we have allocated no user RAM. More details on virtual and physical addresses and partitioning can be found in the Memory Organization section of the Reference Manual.

2.1.4 Configuration Bits

The last four 32-bit words of the boot flash are the Device Configuration Registers, DEVCFG0 to DEVCFG3, containing the *configuration bits*. These configuration bits set a number of important properties of how your PIC32 will function. You can learn more about configuration bits in Section 28 Special Features of the Data Sheet. For example, DEVCFG1 contains configuration bits that determine the frequency ratio between SYSCLK and PBCLK.

2.2 The NU32 Board

The NU32 development board is shown in Figure 2.3, and the layout and pinout are shown in Figure 2.4. The NU32 board was designed to plug into two standard prototyping breadboards, allowing easy prototyping with the PIC32. The NU32 board breaks out 82 pins of the PIC32MX795F512L and provides an 8 MHz resonator to create its clock signal. It also has a mini USB connector for communication with your computer, a USB to RS232 chip that allows your PIC32 to speak RS232 to your computer's USB port, a DC input jack for 6 V input, a 5 V and 3.3 V regulator, two buttons (labeled USER and RESET), two LEDs, and a power switch. The NU32 loads four rails of the breadboards with GND, 3.3 V, 5 V, and the 6 V input, so that these are available for use on your prototyping board. The regulated 3.3 V is also used to power the PIC32.

The DC power supply that comes with the NU32 kit provides up to 1 amp at 6 V. Take this into account when you decide how much current to try to draw from the NU32 board. Even if the NU32 board is powered by a higher-current supply, such as a battery, keep in mind that the onboard 5V and 3.3V regulator can only source approximately 800 mA. You should not try to draw more than, say, half that. Usual rule of thumb: don't try to drive motors, which often draw hundreds of mA up to a few amps, using current flowing through your NU32 board.

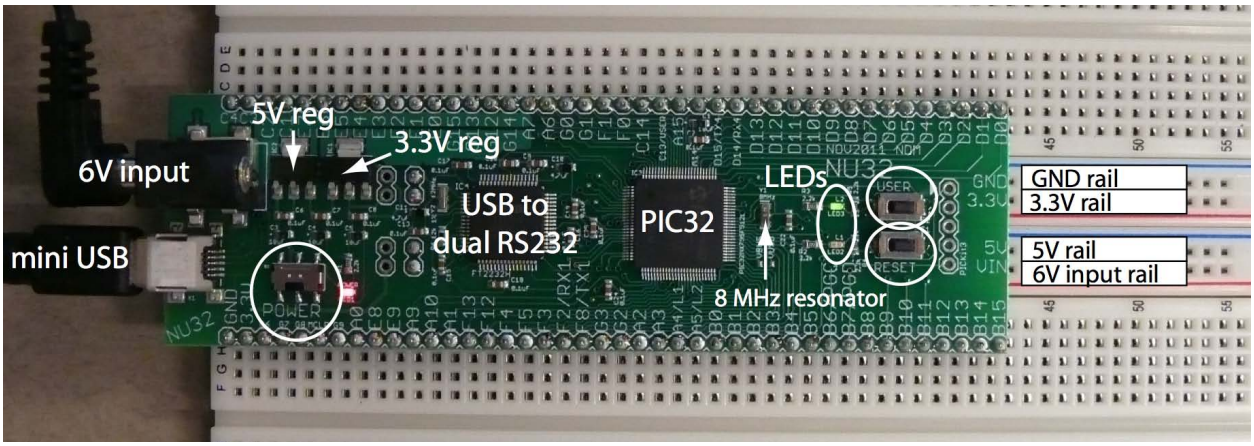
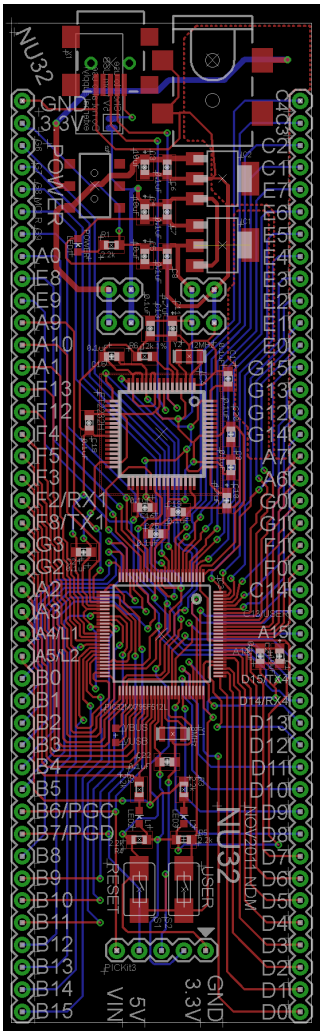


Figure 2.3: The NU32 development board.



Function	Name	Name	Function
GND output from NU32	GND	C4	T5CK/SDI1/C4
3.3V output from NU32	3.3V	C3	T4CK/C3
SCK2/U6TX/U3RTS/CN8/G6	G6	C2	T3CK/C2
SDA4/SDI2/U3RX/CN9/G7	G7	C1	T2CK/C1
SCL4/SDO2/U3TX/CN10/G8	G8	E7	PMD7/E7
MCLR	MCLR	E6	PMD6/E6
SS2/U6RX/U3CTS/CN11/G9	G9	E5	PMD5/E5
A0	A0	E4	PMD4/E4
INT1/E8	E8	E3	PMD3/E3
INT2/E9	E9	E2	PMD2/E2
VREF-/CVREF-/A9	A9	E1	PMD1/E1
VREF+/CVREF+/A10	A10	E0	PMD0/E0
A1	A1	G15	G15
SCK4/U5TX/U2RTS/F13	F13	G13	G13
SS4/U5RX/U2CTS/F12	F12	G12	G12
SDA4/SDI4/U2RX/CN17/F4	F4	G14	G14
SCL5/SDO4/U2TX/CN18/F5	F5	A7	A7
USBID/F3	F3	A6	A6
SDA3/SDI3/U1RX/F2	F2/RX1	G0	C2RX/PMD8/G0
SCL3/SDO3/U1TX/F8	F8/TX1	G1	C2TX/PMD9/G1
D-/G3	G3	F1	C1TX/PMD10/F1
D+/G2	G2	F0	C1RX/PMD11/F0
SCL2/A2	A2	C14	T1CK/CN0/C14
SDA2/A3	A3	C13/USER	CN1/C13
A4	A4/L1	A15	SDA1/INT3/A15
A5	A5/L2	A14	SCL1/INT3/A14
PGED1/AN0/CN2/B0	B0	D15/TX4	SCK3/U4TX/U1RTS/CN21/D15
PGEC1/AN1/CN3/B1	B1	D14/RX4	SS3/U4RX/U1CTS/CN20/D14
AN2/C2IN-/CN4/B2	B2	D13	PMD13.CN19/D13
AN3/C2IN+/CN5/B3	B3	D12	IC5/PMD12/D12
AN4/C1IN-/CN6/B4	B4	D11	IC4/D11
AN5/C1IN+/CN7/B5	B5	D10	SCK1/IC3/D10
PGEC2/AN6/OCFA/B6	B6/PGC	D9	SS1/IC2/D9
PGED2/AN7/B7	B7/PDG	D8	RTCC/IC1/D8
AN8/C1OUT/B8	B8	D7	PMD15/CN16/D7
AN9/C2OUT/B9	B9	D6	PMD14/CN15/D6
AN10/CVREFOUT/B10	B10	D5	CN15/D5
AN11/B11	B11	D4	OC5/CN13/D4
AN12/B12	B12	D3	OC4/D3
AN13/B13	B13	D2	OC3/D2
AN14/B14	B14	D1	OC2/D1
AN15/OCFB/CN12/B15	B15	D0	SDO1/OC1/INT0/D0

5V tolerant pin

Figure 2.4: Pinout of the NU32 development board.

2.3 Problems

You will need to refer to the PIC32 Data Sheet and PIC32 Reference Manual, linked under "Required Reading" from the wiki, to answer some questions.

1. The NUScope is a PIC32-based personal oscilloscope, capable of reading two analog signals and four digital signals. It also provides +5 V and −5 V, 5 V digital outputs, and for 2011-2012, a single analog output channel. Software and instructions for how to use the NUScope are located at

   `http://hades.mech.northwestern.edu/index.php/NUScope`

   If you have an NUScope from ME233 2010, ME233 2011, or BME305 2011, then you are ready to go. If you purchased an NUScope along with your NU32 for ME333 2012, the NUScope does not have the firmware that makes it work with the nuScope2-2 computer program. Follow the instructions on the wiki page to use the bootloader to put the correct firmware onto your NUScope.

   (a) Use the 10k potentiometer from your kit to build a variable voltage source from -5V to +5V. Read it into channel A of the NUScope. Turn the potentiometer to make a sinusoidal signal from -3V to +3V, take a screenshot, and include it in your homework writeup.

   (b) Wire a green LED in series with a 330 Ω resistor to digital output O5 (2010 board) or DO (2011 board). Use the software controls to turn the LED on and off, and demonstrate in class on Thursday for credit.

2. Follow the instructions at

   `http://hades.mech.northwestern.edu/index.php/NU32:_Software_to_Install`

   and

   `http://hades.mech.northwestern.edu/index.php/NU32:_Starting_a_New_Project_and_Putting_it_on_the_NU32`

   and get your first PIC32 program running. Demonstrate the program running and talking to your laptop in class on Thursday for credit.

3. *A Crash Course in C* describes bitwise operators. Calculate the following and give your results in hexadecimal.

   (a) `0x39 | 0xA8`

   (b) `0x39 & 0xA8`

   (c) `~0x39`

   (d) `0x39>>3`

4. In one sentence each, without going into detail, explain the basic function of the following items shown in the PIC32 architecture block diagram: SYSCLK, PBCLK, PORTA...G (and indicate which of these can be used for analog input on our PIC), Timer 1-5, 10-bit ADC, PWM OC1-5, Data RAM, Program Flash Memory, and Prefetch Module.

5. If the ADC is measuring values between 0 and 3.3 V, what is the largest voltage difference that it may not be able to detect? (It's a 10-bit ADC.)

6. Describe the four functions that pin 22 of our PIC32 can have. Is it 5 V tolerant?

7. Go to the Microchip homepage, check out the Parametric Table of PIC32s (32-bit MCUs>Find Products>PIC32 Family), and find a PIC with the following specs: 80 MHz max clock speed, 512 K flash (program memory), 65 K RAM (data memory), 16 A/D channels with 10-bit resolution, USB capabilities, 2 comparators, 5 16-bit timers and 1 32-bit timer, 5 PWM channels and 5 input compare channels, 6 UARTs, 3 SPI, and 4 I2C pins, no CAN modules, Ethernet, and 64 pins. What is the part number? What types of packages does it come in (e.g., DIP, or different kinds of surface mount packages)? How much does it cost in quantity 1? What is the difference in price and features from our PIC, the PIC32MX795F512L?

8. How wide is PORTG on our PIC32 (i.e., how many pins does it have)? You should use Table 1-1 in Section 1 of the Data Sheet, don't search and count for the pins in a diagram.

9. Check out the Special Features section of the Data Sheet. If I want my PBCLK frequency to be four times less than my SYSCLK frequency, which bits of which Device Configuration Register do I have to modify? What values do I give those bits?

10. Referring to the Data Sheet, what is the name of the SFR I have to modify if I want to change pins on PORTC from output to input?

11. PIC32's have increased their flash and RAM over the years. What is the maximum amount of flash memory a PIC32 can have before the current choice of base addresses (for RAM, flash, peripherals, and boot flash) would have to be changed? Give your answer in bytes in hexadecimal.

12. The SFR OC2CON controls the behavior of Output Compare module 2. What is its virtual address? What is its physical address? (Section 4 of the Data Sheet may be helpful.)

13. Your NU32 board provides 5 V output. You plan to put a resistor from that 5 V output to ground. What is the smallest resistance that would be safe? In a sentence, explain how you arrived at the answer.