# Chapter 2

# Looking Under the Hood: Hardware

Now that we've got our first programs running, it's time to take a look under the hood. We begin with the PIC32 hardware, then move to the NU32 development board it sits on.

## 2.1 The PIC32

### 2.1.1 Pin Functions and Special Function Registers (SFRs)

The PIC32MX795F512L is powered by a supply voltage in the range 2.3 to 3.6 V and features a max clock frequency of 80 MHz, 512 KB program memory (flash), and 128 KB 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 pinout diagram for the PIC32MX795F512L (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. Some of 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 PIC32MX5xx/6xx/7xx Data Sheet 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 simply "cleared" for short, or "set to 1," or simply "set" for short.) Now, referring to the I/O Ports section of the Data Sheet, we see that the input/output direction of Port D is controlled by the SFR TRISD, and bits 0-15 correspond to RD0-15. Bit 3 of the SFR TRISD, i.e., TRISD⟨3⟩, should be cleared to 0 to make RD3 (pin 78) a digital output.

In fact, according to the Memory Organization section of the Data Sheet, OC4CON⟨15⟩ is cleared to 0 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.

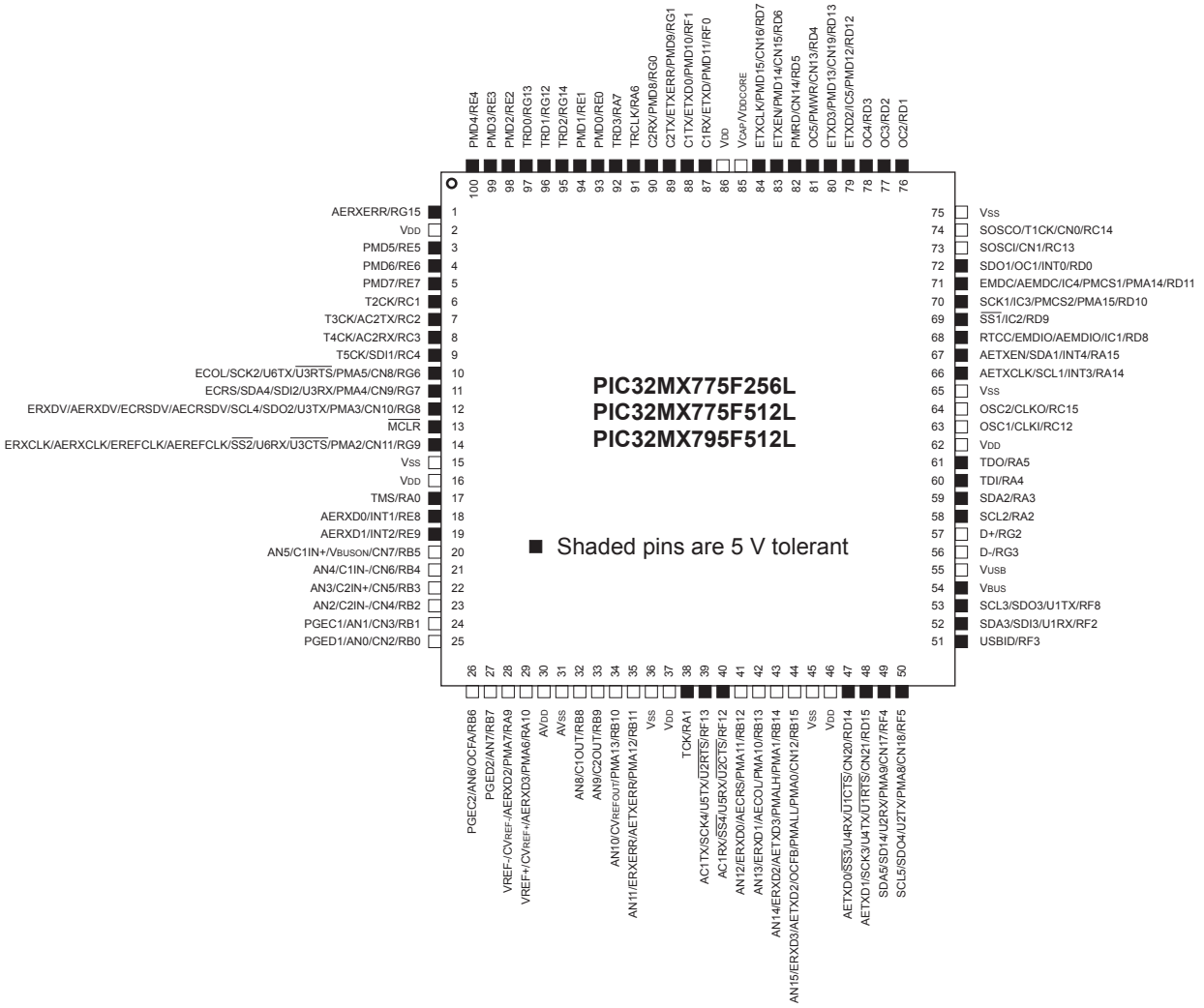


Figure 2.1: The pinout of the PIC32MX795F512L used on the NU32.

### 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 (CN) pins that 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 (PWM) pulse trains (or "output compare" OC); five pins for "input capture" (IC) 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; 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, each of which determines which of two analog inputs has a higher voltage.

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

| Pin Label | Function |
|---|---|
| **ANx (x=0-15)** | analog-to-digital (ADC) inputs |
| AVDD, AVSS | positive supply and ground reference for ADC |
| CxIN-, CxIN+, CxOUT (x=1,2) | comparator negative and positive input and output |
| CxRX, CxTx (x=1,2) | CAN receive and transmit pins |
| 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 (on the NU32 board it is set to VDD to enable the regulator) |
| **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 |
| $\overline{\text{MCLR}}$ | 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 |
| $\overline{\textbf{SS1}}$, $\overline{\textbf{SS2}}$ | 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 |
| $\overline{\textbf{UxCTS}}$, $\overline{\textbf{UxRTS}}$, **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.

communication. The timing generation block that creates these clock signals and other elements of the architecture in Figure 2.2 are briefly described below.

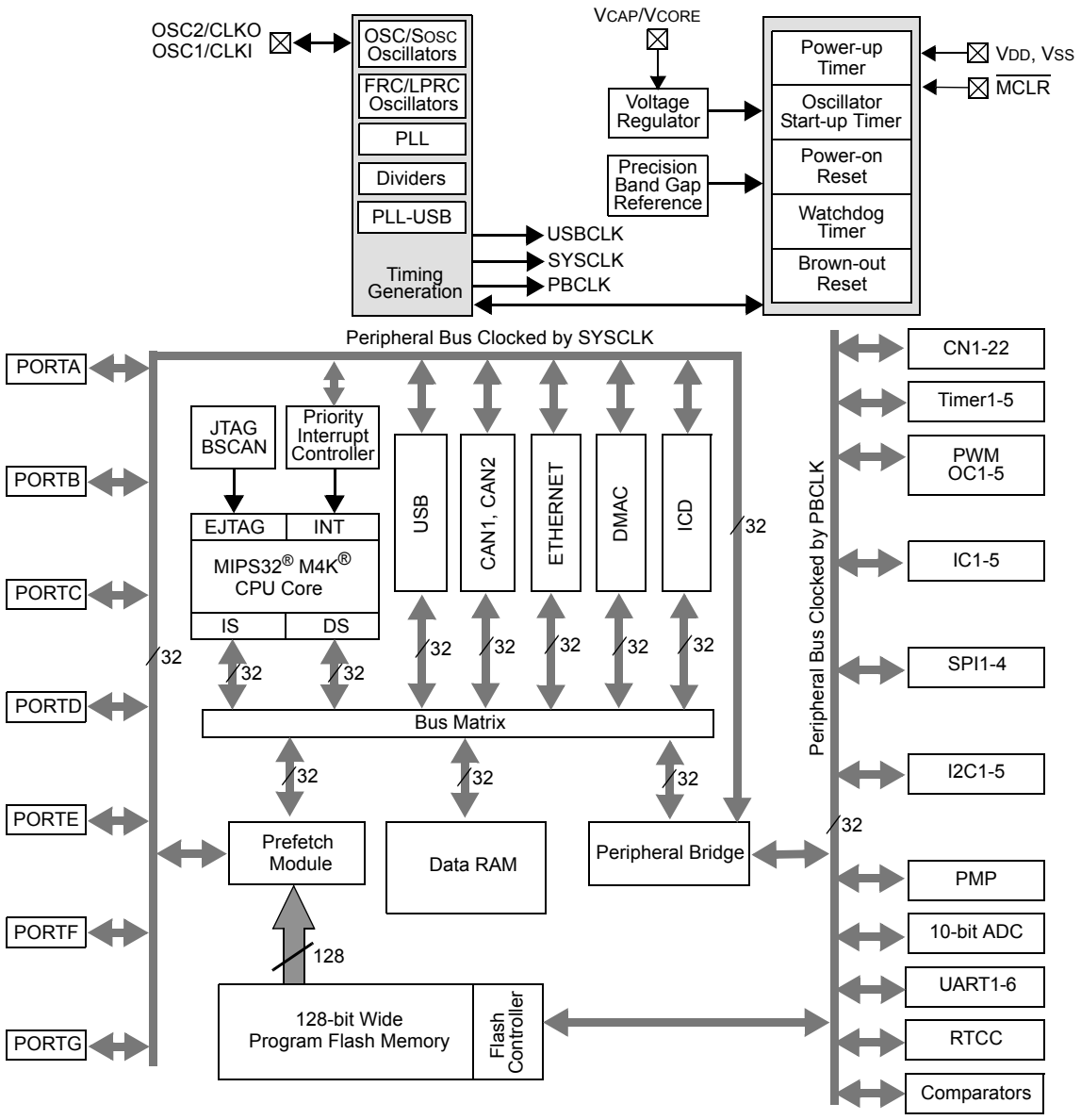


Figure 2.2: The PIC32MX5XX/6XX/7XX architecture.

**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 Imagination 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 read data from, or write data to, program memory (flash), data memory (RAM), or SFRs. 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 (IRQ) may be generated by a 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., 512 KB flash vs. 128 KB 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, with a max speed of 30 MHz for the PIC32MX795F512L, reading a program instruction from flash may take three CPU cycles when operating at 80 MHz (see Electrical Characteristics in the Data Sheet). One job of the prefetch cache module (below) is to minimize or eliminate the need for the CPU to wait around for program instructions to load.

**Prefetch Cache Module** You might be familiar with the term *cache* from your web browser. Your browser's cache stores recent documents or pages you have accessed over the web, so the next time you request them, your browser can provide a local copy immediately, instead of waiting for the download.

The *prefetch cache module* operates similarly—it stores recently executed program instructions, which are likely to be executed again soon (as in a program loop), and, in linear code with no branches, it can even run ahead of the current instruction and predictively *prefetch* future instructions into the cache. In both cases, the goal is to have the next instruction requested by the CPU already in the cache. When the CPU requests an instruction, the cache is first checked. If the instruction at that memory address is in the cache (a cache *hit*), the prefetch module provides the instruction to the CPU immediately. If there is a *miss*, the slower load from flash memory begins.

In some cases, the prefetch module can provide the CPU with one instruction per cycle, hiding the delays due to slow flash access. The module can cache all instructions in small program loops, so that flash memory does not have to be accessed while executing the loop. For linear code, the 128-bit wide data path between the prefetch module and flash memory allows the prefetch module to run ahead of execution despite the slow flash load times.

The prefetch cache module 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, approximately proportional to frequency. 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. You might want to set PBCLK's frequency lower than SYSCLK's if you want to save power. 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 for the pins that can tolerate 5.5 V).

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 turn off (allowing the voltage 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 PBCLK ticks. 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. Conversions are theoretically possible at a maximum rate of 1 million samples per second on the PIC32MX795F512L.

**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 can be used 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. It uses a separate clock, allowing it to run even when the PIC32 is in sleep mode.

**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 without involving the CPU. For example, DMA can allow an external device to dump data through a UART 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$^2$C Serial Communication** The Inter-Integrated Circuit protocol I$^2$C (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 given time. The maximum data rate is less than for SPI.

**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," but the two devices communicating must be set to the same communication rate. 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). Typical data rates are 9600 bits per second (9600 baud) up to hundreds of thousands of bits per second.

**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 PIC32MX795F512L 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, when the counter reaches a specified value, the PIC32 will reset. This is a way to have the PIC32 restart if your program has entered an unexpected state where it doesn't pet the watchdog.

### 2.1.3 The Physical Memory Map

The CPU accesses the peripherals, data, and program instructions in the same way: by writing a memory address to the bus. 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 are not nearly that many things 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 send outputs) or read from (to get sensor input, for example), and *boot flash*. Of these, we have not yet seen "boot flash." This is extra flash memory, 12 KB on the PIC32MX795F512L, that contains program instructions that are executed immediately upon reset of the PIC32.[2] The boot flash instructions typically perform PIC32 initialization and then call 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 the PIC32MX795F512L), "flashsize" bytes of flash (512 KB for the PIC32MX795F512L), 1 MB for the peripheral SFRs, and "bootsize" for the boot flash (12 KB for the PIC32MX795F512L):

[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) | Region |
|---|---|---|
| 0x00000000 | RAMsize (128 KB) | Data RAM |
| 0x1D000000 | flashsize (512 KB) | Program Flash |
| 0x1F800000 | 1 MB | Peripheral SFRs |
| 0x1FC00000 | bootsize (12 KB) | Boot Flash |

The memory regions are not contiguous. For example, the first address of program flash is 480 MB after the first address of data RAM. An attempt to access an address between the data RAM segment and the program flash segment would generate an error.

It is also possible to allocate a portion of RAM to hold program instructions.

In Chapter 3, when we discuss programming the PIC32, we will introduce the *virtual* memory map and its relationship to the physical memory map.

### 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 the PIC32 will function. You can learn more about configuration bits in the Special Features section of the Data Sheet. For example, DEVCFG1 and DEVCFG2 contain configuration bits that determine the frequency multiplier converting the external resonator frequency to the SYSCLK frequency, as well as bits that determine the ratio between the SYSCLK and PBCLK frequencies.

## 2.2 The NU32 Development Board

The NU32 development board is shown in Figure 2.3, and the pinout is given in Table 2.2. The main purpose of the NU32 board is to provide easy breadboard access to 82 of the 100 PIC32MX795F512L pins. The NU32 acts like a big 84-pin DIP chip and plugs into two standard prototyping breadboards, straddling the long rails used for power, as shown in Figure 2.3.

Beyond simply breaking out the pins, the NU32 provides a few other things that make it easy to get started with the PIC32. For example, to power the PIC32, the NU32 provides a barrel jack that accepts a 2.1 mm inner diameter, 5.5 mm outer diameter "center positive" power plug. The plug should provide DC 6 V or more; the NU32 comes with a 6 V wall wart capable of providing 1 amp. The PIC32 requires a supply voltage VDD between 2.3 and 3.6 V, and the NU32 provides a 3.3 V voltage regulator providing a stable voltage source for the PIC32 and other electronics on board. Since it is often convenient to have a 5 V supply available, the NU32 also has a 5 V regulator. The power plug's raw input voltage and ground, as well as the regulated 3.3 V and 5 V supplies, are made available to the user on the power rails running down the center of the NU32, as illustrated in Figure 2.3. Since the power jack is directly connected to the 6 V and GND rails, you could power the NU32 by putting 6 V and GND on these rails directly and not connecting to the power jack.

The 3.3 V regulator is capable of providing up to 800 mA and the 5 V regulator is capable of providing up to 1 amp. However, the wall wart can only provide 1 amp total, and in practice you should stay well under each of these limits. For example, you should not plan to draw more than 200-300 mA or so from any of the power rails. Even if you use a higher current power supply, such as a battery, you should respect these limits, as the current has to flow through the relatively thin traces of the PCB. It is also not recommended to use high voltage supplies greater than 9 V or so, as the regulators will heat up.

Since motors tend to draw lots of current (even small motors may draw hundreds of mA up to several amps), do not try to power them using power from the NU32 rails. Use a separate battery or power supply instead.

In addition to the voltage regulators, the NU32 provides an 8 MHz resonator as the source of the PIC32's 80 MHz clock signal. It also has a mini B USB jack to connect to your computer's USB port to a dual USB-to-RS-232 FTDI chip that allows your PIC32 to speak RS-232 to your computer's USB port. Two RS-232 channels share the single USB cable—one dedicated to programming the PIC32 and the other allowing communication with the host computer while a program is running on your PIC32.

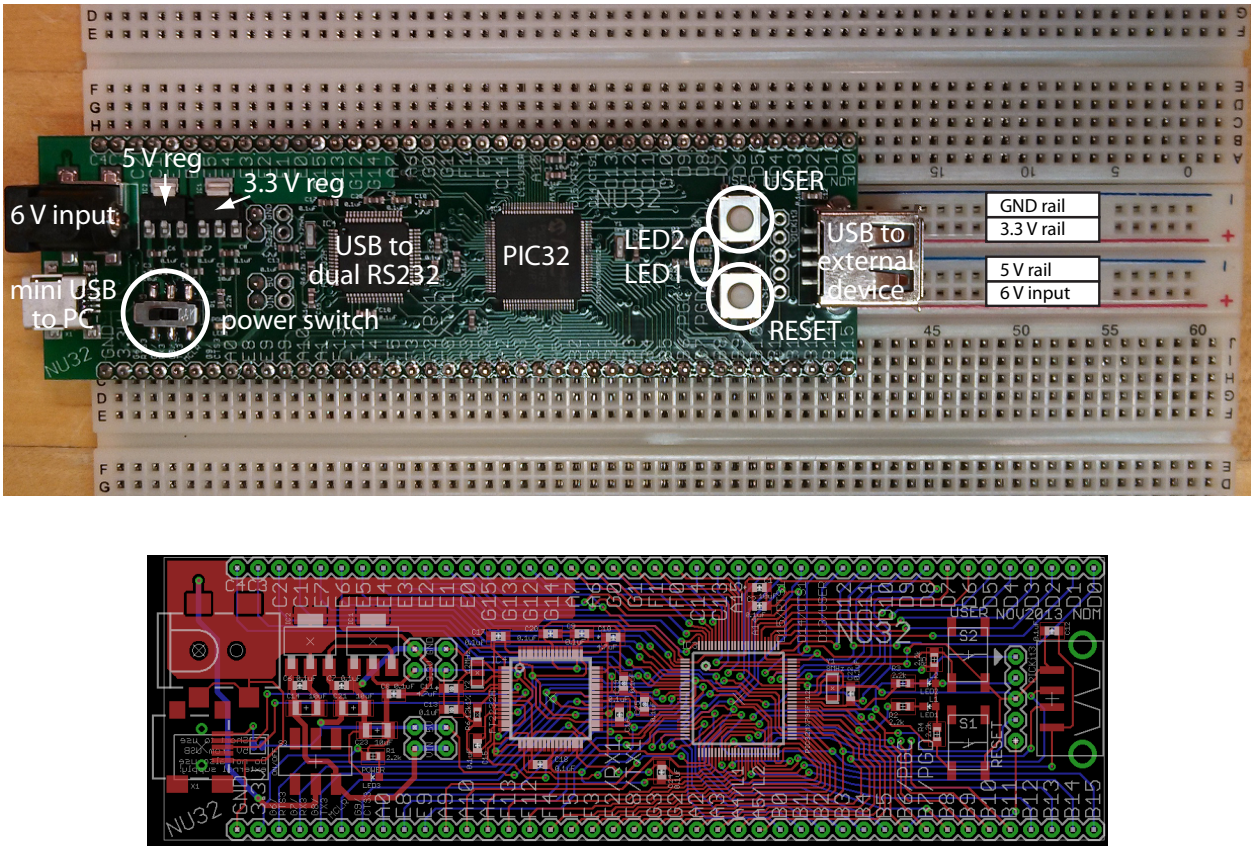


Figure 2.3: The NU32 development board: photo and PCB silkscreen.

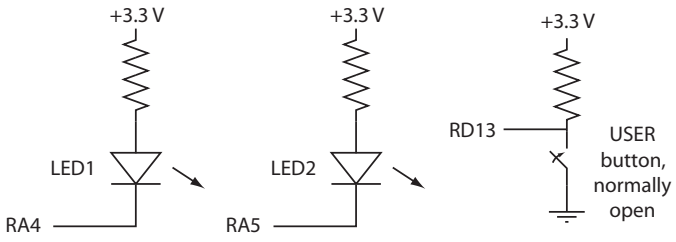


Figure 2.4: The NU32 connection of pins RA4, RA5, and RD13 to LED1, LED2, and the USER button, respectively.

A standard A USB jack is provided to allow the PIC32 to talk to another external device, like a smartphone.

The NU32 board also has a power switch which connects or disconnects the input power supply to the voltage regulators, and two LEDs and two buttons (labeled USER and RESET) allowing very simple input and output. The two LEDs, LED1 and LED2, are connected at one end by a resistor to 3.3 V and the other end to digital outputs RA4 and RA5, respectively, so that they are off when those outputs are high and on when they are low. The USER and RESET buttons are attached to the digital input RD13 and $\overline{\text{MCLR}}$ pins, respectively, and both buttons are configured to give 0 V to these inputs when pressed and 3.3 V otherwise. See Figure 2.4.

While the NU32 comes with a bootloader installed in its flash memory, you have the option to use a programmer to install a standalone program. The five plated through-holes near the USB jack align with the pins of devices such as the PICkit 3 programmer (Figure 1.4).

| Function | PIC32 | | | PIC32 | Function |
|---|---|---|---|---|---|
| GND | | GND | C4 | 9 √ | T5CK/SDI1/C4 |
| 3.3 V | | 3.3 V | C3 | 8 √ | T4CK/C3 |
| SCK2/U6TX/U3RTS/CN8/G6 | √ 10 | **G6/RTS3** | C2 | 7 √ | T3CK/C2 |
| SDA4/SDI2/U3RX/CN9/G7 | √ 11 | **G7/RX3** | C1 | 6 √ | T2CK/C1 |
| SCL4/SDO2/U3TX/CN10/G8 | √ 12 | **G8/TX3** | E7 | 5 √ | PMD7/E7 |
| MCLR | √ 13 | MCLR | E6 | 4 √ | PMD6/E6 |
| SS2/U6RX/U3CTS/CN11/G9 | √ 14 | **G9/CTS3** | E5 | 3 √ | PMD5/E5 |
| A0 | √ 17 | A0 | E4 | 100 √ | PMD4/E4 |
| INT1/E8 | √ 18 | E8 | E3 | 99 √ | PMD3/E3 |
| INT2/E9 | √ 19 | E9 | E2 | 98 √ | PMD2/E2 |
| VREF-/CVREF-/A9 | 28 | A9 | E1 | 94 √ | PMD1/E1 |
| VREF+/CVREF+/A10 | 29 | A10 | E0 | 93 √ | PMD0/E0 |
| A1 | √ 38 | A1 | G15 | 1 √ | G15 |
| SCK4/U5TX/U2RTS/F13 | √ 39 | F13 | G13 | 97 √ | G13 |
| SS4/U5RX/U2CTS/F12 | √ 40 | F12 | G12 | 96 √ | G12 |
| SDA5/SDI4/U2RX/CN17/F4 | √ 49 | F4 | G14 | 95 √ | G14 |
| SCL5/SDO4/U2TX/CN18/F5 | √ 50 | F5 | A7 | 92 √ | A7 |
| USBID/F3 | √ 51 | F3 | A6 | 91 √ | A6 |
| SDA3/SDI3/U1RX/F2 | √ 52 | **F2/RX1** | G0 | 90 √ | C2RX/PMD8/G0 |
| SCL3/SD03/U1TX/F8 | √ 53 | **F8/TX1** | G1 | 89 √ | C2TX/PMD9/G1 |
| D-/G3 | 56 | G3 | F1 | 88 √ | C1TX/PMD10/F1 |
| D+/G2 | 57 | G2 | F0 | 87 √ | C1RX/PMD11/F0 |
| SCL2/A2 | √ 58 | A2 | C14 | 74 | T1CK/CN0/C14 |
| SDA2/A3 | √ 59 | A3 | C13 | 73 | CN1/C13 |
| A4 | √ 60 | **A4/L1** | A15 | 67 √ | SDA1/INT4/A15 |
| A5 | √ 61 | **A5/L2** | A14 | 66 √ | SCL1/INT3/A14 |
| PGED1/AN0/CN2/B0 | 25 | B0 | **D15/RTS1** | 48 √ | SCK3/U4TX/U1RTS/CN21/D15 |
| PGEC1/AN1/CN3/B1 | 24 | B1 | **D14/CTS1** | 47 √ | SS3/U4RX/U1CTS/CN20/D14 |
| AN2/C2IN-/CN4/B2 | 23 | B2 | **D13/USER** | 80 √ | PMD13/CN19/D13 |
| AN3/C2IN+/CN5/B3 | 22 | B3 | D12 | 79 √ | IC5/PMD12/D12 |
| AN4/C1IN-/CN6/B4 | 21 | B4 | D11 | 71 √ | IC4/D11 |
| AN5/C1IN+/CN7/B5 | 20 | B5 | D10 | 70 √ | SCK1/IC3/D10 |
| PGEC2/AN6/OCFA/B6 | 26 | **B6/PGC** | D9 | 69 √ | SS1/IC2/D9 |
| PGED2/AN7/B7 | 27 | **B7/PGD** | D8 | 68 √ | RTCC/IC1/D8 |
| AN8/C1OUT/B8 | 32 | B8 | D7 | 84 √ | PMD15/CN16/D7 |
| AN9/C2OUT/B9 | 33 | B9 | D6 | 83 √ | PMD14/CN15/D6 |
| AN10/CVREFOUT/B10 | 34 | B10 | D5 | 82 √ | CN14/D5 |
| AN11/B11 | 35 | B11 | D4 | 81 √ | OC5/CN13/D4 |
| AN12/B12 | 41 | B12 | D3 | 78 √ | OC4/D3 |
| AN13/B13 | 42 | B13 | D2 | 77 √ | OC3/D2 |
| AN14/B14 | 43 | B14 | D1 | 76 √ | OC2/D1 |
| AN15/OCFB/CN12/B15 | 44 | B15 | D0 | 72 √ | SDO1/OC1/INT0/D0 |

Table 2.2: The NU32 pinout (in green, with power jack at top) with PIC32MX795F512L pin numbers. Board pins in **bold** should only be used with care, as they are used for other functions by the NU32. Pins marked with a √ are 5.5 V tolerant. Not all pin functions are listed; see Figure 2.1 or the PIC32 Data Sheet.

## 2.3 Chapter Summary

- The PIC32 features a 32-bit data bus and a CPU capable of performing some 32-bit operations in a single clock cycle.
- In addition to nonvolatile flash program memory and RAM data memory, the PIC32 provides peripherals particularly useful for embedded control, including analog inputs, digital I/O, PWM outputs, counter/timers, inputs that generate interrupts or measure pulse widths or frequencies, and pins for a variety of communication protocols, including RS-232, USB, ethernet, CAN, $I^2C$, and SPI.
- The functions performed by the pins and peripherals are determined by Special Function Registers. SFR settings also determine other aspects of the behavior of the PIC32.
- The PIC32 has three main clocks: the SYSCLK that clocks the CPU, the PBCLK that clocks peripherals, and the USBCLK that clocks USB communication.
- Physical memory addresses are specified by 32 bits. The physical memory map contains four regions: data RAM, program flash, SFRs, and boot flash. RAM can be accessed in one clock cycle, while flash

access may be slower. The prefetch cache module can be used to minimize delays in accessing program instructions.

- Four 32-bit configuration words, DEVCFG0 to DEVCFG3, set behavior of the PIC32 that should not be changed during execution. For example, these configuration bits determine how an external clock frequency is multiplied or divided to create the PIC32 clocks.
- The NU32 development board provides voltage regulators for power, includes a resonator for clocking, breaks out the PIC32 pins to a solderless breadboard, provides a couple of LEDs and buttons for simple input and output, and makes USB/RS-232 communication and programming simple.

## 2.4 Exercises

You will need to refer to the PIC32MX5XX/6XX/7XX Data Sheet and PIC32 Reference Manual to answer some questions.

1. Search for the "Microchip flash products parametric chart" or navigate to it from the Microchip homepage. You should see a listing of all the PICs made by Microchip. Set the page to show all specs and limit the display to 32-bit PICs.

   (a) Find PIC32s that meets the following specs: at least 128 KB of flash, at least 32 KB of RAM, and at least 80 MHz max CPU speed. (You can choose a range of settings within a single parameter by shift-clicking or ctrl-clicking.) What is the cheapest PIC32 that meets these specs, and what is its volume price? How many ADC, UART, SPI, and $I^2C$ channels does it have? How many timers?

   (b) What is the cheapest PIC32 overall? How much flash and RAM does it have, and what is its maximum clock speed?

   (c) Among all PIC32's with 512 KB flash and 128 KB RAM, which is the cheapest? How does it differ from the PIC32MX795F512L?

2. Based on C syntax for bitwise operators and bit-shifting, calculate the following and give your results in hexadecimal.

   (a) `0x37 | 0xA8`

   (b) `0x37 & 0xA8`

   (c) `~0x37`

   (d) `0x37>>3`

3. Describe the four functions that pin 22 of the PIC32MX795F512L can have. Is it 5 V tolerant?

4. Referring to the Data Sheet section on I/O Ports, what is the name of the SFR you have to modify if you want to change pins on PORTC from output to input?

5. The SFR CM1CON controls comparator behavior. Referring to the Memory Organization section of the Data Sheet, what is the reset value of CM1CON in hexadecimal?

6. In one sentence each, without going into detail, explain the basic function of the following items shown in the PIC32 architecture block diagram Figure 2.2: SYSCLK, PBCLK, PORTA...G (and indicate which of these can be used for analog input on the NU32's PIC32), Timer 1-5, 10-bit ADC, PWM OC1-5, Data RAM, Program Flash Memory, and Prefetch Cache Module.

7. List the peripherals that are *not* clocked by PBCLK.

8. 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.)

9. Refer to the Reference Manual chapter on the Prefetch Cache. What is the maximum size of a program loop, in bytes, that can be completely stored in the cache?

10. Explain why the path between flash memory and the prefetch cache module is 128 bits wide instead of 32, 64, or 256 bits.

11. Explain how a digital output could be configured to swing between 0 and 4 V, even though the PIC32 is powered by 3.3 V.

12. 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 in the physical memory map (for RAM, flash, peripherals, and boot flash) would have to be changed? What is the maximum amount of RAM? Give your answers in bytes in hexadecimal.

13. Check out the Special Features section of the Data Sheet.

    (a) If you want your PBCLK frequency to be half the frequency of SYSCLK, which bits of which Device Configuration Register do you have to modify? What values do you give those bits?

    (b) Which bit(s) of which SFR set the watchdog timer to be enabled? Which bit(s) set the postscale that determines the time interval during which the watchdog must be reset to prevent it from restarting the PIC32? What values would you give these bits to enable the watchdog and to set the time interval to be the maximum?

    (c) The SYSCLK for a PIC32 can be generated in a number of ways. This is discussed in the Oscillator chapter in the Reference Manual and the Oscillator Configuration section in the Data Sheet. The PIC32 on the NU32 uses the (external) primary oscillator in HS mode with the phase-locked loop (PLL) module. Which bits of which device configuration register enable the primary oscillator and turn on the PLL module?

14. Your NU32 board provides four power rails: GND, regulated 3.3 V, regulated 5 V, and the unregulated input voltage (e.g., 6 V). You plan to put a load from the 5 V output to ground. If the load is modeled as a resistor, what is the smallest resistance that would be safe? An approximate answer is fine. In a sentence, explain how you arrived at the answer.

15. The NU32 could be powered by different voltages. Give a reasonable range of voltages that could be used, minimum to maximum, and explain the reason for the limits.

16. Two buttons and two LEDs are interfaced to the PIC32 on the NU32. Which pins are they connected to? Give the actual pin numbers, 1-100, as well as the name of the pin function as it is used on the NU32. For example, pin 57 on the PIC32MX795F512L could have the function D+ (USB data line) or RG2 (Port G digital input/output), but only one of these functions could be active at a given time.