

The Basics – Digital Input

After Chapter 3’s examination of the output mode, we’ll now turn to PIC pins used as *digital input* devices. Many PICs include analog-to-digital converters and we’ll cover analog inputs in Chapter 11.

Digital signal levels are either a logical low (0) or a logical high (1)—what could be simpler? As with the rest of this book, we assume V_{DD} is 5 volts and V_{SS} is 0 volts. In PIC logic, a 0 volt input corresponds to a logical low. Likewise, a 5V input is a logical high. But, suppose our input is 1.7V. Is it a logical low or is it a logical high? Does the answer to this question depend on our choice of an input pin? And, does it depend on the voltage at the input pin earlier in time? We’ll find out in this chapter.

Introduction

First, let’s define a few terms, as illustrated in Figure 4-1:

V_{IL} —The maximum voltage on an input pin that will be read as a logical low.

V_{IH} —The minimum voltage on an input pin that will be read as a logical high.

Undefined region—The undefined region the voltage level between V_{IL} and V_{IH} . Input voltages in the undefined region may be read as a low or as a high, as the PIC’s input circuitry may produce one result or the other. (Obviously the voltage will be read as either a low or a high; it’s just that we have no assurance which one it will be.)

Threshold voltage—The input voltage that separates a low from a high; the threshold voltage, V_T , minus a small increment is read as a low while the threshold voltage plus a small increment is read as a high. The threshold voltage differs from logic family to logic family and somewhat between different chips of the same type. The differences between V_T and V_{IL} and V_{IH} are design margins accounting for device-to-device process tolerances, temperature effects and the like.

In an ideal logic device, V_{IL} equals V_{IH} and there is no undefined region.

Microchip has chosen to build PICs with varying input designs and associated varying V_{IL} and V_{IH} values. We will not consider a few special purpose pins, such as those associated with the oscillator, in this chapter. Even so, the 16F87x series PICs have three input pin variations:

TTL level— Of the many logic families introduced in the early days of digital integrated circuits, transistor-transistor logic (TTL) was by far the most successful with TTL and its descendants still used today. Port A and Port B input pins mimic TTL logic input levels, except for RA4, which has a Schmitt trigger input. (Of course, TTL style Schmitt trigger inputs exist; but since they are not found in the PICs we consider, we won’t further consider them.)

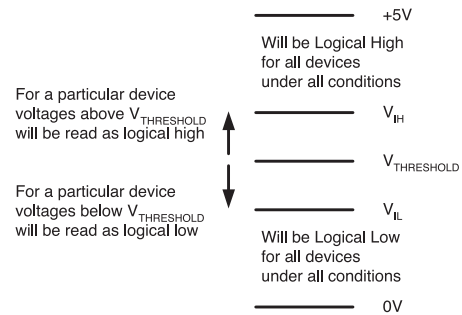


Figure 4-1: Input level relationships.

Schmitt trigger inputs—Almost all other input pins are of Schmitt trigger design. A Schmitt trigger has different transition voltages, depending on whether the input signal is changing from high to low or low to high, as illustrated in Figure 4-2.

Special Schmitt trigger inputs—Pins RC3 and RC4 are software selectable Schmitt trigger or SMBus configuration. (SMBus is a protocol developed by Intel for data exchange between integrated circuits. We will not further discuss SMBus communications in this book.) When RC3 and RC4 are used as normal, general purpose, input pins, their parameters differ slightly from those of the other Schmitt trigger input pins.

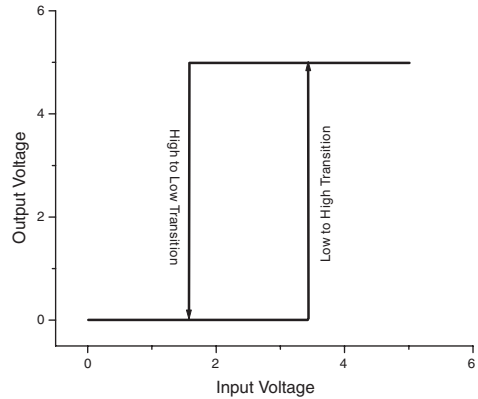


Figure 4-2: Schmitt trigger input.

Input Logic Level Specifications: 16F87x with $4.5V < V_{DD} < 5.5V$		
Input Type	$V_{IL}(max)$	$V_{IH}(min)$
TTL	0.8V	2.0V
Schmitt	$0.2 V_{DD}$	$0.8V_{DD}$
RC3/RC4 Schmitt	$0.3V_{DD}$	$0.7V_{DD}$

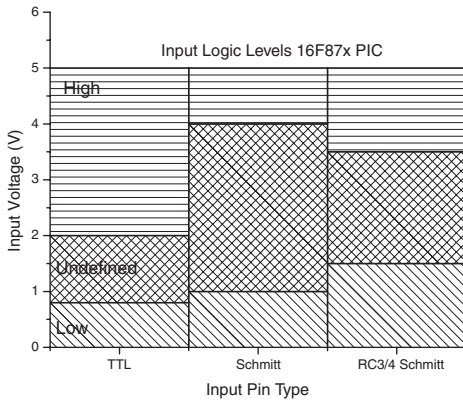


Figure 4-3: Input logic level comparison.

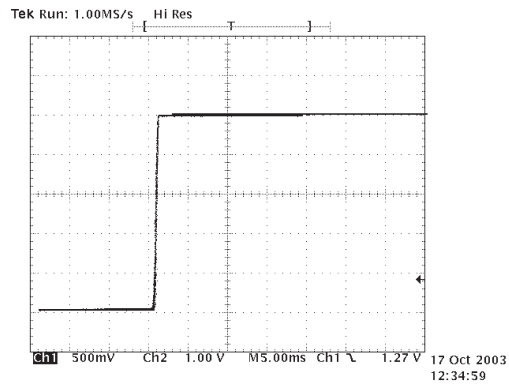


Figure 4-4: SN7400 TTL logic levels Ch1: X-axis (gate input); Ch2: Y-axis (gate output).

Figure 4-3 illustrates the differences between the three input types. To see how these input designs differ in practice, we can apply a time varying 0...5 V signal to an input pin and plot the input voltage against the output value using an oscilloscope. If we have a stand-alone logic gate, this is a straightforward exercise. Figure 4-4, for example, shows the input versus output result for a SN7400 TTL quad NAND gate. (To obtain a noninverting output, the test configuration places two NAND gates in series, and to yield the full 5 V logic high, a 2.2K ohm pull-up resistor was added to the output gate.) We see a clear, narrow transition, with $V_{TRANSITION}$ around 1.6 V, fully consistent with Microchip's V_{IL} and V_{IH} parameters for TTL mimicking input pins.

If we wish to obtain a similar plot with a 16F877, however, we must somehow determine the logical state—high or low—of the input pin corresponding to the input voltage to. The easiest way to do this is simply echo the input pin's value to an output pin. In pseudo-code the algorithm is:

```

ReadPin:
Read Input Pin - If Input Pin <> 1 Make Output Pin = 0
Read Input Pin - If Input Pin <> 0 Make Output Pin = 1
Goto ReadPin

```

It turns out that to be useful, the process of reading the input pin and making the output pin must be done more quickly than possible using MBasic, we'll add some high speed assembler into the mix. The reason for the unusual pseudo-code structure (the not equal operator) will become clear when we look at the real code.

Program 4-1

```

;Program 4-01

Input B0
Output B1

Main

ASM
{
ReadIn
    btfss PortB,0      ;If RB0=1 then skip setting it to 0
    bcf   PortB,1      ;make RB1=0
    btfsc PortB,0      ;If RB0=0 then skip setting it to 1
    bsf   PortB,1      ;make Rb1=1
    GoTo  ReadIn       ;Repeat the loop
}

GoTo Main
End

```

Don't worry if you don't understand the assembler portions of Program 4-1, as we will learn more about mixing assembler and MBasic later. However, the actual program tracks the pseudo-code. We first make RB0 an input and RB1 an output using MBasic. Then, the main program is an endless loop.

The `btfss PortB,0` statement reads the 0'th bit of Port B (RB0) and if it is "set," i.e., if it reads high, the immediately following statement is skipped. If it is low, the statement immediately following is executed. (The mnemonic is Bit Test File, Skip if Set, or `btfss`).

```

    btfss PortB,0      ;If RB0=1 then skip setting it to 0
    bcf   PortB,1      ;make RB1=0

```

The above code reads RB0 and branches, depending on its value. If RB0 is set—that is, RB0 = 1, then the next statement (`bcf PortB,1`) is skipped. Conversely, if RB0=0, then `bcf PortB,1` is executed. The `bcf`—or bit clear file—operator clears or sets to zero a bit, in this case, RB1. These two statements, therefore, read RB0 and set RB1 to zero if RB1 is zero.

```

    btfsc PortB,0      ;If RB0=0 then skip setting it to 1
    bsf   PortB,1      ;make Rb1=1

```

The next two statements perform the inverse operation. RB0 is read a second time and tested—but this time for the clear state—using the `btfsc` operation. (The `btfsc` operator works just like `btfss`, except the next instruction is skipped if the tested bit is clear.) If RB0 is high, then the operation `bsf PortB,1` is performed, thereby setting RB1.

Execution continues with the `GoTo ReadIn`, which loops back to reading RB0.

To terminate this program, power must be removed from the PIC, or another program written into its memory.

Program 4-1 isn't the fastest way to transfer an input pin level to an output pin, but we'll look at more efficient techniques later. It also uses multiple read actions, thereby creating the possibility of mishandling if the input changes value between the two read operations.

Chapter 4

With a 20 MHz clock, this program has a 1.2 μ s operating cycle. (The SN7400 gate, performing these tasks in hardware, requires less than 10 ns, 120 times faster than the PIC's software.) Hence, when we examine the input/output relationship with the same set up we used for the SN7400 gate, we will expect to see horizontal smearing, where the output lags the input by this delay. See Figure 4-5. We see $V_{THRESHOLD}$ is approximately 1.5 V, quite close to the value measured for the SN7400 true TTL gate.

Modifying Program 4-1 to accept RC0 as the input and running the same input/output sweep, we see in Figure 4-6 the hysteresis of the Schmitt trigger. The low-to-high transition occurs at approximately 3.1 V, while the high-to-low transition occurs at approximately 1.8 V. The beauty of separate high-low and low-high transition levels is noise rejection. Suppose the input signal has noise riding on it, perhaps induced from high-speed logic chips on the same board. Once a transition from one state to the other has occurred, it takes a 1.3 V noise excursion to cause a reverse transition. As we shall see later, this hysteresis adds greatly to noise rejection.

We understand Microchip's decision to use TTL mimicking inputs as the product of backward compatibility with earlier PICs and access to the huge base of TTL devices. But, why has Microchip designed the rest of its PIC inputs with Schmitt trigger inputs instead of normal CMOS inputs, such as that seen in Figure 4-7 for a CD4001BE quad NOR gate? After all, PICs are CMOS devices so it makes sense to give them standard CMOS characteristics, right?

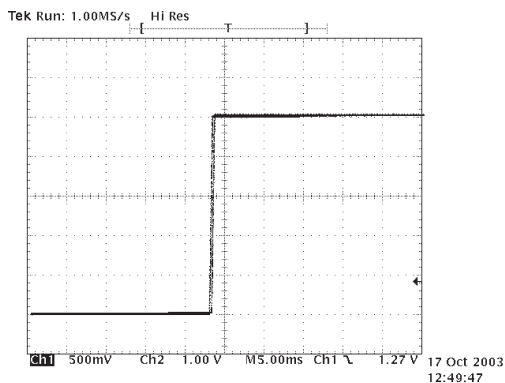


Figure 4-7: CD4001BE quad 2-input NOR CMOS logic levels Ch1: X-axis (input); Ch2: Y-axis (output).

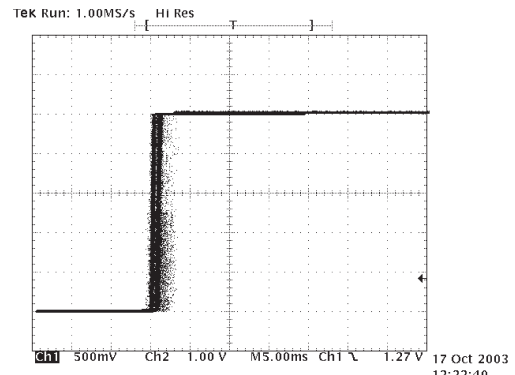


Figure 4-5: 16F876 TTL logic levels Ch1: X-axis (RB0 input); Ch2: Y-axis (RB1 output).

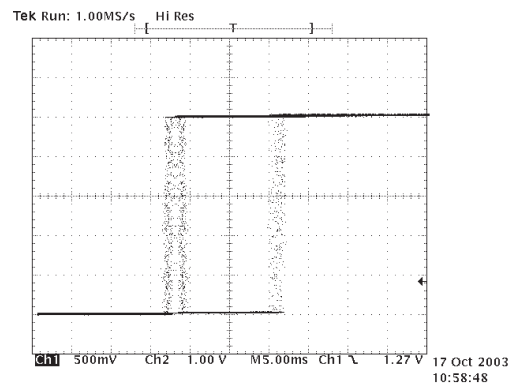


Figure 4-6: 16F876 Schmitt trigger logic levels Ch1: X-axis (RC0 input); Ch2: Y-axis (RB1 output)

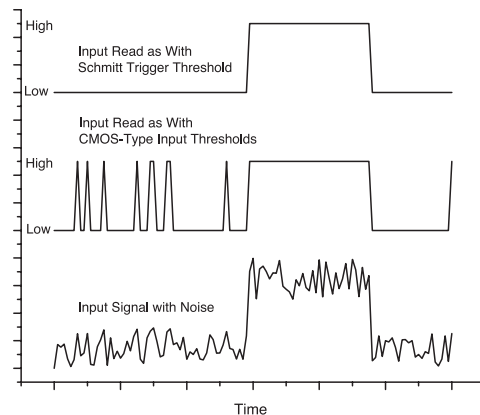


Figure 4-8: Noisy input read differently by standard CMOS and Schmitt inputs.

If Microchip thought its PICs would communicate only with other integrated circuits installed on the same board, it likely would have adopted standard CMOS input levels. But, PICs often must communicate with the real world, through sensors and switches, operating in a much less benign environment where the noise rejecting properties of the Schmitt trigger come to the fore. Figure 4-8 illustrates the ability of a Schmitt trigger input to correctly read an input signal in the presence of large noise voltages.

In order to ensure that levels are correctly read, regardless of the input type, we should aim for logic 0 input levels not exceeding 0.8 V and logic high levels of at least 4.0 V. If we meet these objectives, all input port types will correctly read the input levels. If these levels cannot be achieved, it will be necessary to further investigate the design to ensure reliable data transfer.

Regardless of their type, PIC input pins represent a high input impedance to the outside world. As reflected in Figure 4-9, the only current that flows in or out of an input pin is due to leakage and does not exceed 1 μ A. (In the 16F876 family, pin RA4's leakage current may be up to 5 μ A.) For low impedance circuits we may safely consider an input pin as an open circuit, with zero current flow in or out of the pin. A high impedance input pin carries with it the

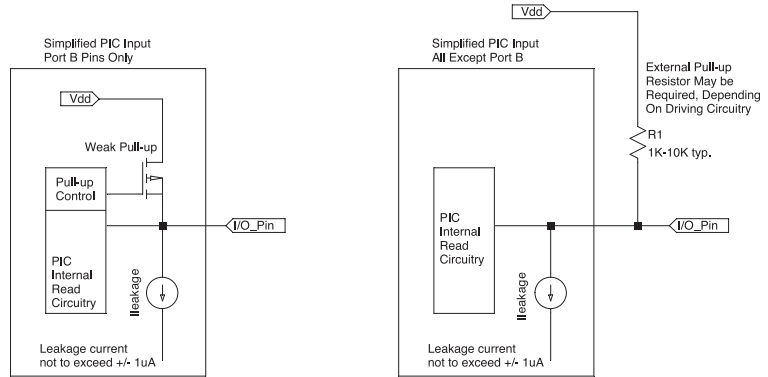
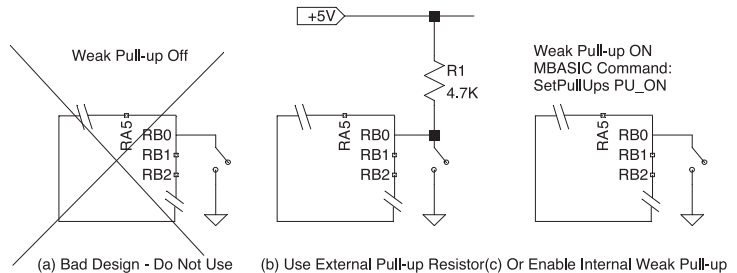


Figure 4-9: Simplified model of input pins.

possibility of static damage, as even a small static charge produces high voltages into a high impedance pin. Although the clamping diodes to V_{DD} and V_{SS} help prevent damage, it's still a good idea to follow anti-static precautions when handling PICs. It also means that input pins should be protected when exposed to outside world voltage and currents. We will briefly cover some of these real world issues in this and later chapters.

The exception to this assumption relates to the software enabled internal pull-up resistors on pins RB0...RB7. If the weak pull-up feature on Port B is enabled via MBasic's procedure `SetPullUps`, the input pins are connected to V_{DD} through an internal 20 Kohm resistor. If enabled through `the SetPullUps PU_On` or disabled through `SetPullUps PU_Off` procedure, the action applies to all PortB pins. In this case, the input pin sources 250 μ A.

We've referred to pull-up resistors without explaining why and where they are used. Suppose we wish to read a switch and determine if it is open or closed. If we simply connect the switch to an input pin, as in Figure 4-10a, we cannot be assured of RB0's status when the switch is open. Certainly, if the switch is closed, RB0 will be at ground potential and will be read as a logical low. When open, however, the voltage at RB0 results from the PIC's internal random leakage current, plus whatever stray signal that outside circuitry may induce in the connections between the switch and



Figures 4-10a,b,c: Pull-up required to read switch.

RB0. The result may be a logical high, or low, and we are without any assurance that the switch’s position will be correctly or consistently read.

If instead, we connect RB0 to V_{DD} , either through the internal pull-up source (Figure 4-10c) or through an external pull-up resistor (Figure 4-10b), when the switch is open RB0 will be “pulled up” to V_{DD} and the switch position will correctly be read as a logical high.

Switch Bounce and Sealing Current

Beyond assuring that an open switch is correctly read, a pull-up resistor provides the switch with enough current to reliably operate. In addition to the mechanical wiping action of switch contacts, a small DC current greatly assists in maintaining reliable conduction between moving switch contacts. And, if the switch is connected through wiring with mechanical splices and crimp or screw pressure terminals, the current helps clean oxidizing film from the mating conductors. From telephone terminology, we often refer to this as a “sealing” current or a “wetting” current. Hence switch circuits are often referred to as wet—carrying a current well in excess of that necessary to pull an input pin high or low—or dry—carrying only a minimal signal current. For most mechanical switches, unless we are trying to save power such as in a battery powered system, select the pull-up resistor to supply 5 to 20 mA current flow through the switch when closed. For 5 V systems, use a pull-up resistor of 1K ohm to 250 ohms. Finally, the smaller the pull-up resistor value, the less stray voltage will be induced into the wiring connecting the PIC with the switch.

When a mechanical switch operates, the same “contact bounce” phenomena we observed in Chapter 3 with relay contacts is seen. In most switches, the contact separation operation (“break”) is relatively clean, but the contact closure (“make”) exhibits multiple bounce events. Figure 4-11 shows nearly 1.5 ms is required to reach a steady state closed condition in the microswitch SPDT switch being tested.

Why should we be worried about switch bounce? The answer is that in many cases, we don’t care. Bounce isn’t a concern, for example, where a baud rate switch is read only at start-up, or where a limit switch senses the position of a part and activates a software stop sequence. In the first case, the switch isn’t operated while the program executes and in the second multiple activations of stop sequence isn’t a concern. Suppose, however, the switch counts the number of times an action is performed. Clearly we want one switch operation to increment the count once, not once for each of a dozen or so bounces. To prevent switch bounce from repeatedly triggering an operation, we must debounce the switch.

We can debounce a switch either with external electronics, or in software.

Hardware Debouncing

Although specialized integrated circuit “debouncers,” such as Fairchild’s FM809 microprocessor supervisor devices, exist we’ll look at a simple circuit described in a recent issue of *EDN Magazine*, as shown in Figure 4-12.^[4-1]

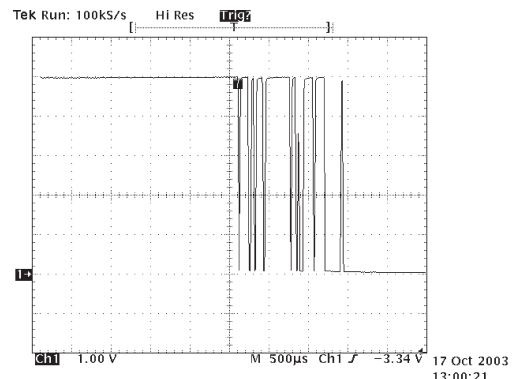


Figure 4-11: Contact bounce upon closure of microswitch.

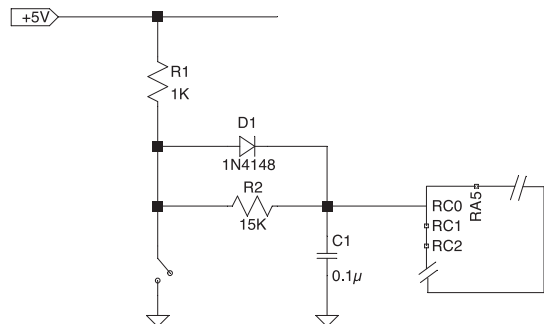


Figure 4-12: Hardware debounce.

When the switch is open, C1 charges to V_{DD} and RC0 is read as high. When the switch makes, C1 is discharged through R2 (D1 is reversed biased and may be neglected) and will be read as low when the voltage across C1 drops below high-to-low transition voltage, approximately 1.8 V. If the time constant of R2-C1 is long compared with the individual bounce intervals, the decay will be smooth and only one transition through $V_{THRESHOLD}$ will occur. But, even if spikes of several hundred millivolts occur at RC0 the output will stay low, as in order to change its read state, RC0 must see the low-to-high transition voltage of approximately 3.1 V.

When the switch is closed, the input pin is connected to ground through R2. The leakage current from the PIC input pin is rated not to exceed $1\mu A$, so in the worst case with R2 at 15K ohm, the input pin will be at 0.015 V, well within the logical low range.

When the switch is opened after closure, C1 charges through R1 and R2 in parallel with D1. Until the voltage across C1 reaches 0.7 V from V_{DD} , C1 charges mostly through R1 and D1. Hence, the charge cycle is significantly shorter than the discharge cycle. This design assumes that the switch has bounce problems only on make and therefore little or no anti-bounce effect is required on break.

The relationship between C1, R2 and the desired debounce time T_B is given by:

$$R2C1 = - \frac{T_B}{Ln \frac{V_{THRESHOLD}}{V_{DD}}}$$

If we design for a Schmitt input where $V_{THRESHOLD}$ for a high to low transition is approximately 1.8 V, and if we make C1 0.1 μF , a convenient value, we may simplify this equation and solve for R2 in terms of T_B :

$$R2 \approx 10T_B$$

R2 is in Kohm, and T_B is in milliseconds. If the desired debounce time is 1.5 ms, R2 should be 15K ohm. Figure 4-13 shows how well this simple circuit removes the bounce from the same switch shown in Figure 4-11.

To study the effect of the debounce circuit, we simply make RB0 equal to RC0 and repeat the test in an endless loop.

Program 4-2

```

;Program 04-02
;Program to echo read of C0 to B0
;Variables
Temp   Var           Byte

Input  C0
Output B0

Main
    PortB.Bit0 = PortC.Bit0
GoTo  Main
End
    
```

The key statement in the program is **PortB.Bit0 = PortC.Bit0** where the assignment operator forces a read of RC0 and a subsequent write of the resulting value to RB0. This read/write sequence is repeated every time the loop executes. Program 4-2 runs quite a bit slower than Program 4-1, with the switch pin RB0 being read once every 68 μs , compared with the once every 1.2 μs in the assembler code.

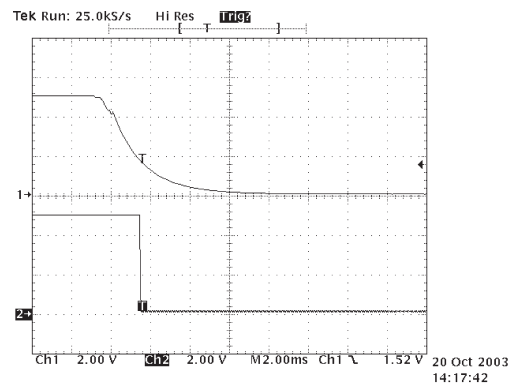


Figure 4-13: External debounce circuit operation
Ch1: RC0 PIC input; Ch2: RB0 PIC output.