AN1104

Capacitive Multibutton Configurations

Author: *Keith Curtis*
Microchip Technology Inc.
Tom Perme
Microchip Technology Inc.

INTRODUCTION

This application note describes how to scan and detect button presses on more than 4 capacitive buttons. The target devices of this application note are the PIC16F616 family, PIC16F690 family and the PIC16F887 family of microcontrollers. It assumes knowledge of the general concept for capacitive sensing described in application note AN1101, "*Introduction to Capacitive Sensing*," and it is recommended to have read AN1101 prior to this application note.

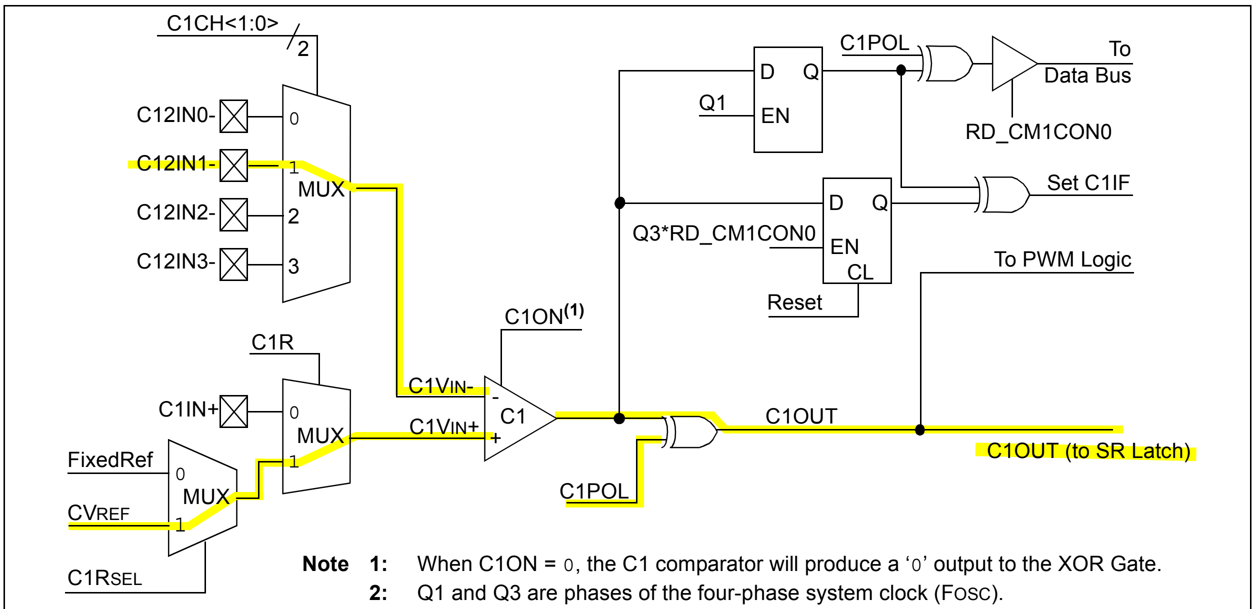
This application note will discuss three different approaches to implementing multiple touch buttons using Microchip microcontrollers. The first approach creates a simple 4 sensor system using the on-chip 4-to-1 analog multiplexers tied to the inputs of the comparator module. The second approach expands the 4 sensor system of the first approach, into a 10 sensor system by combining pairs of the original 4 inputs. The third approach creates an expandable system, which relies on multiplexing additional sensors through an external analog multiplexer.

USING DEFAULT CAPACITY

By default, PIC® microcontrollers with a comparator module capable of capacitive sensing*, may use the internal multiplexor to the comparator inputs to scan up to four buttons. The internal MUX is controlled by the channel select bits, C1CH<1:0> of CM1CON0 and C2CH<1:0> of CM2CON0. The channel must be set the same for both comparators.

A simplified block diagram from the data sheet for the PIC16F887 family is shown in Figure 1. It depicts the proper paths required for capacitive sensing as highlighted. The channel of select must be the same on each interanl multiplexer. So, when switching buttons from one to the next, ensure that they are the same. They must be the same because the basic sensing oscillator circuit requires the voltage on the capacitor to be compared to the upper and lower limit C1IN+ and C2IN+. If the negative inputs were different, the circuit would not oscillate and would be stuck either high or low.

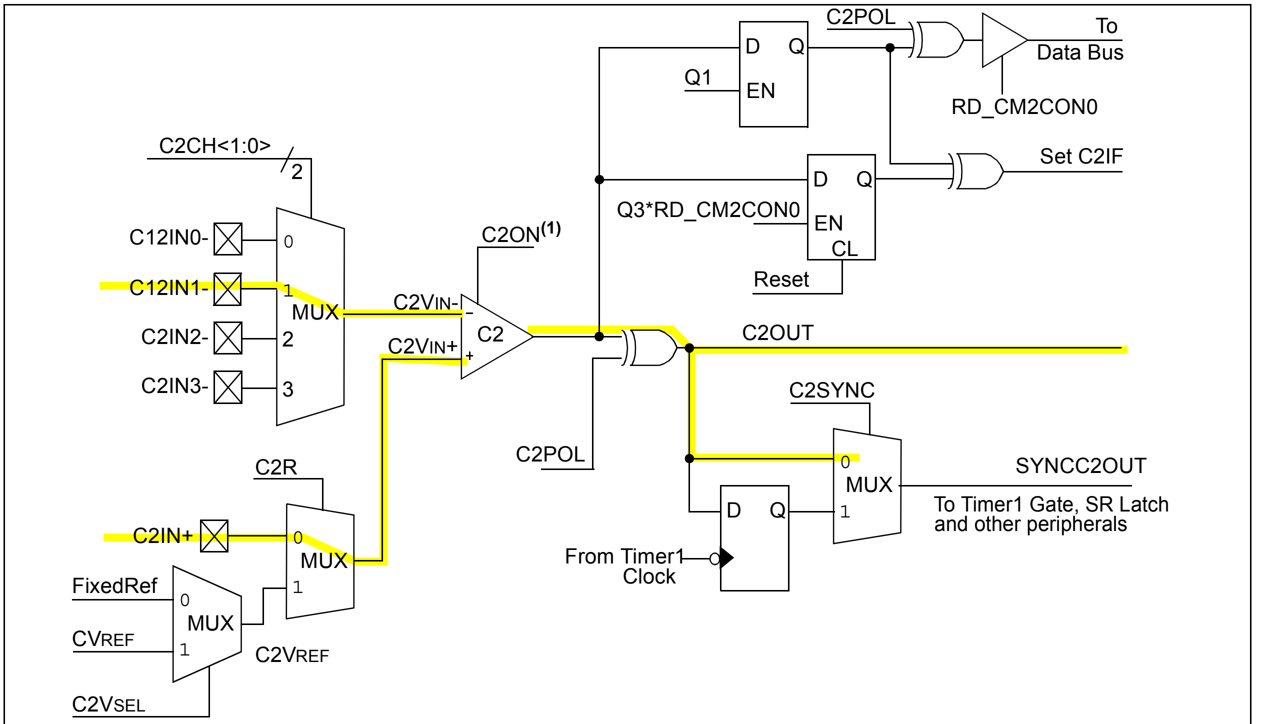
FIGURE 1: COMPARATOR C1 SIMPLIFIED BLOCK DIAGRAM



Note 1: When C1ON = 0, the C1 comparator will produce a '0' output to the XOR Gate.
2: Q1 and Q3 are phases of the four-phase system clock (FOSC).
3: Q1 is held high during Sleep mode.

* Comparator modules capable of touch sense as described must have the SR-latch option.

FIGURE 2: COMPARATOR C2 SIMPLIFIED BLOCK DIAGRAM



Note 1: When C1ON = 0, the C2 comparator will produce a '0' output to the XOR Gate.
2: Q1 and Q3 are phases of the four-phase system clock (FOSC).
3: Q1 is held high during Sleep mode.

To handle scanning four buttons is straightforward in software when using C code. It is easiest to handle the buttons' measured raw and average data as arrays, where each entry is a button's value. It is also nice to have individual trip thresholds for each button, in the event different buttons behave differently. In plain C, these variables would be declared as follows:

```
unsigned int   average [4];
unsigned int   trip [4];
```

Note:	A simple trip threshold will be assumed for purposes of discussion. Application note AN1103, "*Software Handling of Capacitive Sensing,*" contains more detail on various methods of detecting button presses. Averaging is fundamental to all methods.

The array variable average holds the average values for each button, indexed 0 to 3. Likewise,the most recent reading may be stored in an array for viewing or to aid design, but this raw data does not need to be stored since it is measured at the end of a scan when a decision for pressed or not pressed is made in the Interrupt Service Routine (ISR).

To set the big picture before going into further detail, the basic operation is completed in the Interrupt Service Routine. The ISR will be called by a Timer0 overflow interrupt each time a button is ready to have a measurement taken and complete a scan. It may be called due to other interrupts, and those should be handled appropriately by the user, to coexist with the fixed time-base Timer0 interrupts. An abstracted, high level form is as follows:

EXAMPLE 1: SIMPLE ISR

```
void isr {
     // If capacitive interrupt
     if (T0IF == 1) {

        // Grab TMR1 Reading
        StopTimers();
        GetMeasurement();

        // Test if Pressed
        if (IsButtonPressed(index))
          SetFlag(index);
        else
          ClearFlag(index);

        // Perform 16-point average
        PerformAverage(index);

        // Set next sensor
        index = (++index) & 0x03;
        SetComparators(index);
        RestartTimers();
     }
}
```


Handling multiple buttons entails writing the portion below the comment “Set next sensor”. To advance through the four buttons, an index variable will be needed to keep track of which button is being scanned and to initiate proper settings based on that index. Using four buttons, the index will go from 0 to 3, just like the array of average and trip values. This variable will be declared as such:

```
unsigned char    index;
```

At the end of the ISR, the index variable will increment on each scan to prepare for the next scan. After incrementing the index variable, the comparator channel select bits, C1CH<1:0> and C2CH<1:0>, must be set and Timer0 and Timer1 must be restarted. This may be done many ways, but a convenient way is to create an array of 4 constants with the settings for the entire registers, CM1CON0 and CM2CON0, and then use the index to grab the indexed value and load the registers.

For example, assume the constant arrays are declared as COMP1 and COMP2. The values come from the necessary settings for the register, and then changing the channel bits, bit 0 and bit 1 of each.

```
COMP1 = {0x94, 0x95, 0x96, 0x97}
COMP2 = {0xA0, 0xA1, 0xA2, 0xA3}
```

After index change (at end of ISR):

EXAMPLE 2: SETTING COMPARATORS

```
void SetComparators(char index) {
    CM1CON0 = COMP1[index];
    CM2CON0 = COMP2[index];
}
```

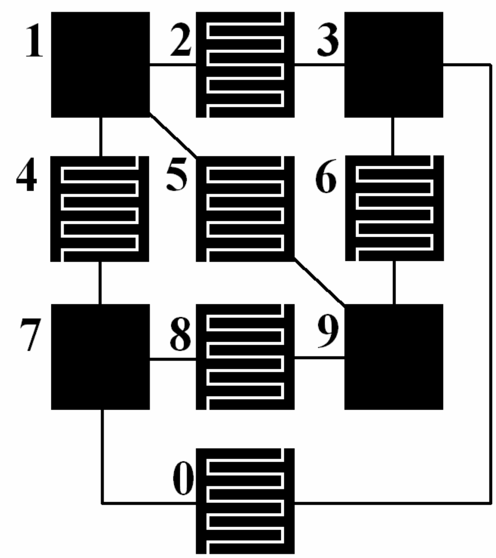
Each pass, the ISR will perform its scan, increment index, and then prepare for the next button to scan, as done by setting the CMxCON0 registers and restarting the timers. The index must only increment to 3 and it must wrap-around from 3 back to 0, but this is a software detail that is easy to handle. One way to count to three and wrap-around is to AND the result with 3, which will clear the uppermost 6 bits, as done in Example 1 above. The variable index increments to 4 (0b100), and then an AND operation with 3 (0b011) makes the result 0.

Now, four buttons are scanned sequentially 0, 1, 2, 3, 0, 1, 2, 3 … over and over, and the remaining portions to complete are the SetFlag(int index), ClearFlag(int index) and PerformAverage(int index) functions. Setting and clearing flags may suffice for some applications, other applications may directly take an action. Base the action of a button on its index, because the index is used to indicate which button is pressed. The same is true for the running 16 point average. The average should be recalculated each pass, and it should be stored in the appropriate index of the average array.

EXPANDING BY PAIRED PRESS

One of the major drawbacks to using the comparator module for a touch button interface is its limited number of inputs. One way to add support for additional sensors is to create new touch sensors that combine pairs of existing touch inputs (see Figure 3). When a combined pair sensor is touched by the user, both of the shared sensor inputs are affected equally and the software differentiates the touch from a single input by the reduce shift and the affect on two inputs instead of just one.

FIGURE 3: MULTIPLE BUTTONS WITH ONLY 4 INPUTS



Because the paired sensor inputs combine existing inputs, no additional circuitry is required and the memory overhead for the averaging system is not increased. The only additional requirement on the decoding logic is the need to search for both single and paired press conditions.

Table 1 lists the number of touch sensors that can be generated, given a fixed number of sensor inputs. Since the comparator input has an internal MUX with four channels, the maximum number of sensors is ten.

TABLE 1: MAXIMUM NUMBER OF SENSORS

Number of inputs	Number of sensors
2	3
3	6
4	10
5	15
6	21
7	28
N	$0.5 (N^2 + N)$

Using the paired sensor system does have limitations. Only one sensor can be pressed at a time and the paired sensors will only shift the frequency of the sensor circuit half as far as full sensors. This will require some additional logic in the decoding routines, and can limit the sensitivity of some sensor inputs. The designer should take this into consideration when laying out the system, placing full sensors in applications requiring greater sensitivity, and placing paired sensors in applications that can tolerate less sensitivity.

The sensor pattern for a paired sensor requires the interleaving of the two sensor inputs (see Figure 4).

FIGURE 4: SINGLE VERSUS PAIRED SENSOR PAD DESIGN

The interleaving of the sensors is required to keep the shift of both inputs as equal as possible, given the uncertainty of finger placement on the sensor. If possible, the areas of the two sensor elements should be equal, and approximately the size of ½ a single sensor. While this does increase the size of the sensor, it allows for more sensitivity on the paired sensor. Note the spacing between the two elements of a paired sensor should also be as large as possible to prevent interaction between the elements when the single sensors attached to each side are activated.

The decode logic for a shared button system starts by testing each frequency value against two touch thresholds, one for single sensor operation, and a smaller threshold for paired operation. The results of these tests are then passed through a search algorithm which checks for paired shifts, first, and then single shifts. If a paired shift is discovered, the button is considered detected and the single shift test is skipped. If a paired shift is not detected, then the single shift test is performed and any press conditions detected are reported. If more than two shifts beyond the paired threshold are detected, then a Fault condition is asserted and the decoding routine is terminated.

EXPANDING BY MULTIPLEXERS

Another option to expand the capacity for buttons is to use an external analog MUX, or several external MUXes. This is a very effective approach to handling very large numbers of buttons; it does not use any special tricks as does the 'paired-press' technique. However, it does come at the expense of more PCB surface area, and each MUX introduces capacitance, which reduces sensitivity. System-wide scan rate also slows as buttons are added. So, it is recommended to start with 1 MUX per comparator input channel, and then build and test progressively with each additional MUX. Chaining an unlimited number of MUXes together is prevented by increasing parasitic capacitance, because eventually, too much parasitic capacitance will make the additional change in capacitance from a finger press undetectable.

Handling an external MUX is much like handling the internal MUX to select which comparator input channel should be routed to the two comparators. Now, additional select channels which select the MUX line are external to the part, and so I/O pins must select the channel with the capacitive pad that the MUX connects, and internally the comparator input channel select bits must determine the appropriate channel to ensure a connection from the MUX's common line to the correct comparator input.

The schematic in Figure 5 shows how to connect external MUXes to a PIC microcontroller. The basic schematic is the same as in AN1101, "*Introduction to Capacitive Sensing*," but a pad passes through the MUX to its common line before connecting to the comparator inputs.

Note:	A suggested 8-channel analog MUX is a 74HC4051, and a 16-channel MUX, the 74HC4067.

Now, it is crucial to pay attention to how the indices of the buttons are assigned on both the MUX the button is on, and what comparator input channel that MUX's common line is tied to. The index is the variable described earlier to identify the appropriate button to scan and to cycle through all the buttons. A logical way to set up the button index values is as follows. Consider 32 buttons to be desired, 4 8-channel MUXes are required. On C12IN0-, place MUX0; this multiplexor will hold buttons 0 through 7. On C12IN1-, place MUX1 for buttons 8 through 15. On C12IN2-, place MUX2 for buttons 16-23, and likewise place MUX3 on C12IN3- for button indices 24-31.

The software to handle such a setup then is fairly simple to handle. If the index is 0-7, enable MUX0 and set the channel select bits to the index value $\in$ [0, 7]. If the index is greater than 7, the modulus of the index by 8 yields the channel select bits, and the MUX to enable is given by the integer division of index by 8. For example, assume that index = 21:

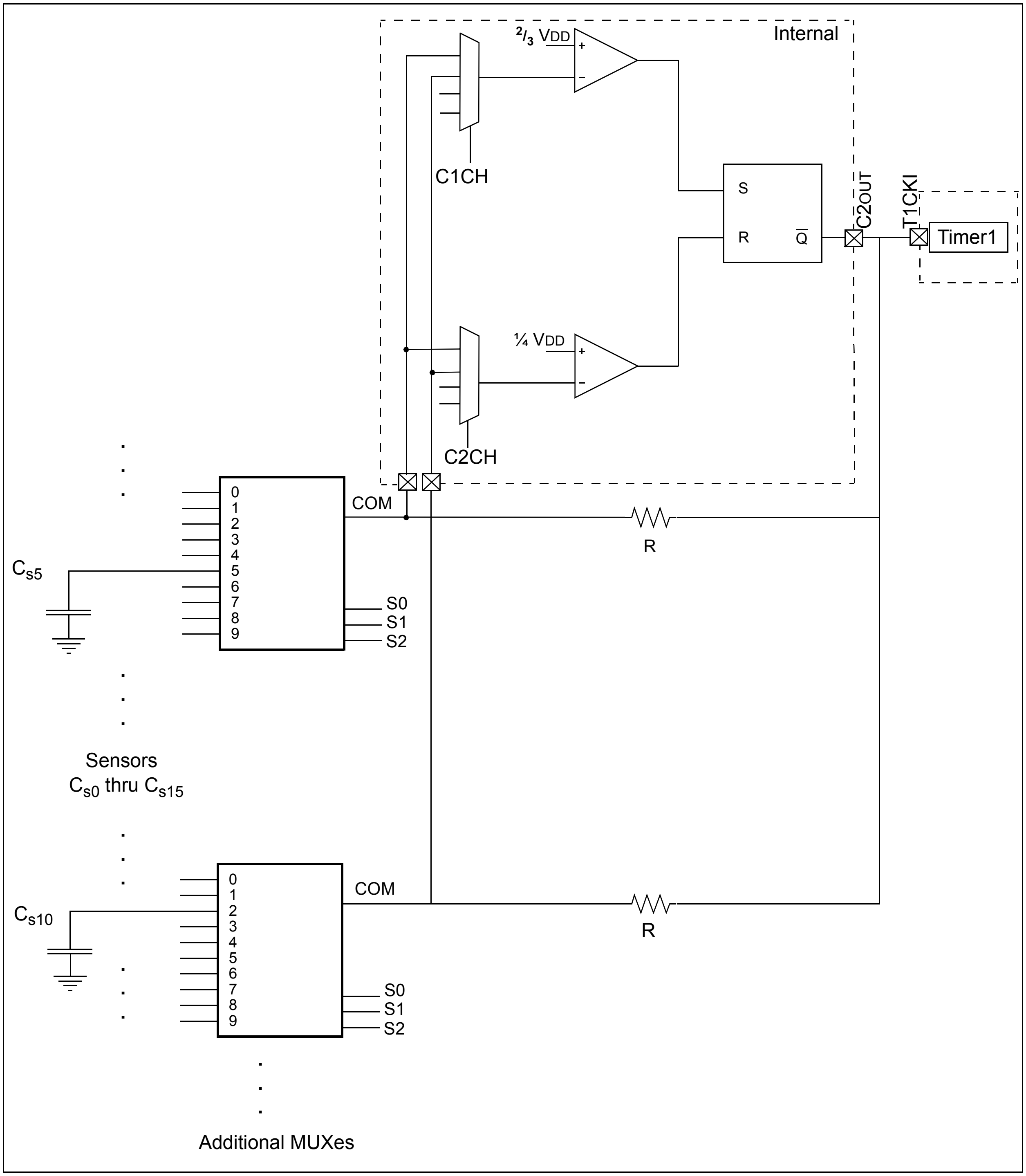
```
index % 8  = 21  %  8 =  5
index / 8  = 21  /  8 =  2
```

MUX channel select bits = 5 = $(101)_2$

MUX to enable = 2 = MUX2

To verify this is correct, look back at how the button indices were assigned. On C12IN2-, MUX2 holds buttons 16-23, and on that MUX the channel for 21 is the 6th channel (whose index is 5).

FIGURE 5: SCHEMATIC DRAWING OF CONNECTION PATH



In addition to enabling the external MUX and its control lines, the appropriate comparator input must be selected for the internal 4-channel MUX. Because of the definition of MUXx on C12INx- chosen before, the integer division index/8, conveniently also yields the comparator input channel bits to select for C1CH and C2CH. The following code example shows how to scan 32 buttons as in the described configuration.

EXAMPLE 3: CODE EXAMPLE

```
void isr {
    // If capacitive interrupt
    if (T0IF == 1) {
        // Grab TMR1 Reading
        StopTimers();
        GetMeasurement();

        // Test if Pressed
        if (IsButtonPressed(index))
            SetFlag(index);
        else
            ClearFlag(index);

        // Perform 16-point average
        PerformAverage(index);


        // Set next sensor
        // *** DIFFERENT THAN PREVIOUS ***

        // Increment index, wrap to 0
        if (index < 31)
            index++;
        else
            index = 0;

        div = index / 8;
        rem = index % 8;

        // Enable Correct MUX
        // (Assumes MUX EN active
        //  low lines on RB<7:4>)
        switch(div) {
            // Enable MUX 0, 1, 2, or 3
            // Disable all, then set correct mux
            // enabled, active low
            case 0:  PORTB |= 0xF0; RB4=0; break;
            case 1:  PORTB |= 0xF0; RB5=0; break;
            case 2:  PORTB |= 0xF0; RB6=0; break;
            case 3:  PORTB |= 0xF0; RB7=0; break;
            default: break;
        }

        // Set MUX Channel
        // (Assumes channel lines = RB<2:0>)
        PORTB &= 0xF8;          // Clear <2:0>
        PORTB |= rem;           // Set 3 LSb's

        // SetComparators
        CM1CON0 = COMP1[div];
        CM2CON0 = COMP2[div];

        RestartTimers();
    }
}
```

COMPARISONS

Each method to handle buttons has its own benefits and downsides. A table comparing some key traits of each method is below.

TABLE 2:

	Max Buttons	2 Buttons At Once	Scan Rate	I/O Lines
Stock	4	YES	++	+
Paired Press	10	NO	+	+
Multi-plexed	N*	YES	-	-

* Number of buttons may not be arbitrarily increased forever due to physical limitation of detection.

Completely stock use is great for applications requiring 4 buttons or less. Often applications requiring only 1 or 2 buttons will best be suited by using this hardware setup. It has the fastest scan rate potential and better distance sensing capability compared to MUXed inputs due to less parasitic capacitance.

Using a paired press technique is economical, when at most, 10 buttons are required and no two buttons need to be pressed simultaneously. Using paired presses requires no additional external parts, aside from extra traces and button pads. This simple hardware expansion requires more complex software.

Multiplexed button systems' largest benefits are the ability to have more than 10 buttons. It also allows that two buttons may be pressed simultaneously. However, the scan rate is tied to the number of inputs to be scanned. As the number of buttons increases, so will the scan rate, as an extreme number of buttons may become limiting. Changing Timer0's prescaler will speed up or slow down the scan rate, and may provide relief to long scan rates, but is also influential in the sensing process itself. With more and more MUXes, parasitic capacitance will eventually grow too large to detect a press, as described in the "Expanding by Multiplexers" section. So, there are several things preventing arbitrary increase of the number of buttons, but the number of capable buttons is easily greater than the stock or paired press techniques.

CONCLUSION

Controlling many capacitive buttons can be handled in a number of ways. The PIC16F616 family, PIC16F690 family and the PIC16F887 family all contain the capability for four buttons inherently, expandable to 10 buttons without external parts. For additional button capacity, external MUXes may be used.

The key sensing method is the same in the capacitive Interrupt Service Routine, but handling many buttons requires keeping track of which index represents which physical button correctly. Planning the software index to physical button relationship during the physical design process will help make the software portion of design easier to implement, write and read.

Other application notes of interest are *AN1101 "Introduction to Capacitive Sensing"*, *AN1102, "Layout And Physical Design Guidelines for Capacitive Sensing"*, and *AN1103 "Software Handling for Capacitive Sensing"*.

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

06/25/07