mTouch™ Conducted Noise Immunity Techniques for the CTMU

Author: Mihnea Rosu-Hamzescu
Microchip Technology Inc.

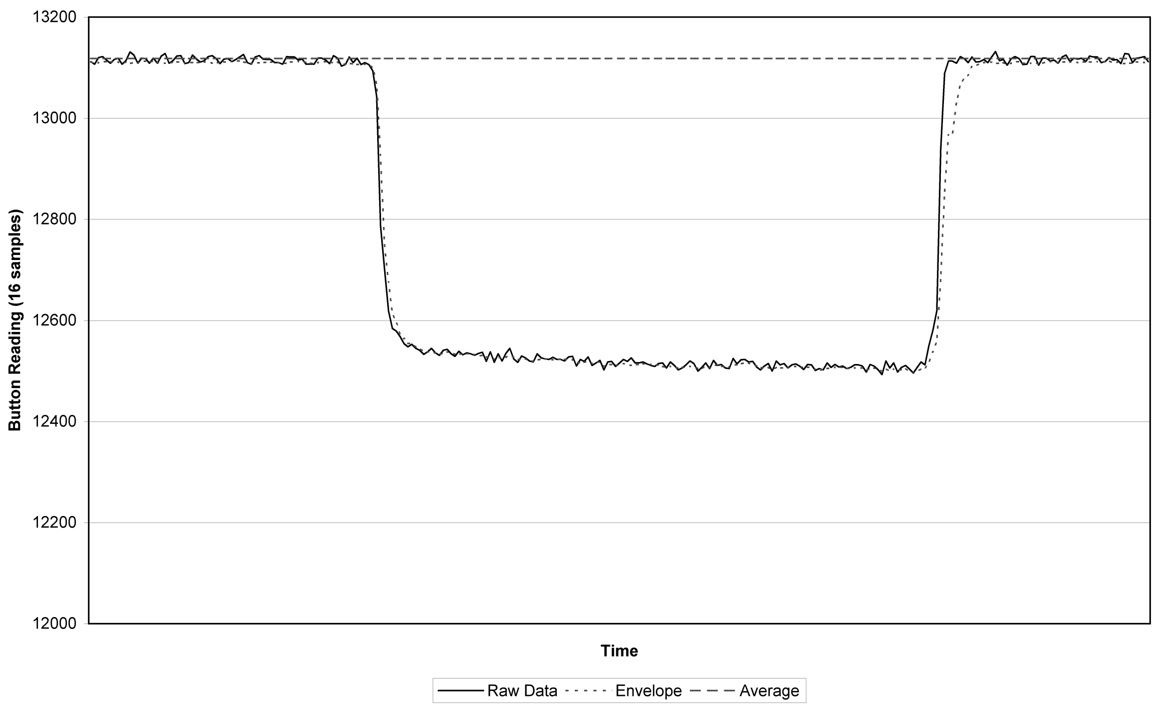
INTRODUCTION

This application note describes the use of special algorithms and techniques with Microchip's Charge Time Measurement Unit (CTMU) for capacitive touch applications in noisy environments. The CTMU is an excellent peripheral for use in touch sensing applications with many benefits such as high scanning speed, charge current trimming and low component count.

Before planning to use capacitive touch interfaces in industrial environments (or any other kind of environment with noisy power supply lines), the user must understand the hazards of conducted noise. Even if regulators keep a constant voltage drop at the circuit input, human interaction couples the noise into the touch pads. The effect on the button readings is clearly illustrated in Figure 2 and Figure 3. The severity of the problem changes depending on the frequency and noise amplitude. In many cases, noise can be countered by heavy signal filtering, but with very long response times. The rest of the cases will result in loss of functionality for the device because filtering is useless when the readings are centered on the average value, as shown in Figure 3.

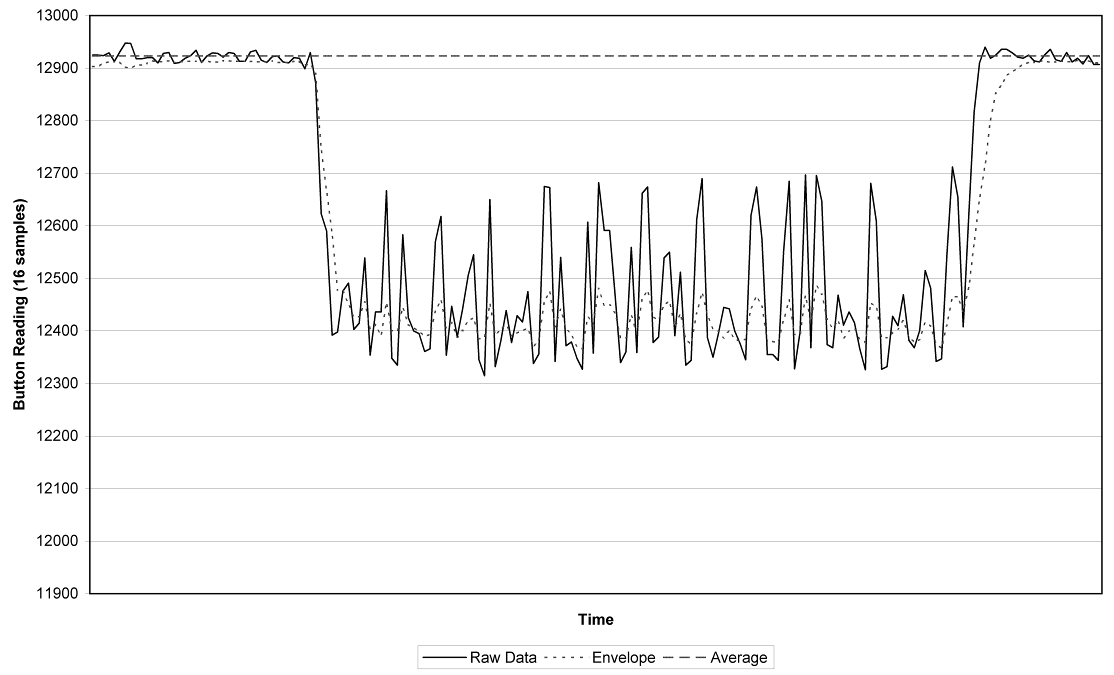
By using a combination of processing techniques (signal envelope and jittered sampling) and good PCB layout, it is possible to detect the capacitive touch keys reliably in the presence of conducted noise.

FIGURE 1: NORMAL OPERATION



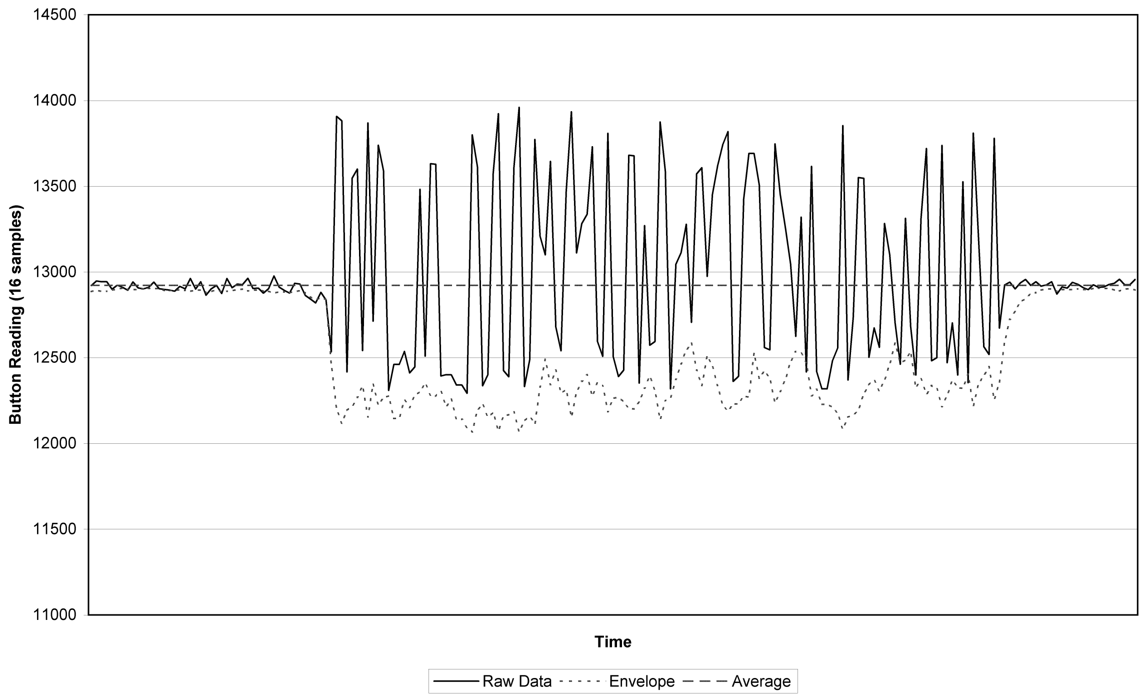
Normal device operation. Readings are steady and the envelope function follows very closely during a press.

FIGURE 2: MEDIUM NOISE SCENARIO



Medium noise scenario. Press detection would be possible without the envelope function using heavy low-pass filtering but with some loss in detection speed.

FIGURE 3: HIGH NOISE SCENARIO



A high noise scenario in which readings are centered on the average. Filtering is useless because the resulting signal will not show any usable deviation from the average. The envelope function quickly follows the peaks of the noise. Mirroring the values above the average keeps it steady enough to have reliable press detection.

© 2010 Microchip Technology Inc.

An envelope function (basically a peak detector) of the button readings is calculated for each channel and all decisions are made based on the difference between the envelope and the average derived from the raw data.

Under normal conditions, the envelope functions follow each button closely, allowing the same functionality. Under noise conditions, raw button readings look like white noise and the envelope function transforms that into a much steadier signal. Modulation in the noise amplitude can be sometimes seen as a low-frequency component in the readings because of under sampling. Since buttons are read in sequence the low frequency component is phase-shifted between them (different deviation from average), leading to problems like false triggering or no button detection. Jittering the sampling rate of the buttons solves this problem by distributing the modulation frequency component energy. The resulting signal looks like white noise and is very suitable for use with the envelope function.

Setting Up the CTMU

For a capacitive touch application, each sensor is connected directly to a channel of the A/D converter.

The CTMU is connected directly to the input of the A/D converter, allowing it to connect to any pin through the analog multiplexer. With this configuration, a single CTMU unit can measure a number of sensors equal to the A/D channels. For details of the CTMU features, please refer to the "*CTMU Family Reference Manual*" (DS39724).

The current source of the CTMU is available in three ranges: 0.55 µA, 5.5 µA and 55 µA. The current range selection is made in the CTMUICON register. For many capacitive touch applications, the highest current range setting (55 µA) works best. This allows for the quickest charging of the capacitive touch circuit. The CTMUICON register also has bits used to trim the current source in ±2% increments up to ±62% for each of the three current ranges. The CTMU current source is enabled and disabled using the software. Two control bits, EDG1STAT and EDG2STAT in the CTMU control register, determine if the current source is enabled. These bits are exclusively ORed. That is, if EDG1STAT and EDG2STAT are both set or cleared, the current source is off. If either bit is set while the other is cleared, the current source is enabled and charging the circuit.

The IDISSEN bit is enabled to drain charge from the A/D converter to insure the charging process begins at zero potential. If the bit is set, the circuit is connected to V_{SS} (grounded). Alternatively, the digital pin circuitry may be used to drain the pad charge very quickly or ground the other channels when they are not being read.

The CTMU is configured so that the external pins are not enabled and all control of the CTMU is handled through software. The A/D converter is also set up to do manual conversion.

For capacitive touch sensing, an absolute capacitance reading is not required, because all decoding decisions are related to the baseline readings.

Example 1 shows the CTMU set up, as described above.

EXAMPLE 1: SETTING UP THE CTMU

```
//configure ADC
AD1CON1bits.ADON        = 1;
    AD1CON1bits.ASAM    = 0;       //automatic sampling off; SAMP bit will start/stop sampling
    AD1CON3             = 0x1F01;
    AD1CHS              = 0x0000;
    AD1PCFGL            = 0xFFFF;  //configure analog channels, but pins are still set as digital
    AD1CON1bits.SAMP    = 1;       // output "0" to drain charge on pads faster than IDISSEN = 0

    //configure CTMU on highest current setting for best noise immunity
    CTMUICON    = 0x0300;
    _IDISSEN    = 0;
    _CTTRIG     = 0;//edge output disabled
    _EDGEN      = 0;//edges are blocked
    _EDG1POL    = 1;//select positive edge response
    _EDG2POL    = 1;
    _EDG1STAT   = 0;//edge bits will be set/cleared manually
    _EDG2STAT   = 0;
    _CTMUIF     = 0;
    _CTMUEN     = 1;//enable CTMU
```

INCREASING NOISE IMMUNITY

High immunity to conducted noise is not a requirement in most applications. Standard button decoding with some degree of debouncing and decent supply filtering is good enough. Unfortunately, this will not work when high frequency noise is coupled into the circuit through the power rails. While the circuit has no problem functioning correctly in these conditions, because voltage regulators maintain a stable difference between V_{DD} and GND, human interaction changes things quite a bit.

The user’s finger couples the power supply noise back into the capacitive buttons and severely compromises the readings. A standard algorithm will not be able to detect a valid touch or false triggers. For example, it is important to take into consideration readings above the average (apparently smaller capacitance). Also, using a precise timing scheme to get button readings might lead to under-sampling of noise frequencies, which causes further complication.

Electrical and electronic equipment immunity requirements to electromagnetic disturbances from RF transmitters are defined in the IEC 61000-4-6 standard. It specifically refers to equipment having at least one conducting cable which can couple the disturbing fields to the equipment. The standard establishes a common reference and a set of testing methods for evaluating electrical and electronic equipment functional immunity to conducted noise induced by electromagnetic fields. The range of frequencies tested is 150 kHz-80 MHz. Three levels of testing are defined, depending on the RMS of the disturbing signal.

The hardware and software presented in this application note have been tested in conditions similar to the IEC 61000-4-6 standard and have passed level 3 tests.

Board Layout and Functionality

A good layout for capacitive touch sense boards is critical in noisy environments. Traces from the microcontroller ADC input pins to the touch pads must be kept short, and have similar geometry. Vias are to be avoided, if possible.

A proper layout is even more important than a “noise resistant” firmware. There is no use having special firmware if a button is resonating at a certain noise frequency, because the trace is too long or is going in loops around the board. A good practice is to place the microcontroller as close as possible to the buttons and route the traces from the pads to the ADC channels first.

Connecting the ADC channels to the touch pads in order is irrelevant as it can be easily handled in the firmware using a descrambling array. It contains the list of ADC channels physically connected to the touch pads. The board presented in Figure 4 has the ADC channels connected in the order [5,4,3,2,1,0,6,7], because they were much more convenient to route this way.

Depending on the application, a small resistor (in the tens of ohms range) may be put in series between each ADC channel pin and the corresponding touch pad, limiting noise energy input into the ADC channel.

Even if it may decrease touch sensitivity to some degree, ground planes must be used on both sides of the board. It will reduce the effects of noise and the crosstalk between buttons.

© 2010 Microchip Technology Inc.

FIGURE 4: CMTU DEMO BOARD SCHEMATIC

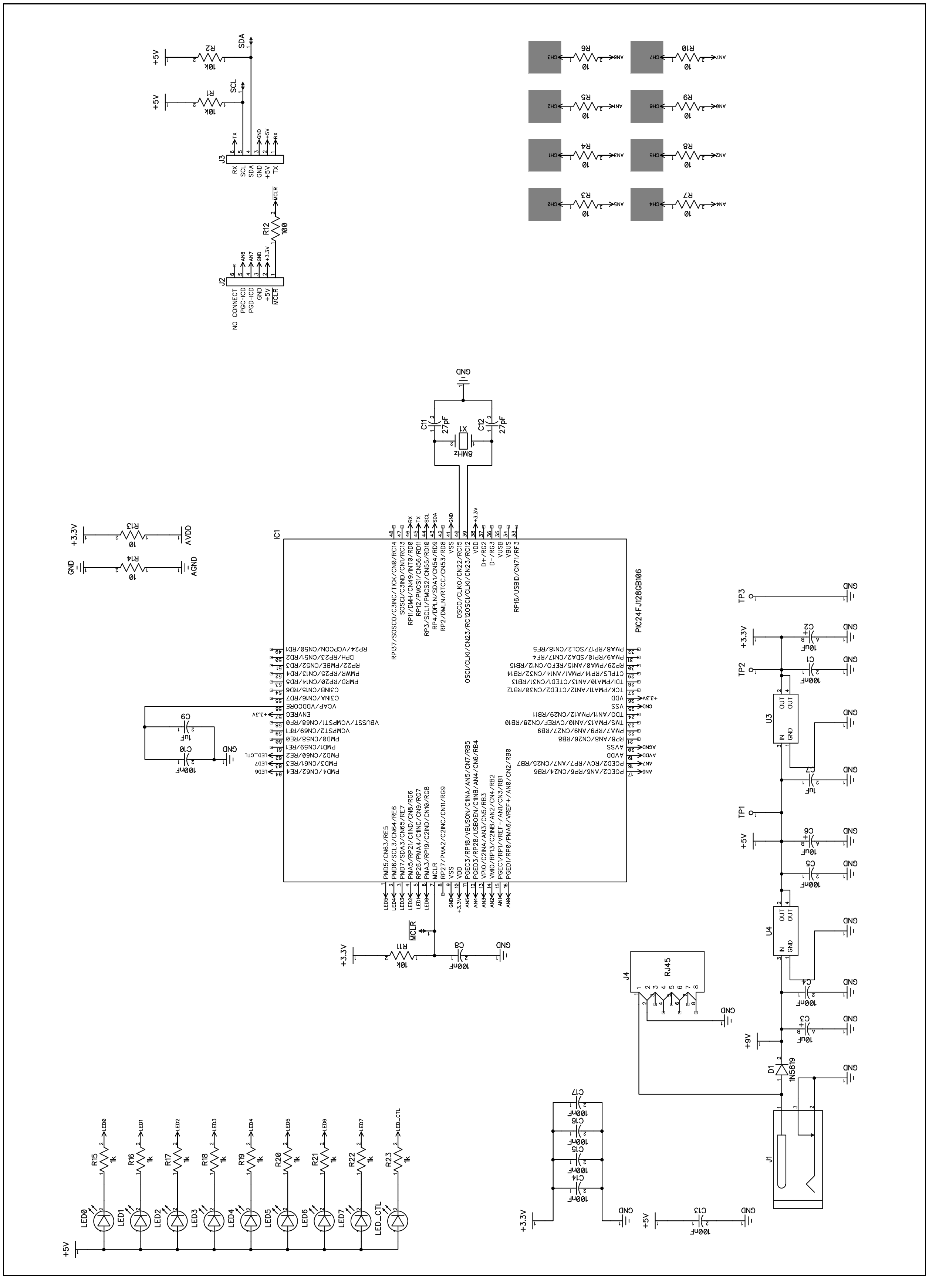
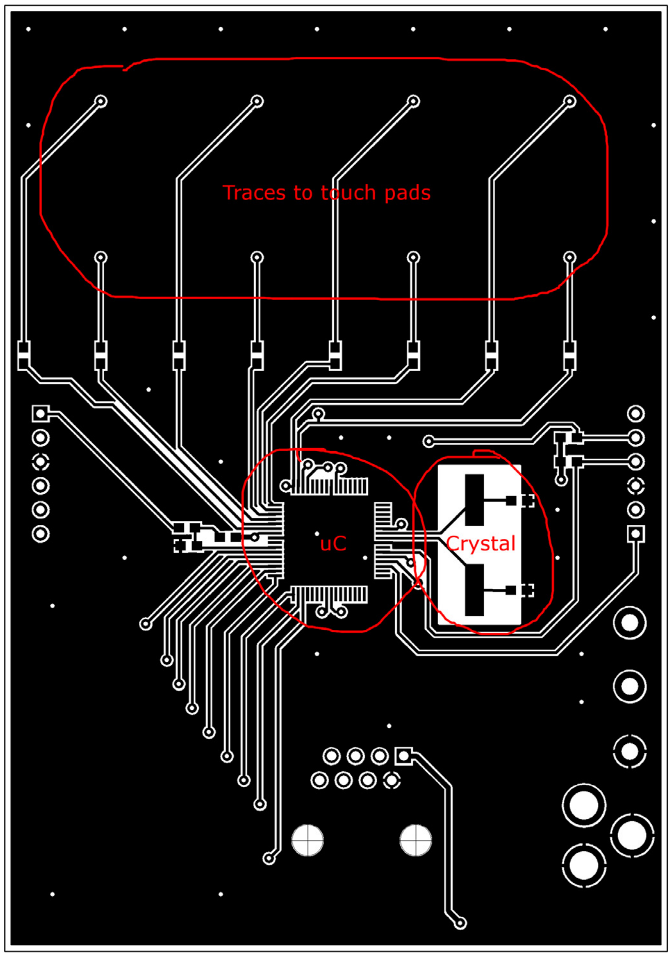
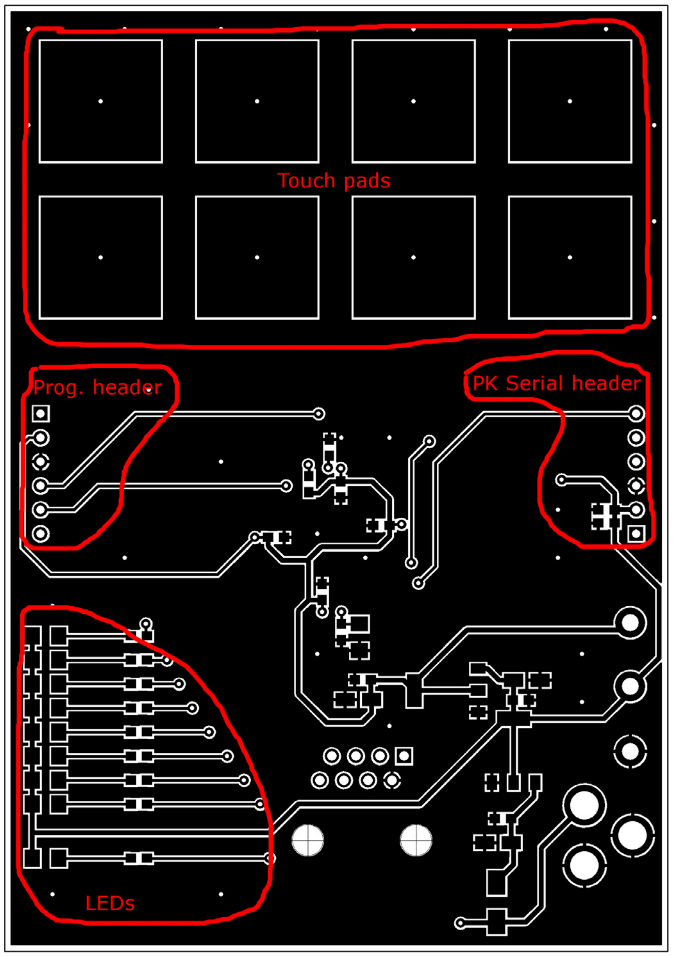


FIGURE 5: PCB LAYOUT (BOTTOM)



© 2010 Microchip Technology Inc.

FIGURE 6: PCB LAYOUT (TOP)



Trim Current Automatic Calibration

Different PCB layouts and trace geometry result in a different touch pad capacitance for each channel. Sometimes, the difference can go up to 20-30% and, if the charge currents are left untrimmed, then the readings for each channel also vary by this amount. This makes it difficult to establish a baseline and a trip value when using a single fixed value for all buttons.

Fortunately, the solution is very simple and the application can calibrate the trim currents for each pad/channel automatically. It is also important to remember that the current source needs a bit of voltage drop to maintain linearity, therefore, it is strongly recommended not to exceed 90% of VDD for the baseline value. Most applications use 80% as a baseline value and trim the currents to get readings very close to this value.

Example code is shown in Example 2.

EXAMPLE 2: CURRENT TRIMMING PROCEDURE

```
for (index = 0; index < CHANNELS; index++)
    {
        for (trimbits = 0x21; trimbits < 0x40; trimbits++)  //start with negative trim values
        {
            trim[index] = trimbits;
            sum          = 0;

            //average 64 samples for a better precision
            for (avnum = 0; avnum < 64; avnum++) sum += ReadButton(index);

            sum /= 64;
            if (sum > CHAN_VAL) break; //stop if the preset 800 (of 1024) value is reached
        }

if (trimbits == 0x40)//use positive trim values if nominal value not reached yet
        for (trimbits = 0x00; trimbits < 0x20; trimbits++)
             {
                 trim[index] = trimbits;
                 sum          = 0;

            //average 64 samples for a better precision
                 for(avnum = 0; avnum < 64; avnum++) sum += ReadButton(index);

                 sum /= 64;
                 if (sum > CHAN_VAL) break; //stop if the preset 800 (of 1024) value is reached
             }

    }
```

The calibration function starts with the lower (negative) trim bits values and averages 64 readings to make the measurement more precise. The trim bits value is increased until the readings are over 800 units (out of 1024 on a 10-bit ADC).

This procedure may be repeated every time the system is powered up or the trim values may be stored into some type of nonvolatile memory after calibration.

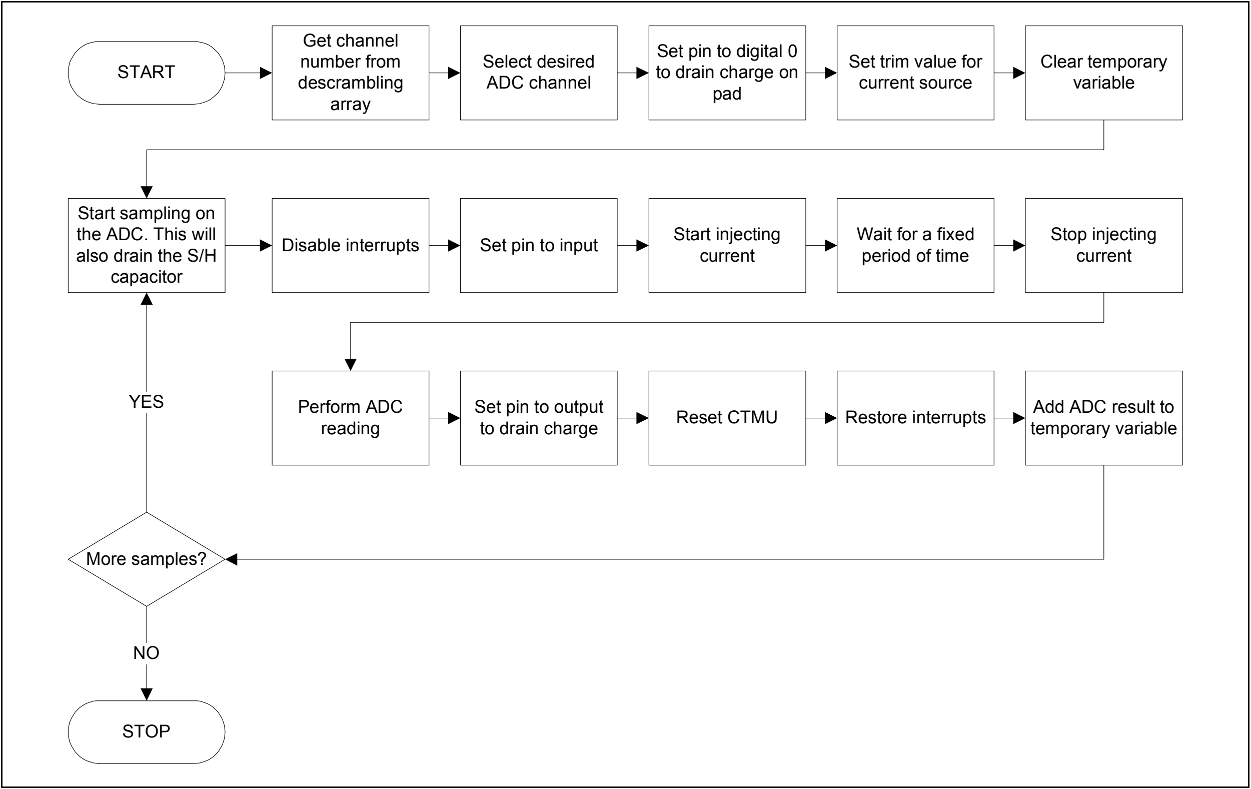
Reading the Sensors

The software routine for reading the touch sensors has a very simple sequence of steps (see also the diagram in Figure 7):

1. Get the real analog channel value from a descrambling array. This can be simply declared as a constant in the firmware. This is related to the fact that the ADC channels are not routed to the pads in order.
2. Produce a bit mask for the desired channel and apply it to the analog/digital functionality select register.
3. Drain all pads by setting pins to 0 digital output.
4. Set the trim value for the current source to the calibrated value for that channel.
5. Clear the temporary variable used for averaging readings.
6. Start sampling on the ADC to allow charging of the internal capacitor and the touch pad at the same time.
7. If interrupts are used, set the processor priority to max (on PIC24) or disable interrupts, to avoid timing problems.
8. Set pin (corresponding to analog channel) to input, allowing ADC to sample voltage on pad.
9. Start injecting current into pad.
10. Wait for a fixed period of time (in this case, 2.0 µS). The charge time should be long enough to allow the pads with higher capacitance to be charged. Of course, shorter time means faster reading.
11. Stop injecting current.
12. Get ADC reading.
13. Set pin (corresponding to analog channel) to output, draining charge on pad almost instantly. The IDISSEN bit can be used for the same purpose, but it is significantly slower. The digital port circuitry can sink up to 25 mA and will do it in 1-2 instruction cycles.
14. If interrupts are used, restore them or set processor priority to normal.
15. Reset the CTMU for a new charging sequence.
16. Add up the ADC readings into a temporary variable.
17. Repeat from step 6 if multiple samples are required.

© 2010 Microchip Technology Inc.

FIGURE 7: READING THE TOUCH SENSORS


START
Get channel number from descrambling array
Select desired ADC channel
Set pin to digital 0 to drain charge on pad
Set trim value for current source
Clear temporary variable
Start sampling on the ADC. This will also drain the S/H capacitor
Disable interrupts
Set pin to input
Start injecting current
Wait for a fixed period of time
Stop injecting current
Perform ADC reading
Set pin to output to drain charge
Reset CTMU
Restore interrupts
Add ADC result to temporary variable
YES
More samples?
NO
STOP

EXAMPLE 3: READING THE TOUCH SENSORS

```
channel=chanord[channel];//get proper channel value from "descrambling" list

    chanmask=(1<<channel);//bitmask for analog/digital functionality
    AD1PCFGL^=chanmask;
    AD1CHS=channel;//set ADC channel

    TRISB=0x0000;

    _ITRIM=trim[index];//set current trim for channel to calibration value

    temp=0;
    for(index=0;index<READINGS;index++)
    {
        _SAMP=1;//turn on ADC sampling
        _IPL=0b111;//set processor priority to maximum so we don't get interrupts during 
measurement
        TRISB^=chanmask;//set the desired channel to input mode
        _EDG1STAT=1;//start injecting current

        NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); //32 NOP = 2.0us @ 16MIPS
        NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); NOP();
        NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); NOP();
        NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); NOP(); NOP();

        _EDG2STAT=1;//stop injecting current
        _SAMP=0;

        while(!_DONE);//get ADC reading

        TRISB^=chanmask;//drain charge on sensor

        _CTMUEN=0;
        _EDG1STAT=0;
        _EDG2STAT=0;
        _CTMUIF=0;
        _CTMUEN=1;//reset CTMU

        _IPL=0b000;//restore processor priority
        temp+=ADC1BUF0;//add up readings
    }
    AD1PCFGL^=chanmask;
```

Touch Decoding Routine

Decoding a valid touch in a noisy environment requires significantly more data conditioning. Relying too much on the noise particularities (noise detectors) might impair device functionality in normal conditions.

After reading the value on a particular channel, the data arrays are tested to determine if they are initialized (first acquisition). Usually, a null value means that there were no prior readings. If low pass filters are used on the data, starting from zero means that some values will take a while (number of samples in this case) to reach their proper level. Avoiding incorrect button press decisions is done by either ignoring a number of initial acquisitions (warm-up time), or by initializing the arrays with values based on the instantaneous button readings.

Low-pass filters used in this firmware are very simple and use only additions/subtractions and bit shifting. Coefficients are all based on the power of two values making the implementation possible even on microcontrollers without hardware multiplication capabilities.

EQUATION 1: DISCRETE LOW-PASS FILTER RECURRENCE

$$Y[i] = Y[i-1] + \alpha * (X[i] - Y[i-1])$$
$$0 \le \alpha \le 1$$

Where X is the input sequence, Y is the output sequence and α is the filter coefficient.

© 2010 Microchip Technology Inc.

We can substitute α by an integer number and scale the coefficients by 2^n. The coefficient resolution depends on "n" (the number of bits used). Smaller coefficients translate into slower updating filters.

On microcontrollers without hardware multiplication, the coefficient "α" can be selected to always have the value 1 and only modify the number of bits used for slower moving filters. This helps with the calculation, since it only uses addition, subtraction and bit shifting.

EQUATION 2: LOW-PASS FILTER WITH QUANTIZED COEFFICIENTS

$$a = \alpha * 2^n ; \alpha = \frac{a}{2^n}$$

$$Y[i] = Y[i-1] + \frac{a*(X[i] - Y[i-1])}{2^n}$$

Dividing by the power of two (bit shifting) means that some lower order bits are discarded every time. The average might not reach the instantaneous reading value in this case. The difference depends on the filter coefficients. To counter this issue, the filtered value may be stored on a higher number of bits. The value is shifted left (bigger) and the lower order bits are used as remainder bits. They eventually add up after a number of operations and the filtered value follows the instantaneous value correctly. The downside is that operations with 16-bit or 32-bit integers are required.

EQUATION 3: LOW-PASS FILTER WITH IMPROVED ACCURACY

$$Y_{BIG}[i] = Y[i] * 2^n$$

$$X_{BIG}[i] = X[i] * 2^n$$

$$Y_{BIG}[i] = Y_{BIG}[i-1] + \frac{a*(X_{BIG}[i] - Y_{BIG}[i-1])}{2^n}$$

$$Y_{BIG}[i] = Y_{BIG}[i-1] + a*(X[i] - Y[i-1])$$

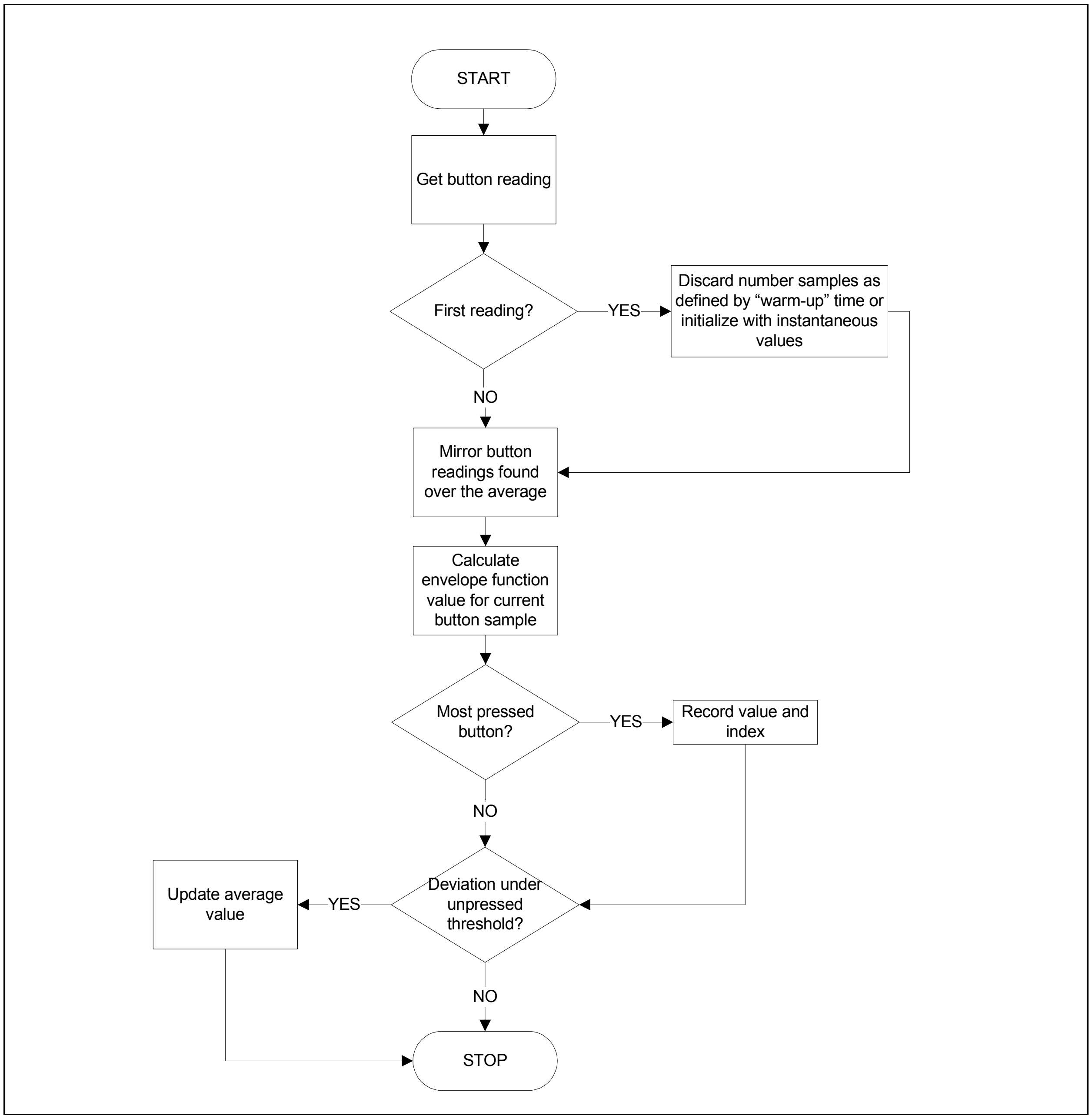
If the deviation while pressing a button is quite large compared to the difference between the instantaneous value and the calculated average (or speed is essential), it can be left alone or a smaller number of bits may used for the division remainder.

In most noise scenarios, button readings are very unpredictable because the acquisition frequency is many orders of magnitude smaller than the noise frequency. One of the working solutions for this problem is to use an envelope detector on the button data, and then use that envelope to decode a touch. Values above and below the average need to be used (see high noise scenario in Figure 3). To simplify things, the values above the average are mirrored instead of calculating two envelopes. Because the number of noise spikes below the average is practically doubled this way, it is much easier to keep the envelope level steady. The envelope function is essentially a filter with different coefficients for attack and decay. If the button value is below the envelope value, then the envelope drops quickly to follow the value. If the button value is over the envelope, then the envelope rises slowly. This way we obtain a sequence of values which is significantly more stable than the button readings and can be used in decoding a touch.

The touch decoding routine needs updated envelope and average values for all channels. They are calculated in a simple sequence of steps. Example code is shown in Example 4 and the flowchart in Figure 8:

1. Read the button value.
2. If warm-up time is not used, then check for null values in the filtered button, average and envelope arrays and initialize them as needed.
3. If the button data filtering is used, calculate the value.
4. Mirror button values above the average (center on the average).
5. Calculate the envelope value for the current button sample.
6. Check if the current button is the most pressed. Record value and index.
7. If the button value is below the depress threshold, update the average value. This means that the average will not move in most noise scenarios.
8. Repeat from step one for each channel, as needed.
9. If all channels have been read, start the decoding routine.

FIGURE 8: ENVELOPE AND AVERAGE CALCULATION


START
Get button reading
First reading?
YES
Discard number samples as defined by "warm-up" time or initialize with instantaneous values
NO
Mirror button readings found over the average
Calculate envelope function value for current button sample
Most pressed button?
YES
Record value and index
NO
Deviation under unpressed threshold?
YES
Update average value
NO
STOP

© 2010 Microchip Technology Inc.

EXAMPLE 4: ENVELOPE AND AVERAGE CALCULATION

```
bt  = ReadButton(index, READINGS);
//first acquisition needs to initialize average and envelope to the instantaneous button
value
//you can do this or start from 0 and have a "warmup time”
if (!bigraw[index]) bigraw[index]   = bt << RAW_BITS;
if (!raw[index]) raw[index]      = bt;

//filter raw button data if needed (for this application there is no raw data filtering)
bigraw[index] += (INT32) bt - (bigraw[index] >> RAW_BITS);
raw[index]= bigraw[index] >> RAW_BITS;
if (!average[index]) average[index] = ((INT32) bt << AVG_BITS);

temp = (average[index] >> AVG_BITS);

if (!envelope[index]) envelope[index] = temp;

bt  = temp - raw[index];      //determine if raw value is over the average
if (bt < 0) bt = temp + bt;   // and mirror it
    else bt = raw[index];

if (bt < envelope[index])
    envelope[index] += ((INT32) bt-envelope[index]) >> ENV_ATTACK;
else if (envelope[index] < temp)
    envelope[index] += ((INT32) bt-envelope[index]) >> ENV_DECAY;

temp = temp - envelope[index];
if (temp < 0) temp=0;    // no negative values allowed
deviation[index] = temp;
if (maxpress < temp)    //determine the "most pressed" button
{
    maxindex  = index;
    maxpress = temp;
}
//update average only if deviation is less than the unpressed threshold
if (temp < DEPRESS_THR)
            average[index] += (INT32) raw[index] - (average[index] >> AVG_BITS);
```

When a touch is present, the button readings affected by noise show a much higher deviation compared to normal conditions. This would help with touch detection unless the board layout has problems. Crosstalk between channels will result in higher deviations on adjacent buttons and some of them might give false triggers.

Solving this problem requires limiting the number of button presses that can be detected at the same time to one. The first step would be to determine the button that has the highest envelope deviation from the average value. This only works if the background noise readings with no press keep the deviation below the press threshold.

Some noise frequencies will boost readings over the press threshold on all buttons (even one is enough) and the one having the highest deviation will be incorrectly flagged as pressed. In this case, the difference between the most pressed button and the rest is small and might not even pass a debouncing test with a long enough sequence. Again, big differences between trace length and geometry on the board might lead to some channels constantly having a higher deviation compared to the others. In this case, the debouncing test won’t protect against false triggers.

This leads to the introduction of another test before deciding whether it’s a press or not. Since the difference between channels in the absence of a touch is usually small, we can compare this difference to a threshold. For example, if the second highest deviation is less than 50% of the maximum, it can be safely decided that it was a press. Of course, the highest value must remain on the same channel all through the debouncing test to get a valid touch.

If a button passes the percentage test and is above the fixed press threshold, then the firmware starts debouncing up to a valid press. If no press has been validated yet, the firmware resets all debouncing counters to 0, except for the most pressed button. This way a button may only be validated if it maintains this “most pressed” status for a number of times in a row larger than the debouncing threshold.

If a button is already flagged as a valid press, but a different one has the highest deviation, we have two choices. We either force the most pressed button index to the flagged button index and still increment the debouncing counter, or simply start debouncing the flagged button down to 0. The first choice helps avoid flickering under heavy noise conditions, but also makes pressing two buttons in a quick succession harder.

On the other hand, if a button does not pass the percentage test or is under the depress threshold, then its debouncing counter is decremented. When the counter reaches 0, the button is no longer set as pressed.

EXAMPLE 5: DECODING PROCEDURE

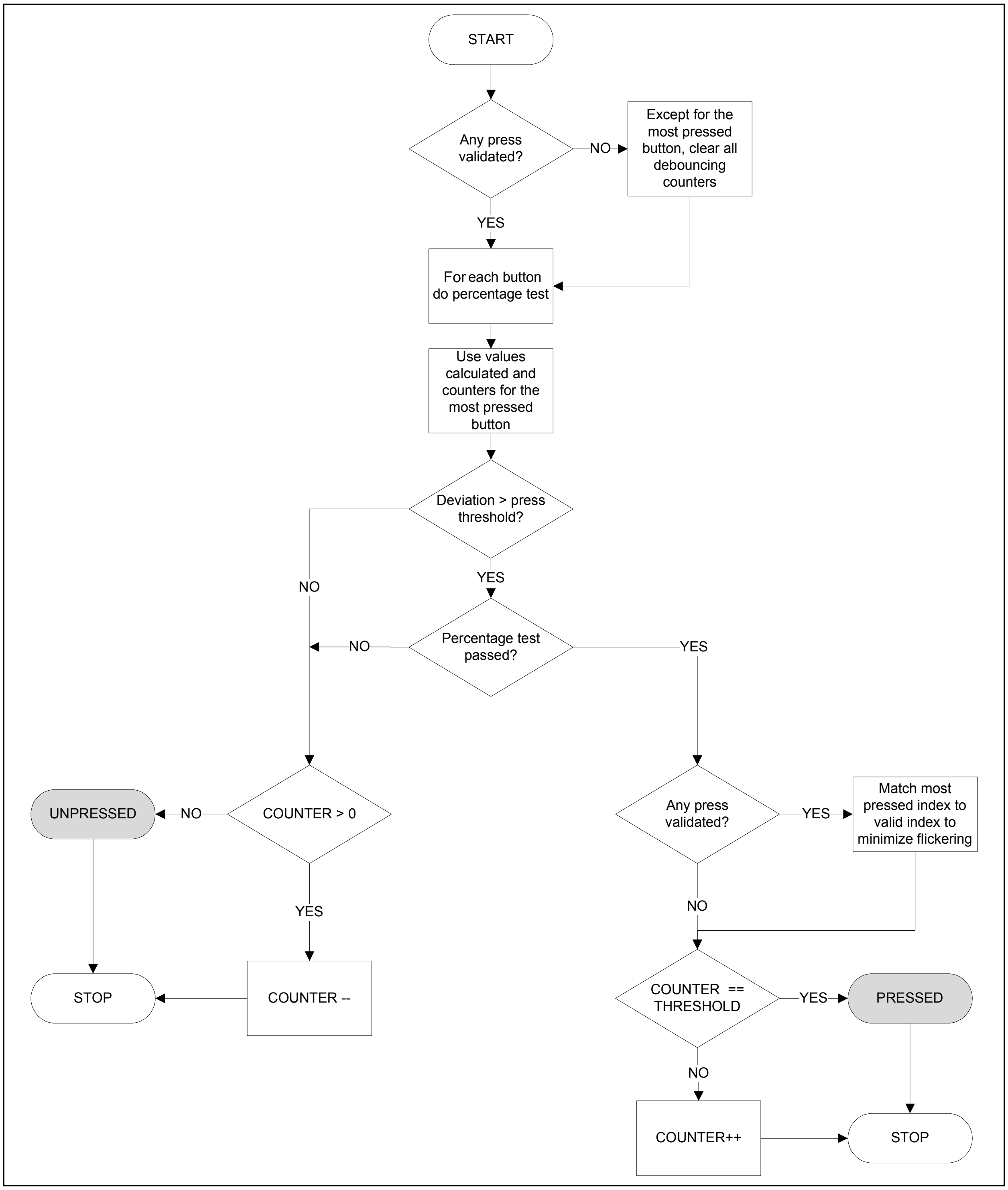
```
//if there is no valid press clear all button debounce counters except the one with the highest
deviation
if (!bpress)
        for (index = 0; index < CHANNELS; index++)
        {
            if (index != maxindex) db[index]=0;
        }

    //see if any button has a deviation higher than a percentage of the most pressed one
    //this is useful in noise conditions to eliminate false presses
    temp = maxpress * WINNER_PCT/100;
    for (index = 0; index < CHANNELS; index++)
    {
         if (index != maxindex && deviation[index] > temp) break;
    }

    if (maxpress > PRESS_THR && index == CHANNELS) //the percentage test passed
    {
        if (!bpress) //no valid press yet
        {
            if (db[maxindex] == PRESS_DEBOUNCE)
            //debounce test passed, set button as pressed
            {
                       bpress  = (1 << maxindex);
                       ldb     = maxindex;
            } else
                       if (db[maxindex] < PRESS_DEBOUNCE) db[maxindex]++;
                       //increment debounce counter for button

        } else //there is a valid press
        {
            //keep the most pressed button to avoid flickering and increment debounce counter
            maxindex  = ldb;
            if (db[maxindex] < PRESS_DEBOUNCE) db[maxindex]++;
        }
    }
    //if most pressed button under depress threshold or percentage test failed
    //start debouncing down
else if (maxpress < DEPRESS_THR || index < CHANNELS)
    {
        //it is possible to set a lower debounce down value for a quicker release
        if (db[maxindex] > DEPRESS_DEBOUNCE)
            db[maxindex] = DEPRESS_DEBOUNCE;
        if (db[maxindex] == 0) bpress = 0;//clear button if counter is 0
        else if (db[maxindex]) db[maxindex]--;//or decrement down to 0
    }
```

© 2010 Microchip Technology Inc.

A typical flow in the decoding procedure has the following steps (see also the diagram in Figure 9):

1. Read the button value.
2. If warm-up time is not used, then check for null values in the filtered button, average and envelope arrays and initialize them as needed.
3. If the button data filtering is used, calculate the value.
4. Mirror button values above the average (center on the average).
5. Calculate the envelope value for the current button sample.
6. Check if the current button is the most pressed. Record value and index.
7. If the button value is below the unpressed threshold, update the average value. This means that the average will not move in most noise scenarios.
8. Repeat from step one for each channel, as needed.
9. If all channels have been read, start the decoding routine from step 10.
10. If there is no press validated yet, clear all debouncing counters except for the most pressed button.
11. Do the percentage test.
12. If the button value is above the press threshold and has passed the percentage test, continue. Otherwise, go to step 15.
13. Check the debouncing counter on the most pressed button. If the debouncing threshold is not reached, increment counter, otherwise set the valid press flag for that button. For better stability, but longer response times, the most pressed button index can be forced to the valid press index (if one exists).
14. Exit the decoding routine (button PRESSED branch).
15. Percentage test failed or the button value is below the depress threshold.
16. If counter is not zero, decrement, otherwise unset the valid press flag.
17. Exit the decoding routine (button UNPRESSED branch).

FIGURE 9: DECODING PROCEDURE


START
Any press validated?
NO
Except for the most pressed button, clear all debouncing counters
YES
For each button do percentage test
Use values calculated and counters for the most pressed button
Deviation > press threshold?
NO
YES
Percentage test passed?
NO
YES
UNPRESSED
NO
COUNTER > 0
YES
STOP
COUNTER --
Any press validated?
YES
Match most pressed index to valid index to minimize flickering
NO
COUNTER == THRESHOLD
YES
PRESSED
NO
COUNTER++
STOP

© 2010 Microchip Technology Inc.

ALGORITHM INTEGRATION

Another important issue is the integration of the described algorithm in different applications. The user might need USB communication or to control an LCD/TFT. Processor usage is not a problem, since the code presented in this note uses only 17% of the processor time on a PIC24F running at 16 MIPS. The difficulty is usually represented by the timing constraints and the critical code sequences that must not be interrupted.

The code attached to this note has all the reading and decoding placed in the Timer1 interrupt routine. Each channel is sampled at a certain interval and, after all of them have been read, the decoding routine is called.

While there are no strict timing constraints regarding the sample rate, it must be kept around 100 samples per second or the filter and envelope function coefficients need to be adjusted.

Custom code can be easily inserted in the main loop or in the other interrupt functions. There are no critical code sections, except the CTMU charging time of about 2-3 µS. This section must run for an exact number of instructions or the readings will be compromised. For this reason, the code sets the processor priority to maximum before the section, making sure no interrupt can be served. After getting the ADC reading, the processor priority is restored to normal.

TABLE 1: PROCESSOR USAGE VS. SAMPLE RATE FOR 8 BUTTON BOARD AT 16 MIPS

Samples Per Second* (all channels)	Samples Per Button	Total ADC samples	T1PERIOD (500ns tick)	Processor Usage Estimation	Detection Speed
570 (max)	16	72960	N/A	100%	Very fast
500	16	64000	4000	88%	Very fast
400	16	51200	5000	70%	Very fast
250	16	32000	8000	44%	Fast
200	16	25600	10000	35%	Fast
100	16	12800	20000	17%	Medium

The jittering algorithm adds a random value to the base timer period, slightly decreasing the sample rate. The actual number of ADC samples taken each pass is calculated by multiplying the number of buttons by the number of samples per button.

TABLE 2: PROGRAM AND DATA MEMORY USAGE ESTIMATION

Buttons	Program Memory (words)	Data Memory (bytes)
16	1480	280
12	1480	220
8	1480	150
4	1480	90
2	1480	60

Data memory usage may be further decreased depending on the number of samples taken per channel and the envelope function coefficients by replacing the 32-bit integer variables with 16-bit integers.

CONCLUSION

The algorithm presented in this application note solves most of the issues resulting from conducted noise in a CTMU-based touch application. The implementation uses a low amount of processor time and is easily customized for the user application.

REFERENCES

http://www.microchip.com/mTouch

AN1250, *"Microchip CTMU for Capacitive Touch Applications"*

"Section 11. Charge Time Measurement Unit (CTMU)" in the *"PIC24F Family Reference Manual"* (DS39724)

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.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE**.** Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

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

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC^{32} logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Octopus, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2010, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.

ISBN: 978-1-60932-191-8

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.

MICROCHIP

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

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

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 - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

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

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

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

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

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

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 - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

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: 886-3-6578-300
Fax: 886-3-6578-370

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

01/05/10

© 2010 Microchip Technology Inc.