MSP430 Advanced Technical Conference 2006

Hands-on: Capacitive Touch Sensing with MSP430

Steve Underwood
MSP430 FAE Asia
Texas Instruments

Agenda

- **How can we sense a finger on a control?**
- **How can we capacitively sense a finger?**
- **The detailed issues in charge rate sensing**
- **Experimenting with some code**
  - LAB1-3: Basic touch pad detection – detecting the center pad of the “4” on the ATC board
  - LAB4: Achieving very low power consumption
  - LAB5: Extending a pad to a slider – detecting around the whole “4”

Strategies for detecting a finger

- **The disturbance of acoustic waves across the surface of the touch pad**
  - Dirt resistant, but can use a lot of energy
- **The disturbance of optical beams over the touch pad**
  - Dirt resistant, but hard to make compact. Good for information booths
- **Resistive coupling between conductive pads**
  - Cheap, simple, low power. Wear and tear, and ESD problems
- **Resistive changes between layers caused by pressure**
  - The usual technique for PDAs. Can require significant pressure
- **Capacitance changes due to the finger**
  - Several variants of this technique exist
  - Low cost and power. Can achieve fine positional detection, with a light touch

Agenda

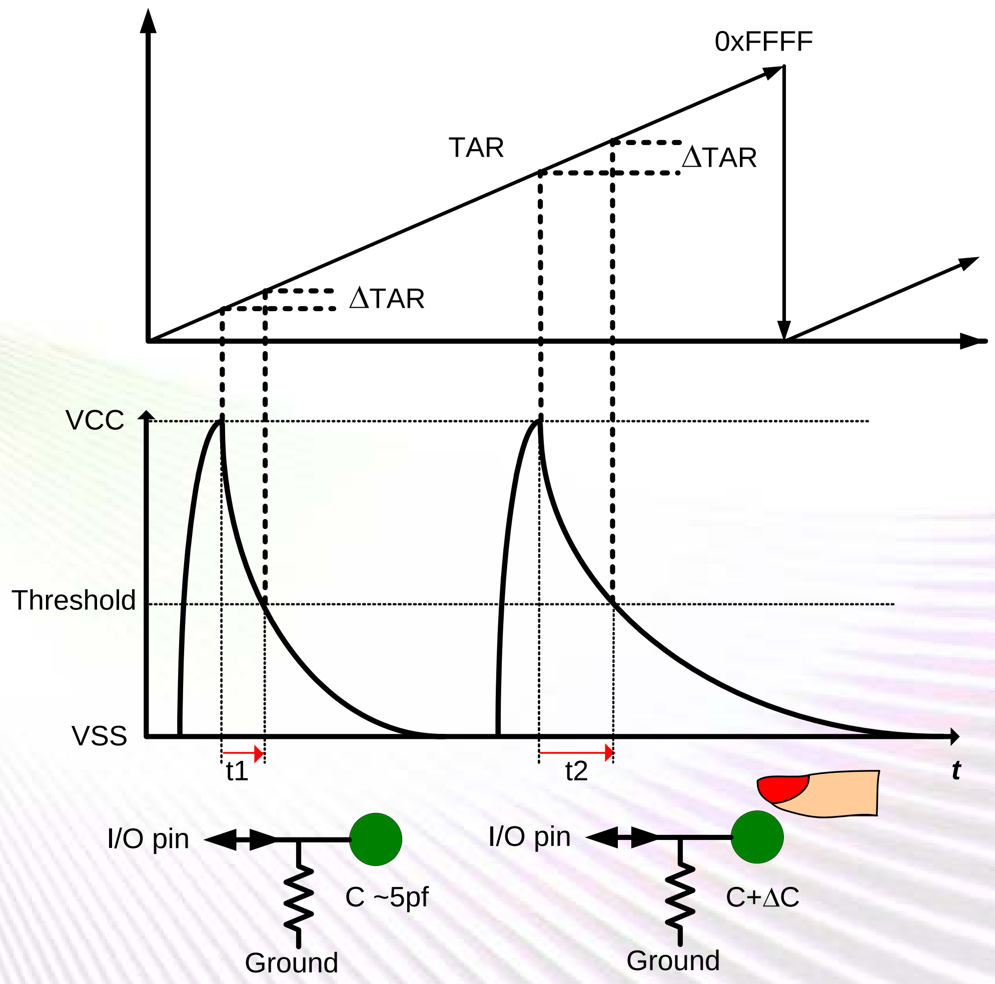
- **How can we sense a finger on a control?**
- **How can we capacitively sense a finger?**
- **The detailed issues in charge rate sensing**
- **Experimenting with some code**
  - LAB1-3: Basic touch pad detection – detecting the center pad of the “4” on the ATC board
  - LAB4: Achieving very low power consumption
  - LAB5: Extending a pad to a slider – detecting around the whole “4”

Capacitive ways to detect a finger

- **A number of strategies for detecting the capacitive effect of a finger's presence have been used:**
  - Make the touch pad a tuning element in an oscillator, and detect the change in frequency as a finger approaches
  - Various charge transfer techniques, some of which resemble a sigma-delta ADC
  - Measure the charge/discharge time of an RC or current source and capacitor circuit, and look for changes as a finger approaches

Using P1 and 2 to sense a touch pad

- **Each P1 or P2 pin can generate an interrupt**
- **MSP430 digital I/O pins have <50nA leakage**
- **Provides up to 16 charge/discharge sensors which can reliably detect with charge currents <<1uA**
  - Low power consumption
  - Long discharge time for good resolution
- **We can free run Timer A or B, and snapshot it on a pin interrupt**



Agenda

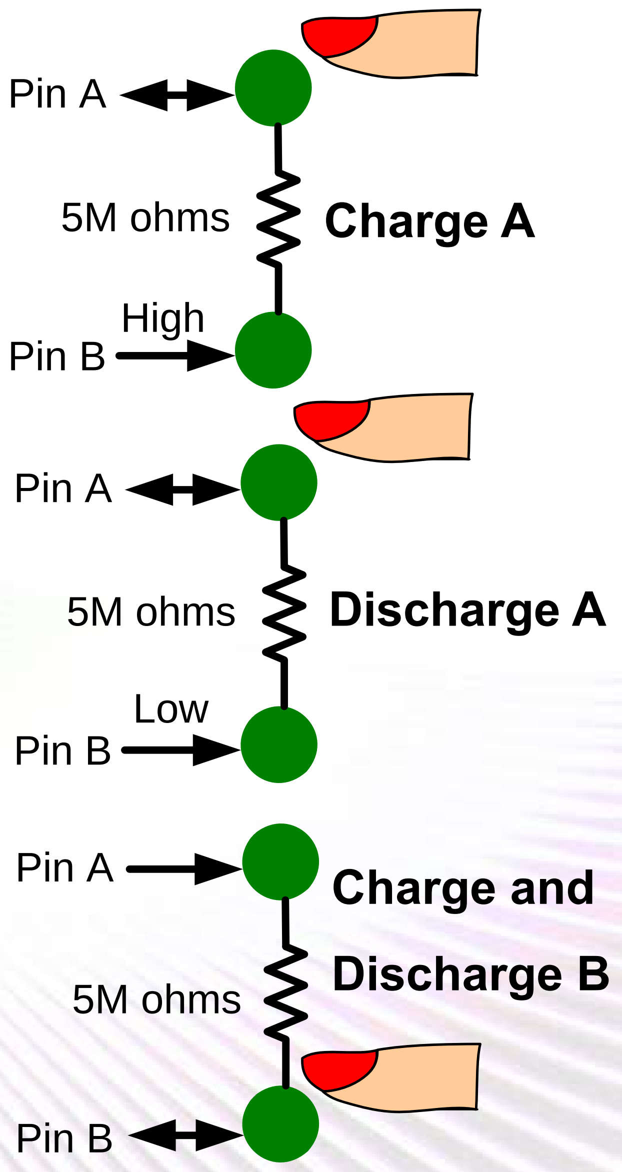
- **How can we sense a finger on a control?**
- **How can we capacitively sense a finger?**
- **The detailed issues in charge rate sensing**
- **Experimenting with some code**
  - LAB1-3: Basic touch pad detection – detecting the center pad of the “4” on the ATC board
  - LAB4: Achieving very low power consumption
  - LAB5: Extending a pad to a slider – detecting around the whole “4”

50Hz/60Hz pickup

- Touching a high impedance oscilloscope probe produces a large mains waveform on the display
- The same effect happens with a touch pad. Mains pickup from touching or waving cables near the pad must be tolerated
- Heavy filtering of touch pad measurements would make the pad unresponsive
- If we measure the charge time, quickly followed by the discharge time, their average is only slightly affected by mains pickup
- Lightweight digital filtering can clean up the remainder of the mains effect. This gives a clean, responsive, measurement of the effects of a finger

Mains tolerant sensing

- **Use 2 I/O pins for 2 pads**
- **5M ohms between pairs**
- **Time charging of a pad**
  - Set pin B to output high
  - Pin A to output low, to discharge
  - Pin A to input, and time the charge
- **Immediately time discharging of that pad**
  - Set pin B to output low
  - Pin A to output high, to charge
  - Pin A to input, and time the discharge
- **Sum the charge and discharge times, use a single pole IIR filter to smooth the results**
- **Swap use of the pins, sense the other pad**



Other sources of interference

- **Tolerance of other internal and external sources of interference may be important**
- **Cellular phones may be a problem**
- **Interference from fast logic in the product itself may be an issue**
  - A clean power supply for the MSP430 is important
  - A stable supply for the MSP430 is important
  - Good screening and ground plane layout on the sensing PCB is important

Labs ahead – let’s get prepared

- The ATC board contains an MSP430FG4619, which has an LCD and an isolated RS232C port. We will use this MCU to receive key information by I2C, display it and also pass it to a PC, by RS232C – you will need a working serial port on your PC for this
- The board contains an MSP430F2013, which connects to the 16 pads which form the large “4”. It also connects by I2C to the MSP430FG4619
  - Eight I/O pins are used for the pads – 2 pads to each I/O pin
  - Four 5.1Mohm resistors are connected between pairs of I/O pins
- Header H1 must have only jumpers 1-2 and 3-4 fitted
- Jumper JP2 must be removed, so LED3 does not affect touch pad sensing

Agenda

- **How can we sense a finger on a control?**
- **How can we capacitively sense a finger?**
- **The detailed issues in charge rate sensing**
- **Experimenting with some code**
  - LAB1-3: Basic touch pad detection – detecting the center pad of the “4” on the ATC board
  - LAB4: Achieving very low power consumption
  - LAB5: Extending a pad to a slider – detecting around the whole “4”

A basic touch button

- We will now look at some core sensing code, implementing the basic technique we have looked at
- In this initial exercise we will only sense the big square pad at the centre of the “4”
- The provided code for the MSP430FG4619 can be compiled, loaded into the device and used without change
- The provided code for the MSP430F2013 will be used as the basis for some experiments
- After this we will extend the sensing technique to more complex requirements

The elements of sensing a key…

```
unsigned int measure_key_capacitance(void)
{
    int sum;
    /* All keys are driven low, forming a "ground" area */
    P1OUT |= BIT1;                /* Charge the key */
    _NOP(); _NOP(); _NOP();       /* Allow time to charge */
    P1IES |= BIT1;                /* Configure a port interrupt */
    P1IE |= BIT1;                /* as the pin discharges */
    P1DIR &= ~BIT1;
    timer_count = TAR;            /* Snapshot the timer... */
    LPM0;                         /* ...and wait for interrupt */
    P1IE &= ~BIT1;                /* Disable key interrupts */
    P1OUT &= ~BIT1;               /* Discharge the key */
    P1DIR |= BIT1;
    sum = timer_count;           /* Record the discharge time */
    ...
```

…The elements of sensing a key…

```
    P1OUT |= BIT0;             /* Prepare for charging */
    _NOP(); _NOP(); _NOP();    /* Allow time to discharge */
    P1IES &= ~BIT1;            /* Configure a port interrupt */
    P1IE |= BIT1;              /* as the pin charges */
    P1DIR &= ~BIT1;
    timer_count = TAR;         /* Snapshot the timer... */
    LPM0;                      /* ...and wait for interrupt. */
    P1IE &= ~BIT1;             /* Disable key interrupts */
    P1OUT &= ~(BIT1 | BIT0);   /* “Ground" the key */
    P1DIR |= BIT1;
    sum += timer_count;
    /* The sum of the two readings is our answer */
    return sum;
}
```

…The elements of sensing a key

```
#pragma vector=PORT1_VECTOR
__interrupt void port_1_interrupt(void)
{
    P1IFG = 0; /* Clear port interrupt flags */
    timer_count = TAR - timer_count; /* Record time */
    LPM0_EXIT;
}
```

```
{
   /* Filtering of individual readings, using a
      single pole IIR low pass filter. */
   margin = measure_key_capacitance() - base_capacitance;
   filtered += (margin - (filtered >> 4));
}
```

Lab 1 – getting started

- **Load IAR workspace "MSP430 Touch Sensing"**
- **Find project "FG4619 host_comms" for the 'FG4619**
  - Open, compile, and load it into the MSP430FG4619 on your board
  - No changes are needed
- **Launch Hyperterminal on your PC**
  - Configure the COM port and set the configuration to 115200 bps with no parity
- **Find project "F2013 touchbutton" for the 'F2013**
  - This is the code we will experiment with, and extend
  - Open, compile, and load it into the MCU on your board
  - Run it and look at the response on the LCD (0 to 255) and your PC (a response waveform), as you touch the square pad of the "4"
  - When this is working we will try some experiments
  - Huge responses mean metal contact – put tape over the touch pad

The effect of insulators

- **The ATC board only separates your finger from the capacitor plates by a thin film of solder resist**
- **A real product will require an insulating layer – typically the plastic case**
- **Insulator material matters**
  - It is the dielectric in a capacitor
  - Use of non-hygroscopic glues is especially important
  - Stable attachment, with no air gaps is important
- **Insulator thickness has a big impact on the response from a pad**
- **Insulator thickness varies considerably**
  - 0.5mm of textured plastic on the touchpad of your notebook
  - 10mm thick glass panel on a kitchen appliance
  - 1 to 2mm of plastic is more common

Lab 2 – the effects of insulators

- Try holding a layer of insulation over the pad, and see the effect
- Try different thicknesses, and look at the effect
- You may need to raise the sensitivity of the display
  - Reduce or remove the shift of the values to the host
  - Remove “ID_3” to stop prescaling Timer A

```
    scan_key();
    to_host = filtered >> 4;
```

```
    TACTL = TASSEL_2 | MC_2 | ID_3;
```

Adapting to environmental change

- **We must calibrate each pad at startup**
  - Components vary
  - Mechanical assembly (glue, etc.) varies
- **We need to dynamically adapt that calibration**
  - Capacitance and the input threshold vary with temperature
  - Supply voltage and other parameters may drift
- **We need to avoid adapting to the wrong thing**
  - Hands waving near the touch pad should not cause the threshold to rise unduly
  - Obviously we should not adapt when a finger is detected
  - We should freeze adaptation of all keys, when any one is detecting
  - Adapt slowly, as real changes occur slowly
  - Adapting downwards faster than we adapt up helps tolerate waving hands
- **A fixed threshold above the base capacitance works OK for touch switches**

Lab 3 – a single touch switch

- **The code already measures and sets an initial “base” capacitance, at startup**
- **1. Try to make it implement a fixed threshold for key detection – i.e. a switch**
  - You can simply implement a fixed threshold that suits the tolerances of your own board, by running and measuring the readings
  - A value between 0 and 255 is being sent to the MSP430FG4619, so try to change this to 0 for off and 255 for on
- **2. Try to adapt the base capacitance dynamically to respond to changes in temperature, etc**
  - What would be a good adaptation scheme?
  - Should you adapt faster when the signal is farther from its base level?
  - How much faster should you adapt downwards than you adapt upwards?

Disturbances from other I/O

- **We have seen that practical insulation cover can make the response from a finger very small**
- **The thickness of the PCB also affects the response**
  - Thin PCBs in hand-held products give a smaller response
  - The increasing popularity of very thin flexible PCBs for small consumer products made things even harder
- **Noise from other I/O on the MCU can disturb results**
- **The most stable results are achieved if I/O only occurs between scans of the touch pad**

Agenda

- **How can we sense a finger on a control?**
- **How can we capacitively sense a finger?**
- **The detailed issues in charge rate sensing**
- **Experimenting with some code**
  - LAB1-3: Basic touch pad detection – detecting the center pad of the “4” on the ATC board
  - LAB4: Achieving very low power consumption
  - LAB5: Extending a pad to a slider – detecting around the whole “4”

Minimizing power consumption

- **Many touch pad applications are in portable appliances, so power consumption is important**
- **We charge and discharge in LPM0 – tens of µA**
- **We can rest between scans in LPM3 – <1 µA**
- **Average consumption can be very low**
- **Rapidly scanning sliders, for good response takes <<1mA overall**
  - We can keep the MCU in the LPM0 state for much of the time
- **Scanning a single pad at a low rate we can achieve 5uA consumption**
  - We can use a low scan rate waiting for the “on” switch to be pressed
  - We don’t need a strong response, as we are not interpolating

Lab 4 – low power operation

- **Minimize power consumption as we wait for a single key to be pressed**
  - Like a real product, saving power, waiting for the “on” button to be pressed
- **Try to adapt the code from the last exercise to minimize its power consumption**
  - The code currently idles in a loop between key scans
  - Try to make it sleep in LPM3 during these idle periods
  - Interrupts can be generated from the watchdog, for simplicity
  - There is no 32kHz crystal for the MSP430F2013 on this board, but there is the VLO. It runs at ~12kHz, and is operating as the ACLK source in this code

Agenda

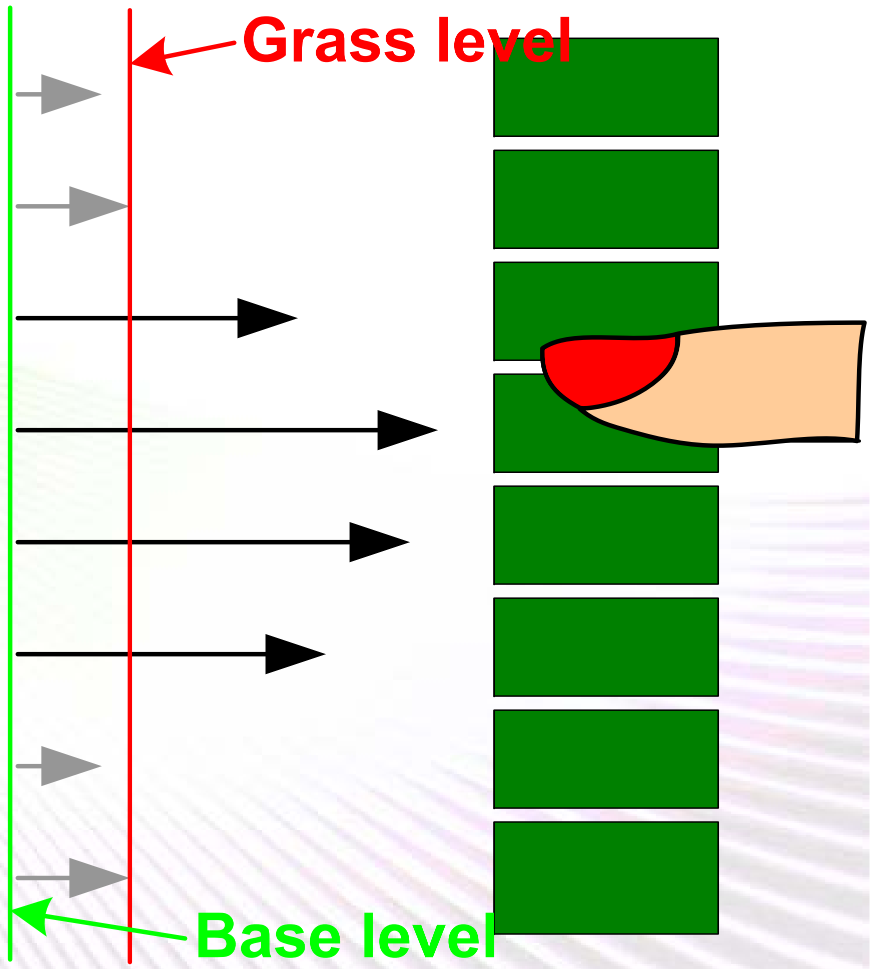
- **How can we sense a finger on a control?**
- **How can we capacitively sense a finger?**
- **The detailed issues in charge rate sensing**
- **Experimenting with some code**
    - LAB1-3: Basic touch pad detection – detecting the center pad of the “4” on the ATC board
    - LAB4: Achieving very low power consumption
    - LAB5: Extending a pad to a slider – detecting around the whole “4”

Extending a button to a slider

- **People tend to be most interested in touch pads for sliders, dials, and other complex configurations**
- **The benefits of tactile feedback reduce the acceptability of individual touch buttons for some applications**
- **Fine resolution sliders can be made with a row of just a few individual pads, using interpolation**
  - Thick insulation would make this harder
- **Usable pad size is related to the size of fingers**
  - A finger needs to excite multiple pads
  - A compromise size can work well for children and large adults
  - The “4” has oversized pads at the extreme top and left, which compromise its performance – **this is a demo, not an end product**

Interpolating slider positions…

- **How can we interpolate for high resolution sliders?**
  - The base reading is large, and varies from pad to pad, so we work with "pad response = current reading – base reading"
  - Perform a linear weighted average of the pad responses
  - "Grass cut" weak responses. These can bias the weighted average
  - There are issues at the ends of rows, because there are no further pads on one side of the peak for a properly weighed calculation
  - Interpolating to 1/16th of a pad is practical
  - There are issues at the ends of rows, because there are no further pads on one side of the peak for a properly weighed calculation

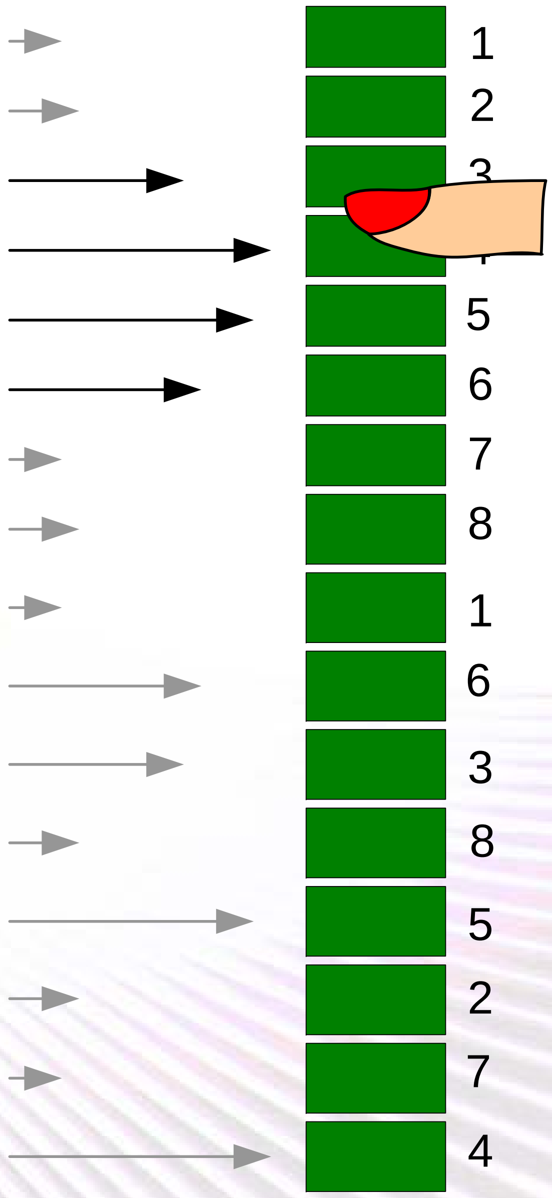


…Interpolating slider positions

- **What will affect the stability of the positional estimate?**
  - Noise will cause some jitteriness in the positional estimate
  - Fingers still move a little while “stationary”
  - As a finger lifts, we need to avoid generating false steps
- **How can we smooth out these effects?**
  - Most applications are looking for movement along a slider, rather than absolute position
  - Use hysteresis in accepting apparent changes of finger direction
  - Up to half a pad of hysteresis is needed for good results as a finger lifts
  - Soft flesh flexes as we change direction, so there is a natural dead zone, regardless of hysteresis we introduce. The final feeling is quite natural

Sliders with less I/O pins

- **10 pads makes a good long slider**
- **For dial type structures more pads – say 16 – may be better**
- **With care we can double up the I/O pins, and use the pattern of responses to work out the real position of the finger**
- **The "4" shape we are experimenting with is like this**
  - 16 pads, connected to 8 I/O lines
  - The large pads at the extreme top and left compromise the results
  - More regular shapes work well



Compact multi-key code…

- Use array of structures in flash to describe each key
- Use array of structures in RAM for their working data

```
/* This defines the static data to maintain one key */
typedef struct
{   unsigned char port;
    unsigned char port_bit;
    int threshold;
} key_config_data_t;
```

```
/* This defines the working data to maintain one key */
typedef struct
{   int base_capacitance;
    int filtered;
} key_data_t;
```

…Compact multi-key code…

```
unsigned int measure_key_capacitance(int key_no)
{
    char active_key;
    const key_line_config_data_t *keyp;
    const key_line_config_data_t *partner;
    int sum;
    keyp = &key_line_config[key_no];
    partner = &key_line_config[key_no ^ 1];
    active_key = keyp->port_bit;
    if (keyp->port == 1)
        P1OUT |= active_key;
    else
        P2OUT |= active_key;
    _NOP();
    _NOP();
    _NOP();
    ……
```

…Compact multi-key code…

```
int scan_keys(void)
{
    int i;
    int margin;

    for (i = 0;  i < NUM_LINES;  i++)
    {
        margin = measure_key_capacitance(i)
               - key_line[i].base_capacitance;
        key_line[i].filtered
            += (margin - (key_line[i].filtered >> 4));
    }
    return 0;
}
```

…Compact multi-key code…

```
int find_finger_position(void)
{
    int i, j, k, l, min, max, max_pos;
    long int a, b;
    /* Find the min and max responses for the key lines */
    min = 32767; max = -32768; max_pos = -1;
    for (i = 0;  i < NUM_KEYS;  i++)
    {
        if (key_line[i].filtered < min)
            min = key_line[i].filtered;
        if (key_line[i].filtered > max)
        {
            max = key_line[i].filtered;
            max_pos = i;
        }
    }
    /* If max response isn't that big, no finger present. */
    if (max < 100) return -1;
```

…Compact multi-key code…

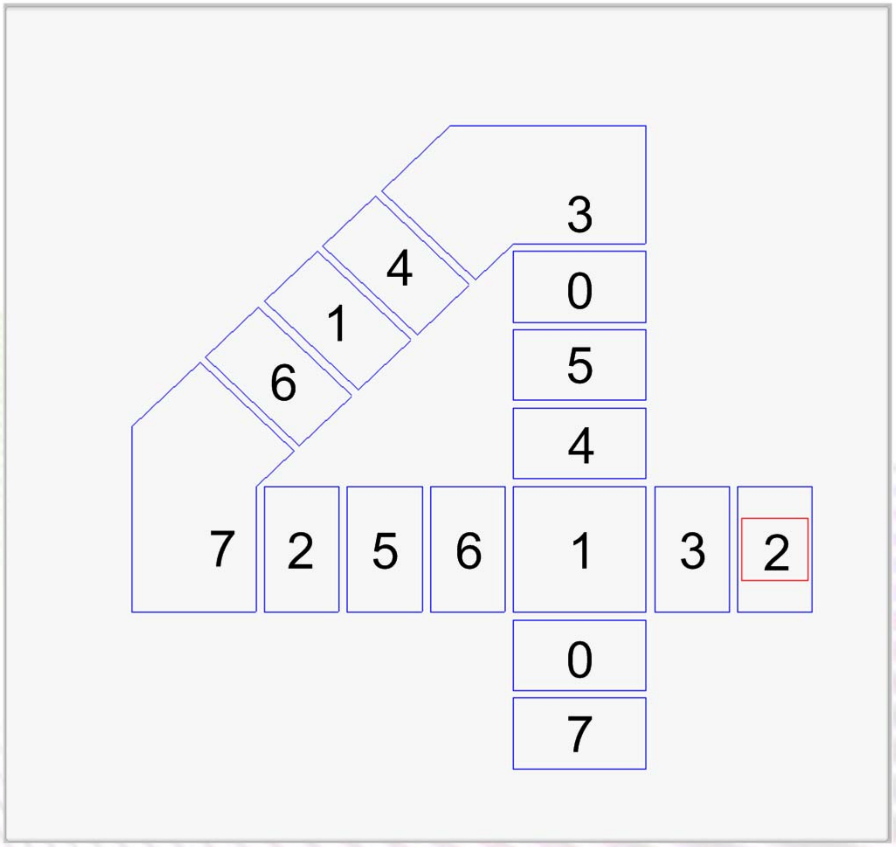
```
    grass_level = (max - min) >> 3;
    a = 0;
    b = 0;
    for (i = 0;  i <= NUM_KEYS;  i++)
    {
        if ((key_line[i].filtered - min) > grass_level)
        {
            a += (key_line[i].filtered - min);
            b += (i + 1)*(key_line[i].filtered - min);
        }
    }
    b /= (a >> 4); /* Calculate average, in 1/16th steps */
    b -= 16;       /* Compensate for adding 1 in the loop */
    return b;
}
```

Lab 5 – a touch slider

- **We will sweep through all the pads of the "4", and estimate the finger position**
- **Launch Hyperterminal on your PC**
  - Configure the COM port and set the configuration to 115200 bps with no parity
  - See the data output on the hyperterminal screen
- **Find project "F2013 touchpad" for the 'F2013**
  - Editing the previous 2013 to work with many pads would take too long
  - This project has the bulk of the editing done for you
  - It uses similar code to that in the previous slides, further extended to deal with the 2 keys per I/O configuration
  - It implements the weighted averaging scheme

Lab 5 – a touch slider

- The picture shows the allocation of I/O lines
- Heavy I/O reuse creates compromises in this complex shape
- When the finger moves across the central pad, it must be tracked to get the movement correct
- Fine progressive tracking of the finger can be seen in the 3 straight runs of pads



Summary

- For requirements up to 16 sense lines, the charge/discharge approach to capacitive sensing works very well with the MSP430
- Very low power consumption is possible
- Simple switches can be implemented with thick insulation
- Interpolation allows fine resolution sliders to be implemented through the typical plastic shell of a handheld device