Modular Capacitive Touchpads Project Proposal

Aaron Pfitzenmaier, Daniel Sheen

1 Overview

We will interface blocks of 16 capacitive touch sensors with the Nexys 4 FPGA. Each block will have 4 ports to which other blocks of 16 sensors or the FPGA can be connected to. The FPGA will be physically connected to only one block and communicate over I2C to all of them. The FPGA will then process the inputs from all sensors and display the layout of the blocks and the status of the sensors on a monitor. Communication between adjacent blocks will also aid the FPGA in determining the overall layout. We will also use the sensors to control various outputs.

This sensor system can be used for custom control systems, for instance, for embedded touch sensing in work surfaces, or for basic projects without requiring complicated or expensive capacitive touch sensor hardware. Essentially, we're hoping it will be a nice tool for FPGA based tinkering projects going forward, since it will provide a simple modular platform for creating and using relatively arbitrary size and shape arrays of sensors for user interfaces.

Extensive test benches will be written to test all modules prior to integration into the system.

2 Design

2.1 Goals and Scope

2.1.1 Baseline Goals

2.1.1.1 At least one working touch sensor block with 16 capacitive sensors.

2.1.1.2 Control and reading of block sensors by the FPGA.

2.1.1.2(a) I2C commands to switch sense line connections.

2.1.1.2(b) FPGA frequency/capacitance sensing.

2.1.1.3 FPGA outputs image of which sensors are pressed to the display.

2.1.1.4 programmable control of other FPGA outputs.

2.1.2 Expected Goals

2.1.2.1 Multiple working touch sensor blocks.

2.1.2.2 FPGA scans I2C address space to detect connected sensors at startup.

2.1.2.3 FPGA automatically determines the layout of sensors at startup, and displays it on screen.

2.1.2.4 Sensors are correctly mapped to an X,Y coordinate designation for use for IO functions (ie, keyboard, tracking, anything arbitrary you might want to do with it)

2.1.3 Stretch Goals

2.1.3.1 Hot swappable sensor blocks

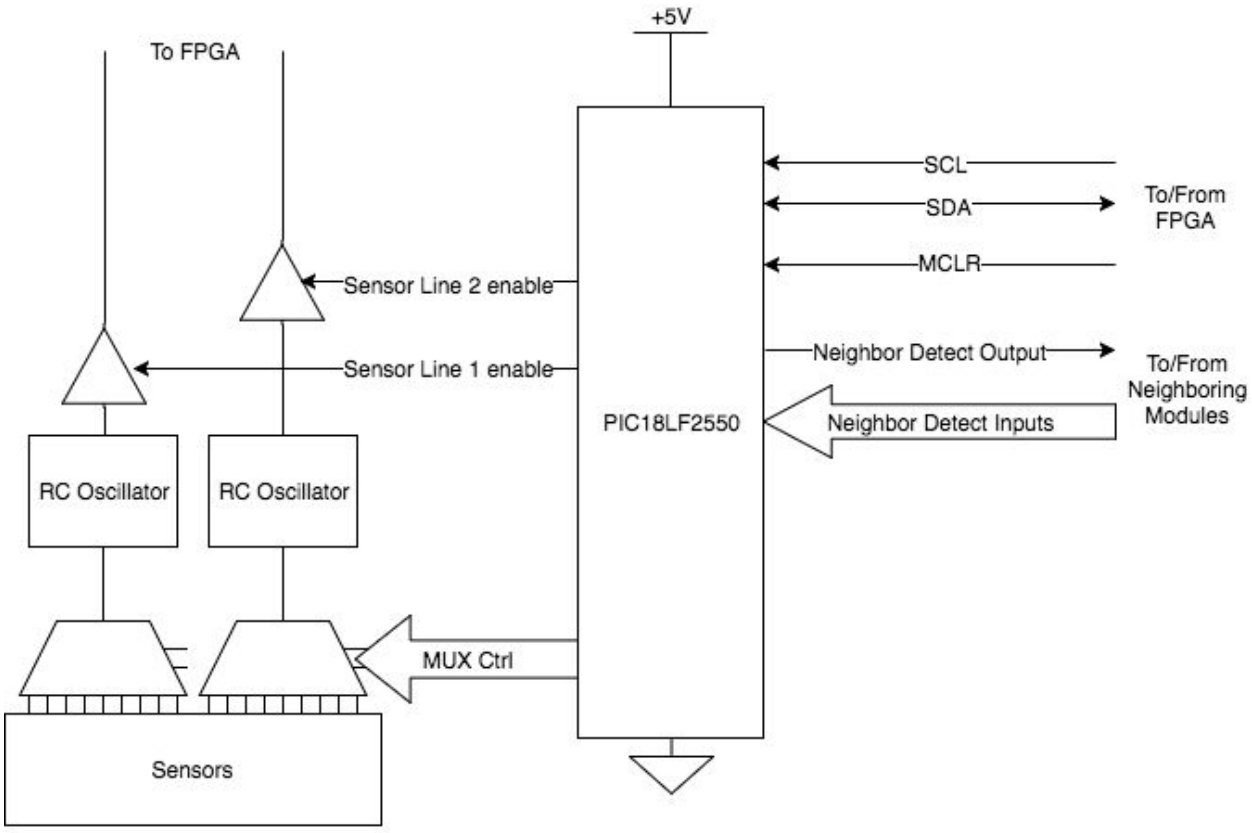
2.1.3.1(a) FPGA continually scans I2C address space during sensor I2C bus dead time (during sensor reads)

2.1.3.1(b) If a change is detected, wait for the current sensor scan to complete, then halt sensor polling and remap sensor layout.

2.1.3.1(c) resume normal operation and update display to new layout without user intervention, don’t crash and burn it the new layout conflicts with IO definitions.

3. Touchpad Block Circuitry

(Aaron and Daniel design circuit, Aaron programs the PIC)



Each block will have 4 connectors on each edge. These connectors will be used to connect adjacent blocks as well as to the FPGA. Each connector will have the following I/O pins:

- SDA/SCL for I2C communication with the FPGA
- 2 sensor data lines
- MCLR (active low PIC reset)
- Neighbor detect pins

The sensor data, MCLR, and I2C communication pins on one edge connector will be electrically connected to their corresponding pins on the other 3 edge connectors so that the entire system

will all share the same sensor data lines, MCLR, SDA, and SCL. In order to prevent contentions on the sensor data lines, tristate buffers will be placed between the sensor outputs on each block and the sensor data lines themselves. Each block will be controlled by a PIC18LF2550 controller which communicates with the FPGA over I2C and handles switching sensors to and from the data lines in response to commands from the FPGA. This PIC will be clocked by the 8MHz internal oscillator and instructions will take 0.5us to execute. (one instruction cycle = 4 clock cylces)

The sensors themselves will be RC square wave oscillators with a no touch frequency of approximately 100kHz. When the user touches one of the touchpads, their parasitic capacitance to the circuit is added in parallel with the capacitor inside the RC oscillator, causing the output frequency to decrease. This change in frequency will be measured by the capacitance sensing module on the FPGA described below. Since each block has two sensor data lines, there will be 2 RC oscillators per block, each of which can be connected to 8 of the 16 sensors on a block. 8 input analog multiplexers will be used to control which sensors are connected to the oscillators and tristate buffers will be used to switch the oscillators on each block to and from the sense lines.

The neighbor detect pins will be used by the mapping module on the FPGA to determine the layout in which all of the blocks are connected. The neighbor detect output pin on the PIC will connect to each of the 4 edge connectors on a block. There will be one neighbor detect input coming from each side of the block, making 4 neighbor detect inputs in total.

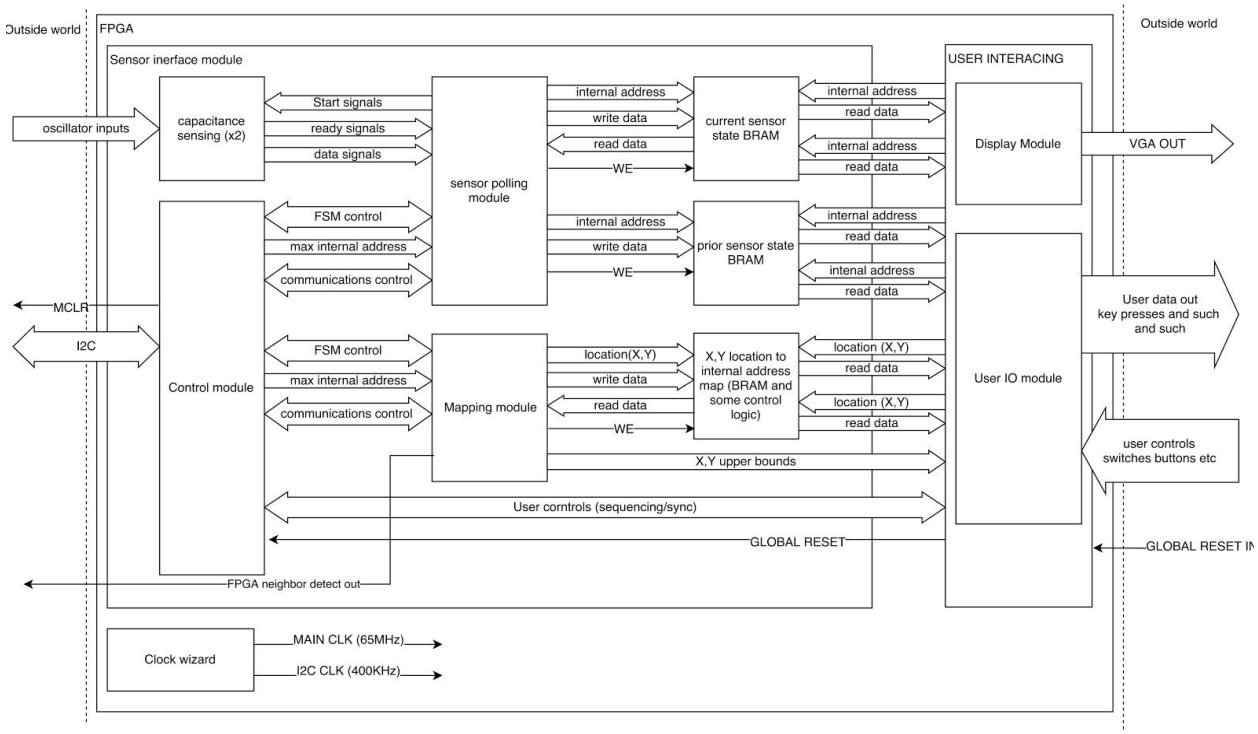
The external components (all of which are relatively inexpensive) needed to make a single block are (excluding passive components such as resistors and capacitors, which are available in lab):

- 2 Analog Multiplexers
- 1 18LF2550 PIC controller
- 2 tristate buffers
- 2 comparators (used in the RC oscillator)
- 2 tristate buffers
- 16 metallic touchpads (which we will make ourselves out of foil)

- 4 DB-9 connectors (the exact connector type we choose to use may change)

If time permits, we will use the EDS PCB mill to fabricate boards that have all 16 sensors as copper pads and boards that contain all of the circuitry for a block.

4 FPGA Block Diagram



5 FPGA Implementation

5.1 Control module (Aaron and Daniel)

The control module will handle most of the communications with the sensor blocks. Its primary function is to provide abstraction of the I2C addresses to an internal address space (a mapping of I2C addresses to a number in the range [0, total number of touchpad blocks]) used by the mapping and sensor polling modules to reference touchpad blocks. It also handles sequencing for the operation of the sensor polling and mapping modules, conflict management if both the mapping and polling modules request to use the I2C bus simultaneously, and synchronization with the user interface (for instance, synchronizing with a display to poll the sensors once per frame refresh).

During startup (or if the global reset is asserted), an FSM inside the control module will send every possible 7 bit I2C address and listen for an ACK from each. If an ACK is detected, that address will be added to a lookup table of addresses implemented using a BRAM, and increments the max internal address pointer (which indicates the highest occupied address occupied in the BRAM and thus the highest valid internal address). The maximum length of the internal address (which we'll call N from here on) will be a parameter shared by all modules. The BRAM will thus need to be 2^N deep by 7 bits wide.

After all I2C addresses have been searched and blocks assigned internal addresses, the sequencing FSM inside the control module will switch I2C control over to the mapping module. Once the mapping module has signaled it is finished, control of the I2C bus will be given to the sensor polling module, which will then cycle through reading the capacitances of the touch sensors. This can either occur continuously, or repeat once every time a sync pulse is sent to the control module, depending upon the desires of the user.

If we implement hot swapping (a stretch goal at best), during the dead time while he polling FSM is measuring the RC oscillator frequencies, the sequencing FSM will hand over control of the I2C bus to an FSM responsible for searching the address space for changes. If a change is detected, it will halt polling, remap the addresses, and trigger the mapping module to update the location to address map, before resuming normal operation.

5.2 Mapping Module (the hardest one) (Daniel)

The mapping module will determine which sensor blocks are next to which, and then use this information to construct a table which stores the internal address and sensor position for a sensor at a given position in (x,y) coordinates. This will be used to allow the display and user IO to access the information from each sensor based on its physical location.

When it receives a start signal from the control module, the mapping module will execute the following procedure for each sensor block (by repeating for address 0 through the max internal address). Note that we need to handle the FPGA location separately.

1. Tell the sensor block at "address" to write it's neighbor detect output pin high.

2. Request each sensor block in the rest of the address space to return which if any of its neighbor detect inputs are currently high.
3. If one of the polled block's pins is high, store that it is abutting the sensor at "address", otherwise just move on to the next one.
4. Repeat until this has been done for all currently valid addresses.
5. Finally, write the FPGA neighbor detect high, use the same procedure to check which block is connected to the FPGA and which side the FPGA is attached to. This will be saved in a special register as {internal_address[N-1:0], side}. Note that side only needs to be one bit, because since we aren't using hermaphrodite connectors, the FPGA can only connect to one of two sides of the board.

The data will be stored in a BRAM with a depth the same as the internal address space. At the location corresponding to each used address, the adjacency information will be stored as follows (where each address is N bits and the valid bit is a one bit flag to clarify whether there is in fact a neighbor)

{left_valid, left_address, top_valid, top_address, right_valid, right_address, bottom_valid, bottom_address}

Once this information is stored, we will construct a second table of signed X,Y coordinates based on the adjacency information. The values for X and Y will initially have to be N+1 bits each to accommodate the sign bit. Data will be stored in a BRAM as {valid, rotated, X, Y}, where valid indicates that the entry is current, and rotated is a 1 bit number that indicates the orientation of the block. From this we then construct the conversion table available to the user IO and display modules. The procedure to generate the addresses to location map will be roughly as follows.

1. Start by assigning the block the FPGA is connected to the coordinates (0,0). If the side the FPGA is connected to is the top, the value of the rotation bit will be defined as 0, if it is connected to the bottom it will be defined as 1.

2. Read the adjacency data for this block. In turn, for each neighbor which doesn't yet have an assigned location, assign a location as follows. base_position(x,y), base_address and base_rotated here refer to the xy coordinates and rotation of the block we're looking at the neighbors of. Plain x, y, rotated, address_right, etc refer to the values associated with the neighbor block. We also need to determine the rotations for the adjacent blocks.

    Left block: x = base_rotated ? (base_position(x) +1) : (base_position(x) - 1)

        y = base_position(y)

        rotated = (address_right == base_ address) ? base_rotated : ~base_rotated

    Right block: x = base_rotated ? (base_position(x) -1) : (base_position(x) + 1)

        y = base_position(y)

        rotated = (address_left == base_ address) ? base_rotated : ~base_rotated

    Top block: x=position(x)

        y = base_rotated ? (base_position(y) -1) : (base_position(y) +1)

        rotated = (address_top == base_ address) ? base_rotated : ~base_rotated

    Top block: x=base_position(x)

        y = base_rotated ? (base_position(y) +1) : (base_position(y) -1)

        rotated = (address_bottom==base_ address) ? base_rotated : ~base_rotated

3. For all addresses less than the max internal address check if there is now a defined location. Repeat step 2 for each address that has a known location.

4. Check if all addresses now have a known location, if not, repeat step 3, if they do, proceed to step 5.

5. Find the minimum X and Y values (this should require one more pass over this BRAM's address space) sign invert these and save them in another register (call these invXmin, invYmin).

6. Write the valid bits for every location in the "X,Y location to internal address" BRAM to zero.

7. For each address retrieve {valid, rotated, X, Y}, store the string {valid, address, rotated} to the location {{X+invXmin}[N-1:0], {X+invXmin}[N-1:0]} in the "X,Y

location to internal address" BRAM. This makes everything positive and drops the sign bit from the location space. Logic internal to the Block containing the Bram will use the rotated bit. Also, record the maximum X and Y coordinates that result from this procedure.

8. Output the maximum X and Y bounds and assert done.

In practice, the "X,Y location to internal address" BRAM should be two BRAMs to allow us to use one as a frame buffer and let us swap them in one cycle. This way we don't ever present a partial map to the user. Note that while not addressed in detail here, the "BRAM" will also have a bit more logic to spit out the mapping of sensors on the blocks based on rotation. This will just rely on a pair of 16 entry roms which we switch between based on the rotation bit (this is the easiest part of this whole thing)

5.3 Capacitive Sensing Module (Aaron)

As mentioned in the Touchpad Block Circuit section, each of the sensor data lines will carry a square wave with frequency dependent on the capacitance added to the block's RC oscillator by a touch. We plan on designing the oscillators to have a frequency of 100kHz when touched and a frequency of no less than 25kHz when touched. The capacitive sensing modules (one module for each sensor data line) will measure count the number of positive edges detected from this square wave in a time period of approximately 0.75ms. This corresponds to 75 positive edges with at the no touch frequency of 100kHz and 19 positive edges at the lowest anticipated frequency of 25kHz, enough to make a reliable measurement.

The inputs to this module will be the two sensor data lines coming from the touchpad blocks and a start signal coming from the polling module telling it to start counting positive edges on the sensor data line. This module will output a ready signal that tells the polling module a measurement is complete as well as output the number of positive edges counted.

5.4 Sensor Polling Module (Aaron)

The sensor polling module is responsible for polling the status of all touchpads in the system. Polling will begin whenever the module receives a start pulse from the Control module. At this point, the polling module will send an I2C command (through the Control module, as described in Section 5.1) to touchpad blocks with internal addresses above 0 to disable their tristate buffers connecting their RC oscillators to the sensor data lines. Commands will then be sent to the touchpad with internal address 0 telling it to enable its tristate buffers and connect the first sensor on each MUX to the corresponding RC oscillator. The polling module will then send a start pulse to each of the capacitive sensing modules and wait until a done pulse is received from them.

After the two sensor data lines have been measured, the new values will be loaded into the current sensor state BRAM, and the value that was previously in the current state BRAM will be loaded into the prior state BRAM. (These two BRAMs will have 2^(N+2) addresses, each of which stores an 8 bit number, where N is the length of the maximum internal address) A command will then be sent telling the touchpad block to connect the second sensor on each MUX to the RC oscillator and the same measurement process will be repeated.

Once all sensors on a touchpad block have been measured, commands will be sent to enable the tristate buffers on the touchpad block with internal address 1 and disable the tristate buffers on all other blocks. The same measurement process as described above will be used to poll the sensors on this block. Once all touchpad blocks have been polled (we will know when all blocks have been polled because the Sensor Polling module has the maximum internal address as an input), the module will pulse its ‘done’ output, telling the control module it has completed.

As mentioned in the above section, the capacitive sensing module will take about 0.75ms to read a sensor. The I2C command telling the touchpad block to switch its MUX outputs will be 20 bits long (Start+Stop+7 bit address+1 bit read/write+8 bit command+2 ACKs). At an I2C clock frequency of 100kHz, this will take 20/100kHz = 0.2ms to complete. Assuming the PIC can switch the multiplexers in under 100 instructions, this means reading the two sensor data lines will take about 1ms. (Since the FPGA clock is significantly faster than I2C communication and a PIC instruction cycle, the time it takes to do operations on the FPGA has been ignored here.) This means we should be able to to poll 200 individual sensors in a tenth of a second,

which is fast enough for smooth operation as long as there are less than around 16 different touchpad blocks.

5.5 User IO (Daniel)

The user IO module will be an interfacing example constructed for our touch sensor platform to allow it to perform various control functions. Basically, we will assign sensors at positions (x,y) to control different IO functions of the FPGA. The module will poll each allocated sensor looking for a change between the current and previous reading greater than some threshold, and if detected, change the appropriate FPGA output.

To do the polling, the module needs to first read the address stored at (x,y) in the "X,Y location to internal address" BRAM. It then uses that address to retrieve the appropriate information from the sensor reading BRAMs. It will then do whatever logic we see fit to make it perform a comparison and change the FPGA outputs.

5.6 Display (Daniel)

The display module will create a visualization of the sensor layout and readings. Based on the max X and Y boundaries output by the mapping module, the display module will divide the display into a grid with one square for each sensor. When a sensor is touched, it will light the corresponding position on screen.

The actual logic behind this will be identical to that in the user IO block. The display module looks up the appropriate address, retrieves the reading, and determines what color the pixels in that patch of screen will be appropriately. It then sends that as a VGA signal.