Lab 7: CapSense

Objective

Demonstrate knowledge of using and tuning CapSense: the capacitive touch solution by Cypress.

Note

This lab is INCOMPLETE. It was originally part of Lab 4, but it was determined that it was too long. Thus, there are still references to USBUART and PWM in this lab. Putting CapSense in its own lab makes more sense too, as teaching about manual tuning is more meaningful than simply copying in a component that is pre-tuned to work with the development kit's sensors. This lab should be modified such that the CapSense schematic is tuned by the student. Additionally, just like Lab 5, the programming section should be less structured so the student can better understand how things work.

Background

B1: CapSense

Capacitive touch sensing is a form of input that detects the presence of a human finger (or an object with similar electrical properties) by detecting the capacitive properties of the human body. Unlike physical switches, buttons that utilize capacitive sensing respond to the proximity of a fingertip, rather than force. Capacitive touch sensing is used in many input applications, including buttons, sliders, and touch surfaces (such as a smartphone touchscreen). Their primary advantage is that force is not required for detection, making applications such as a smartphone less stressful on the hand to use.

There are many implementations, but in all cases, capacitive touch sensing utilizes analog circuitry. Therefore, the physical properties of a touch pad and the connection to the detection circuit changes what the circuit reads. It is important to tune how a system responds to readings from a capacitive sensor so it matches the properties of that particular sensor. For example, one tuning parameter is what the readings are when there is no touch (also known as the baseline).

Cypress' CapSense is an implementation of capacitive touch sensing that can also simplify the tuning process by automatically changing tuning parameters at power-up and during runtime based on the sensor response. Tuning can be a time-consuming process, so automatic tuning can help adjust for manufacturing variations and can make hardware prototyping quicker. CapSense can come in a separate chip, but the PSoC has CapSense hardware built in.

The PSoC 5LP development kit comes with two capacitive buttons and a capacitive slider (see Figure 1). These sensors already come with tuning parameters in an example project that can be found with the development kit example projects. This will be used as the basis for configuring the CapSense component in this lab.

CapSense is a complicated topic that requires knowledge of electronics to fully understand. This background section and the lab will only cover how to use CapSense to make the capacitive buttons and slider on the development kit work. For more information, read the CapSense component datasheet, document no. 001-64846 for the getting started guide, and document no. 001-75400 for the hardware design guide (do a web search or find links to the documents at the beginning of the datasheet).

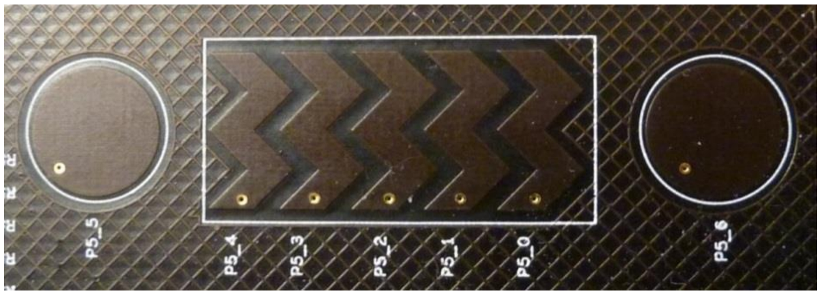
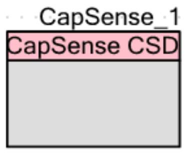


Figure 1: The capacitive touch sensors on the development kit

Configuration of CapSense is contained within the schematic component named CapSense_CSD.



There are a few hidden component inputs and outputs, but these will not be used in this lab.

Here are the most important configuration options. Proper configuration to match the hardware on the development kit will be covered in the procedure.

- General tab
  - Tuning method
    - Auto (SmartSense): tuning parameters for each sensor will be automatically determined at startup and during runtime. This option uses significantly more CPU resources than manual (as tuning is performed on the CPU), but it is easier to start with. Many advanced options are locked out when using this method.
    - Manual: tuning parameters are set to manual values and can be changed using a program on the PC that interfaces with the CapSense component. See the datasheet for more information.
    - None: tuning parameters are set to manual values that cannot be changed after programming. Should only be used when tuning parameters are finalized under Manual mode.
- Widgets Config tab: this is where each sensor or sensor group, collectively called widgets, are defined and configured. There are several types of widgets and each type has its own options. To add a widget, click the folder icon for it and click the Add button in the top left. The widgets may be double clicked to rename them. Here are options for some of the widget types:
  - Buttons
    - Debounce: in order for the button to transition from inactive to active states, a finger must be detected for this number of samples (1-255, 5 default) before the button is considered activated. There are also other filtering methods that work at the analog reading level that reduce bouncing issues.
  - Linear Sliders
    - Number of Sensor Elements: sliders are made of multiple sensors. There are five sensors on the development kit's slider, but others may have more or less.

        - API Resolution: sets the slider resolution. In other words, the value returned for the slider ranges from 0 to this value (max 255). It is usually best to limit this to 20 times the number of sensors, so the default is 100.
- Advanced Tab
    - Scan speed: determines how long sensor readings take. A slower speed allows for a stronger signal to be read and reduces CPU processing time.

Note that for linear sliders, the pin assignment in the DWR should go from the lowest position to the highest position. Also, this component uses many UDB resources. If this becomes a concern, in the advanced tab, some of the timers can be switched from being UDB to FF. It also uses the following analog resources: a CapSense block, a DAC, and a comparator.

Each widget gets a constant value for use in functions that expect a widget as a parameter. For a button named Button0 in the component named CapSense, the constant is named CapSense_BUTTON0__BTN. For a linear slider named LinearSlider0 in the component named CapSense, the constant is named CapSense_LINEARSLIDER0__LS.

The general process for initializing and reading CapSense sensors is as follows:

- Initialize: Start, InitializeAllBaselines
- Repeat:
    - Scan: UpdateEnabledBaselines, ScanEnabledWidgets
    - Wait for scan to complete by checking IsBusy
    - Read: CheckIsWidgetActive for buttons, GetCentroidPos for sliders

Here are the most important functions in the programming interface. “CapSense” is replaced with the name given to the component in the schematic view.

void CapSense_Start()	Initializes the CapSense component.
void CapSense_InitializeAllBaselines()	Initializes the baselines for all sensors, where the baseline is what the sensors read when there is no touch.
void CapSense_UpdateEnabledBaselines()	Updates the baselines for all enabled sensors. This should be done periodically (ideally, before scans) in case the baseline changes due to changes in background noise.
void CapSense_ScanEnabledWidgets()	Starts the scanning of enabled widgets. This is non-blocking because an interrupt handles the scanning.
uint8 CapSense_IsBusy()	Returns 1 if the scanning is still happening, 0 if it’s complete.
uint8 CapSense_CheckIsWidgetActive (uint8 widget)	Checks to see if a given button has been pressed. Note that filtering happens when this function is called, so it should be called on every button after every scan. Returns 1 if pressed, else 0.
uint16 CapSense_GetCentroidPos (uint8 widget)	Returns the finger position of the given linear slider. This value ranges from 0 to API Resolution, or is 0xFFFF if there’s no finger.

Procedure

Outcome

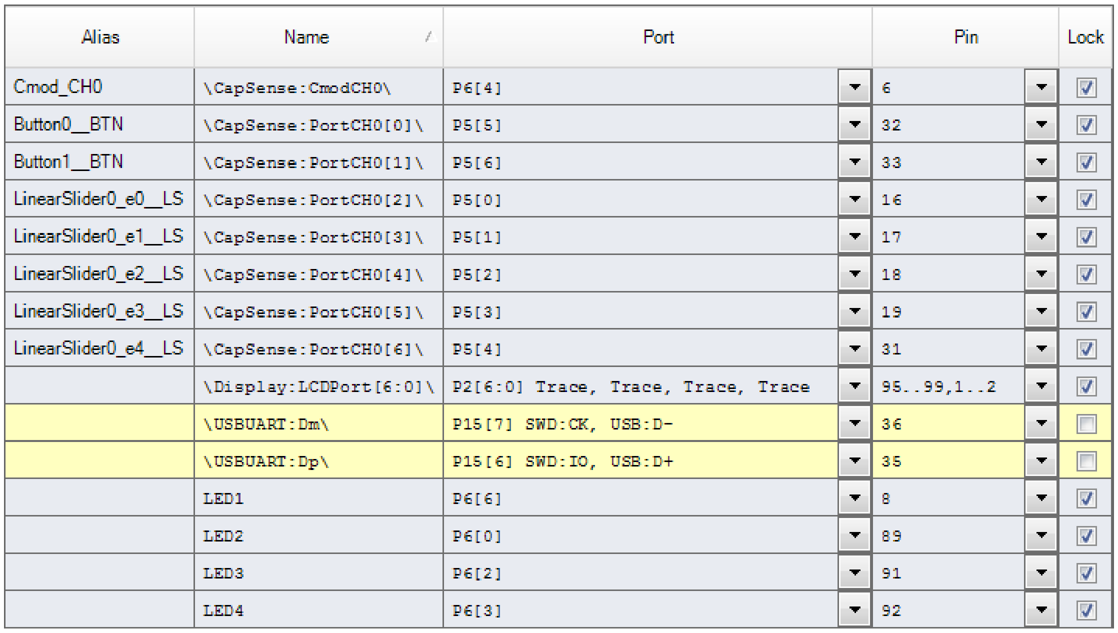
In this part of the lab, the CapSense slider will control the brightness of LED3 by changing the second comparison output of the PWM, which will be connected to LED3. Additionally, the two CapSense buttons will light up LED1 and LED2 while touched. LED1 and LED2 are not directly connected to the PSoC, so making use of them will be demonstrated.

Part 1: Designing the System

1. Add a digital output pin to the schematic and name it LED3. Connect it to the pwm2 output of the PWM block.
2. Add two digital output pins to the schematic named LED1 and LED2. Uncheck HW Connection in both pins' configurations, since there are no hardware connections for CapSense buttons.
3. Open a second copy of PSoC Creator. From the Start Page, there is a section in the left column named Examples and Kits. Choose Kits>CY8CKIT-050LP 3.0>CapSense.cywrk. A dialog will appear asking where to place a new workspace.

If the example project cannot be found, the development kit examples are not installed. Close PSoC Creator, visit http://www.cypress.com/?rID=51577, and download and run the "CY8CKIT-050 PSoC 5LP DVK Only" installer. Do not download the one that ends in "Setup" as that includes PSoC Creator as well.

4. Open TopDesign.cysch in the example project. Select the CapSense CSD component and copy it. Go back to the first copy of PSoC Creator and paste the component into the Lab4 project schematic. The second copy of PSoC Creator may now be closed and the workspace that was created may be deleted.
5. Modify the following settings in the CapSense component.
   - Widgets Config>LinearSlider0: change API resolution to 255. This should be changed because the PWM compare value ranges from 0 to 255.
   - Advanced>Clock Settings: change Scan speed to slow. Since there are only seven sensors, slow scan speed is still sufficient for responsive controls.
6. Open Lab4.cydwr
7. Configure the following pins (Figure 2):
   - Cmod_CH0: P6[4]
     - This pin is connected to a capacitor next to the PSoC. It is required for CapSense.
   - Button0__BTN: P5[5]
     - The buttons and slider on the development kit have the pins they're connected to written next to them, just like the LEDs
   - Button1__BTN: P5[6]
   - LinearSlider0_e0__LS: P5[0]
   - LinearSlider0_e1__LS: P5[1]
   - LinearSlider0_e2__LS: P5[2]
   - LinearSlider0_e3__LS: P5[3]
   - LinearSlider0_e4__LS: P5[4]
   - LED1: P6[6]
     - LED1 and LED2 are not directly connected to the PSoC. Part 3 will explain how to connect them to these pins.
   - LED2: P6[0]
   - LED3: P6[2]
8. Go to the System tab. Change all the voltages under operating conditions from 5.0 to 3.3.
9. Generate the application.



Alias	Name	Port	Pin	Lock
Cmod_CH0	\CapSense:CmodCH0\	P6[4]	6	✓
Button0__BTN	\CapSense:PortCH0[0]\	P5[5]	32	✓
Button1__BTN	\CapSense:PortCH0[1]\	P5[6]	33	✓
LinearSlider0_e0__LS	\CapSense:PortCH0[2]\	P5[0]	16	✓
LinearSlider0_e1__LS	\CapSense:PortCH0[3]\	P5[1]	17	✓
LinearSlider0_e2__LS	\CapSense:PortCH0[4]\	P5[2]	18	✓
LinearSlider0_e3__LS	\CapSense:PortCH0[5]\	P5[3]	19	✓
LinearSlider0_e4__LS	\CapSense:PortCH0[6]\	P5[4]	31	✓
	\Display:LCDPort[6:0]\	P2[6:0] Trace, Trace, Trace, Trace	95..99,1..2	✓
	\USBUART:Dm\	P15[7] SWD:CK, USB:D-	36	
	\USBUART:Dp\	P15[6] SWD:IO, USB:D+	35	
	LED1	P6[6]	8	✓
	LED2	P6[0]	89	✓
	LED3	P6[2]	91	✓
	LED4	P6[3]	92	✓

Figure 2: Pin assigments for CapSense and LEDs

Part 2: Programming the Firmware

Outcome

The firmware will be modified to read the CapSense widgets while continuing to work with the USB UART. The linear slider will control the brightness of LED3 through the second comparison output of the PWM and the buttons will turn on LEDs 1 and 2 while pressed.

Printing LED Brightness

Modify the function that prints LED brightness to print the brightness value of LED3 to the second line of the display. Optionally, make it so updates only overwrite the numbers, rather than erasing the screen.

Initialization

Add the following definition:

```
#define NO_FINGER 0xFFFF
```

Add the following variable to the rest of the variables at the beginning of the main:

```
uint8 led3Bright = 0;
```

Set the initial brightness of LED3 using PWM_WriteCompare2(led3Bright).

Initialize the CapSense component by adding calls to CapSense_Start() and CapSense_IntitializeAllBaselines() after other component intitializations.

Start the first scan of the CapSense sensors by calling CapSense_UpdateEnabledBaselines() and CapSense_ScanEnabledWidgets().

Main For Loop

Place all code for reading CapSense scans at the end of the for loop, after all of the USB UART code.

Check to see if CapSense has finished scanning the sensors using CapSense_IsBusy(). If it is still busy, do nothing. Otherwise, do the following.

Read the status of Button 0 (which is the top button) using:
`CapSense_CheckIsWidgetActive(CapSense_BUTTON0__BTN)`
If the button is pressed (1), light up LED2 using LED2_Write(uint8), otherwise turn LED2 off using the same function.

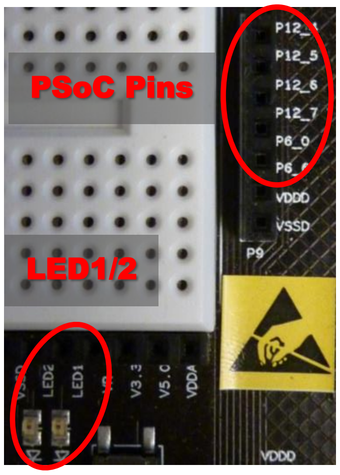
Do the same thing with Button 1 and LED1. The constant to use for CheckIsWidgetActive is CapSense_BUTTON1__BTN.

Get the position on the linear slider using:
`CapSense_GetCentroidPos(CapSense_LINEARSLIDER0__LS)`
If the value returned is NO_FINGER or the same as the last value read, ignore the value returned. Otherwise, cast the value to uint8 and assign it to led3Bright. Then set the brightness of LED3 using PWM_WriteCompare2(led3Bright) and print the new brightness to the LCD display using the print function that was previously updated.

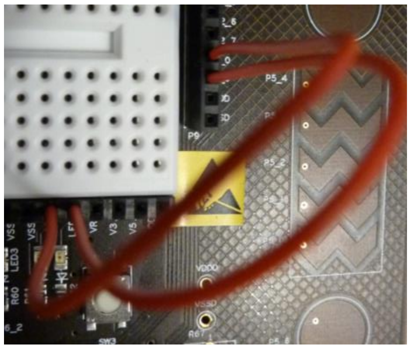
Initiate the next scan of the sensors by calling CapSense_UpdateEnabledBaselines() and CapSense_ScanEnabledWidgets().

Part 3: Running the Project

1. Before running the project, some setup on the development kit is required. Make sure the development kit is unplugged before continuing.
2. LED1 and LED2 are not directly connected to the PSoC. Instead, they are connected to headers next to the white prototyping area above the pushbuttons. There are also headers that are connected to pins on the PSoC. See the image below for where these headers are.



3. In the development kit box, there should be a small bag of wires in the USB cable compartment. Take out two of the wires and connect the P6_0 header to the LED2 header and connect the P6_6 header to the LED1 header.



4. Build and load the project onto the PSoC development kit. Make sure you are not touching any of the capacitive touch sensors, shown in Figure 3, during programming.

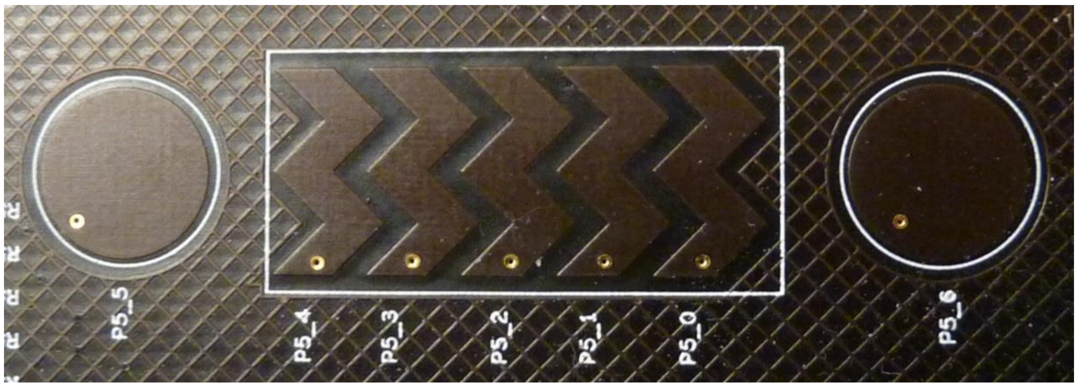
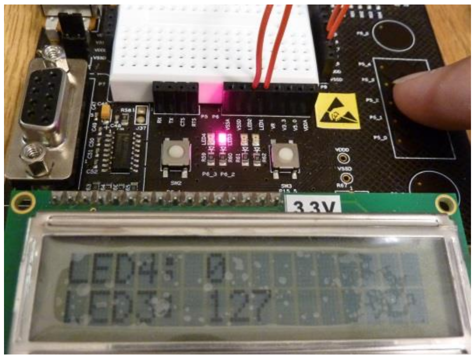


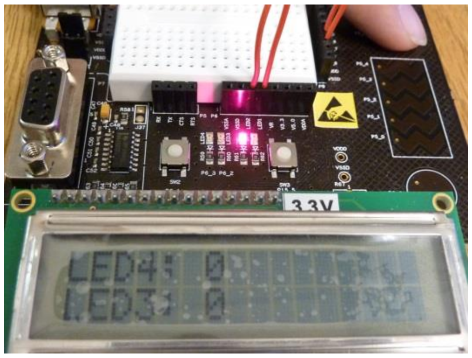
Figure 3: The capacitive touch sensors on the development kit

5. Test the program to make sure it works:
   - Starting out, the LCD display should say both LEDs are 0 and all LEDs should be off.
   - Test the USB UART interface to see if it still works with LED4.

- Place your finger in the middle of the capacitive slider. The value on the display should read somewhere around 127 and LED3 should light up with medium brightness.



- Slide your finger toward the bottom of the slider. LED3 should dim and eventually turn off and the display should similarly indicate a value decreasing to 0.
- Slide your finger toward the top of the slider. LED3 should become bright and the display should similarly indicate a value increasing to 255.
- Release your finger. The brightness and value of LED3 should settle and not change.
- Tap the top capacitive button. LED2 should light up while the button is touched.



- Tap the bottom capacitive button. LED1 should light up while the button is touched.