Haptics and Capacitive Touch Using the MSP430TCH5E: 
the HapTouch BoosterPack

Keith Quiring

ABSTRACT

This report describes a BoosterPack™ for use with the G2 LaunchPad™, such that when they are paired together, they become a haptics-enabled game controller for use with PC-based gaming platforms. The BoosterPack uses the MSP430TCH5E, an MSP430™ derivative specially enabled to use the MSP430TCH5E Haptics Library. This library, in turn, is built around the TouchSense™ TS2200 haptics library from Immersion Corporation, the haptics industry leader. The BoosterPack also features use of MSP430’s Capacitive Touch Library to accomplish inexpensive capacitive buttons. The app note further demonstrates how to use MSP430’s TouchPro GUI to tune these buttons.

1 Introduction .................................................................................................................................. 3
2 What Can I Do With the HapTouch BoosterPack? ..................................................................... 6
   2.1 Standalone Usage Mode: Experiencing Haptics Using the Capacitive Buttons .......... 6
   2.2 Gaming Usage Mode .............................................................................................................. 6
   2.3 GUI Usage Mode: Building Your Own Sequences with the HapTouch GUI ................... 7
   2.4 Audio Haptics Usage Mode ..................................................................................................... 8
   2.5 Writing Your Own Haptics Applications ................................................................................... 9
   2.6 System Requirements ............................................................................................................. 9
3 HapTouch Software Developers Kit (SDK) Overview ............................................................... 10
4 Description of the LaunchPad and BoosterPack System ........................................................ 11
   4.1 The BoosterPack’s Relationship with the LaunchPad ........................................................... 11
   4.2 Communication Paths Between the PC, LaunchPad, and BoosterPack ............................... 17
   4.3 BoosterPack Block Diagrams ................................................................................................ 18
5 Using the HapTouch BoosterPack ............................................................................................ 21
   5.1 Flashing Firmware Images to BoosterPack and LaunchPad ................................................. 21
   5.2 The BoosterPack’s State Machine ........................................................................................ 25
   5.3 Standalone Haptics Usage Mode .......................................................................................... 28
   5.4 Gaming Usage Mode ............................................................................................................... 31
   5.5 Audio Haptics Usage Mode .................................................................................................... 41
   5.6 GUI Usage Mode: the HapTouch GUI .................................................................................. 43
   5.7 Changing Haptics Actuators ................................................................................................. 50
6 Emulating the Software: Stepping, Debugging, Developing .................................................. 52
   6.1 Tool Requirements ................................................................................................................ 52
   6.2 Setting Up Full Emulation for the First Time ......................................................................... 52
   6.3 Emulating Both the LaunchPad and BoosterPack Simultaneously .................................... 57
   6.4 Emulating the BoosterPack Without the LaunchPad, or With Another PCB ..................... 57
7 Hardware Design ........................................................................................................................ 60
   7.1 Power Supply ........................................................................................................................ 60
   7.2 Clocks 60
   7.3 Actuator and Driver Device ................................................................................................. 61
   7.4 Touch Buttons, Wheel, and LEDs ......................................................................................... 62
   7.5 Audio Interface ..................................................................................................................... 63
   7.6 LaunchPad ↔ BoosterPack Connections ............................................................................. 64
Introduction

8 LaunchPad and BoosterPack Software Design ................................................................. 65
  8.1 LaunchPad Application .................................................................................................. 66
  8.2 BoosterPack Application ............................................................................................. 71
  8.3 I2C Commands ............................................................................................................ 92
  8.4 PC→LaunchPad→BoosterPack Communication Dynamics ...................................... 96

9 Tuning the Capacitive Touch Buttons, Using the TouchPro GUI .................................. 100
  9.1 System Communication ............................................................................................. 101
  9.2 Setup 103
  9.3 Tuning 104

10 Ideas for Modifications ................................................................................................. 105
  11.5 The MSP430 Haptics Library Designer’s Guide .................................................... 107
  11.6 The MSP430 Capacitive Touch Resources ............................................................... 107
  11.7 MSP430Ware and the TI Resource Explorer .......................................................... 107

Appendix A. BoosterPack Schematic .................................................................................. 108
Appendix B. BoosterPack Bill of Materials (BOM) ............................................................ 109
Appendix C. Individual Haptics Effects .............................................................................. 110

BoosterPack, LaunchPad, MSP430 are trademarks of Texas Instruments.

TouchSense, Audio2Haptics are trademarks of Immersion Corporation.
1 Introduction

TI makes it easy to add haptics effects to any application, using the MSP430TCH5E Haptics Library. The tactile feedback provided by haptics engages one of the five senses not always brought into play by electronic devices: touch. As such, it can add an exciting new dimension to any handheld device. The library utilizes intellectual property from Immersion, the leader in haptics technologies.

The MSP430TCH5E is an MSP430 derivative created especially for the haptics library. It has small package size and low cost, as well as sufficient memory for adding functions that often go together with haptics, such as capacitive touch.

This report describes a board designed in the BoosterPack format, to be paired with a MSP430 G2 LaunchPad (MSP-EXP430G2). Together, they form a touch- and haptics-enabled gaming controller. It can be used to play games on a variety of web- and PC-based platforms; for example, http://www.virtualnes.com.

The BoosterPack and LaunchPad can also be used to evaluate the full range of haptics features made possible with the library. This means you can sample all 122 effects, as well as build your own sequences of effects.

Complete schematics, code, and layout are included with this document. The board is available for purchase from Element14. Accompanying software -- the HapTouch Software Development Kit (SDK) -- can be downloaded.
Features of the BoosterPack include:

- Works as a gaming controller with popular web- and PC-based gaming platforms
- You can experience all 122 effects in the MSP430TCH5E Haptics Library
- You can design your own haptics effect sequences, and distribute them to others
- Choose between an eccentric rotating mass (ERM) haptics actuator, or a Linear Resonant Actuator (LRA)
- Beyond just listening to music, you can experience it, using Immersion’s audio2haptics™ feature.
- Observe the MSP430TCH5E being used as either a fixed-function serial-slave haptics device, or running a complete on-chip haptics and touch application
- Allows interface with other host MCUs through I2C over an external MCU interface header
- Use of a separate JTAG connection for the BoosterPack allows simultaneous emulation
- Demonstrates use of MSP430’s TouchPro GUI tool to tune the capacitive touch buttons

This application note can be thought of as existing in two halves:

- The first half focuses on how to use the BoosterPack and all its features.
- The second half, starting in Section 7, focuses on the design of the BoosterPack, describing how you can step through the code and make your own modifications.
Figure 2. The BoosterPack’s Features and Controls
2 What Can I Do With the HapTouch BoosterPack?

The HapTouch BoosterPack can be used in many different ways; each is briefly described here. Later in the document, each one is described in full detail.

As you read, keep in mind that every mode gives you a choice of using the Eccentric Rotating Mass (ERM) actuator, or the Linear Resonant Actuator (LRA). Both are present on the BoosterPack, but only one can be used at a time. You can select this at any time, using the ERM/LRA slider switch.

As you read this document, you will encounter two different terms for haptics elements: effect and sequence. These mean different things:

**Haptics Effect:** This is the elemental building block of haptics in the MSP430TCH5E Haptics Library. Compared to sequences, they are simple and usually short. Examples include “strong click”, “soft fuzz 30%, or “soft fuzz 60%”.

**Haptics Sequence:** A sequence is one or more effects chained together, with delay gaps assigned in between. A repeat count is then assigned to the entire chain. This allows creation of more complex effects that might mimic real-world vibrations, like a machine gun.

2.1 Standalone Usage Mode: Experiencing Haptics Using the Capacitive Buttons

The BoosterPack demonstrates all 123 haptics effects in the MSP430TCH5E Haptics Library (listed in Appendix C), as well as several predefined sequences:

- SELECT button: Rolling Dice Sequence
- START button: Explosion Sequence
- Button B: Machine Gun Sequence
- Button A: Strong Heartbeat Sequence

You can access these using the eight capacitive touch buttons (wheel, SELECT, START, B, A). Section 5.3 explains how.

2.2 Gaming Usage Mode

The BoosterPack and LaunchPad can be used to play vintage video games at online sites like [www.virtualnes.com](http://www.virtualnes.com). In fact, if you have two of them, you can do two-player gaming.
Figure 3. Super Mario at [www.virtualnes.com](http://www.virtualnes.com)

Use of the BoosterPack as a game controller is described in detail in Section 5.4.

### 2.3 GUI Usage Mode: Building Your Own Sequences with the HapTouch GUI

The HapTouch GUI makes it easy to find the exact haptics effect you want. 122 effects is a large number, so the GUI uses filters to help you narrow down the one you want.

The GUI also enables you to build your own sequences of effects. You can then assign these sequences to the capacitive touch buttons.

You can also create *game profiles*, for use when playing various video games on various websites or platforms. A profile groups together:

- Haptics sequences for all eight capacitive touch buttons (the wheel’s four directions, SELECT, START, B, and A)
- A PC keystroke assignment for each of these buttons

Each profile enables you to play a particular game, on a particular gaming website or platform. You can create your own haptics sequences to match the game, then build them into a profile. When you are done, you can export these profiles and distribute them freely to others online.
What Can I Do With the HapTouch BoosterPack?

2.4 Audio Haptics Usage Mode

The Immersion haptics technology contained within the MSP430TCH5E Haptics Library enables a unique feature: audio-to-haptics. It lets you experience sounds not only with your ears, but also with touch. If the audio source plays the sound of an explosion, for example, the MSP430TCH5E can simultaneously shake the controller with a strong, reverberating haptics effect.

In this usage mode, you can attach an audio source to the BoosterPack and see the BoosterPack respond to the audio. This is detailed in Section 5.5.
2.5 Writing Your Own Haptics Applications

Although there is a lot you can do with the BoosterPack simply from a user's perspective, you may also want to develop your own applications using the MSP430TCH5E device that controls the BoosterPack. The BoosterPack is fully set up to support haptics development.

The MSP430TCH5E, as well as the MSP430TCH5E Haptics Library, are designed to be used in two different ways:
- As a serial slave device, responding to commands from a host MCU over I2C or SPI
- You write an application for the TCH5E, driving haptics in response to system inputs

The BoosterPack supports both kinds of development. In other words, you can develop haptics code directly on the MSP430TCH5E, or you can drive haptics from a host. The host might be:
- The LaunchPad. An example is provided in the MSP430TCH5E Haptics Library that shows the LaunchPad driving TCH5E haptics using serial commands over I2C
- The PC. The HapTouch GUI essentially acts in the role of the host MCU, sending the same serial commands used by the LaunchPad
- Another MCU. The BoosterPack contains an external host header, which allows you to connect with another PCB using jumper wires.

2.6 System Requirements

The requirements vary by usage mode. They are shown in Table 1

<table>
<thead>
<tr>
<th>Usage Mode</th>
<th>Power Source</th>
<th>Windows PC</th>
<th>Audio Source</th>
<th>MSP-FET430UIF FET Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standalone</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaming</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUI</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Writing your own Applications</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

A few comments apply to this table:
- Power is typically applied by attaching the BoosterPack to a LaunchPad, and plugging the LaunchPad into a USB host (like a PC). But there are alternate means of power; see Section 7.1.
- The PC is specified as Windows, because at this time, the two PC applications included in the HapTouch SDK have been developed for Windows only.
## 3 HapTouch Software Developers Kit (SDK) Overview

The contents of the SDK are shown in Table 2.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Using the Demo's</th>
<th>Gaming</th>
<th>Studying the design</th>
<th>Evaluating Haptics Effects/Sequences</th>
<th>Building Boards</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP430_Haptics_Library</td>
<td>Copy this directory to any project you create that uses the haptics library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HapticsLib.lib</td>
<td>A static library file containing the entire MSP430TCH5E Haptics Library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HapticsLib.h</td>
<td>A header file allowing access to the library</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HapTouch_BoosterPack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSP430_One_Click_Download_Images</td>
<td>Use these to quickly flash your BoosterPack and LaunchPad with the default images needed to use the BoosterPack.</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProgramTheBoosterPack.bat</td>
<td>Invokes the MSP430Flasher to download the default factory image to your BoosterPack, using a UIF430 FET Tool.</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ProgramTheLaunchPad.bat</td>
<td>Invokes the MSP430Flasher to download the LaunchPad software image to a G2553 mounted onto a G2 LaunchPad.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>MSP430_Source_Projects</td>
<td>Code Composer source projects for the images in the one-click-download directory.</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoosterPack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>LaunchPad</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>PC_Tools</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>HapTouch_GUI</td>
<td>Can be used to drive haptics and create new haptics sequences, and program them to the BoosterPack. (Supports Windows only.)</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Touch2Key</td>
<td>Used with the gaming usage mode. Converts capacitive touch button presses to keystrokes, for use with online gaming sites like <a href="http://www.virtualnes.com">www.virtualnes.com</a>. (Supports Windows only.)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PCB_Design_Files</td>
<td>The hardware files needed to create the board.</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HapTouch_Eagle</td>
<td>Design files for use with EAGLE</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Gerbers</td>
<td>Gerber files for building up your own boards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Plastics_and_overlay</td>
<td>Mechanical drawings, including an *.stp CAD file</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HapTouch_Schematic.pdf</td>
<td>A PDF of the schematic</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HapTouch_BOM.txt</td>
<td>Bill of materials</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>AppNote – HapTouch_BoosterPack</td>
<td>Double-click this to download this document.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manifest.html</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4 Description of the LaunchPad and BoosterPack System

This BoosterPack was designed primarily for the G2 LaunchPad (MSP-430EXPG2). Because of this, discussion throughout the document assumes that this is what you are using. However, other LaunchPads might be compatible; see Section 4.1.4 for information on determining compatibility.

4.1 The BoosterPack’s Relationship with the LaunchPad

A diagram of the BoosterPack and a LaunchPad are shown below.

![Diagram of BoosterPack and LaunchPad](image)

Figure 5. Structural Diagram

Most Booster Packs for MSP430 LaunchPads do not have an MSP430 on them. They rely on the MSP430 on the LaunchPad to drive the BoosterPack’s functions, through the 20- or 40-pin BoosterPack interface.

In contrast, this BoosterPack has its own MSP430: the MSP430TCH5E. The TCH5E drives all the BoosterPack functions. The BoosterPack also has its own JTAG header for emulation. (In fact, the LaunchPad and BoosterPack can be emulated simultaneously.)

The reasons the LaunchPad exists in this system are:

- It provides power to the BoosterPack
- It provides a means of communication between the BoosterPack and the PC (via I2C)
- It helps demonstrate the use of the MSP430TCH5E as a haptics serial slave device

The BoosterPack must always be mounted into the LaunchPad for all the available usage scenarios, because it needs power. (There are alternate BoosterPack configurations that provide power in other ways, but these require hardware changes; see Section 7.1.)

If using a G2 LaunchPad, and if the BoosterPack does not need to communicate with the PC, then the G2553 does not need to be in the LaunchPad’s socket. But if communication is required, the G2553 must be there, and it must be running the LaunchPad software included with this application report.

If an MSP430 is present within the LaunchPad, programmed with software other than the code image provided with this document, it is recommended to remove it, so that it cannot interfere with the BoosterPack operation.
4.1.1 LaunchPad’s Backchannel UART

All MSP430 LaunchPads have a feature called the backchannel UART, sometimes called the application UART. This enables easy communication between the target MSP430 device and the PC. This is implemented with the emulator integrated onto the LaunchPad; it contains a USB-to-UART bridge. When you attach the LaunchPad, this bridge silently enumerates on the PC as a virtual COM port based on the Communications Device Class (CDC), without any PC driver installation required. Applications on the PC can then send bytes over the virtual COM port, and they arrive at the target MSP430. The same works in reverse.

This is how the LaunchPad facilitates communication between the BoosterPack and the PC; see Section 4.2.

4.1.2 LaunchPad Settings

Any LaunchPad can provide power, and so this requirement is taken care of simply by mounting the BoosterPack into the LaunchPad and plugging the LaunchPad into a USB host (typically a PC).

Some usage modes require more from the LaunchPad than just power; they require it to facilitate communication with the PC. In these cases, the LaunchPad must be configured properly:

1. The PCB must be v1.5 or later. (This is marked on the PCB silkscreen next to “MSP-EXP430G2”.) This is because earlier versions do not support the USCI-based UART backchannel.

2. Use an MSP430G2553 in the LaunchPad socket, rather than one of the other devices compatible with the LaunchPad.

3. Disconnect the P1.6 LED jumper. If the jumper is present, it creates a strong pulldown on the I2C SCL line, interfering with communication with the BoosterPack.

4. On the jumper block header, turn the two left-most jumpers sideways, as shown in Figure 6. This enables use of the USCI-based hardware UART on the MSP430G2553, rather than the timer-based UART.

5. The G2553 must be running the LaunchPad software provided in the HapTouch SDK.

6. No crystal is required on the LaunchPad.
4.1.3 Proper Placement of the BoosterPack on the LaunchPad

After configuring the LaunchPad as described above (if needed for communication), mount the BoosterPack into the LaunchPad as shown in Figure 7.
Figure 7. Mounting the BoosterPack into a G2 LaunchPad

Notice that the top and bottom edges of the two boards are flush with each other. Also notice that there is a silkcreen key on the BoosterPack that matches with pin 1 on the LaunchPad’s BoosterPack interface.
4.1.4 Plastic Enclosure Handles and Overlay

Three pieces of plastic are defined for the BoosterPack.

Two pieces are handles. Aesthetically, they complete the rectangular form factor of the game controller. Electrically, they prevent the user from touching the underside of the BoosterPack. This is important, because touching the traces for the capacitive touch buttons, or the buttons themselves, could register as a touch.

The handles can be detached by removing the four screws on each handle with a small Phillips screwdriver. This step is actually necessary in order to access the JTAG connector or the external host MCU header.

The other plastic piece is the overlay. Like the handles, the overlay has both an aesthetic purpose and a functional one. Most capacitive touch applications have an insulating overlay for to protect against ESD. Although this setup is already not protected from ESD, the overlay is useful in demonstrating usage of MSP430's capacitive touch solution with one .

Design files for all the components are present within the software package.

4.1.5 Compatibility with LaunchPads Other than the MSP-430EXPG2

This BoosterPack was designed specifically for the G2 LaunchPad (MSP-430EXPG2). The reason a specific one was chosen was not for electrical reasons, but rather because of the intention to create a clean, rectangular form factor.

Electrically, this BoosterPack has very few connections with the LaunchPad. It only connects with it via power and the I2C pins (in addition to one unused, reserved signal). For this reason, it is probably electrically compatible with nearly any LaunchPad.
Physically, the main limitation is the width made available when the plastic enclosure handles are attached. It is always a possibility to detach the plastic enclosure pieces; but be careful about touching the underside of the board, because touching the capacitive touch pads and traces might cause a touch to be seen by the MSP430TCH5E.

![BoosterPack](image)

**Figure 9.** Physically Matching the BoosterPack to Other LaunchPads
4.2 Communication Paths Between the PC, LaunchPad, and BoosterPack

The overall system, and the communication paths, are shown below.

You can see that a multi-section communication path exists between the MSP430TCH5E on the BoosterPack, and the PC. In this way, the LaunchPad essentially serves as a smart USB-to-I2C bridge.

As discussed in Section 4.1.1, the emulator on MSP430 LaunchPads provides a backchannel UART through which the target MSP430 can communicate with the PC. When the LaunchPad is attached to the PC over USB, the backchannel produces a virtual COM port on the PC. This application report provides LaunchPad software that receives UART commands from the PC and translates them to I2C for the BoosterPack.
4.3 BoosterPack Block Diagrams

4.3.1 Complete Block Diagram

A complete block diagram is shown below.

Although having the benefit of being complete, this diagram has quite a few items in it, many of which are not used simultaneously. To help with this, the sections that follow this one show reduced versions of this block diagram, one for each usage.

![Complete HapTouch BoosterPack Block Diagram](image)

Figure 11. Complete HapTouch BoosterPack Block Diagram
4.3.2 Standalone Usage Block Diagram

In standalone usage, the BoosterPack is not communicating with a PC, but rather takes its direction from the on-board capacitive touch buttons.

![Block Diagram During Standalone Use](image)

The mode slider switch needs to select “BTN”, which means you will be interfacing with the BoosterPack using the capacitive touch buttons. You also must select which haptics actuator you want to use, using a slider switch.

A full description of how to use the BoosterPack in this way is given in Section 5.3.
4.3.3 Audio Haptics Block Diagram

While demonstrating the audio haptics feature, the block diagram appears as shown below.

![Block Diagram During Audio Haptics](image)

**Figure 13. Block Diagram During Audio Haptics**

In this scenario, the capacitive touch buttons are not used. Instead, haptics are generated continuously by the audio haptics engine in response to an input audio signal that you provide on the audio jack. You must select this mode by selecting the “AUDIO” position on the slider switch, and you again must select which actuator to use.
4.3.4 **PC Communication Block Diagram**

Finally, you can drive haptics from a PC. This scenario is used during gaming, or when driving haptics from the HapTouch GUI provided with this application report.

![Block Diagram While Interfacing with a PC](image)

**Figure 14. Block Diagram While Interfacing with a PC**

In this case, the I2C interface is now used, in order for the PC to communicate with the BoosterPack via the LaunchPad. Haptics may be triggered by the HapTouch GUI on the PC; or during gaming, haptics are still driven locally by the touch buttons, but the status of the buttons is reported to the PC.

5 **Using the HapTouch BoosterPack**

5.1 **Flashing Firmware Images to BoosterPack and LaunchPad**

The batch files inside the directory \MSP430_One_Click_Download_Images quickly program firmware images into the BoosterPack and LaunchPad. This is the fastest way to get started with using the BoosterPack.
These firmware images are exactly the same as what you would accomplish by installing Code Composer Studio, importing the projects in the \MSP430_Source_Projects directory, building, and downloading the code. Using the one-click images is faster, but unlike with Code Composer, it is not possible to step through the code this way.

The batch files do their work by invoking the MSP430Flasher program. This program spawns a console ("DOS prompt") window, and status text begins to display. It is important to configure the hardware in the way the MSP430Flasher expects.

### 5.1.1 Programming the LaunchPad

The LaunchPad has its own emulator, capable of programming the target MSP430’s flash memory. Simple insert an MSP430G2553 into the socket, attach the LaunchPad to the PC via USB, and double-click on the ProgramTheLaunchPad.bat file. Make sure no other MSP430 hardware tool is also attached to the PC.

When successfully completed, the DOS window opened by the MSP430Flasher should appear as in Figure 16. Note the text highlighted in the red box; this indicates success.
Note that the emulator on your LaunchPad may potentially need a software update; if so, the MSP430Flasher will ask you if you would like to perform one. You will not be able to proceed without doing so, so it is recommended. (Note that if you later use this emulator with different PC software, like an installation of Code Composer Studio, that software may ask to perform an update once again, in order to synchronize the emulator to the software.) This operation is very safe, but it is very important you do not remove the tool during the update process.

5.1.2 Programming the BoosterPack

Note that the HapTouch BoosterPack sold by element14 has the software already pre-programmed in the factory. If you bought this, then you do not need to program it yourself, unless you want to make changes.

Unlike the LaunchPad, the BoosterPack does not have its own emulator. It must be programmed via JTAG, using an MSP-FET430UIF (a “FET tool”).
First, insert the BoosterPack into the LaunchPad. With the default zero-ohm resistor configuration, the BoosterPack derives its power from the LaunchPad, rather than from the FET tool.

Then attach the FET tool’s JTAG cable to the keyed JTAG header on the BoosterPack, attach the FET tool to the PC via USB, and double-click on the ProgramTheBoosterPack.bat file. Make sure no other MSP430 hardware tool is also attached to the PC (including the LaunchPad programmed previously).

As discussed for the LaunchPad emulator in Section 5.1.1, the MSP430Flasher may ask to update the firmware in your FET tool, prior to programming the BoosterPack.

When the MSP430Flasher has finished, you should see similar output text as shown in Figure 16.

With both boards programmed, and with the BoosterPack inserted into the LaunchPad, attach the LaunchPad to the PC via USB.
5.2 The BoosterPack’s State Machine

The BoosterPack’s software implements the state diagram below, with some help from the LaunchPad software and the Touch2Key PC application. Together, these states enable the usage modes briefly described in Sec 2.

Throughout this document, the word state refers specifically to the state of the BoosterPack software; usage mode refers to how you, the user, interact with the BoosterPack, as it was described in Sec 2.

The diagram refers to LP and BP; these are the LaunchPad and BoosterPack, respectively.
Some important general points to draw from the state diagram in Figure 18 include:
- You can quickly move out of the GAME, EFFECTS, or TOUCHTUNE states by moving the BTN/AUDIO slider switch. It resets the board into either the SEQUENCE or AUDIO states. For example, if you want to move from EFFECTS to SEQUENCE, move the slider switch to AUDIO, then back to BTN.

- In fact, the only other way out of these states is to cycle power on the BoosterPack.

- The only way to the EFFECTS state is through the SEQUENCE state. (However, you can play individual effects from the HapTouch GUI from states besides EFFECTS.)

The TOUCHTUNE state is colored a darker gray, because it is a test mode, not a usage mode. For the same reason, it is not generally addressed in Section 5 of this document. It is assumed that unless you are tuning the buttons, you will not be entering this state.

Also be aware that the GUI is capable of interfacing to the BoosterPack while it is in any state except TOUCHTUNE. (It is not recommended to attempt GUI communication while the BoosterPack is in TOUCHTUNE.) Although it is valid to access the BoosterPack from the GUI while it is in AUDIO state, the only command recognized in that state is disable audio, which exits into SEQUENCE state.

A summary of the states is shown in Table 2.

### Table 3. Summary of the BoosterPack States

<table>
<thead>
<tr>
<th>State</th>
<th>Brief Description</th>
<th>AUDIO/BTN Position</th>
<th>Ways In</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQUENCE</td>
<td>Allows the user to experience four predefined haptics effect sequences</td>
<td>BTN</td>
<td>• AUDIO/BTN slider switch set to BTN at power up&lt;br&gt;• While in AUDIO state, move the switch to BTN at any time</td>
</tr>
<tr>
<td>EFFECTS</td>
<td>Allows the user to experience all 123 haptics effects</td>
<td>BTN</td>
<td>• While in SEQUENCE state, press the wheel’s left or right key</td>
</tr>
<tr>
<td>GAME</td>
<td>Allows the user to play PC-based games while experiencing the same sequences as the SEQUENCE state, plus the four wheel buttons now also generate predefined sequences.</td>
<td>AUDIO: Selects Player 1 when Touch2Key is started&lt;br&gt;BTN: Selects Player 2 when Touch2Key is started</td>
<td>• While in any state except AUDIO or TOUCHTUNE, start the Touch2Key PC application</td>
</tr>
<tr>
<td>AUDIO</td>
<td>Allows the user to experience audio haptics, where a changing audio input signal drives the haptics effects</td>
<td>AUDIO</td>
<td>• AUDIO/BTN slider switch set to AUDIO at power up&lt;br&gt;• While in the SEQUENCE, EFFECTS, or GAME states, move the switch to AUDIO</td>
</tr>
<tr>
<td>TOUCHTUNE</td>
<td>A special test mode for tuning the capacitive touch buttons with the TouchPro GUI</td>
<td>Either one</td>
<td>• Hold the pushbutton switch on the LaunchPad down while the boards are powering up. See Section 9 for more information about this state.</td>
</tr>
</tbody>
</table>
5.3 Standalone Haptics Usage Mode

This mode uses the SEQUENCE and EFFECTS states of the BoosterPack software. The SEQUENCE state allows you to experience four predefined haptics effect sequences on the SELECT/START/B/A buttons. The EFFECTS state lets you experience all 123 individual haptics effects in the MSP430TCH5E Haptics Library.

In this mode, the BoosterPack is not communicating with the PC, and therefore it only needs the LaunchPad for supplying power. (In other words, an MSP430G2553 does not need to be in the LaunchPad’s socket, and you do not have to be concerned with configuring it as described in Sec 4.1.2.

Either:
- Attach the LaunchPad+BoosterPack to a PC (or other USB host supplying power), having the AUDIO/BTN slider switch set to “BTN” before doing so; or…
- Simply move the switch into the “BTN” position while the BoosterPack is in any other state.

The BoosterPack’s LEDs respond by flashing several times, and then the SELECT/START/B/A button LEDs all remain on. This tells you it has entered the SEQUENCE state.

![starting in the sequence state](image)

Figure 19. Starting in the SEQUENCE State

In the SEQUENCE state, the SELECT/START/B/A buttons play pre-defined haptics sequences. (See the introduction to Section 2 for a discussion of the difference between haptics effects and sequences.)

Some default sequences are programmed into the BoosterPack when software is first loaded into it:
- SELECT button: Rolling Dice Sequence
- START button: Explosion Sequence
- Button B: Machine Gun Sequence
- Button A: Strong Heartbeat Sequence
Try pressing these buttons, and experience these haptics sequences. In addition to the haptics vibrations, the blue haptics driver indicator LED will light during the haptics drive.

![Haptics Driver Indicator LED, During Haptics Effects](image)

**Figure 20.** Haptics Driver Indicator LED, During Haptics Effects

The defaults shown above can be replaced with new ones you create. This could be done programmatically in the BoosterPack software, but you can also do it with the HapTouch GUI. The GUI can program new values into flash, so that the next time the BoosterPack powers up, it will use those sequences instead. See Section 5.6.

An example where the user might want to create new sequences is in gaming. With the HapTouch GUI, new sequences can be created for each of the buttons, and saved as a *game profile* for a specific online game.

Now, press the left or right side of the touch wheel.
Moving into EFFECTS State

The LEDs flash again, then remain off, and a haptics ramp effect occurs. This indicates the BoosterPack has moved into the EFFECTS state. In this state, any of the 123 individual haptics effects can be executed using the buttons.

Initially with the LEDs off, pressing the four pushbuttons -- START, SELECT, B, and A -- plays effects #0, #1, #2, and #3, respectively.

However, the wheel can be used to change these assignments. If you press the right side of the wheel, the LED embedded in the “A” button turns on, such that if you look at the LEDs from left to right, from the wheel to button “A”, you see a binary “00001b”. If you press the right button again, you now see “00010b”; then “00011b”, and “00100b”, etc. Pressing the left button counts down.

As you count these up or down, the haptics effect assignments of the START, SELECT, B, and A buttons change. A binary “00001b” causes the buttons to play effects #4-7. A binary “00010b” causes them to play effects #8-11, and so on. (A list of effects can be found in Appendix C.)

In this way, all 123 haptics effects can be experienced without a PC data connection. Figure 22 shows the relationship between the LEDs and the effects played.
Press to count LEDs down
Press to count LEDs up

Figure 22. Accessing All the Effects in EFFECTS State

This relationship can be generalized as saying that if \( n \) represents the binary number displayed on the LEDs, the SELECT button always plays the effect with index \( n \times 4 \), and the other buttons play indices \( SELECT+1 \), \( SELECT+2 \), and \( SELECT+3 \). (For more information about the haptics effects, you can read about them in the MSP430TCH5E Haptics Library Designer’s Guide.)

Once you have moved into the EFFECTS state, the easiest way to go back to SEQUENCE state is to move the AUDIO/BTN slider switch briefly to AUDIO, then back to BTN.

5.4 Gaming Usage Mode

The LaunchPad+BoosterPack can be used as a game controller for PC gaming at sites like www.virtualnes.com and www.classicsegaonline.com. These websites let you play vintage video games on your PC, using keyboard presses instead of the original game controller. The Touch2Key application, included in the software package, converts touches of the BoosterPack’s buttons into keystrokes on your PC, allowing you to play the games like you might have done in 1990.
The sections below tell you how.

5.4.1 Sync the Game Controller with Touch2Key

The task of Touch2Key is to convert presses of the BoosterPack’s capacitive touch buttons into keystrokes on your PC, just as if you generated them with taps on your keyboard. For example, perhaps a particular game uses the right arrow to move characters to the right. In this case, you would probably want a press of the wheel’s right side to generate a press of the keyboard’s right arrow. (The default mappings are set up for www.virtualnes.com. However, you can use the HapTouch GUI to change the mappings to whatever you want.)
The game controller must first be configured, and then synced with Touch2Key.

To get started, assemble your game controller. You need the LaunchPad to be programmed with the target software provided with this document, and configured as described in Section 4.1.2. You probably will want at least the plastic enclosure handles fastened, to make sure you don't actually press buttons from the underside of the BoosterPack.

Then, attach the game controller to the PC. Give it a couple seconds to enumerate on the PC. (If this is your first time enumerating the LaunchPad, it may need a little longer.)

Now, make your player 1 and player 2 selections. The AUDIO/BTN slider switch determines this. If the switch is positioned as AUDIO, this is player 1. (Note that this position is to your left, when facing the BoosterPack.) If positioned to the right, at BTN, this selects player 2. If you only have one game controller, it must be configured for player 1.

Finally, locate the Touch2Key Windows application within the provided software package. (Note that Touch2Key should only be started after your game controllers are attached.) Double-click on Touch2Key.exe.

\MSP430_HapTouch_BoosterPack\PC_Tools\Touch2Key
Using the HapTouch BoosterPack

Figure 25. Touch2Key

When Touch2Key starts running, it begins looking for LaunchPads attached to the PC. If it finds one (or two), it tries to sync with it as a game controller.

If it successfully syncs with the game controller, it reports:

![Touch2Key Dialog Box Showing It Found the Game Controller](image)

Figure 26. Touch2Key Dialog Box Showing It Found the Game Controller

If you try to run Touch2Key without the game controller attached, it will quickly report that it was unable to find a game controller and quit.

If Touch2Key finds two game controllers, it will sync with both of them. In this case, you should ensure one of the controllers is configured for player 1 and the other for player 2. Touch2Key will not flag an error if you have two game controllers set as player 1 or as player 2.

After pressing “OK” to the “Found a game controller” message, the application will minimize into the system tray, showing the TI icon below.

![Touch2Key System Tray Icon](image)

Figure 27. Touch2Key System Tray Icon
At the same time, the BoosterPack will display some LED transition effects that show you it has synced with Touch2Key. When they stop, one of two LEDs will remain lit: the ones embedded in the SELECT or START buttons. These indicate whether the controller is configured as player 1 or player 2. (Again, notice that the LED for player 1 is on your left, and the one for player 2 is on your right.)

From this point on, if you press buttons on the game controller, keystrokes will be generated in the system. It will go on this way until you quit Touch2Key, or unplug the game controller.

### 5.4.2 Accessing Touch2Key’s Controls

Touch2Key does not have many controls once it has synced, but the controls that are available can be accessed through the contextual menu in the system tray.

![Touch2Key Menu](image)

Figure 28. Touch2Key Menu

Of particular note is that this is one way to quit Touch2Key. You can also quit by removing the game controller via USB; this will cause Touch2Key to generate an error box and quit.

### 5.4.3 Testing the Connection

The keystrokes are not only directed at the game; they are directed at the entire desktop system. If a text editor program is open on your PC, and it has focus in Windows, it will receive the keystrokes from the game controller just as if from your keyboard.

To test your connection, you can try exactly this. Open a text editor, click inside the window, and press the “A” button; you should see the letter “x” appear in the text editor. You can see the default mappings in Table 4. (Note that some of the keystrokes for the other buttons are arrow keys, and thus do not always result in visible text activity.)

If this works, then your game controller and Touch2Key are properly configured.

### 5.4.4 Launch the Website: [www.virtualnes.com](http://www.virtualnes.com)

Although the game controller and Touch2Key could be used with a wide variety of gaming scenarios, they are designed to be used with the website [www.virtualnes.com](http://www.virtualnes.com). (Please note that this website is not affiliated with Texas Instruments.)
From the site’s opening page, clicking on “Games” brings up a page with a list of games.

The leftmost image in Figure 30 shows the default keystrokes to control the game as “player 1”. By default, the game controller uses these defaults, as well as the ones for “player 2”.

As an example, try running the timeless classic Super Mario Brothers game, by clicking the link shown in the rightmost image in Figure 30. The game will begin to load; this can take several seconds.

A recent Java Runtime Engine (JRE) must be installed on the PC, and if you do not have this, your browser may prompt you to download. You may also be asked to approve of running a Java application on your PC.

Eventually you should see an image like the one shown below.
Now, double-click on the gaming window. This directs Windows’ focus to that window, such that keystrokes will be directed there.

**Note:** If the game controller ever seems to not work, even though you have proven the connection on a text editor, your first action should always be to double-click in the gaming window. Improper focus is usually the source of this problem, and it can happen easily if you are switching focus to other windows on your PC.

If this screen does not appear, or if you cannot play the game using keyboard keystrokes the way the authors of the www.virtualnes.com website intend, the site has a “help” tab to assist in determining the cause of the failure.

### 5.4.5 Button-to-Keystroke Mapping

The mapping between the BoosterPack’s buttons and the Windows keystrokes is stored within the LaunchPad software, not within the BoosterPack or Touch2Key. The default build of the LaunchPad software contains defaults for “player 1” and “player 2” that match www.virtualnes.com.
Table 4. Default Game Controller / PC Keystroke Mappings

<table>
<thead>
<tr>
<th>Game Controller Button</th>
<th>PC Keystroke Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAYER 1 (AUDIO position)</td>
<td></td>
</tr>
<tr>
<td>Wheel up</td>
<td>Arrow up</td>
</tr>
<tr>
<td>Wheel right</td>
<td>Arrow right</td>
</tr>
<tr>
<td>Wheel down</td>
<td>Arrow down</td>
</tr>
<tr>
<td>Wheel left</td>
<td>Arrow left</td>
</tr>
<tr>
<td>SELECT button</td>
<td>Control</td>
</tr>
<tr>
<td>START button</td>
<td>Enter</td>
</tr>
<tr>
<td>B button</td>
<td>z</td>
</tr>
<tr>
<td>A button</td>
<td>x</td>
</tr>
<tr>
<td>PLAYER 2 (BTN position)</td>
<td></td>
</tr>
<tr>
<td>Wheel up</td>
<td>Number pad 8</td>
</tr>
<tr>
<td>Wheel right</td>
<td>Number pad 6</td>
</tr>
<tr>
<td>Wheel down</td>
<td>Number pad 2</td>
</tr>
<tr>
<td>Wheel left</td>
<td>Number pad 4</td>
</tr>
<tr>
<td>SELECT button</td>
<td>Number pad 3</td>
</tr>
<tr>
<td>START button</td>
<td>Number pad 1</td>
</tr>
<tr>
<td>B button</td>
<td>Number pad 9</td>
</tr>
<tr>
<td>A button</td>
<td>Number pad 7</td>
</tr>
</tbody>
</table>

The HapTouch GUI can be used to change these assignments, or restore them to these defaults; see Section 5.6.7.

Note that if you re-flash the LaunchPad as described in Section 5.1 that this, too, will overwrite any new mappings you have programmed with these defaults, because the defaults are stored within the source code project for the LaunchPad software.

The pressing of multiple buttons at the same time results in all the buttons being converted to keystrokes. As a result of this, pressing the wheel in the “corner” positions results in two simultaneous keystrokes. For example, pressing the upper-right corner on the “player 1” controller results in both an “arrow up” and “arrow right” down-keystroke.

Once the game controller has been set as “player 1” or “player 2”, it remains so until Touch2Key is restarted. Therefore, if it’s desired to change the player at any other time, shut down Touch2Key and restart it.

5.4.6 Start Gaming

Gaming can begin if all the following conditions are correct:

- pressing buttons on the game controller results in keystrokes posted to a text editor
- the game can be played in its usual way with keyboard keystrokes
- at least one game controller is configured as “player 1”
Many games are available at www.virtualnes.com, as well as its related sites. Similar sites also exist elsewhere on the web. Because the button-to-keystroke mapping can be changed in the LaunchPad software, the game controller can be used with any game that accepts keystroke control. It could even be used in non-game applications – any application where pressing a button should result in a PC keystroke.

A problem that can commonly happen is for focus to shift to a window besides the gaming window. Focus is what happens when you click on a window in most GUI operating systems: that window comes to the forefront, giving it focus; and the window which previously had focus no longer does. If focus shifts away from a window, then keystrokes no longer get posted to it.

The same applies with the game. If the controller does not seem to be working with www.virtualnes.com, double-click inside the gaming window; this shifts focus to it.

5.4.7 Programming Game Profiles into the Game Controller

The HapTouch GUI lets you program these two aspects of gaming:

- The haptics sequences played by each button
- The button-to-mapping keystrokes for each button

Note that the haptics effects are not driven from the gaming website, because these effects were not written into the game. Instead, the effects are driven from the capacitive touch buttons.

Since the buttons evoke different actions in different games, it makes sense that the haptics sequences driven from the buttons should change, depending on which game you are playing. For example, if in a given game the “A” button shoots a machine gun, then the effects should feel like a machine gun. If it causes a character to jump, the effect should be short and sudden.

The HapTouch GUI collects all this information into a game profile. You can build a game profile and then program it into the game controller.
Figure 32. Setting a Button Profile for Gaming, in the HapTouch GUI

The GUI has profiles and sequences pre-loaded, for two games:

- Super Mario World
- Tecmo Super Bowl

Figure 33. Predefined Game Profiles
Notice that there are separate profiles for player 1 and player 2. This is because the keystroke mappings are different. If you want a controller to be able to function as both player 1 and player 2 for a given game, just by changing the AUDIO/BTN slider switch, program both profiles into the game controller.

See Section 5.6 for a discussion of how to use the GUI in this way.

5.5 Audio Haptics Usage Mode

Immersion’s haptics technology contained within the MSP430TCH5E Haptics Library enables a unique feature: audio-to-haptics. The audio haptics engine within the library samples an audio signal and generates corresponding haptics effects on a continuous basis. Low signal amplitudes produce mild effects; high ones produce strong effects. The result of this is an enriched and enhanced audio experience.

Please note that the audio haptics feature works optimally with the ERM actuator, and not as well with the LRA actuator. This is due to the characteristics of the LRA actuator.

5.5.1 Setting It Up

To enable this mode, move the AUDIO/BTN slider switch to AUDIO. You can either do this prior to powering up the BoosterPack, or at any time during operation.

In this mode, the BoosterPack is not communicating with the PC, and therefore it only needs the LaunchPad for supplying power. As such, an MSP430G2553 does not need to be in the LaunchPad’s socket, nor does the LaunchPad need to be configured as shown in Section 4.1.2.

Then, connect the BoosterPack’s audio jack with an audio source using a 3.5-mm male-to-male stereo cable.

Figure 34. Setting the Audio-to-Haptics Input Signal Configuration in the BoosterPack Source Code

The audio haptics engine inside the MSP430 Haptics Library needs to be configured with the amplitude of the signal you are inputting, for best results. The BoosterPack application configures this for the line-out jack on an Apple iPod Classic. (Note that the headphone output is different from the line-out output.) These settings may or may not work with other sources. (See Section 5.6.9 for information on configuring for other sources.)

When the BoosterPack is powered, the slider is configured, and an audio source is connected, the BoosterPack’s haptics will begin to operate in concurrence with the audio source.

Note that many audio devices having an internal speaker will disable that speaker when something is connected to the line-out jack. If this happens, it may be necessary to use a “Y” adapter that directs audio to both the BoosterPack and an external speaker.
5.5.2 Configuring the BoosterPack for Other Audio Sources

You can configure the settings for other audio sources using the Hap-Touch GUI.

Figure 35. Setting the Audio-to-Haptics Input Signal Configuration in the BoosterPack Source Code

To do this, you must understand what the values shown here mean. You can read about this in the MSP430TCH5E Haptics Library Designer’s Guide.
5.6 GUI Usage Mode: the HapTouch GUI

For an alternate interface to the BoosterPack, and also to access some functionality not otherwise available, the HapTouch GUI is provided. The GUI can be used to:

- select the best effect for your situation
- build and save sequences of effects
- write new predefined sequences into the BoosterPack’s MSP430TCH5E flash memory
- build and save game profiles, with each associated with a given game, and write them into the BoosterPack and LaunchPad’s flash memory.
- share new sequences and game profiles with others

![The HapTouch BoosterPack GUI](image-url)

**Figure 36. The HapTouch BoosterPack GUI**
The MSP430TCH5E Haptics Library includes an I2C serial command interface. With this interface, an external host MCU can drive the MSP430TCH5E as a fixed-function haptics controller. The HapTouch GUI essentially acts in the role of this host MCU, sending these commands through the LaunchPad to the BoosterPack’s MSP430TCH5E. Each section in the GUI window correlates with one of these serial commands.

5.6.1 Configuring the LaunchPad and BoosterPack

This usage mode requires the LaunchPad to be fully configured, as described in Section 4.1.2.

5.6.2 Launching and Running the GUI

To launch the GUI, double-click on HapTouch_GUI.exe within the target directory, shown in Figure 37. If you installed the software package in the default directory, the path will be as shown in the figure, substituting the letter x with the software revision you are using.

```
C:\ti\msp430\MSP430_HapTouchGui_x.xx.xx.xx\HapTouch_GUI.exe
```

(need latest)

Figure 37. Launching the HapTouch BoosterPack GUI

The GUI communicates with the LaunchPad using its backchannel UART, as described in Section 4.2. It sees the backchannel as a virtual COM port on the PC. The LaunchPad software relays the communication to the BoosterPack over I2C.

The GUI automatically finds LaunchPads connected to the PC. It shows all connected LaunchPads in a pulldown menu.

As you press the buttons on the left side of the GUI window, it sends serial commands to the BoosterPack through the LaunchPad. The GUI keeps the COM port closed most of the time; it only opens it during a command transaction. Because of this, it is safe to disconnect the LaunchPad anytime a command transaction is not taking place.

As commands are sent to the virtual COM port, each byte is displayed in the fields at the bottom of the window.

```
Command Sent: 0x12 0x50 0x23 0xa 0x17 0xa 0x23 0xa 0x17 0xa 0x23 0xa 0x17 0xa 0x17 0x
Response Received: 0x1
```

Figure 39. HapTouch GUI: Bytes Sent and Received
Later, if you are studying the serial interface provided with the MSP430TCH5E Haptics Library, you can compare these values to the commands specified in the MSP430TCH5E Haptics Library Designer’s Guide. For example, a response code of 0x01 means the command was successful.

### 5.6.3 Play Effect

The **Play Effect** button plays an individual haptics effect. These are the simple building blocks from which **sequences** are made. There are 123 available effects, indexed by values 0-122.

![Figure 40. The “Play Effect” Command, with the Effect Finder Button](image)

You can also set:

- a **timeout** value, after which the effect will timeout. (This generally is useful only for effect #117, which can last ten seconds.)
- whether the effect will **override** any effects currently playing

123 effects is a lot to choose from! Although you can reference Appendix C to see a complete list, another way to narrow down your options is with the **Effect Finder**.

![Figure 41. The Effect Finder](image)

Use the menus to narrow down the effect you want, click on it, and click on **Close**. The index you selected will be in the Effect Index field. Press **Play Effect**, and the effect will be played on the BoosterPack.

### 5.6.4 Play Sequence

The **Play Sequence** button plays haptics **sequences**, which are a series of effects chained together, separated by programmable time delays.
Figure 42. The “Play Sequence” Command, with the Sequence Builder Button

As with the Play Effect command, you can select whether to override any effects or sequences currently playing.

Several predefined sequences can be selected with the pulldown menu. But you can also build your own sequences, using the Sequence Builder.

Figure 43. The Sequence Builder

To begin, click New. This prompts you to name your new sequence. After doing so, you can select it from the pulldown menu.

Figure 44. New Sequence

You can now build this sequence. Select the first Effect Index and the Wait Gap (time delay) that will follow it before playing the next effect. Then, press Add. It appears in the list box.
Keep doing this for as many effects as you want in your sequence. You can also move the effects around in the sequence, using the **Up** and **Down** buttons. Or, you can remove them with **Remove**.

Finally, set the **Repeat** value.

When the sequence plays, it will be repeated this many times. A value of zero means the sequence will only be played once.

There are a couple buttons at the bottom of the **Sequence Builder** box.

The **Effect Builder** button launches the same Effect Builder discussed in the last section. It helps you find the effects that you need to build your sequence.

The **Save** button saves your changes and also closes the box.

**5.6.5 Stop Playback**

The **Stop Playback** button stops any currently-playing effects or sequences.

**5.6.6 Assign Sequence to Button**

As discussed in Section 5.3, the SELECT, START, B, and A capacitive touch buttons on the BoosterPack play pre-defined sequences during the SEQUENCE state, and also in the GAME state. The **Assign Sequence to Button** command can be used to program new predefined sequences.
Using the HapTouch BoosterPack

5.6.7 Assign Game Profile

When used as a game controller, there are two aspects of the BoosterPack that usually need configuration:

- The haptics sequences played by each button
- The button-to-mapping keystrokes for each button

The HapTouch GUI refers to these collectively as a game profile. You can build a game profile and then download it to the game controller. Each game profile is probably specific to a particular game on a particular gaming website or platform. It is probably also specific to player 1 or player 2, since each profile only contains a single button-to-keystroke mapping.

Figure 50. The “Assign Game Profile” Command, with the Game Profile Builder Button

To begin building a profile, click on the Game Profile Builder button.
The Reset Key Maps button resets the keystrokes to the ones used by www.virtualnes.com, for player 1.

Note that the button-to-keystroke mappings are not stored in the BoosterPack; rather, they are stored in the LaunchPad. The sequences, however, are stored in the BoosterPack.

As such, if you re-flash the LaunchPad with the software image as described in Section 5.1, it will replace any assignments you make with the GUI, using the ones stored in the LaunchPad code. Similarly, if you re-flash the BoosterPack with the software image as described in Section 5.1, it will replace any assignments you make with the GUI, using the ones stored in the BoosterPack code.
5.6.8  **Enable/Disable Audio**

This button puts the BoosterPack into either the AUDIO state (**Enable**) or the SEQUENCE state (**Disable**), depending on the value selected in the pulldown menu. Configure the menu, and then press the **Enable/Disable Audio** button. The BoosterPack’s LEDs will show some transition effects, to convey that it has changed modes.

![Enable/Disable Audio Button](image)

**Figure 52. The “Enable/Disable Audio” Command**

Note that while the BoosterPack is in the AUDIO state – whether it arrived there from the GUI or using the AUDIO/BTN slider switch -- most commands sent from the GUI will receive a response code indicating that the command was invalid while in the AUDIO state. During audio haptics, the BoosterPack software is very busy handling that function; and other commands received would be disruptive, and would not appear very well in the middle of the continuous haptics effects that the audio generates. As a result, the only command that the BoosterPack will receive in the AUDIO state is the **Enable/Disable Audio** button, configured for **Disable**.

5.6.9  **Audio Configuration**

The audio haptics feature needs to know what kind of audio signal is being sent into the jack. The default settings are set up for the line-out signal of an Apple iPod Classic. These settings may or may not work with other sources.

![Audio Configuration](image)

**Figure 53. Setting the Audio-to-Haptics Input Signal Configuration in the BoosterPack Source Code**

Configuring these requires a knowledge of the input waveform, especially its amplitude. For example, line-out jacks typically have different amplitudes than microphone jacks. However, there is no clear standard.

5.7  **Changing Haptics Actuators**

The BoosterPack has two haptics actuators on it:
• an Eccentric Rotating Mass (ERM) actuator
• a Linear Resonant Actuator (LRA)

These actuators provide a tradeoff in performance, lifetime duration, and mechanical form factor. They also simply feel different from each other.

You can select which actuator is used, with the ERM/LRA slider switch. All 123 effects (and sequences of effects) can be driven to either actuator; however, a different effect bank is used by the haptics library, depending on which actuator is used.

Figure 54. Changing Actuators
6 Emulating the Software: Stepping, Debugging, Developing

After using the BoosterPack, you may wish to begin studying how the code works, or write your own applications. This section describes requirements and steps for doing so. You may also wish to refer to the sections after this one, which describe the design of the hardware and software.

6.1 Tool Requirements

Tools needed for full emulation include the following. (Limited emulation is possible with a subset of these, as will be described.)

- The Code Composer Studio IDE (CCS)
- A LaunchPad (for similar reason as described in Section 4)
- An MSP-FET430UIF FET tool (or possibly a LaunchPad an open, software-updatable emulator like the F5529 LaunchPad’s eZ-FET Lite)

Any emulation will require CCS. The free version of CCS compiles up to 16KB of object code, and so it will work with any MSP430TCH5E application. The code was developed using CCS v5.4.0; this is the first version of CCS that included support for the MSP430TCH5E.

Note that CCS is the only IDE supported at this time. There is no reason the software could not be ported to IAR or GCC; however, it will mostly likely not build on these platforms without modifications, because it makes use of special linker segments to store data. Methods of doing so tend to vary from one IDE to another.

Programming of the BoosterPack software generally requires a FET tool. Without a FET tool, you can only use the software that shipped on the board. The G2 LaunchPad does have an emulator on it, but it does not have the ability to recognize MSP430 derivatives not overtly supported by the G2 LaunchPad. (The emulator on some LaunchPads, like the F5529 LaunchPad, should have the ability to recognize the MSP430TCH5E, with appropriate software updates of the emulator software. If updated in this way, and then jumper-wired to the HapTouch BoosterPack’s Spy-Bi-Wire interface, it should theoretically be possible to emulate from these emulators.)

Emulation of the LaunchPad software can be done using its integrated emulator, combined with CCS. If you do not have a FET tool or other means of emulating the BoosterPack, you might choose to drive haptics by writing LaunchPad code that sends serial commands to the BoosterPack, essentially treating the MSP430TCH5E as a fixed-function device.

6.2 Setting Up Full Emulation for the First Time

A full emulation setup can be created as follows. It is assumed that the user has a basic familiarity with CCS, and therefore much of this discussion focuses on the idea of emulating two targets on the same PC at the same time, which is something many have not have done before.

The full setup is shown in the figure below.
Notice that two separate, simultaneous CCS instances are shown. You can run these on the same PC, at the same time. Simply launch CCS twice. As CCS starts running, it asks you for the workspace location. It is recommended to keep separate workspaces for each instance – perhaps C:\CCS_projects1 and C:\CCS_projects2.

It is an advisable practice with any use of CCS (or Eclipse IDE's in general) to put these workspaces in the root directory of your PC. This is because Eclipse builds very deep directory paths in its projects, making it easy to exceed your operating system’s path limit. Even worse, sometimes the reason for the failure is not obvious. Putting them in the root directory, and keeping the workspace and project names short, generally eliminates that problem.
After launching these separate CCS instances, having separate workspaces, import the BoosterPack and LaunchPad projects into their respective workspaces, by choosing Project→Import Existing CCS Eclipse Project.

![Image of CCS project import]

**Figure 56. Importing a Project in CCS**

In the following dialog box, locate the BoosterPack or LaunchPad project directory. If you used the default installation path for the HapTouch SDK, these will be located at the paths below. (The letter "x" has been substituted for specific version numbers.)

C:\ti\msp430\MSP430Haptics_x_xx_xx_xx\MSP430_HapTouch_BoosterPack\MSP430_Source_Projects\BoosterPack

C:\ti\msp430\MSP430Haptics_x_xx_xx_xx\MSP430_HapTouch_BoosterPack\MSP430_Source_Projects\LaunchPad
When importing a project, choose the directory containing the \.settings directory. This helps CCS find the appropriate directory and files.

When you press OK, CCS should find the project; click OK again to proceed with the import.

Do this in one CCS instance for the BoosterPack project. Then, switch to the other CCS instance and do it for the LaunchPad.

Then, attach your LaunchPad (configured as described in Section 4.1.2) and your FET tool. When CCS finds more than one MSP430 emulator attached to the PC, it identifies them by a sequential number: 1, 2, or 3. The determination is made according to which emulator this particular PC initially encountered first. Since you may not remember which this was, this is often something that must be determined by trial and error, described below.
Each project has a setting for this number. Right-click on the project in the Project Explorer, and from the resulting contextual menu, select Properties. The window below should appear. Select **CCS General** from the tree at left, and see the item at right labelled **Connection**.

![Properties for HapTouch_BoosterPack](image)

**Figure 58. Setting Which Emulator Will Be Used**

In this case, “USB1” is selected. As distributed by TI, the BoosterPack project is configured as USB1, while the LaunchPad project is configured as USB2. This may happen to be the correct setting for your PC, or it may be opposite of what you need, depending on the order in which your PC first encountered your LaunchPad and FET tool. Below, we will see how to find out whether this setting is correct for you.

Notice that the chosen variant (the specific MSP430 device, also called a **device derivative**) is MSP430TCH5E. Since the BoosterPack has an MSP430TCH5E, and the a LaunchPad has an MSP430G2553, we can use this to help figure out whether our USB1 and USB2 assignments are correct.

Make sure both your LaunchPad and FET tool are attached to the PC via USB. Make sure the LaunchPad is equipped with a G2553, and make sure the FET tool is attached to the BoosterPack’s JTAG interface. Green power LEDs on each one should be lit.

Then attempt to emulate the BoosterPack project, by clicking the **Debug** icon:

![Beginning Emulation](image)

**Figure 59. Beginning Emulation**
If you receive an error saying that the chosen derivative (MSP430TCH5E) does not match the target MCU (an MSP430G2553), this means the project tried to download to the LaunchPad instead of through the FET tool. In this case, you must reverse the connection settings; the BoosterPack project needs to be set as USB2, and the LaunchPad needs to be set as USB1.

Note that if you only have one emulator attached, CCS will always assign it as USB1; even the one that is normally assigned USB2 when both emulators are attached. This can confuse the settings. The easiest solution is usually to keep both emulators attached, even if you are only using one of them. If this is not possible, you can change the USB2 setting to USB1 on the target project.

If you only intend to emulate the BoosterPack without any communication connection with the PC – that is, only the standalone usage mode described in Section 5.3 – then you do not need to enter emulation on the LaunchPad. However, it is helpful to keep the LaunchPad attached via USB so that the emulator connections are what CCS expects.

### 6.3 Emulating Both the LaunchPad and BoosterPack Simultaneously

If I2C commands are to be sent, then keep in mind that the code makes certain timing assumptions. For example, if you halt execution on the BoosterPack and then send an I2C command from the LaunchPad, the LaunchPad code will hang in the middle of the I2C transmit code, because it needs the BoosterPack software to pull the received data from the USCI module’s interface.

For this reason, it helps to be mindful of these interactions when halting execution. Even when being very mindful, it is easy to put one or both MCUs into a confused state. If this happens, the best thing to do might be to reset execution on both sides and start over. Halting execution on the LaunchPad side is generally a little safer, because it makes more assumptions about the BoosterPack’s behavior than vice versa.

If emulating while commands are sent from the HapTouch GUI, and breakpoints are set on either the LaunchPad or BoosterPack sides, this has the potential to get all three sides confused. This is because the GUI implements timeouts on all commands, which will always expire when breakpoints are encountered. The GUI, in particular, sometimes needs to be shut down and restarted. Usually when doing so, it is a good idea to reset execution on the LaunchPad and BoosterPack as well.

### 6.4 Emulating the BoosterPack Without the LaunchPad, or With Another PCB

As described in Section 4.1, the BoosterPack receives its power from the LaunchPad. This allows both to share a common power rail, so that the I2C lines do not exceed absolute maximum voltage specs on either device.

However, it is possible to change this. There are two scenarios in which you might wish to do so:

- If you do not have a LaunchPad, and you wish to emulate the BoosterPack by itself
- You want to interface the BoosterPack with a different PCB via the external host interface, using an MCU on that PCB as a host to the BoosterPack’s MSP430TCH5E

The *MSP430 Hardware Tools User’s Guide (SLAU278)* describes how pins 2 and 4 on the JTAG connector determine whether the emulator is to provide power into the circuit under test, or whether power is already present and instead its task is simply to verify that this power is present. An excerpted diagram from that document is shown below.
Figure 60. Proper Spy-Bi-Wire (Two-Wire) JTAG Connections

The red rectangle shows this selection. In comparison, see Figure 61, which is an excerpt from the BoosterPack’s schematic.

Figure 61. The BoosterPack’s JTAG Connections: FET Tool Power Source or Sense
Notice that within the blue rectangle, R6 is not populated, and R7 is shorted. With this configuration, the FET tool will sense power, rather than source it.

Therefore, if you want to emulate the BoosterPack without a LaunchPad, it is possible to do so by moving the zero-ohm resistor from R7 to R6. However, if the LaunchPad is later attached, the LaunchPad’s power supply to the target should be severed, so that the LaunchPad’s target G2553 now gets its power from the BoosterPack. This ensures that the voltage sources on the two emulators do not drive into each other.

### 6.5 Encountering Breakpoints During Haptics on the BoosterPack

If a breakpoint is encountered in the middle of a haptics effect, the PWM duty cycle will remain at whatever level it happened to be at the moment the code halted, and the actuator will continue to move. If the duty cycle happened to be high, the actuator might run at a fast rate. Although not dangerous (i.e., it does should not cause any damage), it can be a little annoying.

One way to deal with this is to disable the driver device enable signal in the `enableActuatorCB()` callback function, if you know you will be using breakpoints. Keeping the enable signal de-asserted will allow code to progress, and breakpoints to be quietly hit; but at the expense of not experiencing any haptics.

Another option that solves the latter problem is to add code that temporarily reduces (during debug only) the PWM duty cycle value in `setActuatorDriveLevelCB()`. In other words, modify the value passed in from the library before applying it to Timer_A; perhaps add a divide-by-two. This would halve the effects to a lower level, so that when the breakpoint is encountered, the actuator still remains fixed, but in a more subdued fashion.

The LRA tends to be less distracting than the ERM actuator, so switching to this actuator might be another way of reducing this effect.
7 Hardware Design

The remainder of this application note describes the design of the BoosterPack. It adds to the information provided inside the MSP430TCH5E Haptics Library Designer’s Guide, as well as the Capacitive Touch Software Library Programmer’s Guide. These guides contain information common to any design using these library. In contrast, this document covers things specific to the HapTouch BoosterPack, as well as practical information about applying the library.

This particular section specifically discusses the hardware. This includes the BoosterPack hardware and also the LaunchPad’s I2C connection to the BoosterPack.

Keep in mind that both the MSP430TCH5E on the BoosterPack and the MSP430G2553 on the LaunchPad are G2xx devices. As such, their architecture is described by the MSP430x2xx Family User’s Guide (SLAU144).

7.1 Power Supply

To simplify the design and minimize the chance of violating absolute maximum voltage specifications for either the MSP430TCH5E or the LaunchPad’s MSP430G2553, the BoosterPack and LaunchPad share the same power rail, through the 20- or 40-pin BoosterPack interface. By default, the LaunchPad’s emulator is the source of this power.

It is possible to derive power from an alternate source:

- A FET tool connected to the BoosterPack
- Any other source attached to the external host interface’s VCC pin

You should ensure there is only one voltage source in the circuit; that both the host MCU (the LaunchPad or other PCB) and BoosterPack share that source; and that the FET tool is configured appropriately for emulation depending on whether the voltage source is the FET tool itself or some external source (see Section 6.4).

7.2 Clocks

The 2xx architecture, on which both the MSP430TCH5E and MSP430G2553 are based, provides these clock options:

- DCO: high-frequency, good-precision, low-power. (To achieve this good precision, use the factory-calibrated values in its flash memory for frequencies 1, 8, 12, and 16 MHz.
- VLO: low-frequency, low-precision, ultra-low-power. This is a good choice as a keep-alive clock.
- LFXT1: low-frequency, crystal-precision, ultra-low-power. This can be used as a precision keep-alive clock, or other low-frequency applications; however, the crystal represents a small cost adder to the design.

The software for both the BoosterPack and LaunchPad do not assume the presence of a crystal. They both use the DCO for MCLK and SMCLK, and the VLO as a low-frequency keep-alive clock.

See Sections 8.1.3 and 8.2.8 for a description of clock configuration in software, on the BoosterPack and LaunchPad, respectively.
7.3 Actuator and Driver Device

The basic function of the library is to manage a timer that outputs a pulse-width modulated (PWM) signal. The duty cycle of the PWM determines how the actuator is driven, with the relationship generally being that the higher the duty cycle, the faster the actuator runs. The library changes the duty cycle dynamically to achieve one of 123 haptics effects.

The library allows either of the MSP430TCH5E’s Timer_A modules to be used for this, as well as any of the outputs on those timers. The MSP430TCH5E also controls an enable signal to the driver device, implemented via a general I/O.

The library can be used with one of two haptics driver devices: TI’s DRV2603 or DRV8601. This BoosterPack uses the DRV2603, shown in Figure 62.

![Figure 62. Haptics Driver Device and Actuators](image)

The signals MOTOR_PWM and MOTOR_EN are driven from the MSP430TCH5E, and were described above. The same enable signal drives an indicator LED on this board. Keeping the DRV2603 disabled when not driving the actuator makes a significant difference in its power consumption.

The DRV2603 incorporates advanced features, including the use of auto-resonance to determine when to apply braking or overdrive voltages. The result of this is maximized performance and efficiency. It also makes it easier to use LRA actuators (Linear Resonant Actuators), which are only resonant within a very narrow frequency window. (LRA actuators are growing in popularity because of their performance and power efficiency; see the Designer’s Guide for more information.)

For tips on obtaining data sheets for these driver devices, see Section 11.3.
The BoosterPack has both an ERM (Eccentric Rotating Mass) and LRA actuator. The user selects this using a double -pole, double-throw (DPDT) slider switch. This switch directs the driver’s OUT+ signal to either of the actuators, which in turn share its differential, OUT-, in common. At the same time, the switch enables a pullup or pulldown on the ERM/LRA signal, which goes to both the MSP430TCH5E and the DRV2603. The DRV2603 drives the LRA differently than it does the ERM actuator, and so it uses this signal to know how to drive. The MSP430TCH5E must configure a different haptics effect bank for LRA compared to ERM actuators. (An effect bank is essentially a map between the desired actuator speed and duty cycle, which accounts for variations between actuator specifications.)

7.4 Touch Buttons, Wheel, and LEDs

TI MSP430 enables the adoption of capacitive touch controls using its touch-enabled IO pins, in conjunction with the MSP430 Capacitive Touch Library (SLAA490). These can be used to implement buttons, wheels, sliders, and any other touch control.

For gaming, buttons are a good choice. The BoosterPack achieves eight-directional control by using a button for up, right, down, and left, and then also registering the combination of any adjacent buttons. In Figure 63, notice that the boundaries of the buttons on the overlay at right, are drawn in relation to the copper at left, to help achieve this.

![Figure 63. The Wheel: PCB and Overlay](image)

Four buttons are then implemented for SELECT, START, B, and A. Ground fill around all these buttons completes the capacitor formed in the touch circuit.

Buttons that have discrete on and off states are inherently easier to tune than controls having incremental values, like a slider or wheel. For the same reason, they are also more immune to noise. For more information on the tuning of the buttons, see Section 9.

Five button-corresponding LEDs are implemented. Four of them are located within the SELECT, START, B, and A buttons; a fifth is located within the wheel. Together they enable counting up and down through the effects in the software’s EFFECTS state.
One can see the correlation in signal names between the LEDs and buttons. PROX_BTN was provided for implementation of proximity sensing. This was not implemented in the software, but there is no technical reason it could not be done.

7.5 Audio Interface

The MSP430TCH5E Haptics Library has a unique feature that converts an audio signal into haptics effects. In general, the higher the amplitude, the faster the actuator moves. The resulting effect is, in some ways, similar to a subwoofer. Another way of describing the effect is that it is similar to video games that shake the controls during explosions; this feature could be used to implement the same experience in your device.

The library requires an input in the form of an 8-bit streaming sample of an audio source. This source is typically the MSP430TCH5E’s ADC10 module, with the two least significant bits discarded. Since the entire digitized version of the analog waveform is expected to exist within the 8-bit space, the amplitude can be maximized by biasing the signal at mid-voltage.

VREF+ is the buffered output of the ADC10’s reference voltage. It goes through a 50% voltage divider to achieve a bias at mid-voltage, maximizing the possible amplitude of the incoming signal. The resistor values are selected to not exceed the VREF+ maximum load current, which is indicated in the data sheet.
The 10-µF series capacitor decouples the input signal as it arrives from the audio source, which is then superimposed on the bias voltage.

Finally, a low-pass anti-aliasing filter is incorporated. The library causes the audio signal to be sampled at a 4-kHz rate; therefore, the analog input should not exceed 2 kHz, per Nyquist, to avoid aliasing. R10 and C6 combine to form a passive low-pass filter with a cutoff frequency of \( \frac{1}{2\pi RC} = 1940 \text{ Hz} \). (Since audio typically has components above 2 kHz, it can be seen that the library responds to lower-frequency audio components only.)

Note that, using the 50% voltage divider, either of the ADC10’s reference voltages (1.5 V or 2.5 V) could be used. However, using 2.5 V creates a wider input range, allowing the highest compatibility with most audio sources without saturating the input.

7.6 LaunchPad \( \leftrightarrow \) BoosterPack Connections

Only a few connections are made between the LaunchPad and BoosterPack. Mostly they fall into the categories of:

- Power and ground
- I2C
- RESERVED0 and RESERVED1

Figure 66. LaunchPad \( \leftrightarrow \) BoosterPack Connections

The I2C pullups can be seen in this schematic excerpt. A power LED is provided, to let the user know the BoosterPack is operational.

Zero-ohm resistor options are provided for pulling RESERVED1 high or low. These are not used by the software, but they are provided for the benefit of the person that wants to use them for application-specific purposes.
The main reason zero-ohm resistor R1 is provided is for the measurement of power consumption on the BoosterPack. R1 can be removed and replaced by ammeter probes in series. Alternatively, it could be used to isolate power between the LaunchPad and BoosterPack, but if doing so, it is important to ensure that the voltages on either board match, since they make connection via I2C.

7.7 External Host Header

As described in the Designer’s Guide, the MSP430TCH5E with the haptics library can be used either as a standalone, integrated-function device, or as a fixed-function serial slave to another host MCU. In the context of most of this document, the LaunchPad acts in the role as host. However, if developing your own application, you may wish to interface the BoosterPack to another PCB. The external host header is provided for this purpose.

Figure 67. External Host Header and MSP430TCH5E Connections

In Figure 67, the external header is shown at top left. Its connections to the MSP430TCH5E are shown at top right. (These are excerpted from the schematic in Appendix A.) Finally, the MSP430TCH5E data sheet is excerpted at the bottom, so you can see the full list of options available for these pins.

One can see that I2C is implemented with the USCI_B0 module, using the UCB0SDA and UCB0SCL pins. One can also see that the RESERVED0 and RESERVED1 pins provide access to additional USCI pins that could enable serial communication using SPI or UART. They are marked as reserved in this schematic, because the software does not implement them. However, the option is made available in hardware.

8 LaunchPad and BoosterPack Software Design

This section focuses on the design of the LaunchPad and BoosterPack software. Software projects are provided for both boards for use with Code Composer Studio. (As discussed in Section 6.1, CCS is the only IDE supported at this time.)

The information in this section builds on a knowledge of the system gained in previous sections, especially the state diagram in Fig. 5.2.
8.1 LaunchPad Application

This application is fairly simple. At a fundamental level, it acts as a UART-to-I2C bridge between the LaunchPad’s backchannel UART (and thus the PC), and the MSP430TCH5E’s I2C interface.

In actuality, it does a little more than that:

- It has an awareness of the commands the MSP430TCH5E is allowed to receive, and will reject any other command code.
- It knows that the BoosterPack will take longer to respond to certain commands than others, and adjusts its I2C wait time accordingly.
- In the case of two of the commands it can receive from the PC, it does not act as a bridge, but rather acts directly as the target of the command from the PC, and acts directly as the master to the MSP430TCH5E.

8.1.1 File Organization

The directory structure for the example project is shown below.

![Project Structure](image)

Table 5 describes the purpose of each file.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>main()</td>
</tr>
<tr>
<td>main.h</td>
<td>A couple high-level configuration constants.</td>
</tr>
<tr>
<td>commands.c/h</td>
<td>Command handling.</td>
</tr>
<tr>
<td>I2C.c/h</td>
<td>Any code related to low-level I2C, except the USCI ISR.</td>
</tr>
<tr>
<td>UART.c/h</td>
<td>Any code related to low-level UART, except the USCI ISR.</td>
</tr>
<tr>
<td>UART_I2C_ISR.c/h</td>
<td>Given the way that the USCI interrupt vectors are configured, the I2C and</td>
</tr>
</tbody>
</table>
UART functions must share the same set of ISR's. They are located in this file.

<table>
<thead>
<tr>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinKeyCodes.c/h</td>
<td>When synchronizing with the Touch2Key PC application, the LaunchPad sends a</td>
</tr>
<tr>
<td></td>
<td>mapping of BoosterPack buttons to keyboard presses. The keyboard presses</td>
</tr>
<tr>
<td></td>
<td>are stored as Windows key code. These files handle the mapping, and contain</td>
</tr>
<tr>
<td></td>
<td>a complete list of Windows key codes.</td>
</tr>
<tr>
<td>Vectors.c</td>
<td>Any vectors not used as part of the application are defined here. Defining</td>
</tr>
<tr>
<td></td>
<td>unused vectors is a good practice in embedded design.</td>
</tr>
<tr>
<td>TouchPro.c/h</td>
<td>The LaunchPad and BoosterPack contain code dedicated to the tuning of the</td>
</tr>
<tr>
<td></td>
<td>capacitive touch buttons. The LaunchPad’s is stored in this file. This</td>
</tr>
<tr>
<td></td>
<td>functionality is described in Section 9.</td>
</tr>
<tr>
<td>Ink_msp430g2553_for_HPBP</td>
<td>This is a modified version of the standard G2553 linker file.</td>
</tr>
</tbody>
</table>

### 8.1.2 Project Setting Considerations

For the most part, there are no special considerations for this code regarding project settings. Optimization settings are not of much concern, because resources are not tight. For this reason, and since optimization can make stepping through the code less linear, optimization has been turned off.

The project does use a customized linker file, compared to the standard one for the MSP430G2553. This file sets aside a linker segment for storing the button-to-keystroke mappings. Storing them in this segment allows them to easily be erased and re-written when the CMD_ASSIGN_GAME_MAPPING command is received from the PC. The detailed description of this linker modification is very similar to the one for the BoosterPack software; please see Section 8.2.2.6.

### 8.1.3 Clocks

The clocking on this application is very simple. The DCO is configured to use the calibrated 1MHz rate, which is used for MCLK and SMCLK. The program rests in LPM0. In LPM0, SMCLK remains available for the communication functions at all times.

ACLK is configured to use the VLO, but is not used for any function.

### 8.1.4 Normal or Touchtune Mode

At the beginning of execution, the application reads the LaunchPad’s pushbutton. If the button is held down, the application will spend its execution in the touchtune mode, described in Section 9. If not held down, then the application will fulfill its normal function as a bridge between the BoosterPack and the PC.

### 8.1.5 UART Interface with the PC

The software uses the USCI_A0 module to interface with the LaunchPad’s backchannel UART. The G2 LaunchPad, which is the default choice for use with this BoosterPack, only runs at 9600 baud, and so this is how the USCI is configured.

All commands from the PC are expected to be of the format shown in Table 6.

<table>
<thead>
<tr>
<th>Byte Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>The number of bytes, beginning with the cmd byte, and not including the 0x0D delimiter.</td>
</tr>
<tr>
<td>cmd</td>
<td>The command code. The same set of command codes is shared between the BoosterPack</td>
</tr>
</tbody>
</table>
The 0x0D end delimiter is part of the protocol to aid compatibility on the PC side, and would not ordinarily be necessary from the LaunchPad side. However, since the HapTouch GUI must send it, the LaunchPad software is written to expect it, and requires it.

### 8.1.6 I2C Interface with the BoosterPack

The LaunchPad’s MSP430G2553 serves as the I2C master. It clocks the interface at 100kbps.

The address by which it will try to address the slave is defined by the value `I2C_SLAVE_ADDR_OF_BOOSTERPACK` in the file `main.h`.

For a brief discussion of the I2C interface, see Section 8.3. For more detail, see the MSP430 Haptics Designer’s Guide.

### 8.1.7 Command Execution

Command handling is performed by the function `processCommandFromPC()`, which is mostly a large `switch()` statement. Within the `switch()`, the commands are separated into groups:

- Commands that are directly related to Haptics Library function calls, and thus are common to any application implementing the library for haptics and using the serial interface
- Commands that are specific to this BoosterPack

The commands are handled in a simple, straightforward manner:

1. The command code is evaluated, and handling is directed down slightly different paths, depending on the command
2. In almost all cases, the same set of data received from the PC are passed along to the BoosterPack over I2C (with the exception that the 0x0D terminator is omitted).
3. The LaunchPad software begins fetching a response from the BoosterPack. Until the BoosterPack is ready, the command response code will be 0x0C, which indicates `CMDRESP_WAIT_FOR_RESP`, which tells the LaunchPad it needs to keep waiting for a response.
4. When it receives any other response code, it passes the response back to the PC over UART
5. After re-trying to read the response for a fixed number of tries, it gives up, and passes the `CMDRESP_WAIT_FOR_RESP` code to the PC.

There are two exceptions to the procedure above, both related to gaming. See Section 8.4.2 for a description. Note that the code for tuning the capacitive touch buttons is not covered in this section; see Section 9.

Most of the commands are handled with the same function: `handleBasicCommand()`. The exceptions are:

- the gaming functions, described in Section 8.4.2
the read-back functions

The readback functions differ from those handled by handleBasicCommand() in that the latter only receive a single response code byte in response from the MSP430TCH5E. In contrast, a readback command usually receives more than one byte, where the number of bytes received varies according to the readback index that followed the readback command byte.

8.1.8 Gaming-Related Functions

The gaming function is somewhat more complex. It represents several departures from the way in which things are handled in the other areas of functionality.

The three gaming-related commands are briefly shown in Table 7. They are described in more detail in Section 8.4.2
Table 7. Gaming-Related Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_ASSIGN_GAME_MAPPING</td>
<td>Sent by the HapTouch GUI prior to gaming, to program a new mapping of the BoosterPack’s buttons to system keystrokes. This command does not get passed to the BoosterPack; the mapping is stored in the flash memory of the LaunchPad’s G2553. Note that each call to this command only reprograms player 1 or player 2, but not both.</td>
</tr>
<tr>
<td>CMD_GAME_SYNC</td>
<td>Sent by Touch2Key at the beginning of gaming. The LaunchPad responds by sending CMD_ENTER_GAME_STATE to the BoosterPack, and sending the button-to-keystroke mapping to Touch2Key. If the PC receives the mapping in response to this command, it considers the syncing successful, and it begins sending CMD_GAMING_GET_BTN_TRANSITIONS on a periodic basis.</td>
</tr>
<tr>
<td>CMD_GAMING_GET_BTN_TRANSITIONS</td>
<td>The LaunchPad responds to this command by polling the BoosterPack for the status of each of its buttons: BTN_HELD_DOWN, BTN_NOT_PRESSED, or BTN_IN_DEBOUNCE. It converts these levels into transitions, and sends the transition information back to the PC. This function gets called by Touch2Key on fixed periods, according to a pre-determined button scan rate.</td>
</tr>
</tbody>
</table>

None of these commands fits the same bridge-type model followed by the others, and each is different from the other.

8.1.9 Changing the Button-to-Keystroke Mapping

There are two ways to change this mapping.

The initial mapping, which corresponds to the mapping use by www.virtualnes.com, is located within the LaunchPad’s source code. A special linker segment is provided to hold an array called btnKeyMappingInFlash[], and it is this array that stores the mapping. So, one option to change the mapping is to change these values, and re-download the code to the LaunchPad.

The other way is to use the HapTouch GUI’s Assign Game Profile command. This command sends eight instances of the CMD_ASSIGN_SEQ_TO_BTN command (which re-configures the predefined haptics sequences played by the BoosterPack’s eight buttons), followed by one instance of CMD_ASSIGN_GAME_MAPPING. As described in Section 8.1.8, this latter command causes the mapping passed after the command byte to be programmed into this dedicated linker segment. Therefore, the next time CMD_GAME_SYNC is received, this is the mapping that will be returned to Touch2Key.

Note that the keystroke codes provided in WinKeyCodes.h are Windows-specific.
8.2 BoosterPack Application

The BoosterPack application has considerably more content than the LaunchPad application. Much of this is because of the numerous modes of operation.

8.2.1 File Organization

The directory structure for the example project is shown below.

![Software Stack Diagram](image)

Figure 69. Software Stack Diagram
Table 8 describes the purpose of each file.

### Table 8. LaunchPad Application File Description

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>main()</td>
</tr>
<tr>
<td>HardwareConfig.c/h</td>
<td>Clocks, power, and I/O’s</td>
</tr>
<tr>
<td>\Comms</td>
<td>I2C and command handling</td>
</tr>
<tr>
<td>\Touch_and_Haptics</td>
<td>Application-level handling of touch and haptics; makes calls to the MSP430</td>
</tr>
<tr>
<td></td>
<td>Haptics Library and Capacitive Touch Library. Also includes callbacks.c/h,</td>
</tr>
<tr>
<td></td>
<td>which contains the callbacks for the haptics library.</td>
</tr>
<tr>
<td>StateMachine.c/h</td>
<td>Manages the software state machine</td>
</tr>
<tr>
<td>Timer.c/h</td>
<td>Manages the heartbeat timer</td>
</tr>
<tr>
<td>\MSP430_Haptics_Library</td>
<td>The MSP430 Haptics Library</td>
</tr>
<tr>
<td>\CTS_Library</td>
<td>The MSP430 Capacitive Touch Library</td>
</tr>
<tr>
<td>Vectors.c</td>
<td>Definition of the unused ISRs; generally it is a good embedded practice to do so</td>
</tr>
<tr>
<td>HapTouchBoosterPack_link.cmd</td>
<td>This is a modified version of the standard TCH5E linker file. See Section 8.2.2.6 for a description of the modifications.</td>
</tr>
</tbody>
</table>

### 8.2.2 Project Setting Considerations

There are some specific considerations in the projects settings, described below.

Please remember to consult both the [MSP430 Capacitive Touch Library (SLAA490)](SLAA490) document, and the [MSP430 Haptics Library Designer's Guide (SLAU543)](SLAU543), for the project setting requirements for these constituent libraries.

#### 8.2.2.1 MSP430TCH5E

It bears mentioning again in this section that the MSP430 Haptics Library only runs on the MSP430TCH5E device derivative. The library may build on other derivatives, but it will not execute. If attempted, the function `HAPTIC_init()` will return an error indicating that the device is not supported.

#### 8.2.2.2 Output Format

The MSP430TCH5E Haptics Library is provided as object code *.lib files, which means they've been precompiled in a particular output format. That format is ELF.

![Output Format](Figure 70. Output Format)

The entire project needs to use this same format, otherwise errors will result. Fortunately ELF is the default setting in CCS, and so unless you explicitly set it to COFF, the setting should be correct.

#### 8.2.2.3 Optimization

Speed is probably not a major concern in this application, whereas memory is tight. The application does not fit in the available flash memory without some optimization for size.
If changing the optimization from the default settings, keep in mind that system timing will likely be affected. The LaunchPad application may possible need to wait longer for I2C responses, and in turn, the HapTouch GUI and Touch2Key’s wait times are based on the BoosterPack timing resulting from the default optimization settings.

8.2.2.4 GCC Extensions

The capacitive touch library requires GCC extensions in order to build. Without this, building the project will experience numerous errors.

The location of this setting is shown in the figure below.

![Image showing GCC Extensions setting]

Figure 71. Enabling GCC Extensions

This and other project settings for this library are described in the Capacitive Touch Library Software Programmer’s Guide.

8.2.2.5 Include Paths

There are several global include paths to help the compiler and linker find all the necessary files. In particular, there are global paths to `\MSP430_Haptics_Library` and `\CTS_Library`, the directories for the haptics and capacitive touch libraries.

8.2.2.6 Linker File Modifications

This software stores two types of re-writable data in flash:

- Eight predefined haptics sequences, playable by the eight capacitive touch buttons
- Configuration data for the audio haptics feature

To make room for this, a customized version of the standard MSP430TCH5E linker file was created. It sets aside a segment of the MSP430TCH5E’s flash memory to store both the items above. This same flash segment houses two linker segments, corresponding with those items.

<table>
<thead>
<tr>
<th>SEQ_IN_FLASH1</th>
<th>origin = 0xC000, length = 0x0180</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIO_CONFIG1</td>
<td>origin = 0xC180, length = 0x0080</td>
</tr>
</tbody>
</table>
In the MSP430x2xx architecture, each flash segment is 0x0200 bytes (or 512 bytes) in length. 0xC000 aligns with the beginning of one of these segments. The lengths of the two linker segments above add up to 0x0200. Therefore, these two linker segments fit within a single flash segment. (Notice that the term segment is used here in two different contexts: flash and linker.)

It is desirable to be able to change only one predefined haptics sequence at a time, or to change audio configuration information without changing any sequence. A flash segment is the smallest unit of MSP430 flash memory that can be erased at a time; therefore, before erasing, it is necessary to copy all the data that is not to be lost into a safe location. These 512 bytes would fill all the RAM that exists on the MSP430TCH5E, and of course the application itself uses some of that RAM for other purposes. Therefore the only way to preserve the data is to copy it to another location in flash.

So another flash segment is allocated, which mirrors the one shown above:

```
SEQ_IN_FLASH2           : origin = 0xC200, length = 0x0180
AUDIO_CONFIG2           : origin = 0xC380, length = 0x0080
```

This is the scratchpad segment. Anytime the flash data is changed, the entire contents are written to the scratchpad, with the new data interleaved among the old existing data. Then, the entire scratchpad is copied back to the original flash segment, byte for byte.

Having set aside these linker segments in the linker file, they are assigned to sections:

```
SECTIONS
{
  .bss     : {} > RAM                /* GLOBAL & STATIC VARS              */
  .data    : {} > RAM                /* GLOBAL & STATIC VARS              */
  .sysmem  : {} > RAM                /* DYNAMIC MEMORY ALLOCATION AREA    */
  .stack   : {} > RAM (HIGH)         /* SOFTWARE SYSTEM STACK             */
  .text    : {} > FLASH              /* CODE                              */
  .cinit   : {} > FLASH              /* INITIALIZATION TABLES             */
  .const   : {} > FLASH              /* CONSTANT DATA                     */
  .cio     : {} > RAM                /* C I/O BUFFER                      */
  .seq_in_flash1 : {} > SEQ_IN_FLASH1 // ADDED
  .audio_config1 : {} > AUDIO_CONFIG1 // ADDED
  .seq_in_flash2 : {} > SEQ_IN_FLASH2 // ADDED
  .audio_config2 : {} > AUDIO_CONFIG2 // ADDED
}
```
Finally, these sections are referenced in the code, all within AppHaptics.c; for example:

```c
#pragma DATA_SECTION (predefinedSeqsInFlash, "seq_in_flash1");
const uint8_t predefinedSeqsInFlash[] = {
    1, 0,
    .
    .
};
```

This `#pragma` causes the data element `predefinedSeqsInFlash[]` (which in this example is later assigned default sequence values) to be stored in the linker section `seq_in_flash1`.

### 8.2.3 Usage of MCU Peripheral Resources

Usage of peripherals by this project is shown in the table below.

#### Table 9. Peripheral Usage

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/Os</td>
<td>Nearly all I/Os on the MSP430TCH5E are used, for the purposes of:</td>
</tr>
<tr>
<td></td>
<td>- Capacitive touch buttons</td>
</tr>
<tr>
<td></td>
<td>- Enabling/disabling the actuators</td>
</tr>
<tr>
<td></td>
<td>- Mode inputs (BTN/AUDIO and ERM/LRA slider switches)</td>
</tr>
<tr>
<td></td>
<td>- Driving LEDs</td>
</tr>
<tr>
<td>USCI module</td>
<td>I2C interface</td>
</tr>
<tr>
<td>Timer_A0</td>
<td>Used for two purposes:</td>
</tr>
<tr>
<td></td>
<td>- Heartbeat timer. Generates timing for the haptics library and capacitive touch measurements.</td>
</tr>
<tr>
<td></td>
<td>- Capacitive touch measurement; measuring the frequency of the oscillation of the capacitive circuit.</td>
</tr>
<tr>
<td></td>
<td>See Section 8.2.7 for a discussion of how these two functions are handled on the same timer.</td>
</tr>
<tr>
<td>Timer_A1</td>
<td>PWM output, to drive the actuator</td>
</tr>
</tbody>
</table>
8.2.4 Memory Requirements

As an example of the library’s memory requirements, the table below shows a breakdown of flash/RAM usage, in CCS, with optimization set to “maximum for size”.

Note: These are only shown as an example; please refer to the *.map file in the \Debug directory of the project, for an actual breakdown.

Table 10. Memory Usage

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
<th>Usage (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM (not incl. stack)</td>
<td>Haptics Library</td>
<td>163</td>
</tr>
<tr>
<td></td>
<td>Capacitive Touch Library</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Application</td>
<td>147</td>
</tr>
<tr>
<td>Flash – code</td>
<td>Haptics Library</td>
<td>2.6K</td>
</tr>
<tr>
<td></td>
<td>Capacitive Touch Library</td>
<td>1.2K</td>
</tr>
<tr>
<td></td>
<td>Application</td>
<td>6K</td>
</tr>
<tr>
<td>Flash – data</td>
<td>Each Haptics Library effect bank added with HAPTIC_addBank()</td>
<td>1.8K</td>
</tr>
<tr>
<td></td>
<td>Capacitive Touch Library</td>
<td>192</td>
</tr>
<tr>
<td></td>
<td>Application</td>
<td>24</td>
</tr>
</tbody>
</table>

The RAM usage is entirely static; no heap space is required.

8.2.5 Execution Flow

A pseudo-representation of the execution flow is shown in the figure below.
In its spare time, main() rests in LPM0 power mode. LPM0 keeps the DCO active to drive Timer_A0, as well as Timer_A1 if a haptic effect is being played.

It is awakened from LPM0 from one of two interrupt sources:

- The Timer_A0 heartbeat timer
- The USCI_B0 module receiving I2C data from the host MCU (LaunchPad)

The ISRs for these interrupts use the CPU to perform certain actions, and then may set some system flags and revive main() to handle those flags. The flags are typically set within the bitfield variable bSystemFlags.

The heartbeat timer ISR calls some of the haptics functions itself, because they complete quickly and do not keep the MCU in a long-term state of disabling interrupts. In contrast, capacitive touch sensing takes a few milliseconds, and so when it comes time to perform this action, the ISR sets flags and requests main() to handle it.

Similarly, the USCI ISR handles some tasks itself, because they complete quickly. In the case of the two I2C commands that involve flash writes, it uses the state machine mechanism to request main() to handle it.

The mechanism of setting a flag within an ISR, and then modifying the stack’s contents to keep the CPU active when the ISR completes (thus reviving main()) is a standard MSP430 technique; see Section 8.2.10.
8.2.6 System Timing

This software environment is a complex one, with multiple interrupt sources, making it possible for things to happen at unexpected times. This section explains how timing is managed in this software.

A good place to begin is to understand any timing requirements the various functions have. These are shown in Table 11.

Table 11. Periodic Functional Timing Requirements

<table>
<thead>
<tr>
<th>Function</th>
<th>Timing Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haptics, when audio is not enabled</td>
<td>• HAPTIC_update() must be called every 5ms, +/-10%</td>
</tr>
<tr>
<td>Haptics, when audio is enabled</td>
<td>• HAPTIC_audioGetConversion() must be called every 250us, +/-10%</td>
</tr>
<tr>
<td></td>
<td>• HAPTIC_audioUpdate() must be called every four calls to HAPTIC_audioGetConversion()</td>
</tr>
<tr>
<td>Capacitive touch button scanning</td>
<td>• There is some flexibility in this rate, but a 20ms period was chosen</td>
</tr>
</tbody>
</table>

Because these are periodic, their timing can be arranged in synchronous fashion, triggered by the heartbeat timer. In non-audio situations, the timer is set to 5ms intervals, and HAPTIC_update() is called in each one. After several of these intervals, capacitive touch measurements are triggered. In audio situations, the timer is set to 250us intervals, calling HAPTIC_audioGetConversion(). After four of these, HAPTIC_audioUpdate() is called. (No capacitive touch is performed during audio.) The haptics functions are called directly out of the Timer_A0 ISR. If it is time to scan buttons, the ISR then awakens main() to perform this action.

Therefore, all of these events are synchronized to the heartbeat timer. In contrast, since the MSP430TCH5E is the I2C slave, I2C interrupts can happen at any time. I2C transactions involve multiple interrupts, once for each byte or bus condition. In this software, these interrupts fit into timing gaps where other functions are not being performed. When observed on a scope, this can stretch out the transaction with gaps in between. But provided that the I2C master waits long enough for a response, this is tolerable.

main() does not often disable interrupts. It only does so in one situation: when capacitive touch measurements are taking place. If I2C communication were allowed to interrupt this measurement, it would skew the measurement, potentially registering a touch that did not really happen. Any place the capacitive touch library is called, interrupts are disabled immediately prior, and re-enabled immediately after.

Notice that only each individual capacitive touch call is protected in this way. Another option might have been to disable it once at the beginning of scanning all the buttons, and only re-enable after; however, doing so would have further delayed I2C response to the host MCU. Re-enabling in between allows at least intermittent I2C communication with the host, interspersed among capacitive touch measurement.

An example of what can happen to the I2C communication is shown below.
Figure 73. I2C Communication Interrupted by Other Functions

In this case, the host MCU is writing a 0x30 (readback command), followed by the readback index (0x02). However, in between, the BoosterPack software began a capacitive touch measurement, delaying its ability to move data out of the RXBUF register.

These delays cause no harm, as long as the host MCU knows to wait a long enough period of time for an answer; including:

- The time for the BoosterPack to process the command
- Several interspersed capacitive touch measurements, and maybe the heartbeat timer ISR

The LaunchPad software’s retry counts were chosen for worst-case scenarios of this kind. Similarly, the HapTouch GUI and Touch2Key, which must wait for the LaunchPad, also have timeout periods that take into account these worst-case response times; see Section 8.4.2.

8.2.7 Timer Allocation

Using the haptics library and the capacitive touch library at the same time means there are these simultaneous requirements:

- Heartbeat timer; must be able to achieve 5ms and 250us periods, as well as capacitive touch scan periods on the order of 20ms.
- Capacitive touch measurement; the touch library uses a timer to measure the number of oscillations of the touch oscillator. Must be Timer_A-based, because this is what the touch library requires.
- Haptics actuator PWM; needs a PWM output pin, which means it must be Timer_A-based

To fulfill these requirements, the MSP430TCH5E has three timers available:

- Timer_A0
- Timer_A1
- WDT (watchdog timer)

The haptics actuator PWM needs a dedicated timer, since it is undesirable to have restrictions or delays in when haptics can be generated. In trying to fit the remaining two functions with the remaining two timers, a relevant question becomes: can the WDT be used as the heartbeat timer?

This depends on the application, and whether you want to use the WDT for its watchdog function. If not, then it is possible to use it for the heartbeat timer; this is discussed in Section 8.2.7.2. If you do want to use the WDT as a watchdog, then it cannot be used as the heartbeat timer, and some creative multiplexing of one of the Timer_A modules becomes necessary so that it can share the heartbeat timer and the capacitive touch functions. This is discussed in Section 8.2.7.1.

Note that this choice must be made by anyone combining the haptics and capacitive touch libraries.
Whichever timer is used, it is necessary to derive subintervals of that heartbeat. For example, in this software, the capacitive touch scan period (20ms) was chosen as a multiple of the 5ms haptics period. The timer ISR is triggered every 5ms; a software sub-timer is kept (incClockDivider()) that reduces this to 20ms.

This discussion is tightly tied to the selection of clocks; see Section 8.2.8 for this discussion.

### 8.2.7.1 Multiplexing a Timer_A Between Touch and Heartbeat Timer

The BoosterPack software multiplexes these functions on Timer_A0. If the WDT were desired for use as a watch; multiplexing on Timer_A preserves this option.

Every time the capacitive touch library samples the buttons, it saves the registers of the Timer_A it is configured to use, and restores them when it is done with the timer. If it is acceptable to have a subtle compromise in haptics performance, it might actually be alright to simply assign the same Timer_A to both functions.

However, the capacitive touch library’s saving and restoring of the Timer_A registers effectively extends the heartbeat timer’s period by several milliseconds, every time the capacitive measurement is performed. This would impact haptics performance; however, it is preventable.

Compensating for this requires some creative use of Timer_A: the time that is lost from the heartbeat timer during capacitive touch measurement can be added back to the timer. The function below is called after the touch measurements are performed.

```c
void addTimeToHeartbeatTimer(uint16_t time)
{
    TA0CTL &= ~MC_1;
    TA0R += time;
    TA0CTL |= MC_1;
}
```

The value `time` is best found empirically. For example, an I/O pin can be set within the heartbeat timer (Timer_A0) ISR and watched on a scope, as the value is tuned.

Note that the majority share of the lost time is determined by the `accumulationCycles` value in each sensor’s definition in structure.c. The longer `accumulationCycles` is, the longer the measurement is performed, allowing higher count values at the expense of more system time. The value is not an exact time value, hence the need to determine the time empirically.

```c
const struct Sensor WUP_Sensor =
{
    .halDefinition = RO_PINOSC_TA0,
    .numElements = 1,
    .baseOffset = 4,
    .arrayPtr[0] = &WUP_Element,
    .accumulationCycles = 15
};
```
8.2.7.2 Using the WDT as the Heartbeat Timer

In Section 8.2.7, it was said that the WDT module could perform the heartbeat timer function, provided it was not needed for the watchdog function. This would allow Timer_A0 to only be used for capacitive touch, avoiding the need to add back lost time. Using the WDT is possible, but again requires some creativity, because the WDT module does not have Timer_A's flexibility in setting its period. Its possible interval configurations are best seen within the msp430tch5e.h header file:

```c
/* WDT-interval times [1ms] coded with Bits 0-2 */
/* WDT is clocked by fSMCLK (assumed 1MHz) */
#define WDT_MDLY_32  (WDTPW+WDTTMSEL+WDTCNTCL) /* 32ms interval */
#define WDT_MDLY_8   (WDTPW+WDTTMSEL+WDTCNTCL+WDTISO) /* 8ms      */
#define WDT_MDLY_0_5  (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1) /* 0.5ms    */
#define WDT_MDLY_0_064 (WDTPW+WDTTMSEL+WDTCNTCL+WDTIS1+WDTISO) /* 0.064ms  */

/* WDT is clocked by fACLK (assumed 32KHz) */
#define WDT_ADLY_1000 (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL) /* 1000ms   */
#define WDT_ADLY_250  (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTISO) /* 250ms    */
#define WDT_ADLY_16   (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1) /* 16ms     */
#define WDT_ADLY_1_9  (WDTPW+WDTTMSEL+WDTCNTCL+WDTSSEL+WDTIS1+WDTISO) /* 1.9ms    */
```

These are values that can be assigned to the WDTCTL register. As can be seen, the intervals are based on powers of two. Note that they are all based on source clock assumptions: SMCLK=1MHz and ACLK=32kHz, the latter of which can only be assumed when a 32kHz crystal is sourcing ACLK.

Although these do not appear to match the haptics library’s requirements very well (as shown in Table 11), it is still possible to achieve them. In the case of this BoosterPack, SMCLK is 16MHz. If the value `WDT_MDLY_8` were assigned to WDTCTL, then the WDT would generate an interrupt every 500us. A counter could be maintained that reduced this period to 5ms. (The software includes a function `incrClockDivider()` that could be used for this purpose.)

The best means of getting the WDT to trigger the proper periods will depend on your SMCLK settings, whether you have a 32kHz crystal attached, and whether you are using the audio haptics feature or not.
8.2.8 Clocks and Low-Power Modes

The MSP430x2xx family has these available clock sources.

<table>
<thead>
<tr>
<th>Clock Source</th>
<th>Frequency*</th>
<th>Typical System Clock Path(s)</th>
<th>Tolerance</th>
<th>Power Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>The DCO (digitally-controlled oscillator)</td>
<td>Up to 16MHz</td>
<td>MCLK (CPU) SMCLK (peripherals - fast)</td>
<td>If using one of the four calibrated settings provided in info memory flash, very good.</td>
<td>Good</td>
</tr>
<tr>
<td>The VLO (very low-power, low-frequency oscillator)</td>
<td>4-20kHz</td>
<td>ACLK (peripherals - slow)</td>
<td>Low</td>
<td>Ultra-low-power</td>
</tr>
<tr>
<td>The LFXT1 32kHz crystal</td>
<td>32kHz</td>
<td>ACLK (peripherals - slow)</td>
<td>High</td>
<td>Ultra-low-power</td>
</tr>
</tbody>
</table>

* Always see the device data sheet for actual parametrics

Note that the BoosterPack does not have a 32kHz crystal, and therefore this is not an option for this software.

The table below shows how the system clocks were configured in the BoosterPack software.

<table>
<thead>
<tr>
<th>System Clock</th>
<th>Clock Source</th>
<th>Used For</th>
<th>Frequency*</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCLK</td>
<td>DCO</td>
<td>CPU</td>
<td>16MHz</td>
</tr>
<tr>
<td>SMCLK</td>
<td>DCO</td>
<td>Timer_A0 (heartbeat timer) Timer_A1 (haptics PWM)</td>
<td>16MHz</td>
</tr>
<tr>
<td>ACLK</td>
<td>VLO</td>
<td>N/A</td>
<td>4-20kHz</td>
</tr>
</tbody>
</table>

The sections below discuss why these settings were chosen, and things to consider if choosing other options.

8.2.8.1 MCLK

The primary reason for using 16MHz MCLK is the audio function. When performing audio haptics, the HAPTIC_audioGetConversion() and HAPTIC_audioUpdate() functions must both be able to execute within the 250us sampling window. This is not much time, and as soon as the 250us window is reached, HAPTIC_audioGetConversion() must get called again. To execute within this window, an MCLK speed of approximately 15MHz is required. 16MHz is the next highest calibrated clock value placed within the MSP430TCH5E flash at the TI factory.

If audio had not been used, then instead of very tight clocking requirements for MCLK, MCLK would be valid across a very wide range, down to 1MHz or lower.

To achieve reliable frequencies, the pre-programmed, calibrated values in the device’s info memory are used. Devices in the MSP430x2xx family provide values in their info memory flash that are calibrated on that particular physical device, for frequencies 1/8/12/16MHz.
8.2.8.2 Heartbeat Timer Clock and Haptics PWM

The DCO is used to drive the heartbeat timer in this software, via the SMCLK system clock. Since MCLK was already configured to use the DCO at 16MHz, the same frequency is applied to SMCLK, and thus to the heartbeat timer.

Use of the DCO means that LPM0 is the lowest low-power mode that can be entered. The only way to enable LPM3 entry would have been to use either the VLO or LFXT1 to run the heartbeat timer. LFXT1 would do this very well, if a crystal were attached. On the BoosterPack, there is no crystal.

The other choice for enabling LPM3 is the VLO. As seen in Table 13, the VLO has a very wide tolerance. Given the recommended +/-10% tolerances on the haptics library timing requirements discussed in Section 8.2.6, it is not a very good choice for the heartbeat timer.

Having said this, there is a technique that allows the VLO to be used in higher-precision applications, where it is periodically measured against the DCO’s calibrated frequencies. This may work with the haptics library if audio is not required, where the 5ms period allows some room for the VLO count to be tuned. (See Using the VLO Library (slaa340).) If the VLO were used as the heartbeat timer, then LPM3 could be entered, as long as haptics were silent. While a haptics effect is played, the DCO is needed to achieve the DRV2603’s input range of 10-250kHz. This is because the VLO cannot be relied upon to always be above 10kHz.

8.2.9 Organization of main()

The software performs initialization, and then enters a main loop.
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;

    initClocks();
    initIO();
    initI2C();
    determineBoardConfig();
    initHapticsLib();
    initTouchBaselines();
    initHeartbeatTimer();
    bSystemFlags = NO_FLAGS;

    while(1)
    {
        _DINT();
        if(bSystemFlags == NO_FLAGS)
        {
            __bis_SR_register(LPM0_bits+GIE);
        }

        if((bSystemFlags & NEED_TO_CHANGE_STATE))
        {
            changeStates();
        }
        _EINT();

        if(bSystemFlags & TIME_TO_READ_BUTTONS)
        {
            checkButtonsAndHandle();
            bSystemFlags &= ~TIME_TO_READ_BUTTONS;
        }
        if(bSystemFlags & TIME_TO_READ_WHEEL)
        {
            checkWheelAndHandle();
            bSystemFlags &= ~TIME_TO_READ_WHEEL;
        }
        if(bSystemFlags & TIME_TO_SAMPLE_FOR_TOUCHTUNING)
        {
            getTouchCountsForTuning();
            bSystemFlags &= ~TIME_TO_SAMPLE_FOR_TOUCHTUNING;
        }
    }
}

main() spends its spare time in low-power mode 0 (LPM0), in which the CPU is disabled but
SMCLK remains active, powered by the DCO.

It can be awakened from LPM0 by:

- Heartbeat timer events
- USCI ISR (I2C commands from a host MCU)
- PORT1 ISR (changes on the slider switch)

When awakened, the main loop executes one iteration. It may execute more iterations if a flag is
still set when the LPM0 entry is re-encountered. Flags are set primarily in interrupt service
routines (ISRs), which then also wake the main loop so that it can execute the flags.

Section 8.2.10 discusses the low-power mode management in more detail.
8.2.10 Low-Power Mode Management

This software uses a standard MSP430 technique for managing low-power modes. This is briefly described here, both as a general discussion and how it applies to the BoosterPack.

Low-power modes are entered in a non-interrupt context (i.e., `main()`) through an intrinsic function that modifies the MSP430’s status register (SR). The combination of certain SR bits being set or cleared determines the mode.

```c
__bis_SR_register(LPM0_bits+GIE);
functionThatRunsAfterLPM();
```

The above code sets the bits in the SR that cause LPM0 to be entered. (See the MSP430x2xx Family User’s Guide for the exact SR definition.)

Simultaneously, this intrinsic sets the general interrupt enable (GIE) bit, which ensures that interrupts will be able to run while the MCU rests in this low-power mode. (Without GIE, the low-power mode could never be exited.)

Interrupt service routines (ISRs) are then free to execute while `main()` remains in LPM0. But when each one completes, the MCU goes back into LPM0. It works this way because the contents of the SR are pushed to the stack prior to running the ISR, and are restored when the ISR exits. The information in those SR contents said that the MCU had been in LPM0 prior to the ISR, and so that is what it returns to afterwards.

It is possible to write an application this way, where all useful functionality happens within an ISR context. Unfortunately it is not a good idea to spend large amounts of contiguous time within a single ISR’s context, because GIE is automatically cleared when an ISR is entered; as a result, the system remains unresponsive during that time. (Clearing GIE prevents nested interrupts and stack overflows.)

Therefore, ISRs should be kept short. If a large amount of functionality needs to execute from a given ISR, it is better for the ISR to set a flag and wake `main()` to handle that functionality.

Indeed, this is done in several places in the BoosterPack software:

- The two I2C commands that cause flash memory to be erased and written; these are triggered from the USCI ISR, but require on the order of 50ms to complete.
- Capacitive touch measurements; these are triggered from the Timer_A0 ISR, but take a few milliseconds to complete.
For these functions, the ISRs set a flag, and then wake main:

```c
bSystemFlags |= TIME_TO_READ_BUTTONS;
__bic_SR_register_on_exit(LPM3_bits);
```

Once again, an intrinsic function is used to accomplish this. But this time, the intrinsic does not modify the SR itself, but rather modifies its stored contents on the stack. The intrinsic can do this because it knows exactly where the SR was stored on the stack. It clears the bits related to low-power mode. (Notice that LPM3 was used above; clearing the bits for LPM3 also clears the bits for LPM0.)

So when the ISR exits, the Program Counter (PC) register is restored to what it had been prior to the ISR, pointing to `functionThatRunsAfterLPM()`. The modified SR value is then placed into the SR. These contents now tell the CPU to begin executing. (More accurately, since the CPU had been executing the ISR, it continues to execute; it just does so within `main()`, rather than the ISR.)

It therefore resumes full execution of `main()`, at `functionThatRunsAfterLPM()`. Given that low-power mode entry happens on a cyclical basis – perhaps in a main loop – execution usually eventually arrives back at the low-power mode entry. This BoosterPack software does employ a main loop, and this does happen here.

This technique is commonly used in MSP430 programming.

The next section describes at a higher level how this technique is used to implement the software state machine.

### 8.2.11 Software State Management

The global `gBoardState` always contains the state in which the software currently resides. The values of this global correspond with the states that were shown in Figure 18, and re-printed below.
Figure 74. HapTouch BoosterPack Modes of Operation

State changes can be initiated from these sources:

- Capacitive button touches (Touch.c; executes from `main()`)
• I2C communication (Comms.c; executes out of USCI ISR)
• Slider switch changes (HardwareConfig.c; executes out of PORT1 ISR)

The capacitive touch events happen at a consistent location within the flow of main(). That is, the time from a heartbeat timer interrupt to a capacitive touch measurement is relatively fixed. The other two however -- being driven by externally-originating interrupts – can happen at any time. Changing state at random times could provoke unexpected software failures; therefore, the software synchronizes all state changes to a consistent and convenient location within main(): directly after exiting LPM0.

```c
_DINT();
if(bSystemFlags == NO_FLAGS)
{
    __bis_SR_register(LPM0_bits+GIE);
}
if((bSystemFlags & NEED_TO_CHANGE_STATE))
{
    changeStates();
}
_EINT();
```

To implement synchronized state change, the locations in code that change state do not modify gBoardState directly; they modify gNextBoardState, and then set the NEED_TO_CHANGE_STATE flag in bSystemFlags indicating that a state change is in order. If this happens within an ISR context, software then calls __bic_SR_register_on_exit(LPM3_bits) to wake main().

Whether this happens within an ISR context or not, execution is assured to not enter LPM0 if NEED_TO_CHANGE_STATE is set. Execution arrives at changeStates(), which:

• Handles any actions to re-configure peripherals for the new state
• Performs visible transition effects, allowing the user to see visible signs of entering a new state
• Officially changes the state by assigning gNextBoardState to gBoardState

Notice that interrupts are disabled (using the intrinsic function _DINT()) prior to evaluating bSystemFlags. The action of evaluating the flag and entering LPM0 is not atomic. Without disabling interrupts, it would be possible for an ISR to set the flag and trigger a wake from main() between the time that the flag is evaluated and LPM0 is entered. If this happened, the CPU would not wake until a subsequent ISR did the same thing. With interrupts disabled, and with the scenario described above, the ISR will not execute until after LPM0 is entered, and thus the CPU will wake in a normal fashion.

### 8.2.12 Capacitive Touch Button Sensing

All calls to the capacitive touch library are contained within the file Touch.c. The software treats all the touch elements as discrete pushbuttons, calling the library function TI_CAPT_Buttons().
In the capacitive touch library, an individual touch button, connected to an individual I/O pin, is called an \textit{element}. Elements can be grouped into \textit{sensors}. To allow simultaneous pressing of any two buttons, each element in the BoosterPack software is given its own sensor.

In general, handling of the wheel and the four discrete pushbuttons (SELECT/START/B/A) are kept separate in the code. For example, the sensor structs are divided into two arrays:

- pushbuttonSensorList[]
- wheelSensorList[]

Two separate touch functions then scan through the sensors in these arrays:

- checkButtonsAndHandle()
- checkWheelAndHandle()

These are divided because the handling of each group differs slightly. Among the discrete buttons, variables are maintained that indicate the state of that button. This is shown in Figure 75.

![Figure 75. The Lifecycle of a Capacitive Touch Pushbutton Continually Pressed](image)

In contrast, presses of the wheel’s four buttons are converted into a position 0-7, and handling is based on those positions.

![Figure 76. Capacitive Touch Wheel Positions](image)

In this conversion, the press of two adjacent buttons is registered as an independent position, allowing the 45-degree angle positions to be registered. Depending on the position, different actions might be taken.
A capacitive touch wheel-style sensor could have been used here, instead of four pushbuttons. A wheel sensor is essentially a slider that, instead of being linear, is wrapped around a central point. Sliders are capable of many more than eight positions, and so are wheels. However, they are more difficult to tune than pushbuttons, and since only eight positions were needed for this application, discrete buttons were used instead.

### 8.2.13 Triggering Haptics Effects and Sequences

Most of the effort of using the haptics library is in setting up the periodic update functions and callbacks. Once this is done, triggering haptics effects and sequences is pretty simple.

Playing a single effect looks something like this.

```c
result = HAPTIC_playEffect(31, 0, FALSE);
```

This plays effect #31, with no timeout period. If an effect or sequence is currently playing, it will not override it.

Playing sequences can look like the code below. The sequence is first built, and then played. The TRUE parameter means that if an effect or sequence is currently being played, this new one will override it.

```c
HAPTIC_sequence seqBuiltInRAM;
seqBuiltInRAM.pairCount = 3;
seqBuiltInRAM.repeatCount = 2;
seqBuiltInRAM.effPairs[0].effIndex = 4;
seqBuiltInRAM.effPairs[0].timeGap = 2;
seqBuiltInRAM.effPairs[1].effIndex = 4;
seqBuiltInRAM.effPairs[1].timeGap = 2;
seqBuiltInRAM.effPairs[2].effIndex = 4;
seqBuiltInRAM.effPairs[2].timeGap = 2;
result = HAPTIC_playSequence(&seqBuiltInRAM, TRUE);
```

The BoosterPack project also provides for *predefined sequences*. To understand where this is beneficial, consider an application where a haptics sequence is known at compile-time and will never change; it can be stored as a simple `HAPTIC_sequence` struct instance built with the `const` modifier in flash. No special linker segments, or read and write capability, would be needed. (This is demonstrated in the MSP430 Haptics Library examples.)

However, perhaps these need to be set at run-time. For example, the BoosterPack supports these being re-written from the serial interface, and for them to still be available after a power cycle. This means writing them in flash. To support this, special linker segments are set up, as described in Section 8.2.2.6. Functions are then provided in AppHaptic.c for reading and writing:

- `writePredefinedSeqtoFlash()`
- `readPredefinedSeqFromFlash()`
- `playPredefinedSeq()`
With predefined sequences, the host MCU can write a new predefined sequence to the MSP430TCH5E flash memory, and then it can be played back later, by index.
8.3 I2C Commands

The BoosterPack software can receive haptics commands via I2C. The scheme for this is based on the one provided with the MSP430TCH5E Haptics Library. The basic library distribution provides many such commands; these are used by the BoosterPack software.

The BoosterPack software also adds a few new commands to enable application-specific functionality. They were essentially added as extensions to the same command framework used by the library.

Refer to the *MSP430TCH5E Haptics Library Designer’s Guide* (SLAU543) for a complete reference for the I2C interface, including command formatting and protocol. The remainder of this section can then be viewed as an addendum to that reference.
CMD ENTER GAME STATE

Description

This command puts the BoosterPack into GAME_STATE.

In responding back to the master, the BoosterPack software reads the state of the BTN/AUDIO slider switch. If the switch is positioned toward AUDIO, then software determines that the LaunchPad+BoosterPack game controller will represent player 1. If it’s in the BTN position, then the game controller will represent player 2. It returns this as the sole data parameter for this command. The LaunchPad software uses this byte to determine what button-to-keystroke mapping to send to the host.

Parameters

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command ID</td>
<td>0x55</td>
</tr>
</tbody>
</table>

Return

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Response code</td>
<td>CMDRESP_SUCCESS</td>
</tr>
<tr>
<td>1</td>
<td>PlayerOneTwo</td>
<td>An integer 1 or 2, reflecting whether the game controller should represent player 1 or player 2.</td>
</tr>
</tbody>
</table>

CMD TOUCHTUNE

Description

This command puts the BoosterPack into the TOUCHTUNE_STATE. It is only intended for use during development, to tune the capacitive touch buttons using the TouchPro GUI.

Parameters

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command ID</td>
<td>0x71</td>
</tr>
</tbody>
</table>

Return

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Response code</td>
<td>CMDRESP_SUCCESS</td>
</tr>
</tbody>
</table>

The following are not commands per se, but rather indices to the CMD_READBACK command.
CMD READBACK:  RB_INDEX_BTNSTATUS

Description

CMD READBACK is a command within the MSP430 Haptics Library. An index can be passed into CMD READBACK, to pull a specific kind of information from the MSP430TCH5E.

RB_INDEX_BTNSTATUS is an index added by the BoosterPack software, and not contained within the standard library. Its purpose is to return the status of all eight capacitive touch buttons: not pressed, waiting in a repeat delay, or held down. The BoosterPack software is written with the intention it be received while in GAME_STATE.

Parameters

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command ID</td>
<td>0x30</td>
</tr>
<tr>
<td>1</td>
<td>Readback_table_index</td>
<td>Determines which information is returned by the readback command, according to the table below. To determine button status, use RB_INDEX_BTNSTATUS.</td>
</tr>
</tbody>
</table>

Return

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Wheel: up</td>
<td>BTN_NOT_PRESSED, BTN_REPEAT_DELAY, or BTN_HELD_DOWN</td>
</tr>
<tr>
<td>1</td>
<td>Wheel: right</td>
<td>BTN_NOT_PRESSED, BTN_REPEAT_DELAY, or BTN_HELD_DOWN</td>
</tr>
<tr>
<td>2</td>
<td>Wheel: down</td>
<td>BTN_NOT_PRESSED, BTN_REPEAT_DELAY, or BTN_HELD_DOWN</td>
</tr>
<tr>
<td>3</td>
<td>Wheel: left</td>
<td>BTN_NOT_PRESSED, BTN_REPEAT_DELAY, or BTN_HELD_DOWN</td>
</tr>
<tr>
<td>4</td>
<td>SELECT button</td>
<td>BTN_NOT_PRESSED, BTN_REPEAT_DELAY, or BTN_HELD_DOWN</td>
</tr>
<tr>
<td>5</td>
<td>START button</td>
<td>BTN_NOT_PRESSED, BTN_REPEAT_DELAY, or BTN_HELD_DOWN</td>
</tr>
<tr>
<td>6</td>
<td>B button</td>
<td>BTN_NOT_PRESSED, BTN_REPEAT_DELAY, or BTN_HELD_DOWN</td>
</tr>
<tr>
<td>7</td>
<td>A button</td>
<td>BTN_NOT_PRESSED, BTN_REPEAT_DELAY, or BTN_HELD_DOWN</td>
</tr>
</tbody>
</table>
CMD_READBACK: _RB_INDEX_BTNCOUNTS

Description

CMD_READBACK is a command within the MSP430 Haptics Library. An index can be passed into CMD_READBACK, to pull a specific kind of information from the MSP430TCH5E.

_RB_INDEX_BTNCOUNTS is an index added by the BoosterPack software, and not contained within the standard library. Its purpose is to return the capacitive touch counts of all eight capacitive touch buttons. These values are the values returned by the TI_CAPT_Custom() call from the capacitive touch library. This is the information the TouchPro GUI needs, to display the performance of the buttons.

The BoosterPack software is written with the intention this readback index be received while in TOUCHTUNE_STATE.

Parameters

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Command ID</td>
<td>0x30</td>
</tr>
<tr>
<td>1</td>
<td>Readback_table_index</td>
<td>Determines which information is returned by the readback command, according to the table below. To determine button status, use RB_INDEX_BTNSTATUS.</td>
</tr>
</tbody>
</table>

Return

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Wheel: up</td>
<td>The counts returned by TI_CAPT_Custom() for this button</td>
</tr>
<tr>
<td>1</td>
<td>Wheel: right</td>
<td>The counts returned by TI_CAPT_Custom() for this button</td>
</tr>
<tr>
<td>2</td>
<td>Wheel: down</td>
<td>The counts returned by TI_CAPT_Custom() for this button</td>
</tr>
<tr>
<td>3</td>
<td>Wheel: left</td>
<td>The counts returned by TI_CAPT_Custom() for this button</td>
</tr>
<tr>
<td>4</td>
<td>SELECT button</td>
<td>The counts returned by TI_CAPT_Custom() for this button</td>
</tr>
<tr>
<td>5</td>
<td>START button</td>
<td>The counts returned by TI_CAPT_Custom() for this button</td>
</tr>
<tr>
<td>6</td>
<td>B button</td>
<td>The counts returned by TI_CAPT_Custom() for this button</td>
</tr>
<tr>
<td>7</td>
<td>A button</td>
<td>The counts returned by TI_CAPT_Custom() for this button</td>
</tr>
</tbody>
</table>
8.4 PC <-> LaunchPad <-> BoosterPack Communication Dynamics

In the gaming and GUI usage modes, the PC communicates with the BoosterPack through the LaunchPad. In some respects, the LaunchPad simply acts as a bridge; but the reality is that it plays a more advanced role. This section provides some detail on what the communication looks like in these modes.

Note that Section 8.1.7 discussed the way in which the LaunchPad software handles commands, which is relevant to understanding this topic. For the most part, this section adds to that description, rather than repeating it. It takes a high-level system view, with a particular focus on managing timing. Please refer to Section 8.1.7 for detail on the LaunchPad software itself.

8.4.1 GUI Usage Mode

From the HapTouch GUI's perspective, it sends a command to the combined LaunchPad-and-BoosterPack unit over the UART, and later it receives a response. The GUI has a very liberal timeout length (which it can afford to do because its client is a human). If that time expires, a timeout is reported to the user.

Between the time that the GUI sends a command and receives a response, the GUI is not allowed to send a second command. The LaunchPad software is not written to anticipate this.

As discussed in Section 8.1.7, the Boosterpack’s I2C interface reports a value of CMDRESP_WAIT_FOR_RESP until the actual, final response is generated and placed in the I2C return buffer. The LaunchPad keeps reading from the I2C buffer until the CMDRESP_WAIT_FOR_RESP value is no longer reported, or until its retry count is exceeded. Either way, it sends the result over the UART to the GUI. If the GUI receives a CMDRESP_WAIT_FOR_RESP response code, it reports this to the user. The LaunchPad’s retry values have been tuned to anticipate the BoosterPack’s worst-case response times.

These commands are special cases that do not follow the protocol in Section 8.1.7:

- CMD_GAME_SYNC: This is discussed in the next section.
- CMD_ASSIGN_GAME_MAPPING: The button-to-keystroke mapping is stored in the LaunchPad, and the BoosterPack software has no knowledge of it. When the GUI wants to re-program these mappings, it sends this command targeted directly at the LaunchPad, not the BoosterPack. The LaunchPad writes the new mapping into a special linker segment that stores the mappings.
8.4.2 Gaming Protocol

Communication during the gaming usage mode can be broken into two categories: syncing and button polling.

The sync process happens immediately upon starting Touch2Key, and completes (or fails) quickly. The result of the sync process is displayed to the user in a dialog box.
The sync process is described in Figure 78.

Figure 78. Game Controller Syncing Process
After syncing succeeds, the BoosterPack is in GAME_STATE and begins polling its capacitive touch buttons in the exact same way it does in EFF_STATE or SEQ_STATE. This happens independently of Touch2Key.

After the user accepts the dialog box indicating that syncing succeeded, Touch2Key begins sending periodic commands to poll the status of the capacitive touch buttons. This simply returns the result of the last scan the BoosterPack performed on its buttons.

Timing becomes an important consideration during button polling. At Touch2Key's level, there are multiple layers of worst-case downstream delays that need to be accounted for during each polling period, which add up into the double-digit millisecond range (but less than the desired button scan period). Touch2Key has been tuned to account for the worst-case response time.
9 Tuning the Capacitive Touch Buttons, Using the TouchPro GUI

TI provides a GUI to help tune MSP430 capacitive touch buttons, called the TouchPro GUI. You write code into your device that streams touch count values to the PC over a virtual COM port, and the GUI displays them. You can then:

- See how responsive your buttons are
- See your noise level
- Increase or decrease your measurement window, to balance the tradeoff of time and power vs. sensitivity
- Set touch thresholds

![The TouchPro GUI](image.png)

Figure 80. The TouchPro GUI

The GUI has a complete user’s guide and examples to help in its use.

In developing this BoosterPack, it was necessary to tune the buttons, just as with any other design would need to do. To help demonstrate use of the TouchPro GUI, the software used for this has been retained in the LaunchPad and BoosterPack examples, and this chapter has been written to help you understand how it was done.
9.1 System Communication

In most touch applications, the MSP430 performing the touch measurements would be the same MCU sending the results to the PC. But as with all communication with the HapTouch BoosterPack, it must send its data via the LaunchPad.

The TouchPro GUI allows the MSP430 to send a variable number of button elements, as long as the data fits a predefined packet format. It expects the MSP430 to repeatedly send these packets. As quickly as it receives these packets, it updates its PC display.

In the HapTouch system, the LaunchPad takes the initiative to request the BoosterPack to sends its button counts, then forms a valid TouchPro packet and sends it over the UART. It does this repeatedly, on a fixed period.

Before it can do this, it needs to put the BoosterPack into a special mode where the BoosterPack gathers not button status, but rather button counts – the output of the TI_CAPT_Custom() call of the capacitive touch library.

Since the LaunchPad is performing a function it does not otherwise do, it must be triggered to go into this mode. This is done with the P1.3 pushbutton on the LaunchPad. If it is held down at powerup, the software irreversibly goes into this state (until the next power cycle).

The communication occurs as in the diagram below.
GUI is open and running, with COM port to LaunchPad open

LaunchPad powers up

Is P1.3 pushbutton held down?

YES

NO

Ordinary operation (not touch tuning)

LaunchPad sends CMD_TOUCHTUNE to BoosterPack

BoosterPack enters TOUCHTUNE_STATE

BoosterPack begins periodically updating an array containing button touch counts

BoosterPack responds

LaunchPad sends CMD_READBACK → RB_INDEX_BTNCOUNTS

LaunchPad packetizes the data and sends it to the GUI

GUI displays the data

Figure 81. Communication During Touch Tuning
9.2 Setup

First, download the TouchPro GUI and install it. Then, start the GUI. If you used the default location, its path will be similar to:

C:\ti\msp430\TouchPro_x_xx_xx_xx\32bit

The GUI is a Java file, so it does not have an *.exe file to open. Instead, double-click on touchPro.bat.

With the LaunchPad not attached to the PC, mount the BoosterPack into it; hold down the P1.3 pushbutton, and attach to the PC. After some initial LED activity on the BoosterPack, then should eventually all turn off (except for the power LED). This tells you that the LaunchPad and BoosterPack are both in TouchTune mode.

The GUI can locate LaunchPads on the PC. Select your LaunchPad’s backchannel COM port. The connection button should show “Connected”.

The GUI has two views: “Oscilloscope” and “Bar Chart”. Initially, it is in “Oscilloscope” mode, and like an oscilloscope, the values being sent from the LaunchPad roll across the screen in the time domain. Initially these values are all very low. As you touch a button on the BoosterPack, you should see one of the lines move sharply upward, and vice versa when you release it.

If you do not see this, then something is wrong:

- Check the LEDs on the BoosterPack; they should all be off, except the power LED.
- Check to make sure the icon in the upper right of the GUI window says “Connected”.

Sometimes the GUI’s connection with the LaunchPad becomes corrupted. It may be necessary to shut down the GUI, and re-start it.

The other view is “Bar Chart”. Click on “Bar Chart” in the radio buttons at the top:
The “Threshold” setting can be used to help you see how the capacitive touch library’s threshold setting might appear, for any given touch element. At least initially, you might want to turn it all the way down so you can see the sensitivity at the lowest levels.

9.3 Tuning

You are now ready to begin tuning. To some extent, you can see intuitively in the GUI how well the buttons are responding. Push each button and see its response.

If the threshold is too high, a light touch may get caught up in baseline tracking and quickly disappear. If this happens, lower the threshold.

In principle, tuning a touch pushbutton is easy, compared to more complex controls like sliders: set each Element struct’s maxResponse field to the maximum value, and set the threshold value to half of this. This is sufficient for very basic bench work, but it is insufficient for a product. Ultimately, the threshold for each element should be halfway between the lowest intended press’ value, and the highest unintended press’ value.

For the discrete pushbuttons (SELECT/START/B/A), this is fairly simple. The wheel’s buttons are more difficult. This is because, on the 45-degree angle virtual buttons, your press is divided between two physical buttons. This means the counts are lower on each physical button than they would be if you were pressing that button only. Therefore, the lowest intended press on each physical button is significantly reduced.

Similarly, on this game controller, you may use your thumb to reach across unintended buttons to reach the one you want. This may result in some pressing of buttons you do not intend, and this may register a small amount of counts on those unintended buttons. This raises the counts for the highest unintended press.

This highlights a good point: always tune buttons in the way you intend them to actually be used. Hold the game controller in the same way as you intend to when used, etc. It can be easy to treat the buttons differently during tuning, with realizing it.

When you find the highest unintended value, and lowest intended value, divide that distance in half, and this can be your threshold value. It is best to find this values for multiple boards, and use a threshold that will perform well on all boards in production. Boards can vary in their performance over volume.

There is a document specifically for tuning pushbuttons – Capacitive Touch Sensing, MSP430 Button Gate Time Optimization and Tuning Guide (slaa574). It is recommended to use this document as a guide.
10 Ideas for Modifications

The BoosterPack is really meant as a tool to give you ideas for your own designs and to help you learn the MSP430TCH5E Haptics Library. It does a lot of things, but still leaves open a lot of functionality you might choose to implement on your own. Some ideas include:

- Use the BoosterPack with different gaming websites or platforms
- Generating new sequences and button profiles
- Port the projects to GCC or IAR
- Drive the MSP430TCH5E from a host MCU besides the LaunchPad. Wire the BoosterPack to another MCU using the external host header, and perhaps port the LaunchPad code to that processor since it contains the necessary command handling.
- Add proximity sensing. There is a signal PROX_BTN on the schematic that could be used to implement proximity sensing, but was not implemented in software. You might try adding this yourself; see the Capacitive Touch Library documentation.
- Modify the audio haptics configuration settings to use a different audio input. Perhaps you could add an auto-adaptive algorithm that automatically senses the audio amplitude and adapts the configuration accordingly?
- Change the audio interface to use a kind of audio input than ADC10. Perhaps stream a digital input from a PC?
- Use the haptics actuator of your choice, besides the ERM and LRA used on the BoosterPack. De-solder one of these actuators and solder your own instead. The MSP430TCH5E Haptics Library Designer’s Guide contains information about configuring it for various actuators.
- Similarly, perhaps you could put the actuator on a different PCB, and wire to it from the BoosterPack’s MSP430TCH5E or DRV2603.
- Modify the serial interface to use SPI instead of I2C. (This is the reason the RESERVED0 and RESERVED1 signals are wired to the BoosterPack interface.)
- Modify the LaunchPad software to use a timer-based UART instead of the USCI-based UART; this would allow it to run on other devices that run on the G2 LaunchPad, besides the MSP430G2553.
11 Suggested Reading and Resources

11.1 The LaunchPad Websites

More information about the G2 LaunchPad, supported Booster Packs, and available resources can be found at:

- The G2 LaunchPad’s tool page: resources specific to the G2 LaunchPad
- TI’s LaunchPad portal: information about all LaunchPads from TI, for all MCUs
- The LaunchPad wiki: design resources and example projects from the community

11.2 Information on the MSP430TCH5E

At some point, you will probably want detailed information about the MSP430TCH5E device. For every MSP430 device, the documentation is organized as shown below.

<table>
<thead>
<tr>
<th>Document</th>
<th>For MSP430TCH5E</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user's guide for the device’s “family”</td>
<td>The MSP430x2xx Family User's Guide (SLAU144)</td>
<td>Architectural information about the device, including clocks, timers, A/D, etc.</td>
</tr>
<tr>
<td>The device’s data sheet</td>
<td>The MSP430TCH5E data sheet (SLAS895)</td>
<td>Device-specific information and all parametric information for this device</td>
</tr>
</tbody>
</table>

11.3 Information on the DRV2603

You will need information on this driver device. A data sheet can be found on the page linked above. (At the time of writing, only a very limited data sheet is obtained when clicking on the data sheet link. The full data sheet is available by clicking on the “Request Full Datasheet” link immediately below the data

Figure 83. DRV2603 Product Page: Requesting the Full Data Sheet
11.4 Download Code Composer Studio

If emulating or building the code, you will need Code Composer Studio. At this point in time, the library and the HapTouch SDK only support development with Code Composer Studio. Development with IAR and MSP430GCC has not been tested.

11.5 The MSP430 Haptics Library Designer’s Guide

The designer’s guide (SLAU543) contains much more detail on the library and deeper information about haptics applications.

11.6 The MSP430 Capacitive Touch Resources

TI has many materials available for doing capacitive touch on the MSP430. Many of these can be found from the tool folder page for the MSP430 Capacitive Touch Library.

11.7 MSP430Ware and the TI Resource Explorer

MSP430Ware is a complete collection of libraries and tools. By default, MSP430Ware is included in a CCS installation. CCS and MSP430GCC users must download it separately.

MSP430Ware includes the TI Resource Explorer, for easy browsing of the tools.

11.8 The MSP430TCH5E Code Examples

This is a set of very simple code examples that demonstrate how to use the MSP430TCH5E’s entire set of peripherals: ADC10, Timer_A, etc. Every MSP430 derivative has a set of these code examples. When writing code that uses a peripheral, they can often serve as a starting point.

11.9 MSP430 Application Notes

There are many application notes at www.ti.com/msp430, with practical design examples and topics.

11.10 The TI E2E Community

Search the forums at e2e.ti.com. If you cannot find your answer, post your question to the community.

11.11 The Community at Large

Many online communities focus on the MSP430 – for example, http://www.43oh.com. You can find additional tools, resources, and support from these communities.
## BoosterPack Bill of Materials (BOM)

<table>
<thead>
<tr>
<th>Qty</th>
<th>Part Designators</th>
<th>Value</th>
<th>Description</th>
<th>Manufacturer Part Number</th>
<th>Manufacturer</th>
<th>Footprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>C1, C3, C5, C6</td>
<td>100nF</td>
<td>X7R Ceramic Capacitor,16V,10%,SMD</td>
<td>GRM188R71C104KA01D</td>
<td>MURATA</td>
<td>0603</td>
</tr>
<tr>
<td>4</td>
<td>C2, C4, C7, C46</td>
<td>10uF</td>
<td>X5R Ceramic Capacitor,16V,10%,SMD</td>
<td>GRM21BR61C106KE15L</td>
<td>MURATA</td>
<td>0805</td>
</tr>
<tr>
<td>6</td>
<td>D1, D2, D3, D4, D5, D7</td>
<td>Red</td>
<td>Chip LED,Red</td>
<td>15-21SURC/5530-A2/TR8</td>
<td>Everlight</td>
<td>1206</td>
</tr>
<tr>
<td>1</td>
<td>D6</td>
<td>Blue</td>
<td>Chip LED,Blue</td>
<td>15-21/BHC-AN1P2/2T</td>
<td>Everlight</td>
<td>1206</td>
</tr>
<tr>
<td>2</td>
<td>FB1, FB2</td>
<td>220R</td>
<td>Ferrite Bead,220ohm@100MHz,25%,2A,0.045ohm,SMD</td>
<td>BLM21PG221SN1D</td>
<td>MURATA</td>
<td>0805</td>
</tr>
<tr>
<td>2</td>
<td>J1, J2</td>
<td>Socket 2x10</td>
<td>Double Row Header Sockets,2.54mm Pitch,20 pins</td>
<td>FH254-S180-850-210WP</td>
<td>JXT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>J12</td>
<td>Header 2x7</td>
<td>Double Row Pin Header,2.54mm Pitch,6 pins</td>
<td>P101-2*07MGF-116-1X</td>
<td>TOWNES</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>J13</td>
<td>Header 2x3</td>
<td>Double Row Pin Header,2.54mm Pitch,6 pins</td>
<td>PD254NV-06GR310</td>
<td>JXT</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>J3</td>
<td>KJ-328</td>
<td>3.5mm ,Audio Jack,Right Angle,Black,SMD</td>
<td>KJ328KYS-1BBI02A</td>
<td>Kaler</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>M1</td>
<td>LRA</td>
<td>LRA Vibrator Motor ,Ø10×3.6mm</td>
<td>ELV1036C</td>
<td>AAC</td>
<td>Ø10×3.6mm</td>
</tr>
<tr>
<td>1</td>
<td>M2</td>
<td>ERM</td>
<td>ERM Vibrator Motor,15.5x4.5x4.85mm</td>
<td>RP1342</td>
<td>ZLIFE</td>
<td>15.5x4.5x</td>
</tr>
<tr>
<td>3</td>
<td>R1, R7, R35</td>
<td>0R</td>
<td>Resistor,5%,1/10W,SMD</td>
<td>RC0603JR-070RL</td>
<td>YAGEO</td>
<td>0603</td>
</tr>
<tr>
<td>1</td>
<td>R10</td>
<td>820R</td>
<td>Resistor,5%,1/10W,SMD</td>
<td>RC0603JR-07820RL</td>
<td>YAGEO</td>
<td>0603</td>
</tr>
<tr>
<td>2</td>
<td>R11, R12</td>
<td>49.9K</td>
<td>Resistor,1%,1/10W,SMD</td>
<td>RC0603FR-0749K9L</td>
<td>YAGEO</td>
<td>0603</td>
</tr>
<tr>
<td>7</td>
<td>R13, R14, R15, R16, R17, R18, R21</td>
<td>470R</td>
<td>Resistor,5%,1/10W,SMD</td>
<td>RC0603JR-07470RL</td>
<td>YAGEO</td>
<td>0603</td>
</tr>
<tr>
<td>8</td>
<td>R2, R4, R5, R19, R20, R22, R23, R24</td>
<td>4.7K</td>
<td>Resistor,5%,1/10W,SMD</td>
<td>RC0603JR-074K7L</td>
<td>YAGEO</td>
<td>0603</td>
</tr>
<tr>
<td>1</td>
<td>R8</td>
<td>330R</td>
<td>Resistor,5%,1/10W,SMD</td>
<td>RC0603JR-07330RL</td>
<td>YAGEO</td>
<td>0603</td>
</tr>
<tr>
<td>2</td>
<td>S1, S2</td>
<td>SS-2205S-WV1</td>
<td>Slide Switch,Right Angle,SMD</td>
<td>SS-2205S-WL11</td>
<td>TONE PARTS</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>U1</td>
<td>DRV2603</td>
<td>DC Motor Driver</td>
<td>DRV2603RUNT</td>
<td>TI</td>
<td>QFN10</td>
</tr>
<tr>
<td>1</td>
<td>U2</td>
<td>MSP430TCH5E</td>
<td>MSP430 Microcontroller</td>
<td>MSP430TCH5E</td>
<td>TI</td>
<td>QFN32</td>
</tr>
</tbody>
</table>
Appendix C. Individual Haptics Effects

This table shows all 123 individual haptics effects that can be driven from the MSP430TCH5E Haptics Library.

The binary numbers in the “LEDs” column reflect the values shown by the LEDs embedded within the wheel, SELECT, START, B, and A buttons. Together these LEDs represent a five-digit binary number, where if the LED is on, it is considered a “1”, and if off is considered a “0”.

The far right columns give suggested scenarios in which the effect might be used in an end application.

Note that the HapTouch GUI organizes these effects into groups, allowing the user to more easily find the desired effect.

<table>
<thead>
<tr>
<th>LED Indicators</th>
<th>Button#</th>
<th>Effect Index (in hex)</th>
<th>Description</th>
<th>Button</th>
<th>Alert</th>
<th>Gesture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wheel → A LEDs</td>
<td>SELECT 0</td>
<td>0</td>
<td>Strong Click – 100%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start 1</td>
<td>1</td>
<td>Strong Click – 60%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B 2</td>
<td>2</td>
<td>Strong Click – 30%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A 3</td>
<td>3</td>
<td>Sharp Click – 100%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000</td>
<td>SELECT 4</td>
<td>4</td>
<td>Sharp Click – 60%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start 5</td>
<td>5</td>
<td>Sharp Click – 30%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B 6</td>
<td>6</td>
<td>Soft Bump – 100%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A 7</td>
<td>7</td>
<td>Soft Bump – 60%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00010</td>
<td>SELECT 8</td>
<td>8</td>
<td>Soft Bump – 30%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start 9</td>
<td>9</td>
<td>Double Click – 100%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B A</td>
<td>A</td>
<td>Double Click – 60%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A B</td>
<td>B</td>
<td>Triple Click – 100%</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>00011</td>
<td>SELECT C</td>
<td>C</td>
<td>Soft Fuzz – 60%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start D</td>
<td>D</td>
<td>Strong Buzz – 100%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B E</td>
<td>E</td>
<td>Empty</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>A F</td>
<td>F</td>
<td>Empty</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>00100</td>
<td>SELECT 10</td>
<td>10</td>
<td>Strong Click 1 – 100%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start 11</td>
<td>11</td>
<td>Strong Click 2 – 80%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B 12</td>
<td>12</td>
<td>Strong Click 3 – 60%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A 13</td>
<td>13</td>
<td>Sharp Click 4 – 30%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101</td>
<td>SELECT 14</td>
<td>14</td>
<td>Medium Click 1 – 100%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start 15</td>
<td>15</td>
<td>Medium Click 2 – 80%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B 16</td>
<td>16</td>
<td>Medium Click 3 – 60%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED Indicators</td>
<td>Button#</td>
<td>Effect Index (in hex)</td>
<td>Description</td>
<td>Button</td>
<td>Alert</td>
<td>Gesture</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>----------------------</td>
<td>-------------</td>
<td>--------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>Wheel→A LEDs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>17</td>
<td></td>
<td>Sharp Tick 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>00110 SELECT</td>
<td>18</td>
<td></td>
<td>Sharp Tick 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Start</td>
<td>19</td>
<td></td>
<td>Sharp Tick 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>1A</td>
<td></td>
<td>Short Double Click Strong 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>1B</td>
<td></td>
<td>Short Double Click Strong 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>00111 SELECT</td>
<td>1C</td>
<td></td>
<td>Short Double Click Strong 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Start</td>
<td>1D</td>
<td></td>
<td>Short Double Click Strong 4 – 30%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>1E</td>
<td></td>
<td>Short Double Click Medium 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>1F</td>
<td></td>
<td>Short Double Click Medium 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>01000 SELECT</td>
<td>20</td>
<td></td>
<td>Short Double Click Medium 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Start</td>
<td>21</td>
<td></td>
<td>Short Double Sharp Tick 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>22</td>
<td></td>
<td>Short Double Sharp Tick 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>23</td>
<td></td>
<td>Short Double Sharp Tick 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>01001 SELECT</td>
<td>24</td>
<td></td>
<td>Long Double Sharp Click Strong 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Start</td>
<td>25</td>
<td></td>
<td>Long Double Sharp Click Strong 2 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>26</td>
<td></td>
<td>Long Double Sharp Click Strong 3 – 30%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>27</td>
<td></td>
<td>Long Double Sharp Click Strong 4 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>01010 SELECT</td>
<td>28</td>
<td></td>
<td>Long Double Sharp Click Medium 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Start</td>
<td>29</td>
<td></td>
<td>Long Double Sharp Click Medium 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>2A</td>
<td></td>
<td>Long Double Sharp Click Medium 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>2B</td>
<td></td>
<td>Long Double Sharp Tick 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>01011 SELECT</td>
<td>2C</td>
<td></td>
<td>Long Double Sharp Tick 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Start</td>
<td>2D</td>
<td></td>
<td>Long Double Sharp Tick 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>2E</td>
<td></td>
<td>Buzz 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>2F</td>
<td></td>
<td>Buzz 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>01100 SELECT</td>
<td>30</td>
<td></td>
<td>Buzz 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Start</td>
<td>31</td>
<td></td>
<td>Buzz 4 – 40%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>32</td>
<td></td>
<td>Buzz 5 – 20%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>33</td>
<td></td>
<td>Pulsing Strong 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>01101 SELECT</td>
<td>34</td>
<td></td>
<td>Pulsing Strong 2 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Start</td>
<td>35</td>
<td></td>
<td>Pulsing Medium 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>B</td>
<td>36</td>
<td></td>
<td>Pulsing Medium 2 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>A</td>
<td>37</td>
<td></td>
<td>Pulsing Sharp 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>LED Indicators</td>
<td>Button#</td>
<td>Effect Index (in hex)</td>
<td>Description</td>
<td>Button</td>
<td>Alert</td>
<td>Gesture</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>-----------------------</td>
<td>-------------</td>
<td>--------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>Wheel→A LEDs</td>
<td>SELECT</td>
<td>38</td>
<td>Pulsing Sharp 2 – 60%</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>39</td>
<td>Transition Click 1 – 100%</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>3A</td>
<td>Transition Click 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>3B</td>
<td>Transition Click 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>3C</td>
<td>Transition Click 4 – 40%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>3D</td>
<td>Transition Click 5 – 20%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>3E</td>
<td>Transition Click 6 – 10%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>3F</td>
<td>Transition Hum 1 – 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>40</td>
<td>Transition Hum 2 – 80%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>41</td>
<td>Transition Hum 3 – 60%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>42</td>
<td>Transition Hum 4 – 40%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>43</td>
<td>Transition Hum 5 – 20%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>44</td>
<td>Transition Hum 6 – 10%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>45</td>
<td>Transition Ramp Down Long Smooth 1 – 100 to 0%</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>46</td>
<td>Transition Ramp Down Long Smooth 2 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>47</td>
<td>Transition Ramp Down Medium Smooth 1 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>48</td>
<td>Transition Ramp Down Medium Smooth 2 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>49</td>
<td>Transition Ramp Down Short Smooth 1 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4A</td>
<td>Transition Ramp Down Short Smooth 2 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4B</td>
<td>Transition Ramp Down Long Sharp 1 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>4C</td>
<td>Transition Ramp Down Long Sharp 2 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>4D</td>
<td>Transition Ramp Down Medium Sharp 1 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4E</td>
<td>Transition Ramp Down Medium Sharp 2 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4F</td>
<td>Transition Ramp Down Short Sharp 1 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>50</td>
<td>Transition Ramp Down Short Sharp 2 – 100 to 0%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>51</td>
<td>Transition Ramp Up Long Smooth 1 – 0 to 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>52</td>
<td>Transition Ramp Up Long Smooth 2 – 0 to 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>53</td>
<td>Transition Ramp Up Medium Smooth 1 – 0 to 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>54</td>
<td>Transition Ramp Up Medium Smooth 2 – 0 to 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>55</td>
<td>Transition Ramp Up Short Smooth 1 – 0 to 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>56</td>
<td>Transition Ramp Up Short Smooth 2 – 0 to 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>57</td>
<td>Transition Ramp Up Long Sharp 1 – 0 to 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td>58</td>
<td>Transition Ramp Up Long Sharp 2 – 0 to 100%</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>LED Indicators</td>
<td>Button#</td>
<td>Effect Index (in hex)</td>
<td>Description</td>
<td>Button</td>
<td>Alert</td>
<td>Gesture</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>----------------------</td>
<td>-------------</td>
<td>--------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>Wheel→A LEDs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10110</td>
<td>Start</td>
<td>59</td>
<td>Transition Ramp Up Medium Sharp 1 – 0 to 100%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>5A</td>
<td>Transition Ramp Up Medium Sharp 2 – 0 to 100%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>5B</td>
<td>Transition Ramp Up Short Sharp 1 – 0 to 100%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>10111</td>
<td>SELECT</td>
<td>5C</td>
<td>Transition Ramp Up Short Sharp 2 – 0 to 100%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>5D</td>
<td>Transition Ramp Down Long Smooth 1 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>5E</td>
<td>Transition Ramp Down Long Smooth 2 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>5F</td>
<td>Transition Ramp Down Medium Smooth 1 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11000</td>
<td>SELECT</td>
<td>60</td>
<td>Transition Ramp Down Medium Smooth 2 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>61</td>
<td>Transition Ramp Down Short Smooth 1 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>62</td>
<td>Transition Ramp Down Short Smooth 2 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>63</td>
<td>Transition Ramp Down Long Sharp 1 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11001</td>
<td>SELECT</td>
<td>64</td>
<td>Transition Ramp Down Long Sharp 2 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>65</td>
<td>Transition Ramp Down Medium Sharp 1 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>66</td>
<td>Transition Ramp Down Medium Sharp 2 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>67</td>
<td>Transition Ramp Down Short Sharp 1 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11010</td>
<td>SELECT</td>
<td>68</td>
<td>Transition Ramp Down Short Sharp 2 – 50 to 0%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>69</td>
<td>Transition Ramp Up Long Smooth 1 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>6A</td>
<td>Transition Ramp Up Long Smooth 2 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>6B</td>
<td>Transition Ramp Up Medium Smooth 1 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11011</td>
<td>SELECT</td>
<td>6C</td>
<td>Transition Ramp Up Medium Smooth 2 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>6D</td>
<td>Transition Ramp Up Short Smooth 1 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>6E</td>
<td>Transition Ramp Up Short Smooth 2 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>6F</td>
<td>Transition Ramp Up Long Sharp 1 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11100</td>
<td>SELECT</td>
<td>70</td>
<td>Transition Ramp Up Long Sharp 2 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>71</td>
<td>Transition Ramp Up Medium Sharp 1 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>72</td>
<td>Transition Ramp Up Medium Sharp 2 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>73</td>
<td>Transition Ramp Up Short Sharp 1 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11101</td>
<td>SELECT</td>
<td>74</td>
<td>Transition Ramp Up Short Sharp 2 – 0 to 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>75</td>
<td>Long buzz for programmatic stopping – 100%, 10 seconds max</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>76</td>
<td>Smooth Hum 1 (No kick or brake pulse) – 50%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>77</td>
<td>Smooth Hum 2 (No kick or brake pulse) – 40%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>11110</td>
<td>SELECT</td>
<td>78</td>
<td>Smooth Hum 3 (No kick or brake pulse) – 30%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>79</td>
<td>Smooth Hum 4 (No kick or brake pulse) – 20%</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>LED Indicators</td>
<td>Button#</td>
<td>Effect Index (in hex)</td>
<td>Description</td>
<td>Button</td>
<td>Alert</td>
<td>Gesture</td>
</tr>
<tr>
<td>----------------</td>
<td>---------</td>
<td>-----------------------</td>
<td>-------------</td>
<td>--------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>Wheel→A LEDs</td>
<td>B</td>
<td>7A</td>
<td>Smooth Hum 5 (No kick or brake pulse) – 10%</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>7B</td>
<td>Empty</td>
<td>NA</td>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>11111</td>
<td>SELECT</td>
<td>7C</td>
<td>Empty</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Start</td>
<td>7D</td>
<td>Empty</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>7E</td>
<td>Empty</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>7F</td>
<td>Empty</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have not been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products

| Audio       | www.ti.com/audio |
| Amplifiers  | amplifier.ti.com |
| Data Converters | dataconverter.ti.com |
| DLP® Products | www.dlp.com |
| DSP         | dsp.ti.com |
| Clocks and Timers | www.ti.com/clocks |
| Interface   | interface.ti.com |
| Logic       | logic.ti.com |
| Power Mgmt  | power.ti.com |
| Microcontrollers | microcontroller.ti.com |
| RFID        | www.ti-rfid.com |
| OMAP Applications Processors | www.ti.com/omap |
| Wireless Connectivity | www.ti.com/wirelessconnectivity |
| Applications | Automotive and Transportation | www.ti.com/automotive |
|             | Communications and Telecom | www.ti.com/communications |
|             | Computers and Peripherals | www.ti.com/computers |
|             | Consumer Electronics | www.ti.com/consumer-apps |
|             | Energy and Lighting | www.ti.com/energy |
|             | Industrial | www.ti.com/industrial |
|             | Medical | www.ti.com/medical |
|             | Security | www.ti.com/security |
|             | Space, Avionics and Defense | www.ti.com/space-avionics-defense |
|             | Video and Imaging | www.ti.com/video |
|             | TI E2E Community | e2e.ti.com |
|             | TI E2E Community | e2e.ti.com |

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2013, Texas Instruments Incorporated