Luis Reynoso

TI Designs: Reference Designs
Keyboard Controller using MSP430

TEXAS INSTRUMENTS

TI Designs

TI Designs provide the foundation that you need including methodology, testing and design files to quickly evaluate and customize and system. TI Designs help ***you*** accelerate your time to market.

Design Resources

TIDM-KEYBOARD	Design Folder
MSP430F5529	Product Folder
MSP430G2744	Product Folder
TPS73533	Product Folder
TPD2E001	Product Folder

TI E2E™ Community

ASK Our Analog Experts
WEBENCH® Design Center

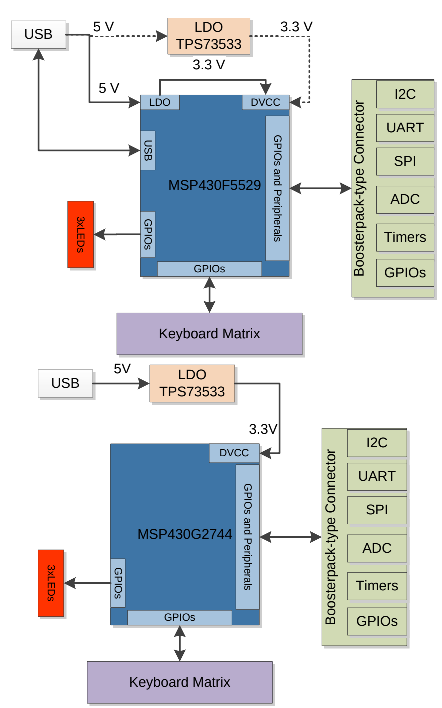
Design Features

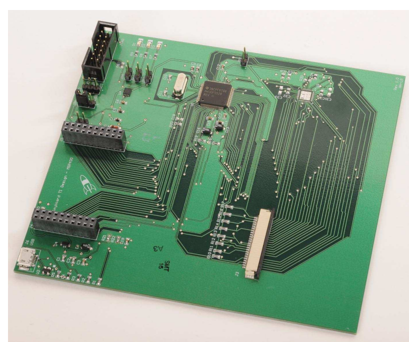
- Low-power implementation
- Cost-effective
- Customizable for different keyboard layouts
- Supports different communication interfaces (USB and I2C examples included)
- Supports multimedia keys
- "Ghost" key detection
- Composite USB allows users to send custom data through HID-datapipe

Featured Applications

- PC Keyboards
- Gaming
- Sensor Hub Aggregation
- Smart TV

Block Diagram





1 System Description

This reference design describes the implementation of a keyboard controller with the following characteristics:

- Supports standard matrix keyboards: this design shows an implementation using a 15x8 matrix but different keyboard layouts can be used
- Independent of the communication interface: examples for USB and I2C are included
- HID compliant: can interface directly with PC using USB or HID Over I2C.
- “Ghost” key handling in software: prevents incorrect key detection from multiple simultaneous key presses
- HID boot protocol support: allows keyboard to be used to interface with a PC’s BIOS
- Supports multimedia keys: common multimedia and power keys are implemented
- Low power consumption: device goes to low power mode when idle
- Composite USB device: an HID-datapipe back-channel is implemented to send custom data to the PC
- Can be implemented in practically any MSP430 platform: examples for MSP430F5529 and MSP430G2744 are included

1.1 MSP430 Family of Microcontrollers

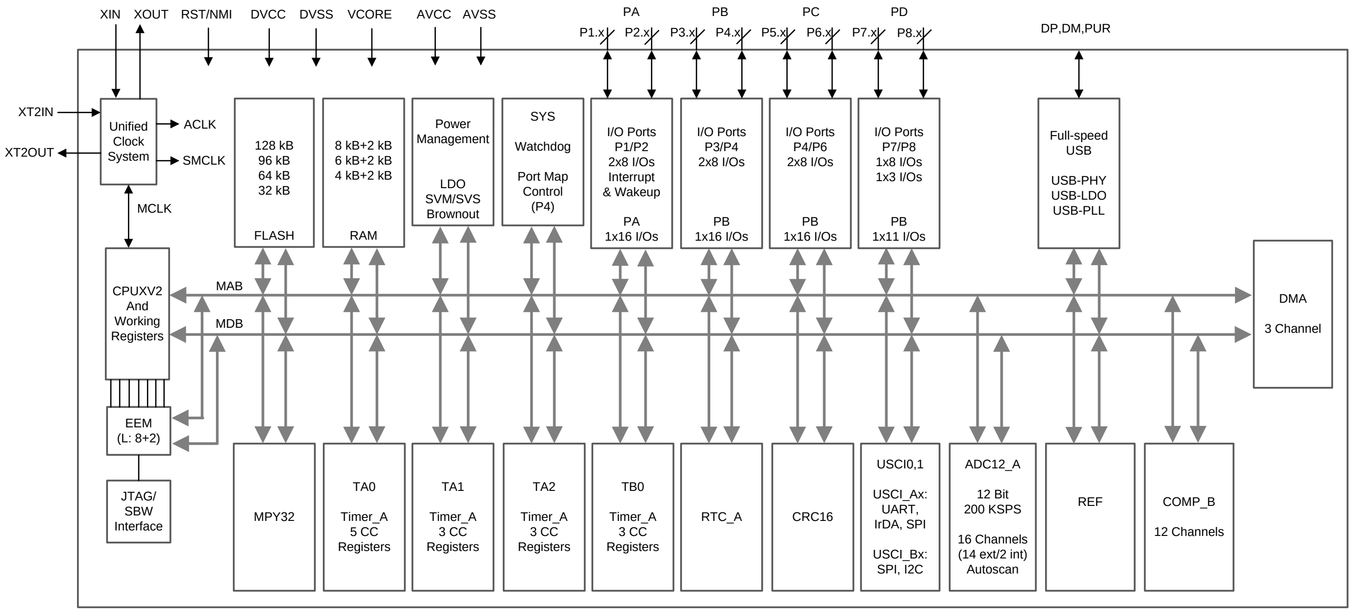
The Texas Instruments MSP430 family of ultra-low-power microcontrollers consists of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with extensive low-power modes, is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency.

The software included in the design can be migrated to practically any MSP430 with enough GPIOs, but the available examples are implemented for MSP430F5529 and MSP430G2744.

1.1.1 MSP430F5529

The MSP430F552x series of microcontrollers include an integrated USB and PHY supporting USB 2.0 full-speed communication, four 16-bit timers, a high-performance 12-bit analog-to-digital converter (ADC), two universal serial communication interfaces (USCI), hardware multiplier, DMA, real-time clock module with alarm capabilities, and up to 63 I/O pins.

This reference design uses the MSP430F5529 derivative to show an implementation using USB or I2C, but with plenty of resources left to allow for further customization. This device is the superset of the family, with 128KB of Flash, 10KB of RAM and 63 I/Os in an 80-pin LQFP package.



Note: Memory size and available peripherals and ports may vary, depending on the device.

Figure 1. Functional Block Diagram – MSP430F552x

1.1.2 MSP430G2744

The MSP430G2x44 series is an ultra-low-power mixed signal microcontroller with two built-in 16-bit timers, a universal serial communication interface (USCI), 10-bit analog-to-digital converter (ADC) with integrated reference and data transfer controller (DTC), and up to 32 I/O pins. Typical application include sensor systems that capture analog signals, convert them to digital values, and then process the data for display or for transmission to a host system. Stand-alone radio-frequency (RF) sensor front ends are another area of application.

This reference design also uses the MSP430G2744 to show an I2C implementation using a smaller device from the value-line family of MSP430 microcontrollers. This device is the superset of the MSP430G2x44 family with 32KB of Flash, 1KB of RAM and is available in 40-QFN, 38-TSSOP and the ultra-small 49-DSBGA package for space constrained application.

Although this implementation is more limited in resources, some pins and enough memory is available for further customization.

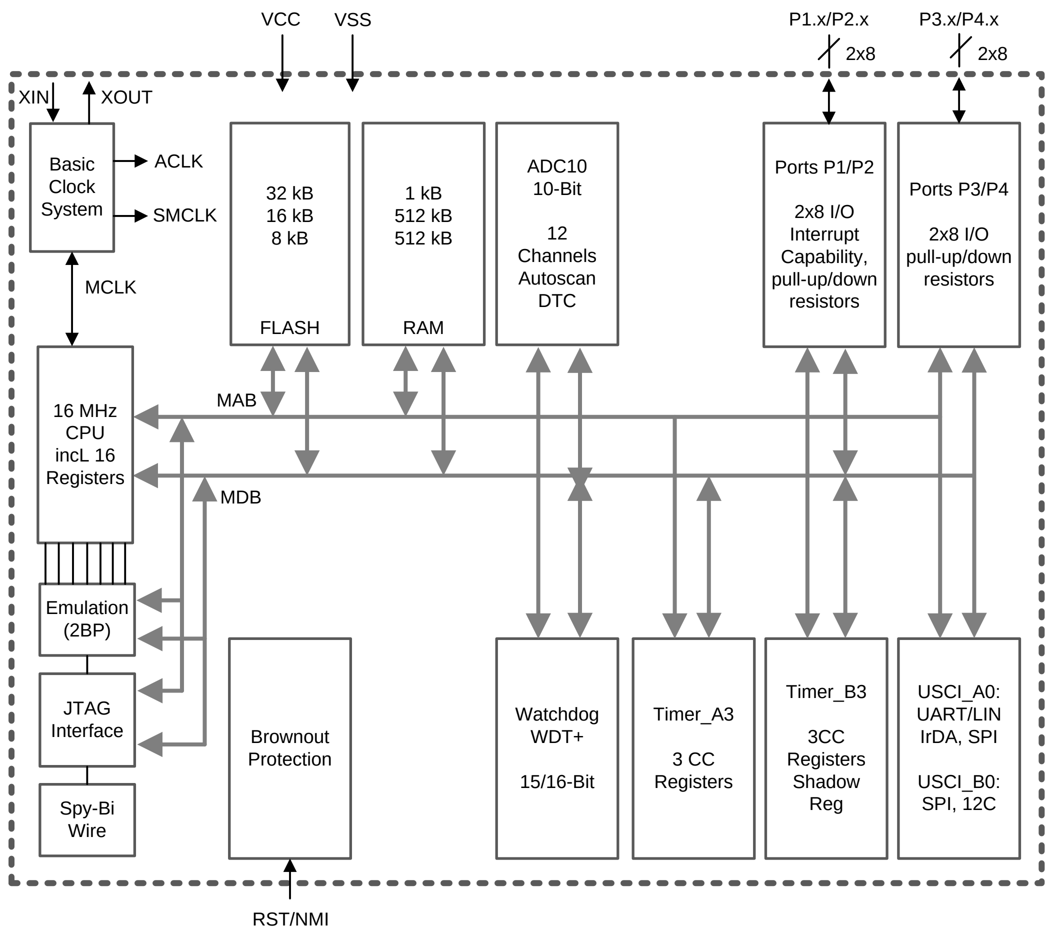


Figure 2. Functional Block Diagram – MSP430G2x44

1.2 TPS73533

The TPS735xx family of low-dropout (LDO), low-power linear regulators offers excellent AC performance with very low ground current. High power-supply rejection ratio (PSRR), low noise, fast start-up, and excellent line and load transient response are provided while consuming a very low 46µA (typical) ground current. The TPS735xx is stable with ceramic capacitors and uses an advanced BiCMOS fabrication process to yield a typical dropout voltage of 250mV at 500mA output. The TPS735xx uses a precision voltage reference and feedback loop to achieve overall accuracy of 2% (VOUT > 2.2V) over all load, line, process, and temperature variations. It is fully specified from TJ = –40°C to +125°C and is offered in low-profile, 2mm × 2mm SON and 3mm × 3mm SON packages that are ideal for wireless handsets, printers, and WLAN cards.

This reference design uses the TPS73533 regulator to convert 5V from USB to 3.3V used by the MSP430 microcontrollers.

2 Block Diagram

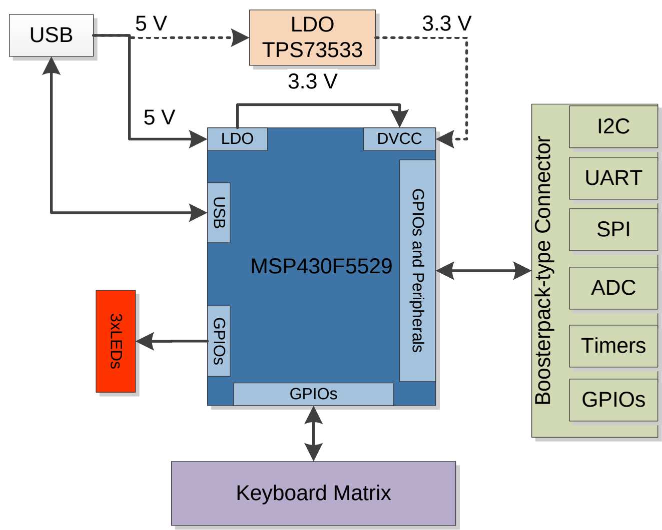


Figure 3. Block Diagram using MSP430F5529

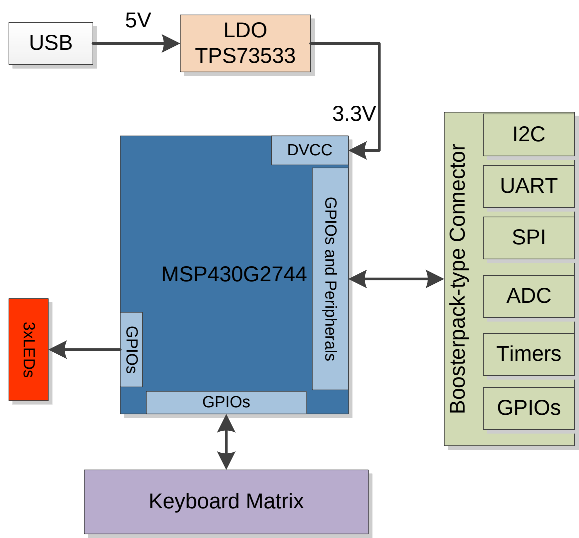


Figure 4. Block Diagram using MSP430G2744

3 System Design Theory

3.1 Key Matrix

The keyboard controller presented in this document implements a key matrix of rows and columns similar to smaller keypads like the one shown in the application report *Implementing An Ultralow-Power Keypad Interface with MSP430* (SLAA139).

The implementation shown uses a 15 rows x 8 column matrix, which allows up to 120 keys, but it only uses 84 keys in total.

The key matrix used implemented in this example is shown in Figure 5.

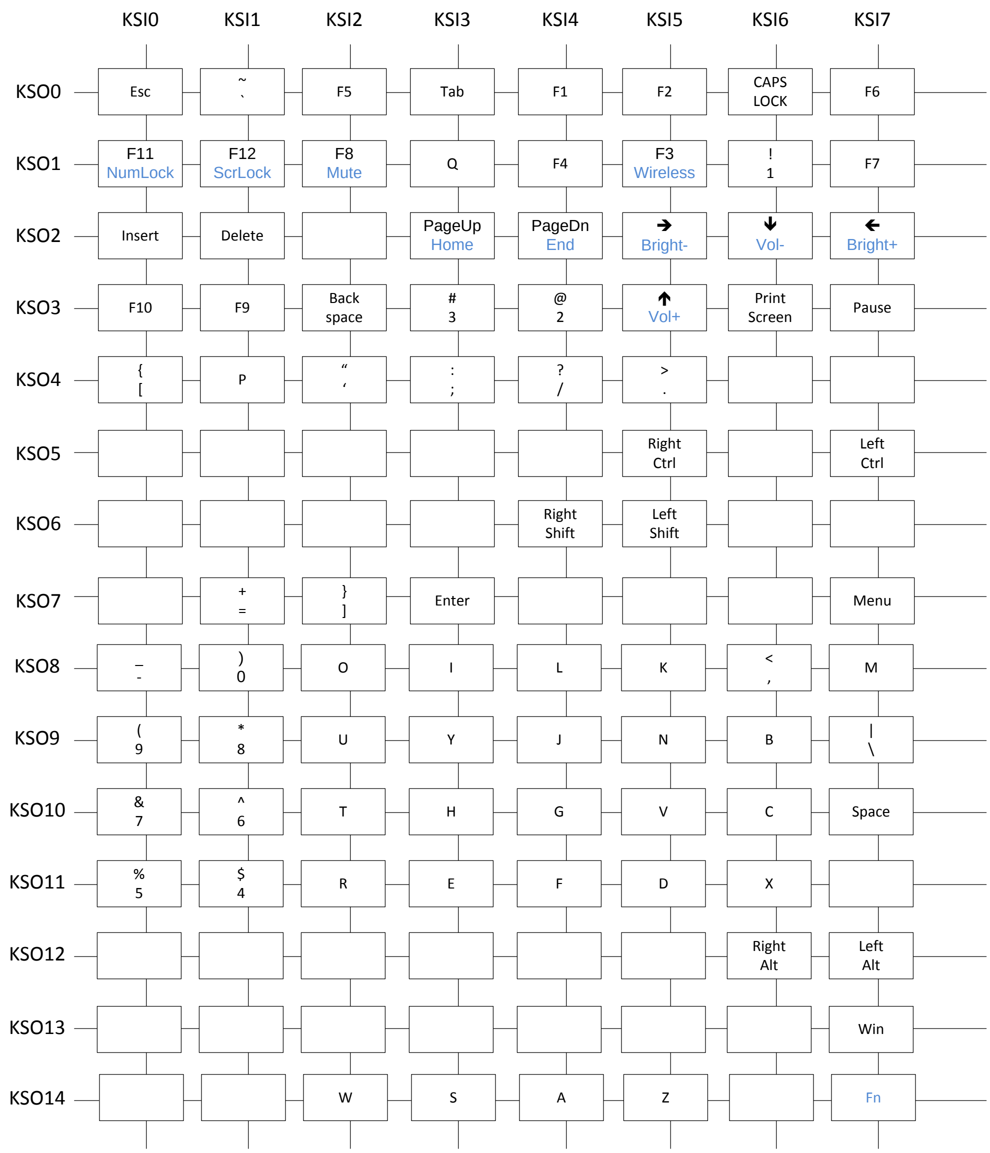


Figure 5. Key Matrix

Each key works like a switch and pull-ups are required for each of the columns (KSI pins), keeping the idle state high (see Figure 6).

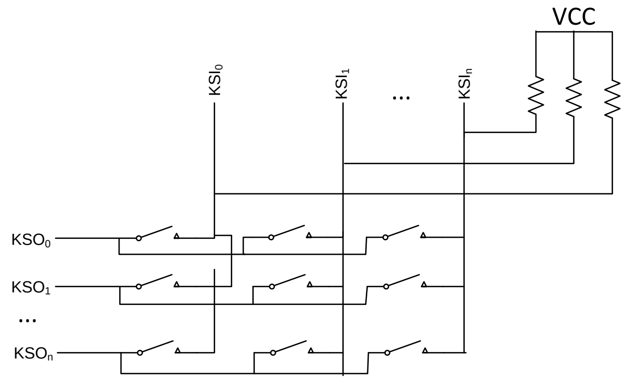


Figure 6. Keyboard Schematic Model

There are multiple ways to scan a key matrix, but this implementation uses two methods, referred in this application report as: column-interrupt and polling.

In the column-interrupt approach, all KSO pins are actively driven at the same time and KSI pins are configured to interrupt the processor when any key is pressed.

This method is useful in low-power modes, because any key can wake up the microcontroller; however, it is important to remark that the key press is only used for that purpose, because this method does not provide the exact key being pressed.

Figure 7 shows the key matrix behavior when the “Enter” Key is pressed in column-interrupt mode. Pressing this key will close a path between KSO7 and KSI3, thus causing a state change in KSI3. This is shown by red lines which indicate the lines which are not in an idle state. Notice that KSI3 would detect the event when the “Enter” key is pressed, but the effect would be the same for any other pin pressed in the same column.

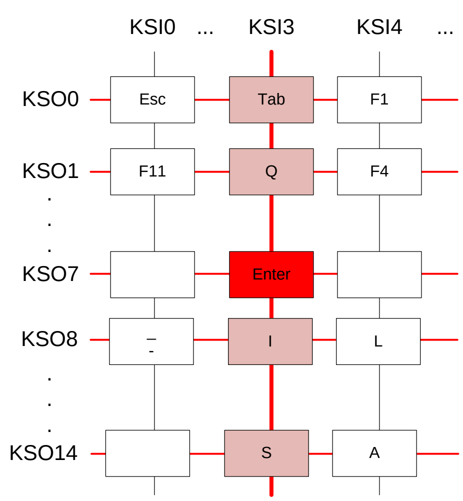


Figure 7. Detection of a Key Using Column-Interrupt Method

After the system is awake due to a key press using the column-interrupt approach, the Polling method can be used to determine which key(s) is (are) being pressed.

In the Polling method, each row is scanned separately by driving one KSO at a time in sequential order. KSI pins are then read giving the exact keys being pressed.

The following image shows the result of pressing the same “Enter” key in Polling method. When KSO7 is driven, the pressed key will close a path between KSO7 and KSI3. Since all the other KSO pins are idle, we know that the key has coordinates (KSI3, KSO7).

Note that KSI pins are in idle state when the rest of KSO pins are driven since all the switches are open and there is no path between KSO and KSI pins.

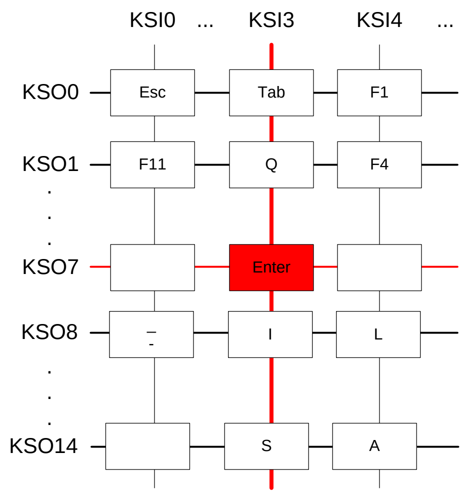


Figure 8. Detection of a Key Using Polling Method

3.1.1 "Ghost" Key Detection

One of the caveats when using the polling method is that particular patterns can cause unwanted connections, known as “ghost” keys. This behavior is caused when three or more keys sharing rows and columns are pressed at the same time.

The following image shows a “ghost” key condition caused by pressing 3 keys at a time.

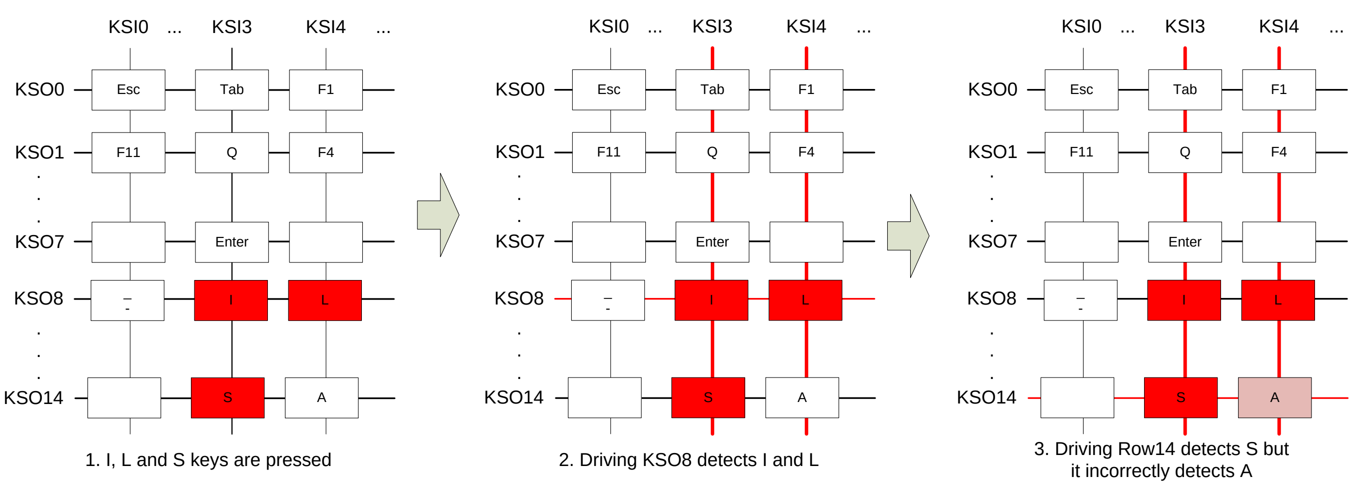


Figure 9. Ghost Key Detection

The software included in this application report detects potential “ghost” keys and does not report them to the host.

In addition, the software also detects unimplemented keys which can’t cause “ghost” keys, even when 3 keys are pressed at the same time. This condition is shown in the following image.

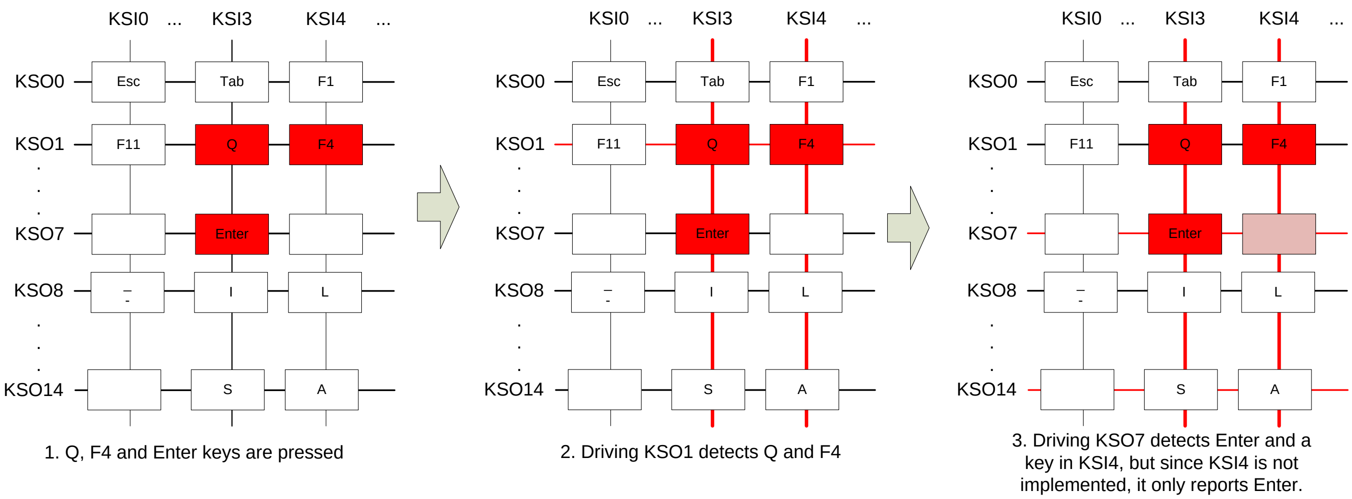


Figure 10. Ignored "Ghost" Key Condition Due to Unimplemented Key

3.2 USB HID

This application report uses the MSP430 application programming interface (API) stack found in the MSP430 USB Developers Package (msp430usbdevpack).

The stack is configured to work as a composite HID interface with the following interfaces:

- HID0: Standard Keyboard
- HID1: DataPipe
- HID2: Consumer Control (Multimedia keys)
- HID3: Wireless Radio Control

Since all interfaces are HID-compliant, no drivers are required.

Basic keyboard implementations only need the Standard keyboard interface to report keys to the host and control the keyboard LEDs.

The DataPipe interface is optional but it allows the MSP430 to not only perform the job of a digital keyboard, but also to do other jobs taking advantage of the same USB interface and the rest of the peripherals. Some examples include: reporting the status of sensors which are read using the ADC, controlling actuators using timer PWMs, etc.

It should be noted that while the host OS interprets and uses the data from the standard keyboard interface without additional applications or drivers, in the case of the DataPipe interface, a host application is required. Texas Instruments provides a Java-based HID Demo which enables communication between a PC and a MSP430 microcontroller running the HID API stack. The Java HID Demo is available in executable format and source code in the MSP430 USB Developers Package (msp430usbdevpack).

The Consumer Control and Wireless Radio Control interface were added to show the implementation of function (Fn) keys. It's important to remark that there's some standardization for these keys, but in some cases, the implementation depends on the vendor. The function keys implemented in this software are:

Table 1. Supported Function (Fn) Keys

Key	Function	Interface
Fn + F8	Mute	Consumer Control
Fn + F11	NumLock	Consumer Control
Fn + F2	Scroll Lock	Consumer Control
Fn + UP	Increase Volume	Consumer Control
Fn + Down	Decrease Volume	Consumer Control
Fn + Left	Increase Brightness	Consumer Control
Fn + Right	Decrease Brightness	Consumer Control
Fn + F3	Turn ON/OFF Wireless	Wireless Radio Control

The keyboard interface supports Boot protocol, which allows it to work with HID-limited hosts (such as some BIOS).

The VID and PID can be modified according to the particular application but the default code used for this example uses the following values:

Table 2. VID/PID Used by the Device

VID	0x2047
PID	0x0401

3.3 HID over I2C

This application report uses the HIDI2C Development API for MSP430 (ti_hidi2c_msp430).

The stack is configured to work as a single HID interface with the following report IDs:

- 0x01: Standard Keyboard
- 0x03: Consumer Control (Multimedia keys)
- 0x04: Wireless Radio Control

Since the interface is HID-compliant, no drivers are required.

Basic keyboard implementations only need the Standard keyboard report ID in order to report keys to the host and control the keyboard LEDs.

The Consumer Control and Wireless Radio Control reports are used to show the implementation of function (Fn) keys. It's important to remark that there's some standardization for these keys, but in some cases, the implementation depends on the vendor. The function keys implemented in this software are shown in Table 1.

3.4 Software

The following figure shows the software layers for the keyboard controller:

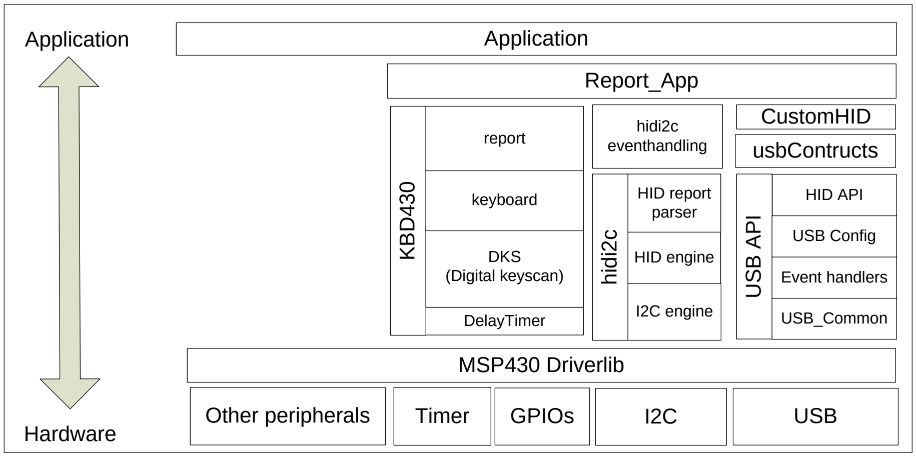


Figure 11. Software Architecture

Software is designed in a modular way, re-using existing TI libraries and adding new modules from low-level drivers to application level.

These modules include:

- **Application**
  Description
  Main application initializing the microcontroller and peripherals, and executing a loop checking and servicing the rest of the modules.
  Files
  : .\Projects*\Src\main.c
  Flow diagram:

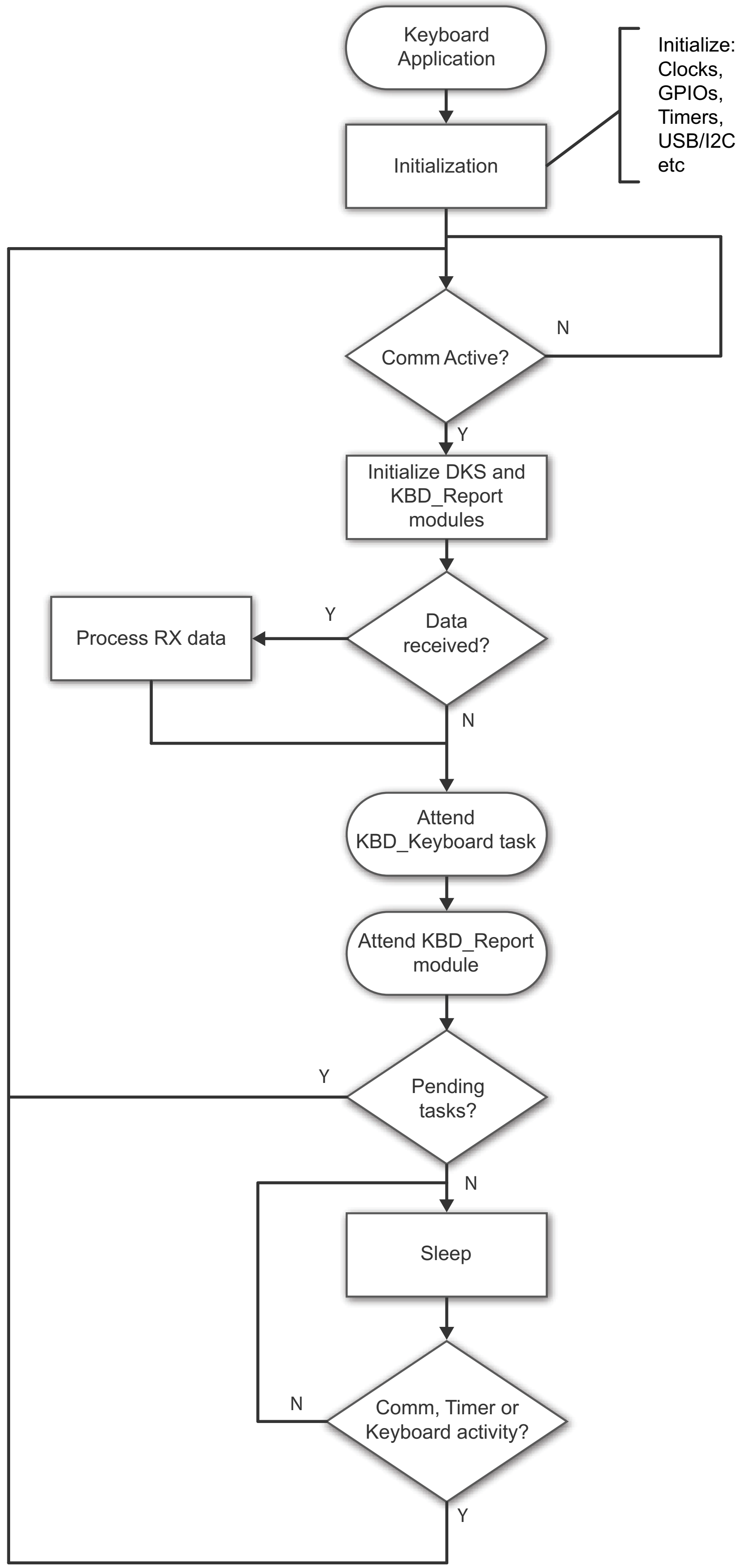


Figure 12. Application Flow Diagram

- **Report_App**
  Description
  This file provides an abstraction layer between application, DKS and the communication interface. When the DKS detects a new key, it will call the callback function KBD430_ReportSend. This function can then send the data to the corresponding communication interface (for example, USB or I2C).
  Files:
  .\Projects*\Src\KBD430_report_App.c
- **KBD430_Report**
  Description
  Handles the HID Keyboard report, adding and removing keys from the report on press/release events, then sends the data to the Report_App layer.
  Files:
  .\KBD430\Src\KBD430_report.c .\KBD430\Include\KBD430_report.h
  HID Keyboard Report format:

Table 3. Standard Keyboard Input Report

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	Right GUI	Right Alt	Right Shift	Right Ctrl	Left GUI	Left Alt	Left Shift	Left Ctrl
Byte1	Reserved							
Byte2	Key_array[0]							
Byte3	Key_array[1]							
Byte4	Key_array[2]							
Byte5	Key_array[3]							
Byte6	Key_array[4]							
Byte7	Key_array[5]							

Table 4. Standard Keyboard Output Report

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	Ignored					ScrollLock	CAPSLock	NumLock

Table 5. Consumer Control Input Report

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	Bright-	Bright+	Play/Pause	PrevTrack	NextTrack	Mute	Vol-	Vol+

Table 6. Wireless Radio Control Input Report

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Byte0	Unused							Wireless Toggle

- **KBD430_Keyboard**
  Description
  This layer gets the keys from the DKS module and reports them to the KBD430_Report module. It can handle special key combinations such as Function (Fn) keys.
  Files:
  .\KBD430\Src\KBD430_Keyboard.c
  .\KBD430\Include\KBD430_Keyboard_public.h
  Flow diagram:

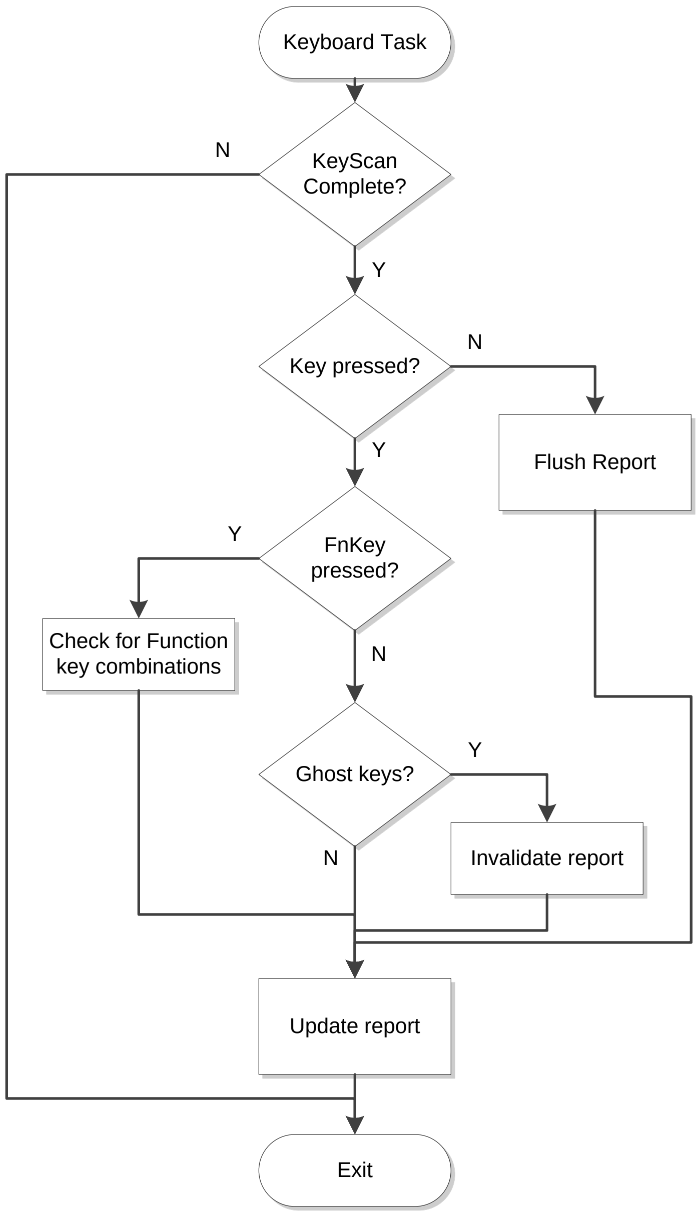


Figure 13. Flow Diagram for Keyboard Task

- **KBD430_DKS (Digital Keyscan)**
  Description
  This layer handles the digital keyboard scanning, detecting key press/release events, and reporting them to higher layers.
  Files:
  .\KBD430\Src\KBD430_DKS.c
  .\KBD430\Include\KBD430_DKS.h
  State Diagram:

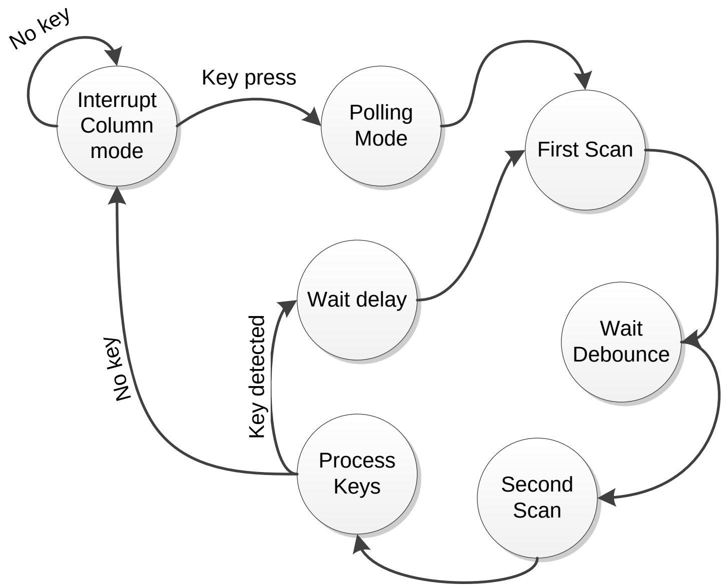


Figure 14. State Diagram for Keyboard Scan

- **KBD430_DelayTimer**
  Description
  This module handles a general purpose interrupt timer used to implement a delay.
  This timer is implemented using TA0.0.
  Files:
  .\KBD430\Src\KBD430_delaytimer.c
  .\KBD430\Include\KBD430_delaytimer.h

- **CustomHID**
  Description
  This layer handles the HID Custom interface, which is used to transfer data to/from an USB host. The current implementation shows a template that can be used for custom development. This module uses the HID-Datapipe as defined in the USB API included in MSP430 USB Developers Package (msp430usbdevpack).
  Files:
  .\Projects\USBKBD\Src\CustomHID.c
  .\Projects\USBKBD\Src\CustomHID.h
  Custom HID Report format:

Table 7. CustomHID Report Descriptor

Field	Size	Description
IN Report		
Report ID	1 Byte	Report ID (automatically assigned to 0x3F by the HID-datapipe calls)
Size	1 Byte	Number of valid bytes in the data field
Data	62 Bytes	Data payload
OUT Report		
Report ID	1 Byte	Report ID (automatically assigned to 0x3F by the HID-datapipe calls)
Size	1 Byte	Number of valid bytes in the data field
Data	62 Bytes	Data payload

In addition, the following TI libraries used by this design are:

- MSP430 DriverLib: Driver Library's abstracted API keeps you above the bits and bytes of the MSP430 hardware by providing easy-to-use function calls. Thorough documentation is delivered through a helpful API Guide, which includes details on each function call and the recognized parameters. Developers can use Driver Library functions to write complete projects with minimal overhead. DriverLib is used in this project to initialize MSP430F5529 peripherals and perform basic functions.
  Files:
  .\driverlib\MSP430F5xx_6xx*.*
  .\driverlib\MSP430F5xx_6xx\inc*.h
- MSP430 USB Developers Package: The USB Developers Package for MSP430 is a software package containing all necessary source code and sample applications required for developing a USB-based MSP430 project. The package only supports MSP430 USB devices.
  The USB API is used in the USBKBD configuration to enable USB and utilize the HID class.
  Files:
  .\USB_API*.*
  .\Projects\USBKBD\Src\USB_config*.*
  .\Projects\USBKBD\Src\USB_App*.*
- HIDI2C API for MSP430: Development API driver for Microsoft HIDI2C Protocol for the Texas Instruments MSP430.
  The HIDI2C API is used in the I2CKBD and I2CKBD_G2xx4 configurations to enable HID over I2C.
  Files:
  .\hidi2c*.*
  .\Projects\I2CKBD\Src\HIDI2C*.*
  .\Projects\I2CKBD_G2xx4\Src\HIDI2C*.*

3.5 Hardware

The hardware included in this reference design provides users with the flexibility to test and develop their keyboard controller application using two different microcontrollers: MSP430F5529 and MSP430G2744.

NOTE: The hardware included in this Reference Design has support for the MSP430F5529 or MSP430G2744, but it doesn't support both devices at the same time. Populating both devices could cause electrical problems. Check the corresponding BOMs in Section 8.1.

The evaluation board contains the following connectors common to both board configurations:

Table 8. Connectors in Evaluation Board

Connector	F5529	G2744
J1	Standard 2x7 JTAG/SBW (only SBW is supported)	
J2	Provide external VCC and GND	
J3	24-pin Keyboard connector. Check Section 4.1 for information about the keyboard used by this reference design	
J4-J5	Boosterpack-compatible connector.	
	Check Figure 15	Check Figure 16
J6	Used to power board and/or communication with host	Used to power board

The board also includes jumpers to provide more flexibility to the developer (options in bold shows the default configuration):

Table 9. Jumpers in Evaluation Board

Jumper	F5529	G2744
JP1	1-2 VUSB: Use VUSB (MSP30F5529 internal LDO) for VCC. **2-3 LDO: Uses TPS73533 for VCC** OFF: EXT power using J2	1-2 VUSB: Unused **2-3 LDO: Uses TPS73533 for VCC** OFF: EXT power using J2
JP2	**ON: Provides power to MSP430** OFF: Allows for power consumption measurement	
JP3	**ON: Enables LED3** OFF: LED3 pin can be used in boosterpack connector	
JP4	**ON: Enables LED2** OFF: LED2 pin can be used in boosterpack connector	
JP5	**ON: Enables LED1** OFF: LED1 pin can be used in boosterpack connector	
JP6	**ON: Connects VUSB to JP1** OFF: Disconnects VUSB from JP1	Unused

3.5.1 Using MPS430F5529

Using KBD430_BOM_F5529 from Table 19, which utilizes the MSP430F5529 microcontroller, provides a lot of flexibility to developers thanks to the microcontroller’s rich set of peripherals, large memory size, and amount of available I/Os. The software included in this reference design supports the following interfaces:

Table 10. Communication Interfaces Supported for MSP430F5529

Target Configuration	Communication Interface
USBKBD	USB
I2CKBD	I2C

In addition, the evaluation board provides access to:

- Additional communication peripherals (for example, UART and SPI): allowing developers to send the keyboard information to the host using other methods, or simply to implement other communication interfaces
- Analog peripherals (ADC and analog comparator): providing flexibility to implement functions such as, reading sensors and transducers, using the same keyboard controller
- Timer input/output pins: allowing implementation of PWMs, pulse detection, custom communication interfaces, etc
- Ample memory resources: allowing for more complex applications, or the implementation of protocols such as Bluetooth
- Up to 35 GPIOs available

The application reserves the following peripherals and pins for keyboard functionality:

Table 11. Peripherals and Pinout Used for MSP430FF529

Function	Description	USBKBD	I2CKBD
DelayTimer	Low power timer delay	TA0.0	TA0.0
USB	Communication with host	PU.0/DP PU.1/DM PUR	N/A
I2C	Communication with host	N/A	SDA:P4.1/UCB1SDA SCL:P4.2/UCB1SCL
I2C_INT	Interrupt output to host	N/A	P1.0
KSO0	Keyboard output (row)	P4.7	
KSO1		P5.4	
KSO2		P5.5	
KSO3		P5.6	
KSO4		P5.7	
KSO5		P6.6	
KSO6		P6.7	
KSO7		P7.0	
KSO8		P7.1	
KSO9		P7.2	
KSO10		P7.3	
KSO11		P7.7	
KSO12		P8.0	
KSO13		P8.1	
KSO14		P8.2	
KSI0	Keyboard input (column)	P2.0	
KSI1		P2.1	
KSI2		P2.2	
KSI3		P2.3	
KSI4		P2.4	
KSI5		P2.5	
KSI6		P2.6	
KSI7		P2.7	
LED1	NumLock LED	P1.1	
LED2	CapsLock LED	P1.6	
LED3	ScrollLock LED	P1.7	

Additional pins are available in J4-J5 connectors which are Boosterpack-compatible:

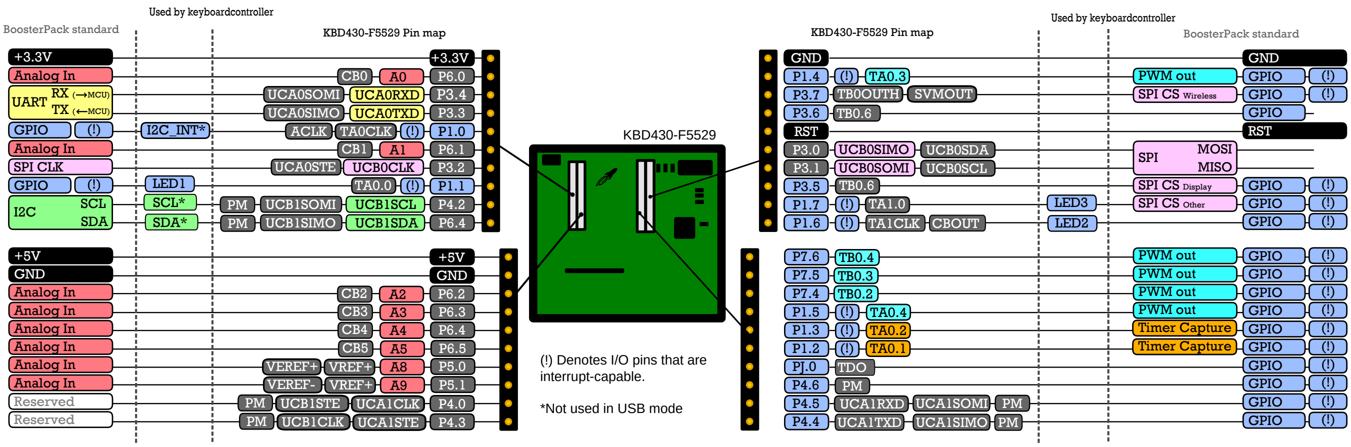


Figure 15. Pinout for Boosterpack Connector using MSP430F5529

3.5.2 Using MSP430G2744

KBD430_BOM_G2744 from Table 20, which utilizes the MSP430G2744 microcontroller, shows a smaller, lower-cost implementation using a value-line device, but it still provides enough flexibility to implement custom functionality in the application.

The software included in this reference design supports the following interfaces:

Table 12. Communication Interfaces Supported for MSP430G2744

Target Configuration	Communication Interface
I2CKBD_G2xx4	I2C

In addition, the evaluation board allows developers to:

- Implement a different communication interface (for example, UART or SPI) to send the keyboard information to the host using other methods, or simply to implement other communication interfaces
- The functionality of LEDs can be disabled allowing developers access to ADC pins to implement functions such as, reading sensors and transducers, using the same keyboard controller; or to Timer input/output pins allowing implementation of PWMs, pulse detection, custom communication interfaces, etc
- 3 GPIOs available by default, and up to 9 available if not using I2C and LEDs.

The application reserves the following peripherals and pins for keyboard functionality:

Table 13. Peripherals and Pinout Used for MSP430G2744

Pin/Peripheral	Description	USBKBD
DelayTimer	Low power timer delay	TA0.0
I2C	Communication with host	SDA:P3.1/UCB0SDA SCL:P3.2/UCB0SCL
I2C_INT	Interrupt output to host	P2.0

Table 13. Peripherals and Pinout Used for MSP430G2744 (continued)

Pin/Peripheral	Description	USBKBD
KSO0	Keyboard output (row)	P2.4
KSO1		P2.5
KSO2		P2.6
KSO3		P2.7
KSO4		P3.0
KSO5		P3.6
KSO6		P3.7
KSO7		P4.0
KSO8		P4.1
KSO9		P4.2
KSO10		P4.3
KSO11		P4.4
KSO12		P4.5
KSO13		P4.6
KSO14		P4.7
KSI0	Keyboard input (column)	P1.0
KSI1		P1.1
KSI2		P1.2
KSI3		P1.3
KSI4		P1.4
KSI5		P1.5
KSI6		P1.6
KSI7		P1.7
LED1	NumLock LED	P2.1
LED2	CapsLock LED	P2.2
LED3	ScrollLock LED	P2.3

Additional pins are available in J4-J5 connectors which are Boosterpack-compatible. Note that many pins are not available because they are reserved for keyboard functions and due to the smaller package of this device.

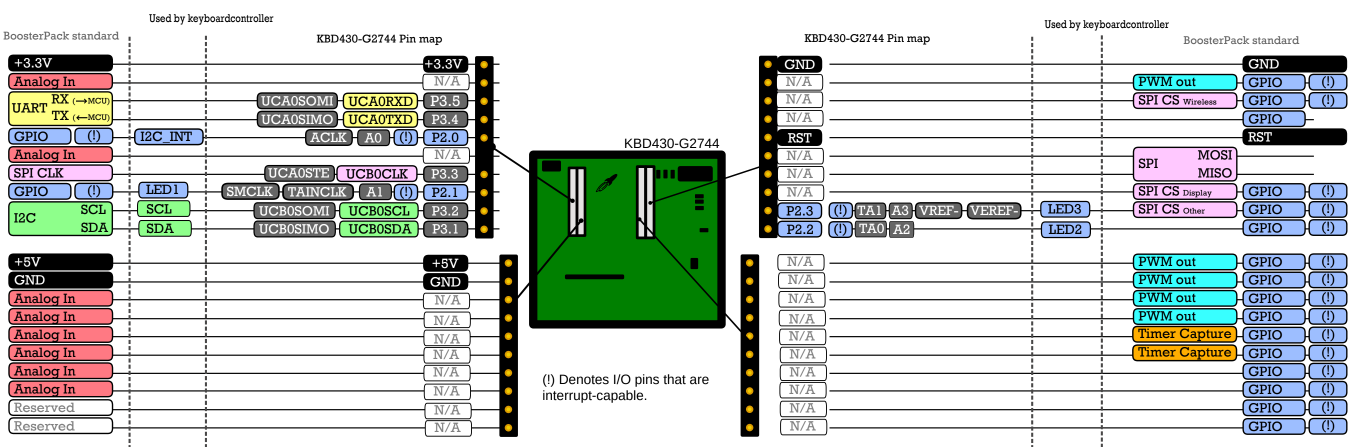


Figure 16. Pinout for Boosterpack Connector using MSP430G2744

4 Getting Started Hardware

4.1 Keyboard

This reference design uses the keyboard **Acer V11102AS1**, which is a replacement for some laptops, including the Acer Aspire One AO532H.

Figure 17. Keyboard used by Reference Design

The software and hardware can be customized for other keyboards as explained in Section 6.3.

4.2 Basic Connections

1. Connect Keyboard to J3

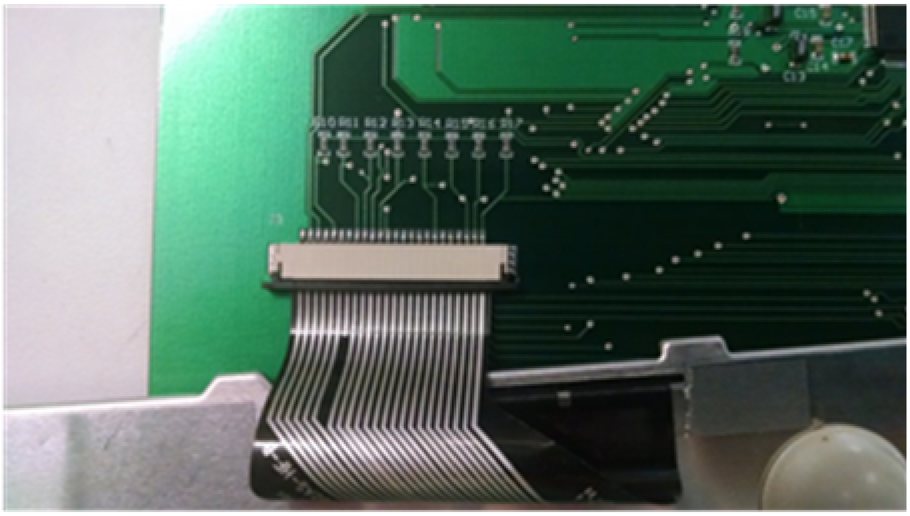


Figure 18. Keyboard Connection

2. Set default jumpers according to Table 9.
3. Optionally, connect I2C host when using *I2CKBD* or *I2CKBD_G2xx4* configurations. Note that there's no standard connector for HID over I2C but the pins are available in boosterpack connector J4-J5 as shown in the following table:

Table 14. I2C Connections

Functions	Connector.pin
I2C SDA	J4.20
I2C SCL	J4.18
I2C INT	J4.10
VCC	J4.2
GND	J4.3

4. Connect USB from PC to J6. This will provide power to the board and it also allows for communication with the host when using *USBKBD* configuration.
5. The device will start running when pre-programmed. Follow steps in Section 4.3 and Section 4.4 to test the USB or I2C applications.
6. Program board if necessary. If the MSP430 hasn't been programmed or when debugging/customizing code:
   - Connect JTAG tool (for example, MSP-FET) to J1
   - Follow steps described in Section 5 to build the software and download code.
   - Run code, or repeat steps 4-5

4.3 Testing USBKBD Configuration

1. Follow steps described in Section 4.2 to execute the application
2. The 3 LEDs on the board will light up in sequence to indicate that the keyboard is running
   - Some LEDs can stay ON depending on the current status of CapsLock, NumLock, and ScrollLock.
3. When connected to a PC, the USB keyboard should be detected by the operating system and enumerated without drivers. Windows shows 5 new devices in the Device Manager (see Figure 19):
   - Human Interface Devices
     – USB Input Device: Standard keyboard in HID0 (MI_00)
     – USB Input Device: Datapipe in HID1 (MI_01)
     – USB Input Device: Consumer Control in HID2 (MI_02)
     – USB Input Device: Wireless Radio Control in HID3 (MI_03)
   - Keyboards
     – HID Keyboard Device: Standard keyboard in HID0 (MI_00)

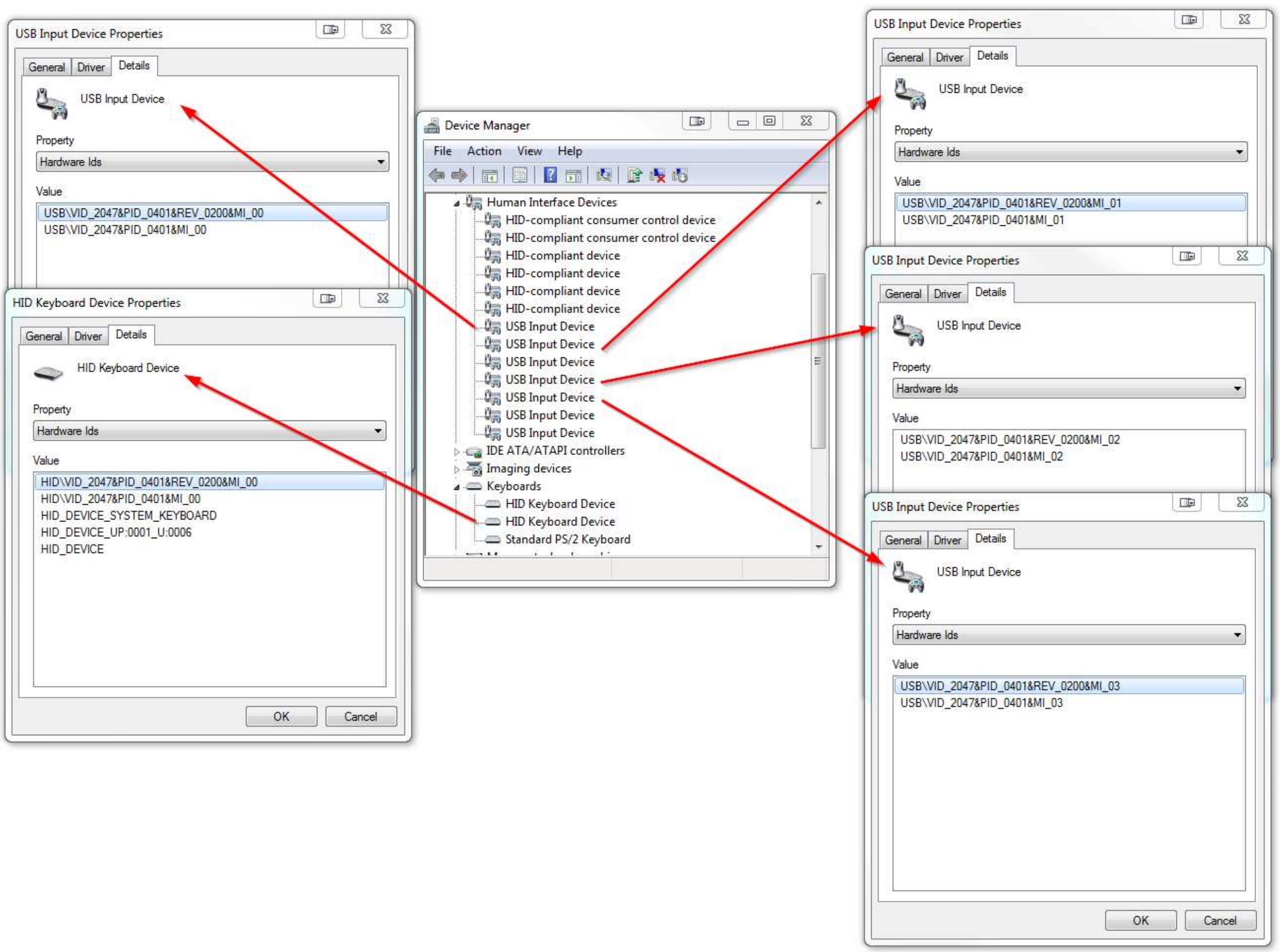


Figure 19. USB Keyboard in Windows Device Manager

4. The keyboard can now be tested and used as a standard keyboard

---

NOTE: The example implements Function keys shown in Table 1 but it's important to remark that the implementation of some of these keys varies depending on the Host.

---

5. In addition to the keyboard functionality, the custom interface can be tested using the MSP430 HID

Demo, available in MSP430 USB Developers Package (msp430usbdevpack).

(a) Open the Java HID Demo.

(b) Select the VID and PD (default: VID = 0x2047, PID = 0x0401).

(c) Click “Set VID PID” button

(d) Click the USB button to connect

(e) The LED should turn green

(f) Write one of the supported commands in the Send and Receive field
1 – Toggles LED1
2 – Toggles LED2
3 – Toggles LED3

(g) Observe the response from USB controller

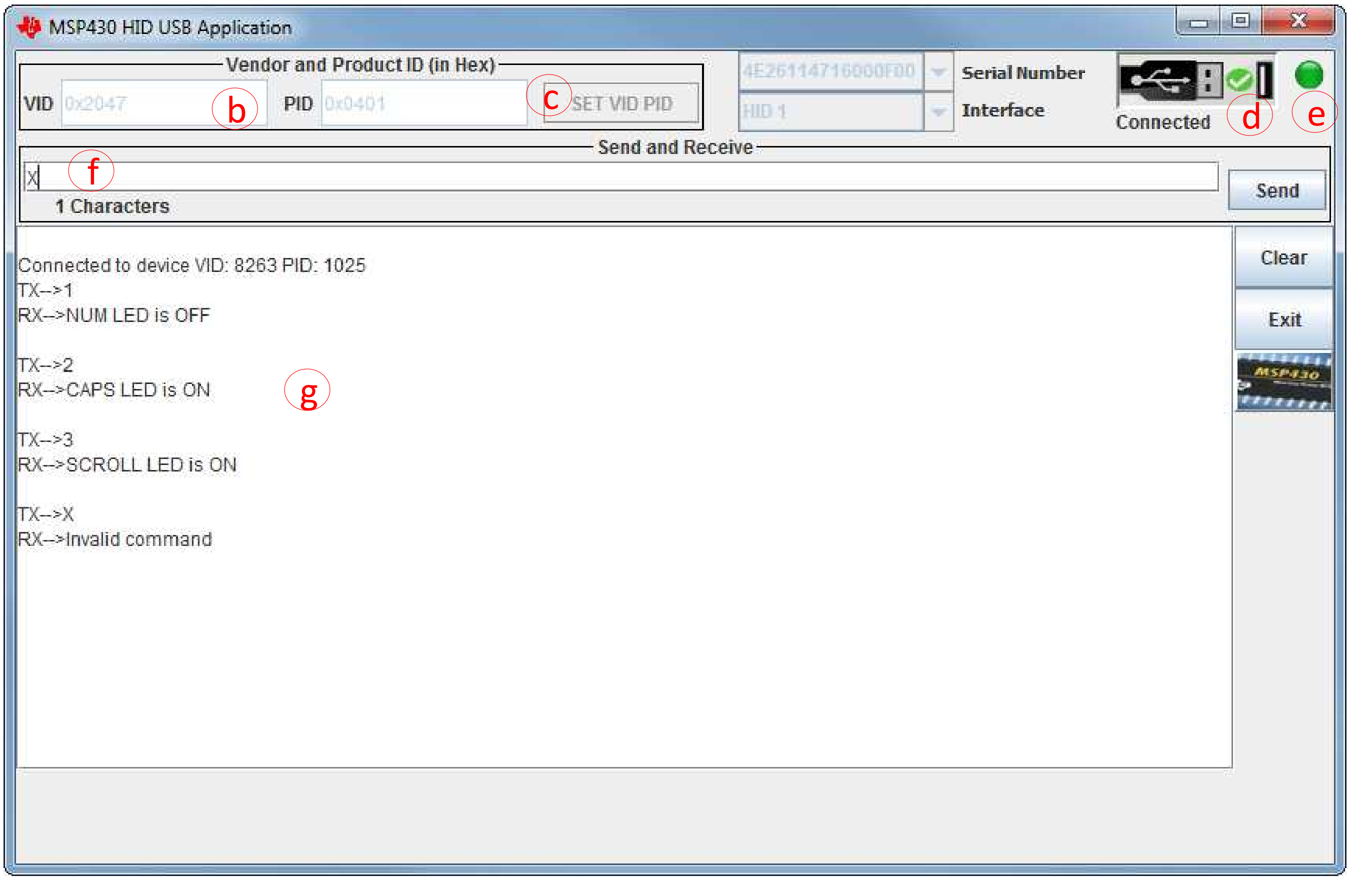


Figure 20. Testing the HID Custom Interface using MSP430 HID Demo

4.4 Testing I2CKBD and I2CKBD_G2xx4 Configurations

1. Follow steps described in Section 4.2 to execute the application
2. The 3 LEDs on the board will light up in sequence to indicate that the keyboard is running
   - Some LEDs can stay ON depending on the current status of CapsLock, NumLock, and ScrollLock.
3. Turn on the I2C Host device, the device will perform enumeration of I2C slave devices during start-up.
4. The keyboard should be detected by the operating system and enumerated without drivers. Windows shows the new HID devices including the keyboard in the device manager:

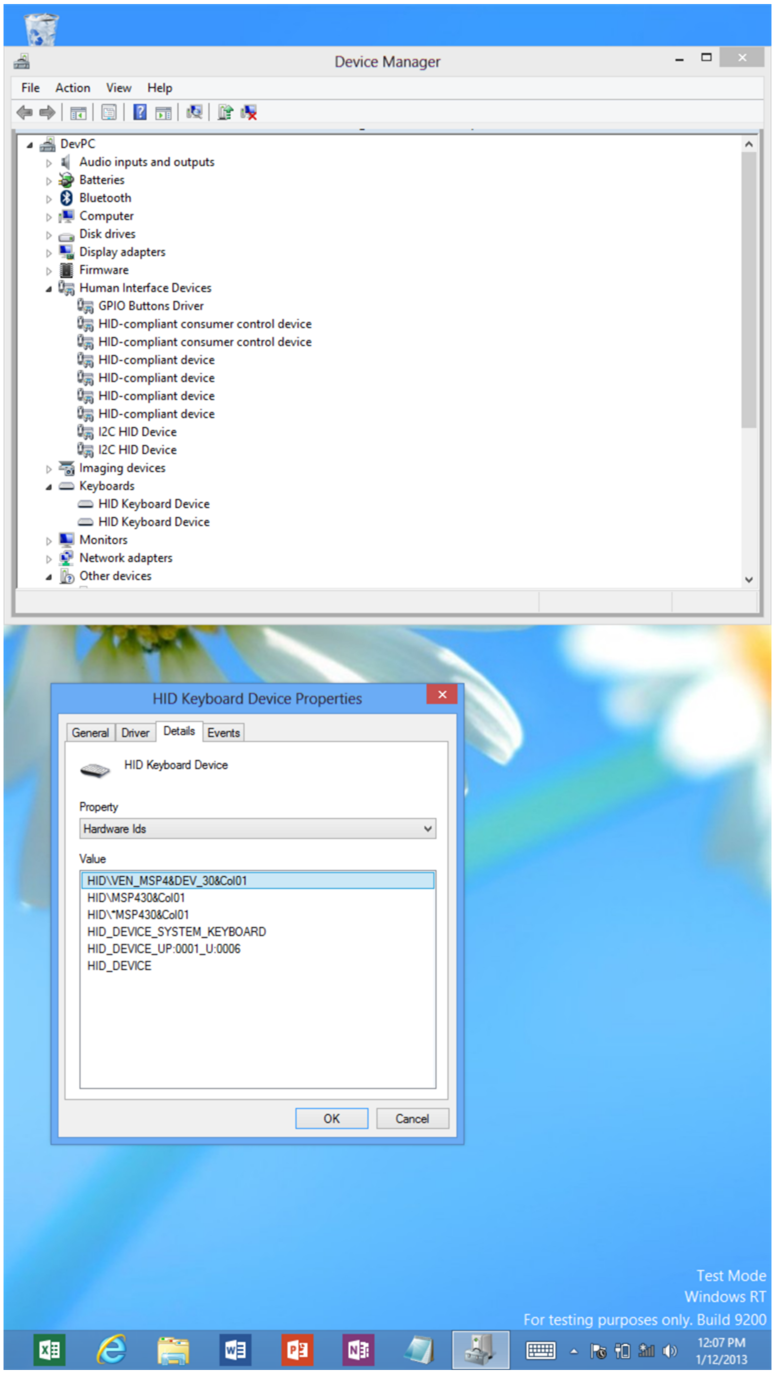


Figure 21. I2C Keyboard in Windows Device Manager

5. The keyboard can now be tested and used as a standard keyboard

NOTE: The example implements Function keys shown in Table 1 but it's important to remark that the implementation of some of these keys varies depending on the Host.

5 Getting Started Firmware

The firmware included in this reference design has the following structure:

```
KBD430_SW
|----driverlib                     ←MSP430 DriverLib
|       |----MSP430F5xx_6xx
|                  |----deprecated
|                  |----inc
|----hidi2c                        ←HIDI2C API for MSP430
|       |----hid
|       |----i2c
|
|----USB_API                       ←MSP430 USB Developers Package
|       |----USB_CDC_API
|       |----USB_Common
|       |----USB_HID_API
|       |----USB_MSC_API
|       |----USB_PHDC_API
|
|----KBD430                        ← Keyboard controller driver
|       |----Include                  ← Header files
|       |----Src                      ← Source code
|
|----Projects
|       |----USBKBD                ← Project supporting USB with MSP430F5529
|             |----CCS                ← CCS project folder
|             |----IAR                ← IAR project folder
|             |----Src                ← Source code for this project
|       |----I2CKBD                   ← Project supporting I2C with MSP430F5529
|             |----CCS                ← CCS project folder
|             |----IAR                ← IAR project folder
|             |----Src                ← Source code for this project
|       |----I2CKBD_G2xx4             ← Project supporting I2C with MSP430G2744
|             |----CCS                ← CCS project folder
|             |----IAR                ← IAR project folder
|             |----Src                ← Source code for this project
```

The projects included in the software package have been built and tested in the following IDEs:

- Code Composer Studio 6.0.1
- IAR for MSP430 6.10.2

The procedure to build code for these IDEs is explained in the following sections.

5.1 Building Projects in IAR

1. Select a project to build:
   - *USBKBD*: USB using MSP430F5529
   - *I2CKBD*: I2C using MSP430F5529
   - *I2CKBD_G2xx4*: I2C using MSP430G2744
2. Open the IAR workspace for the corresponding project:
   `KBD430_SW\Projects\<project>\IAR\<project>.eww`
3. Build project (F7, Menu → Project → Rebuild All, or)

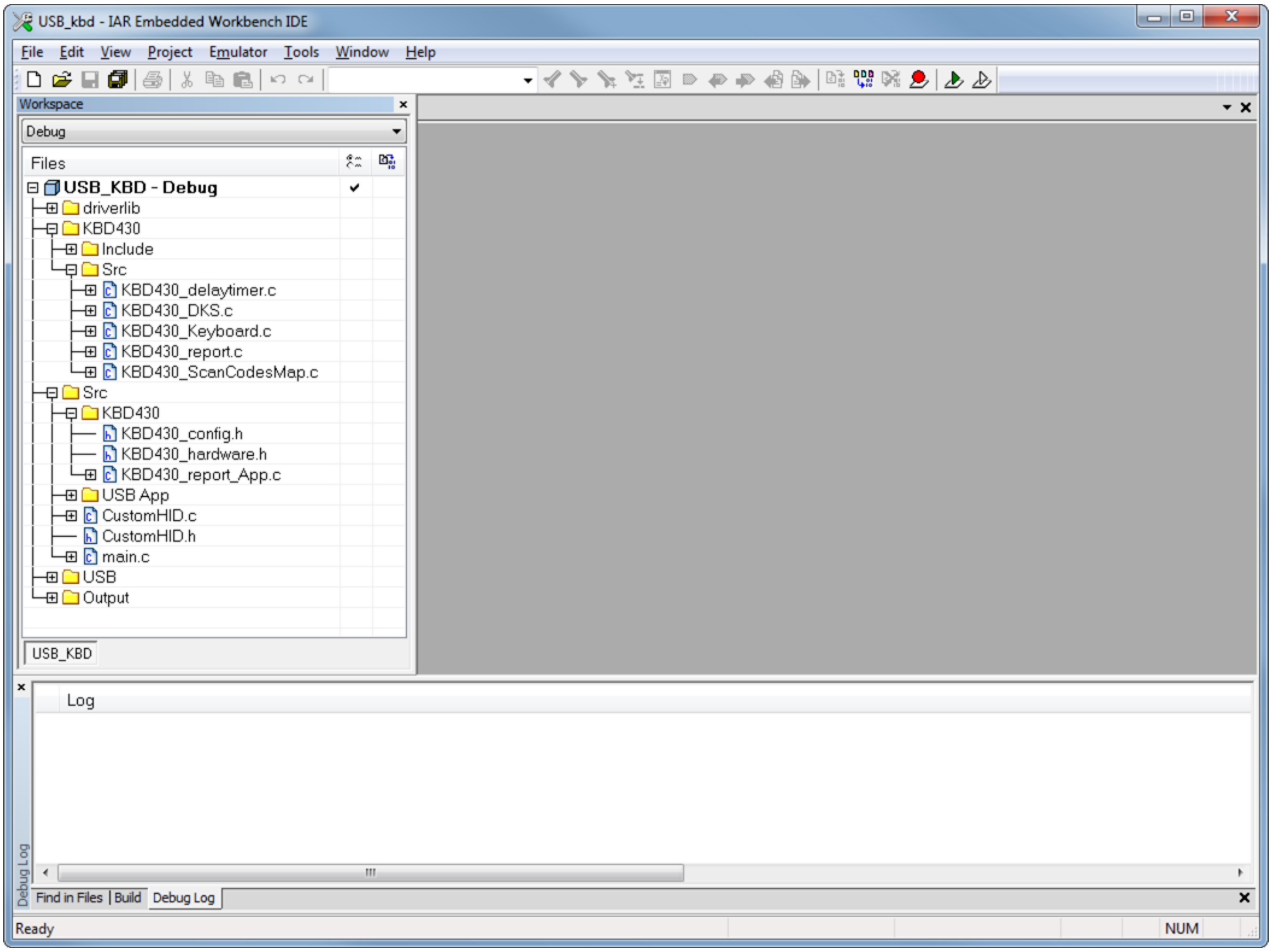


Figure 22. USBKBD Project in IAR

4. Connect board as described in Section 4.2
5. Download project to device (Ctrl+D, Menu → Project → Download and Debug, or)
6. Execute the program ( ) or close debugger and reset device.

5.2 Building Projects in CCS

1. Select a project to build:
   - *USBKBD*: USB using MSP430F5529
   - *I2CKBD*: I2C using MSP430F5529
   - *I2CKBD_G2xx4*: I2C using MSP430G2744
2. Import the corresponding project in CCS (Menu → Project → Import CCS Project).

```
KBD430_SW\Projects\<project>\CSS
```

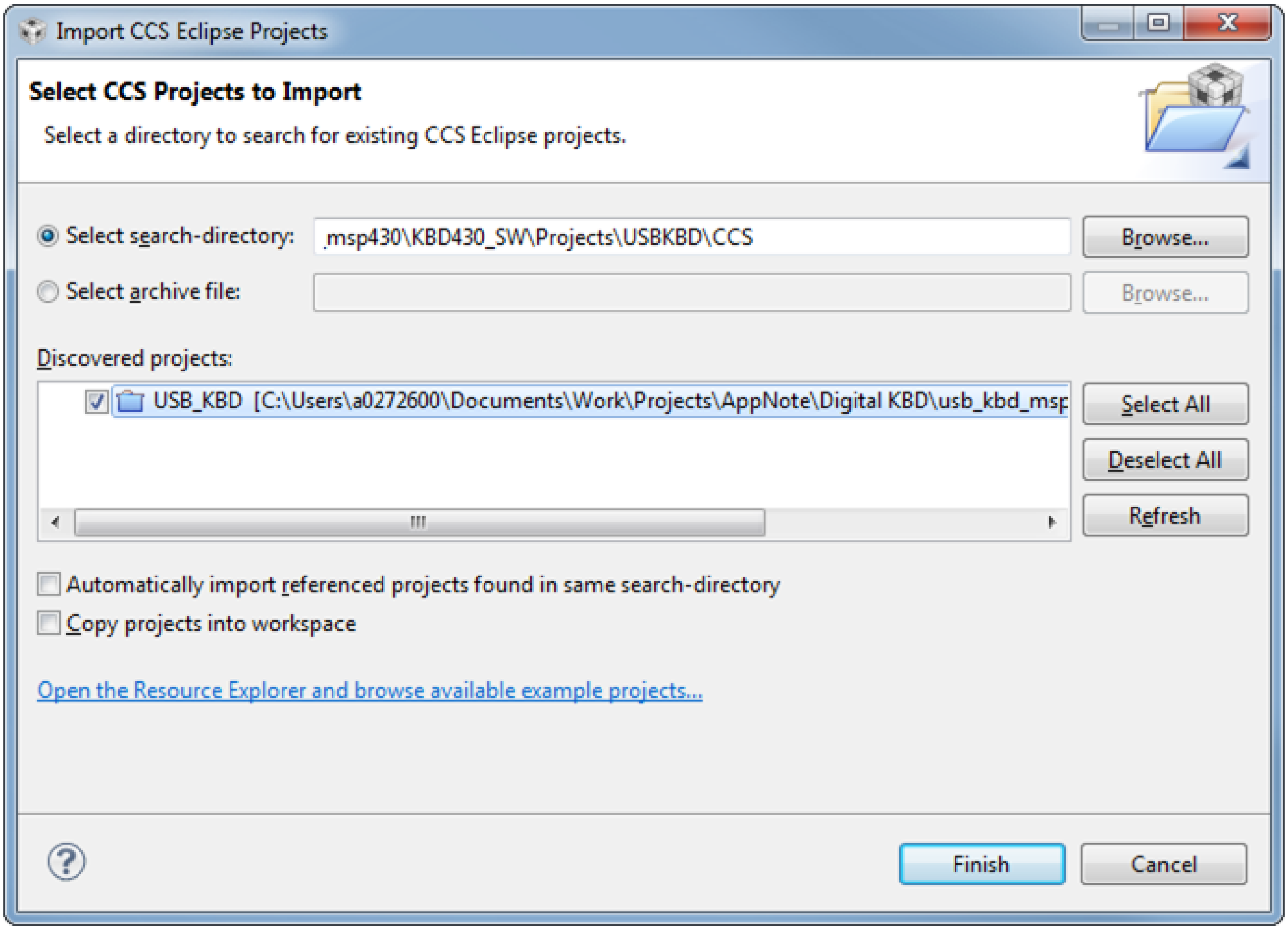


Figure 23. Importing USBKBD project in CCS

3. Build project (Ctrl+B, Menu → Project → Build All, or )

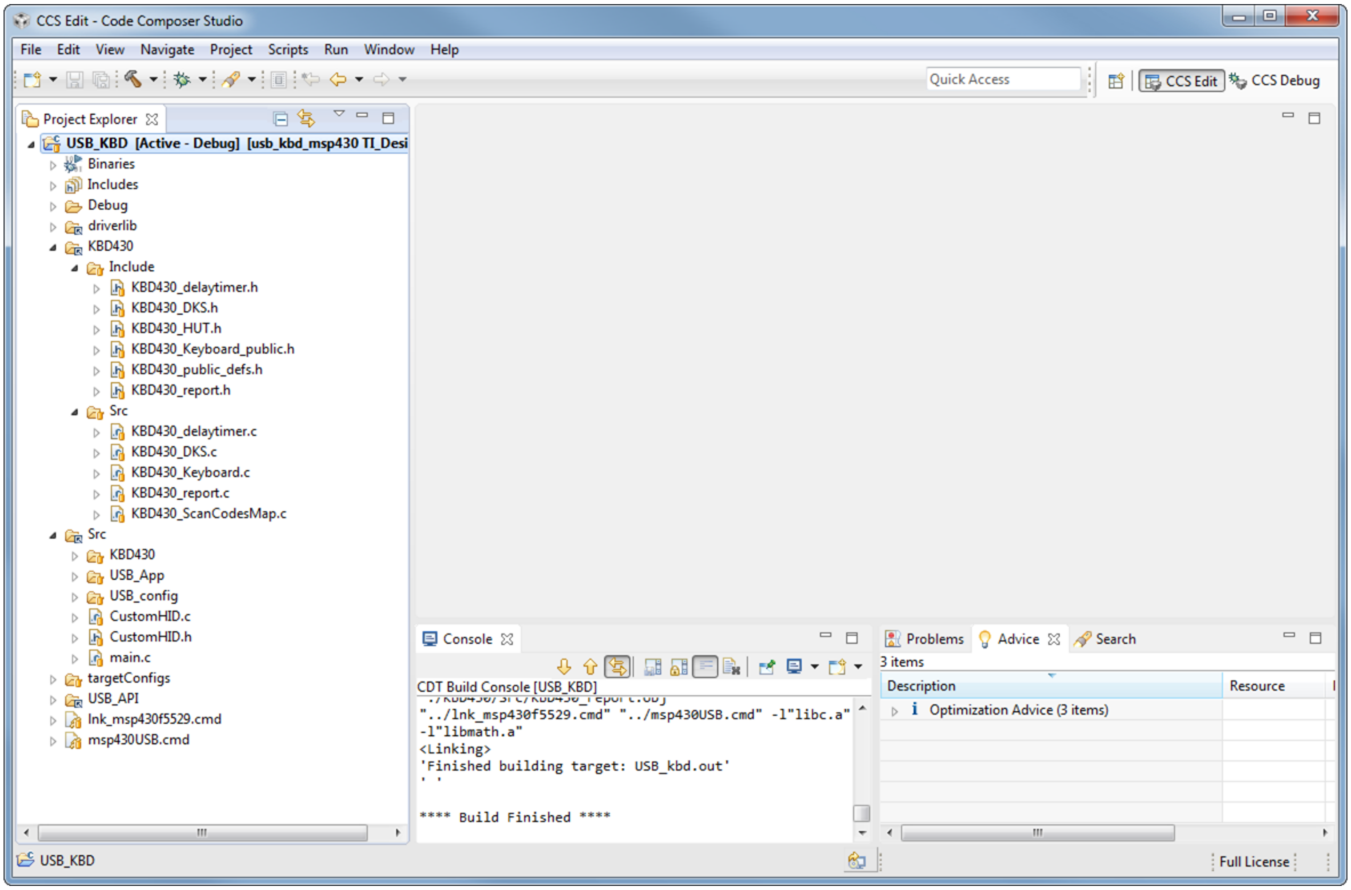


Figure 24. USBKBD Project in CCS

4. Connect board as described in Section 4.2
5. Download project to device (F11, Menu → Run → Debug, or )
6. Execute the program () or close debugger and reset device.

6 Customizing the Keyboard Controller

The software and hardware provided in this reference design provide an easy-to-use out-of-box experience for demos and testing, but it also provides a starting point for developers trying to implement their own keyboard controller in different applications.

The following sections describe some common customizations, but many more can be implemented by developers.

6.1 USB Interface Customizations

6.1.1 USB VID/PID

Developers can modify the VID/PID of the application in order to use their own

Instructions:

1. Modify Macros USB_VID and USB_PID in:
   `.\Projects\Projects\USBKBD\Src\USB_config\descriptors.h`

6.1.2 Crystal XT2

Use a different crystal for the design.

Instructions:

1. Modify USB_XT_FREQ_VALUE in:
   `.\Projects\Projects\USBKBD\Src\USB_config\descriptors.h`

6.1.3 USB Descriptors

USB interfaces can be modified to meet particular needs. This includes removing unwanted interfaces, or adding other interfaces such as CDC or MSC.

Instructions:

1. Modify the USB descriptors in the following files:
   `.\Projects\Projects\USBKBD\Src\USB_config\descriptors.c/h`
2. The MSP430 USB Developers Package (msp430usbdevpack) includes a USB Descriptor tool which can help creating descriptors for the USB API
3. Add/remove application code as necessary to support descriptors. Definitions USE_CONSUMER_REPORT, USE_WRC_REPORT, and USE_CUSTOM_HID can be used to disable application calls to these interfaces. These definitions are in:
   `.\Projects\Projects\USBKBD\Src\KBD430\KBD430_config.h`

6.1.4 Polling Interval

The USB interface defines the interface polling rate in the interface descriptors. This polling rate can be modified with a direct impact on the response time of the keyboard.

Instructions:

1. Modify the bInterval parameter (in ms) in the corresponding interface descriptor found in:
   `.\Projects\Projects\USBKBD\Src\USB_config\descriptors.c/h`

6.2 HID-I2C Interface Customizations

6.2.1 I2C Slave Address

Developers can modify the I2C slave address of the device.

Instructions:

1. Modify Macro USCIBx_ADDR in:

   `.\Projects\Projects\<I2CKBDproject>\Src\HIDI2C\hidi2c_settings.h`

6.2.2 I2C Peripheral/Pins

The hidi2c driver can be modified to use other USCI interfaces and pins.

Instructions:

1. Uncomment/comment USCIBx_XXXX macros in:

   `.\Projects\Projects\<I2CKBDproject>\Src\HIDI2C\hidi2c_settings.h`

2. Modify appropriately:

USCIBx	← Constant definition
USCIBx_ADDR	← Slave address used for this interface
USCIBx_GPIO_POUT	← Slave address used for this interface
USCIBx_GPIO_PDIR	← PxDIR register used for I2C_INT
USCIBx_GPIO_PIN	← PxIN register used for I2C_INT
USCIBx_PORT	← PxSEL register used to initialize I2C pins functionality
USCIBx_PINS	← Pins used for I2C (SDA/SCL)

6.2.3 HIDI2C Report Descriptors

The HID interface can be modified to meet particular needs of the developer. Reports can be modified, removed or added as needed.

Instructions:

1. Modify the HID descriptors in the following file:

   `.\Projects\Projects\<I2CKBDproject>\Src\HIDI2C\keyboard_descriptors.h`

2. Add/remove application code as necessary to support descriptors. Definitions USE_CONSUMER_REPORT and USE_WRC_REPORT can be used to disable application use of these reports. These definitions are in:

   `.\Projects\Projects\<I2CKBDproject>\Src\KBD430\KBD430_config.h`

6.3 Keyboard

6.3.1 Matrix Layout

The software can be easily adjusted to support different keyboard layouts.

Instructions:

1. Obtain the key matrix for your keyboard, similar to Figure 5.
2. Modify the USBKBD_scancodes_s table in the following file:

   ```
   .\KBD430\Src\KBD430_ScanCodesMap.c
   ```

   Each entry of this table corresponds to a key in each (column, row) in the following order:

(0,0), (0,1), (0,2), (0,3), (0,4), (0,5), (0,6), (0,7)
(1,0), (1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7)
...
(13,0), (13,1), (13,2), (13,3), (13,4), (13,5), (13,6), (13,7)
(14,0), (14,1), (14,2), (14,3), (14,4), (14,5), (14,6), (14,7)

   Codes are defined by USB.org HID Usage tables and they are defined in the following file:

   ```
   .\KBD430\Include\KBD430_HUT.h
   ```

3. hidUsageReserved must be used for unavailable keys. During keyboard initialization, the driver will check for unimplemented keys in order to detect “ghost” keys properly.
4. Additional entries can be added to this table for matrix arrays different from 15x8. Note that this will also require changes to the key scan driver which are described in Section 6.3.3.

6.3.2 Function Keys

The software included in this reference design has support for function keys defined in Table 1. The implementation of function keys can vary, but developers can customize these keys as needed and/or add new function keys to their implementations.

Instructions:

1. Make sure the Fn key is defined as hidUsageReservedFn in the USBKBD_scancodes_s table (check Section 6.3.1). The function CheckforFnKey() will detect when the Fn key is pressed and it will set a flag in order to handle this special case.
2. If required, modify the report descriptors to add new functions or customize existing ones (check Section 6.1.3 and Section 6.2.3for USB and I2C respectively). The report descriptors included in the software example include support for some keys defined in the Consumer Control Usage table and one key defined in the Wireless Radio Control usage table. More details of HID report descriptors can be found in HID Usage tables.
3. Customize the keyboard response for each Function key combination. GetFnKey() includes a switch-case statement with the implementation of all key combinations. This function is located in the following file:

   ```
   .\KBD430\Src\KBD430_Keyboard.c
   ```

   Note that in some cases, the keyboard combination can simply return a new standard key:

   ```
   case hidUsageF11:
      return hidUsageKeypadNumlock;          // Fn + F11 = Num Lock
   ```

   But in order cases, it will have special functionality as follows:

   ```
   case hidUsageF8:
      return CONSUMER_KEY(0x04);             // Fn + F8 = Audio Mute
   ```

   The macros CONSUMER_KEY and WRC_KEY are simply used to differentiate between normal keys and special ones.
4. If implementing a report different from Consumer Control and Wireless Radio Control, add the corresponding handlers to add and remove keys in UpdateHIDReport() and to send the report in KBD430_Report_Update(). Use the existing implementation, conditionally built using USE_CONSUMER_REPORT and USE_WRC_REPORT macros, as a base.

6.3.3 Keyboard Hardware Change

The software can be modified to support different keyboard connectors, or when porting to a different MSP430.

Instructions:

1. Define number of KSI pins (columns) and KSO pins (rows) in:

   `.\Projects\Projects\<I2CKBDproject>\Src\KBD430\KBD430_config.h`

   The number of keys resulting from multiplying KSIxKSO must be the same as the number of entries in USBKBD_scancodes_s table (check Section 6.3.1).

2. Define the KSI port (columns) in the following file:

   `.\Projects\<project>\Src\KBD430\KBD430_hardware.h`

   Note that this implementation uses a single 8-bit port to implement al columns. This allows for an easier and faster read of the whole column at the same time and easier handling of the ISR. The following definitions must be modified:

KSI_IN	← PxIN register for KSI
KSI_OUT	← PxOUT register for KSI
KSI_DIR	← PxDIR register for KSI
KSI_IES	← PxIES register for KSI
KSI_IFG	← PxIFG register for KSI
KSI_IE	← PxIE register for KSI
KSI_ALL	← Bits used for KSI
KSI_READ()	← Macro to read KSI port
KSI_VECTOR	← KSI interrupt vector

3. Define the KSO pins (rows) in the same file. KSO pins can be located anywhere in the microcontroller. The pins are defined as follows:

KSOx_POUT	← Address of PxOUT register for this KSO pin
KSOx_BIT	← Bit used by this KSO pin

   Note that the KSO pins are stored in the KSO_Pinmap array.

4. Define KSO_Px_ALL for each port used by KSO pins. This definition is optional but it can be used by macros which write to the whole ports in a faster way.
5. Modify KSO_PDIR_OFFSET if needed. The driver uses KSOx_POUT as the base address for each port. But it uses the KSO_PDIR_OFFSET to access the corresponding PxDIR register. For example, if the address of P1OUT is 0x0202 in MSP430F5529, and P1DIR is 0x0204, the value of KSO_PDIR_OFFSET will be 0x02.
6. Modify keyboard macros SET_KSO_INTERRUPT, SET_KSO_POLL and SET_KSO_IDLE. These macros set the state of the KSO pins in column-interrupt, polling or idle mode. The macros write directly to complete KSO ports in order to run faster. When possible, write to the 16-bit register (for example, PAOUT/PADIR/PAREN) instead of 8-bit registers (for example, P1OUT/P1DIR/P1REN).
7. If the number of KSO pins is different, add/remove entries in KSO_Pinmap, found in:

   `.\KBD430\Src\KBD430_DKS.c`

6.3.4 Scan Rate

The scan rate of the keyboard can be modified to reduce power consumption, or increase the response time.

Instructions:

1. Adjust DELAY_SCAN_CYCLES as required in the following file:

   `.\Projects\Projects\<project>\Src\KBD430\KBD430_config.h`

   By default, the keyboard controller waits 10ms between scans.

6.4 Application

6.4.1 MCU System Initialization

The software included in this reference design initializes basic MCU system peripherals as follows:

- Watchdog disabled
- MCLK=SMCLK = DCO = 8MHz
- Unused GPIOs = Output Low
- SVSL/SVML/SVMH = disabled, SVSH=full performance → MSP430F5529 only
- VCore=2 (USBKBD) or VCore=0 (I2CKBD) → MSP430F5529 only

Developers can modify these settings as required or when using a different hardware or MSP430 derivative.

Instructions:

1. Modify Init_Clock() to initialize MSP430 clocks. This function is located in:
   ```
   .\Projects\<project>\Src\main.c
   ```
2. In the same file, modify Init_Ports() to initialize MSP430 GPIOs.
   - KSI pins should be initialized as inputs.
   - KSO pins as output low.
   - LED pins as output low.
   - Unused pins should have an internal/external pull-up/down resistor or be configured as output.
3. Other basic initialization can be performed in main().

6.4.2 Keyboard LEDs

The application uses 3 LEDs for NumLock, CapsLock and ScrollLock. Developers can use different pins for these functions or simply remove the functionality.

Instructions:

1. The LED pins are defined in:
   ```
   .\Projects\<project>\Src\KBD430\KBD430_hardware.h
   ```
   The following definitions are available:

LED_PORT_W	← PxOUT register for LEDs
LEDNUM_1_1	← Pin used for Num Lock
LEDCAPS_1_6	← Pin used for Caps Lock
LEDSCROLL_1_7	← Pin used for Scroll Lock

2. The LEDs are handled in the application layer after getting a report from the HID interface. Modify the implementation in main() if needed.

6.4.3 USB Custom Interface

The *USBKBD* configuration includes an HID custom interface which can be used to exchange custom data with the USB host. The implementation shown in this reference design is very simplistic but it can be used as a base for further development.

Instructions:

1. Modify CustomHID_Parse() to parse and interpret data from USB Host. This function is declared in:

```
.\Projects\USBKBD\Src\CustomHID.c
```

The current implementation implements a simple switch-case statement handling the different commands from USB Host.

6.5 Firmware Updates

Section 5 explains the procedure required to program the device using the JTAG connector. While this option is useful during development, it can be inconvenient for field upgrades in many applications.

In addition to the JTAG interface, this reference design allows developers to implement BSL communication.

The MSP430 bootstrap loader (BSL) enables users to communicate with embedded memory in the MSP430 microcontroller during the prototyping phase, final production and in service. By using common interfaces such as USB, UART or I2C, BSL can be more convenient and easier to implement on the field.

Two BSL methods are implemented depending on the device, and they are explained in more detail in the following sections.

6.5.1 USB BSL in MSP430F5529

MSP430F5529 is shipped with a USB BSL which resides in Flash. This BSL can be forced in hardware using the PUR pin, however the development board included in this design doesn't have easy access to this pin, so a software method is used instead.

The procedure is explained in the following steps:

1. The USB BSL method can be used in the following projects:
   - *USBKBD*: USB using MSP430F5529
   - *I2CKBD*: I2C using MSP430F5529
2. The default configuration for these projects includes a definition ENABLE_SW_BSL which enables software BSL calls.
   If this definition is not enabled, the device can't force BSL mode.
3. Disconnect the device from the USB port
4. Press 'm', 's', and 'p' keys at the same time
5. With the keys pressed, connect to USB port.
   This will force a BOR, the MSP430 will reset and check for the 'm','s',and 'p' keys. If the keys are pressed, the device jumps to BSL; if not, the device will execute the application.
6. The 3 LEDs will blink at the same time to indicate entry to BSL mode
7. The 3 keys can be released at this point
8. A host application such as "MSP430 USB Firmware Upgrade Example", available here, can be used to update the firmware:

NOTE: The new firmware should enable ENABLE_SW_BSL in order to be able to update again.

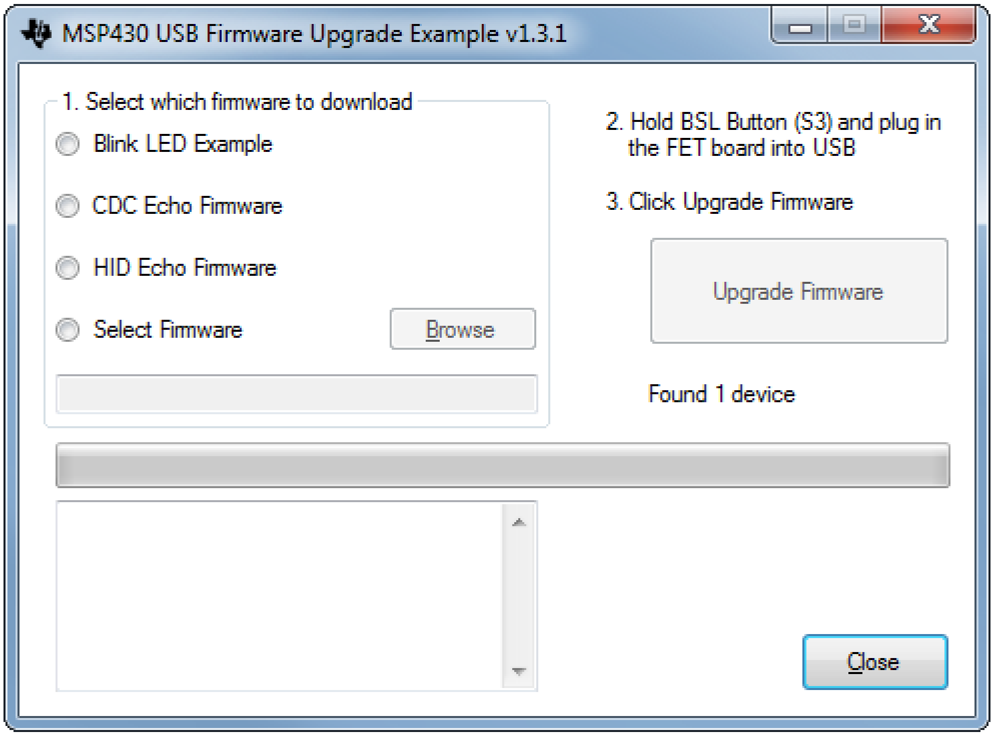


Figure 25. USB Firmware Upgrade Tool

The USB BSL is explained in more detail in SLAU319 and SLAA452.

Thanks to the flexibility of BSL, developers can implement customizations such as:

- Use a different software invocation sequence (check a pin on reset, use different keys, etc). The implementation of the software call to BSL is included in main.c and can be used as a guide.
- Enable hardware entry sequence using PUR pin (i.e. using push button during reset) The procedure and schematics are explained in more detail in SLAA452.
- Implement BSL using other interfaces such as I2C or UART.
  BSL resides in Flash in the MSP430F5xx family, allowing for these and more customizations. For more details, please refer to SLAA450.

6.5.2 UART BSL in MSP430F2744

MSP430G2744 includes a ROM BSL supporting UART and using the following pins:

Table 15. BSL pins in MSP430G2744

BSL Fuction	Pin	Usage
Data Transmit	P1.1	KSI1
Data Receive	P2.2	GPIO3_LED2

It’s important to remark that the development board doesn’t have special provisions to access the UART BSL pins easily:

- P2.2 can be accessed in Jumper JP4 or connector J5,
- P1.1 is used as a keyboard input and as such, it’s only available in surface mount connector J3 or resistor R11.

This BSL is usually invoked in hardware using the TEST and RESET using the entry sequence described in SLAU319; however this reference design includes a software invocation sequence described in the following steps:

1. The UART BSL method can be used in the followuing projects
   - *I2CKBD_G2xx4*: I2C using MSP430G2744
2. The default configuration for these projects includes a definition ENABLE_SW_BSL which enables software BSL calls. If this definition is not enabled, the device can't force BSL mode.
3. Disconnect the device from the USB port or power off device.
4. Press 'm', 's', and 'p' keys at the same time
5. With the keys pressed, connect to USB port or power up the device.
   This will force a BOR, the MSP430 will reset and check for the 'm','s',and 'p' keys. If the keys are pressed, the device jumps to BSL; if not, the device will execute the application.
6. The 3 LEDs will blink at the same time to indicate entry to BSL mode
7. The 3 keys can be released at this point
8. A host application such as BSLDEMO -included in SLAU319- and hardware such as MSP430-BSL Rocket can be used to update the firmware.

NOTE: The new firmware should enable ENABLE_SW_BSL in order to be able to update again.

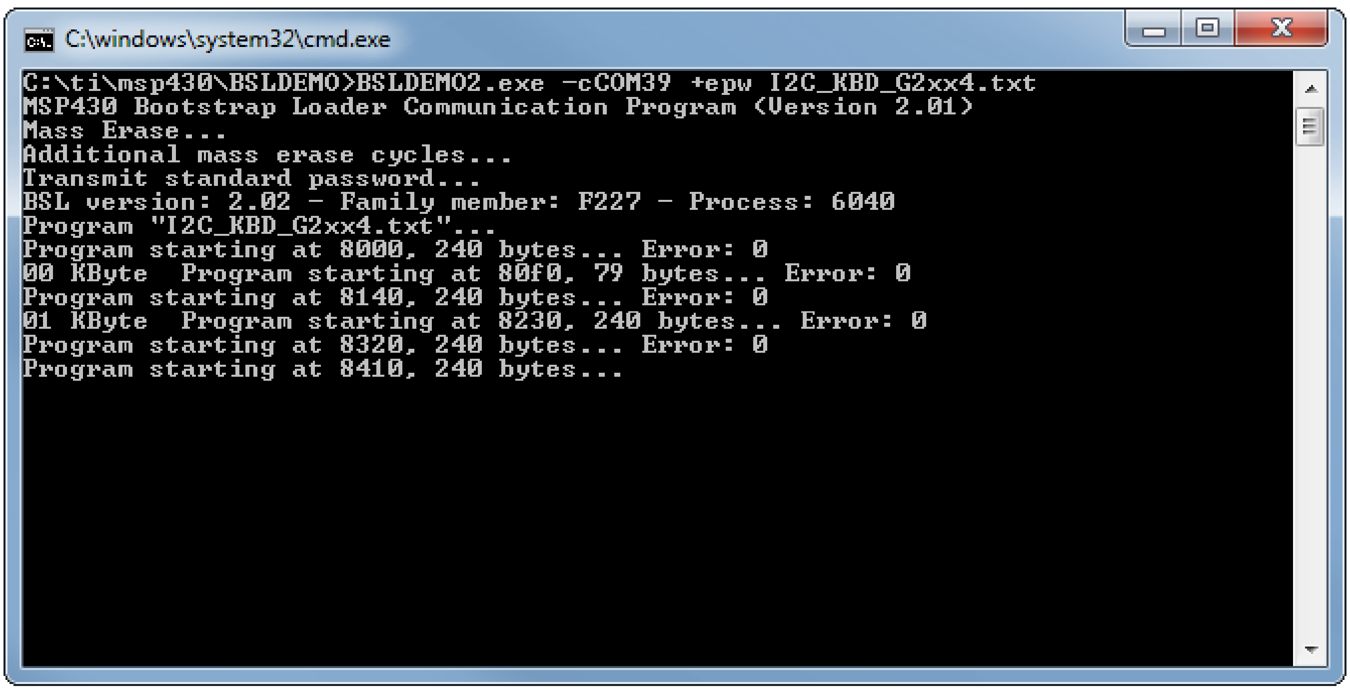


Figure 26. BSLDEMO Tool

The UART BSL is explained in more detail in SLAU319.

Developers can implement customizations such as:

- Use a different software invocation sequence (check a pin on reset, use different keys, etc).
  The implementation of the software call to BSL is included in main.c and can be used as a guide.
- Make the BSL pins easily available.
  As mentioned previously, the development board doesn't have special provisions to access BSL pins. Developers can make a special connector for BSL pins and reserve these pins for this purpose. Additionally, this connector can include the TEST and RESET pins used to force the hardware entry sequence.

- The ROM BSL is fixed to UART, but developers can implement a bootloader using other interfaces. MSPBoot included in SLAA600 shows the implementation of a bootloader which resides in main flash and supports UART, I2C and SPI.

7 Test Data

7.1 Test Setup

The board should be connected following the guidelines described in Section 4.

To measure power consumption:

- Disconnect JP2 and connect an ammeter in series

7.2 Response Time

The response time for keyboards is approximately 5ms to 50ms. While this depends on different factors such as the mechanical implementation of the keyboard, communication bus load, etc., by using this reference design, developers have more flexibility to customize the application according to their needs. Whether response time, price or power consumption is the most important requirement, parameters such as debounce time, USB polling interval, and microcontroller internal frequency can be adjusted to meet particular requirements.

One important factor affecting the response time is the scan time, which defines the time required to scan all keys. While a key press is detected in a few cycles in column-interrupt mode, the algorithm to recognize the particular pressed key, debounce it, discard "ghost" keys, etc. can take more cycles.

The following measurements were observed on bench tests:

	Time	Cycles
DKS Scan	~182us@8MHz	~1450 cycles
DKS Process (1st key)	~339us@8MHz	~2715 cycles
DKS Process (additional keys)	~108us@8MHz	~860 cycles

7.3 Power Consumption

The expected power profile for the *I2CKBD* configuration is shown in Figure 27:

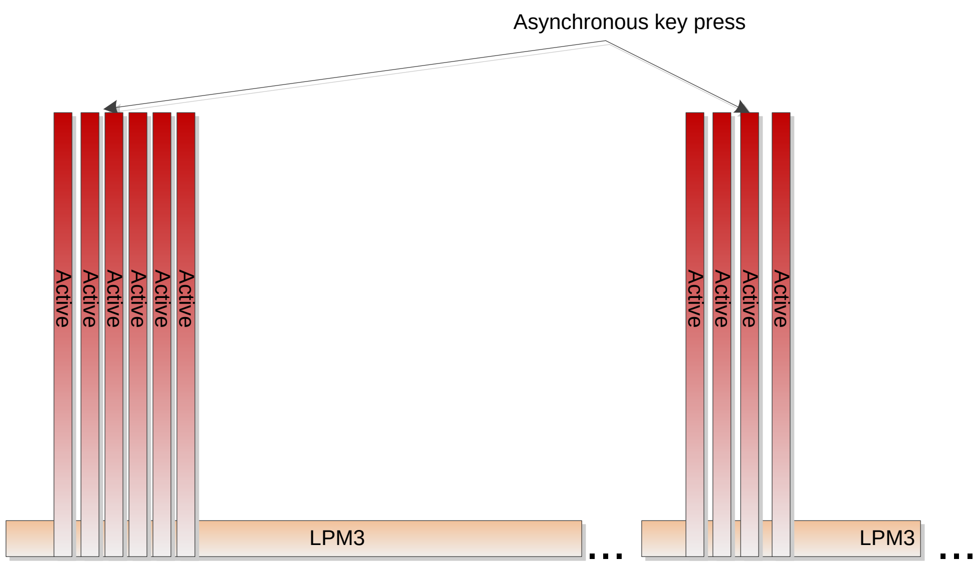


Figure 27. Expected Power Profile for I2CKBD and I2CKBD_G2xx4 Configurations

The following profile was observed on bench tests:

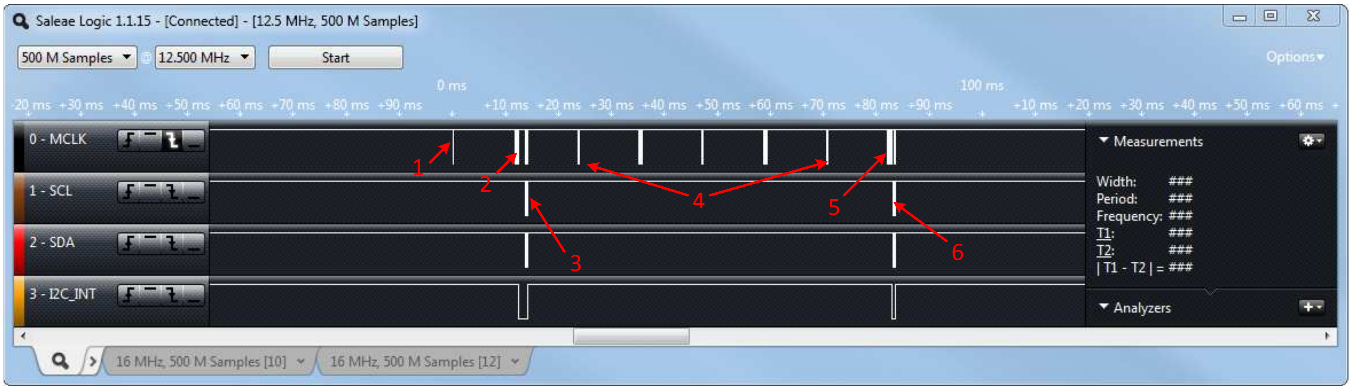


Figure 28. Power Profile Observed on I2CKBD and I2CKBD_G2xx4

The labels from Figure 28 show the different steps of the process:

1. Device wakes-up from key detection and performs first scan
2. Second scan is performed and key is processed
3. Key press is reported to host
4. Scans performed in polling mode checking for new keys and waiting for key release
5. Scan detects key release
6. Key release is reported to host

The power consumption measured for I2C configurations was:

Table 16. Power Consumption for I2C Examples

Device	Mode	Current	Power
MSP430G2744	Active MCLK=SMCLK=DCO= 8MHz ACLK = VLO	3.2mA	10.56mW
	LPM3 ACLK=VLO	0.8uA	2.64uW
MSP430F5529	Active FLLref=REFO=32.768Khz MCLK=SMCLK=DCO=8MHz PMMCOREV=0 ACLK = REFO SVSL/SVML/SVMH=Off SVSH=Full Performance	2.154mA	7.1082mW
	LPM3 ACLK=REFO SVSL/SVML/SVMH=Off SVSH=Full Performance	5.91uA	19.503uW

The expected power profile for the *USBKBD* configuration is shown in Figure 27:

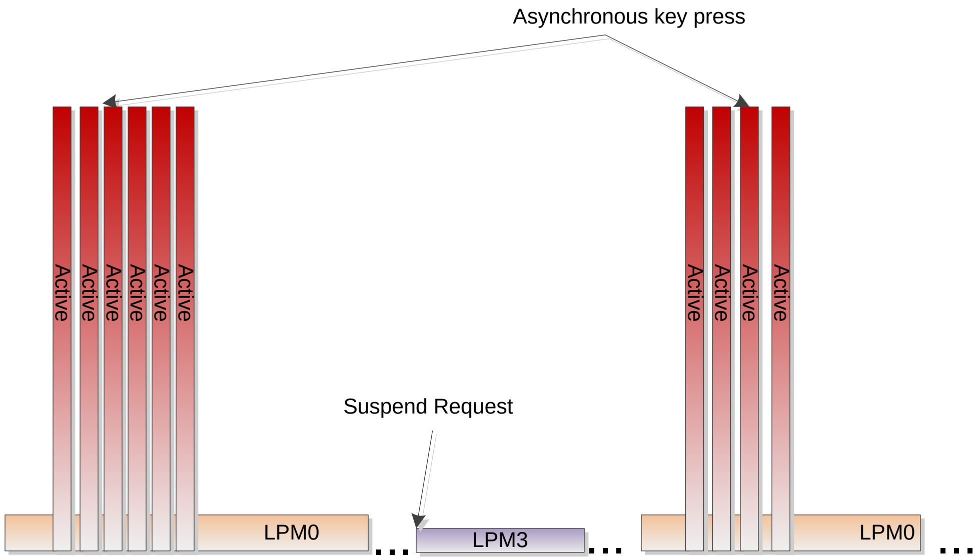


Figure 29. Expected Power Profile for USBKBD Configuration

The following profile was observed on bench tests:

Figure 30. Power Profile Observed on USBKBD

The labels from Figure 30 show the different steps of the process:

1. Device wakes-up from key detection and performs first scan
2. Second scan is performed, key is processed and sent to host
3. Scans performed in polling mode checking for new keys and waiting for key release
4. Scan detects key release and key is reported to host
5. Device detects a request to go to Suspend mode and goes to LPM3
6. Keyboard controller detects a key event and requests a USB remote wake-up

The power consumption measured for USB configuration was:

Table 17. Power Consumption for USB Example

Device	Mode	Current	Power
MSP430F5529	Active FLLref=REFO=32.768Khz MCLK=SMCLK=DCO=8MHz PMMCOREV=2 ACLK = REFO SVSL/SVML/SVMH=Off SVSH=Full Performance XT2 = 4MHz USB, PLL = On	3.3mA	10.89mW
	LPM0 FLLref=REFO=32.768Khz SMCLK=DCO=8MHz PMMCOREV=2 ACLK = REFO SVSL/SVML/SVMH=Off SVSH=Full Performance XT2 = 4MHz USB, PLL = On	977uA	3.224mW
	LPM3 ACLK=REFO SVSL/SVML/SVMH=Off SVSH=Full Performance XT2= Off USB, PLL = Off	6.22uA	20.52uA

7.4 Memory Footprint

The following memory footprint was obtained using IAR for MSP430 6.10.2 using optimization level "High-Balanced":

Table 18. Memory Footprint

	USBKBD	I2CKBD	I2CKBD_G2xx4
Code	10,926B	9,348B	7,528B
KBD430	2,252B	2,228B	2,098B
HIDUSB	5,566B	-	-
HIDI2C	-	4,460B	4,532B
Constants	1,486B	323B	323B
ScanCodeMap	124B	124B	124B
Descriptors	1,142B	139B	139B
Data	492B	638B	638B
Stack	160B	320B	320B
Heap	-	200B	200B

8 Design Files Schematics

To download the Schematics for each board, see the design files at http://www.ti.com/tool/DESIGNNUMBER.

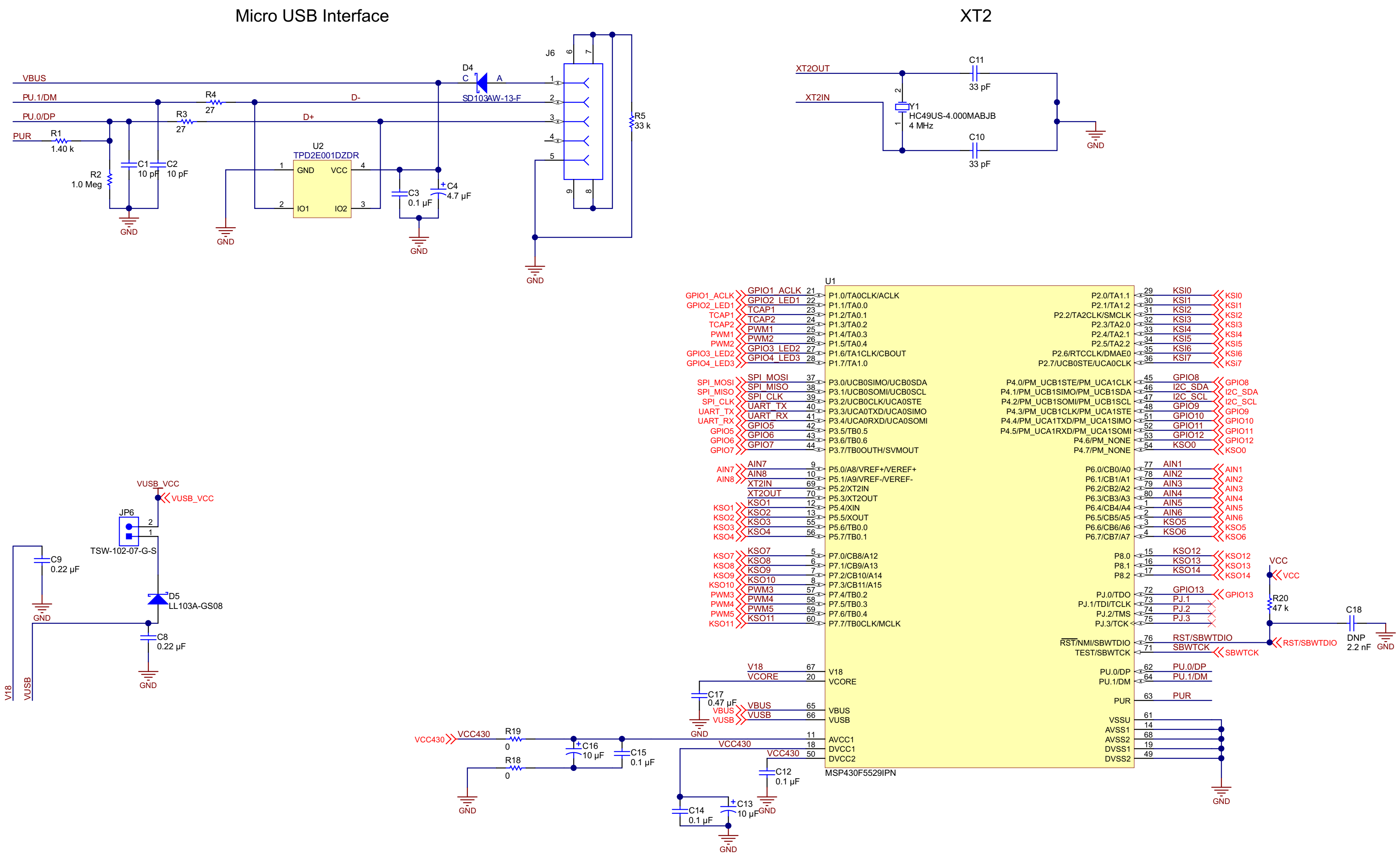


Figure 31. Schematics - Page 1

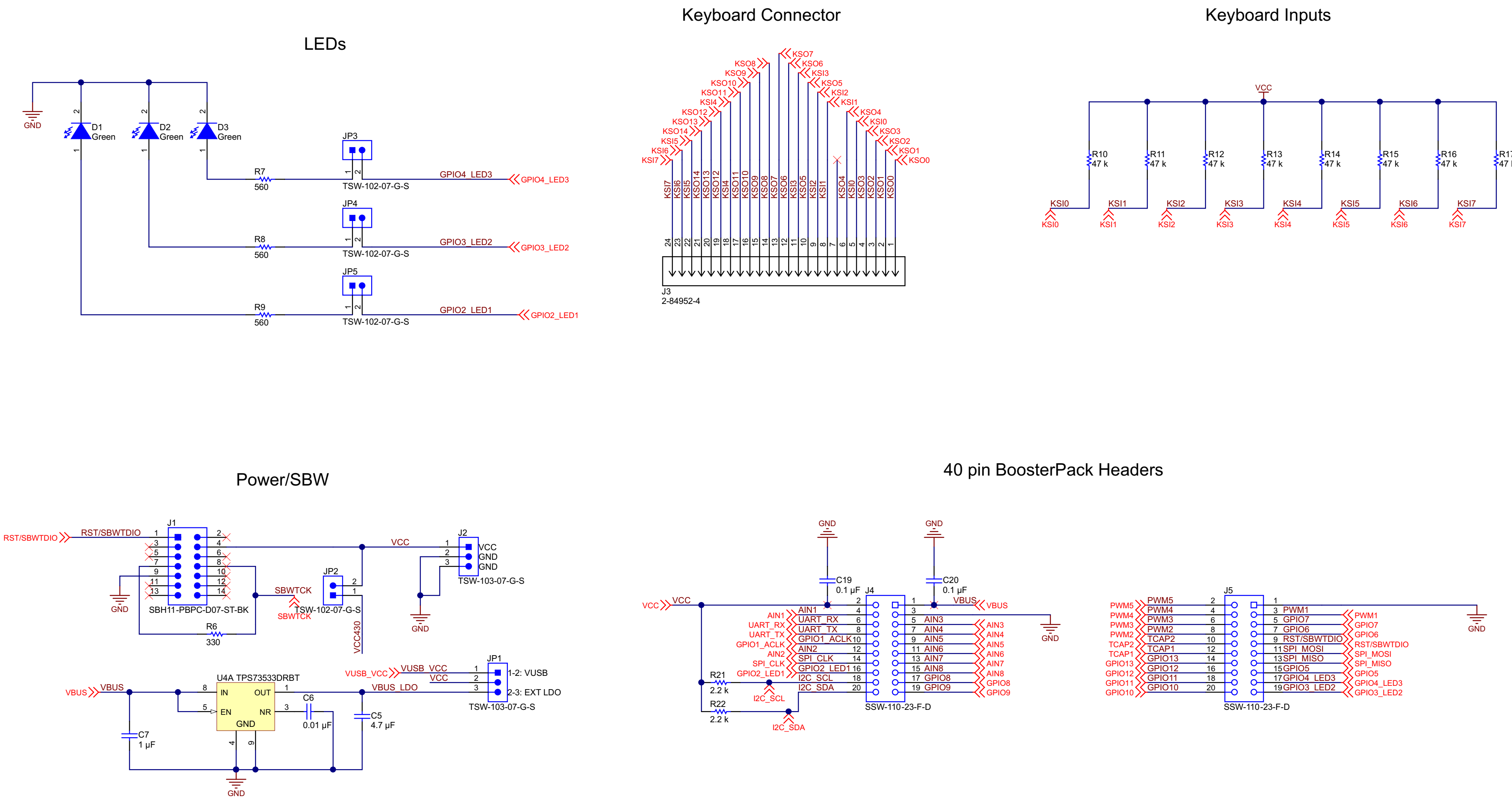


Figure 32. Schematics - Page 2

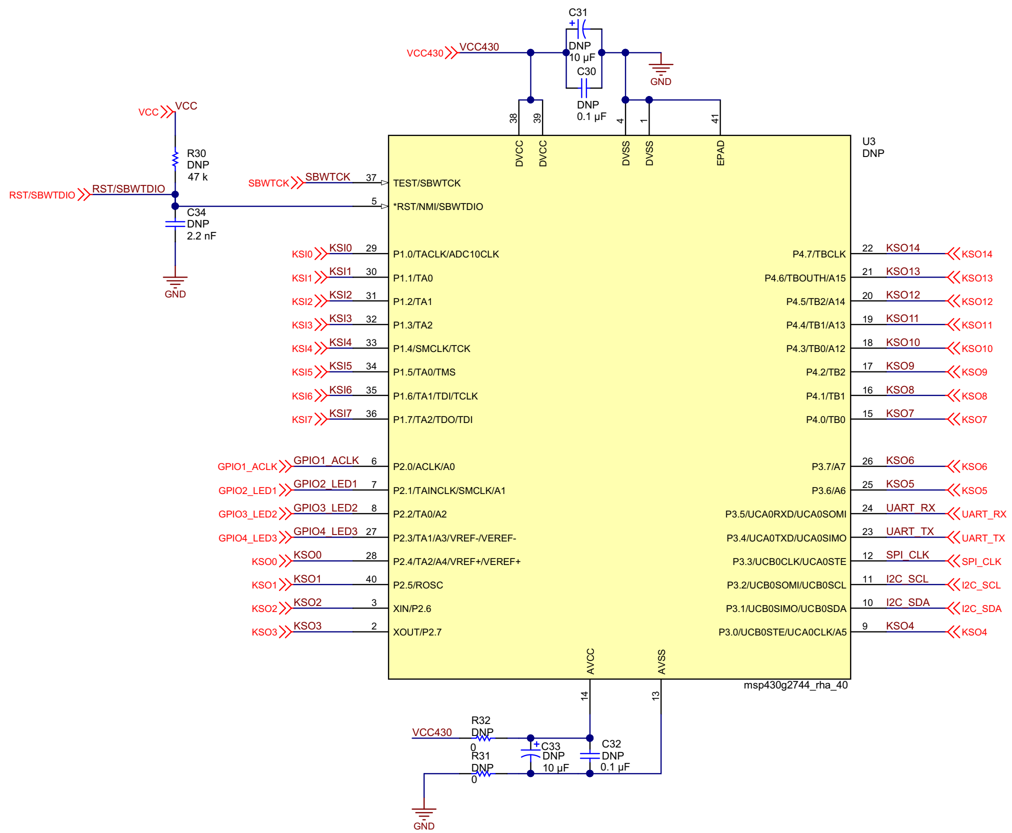


Figure 33. Schematics - Page 3

8.1 Bill of Materials

To download the Bill of Materials for each board, see the design files at http://www.ti.com/tool/DESIGNNUMBER.

Table 19. KBD430_BOM_F5529

Item	Qty	Reference	Value	Part Description	Manufacturer	Manufacturer Part Number	Alternate Part	PCB Footprint	Note
1	2	C1, C2	10pF	CAP, CERM, 10pF, 50V, +/-5%, C0G/NP0, 0603	Kemet	C0603C100J5GAC TU		0603	
2	1	C3	0.1uF	CAP, CERM, 0.1uF, 50V, +/-10%, X7R, 0603	Kemet	C0603C104K5RAC TU		0603	
3	1	C4	4.7µF	CAP, TA, 4.7uF, 10V, +/-10%, 5 ohm, SMD	Vishay-Sprague	293D475X9010A2T E3		3216-18	
4	1	C5	4.7uF	CAP, CERM, 4.7uF, 10V, +/-10%, X5R, 0603	Kemet	C0603C475K8PAC TU		0603	
5	1	C6	0.01uF	CAP, CERM, 0.01uF, 50V, +/-10%, X7R, 0603	MuRata	GRM188R71H103K A01D		0603	
6	1	C7	1uF	CAP, CERM, 1uF, 16V, +/-10%, X5R, 0603	Kemet	C0603C105K4PAC TU		0603	
7	2	C8, C9	0.22uF	CAP, CERM, 0.22uF, 16V, +/-10%, X7R, 0603	TDK	C1608X7R1C224K 080AC		0603	
8	2	C10, C11	33pF	CAP, CERM, 33pF, 50V, +/-5%, C0G/NP0, 0603	TDK	C1608C0G1H330J 080AA		0603	
9	3	C12, C14, C15	0.1uF	CAP, CERM, 0.1uF, 16V, +/-10%, X7R, 0603	Kemet	C0603C104K4RAC TU		0603	
10	2	C13, C16	10uF	CAP, TA, 10uF, 10V, +/-20%, 3.4 ohm, SMD	Vishay-Sprague	293D106X0010A2T E3		3216-18	
11	1	C17	0.47uF	CAP, CERM, 0.47uF, 10V, +/-10%, X5R, 0603	Kemet	C0603C474K8PAC TU		0603	
12	0	C18, C34	2200pF	CAP, CERM, 2200pF, 100V, +/-5%, X7R, 0603	AVX	06031C222JAT2A		0603	DNP_F5529
13	2	C19, C20	0.1uF	CAP, CERM, 0.1uF, 16V, +/-5%, X7R, 0603	AVX	0603YC104JAT2A		0603	
14	0	C30, C32	0.1uF	CAP, CERM, 0.1uF, 16V, +/-10%, X7R, 0603	Kemet	C0603C104K4RAC TU		0603	DNP_F5529
15	0	C31, C33	10uF	CAP, TA, 10uF, 10V, +/-20%, 3.4 ohm, SMD	Vishay-Sprague	293D106X0010A2T E3		3216-18	DNP_F5529
16	3	D1, D2, D3	Green	LED, Green, SMD	Lite-On	LTST-C171GKT		LED_LTST-C171	
17	1	D4	SD103AW-13-F	DIODE, SCHOTTKY, 0.35A, 40V, SOD-123	DIODES INC	SD103AW-13-F		D_SOD123	
18	1	D5	LL103A-GS08	Diode, Schottky, 40V, 0.2A, 3.7x1.6x1.6mm	Vishay-Semiconductor	LL103A-GS08		SOD-80	
19	1	J1	Connector	Header (shrouded), 100 mil, 7x2, Gold, TH	Sullins Connector Solutions	SBH11-PBPC-D07-ST-BK		CONN_SBH11-PBPC-D07-ST-BK	
20	2	J2, JP1	Connector	Header, 100mil, 3x1, Gold, TH	Samtec	TSW-103-07-G-S		TSW-103-07-G-S	

Table 19. KBD430_BOM_F5529 (continued)

21	1	J3	Keyboard connector	CONN FPC 24POS 1MM RT ANG SMD	TE Connectivity	2-84952-4		CON24_SMT-RA_FCC	
22	2	J4, J5	Connector	Connector, Receptacle, 100mil, 10x2, Gold plated, TH	Samtec	SSW-110-23-F-D		CONN_SSW-110-23-F-D	
23	1	J6	USB Connector	CONNECTOR, MICRO-USB-AB, RECEPTACLE, RIGHT ANGLE, 5-PIN	HIROSE	ZX62D-AB-5P8		USB-ZX62D-AB-5P8_REVISED	
24	5	JP2, JP3, JP4, JP5, JP6	Jumper	Header, 100mil, 2x1, Gold, TH	Samtec	TSW-102-07-G-S		TSW-102-07-G-S	
25	1	R1	1.40k	RES, 1.40k ohm, 1%, 0.1W, 0603	Vishay-Dale	CRCW06031K40FKEA		0603	
26	1	R2	1M	RES, 1.0Meg ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW06031M00JNEA		0603	
27	2	R3, R4	27	RES, 27 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060327R0JNEA		0603	
28	1	R5	33k	RES, 33k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060333K0JNEA		0603	
29	1	R6	330	RES, 330 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW0603330RJNEA		0603	
30	3	R7, R8, R9	560	RES, 560 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW0603560RJNEA		0603	
31	9	R10, R11, R12, R13, R14, R15, R16, R17, R20	47k	RES, 47k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060347K0JNEA		0603	
32	2	R18, R19	0	RES, 0 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW06030000Z0EA		0603	
33	2	R21, R22	2.2k	RES, 2.2k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW06032K20JNEA		0603	
34	0	R30	47K	RES, 47k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060347K0JNEA		0603	DNP_F5529
35	0	R31, R32	0	RES, 0 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW06030000Z0EA		0603	DNP_F5529
36	1	U1	MSP430F5529	Mixed Signal MicroController, PN0080A	Texas Instruments	MSP430F5529IPN		PN0080A_L	
37	1	U2	TPD2E001DZDR	IC, LOW-CAPACITANCE 2-Chan ±15-kV ESD-PROTECTION ARRAY, SOP-4	TEXAS INSTRUMENTS	TPD2E001DZDR		DZD	
38	0	U3	MSP430G2744	MSP430G2744, VQFN40	Texas Instruments	MSP430G2744IRHA40R		QFN50P600X600X100-40N	DNP_F5529
39	1	U4	TPS73533	Single Output High PSRR LDO, 500 mA, Fixed 3.3 V Output, 2.7 to 6.5 V Input, with Low IQ, 8-pin SON (DRB), -40 to 125 degC, Green (RoHS & no Sb/Br)	Texas Instruments	TPS73533DRBT		DRB8_1P75X1P5	
40	1	Y1	XTAL4M	Crystal, 4MHz, 30pF, SMD	Auris-GmbH	Q- 4,000000M-HC49USSMD-F-30-30-D-16-TR	HCM49-4.000MABJT	Auris_HC49USSMD	

Table 20. KBD430_BOM_G2744

Item	Qty	Reference	Value	Part Description	Manufacturer	Manufacturer Part Number	Alternate Part	PCB Footprint	Note
1	2	C1, C2	10pF	CAP, CERM, 10pF, 50V, +/-5%, C0G/NP0, 0603	Kemet	C0603C100J5GACTU		0603	
2	1	C3	0.1uF	CAP, CERM, 0.1uF, 50V, +/-10%, X7R, 0603	Kemet	C0603C104K5RACTU		0603	
3	1	C4	4.7µF	CAP, TA, 4.7uF, 10V, +/-10%, 5 ohm, SMD	Vishay-Sprague	293D475X9010A2TE3		3216-18	
4	1	C5	4.7uF	CAP, CERM, 4.7uF, 10V, +/-10%, X5R, 0603	Kemet	C0603C475K8PACTU		0603	
5	1	C6	0.01uF	CAP, CERM, 0.01uF, 50V, +/-10%, X7R, 0603	MuRata	GRM188R71H103KA01D		0603	
6	1	C7	1uF	CAP, CERM, 1uF, 16V, +/-10%, X5R, 0603	Kemet	C0603C105K4PACTU		0603	
7	0	C8, C9	0.22uF	CAP, CERM, 0.22uF, 16V, +/-10%, X7R, 0603	TDK	C1608X7R1C224K080AC		0603	DNP_G2744
8	0	C10, C11	33pF	CAP, CERM, 33pF, 50V, +/-5%, C0G/NP0, 0603	TDK	C1608C0G1H330J080AA		0603	DNP_G2744
9	0	C12, C14, C15	0.1uF	CAP, CERM, 0.1uF, 16V, +/-10%, X7R, 0603	Kemet	C0603C104K4RACTU		0603	DNP_G2744
10	0	C13, C16	10uF	CAP, TA, 10uF, 10V, +/-20%, 3.4 ohm, SMD	Vishay-Sprague	293D106X0010A2TE3		3216-18	DNP_G2744
11	0	C17	0.47uF	CAP, CERM, 0.47uF, 10V, +/-10%, X5R, 0603	Kemet	C0603C474K8PACTU		0603	DNP_G2744
12	0	C18, C34	2200pF	CAP, CERM, 2200pF, 100V, +/-5%, X7R, 0603	AVX	06031C222JAT2A		0603	DNP_G2744
13	2	C19, C20	0.1uF	CAP, CERM, 0.1uF, 16V, +/-5%, X7R, 0603	AVX	0603YC104JAT2A		0603	
14	2	C30, C32	0.1uF	CAP, CERM, 0.1uF, 16V, +/-10%, X7R, 0603	Kemet	C0603C104K4RACTU		0603	
15	2	C31, C33	10uF	CAP, TA, 10uF, 10V, +/-20%, 3.4 ohm, SMD	Vishay-Sprague	293D106X0010A2TE3		3216-18	
16	3	D1, D2, D3	Green	LED, Green, SMD	Lite-On	LTST-C171GKT		LED_LTST-C171	
17	1	D4	SD103AW-13-F	DIODE, SCHOTTKY, 0.35A, 40V, SOD-123	DIODES INC	SD103AW-13-F		D_SOD123	
18	0	D5	LL103A-GS08	Diode, Schottky, 40V, 0.2A, 3.7x1.6x1.6mm	Vishay-Semiconductor	LL103A-GS08		SOD-80	DNP_G2744
19	1	J1	Connector	Header (shrouded), 100 mil, 7x2, Gold, TH	Sullins Connector Solutions	SBH11-PBPC-D07-ST-BK		CONN_SBH11-PBPC-D07-ST-BK	
20	2	J2, JP1	Connector	Header, 100mil, 3x1, Gold, TH	Samtec	TSW-103-07-G-S		TSW-103-07-G-S	
21	1	J3	Keyboard connector	CONN FPC 24POS 1MM RT ANG SMD	TE Connectivity	2-84952-4		CON24_SMT-RA_FCC	
22	2	J4, J5	Connector	Connector, Receptacle, 100mil, 10x2, Gold plated, TH	Samtec	SSW-110-23-F-D		CONN_SSW-110-23-F-D	
23	1	J6	USB Connector	CONNECTOR, MICRO-USB-AB, RECEPTACLE, RIGHT ANGLE, 5-PIN	HIROSE	ZX62D-AB-5P8		USB-ZX62D-AB-5P8_REVISED	
24	4	JP2, JP3, JP4, JP5	Jumper	Header, 100mil, 2x1, Gold, TH	Samtec	TSW-102-07-G-S		TSW-102-07-G-S	
25	0	JP6	Jumper	Header, 100mil, 2x1, Gold, TH	Samtec	TSW-102-07-G-S		TSW-102-07-G-S	DNP_G2744
26	1	R1	1.40k	RES, 1.40k ohm, 1%, 0.1W, 0603	Vishay-Dale	CRCW06031K40FKEA		0603	
27	1	R2	1M	RES, 1.0Meg ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW06031M00JNEA		0603	
28	2	R3, R4	27	RES, 27 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060327R0JNEA		0603	
29	1	R5	33k	RES, 33k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060333K0JNEA		0603	

Table 20. KBD430_BOM_G2744 (continued)

30	1	R6	330	RES, 330 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060330RJNEA		0603	
31	3	R7, R8, R9	560	RES, 560 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW0603560RJNEA		0603	
32	9	R10, R11, R12, R13, R14, R15, R16, R17	47k	RES, 47k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060347K0JNEA		0603	
33	0	R20	47k	RES, 47k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060347K0JNEA		0603	DNP_G2744
34	0	R18, R19	0	RES, 0 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW06030000Z0EA		0603	DNP_G2744
35	2	R21, R22	2.2k	RES, 2.2k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW06032K20JNEA		0603	
36	1	R30	47K	RES, 47k ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW060347K0JNEA		0603	
37	2	R31, R32	0	RES, 0 ohm, 5%, 0.1W, 0603	Vishay-Dale	CRCW06030000Z0EA		0603	
38	0	U1	MSP430F5529	Mixed Signal MicroController, PN0080A	Texas Instruments	MSP430F5529IPN		PN0080A_L	DNP_G2744
39	1	U2	TPD2E001DZDR	IC, LOW-CAPACITANCE 2-Chan ±15-kV ESD-PROTECTION ARRAY, SOP-4	TEXAS INSTRUMENTS	TPD2E001DZDR		DZD	
40	1	U3	MSP430G2744	MSP430G2744, VQFN40	Texas Instruments	MSP430G2744IRHA40R		QFN50P600X600X100-40N	
41	1	U4	TPS73533	Single Output High PSRR LDO, 500 mA, Fixed 3.3 V Output, 2.7 to 6.5 V Input, with Low IQ, 8-pin SON (DRB), -40 to 125 degC, Green (RoHS & no Sb/Br)	Texas Instruments	TPS73533DRBT		DRB8_1P75X1P5	
42	0	Y1	XTAL4M	Crystal, 4MHz, 30pF, SMD	Auris-GmbH	Q-4,000000M-HC49USSMD-F-30-30-D-16-TR	HCM49-4.000MABJT	Auris_HC49USSMD	DNP_G2744

8.2 PCB Layout

To download the Layout Prints for each board, see the design files at http://www.ti.com/tool/DESIGNNUMBER.

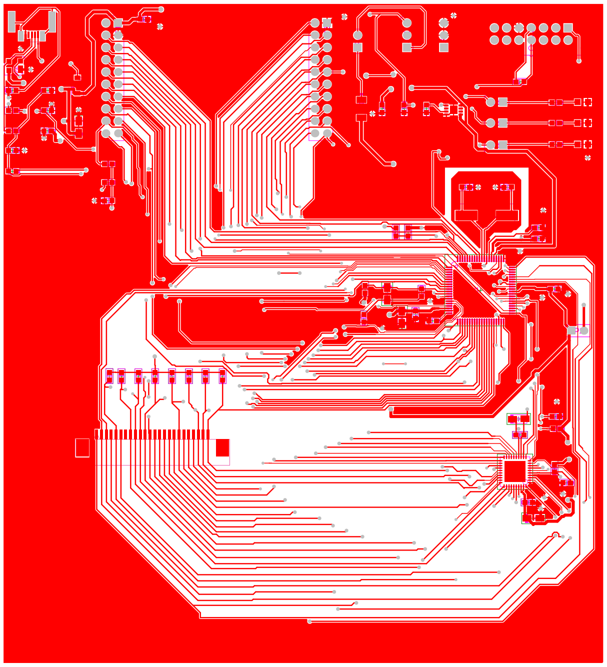


Figure 34. Layout - Top Layer

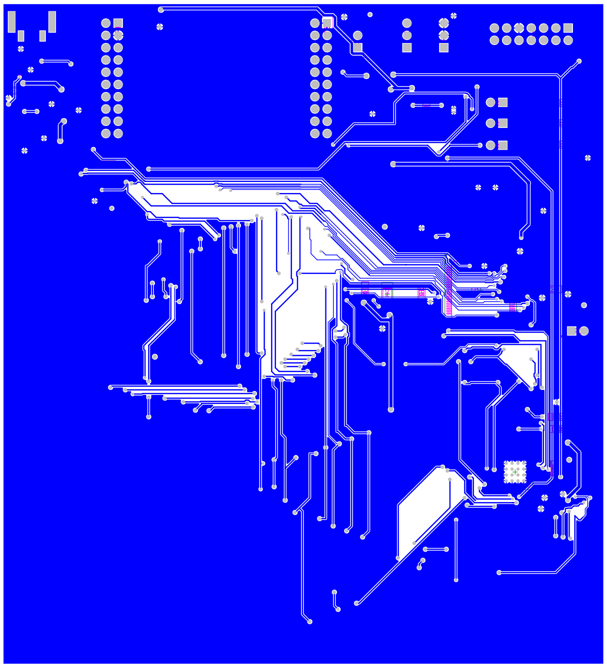


Figure 35. Layout – Bottom Layer

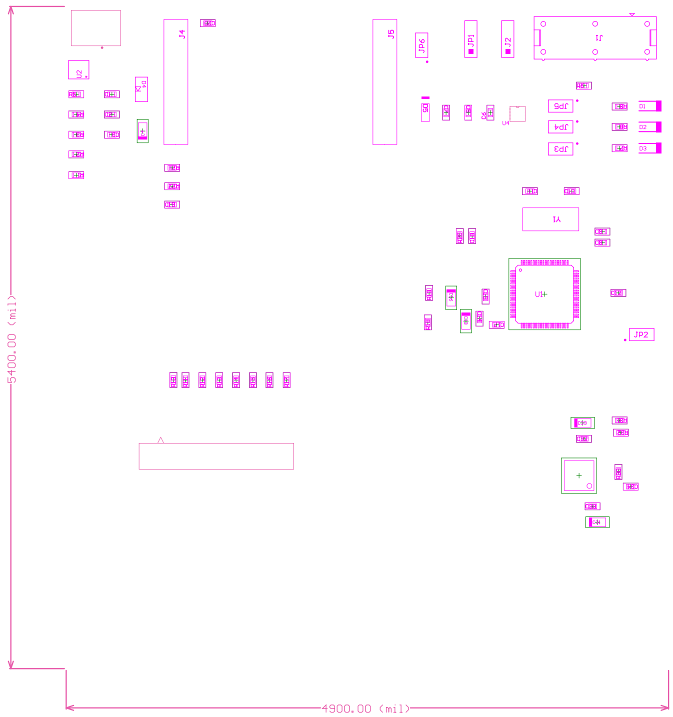


Figure 36. Mechanical Dimensions

8.3 Altium Project

To download the Altium project files for each board, see the design files at http://www.ti.com/tool/DESIGNNUMBER.

8.4 Gerber FIles

To download the Gerber files for each board, see the design files at http://www.ti.com/tool/DESIGNNUMBER.

9 Software Files

To download the software files for this reference design, please see the link at http://www.ti.com/tool/DESIGNNUMBER.

10 References

1. 1. MSP430x5xx and MSP430x6xx Family User's Guide (SLAU208)
2. 2. MSP430x2xx Family User's Guide (SLAU144)
3. 3. Implementing an Ultralow-Power Keypad Interface with the MSP430, 2002 (SLAA149)
4. 4. Enabling Low-Power Windows 8 HID Over I2C Applications using MSP430 Microcontrollers, 2012 (SLAA569)
5. Programming With the Bootstrap Loader (BSL), 2014 (SLAU319)
6. Creating a Custom Flash-Based Bootstrap Loader (BSL), 2013 (SLAA450)
7. USB Field Firmware Update on MSP430™ MCUs, 2011 (SLAA452)
8. MSPBoot - Main Memory Bootloader for MSP430™ Microcontrollers, 2014 (SLAA600)
9. 5. USB HID Usage Tables. http://www.usb.org/developers/hidpage/

11 About the Author

Luis Reynoso is an Applications Engineer at Texas Instruments. He has taken multiple customer-facing roles in the embedded industry, and during this time he has published several Applications Notes and papers for microcontrollers. He joined the MSP430 Applications team in 2010.

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265