ABSTRACT

This application report describes a low-cost highly-flexible composite USB keyboard implementation based on MSP430F5xx/MSP430F6xx families. Schematics and software are included allowing for an easy implementation and customization.

The document explains basic necessary concepts but familiarity with the MSP430™ USB Developers Package (MSP430USBDEVPACK) and USB HID specification is assumed.


Contents

1 Introduction ................................................................. 2
2 Implementation ............................................................. 2
3 Software ........................................................................ 6
4 Hardware and Peripheral Usage ....................................... 12
5 Using the USB Keyboard ............................................... 12
6 Schematics ...................................................................... 15
7 References ...................................................................... 16

List of Figures

1 Key Matrix ................................................................... 3
2 Keyboard Schematic ......................................................... 3
3 Detection of a Key Using Column-Interrupt Method ................. 4
4 Detection of a Key Using Polling Method ................................. 4
5 “Ghost” Key Detection ....................................................... 5
6 USB Keyboard Software Modules .................................... 6
7 USB Keyboard Flow Diagram ............................................ 7
8 Digital Keyscan Flow Diagram ......................................... 9
9 USB Keyboard in Windows Device Manager ......................... 13
10 Testing the HID Custom Interface ................................... 14
11 Schematics ................................................................. 15

List of Tables

1 VID/PID Used by the Device ............................................ 5
2 HID Keyboard Report Format ......................................... 8
3 Communication Protocol Report Descriptor ........................ 8
4 Implemented Protocol ...................................................... 8
5 Configuration Constant Table ........................................... 11
6 ScanCodes ................................................................. 11
7 MSP430F550x/5510 Peripheral Usage ................................. 12
8 MSP430F550x/5510 Pinout Usage ....................................... 12
1 Introduction

This application report describes the implementation of a USB keyboard with the following characteristics:

• 101 keys, 2 LEDs: standard HID keyboard and LED usage
• 16x8 matrix: allows for easy customization of different keyboard layouts
• Composite USB device: In addition to the keyboard interface, it includes an HID-datapipe back-channel which can be used to transmit any custom data
• HID boot protocol support, allowing keyboard to be used to interface with a PC's BIOS
• "Ghost" key handling in software, to prevent errors from multiple key presses
• Uses MSP430F550x/5510 low-cost USB family

The Texas Instruments MSP430F550x/5510 devices are ultra-low power microcontrollers featuring a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. In addition, this MSP430 family includes an integrated USB and PHY supporting USB 2.0 full-speed communication, four 16-bit timers, a high-performance 10-bit analog-to-digital converter (ADC), two universal serial communication interfaces (USCI), hardware multiplier, DMA, real-time clock module with alarm capabilities, and 31 or 47 I/O pins.

2 Implementation

2.1 Key Matrix

The USB keyboard presented in this application report 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).

This implementation uses a 16 rows x 8 columns matrix, which allows up to 128 keys, but it actually uses only 101 keys in total.

The key matrix is shown in Figure 1.
Each key works like a switch, and pulldowns are implemented on each column, keeping the idle state low (see Figure 2).

There are multiple ways to scan a key matrix, but this application report uses two methods, referred in this application report as: column-interrupt and polling.
In the column-interrupt approach, all rows are actively driven at the same time and columns are configured to interrupt the processor when any single 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 it does not provide the exact key being pressed.

Figure 3 shows the key matrix behavior when the Enter key is pressed in column-interrupt mode. Actively driven rows and columns are shown in red. Notice that the Col1 pin would detect a change when the Enter key is pressed, but the effect would be the same for any other pin pressed in the same column.

Figure 3. 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 (see Figure 4). In the polling method, each row is scanned separately driving one row at a time in sequential order. The columns are then read giving the exact keys being pressed.

One of the caveats when using this 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 (see Figure 5).
The software included in this application report detects potential "ghost" keys and does not report them to the host.

2.2 **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-HID interface with the first interface being a standard Keyboard and the second interface used as a DataPipe. One of the advantages of using this implementation, which using only HID interfaces, is that no drivers are required.

Although the relevant code for the keyboard implementation uses the standard keyboard interface, the DataPipe interface was added to provide users with more flexibility and to facilitate customization.

This interface can be used to send or receive any type of data to/from the host, so that the MSP430 microcontroller not only performs the job of a digital keyboard, but it can also be used to perform other jobs taking advantage of the same USB interface and the rest of the peripherals. Some examples include reading sensors using ADC and reporting to PC, 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 an HID API which enables communication between a PC and a MSP430 microcontroller running the HID API stack. This HID API is available in executable format and source code in the MSP430 USB Developers Package (MSP430USBDEVPACK).

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

VID and PID can be modified according to the particular application but the default code used for this example uses the values shown in Table 1.

<table>
<thead>
<tr>
<th>Table 1. VID/PID Used by the Device</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>VID</strong></td>
</tr>
<tr>
<td><strong>PID</strong></td>
</tr>
</tbody>
</table>
3 Software

3.1 Tools
The software included in this application report was built and tested using:
• IAR Embedded Workbench™ for MSP430 5.30.4 IDE
• Code Composer Studio™ (CCS) 5.1.0 IDE

3.2 Software Implementation
Figure 6 shows the software layers for the USB keyboard.

![Figure 6. USB Keyboard Software Modules](image)

Software is designed in a modular way, re-using existing TI libraries such as driverlib and the USB API and adding new modules from low-level drivers to application level. These modules include:

- **USB Keyboard**
  
  *Description*
  Main application initializing the microcontroller, peripherals, and executing a loop checking and servicing the rest of the modules.

  *Files*
  Src\TI_USBKBD_main.c

  *Flow Diagram*
Figure 7. USB Keyboard Flow Diagram
• **Keyboard Report**

  **Description**
  Handles the HID Keyboard report, adding and removing keys from the report depending on press/release events and sends the report to the USB Host.

  **Files**
  Src\TI_USBKBD_HIDKBD_report.c  
  Src\Include\TI_USBKBD_HIDKBD_report_public.h

  **HID Keyboard Report Format**

<table>
<thead>
<tr>
<th>Bit7</th>
<th>Bit6</th>
<th>Bit5</th>
<th>Bit4</th>
<th>Bit3</th>
<th>Bit2</th>
<th>Bit1</th>
<th>Bit0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte0</td>
<td>Right GUI</td>
<td>Right Alt</td>
<td>Right Shift</td>
<td>Right Ctrl</td>
<td>Left GUI</td>
<td>Left Alt</td>
<td>Left Shift</td>
</tr>
<tr>
<td>Byte1</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte2</td>
<td></td>
<td>Key_array[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte3</td>
<td></td>
<td>Key_array[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte4</td>
<td></td>
<td>Key_array[2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte5</td>
<td></td>
<td>Key_array[3]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte6</td>
<td></td>
<td>Key_array[4]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte7</td>
<td></td>
<td>Key_array[5]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• **Communication Protocol**

  **Description**
  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**
  Src\TI_USBKBD_comm_protocol.c  
  Src\Include\TI_USBKBD_comm_protocol_public.h

  **HID Custom Interface Report Descriptor**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN Report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report ID</td>
<td>1 byte</td>
<td>The report ID of the chosen report (automatically assigned to 0x3F by the HID-Datapipe calls)</td>
</tr>
<tr>
<td>Size</td>
<td>1 byte</td>
<td>The number of valid bytes in the data field</td>
</tr>
<tr>
<td>Data</td>
<td>62 bytes</td>
<td>Data payload</td>
</tr>
<tr>
<td>OUT Report</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report ID</td>
<td>1 byte</td>
<td>The report ID of the chosen report (must be assigned to 0x3F by the host)</td>
</tr>
<tr>
<td>Size</td>
<td>1 byte</td>
<td>The number of valid bytes in the data field</td>
</tr>
<tr>
<td>Data</td>
<td>62 bytes</td>
<td>Data payload</td>
</tr>
</tbody>
</table>

  **Data Payload Protocol**

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD</td>
<td>1 byte</td>
<td>1 = Toggle CAPS LED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = Toggle NUM LED</td>
</tr>
<tr>
<td>Data</td>
<td>61 bytes</td>
<td>Unused</td>
</tr>
</tbody>
</table>
**DKS (Digital KeyScan)**

*Description*
Handles the digital keyboard scanning, detecting key press/release events, and reporting them to the keyboard report module.

*Files*
- `Src\TI_USBKBD_DKS.c`
- `Src\Include\TI_USBKBD_DKS_public.h`

*Flow Diagram*

![Flow Diagram](image)

*Figure 8. Digital Keyscan Flow Diagram*
• **USB API / USB HID**

**Description**
The MSP430 USB API stack is a software solution provided by Texas Instruments that includes support for:
- Communications Device Class (CDC)
- Human Interface Device class (HID)
- Mass Storage Class (MSC)
- Personal HealthCare Device Class (PHDC)

This software solution, including detailed documentation, is available in the MSP430 USB Developers Package ([MSP430USBDEVPACK](#)).

**Files**
- `Src\USB_API\*.*`
- `Src\USB_config\*.*`
- `Src\USB_App\*.*`

• **Ticktimer**

**Description**
Handles a general purpose interrupt timer that is used as a timebase, to wake-up the processor, and to trigger a new keyboard scan, among other functions.

The ticktimer is implemented using TA0.0 with a default time base of 2 ms.

**Files**
- `Src\TI_USBKBD_ticktimer.c`
- `Src\Include\TI_USBKBD_ticktimer_public.h`

• **MSP430 Driverlib**

**Description**
The Texas Instruments MSP430 Peripheral Driver Library (Driverlib) is a set of drivers that provide an easy mechanism to use the MSP430 peripherals. This software uses Driverlib to initialize the PMM and UCS modules.

Source code and detailed documentation are available in MSP430Ware ([www.ti.com/msp430ware](http://www.ti.com/msp430ware)). For simplicity purposes, this project includes only the pre-compiled libraries for IAR and CCS using a small memory model and header files.

**Files**
- `Src\driverlib\*.h`
- `Src\driverlib\driverlib_small_CCS.lib`
- `Src\driverlib\driverlib_small_IAR.r43`

### 3.3 Configuration and ScanCode Tables

For modularity purposes and to allow for an easier optimization or upgrade, the USB keyboard software reserves some Flash sectors for constant tables that define some of the functionality of the application and define the ScanCode table.

• **Configuration Constant Table**

**Description**
Contains the USB keyboard version and configuration constants defining the KeyScan functionality, such as debounce counter, ticktimer period, etc.

**Files**
- `Src\TI_USBKBD_SharedTables.c` (declaration)
- `Src\Include\TI_USBKBD_public.h` (typedef)

**Declaration**
```c
const USBKBD_config_const_t USBKBD_configconst_s
```
### Configuration Constant Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagicKey</td>
<td>4 bytes</td>
<td>Indicates the start of the table. 0xDEADC0DE is used by default.</td>
</tr>
<tr>
<td>Version</td>
<td>2 bytes</td>
<td>USB keyboard version in BCD format: 0x0101 - 1.0.1</td>
</tr>
<tr>
<td>ticktimer_div</td>
<td>2 bytes</td>
<td>TickTimer divider (based on ACLK): 66 represents a period of 66 / 32768 = ~2 ms</td>
</tr>
<tr>
<td>debounce_cycles</td>
<td>2 bytes</td>
<td>Number of debounce cycles in tick counts: 2 represents a debounce of 4 ms with Ticktimer = 2 ms</td>
</tr>
<tr>
<td>inactive_timeout</td>
<td>2 bytes</td>
<td>Number of tick counts before going to interrupt_column mode if no key is detected: 8 represents 16 ms with Ticktimer = 2 ms</td>
</tr>
</tbody>
</table>

### ScanCode Table

**Description**

Contains the USB Keyboard scancode table, mapping each row and column to the corresponding value based on HID Usage Tables.

**Files**

Src\TI_USBKBD_SharedTables.c(declaration)
Src\Include\TI_USBKBD_public.h(typedef)

**Declaration**

`const USBKBD_scancodest_t USBKBD_scancodes_s`

**Location**

USBKBD_SCANCODES_SEGMENT (0xFA00-0xFBFF)

### ScanCodes Table

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagicKey</td>
<td>4 bytes</td>
<td>Indicates the start of the table. 0xDEADC0DE is used by default.</td>
</tr>
<tr>
<td>keycode</td>
<td>128 bytes</td>
<td>Keycodes for each key in the following order: Row0,Col0 Row0,Col1 ... Row0,Col7 Row1,Col0 Row1,Col1 ... Row15,Col6 Row15,Col7</td>
</tr>
</tbody>
</table>
## 4 Hardware and Peripheral Usage

In addition to system modules (UCS, PMM), this keyboard implementation uses the peripherals shown in Table 7.

### Table 7. MSP430F550x/5510 Peripheral Usage

<table>
<thead>
<tr>
<th>Peripheral</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB</td>
<td>Communication with host (Composite HID-HID)</td>
</tr>
<tr>
<td>Timer_A0 (TA0.0)</td>
<td>TimerTick used as a time base to perform periodic polling, debounce, etc.</td>
</tr>
</tbody>
</table>

In addition to the circuitry required for USB and common functionality (reset, VCC, VSS, crystal, etc.), the USB keyboard uses the pins shown in Table 8.

### Table 8. MSP430F550x/5510 Pinout Usage

<table>
<thead>
<tr>
<th>Rows</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>KSO0</td>
<td>P2.0</td>
</tr>
<tr>
<td>KSO1</td>
<td>P2.1</td>
</tr>
<tr>
<td>KSO2</td>
<td>P2.2</td>
</tr>
<tr>
<td>KSO3</td>
<td>P2.3</td>
</tr>
<tr>
<td>KSO4</td>
<td>P2.4</td>
</tr>
<tr>
<td>KSO5</td>
<td>P2.5</td>
</tr>
<tr>
<td>KSO6</td>
<td>P2.6</td>
</tr>
<tr>
<td>KSO7</td>
<td>P2.7</td>
</tr>
<tr>
<td>KSO8</td>
<td>P3.0</td>
</tr>
<tr>
<td>KSO9</td>
<td>P3.1</td>
</tr>
<tr>
<td>KSO10</td>
<td>P3.2</td>
</tr>
<tr>
<td>KSO11</td>
<td>P3.3</td>
</tr>
<tr>
<td>KSO12</td>
<td>P3.4</td>
</tr>
<tr>
<td>KSO13</td>
<td>P5.0</td>
</tr>
<tr>
<td>KSO14</td>
<td>P5.1</td>
</tr>
<tr>
<td>KSO15</td>
<td>P5.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEDs</th>
<th>KSI0</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED0 (CAPS)</td>
<td>P4.7</td>
</tr>
<tr>
<td>LED1 (NUM)</td>
<td>P4.6</td>
</tr>
<tr>
<td>KSI1</td>
<td>P1.1</td>
</tr>
<tr>
<td>KSI2</td>
<td>P1.2</td>
</tr>
<tr>
<td>KSI3</td>
<td>P1.3</td>
</tr>
<tr>
<td>KSI4</td>
<td>P1.4</td>
</tr>
<tr>
<td>KSI5</td>
<td>P1.5</td>
</tr>
<tr>
<td>KSI6</td>
<td>P1.6</td>
</tr>
<tr>
<td>KSI7</td>
<td>P1.7</td>
</tr>
</tbody>
</table>

Schematics showing the implementation on the USB keyboard are found in Section 6.

## 5 Using the USB Keyboard

When connected to a PC, the USB keyboard should be detected by the operating system and enumerated without drivers. Windows shows three devices in the Device Manager (see Figure 9).

- **Human Interface Devices**
  - USB Human Interface Device: Standard keyboard in intf0 (MI_00)
  - USB Human Interface Device: Custom interface in intf1 (MI_01)

- **Keyboards**
  - HID Keyboard Device: Standard keyboard in intf0 (MI_00)
The keyboard can now be tested and used as a standard keyboard.

In addition to the regular key functionality, the custom interface can be tested using the MSP430 HID USB Application following these steps (see Figure 10):
1. Select the VID and PID (default: VID = 0x2047, PID = 0x0401).
2. Click Set VID PID.
3. Click Connect.
4. The LED should turn green.
5. Write one of the commands in the Send & Receive field.
6. Observe the response from the USB keyboard.
5.1 Performance

The usual response time for keyboards is approximately 5 to 50 ms. While this depends on different factors such as the mechanical implementation of the keyboard, USB bus load, etc., by using this software, 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, polling scan rate, USB polling interval, and microcontroller internal frequency can be adjusted to meet particular requirements.

One important factor affecting the response time is the polling rate, 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.

During bench tests, this implementation was measured to take ~1870 cycles (which is equivalent to ~233 µs at 8 MHz) for the first pressed key and ~520 cycles (~65 µs at 8 MHz) for each additional pressed key.

5.2 Memory Footprint

The following memory footprint was obtained using IAR Embedded Workbench 5.30.4 using the maximum optimization level:

- Code: 7626 Bytes
- Constants: 1096 Bytes
- Data: 679 Bytes
Figure 11. Schematics
7 References

1. USB HID specification (www.usb.org/developers/devclass_docs/HID1_11.pdf)
2. MSP430x5xx/MSP430x6xx Family User's Guide (SLAU208)
3. MSP430F550x/MSP430F5510 Mixed Signal Microcontroller data sheet (SLAS645)
5. MSP430Ware (www.ti.com/msp430ware)
6. Implementing an Ultralow-Power Keypad Interface With the MSP430 (SLAA139)
IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

TI products are neither designed nor intended for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal and regulatory requirements in connection with such use. TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or “enhanced plastic.” Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use. TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

<table>
<thead>
<tr>
<th>Products</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Automotive and Transportation</td>
</tr>
<tr>
<td>Amplifiers</td>
<td>Communications and Telecom</td>
</tr>
<tr>
<td>Data Converters</td>
<td>Computers and Peripherals</td>
</tr>
<tr>
<td>DLP® Products</td>
<td>Consumer Electronics</td>
</tr>
<tr>
<td>DSP</td>
<td>Energy and Lighting</td>
</tr>
<tr>
<td>Clocks and Timers</td>
<td>Industrial</td>
</tr>
<tr>
<td>Interface</td>
<td>Medical</td>
</tr>
<tr>
<td>Logic</td>
<td>Security</td>
</tr>
<tr>
<td>Power Mgmt</td>
<td>Space, Avionics and Defense</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>Video and Imaging</td>
</tr>
<tr>
<td>RFID</td>
<td></td>
</tr>
<tr>
<td>OMAP Mobile Processors</td>
<td></td>
</tr>
<tr>
<td>Wireless Connectivity</td>
<td></td>
</tr>
</tbody>
</table>

TI E2E Community Home Page e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2012, Texas Instruments Incorporated