

# ***Interrupt and Exception Handling on Hercules™ ARM® Cortex®-R4/5-Based Microcontrollers***

*Christian Herget, Zhaohong Zhang*

## **ABSTRACT**

This application report describes the interrupt and exception handling of the ARM Cortex-R4/5 processor as implemented on Hercules-based microcontrollers, as well as the related operating modes of the processor.

Project collateral and source code discussed in this application report can be downloaded from the following URL: <http://www.ti.com/lit/zip/spna218>.

---

**NOTE:** TI assumes no liability that the discussed implementation and provided code are free from faults, compliant to certain coding guidelines, nor was it developed in accordance with certain standards. The implementer has to ensure and verify that the code conforms to appropriate rules and standards, and he has the sole responsibility to ensure and verify correct functionality in this application.

---

## **Contents**

|   |   |    |
|---|---|----|
| 1 | Operating Modes of the Cortex-R4/5 Processor.....                           | 2  |
| 2 | Exception Handling on Hercules ARM Cortex-R4/5-Based Microcontrollers ..... | 4  |
| 3 | Type of Aborts.....   | 6  |
| 4 | Interrupt Handling on Hercules ARM Cortex-R4/5-Based Microcontrollers ..... | 7  |
| 5 | Example Code Snippets.....  | 12 |
| 6 | References .....  | 15 |

## **List of Figures**

|   |   |    |
|---|---|----|
| 1 | Legacy ARM7 Interrupt Flow .....                    | 8  |
| 2 | Legacy ARM7 Interrupt Connection Scheme .....       | 8  |
| 3 | Vectored Interrupt Flow .....                       | 9  |
| 4 | Vectored Interrupt Connection Scheme .....          | 9  |
| 5 | Hardware Vectored Interrupt Flow.....               | 9  |
| 6 | Hardware Vectored Interrupt Connection Scheme ..... | 10 |
| 7 | Encoding of SVC Instruction in ARM Mode.....        | 12 |
| 8 | Encoding of SVC Instruction in Thumb Mode .....     | 12 |

## **List of Tables**

|   |  |   |
|---|--|---|
| 1 | Operating Modes.....                   | 2 |
| 2 | Processor Registers Organization ..... | 3 |
| 3 | Exception Vector Table .....           | 5 |

Hercules is a trademark of Texas Instruments.  
 ARM, Cortex are registered trademarks of ARM Limited.  
 All other trademarks are the property of their respective owners.

## 1 Operating Modes of the Cortex-R4/5 Processor

The Cortex-R4/5 processor implements seven different operating modes. Specific access rights and a specific set of program registers are associated to each operating mode. [Table 1](#) lists the seven different operating modes, along with the applicable access rights (privilege levels).

**Table 1. Operating Modes**

| Mode Name      | Abbreviation | Access Rights |
|----------------|--------------|---------------|
| SuperVisor     | SVC          | Privileged    |
| Undefined      | UND          | Privileged    |
| Abort          | ABT          | Privileged    |
| Fast Interrupt | FIQ          | Privileged    |
| Interrupt      | IRQ          | Privileged    |
| System         | SYS          | Privileged    |
| User           | USR          | Unprivileged  |

As shown in [Table 1](#), modes other than user mode have privileged access rights; the user mode is the only mode with unprivileged access rights. Privileged access rights are needed to access certain configuration registers in the processor (CP15) as well as certain registers in the system and peripheral modules implemented in the Hercules-based microcontrollers. The Cortex-R4/5 processor, as implemented in the Hercules-based microcontrollers, has a Memory Protection Unit (MPU) that can be used to set the necessary access rights for certain regions in the processors memory space [\[1\]](#).

The individual operating modes are described in the following sections.

### 1.1 Program Registers

The Cortex-R4/5 processor has a set of 37 32-bit program registers. Not all of these registers are accessible at the same time, which registers are available depends on the processor state (ARM/Thumb2) and the operating mode. [Table 2](#) shows a matrix of the available registers for each processor state and operating mode. The program registers are called R0-R15 (general-purpose registers) and CPSR and SPSR (status registers). In addition, there are synonyms defined for the general-purpose registers R0-R15 based on their special use ([\[1\]](#), [\[3\]](#)):

- R0 - R3 are also called A1 - A4, as a synonym for Argument registers, as these registers are used to hold the C function arguments (A1-A4) and return values (A1 and A2).
- R4 - R11 are also called V1 - V8, as a synonym for Variable registers, as these registers are used to hold variables within a C function.
- R12 is also called IP, as a synonym for intra-procedure-call scratch register.
- R13 is also called SP, as a synonym for stack pointer.
- R14 is also called LR, as a synonym for Link Register.
- R15 is also called PC, as a synonym for program counter.

Some of the registers are operating mode-specific banked registers, which are only available in a certain operating mode. The banked registers in [Table 2](#) are shaded in gray.

Some derivatives of the Hercules family have a built-in Floating-Point Coprocessor (VFPv3). The VFP comes along with its own set of processor registers, which are not listed in [Table 2](#) and are independent of the processors operating mode.

**Table 2. Processor Registers Organization**

|                   |   | Modes             |                  |          |          |          |          |          |
|-------------------|---|-------------------|------------------|----------|----------|----------|----------|----------|
|                   |   | Unprivileged Mode | Privileged Modes |          |          |          |          |          |
|                   |   |                   | Exception Modes  |          |          |          |          |          |
|                   |   | USR               | SYS              | SVC      | ABT      | UNDEF    | IRQ      | FIQ      |
| ARM Mode (32 Bit) | Thumb Mode (16 Bit)   | R0, A1            | R0               | R0       | R0       | R0       | R0       | R0       |
|                   |   | R1, A2            | R1               | R1       | R1       | R1       | R1       | R1       |
|                   |   | R2, A3            | R2               | R2       | R2       | R2       | R2       | R2       |
|                   |   | R3, A4            | R3               | R3       | R3       | R3       | R3       | R3       |
|                   |   | R4, V1            | R4               | R4       | R4       | R4       | R4       | R4       |
|                   |   | R5, V2            | R5               | R5       | R5       | R5       | R5       | R5       |
|                   |   | R6, V3            | R6               | R6       | R6       | R6       | R6       | R6       |
|                   |   | R7, V4, AP        | R7               | R7       | R7       | R7       | R7       | R7       |
|                   | Can only be accessed via special MOV instructions in Thumb Mode | R8, V5            | R8               | R8       | R8       | R8       | R8       | R8_fiq   |
|                   |   | R9, V6            | R9               | R9       | R9       | R9       | R9       | R9_fiq   |
|                   |   | R10, V7           | R10              | R10      | R10      | R10      | R10      | R10_fiq  |
|                   |   | R11, V8           | R11              | R11      | R11      | R11      | R11      | R11_fiq  |
|                   |   | R12, V9, 1P       | R12              | R12      | R12      | R12      | R12      | R12_fiq  |
|                   | Thumb Mode  | R13, SP           | R13              | R13_svc  | R13_abt  | R13_und  | R13_irq  | R13_fiq  |
|                   |   | R14, LR           | R14              | R14_svc  | R14_abt  | R14_und  | R14_irq  | R14_fiq  |
|                   |   | R15, PC           | PC               | PC       | PC       | PC       | PC       | PC       |
|                   |   | CPSR              | CPSR             | CPSR     | CPSR     | CPSR     | CPSR     | CPSR     |
|                   |   |                   |                  | SPSR_svc | SPSR_abt | SPSR_und | SPSR_irq | SPSR_fiq |
|                   |   |                   |                  |          |          |          |          |          |
|                   | Banked Registers  |                   |                  |          |          |          |          |          |

## 1.2 SuperVisor Mode

This is the mode entered after a reset of the CPU has occurred; it can also be entered by the SuperVisor Call (SVC) instruction. Usually the initial setup of the processor and MCU is done in this mode after cold or warm reset. The processor is usually switched to System or User mode afterwards, to start with the execution of the application. Operating systems can use the SuperVisor mode to gather privileged access rights for certain tasks.

For more information regarding the SVC exception/instruction, see [Section 2.1](#).

The SuperVisor mode has three processor registers banked: the SP, LR and the SPSR.

## 1.3 Undefined Mode

This mode is entered when an undefined instruction exception occurs, see [Section 2.2](#).

The undefined operating mode has three processor registers banked: the SP, LR and the SPSR.

## 1.4 Abort Mode

This mode is entered when a Prefetch abort or data abort exception occurred, [Section 2.3](#) and [Section 2.4](#).

The abort operating mode has three processor registers banked: the SP, LR and the SPSR.

## 1.5 Fast Interrupt Mode

This mode is entered after a *Fast Interrupt Request* (FIQ) has been received by the processor, see [Section 2.5](#).

The *Fast Interrupt* operating mode has eight processor registers banked: R8 - R12, the SP, LR and the SPSR. Having registers R8 - R12 banked has the advantage that these registers do not have to be preserved or saved to the stack in order to use them in an interrupt handler, and can, thus, improve interrupt latency.

## 1.6 Interrupt Mode

This mode is entered after an *Interrupt Request* (IRQ) has been received by the processor, see [Section 2.6](#).

The interrupt operating mode has three processor registers banked: the SP, LR and the SPSR.

## 1.7 System and User Mode

Both modes, the *System mode* and *User mode* share the same set of processor registers and, thus, also the same stack. The *System mode* can be used as the privileged operating mode for an operating system or scheduler. The *User mode* can be used as the unprivileged operating mode for an operating system.

Sharing the same set of processor registers and stack makes it easier to switch between these modes within a function and access rights, as no special precautions have to be taken into account to keep the C environment (stack) intact. A switch between the two modes can be done with the help of a SuperVisor Call (SVC) handler. For more details, see [Section 2.1](#).

However, the system designer should be careful with choosing the right mode for certain tasks and whether dynamical switching between *User* and *System mode* is allowed or not. Dynamical switching increases overall system complexity and might result in unwanted behavior, if not used the right way.

## 2 Exception Handling on Hercules ARM Cortex-R4/5-Based Microcontrollers

Exceptions are interruptions of the normal program flow this includes peripheral interrupts. The Cortex-R4/5 processor usually takes care to preserve the critical parts of the current processor state, so that the normal program flow could be resumed after the exception was handled by the application (saving and restoring of the CPSR and banked Stack Pointers).

The Cortex-R4/5 processor implements the following exceptions:

- SuperVisor Call (SVC) also known as Software Interrupt (SWI) [Section 2.1](#)
- Undefined Instruction (UNDEF) [Section 2.2](#)
- Prefetch Abort (PABT) [Section 2.3](#)
- Data Abort (PABT) [Section 2.4](#)
- Fast Interrupt Request (FIQ) [Section 2.5](#)
- Interrupt Request (IRQ) [Section 2.6](#)
- Reset [Section 2.7](#)
- Break Point (BKPT) [Section 2.8](#)

The processor state (ARM/Thumb2) and the operating mode can and will change on exception entry. The Cortex-R4/5 processor supports exception entry in ARM and in Thumb2 state, the default after reset as implemented in the Hercules family is the ARM state. However, it is possible to change this behavior during runtime (usually during the startup phase), but changing the default behavior might cause undesired side effects.

---

**NOTE:** The processor core as implemented always defaults to the ARM state exception vectors after reset. If the Thumb(2) state exception vectors should be used, the software (application) has to reconfigure the processor core via the CP15. Therefore, there will be a short time frame where an exception handler will be entered in ARM state (for example, Prefetch or Data abort). In case the handler routine is coded with Thumb(2) instructions, this might result in a deadlock of the processor trying to execute the undefined instruction handler (best case).

---

The operating mode changes according to the type of the exception. [Table 3](#) shows the seven exceptions, the mode on entry and the changes in the A, F and I bits in the CPSR.

**Table 3. Exception Vector Table**

| Vector Table Offset | Exception | Mode on Entry    | Bits in CPSR |           |     |
|---------------------|-----------|------------------|--------------|-----------|-----|
|                     |           |                  | A            | F         | I   |
| 0x00                | Reset     | SuperVisor       | Set          | Set       | Set |
| 0x04                | UNDEF     | Undefined (Mode) | Unchanged    | Unchanged | Set |
| 0x08                | SVC       | SuperVisor       | Unchanged    | Unchanged | Set |
| 0x0C                | PABT      | Abort            | Set          | Unchanged | Set |
| 0x10                | DABT      | Abort            | Set          | Unchanged | Set |
| 0x14                | Reserved  |                  |              |           |     |
| 0x18                | IRQ       | IRQ              | Set          | Unchanged | Set |
| 0x1C                | FIQ       | FIQ              | Set          | Set       | Set |

The A, F and I bits in the CPSR have the following meanings:

- A (bit [8]) Asynchronous (imprecise) abort disable bit
- I (bit [7]) Interrupt disable bit
- F (bit [6]) Fast Interrupt disable bit

A '1' in these registers means that the corresponding exceptions are disabled. For more information, see the *Program Status Registers* section of the *ARM Cortex-R4 and Cortex-R4F Technical Reference Manual* (ARM DDI 0363C). For more details about the CPSR, see the *Program status registers* section [1].

## 2.1 SuperVisor Call (SVC)

The *SuperVisor Call* instruction SVC is used to enter the *SuperVisor Mode*. Usually this is done to execute an operating system service, like a processor mode change or writing to a protected configuration register. Therefore, the SVC instruction includes an 8 bit (Thumb2) or 24 bit (ARM) integer constant to be able to determine the requested service. The program has to manually read the instruction and extract the integer constant [1].

For more details on how to use and write a SVC handler, see [Section 5.1](#).

## 2.2 Undefined Instruction (UNDEF)

The processor takes the *undefined instruction* exception when it encounters an instruction that is undefined in the appropriate version of the ARM instruction set, or which is for the VFP when the VFP is disabled. The undefined instruction exception can be used to emulate undefined instructions, or simply to handle fault situations [1].

## 2.3 Prefetch Abort (PABT)

The processor takes the *prefetch abort* if it tries to execute an instruction from a protected or faulty memory location. This is usually the case if the memory location is not implemented in the system, if the memory region is protected by the MPU, or if an error is detected in the data by the ECC checking logic. All *prefetch aborts* are precise [1].

## 2.4 Data Abort (DABT)

The processor takes the *data abort* if data should be read from or written to a protected or faulty memory location. This could be because of:

- The memory location is not implemented
- The memory location is read or write only in privileged mode (when processor is in *User mode*)
- The memory location is read or write protected by the MPU
- If an error is detected in the data by the ECC checking logic

The above list may not be complete. For more details about the data abort, see the *ARM Cortex-R4 and Cortex-R4F Technical Reference Manual* (ARM DDI 0363C). Data aborts can be precise or imprecise [1].

## 2.5 Fast Interrupt Request (FIQ)

The nFIQ is an input to the processor core, a low signal on the nFIQ input causes the processor to take the FIQ exception, if not masked. On the Hercules MCU's, the nFIQ pin is connected to the Vectored Interrupt Manager (VIM) (see [Section 4.1](#)). It is typically used to inform the application about severe errors via the Error Signaling module (ESM), but it could also be used as general-purpose interrupts line, if required by the application.

---

**NOTE:** Using the FIQ for peripheral interrupts might delay interrupts generated by the ESM and, thus, handling of these.

---

The FIQ mode has several banked processor registers to improve interrupt latency. For more information about the processor registers, see [Section 1.5](#). Furthermore, it is placed at the end of the [Table 3](#). This has the advantage that a software interrupt dispatcher or in the case that only one interrupt is mapped to the FIQ, the whole interrupt service handler could be placed at this address to further improve interrupt latency (avoiding unnecessary branches).

The FIQ is implemented as a non-maskable interrupt in the Hercules MCU's. This means, that once activated by the application (by clearing the F bit in the CPSR, see [Section 2](#)) it could not be masked again. The only way to mask the FIQ once activated is to perform a reset of the CPU system.

## 2.6 Interrupt Request (IRQ)

The nIRQ is an input to the processor core, a low signal on the nIRQ input causes the processor to take the IRQ exception, if not masked. On the Hercules MCU's the nIRQ pin is connected to the VIM (see [Section 4.1](#)). It is usually used as "general-purpose" interrupt line and interrupt dispatching is usually handled by the VIM. The Cortex-R4/5 processor offers a so called VIC port to supply the interrupt vector address directly to the processor, in order to reduce interrupt latency for IRQ's. For more information about the VIM, see [Section 4.1](#).

## 2.7 CPU Reset

The reset abort is taken after the reset input to the Cortex-R4/5 processor was released. The processor will be in SuperVisor mode (see [Section 1.2](#)) after reset and will start executing code beginning from address 0x00000000, which is the top of [Table 3](#).

## 2.8 Break Point (BKPT)

The Brake Point instruction is not discussed in this document. For more details about the Brake Point instruction, see [\[1\]](#).

# 3 Type of Aborts

## 3.1 Precise or Synchronous Aborts

Aborts for which it is ensured that the exception is taken at the instruction that caused it are called precise aborts or synchronous aborts. This means, that the abort handler could use the SPSR\_abt and R14\_abt (LR\_abt) registers to determine the instruction that generated the abort and the state of the processors when the abort occurred [\[1\]](#).

### 3.2 Imprecise or Asynchronous Aborts

If the exception is taken on an instruction later than the instruction that caused the exception, the abort will be an imprecise or asynchronous abort. This means, that it is not possible to determine the exact instruction that caused the abort. This could be the case on writes to normal-type memory, were the write is stored in a buffer until the memory system is ready to perform it. In such cases the exception will be generated and the abort will be taken after the appropriate store instruction was executed by the processor.

### 3.3 How to Determine the Cause of an Abort

The Cortex-R4/5 processor has a system control coprocessor implemented, the CP15. The CP15 offers the possibility to readout additional information about an abort. Registers in the CP15 that hold information about the cause of an abort are:

- Data Fault Status Register
- Auxiliary Fault Status Registers
- Data Fault Address Register
- Instruction Fault Address Register

For more information about the CP15 registers for fault determination, see the *Fault Status and Address Registers* section in the *ARM Cortex-R4/5 Technical Reference Manual* [1].

## 4 Interrupt Handling on Hercules ARM Cortex-R4/5-Based Microcontrollers

The Hercules ARM Cortex-R4 based microcontrollers support three different modes to handle peripheral interrupts in hardware and software:

- The first one is called Legacy mode. This mode is fully backward compatible to ARM7-based microcontrollers. In this mode, the interrupt dispatching has to be done completely in software (software dispatcher). For more details, see [Section 4.2](#).
- The second one is called Vectored Interrupt mode. This mode allows the interrupt dispatching to be done in hardware, the software has only to load the interrupt vector of the ISR from the VIM module and branch to the vector. For more details, see [Section 4.3](#).
- The third mode is only available for IRQ not for FIQ and called Hardware Vectored Interrupt mode. This mode has the advantage that the vector of the ISR has not been loaded by software. Instead, the vector is directly supplied to the Cortex-R4 core via the VIC port and saves some CPU cycles for lower interrupt latency compared to the second mode. For more details, see [Section 4.4](#).

### 4.1 Vectored Interrupt Manager (VIM) Module

The Vectored Interrupt Manager (VIM) module provides hardware assistance for prioritizing and controlling the many interrupt sources present on Hercules family devices. Interrupts are caused by events outside of the normal flow of program execution. Normally, these events require a timely response from the central processing unit (CPU); therefore, when an interrupt occurs, the CPU switches execution from the normal program flow to an ISR.

The VIM module has the following features:

- Currently supports up to 96 interrupt channels (can be extended in the future)
  - Provides programmable priority and masks for interrupt request lines
- Provides a direct hardware dispatch mechanism for fastest IRQ dispatch via the VIC port (Hardware Vectored Interrupts, see [Section 4.4](#))
- Provides two software dispatch mechanisms when the CPU VIC port is not used:
  - Index interrupt (Legacy ARM7 Interrupts, see [Section 4.2](#))
  - Register vectored interrupt (Vectored Interrupts, see [Section 4.3](#))
- Parity protected vector interrupt table to be able to detect soft errors



## 4.2 Legacy ARM7 Interrupts

The flow is visualized in [Figure 1](#), the connection scheme of the peripherals, VIM and R4(F) inside the MCU in [Figure 2](#).



Figure 1. Legacy ARM7 Interrupt Flow

1. Request FIQ/IRQ.
2. Fetch from 0x18/0x1C.
3. Branch to ISR dispatcher.
4. Load Interrupt offset.
5. Decide which ISR to execute.
6. Branch to ISR.

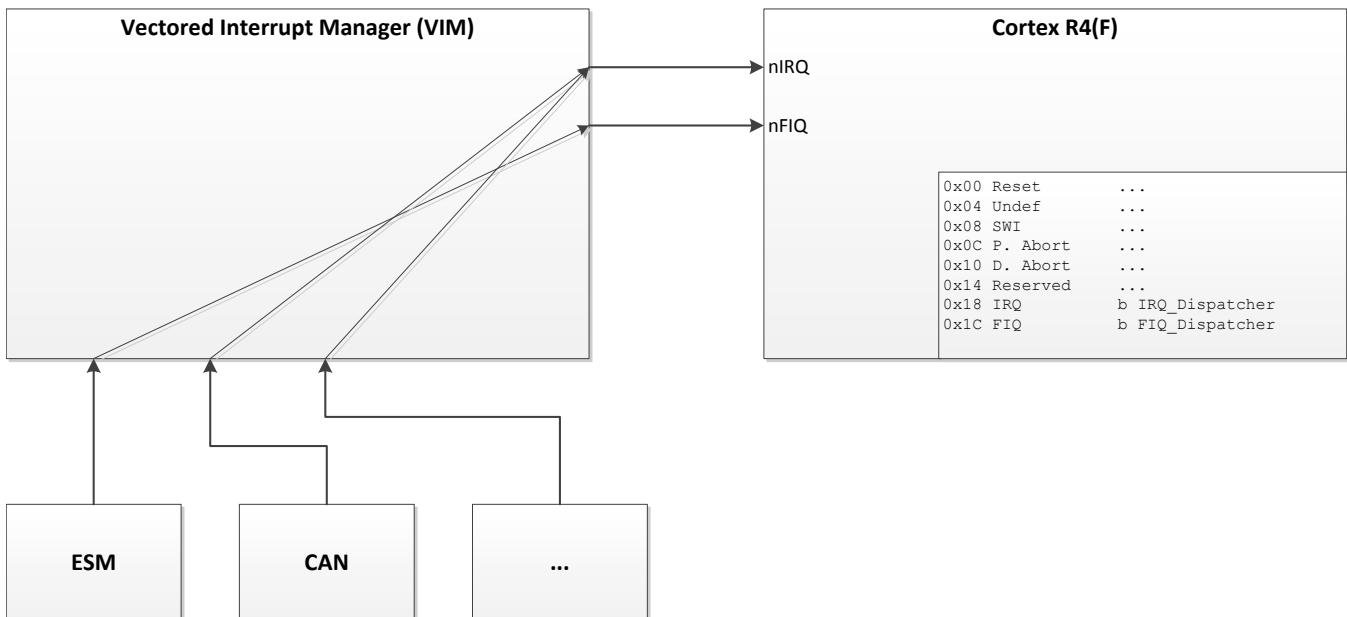


Figure 2. Legacy ARM7 Interrupt Connection Scheme

For the interrupt dispatchers, see [Section 5.2](#) regarding example code.

---

**NOTE:** The LDR instruction on address 0x20 can potentially be replaced with an unconditional jump to the ESM fault handler or even with the handler itself, in case no other interrupt except the ESM high level fault interrupt is mapped to the nFIQ request line.

---

## 4.3 Vectored Interrupts

The flow is visualized in [Figure 3](#), the connection scheme of the peripherals, VIM and R4/5(F) inside the MCU in [Figure 4](#).



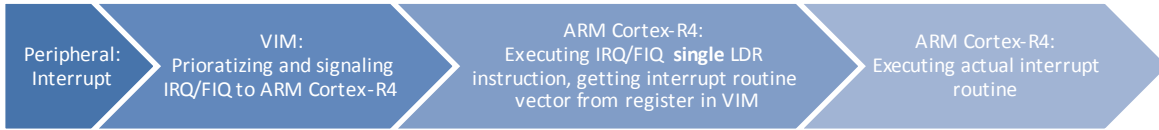


Figure 3. Vectored Interrupt Flow

1. Request FIQ/IRQ.
2. Fetch from 0x18/0x1C
3. Branch to ISR  
(LDR PC, [PC, #-0x1B0]),  
Address for highest active interrupt request derived from IRQVECREG/FIQVECREG.

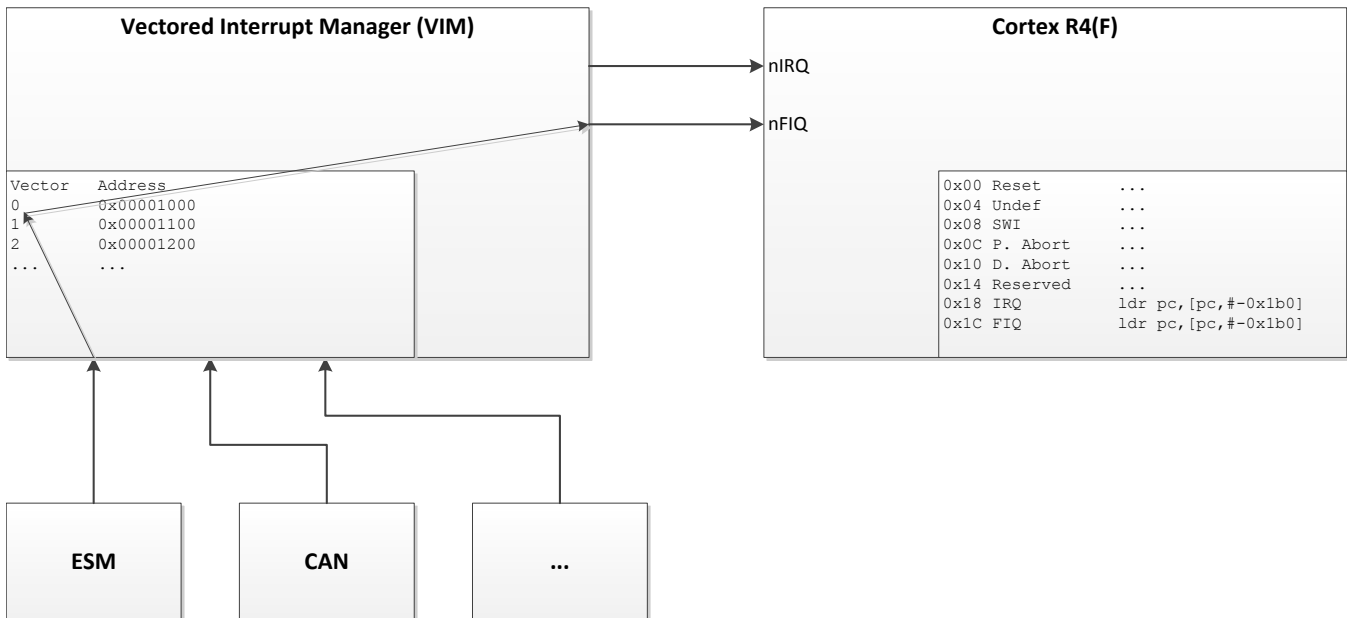


Figure 4. Vectored Interrupt Connection Scheme

For more details how these instructions work, see [Section 5.3](#).

**NOTE:** The LDR instruction on address 0x20 can potentially be replaced with an unconditional jump to the ESM fault handler or even with the handler itself, in case no other interrupt except the ESM high level fault interrupt is mapped to the nFIQ request line.

#### 4.4 Hardware Vectored Interrupts (only IRQ)

The flow is visualized in [Figure 5](#), the connection scheme of the peripherals, VIM and R4(F) inside the MCU in [Figure 6](#).

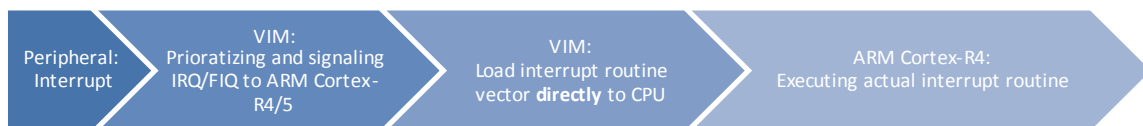
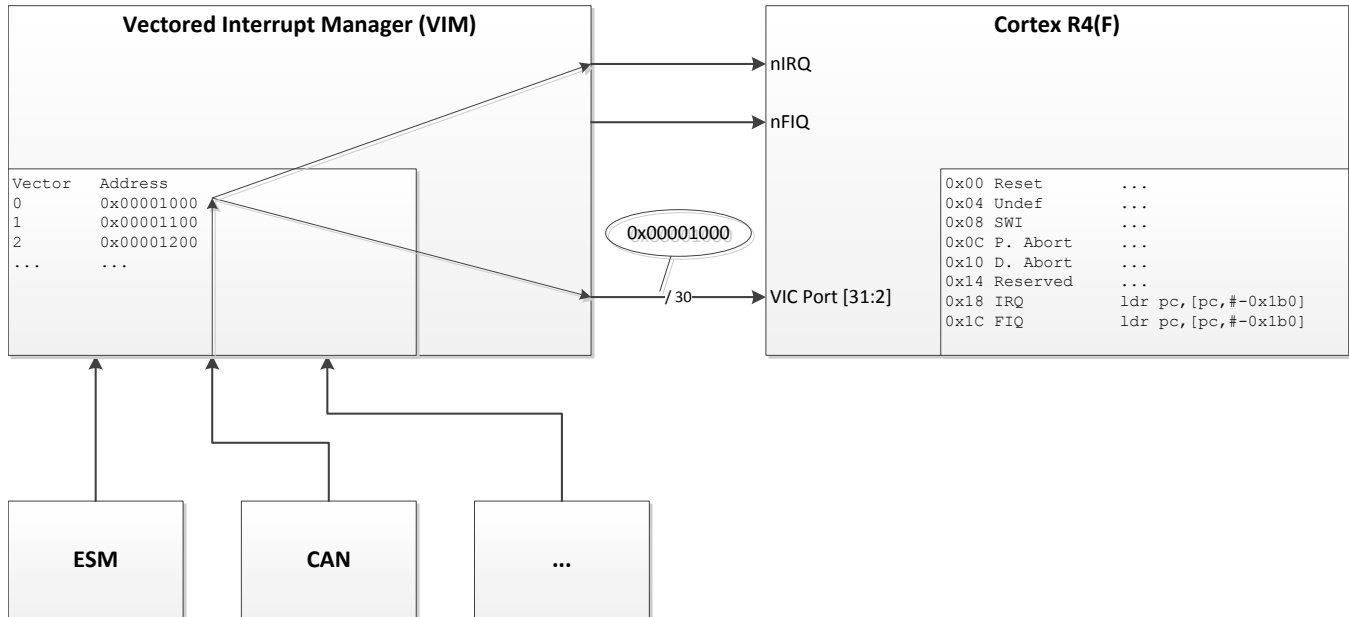


Figure 5. Hardware Vectored Interrupt Flow

1. Request IRQ only.
2. CPU reads IRQ vector address instead of 0x18.
3. VIM provides address of highest pending request directly to the processors VIC port.
4. CPU branches directly to ISR.



**Figure 6. Hardware Vectored Interrupt Connection Scheme**

**NOTE:** The LDR instruction on address 0x18 could be a jump to a fault handler, as this instruction should never be executed when the MCU is configured correctly for Hardware Vectored Interrupts.

The LDR instruction on address 0x20 can potentially be replaced with an unconditional jump to the ESM fault handler or even with the handler itself, in case no other interrupt except the ESM high level fault interrupt is mapped to the nFIQ request line.

## 4.5 Phantom Interrupts

A phantom interrupt is an interrupt that the CPU cannot identify the source. When a phantom interrupt occurs, the CPU internal nIRQ or nFIQ request signal is low but the contents in the VIM interrupt offset register (IRQIVEC/FIQIVEC) and interrupt request register (INTREQ) stay at zero. It is undesirable because it not only adds a lot overhead in the processing but also would lead to unexpected behavior in the system operation. It could be caused by improper handling of a legitimate interrupt.

### 4.5.1 How Does it Occur?

Although malfunction of the hardware could result in a phantom interrupt, the most common cause of the phantom interrupt is the improper handling of a legitimate interrupt.

There are two typical scenarios where improper handling of a legitimate interrupt would cause a phantom interrupt. To simplify the discussion, the example will use only one peripheral interrupt enabled.

In the first scenario, the IRQ or FIQ interrupts are handled in the legacy mode where the VIM interrupt offset register (IRQIVEC/FIQIVEC) has to be read to determine the source of interrupt. If the VIM mask register (REQMASKSET/REQMASKCLR) is modified before the reading of the VIM interrupt offset register, the read back value of the VIM interrupt offset or flag registers could be incorrect and the legitimate interrupt could be falsely recognized as a phantom interrupt.

The second scenario is related to the timing between the internal interrupt control signals. It could happen when FIQ and IRQ interrupts are handled in either vectored or legacy modes.

When a peripheral interrupt occurs, the peripheral sends the interrupt request signal. This peripheral interrupt request signal sets the CPU nIRQ or nFIQ interrupt request signal if this peripheral interrupt is enabled in the VIM interrupt request mask register (REQMASKSET). In order for the peripheral interrupt to occur next time, the peripheral interrupt flag needs to be cleared in the module requesting the interrupt before exiting the interrupt service routine (ISR). The VIM register will be cleared 1-2 VBUS clock cycles after the peripheral interrupt request signal becomes inactive. The nIRQ or nFIQ interrupt request signal will become inactive about 2-4 VBUS clock cycles after the peripheral interrupt request signal is inactive.

A phantom interrupt will occur if the I and/or F bits in the CPSR register is cleared (IRQ and/or FIQ interrupts are enabled) during the time where the CPU nIRQ or nFIQ interrupt request signal is high but the VIM registers are cleared.

This scenario is commonly related to the situation where the peripheral interrupt pending flag is cleared by a write operation. Take the real time interrupt module (RTI) for example. In order to clear the interrupt flag in the RTI module, a "1" needs to be written to the proper bit location in the RTI interrupt flag register. In this case, due to the imprecise timing of the VBUS write operation the interrupt service routine (ISR) could return before the CPU nIRQ or nFIQ interrupt request signal becomes inactive. It will cause a phantom interrupt.

When the peripheral interrupt pending flag is cleared by a read operation, the read instruction will return after at least 3 VBUS cycles and the CPU nIRQ or nFIQ interrupt request signal is already low when the interrupt service routine exits. Phantom interrupts will not occur in this situation.

#### **4.5.2 How to Avoid it?**

The following interrupt handling procedures are recommended to avoid phantom interrupts. When serving an IRQ or FIQ interrupt in the legacy mode, the following steps should be followed in the ISR:

1. Read the VIM interrupt offset register (IRQIVEC/FIQIVEC) to determine the source of interrupt.
2. Clear the peripheral interrupt flag after the interrupt source is identified.
3. If the peripheral interrupt flag is cleared by a write operation, read the peripheral interrupt flag register back to ensure the proper timing. Due to the imprecise timing of the write operation, adding NOP after the write operation will not ensure the required delay.
4. Start other processing.

When the IRQ or FIQ interrupt are served in the vectored mode, the VIM hardware will automatically identify the source of the highest priority pending interrupt and the reading of the VIM interrupt offset register (IRQIVEC/FIQIVEC) is no longer needed. The following steps should be followed in the ISR when processing an interrupt in the vectored mode.

1. Clear the peripheral interrupt flag.
2. If the peripheral interrupt flag is cleared by a write operation, read the peripheral interrupt flag register back to ensure the proper timing.
3. Start other processing.

## 5 Example Code Snippets

### 5.1 Generate and Handle SVC

#### 5.1.1 Declare SVC in C Language

The TI ARM Code Generation Tools (Compiler) allows declaring special function prototypes that are translated to SuperVisor Calls. For this purpose, the `#pragma SWI_ALIAS` was introduced (see the *The SWI\_ALIAS Pragma* section in the TI ARM Compiler User's Guide [4]). The ARM/Keil tool chain supports similar functionality with the keyword `__svc()`.

The following code snippet shows how to declare SVC functions with the TI tool chain:

```
#pragma SWI_ALIAS(unimplementedSVC, 0);
#pragma SWI_ALIAS(switchCpuMode, 1);
#pragma SWI_ALIAS(switchToSystemMode, 2);
#pragma SWI_ALIAS(switchToUserMode, 3);
#pragma SWI_ALIAS(writePrivRegister32, 33);
#pragma SWI_ALIAS(writePrivRegister8, 34);
#pragma SWI_ALIAS(testReentrantSVC, 35);

void unimplementedSVC(void); /* Used to test fault handler */
uint32_t switchCpuMode(uint32_t u32ModeNum);
void switchToSystemMode(void);
void switchToUserMode(void);

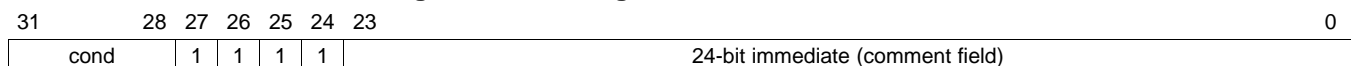
/* The following SVC functions are implemented at C level */
void writePrivRegister32(uint32_t * pu32Address, uint32_t u32Value);
void writePrivRegister8(uint8_t * pu8Address, uint8_t u8Value);
void testReentrantSVC(uint32_t * pu32Address, uint32_t u32Repetition);
```

Other tools have similar ways to declare aliases for SuperVisor Calls. The ARM tool chain for example uses the `__svc(#imm)` keyword.

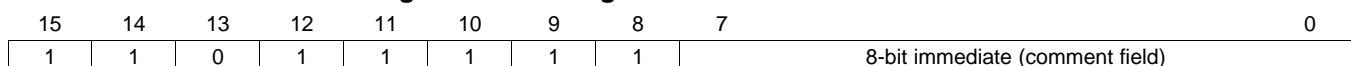
#### 5.1.2 Handle SVC in Assembly

The SVC handler itself has to be implemented in assembly to extract the 8/24-bit value from the SVC instruction (Thumb 8 bit, ARM 24 bit). [Figure 7](#) and [Figure 8](#) show the encoding of the SVC instruction in both ARM and Thumb mode.

**Figure 7. Encoding of SVC Instruction in ARM Mode**



**Figure 8. Encoding of SVC Instruction in Thumb Mode**



1. Figure out the instruction set mode from which the SVC handler was called from. This can either be the ARM mode or the Thumb mode, depending on this we have to decide which part of the instruction we have to load and mask to get the immediate value which tells us the service being requested.

```
MRS    A4, SPSR           ; Get Saved Program Status Register
TST    A4, #0x20          ; Called in Thumb state?
```

2. Load the immediate value from the SVC instruction.

```
LDRNEH A3, [lr,#-2]      ; Yes: Load halfword and...
BICNE  A3, A3, #0xFF00   ; ...extract comment field
LDREQ  A3, [lr,#-4]      ; No: Load word and...
BICEQ  A3, A3, #0xFF000000 ; ...extract comment field
```

3. Register A3 (R2) holds now the immediate value which tells us the service being requested. This can be used along with a jump table coded in assembly to dispatch the service or be forwarded to a C level SVC handler and evaluated there.

More details on how to write a SVC handler can be found in the *SVC handlers* section of the *Developing Software for ARM Processors* handbook [3]. There is also an example CCS project available from the project collateral, where the code snippets in this document are copied from.

In the example project the first SVC handler is intended to test for an unimplemented SVC handler. Handlers 1 to 31 are intended to be implemented in assembly. Handlers 32 to 255 are intended to be implemented in C language. Handlers 1, 2, 3, 33, 34 and 35 implemented, handler 35 is used to verify that reentrant SVC handlers are functional.

The C level handler supports for 2 function parameters and a return value, the third function parameter is used to pass the SVC handler number from the assembly handler to the C level handler.

## 5.2 Interrupt Dispatcher for Legacy ARM7 Interrupts Mode

### 5.2.1 Assembly Handler

The following code snippet can be used to dispatch IRQ interrupts because FIQ interrupt, the pointer to the IRQINDEX register (0xFFFFE03), has to be changed to the FIQINDEX register (0xFFFFE07).

```
-----
; IRQ Dispatcher

    .import PHANTOM
    .import _ISR1
    .import _ISR2
    .import _ISR96

    .global    _irq
    .text
    .arm
    .armfunc    _irq

    .align 4
_irq:
    .asmfunc

    MOV  R8, #0xFE03
    MOVT R8, #0xFFFF
    LDRB R8, [R8] ; FIQ INTERRUPT ENTRY
    ; R8 used to get the FIQ index
```

```

; with address pointer to the
; first FIQ banked register

ldr PC, [PC, R8, LSL#2] ; Branch to the indexed interrupt
; routine. The prefetch
; operation causes the PC to be 2
; words (8 bytes) ahead of the
; current instruction, so
; pointing to _INT_TABLE.

nop ; Required due to pipeline.

;=====
_INT_TABLE ; FIQ INTERRUPT DISPATCH
;=====

.word PHANTOM ; beginning of FIQ Dispatch
.word _ISR1 ; dispatch to interrupt routine 1
.word _ISR2 ; dispatch to interrupt routine 2
;...
.word _ISR96 ; dispatch to interrupt routine 96

.endasmfunc

```

## 5.2.2 C Handler

An interrupt dispatcher can also be efficiently written in C with the help of function pointers. Note that the interrupt handler table itself is not shown here but in the project collateral.

```

#pragma INTERRUPT(fiqDispatch, FIQ);
#pragma RETAIN(fiqDispatch);

void fiqDispatch(void)
{
    uint32_t u32FiqIndex = vimREG->FIQINDEX;

    if (0ul == u32FiqIndex)
    {
        /* TODO: Add Fault Handler for Phantom Interrupt */

        phantomInterrupt();
    }
    else if (VIM_CHANNELS < u32FiqIndex)
    {
        /* TODO: Add fault handler, as index has exceeded max interrupts. */

        phantomInterrupt();
    }
    else
    {
        /* Read FIQ Interrupt Vector */
        t_isrFuncPTR fiq_func_pnt = ISR_TABLE[u32FiqIndex];

        (*fiq_func_pnt)(); /* Execute interrupt routine */
    }

    return;
}

```

### 5.3 Load Instructions for Vectored Interrupts Mode

The following two load instructions have to be placed in the interrupt vector table at addresses 0x18 for IRQ and 0x1C for FIQ.

```
ldr pc,[pc,#-0x1b0]
ldr pc,[pc,#-0x1b0]
```

The intention is to load the content of the IRQ/FIQ Interrupt Vector Registers from the VIM into the PC. The Registers are placed at addresses 0xFFFFFE70 and 0xFFFFFE74. As it is not possible to code these addresses directly in a single load instruction, instead a PC relative load should be used. The two instructions load from addresses (PC + 8) - 0x1B0, which calculates to the following addresses:

- $(0x18 + 8) - 0x1B0 = 0xFFFFFE70$
- $(0x1C + 8) - 0x1B0 = 0xFFFFFE74$

With this the highest priority pending interrupt vector address gets read from the VIM and loaded into the PC.

---

**NOTE:** In ARM state, the value of the PC is the address of the current instruction plus 8 bytes.

In Thumb state:

- For B, BL, CBNZ, and CBZ instructions, the value of the PC is the address of the current instruction plus 4 bytes.
  - For all other instructions that use labels, the value of the PC is the address of the current instruction plus 4 bytes, with bit[1] of the result cleared to 0 to make it word-aligned.
- 

## 6 References

1. *ARM Cortex-R4 and Cortex-R4F Technical Reference Manual*: (ARM DDI 0363C): available at [http://infocenter.arm.com/help/topic/com.arm.doc.ddi0363e/DDI0363E\\_cortexr4\\_r1p3\\_trm.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0363e/DDI0363E_cortexr4_r1p3_trm.pdf)
2. *ARM Cortex-R5 and Cortex-R5F Technical Reference Manual* (ARM DDI 0460D): available at [http://infocenter.arm.com/help/topic/com.arm.doc.ddi0460d/DDI0460D\\_cortex\\_r5\\_r1p2\\_trm.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0460d/DDI0460D_cortex_r5_r1p2_trm.pdf)
3. *ARM Compiler Tool Chain Developing Software for ARM Processors* (ARM DUI 0471): available at [http://infocenter.arm.com/help/topic/com.arm.doc.dui0471c/DUI0471C\\_developing\\_for\\_arm\\_processors.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.dui0471c/DUI0471C_developing_for_arm_processors.pdf)
4. *ARM Optimizing C/C++ Compiler v5.2 User's Guide* ([SPNU151](#))



## IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

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

### Products

|                              |  |
|------------------------------|--|
| Audio                        | <a href="http://www.ti.com/audio">www.ti.com/audio</a>                               |
| Amplifiers                   | <a href="http://amplifier.ti.com">amplifier.ti.com</a>                               |
| Data Converters              | <a href="http://dataconverter.ti.com">dataconverter.ti.com</a>                       |
| DLP® Products                | <a href="http://www.dlp.com">www.dlp.com</a>   |
| DSP                          | <a href="http://dsp.ti.com">dsp.ti.com</a>   |
| Clocks and Timers            | <a href="http://www.ti.com/clocks">www.ti.com/clocks</a>                             |
| Interface                    | <a href="http://interface.ti.com">interface.ti.com</a>                               |
| Logic                        | <a href="http://logic.ti.com">logic.ti.com</a>                                       |
| Power Mgmt                   | <a href="http://power.ti.com">power.ti.com</a>                                       |
| Microcontrollers             | <a href="http://microcontroller.ti.com">microcontroller.ti.com</a>                   |
| RFID                         | <a href="http://www.ti-rfid.com">www.ti-rfid.com</a>                                 |
| OMAP Applications Processors | <a href="http://www.ti.com/omap">www.ti.com/omap</a>                                 |
| Wireless Connectivity        | <a href="http://www.ti.com/wirelessconnectivity">www.ti.com/wirelessconnectivity</a> |

### Applications

|                               |  |
|-------------------------------|--|
| Automotive and Transportation | <a href="http://www.ti.com/automotive">www.ti.com/automotive</a>                         |
| Communications and Telecom    | <a href="http://www.ti.com/communications">www.ti.com/communications</a>                 |
| Computers and Peripherals     | <a href="http://www.ti.com/computers">www.ti.com/computers</a>                           |
| Consumer Electronics          | <a href="http://www.ti.com/consumer-apps">www.ti.com/consumer-apps</a>                   |
| Energy and Lighting           | <a href="http://www.ti.com/energy">www.ti.com/energy</a>                                 |
| Industrial                    | <a href="http://www.ti.com/industrial">www.ti.com/industrial</a>                         |
| Medical                       | <a href="http://www.ti.com/medical">www.ti.com/medical</a>                               |
| Security                      | <a href="http://www.ti.com/security">www.ti.com/security</a>                             |
| Space, Avionics and Defense   | <a href="http://www.ti.com/space-avionics-defense">www.ti.com/space-avionics-defense</a> |
| Video and Imaging             | <a href="http://www.ti.com/video">www.ti.com/video</a>                                   |

### TI E2E Community

[e2e.ti.com](http://e2e.ti.com)