Usage of MPU Subregions on the TI Hercules™ ARM® Safety MCUs

Bernhard Rill

1 Introduction

A Memory Protection Unit (MPU) is a module used to modify the memory types and attributes as defined in a processor's memory ordering model. The MPU is specific to each core in the system and can only modify the memory ordering model of the CPU to which it is attached. Texas Instruments Hercules devices including a CortexR4 or a CortexM3 core have as part of the core architecture an MPU implemented with either 0, 8, or 12 regions supported. This documents provides an overview on how to use the subregions of an MPU region when 8 or 12 regions are implemented.

2 Hercules MPU Setup Considerations

The following list includes the setup procedures for the MPU.

- Via the MPU you can configure memory regions (named frames). For every region it is possible to configure memory access permissions, memory type, memory attributes, and region size individually.
- Using the MPU, a maximum of twelve different regions can be defined with different region numbers. Please refer to the device datasheet how much regions are implemented on a dedicated device.
- The higher the region number, the higher is the priority (e.g., in the case of 8 MPU regions the attributes for region 7 take highest priority, and those for region 0 take lowest priority).
- If regions are overlapping, every attribute of a lower prior region gets overwritten by the attributes of the higher prior region.
- The region frame start address needs to be a multiple of the frame region size.
- On both cores, the default background region is only valid in Supervisor mode, when enabled. Any access in User mode will cause an MPU fault when there is no appropriate section defined for this access. A background region refers to a region that matches the entire 4GB physical address map, and has a lower priority than any other region. Therefore, a background region provides the memory attributes for any memory access that does not match any of the defined memory regions.
- An MPU region (frame) is divided into eight subregions; each of the subregion has the same size.

3 Subregions Usage

In the MPU regions' size and enable registers you can disable dedicated subregions. In default when a MPU region is enabled, all the eight subregions are activated. When you write a 0x1 to one of the bits in the bitfield [15:8] of this register you can disable this subregion.

An illustration of the region size and enable register format is shown below.

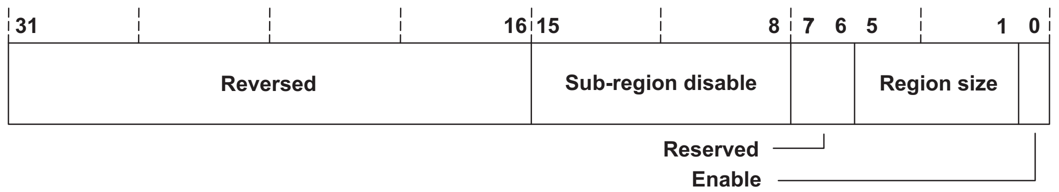

Figure 1. MPU Region Size and Enable Registers Format

Once dedicated subregions are disabled, the attributes of the next active low priority region apply.

Figure 2 provides an overview of how subregions can be used to protect certain RAM regions: the MPU region 2 (green) is configured as a background region and allows read/write access in User and in Privilege mode. Region 5 is configured with the memory attribute No Access; thus every access to this region leads to a data abort.

As region 5 (red) has a higher priority than region 2, all the attributes from region 5 succeed over the region 2 attributes. Region 5 starts at address 0x08004000 and ends at 0x08005000, so the region frame size is 0x1000 and subsequently leads to a subregion size of 0x200. In the MPU Region Size and Enable Register for Region 5, the second and third subregion are now disabled (bit 10 for the 2nd subregion and bit 11 for the 3rd subregion) by writing a ‘1.’ With this setup, the MPU memory attributes of Region 5 are NOT valid for the address space from 0x08004200 to 0x08004600 (disabled subregions 2 and 3); thus for this address space the MPU memory attributes of the next valid region apply (region 2).

Within the table on the left hand side, different accesses within the memory region are described, and in the middle of the table every reaction to the access is listed.

Background region is R/W for user and priviledged mode, testregion 5
from 0x8004000 to 0x8005000 is NO_ACCESS its subregions 2 and 3 are disabled

Memory	Reaction
test lower border of first subregion, inside subregion	data abort
test upper border of first subregion, inside subregion	data abort
test upper border of first subregion, with border overlap into second region	data abort
test lower border of second subregion, inside subregion	write possible
test upper border of second subregion, inside subregion	write possible
test upper border of second subregion, with border overlap into third region	write possible
test upper border of third subregion, inside subregion	write possible
test upper border of third subregion, with border overlap into fourth region	data abort
test lower border of fourth subregion, inside subregion	data abort

⬜ disabled subregions within testregion
🟥 testregion with NO_ACCESS attribute
🟩 background region with R/W for user and previledged mode

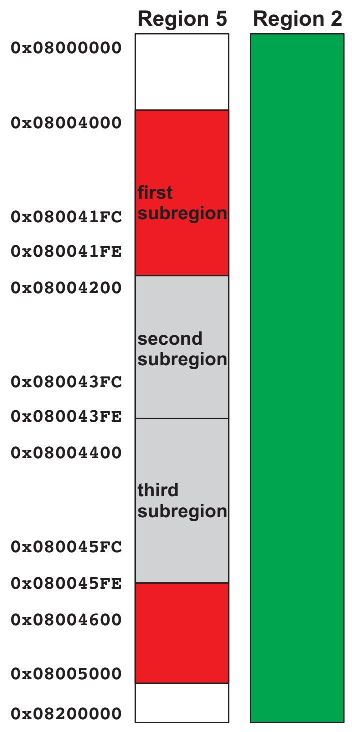

Figure 2. Memory Regions Access

The terminus border should be used in this context to describe the start or the end of a memory region.

In the above described MPU setup, two disabled subregions (2 and 3) define an address space which includes a subregion border at address 0x08004400. If there is a write which is overlapping this subregion border (e.g., a 32-bit write to address 0x080043FE) this write is possible without an issue. In case the 32-bit write is to address 0x080045FE (write which overlaps into the 4th subregion which is NOT disabled and which has the MPU memory attribute, Execute Never (XN) from region 5), the write will lead to a Data Abort.

On CortexR4 devices if there is this kind of write via a border of a subregion (lower subregion is enabled, higher subregion is disabled), the lower bytes are still written, and then an abort is generated. The lower 16 bits are still written due to the architecture of the R4 Tightly Coupled Memories (TCMs). If the test was executed on the R4 RAM, the ECC calculation for the R4 RAM is based upon 64 bits. Thus if there is a 32-bit write to an address 0x...FFFC, the write is split into two parts. The lower 64-bit which is affected by this write (starting from 0x...FFF0) calculates a new ECC value as well as the higher 64-bit (starting from 0x...0000). So because of the ECC logic there are two write accesses, and only the higher one is blocked by the MPU and leads to a data abort.

4 Programming Examples of MPU Configurations

NOTE: Please be aware that it is up to the USER to configure the MPU based upon the system requirements. The configuration examples given below just should give an indication how from programmers perspective the MPU needs to be set up.

4.1 *Programming Example Cortex™-R4*

The CortexR4 MPU has to be configured via CP 15 instructions.

```
;*****************************************************************************
;                              FUNCTION INFO
;NAME:
;    CP15_ENABLE_MPU
;
;DESCRIPTION:
;
;                     Arguments:
;                 REG - register to use as scratch pad
;
;*****************************************************************************
CP15_ENABLE_MPU:    .macro REG
                     mrc        P15, #0, REG, C1, C0, #$C1C0_SCTRL
                     orr        REG, REG, #$C1C0_SCTLR_M_MASK
                     dmb
                     mcr        P15, #0, REG, C1, C0, #$C1C0_SCTRL
                     isb
                     .endm


;*****************************************************************************
;                              FUNCTION INFO
;NAME:
;    CP15_DISABLE_MPU
;
;DESCRIPTION:
;
;                     Arguments:
;                   REG - register to use as scratch pad
;
;*****************************************************************************

CP15_DISABLE_MPU:    .macro REG
                    mrc        p15, #0, REG, C1, C0, #$C1C0_SCTRL ; read CP15
                    BIC        REG, REG, #0x1
                    DSB
                    mcr        P15, #0, REG, C1, C0, #$C1C0_SCTRL ; disable MPU
                    isb
                    .endm


;*****************************************************************************
;                              FUNCTION INFO
;NAME:
;    CP15_SET_MPU_REGION
;
;DESCRIPTION:
;
;                     Arguments:
;                     r0_NUM  - region number
;                 r1_ADDR - region base address
;                 r2_SIZE - region size and control bits
;                 r3_CTRL - region access control
;
```

```
;******************************************************************************
CP15_SET_MPU_REGION:    .macro r0_NUM,  r1_ADDR, r2_SIZE, r3_CTRL
                    mcr        P15, #0, r0_NUM,  C6, C2, #$C6C2_MPU_REGION_NUMBER
                     mcr        P15, #0, r1_ADDR, C6, C1, #$C6C1_MPU_BASE_REGION_ADDR
                    mcr        P15, #0, r2_SIZE, C6, C1, #$C6C1_MPU_SIZE_AND_ENABLE
                    mcr        P15, #0, r3_CTRL, C6, C1, #$C6C1_MPU_ACCESS_CNTRL
                    .endm


$C1C0_SCTRL             .equ 0
$C1C0_SCTLR_M_MASK         .equ (1 < < 0)


;******************************************************************************
;
;   PRIVATE VARIABLE DECLARATIONS
;
;******************************************************************************
$MEM_FRAME_SIZE             .equ $C6C1_MPU_SIZE_4G
$ITCM_FRAME_SIZE            .equ $C6C1_MPU_SIZE_2M
$DTCM_FRAME_SIZE            .equ $C6C1_MPU_SIZE_2M
$HPI_FRAME_SIZE             .equ $C6C1_MPU_SIZE_8M
$PERIPHERAL_FRAME_SIZE      .equ $C6C1_MPU_SIZE_32M

$MEM_MPU_REGION             .equ 0
$ITCM_MPU_REGION            .equ 1
$DTCM_MPU_REGION            .equ 2
$HPI_MPU_REGION             .equ 4
$PERIPHERAL_MPU_REGION      .equ 3
$ADVANCED_TEST_REGION        .equ 5

$Dtcm_Frame_Start           .long 0x08000000
$Hpi_Frame_Start            .long 0xFD000000
$Periph_Frame_Start         .long 0xFE000000



;******************************************************************************
;
;   PRIVATE DEFINITIONS
;
;******************************************************************************
; CP15 C6 MPU REGION PROGRAMMING REGISTERS
$C6C1_MPU_BASE_REGION_ADDR    .equ 0
$C6C1_MPU_SIZE_AND_ENABLE     .equ 2
$C6C1_MPU_ACCESS_CNTRL        .equ 4
$C6C2_MPU_REGION_NUMBER       .equ 0

$MPU_NUM_OF_REGIONS           .equ 8

; CP15 C6 MPU PROGRAMMING DEFINITIONS
; C6C1_MPU_SIZE_AND_ENABLE:
; MPU sub-region enable control
$C6C1_MPU_EN_ALL_SUBREG          .equ 0
$C6C1_MPU_DIS_ALL_SUBREG      .equ 0x0000F0
;Here an individual configuration of enabled/disabled subregions can be added

; MPU region esizes
$C6C1_MPU_SIZE_32             .equ (00100B << 1)
$C6C1_MPU_SIZE_64             .equ (00101B << 1)
$C6C1_MPU_SIZE_128            .equ (00110B << 1)
$C6C1_MPU_SIZE_256            .equ (00111B << 1)
$C6C1_MPU_SIZE_512            .equ (01000B << 1)
$C6C1_MPU_SIZE_1K             .equ (01001B << 1)
$C6C1_MPU_SIZE_2K             .equ (01010B << 1)
$C6C1_MPU_SIZE_4K             .equ (01011B << 1)
```

```
$C6C1_MPU_SIZE_8K              .equ (01100B << 1)
$C6C1_MPU_SIZE_16K             .equ (01101B << 1)
$C6C1_MPU_SIZE_32K             .equ (01110B << 1)
$C6C1_MPU_SIZE_64K             .equ (01111B << 1)
$C6C1_MPU_SIZE_128K            .equ (10000B << 1)
$C6C1_MPU_SIZE_256K            .equ (10001B << 1)
$C6C1_MPU_SIZE_512K            .equ (10010B << 1)
$C6C1_MPU_SIZE_1M              .equ (10011B << 1)
$C6C1_MPU_SIZE_2M              .equ (10100B << 1)
$C6C1_MPU_SIZE_4M              .equ (10101B << 1)
$C6C1_MPU_SIZE_8M              .equ (10110B << 1)
$C6C1_MPU_SIZE_16M             .equ (10111B << 1)
$C6C1_MPU_SIZE_32M             .equ (11000B << 1)
$C6C1_MPU_SIZE_64M             .equ (11001B << 1)
$C6C1_MPU_SIZE_128M            .equ (11010B << 1)
$C6C1_MPU_SIZE_256M            .equ (11011B << 1)
$C6C1_MPU_SIZE_512M            .equ (11100B << 1)
$C6C1_MPU_SIZE_1G              .equ (11101B << 1)
$C6C1_MPU_SIZE_2G              .equ (11110B << 1)
$C6C1_MPU_SIZE_4G              .equ (11111B << 1)
; MPU region enable control
$C6C1_MPU_EN_REG               .equ 1
$C6C1_MPU_DIS_REG              .equ 0

; C6C1_MPU_ACCESS_CNTRL
; XN: Execution control

$C6C1_MPU_EXE_ALWAYS           .equ 0
$C6C1_MPU_EXE_NEVER            .equ (1 << 12)
; AP: Privileged/User Read/Write control
$C6C1_MPU_NO_ACCESS            .equ (000B << 8)
$C6C1_MPU_RP_WP                .equ (001B << 8)
$C6C1_MPU_RPU_WP               .equ (010B << 8)
$C6C1_MPU_RPU_WPU              .equ (011B << 8)
$C6C1_MPU_RP                   .equ (101B << 8)
$C6C1_MPU_RPU                  .equ (110B << 8)
; S: Sharability
$C6C1_MPU_NON_SHARED           .equ 0
$C6C1_MPU_SHARED               .equ (1 << 2)
; C: Cacheability
$C6C1_MPU_NON_CACHED           .equ 0
$C6C1_MPU_CACHED               .equ (1 << 1)
; B: Bufferability
$C6C1_MPU_NON_BUFFERED         .equ 0
$C6C1_MPU_BUFFERED             .equ 1

; The Region Access Control Registers use five bits to encode the memory
; region type.These are TEX[2:0], and the C and B bits.
; Additionally, the Region Access Control Registers contain
; the shared bit, S. This bit only applies to Normal memory, and determines
; if the memory region is Shared (1), or Non-Shared (0).

; Normal, non-cachable, non-shared
$C6C1_MPU_NORMAL               .equ ((001B << 3) | ($C6C1_MPU_NON_CACHED | $C6C1_MPU_NON_BUFFERED
| $C6C1_MPU_NON_SHARED))
; Device, non-cachable, non-shared
$C6C1_MPU_DEVICE               .equ ((000B << 3) | ($C6C1_MPU_NON_CACHED | $C6C1_MPU_NON_BUFFERED
| $C6C1_MPU_NON_SHARED))


        .text
        .arm

        .global __mpuSetup_
        .asmfunc
```

```
__mpuSetup_:
        ; BOOT MPU SETUP FOR AXI
        ; ITCM frame
        MOV        R0, #$ITCM_MPU_REGION        ; Select $ITCM_MPU_REGION
        MOV        R1, #0                        ; Region 0x00000000-0x00800000
        MOVW            R2, #($ITCM_FRAME_SIZE      | $C6C1_MPU_EN_REG)
        MOVW            R3, #($C6C1_MPU_EXE_ALWAYS  | $C6C1_MPU_RPU_WPU
                                                    | $C6C1_MPU_NC_NB_NORMAL
                                                    | $C6C1_MPU_SHARED)
        CP15_SET_MPU_REGION    R0, R1, R2, R3

        ; DTCM frame
        MOV        R0, #$DTCM_MPU_REGION        ; Select $DTCM_MPU_REGION
        LDR        R1, $Dtcm_Frame_Start        ; Region 0x08000000-0x08800000
        MOVW            R2, #($DTCM_FRAME_SIZE      | $C6C1_MPU_EN_REG)
        MOVW            R3, #($C6C1_MPU_EXE_ALWAYS  | $C6C1_MPU_RPU_WPU
                                                    | $C6C1_MPU_NC_NB_NORMAL
                                                    | $C6C1_MPU_SHARED)
        CP15_SET_MPU_REGION    R0, R1, R2, R3

        ; HPI frame
        MOV        R0, #$HPI_MPU_REGION        ; Select $HPI_MPU_REGION
        LDR        R1, $Hpi_Frame_Start        ; Region 0xFD000000-0xFDFFFFFF
        MOVW            R2, #($HPI_FRAME_SIZE       | $C6C1_MPU_EN_REG)
        MOVW            R3, #($C6C1_MPU_EXE_NEVER   | $C6C1_MPU_RPU_WPU
                                                    | $C6C1_MPU_S_B_DEVICE)
        CP15_SET_MPU_REGION    R0, R1, R2, R3


        ; R4 CRC frame + Peripheral frame
        MOV        R0, #$PERIPHERAL_MPU_REGION  ; Select $PERIPHERAL_MPU_REGION
        LDR        R1, $Periph_Frame_Start         ; Region 0xFE000000-0xFFFFFFFF
        MOVW            R2, #($PERIPHERAL_FRAME_SIZE | $C6C1_MPU_EN_REG)
        MOVW            R3, #($C6C1_MPU_EXE_NEVER    | $C6C1_MPU_RPU_WPU
                                                     | $C6C1_MPU_S_B_DEVICE)
        CP15_SET_MPU_REGION    R0, R1, R2, R3

        ; Enable MPU
        CP15_ENABLE_MPU     r0
        bx      lr

.endasmfunc




        .global __mpu_advanced_Setup_
        .asmfunc

__mpu_advanced_Setup_:

        ; Disable MPU
        CP15_DISABLE_MPU     r4


        ; ADVANCED_TEST_REGION frame
        CP15_SET_MPU_REGION r0, r1, r2, r3


        ; Enable MPU
        CP15_ENABLE_MPU     r0
        bx      lr

        .endasmfunc
```

4.2 Programming Example CortexM3

The CortexM3 MPU can be configured via memory mapped registers:

```
...

struct m3core
{
    unsigned int MASTERCTRL;      /* 0x0000        */
    unsigned int INTCONTRTYPE;    /* 0x0004        */
    unsigned int : 32;            /* 0x0008        */
    unsigned int : 32;            /* 0x000C        */
    unsigned int SYSTICKCTRLSR;   /* 0x0010        */
    unsigned int SYSTICKRELOAD;   /* 0x0014        */
    unsigned int SYSTICKCURRENT;  /* 0x0018        */
    unsigned int SYSTICKCAL;      /* 0x001C        */
    unsigned int Reserved_0[56];  /* 0x0020-0x00FF */
    unsigned int SETENA[8];       /* 0x0100-0x011F */
    unsigned int Reserved_1[24];  /* 0x0120-0x017F */
    unsigned int CLRENA[8];       /* 0x0180-0x019F */
    unsigned int Reserved_2[24];  /* 0x01A0-0x01FF */
    unsigned int SETPEND[8];      /* 0x0200-0x021F */
    unsigned int Reserved_3[24];  /* 0x0220-0x027F */
    unsigned int CLRPEND[8];      /* 0x0280-0x029F */
    unsigned int Reserved_4[24];  /* 0x02A0-0x02FF */
    unsigned int ACTIVE[8];       /* 0x0300-0x031F */
    unsigned int Reserved_5[24];  /* 0x0320-0x037F */
    unsigned int Reserved_6[32];  /* 0x0380-0x03FF */
    unsigned int INTPRIO[61];     /* 0x0400-0x07EF */
    unsigned int Reserved_7[515]; /* 0x07F0-0x0CFF */
    unsigned int CPUID;           /* 0x0D00        */
    unsigned int INTCTRL;         /* 0x0D04        */
    unsigned int VTBLOFF;         /* 0x0D08        */
    unsigned int AIRCR;           /* 0x0D0C        */
    unsigned int SYSCTRL;         /* 0x0D10        */
    unsigned int CONFCTRL;        /* 0x0D14        */
    unsigned int SYSPRIO[3];      /* 0x0D18-0x0D23 */
    unsigned int SYSHCTRL;        /* 0x0D24        */
    unsigned int CONFFAULTSR;     /* 0x0D28        */
    unsigned int HARDFAULTSR;     /* 0x0D2C        */
    unsigned int DEBUGFAULTSR;    /* 0x0D30        */
    unsigned int MEMMANAGEADDR;   /* 0x0D34        */
    unsigned int BUSFAULTADDR;    /* 0x0D38        */
    unsigned int AUXFAULTADDR;    /* 0x0D3C        */
    unsigned int PROCFEATURE0;    /* 0x0D40        */
    unsigned int PROCFEATURE1;    /* 0x0D44        */
    unsigned int DEBUGFEATURE;    /* 0x0D48        */
    unsigned int AUXFEATURE;      /* 0x0D4C        */
    unsigned int MEMMODFEATURE0;  /* 0x0D50        */
    unsigned int MEMMODFEATURE1;  /* 0x0D54        */
    unsigned int MEMMODFEATURE2;  /* 0x0D58        */
    unsigned int MEMMODFEATURE3;  /* 0x0D5C        */
    unsigned int ISAFEATURE0;     /* 0x0D60        */
    unsigned int ISAFEATURE1;     /* 0x0D64        */
    unsigned int ISAFEATURE2;     /* 0x0D68        */
    unsigned int ISAFEATURE3;     /* 0x0D6C        */
    unsigned int ISAFEATURE4;     /* 0x0D70        */
    unsigned int ISAFEATURE5;     /* 0x0D74        */
    unsigned int : 32;            /* 0x0D78        */
    unsigned int : 32;            /* 0x0D7C        */
    unsigned int : 32;            /* 0x0D80        */
    unsigned int : 32;            /* 0x0D84        */
    unsigned int COPROCACESS;     /* 0x0D88        */
    unsigned int : 32;            /* 0x0D8C        */
    unsigned int MPUTR;           /* 0x0D90        */
    unsigned int MPUCR;           /* 0x0D94        */
```

```
        unsigned int MPURNR;          /* 0x0D98        */
        unsigned int MPURBAR0;        /* 0x0D9C        */
        unsigned int MPURASR0;        /* 0x0DA0        */
        /* MPURASR0 MPU Region Attribute and Size Register */
        /* bits [8:15] can be used to disable certain subregions /*
        unsigned int MPURBAR1;        /* 0x0DA4        */
        unsigned int MPURASR1;        /* 0x0DA8        */
        unsigned int MPURBAR2;        /* 0x0DAC        */
        unsigned int MPURASR2;        /* 0x0DB0        */
        unsigned int MPURBAR3;        /* 0x0DB4        */
        unsigned int MPURASR3;        /* 0x0DB8        */
        unsigned int Reserved_8[15];  /* 0x0DBC-0x0DEF */
        unsigned int DHCSR;           /* 0x0DF0        */
        unsigned int DCRSR;           /* 0x0DF4        */
        unsigned int DCRDR;           /* 0x0DF8        */
        unsigned int DEMCR;           /* 0x0DFC        */
        unsigned int Reserved_9[64];  /* 0x0E00-0x0EFF */
        unsigned int SWTRIGINT;       /* 0x0F00        */
        unsigned int Reserved_10[51]; /* 0x0F04-0x0FCF */
        unsigned int PERID[8];        /* 0x0FD0-0x0FEF */
        unsigned int CID[4];          /* 0x0FF0-0x0FFF */
    };

    #define M3CORE ((volatile struct m3core *)0xE000E000)
    ...

        /* Configuring of 3 MPU regions */
        /* MPU  Region 0 – Flash, Device */
        /* Start address 0x00000000, size 128MB, PRV_RW, USR_RW, shared, buffered */
        M3CORE->MPURBAR0 = 0x00000010;
        M3CORE->MPURASR0 = 0x03050035;

        /* MPU  Region 1 – RAM, Device */
        /* Start address 0x08000000, size 8MB, PRV_RW, USR_RW, shared, buffered */
        M3CORE->MPURBAR0 = 0x08000011;
        M3CORE->MPURASR0 = 0x0305002D;

        /* MPU  Region 2 – System and Peripheral Frame, Strongly ordered */
        /* Start address 0xFE000000, size 32MB, PRV_RW, USR_RW, non- shared, */
        /* non-buffered */
        M3CORE->MPURBAR0 = 0xFE000012;
        M3CORE->MPURASR0 = 0x03000031;

        /* Enable MPU */
        asm(" DMB");
        M3CORE->MPUCR = 0x00000001;
        asm(" ISB");
    ...
```

5 Reference Documentation

- ARMv7-R Architecture Reference Manual (DDI0406B)
- ARMv7-M Architecture Reference Manual (DDI0403B)

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