AND9547/D

Multi-Core Configuration of LC823450 Series for Audio Applications

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE

Introduction

This application note describes multi-core configuration for the implementation of multi-core RTOS (Real Time Operating System).

At first, H/W requirements for the implementation of multi-core RTOS are described, and then S/W requirements are described.

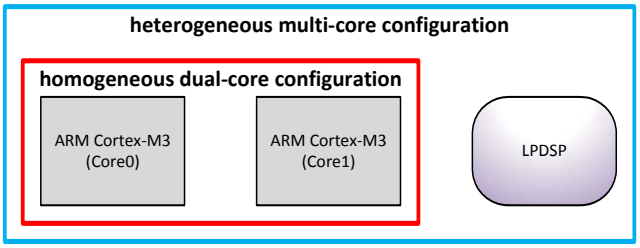
Intended audience is customers who are building audio application using LC823450 Series (called LC823450 hereafter).

BACKGROUND

Implemented cores

LC823450 has two ARM® Cortex®-M3 (Core0 and Core1) and one original-DSP called LPDSP. LPDSP is used for audio processing, and its instruction set is different from Cortex-M3's one. Therefore, the combination of two Cortex-M3 and LPDSP is categorized to heterogeneous multi-core as described in the blue frame of Figure 1. On the other hand, combination of two Cortex-M3 is categorized to homogeneous dual-core as described in the red frame of Figure 1.

Figure 1. Core configuration of LC823450



HARDWARE REQUIREMENTS

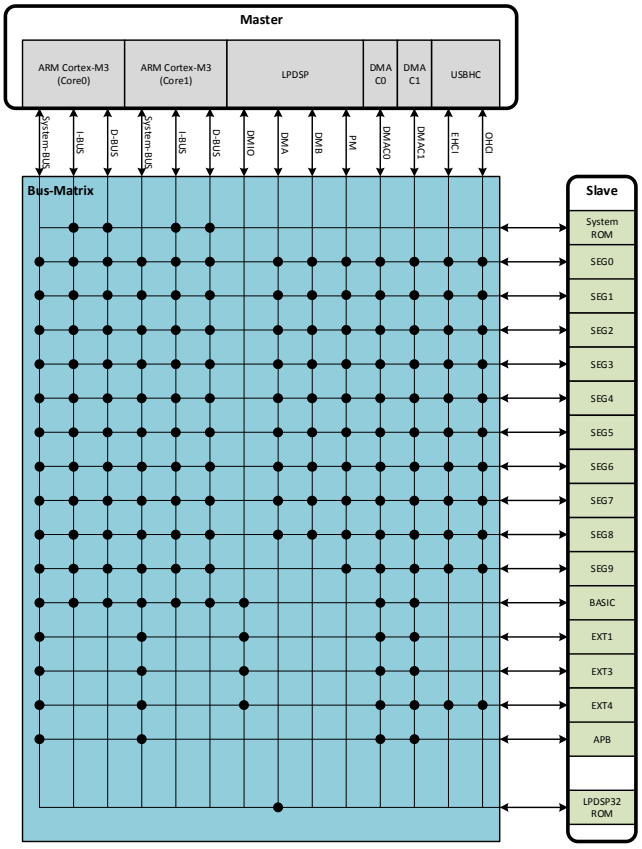
Memory access from cores

LC823450 has multiple bus-masters including two Cortex-M3 and multiple bus-slaves as described in Figure 2. These bus-masters access to bus-slaves via bus-matrix shown by ● as described in Figure 2.

Cortex-M3 (core0 and core1) can access to all bus-slaves except LPDSP32-ROM. Access latency to target bus-slave from Cortex-M3 core0 is the same from Cortex-M3 core1 if Cortex-M3 core0 and Cortex-M3 core1 don't access to same bus-slave.

Via bus-matrix, multiple bus-masters can access to different bus-slaves at the same time. Only when multiple bus-masters access to same bus-slave at the same time, bus-matrix performs bus arbitration.

Figure 2. Bus-matrix of LC823450



Note about bus arbitration:

- PM, DMA and DMB access from LPDSP are top priority and always preferred.
- Access from master except LPDSP is usually scheduled in round-robin arbitration.

Mutex

Mutual exclusion is called mutex. It is used to perform exclusive control to critical section. By using it, it is possible to maintain the consistency of data. LC823450 has 16 mutex registers.

Atomic instruction

Cortex-M3 supports bit-band instruction. This instruction performs atomic read-modify-write operation and can access exclusively to memory. By using it, it is possible to maintain the consistency of data.

LPDSP doesn't support atomic instruction.

Unique id

Cortex-M3 core0 and core1 have unique CPUID placed in address 0xE00F_E000 respectively. As this address is accessed through the Private Peripheral Bus (PPB), each core can access to unique register.

CPUID for Cortex-M3 core0 indicates 0 ("0" is allocated in 0xE00F_E000 for Cortex-M3 core0).

CPUID for Cortex-M3 core1 indicates 1("1" is allocated in 0xE00F_E000 for Cortex-M3 core1)..

When unique program code for each core is included in common program code for Cortex-M3 core0 and core1, each core identifies CPUID on boot and then performs unique program code respectively.

Interrupt among each core

LC823450 has 12 interrupts among each core as described in Figure 3. Interrupts to Cortex-M3 (core0 and core1) are connected via NVIC. Interrupts to LPDSP are connected via SELECTOR.

INTISR0_0, INTISR0_1, INTISR0_2 and INTISR0_3 are used to generate interrupts from Cortex-M3 core0. When Cortex-M3 core0 sets "1" in INTISR0_0, interrupt from Cortex-M3 core0 is generated to Cortex-M3 core0, core1 and LPDSP. NTISR0_1, INTISR0_2 and INTISR0_3 are also same function as INTISR0_0.

INTISR1_0, INTISR1_1, INTISR1_2 and INTISR1_3 are used to generate interrupts from Cortex-M3 core1. When Cortex-M3 core1 sets "1" in INTISR1_0, interrupt from Cortex-M3 core1 is generated to Cortex-M3 core0, core1 and LPDSP. NTISR1_1, INTISR1_2 and INTISR1_3 are also same function as INTISR1_0.

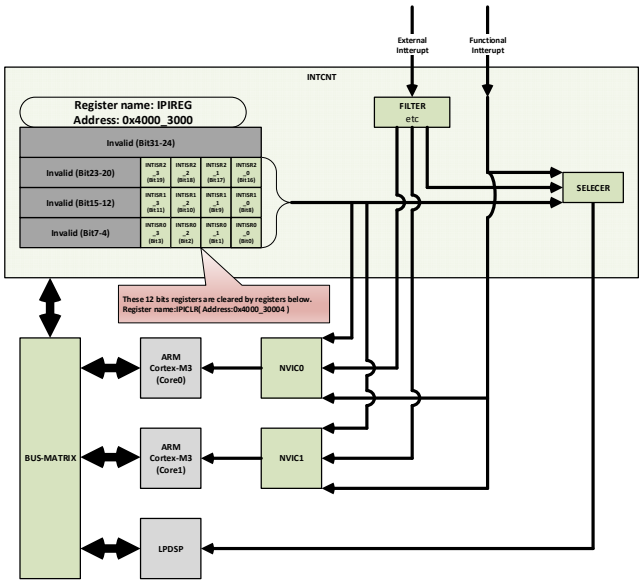
INTISR2_0, INTISR2_1, INTISR2_2 and INTISR2_3 are used to generate interrupts from LPDSP. When LPDSP sets "1" in INTISR2_0, interrupt from LPDSP is generated to Cortex-M3 core0, core1 and LPDSP. NTISR2_1, INTISR2_2 and INTISR2_3 are also same function as INTISR2_0.

Interrupt from peripherals

External interrupts and Functional interrupts are provided to Cortex-M3 (core0 and core1) and LPDSP as described in Figure 3.

External interrupts mean interrupt by GPIO. And Functional interrupts mean interrupt from each function block. External interrupts are provided via FILTER for noise countermeasure.

Figure 3. Interrupt of LC823450



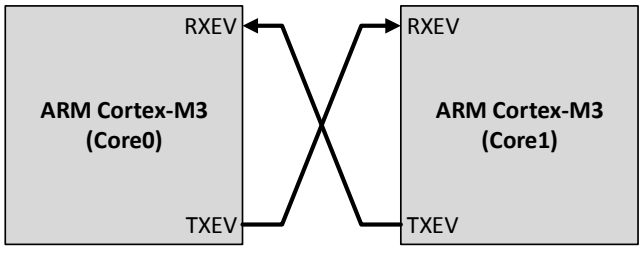
Event communication between Cortex-M3 core0 and Cortex-M3 core1

Figure 4 shows the image of event communication between Cortex-M3 core0 and Cortex-M3 core1. TXEV output of Cortex-M3 Core0 is connected to RXEV input of Cortex-M3 Core1. TXEV output of Core1 is connected to RXEV input of Core0.

Both cores can synchronize their tasks by entering into sleep state with WFE command, and by issuing an event to release the sleep state with SEV command.

WFE command behavior is affected by the event latch in core. If the event latch is not set, it will enter into the sleep state. If the event latch is set, the latch is cleared and command execution does not enter into the sleep state and continues on. Please refer to ARM document such as Cortex-M3 WFE command for more information about the event latch.

Figure 4. Image of event communication



Clock and reset

Common clock is provided to Cortex-M3 core0, core1 and LPDSP as described in Figure 5.

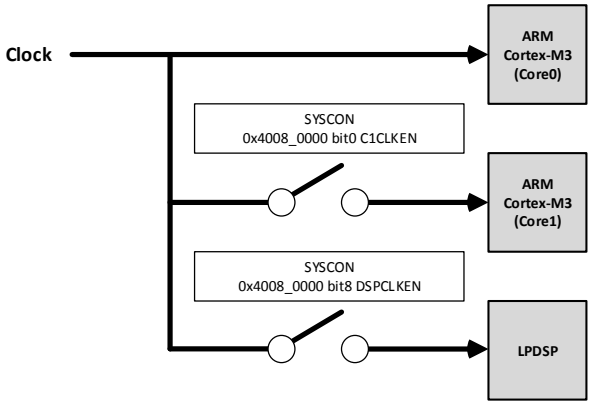
Clock for Cortex-M3 core1 is controlled by register C1CLKEN (0x4008_0000 bit0).

Clock for LPDSP is controlled by register DSPCLKEN (0x4008_0000 bit8).

Also reset for Cortex-M3 core1 is controlled by register C1RSTN (0x4008_0000 bit1) and reset for LPDSP is controlled by register DSPRSTN (0x4008_0000 bit9).

On boot, clocks for Cortex-M3 core1 and LPDSP are stopped and Cortex-M3 core1 and LPDSP are reset. When you use Cortex-M3 core1 or LPDSP, clock is supplied, and then release the reset. Note: Supply at least 3 clock cycles before release the reset.

Figure 5. Clock tree of LC823450



SOFTWARE REQUIREMENTS

OS

We can provide OS as binary codes called TOPPERS/FMP. It conforms to uITRON4 standard profile designed as the operating system for embedded system and supports dual core system. Also we can provide the development environment including these source code by concluding NDA. It is ready for mass production, but if customers have their own OS, it is possible to use it.

Task assignment example regarding TOPPERS/FMP

When you add task, you should specify the task assignment by using CRE_TSK in configure file as described in Figure 6.

taskA written in CLASS(TCL_1) is assigned to Cortex-M3 Core0 initially. But it is possible to assign to Cortex-M3 Core1 later.

taskB written in CLASS(TCL_2) is assigned to Cortex-M3 Core1 initially. But it is possible to assign to Cortex-M3 Core0 later.

taskC written in CLASS(TCL_1_ONLY) is assigned to Cortex-M3 Core0. It is impossible to assign to Cortex-M3 Core1.

taskD written in CLASS(TCL_2_ONLY) is assigned to Cortex-M3 Core1. It is impossible to assign to Cortex-M3 Core0.

And you should write the task processing as described in Figure 7. By the way, ext tsk () is optional.

Regarding argument detail of CRE_TSK, please refer to TOPPERS/FMP manual provided by concluding NDA.

Figure 6. Example of configure file

```
This is configure file example.

CLASS(TCL_1){
  //create taskA
   CRE_TSK(NEW_TSKA, { TA_ACT, 0, taskA, 1, 1024, NULL });
   //define interrupt0
    DEF_INH(INHNO_I0, { TA_NULL, interrupt0 });
   //set interrupt0 attribute
    CFG_INT(INTNO_I0, { LOWLEVEL_DETECT, INTPRIORITY });
}

CLASS(TCL_2){
  //create taskB
   CRE_TSK(NEW_TSKB, { TA_ACT, 0, taskB, 2, 1024, NULL });
}

CLASS(TCL_1_ONLY){
  //create taskC
   CRE_TSK(NEW_TSKC, { TA_ACT, 0, taskC, 3, 1024, NULL });
}

CLASS(TCL_2_ONLY){
  //create taskD
   CRE_TSK(NEW_TSKD, { TA_ACT, 0, taskD, 4, 1024, NULL });
}
```

TOPPERS/FMP

When you add interrupt handler, you should define the interrupt handler by using DEF_INH, and set the interrupt attribute by CFG_INT in configure file as described in Figure 6.

Regarding argument detail of DEF_INH and CFG_INT, please refer to TOPPERS/FMP manual provided by concluding NDA.

And you should write the interrupt handler processing as described in Figure 6.

Figure 7. Example of C source code

```
This is C source file example.

Void taskA( intptr_t exinf ){
   //process
   ************************
   ext_tsk();
}

Void taskB( intptr_t exinf ){
   //process
   ************************
   ext_tsk();
}

Void taskC( intptr_t exinf ){
   //process
   ************************
   ext_tsk();
}

Void taskD( intptr_t exinf ){
   //process
   ************************
}

Void Interrupt0(void){
   //process
   ************************
}
```

ARM, the ARM logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere.