Chapter 7: Multi-core applications

After completing this chapter, you will understand how to create multi-core applications and how to communicate between the cores in XMC7000 devices.

Table of contents

Document conventions

Convention	Usage	Example
`Courier New`	Displays code and text commands	`CY_ISR_PROTO(MyISR);` `make build`
Italics	Displays file names and paths	*sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures	[**Enter**] or [**Ctrl**] [**C**]
Menu > Selection	Represents menu paths	**File > New Project > Clone**
Bold	Displays GUI commands, menu paths and selections, and icon names in procedures	Click the **Debugger** icon, and then click **Next**.

7.1 XMC7000 processors

The XMC7000 series devices have one or two Arm® Cortex®-M7-based CPUs (CM7) and one Cortex®-M0+-based CPU (CM0+). The CM7 CPUs have Instruction/Data cache (I-cache/D-cache) and Instruction/Data tightly-coupled Memories (ITCM/DTCM). The CPU subsystems of the XMC7000 series MCUs have bus masters for P-DMA (DW), M-DMA (DMAC), and a Crypto block.

By default, the CM0+ CPU is the primary CPU, which is responsible for initializing the system. The CM0+ enables the other CPUs (CM7_0 and CM7_1), which run the main application.

7.1.1 CM0+

The CM0+ processor is a very low gate count, highly energy-efficient processor intended for microcontroller and deeply embedded applications that require an area-optimized, low-power processor. For more information on this core, you can refer to:

https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m0-plus

7.1.2 CM7

The CM7 processor is a highly efficient, high-performance embedded processor that features low-interrupt latency and low-cost debug. It also provides backward compatibility with existing Cortex-M profile processors. The processor has an in-order, super-scalar pipeline, which means many instructions can be dual-issued, including load/load and load/store instruction pairs because of multiple memory interfaces. For more information on this core, you can refer to:

https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m7

7.2 Inter-Process Communication (IPC)

Architectures with multiple CPUs often require exclusive control, synchronization, and data passing between CPUs. The XMC7000 can use IPC for such control.

The features of IPC are as follows:

- Implements locks for mutual exclusion between processors.
- Allows sending messages between processors.
- Supports multiple channels for communication.
- Supports multiple interrupts, which can be triggered using notify or release events from the channels.

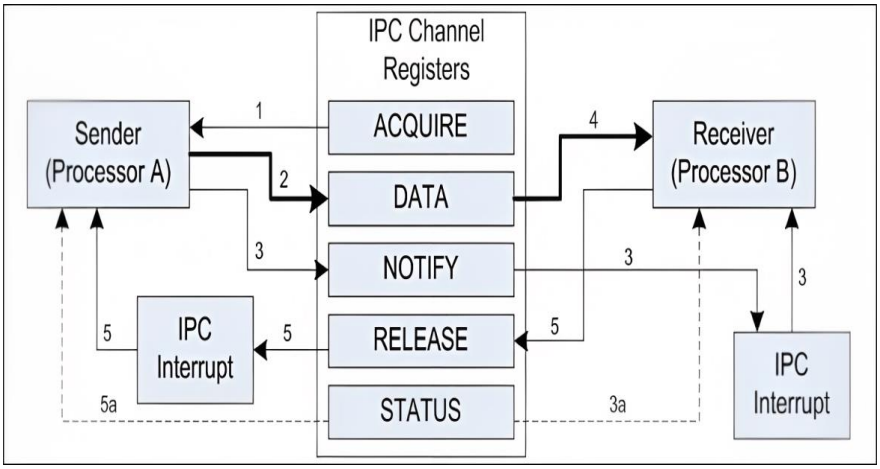
7.2.1 Operation

IPC is implemented in hardware as a collection of individual communication channels, each with a set of 32-bit registers. The IPC design implements a set of interrupts that enable each processor to notify the other that data is available, or has been processed. There is also a locking mechanism that allows only one CPU to gain access at a time.

At the hardware level, communication is a five-step process:

1. The sending processor acquires a channel.
2. The sender puts data into the channel.
3. The sender generates a notify event (interrupt).
4. The receiving processor identifies the sender and retrieves the data.
5. The receiving processor generates a release event (interrupt).

The following diagram illustrates the steps:



These transactions are handled transparently by the DRV-level API. You can also use the PIPE and SEMA layers of the API to implement communication in your application Configuration. Kindly refer to the PDL-IPC Documentation to know more about the PIPE and SEMA layers.

7.2.2 Configuration

The configuration for message passing can be done using the **IPC PDL Driver Layer**.

These APIs and their functions include:

- `Cy_SysInt_Init`: set up the IPC interrupt line
- `NVIC_EnableIRQ`: enable the interrupt
- `Cy_IPC_Drv_SetInterruptMask`: set the interrupt mask for an IPC Interrupt

  The mask configures release or acquire notification events for all IPC channels.
- `Cy_IPC_Drv_ClearInterrupt`: clear IPC interrupt
- `Cy_IPC_Drv_SendMsgWord`: send a 32-bit word message through an IPC channel
- `Cy_IPC_Drv_ReadMsgWord`: read a 32-bit word message through an IPC channel

  This function assumes that the channel is locked (for a valid message). If the channel is not locked, the message is invalid. You must call the `Cy_IPC_Drv_Release` function after reading the message to release the IPC channel.
- `Cy_IPC_Drv_LockRelease`: release an IPC channel from the locked state
- `Cy_IPC_Drv_IsLockAcquired`: test the status of an IPC channel
- `Cy_IPC_Drv_GetIpcBaseAddress`: take an IPC channel index as a parameter and return the base address the IPC registers corresponding to the IPC channel

The documentation for the PDL IPC functions can be found under **Peripheral Driver Library > PDL API Reference > IPC > IPC Driver Layer > Functions**.

In the upcoming exercises, we will see how the above shown functions work and how they are implemented for passing messages between two CPUs.

7.3 Exercises

Exercise 1: Create the "Multicore Empty App" application

This exercise uses the KIT_XMC72_EVK BSP.

1. Use the Project Creator to create a new application called **ch05_ex01_MulticoreEmptyapp** using the **Multicore_Empty_App** as the template application.

Note: *You can also find this example in the project section of the training material in the name key_ch05_ex01_MulticoreEmptyapp.*

2. Explore how the empty application is created for all three cores.

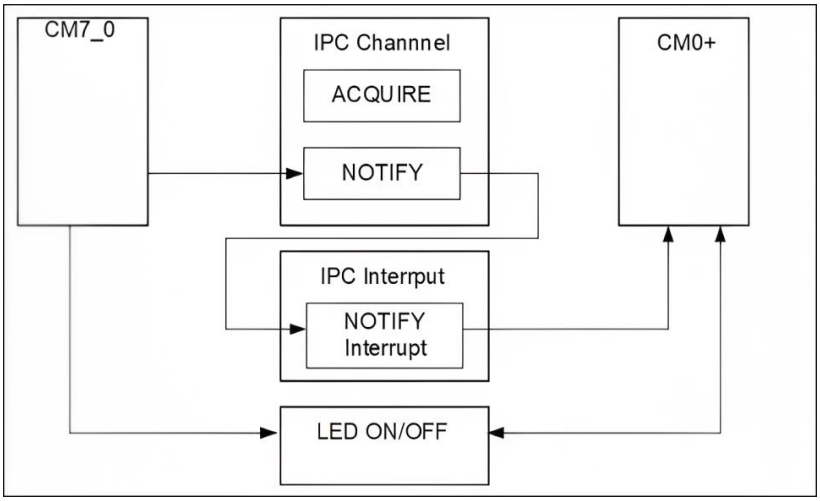
Note: *In this example, the CM0+ CPU is the primary CPU, which is responsible for initializing the system. The CM0+ enables the CM7_0 for single mode and the CM7_0 and CM7_1 for dual mode operation. Then, the CM0+ is put into deep sleep. The main application runs on the CM7 cores. The single mode and dual mode can be configured in the Makefile of the project.*

Exercise 2: Create the "Multicore IPC pipes" application

This exercise uses the KIT_XMC72_EVK BSP. In this exercise we will examine and create the Infineon XMC7000 "Multicore IPC pipes" application, which:

- Demonstrates **how to use the IPC driver to implement a message pipe** in the XMC7000 MCU.
- Shows how to use the IPC low-level interface. The low-level interface directly controls the IPC.
- Sends data from CM7_0 to CM0+, which changes LED ON/OFF with received data.

Design and implementation



- The CM7_0 core initializes the peripheral `cybsp_init()`.
- After the CM7_0 completes peripheral initialization, it force releases the lock state through the `Cy_IPC_Drv_LockRelease` function, and then waits until the CM0+ IPC server is started.
- The CM7_0 generates a notify interrupt to CM0+.
- Then, a notify interrupt occurs in CM0+.
- The CM0+ starts the operation of LED ON/OFF after returning from the interrupt routine.

Note: *All three LEDs glow sequentially by the CM7_0 core passing values such as 0, 1 and 2 to the CM0+ core through the notify interrupt and checking it in a switch case in the CM0+ core.*

Steps

1. Use the Project Creator to create a new application called **ch05_ex02_MulticoreIPCpipes** using the **Multi-core IPC pipes** as the template application.

   Note: *You can also find this example in the project section of the training material in the name key_ch05_ex02_MulticoreIPCpipes.*

2. Program and see the LEDs blinking in a serial fashion.
3. Understand the above explained flow of IPC by going through the code in *main.c* file for each core.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2022 Infineon Technologies AG.
All Rights Reserved.

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.