Application Brief

GPIO Use Cases and Implementation With the AM6x Family of Sitara™ MPU Devices

Tushar Thakur, Anil Swargam *Sitara MPU*

Introduction

This application brief explores the General Purpose Input/Output (GPIO) functionality of the AM6x family of processors, highlighting the versatility in embedded system design. The AM6x processors provide a robust GPIO interface that enables efficient control and monitoring of various peripherals and sensors. This document details key features including pin configuration, interrupt handling and resource management, alongside practical use cases demonstrating GPIO implementation in real-world applications. By providing step-by-step guidance and links to frequently-asked-questions (FAQ) with example code, this application brief serves as a valuable resource for engineers seeking to leverage GPIO capabilities in projects, providing best-in-class performance and reliability in diverse environments.

What is GPIO?

The General-Purpose Input/output (GPIO) peripheral provides dedicated general-purpose pins that can be configured as either inputs or outputs.

GPIO can be used in three modes: Input, Output, and Interrupt

- **Output**: When configured as an output, the user can write to an internal register to control the state driven on the output pin.
- **Input:** When configured as an input, the user can obtain the state of the input by reading the state of an internal register.
- **Interrupt:** When configured as interrupt or event generation mode the GPIO peripheral can produce host CPU interrupts and DMA synchronization events.

The GPIO pins are grouped into banks (16 pins per bank and 9 banks per module), which means that each GPIO module provides up to 144 dedicated general-purpose pins with input and output capabilities; thus, the general-purpose interface supports up to 432 (3 instances × (9 banks × 16 pins)) pins.

For more information on GPIO module, see the device-specific Technical Reference Manual (TRM).

GPIO Interrupt Connectivity

Due to the large number of possible GPIO interrupt sources, routing all interrupt events to each processing element is impractical. Since most applications do not typically require a large number of GPIO interrupts, the interrupt uncertainty is resolved by mapping all GPIO interrupts to a series of event MUXes implemented using Interrupt Router (`IntRouter`) modules. These MUXes allow any one of the available GPIO interrupts to be selected and passed on as an event to the various processor interrupt controllers and DMA controllers. Event selection is controlled through associated registers within each `IntRouter`.

What are Interrupt Routers?

Interrupt routers are a hardware module in the AM6x system on chip (SoC) that manage and route interrupts between various processor and peripheral components. The interrupt router is responsible for taking interrupt requests generated by hardware peripherals or other sources and then routing them to the appropriate CPU or processor core for processing.

The interrupt routers are configurable, allowing developers to specify interrupt routing configurations to optimize system performance and reduce latency. The interrupt router configuration can be controlled through Resource Management (RM) file (`sciclient_defaultBoardcfg_rm.c` file).

There are two interrupt routers associated with GPIO:

1. MAIN_GPIOMUX_INTROUTER0
2. MCU_GPIOMUX_INTROUTER0

What is a Resource Management File (`sciclient_defaultBoardcfg_rm.c`)?

The RM firmware uses the *rm_c* (`sciclient_defaultBoardcfg_rm.c`) file to initialize the system resources during boot and to manage the allocation and usage of these resources during runtime. This helps to make sure that system resources are used efficiently and that there are no conflicts between different software components that can require access to the same resources. The file is typically created by the system integrator and provided to the RM firmware as part of the boot process.

What is the Difference Between GPIO Pin Interrupt versus GPIO Bank Interrupt?

- **Pin Interrupt:** Interrupt is generated only for a single pin.
- **Bank Interrupt:** Interrupt is generated for all the pins lies in particular bank (16 pins per bank). In the interrupt service routine, check for which pin the interrupt is generated.

E2E™ FAQs

This section lists different GPIO use-case possibilities with FAQ links, where applicable:

1. **Configure multiple GPIO interrupts under a single bank interrupt.**

   Consider a scenario where you want two or more pins of same bank to generate interrupt. On the basis of the pin number that generates interrupt for the core, a certain action is performed. A limited number of routers can be configured and there are more GPIO pins than routers. Using the same router to interrupt the core is always a good idea, when multiple pins of the same GPIO bank are configured.

   Suppose GPIO4 and GPIO5 are configured as an interrupt source which lies under the same GPIO bank and two different sensors are connected to those GPIO pins. Now individual actions can be taken for each sensor based on interrupt generation.

   FAQ - https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1337322
2. **Route the same GPIO bank interrupt to two different cores.**

   Consider a scenario where a multicore image uses pins from the same GPIO bank but from different cores.

   Suppose there is a multicore appimage (that is, using the R5F0-0 and R5F0-1 core) where the GPIO_5 pin is used by R5F0-0 and the GPIO_8 pin is used by R5F0-1 core. A sensor is connected to R5F0-0 and another sensor is connected to the R5F0-1 core and each sensor has an independent ISR to process. When such a scenario exists, interrupt generated from pins lying in the same GPIO bank need to be routed to different cores.

   FAQ - https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1260818
3. **R5F GPIO interrupt example to run parallelly when running Linux® on A53.**

   Consider a scenario where Linux® is running on A53 cores and the R5F core wants to use GPIO pins.

   The problem with this use case is that resources are now managed by Linux running on the A53 core. The Processor SDK Linux and MCU+SDK use different board configurations. In Linux, resources are mostly allocated to A53 core. The user needs to allocate the resources to the R5F core and configure the system properly while using the R5F application parallelly with running Linux without having a resource conflict.

   FAQ - https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1198105

4. **Enable pin interrupt rather than bank interrupt.**

   There are 16 GPIO pins in a GPIO bank. Consider, if you want to configure GPIO_5 and GPIO_7 (lies in same bank) to control two different sensors whose working is independent of each other. In such scenarios, GPIO needs to be configured as pin interrupt to handle the ISR of each sensor individually.

   FAQ - https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1196137
5. **Change the router assignments for custom hardware.**

   There are scenarios where router assignment must be changed for the GPIO pins of the SoC. Consider an application, where a number of sensors (say, 6 sensors) are connected to a single core which are independent of each other and needs to have an independent ISR. In such a use case, configure the GPIO to work as pin interrupt but it is possible that the router assigned to the core are less in number.

   For example – If a system has 12 routers; four of them are allocated to the R5F0-0 core and the next four are allocated to an A53 core, and another 2 are allocated to the R5F0-1 and M4F core. The R5F0-0 core needs six routers for six GPIO pins to work as pin interrupt. Routers allocated to the R5F0-1 core are not used, so it can be allocated to R5F0-0. All sensors can be controlled individually by R5F0-0.

   FAQ - https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1338545
6. **Trigger DMA with the help of GPIO on AM64X, AM243, and AM62X devices.**

   Consider a scenario where a 16-bit parallel ADC needs to be connected to the SoC. An interrupt needs to be generated when the data is ready to be read by the SoC. This interrupt can be generated when the ready pin of the ADC sensor sends high signals to connected GPIO. Another similar use case is connecting an FPGA to the SoC. Once the interrupt is generated or the data is ready, the data can be read and transferred using the DMA peripheral.

   FAQ - https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1378150
7. **Route MCU GPIO pin interrupts to MAIN domain cores.**

   Consider a scenario where the GPIO pins of MAIN domain are used and there are spare MCU domain GPIO pins which are not used by the MCU domain. The designer wants to use those pins to generate interrupt for MAIN domain so that the GPIO functionality can fully be utilized.

   Check first whether or not the SoC supports cross-domain peripheral access. If support is present, then check whether or not the Application has the Reset Isolation feature enabled.

   Note

   Accessing the MCU domain peripherals from MAIN core is not allowed when reset isolation is enabled in the application.

   FAQ - https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1236579
8. **Route MAIN GPIO pin interrupts to MCU domain cores.**

   Check first whether or not the SoC supports cross-domain peripheral access. If support is present, then determine whether or not the Application has any Reset Isolation feature enabled.

   Note

   Accessing the MAIN domain peripherals from MCU core is not allowed when reset isolation is enabled in the application.

   If concerns remain, begin a new thread on TI's E2E™ forum.

   In the AM62x and AM64x family of devices, this is not possible because of SoC limitations.

Summary

This application brief presents an in-depth exploration of GPIO functionality in Sitara™ MPU devices, specifically in the AM6x series. The document outlines the critical role of GPIO in connecting and controlling external hardware components within embedded applications. The GPIO configuration process is detailed, including how to set up pins, do resource allocation, and managing interrupt handlers.

Real-world use cases illustrate how to implement GPIO for various applications such as motor control, sensor integration, and communication tasks. Additionally, it offers insights into performance optimization and troubleshooting techniques to provide robust system operation. This resource serves as a practical guide for developers and engineers looking to maximize the capabilities of GPIO in terms of efficiency and reliability for embedded system design.

References

In addition to this document, see the following references at www.ti.com.

1. Texas Instruments, AM62x Technical Reference Manual
2. Texas Instruments, AM62x Sitara™ Processors Data Sheet
3. Texas Instruments, AM62Ax Sitara Processors Technical Reference Manual
4. Texas Instruments, AM62Ax Sitara™ Processors Data Sheet
5. Texas Instruments, AM62Px Sitara Processors Technical Reference Manual
6. Texas Instruments, AM62Px Sitara™ Processors Data Sheet
7. Texas Instruments, AM64x /AM243x Processors Silicon Revision 2.0 Texas Instruments Families of Products Technical Reference Manual
8. Texas Instruments, AM64x Sitara™ Processors Data Sheet
9. Texas Instruments, Processors Academy

Trademarks