Table of Contents

Introduction

This document introduces the ARM Generic Interrupt Controller (GIC), which is included as part of the ARM Cortex-A9 MPCORE processor.

- GIC Architecture
- GIC Programmer’s Interface

Interrupt line signals

A.3 Interrupt line signals

Details for the interrupt line signals that are present in the Cortex-A9 processor.

Table A-3 Interrupt line signals

Name	I/O	Source	Description
nFIQ	I	Interrupt sources	Cortex-A9 processor FIQ request input line. Active-LOW fast interrupt request: **0** Activate fast interrupt. **1** Do not activate fast interrupt. The processor treats the **nFIQ** input as level sensitive.
nIRQ	I	Interrupt sources	Cortex-A9 processor IRQ request input line. Active-LOW interrupt request: **0** Activate interrupt. **1** Do not activate interrupt. The processor treats the **nIRQ** input as level sensitive.

ARM generic interrupt controller (GIC)

The generic interrupt controller (GIC) is a centralized resource for managing interrupts sent to the CPUs . The controller enables, disables, masks, and prioritizes the interrupt sources and sends them to the selected CPU (or CPUs) in a programmed manner as the CPU interface accepts the next interrupt.

The registers are accessed via the **CPU private bus for fast read/write response by avoiding temporary blockage or other bottlenecks in interconnect.**

The interrupt distributor centralizes all interrupt sources before dispatching the one with the highest priority to the individual CPUs. The GIC ensures that an interrupt targeted to several CPUs can only be

taken by one CPU at a time. **All interrupt sources are identified by a unique interrupt ID number.**

GIC is a part of the ARM A9 MPCORE processor. The GIC is connected to the IRQ interrupt signals of all I/O peripheral devices that are capable of generating interrupts. Most of these devices are normally external to the A9 MPCORE, and some are internal peripherals (such as timers). The GIC included with the A9 MPCORE processor in the SoC **family handles up to 255 sources of interrupts.** When a peripheral device sends its IRQ signal to the GIC, then the GIC can forward a corresponding IRQ signal to one or both of the A9 cores. Software code that is running on the A9 core can then query the GIC to determine which peripheral device caused the interrupt, and take appropriate action. The procedure for working with interrupts for the ARM Cortex-A9 and the GIC are described in the following sections.

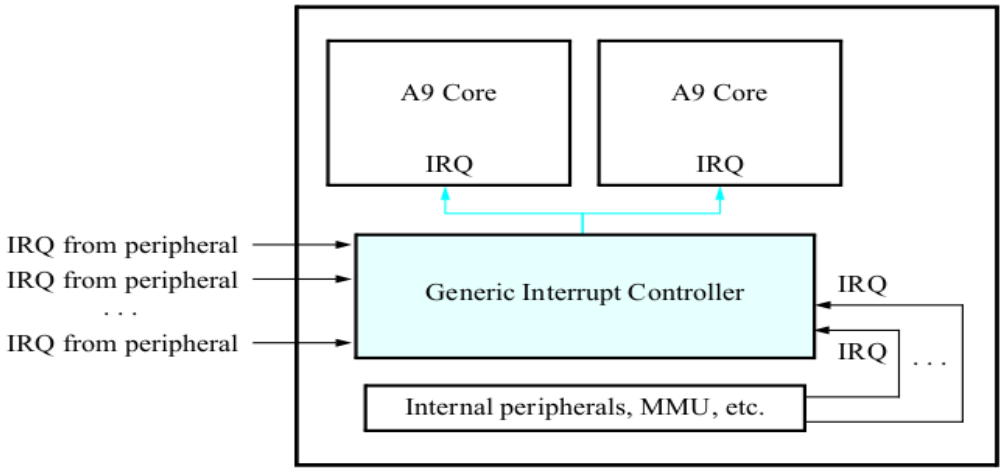

Figure 1. The ARM A9 MPCORE processor.

Programmer's Interface to the GIC

The GIC includes a number of memory-mapped registers that provide an application programmer's interface (API). As illustrated in Figure , the GIC architecture is **divided into two main parts, called the CPU Interface and the Distributor.** The CPU Interface is responsible for sending IRQ requests received by the Distributor to one or both of the A9 processors in the MPCORE. The Distributor receives IRQ interrupt signals from I/O peripherals.

GIC CPU Interface

The CPU Interface in the GIC is used to send IRQ signals to the A9 cores. There is one CPU Interface for each A9 core in the MPCORE. API registers in each CPU Interface

GIC Distributor

The Distributor in the **GIC can handle 255 sources of interrupts**. As indicated in Figure , Interrupt IDs in the range **from 32 − 255 correspond to shared peripheral interrupts (SPIs)**. These interrupts are connected to the IRQ signals of up to 224 I/O peripherals, and these sources of interrupts are common to (shared by) both CPU Interfaces. The Distributor also handles private peripherals interrupts (PPIs) for each of the A9 processors, with these interrupts using IDs in the range from 0 − 31 . The software generated interrupts (SGIs) are a special type of private interrupt that are generated by writing to a specific register in the GIC; Interrupt IDs from 0−15 are used for SGIs.

ARM Generic Interrupt Controller – Device Tree

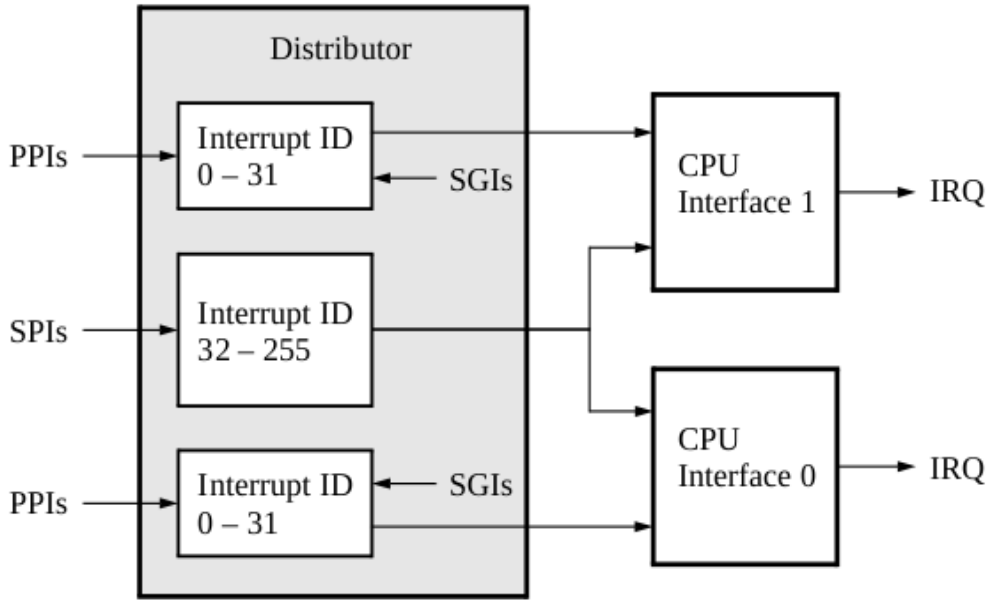

Figure 4. The GIC Architecture.

ARM SMP cores are often associated with a GIC, providing per processor generated interrupts(PPI). shared processor interrupts (SPI) and software generated interrupts (SGI).

Primary GIC is attached directly to the CPU and typically has PPIs and SGIs. Secondary GICs are

cascaded into the upward interrupt controller and do not have PPIs or SGIs.

Main node required properties:

- compatible : should be one of: **"arm,cortex-a9-gic","arm,arm11mp-gic"**
- interrupt-controller : **Identifies the node as an interrupt controller**
- #interrupt-cells : Specifies the number of cells needed to encode an interrupt source. The type shall be a <u32> and the value shall be 3.

The 1st cell is the interrupt type; 0 for SPI interrupts, 1 for PPI interrupts.
The 2nd cell contains the interrupt number for the interrupt type. SPI interrupts are in the range [0-987]. PPI interrupts are in the range [0-15].

The 3rd cell is the flags, encoded as follows:
bits[3:0] trigger type and level flags.
    1 = low-to-high edge triggered
    2 = high-to-low edge triggered
    4 = active high level-sensitive
    8 = active low level-sensitive
bits[15:8] PPI interrupt cpu mask. Each bit corresponds to each of the 8 possible cpus attached to the GIC. A bit set to '1' indicated the interrupt is wired to that CPU. Only valid for PPI interrupts.

- reg : Specifies base physical address(s) and size of the GIC registers. The first region is the GIC distributor register base and size. The 2nd region is the GIC cpu interface register base and size.

intc: interrupt-controller@fff11000 {
    compatible = "arm,cortex-a9-gic";
    #interrupt-cells = <3>;
    #address-cells = <1>;
    interrupt-controller;
    reg = <0xfff11000 0x1000>,
    <0xfff10100 0x100>;
};

Linux Kernel driver:

drivers/irqchip/irq-gic.c

GIC Distributor

#define GIC_DIST_CTRL	0x000	
#define GIC_DIST_CTR	0x004	
#define GIC_DIST_IIDR	0x008	
#define GIC_DIST_IGROUP	0x080	

#define GIC_DIST_ENABLE_SET	0x100	
#define GIC_DIST_ENABLE_CLEAR	0x180	
#define GIC_DIST_PENDING_SET	0x200	
#define GIC_DIST_PENDING_CLEAR	0x280	
#define GIC_DIST_ACTIVE_SET	0x300	
#define GIC_DIST_ACTIVE_CLEAR	0x380	
#define GIC_DIST_PRI	0x400	
#define GIC_DIST_TARGET	0x800	
#define GIC_DIST_CONFIG	0xc00	
#define GIC_DIST_SOFTINT	0xf00	
#define GIC_DIST_SGI_PENDING_CLEAR	0xf10	
#define GIC_DIST_SGI_PENDING_SET	0xf20	

CPU Interface

#define GIC_CPU_CTRL	0x00	
#define GIC_CPU_PRIMASK	0x04	
#define GIC_CPU_BINPOINT	0x08	
#define GIC_CPU_INTACK	0x0c	
#define GIC_CPU_EOI	0x10	
#define GIC_CPU_RUNNINGPRI	0x14	
#define GIC_CPU_HIGHPRI	0x18	
#define GIC_CPU_ALIAS_BINPOINT	0x1c	
#define GIC_CPU_ACTIVEPRIO	0xd0	
#define GIC_CPU_IDENT	0xfc	
#define	0x1000	

GIC_CPU_DEACTIVATE		

```
enum irqchip_irq_state {
	IRQCHIP_STATE_PENDING,		/* Is interrupt pending? */
	IRQCHIP_STATE_ACTIVE,		/* Is interrupt in progress? */
	IRQCHIP_STATE_MASKED,		/* Is interrupt masked? */
	IRQCHIP_STATE_LINE_LEVEL,	/* Is IRQ line high? */
};
```