Platform Bus and Platform Devices

Depending on the bus connected to the device we can decide whether the device is a platform device or not a platform device.

The following table is used to differentiate between these two kinds of devices.

Field	Platform	Non Platform
Bus	A pseudo/virtual bus which is used to connect devices on buses with minimal infrastructure, like those used to integrate peripherals on many system-on-chip, processors, or some "legacy" PC interconnects.	A physical PCI fabric or USB fabric is present which is the actual bus. Has specific properties/features as per the appropriate standard.
Device	Devices are Inherently Not Discoverable The hardware cannot say "Hey! I'm present!" to the software. Typical examples are i2c devices, spi devices. These are callled non hotpluggable devices.	Devices are Discoverable In PCI automatic hotplugging is supported at the boot time only. After that automatic loading is not present. Though probing is not automatic, but can be done after boot with command -- echo 1 > /sys/bus/pci/rescan. In USB hotplugging is fully supported and devices may be directly added or removed dynamically.
	They are bound to drivers by matching names.	The correct driver is selected by the PCI/USB Vendor:Device ID. This is baked into every device, and vendors must ensure uniqueness.
	Platform devices should be registered very early during system boot. This is because they are often critical to the rest of the system (platform) and its drivers.	Register and interrupt addresses are dynamically allocated by the PCI/USB system.
x86	A platform device is represented by the interface : struct platform_device	Device properties are not specified in any kernel structure. Device details are obtained during bus scanning.

	The platform driver can bind to this device by the either of the two ways:
	Method 1
	----------- 	
	The first is the id_table argument.
	The structure used is :
	struct platform_device_id {
	  char name[PLATFORM_NAME_SIZE];
	  kernel_ulong_t driver_data;
	};
	If an ID table is present, the platform bus code will scan through it every time it has to find a driver for a new platform device.
	If the device's name matches the name in an ID table entry, the device will be given to the	

	driver for management.
	A pointer to the matching ID table entry will be made available to the driver as well.
	Method 2
	----------- 	
	Most platform drivers do not provide an ID table at all, they simply provide a name for the driver itself in the driver field.
	static struct platform_driver i2c_gpio_driver = {
	  .driver = {
	    .name = "i2c-gpio",
	    .owner = THIS_MODULE,
	  },
	  .probe = i2c_gpio_probe,
	  .remove =__devexit_p(i2c_gpio_remove),
	};
	With this setup, any device identifying itself as "i2c-gpio" will be bound to this driver; no ID table is needed.	

SoC	Register and interrupt addresses are hardcoded in the device tree and match the machine description, which represents the SoC.	No device tree is required as we dont have to specify the device properties. The device details are obtained during bus scanning.
	There is no way to remove the device hardware (since it is part of the SoC)	
	The correct driver is selected by the compatible device tree property which matches platform_driver.name in the driver	
	platform_driver_register is the main register interface	

References
1. LWN Article by Jonathan Corbet - https://lwn.net/Articles/448499/