Xilinx 快速嵌入式设计导论

一、设计队伍：

1、系统架构

2、硬件设计

3、软件设计

二、设计检查表

xtp397-embedded-design-methodology-checklist

DocNav软件design hub tab下可生成checklist

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/overview> 重要网页

三、总体设计内容

A、Performence

The system architect can makeearly performance estimates of data and communication paths, and fine tune them laterusing the system performance-monitoring points and tools. 系统性能监视调试工具？

关注点：

（1）系统数据传输

Moving data through a system is a common system-level performance problem. A Zynq device has several AXI masters that can drive transactions either directly or with assistance from DMAs. This section describes the various options and trade-offs for addressing datamovement in a Zynq device.

PS and PL Ethernet Performance and Jumbo Frame Support with PL Ethernet in the Zynq-7000 AP SoC (XAPP1082)

For example, a PS DMA data transfer to PL will typically go through the 32-bit master GP ports.

For on-chip buffering, the OCM, L2 cache, and DDR controller are the three main sources of sharable buffer space within the PS. The L2 cache and DDR controller provide excellent buffer-access latency for sharing data between the processor and ACP port. Only the ACP can access the L2 cache from PL. For high-bandwidth accesses to DDR, the HP ports are better suited than ACP. The OCM can be used by software applications as a 256 KB scratchpad accessible by all masters in the PL. A benefit to using OCM is its excellent random-access latency, whereas the L2 cache and DDR memory benefit from memory-access locality.

（2）System Monitoring

1、SCU Global Timer (PS). The SCU global timer can be used to timestamp system events in a single clock domain.

2、ARM Performance Monitoring Units (PS). Each ARM core has a performance monitoring unit (PMU) that is used to count micro-architectural events.

3、L2 Cache Event Counters (PS). The L2 cache has event counters that can be accessed to measure cache performance.

4、GigE Controller (PS). The gigabit Ethernet controller has statistical counters to track bytes received and transmitted on its interface.

5、AXI Performance Monitor (PL). This core can be added in PL to monitor AXI performance metrics such as throughput and latency.See the AXI Performance Monitor web page.

6、AXI Timer (PL). This core can be added in PL to provide a free-running timer in PL. This

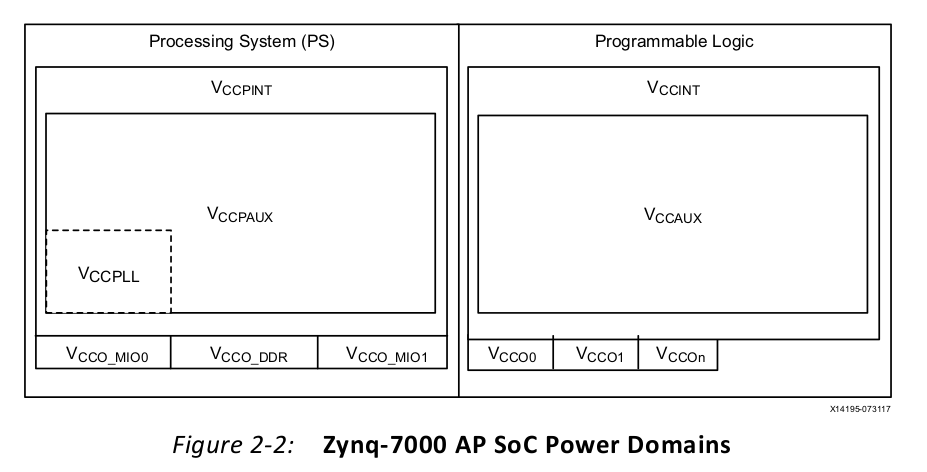
timer is useful for time-stamping events in PL clock domains. See the AXI Timer/Counter web page.

7、AXI Traffic Generator (PL). This core can generate a variety of traffic patterns to the PS interfaces.For more information, refer to the LogiCORE™ AXI Traffic Generator web page.

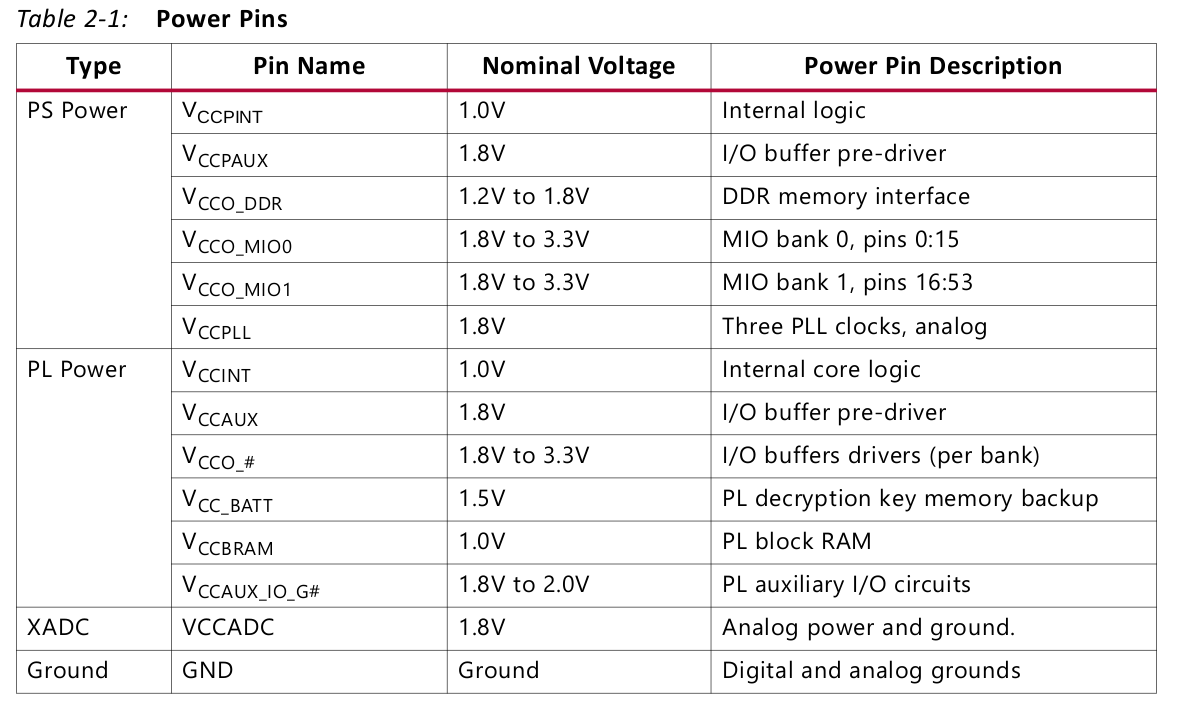
B、Power Consumption

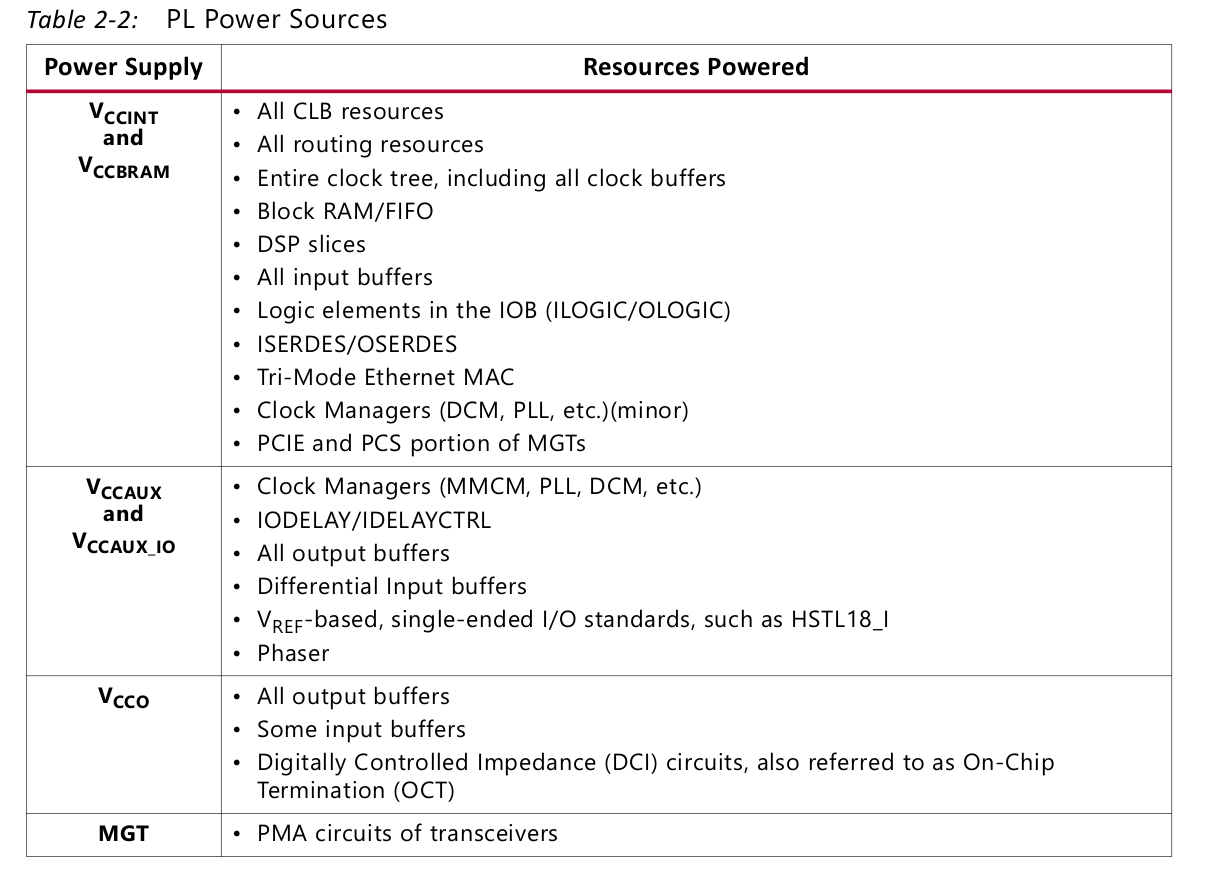
transistor size decreases with each process technology generation. As size decreases the amount of current each transistor leaks increases, causing an increase in static power consumption. Increasing SoC performance requires higher-frequency clocks,resulting in increased dynamic power consumption. Thus, static power is driven by transistor leakage current and dynamic power is driven by the transistor switching frequency. There are different ways to reduce system power. The following sections provide tips that can be used to optimize a design to meet system power requirements.

Zynq-7000 AP SoC devices are divided into several power domains, as illustrated in Figure 2-2.



The PS and PL power supplies are independent; however, the PS power supply must be present when the PL power supply is active. The PL can be powered off in applications that do not require the PL. The PS and PL power pins are summarized in Table 2-1. The voltage sequencing and electrical specifications are described in Zynq-7000 All Programmable SoC (Z-7010, Z-7015, and Z-7020): DC and AC Switching Characteristics (DS187).





PS power management

Refer to the Zynq Power Management wiki page

PL power management

The PL can be powered off in applications that do not require the PL. To do this, independently-connected power supplies are needed for the PS and PL. The configuration is lost when the PL is powered down and must be reconfigured when it is powered on again. Software should determine when it is safe to power down the PL.

Avoid using both a set and reset on a register or latch.

Use active-high control signals, because the control ports on registers are active high. Active-low signals use more lookup tables because they require an inversion before they drive the register control port.

The equation for dynamic power is:

Dynamic Power = α x fclk x C x V2

I/O power can be a major contributor to total power consumption. In some designs, as much as half of the total power consumption comes from the I/Os, particularly in memory-intensive systems.

The devices support digitally controlled impedance (DCI) technology, and can be tri-stated. DCI eliminates termination power when the I/O’s output is enabled, so that the device consumes termination power only during ingress cycles.

Zynq-7000 AP SoCs incorporate a user-programmable receiver power mode for HSTL and SSTL. By controlling the programmable power modes on each I/O, DC power can be reduced by making trade-offs between power and performance.

To maximize power savings when running the power optimizer in the Vivado tools, run power optimization on the entire design and do not exclude portions of the design. （应进行整体优化，而不要排除局部）If power savings are not realized after enabling power optimization, there are three areas that should be examined more closely:

• Global set and reset signals

• Block RAM enable generation

• Register clock gating

a、Understanding and implementing power-sensitive design techniques before coding is the

single-largest method for reducing system power.

b、Using the various Xilinx tools at the appropriate design cycle stages also helps in meeting power specifications, and provides the board designer with information on selecting the number, type, and size of the power supplies.

C、Clocking and Reset

（1）External Clocks

On the PS side, a fixed-frequency oscillator in the range of 30–60 MHz is typically used to provide the processor clock PS\_CLK. The clock must be a single-ended LVCMOS signal, using the same voltage level as the I/O voltage for MIO bank 0. From this clock, all other PS internal clocks are generated based on three PLLs: ARM, DRM, and IO PLL.

The default PS\_CLK frequency used on Xilinx evaluation boards is 33.3 MHz. Other clock frequencies can be used, but the following items are dependent on the PS clock frequency

and must be adjusted accordingly:

• The LogiCORE IP Processing System 7 configuration wizard calculates the derived clock

dividers and multipliers of each of the PLLs as well as the I/O peripheral clocks such as

SPI or UART based on the selected PS\_CLK frequency. These values are later used by the

first stage boot loader (FSBL) during initialization of the PS.

• The U-Boot board configuration include file.

• The Linux design-specific device tree.

On the PL side, single-ended or differential fixed-frequency oscillators can be used as additional clock sources for greater flexibility. They should be connected to multi-region clock capable (MRCC) input pins and adhere to the I/O standard and voltage of the corresponding PL bank.

（2）internal clocks

1、PS端

All clocks generated by the PS clock subsystem are derived from one of three programmable PLLs: CPU, DDR, and I/O.

a、The CPU clock domain is composed of four separate clocks: CPU\_6x4x, CPU\_3x2x, CPU\_2x,

and CPU\_1x. These four clocks are named according to their frequencies, which are related by one of two ratios: 6:3:2:1 or 4:2:2:1 (abbreviated 6:2:1 and 4:2:1, respectively). All of the CPU clocks are synchronous to each other.

b、There are two independent DDR clock domains:

DDR\_3x and DDR\_2x. These clocks are asynchronous to each other and the CPU clocks.

c、Most I/O peripherals clocks have dedicated dividers. Each peripheral clock is completely

asynchronous to all other clocks.

2、PS - PL 界面

PL AXI channels (AXI\_HP, AXI\_ACP, and AXI\_GP) have asynchronous interfaces between the

PS and the PL. The synchronization, where the clock domain crossing occurs, is locatedinside the PS. Therefore, the PL provides the interface clock to the PS. Each of the aforementioned interfaces could use unique clocks in the PL.

The PS provides four frequency-programmable fabric clocks (FCLK [3:0]) to the PL that are physically spread out along the PS-PL boundary. The clocks can be controlled individually by setting the clock's source (ARM, DDR, or I/O PLL) and the clock's output frequency. There

is no guaranteed phase relationship between any of the four FCLK clocks, even when sharing the same clock source. Make sure to use appropriate design constraints when interfacing between multiple FCLK regions. The FCLK clocks are disabled until the PS – PL level shifters are enabled.

RECOMMENDED: It is good practice to route a single FCLK into a clocking wizard IP core instantiated inside the PL to generate more than one phase-aligned output clocks.

The following are pros and cons of using FCLK:

• The processor controls the PL clock frequency.

• An on-board clock generator is not available.

FCLK is the preferred PL clock under the following circumstances:

° The processor controls the PL clock frequency.

° An on-board clock generator is not available.

FCLK is not the preferred PL clock under the following circumstances:

° The PL clock frequency is outside the frequency range supported by FCLK.

° The PL uses a clock provided by the FPGA pins. This is common in source-synchronous protocols that use the input clock to sample receive data.

° Low clock jitter is required.

° Some IP blocks that require specific clocking cannot use FCLK:

- The memory-interface generator (MIG) requires a differential clock, Therefore, FCLK cannot be used for the MIG except at reduced frequencies due to jitter.

- GTs should use a differential clock from the board as a reference clock.

3、PL端

The PL provides clock primitives that are commonly found on FPGA devices, such as global

or regional clock buffers (BUFG, BUFR), phased-locked loops (PLL), or mixed-mode clock

managers (MMCM).

综上，In the simplest case, a complete Zynq-7000 system can be built with a single input clock based on PS\_CLK, and all PL clocks are generated from the provided FCLKs and FPGA clocking resources.

（3）Processing system reset

a、PS Power-On Reset

the PS power-on reset (PS\_POR\_B) is an active-low signal used to hold the PS in reset until

all PS power supplies are stable and at their required voltage levels. This signal should be generated from the power supply power-good signal or from a voltage supervisor chip. At the time PS\_POR\_B is released, the system clock (PS\_CLK) must have been stable for 2,000 clock cycles. PS\_POR\_B should be pulled high to VCCO\_MIO0. When asserting PS\_POR\_B, the pulse length must be longer than 100 μs.

b、PS System Reset

The PS system reset (PS\_SRST\_B) is an active-low signal that is used primarily for debugging

proposes. If both PS\_SRST\_B and PS\_POR\_B are used, PS\_POR\_B must be the last signal that is de-asserted.

c、System Software Reset

The System Software Reset, also called SLCR Soft Reset, is asserted by writing to PSS\_RST\_CTRL[SOFT\_RST] and has the same effect as asserting the PS\_SRTS\_B pin. All of the

PS RAMs are cleared and the PL is reset as well.

d、Watchdog Timer Resets

There are two sources of watchdog timer resets: the System Watchdog, SWDT, and the two

ARM Watchdog Timers, AWDT0 and AWDT1. The SWDT always resets the entire system,

while each of the AWDTs can be used to reset either the associated ARM core (same effect

as CPU Reset) or the entire system (same effect as System Software Reset).

e、CPU Resets

There are two CPU Resets, one for each ARM core asserted by writing to

A9\_CPU\_RST\_CTRL[A9\_RSTx]. A CPU Reset to a single processor must be applied from the

other CPU, through JTAG or the PL.

f、Debug Resets

There are two debug resets, Debug System Reset and Debug Reset, that originate from the

ARM DAP and are controlled by JTAG. The Debug System Reset has the same effect as a

System Software Reset, whereas the Debug Reset only resets the debug logic.

g、Peripheral Resets

Individual peripheral resets can be asserted under software control, using programmable

bits within the SLCR. However, asserting reset at the peripheral block level is not recommended. Resetting a peripheral without completing all in-flight or pending transactions will cause the system to hang, because AXI transactions do not support a timeout mechanism. When asserting reset to a peripheral, all pending and in-flight transactions must be completed and no future transactions can be issued prior to the reset.

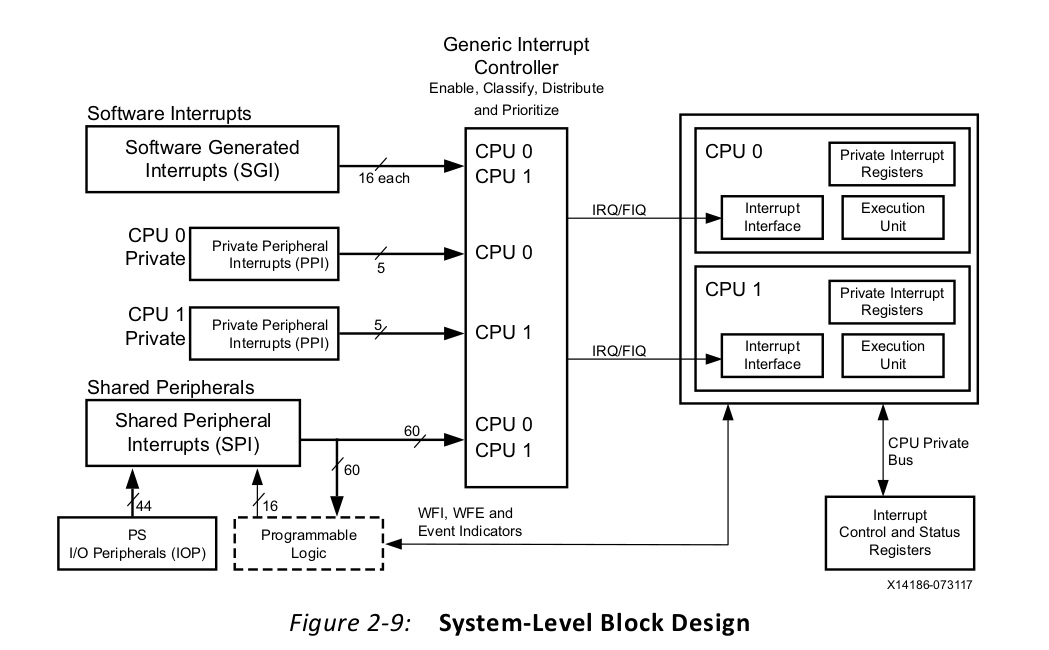
（4）program logic reset

The PS provides four programmable reset signals to the PL (FCLK\_RESET [3:0]).The FCLK\_RESET is loosely associated with the FCLK of the same number. That is, the FCLK

needs to be toggling for the FCLK\_RESET to propagate out of the PS. The reset is an asynchronous reset to the PL and you must synchronize the reset inside the PL if required.

RECOMMENDED: To synchronize an FCLK\_RESET, connect the signal to the external reset input port of a proc\_sys\_reset IP core. By doing so, the reset output signals connected to other PL IP cores can be synchronized to the slowest clock.

D、Interrupts



the generic interrupt controller (GIC) architecture contains an interrupt distribution block (distributor) and two processor-interrupt blocks. Each block contains a set of registers. Registers in the processor-interrupt blocks can only be accessed by the processor they are attached to using its private bus and cannot be accessed by the other CPU or other AXI masters in the system. Distributor registers can be accessed by either processor, and some registers are also banked for secure and non-secure access.

Interrupts sent to the GIC can be unmanaged (legacy) or managed.

a、Legacy interrupts are not controlled by the GIC.

b、Managed interrupts are controlled by the GIC and the interrupt handler must interact with

GIC registers.The distributor can control managed interrupt sources as follows:

• Define the interrupt as edge-sensitive or level-sensitive, subject to hardware

configuration limitations.

• Assign a 5-bit priority to the interrupt, with a programmable binary point.

• Assign a TrustZone technology security state to the interrupt. Interrupts labeled as

secure are called Group 0 interrupts, and non-secure interrupts are labeled as Group 1

interrupts.

• Route shared peripheral interrupts to one or both processors.

Note: Private peripheral interrupts (PPIs) are dedicated to each CPU and cannot be routed

elsewhere other than through the GIC.

• Route a software-generated interrupt to one or both processors.

• Save and restore the pending state of each interrupt. This is useful in low-power

applications.

（1）PS Interrupt Sources

Each Cortex-A9 processor can be interrupted by the following sources:

• Sixteen software-generated interrupts (SGIs) are available for software to interrupt

either processor.

• Five private peripheral interrupts (PPIs) are available. There are two interrupts from PL

(FIQ and IRQ) and one each from the global timer, the private timer, and AWDT.

• Sixty shared peripheral interrupts are available. There are 44 PS I/O peripheral

interrupts and 16 PL interrupts.

• Four PL interrupts can bypass the GIC and directly interrupt the processors, reducing

interrupt latency.

• Although it is not an interrupt, a CPU may use the WFE instruction to deliberately stall

and wait for an event on a dedicated input line from the other CPU or the PL.

（2）PL Interrupt Sources

Each device in the PL that responds to interrupts must arrange for its own interrupt

controller, if needed. The following interrupts can be sent from the PS to the PL:

• Twenty-nine shared peripheral interrupts from the PS. These correspond to many (but

not all) of the PS peripherals.

• A processor can also use software-generated interrupts to interrupt the PL by using

EMIO GPIOs, an AXI\_GPIO output channel in the PL, or by asserting a per-processor

hardware event line using the SEV instruction.

A Zynq-7000 AP SoC supports IRQ IDs #0 through #95。

Interrupts can originate from the PS or the PL, and a PS processor or dedicated hardware in

the PL fabric (such as MicroBlaze™) can respond to interrupts. This enables the unique

capabilities of Zynq-7000 AP SoCs。

a、Interrupt Processing in Fabric

b、Processor as an Extension of Fabric

c、Using OCM for Handlers

d、Processor Affinity

e、Using FIQ and IRQ，IRQ可以被FIQ剥夺

f、TrustZone

interrupt implementation considerations:

• Latency and associated jitter should be characterized and managed by hardware or the

operating system. Jitter occurs if the OS masks interrupts for an unpredictable time

period, or because higher priority interrupts are being serviced.

• Policies for peripherals and associated software should be defined for cases when

interrupt handling is delayed. Choices include dropping data, throttling or pausing the

remote data source, and allowing hardware to run with degraded performance.

• Processor utilization is sufficient for peripheral services and other tasks, assuming

average and worst case scenarios.

• The interrupt handler's memory location can affect instruction-fetch performance.

Memory closer to the processor will have lower latency. Aligning the interrupt service

routine so that it starts on a cache line boundary can improve latency.

• When an interrupted peripheral operates on data that could be cached by the

processor, the processor must maintain cache coherency by invalidating or flushing its

caches, or by using hardware coherency mechanisms such as the ACP. On Zynq-7000 AP

SoCs, PL peripherals that use the ACP port avoid this restriction.

• In systems implementing asymmetric multi-processing (AMP), an ownership policy for

interrupts (and other system resources) must be established.

E、Embeded Device Security

Potential threats to embedded devices include:

• Data privacy in the embedded device

• Cloning the embedded device

• Denial of service

• Malware insertion to change the embedded device behavior

• An insider providing keys to an adversary

Zynq-7000 AP SoCs that can be used to

provide different levels of security, such as:

• Boot image and bitstream encyption and authentication (secure boot).

• Partitioning the system into separate secure zones (TrustZone).

• Deploying asynchronous multiprocessing (AMP) on Zynq-7000 AP SoCs.

• Linux deployment on Zynq-7000 AP SoCs.

Boot sequence

a、Power on Reset

b、BootROM execution，boot securely or non-securely by boot header

c、（optional）BootROM CRC by eFuse settings （ref to XAPP1175 application note)

d、（selectable）RSA authentication performed on FSBL and prior to decryption of FSBL

如果第C步中，boot header选择 boot securely，那么RSA 使用public-key 对FSBL authentication，可见FSBL应该事先使用private-key签名，防止被篡改。注意 eFuse是用于存放public-key的，是一次性的，谨慎使用

e、AES decryption of FSBL（security boot）

f、FSBL loaded to OCM(on chip memory) from one of static mems or loaded JTAG(non-security)

g、HMAC(hash message authentication code) authentication of FSBL（security boot）

h、Disable OCM ROM memory

i、Pass control to FSBL

j、SSBL（second step boot loader，eg U-Boot）

Secure BOOT Image

The programmable components of a monolithic boot image (BOOT.bin) include:

•PS image components

°An initialization header that can optionally write values to registers. For example the

initialization header can be used to increase the CPU clock speed or boot device

speed before the BootROM copies and executes the FSBL.

° FSBL.

° Optional secondary-boot loader, such as U-Boot or bare-metal software.

° Optional data images and multiple ELF.

° Optional Linux uImage.

•PL bitstream

Bootgen

The programmable components of BOOT.bin are assembled using a Xilinx software tool

called Bootgen.

The keys used by Zynq-7000 AP SoCs are:

• AES 256-bit key， 用于解密FSBL

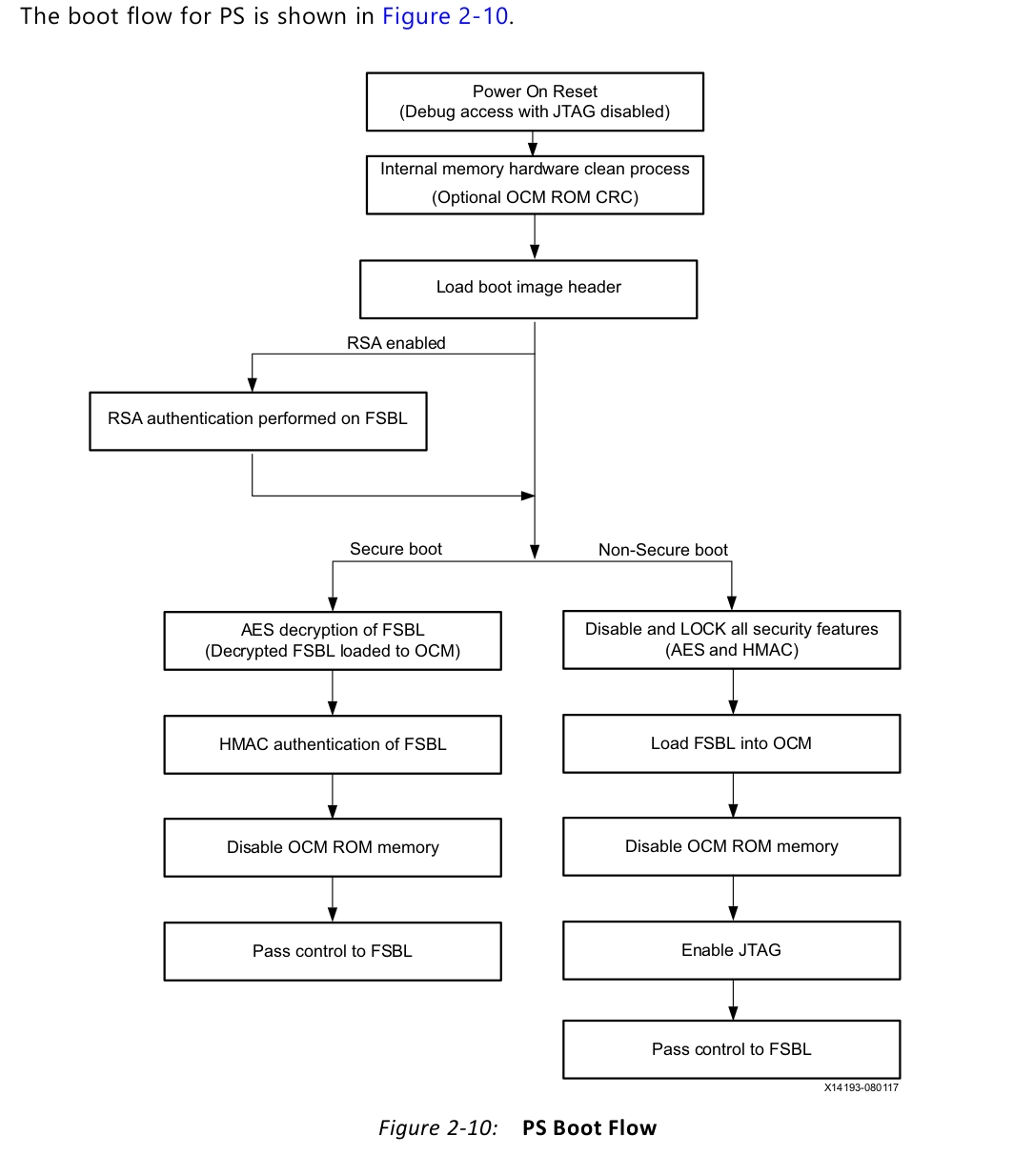
• HMAC key， 用于验证FSBL

• RSA Primary Secret Key (PSK)，用于econdary Secret Key的签名

• RSA Primary Public Key (PPK)，用于econdary Secret Key的验证

• RSA Secondary Secret Key (SSK)，用于签名software, data, and bitstream

• RSA Secondary Public Key (SPK)，用于验证software, data, and bitstream



Partitioning a System in Separate Secure Zones (Trust Zone)

ARM TrustZone technology ensures runtime security by enabling the system designer to

divide the system into logical partitions that allow only trusted software to access secure

components at the hardware level.For example, a system designer might not want third-party software to access system flash that is used to store private data, such as bank account details and passwords.（UG1019，这个是arm的功能）

Security using Asynchronous Multiprocessing (AMP)

For example, one ARM Cortex-A9 processor can be used to execute untrusted third-party applications and the other ARM Cortex-A9 processor can be used to execute trusted software. This approach ensures runtime security. Preventing a malicious boot of a Zynq-7000 AP SoC by an unauthorized programmable component is covered by secure boot.

Linux Deployment

Because Linux is ported to Zynq-7000 AP SoCs, Linux security features are available.

F、Profiling（扼要介绍） and Patitioning

Profiling tools help you determine how to partition an application’s functions between

hardware and software for optimal performance.

Software Profiling is a form of program analysis：

• Memory usage

• Function call execution time

• Function call frequency

• Instruction usage

profilling tools

Program Trace Module (PTM)

Performance Monitor Unit (PMU)

Level 2 (L2) Cache Event Counters

AXI Performance Monitor (APM)

Ethernet Statistics Registers

Software/Hardware Partitioning

functions can then be compiled into hardware and migrated into the PL for higher performance.

At the interface between hardware and software is a communication mechanism that allows data exchange between the two.This communication can be done over one of the PS AXI ports (AXI\_ACP, AXI\_HP, or slave AXI\_GP) using any of the following data movement schemes:

• Memory-mapped registers

• AXI-Stream FIFO

• AXI-DMA

四、Hardware Design Considerations 硬件设计

• Configuration and Boot Devices

• Memory Interfaces

• Peripherals Application Processing Unit(APU)是什么？

• Designing IP Blocks

• Hardware Performance Considerations

• Dataflow

• PL Clocking Methodology

• ACP and Cache Coherency

• PL High-Performance Port Access

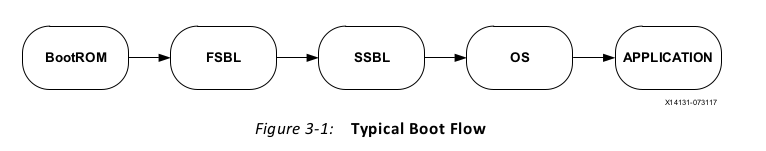
• System Management Hardware Assistance

• Managing Hardware Reconfiguration

• GPs and Direct PL Access from APU

1、Configuration and Boot Devices

Primary boot options like Quad-SPI, SD, NAND, NOR flash, and JTAG boot mode, and secondary boot options like eMMC and PCIe in the Zynq-7000 AP SoC。The JTAG boot mode is considered a slave boot mode and is always a non-secure boot mode.



Considerations:

The QSPI boot option requires fewer pins than the NAND or NOR boot options.

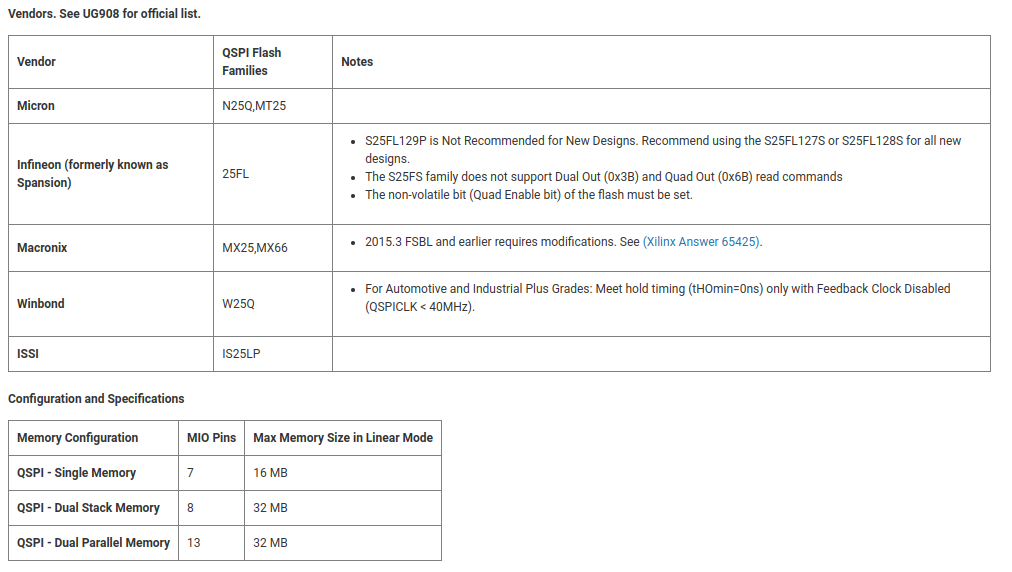
QSPI is the fastest available boot option.

NAND and SD boot options provide higher memory density than the QSPI and NOR options.

## 50991 - Zynq-7000 SoC - What devices are supported for configuration?

## <https://support.xilinx.com/s/article/50991?language=en_US>

QSPI Flash：



NAND Flash：

Only On-Die ECC and 1-bit ECC NAND devices can be used with Zynq-7000 SoC.

* In order to use On-Die ECC with Zynq-7000 SoC, the flash MUST be a MICRON and MUST support bit 3 (Enable/Disable ECC) in Feature Address 90h.

Only 1 chip-select NAND devices can be used with Zynq-7000 SoC.

For Guidelines on Xilinx Unverified QSPI Flash Devices

<https://support.xilinx.com/s/article/63798?language=en_US>

For Guidelines on Xilinx Unverified NAND Flash Devices

<https://support.xilinx.com/s/article/62743?language=en_US>

SD Memory card

Jtag

eMMC

You can use the eMMC secondary boot option when QSPI is the primary boot option and a

small QSPI memory is used. Typically, the FSBL will be loaded in to the QSPI and other

partitions will be loaded into eMMC. Xilinx recommends using eMMC with the SDIO

controller only in standard speed mode (max frequency of 25 MHz).

PCIe, Ethernet, USB, UART, and Custom FPGA Interface

The boot flow is:

• The FSBL will be loaded from the primary boot device.

• The APU waits until the FSBL performs the peripheral initialization process.

• The FSBL programs the bit file with the PIO design and handshake status register set.

• The APU programs the bitDone register after the bit file programming is done.

• The host PIO driver reads the bitDone register and writes the U-Boot.elf file to PL

block RAM.

• After U-Boot.elf is written, the status is written to the U-BootDone register.

• The APU polls the U-BootDone bit, and after it is done copies the file to PS DDR

memory.

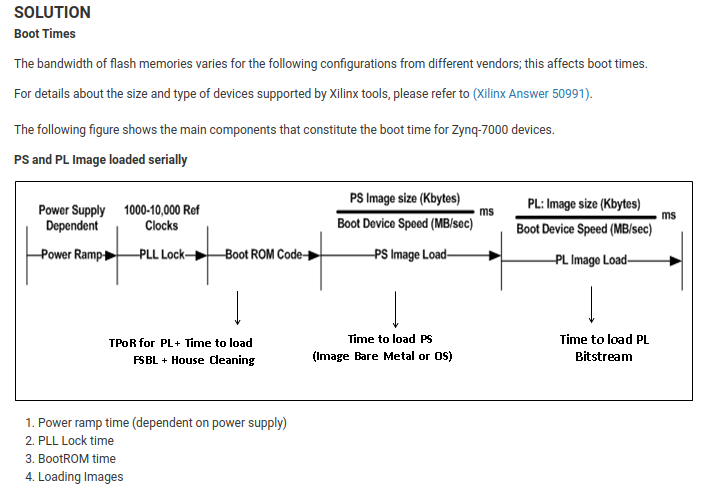
File System

Conventional file systems, such FAT or ext3, work with block devices and can be implemented using eMMC and SD cards。

Optimizing Boot Time

the Xilinx Answer Record 55572

<https://support.xilinx.com/s/article/55572?language=en_US>



如何理解系统启动步骤：

第一步 模拟电路系统启动，1、电源稳定，2、时钟锁相环路锁定。

第二部 数字电路系统启动，1、BootROM+FSBL启动（FBSL中多为用户定义的参数，所以独立）

2、SSBL+OS\_image(操作系统的引导及系统本身)

3、PL的bitstream

2、Memory Interfaces

1) DDR

the Zynq-7000 AP SoC supports 1.8V DDR2, 1.2V LPDDR2, 1.5V DDR3, and 1.35V DDR3L.

All devices support the 16-bit and 32-bit data bus width options, except the 7z010 CLG225 device that supports only the 16-bit data bus width. The controller optionally supports ECC in 32-bit configurations, with 16 data bits and 10 check bits. When ECC is enabled the data width is limited to 16 bits. A 1 GB address map is allocated to the DDR. However, if ECC is used, only 512 MB of address space is available.

The maximum supported bus clock is 666.6666 MHz in DDR3 mode on the fastest speed grade

The maximum supported bus clock is 533 MHz in DDR3 mode for all other speed grades.

Data transfer rate = 666.6666 MHz \* 2 bits (for double data rate) = 1333 Mb/s per data IO

Using the maximum bus width of 32 bits, the maximum bus bandwidth is 42.6 Gb/s, or 5.3 GB/s.

The DDR memory controller contains three major blocks: an AXI memory port interface

(DDRI), a core controller with a transaction scheduler (DDRC), and a controller with digital

PHY (DDRP).

<https://www.xilinx.com/cgi-bin/docs/ndoc?t=user_guides;d=ug585-Zynq-7000-TRM.pdf;a=xDDRMemoryController>

The controller DDRC block includes a three-stage arbiter for improved DDR control latency.

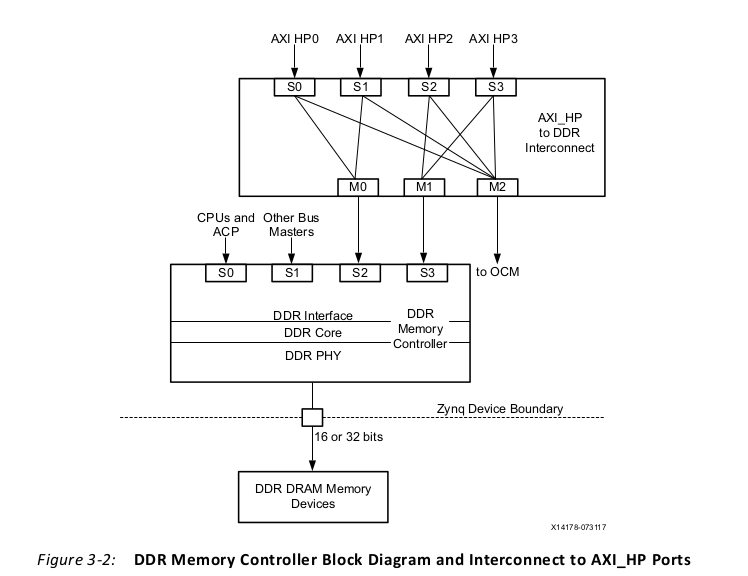
Latency can be controlled using register settings.

DDRP includes a DRAM training feature to help automatically determine the timing delays required to align data to the optimal window for reliable data capture. The Zynq-7000 AP SoC tool flow helps automate the DDR bring-up. To do this, the PS DDRC board parameters need to be configured in accordance with Xilinx Answer Record 46778

[ <https://support.xilinx.com/s/article/46778?language=en_US> ]. This will import the delay characteristics of DDR signals on the board during the

hardware design process. Those characteristics are used to determine the initial values used

by the automatic training algorithm, or for calculating static interface timing when the automatic algorithm is not supported by the particular DDR standard. The timing values are part of the design's hardware platform specification and are exported to the SDK and used in PS initialization code that is called by the FSBL. DDR is not used by the BootROM.



The DDRI block connects to four 64-bit synchronous AXI interfaces to serve multiple AXI masters simultaneously.

Port S0 is connected to the L2-cache and services only the PL CPU and ACP interfaces.

Port P1 is shared by all central-interconnect masters, such as PS peripherals and AXI GP ports.

Ports 2 and 3 are connected to The four PL AXI\_HP interfaces are multiplexed down in pairs.

2)QSPI

QSPI NOR flash can behave like a standard address-mapped memory and is well-suited for code storage. Also, it supports the execute-in-place (XIP) feature where a CPU can execute code directly out of QSPI without reading the code into DDR or OCM first.

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842377/Zynq-7000+AP+SoC+Boot+-+Booting+and+Running+Without+External+Memory+Tech+Tip>

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/overview?homepageId=18844350>

The QSPI flash controller supports three different modes of operation:

**I/O mode**,

**linear-addressing mode**, The controller supports only 24-bits of flash address, therefore the maximum size of QSPI is limited to 16MB in linear mode. BootROM accesses the QSPI in linear mode.

**legacy SPI mode**.

The QSPI controller supports only 24-bits of flash address, therefore the maximum size of QSPI is limited to 16MB in linear mode. BootROM accesses the QSPI in linear mode.When a QSPI larger than 16MB is used on the board, any Zynq-7000 AP SoC platform reset must also trigger a reset to the QSPI so that the page register is reset. <https://support.xilinx.com/s/article/57744?language=en_US>

In a high-speed QSPI application where the memory-interface clock is greater than 40 MHz,

QSPI feedback mode must be used.

IO mode ref to UG585

**Static Memory Controller**

The static memory controller has two interface modes:

**a parallel port memory interface mode**. It supports NOR flash and asynchronous SRAM. it supports NOR flash sizes only up to 64MB(address lines to 26 bits, parallel port needs 40 pin, compared to 128MB devices supported by QSPI using just 8 pins), QSPI a preferred solution over NOR flash.

**a NAND flash interface mode，**only NAND flash is supported. The NAND flash controller can support up to 1GB of external NAND flash with either an 8-bit or 16-bit I/O bus for address/data/command. It supports the Open NAND flash Interface 1.0 specification. Because the controller only supports 1-bit ECC, only NAND devices with on-chip ECC or one-bit ECC can be used with Zynq-7000 AP SoCs. Currently, only single-level-cell (SLC) devices meet the ECC criteria and multi-level-cell (MLC) devices are not supported. Also, the controller only supports a single chip select.

Because NAND flash does not behave like random access memory, Linux systems use it as a memory technology device (MTD) that provides an abstraction layer, allowing software to access the device using the MTD subsystem API. That API is common among different flash types and

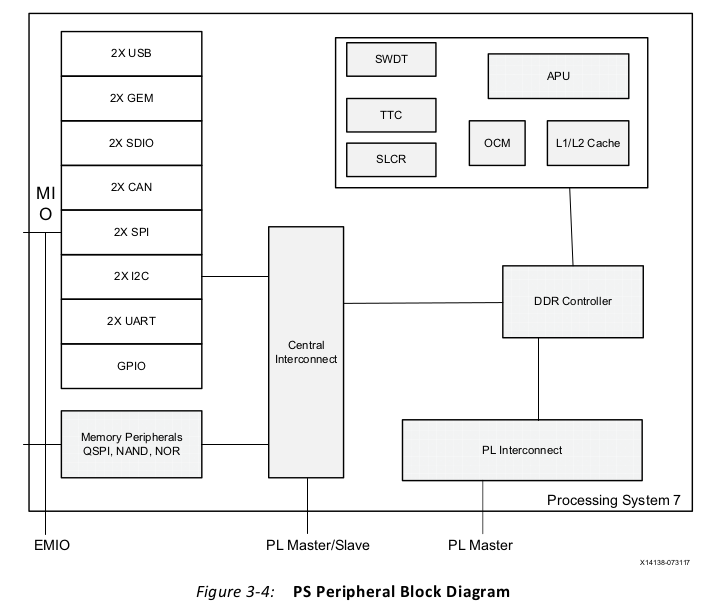
technologies. MTD is not a block device and it lacks the software management algorithms to handle issues like wear leveling and bad block management. Instead of traditional file systems like ext2, ext3, and FAT (which work on top of block devices), the file system must be designed to work on top of raw flash, such as JFFS2 or UBIFS. JFFS2 works on top of MTD subsystems. UBIFS works on top of UBI subsystems, and those(JFFS2 or UBIFS) work on top of MTD subsystems to provide software management algorithms required for NAND devices.

3、Peripherals

The Zynq-7000 AP SoC peripherals can be broadly categorized as:

• PS peripherals

ASSP（ Application Specific Standard Parts，专用标准产品）与ASIC是两个端。



• PL peripherals

**usb2.0**

**Gigabit Ethernet MAC(GEM).** To save pins, each controller uses an RGMII interface through the MIO. Access to the PL is through the EMIO(external MIO) which provides the GMII interface.

RGMII（Reduced Gigabit Media Independent Interface，精简吉比特介质独立接口）是Reduced GMII（[吉比特](https://baike.baidu.com/item/吉比特/316597?fromModule=lemma_inlink)介质独立接口）。RGMII均采用4位[数据接口](https://baike.baidu.com/item/数据接口/6659495?fromModule=lemma_inlink)，工作时钟125MHz，并且在[上升沿](https://baike.baidu.com/item/上升沿/3110757?fromModule=lemma_inlink)和[下降沿](https://baike.baidu.com/item/下降沿/6264539?fromModule=lemma_inlink)同时传输数据，因此[传输速率](https://baike.baidu.com/item/传输速率/10839944?fromModule=lemma_inlink)可达1000Mbps。

**SDIO Peripheral**

[https://www.sdcard.org/downloads/pls/](https://www.sdcard.org/downloads/pls/simplified_specs/)

SD cards are essentially NAND flash devices with built-in controllers that implement the flash translation layer (FTL). The FTL handles ECC, block management, and wear leveling so that the memory behaves like a block device. Because of this, conventional file systems (such as FAT, ext2, and ext3) can be implemented.

eMMC consists of flash memory and a controller packaged in a small ball grid array (BGA) that can be directly mounted onto a circuit board without a mechanical connector. The SD and

eMMC solutions differ in their ability to function as boot devices. A Zynq-7000 AP SoC can

boot directly from an SD card, but it cannot do so from an eMMC. eMMC solutions require

an additional boot device, such as QSPI.

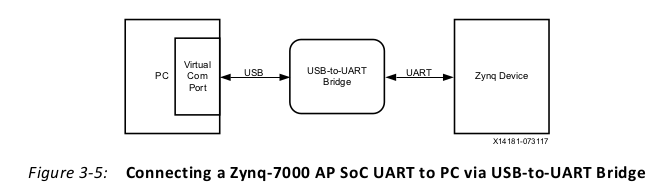
**UART Peripheral**

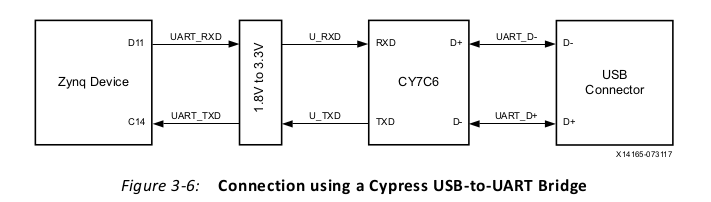
It is recommended that the user have access to UART1 for debug, even when using a PMOD with

two test points, TX and RX.

Zynq-7000 AP SoCs include two UART controllers that are commonly used as debug ports

for embedded systems.





**CAN Peripheral**

It supports bit rates up to 1Mb/s, 64 messages, 16-bit time stamping for receive messages, receive and transmit error counters. You can route the CAN transmit and receive signals to either MIO or the PL through EMIO.

The CAN controller supports five modes of operation:

• Configuration mode

• Normal mode

• Sleep mode

• Loopback mode

• Snoop mode

**I2C Peripheral**

Inter Integrated Circuit(集成电路间总线（协议），inter表示“之间”，inner表示“之内”) is a multi-master serial single-ended bus that uses two bidirectional open-drain lines, Serial Data (SDA) and Serial Clock (SCL), both pulled up with resistors which shall be placed at the far end of the SCL and SDA lines. The protocol includes a slave address, an optional register address within the slave device, and per-byte ACK/NACK bits.I2C is used for attaching low-speed peripherals such as sensors, EEPROMs, I/O expanders, programmable clock devices, or A/D and D/A converters to an embedded system.

Several bus implementations are derived from the I2Cbus, including the System Management Bus (SMB), Power Management Bus (PMB), and Intelligent Platform Management Interface (IPMI).

Pull-up resistors calculating reference to <http://www.edn.com/design/analog/4371297/Design-calculations-for-robust-I2C-communications>

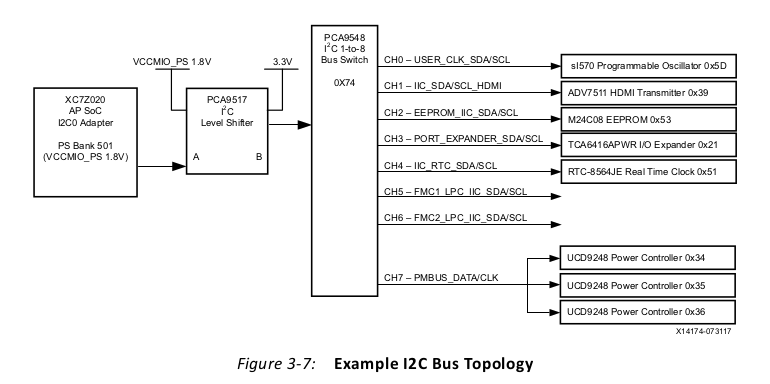
Zynq-7000 AP SoCs include two I2C controllers that can operate at the common I2C bus

speeds of 100 Kb/s (standard mode) and 400 Kb/s (fast mode). Each controller can function

as a master or a slave in a multi-master design. The master can be programmed to use both

normal (7-bit) addressing and extended (10-bit) addressing modes. The master is responsible for generating the clock and controlling the data transfer.

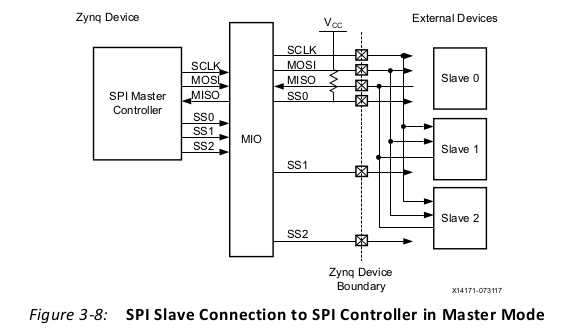
相同地址的从设备的连接需使用I2C交换机，如下图：

交换机可以将总线分为若干段。由此解决相同地址问题。当然地址各不相同，也可以不用交换机。

**SPI peripheral**

SPI(Serial Peripheral Interface) is a synchronous serial bus that operates in full-duplex mode. It uses four wires: SCLK (Serial Clock), MISO (Master Input, Slave Output), MOSI (Master Output, Slave Input), and SS (Slave Select). SPI is commonly used to talk to A/D and D/A converters, flash and EEPROM memories, LCDs, and many other peripherals. The JTAG standard is essentially an application stack for a three-wire SPI protocol. SPI可看作UART（Universal Asynchronous receiver and transmitter）升级的USRT（Universal Synchronous receiver and transmitter）

The SCLK clock frequency can operate up to 50 MHz when the I/O signals are routed to MIO pins. When the I/O signals are routed through EMIO, the SCLK frequency can operate up to 25 MHz.



It is recommended that the SCLK, MISO, MOSI, and SS lines have matched lengths to help

meet setup and hold times. PCB and package delay skew for the MISO, MOSI, and SS lines

relative to SCLK should be less than ±50 ps.

**GPIO Peripheral**

The GPIO controller has four banks. Bank 0 has 32 pins and bank 1 has 24 pins. The total of

54 GPIO pins in the two banks are dedicated to MIO.

Banks 2 and 3 have 32 pins each, and the total of 64 GPIOpins are connected to the PL using EMIO. These GPIO contain three signals: input, output, and output enable.

**Cortex-A9 Multiprocessing Peripherals**

The Cortex-A9 multiprocessing peripherals include the SWDT and TTC with auto decrement

feature and they can be used as general purpose timers.

**PS DMA Controller**

**XADC**

4、Designing IP Blocks

**Soft IP Blocks**: You can implement these blocks in an FPGA fabric and are specified

using RTL or higher-level descriptions.

**PS IP Blocks**: These blocks have fixed layouts and are optimized for a specific

application and process.

**IP Core Design Methodology**：

System Generator(from matlab)

HDL Coder(from matlab)

VHLS(from Vivado High level synthesis)

**IP Core Design Considerations**：

1）Parameterization:The basic level of parameterization can be implemented in the HDL code. The HDL parameters can be configured from the IP configuration wizard available after packaging the IP block using Vivado IP integrator tool.

2）Bus Topology:

The AXI4-Lite bus is primarily used as a control bus from the PS, and the AXI4 Memory-Mapped bus is for memory transfers between the IP core. The AXI4-Stream bus is used to connect to streaming blocks in the design that do not have addressing context.

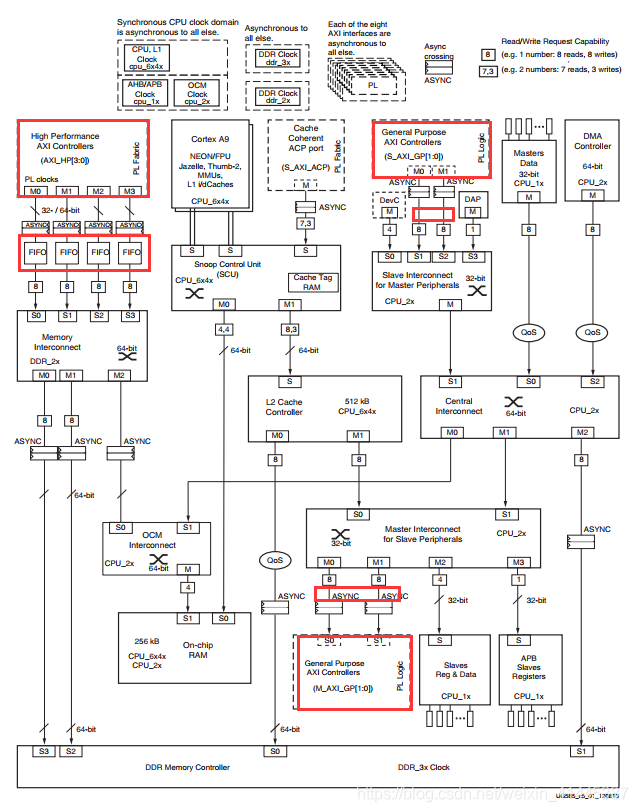
3）IP Security and Documentation:Protecting a design against copyright infringement is an important consideration in designing IP blocks.

**Connecting IP Blocks to the PS：**

When the interface between the PS and PL uses AMBA-compliant interfaces, you can implement a design using soft IP blocks.

5、Hardware Performance Considerations

**Hardware Performance Metrics，**An important metric of PL performance is the AXI throughput and latency。



**High-Performance AXI Masters**

**Using Performance-Critical AXI Slaves**

**High-Performance Datapaths**

**Monitoring Hardware Performance**

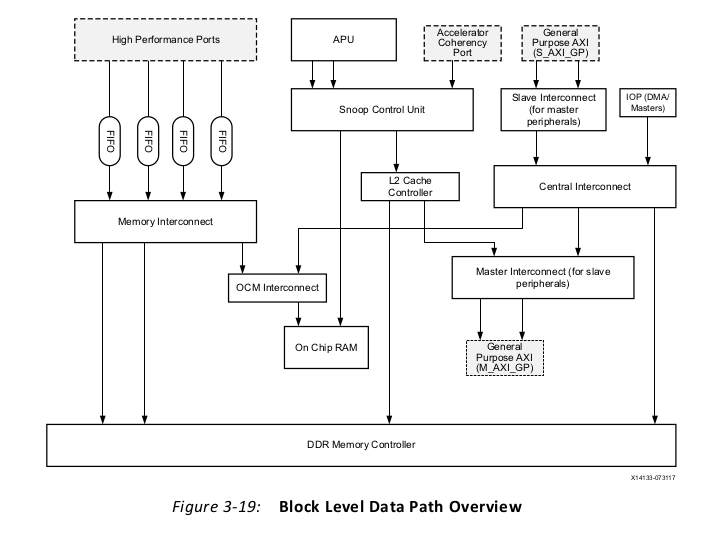
6、Dataflow

The first part describes the dataflow within the PS, focusing on APU access of PS DDR, and of PS

peripherals to PS DDR. The second part describes dataflow from the PL to the PS, focusing on the following PS-PL AXI interfaces

The PS interconnect is based on high-performance data path switches that enable data transfers between various peripherals and DDR memory, or between the APU and DDR or OCM memories.

The PS-PL interface provides an AXI interface in the PL, enabling PL peripherals to connect to the PS and subsequent dataflow.



**Interconnect within PS：**

There are two primary data paths within the PS:

the APU interconnect to PS DDR, APU interconnect to PS peripheral（APB） and the PS peripheral interconnect to PS DDR（AHB）.

**PS-PL AXI Interfaces：**

General Purpose AXI Interface (AXI\_GP)

Accelerator Coherency Port (ACP)

High Performance Ports (AXI\_HP)

System Level Design Considerations

The general-purpose ports should be used for low-to-medium types of traffic.

The accelerator coherency port is useful when applications in the PL need to maintain

coherency with the processor L1 caches, and need reduced software overhead.

The high-performance port is recommended when applications in the PL need high-bandwidth

access to the DDR controller or the OCM.

7、PL Clocking Methodology

The different clock sources available in the PL and their recommended usage. Those clock sources are:

• Clock from PS (FCLK)

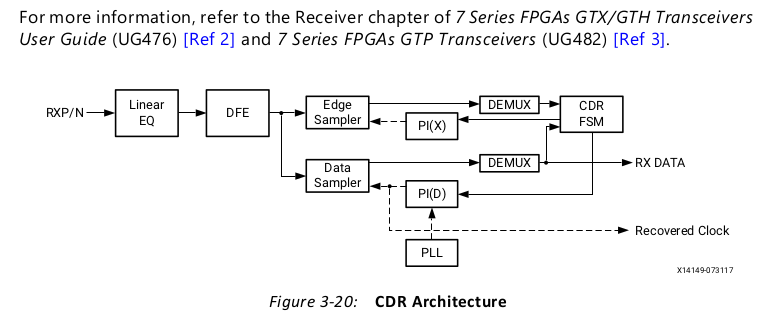
• Clock Recovered from a GT

• Clock from External Source

• Clock Generated by MMCM

**Clock Recovered from a GT**

The receiver clock data recovery (CDR) circuit in the GT transceiver extracts clock and data from an incoming data stream.

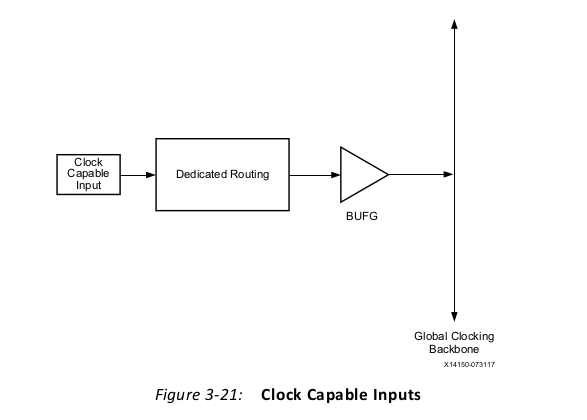


**Clock from External Source**

If a single-ended clock is used, it should be connected to the P (master) side of the clock-capable input pin pair. Differential clocking is recommended because it eliminates power-supply noise and line-coupling noise. If the design is pin limited, you can use single-ended clocking.

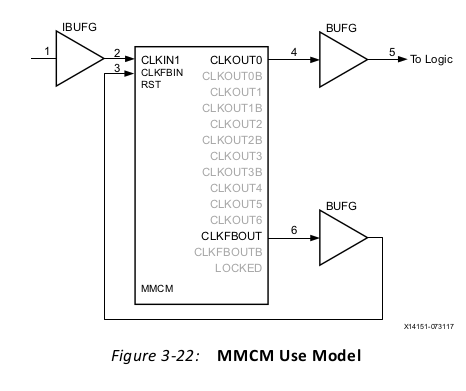
When a single-ended clock is connected to the P side of an input pin pair, the N side cannot be used for another single-ended clock. However, you can use it as a user IO. The clock-capable pins provide dedicated, high-speed access to the internal global and

regional clock sources, refer to 7 Series FPGAs Clocking Resources User Guide (UG472).



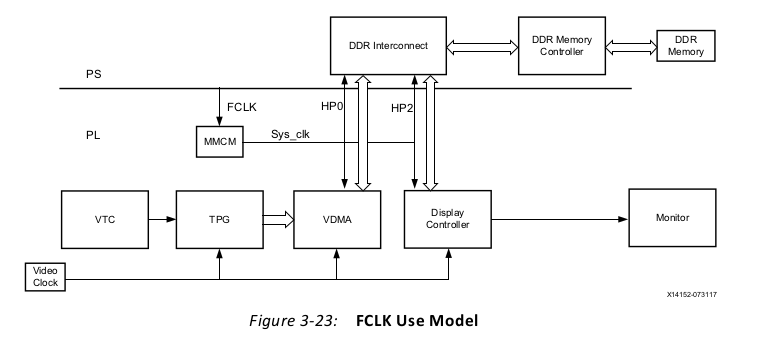
**Clock Generated by MMCM (mixed-mode clock managers)**

The MMCM is used to generate multiple clocks with different frequency and phase relationships. The MMCM also de-skews the clock output.The MMCM primitive, MMCME2\_ADV, provides the functions listed above and the input clock selection using the Dynamic Reconfiguration Port (DRP).



FCLK Use Model

An example FCLK use model is a video application that displays a test pattern on a display monitor.



• Test pattern generator (TPG): Generates video frames of different test patterns.

• Video timing controller (VTC): Generates the timing for TPG.

• Video direct memory access module (VDMA): Writes the video frames generated by TPG in to the DDR memory.

• Display controller: Fetches the frames from the DDR through HP2 port and displays on monitor.

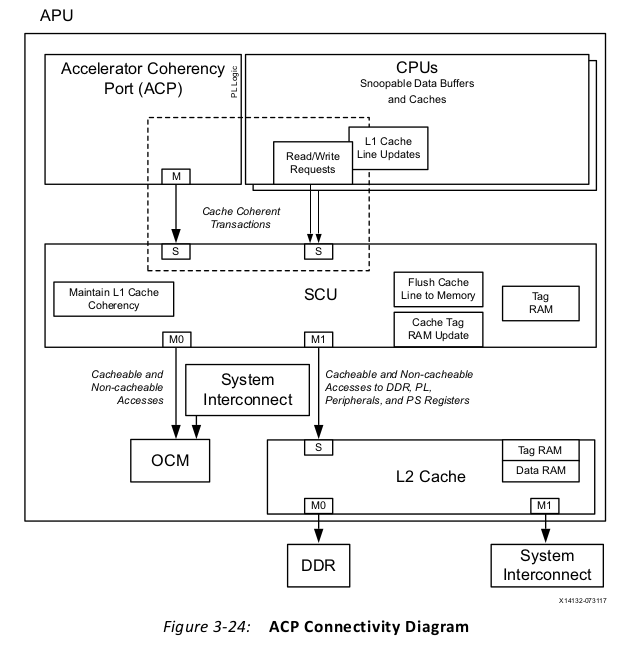
• On-board clock synthesizer: Used as a video clock driving all input video modules and the display controller. It is programmable by the PS.

• Clock derived from FCLK using the MMCM: This clock has a higher frequency than Sys\_clk. It is used by the VDMA and display controller AXI master interfaces because of display controller latency requirements.

**8、ACP and Cache Coherency**

The accelerator coherency port provides low-latency access for PL masters, including optional cache coherency with the dual core ARM Cortex-A9 CPUs.

注意下图中的APU（Application Processing Unit）的描述



The ACP's main benefit, compared to other PL masters, is access to CPU cache-coherent

memory and use of the low-latency 512KB L2 cache. In a common Zynq-7000 AP SoC

implementation, a PL accelerator is connected to the ACP port. It is configured by the CPU

over one of the GP ports, and then the accelerator does its data movement over the ACP.

Coherent data movement is managed in hardware, allowing the PL ACP-master to issue

standard AXI reads and writes. The only requirement on ACP cache-coherent accesses is

that those transactions have the A\*CACHE and A\*USER bits set to all 1s.

This hardware-managed coherency model is an alternative to having PL accelerators use the

HP or GP ports to access non-coherent memory directly from the DDR Controller. Here,

software-managed coherency would be necessary, whereby the processor would have to

flush shared memory back to the DDR to guarantee coherency. While there are many ways

to implement software-managed coherency, the ACP does so in hardware, easing software

design.

Because the ACP supports cache coherency, ACP-attached AXI masters compete with the

CPUs for L2-cache resources (the L1 cache is only available to the CPU). Any AXI read or

write transaction coming from the ACP will use L2-cache resources to service the data

movement (arbitration, tag matching, and cache-line fetches to DDR Controller). If this data

movement is coordinated with CPU activity (such as the accelerator example above), then

sharing the memory resources benefits the larger system application. If this data movement

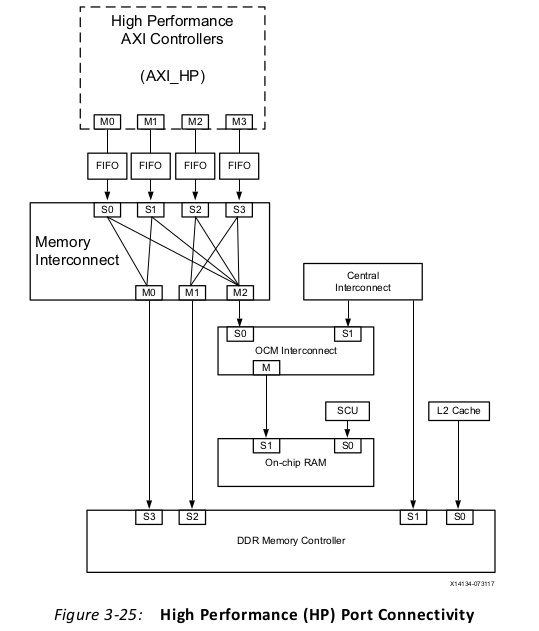
operates in parallel with unrelated CPU tasks, there are additional opportunities to mitigate

contention on the CPUs and ACP masters.

9、PL High-Performance Port Access

The high performance (HP) ports give the PL direct access to the DDR controller and to the

On-Chip Memory (OCM). The HP ports are typically used in a design needing PL high-throughput access to either the OCM or, more commonly, DRAM attached to the DDR controller.



10、System Management Hardware Assistance

Xilinx Analog-to-Digital Converter

TrustZone Security

DDR Error Recovery：error correction code (ECC) mechanism. Ten ECC bits for each 16 data bits support correction of single-bit errors and detection of two-bit errors.

Low Power: PL power off, processor standby mode, clock gating of PS subsystems, PLL configuration, or I/O voltage control.

11、Managing Hardware Reconfiguration

Partial reconfiguration is the ability to reconfigure a portion of an FPGA chip to implement

different logic functions without disturbing the remaining logic on the chip. The technology

enables an FPGA to implement different functional blocks in a time-sliced manner.

In the Zynq-7000 AP SoC PL architecture, the basic programmable unit is a configuration

frame composed of a LUT, DSP48, and BRAM. In partial reconfiguration, the configuration

frames are reprogrammed using a Processor Configuration Access Port (PCAP) that acts as

a reconfiguration agent. Each configuration frame has a unique address identified with a

top/bottom address bit, a major address, and a minor address. The partial bitstreams are

generated using a Xilinx Vivado Tool flow, and the partial bitstreams are parsed by the PCAP

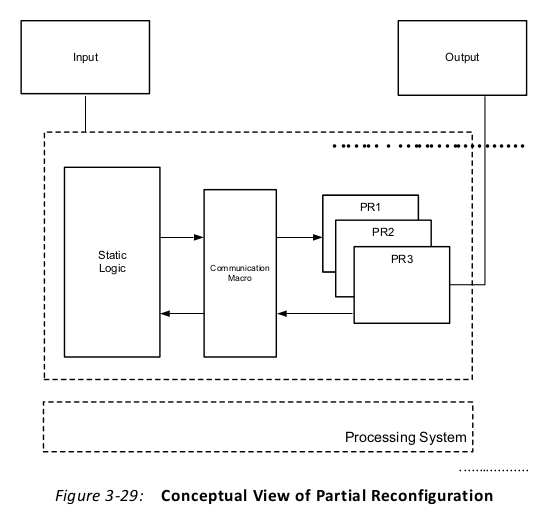
to reconfigure the frames.

The static design does not change over time and establishes communication between the partial

reconfiguration modules and the static design. The communication between the static logic

and the partial reconfiguration modules occurs over a set of communication macros

consisting of a bidirectional communication entity and tri-state gates.



The Zynq-7000 AP SoC has a PL portion that you can reconfigure using partial

reconfiguration. The PL is programmed using the Processor Configuration Access Port

(PCAP) that is part of the PS Device Configuration Interface. The PCAP block communicates

with the CPU and system memory using a PCAP to APB bridge that converts APB

transactions to PCAP read/write transactions. You can store the partial bitstreams

generated using the Xilinx Vivado design tool into the PS DDR memory, and retrieve them

to configure the PL over the PCAP interface using the partial reconfiguration methodology.

A configuration **defines** a complete FPGA design and **produces** **a full bitstream** for a reconfigurable module and static logic, plus **a partial bitstream** for the reconfigurable module.

**完整配置**包括**静态配置与首要部分配置**的full比特文件，加上其他部分配置模块的partial比特文件，见图3-30。

ECOMMENDED: To learn how to design with Partial Reconfiguration in Vivado, refer to the Partial Reconfiguration Design Hub in the Documentation Navigator. For more information, see Related Design Hubs.

**System Level Considerations**

1）Hardware Design Flow

The first step in creating a partially reconfigurable design is to create a design using a reconfigurable module and static design flow.

• Create a partial reconfiguration project using the Xilinx Vivado design tool.

• Define the reconfigurable partition.

• Create reconfigurable modules for the reconfigurable partition by adding the corresponding netlist and constraint files.

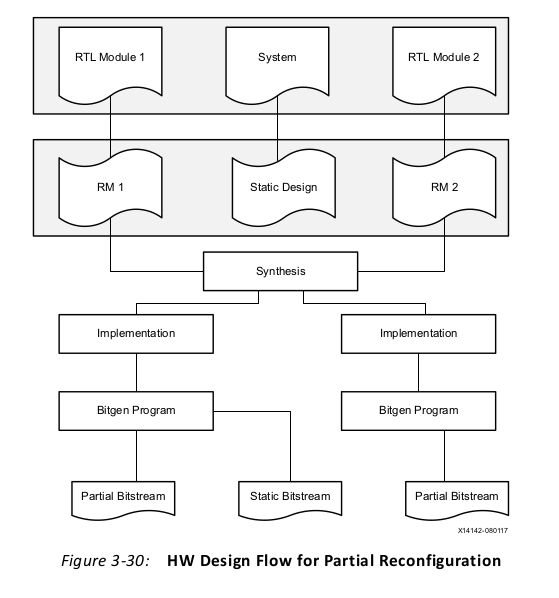
• Floorplan the reconfigurable partition by setting the physical size of the partition and

the types of resources desired.

• When building reconfigurable design configurations, the first configuration implemented should be the most challenging one.

• Run the partial reconfiguration verify-configuration utility to validate consistency between the reconfigurable-module implementations.

• Generate full and partial bitstreams for the reconfigurable modules.



2）System Design Flow

The Device Configuration (DevC) interface containing the AXI-PCAP bridge is used to

implement the partial reconfiguration flow on a Zynq-7000 AP SoC.

1. After power-on reset, the BootROM determines the external memory interface or boot

mode (SD flash memory) and the encryption status (non-secure). The BootROM uses the

DevC's DMA to load the First Stage Boot Loader (FSBL) into on-chip RAM (OCM).

2. The BootROM shuts down and releases CPU control to the FSBL which in turn configures

the PL with the full bitstream via the PCAP.

3. The FSBL loads and releases control to the user application in bare-metal OS.

4. The user application loads the partial bitstreams into DDR memory during start-up. This

is to maximize the configuration throughput over the PCAP interface, speed up the

configuration time, and take advantage of cache.

5. The application can use the partial bitstreams at any time to modify the pre-defined PL

regions while the rest of the FPGA remains fully active and uninterrupted. This is done

by transferring the reconfigurable module bitstream from DDR to the PL via PCAP.

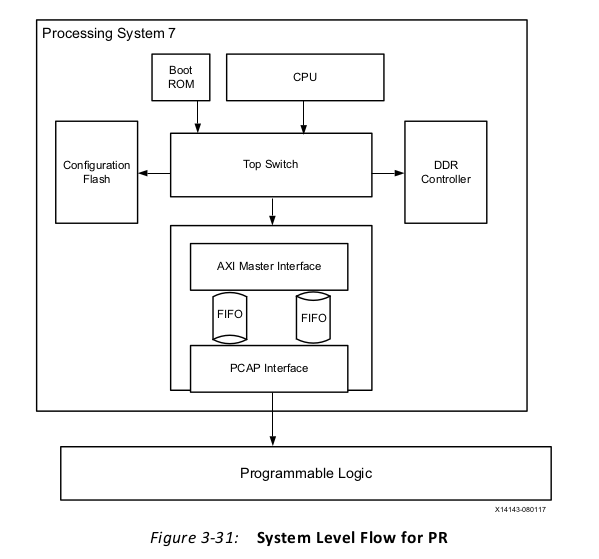
6. A single configuration engine handles both full configuration and partial

reconfiguration. The task of loading a partial bitstream into the PL does not require

knowledge of the reconfigurable module's physical location because

configuration-frame addressing information is included in the partial bitstream.

7.You can use Linux and U-Boot to dynamically load the reconfigurable bitstream to the PL.

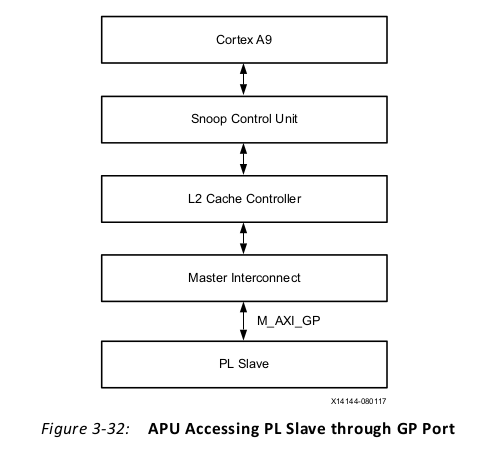


12、GPs and Direct PL Access from APU

The APU in PS can access registers and memory in PL using GP master AXI interfaces, which

are part of the PS-PL interface. There are two GP master AXI interfaces that the APU can use

to initiate an AXI transaction to a slave implemented in PL.



**Performance**

GP ports are for general purpose and not for high performance. The

GP port has a latency of 38 M\_AXI\_GP interface clock cycles.

**Design Considerations for AXI Master GP Port**

• When AXI interconnect in the PL is connected to the PS GP master AXI interface and the

HP AXI slave interface(如果PL中的AXI端口同时与PS中的GP master以及HP slave接口相连接), you need to disable HP slave port access from the master portGP. In the IP integrator address editor under processing\_system7 addressing, un-assign the HP port address to avoid the address conflict.

• Accesses to the GP interface are not cached by default. You can cache accesses by

changing the Translation Lookaside Buffers (TLBs).

• When using the GP interface with an AXI interconnect in PL, system software should

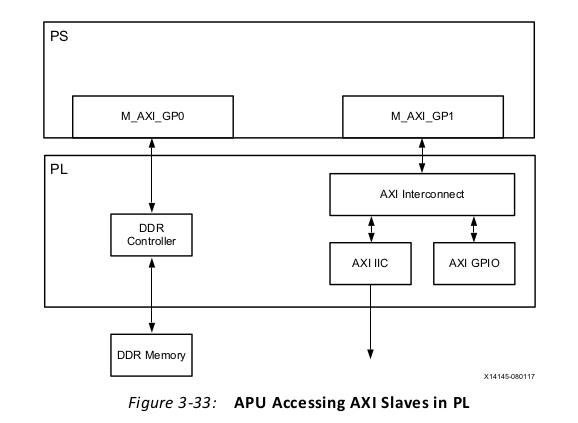
never access an address that does not exist in the AXI interconnect. This will avoid

permanent AXI interconnect lock-up (there is no timeout set by the AXI interconnect).

• The voltage translators at the PS-PL boundary must be configured before the APU uses

the master GP interface to access the AXI slave in PL. This is normally done by the FSBL.

**示例**



五、Software Design Considerations 软件设计（五个方面）

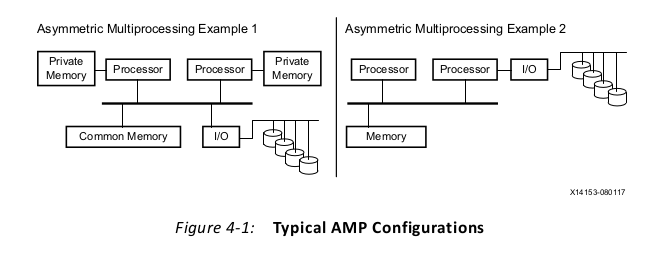
1、Processor Configuration

**Clock Speed and Multiple Cores**

If an operating system supports multiple processor cores, power could be saved by running the cores at a lower frequency instead of a single core at a high frequency.（根据公式Dynamic Power = α x fclk x C x V2，上所结论普遍成立，但不绝对。需要对比，才能得出这一结论。）

**SMP and AMP Configuration**

Most multicore SoCs have an option to run the system as an asymmetric multiprocessor (AMP) or symmetric multiprocessor (SMP).



• In example 1, each processor is allocated private memory, and that memory is not

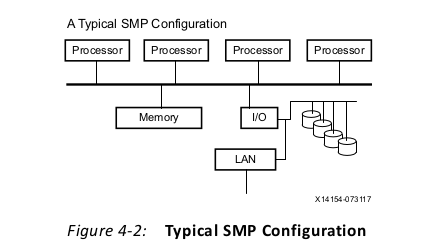
shared between the processors. Both processors have access to common memory and

a set of peripherals.

• In example 2, one processor does all memory operations but cannot access the

peripherals. The second processor accesses the peripherals but does not perform

memory operations.



**Using Coprocessors**

The Zynq-7000 AP SoC Spectrum Analyzer part 2-Building ARM NEON Library Tech Tip [ <http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Spectrum+Analyzer+part+2> [Building+ARM+NEON+Library+Tech+Tip](http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Spectrum+Analyzer+part+2-Building+ARM+NEON+Library+Tech+Tip) ] shows how to use NEON libraries to target the NEON capabilities of a Zynq-7000 AP SoC.

**Cache Considerations**

Coherency between the caches is maintained by a snoop control unit (SCU), and coherency decisions are determined by the AMP or SMP multi-processor configuration.

The L1 **instruction** cache must be enabled at the beginning of the boot process (typically, by the first-stage boot loader), and it should **not** be disabled thereafter. The L1 and L2 data caches can be configured and enabled at a later time.

**Processor State after Power on Reset**

In the Zynq-7000 AP SoC, both processor cores start after power-on reset (POR). CPU0

starts at address 0x0, where BootROM loads the first-stage boot loader. CPU1 is sent into a

wait-for-event (WFE) loop. Any interrupt will cause CPU1 to wake up and exit the WFE state.

The boot loader, kernel, or equivalent software is typically responsible for starting CPU1 at

a later time, when it is going to be used.

In most cases, DDR memory is not initialized at POR, so the first-stage boot loader is loaded

into simpler memory, such as on-chip memory, and is mapped to address 0x0 in the SoC.

ARM processors start in supervisor (SVC) mode after POR. Then, the kernel or equivalent

software changes the ARM processor mode as required.

**Interrupt Handling**

There are three possible interrupt sources:

•Private Peripheral Interrupts (PPIs):

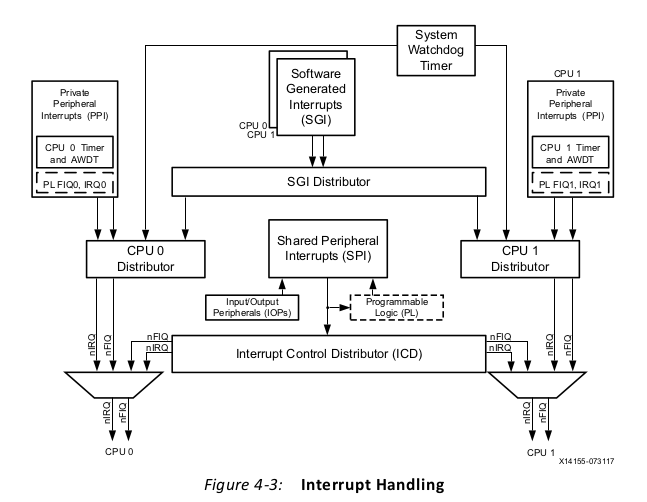
These per-processor-core interrupts have limited function.

•Shared Peripheral Interrupts (SPIs):

These interrupts are shared between the processor cores.

•Software Generated Interrupts (SGIs):

These are synchronous interrupts generated by writing an interrupt number on a particular SGI register on the system.



**Timers**

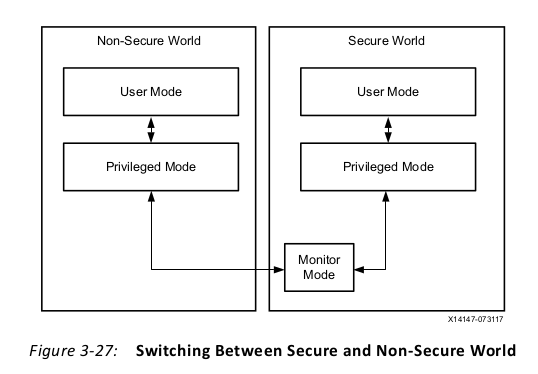
Timer interrupts are often essential to system operation, because they are used for scheduling and time-slicing tasks.

**MMU Configurations**

**Secure Configurations**

A processor switches between two separate worlds, secure world and normal world,

through a processor mode called monitor mode.



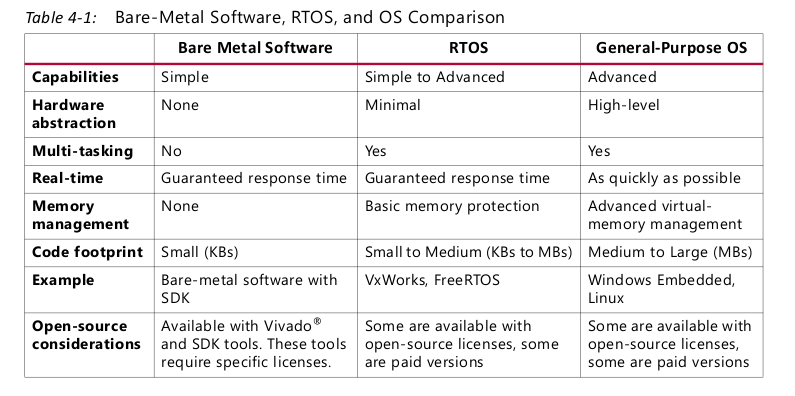
2、OS and RTOS Choices

**Types of Embedded-System Software**

An OS or RTOS can support multitasking and manage resource usage. Bare-metal software can serve as a minimal implementation of necessary system-management functions.

A full-featured monolithic OS like Linux uses the MMU for secure and non-secure memory

regions, whereas a microkernel RTOS does not use the MMU and thus will not offer the same memory protection as a monolithic kernel.



**Application Requirements:**

° If an application needs advanced software with ready-made libraries, then consider

a full-featured OS, such as Linux.

° If an application needs hard real-time response for many tasks, then consider an

RTOS like freeRTOS.

°If an application needs both a full-featured OS and real-time response, then

consider a multi-OS environment like Linux Symmetric Multi-Processing (SMP) with

real-time patches, or asymmetric multiprocessing (AMP) with Linux and RTOS

running on the dual ARM cores or hypervisor.

**Hardware Requirements for Running an OS or RTOS**

• System Timer: This is a hardware timer for the kernel heartbeat, like jiffies in the Linux

OS. The timer is used by the OS or RTOS for all time-related kernel activities, such as

scheduling and delays.

• Memory Footprint: As low as 5 to 30 KB of RAM and ROM for an RTOS. A few KBs to

100 MBs for Linux OS, based on the kernel and services configuration selections.

• Memory Management Unit (MMU): Required for a full-featured, general-purpose OS

like Linux. May be used for memory protection by some RTOS implementations.

**Linux OS and PetaLinux Tools**

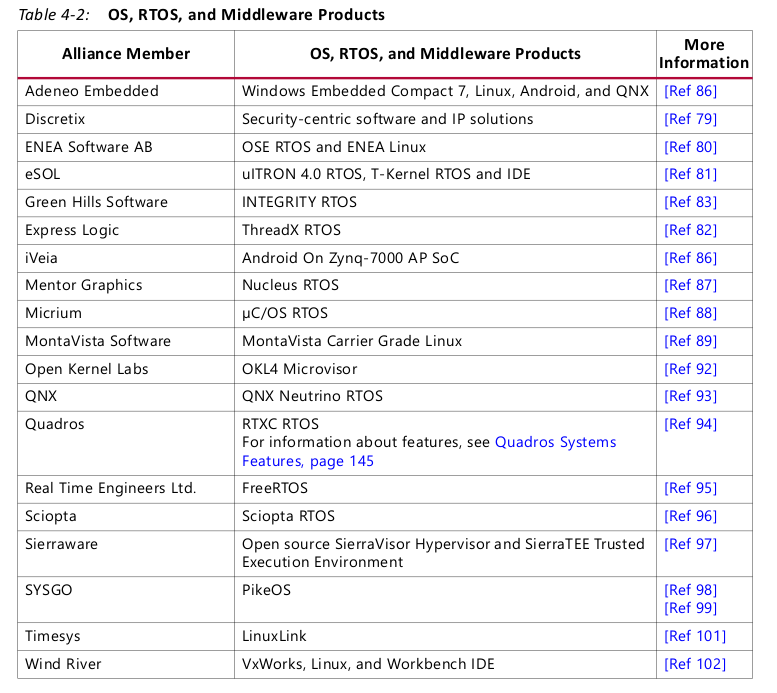
https://www.xilinx.com/tools/petalinux-sdk.htm

http://www.wiki.xilinx.com/Linux

http://www.wiki.xilinx.com/petalinux

**OS and RTOS Ecosystem**

https://www.xilinx.com/products/silicon-devices/soc/zynq-7000/ecosystem/index.htm



**Multi-OS**

There are multiple ways to enable multiple operating systems running on both CPU cores of

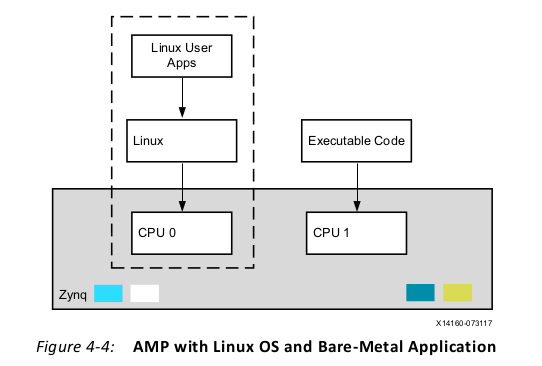
the dual-core ARM Cortex-A9 MPCore processor in the Zynq-7000 AP SoC.

[http://www.wiki.xilinx.com/Multi-OS+Support+%28AMP+%26+Hypervisor%29](http://www.wiki.xilinx.com/Multi-OS+Support+(AMP+%26+Hypervisor))

**Simple AMP（asymmetric multi-processing）**

A simple multi-OS AMP implementation can consist of running, for example, Linux OS and

a bare-metal application or of running two bare-metal applications.



refer to PetaLinux Tools User Guide: Zynq All Programmable SoC Linux-FreeRTOS AMP Guide (UG978)

**ARM Cortex-A9 TrustZone**

The Secure domain has the same capability as the Normal domain but it operates in a separate memory space. A Secure Monitor acts as a virtual gatekeeper to control migration between the domains.

**Development and Debugging Tools**

Xilinx Software Development Kit

ARM DS-5 Development Studio

IAR ref to <http://www.iar.com/Products/RTOS/Integrated-RTOSes/>

3、Libraries and Middleware

**Libraries**

Libraries support static and dynamic linking.

Libraries can be categorized into commonly used libraries and domain-specific libraries:

• Examples of commonly used libraries include GNU C, GNU C++ library, POSIX Pthread

libraries.

• Examples of domain-specific libraries include Intel Threading Building Blocks (parallel

processing), FFmpeg audio, OpenCV, direct rendering manager (DRM), kernel mode

setting (KMS), ARM OpenMAX DL sample software library (audio/video processing),

and MATLAB engine library (mathematical processing).

**Xilinx Libraries for Stand-Alone Systems**

• libxil.a — Device drivers for peripherals.

• LibXil MFS — A memory file system.

• LibXil FFS — A generic FAT file system based on an open source implementation. It is

primarily used with the SD/eMMC driver and a glue layer is implemented to link it to

that driver.

• LibXil Flash — A library that provides read, write, erase, lock, unlock, and

device-specific functions for parallel flash devices.

• LibXil Isf — An in-system flash library that supports the Xilinx in-system flash hardware

and serial flash on SPI/QSPI.

• LibXil SKey — The LibXil SKey library provides a programming mechanism for

user-defined eFUSE bits. The PS eFUSE holds the RSA primary key hash bits and user

feature bits, which can enable or disable some Zynq-7000 AP SoC processor features.

• lwIP — A third-party, light-weight TCP/IP networking library.

<http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Spectrum+Analyzer+part+2-Building+ARM+NEON+Library+Tech+Tip>

The Tech Tip describes the process of obtaining and building a set of filtering functions

targeting the Zynq-7000 AP SoC ZC702 platform. Many applications that can take

advantage of the processing capabilities of the Zynq-7000 AP SoC involve complex

calculations used in filtering, video manipulation, and signal processing. This has

evolved into the Ne10 project and the Ne10 library.

**Middleware**

Middleware is a set of libraries that runs between the operating-system and application

layers. In embedded systems, middleware is system software that sits on top of the

operating system or is sometimes included as part of the operating system.

**usecase**

The lwIP stack supports IP, ICMP, IGMP, UDP, TCP, and other networking protocols. By using the lwIP stack, you can focus on developing your application core and leave the TCP/IP implementation to the lwIP stack. Ref to LightWeight IP (lwIP) Application Examples (XAPP1026)

4、Boot Loaders

A boot loader is the software that initializes the system in preparation for execution of the

next level of software, such as an operating system. Usually, each operating system has a set

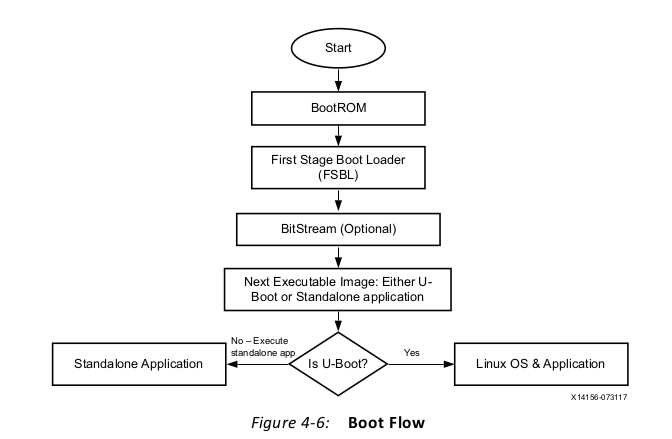
of boot loaders specific for it. Boot loaders usually contain several ways to boot the OS

kernel and also contain commands for debugging and/or modifying the kernel

environment.

**The Zynq-7000 AP SoC Boot Process**

Stage 0, 1, 2



5、Software Development Tools

**Stage 0**

At power-on or reset, the ARM core runs initialization code from the BootROM. The

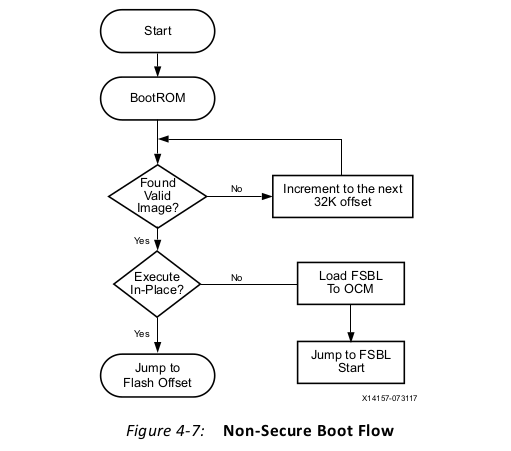
BootROM code cannot be changed; it is a factory pre-programmed code that comes with

each Zynq-7000 AP SoC. The BootROM code determines the device on which the next-level

loader is located by reading the boot-mode pins. Depending on the boot-mode setting, the

FSBL is copied from either NAND, parallel NOR, serial NOR (Quad-SPI), or Secure Digital

(SD) flash memories to the OCM.



**Stage 1**

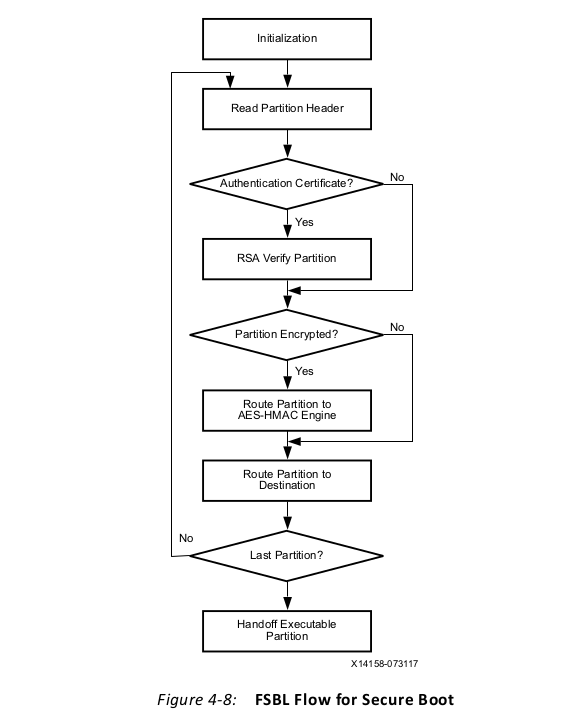
The FSBL gets execution control from the BootROM and either runs from the OCM or

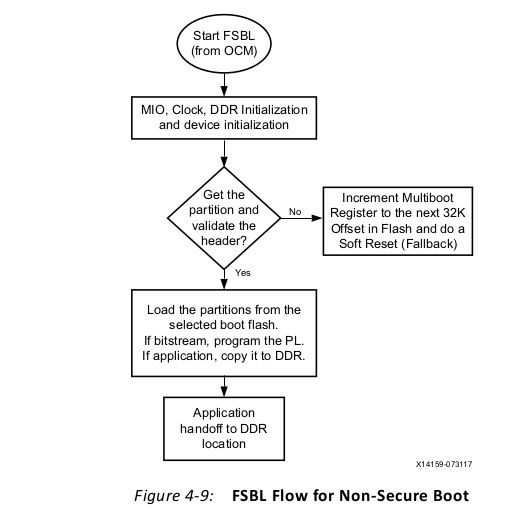
execute-in-place flash based on the boot mode settings. The FSBL initializes the PS and

looks for a bit file in the boot device. If found, the FSBL writes the bit file to the PL.

Typically, the FSBL initializes the external RAM and loads the second-stage boot loader

(SSBL) or a stand-alone application.





**Stage 2**

This stage could be a second-stage boot loader (SSBL) like U-Boot, or it could be an RTOS

or application. In the case of a full-featured OS like embedded Linux, the SSBL U-Boot runs

in CPU0 to initialize and set up the environment in which the OS will boot.

In the case of Linux, the OS detects and enables the second processor core, configures and

activates the MMU and data caches, and performs other actions to make a complete system

available to applications.

**Software Development Tools**

Xilinx provides a variety of software development tools that can be used to build various

**software components** used by the Zynq-7000 AP SoC, including:

• Board Support Package (BSP). This is a set of APIs used to access low-level hardware.

• Stand-alone (bare-metal) applications. These are simple applications that do not

support complex kernel features such as multi-tasking. Such applications use the BSP

APIs to access low-level hardware.

• FSBL (first stage boot loader). This is a small application that performs PS hardware

initialization, loads the fabric with a bit stream, optionally loads additional data, and

loads the second-stage boot loader. The FSBL is an example of a stand-alone

(bare-metal) application.

• U-Boot (second stage boot loader). This boot loader performs the necessary

hardware initialization for the kernel to begin execution. When done, the Linux kernel is

loaded and starts executing.

• Linux kernel. Linux is an open source operating system. The **hardware-dependent**

parts of the Linux kernel (device drivers, etc.) are provided for the Zynq-7000 AP SoC.

The **hardware independent** parts of the kernel (file systems, networking, etc.) are similar

to other Linux machines. The kernel is configured as needed and targeted to the ARM

processor.

• User applications. Typical user applications run in the Linux environment and are

hardware independent.

• Device tree blob. The device tree is a structure of nodes and properties describing

components and features supported by the hardware (PS peripherals, ARM processors,

and PL peripherals).

The software development tools for Zynq-7000 AP SoCs include:

•GUI-based tool chain.

•Command-line based tool chain.

•Hybrid tool chain.

**Stand-Alone Platform**

In a stand-alone platform, an application is built as monolithic, executable code.

Multi-tasking is not supported, so device drivers are loaded with the application and the

application invokes the driver APIs directly.

Stand-alone platforms are used by small, dedicated systems that do not require an

operating system. The advantages include:

• Application development is typically simple and quick.

• The overall complexity is minimized because context switching and multi-tasking is not

used.

• Developers can have a more detailed and comprehensive understanding of the

top-to-bottom system implementation.

• Ideal for the initial development stages of new hardware.

• A console (text) based application can be built using this platform.

The disadvantages include:

• A complex application system requiring multi-tasking cannot be built.

• Not ideal for GUI-based applications.

**Linux Platform**

Linux is a multi-process operating system. The applications use the Linux library APIs

and therefore access the hardware resources in an abstract manner. The procedure for

creating applications in the SDK workspace is similar to the stand-alone platform except the

Linux-platform BSP is selected.

The Zynq-7000 AP SoC supports launch of the GNU tool chain in two ways:

• GNU open-source tool chain

• PetaLinux framework <https://www.xilinx.com/tools/petalinux-sdk.htm>

六、硬件设计流程

• Using the Vivado IDE to Build IP Subsystems

• Rule-Based Connection

• Creating Hierarchical IP Subsystems

• Board Window

• Generating Block Designs

• Creating and Packaging IP for Reuse

• Creating Custom Interfaces: requires that all memory-mapped interfaces use an AXI interface.

• Managing Custom IP

• Vivado High-Level Synthesis (HLS)

**Overview**

The IP integrator enables the creation of block designs. These block designs are essentially

IP subsystems containing any number of user-configured IP and interconnect. IP integrator

is the feature for doing embedded-processor design in Vivado design tools with Zynq-7000

AP SoCs, MicroBlaze™ processor designs, and non-processor-based designs.

**Using the Vivado IDE to Build IP Subsystems**

Using the GUI to Create an IP Subsystem

Using a Scripted Flow to Create an IP Subsystem

Using designer Assistance: Block Automation, Connection Automation

**Vivado High-Level Synthesis (HLS)**

The high-level synthesis (HLS) tool transforms a C, C++, OpenCL Kernel, or SystemC design

specification into a register transfer level (RTL) implementation that in turn can be

synthesized into a Xilinx All Programmable device.

HLS performs two types of synthesis on the design:

• Algorithm Synthesis: This synthesizes the functional statements into RTL statements

over potentially multiple clock cycles.

• Interface Synthesis: This transforms the function arguments (or parameters) into RTL

ports with specific timing protocols, allowing the design to communicate with other

designs in the system.

Typically, HLS is used after a software bottleneck has been identified by profiling the

software application within the SDK or third-party tools. When the bottleneck function(s)

have been identified, the next step is to move those function(s) into hardware using HLS.

Finally, the HLS design can be exported as an IP core to be used in IP integrator. HLS also

generates a simulation model for the IP core that can be used to verify the IP function. Refer

to the Vivado Design Suite User Guide: High-Level Synthesis (UG902).

**Summary**

In general, the following steps should be followed to capture an embedded design in IP

integrator:

1. Add processor IP, such as the Zynq-7000 AP SoC or the MicroBlaze processor.

2. When the processor IP is instantiated, designer assistance is available. Use designer

assistance to configure the processor and peripherals.

3. Customize the processor further, if needed. This step is needed if more function and

control over clocks, resets, I/O ports, etc., is desired.

4. Add peripherals, and connect them using designer assistance, when available.

5. Add connectivity IP for external interfaces, such as GPIO, Ethernet, etc.

6. Add custom accelerators for the Processing Logic, if needed.

7. Connect and review clock and reset domains, using the Signals tab or the Make

Connection wizard.

8. Run design-rule checks by validating design. Resolve any errors or warnings flagged

during design validation, and run design validation until no further errors are flagged.

9. Synthesize, implement, and generate bit-stream for the design.

10. Export the design to the SDK for software development.

七、软件设计流程

八、调试