



基于*DSPs*的系统开发过程

www.farsight.com.cn

今天的内容

- ✓ 1) TMS320 C6000系列DSP的关键技术，包括哈佛总线，多MAC，流水，多线程等内容。
- ✓ 2) CCS的使用
- ✓ 3) BIOS核心技术，包括HWI，SWI，TASK等的调度。
- ✓ 4) DMA关键外设的使用
- ✓ 5) 线性汇编优化代码等。
- ✓ 6) 现场编程演示与问题交流。

1、DSP概述

✓ 1、DSP（Digital Signal Processing）也就是我们常说的数字信号处理，它是利用计算机或专用处理设备，以数字形式对信号进行采集，变换，滤波，估值，增强，压缩，识别等处理，以得到符合人们需要的信号形式。

✓ 狭义的DSP是指，DSP芯片就是一种特别适合于进行数字信号处理运算的微处理器，其主要应用是实时快速地实现各种数字信号处理算法。

Why DSP?

√2、DSP 的优势

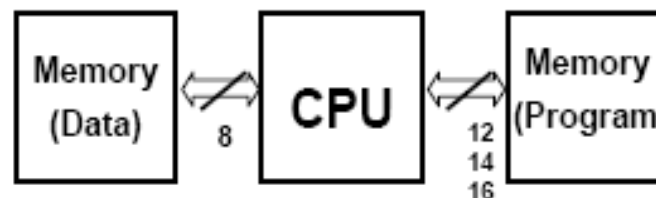
- √1) . 在一个指令周期内可完成一次乘法和一次加法;
- √2) . 程序和数据空间分开, 可以同时访问指令和数据;
- √3) . 片内具有快速RAM, 通常可通过独立的数据总线在两块中同时访问;

Why DSP?

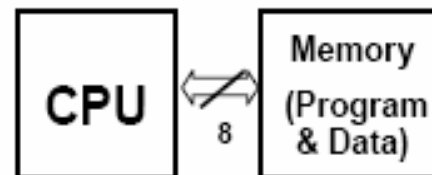
- ✓4) .具有低开销或无开销循环及跳转的硬件支持;
- ✓5) .快速的中断处理和硬件I/O支持;
- ✓6) .具有在单周期内操作的多个硬件地址产生器;
- ✓7) .可以并行执行多个操作;
- ✓8) .支持流水线操作,使取指,译码和执行等操作可以重叠执行。

2、DSP关键技术?

√ 1、什么是哈佛总线?



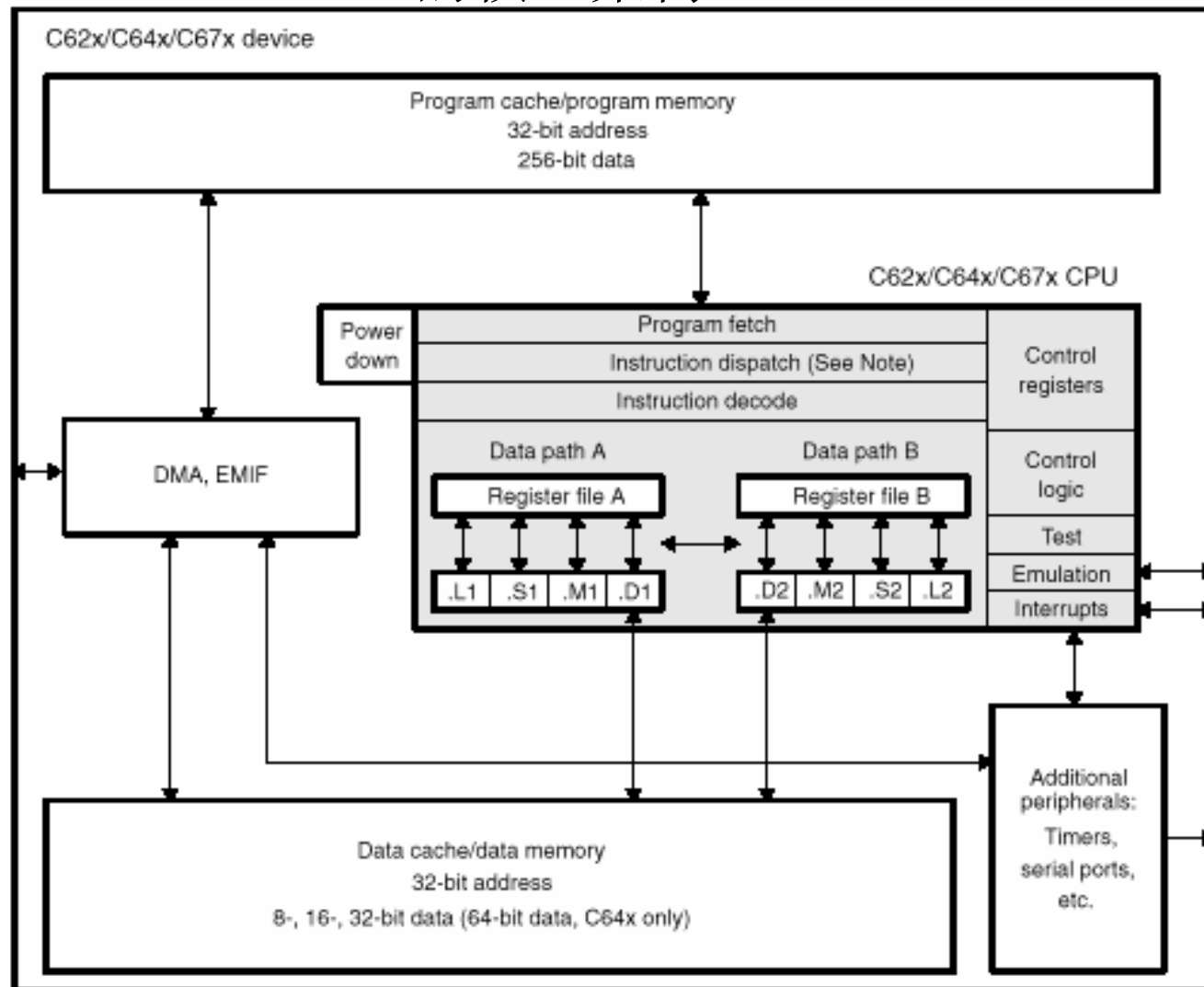
√ 冯. 诺依曼接口



√ 核心就是将数据总线与程序总线独立开来

2、*DSP*关键技术

•2、C6000的核心架构

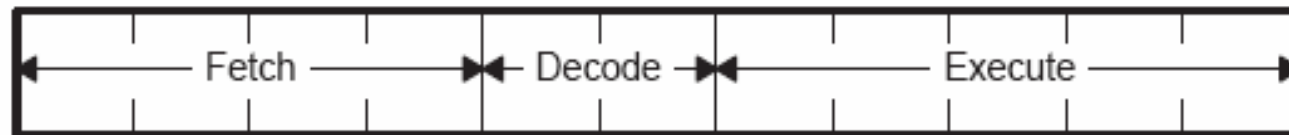


√3、流水操作（pipeline）

√1）取指

√2）解码，包括指令派遣，指令译码

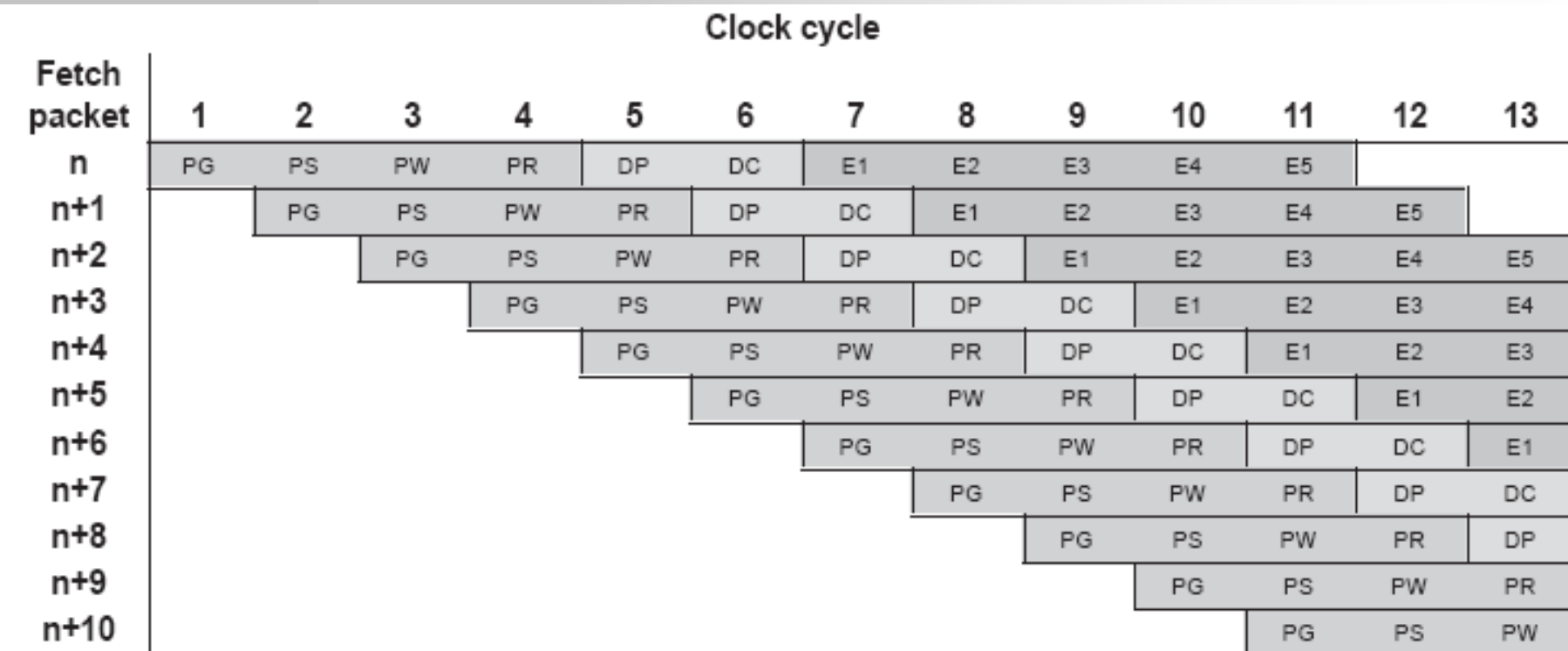
√3）执行



远见品质

DSP关键技术

流水操作示意图



DSP关键技术

- √ **数据格式**: 定点/浮点
- √ **数据宽度**: 16bit/32bit
- √ **速度**: MIPS, FLOPS, 每秒乘加运算次数
- √ **存储器结构**: 一个MAC需要一个指令周期
读一个指令字和2个数据字
- √ **开发配套工具的完善**: CCS
- √ **I/O能力和多片互连能力**: link口
- √ **功耗和电源管理**: 休眠/等待模式
- √ **成本**

✓ 当有如下需求时使用DSPs:

∅ Cost saving.

∅ Smaller size.

∅ Low power consumption.

∅ Processing of many “high” frequency signals in real-time.
—————→ 实时数字信号处理

} 嵌入式

✓ 当有如下需求时使用GPPs::

∅ Large memory.

∅ Advanced operating systems.

3、1 DSPs系统的系统设计： 设计流程

DSPs设计可以被分为三种主要的任务：

1. **-功能划分：** 把需要实现的功能划分成更小的、相互作用的模块；
2. **-模块分配：** 把那些模块分配到微处理器或者是其它硬件单元中，由微处理器上运行的软件或者直接由硬件执行功能
3. **-调度安排：** 调度安排好各功能的执行时间，这对于几个功能模块共享一个硬件单元的时候尤其重要。

补： DSPs系统的迭代开发：原型验证

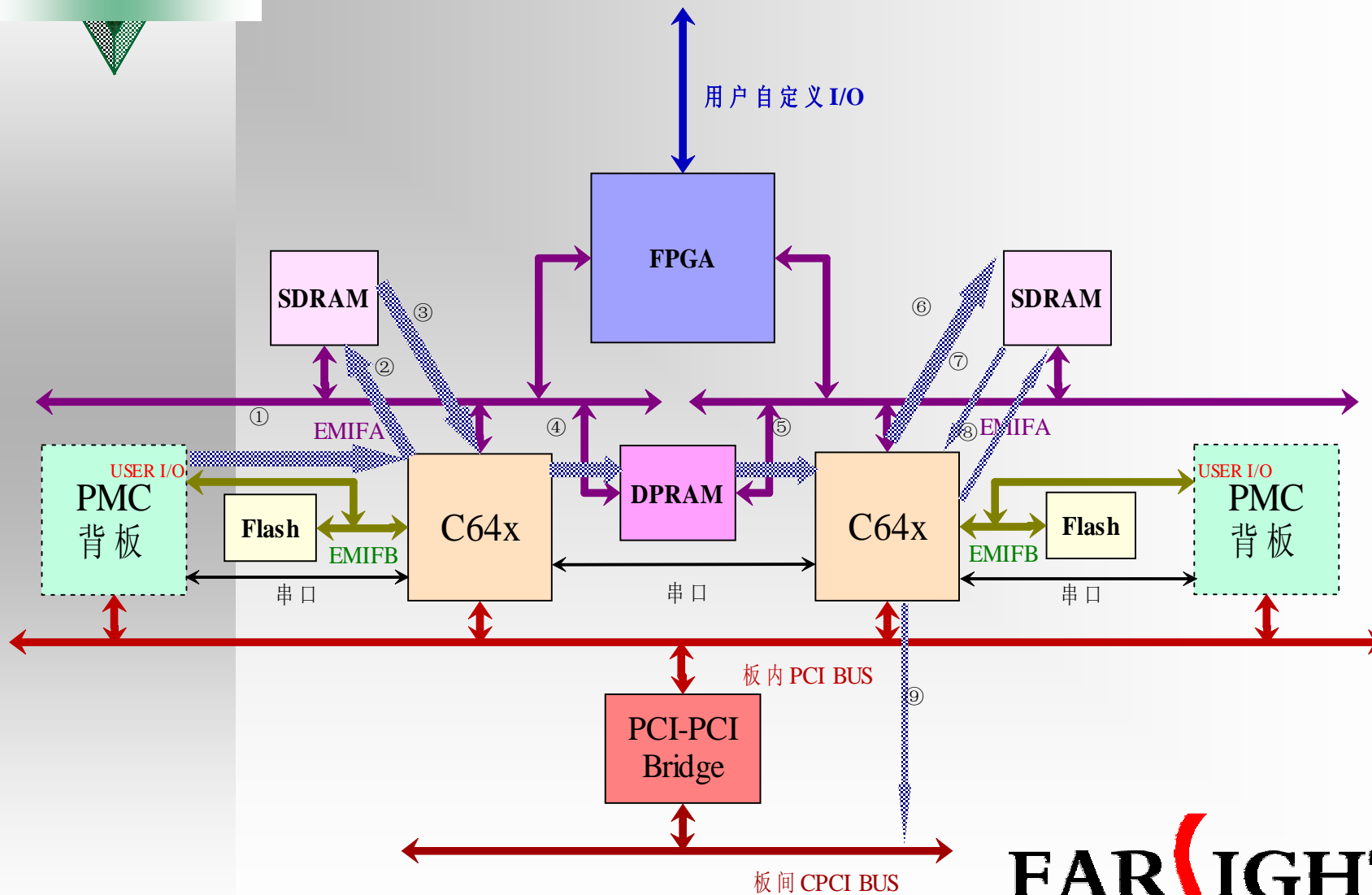
FAR  **IGHT**

DSPs系统的系统设计的重要指标

- ✓ 算法 → 运算量
- ✓ 算法 → 数据率
输入数据率，输出数据率
- ✓ 运算量对DSPs的CPU核的运算单元的压力
- ✓ 数据率对总线和接口有压力

远见品质

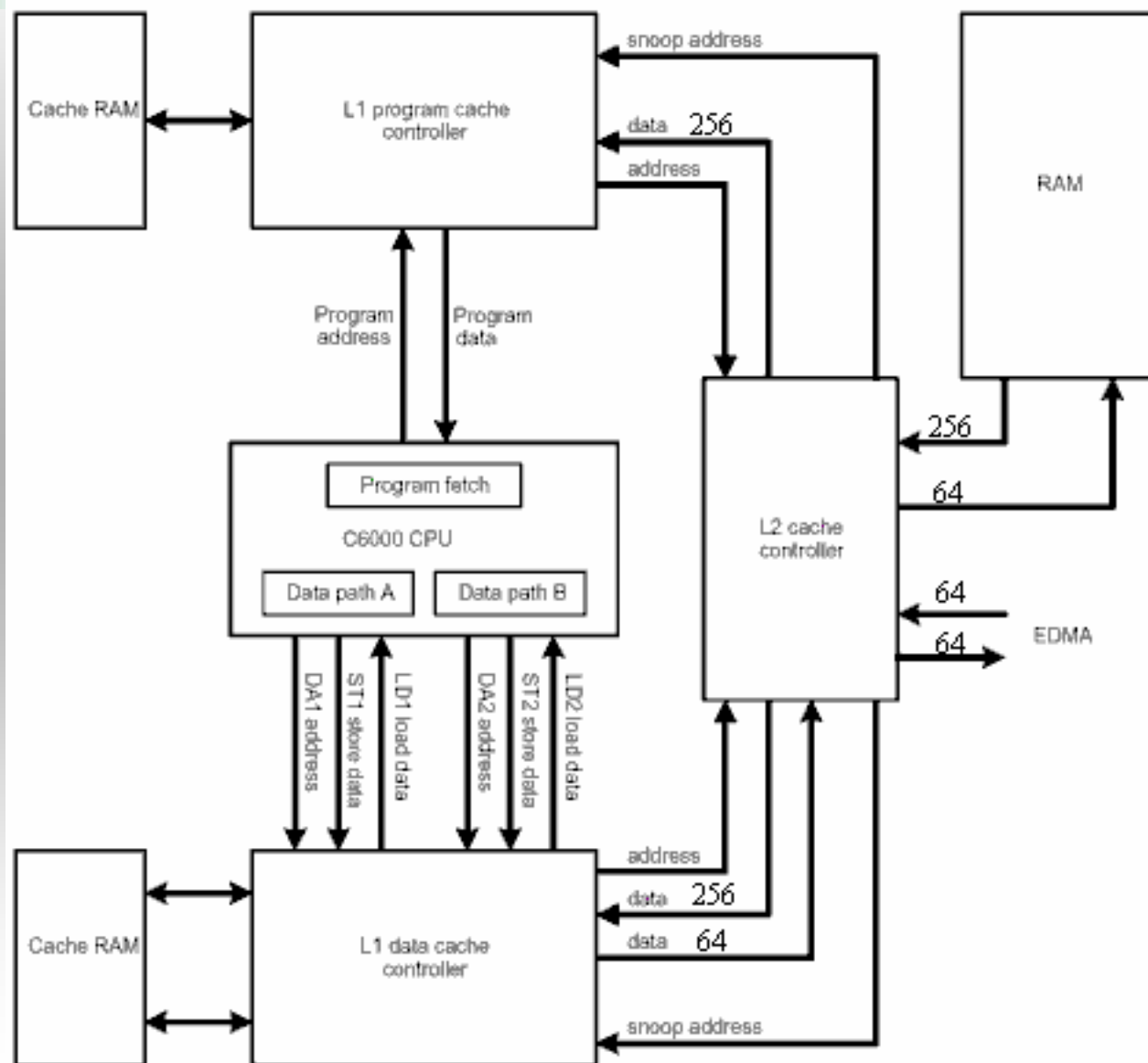
例子：双C64板的数据流图



FAR SIGHT

远见品质

C6416结构图



GHT

远见品质

数据率 (IO总线压力分析)

序号	功能	数据源	数据目的	使用的板内总线	数据率(Mbyte/s)
①		PMC 数字接收板	C6416_A	C6416_A 的 EMIFB	53.4
②		C6416_A	C6416_A 的 SDRAM	C6416_A 的 EMIFA	53.4
③		C6416_A 的 SDRAM	C6416_A	C6416_A 的 EMIFA	53.4
④		C6416_A	双口 RAM	C6416_A 的 EMIFA	53.4
⑤		双口 RAM	C6416_B	C6416_B 的 EMIFA	53.4
⑥		C6416_B	C6416_B 的 SDRAM	C6416_B 的 EMIFA	53.4
⑦		C6416_B 的 SDRAM	C6416_B	C6416_B 的 EMIFA	26.7
⑧		C6416_B	C6416_B 的 SDRAM	C6416_B 的 EMIFA	26.7
⑨		C6416_B	终端计算机	板内 PCI 和系统 PCI	---

C6416 的总线压力

	C6416_A	C6416_B
EMIFA	54%	45%
EMIFB	27%	0
L2	11%	6%

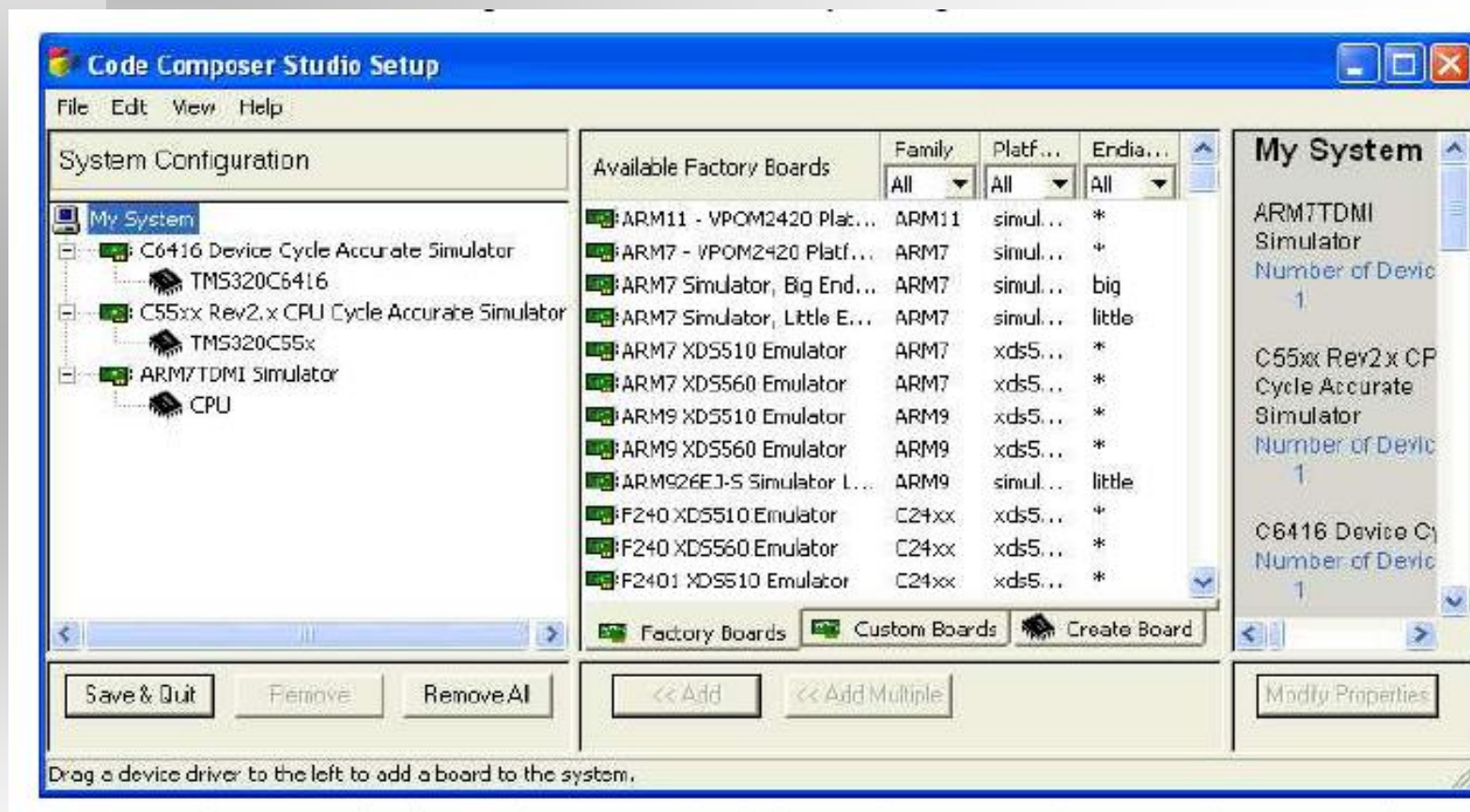
3、2 DSPs系统的硬件和软件的协同设计

- √-软件开发快、灵活性好、容易修改，但处理速度慢
- √-硬件处理速度快、实时性好，但成本高、修改困难
例如：FPGA对大量数据的存储困难，数据调度复杂。所以FPGA适合做处理量大但是数据调度简单的运算。

远见品质

3.3 CCS的使用

√1、CCS Setup 环境的配置



FAR SIGHT

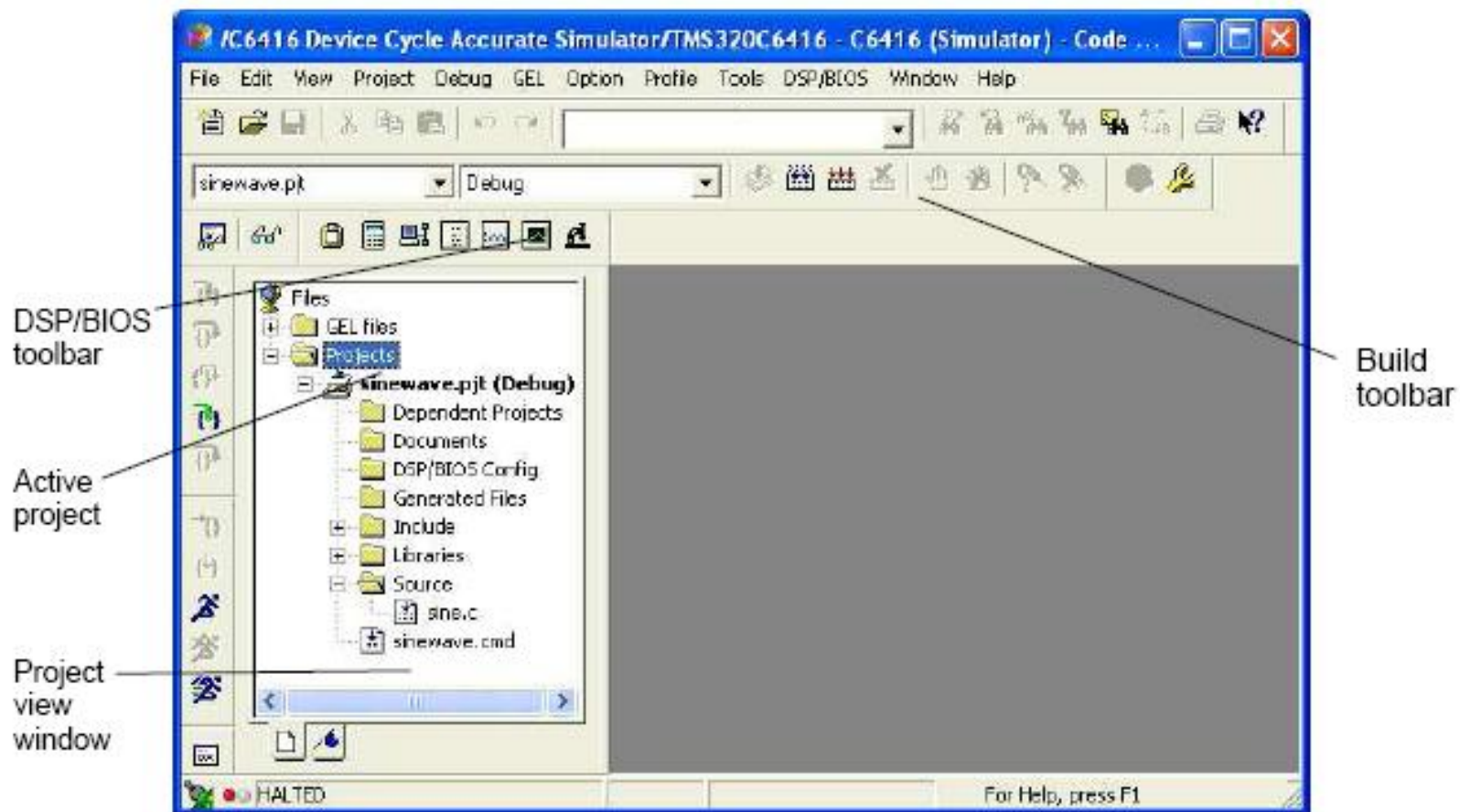
3.3 CCS的使用

- √2、CCS 集成开发环境功能
- √1) C, 汇编语言的编辑与编译
- √2) 代码调试
- √3) 代码优化
- √4) 强大的软件仿真功能
- √5) spru509f.pdf

远见品质

3.3 CCS的使用

3、CCS 集成开发环境介绍



3.3 CCS的使用

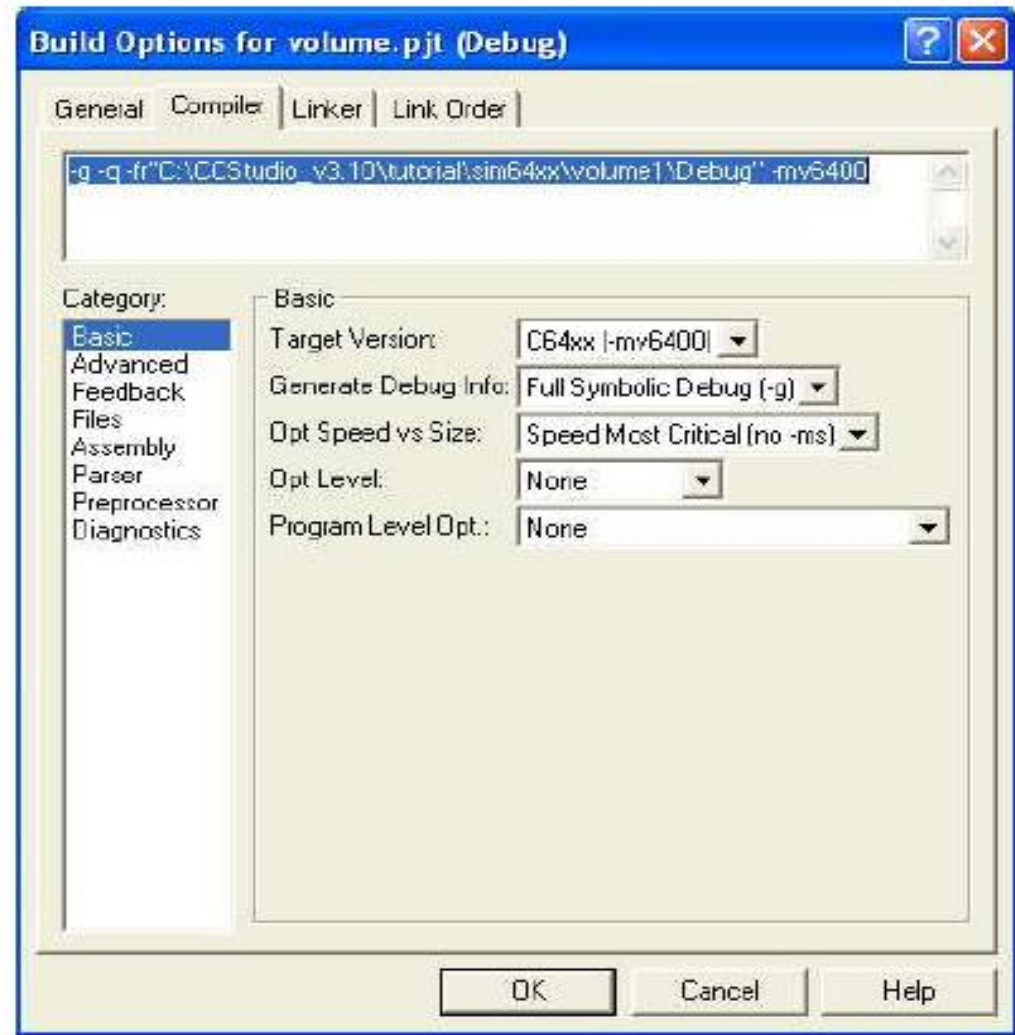
- ✓ 软件和硬件并行开发
开发时间压力
- ✓ DSPs的Simulator支持对硬件中断的
模拟功能和数据输入输出的功能
- ✓ TI CCS的pin connect和FILE IO
- ✓ DSK上的硬件验证, Timer驱动中断

远见品质

3.3 CCS的使用

✓ Build的功能

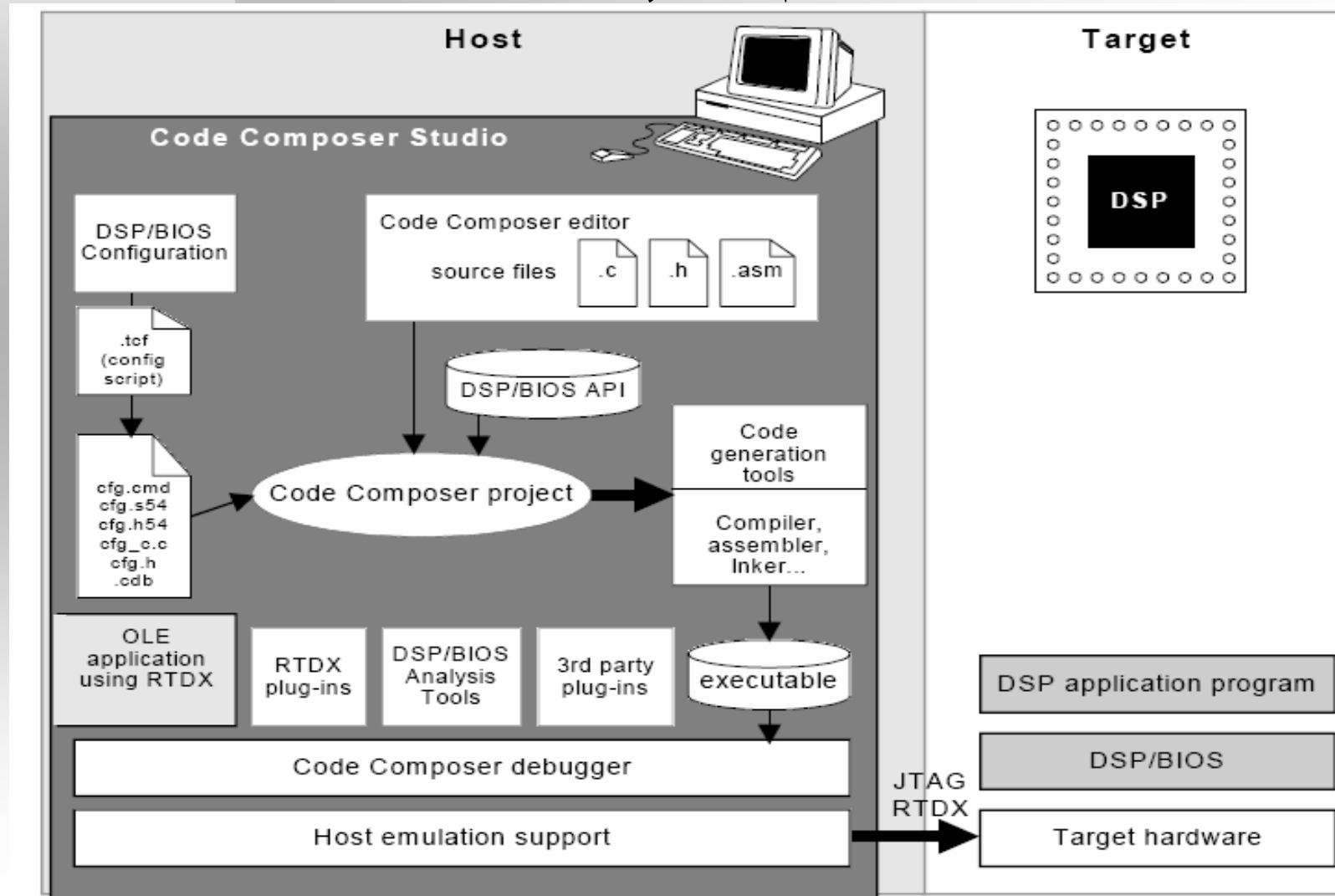
✓ (详细使用过会电脑现场演示)



远见品质

3.4 BIOS核心技术

1、BIOS 的组件



T

3.4 BIOS核心技术

√2、BIOS配置组

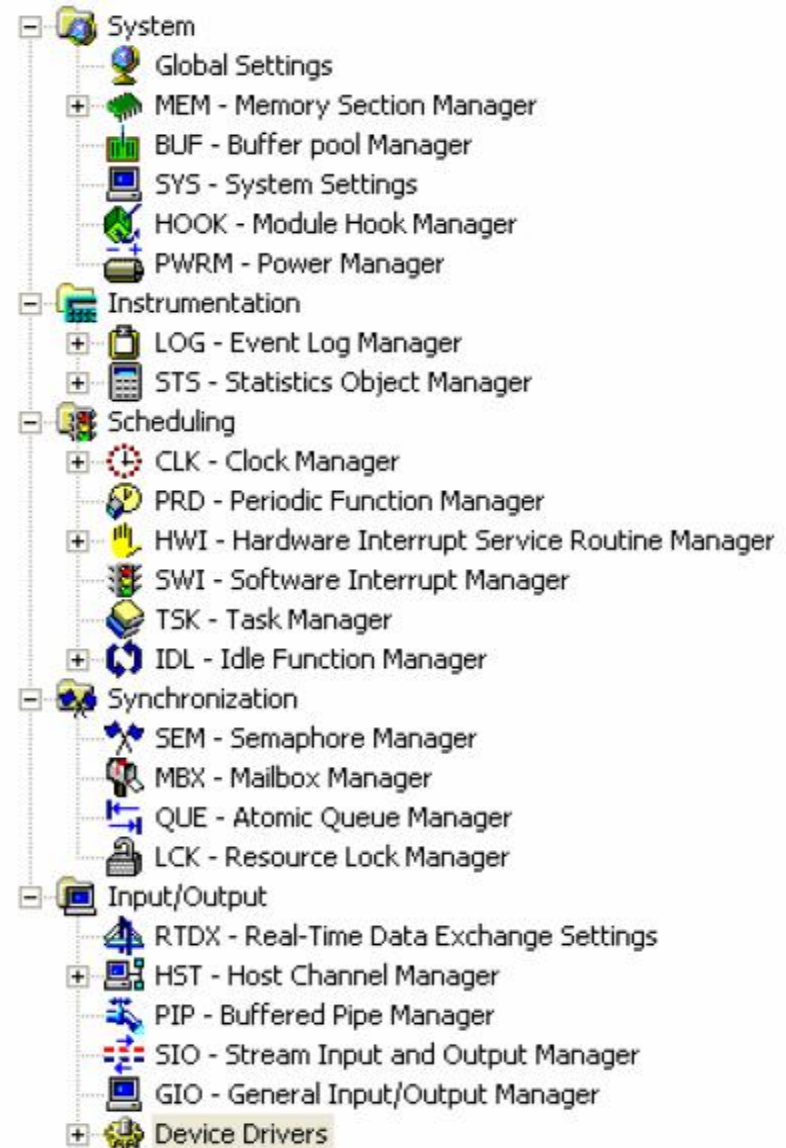
√1) 系统

√2) 设备

√3) 调度

√4) 同步

√5) 输入输出



3.4 BIOS核心技术

3、线程类型以及优先级别

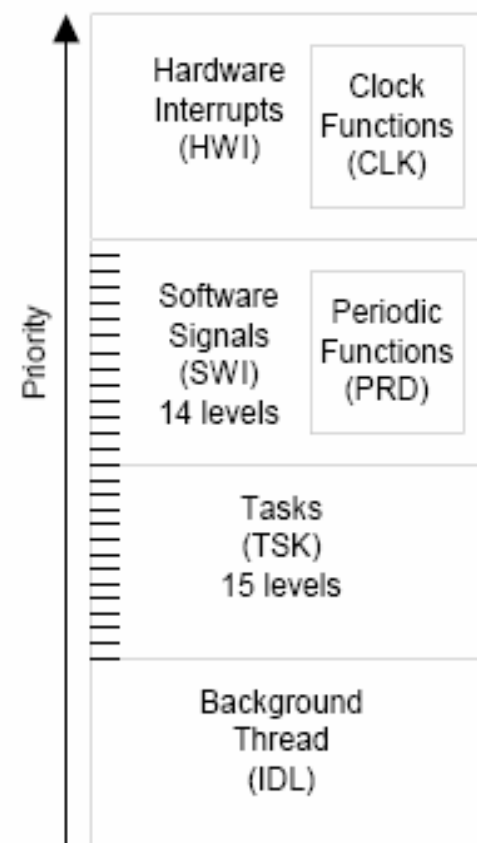
✓ 1) HWI

✓ 2) SWI

✓ 3) TASK

✓ 4) Idle

4、线程之间的比较



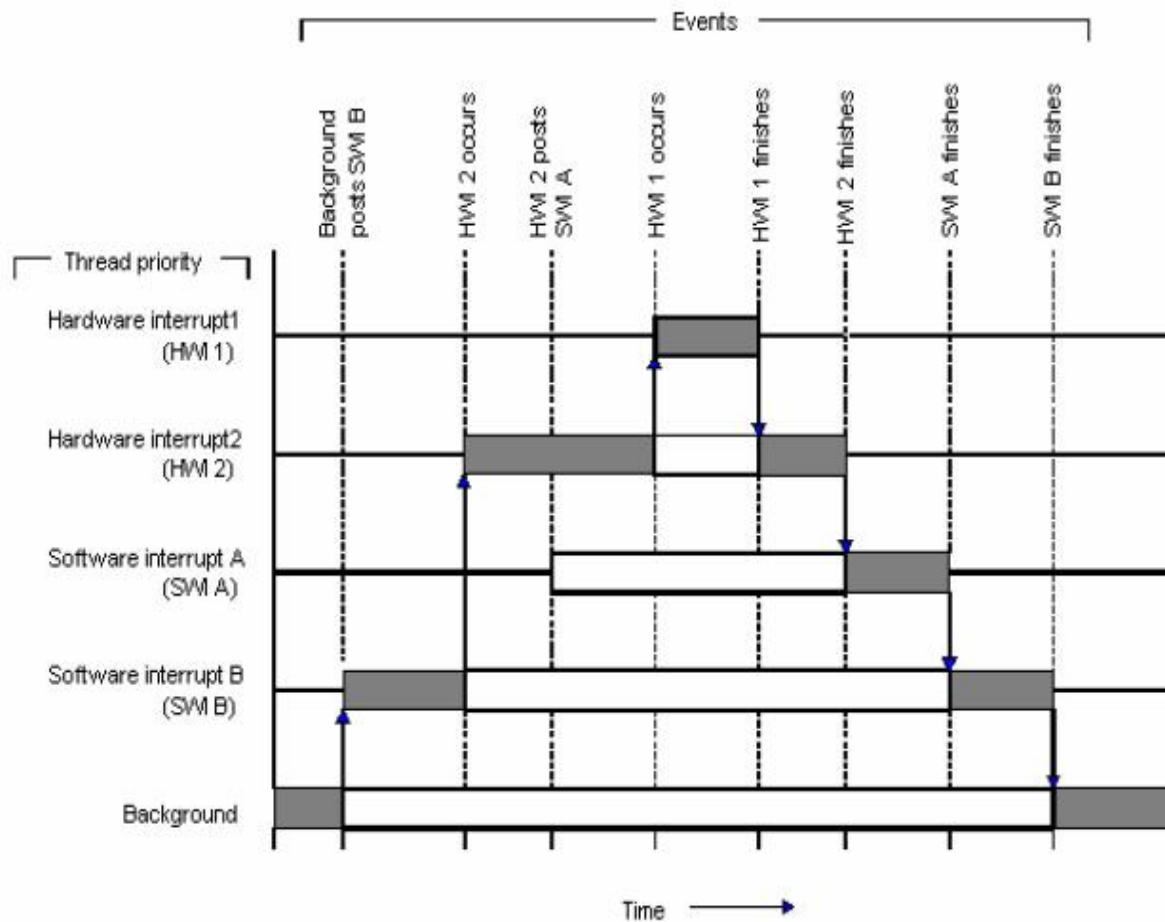
3.4 BIOS核心技术

Characteristic	HWI	SWI	TSK	IDL
Priority	Highest	2nd highest	2nd lowest	Lowest
Number of priority levels	DSP-dependent	15. Periodic functions run at priority of the PRD_swi SWI object. Task scheduler runs at lowest priority.	16 (Including 1 for the ID loop)	1
Can yield and pend	No, runs to completion except for preemption	No, runs to completion except for preemption	Yes	Should not; would prevent PC from getting target information
Execution states	Inactive, ready, running	Inactive, ready, running	Ready, running, blocked, terminated	Ready, running
Scheduler disabled by	HWI_disable	SWI_disable	TSK_disable	Program exit
Posted or made ready to run by	Interrupt occurs	SWI_post, SWI_andn, SWI_dec, SWI_inc, SWI_or	TSK_create	main() exits and no other thread is currently running
Stack used	System stack (1 per program)	System stack (1 per program)	Task stack (1 per task)	Task stack used by default (see Note 1)
Context saved when preempts other thread	Customizable	Certain registers saved to system stack (see Note 2)	Entire context saved to task stack	--Not applicable--

3.4 BIOS核心技术

Characteristic	HWI	SWI	TSK	IDL
Context saved when blocked	--Not applicable--	--Not applicable--	Saves the C register set (see optimizing compiler user's guide for your platform)	--Not applicable--
Share data with thread via	Streams, queues, pipes, global variables	Streams, queues, pipes, global variables	Streams, queues, pipes, locks, mailboxes, global variables	Streams, queues, pipes, global variables
Synchronize with thread via	--Not applicable--	SWI mailbox	Semaphores, mailboxes	-Not applicable--
Function hooks	No	No	Yes: initialize, create, delete, exit, task switch, ready	No
Static creation	Included in default configuration template	Yes	Yes	Yes
Dynamic creation	Yes (see Note 3)	Yes	Yes	No
Dynamically change priority	No (see Note 4)	Yes	Yes	No
Implicit logging	None	Post and completion events	Ready, start, block, resume, and termination events	None
Implicit statistics	Monitored values	Execution time	Execution time	None

v5、多线程调度实例

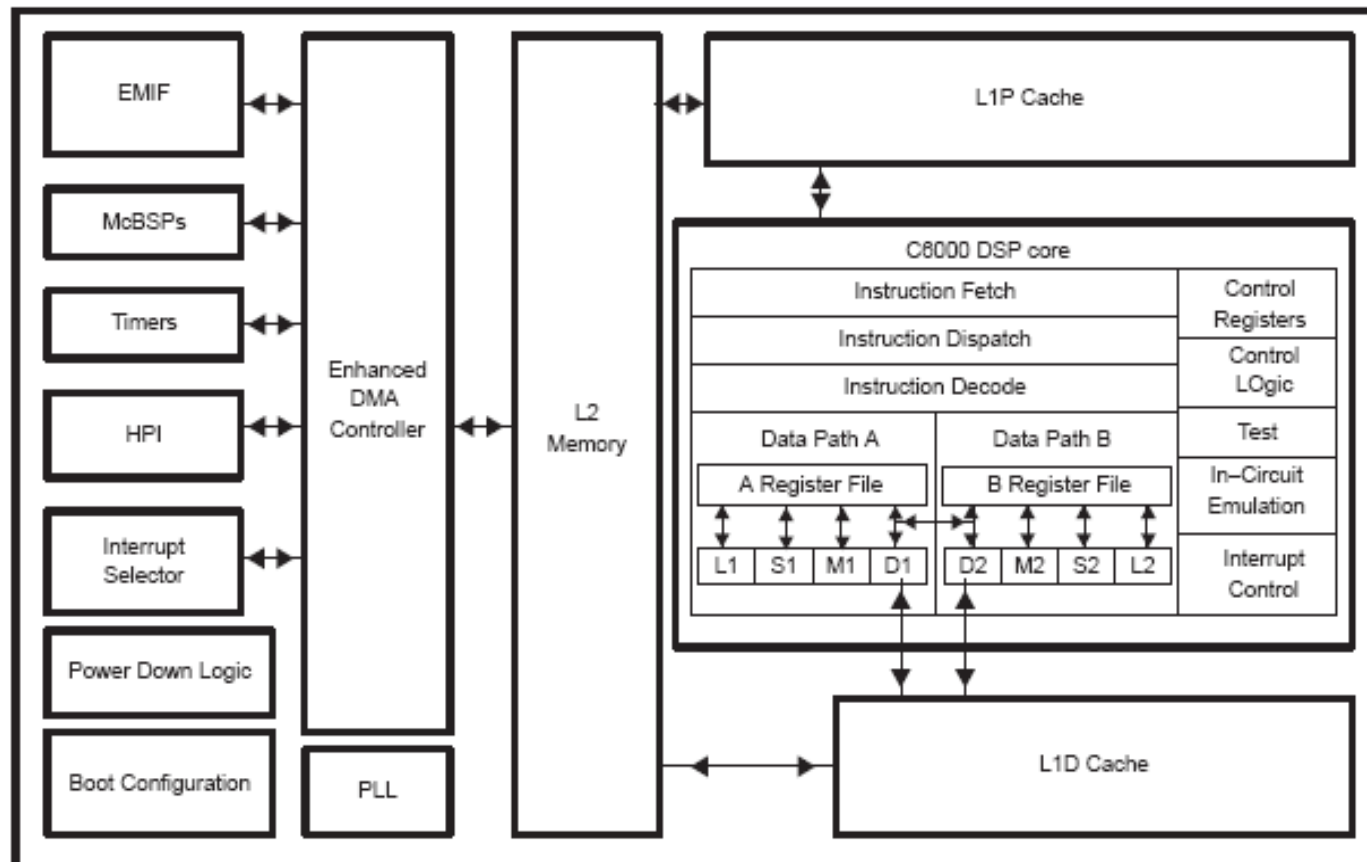


3.5 DMA关键外设的使用

- √ 1、C6000的主要外部设备
- √ 1) HPI
- √ 2) EMIF
- √ 3) DMA
- √ 4) McBSP
- √ 5) TIMER等

远见品质

3.5 DMA关键外设的使用



FAR  **IGHT**

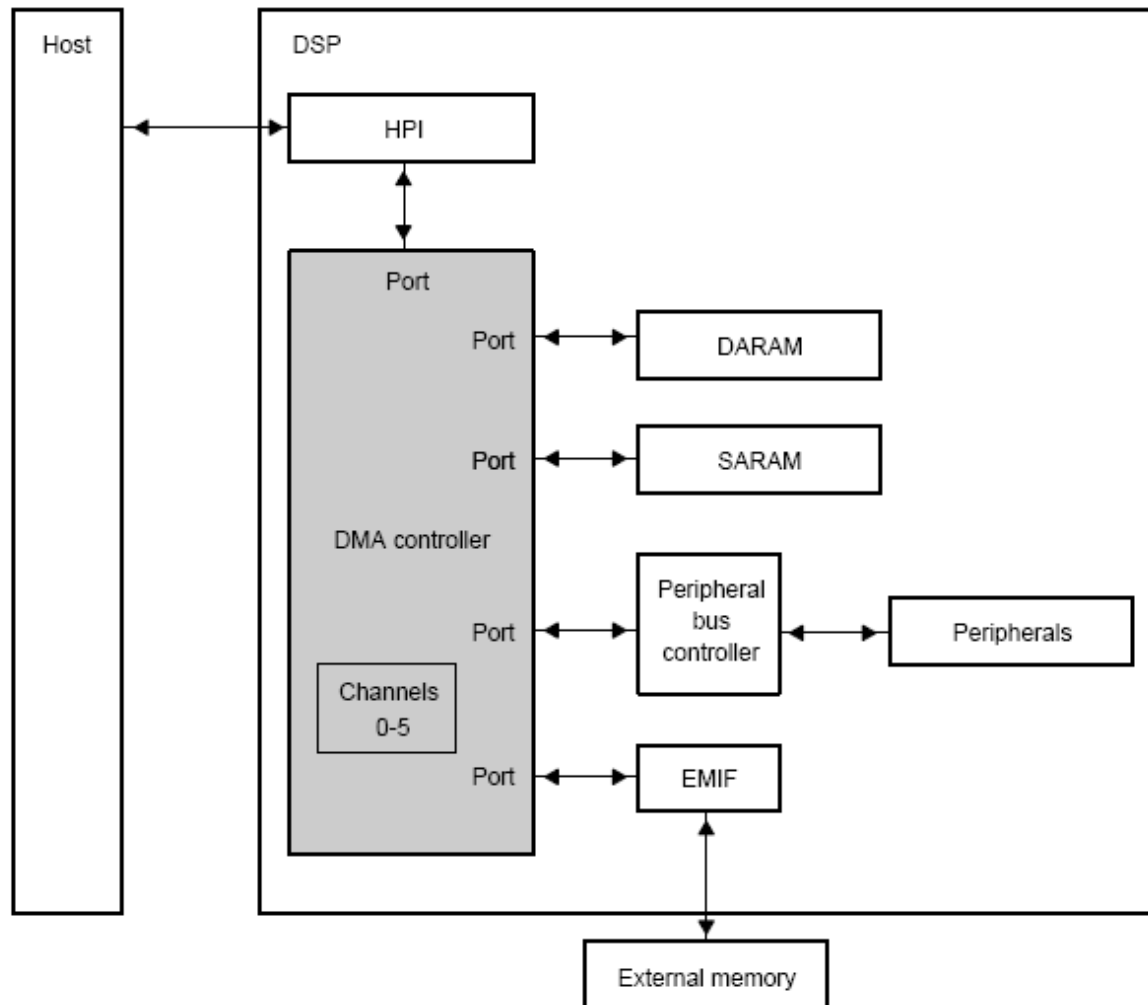
3.5 DMA关键外设的使用

- √2、什么是DMA? Direct Memory Access
- √3、DMA工作的几个准备条件:
 - √1) 源地址
 - √2) 目的地址
 - √3) 传输方式, element size, 帧大小, 读/写等
 - √4) 同步触发方式等

远见品质

3.5 DMA关键外设的使用

v4、DMA的连接方式



SIGHT

3.5 DMA关键外设的使用

✓ 5、通过BIOS配置DMA

✓ Structure DMA_Config

✓ Members

- ✓ Uint32 prictl DMA primary control register value
- ✓ Uint32 secctl DMA secondary control register value
- ✓ Uint32 src DMA source address register value
- ✓ Uint32 dst DMA destination address register value
- ✓ Uint32 xfrcnt DMA transfer count register value
- ✓ Description

✓ This DMA configuration structure is used to set up a DMA channel. You create and initialize this structure and then pass its address to the DMA_config() function. You can use literal values or the _RMK macros to create the structure member values.

✓ Example

```

✓ DMA_Config MyConfig = {
✓   0x00000050, /* prictl */
✓   0x00000080, /* secctl */
✓   0x80000000, /* src   */
✓   0x80010000, /* dst   */
✓   0x00200040 /* xfrcnt */
✓ };

```

✓ DMA_config(hDma,&MyConfig);

远见品质

3.5 DMA关键外设的使用

The screenshot displays the Code Composer Studio interface for configuring the DMA controller. The main window shows the 'dmaCfg0 properties' table, and a 'dmaCfg0 属性' dialog box is open in the foreground.

Property	Value
comment	<add comments here>
Start/Autoinit, Pause (START)	Stop
Source Address Modification (SRC DIR)	None
Destination Address Modification (DST DIR)	None
Element Size (ESIZE)	32-bit
Split Channel Mode (SPLIT)	Disable
Transfer Count Reload (CNT RELOAD)	Count Reload Reg A
Select Programmable Index (INDEX)	Global Index Register A
Read Transfer Sync (RSYNC)	None
Write Transfer Sync (WSYNC)	None
Priority Mode (PRI)	CPU
Transfer Controller Interrupt (TCINT)	Disable
Frame Sync (FS)	Disable
Emulation Mode (EMOD)	Continue
Src. Addr. Reload (SRC RELOAD)	No Reload
Dst. Addr. Reload (DST RELOAD)	No Reload
Split Transmit Overrun Receive Condition (SX COND)	Clear
Frame Complete Condition (FRAME COND)	Clear
Last Frame Condition (LAST COND)	Clear
Block Transfer Finished Condition (BLOCK COND)	Clear
Dropped Read Synchronization Condition (RDROP COND)	Clear
Dropped Write Synchronization Condition (RDROP C...)	Clear
Split Transmit Overrun Receive IE (SX IE)	Disable
Frame Complete IE (FRAME IE)	Disable
Last Frame IE (LAST IE)	Disable
Block Transfer Finished IE (BLOCK IE)	Enable
Dropped Read Synchronization IE (RDROP IE)	Disable
Dropped Write Synchronization IE (WDROP IE)	Disable
Read Sync Status (RSYNC STAT)	Not Received
Write Sync Status (WSYNC STAT)	Not Received
Read Sync Status Clear (RSYNC CLR)	None
Write Sync Status Clear (WSYNC CLR)	None
DMA Action Complete (DMAC EH)	Low
Frame Sync Ignore (FSIG, C6202 only)	None
Read Sync Event Polarity (RSPOL, C6202 only)	Active Low
Write Sync Event Polarity (WSPOL, C6202 only)	Active Low
Source Address Format	Numeric
Source Address - Numeric	0x00000000
Src Addr - Extern Decl. Symbol name	NULL
Src Addr - Extern full address	NULL

The 'dmaCfg0 属性' dialog box shows the following configuration:

- Primary Control Register: 0x00000000
- Secondary Control Register: 0x00000080
- Source Address Format: Numeric
- Source Address - Numeric: 0x00000000
- Destination Address Format: Numeric
- Destination Address - Numeric: 0x00000000
- Transfer Counter Format: Numeric
- Transfer Counter Value: 0x00000000

3.6 线性汇编以及代码优化

- ✓ 1、什么是线性汇编？
- ✓ 线性汇编类似于汇编代码，不同的是线性汇编代码中不需要给出汇编代码必须指出的所有信息，线性汇编代码对这些信息可以进行一些选择，或者由汇编优化器确定。下面是不需要给出的信息：
 - ✓ 1) 使用的寄存器
 - ✓ 2) 指令的并行与否
 - ✓ 3) 指令的延时周期
 - ✓ 4) 指令使用的功能单元

3.6 线性汇编以及代码优化

√2、点积的定点代码优化

√1)C语言代码

```
int dotp(short a[], short b[])
{
    int sum, i;
    sum = 0;
    for(i=0; i<100; i++)
        sum += a[i] * b[i];
    return(sum);
}
```

3.6 线性汇编以及代码优化

√ 2) ASM语言代码

```

LDH      .D1      *A4++,A2          ; load ai from memory
LDH      .D1      *A3++,A5          ; load bi from memory
MPY      .M1      A2,A5,A6          ; ai * bi
ADD      .L1      A6,A7,A7          ; sum += (ai * bi)
SUB      .S1      A1,1,A1           ; decrement loop counter
[A1] B    .S2      LOOP             ; branch to loop
    
```

√ 使用的逻辑单元

- Load (LDH and LDW) instructions must use a .D unit.
- Multiply (MPY and MPYSP) instructions must use a .M unit.
- Add (ADD and ADDSP) instructions use a .L unit.
- Subtract (SUB) instructions use a .S unit.
- Branch (B) instructions must use a .S unit.

3.6 线性汇编以及代码优化

√3) 非并行的ASM代码

```

        MVK    .S1    100, A1        ; set up loop counter
        ZERO   .L1    A7            ; zero out accumulator
LOOP:
        LDH    .D1    *A4++,A2      ; load ai from memory
        LDH    .D1    *A3++,A5      ; load bi from memory
        NOP    4                ; delay slots for LDH
        MPY    .M1    A2,A5,A6      ; ai * bi
        NOP    1                ; delay slot for MPY
        ADD    .L1    A6,A7,A7      ; sum += (ai * bi)
        SUB    .S1    A1,1,A1       ; decrement loop counter
[A1] B     .S2    LOOP              ; branch to loop
        NOP    5                ; delay slots for branch
; Branch occurs here

```

3.6 线性汇编以及代码优化

√4) 并行的ASM代码

```

        MVK    .S1    100, A1        ; set up loop counter
    ||
        ZERO  .L1    A7            ; zero out accumulator
LOOP:
        LDH   .D1    *A4++,A2      ; load ai from memory
    ||
        LDH   .D2    *B4++,B2      ; load bi from memory
        SUB   .S1    A1,1,A1       ; decrement loop counter
    [A1] B     .S2    LOOP          ; branch to loop
        NOP   2                ; delay slots for LDH
        MPY   .M1X   A2,B2,A6      ; ai * bi
        NOP   2                ; delay slots for MPY
        ADD   .L1    A6,A7,A7      ; sum += (ai * bi)
; Branch occurs here
    
```

远见品质

3.6 线性汇编以及代码优化

v5) 性能比较

Code Example	100 Iterations	Cycle Count
Example 5-9 Fixed-point dot product nonparallel assembly	$2 + 100 \times 16$	1602
Example 5-10 Fixed-point dot product parallel assembly	$1 + 100 \times 8$	801

FAR SIGHT

3.6 线性汇编以及代码优化

v6) 线性汇编代码

```

LDW      *a++,ai_i1      ; load ai & a1 from memory
LDW      *b++,bi_i1      ; load bi & b1 from memory
MPY      ai_i1,bi_i1,pi   ; ai * bi
MPYH     ai_i1,bi_i1,pi1  ; ai+1 * bi+1
ADD      pi,sum0,sum0    ; sum0 += (ai * bi)
ADD      pi1,sum1,sum1   ; sum1 += (ai+1 * bi+1)
[ctr] SUB      ctr,1,ctr  ; decrement loop counter
[ctr] B      LOOP        ; branch to loop
    
```

v7) 完整的线性汇编代码

3.6 线性汇编以及代码优化

```

        .global _dotp
_dotp: .cproc  a, b

        .reg    sum, sum0, sum1, a, b
        .reg    ail:ai, bil:bi, pi, pil

        MVK     50,cntr          ; cntr = 100/2
        ZERO    sum0            ; multiply result = 0
        ZERO    sum1            ; multiply result = 0

LOOP:   .trip 50
        LDDW    *a++,ail:ai     ; load ai & ai+1 from memory
        LDDW    *b++,bil:bi     ; load bi & bi+1 from memory
        MPYSP   ai,bi,pi        ; ai * bi
        MPYSP   ail,bil,pil     ; ai+1 * bi+1
        ADDSP   pi,sum0,sum0    ; sum0 += (ai * bi)
        ADDSP   pil,sum1,sum1   ; sum1 += (ai+1 * bi+1)
[cntr]  SUB     cntr,1,cntr     ; decrement loop counter
[cntr]  B       LOOP           ; branch to loop

        ADDSP   sum,sum1,sum0   ; compute final result

        .return sum

        .endproc
    
```



远见品质

C6000 DSP 系统开发培训班课程内容

- √ 一. DSP技术概述
- √ 二. C6000 体系结构和汇编语言
- √ 三. TI DSP开发工具-CCS
- √ 四. C6000 C 运行时环境
- √ 五. 嵌入式实时系统软件开发与调试
- √ 六. C6000程序的优化
- √ 七. TI的实时操作系统: DSP/BIOS

FAR  **IGHT**



The success's road

www.farsight.com.cn

谢谢！