第3章 MicroBlaze软核处理器结构

本章详细介绍Xilinx公司的MicroBlaze处理器的结构及其原理，内容包括MicroBlaze的结构框架、MicroBlaze寄存器、MicroBlaze虚拟存储器管理、MicroBlaze事件及处理、MicroBlaze指令和数据缓存以及MicroBlaze调试和跟踪。

MicroBlaze处理器结构体现了最新的计算机体系结构和微处理器技术的发展趋势。学习这部分内容对掌握软核处理器的一些关键技术和后续基于软核处理器设计片上系统非常重要。

3.1 MicroBlaze 处理器结构框架

MicroBlaze 处理器软核是用户可配置的 RISC 精简指令集计算机，该 RISC 核针对 Xilinx 的 FPGA 芯片进行了优化。图 3.1 给出了 MicroBlaze 处理器核的内部结构图。

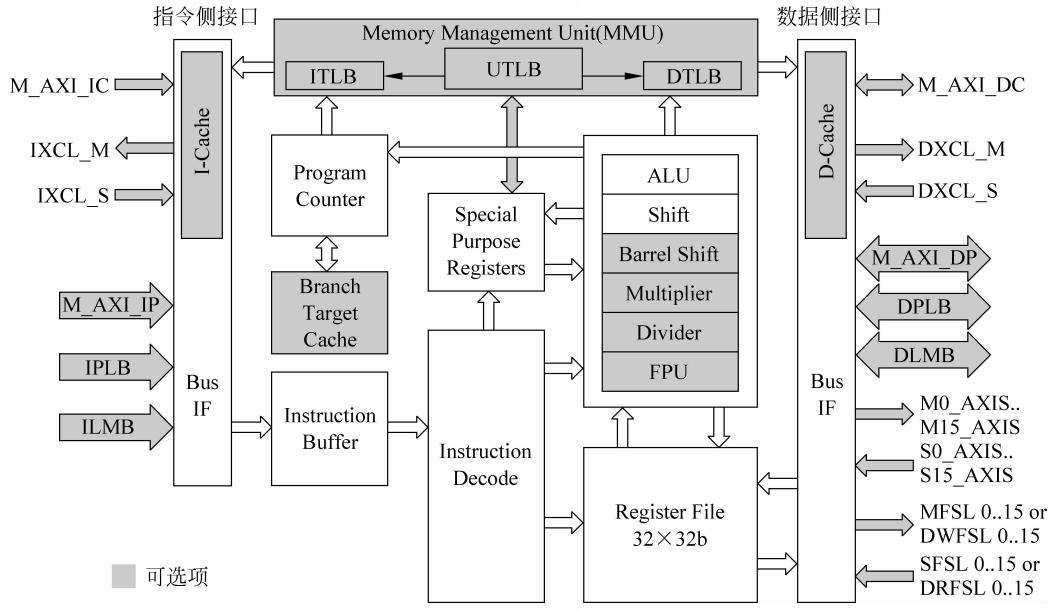

图 3.1 MicroBlaze 核结构图

从图 3.1 中可以看出该处理器有以下几个特点：

(1) 采用指令和数据空间分离的哈佛结构；

(2) 32 个 32 位通用寄存器；

(3) 32 位的地址总线，可寻址空间 4GB；

(4) 32 位 3 个操作数的指令字，指令字有 2 种寻址模式；

(5) 单发(single-issue，一个时钟周期发出一条指令)流水线结构；

(6) 采用指令预测分支策略和预取缓冲区；

(7) 有独立的存储器管理单元对存储器空间进行管理；

(8) 提供了可以使用硬件实现的功能单元——桶形移位寄存器，乘法器，除法器，浮点处理单元；

(9) 提供了丰富的外设接口资源；

(10) 采用点对点的流连接结构和共享总线的结构。

图 3.1 中 MicroBlaze 处理器具体外部接口定义如下：

(1) M_AXI_DP，外设数据总线，AXI4-Lite 或者 AXI4 接口；

(2) DPLB，数据接口，处理器本地总线；

(3) DLMB,数据接口,本地存储器总线(只有 BRAM);
(4) M_AXI_IP,外设指令接口,AXI4-Lite 接口;
(5) IPLB,指令接口,处理器本地总线;
(6) ILMB,指令接口,本地存储器总线(只有 BRAM);
(7) M0_AXI..M15_AXIS,AXI4_Stream 接口,主直接连接接口;
(8) S0_AXIS..S15_AXIS,AXI4_Stream 接口,从直接连接接口;
(9) MFSL0..15,FSL 主接口;
(10) DWFSL0..15,FSL 主直接连接接口;
(11) SFSL0..15,FSL 从接口;
(12) DRFSL0..15,FSL 从直接连接接口;
(13) DXCL,数据侧 Xilinx CacheLink 连接接口(FSL 主/从对);
(14) M_AXI_DC,数据侧高速缓存 AXI4 接口;
(15) IXCL,指令侧 Xilinx CacheLink 连接接口(FSL 主/从对);
(16) M_AXI_IC,指令侧高速缓存 AXI4 接口;
(17) Core,杂项信号时钟、复位、调试和跟踪。

3.1.1 MicroBlaze 存储器结构

MicroBlaze 处理器采用哈佛存储器结构,即指令和数据访问使用独立的地址空间。每一个地址空间都是 32 位(即它们可以独立访问 4GB 地址空间的指令和数据存储器)。通过控制,使重叠的数据和指令空间映射到相同的物理存储器上,这对于软件调试非常有用。

MicroBlaze 所有的指令和数据接口,默认情况下是 32 位,使用大段或小段,位反转格式(取决于 C_ENDIANNESS)。MicroBlaze 支持对数据存储器的字、半字和字节访问。表 3.1 给出大段、小段及位反转格式在存储器空间的表示。

表 3.1 大段、小段及位反转的表示

字数据类型	表示方法			
大段字节地址(Big-Endian Byte Address)	n	n+1	n+2	n+3
大段字节意义(Big-Endian Byte Significance)	MSByte	—	—	LSByte
大段字节顺序(Big-Endian Byte Order)	n	n+1	n+2	n+3
大段字节反转顺序(Big-Endian Byte-Reversed Order)	n+3	n+2	n+1	n
小段字节地址(Little-Endian Byte Address)	n+3	n+2	n+1	n
小段字节意义(Little-Endian Byte Significance)	MSByte	—	—	LSByte
小段字节顺序(Little-Endian Byte Order)	n+3	n+2	n+1	n
小段字节反转顺序(Little-Endian Byte-Reversed Order)	n	n+1	n+2	n+3
位标号(Bit Label)	0			31
位意义(Bit Significance)	MSBit			LSBit

数据访问必须对齐(字访问对齐字边界,半字访问对齐半字边界,除非处理器配置支持非对齐访问异常)。所有指令的访问必须是字对齐方式。

MicroBlaze 通过预取缓冲器和指令高速缓存流来预取指令,以提高处理性能。

MicroBlaze 处理器采用存储器映射方式访问 I/O 设备,即存储器和 I/O 设备采用统一编址方式。处理器有下面三种接口用于存储器访问:

(1) 本地存储器总线(LMB);

(2) 高级可扩展接口 AXI4 或处理器本地总线(PLB);

(3) 高级可扩展接口 AXI4 或 Xilinx CacheLink(XCL)。

当时用 AXI4 时,C_ENDIANNESS 自动设置为小段;使用 PLB 时,自动设置为大段。

处理器指令和数据缓存可以配置成 4/8 字的缓存行(缓存最小结构单位)。缓存行越大,可存放的代码就越长,执行效率也就越高。但是当程序中使用很多的随机访问模式时,会降低缓存的命中率,因此给定大小的缓存行反而会降低性能。

3.1.2 MicroBlaze 浮点单元

MicroBlaze 处理器的浮点单元 FPU 基于 IEEE754 标准:

(1) 使用 IEEE754 单精度浮点格式,包括无穷大定义,不是一个数(NaN)和零;

(2) 支持加、减、乘、除、比较、转换和平方根指令;

(3) 实现最近舍入(round-to-nearest)模式;

(4) 产生状态位用于下溢、上溢、除数为零和无效操作。

为了提高性能,使用下面的非标准简化:

(1) 不支持非规范化的操作数(这种操作数接近 0,无法用完整精度表示),使用硬件浮点对非规范化的操作数进行操作,将返回 NaN。并且在 FSR 中设置非规范化操作数错误标志;

(2) 非规范化的操作数结果保存为 0,并且在 FSR 的寄存器中设置下溢标志,这种方法通常称为清零(Flush-to-zero)模式;

(3) 对 NaN 操作返回固定的 NaN-0xFFC00000;

(4) 浮点操作的上溢总是返回有符号的∞。

IEEE754 单精度浮点数由 1 位符号位、8 位偏置指数和 23 位小数(尾数)部分组成。表 3.2 给出了 IEEE754 单精度格式。

表 3.2 IEEE754 单精度格式

0	1 8	9 31
符号	指数	小数

在 MicroBlaze 中浮点数 V 可以用以下方法描述:

(1) 如果指数=255,小数< >0,值=NaN,和符号无关;

(2) 如果指数=255,小数=0,值=$(-1)^{sign} * \infty$;

(3) 如果 0<指数<255, 值=$(-1)^{sign} * 2^{(指数-127)} * (1.尾数)$;

(4) 如果指数=0,小数<>0,值=$(-1)^{sign} * 2^{-126} * (0.尾数)$;

(5) 如果指数=0,小数=0,值=$(-1)^{sign} * 0$。

在实际中,只有规则 3 和 5 是有用的。其浮点单元支持浮点算术操作、浮点比较操作、浮点和有符号整数之间的转换操作。

基于 GCC 的 SDK 编译器系统,为 FPU 提供支持。当使用 SDK 时,基于系统中 FPU 的类型,编译器标志自动添加到 GCC 命令行中。

SDK 编译器系统只包含软件浮点 C 实时库。为了使用硬件 FPU 的优势,使用合适的编译器开关重新编译库。

为了从 FPU 中得到最大的益处,而不需要底层汇编语言编程,就需要考虑 C 编译器如何理解所编写的源代码。下面的规则可以帮助提高浮点处理的效率。

1. 默认情况下,C 中的浮点常数是双精度

当使用单精度的 FPU 时,不经心的编码将导致使用双精度的软件仿真程序而不使用真正的单精度指令,为了避免这种情况,需要明确将常数指明为单精度数。

```
float x = 1.0;
          …
          x += (float)1.0; or x += 1.0F
```

2. 硬件 FPU 支持整数和浮点格式之间的转换

下面一个不好的例子就是计算 1～10 的整数平方的和,使用浮点表示。在每个循环中,要求整数到浮点的转换。

```
float sum,t;
int i;
sum = 0.0f;
for(i = 0;i <= 10;i++){
  t = (float)i;
  sum += t * t;
}
```

⟹

```
float sum,t;
int i;
t = sum = 0.0f;
for(i = 0;i <= 10;i++){
  t += 1.0f;
  sum += t * t;
}
```

3. 均方根实时库函数

标准 C 实时数学库函数使用双精度算法。当使用单精度的 FPU 时,调用均方根函数 sqrt( )导致低效率的仿真例程,而不使用 FPU 指令。

解决方法是使用非-ANSI 的函数 sqrtf( ),使用 FPU 单精度的算法。

```
#include <math.h>
...
float x = -1.0f;
...
x = sqrt(x); /* 双精度
```

⇒

```
#include <math.h>
...
float x = 1.0f;
...
x = sqrtf(x); /* 单精度
```

3.1.3 MicroBlaze 流连接接口

MicroBlaze 处理器可配置为 16 个快速单一链接(FSL)接口或者 AXI4-Stream 接口,每个接口由一个输入和输出端口组成。通道是专用单向的点对点数据流接口。

MicroBlaze 接口为 32 比特宽度。一个独立的比特用来描述发送/接收字是控制类型还是数据类型。MicroBlaze ISA 中的 get 指令用来从端口到通用寄存器传输信息;put 指令以相反的方向传输信息。指令有 4 种类型:阻塞数据、非阻塞数据、阻塞控制、非阻塞控制。

每个链路提供到处理器流水线低延迟的专用接口。这是使用定制加速器和处理器扩展的理想接口。图 3.2 给出了使用定制硬件加速的例子。

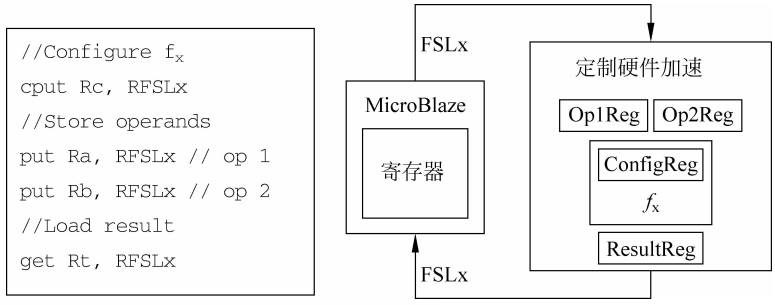

图 3.2 定制硬件加速原理

3.1.4 MicroBlaze 流水线结构

1. 流水线

MicroBlaze 处理器的指令采用流水方式执行。大多数指令需要一个时钟周期完成。所以,用于一个特定指令完成所需要的时钟周期和流水线级数是一致的。只有少数指令在执行阶段需要多个时钟周期完成,通过流水线断流方式实现。

当从较慢的存储器执行时,需要多个时钟周期完成取指令过程。这个过程直接影响流水线的效率。MicroBlaze 处理器通过指令预取缓冲区来减少这个过程对流水线性能的影响。当流水线在执行过程中断流时,预取缓冲区继续加载顺序指令。当执行指令继续进行时,可以提取新的指令到预取缓冲区中,而不需要等待对指令存储器的访问完成。

当执行时，指令被修改，在执行修改的指令前，将预取缓冲区清空。

当采用面积优化时，流水线分为三级，即取指、译码和执行，这样可以减少硬件开销。图 3.3 给出了三级流水结构。

图 3.3　三级流水结构

当不使用面积优化时，流水线分为五级，即取指、译码、执行、访问存储器和回写，这样可以提高性能。图 3.4 给出了五级流水结构。

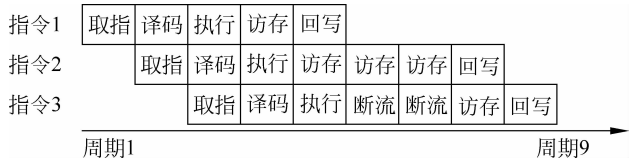

图 3.4　三级流水结构

2. 分支

当执行采纳分支(taken branch，分支就是程序中跳转指令)时，一般情况下，流水线的取指、译码和预取缓冲区将被清空。取指级从所计算的分支地址中重新取指。MicroBlaze 中的采纳分支需要三个时钟周期才能完成，而其中的两个时钟周期用来填充流水线。为了减少所产生的延迟，处理器采用延迟隙(delay slot)的分支处理方法。

3. 延迟隙

当采用延迟隙的采纳分支时，只有取指阶段被清除。而允许完成译码过程。这种技术，将延迟从两个周期减少到一个周期。执行带有延迟隙的分支时，只需要在指令助记符后增加字母 D 即可，例如 BNE 指令不执行后面的指令，而 BNED 在找到分支的位置前，就已经执行下一条指令。

延迟隙不能使用的指令：IMM、分支、断点。那些引起可恢复异常的指令是允许使用延迟隙的。如果在延迟隙中产生异常，则异常句柄(异常服务程序)负责转向执行分支目标(这个分支目标保存在 BTR 寄存器内)。如果设置了 ESR[DS]位，R17 寄存器就不是有效的，除非该寄存器保存着引起异常指令的下一条指令的地址。

4. 分支目标高速缓存

为了提高分支的性能，MicroBlaze 提供分支目标高速缓存(branch target cache，BTC)和分支预测策略。

BTC 中保存着每个最开始指令所遇到的立即分支和返回指令的目标地址。下一次再遇到时，通常能在 BTC 中找到，指令加载 PC 简单地变为所保存的应采纳的目标地址。无条件分支和返回指令通常被采纳，然而有条件分支使用分支预测，即不采纳不执行的指令。

3.1.5 MicroBlaze 特权指令

MicroBlaze 处理器的特权指令有以下几类：

(1) GET、PUT、NGET、NPUT、CGET、CPUTNCGET、NCPUT，这些指令和 FSL 接口操作有关；

(2) WIC、WDC，这些指令和写数据、指令缓存有关；

(3) MTS，写特殊寄存器指令；

(4) MSRCLR、MSRSET，分别是读，然后清除/置位特殊寄存器；

(5) BRK，断点指令；

(6) RTID、RTBD、RTED，分别是从中断返回、从断点返回和从异常返回指令；

(7) BRKI，立即打断指令。

这些指令都是对处理器的内核操作，试图在用户模式下使用这些特权指令，将引起异常，可以使用以下几种方法脱离用户模式和虚拟模式：

(1) 硬件复位(包括调试复位)；

(2) 硬件异常；

(3) 不可屏蔽断点或硬件断点；

(4) 中断；

(5) 执行指令 Bralid Re，0x8，执行用户向量异常操作；

(6) 执行软件断点指令 BRKI，跳转到地址 0x8 或 0x18。

在以上这些情况(除硬件产生复位外)，虚拟模式和用户模式的状态保存在 MSR 寄存器内的 UMS 和 VMS 比特位。

通过使用 BRALID 和 BRKI 指令，用户模式下的应用程序将控制权交给系统服务例程(特权模式程序)，并且跳转到物理地址 0x8。执行这条指令产生系统异常条件。异常句柄(异常服务程序)决定调用哪个系统服务例程和是否允许调用服务。如果授权许可，异常句柄代表应用程序执行真正的程序调用。

异常服务例程所希望执行环境要求执行序言指令(prologue instruction)来建立所要求的环境。这些指令用于创建保持程序信息(活动的纪录)的存储块、更新和初始化指针和保存易失性存储器(这些寄存器是系统服务例程需要使用的)。当创建可执行模块时，链接器能插入序言指令，或者作为系统调用中断句柄或者系统库例程的存根代码(stub code)。

从系统服务例程返回和上面描述的过程相反。执行结束代码(epilog code)打开和取消分配的活动记录，恢复指针和恢复存易失性存储器。中断句柄通过执行异常指令 RTED 返回到应用程序。

3.1.6 MicroBlaze 指令类型

MicroBlaze 处理器指令为 32 比特长度，指令分为类型 A 和类型 B。表 3.3a 给出了类型 A 的指令结构，表 3.3b 给出了类型 B 的指令结构。

表 3.3a 类型 A 的指令结构

操作数	目标寄存器	源寄存器 A	源寄存器 B	00000000000

表 3.3b 类型 B 的指令结构

操作数	目标寄存器	源寄存器 A	立即数(16 位)

类型 A 指令包含两个源寄存器操作数和一个目的寄存器操作数。类型 B 指令一个源寄存器、一个 16 位的立即数(通过 imm 指令扩展到 32 位)和一个目的寄存器操作数。MicroBlaze 处理器的指令完成的功能有：算术操作、逻辑操作、分支操作、加载/存储操作和其他特殊操作。

3.2 MicroBlaze 寄存器

MicroBlaze 处理器内有 32 个 32 位的通用寄存器和最多 18 个特殊寄存器(取决于配置)。

3.2.1 通用寄存器

32 个 32 位的通用寄存器为 R0～R31。寄存器的值在下载比特流文件时复位为 0x00000000，表 3.4 给出了通用寄存器的功能。

表 3.4 通用寄存器(R0～R31)

寄存器名称	功　能
R0	值为 0，写该寄存器无效
R1～R13	32 位通用寄存器
R14	32 位寄存器用来保存中断时的返回地址
R15	32 位寄存器，推荐用于保存用户向量表内的返回地址
R16	32 位寄存器，用于保存断点的返回地址
R17	当处理器配置成允许支持硬件异常时，用于加载使硬件异常指令后的下一条指令的地址(使用 BTR 例外)；否则是通用寄存器
R18～R31	32 位通用寄存器

3.2.2 特殊寄存器

1. 程序计数器(program counter，PC)

程序计数器是一个 32 位的被执行指令的地址。该地址可以通过 MFS 指令读取，但不能使用 MTS 指令写。表 3.5 给出了 PC 寄存器。

表 3.5　PC 寄存器

0	31
PC	

2. 机器状态寄存器(machine status register,MSR)

MSR 保存处理器的控制和状态标志,该寄存器可以通过 MFS 指令读取。MSR 寄存器也可以通过 MTS 指令进行写操作,或者使用专用的 MSRSET 和 MSRCLR 指令对寄存器进行操作。表 3.6 给出了 MSR 寄存器的标志,表 3.7 给出了 MSR 寄存器各个标志位的含义。

表 3.6　MSR 寄存器

0	1～16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CC	RES	VMS	VM	UMS	UM	PVR	EIP	EE	DCE	DZO	ICE	FSL	BIP	C	IE	RES
—	—	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

表 3.7　MSR 寄存器的标志

比特位	名　字	功　　能	复位值
0	CC	算术进位复制,对第 29 比特算术进位标志复制,与其一样	0
1:16	保留	—	
17	VMS	虚拟保护模式保存,只有在使用 MMU(C_USE_MMU＞1)时可用	0
18	VM	虚拟保护模式,只有在使用 MMU(C_USE_MMU＞1)时可用: 0＝禁止 MMU 地址翻译和访问保护(C_USE_MMU＝3),禁止访问保护(C_USE_MMU＝2);1＝使能 MMU 地址翻译和访问保护(C_USE_MMU＝3),使能访问保护(C_USE_MMU＝2)	0
19	UMS	用户模式保存,只有在使用 MMU(C_USE_MMU＞0)时可用	0
20	UM	用户模式,只有在使用 MMU(C_USE_MMU＞0)时可用: 0＝特权模式,所有指令均可用;1＝用户模式,某些指令不能用	0
21	PVR	处理器版本寄存器存在: 0＝无处理器版本寄存器;1＝存在处理器版本寄存器	基于 C_PVR
22	EIP	处理异常,只有 C_＊_EXCEPTION 或 C_USE_MMU 有效时可用: 0＝无硬件处理异常;1＝有硬件处理异常	0
23	EE	异常使能,只有 C_＊_EXCEPTION 或 C_USE_MMU 有效时可用: 0＝禁止硬件异常;1＝使能硬件异常	0