第 4 章 系统性能获取

缉蛮黄鸟，止于丘隅 《诗经·商颂·玄鸟》

智能嵌入式系统优化设计方法是通过系统功能的软件与硬件实现,实现整个系统性能的优化。在系统实现之前人们要获得系统软件实现的性能（简称软件性能）以及硬件实现的性能（简称硬件性能）。本章提供软件性能、硬件性能、以及通信性能的获取方法。

第 4.1 节 软件性能属性获取

软件性能一般包括软件执行时间和功耗。这里分两小节介绍软件性能属性的获取方法。

4.1.1 软件时间

定义 4.1 软件执行时间（简称软件时间）是指任务使用软件执行所需要的时间。

一般来说，软件时间 T 通常可以表示成执行该软件的时钟周期数 N 与时钟频率 H 倒数之积：

$$T = N \times 1/H \quad \text{(公式 4-1)}$$

其中时钟频率 H 倒数为计算机中最基本的、最小的时间单位。

在软件运行中获得的时间，也就是把任务编程实现后在微处理器上执行获得的时间。算法建模阶段可以使用 Matlab 仿真工具、C 和 C++，将其代码执行时间作为软件时间，从而获取到可参考的软件任务执行时间，为系统开发节约了时间。

在 Matlab 工具中获取软件时间可以通过在需要测量的软件代码前加命令 tic，软件任务结尾加命令 toc，即可获得软件时间。

例 4.1 测试排序算法的执行时间。将数据 3,4,1,8,0,5,14,10 按照从小到大进行排序，并记录排序结果和执行的时间。

解：下面是该算法（微处理器为 Intel 酷睿 2 Duo L9400）在 Matlab 上的运行代码以及运行结果：

```
>> tic
a=[3,4,1,8,0,5,14,10];
temp=0;
for i=1:7
    for k=0:7-i
```

```
        if a(k+1)>a(k+2)
            temp=a(k+1);
            a(k+1)=a(k+2);
            a(k+2)=temp;
        end
    end
end
a
 toc
```

a = 0 1 3 4 5 8 10 14

Elapsed time is 0.007076 seconds.

软件执行时间为 0.007076 秒。但若再次执行，执行时间会发生变化。

```
>> tic
a=[3,4,1,8,0,5,14,10];
temp=0;
for i=1:7
    for k=0:7-i
        if a(k+1)>a(k+2)
            temp=a(k+1);
            a(k+1)=a(k+2);
            a(k+2)=temp;
        end
    end
end
a
 toc
```

a =0 1 3 4 5 8 10 14

Elapsed time is 0.018493 seconds.

这次软件执行时间为 0.018493 秒，是上次执行时间的 25 倍。由此可以看出，同一个程序在多次运行时所产生的执行时间是不同的，而且相差较大。因此，可以多运行几次记录最大执行时间和最小执行时间以及平均时间，作为软件运行时间的属性值。

例 4.2 测试排序算法的执行时间。将数据 3,4,1,8,0,5,14,10 按照从小到大进行排序，使用 Matlab 记录 10 次运行时间以及最小、最大和平均执行时间。

解： 下面是运行环境（微处理器为 Intel(R)i5-5200U@2.2GHZ）在 Matlab 上的运行代码以及运行结果：

```
t=[0,0,0,0,0,0,0,0,0,0];
for i=1:10
     a=[3,4,1,8,0,5,14,10];
    tic
    %small =a(1);
    for j=1:7
        for k=1:8-j
            if (a(k)>a(k+1))
                a(k)=a(k)+a(k+1);
                a(k+1)=a(k)-a(k+1);
                a(k)=a(k)-a(k+1);
            end
        end
    end
    a;
    t(i)=toc
end
min=t(1);
max=t(1);
avg=t(1);
for k=2:10
    if (t(k)<min)
        min=t(k);
    end
    if (t(k)>max)
        max=t(k);
    end
    avg=avg+t(k);
end
min
max
avg=avg/10
```

```
min =8.4000e-06

max =1.94000e-05

avg = 1.2390e-05

t=(1.7100e-05,1.9400e-05,1.8000e-05,1.0800e-05,9.2000e-06,9.70000e-06,8.4000e-06,1.2200e-05,1.0400e-05,8.7000e-06)
```

注：8.4000e-06 是 Matlab 科学计数法，表示 $8.4\times10^{-6}=0.0000084$.

在 C、C++代码上分别加上一行带有时间戳的代码：//cout<<"timer = "<<fp_ns<<"ms"<<endl;和 //count<<”timer=”<<cfp_ns<<”ms”<<endl;可获得软件执行时间。

例 4.3 实现欧几里得（Euclid）整数最大公约数算法，求得(200, 148)最大公约数以及软件执行时间。

解：Matlab 软件执行时间是 39ms， C++软件执行时间是 57ms，而 C 软件执行时间为 35ms。

```
tic
% gcd codes
% input two numbers: a and b;
% output their greatest common divisor;
a = 200; %input('Please input the first number:\n a =');
b= 148;%input('Please input the second number:\n b =');
c =mod(a,b);
 while c~=0
    a=b;
    b=c;
c=mod(a,b);
end
disp('The Greatest common divisor of 200 and 148 is: ');
disp(b);
toc
```

GCD Matlab 代码

```
#include<stdio.h>
#include<time.h>
int main(){
	double t0= clock();
	int a = 200;
	int b = 148;
	    //input the value of a and b
	int c = a % b;
	while(c){
		a = b;
		b = c;
		c = a % b;
		    }
	printf("The Greatest common divisor of 200 and 148 is : %d\n",b);
	double t1= clock();
	double fp_ns=(t1-t0);
	printf("execute time = %.1f ms\n", fp_ns);
	//cout<<"timer = "<<fp_ns <<"ms"<<endl;
	return 0;
}
```

GCD C 代码

```
#include <iostream>
#include <chrono>
#include<time.h>
using namespace std;
int main(){
	auto t0 = clock();
	int a = 200;
	int b = 148;
	    //input the value of a and b
	int c = a % b;
	while(c){
		a = b;
		b = c;
		c = a % b;
	}
	cout << "The Greatest common divisor of 200 and 148 is : " << b << endl;
	auto t1 = clock();
	double fp_ns=(t1-t0);
	cout<<"timer = "<<fp_ns<<"ms"<<endl;
	return 0;
}
```

GCD C++代码

4.1.2 软件功耗

定义 4.2 软件功耗 软件功耗是指软件从开始运行到运行结束产生的功耗。

软件执行时间越长其功耗越大。因此软件执行时间是计算软件功耗的一个因素。

微处理器在运行过程中会消耗电能量，因此会产生功耗。 这种功耗通常定义为电压与电流的乘积，单位为瓦(W)。软件在微处理器上运行会产生功耗，需要实时测量获得。在软件设计阶段，可以预估软件运行产生的功耗值。软件运行的功耗 E 定义为微处理器单位功耗 P 与软件运行时间 T 的乘积:

$$E = P \times T \qquad \text{(公式 4-2)}$$

微处理器的参数一般给出热设计功耗（TDP）值， 如：Intel 酷睿 2 双核 SL9400 的热设计功耗为 17w，Intel(R)i5-5200U@2.2GHZ 热设计功耗为 15w，酷睿 2 双核 T9500 的热设计功耗为 35w，酷睿 i5 3470 热设计功耗为 77w。热设计功耗的含义是当微处理器达到最大负荷时热量释放的指标，是冷却系统必须有能力驱散热量的最大限度。很明显，热设计

功耗不是程序在微处理器上运行时产生的单位时间功耗。一般来说，程序运行时产生的单位功耗要小于热设计功耗，为了能预估软件运行产生的功耗，可以把热设计功耗的 10%作为程序运行产生的单位功耗。这样就很方便给出软件运行功耗的预估值，为软件开发提供功耗参考。

例 4.4 测试排序算法的执行能耗。将数据 3,4,1,8,0,5,14,10 按照从小到大进行排序，使用的微处理器为 Intel(R)i5-5200U@2.2GHZ，其热设计功耗为 15w，其 10%为 1.5w。例 4.2 显示在 Matlab 上的运行十次的运行时间：最小为 8.4000e-06、最大为 1.9400e-05、平均为 1.2390e-05。这样这个程序最小、最大、平均能耗分别为：1.2600e-05w、2.9100e-05w、1.8585e-05w。

注：软件功耗=微处理器单位功耗 P×程序时钟周期数 N×时钟频率 H 倒数。一般认为电压保持不变，因此只需测量出执行时处理器的平均电流，便可以计算出软件所消耗的功耗。通过直接测量目标处理器执行指令时平均电流的方法可以建立基本的软件功耗模型。通过执行一个足够多次循环的指令序列可以得到稳态平均电流。循环体内指令重复的次数要足够多，以消除循环结束时无条件跳转等指令的影响。

只在微处理器内部进行计算的指令功耗可以用指令数来算，但软件执行中的内存操作功耗也是需要计算的，读写导致的电流变化也是不一样的，也应该计算在内。软件功耗方面应建立这种计算模块：

指令执行功耗+数据读访问功耗+数据写访问功耗+指令无效读取功耗

这些与具体各程序有关，每个程序设计人员应给出程序执行量和数据访问次数，用这个可以比较精确地预估软件运行功耗。

降低软件功耗一直得到了软件工程界的关注，软件的优化可以降低软件功耗。

早在 1999 年，Simunic 等人开始了嵌入式软件功耗的研究，其研究结果[19，20]表明，对软件源代码进行全局优化，软件系统最高可节能 90%；2000 年，Kandemir 等人对目标代码进行了功耗优化设计，研究结果[21]表明，对目标代码进行功耗优化设计可节能 25%以上。智能手机、智能传感器等智能移动终端的功耗更是需要关注的， 2014 年，郭兵团队[22]利用 Androi 操作系统提供的基于组件应用程序能耗测量方法，提出了一种基于应用程序运行时间的时间能耗模型，实践表明该模型能保证在能耗误差 0.001%-7.82%基础上，方便终端用户利用独立于硬件功耗特性(power characteristics) 的时间变量估算应用程序能耗。

第 4.2 硬件性能属性获取

硬件性能包括硬件时间、硬件面积，以及 FPGA 查找表数等。

4.2.1 硬件时间与面积属性

定义 4.3 硬件时间 任务硬件执行完成所需要的时间。

定义 4.4 硬件面积 任务在硬件上执行所需要的硬件面积。

硬件性能属性一般包括硬件执行时间（简称硬件时间）和硬件面积。而系统硬件一般是指 ASIC（即 Application Specific Integrated Circuit）专用集成电路的面积大小。但是由于 ASIC 全定制或者半定制的设计方法，流片需要花费大量的时间与人力进行人工布局布线，而且一旦需要修改内部设计，将不得不影响到其它部分的布局。所以，进行硬件面积估算若采用 ASIC 将会带来很大的成本。硬件时间若通过 ASIC 的设计来获取同样会带来很大的成本。

因此，本节采用 FPGA 的设计工具 Vivado HLS 进行硬件面积和硬件时间的估计。FPGA 硬件面积是指查找表 LUT（Look Up Table）的个数。硬件时间是时钟周期数与时钟周期的乘积。

4.2.2 硬件查找表

定义 4.3 硬件查找表 FPGA 的查找表 LUT 本质上是一个静态随机存储器 SRAM（Static Random Access Memory）。

FPGA 查找表多采用 4 输入的查找表，每个查找表可以看作一个有 4 位地址线的 16×1 的随机存储器 RAM(Random Access Memory)。近来也有 6 位地址输入的查找表。在 FPGA 工作时，每输入一个信号进行逻辑运算就等于输入一个地址进行查表，找出地址对应的内容，然后输出。因此，查找表个数反映了 FPGA 的能力，查找表个数越多，FPGA 的可编程能力就越好，其成本也就越高。

4.2.3 硬件时间与查找表获取

Vivado HLS（High Level Synthesis）高阶层次综合工具可直接使用 C，C++以及 System C 语言规范对 Xilinx 可编程器件进行编程，无需手动创建 RTL，从而可加速 IP 创建，同时会对 IP 资源和占用时间进行模拟，也可以对编程代码进行优化，以减少执行时间。

本节使用工具是 Vivado HLS 2018.2 工具平台。有关安装 Vivado 工具平台和如何使用可以参考本章附录或 Vivado 工具使用说明。

例 4.5 在 Vivado HLS 工具平台上实现欧几里得整数最大公约算法。输入输出要求如下：有两个整数输入 m， n (可用 8bits 表示，但要依据整数的大小来决定位数，可以使用 16bits、32bits 甚至 64bits) 输出返回两整数的最大公约数，统一给出硬件时间和 LUT 数。

解： 首先，按照 Vivado HLS 规范将一般的 C/C++代码改写为符合 HLS 标准的代码。

根据 Vivado HLS 的 C 语言规范需要对下面 C 语言代码进行改写，把“int”类型改写为符合 HLS 标准的“int8”型，“int8”表示 8 位整数型变量，同时为了能正确使用“int8”数据类型，要包含头文件“ap_cint.h“。

```
int gcd(int a, int b)
 {
    int c;
     c=a%b;
    while(c!=0){
          a=b;
          b=c;
          c=a%b;
          }
 return b;
}
```

gcd C 语言代码

```
#include<ap_cint.h>

uint8 gcd(uint8 m,uint8 n);
```

gcd.h 文件

```c
#include "gcd.h"
uint8 gcd(uint8 m,uint8 n)
{
    uint8 r;
    while(n!=0){
        r = m%n;
        m = n;
        n = r;
    }
    return m;
}
```

gcd.c HLS 文件

完成了 C 代码的改写后，需要编写一个测试文件，对修改后符合 HLS 标准的 C 代码进行逻辑检查，通过验证实际输出是否符合预期输出，确保改写没有影响原本代码想要实现的逻辑，形成测试文件 gcd_tb.c。

```c
#include <stdio.h>
#include "gcd.h"
int main(void)
{
    uint8 result1 = 0;
    uint8 result2 = 0;
    result1 = gcd(200,148);
    printf("result=%d \n",result1);
    return 0;
}
```

gcd_tb.c HLS 文件

打开 Vivado HLS 工具，选择 Create New Project，进行 Project 定义。取 GCD 为 Project 名，为 GCD project 设置一个目录，如 C://Vivado-files。按照操作步骤依次加入 Top Function (GCD)和 Source 目录下的 gcd.c 和 gcd.h 文件以及 Test Bench 目录下的 gcd_tb.c。打开 Board 界面选择开发板，在 Boards 环境下，选择目标开发板 xc7z020clg484-1。

运行 Solutioin 环境下运行 C Synthesis Active Solution，获得图 4-1 和图 4-2，中部上侧窗口显示“General Information（基本信息）”、“Performance Estimates（性能估算）”、“Utilization Estimates（资源利用估算）”、“（Interface 接口）”等信息。从这两张图可以获得欧几里得算法最大公约数的硬件资源信息，其中图 4-1 是有关 Timing 信息， 而图 4-2 是有关查找表 LUT 等信息。

运行 Solution 环境下运行 C/RTL Cosimulation 可以获得图 4-3，给出硬件执行时间和 Latency 以及 Interval 资源信息。

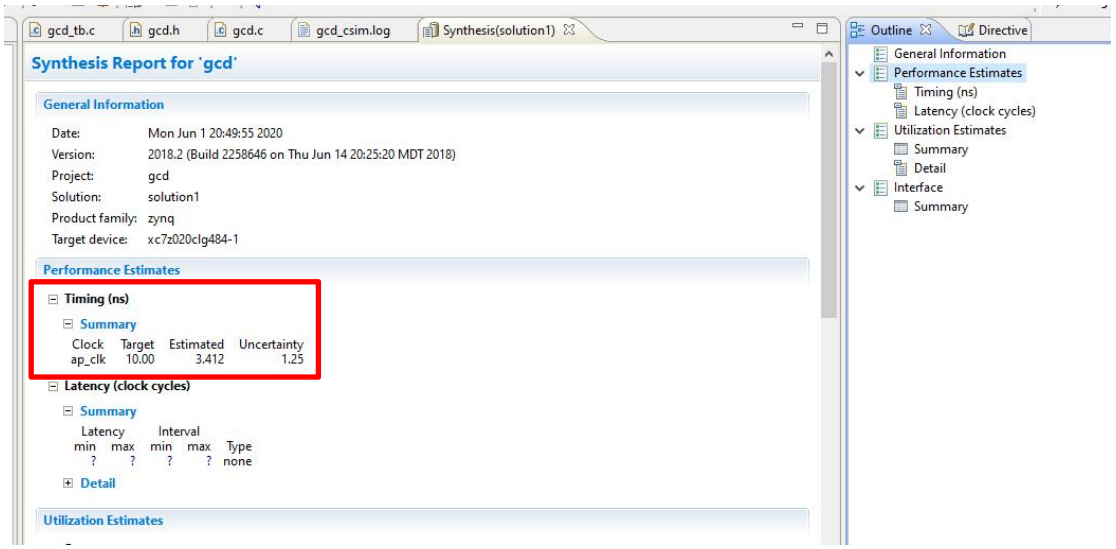

图 4-1 Timing 资源信息图

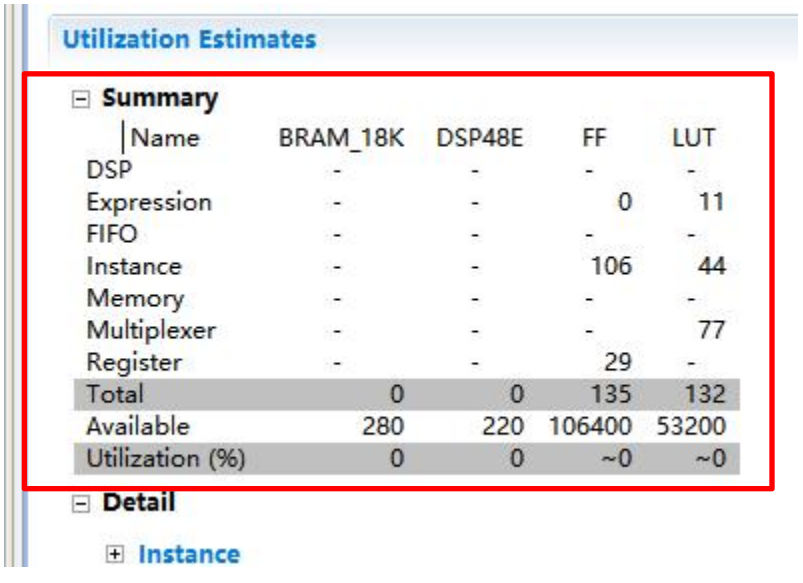

图 4-2 LUT 资源信息图

FPGA 查找表 LUT 数：可以直接从“Utilization Estimates（资源利用估算）”(图 4-2）中获得 LUT 数，得出硬件面积为：132 个 LUT。资源利用估算表中还有触发器 FF（Flip Flop）数 135。查找表 LUT 归于组合逻辑，而触发器 FF 归于时序逻辑。数字信号处理器 DSP（Digital Signal Processor）用于数字信号处理，具有灵活、精确、抗干扰强、设备尺寸小、造价低、速度快等突出优点。

硬件时间：任务硬件时间是任务硬件实现执行的时间，与测试用例有关。硬件时间可以从硬件资源信息获得，如图 4-3 可以看到测试用例为（200，48）的硬件执行时间为 795ns，latency 为 61。

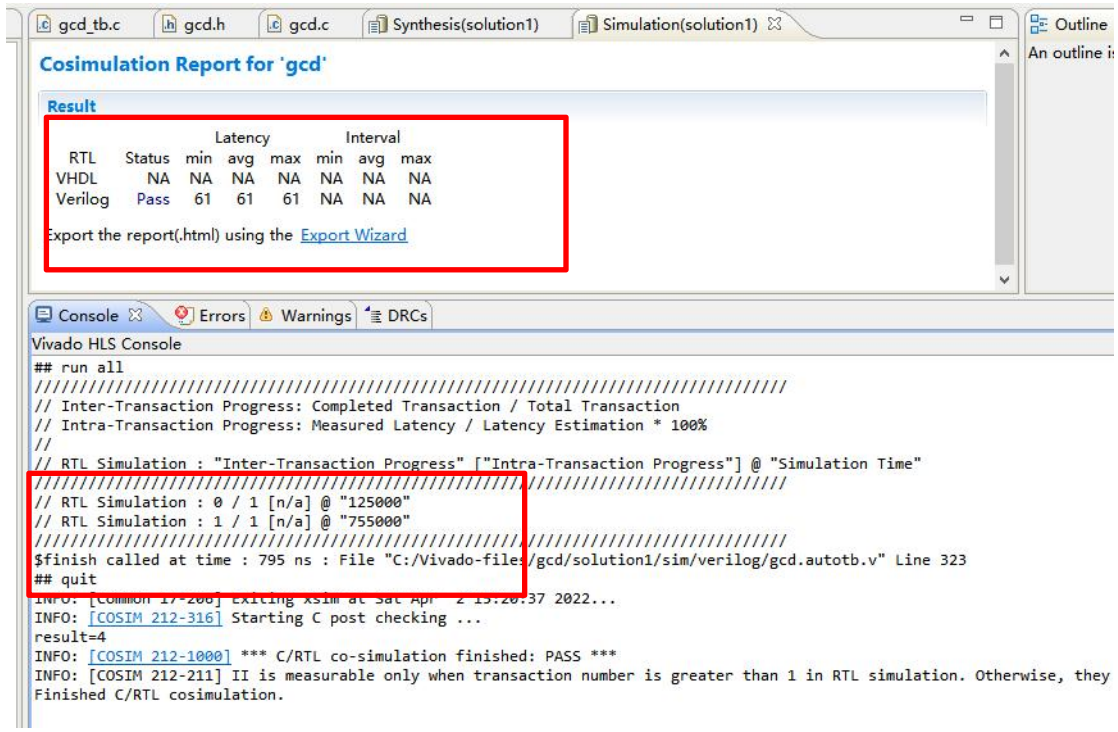

图 4-3 硬件执行时间、latency 和 Interval 资源信息图

调整 gcd_tb.c 中的参数 200 为 2000，148 调整为 48，计算 2000 和 48 最大公因子所需要的时间为 435ns，Latench 为 25。再次调整 gcd_tb.c 中的参数为 2000 和 1481，计算 2000 和 1481 最大公因子的时间为 795ns，Latency 为 61。

设置格式[陈年老酒]: 字体: （默认）Calibri, 图案: 清除(自动设置), 字体颜色: 自动设置

设置格式[陈年老酒]: 字体: （默认）Calibri, （中文）宋体, 小四, 图案: 清除(自动设置), 字体颜色: 自动设置

设置格式[陈年老酒]: 字体:（默认）Calibri, 小四, 图案: 清除(自动设置), 字体颜色: 自动设置

设置格式[陈年老酒]: 字体: （默认）Calibri, 图案: 清除(自动设置), 字体颜色: 自动设置

设置格式[陈年老酒]: 字体: （默认）Calibri, （中文）宋体, 小四, 图案: 清除(自动设置), 字体颜色: 自动设置

设置格式[陈年老酒]: 默认段落字体, 字体:（默认）Calibri, （中文）宋体, 小四, 非倾斜, 无下划线, 图案: 清除(自动设置), 字体颜色: 自动设置, 非加宽量/紧缩量, 非全部大写

设置格式[陈年老酒]: 字体: （默认）Calibri, （中文）宋体, 小四, 图案: 清除(自动设置), 字体颜色: 自动设置

删除[陈年老酒]: 这些性能指标与测试用例无关

设置格式[陈年老酒]: 字体: （默认）Calibri, 图案: 清除(自动设置), 字体颜色: 自动设置

第 4.3 节 通信时延属性获取

本节介绍通信时延属性的获取方法。4.3.1 介绍通过依据传输数据量来获取时延的简单方法。4.3.2 以 ZYNQ702 开发板为例，通过一个实例来分析直接内存访问 DMA 和非直接内存访问 DMA 两种方式的通信时延估计方法。

4.3.1 通信时延的简单估测

假定一个总线访问周期为 TB，通信时延 Tcomm 可以简单地从传输的数据量 N（字节）和总线宽度 W（位数）估计出来。首先从总线宽度 W 计算出一次传输字节数 NB（=总线宽度 W 与数字 8 的商，即 NB=w/8，因为一个字节是 8 位表示），然后计算共需传输的次数（=通信总量 N 与一次传输字节数 NB 的商），每次总线时长即访问周期 TB，通信时延 Tcomm 计算如下公式 4-3。

$$Tcomm=(N/NB)\times TB \qquad \text{(公式 4-3)}$$

例如，通信总量 N 为 100 字节，总线宽度 W 为 32 位，一次可以传输 4 个字节，即 NB=4，共需要传输 100/4=25 次，每次总线时长即访问周期 TB 为 20ns，得通信时延约为 25×20ns=500ns。

4.3.2 基于异构系统平台的通信时延估测

本段介绍基于具体异构系统平台（如 ZYNQ702）的通信时延方法。通过编写 PS 与 PL 端的通信代码，进行仿真获取基于时钟的通信时延计算公式。通信代码见 9.3.5 段。

在 ZYNQ7020 中，双口 RAM 是由在 PL 端的 BRAM 构建而成，PS 端通过工作时钟为 100MHz 的 AXI 总线与 BRAM 控制器相联，由 BRAM 控制器对该 BRAM 进行读写操作。

4.3.2.1 非 DMA 方式

在 VIVADO 中构建如图 4-4 所示的非 DMA 通信的测试电路图。BRAM 端口 A 为 PS 使用端口，端口 B 为 PL 逻辑处理电路直接使用端口。

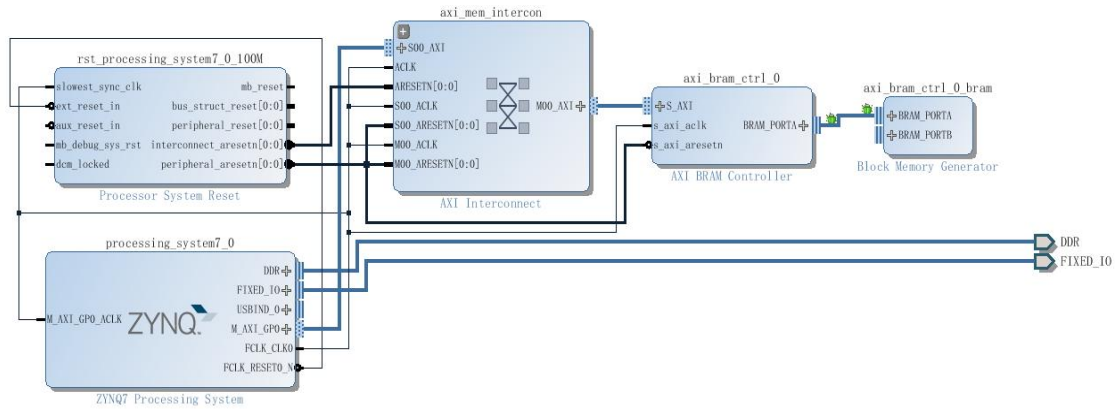

图 4-4 非 DMA 通信代码 IP 核图

①PS 至 PL 通信操作

PS 向 PL 传输数据测试程序是对端口 A 连续 4 次写操作。运行后可通过集成逻辑分析器 ILA (Integrated Logic Analyzer) 获取 BRAM PORT A 的总线时序图，图 4-5 是一次单字传输的总线时序图。

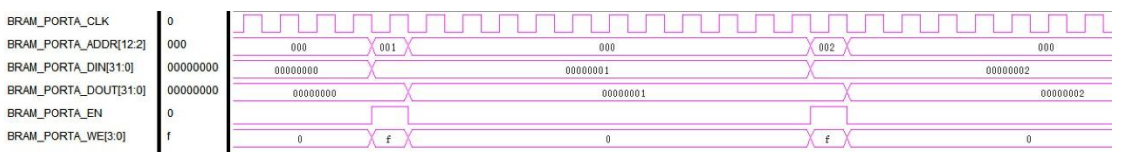

图 4-5 PS 至 PL 非 DMA 方式传输的总线时序图

由图中可知，一个单字传输操作时长为两个相邻 BRAM_PORTA_EN 之间的间隔，共有 12 个总线时钟周期（BRAM_PORTA_CLK），即 120ns。

值得注意的是，程序代码中一个字的通信指令在执行中，虽然总线上数据写操作只占用 1 个总线时钟周期，即 10ns，但总线上还会有 10 个时钟周期的其它时延。所以代码中一次 PS 至 PL 的通信操作实际用 12 个总线时钟周期，不能简单用总线上的写操作时长（1 个总线时钟周期）来计算。

②PL 至 PS 通信操作

PL 向 PS 传输数据测试程序是对端口 A 进行 4 次连续读入。

运行后可通过集成逻辑分析器 ILA 获取 BRAM PORT A 的总线时序图，图 4-6 是一次单字传输的总线时序图。

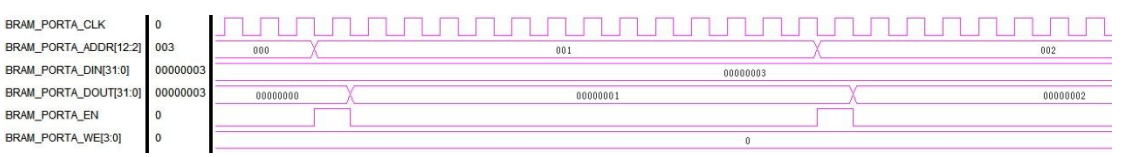

图 4-6 PL 至 PS 非 DMA 方式传输的总线时序图

由图中可知，一次传输操作时长为两个相邻 BRAM_PORTA_EN 之间的间隔，共有 14 个总线时钟周期（BRAM_PORTA_CLK），即 140ns。

4.3.2.2 DMA 方式

在 VIVADO 中构建如图 4-7 所示的测试电路图，在非 DMA 的测试电路图上增加了 AXI Central DMA 和 AXI SmartConnect 两个部件。

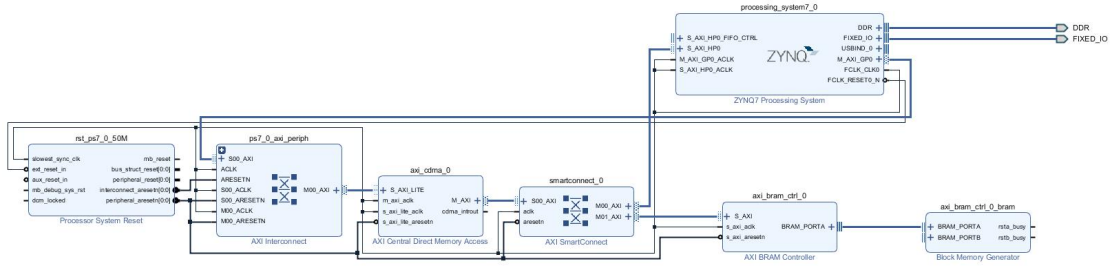

图 4-7 DMA 方式软硬通信测试电路连接图

与非 DMA 方式相似，BRAM 端口 A 为 PS 使用端口，端口 B 被逻辑处理电路直接使用。软硬件通信主要体现在 MPU 对 BRAM 端口 A 的访问。

①PS 至 PL 通信操作

PS 向 PL 传输数据测试程序是对端口 A 采用 DMA 方式写入，一次写 4 个字。运行后可通过集成逻辑分析器 ILA 获取 BRAM PORT A 的写入总线时序图，如图 4-8 所示。两次连续的 DMA 方式通信间隔 76 个时钟（如图 4-8 所示），对 BRAM 进行数据访问共用 4 个时钟周期，其时序如图 4-9 所示。

图 4-8 PS 至 PL DMA 方式传输的 BRAM 总线时序图

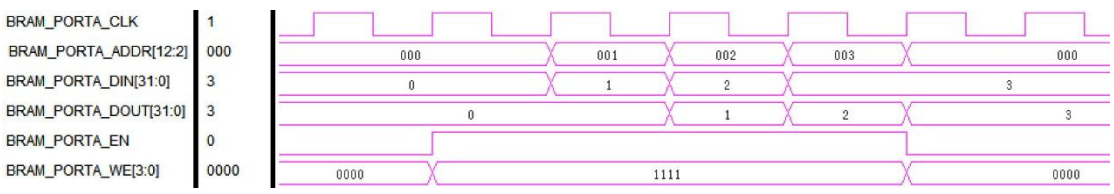

图 4-9 PS 至 PL DMA 方式传输的 BRAM 总线数据传输时序图

两次 DMA 数据传输之间需要有 76 个时钟周期用来对 DMA 控制器进行三次寄存器写操作。

因此，一次 N 个字的 DMA 方式通信所用的时长为（N+76)*10ns。

②PL 至 PS 通信操作

PL 向 PS 传输数据测试程序是对端口 A 采用 DMA 方式读入，一次读 4 个字。运行后可通过集成逻辑分析器 ILA 获取 BRAM PORT A 的读取总线时序图，如图 4-10 所示。两次连续的 DMA 方式通信间隔 74 个时钟，对 BRAM 进行数据访问共用 4 个时钟周期，其时序如图 4-11 所示。

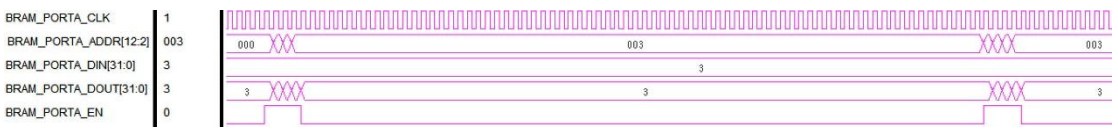

图 4-10 PL 至 PS DMA 方式传输的 BRAM 总线时序图

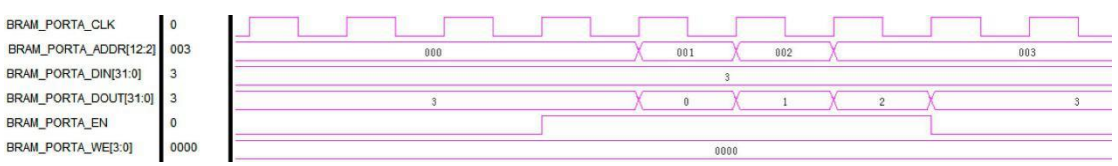

图 4-11 PL 至 PS DMA 方式传输的 BRAM 总线数据传输时序图

两次 DMA 数据传输之间需要有 74 个时钟周期用来对 DMA 控制器进行三次寄存器写操作。因此，一次 N 个字的 DMA 方式通信所用的时长为（N+74)*10ns。

综合前面的实测数据可以看出，每次通信的时长不仅仅有数据传输的时长，而且还有通信总线上的控制时长。对于任一次通信，其时长T_c可以表示为

$$T_c = T_d + T_w$$

其中T_d为数据传输时长，通常与数据量有关，而 T_W 为通信控制操作时长，通常与操作类型有关，不同操作类型所需要的额外控制操作不同。

通信分为 PS 至 PL 和 PL 至 PS 两个方向，每个方向分为单字通信和批量通信。

对于 PS 至 PL 方向，单字通信的数据传输时长为 T^{sl}_d，单字通信的控制操作时长为 T^{sl}_{ws}，一次 N 个字的批量通信数据传输时长为 $N\times T^{sl}_d$，一次批量通信的控制操作时长为 T^{sl}_{wb}。

对于 PL 至 PS 方向，单字通信的数据传输时长为 T^{ls}_d，单字通信的控制操作时长为 T^{ls}_{ws}，一次 N 个字的批量通信的数据传输时长为 $N\times T^{ls}_d$，一次批量通信的控制操作时长为 T^{ls}_{wb}。

在前面所用的测试平台中，这些参量分别为：T^{sl}_d=10ns，T^{sl}_{ws}=110ns，T^{sl}_{wb}=760ns，T^{ls}_d=10ns，T^{ls}_{ws}=130ns，T^{ls}_{wb}=740ns。

至此，某一个软件模块与相连的硬件模块之间的通信时延估计可以采用如下方法进行估计。

假定某软件模块的代码中通信信息如下：

PS 至 PL 的批量通信量为 N_{sl} 个字，共用 K_{sl} 次批量传输操作；PS 至 PL 的单字通信共 M_{sl} 次。 PL 至 PS 的批量通信量为 N_{ls} 字，共有 K_{sl} 次批量传输操作；PL 至 PS 的单字通信共 M_{ls} 次。其通信时延估计公式 4-4。

$$T_d = N_{sl} \times T_d^{sl} + K_{sl} \times T_{wb}^{sl} + M_{sl} \times (T_d^{sl} + T_{ws}^{sl}) + N_{ls} \times T_d^{ls} + K_{ls} \times T_{wb}^{ls} + M_{ls} \times (T_d^{ls} + T_{ws}^{ls}) \quad \text{(公式 4-4)}$$

若使用被段的测试用例可以得到通信时延计算：

$$T_d = N_{sl} \times 10 + K_{sl} \times 760 + M_{sl} \times 120 + N_{ls} \times 10 + K_{ls} \times 740 + M_{ls} \times 140$$

在智能嵌入式系统设计中，根据控制操作通常采用单字通信，少量数据采用单字通信，大量数据采用批量通信。以前面的测试平台来说，少于 8 个字的数据采用多次单字通信比采用批量通信时延小。

第 4.4 节本章小结

本章介绍了智能嵌入式系统软件任务、硬件任务以及 PS 与 PL 端通信性能获取方法，这些方法提供了初步性能指标，但在实际开发中，特别在性能指标为非常关键的智能嵌入式系统设计和开发中，要获取精确的性能指标，以便为智能嵌入式系统设计和开发提供准确的性能指标，使得开发的系统具有最优的性能。

习题

4.1 分别使用 Matlab 工具和 C++语言获得计算下列算法的最大时间，最小时间及平均时间,以及相应的软件功耗：

(1) 请实现对一维数组[3,4,1,8,0,5,14,10,12,20,23,24,2,17,6,18,9,19]的排序。（请勿使用 Matlab 自带的 sort 函数。建议使用冒泡、快排等常用算法。）

(2) 实现 1+2+3+……+10000 的求和。

4.2 使用 Vivado HLS 把上题 4.1 中(1)和(2)再做一遍，获取硬件执行时间、LUT 数和触发器 FF 数。（选择至少两个目标开发板，比较获得硬件性能指标）

4.3 设通信总量 N 为 10000 字节，总线宽度 W 为 32 位，一次传输一个字节需要 8 位，每次总线时长即访问周期 TB 为 20ns，计算完成这个通信量传输所需要的时延。若通信总量 N 为 9981 字节，其他相同，问完成这个通信量传输需要多长时间？

4.4 在一款异构系统平台上通过编写通信代码测试 PS 与 PL 端通信时延。