Co-Projeto Hw/Sw

Grupo: 11
Alunos: - João Ramiro nº 81138
- José Vieira nº 90900

Part 1

Consider your PS-only system

1. List the memory regions available (linker script) in your system to place your program (code and data sections) and explain briefly to which physical memory components they correspond

Name	Base Address	Size (Hex)	Size (Bytes)
ps7_ddr_0	0x100000	0x1FF00000	511MB
ps7_ram_0	0x0	0x30000	192KB
ps7_ram_1	0xFFFF0000	0xFE00	~63,5KB

Quando se gera o *linker script*, é dada ao programador a possibilidade de escrever as *data sections*, *code sections* e a *heap* e *stack*, tanto na DDR como nas RAMs. Neste caso foi escolhida a RAM0 por se encontrar fisicamente mais próxima do processador, diminuindo o *critical path*. As RAMs correspondem à *On- Chip Memory* (OCM) que se encontra ligada no PS. Por *default*, está dividida em duas memórias, RAM0 e RAM1, cuja a soma das suas capacidades corresponde a 256KB.

2. For the matprod1 application:

a) What is the total size of your program (including heap and stack)?

text	data	bss	dec	hex	filename
73368	2584	27024	102976	19240	matprod_1.elf

O tamanho da *heap* e da *stack* já estão considerados dentro destes valores, pois, o ficheiro matprod_1.elf.size resume todas as secções existentes (matprod_1.elf), onde se incluem estas duas.

b) Indicate the base address of the memory region where you placed your program (all sections).

Neste caso foi utilizada a RAM0, cujo o seu *base address* é 0x0.

c) What matrix dimensions did you use for the demonstration? #define MAT_SIZE ??

d) Which physical memory component did you select to store the matrices and which base address did you set for the matrices storage? #define MATA_START_ADD 0x????????

Foi verificado que o tamanho total do programa era de 0x19240 bytes. Como o *base address* da RAM0 é 0x0, então apenas foi utilizado um endereço superior ao referido, e divisível por 4 (endereços de 32-bit). Sendo escolhido o endereço 0x19244.

Onde último endereço ocupado (incluindo as 3 matrizes) é:

l_addr = MATA_START_ADD + 3×(4×50×50)

⇔l_addr = 0x19244 + 0x7530 = 0x20774 < 0x30000

O último utilizado é menor a RAM0, logo todo o programa ficou dentro desta memória.

e) What was the execution time of the matprod1 matrix multiplication application?

Output took 8989156 clock cycles.

Output took 13829.47 us.

Part 2

Consider your PS+PL system

3. Which base address has been assigned to the axil_mult_ip ?

0x43c0_0000

4. Indicate the number of resources needed to implement your PL design (in terms of LUTs, BRAMs and DSPs).

Resource	Utilization	Available	Utilization %
LUT	425	17600	2.41
LUTRAM	60	6000	1.00
FF	622	35200	1.77
DSP	3	80	3.75

São utilizados 3 DSPs, pois, como se trata de uma multiplicação de inteiros de 32-bit, então, estes são separados em dois números de 16-bit cada. Considerando os inteiros A e B:

$A = A_1 \times 2^{16} + A_0$

$B = B_1 \times 2^{16} + B_0$

A multiplicação de A com B, será:

$A \times B = A_1 \times B_1 \times 2^{32} + A_1 \times B_0 \times 2^{16} + B_1 \times A_0 \times 2^{16} + A_0 \times B_0$

O resultado é truncado, sendo igual aos 32-bit menos significativos, resultantes da multiplicação.

5. Could your PL system use a faster clock frequency? Justify briefly.

Como o Worst Negative Slack (WNS) deu negativo ($-9.734ns$), não podemos aumentar a frequência de relógio. Em vez disso, para que o seu valor fosse positivo, deveria ser diminuída, pois este valor corresponde à margem entre o *slowest delay path* entre registos e o fim do ciclo de relógio. No entanto, como testado, o programa funciona corretamente, onde o resultado da multiplicação é o real, o que significa que é um *false critical path*.

6. Include below the C code (kernel of the main function) you used to implement the multiplication of the matrix elements using the hw_mult_ip.

```
volatile int *reg0 = (int *)0x43C00000; // Input 1
volatile int *reg1 = (int *)0x43C00004; // Input 2
volatile int *reg3 = (int *)0x43C0000C; // Result

main:
for (i=0; i<N1; i++) {
        for (j=0; j<N3; j++) {
                MEMC(i,j) = 0;
                for (k=0; k<N2; k++){
                        *reg0 = MEMA(i,k);
                        *reg1 = MEMB(k,j);
                        MEMC(i,j) += *reg3;}}}
```

7. Explain briefly how the interface between the PS and the PL is implemented.

Para conseguir comunicar entre o PS e o PL, existem os portos GP *slave* e/ou *master*, dependendo se os periféricos adicionados no PL são *slave* ou *master*. Como o *Processing System* é *master*, foi habilitado um dos *General purpose* AXI *master*, neste caso, a *interface* 0. Nesta interface, os endereços dos periféricos inseridos no PL estão compreendidos entre 0x4000_0000 e 0x7FFF_FFFF. Como referido o *base address* atribuído ao multiplicador implementado é o 0x43c0_0000 e o número de registos é 4, ou seja:

&Reg0 = 0x43C0_0000
&Reg1 = 0x43C0_0004
&Reg2 = 0x43C0_0008
&Reg3 = 0x43C0_000C

Quando se quer escrever no registo Reg0, apenas temos de realizar a operação: *reg0 = x;. Já para a sua leitura é a operação: y = *reg0;.

8. What is the performance overhead of executing the integer multiplications (one by one) on the PL? Explain briefly the overhead obtained.

O *overhead* deve-se, essencialmente, às comunicações utilizadas entre o PS-PL e vice-versa, pois as operações de multiplicação são feitas em paralelo ao PS, logo o seu tempo de cálculo não representa um atraso temporal. Neste caso são feitas $2N^3 writes + N^3 reads = 3N^3$ comunicações, onde foi obtido o seguinte tempo de computação:

Output took 52767682 clock cycles. Output took 81181.05 us.

Comparando com o tempo obtido no primeiro laboratório (13829.47 us), o *overhead* é de:

$$81181.05us - 13829.47us = 67351.58us$$

O que torna o programa $81181.05\ us\ /13829.47us \simeq 5.87$ vezes mais demorado.