POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE

École Polytechnique Montreal

ELE8307 Laboratoire

Rapid prototyping of digital systems
Labratory 2

Élève :
Hossein Askari

Enseignant :
Jean-Pierre David

September 21, 2018

Contents

1 Lab2 Overview

In this Lab, we got familiar with Intel's NIOSII soft processor. We used DE2 board to implement our design on it. We attached a VGA IP core to the processor so that we would be able to show images on the monitor. In the other part of the lab, we generated a Board Support Package (BSP) for our NIOSII + VGA IP so that we would be able to run software and have access to the monitor.

In Part 4 of the lab we got familiar with NIOSII assembly language. We were asked to re-write a C function in assemblyto see the performance boost and get to know the assembly language of NIOSII.

Finally, in the last part of this lab, we ran a solar planet animation program on our system. We then investigated and profiled the performance of the code that is running on the system. We used GNU profiler to find out which functions are the best candidates for optimization. We optimized different part of the code and checked for any performance speed up.

2 Lab2 Part 1-3

In this part of the lab, we got familiar with NIOSII processor. NIOSII is a 32-bit embedded-processor architecture designed specifically for the Intel's family of FPGAs. NIOSII is a soft processor, meaning that the processor is not fabricated inside in the FPGA silicon. However, it can be configured through software.

We started by instantiating the NIOSII processor, we then added our VGA IP to the project. Next, we instantiated a PLL module. The PLL had to provide a 50MHz clock for the NIOSII processor, a 25MHZ clock for the VGA IP and a 50MHZ clock with -3ns phase delay for DRAM. The following image shows the configured system in Qsys.

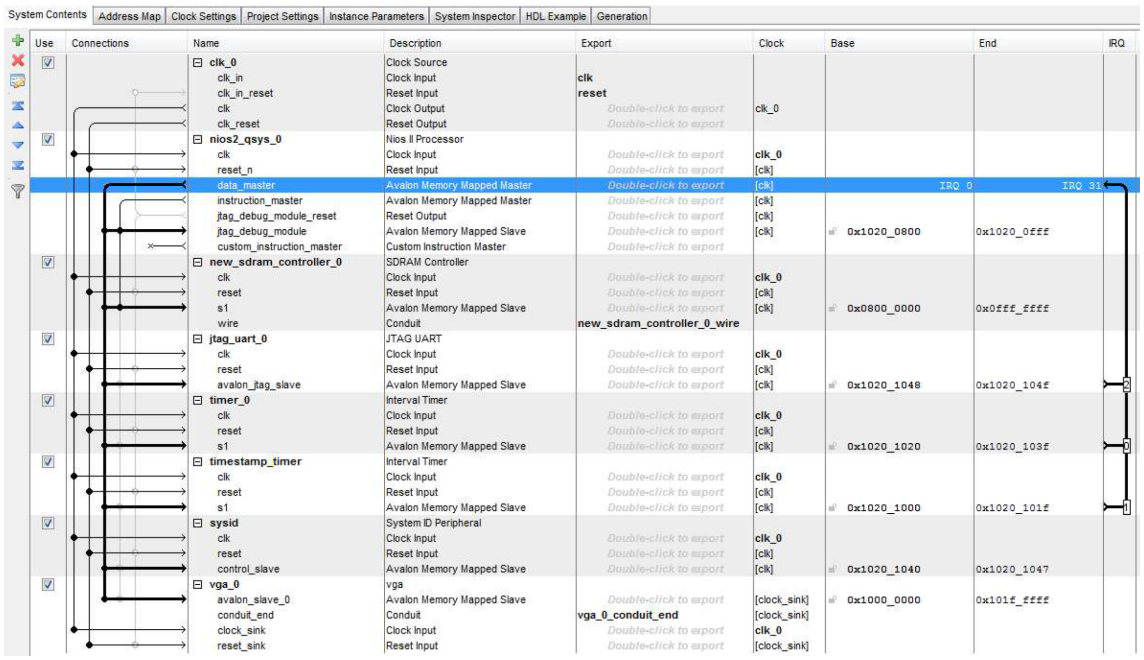

Figure 1: This figure illustrates the configured system in Qsys. As it can be seen, our system has a VGA IP and a NIOSII processor. We also have j-tag for debugging.

We then exported our design to generate the BSP for the system. For this, we used a Quartus built-in software that is built on top of Eclipse IDE. Generating BSP from the configured hardware system is done automatically by the tool. This process is essential since it will gives software access to hardware. In this process, the base address for different modules is automatically assigned. Also, the required libraries for running software on a NIOSII system is also added to the project. The crucial tip that we found while doing this part was the base addresses for reset vector. It is very important to properly set this address. Otherwise, the system might not be able to come out of reset. We used the default values provided by the BSP editor. However, occasionally, the system goes to an unstable state. In these scenarios, reseting the processor several times was the only solution to get back to a stable state.

Once we got everything setup properly, we were asked to run a basic Hello World application on NIOSII. Running this test would show that the system is configured correctly. It also shows that the system could communicate with DRAM. However, to check if the VGA IP is accessible or not, a Hello World application would not be sufficient.

To test whether the VGA IP is accessible to the system or not, we were asked to add the source code for **SolarSystem3D** application. This application uses some simple drivers to draw 3-D planets on the VGA monitor. After adding the source code for the project, we observed that **SolarSystem3D** starts an animation in which it draws 3-D planets on the VGA. However, the delay on drawing the planet was very noticeable. In the following parts of the lab, we will try to fix this by optimizing the source code to run the code better and faster.

3 Lab2 Part 4

In this part, we were asked to review the code base for the **SolarSystem3D** and come up with an assembly code to replace **ss_planet_drawBall** function written in C. We started by looking at the dis-assembly code for the **ss_planet_drawBall** which is written in C. We got some intuition on how to write a NIOSII assembly code. Also, it helped us to understand why this code can benefit from optimization. On the first look, we notice that the original C function (which has less than 20 lines of code) is mapped to more than 140 lines of assembly code. Also, the dis-assembly showed us that the compiler tried to put everything on the stack and use the frame pointer as a reference to the variables.

```
1  void ss_planet_drawBall( int x0, int y0, int radius, int color ) {
2            ss_planet_drawBall:
3  00803f40:   addi sp,sp,-44
4  00803f44:   stw ra,40(sp)
5  00803f48:   stw fp,36(sp)
6  00803f4c:   addi fp,sp,36
7  00803f50:   stw r4,-16(fp)
8  00803f54:   stw r5,-12(fp)
9  00803f58:   stw r6,-8(fp)
10 00803f5c:   stw r7,-4(fp)
```

Figure 2: Dis-assembly of the original code for **ss_planet_drawBall** written in C. The figure shows that compiler uses the stack to store values.

For instance, the first few lines of dis-assembly for **ss_planet_drawBall** is illustrated in Figure 2. As it can be seen, the function arguments that are stored in r4, r5, r6 and r7 are immediately stored back in the stack. This will make processor to have a longer access time to these variables compared to the case where these values are stored in the registers. We took this into account while writing the assembly code for **ss_planet_drawBall**. To write into the IO, the original C code was using the **ecran2d_setPixel** function which uses **IOWR** function. This function uses the base address and an offset to write data to the IO located at the based address plus the offset. We used two different method to perform the same task. As suggested in the lab document, we used **stwio** instruction to write the data to the base address. We also tried to call the C function from the assembly code. Figure 3 shows how we accomplished this.

```
add     r4,  r0,  r8          /* pass i        */
add     r5,  r10, r15         /* pass y+y0     */
add     r6,  r0,  r17         /* pass color    */
call    ecran2d_setPixel      /* call function */
add     r4,  r0,  r8          /* pass i        */
sub     r5,  r15, r10         /* pass y-y0     */
add     r6,  r0,  r17         /* pass color    */
call    ecran2d_setPixel      /* call function */
```

Figure 3: Assembly code that shows how to call a C function from assembly code.

As it can be seen, to correctly call the function, the arguments for the C function needs to be stored inside r4, r5 and r6 registers. Then, we need to use the assembly instruction **call** to call the function. This method works just fine however the first methd which uses the **stwio** has a better performance.

Warning!

Calling a function inside an assembly code requires the programmer to properly store the return address. Otherwise when the code is executed completely, executing **ret** will return the **p**c to the last called function. Figure 4 shows how to store the return address properly in an assembly code.

```
.section .text
.global _planete_drawball_opt
_planete_drawball_opt :
    add  r18,  r0,  ra

...

    END_OF_CALL:
        sub  ra,   ra,  ra
        add  ra,   r18, r0
        ret
.end
```

Figure 4: Assembly code that shows how to store return address.

As it can be seen, the return address is is stored into r18 register right at the beginning of the function call. Then, at the end of the function call, we restore the return address from r18. In between, the programmer has to make sure he does not over write the r18 value.

```
Flat profile:

Each sample counts as 0.001 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 15.76     10.66    10.66   894165     0.00     0.00  __muldf3
 14.44     20.44     9.78  2097851     0.00     0.00  __pack_d
  9.96     27.18     6.74  3852527     0.00     0.00  __muldi3
  7.64     32.35     5.17   935469     0.00     0.00  _fpadd_parts
  7.38     37.35     5.00  4232607     0.00     0.00  __unpack_d
  6.49     41.74     4.39   109127     0.00     0.00  __divdf3
  5.59     45.52     3.78   109362     0.00     0.00  __ieee754_sqrt
  3.32     47.77     2.25  3337750     0.00     0.00  __unpack_f
  3.12     49.88     2.11  1806319     0.00     0.00  __pack_f
  2.85     51.81     1.93                             read
  2.73     53.66     1.85   800766     0.00     0.00  __mulsf3
  2.06     55.06     1.40    38848     0.00     0.00  __ieee754_acos
  1.71     56.21     1.16   761782     0.00     0.00  _fpadd_parts
  1.40     57.16     0.94    39564     0.00     0.00  matrice4d_product_v4d
  1.32     58.05     0.89   499950     0.00     0.00  __subdf3
  1.08     58.79     0.73    39564     0.00     0.00  ss_planet_getBall3DIntensity
  0.94     59.42     0.63                             __malloc_unlock
  0.92     60.04     0.62   721850     0.00     0.00  __addsf3
  0.81     60.59     0.55   199026     0.00     0.00  __clzsi2
  0.81     61.14     0.55    22078     0.00     0.00  __kernel_sin
  0.79     61.68     0.53   435519     0.00     0.00  __adddf3
  0.77     62.20     0.52    80443     0.00     0.00  __make_fp
  0.61     62.61     0.41    17922     0.00     0.00  __kernel_cos
  0.60     63.02     0.41    79340     0.00     0.00  sqrt
  0.60     63.43     0.40   118188     0.00     0.00  __isnand
  0.56     63.81     0.38   119522     0.00     0.00  __extendsfdf2
  0.47     64.13     0.32    38848     0.00     0.00  acos
  0.46     64.44     0.31    18336     0.00     0.00  __ieee754_rem_pio2
  0.39     64.70     0.27   161152     0.00     0.00  __floatsisf
  0.35     64.94     0.23    39424     0.00     0.00  sin
  0.33     65.16     0.23    80443     0.00     0.00  __truncdfsf2
```

Figure 5: GNU profiler result for Part4 of the lab.

4 Part 5

We were asked to report the GNU profiler result at the of Part 5 of the lab. Figure 5 illustrates the result of GNU profiler for Part4.

? Question

Outre les calculs en points flottants, quel est le point chaud de l'application sur lequel il faudrait s'attarder afin de l'accélérer ?

As it can be seen in 5, the floating point calculation specially `__muldf3` and `__muldi3` are taking a lot of calculation time. On the other hand, `__pack_d` , `__unpack_d` , `_fpadd_parts` , `__divdf3` , `__ieee754_sqrt` , `__pack_f` , `__unpack_f` , `__mulsf3` and `__ieee754_acos` are among instructions that are taking a lot of computation time.

5 Part 6

In this part, based on the analysis in the previous part, we were asked to optimize the code. There were three parts that were asked to be optimized.

5.1 Part 6.A

In this part, we were asked to replace all double variables to use float in their calculations. The following shows the GNU performance analysis after we made this change.

```
Flat profile:

Each sample counts as 0.001 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 16.78     10.73    10.73   894165     0.00     0.00  __muldf3
 14.19     19.81     9.07  2020155     0.00     0.00  __pack_d
 10.21     26.34     6.53  3852527     0.00     0.00  __muldi3
  8.89     32.02     5.68   935469     0.00     0.00  _fpadd_parts
  7.51     36.82     4.80  4154911     0.00     0.00  __unpack_d
  4.68     39.82     2.99   109362     0.00     0.00  __ieee754_sqrt
  4.66     42.79     2.98    70279     0.00     0.00  __divdf3
  4.10     45.42     2.62  3415446     0.00     0.00  __unpack_f
  3.34     47.55     2.14  1845167     0.00     0.00  __pack_f
  2.91     49.42     1.86   800766     0.00     0.00  __mulsf3
  2.33     50.91     1.49    38848     0.00     0.00  __ieee754_acos
  1.54     51.90     0.99   761782     0.00     0.00  _fpadd_parts
  1.54     52.88     0.99   499950     0.00     0.00  __subdf3
  1.47     53.82     0.94    39564     0.00     0.00  matrice4d_product_v4d
  1.22     54.60     0.78    39564     0.00     0.00  ss_planet_getBall3DIntensity
  0.98     55.23     0.63   721850     0.00     0.00  __addsf3
  0.92     55.82     0.59    41041     0.00     0.00  __divsf3
  0.87     56.38     0.56    22078     0.00     0.00  __kernel_sin
  0.85     56.92     0.55   160178     0.00     0.00  __clzsi2
  0.84     57.46     0.54    17922     0.00     0.00  __kernel_cos
  0.81     57.98     0.52   435519     0.00     0.00  __adddf3
  0.76     58.47     0.49                             __malloc_unlock
```

Figure 6: GNU profiler result for Part6.A.

The following snippet of code shows where we changed the double precision to float to achieve the results in Figure 6.

```
inline int  ss_planet_getBall3DReferenceIntensity( float x, float y, float z, int radius

        // Balle 3D : R2 = (x-x0)2 + (y-y0)2 + (z-z0)2
    int         intensity;
        int             intensity_min = 0;
    float       r = sqrt(x*x + y*y);

    if( z <= 0 )
        intensity = 30*sin(acos(r/(float)radius));
    else
        intensity = intensity_min;

    return (int)intensity;
}
```

Figure 7: Code snippet of changing double to float in `ss_planet.c` code.

5.2 Part 6.B

In this part, we were asked to pre-calculate the function $sin(acos(x))$ and store them in a vector of size 100. To do that, we wrote a simple python code to generate these values. After using this vector and replace it with $sin(acos(x))$, we ran the GNU profiler again to see the results, Figure 8 illustrates the result. As it can be seen, there no more calls to `acos` and `sin` in this report

```
Flat profile:

Each sample counts as 0.001 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 12.91      2.58     2.58  3803926     0.00     0.00  __unpack_f
 11.55      4.88     2.31  2039407     0.00     0.00  __pack_f
 11.51      7.18     2.30    79340     0.00     0.00  __ieee754_sqrt
  9.61      9.10     1.92   878462     0.00     0.00  __mulsf3
  6.10     10.32     1.22   839478     0.00     0.00  _fpadd_parts
  4.67     11.25     0.93    39564     0.00     0.00  matrice4d_product_v4d
  4.33     12.12     0.87   417757     0.00     0.00  __muldi3
  3.58     12.83     0.72   114723     0.00     0.00  __pack_d
  3.48     13.53     0.69   305085     0.00     0.00  __unpack_d
  3.40     14.20     0.68    39564     0.00     0.00  ss_planet_getBall3DIntensity
  2.98     14.80     0.59   199010     0.00     0.00  __clzsi2
  2.94     15.39     0.59   721850     0.00     0.00  __addsf3
  2.44     15.87     0.49    79340     0.00     0.00  sqrt
  2.01     16.27     0.40    41041     0.00     0.00  __divsf3
  1.64     16.60     0.33        3     0.11     0.11  altera_avalon_jtag_uart_close
  1.37     16.88     0.27    80674     0.00     0.00  __extendsfdf2
  1.26     17.13     0.25    79340     0.00     0.00  __isnand
  1.12     17.35     0.22                             __malloc_unlock
  1.12     17.57     0.22                             read
  1.04     17.78     0.21   200000     0.00     0.00  __floatsisf
  0.98     17.98     0.20    15955     0.00     0.00  __muldf3
  0.97     18.17     0.19   117628     0.00     0.00  __subsf3
  0.84     18.34     0.17    80674     0.00     0.00  __make_dp
  0.82     18.50     0.16      587     0.00     0.00  matrice4d_product_m4d
  0.75     18.65     0.15    80443     0.00     0.00  __truncdfsf2
  0.73     18.80     0.15   118874     0.00     0.00  __fixsfsi
  0.68     18.94     0.14    42491     0.00     0.00  __fpcmp_parts_f
  0.60     19.06     0.12    78623     0.00     0.00  __fpcmp_parts_d
  0.60     19.18     0.12    15851     0.00     0.00  _fpadd_parts
  0.60     19.29     0.12        9     0.01     1.92  ss_planet_drawBall3D
```

Figure 8: GNU profiler result for Part6.B of the lab. There no more calls to `acos` and `sin` in this report.

5.3 Part 6.C

Finally, we were asked to remove the `sqrt` in the `ss_planet_getBall3DReferenceIntensity` function. We followed the instruction in the lab doc and we ran the profiler one more time. Figure 9 shows results for this part of the lab. As it can be seen, the amount of time spent for calculating `sqrt` has been reduced compared to the previous part.

```
Flat profile:

Each sample counts as 0.001 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 16.46      3.17     3.17  3720502     0.00     0.00  __unpack_f
 14.52      5.97     2.80   857248     0.00     0.00  __mulsf3
 13.37      8.54     2.57  1977913     0.00     0.00  __pack_f
  5.63      9.62     1.08    39564     0.00     0.00  matrice4d_product_v4d
  5.56     10.69     1.07    39776     0.00     0.00  __ieee754_sqrt
  5.44     11.74     1.05                             read
  4.83     12.67     0.93   838762     0.00     0.00  _fpadd_parts
  4.54     13.54     0.87   721134     0.00     0.00  __addsf3
  3.44     14.21     0.66   397975     0.00     0.00  __muldi3
  2.90     14.77     0.56        9     0.06     1.80  ss_planet_drawBall3D
  2.82     15.31     0.54    41041     0.00     0.00  __divsf3
  2.46     15.78     0.47    73727     0.00     0.00  __pack_d
  2.12     16.19     0.41   198250     0.00     0.00  __clzsi2
  1.97     16.57     0.38   185677     0.00     0.00  __unpack_d
  1.79     16.91     0.34   199284     0.00     0.00  __floatsisf
  1.31     17.16     0.25   118874     0.00     0.00  __fixsfsi
  1.14     17.38     0.22    39776     0.00     0.00  sqrt
  1.10     17.59     0.21   117628     0.00     0.00  __subsf3
  1.01     17.79     0.19                             __malloc_unlock
  0.78     17.94     0.15    15239     0.00     0.00  __muldf3
  0.74     18.08     0.14    41595     0.00     0.00  __truncdfsf2
  0.57     18.19     0.11    39775     0.00     0.00  __fpcmp_parts_d
  0.50     18.29     0.10    15135     0.00     0.00  _fpadd_parts
  0.48     18.38     0.09    41110     0.00     0.00  __make_dp
  0.47     18.47     0.09    39775     0.00     0.00  __ltdf2
  0.37     18.54     0.07    39776     0.00     0.00  __isnand
  0.36     18.61     0.07    41595     0.00     0.00  __make_fp
  0.36     18.68     0.07    41110     0.00     0.00  __extendsfdf2
  0.27     18.73     0.05    45324     0.00     0.00  ecran2d_setPixel
  0.25     18.78     0.05    39564     0.00     0.00  vecteur4d_init
  0.23     18.83     0.05        1     0.05     0.05  ecran2d_clear
  0.23     18.87     0.04      990     0.00     0.00  __ieee754_rem_pio2
```

Figure 9: GNU profiler result for Part6 of the lab. Time spent calculating `sqrt` has been reduced compared to the previous part.