SoC development on FPGA with application profiling

Tutorial 10 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2018-19

Table of Contents

tutorial outline

this tutorial deals with:

- system integration of hardware components for SoC development with Qsys
- tools for software application profiling
- design of a Nios II system equipped with a performance counter
- use of the performance counter API in a well-known example: delay computation on a sequence of Collatz trajectories, with user input of the sequence length
- HW/SW integration through BSP generation in the Monitor Program, compilation, loading on FPGA, debugging, and execution
- lab experience:
  design and implementation of a HW/SW system with similar structure and features as those of the example presented in this tutorial, using the same development and profiling tools, but for a different application

system integration in SoC development

development of a SoC with applications is a typical HW/SW codesign activity

it consists of design and development of components of both kinds, as well as their *integration* to form a single system

the Quartus tool utilized in this lab tutorial for the integration of hardware components in SoC development is *Qsys*

it is advised to consult the introduction to Qsys and to run the therein provided example on the DE1-SoC, to get familiar with using the tool

a slightly more complex example is the subject of the present tutorial:

- development of a SoC similar to the aforementioned one, but equipped with a component that enables accurate *profiling* of software applications (see next)
- development of a software implementation of the delay computation of Collatz trajectories, and of a software application to measure its execution time

tools for software application profiling

profiling a program: measuring the time spent in different parts of the program, to identify those which are critical to execution speed

useful in HW/SW codesign to figure out which program parts may deserve possible hardware acceleration, thence to estimate the achievable speedup

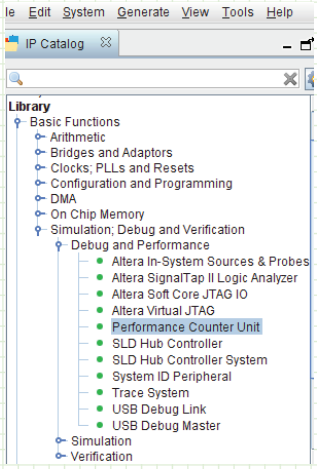
three tools considered in (fairly dated) document *Profiling Nios II Systems*:

GNU gprof: software measurement, high software overload, high measure distortion

Interval Timer: hardware measurement, minimal resource overload, limited distortion

Performance Counter Unit: hardware measurement, significant hardware overload, minimal distortion, upperbound (7) on no. of measurable program sections

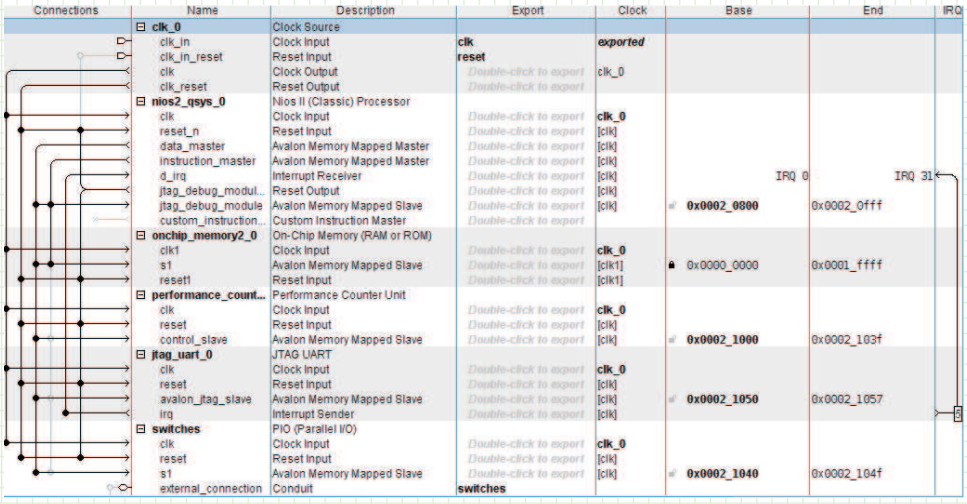
the third method is utilized here, since it yields the best accuracy and the easiest use within the program, while the aforementioned upperbound is no problem for the application at stake



construction of a Nios II system with performance counter

the figure shows the Qsys contents of the Nios II system with Performance Counter Unit

the scheme is similar to that of the example in the introduction to Qsys, except for the absence of the LEDs PIO and the presence of the profiling component



a simple, well-known example

the C function in the figure is a software implementation of the delay computation of a Collatz trajectory with given start point

the preprocessing directives, except the fourth one, relate to the hardware platform previously built with Qsys, see next

delay_collatz_timing.c +

```c
#include "altera_avalon_performance_counter.h"
#include "system.h"
#define switches (volatile unsigned int *) SWITCHES_BASE // from "system.h"
#define N_FACTOR (unsigned int const) 8           // *switches scale factor
#define pca (void *) PERFORMANCE_COUNTER_0_BASE          // from "system.h"

unsigned int delay_collatz(unsigned int x0) {
  int d = 0;
  int x = x0;
  int hx;
  while (x > 1) {
    d++;
    hx = x >> 1;
    if ((x % 2) > 0) {
      d++;
      x += hx + 1;
    }
    else
      x = hx;
  }
  return d;
}
```


use of the performance counter in the software application

unlike the previous lab experiences relating to hardware implementations of the subject function, the user input here determines the length of the sequence of trajectories to be generated in the main program, that is the number of function invocations

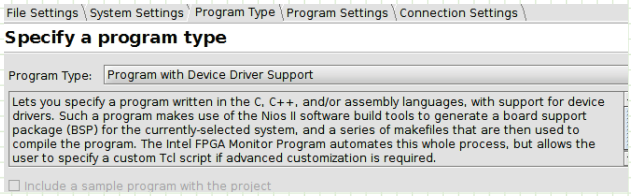
the switches input is multiplied by a scale factor, to get a reasonable test duration

delay_collatz_timing.c +

```c
int main() {
  unsigned int i, n, taps, c, x, d;
  n = *switches << N_FACTOR;
  alt_u32 cpu_freq = alt_get_cpu_freq();
  unsigned int seed = 0x1234;
  c = seed;
  PERF_RESET(pca);
  PERF_START_MEASURING(pca);
  for (i = 0; i < n; i++) {
  PERF_BEGIN(pca, 1);
    x = c;
    taps = (c & 0x1) ^ ((c & 0x4) >> 2) ^ ((c & 0x8) >> 3) ^ ((c & 0x20) >> 5);
    c = (taps << 15) | (c >> 1);
  PERF_END(pca, 1);
  PERF_BEGIN(pca, 2);
    d = delay_collatz(x);
  PERF_END(pca, 2);
  }
  PERF_STOP_MEASURING(pca);
  perf_print_formatted_report(
    pca,                            // peripheral's HW base address
    cpu_freq,                       // CPU frequency (Hz)
    2,                              // how many sections to print
    "traject_start",                // display-names of sections
    "delay_collatz");
  return 0;
}
```

the preprocessing directives, previously shown, enable the use of the *performance counter API* as well as of other symbols (SWITCHES_BASE in this case) defined in the software interface of the system built with Qsys

the interface is provided by the BSP, whose construction here is automated by the *Monitor Program*, following the choice of program type Program with Device Driver Support



other aspects of the BSP (e.g. compiler or linker options) may be specified by providing a custom Tcl script

in particular, while the default optimization level fixed by the Monitor Program is -O1, a different level, e.g. -O3, may be obtained by creating a one-line script (with extension .tcl):

```
set_setting hal.make.bsp_cflags_optimization -O3
```

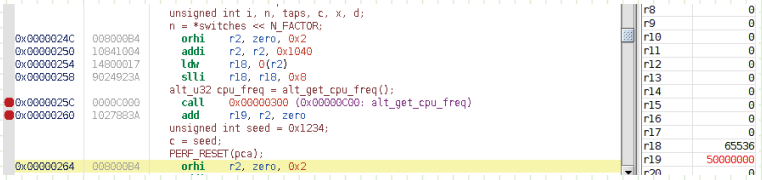
and providing its path in the input box BSP settings Tcl script (optional) within the Program Settings tab

C source-level debugging is also available in the Monitor Program (visualization of values of variables)

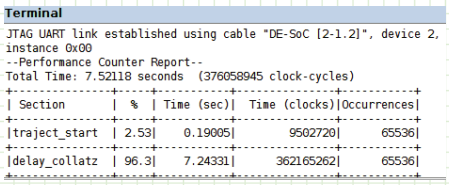
this requires compilation with optimization level -O0

the program disassembly remains accessible anyway, where to set breakpoints and to examine its execution status at critical points for correctness verification

for example, with breakpoints as in the figure, one may check the correctness of the computed no. of iterations and of the clock frequency, resp. in r18 and r19



after removal of all breakpoints, system reset and execution restart, the profiling module generates the performance report displayed in the figure



lab experience

the proposal aims at the design and implementation of a HW/SW system with similar structure and features as those of the example presented in this tutorial, using the same development and profiling tools, but for a different application; precisely, the work goes about:

- building a Nios II system on FPGA, equipped with a performance counter component for application profiling
- software development of a coprimality test program, using a GCD computation function while processing a sequence of number pairs
- HW/SW integration of system and application, with: application profiling using the API of the aforementioned component, BSP generation, compilation, loading and execution on the FPGA, with debugging if needed

references

recommended readings:

Introduction to the Qsys System Integration Tool - *For Quartus Prime 16.1*, Intel Corp. - FPGA University Program, November 2016

readings for further consultation:

Profiling Nios II Systems, AN-391-3.0, Altera Corp., July 2011

useful materials for the proposed lab experience:

Performance Counter Unit Core, Ch. 31 in: *Embedded Peripherals IP User Guide*, Intel Corp., UG-01085 | 2017.11.06

Intel FPGA Monitor Program Tutorial for Nios II - *For Quartus Prime 16.1*, Intel® FPGA University Program (November 2016)

source files for running the lab experience (in the lab reserved area)