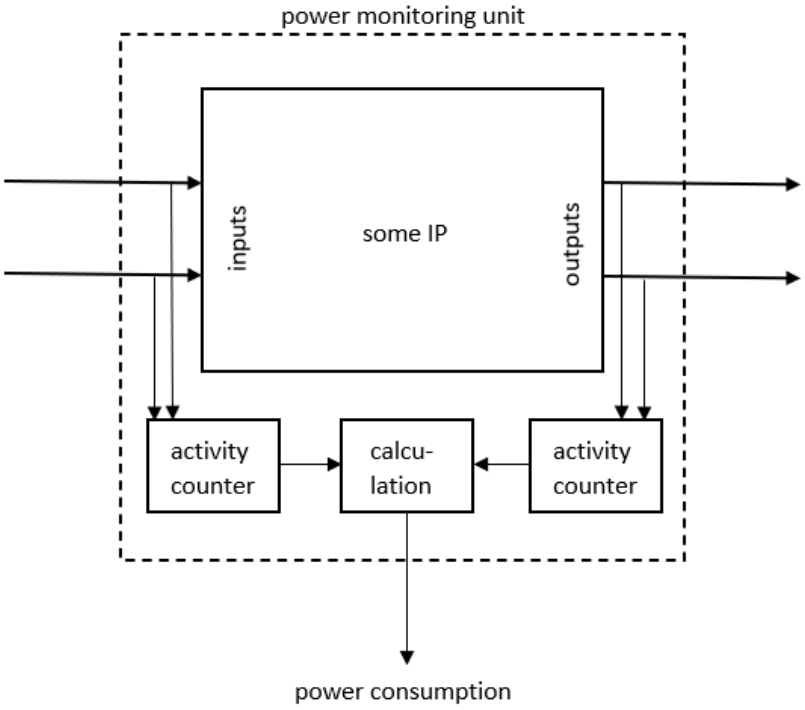
README.md

Run Time Module Power Monitoring for FPGA Applications

This is a university project at TU Vienna in the SoC Lab.

Abstract

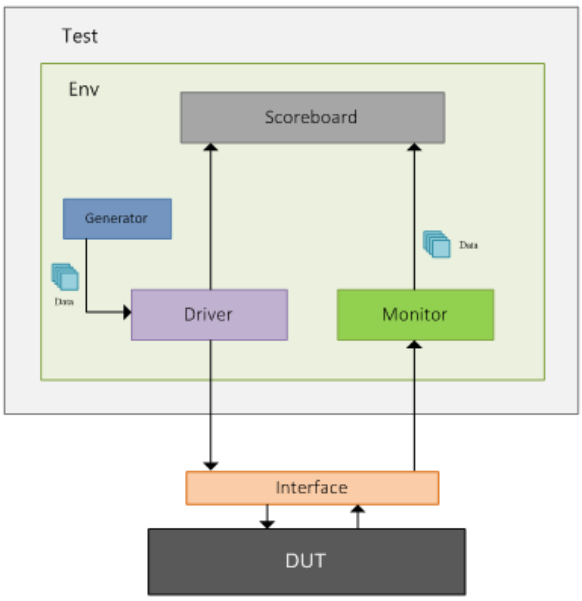
The usage of field programmable gate arrays (FPGAs) in low power applications increases, so the reduction of power consumption is of great importance. Although power analysis tools from different vendors exist, it can be important to be aware of the run time dynamic power consumption of individual applications.The goal of this project is to create a design which can measure the power consumption of basic modules or IP cores in run time by counting the toggling of specific signals. Counting the toggles of each signal from a module can strongly increase the required board area for the measurement unit. Therefore it is important to identify the most indicative signals. This can be done by sorting the signal activities based on previous generated switching activity files (.saif). The signals with the most activity are selected and a activity counter identifies the edges of these signals. Finally, a relationship between switching activity and power dissipation gives the power consumption. This method is described in more detail in the work of Lin et al. [1].
The following picture gives an idea of the structure. The "module-under-monitoring" is enveloped by the power monitoring unit, which includes the activity counters and power consumption calculation.



Description

Testbench:

The results in the vivado power report heavily depend on the applied testbench. So it is very important to create a testbench, which simulates the real behaviour as precisely as possible. Because testbenches in vhdl are not modular, scalable or flexible, we chose a SystemVerilog-Testbench.
We took the Testbench from here and modified it for our needs. The vivado project for our testbench is located in "vivado/SV_Testbench-vhdlMemory". The source files of our testbench can be found at "vivado\SV_Testbench_vhdlMemory\SV_Testbench_vhdlMemory.srcs\sim_1\imports\SV_Testbench_Memory.srcs\sources_1\new".



Our module-under-monitoring:

We chose a single port RAM as our module-under-monitoring. This allows us to generate a lot of traffic (by writing to and reading out of the RAM).

Activity counters:

The activity counters are used to detect signal toggles. At every rising edge of the system clock, they xor the input signal with the input signal from the previous clock cycle. Because the counter values are read every 10ms, the counter has to get resetted every 10ms.

Calculation:

In order to be able to calculate the power values as precisely as possible, we implemented a fixed point library ("fxd_arith_pkg.vhd"). The calculation is based on the following formula:

$$P_{dyn} = \sum_{i \in N} \alpha_i C_i V_{dd}^2 f$$

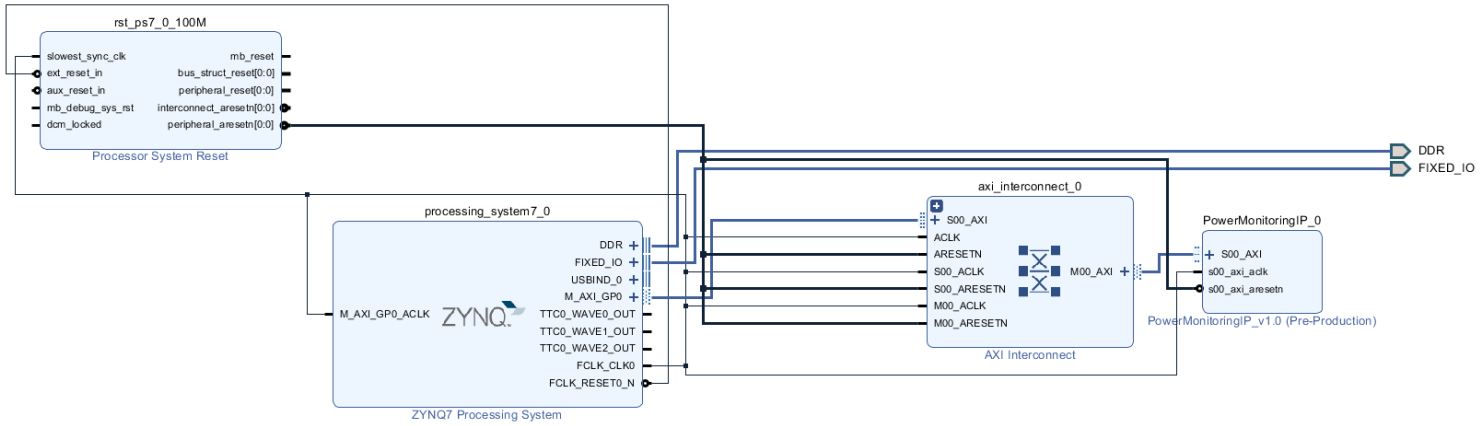
alpha stands for the switching activity (determined by the activity counters), C for the capacitance, Vdd for the supply voltage and f for the operating frequency.

VHDL Top-design:

The top-design connects all sub-modules (RAM, activity-counter and calculation). Changes to the signals-to-be-monitored can be made here (connect the signals you want to monitor with "s_ac_inp").

AXI and creating custom IP-Core:

We use the advanced extensible Interface (AXI) protocol for the communication between the Programmable System (PS) and Programmable Logic (PL) on the Zynq.
Vivado offers a guide, which helps with the creation of a IP-Core (for details see here). Our IP-Core is located at "vivado/ip_repo/PowerMonitoringIP_1.0/". Basically, we added the instantiation of our top-design in "PowerMonitoringIP_v1_0_S00_AXI.vhd".
The following picture presents the block design:



Embedded Application:

Basically we just modified "main.c" and left everything else unchanged.
First, we initialize the RAM by resetting and enabling it. Afterwards, the actual monitoring starts. Every 10ms a random value is written to the RAM and every second the calculated power value is read out. For that, we created functions like "WriteRAMData" or "PrintPower".

How to use:

Signal Selection:

1. You need to have the rtl-design of the modul-to-be-monitored. A description in VHDL would be ideal, because otherwise we can not guarantee that vivado can generate a bitstream out of 2 different description languages.
2. You have to write a testbench for the vivado power report, which should be as close as possible to the real world application (for details see testbench)
3. Synthesize and create the implementation of the modul-to-be-monitored in vivado.
4. Go to simulation settings and add the name of the saif-file (simply by typing in the name of the file)
5. Run the functional simulation on the implemented design. In this step, the saif-file is actually created.
6. Create the power report - dont forget to include the generated saif-file.
7. In the power report - go to I/O and sort them by highest to lowest switching rate.
8. Select the I/Os with the highest activity. In our design (RAM), we selected the three signal with the highest activity. Please note: signals with a switching rate higher than FCLK/2 cannot be monitored by our activity-counters. To take that into account, the multiplier in "rtpm_pkg.vhd" can be adjusted. If the I/O has several bits, our way was to select the 4 bits with the highest rate, wich is of course highly dependent on the data wich is read from/written to the RAM.

Adjustment of the monitoring-design:

1. Set the "activity_count" constant in "rtpm_pkg.vhd". (e.g.: if you monitor 9 signals, set it to 9)
2. Connect the signals (you want to monitor) with a activity counter in "top.vhd". This is done by connecting the bits of the signal with "s_ac_inp".
3. Open the vivado project "PowerMonitoring", refresh the ip catalog and update the PowerMonitoring-IP.
4. Generate output products
5. Generate the bitstream
6. Export hardware (with bitstream!)
7. Launch Xilinx SDK (workspace has to be "local to project")

Xilinx SDK:

1. Actually you don't have to change anything in "main.c" if you want to read out the values via UART.

How to run & preparations

1. Open the vivado project ("PowerMonitoring.xpr") and make sure that the bitstream is up-to-date.
2. Launch the SDK (workspace has to be "local to project")
3. Connect the Zedboard: UART and PROG via USB to the PC (and of course Power)
4. Program the FPGA: Go to Xilinx -> Program FPGA: Select the bitstream-file and press "Program"
5. Set up the serial communication via a terminal (e.g. Tera Term) and set baud rate to 115200
6. Go to the project navigator, right click on "PowerMonitoring" -> Run As -> Launch on Hardware (GDB)

Team organization

- Lead Architect: Alexander Ludwig (Team leader)
  Contact: e1525371@student.tuwien.ac.at
- Lead Hardware: Thomas Kotrba
  Contact: e1525909@student.tuwien.ac.at
- Lead Test and Integration: Luke Maher
  Contact: e1225441@student.tuwien.ac.at

Timetable

Task	Due Date	Status
Activitycounter RTL Design & Simulation	17.11.2020	Done
(Power) calculation RTL Design & Simulation	24.11.2020	Done
Create a signal selection algorithm	15.12.2020	Done
Selection of the module to be analyzed	15.12.2020	Done
Top module RTL Design & Simulation	15.12.2020	Done
Zedboard implementation (in vivado)	12.01.2021	Done
(Final) Testing and modifications	23.02.2021	Done

Prerequisites

- ghdl: for compiling and simulating VHDL code
- GTKWave: for viewing the simulation results
- Vivado: for power report generation, bitstream generation
- Xilinx SDK: for creating the embedded application
- ModelSim: for compiling and simulating VHDL code

References

[1] Z. Lin, W. Zhang, and S. Sharad. Decision tree based hardware power monitoring for run time dynamic power management in fpga. In 2017 27th International Conference on Field Programmable Logic and Applications (FPL), pages 1-8, 2017.