Embedded Systems project report

Alexandru Gabriel Bradatan

August 7, 2024

Contents

1 Project Data

Supervisors	Davide Galli, Davide Zoni
Participants	Alexandru Gabriel Bradatan - 10658858
Tags	Complex

2 Project Description

CVA6 is a 6-stage in-order and single issue processor core which implements the RISC-V instruction set with the I, M, A and C extensions. It can be configured as a 32- or 64-bit core (RV32 or RV64), called CV32A6 and CV64A6 respectively. It also implements three privilege levels M, S, U to fully support a Unix-like operating system, like Linux. The core is fully open source (repository) and written in SystemVerilog.

This project has two main goals:

1. Instantiate a CVA6 core
2. Boot a Linux image on a simulated CVA6 testbench and execute some programs

We are going to work on the latest release of the CVA6 core (5.1.0), use Vivado 2023.2 as an IDE and synthesizer and a customized version of Verilator as a simulator.

All artifacts of this project are available in this repository together with instructions.

2.1 CVA6 Instance

The FPGA family used for synthesis is the Artix-7. Specifically, the part used is `xc7a35tcsg324-1`. This part was chosen mainly because it is the one found on the entry level Arty A7-35T board and it is just enough for what we need. The CVA6 configuration we are going to use for this part is the `CV32A6_IMA_SV32_FPGA` config.

To instantiate a CVA6 core, we must create a Verilog module describing a minimal SoC composed of the CVA6 core and some essential peripherals, which are the Core-Level interrupt controller (for the timer interrupt) and the DRAM, all connected together by an AXI crossbar. The resulting module will receive as inputs the clock from the board's oscillator, the reset signal and other auxiliary signals for interfacing with the onboard memory.

Our board provides one 100MHz oscillator. This clock will feed directly into Xilinx's DDR3 Memory Interface Generator IP running at 325 MHz. This frequency has been chosen so that the memory has an input clock of 100MHz. The MIG IP outputs a clock signal (`ui_clk`) with frequency $325/4 = 81.25$ MHz. This signal, in the code reassigned to `dram_clk` for naming consistency with other DRAM-related signals, will be rescaled by the Clock Wizard Xilinx IP into 2 other clocks:

1. A 200 MHz reference clock used by the DRAM
2. A 50 MHz clock used by the CPU, the AXI bus and the peripherals

Since the memory and the AXI bus are in two different clock domains, we need to handle clock domain crossing. This is done by the AXI Clock Converter Xilinx IP. A diagram of the module's clock generation is provided in Figure 1.

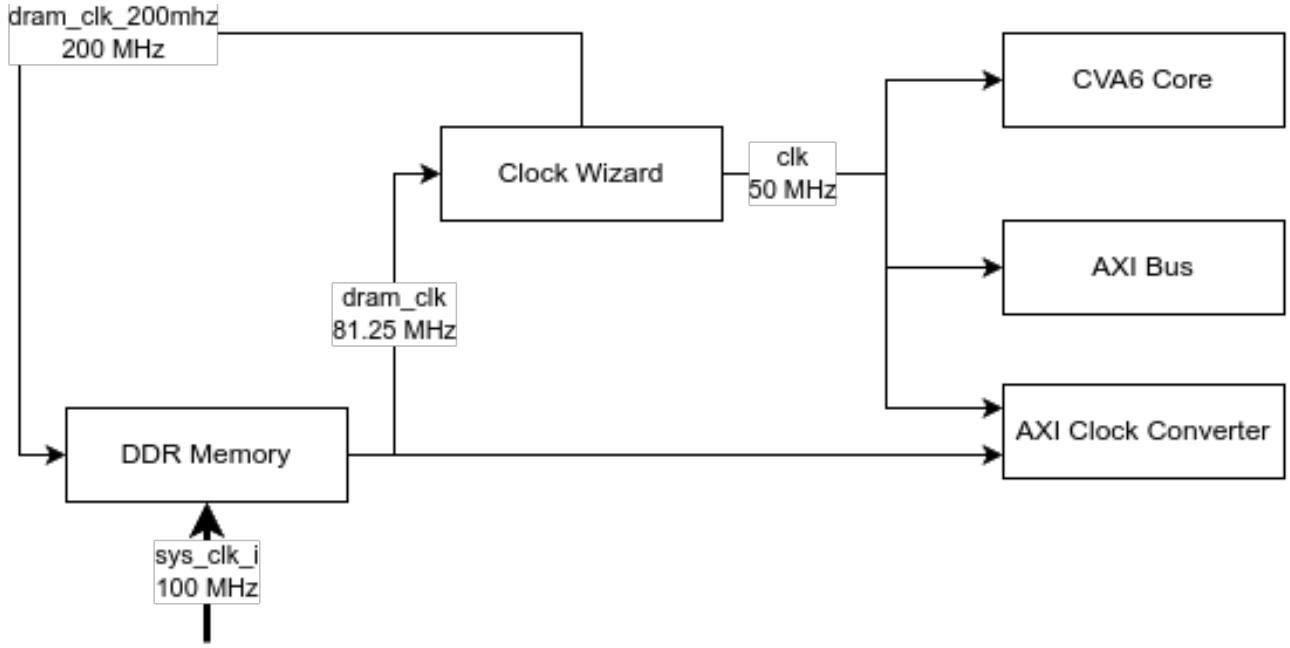

Figure 1: Diagram of the clock generation

This design is described by the `synth_harness` module. Synthesis succeeds and results in the resource usage outlined in Table 1.

Resource	Utilization	Available	Utilization %
LUT	16421	20800	78.95 %
FF	12096	41600	29.08 %
BRAM	16.50	50	33.00 %
DSP	4	90	4.44 %
IO	50	210	23.81 %
MMCM	2	5	40.00 %
PLL	1	5	20.00 %

Table 1: Post-synthesis utilization report

2.2 CVA6 Simulation

To have maximum compatibility with both the upstream Verilog code and any upstream scripts that may be needed, we have decided to use Verilator compiled with the in-tree patches. The same logic will be applied to any other auxiliary tools required for simulation. Vivado has been excluded from the beginning as the simulator since it is not able to correctly parse some otherwise valid Verilog code (specifically this and this lines).

The simulation testbench used is the one provided by upstream. It comprises of a 64-bit core with the `CV64A6_IMAFDC_SV39` configuration, a JTAG module, a behavioural SRAM for the text and a behavioural UART for interfacing with the host.

Compilation of the tools with the host compiler, compilation of the cross compiler and cross compilation of the Linux image are handled by scripts provided by the main repo and the SDK repo. Invocation of the build scripts with the correct parameters has been automated by us with a Makefile. To compile all tools and images needed for simulation, one can execute `make tools`.

Simulation is started by invoking an upstream provided Python script with the ELF image to run and the desired board configuration. As with compilation, correct invocation has been implemented as a Make target: one can simply run `make sim` to start a simulation of the core running a Linux image.

To connect to the simulated core, we can use OpenOCD with JTAG Remote BitBang. A simple OpenOCD script has been provided in the project repo.

3 Project Outcomes

3.1 Concrete Outcomes

The project created a Verilog module that correctly instantiates a configuration of the CVA6 core and synthesizes.

The project also built upon the tools provided by the upstream developers to enable us to easily build a RISCV Linux image and start a simulation of a CVA6 core running said image using Verilator.

3.2 Problems encountered

Unfortunately, we were not able to get a shell prompt through the simulated UART and execute programs. However, the simulation is working since we can run without errors until timeout and, by watching the utilization of the host CPU, we can see that something is being executed.

Adding to the problems is the fact that we could not find any documentation for simulating a full OS image. This left us with only trial and error and reverse engineering. Due to time constraints on our part, we did the best we could with our knowledge and skills. In the following, we will outline the main things we tried and various possibilities we considered.

To connect to the core, we tried using OpenOCD with the simple script contained in the project repo, which itself is an adaptation of an example one we found one in Spike's documentation. Once we tried to connect to the core, the connection would be established but the whole thing would crash after some time (usually around a couple of minutes). The OpenOCD output up until the crash is the following:

```
Open On-Chip Debugger 0.12.0
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : only one transport option; autoselect 'jtag'
Warn : `riscv set_prefer_sba` is deprecated. Please use
```

```
    ‘riscv set_mem_access‘ instead.
Info : Initializing remote_bitbang driver
Info : Connecting to localhost:9824
Info : remote_bitbang driver initialized
Info : This adapter doesn’t support configurable speed
Info : JTAG tap: riscv.cpu tap/device found:
    0x00000001 (mfg: 0x000 (<invalid>), part: 0x0000, ver: 0x0)
Info : datacount=2 progbufsize=8
Error: unable to halt hart 0
Error:   dmcontrol=0x80000001
Error:   dmstatus =0x00000c82
Error: Fatal: Hart 0 failed to halt during examine()
Warn : target riscv.cpu examination failed
Info : starting gdb server for riscv.cpu on 3333
Info : Listening on port 3333 for gdb connections
Error: Target not examined yet
Info : remote_bitbang interface quit
```

We think that this may be caused by an incorrect configuration of OpenOCD or the testbench. This is very possible due to our lack of familiarity with these tools. It may also be that the image was built incorrectly, however we don't think that is the case since we used the official SDK with the default options without any modification. Another possibility may be that OpenOCD is simply not supported. We do not think that's the case since OpenOCD connects just fine. Moreover, the Verilator logs explicitly output a port to which we can attach OpenOCD.

We also considered the simplest case: maybe the UART uses the same TTY as the simulation script. This seems not to be true because even after waiting for several minutes, the script's output doesn't display anything other than some debug information and doesn't accept input. To be sure, however, we would need to reverse the simulation script.