User's Guide to the Aquila SoC v1.0

Chun-Jen Tsai
Sep. 10, 2020

Aquila is a RISC-V RV32IMA compliant processor core that is designed as a reusable AXI4 IP for the Xilinx Vivado EDA tool. Currently, Aquila only supports M mode of RISC-V. Before synthesis of the hardware and compilation of the software, you must install Xilinx Vivado (2018.2 was used) and 32-bit RISC-V GNU toolchain (gcc 8.2.0 was used) on your system. The source-tree structure for Aquila HW-SW is as follows:

```
/ --+-- hw -- aquila_soc (the Aquila Vivado project workspace)
    |
    +-- sw --+-- uartboot (the boot code ROM)
             |
             +-- elibc (very small C standard library)
             |
             +-- Dhrystone (the classical CPU benchmark)
             |
             +-- test (testing of malloc(), uart I/O, and timer ISR.
```

1   Introduction to the Aquila Platform

The Vivado project "aquila_soc" contains the minimal architecture of an Aquila platform. The block diagram of the project is shown in Fig. 1. The red dash box in Fig. 1 contains two IPs, aquila, of the Aquila processor subsystem. By default, the boot code is loaded at the on-chip tightly-coupled memory (TCM) at 0x00000000 ~ 0x0000FFFF. The code, data, and stack areas can be changed through the boot code linker script. Upon the reset of Aquila, the boot code will be waiting for an application program to be transferred through the UART device, Xilinx axi_uartlite, at a baud rate of 115200. A user can use the "send file" menu item of TeraTerm to transfer a RISCV binary for execution. When the program is received, the boot code will load it into the addresses specified in the application linker script, either in TCM or in DDR3 DRAM.
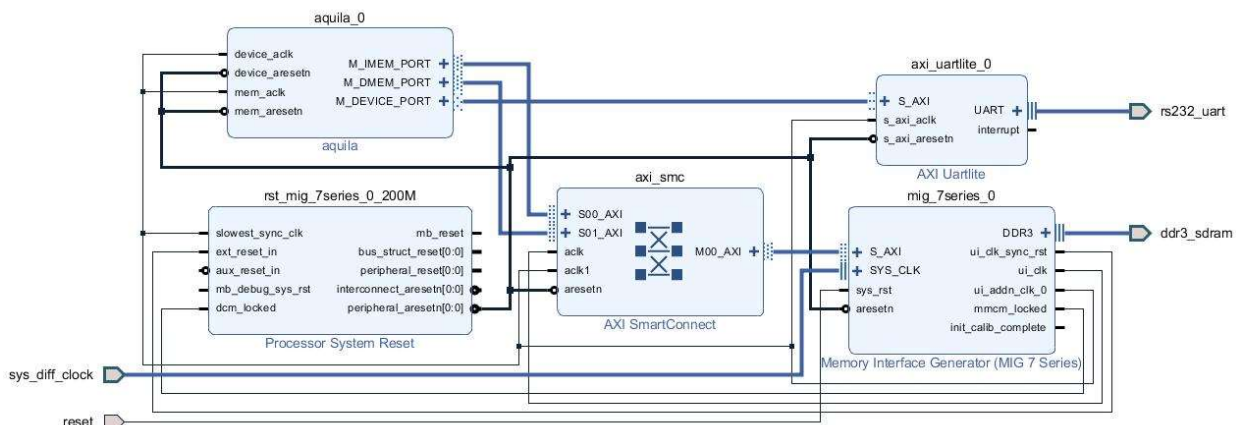

Fig. 1. The Vivado block diagram of a minimal Aquila SoC.

The Aquila processor core has three master ports for external I-Memory, D-Memory, and I/O Devices, respectively. In Fig. 1, the device port is connected directly to the axi_uartlite device. More devices can be connected to the Aquila core via an AXI Interconnect. Do not forget to edit the device addresses in Vivado block design tab as shown in Fig. 2. However, there is an address decoder in the Aquila IP, in `aquila_top.v`, that presumes the following system address map:

```
0000_0000 - 0FFF_FFFF: Tightly-Coupled Memory (TCM)
8000_0000 - BFFF_FFFF: DDRx DRAM memory (cached)
C000_0000 - CFFF_FFFF: device memory (uncached)
F000_0000 - F000_0010: CLINT I/O registers
```

Therefore, you must assign an address between 0xC0000000 and 0xCFFFFFFF to your I/O devices so that the I/O requests from the Aquila core will be redirected properly to the device port.

The program memory area (either in TCM or in DRAM) is further divided into code section, data section, heap section, and stack section specified by the linker script of a program. A typical setup of these sections for a Xilinx KC705 development board (with 1GB of DDR3 DRAM) are as follows:

```
0x8000_0000 - 0x8000_8FF0: program code section
0x8000_8000 - 0x8100_7FF0: data section
0x9000_0000 - 0xAFFF_FFF0: heap section
0xBFFE_0000 - 0xBFFF_0000: stack section
```

These section areas can be modified through the linker script. Note that the stack section is physically initialized in the boot code.
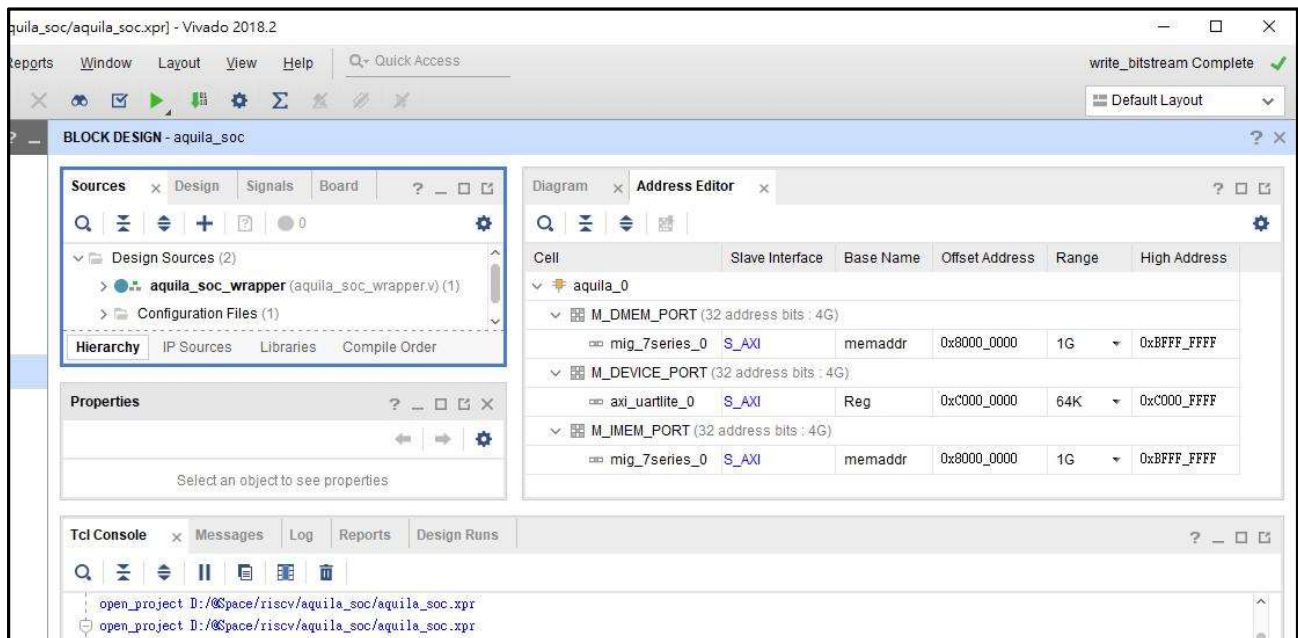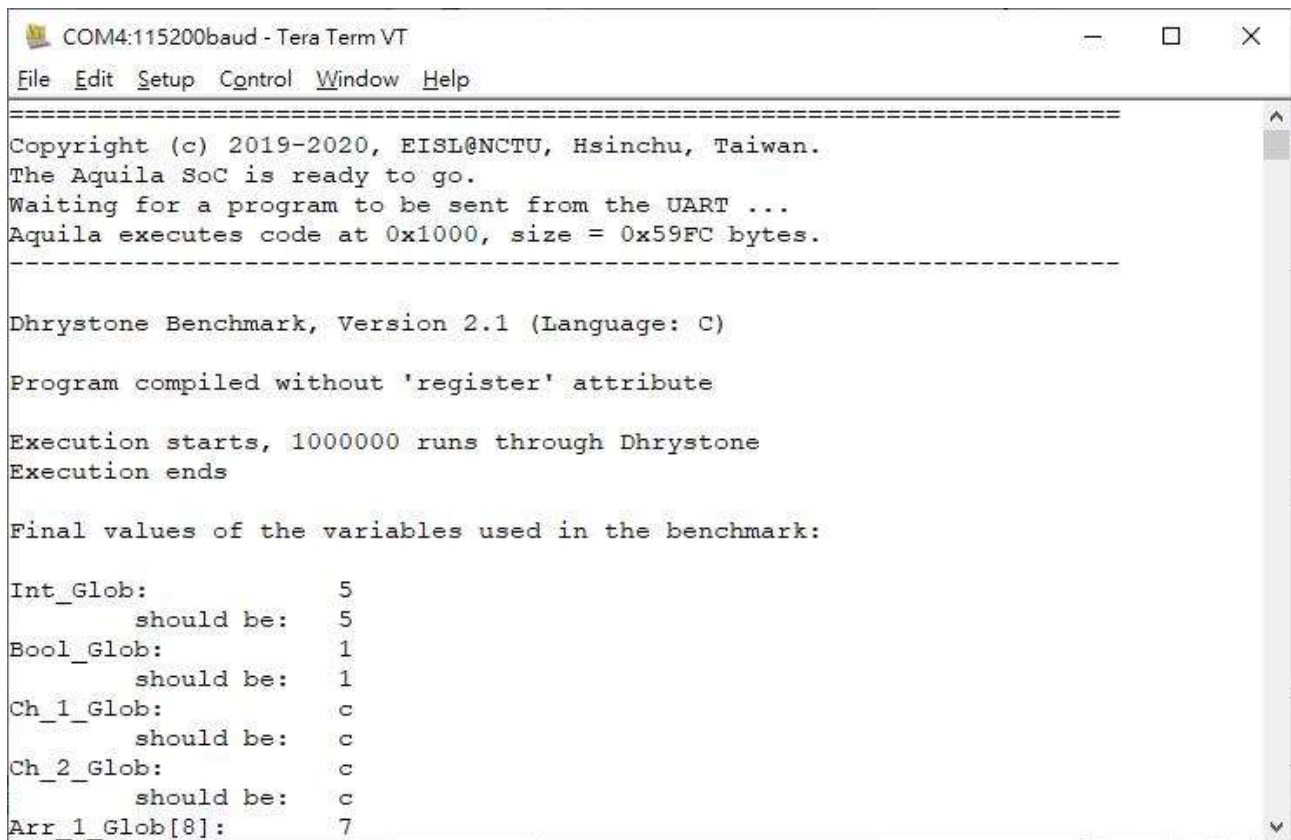

Fig. 2. Device address editor in Vivado.

Once you have generated and configured the bitstream for the target FPGA (KC705), you will see a welcome message showing on the terminal emulator (such as Tera Term) on the host PC (see Fig. 3).

The parameters for the UART terminal are 115200, 8-N-1.

Fig. 3. The boot message from the Aquila SoC.

2 The Software Samples of the Aquila SoC

In the software directory, `sw/`, there are four subdirectories that contains a UART boot code, a very small standard C library, a Dhrystone benchmark application, and a test application, respectively. To build the software, you can simply run "make" in each directory. For the small C library, there is no Makefile. Any program that depends on this library, such as the test program, should refer to the library in its Makefile. The software project under `aquila/sw/uartboot` generates a boot code with starting address at 0x00000000 and stack area at 0x00000A00 ~ 0x00000FF0. The target file `bootrom.mem` shall be copied to the Aquila Vivado project directory, under `aquila_soc/ip_repo/aquila/hdl/mem`. This is the ROM memory file of a small boot code that will be loaded into the DDRx main memory for execution.

Once the boot code is executed, a message will be sent to the UART device (Fig. 3). A terminal emulator running on the host computer can be used to interact with the Aquila SoC. A user can use the "Send file" menu command in Tera Term (see Fig. 4) to transfer a RISC-V binary executable in *.ebf format to the main memory for execution. Note that you must check the binary transfer mode in the "Send file" option. The *.ebf format is a binary executable created using 'objcopy', then prepended by a 8-byte length & starting address header.

A test program is located in the directory `sw/test`. To build `test.ebf`, simply type "make" under this directory. The test program depends on the 'elibc' library for basic I/O, tick count, and memory management support. Fig. 6 shows the execution result of the test program.
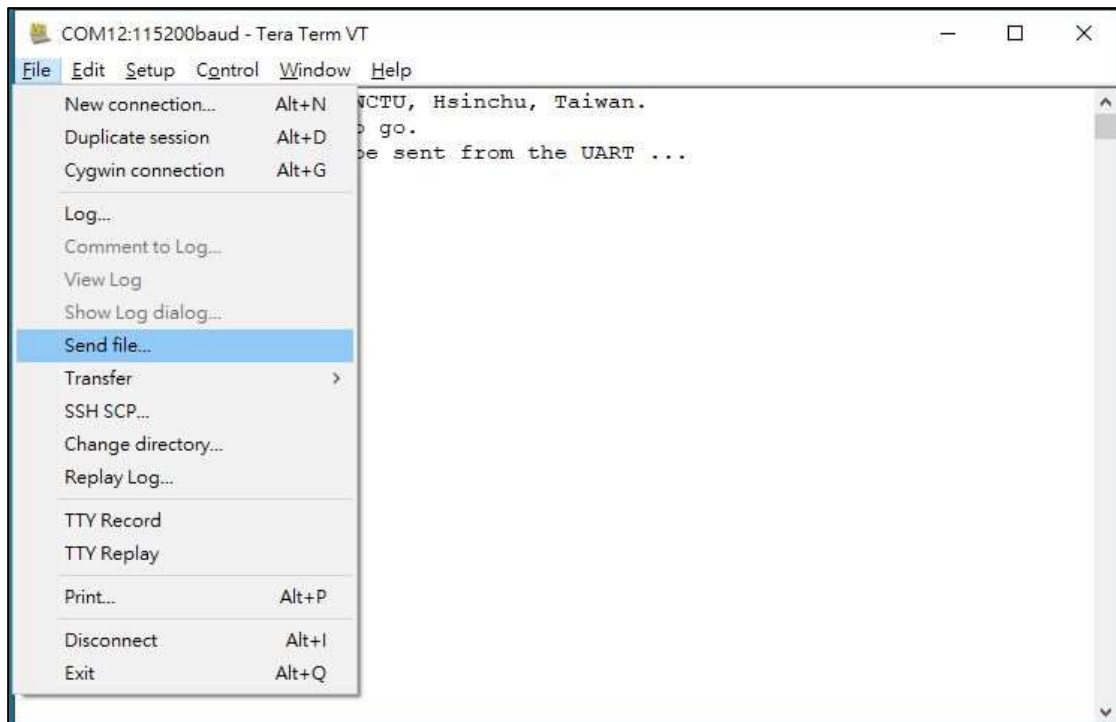

Fig. 4. Use "Send file" in Tera Term to load a binary executable to the DDRx main memory for execution.
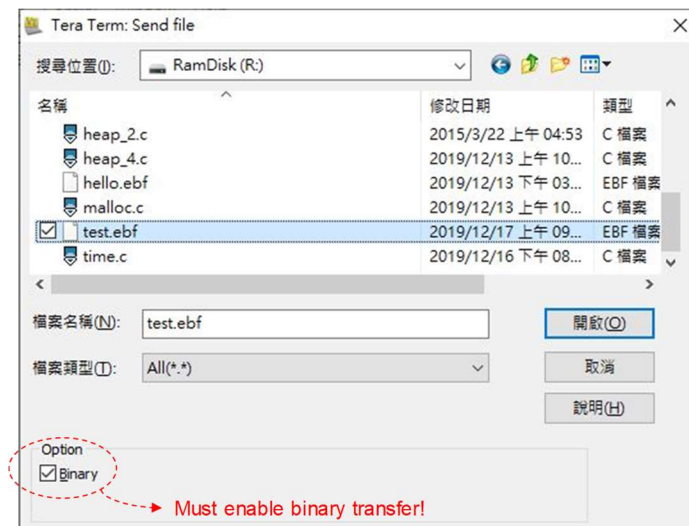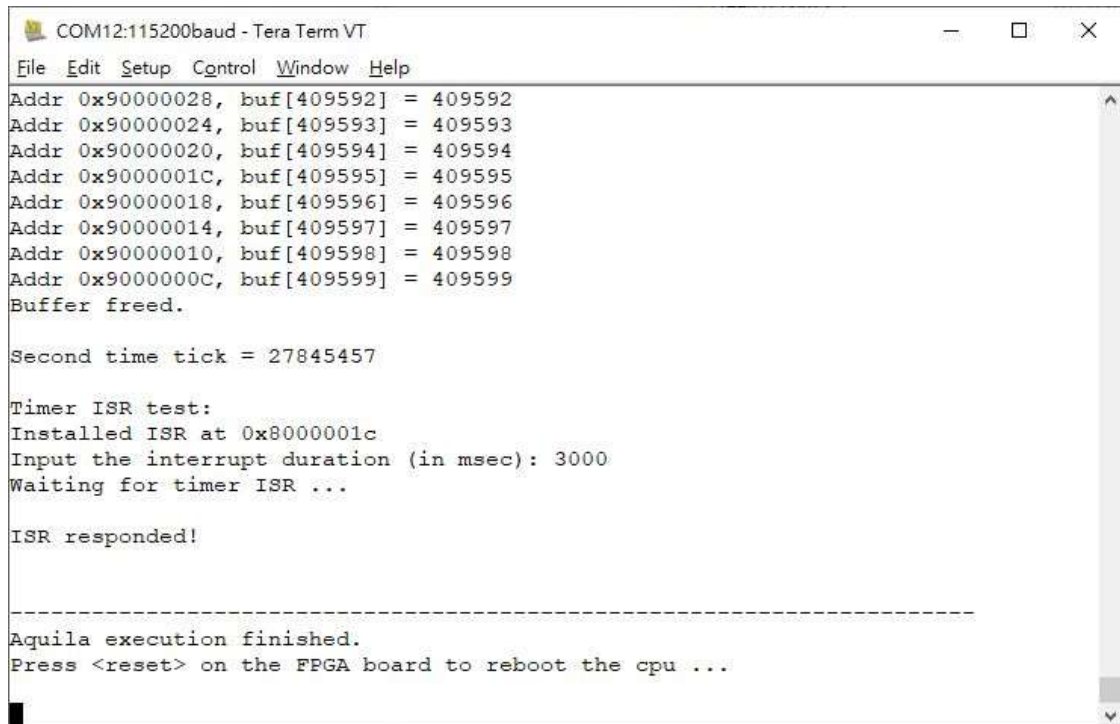

Fig. 5. Selection of the *.ebf executable using binary transfer mode.

```
COM12:115200baud - Tera Term VT                                    —    □    ×

File  Edit  Setup  Control  Window  Help

Addr 0x90000028, buf[409592] = 409592
Addr 0x90000024, buf[409593] = 409593
Addr 0x90000020, buf[409594] = 409594
Addr 0x9000001C, buf[409595] = 409595
Addr 0x90000018, buf[409596] = 409596
Addr 0x90000014, buf[409597] = 409597
Addr 0x90000010, buf[409598] = 409598
Addr 0x9000000C, buf[409599] = 409599
Buffer freed.

Second time tick = 27845457

Timer ISR test:
Installed ISR at 0x8000001c
Input the interrupt duration (in msec): 3000
Waiting for timer ISR ...

ISR responded!


----------------------------------------------------------------
Aquila execution finished.
Press <reset> on the FPGA board to reboot the cpu ...
```

Fig. 6. Execution of the test program.