Implementing VexRiscv Based Murax SoC on Arty A7 Artix-7 PCB from Digilent and Enabling JTAG Connection through Xilinx's BSCANE2 Debug IP

By
Pradeep Krishnamurthy – Student Research Assistant, OFFIS e.V.
Frank Poppen – Senior Research Engineer, OFFIS e.V.

www.offis.de

Acknowledgement
This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project SATiSFy under contract no. 16KIS0821K, and within the project Scale4Edge under contract no. 16ME0127.

Contents

1. Introduction

Up-to-date FPGA evaluation boards, like the Digilent Arty A7 mounting a Xilinx Artix-7 FPGA, come with an integrated FTDI chip which makes programming and debugging quite easy. In our work, we synthesized the VexRiscv based Murax processor to an Artix-7 FPGA and at first lead out the JTAG relevant signals of the Riscv core to the board's Pmod Header to connect to a dedicated Olimex JTAG Adapter through a second USB cable. As it turns out, this extra effort on hardware can be minimized by use of some Xilinx Debug IP named BSCANE2. Collecting the required information on how to do this was tedious. So we came to the decision to document our path to success with this short report. We expect that the reader is familiar with the `README.md` to be found at https://github.com/SpinalHDL/VexRiscv and that the reader is capable of generating the Murax SoC as it is described there.

2. SpinalHDL - Generation of Murax SoC with BSCANE2

The BSCANE2 allows access between the internal FPGA logic and the JTAG Boundary Scan logic controller. This allows for communication between the internally running design and the dedicated JTAG pins of the FPGA.

Steps to enable Bscane2

- After cloning all files from https://github.com/SpinalHDL/VexRiscv, go to the path: `src/main/scala/vexriscv/demo` and find the `Murax.scala` file.
- Comment out the following lines to remove the toplevel jtag I/O pins in `Murax.scala`. Be aware that line numbers as given could move with future changes to the file:

```
[164] val jtag = slave(Jtag())
      …
[392] val jtagClkBuffer = SB_GB()
[393] jtagClkBuffer.USER_SIGNAL_TO_GLOBAL_BUFFER      <>
        io.jtag_tck
[394] jtagClkBuffer.GLOBAL_BUFFER_OUTPUT              <>
        murax.io.jtag.tck
      …
[398] murax.io.jtag.tdi <> io.jtag_tdi
[399] murax.io.jtag.tdo <> io.jtag_tdo
```

```
[400] murax.io.jtag.tms <> io.jtag_tms
```

- In the `Murax.scala` file, delete the line:
  ```
  [253] io.jtag <> plugin.io.bus.fromJtag()
  ```
- And add the lines:
  ```
  [254] val jtagCtrl = JtagTapInstructionCtrl()
  [255] val tap = jtagCtrl.fromXilinxBscane2(userId = 2)
  [256] jtagCtrl                                          <>
          plugin.io.bus.fromJtagInstructionCtrl(ClockDoma
          in(tap.TCK))
  ```

  Changing the above lines, removes the Murax SoC's JTAG ports as pins of the FPGA and inserts the BSCANE2 Xilinx Debug IP to which the JTAG signals are now connected.
- Add the following import statement at the beginning of `Murax.scala`:
  ```
  import spinal.lib.com.jtag.JtagTapInstructionCtrl
  ```

With these changes in place, you generate the SoC with a demo program already in ram by use of:

```
sbt "runMain vexriscv.demo.MuraxWithRamInit"
```

A Verilog file is generated with the name `Murax.v` next to four `.bin` files inside the `VexRiscv` folder. These files are the input to the Xilinx FPGA synthesis. Inside the `Murax.v` file, we can see that the BSCANE2 ports are instantiated, confirming that the BSCANE2 has successfully been instantiated within the Murax SoC as a debug brige to JTAG.

3. Xilinx Vivado - Programming Arty A7 FPGA

There are many applications to program a FPGA. In our work we referred to the freely available Xilinx Vivado 2020 application to synthesize and program the FPGA. Vivado is readily available at Xilinx website and free of cost to download. This document assumes that the reader is able to setup and execute FPGA synthesis projects. The following is not a step by step tutorial, but gives general guiding information.

Programming the FPGA

- Create a new project and choose the board. In our case it is the Arty A7-35 (`xc7a35ticsg324-1L`).
- Copy the mentioned files (`.v` and `.bin`) of the previous section from the Vexriscv folder into the Vivado project in e.g. the path:
  `project_name.srcs\sources_1\imports\Downloads`
- Create a toplevel file by instantiating Murax I/O ports in it to blink the LED's on the Digilent board. (Note: The program to blink the LED's is already present in the four `.bin` files with the `Murax.v` file). The toplevel file and constraint `arty_a7.xdc` file, if required, can be found and reused from the path: `VexRiscv/scripts/Murax/arty_a7`, but you need to make sure that all the JTAG ports of Murax are commented or deleted in the toplevel file. Remember: we removed them in Section 2 and connected them internally to the BSCANE2 debug

bridge.

- Be aware that line numbers as given could move with future changes to the file. The lines to remove from toplevel file are:

```
[43]  reg  tesic_tck,tesic_tms,tesic_tdi;
[44]  wire tesic_tdo;
[45]  reg  soc_tck,soc_tms,soc_tdi;
[46]  wire soc_tdo;
[47]
[48]  always @(*) begin
[49]    {soc_tck,  soc_tms,  soc_tdi  } = {tck,tms,tdi};
[50]    tdo = soc_tdo;
[51]  end
      …
[56]  .io_jtag_tck(soc_tck),
[57]  .io_jtag_tdi(soc_tdi),
[58]  .io_jtag_tdo(soc_tdo),
[59]  .io_jtag_tms(soc_tms),
```

- Also remove any JTAG port to pin assignments from any constraint file.
- Next, click Generate Bitstream and program the FPGA with the bit file. You can see the LED's blink and Murax SoC has been programmed into the FPGA.

4. Debugging - Using OpenOCD and GDB

- Clone and setup openocd with the steps as provided by https://github.com/SpinalHDL/openocd_riscv
- You basically have to provide two files for OpenOCD to connect successfully through the FPGA into the Murax SoC inside it:
  1. `usb_connect.cfg` (interface configuration)
  2. `soc_init.cfg` (take over the control of the CPU)
- `usb_connect.cfg`
  You can take it from …
  https://github.com/SpinalHDL/SaxonSoc/blob/dev-0.3/bsp/digilent/ArtyA7SmpLinux/openocd/usb_connect.cfg …
  without modifications as we would say, but make sure to check the entire path in your system for the files `xilinx-xc7.cfg` and `jtagspi.cfg`. If required, adapt the find and path for the lines:

```
[29]  source [find cpld/xilinx-xc7.cfg]
[30]  source [find cpld/jtagspi.cfg]
```

- `soc_init.cfg`
  https://github.com/SpinalHDL/SaxonSoc/blob/dev-0.3/bsp/digilent/ArtyA7SmpLinux/openocd/soc_init.cfg
  You can take it but you need to: set cpu_count to 1 and remove lines 22 to 35 as shown in the result below:

```
set cpu_count 1

for {set i 0} {$i < $cpu_count} {incr i} {
```

```
        target create saxon.cpu$i vexriscv -endian little -
            chain-position  $TAP_NAME  -coreid  $i  -dbgbase
            [expr $i*0x1000+0x10B80000]
        vexriscv readWaitCycles 40
        vexriscv cpuConfigFile $CPU0_YAML
        if {$SPINAL_SIM != "yes"} {
            vexriscv jtagMapping 3 3 0 1 2 2
        }
    }

    for {set i 0} {$i < $cpu_count} {incr i} {
        targets saxon.cpu$i
        poll_period 50
        init
        soft_reset_halt
    }

    puts " done"
```

- Run openocd:

```
openocd -c "set CPU0_YAML ../VexRiscv/cpu0.yaml"\
        -f tcl/interface/usb_connect.cfg   \
        -f tcl/interface/soc_init.cfg
```

On success you should be able to see something like

```
Open  On-Chip  Debugger  0.10.0+dev-01231-gf8c1c8ad-dirty
        (2021-05-03-10:57)
Licensed under GNU GPL v2
For bug reports, read
      http://openocd.org/doc/doxygen/bugs.html
../../cpu0.yaml
Info : auto-selecting first available session transport
        "jtag".  To  override  use  'transport  select
        <transport>'.
xc7.tap
Info : set servers polling period to 50ms
Info : clock speed 5000 kHz
Info  :  JTAG  tap:  xc7.tap  tap/device  found:  0x0362d093
        (mfg: 0x049 (Xilinx), part: 0x362d, ver: 0x0)
Info : starting gdb server for saxon.cpu0 on 3333
Info : Listening on port 3333 for gdb connections
requesting target halt and executing a soft reset
 done
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

- Information on setting up a riscv compiler and debugger toolchain are to be found at:
  `https://github.com/riscv/riscv-gnu-toolchain`
- With openocd running you can now connect a debugger to port `3333`.
- A demonstration software to compile and debug with the Murax SoC can be found at https://github.com/SpinalHDL/VexRiscvSocSoftware in the path `VexRiscvSocSoftware/projects/murax/demo`. With a `make` you create

the `.elf` in the `build` directory from which you then give the command:

```
riscv64-unknown-elf-gdb demo.elf
```

- The riscv debugger is started with the `demo.elf` program and is ready to be connected to the CPU. Do so by issuing the following command in its window:
- `target remote localhost:3333`
  This command will connect the GDB server to OpenOCD
- `load`
  This command will load the program into the FPGA. Whenever you decide to make changes to the demo software and recompiled it, you need to upload the resulting new executable to the CPU in this way.
- `monitor reset halt`
  This command resets the Murax CPU and halts it to receive further commands.
- `continue`
  From her on you should be able to execute a regular debug session with your VexRiscv based Murax SoC on the FPGA.