A Single-board computer using the AJIT lite processor with an attached accelerator

Madhav Desai
Department of Electrical Engineering
IIT Bombay

May 4, 2024

The processor

- AJIT 1x1x32 Lite: AJIT single core, SPARC-V8 ISA.
  - 8KB Icache (2-way set associative).
  - 8KB Dcache (2-way set associative).
- Hardware debug support unit.
- Integrated peripherals: interrupt controller, timer, serial, scratch pad, performance counters, SPI master, I2C master.

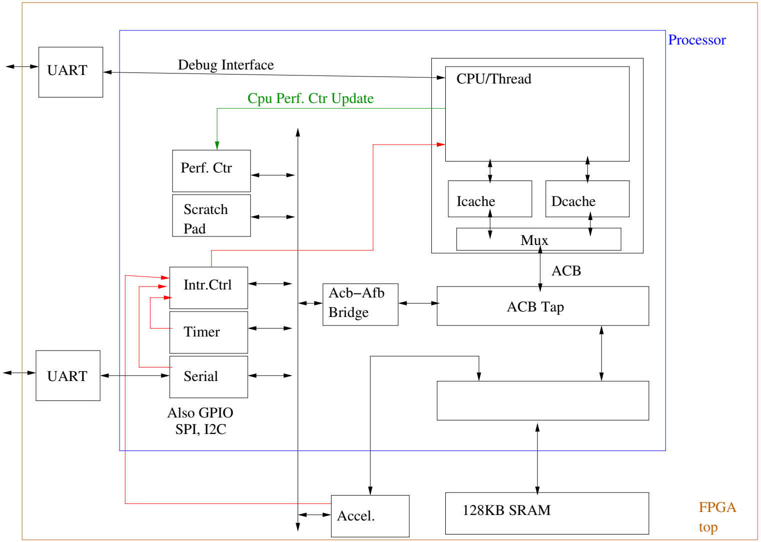
Processor VHDL code hierarchy

```
processor_1x1x32_lite
  mcore_1x1x32_lite
    core_1x32_no_fp_no_mmu
      munit_no_mmu
        dcache
        icache
        dummy_mmu
        mmu_mux
      cpu_no_fp
        teu_no_fp
        stream_corrector
```

Processor VHDL code hierarchy

```
        iunit
          iu_exec
          iu_registers
          iu_writeback_in_mux
        ifetch
        iretire
        loadstore
        idecode_idispatch
        dummy_fpunit
        ccu
        debug_interface
    peripherals
```

The top-level



The connections

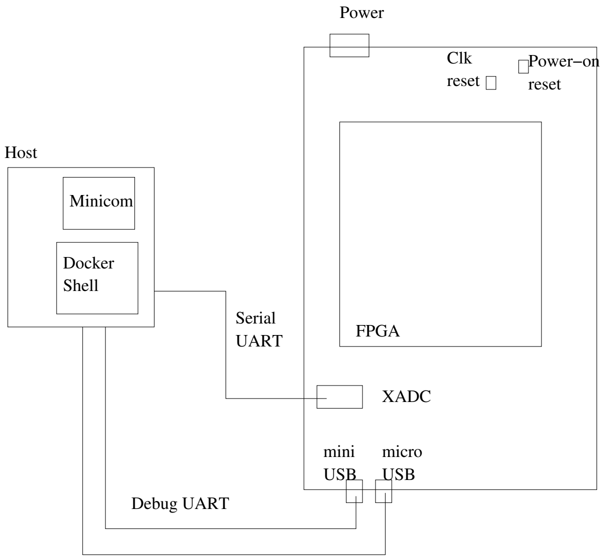


Figure: Connections

The connections: confirmation

- Programming the FPGA (note the USB tty, suppose it is /dev/ttyUSBP).
- Serial cable to FPGA (note the USB tty, suppose it is /dev/ttyUSBQ).
- USB-UART (note the USB tty, suppose it is /dev/ttyUSBR).

The connections: summary

- For FPGA programming, 1x USB (USB micro connector on FPGA board).
- For debugging UART, 1x USB (USB mini connector on FPGA board).
- For serial UART, 1x USB-UART (preferably 1.8V/3.3V).
- On host computer
  - One docker shell, in which we will run ajit_debug_monitor_mt application.
  - One minicom terminal, set to 115200 baud.

First test program

```
.section .text.ajitstart
_start:
mov 0x1, %g1
mov 0x2, %g2
add %g1, %g2, %g3
ta 0
```

Compile this code to generate a memory map file.

Program the FPGA

Start VIVADO and program using the bit-file

`processor_1x1x32_lite.with_accelerator.kc705.bit`

Start the AJIT debug monitor

```
calibrateUart /dev/ttyUSBQ
ajit_debug_monitor_mt /dev/ttyUSBQ
```

This brings up a prompt.

```
[0:0]>
```

Check if debug monitor is connected

At the AJIT debug monitor prompt, type

```
[0:0]> r mode
```

It should return 0.

The AJIT debug monitor gives you access to all internal state

- Read/write from any integer register.
- Read/write from any state register (PSR, WIM, TBR, Y, ASR).
- Read/write from any memory location, using any ASI.
- The AJIT debug monitor also hooks up to a remote GDB client and provides real-time debug capability.

Processor start sequence

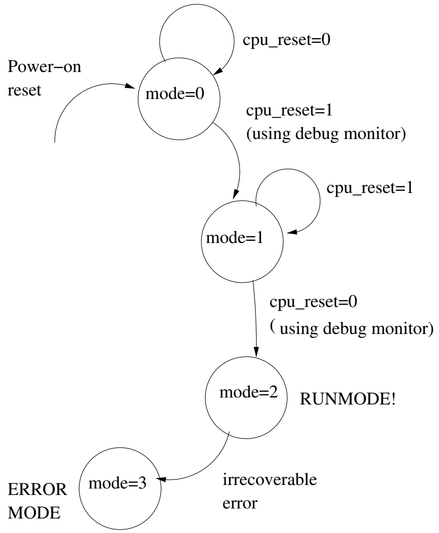


Figure: Start sequence

Run the debug monitor script

```
[0:0]> s run.script
```

The script makes CPU reset "1", downloads the memory map, and then sets the CPU reset to "0", putting the CPU into RUN mode (the LED's indicate this mode transition from 0 to 1 to 2 and then eventually 3.
The CPU mode LEDs then becomes 0x3, indicating that the program has completed and the processor is halted in error mode.
Read back register 3, ensure it holds value 0x3.

```
[0:0]> r iureg 3
```

Should return 3.

Other post run observations

- The processor is in mode 0x3 (error-mode).
  - By default the processor is started with traps disabled. Instruction at PC=0xc generates a trap with traps disabled. The processor goes into error state and halts.
  - Registers g1, g2 hold values 0x1 and 0x2 respectively.

    ```
    [0:0]> r iureg 1
    [0:0]> r iureg 2
    ```
  - The PSR will indicate that the processor is in window 7. This is the default window when the processor starts off, and it stays there.

Second test program: also a small assembly program

- A GCD calculation program.
- The example illustrates branches and delay slots.
- Go through the whole sequence (given earlier).
- Check that register 2 holds the final value, which is 0x1.
- Processor halts in mode 0x3.
- PSR window is 0x7.

Moving on: A bare-metal C program, the hello_world benchmark

- We will print "Hello, world".
- First we use an assembly file to set up the run-time that the C program requires
  - Set up the stack.
  - Set up the WIM so that we use 8 windows.
  - Set up the PSR, enable traps.
  - Jump to the main program.
  - Trap on return.
- We will have to supply trap handlers!
- Compile and link.

Setting up the run-time

```
.section .text.ajitstart
.global _start;
_start: ! we reach here in window 7, post reset.
set 0x20000, %fp  ! Stack
sub %fp, 64, %sp  ! High Mem

set 0x1, %l0      ! window 0 invalid
wr %l0, 0x0, %wim

set 0x10E7, %l0   ! enable traps.
wr %l0, %psr

set trap_table_base, %l0
wr %l0, 0x0, %tbr ! TBR

call main
nop
ta 0    ! return here, goto error.
```

Moving on: compiling and linking

```
MAIN=hello_world
AAR=$AJIT_ACCESS_ROUTINES_MT
PT=$AJIT_MINIMAL_PRINTF_TIMER
INCLUDES="-I ../ -I ../include/ \
         "-I $AAR/include -I $PT/include "
SRCS=" -C ./ -C $AAR/src -C $PT/src -s \
         "  ./init.s -s ./trap_handlers.s"
DEFINES=" -D CLK_FREQUENCY=80000000 "

#Step 1: Generate the Linker Script
makeLinkerScript.py -t 0x0 -d 0x10000\
          -o customLinkerScript.lnk

#Step 2: Compile the application using uclibc
compileToSparcUclibc.py -o 3 -U -N ${MAIN} \
        $INCLUDES $SRCS $DEFINES \
        -L customLinkerScript.lnk
```

Run it on hardware

- But first, we need to start a terminal program (e.g. minicom) for program I/O.

  ```
  minicom -s
  ```

- Setup minicom for 115200 baud, connect to /dev/ttyUSBR.
- In the Docker shell, run ajit_debug_monitor_t as before.
  - Power-on reset, calibrate UART as before.
  - Put CPU into reset.
  - Download memory map.
  - Move CPU out of reset.
- You should see the print on the minicom.

Developing applications using CORTOS2

- Describe the system configuration
  - Hardware (ISA, number of cores, number of threads/core, MMU, FPU).
  - Memory regions (Flash, RAM, Non-cacheable RAM, IO).
  - Build options (stack size, debug, compiler flags, defines etc.).
  - Dynamic memory size.
  - Interrupt handlers.
  - Software trap handlers.
- CORTOS2 takes care of lower level details.
- Program can use CORTOS2 routines for managing logging, printing, dynamic memmory management, locks, message queues.
- Running the application on the FPGA follows same pattern as above.

CORTOS2 examples

- Hello world.
- Display processor configuration, performance counters.
- Illustrate use of GDB.
- Serial interrupts.
- Timer interrupts.
- Software traps.
- Accelerator.

Go through the examples one by one

For each example we review:

- The configuration.
- The program.
- The build process.
- Using the AJIT debug monitor to run the program.
- For one of the examples, illustrate the use of the debugger.