Using the AJIT processor: a short course

Madhav Desai
Department of Electrical Engineering
IIT Bombay

October 27, 2023

AJIT core family

- The AJIT family of processors consists of various implementations of the SPARC-V8 instruction set architecture.
- The simplest implementation is a single-core-single-thread processor and the most complex (thus far) is a quad-core-eight-thread implementation.
- For the purposes of this course, we will use a single-core single-thread implementation:
  - The processor includes a floating point unit, 16KB Data/Instruction caches, a memory management unit and various peripherals.
- A full tool-chain is available, and hosted on github.
- A cooperative real-time operating system (CORTOS2) has been developed at IIT Bombay and will be used to develop applications.
- Linux is also supported.

Single core single thread AJIT processor core

- IEEE 1754 standard ISA (Sparc V8).
- IEEE floating point unit.
- VIVT ICACHE (32kB) DCACHE (32kB) with 2-cycle latency on hit, MMU with 256 entry TLB and hardware page table walk.
- Single issue, in-order, seven stage one instruction-per-cycle instruction pipeline, with branch-predictor.
- C compiler, debugger, Linux.

Downloading and installing the tool-chain

The tool chain can be downloaded from github using

```
git clone https://github.com/adhuliya/ajit-toolchain
```

Now, download the following docker image

```
https://docs.google.com/uc?export=download&id=
                1ck2jSq8LTO-q2RFrwc49mPkXTH9LecVxf
```

The downloaded file will be named

```
ajit_build_dev.tar
```

Downloading and installing the tool-chain

Load the tar-file into Docker:

```
docker load -i ajit_build_dev.tar
```

Remove any containers that may be running

```
docker rm --force ajit_build_dev
```

Navigate to your home directory and start a container with attached bash shell:

```
docker run -u ($id -nu)  -v $(pwd):$(pwd)
           -w $(pwd) -i -t ajit_build_dev  bash
```

This should start bash in the container and give you a shell to use the tool chain.

Potential problems

If the user name is not recognized, go to the /etc/passwords file to find your user-id and use

```
docker run -u user-id  -v $(pwd):$(pwd)
           -w $(pwd) -i -t ajit_build_dev  bash
```

If ajit_build_dev is not recognized, use

```
docker images
```

to find the image-id of ajit_build_dev and use

```
docker run -u user-id  -v $(pwd):$(pwd)
           -w $(pwd) -i -t  image-id  bash
```

Build the tool-chain

Navigate to the ajit-tool-chain directory that you have cloned into.
Run

```
source set_ajit_home
source ajit_env
```

and then

```
./setup.sh
```

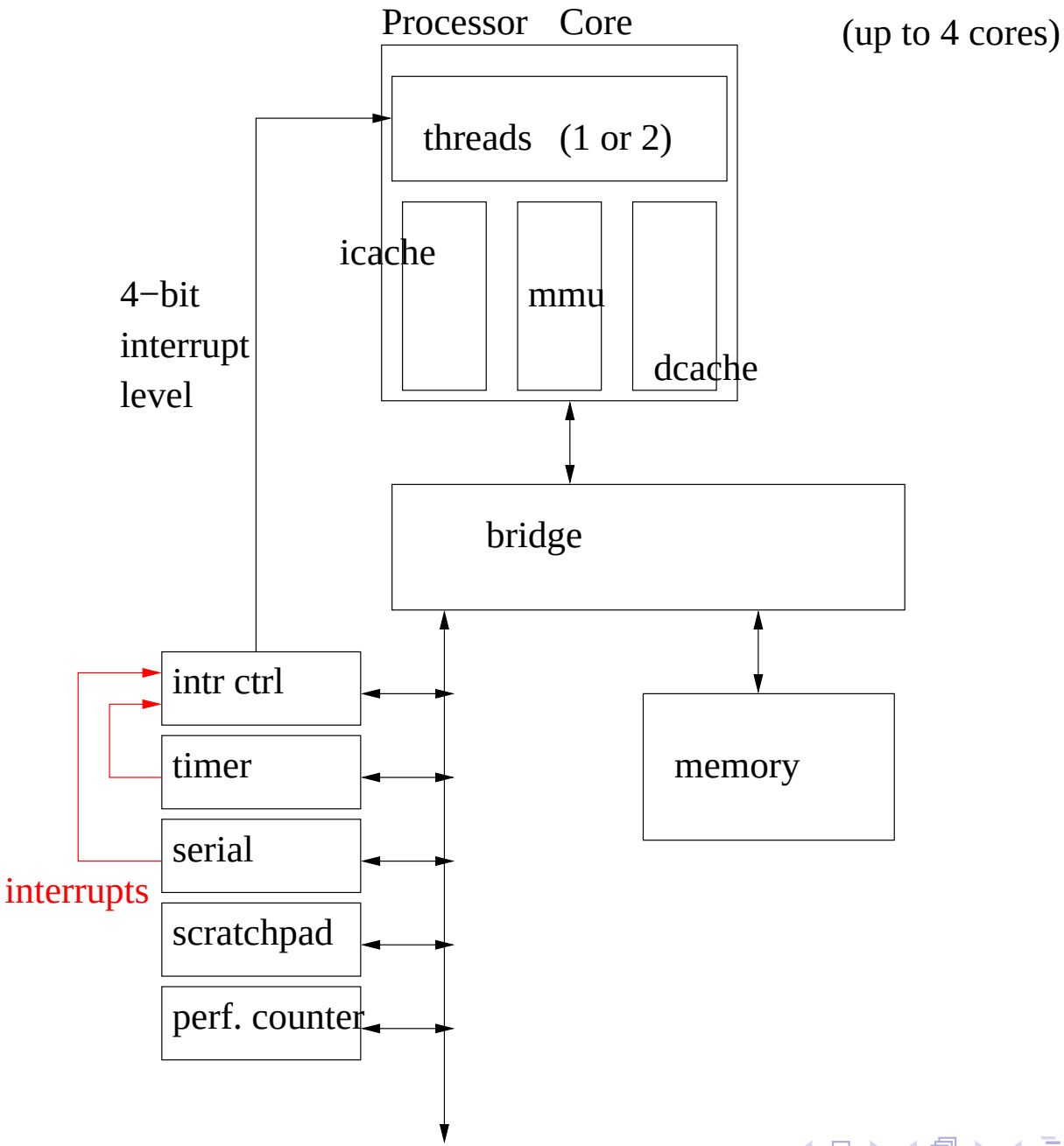
After some time, your tool-chain will be ready.
Documentation on the AJIT processor and environment around it is provided in the docs/processor/ directory.

AJIT C-model

The AJIT C-model implements a software model of a processor platform with

- Up to 4 cores, each with 1 or 2 threads.
- Peripherals: serial devices (2), timer, interrupt controller, scratch-pad, performance counters.
- Main memory.

Platform modeled by AJIT C-model



The AJIT C-model

The following command

```
ajit_C_system_model -m add_test.mmap -i 0x40000000
```

loads memory map file into memory, and executes starting from address 0x40000000.
For all the options, type

```
ajit_C_system_model -h
```

We will describe these options in detail later.

Compiling and running a program

Go to the directory

`validation_ladder/basic_tests/mp_printf`

The directory contains the following:

```
init.s      initialization code to
            set up stack etc.
main.c      the C routine
BIGMEM/VMAP.TXT
            a virtual to physical
            mapping of pages.
BIGMEM/compile_for_ajit_uclibc.sh
            compile directory
```

Compiling and running a program: init.s

```
.section .text.ajitstart
.global _start;
_start:
set 0xfff01ff8, %sp ! stack pointer
        set 0xfff01f00, %fp ! frame pointer
... set up wim, tbr registers.....
        ... set up virtual->physical mapping...
set 0x10E7,%l0
wr %l0,%psr    // psr.

set 0x1,%o0
sta %o0,[%g0] 0x4 // enable mmu

call main // jump to main
nop
ta 0 // halt on return.
```

Compiling and running a program: main.c

- The main program does a series of prints. Note the included directories!
- The VMAP.TXT file defines a virtual to physical mapping. For example, the line

  `0x0 0x40000000 0x40000000 0x2 0x1 0x3`

  means, "for context 0, map virtual page at address 0x40000000 to physical page at address 0x40000000 (page size is 256KB), and the page is cacheable, with access permissions 0x3 (supervisor and user can read/write execute)". For more documentation, read the processor description in AJIT processor description document.
- The compile script uses a Python script, described next.

Compilation script

```
... some stuff omitted ...
TEXTBASE=0x40000000  # location of
                     # first instruction to be executed
DATABASE=0x40040000  # location of data
VMAP=VMAP.TXT
CLKFREQ=80000000
#1 generate linker script
makeLinkerScript.py -t $TEXTBASE -d $DATABASE -o customLink
#2 compile the application (run with -h to get options).
compileToSparcUclibc.py .... lots of options ....
                     (use -h flag to see all options)
```

Compile and run

```
./compile_for_ajit_uclibc.sh

ajit_C_system_model \
            -m printf_test.mmap.remapped  \
            -i 0x40000000
```

Summary

- Download and installation of AJIT tool chain.
- Description of AJIT C model.
- Compiling and running a simple program.