# The AJIT tool chain

Anshuman Dhuliya, Gauri Patrikar, Vishnu Easwaran
Madhav Desai
Department of Electrical Engg.
IIT Powai
Mumbai 400076

July 26, 2021

# Overview

- Hosted at github.

  `git@github.com/adhuliya/ajit-toolchain`
- Based on buildroot 2014.08.
  - Compilers, binary utils, uclibc.
  - Linux 3.16 port.
- AJIT multi-core processor system simulator.
- Utilities for building memory maps, flash images, managing (static) virtual memory maps.
- C libraries for processor monitoring, observation, minimal print routines, timer routines, basic trap handlers, generic interrupt service routine.
- Verification tests and examples.
- FPGA bitfiles (for VC709, KC705 FPGA cards).

# Installation

- You will either need to use ubuntu 16.04, or install Docker.
- Clone and check out "marshal" branch.
- Follow the instructions in the README.md file.

```
source set_ajit_home
./setup.sh
source ajit_env
```

  - The basic setup gives you the 32-bit ISA tools.
  - For 64-bit ISA tools, go to the buildroot_src_64 directory and follow the README.
- At this point you are ready to go in two possible ways.
  - Using Docker.
  - Work directly on the native system.
- Will be doing a walk through later.

# AJIT multi-core processor system

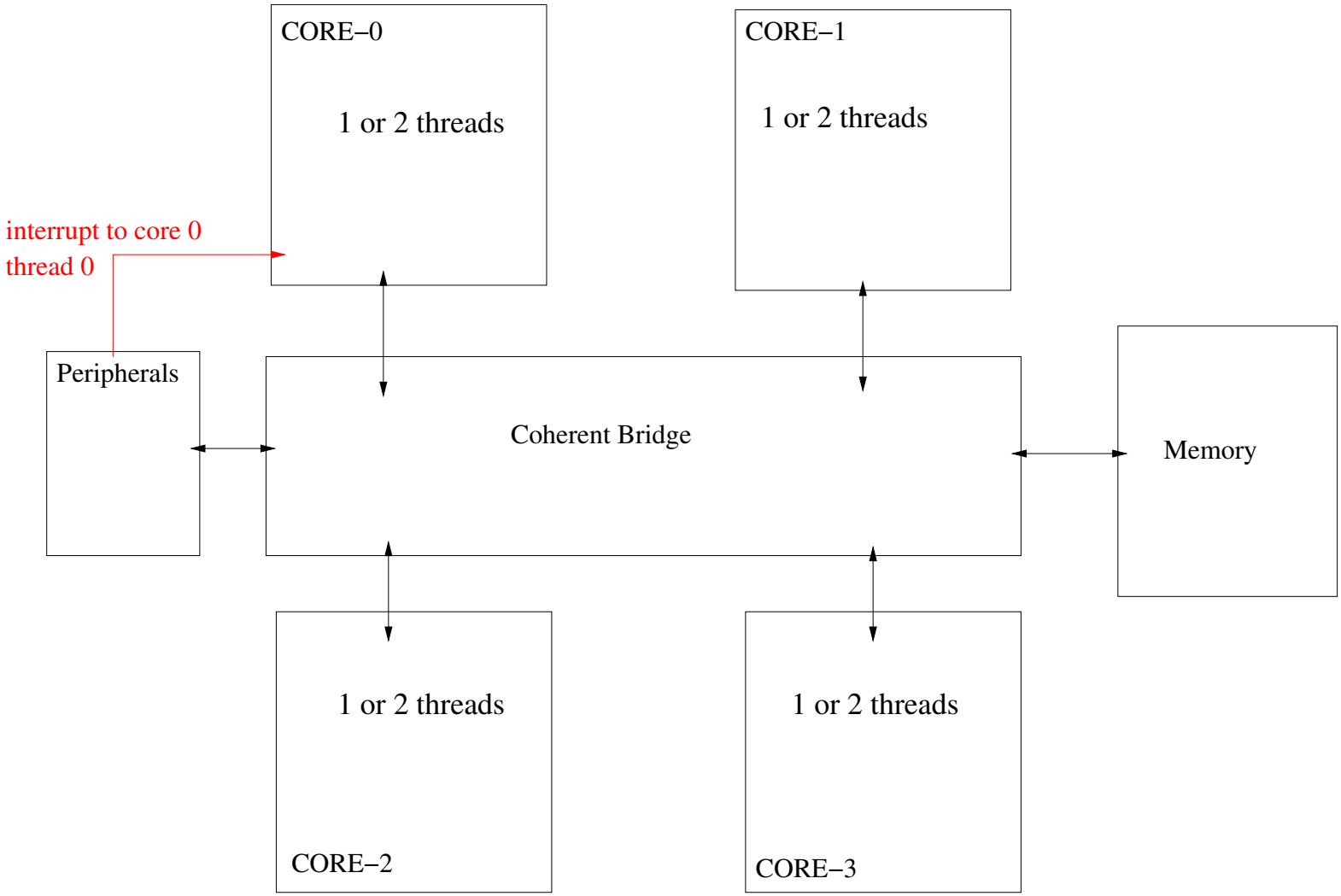
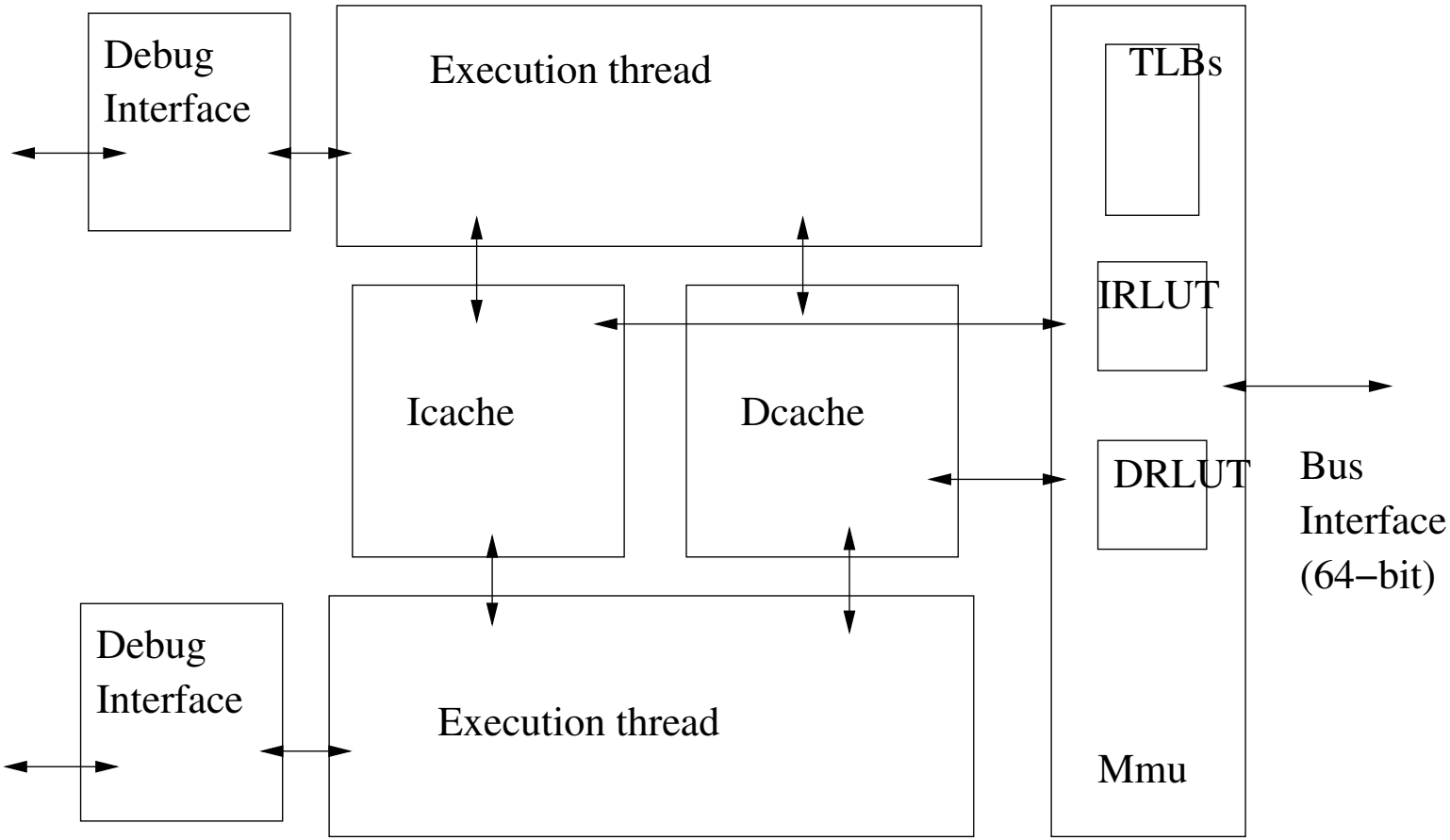



Figure: AJIT Multi-core Processor System

# AJIT core




CORE can be configured with one or two threads.

Figure: AJIT Core with two threads

# Core and thread identification etc.

- In each thread, a read of asr29 gives you the following word

| 31:16 | 15:8 | 7:0 |
|---|---|---|
| 0x5052 | core-id | thread-id |

  For instance, if you are in core 2, thread 1, then the read from asr29 will return 0x50520201.
- Note: two threads in the same core will share the caches and the MMU. This should be exploited by software.
- Note: two threads in two different cores use distinct caches and MMU. In principle they can be executing distinct processes with distinct virtual to physical address mappings.

# System simulator

```
ajit_C_system_model    ...options...

Important options:
     -n <number-of-cores>
     -t <number of threads per core>
     -u <32/64>   (ISA version: default is 32)
     -m <memory-map-file>  (Initial contents of RAM)

     [-w  <execution_trace_log_file>]
     [-d]  for post-condition checks
     [-r]  results file.
     [-l]  post-condition check log.

 ... lots more ....
```

# Writing an application for bare-metal

- Initialize the run time environment
  - Set up stack for each thread.
  - Set up virtual-¿physical mapping for each thread.
  - Initialize processor state register, trap base register.
- User program: as usual, but can use the following resources

  `AjitPublicResources/tools/ajit_access_routines`
  `AjitPublicResources/tools/minimal_printf_timer`
- Run compile script. This will generate and ELF file, a memory map file and an object dump file.
- Run the program on the system simulator.

# Walk through

- Clone, and build.
- Examine a test application.
  - Initialization.
  - Compilation.
  - Run on the system model.
  - Examine results.