RCE Programmer's Training

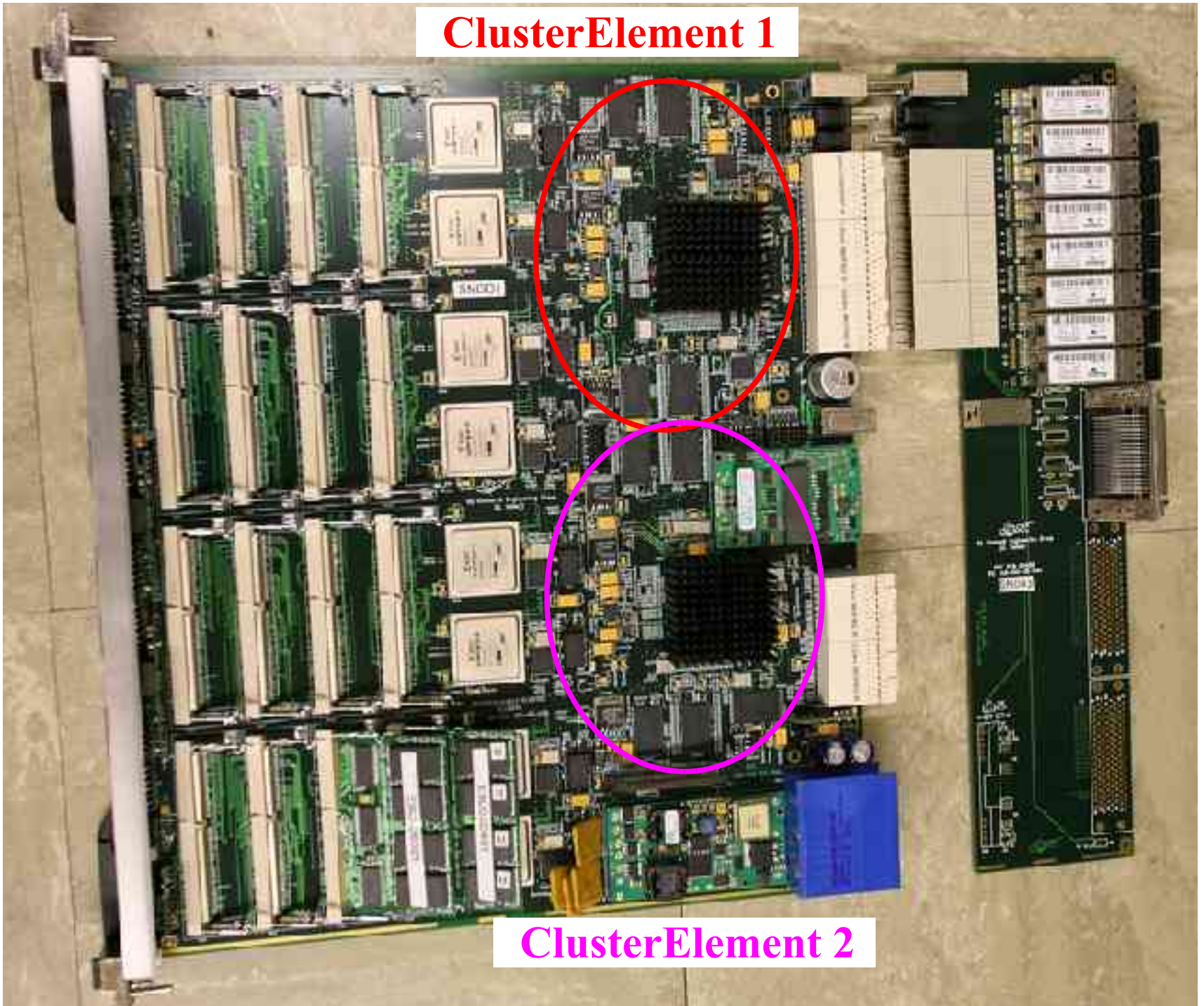
Informal walk-through of software environment and tools for programming the Reconfigurable Cluster Element.

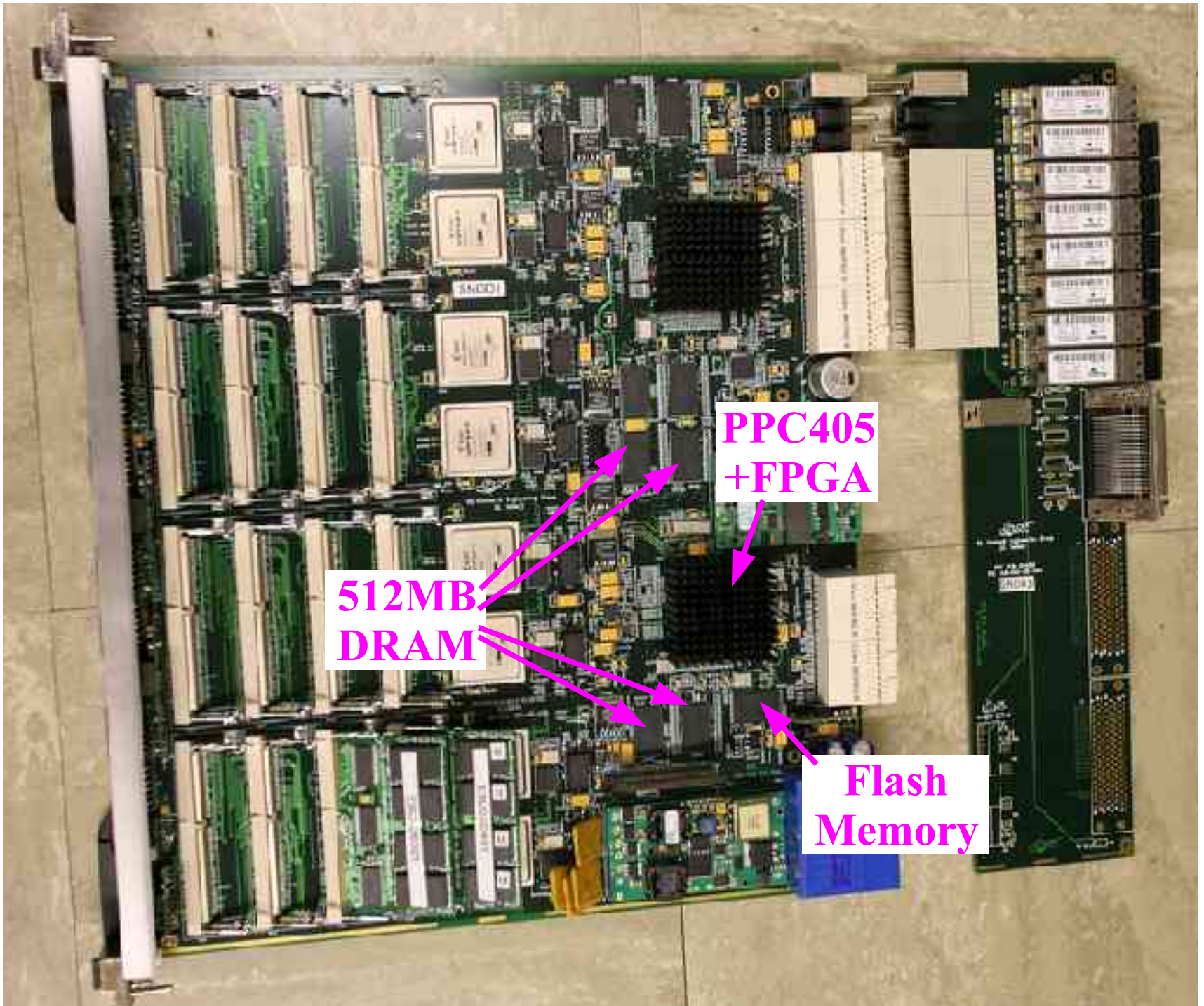
Material covered can be found on
Confluence → Cluster Computing Initiative
(http://confluence/display/CCI/CCI+Home)

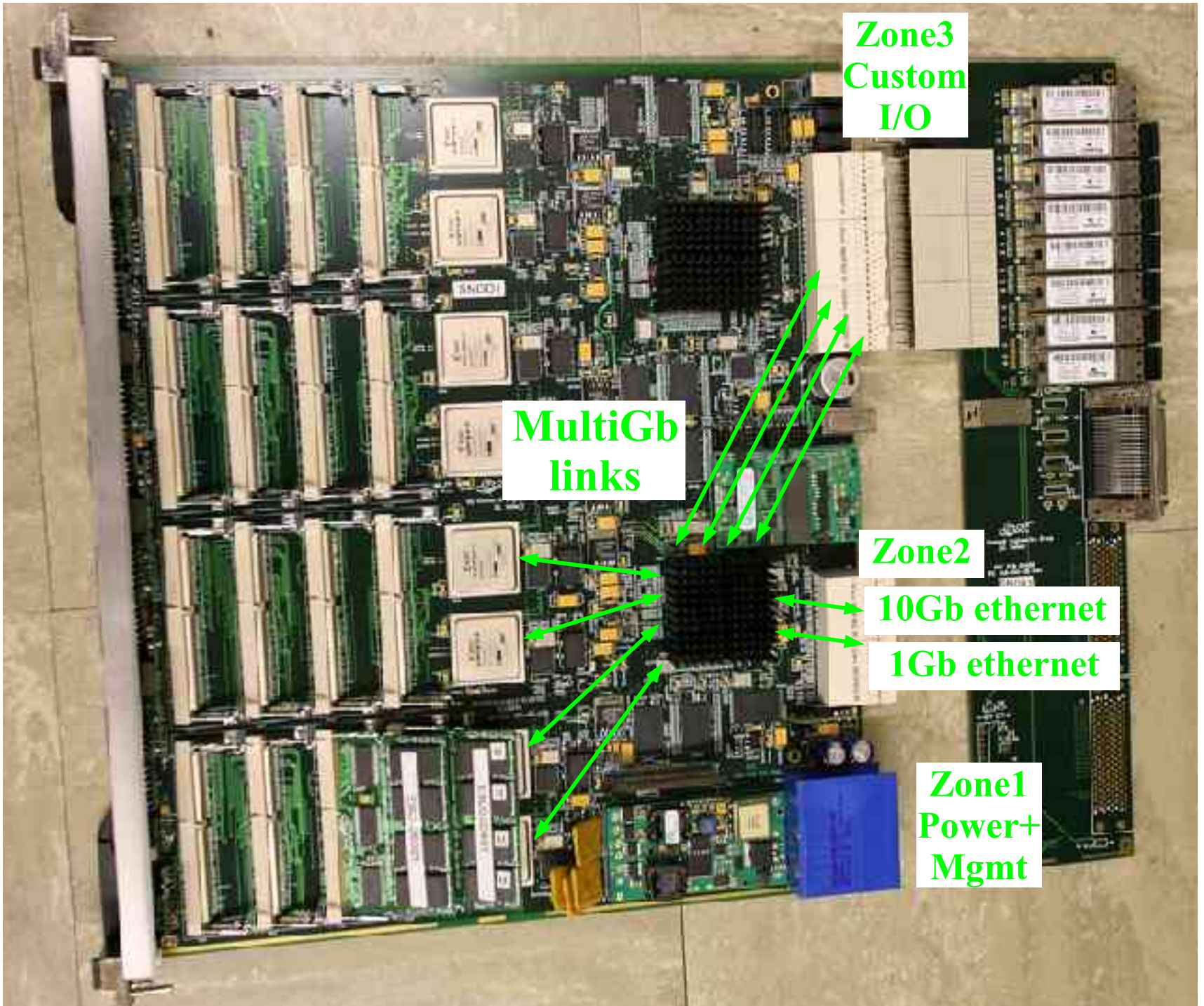
Plan

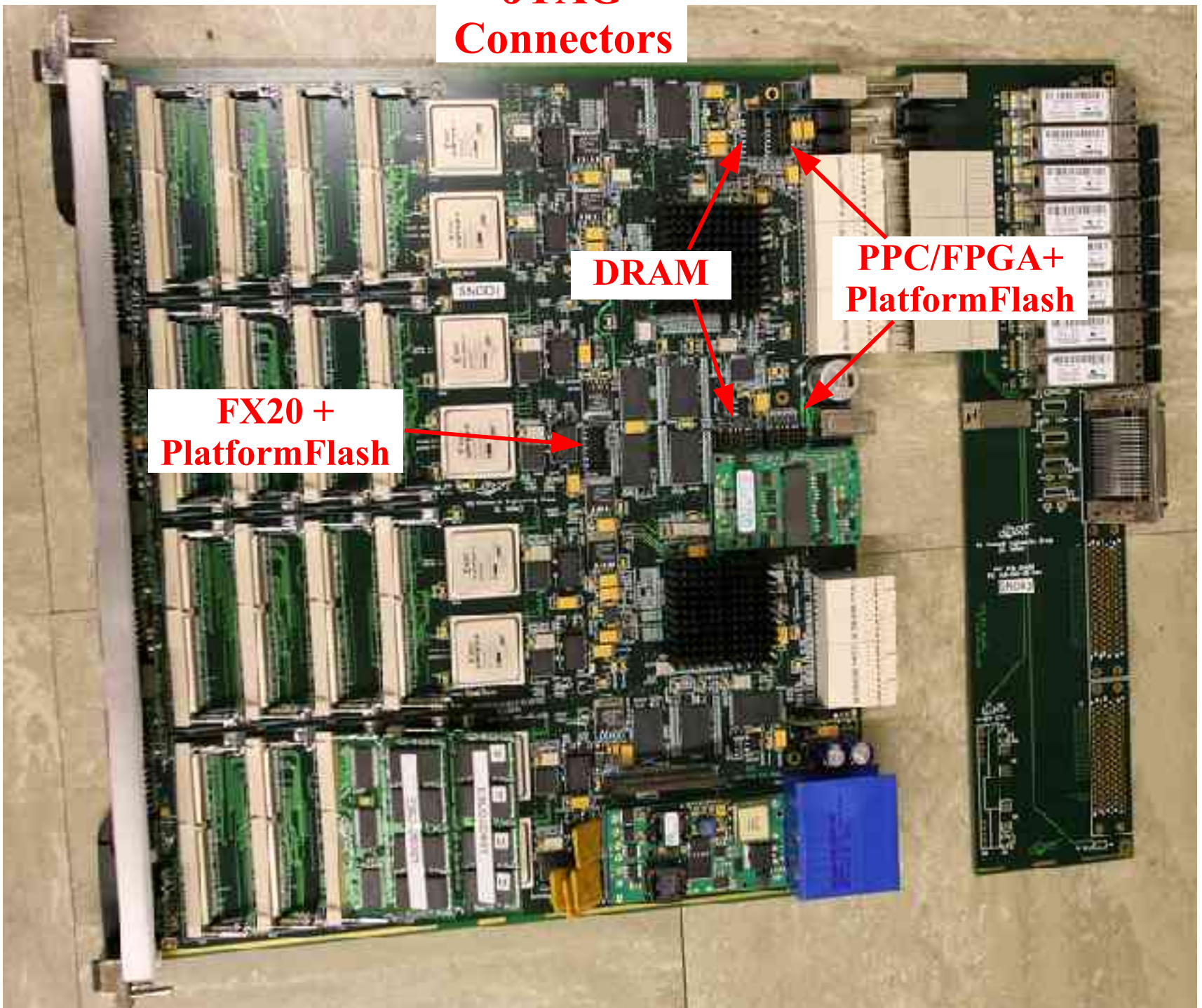
- Quick look at the RCE
- Discuss building applications
- Discuss debugging applications
- Try some examples

RCE Board and RTM


ClusterElement 1
ClusterElement 2


PPC405
+FPGA
512MB
DRAM
Flash
Memory


Zone3
Custom
I/O
MultiGb
links
Zone2
10Gb ethernet
1Gb ethernet
Zone1
Power+
Mgmt


JTAG
Connectors
DRAM
PPC/FPGA+
PlatformFlash
FX20 +
PlatformFlash

RCE Software Environment

- RTEMS (4.7.1) on PowerPC405
  - POSIX API
  - BSD network stack with some additions
- cross-compiler via gcc
- makefile system and project/package release structure
- Xilinx JTAG debugger

Building Applications

- checkout build environment (“release”)
- checkout RCE support projects
- compile your code against these projects
- upload the executable to the RCE flash memory
- boot RCE target

Building Applications

- checkout build environment (“release”)
  - *source /afs/slac/g/npa/setup/npa.csh*
    - defines build/debug environment
  - *cvs co -d <mydir> release*
  - *cd <mydir>*

Building Applications

- checkout RCE support projects
  - *cvs co rce*
  - *cvs co rceusr*
  - *cvs co rceapp*

Building Applications

- create your own project/package
  - *make/tools/pkgcreate.py --project myapps --package example*
  - (edit code, include targets/libs in constituents.mk)
- or checkout from cvs
  - *cvs co myapps*
- and compile
  - *gmake ppc-rtems-rce405*
  - *gmake i386-linux*

Building Applications

- upload the executable to the RCE flash memory
  - *build/rceapp/bin/i386-linux/upload_elf_host*
  - extracts the image from the compiled executable and transmits over the network to the RCE; a corresponding RCE thread receives the image and writes it to the indicated block in flash.
  - Alternatively, the image can be loaded directly to RAM over JTAG with the debugging tools and written to flash (via core code contained within that image)

Building Applications

- boot RCE target
  - *build/rceapp/bin/i386-linux/console_host*
    console> reboot
  - or get up and push the front panel button,
  - or reset the processor via the JTAG debugger.
  - Eventually, the RCE will have a daughtercard (IPMI) which allows it to receive resets from the “shelf manager”.

Debugging Applications

- the JTAG debugger
- the multi-function display
- the console

Debugging Applications

- the JTAG debugger
  - xmd (Xilinx Multiprocessor Debugger)
    - read/write memory
    - read/write processor cache/tags
    - read/write PPC and DCR registers
    - set breakpoints and step through instructions

Debugging Applications

- the JTAG debugger
  - *weaver> xmd*
  - *XMD> rst -processor* (resets processor execution)
  - *XMD> dow pgpforward* (downloads an executable)
  - *XMD> con* (continues execution)

Debugging Applications

- the JTAG debugger + gdb
  - full source code debugger
  - *weaver> powerpc-rtems-gdb pgpforward*
  - *(gdb) target remote localhost:1234* (connect to xmd)
  - *(gdb) break init_executive* (set a breakpoint)
  - *(gdb) c* (continue execution)
  - (gdb) ...

Debugging Applications

- the multi-function display
  - a 32-b value can be written to the front panel display

```
void writeLED(unsigned val) {
  asm volatile ("mtdcr 0x2f7,%0" : : "r" (val));
}
```

  - the display can be configured (firmware change) to display any characters (5x5 bitmap) if useful

Debugging Applications

- the “console”
  - host access via the network to a target thread which handles interactive commands
  - involves a host (linux) process and a target (ppc) thread
  - found in *rceusr* project to allow direct reuse or extension

Debugging Applications

- *ConsoleHandler* class instanciated in rce application
  - see rceapp/console.cc for rce application example
- *console_host -h <hostname>* linux executable
  - reboot: Reboot the processor
  - remove <filenum>: Remove the file specified by <filenum>
  - bootcfg <index> <image> <flags>: Sets boot vector <index> to load image <image> and user configuration <flags>
  - bootdir: Dumps flash boot directory contents
  - filedir: Dumps flash file directory contents
  - echo <message>: Enter a time-stamped message in the system log
  - log: Dumps system log
  - clear: Clears system log

Debugging Applications

- *DebugHandler* class instanciated in rce application
- *debug_host -h <host> -f <executable>*(linux executable)
  - getexceptions: print the exceptions (if any) recorded by the RCE since the last reset/power-on; includes CPU registers, stack trace
  - clearexceptions: clear all the recorded exceptions
  - getmessages: print all messages recorded by the RCE since the last reset/power-on
  - clearmessages: clear all the recorded messages
  - getcontext: shows a stack dump of the different threads running in the RCE
  - dumpstats: shows a dump of the RCE network statistics

Examples

- *rceapp/console/console.cc* application
  - (examine constituents.mk – makefile support)
  - (examine rtems_config.cc – per executable definition of rtems resources)
  - (examine *rce/init/src/Init.cc*)
  - initializes ethernet driver and attaches to the network stack
  - instanciates ConsoleHandler, DebugHandler, and UploadManager threads
    - uses *rce/service/Thread* POSIX thread wrapper
    - uses *rce/net/Socket** socket API wrappers
  - (connect debugger and run the executable)

Examples

- *rceapp/pgpforward/pgpforward.cc* application
  - initializes ethernet driver and attaches to the network stack
  - initializes pgp driver
  - starts thread which
    - waits for a datagram over the network
    - forwards the network data over the pgp link
    - forwards the pgp response back over the network
    - increments a counter
  - instanciates ConsoleHandler, DebugHandler, and UploadManager threads
    - uses *rce/service/Thread* POSIX thread wrapper
    - uses *rce/net/Socket** socket API wrappers