

THE WORLD'S FIRST HYBRID-CORE COMPUTER.





PERSONALITY DEVELOPMENT KIT

Topics

- **PDK Overview**
- **PDK Instruction Set Architecture**
- **Hardware Interfaces**
- **Simulation Environment**
- **PDK Tool Flow**
- **Debugging a Custom Personality**
- **Sample Personality**

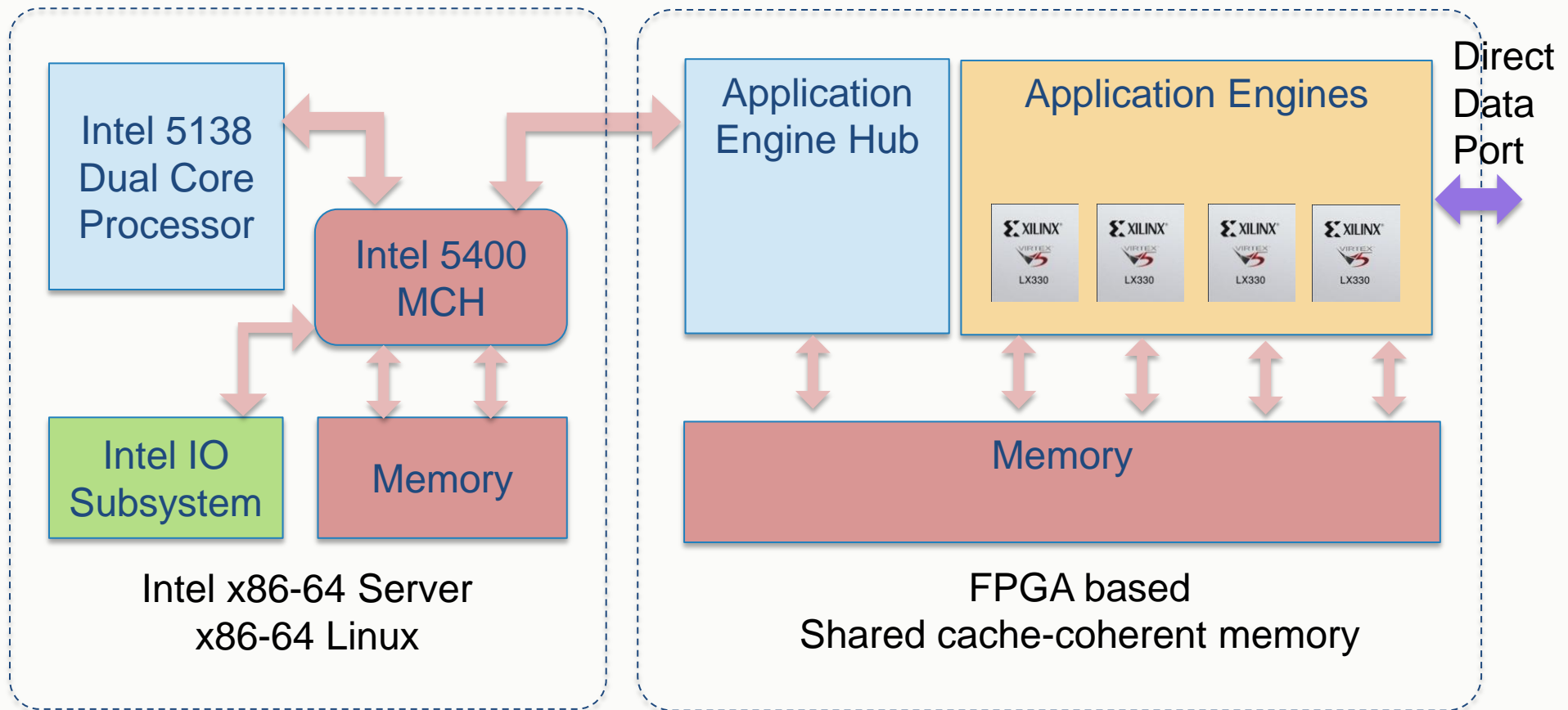


PDK OVERVIEW

HC-1 Architecture

“Commodity” Intel Server

Convey FPGA-based coprocessor

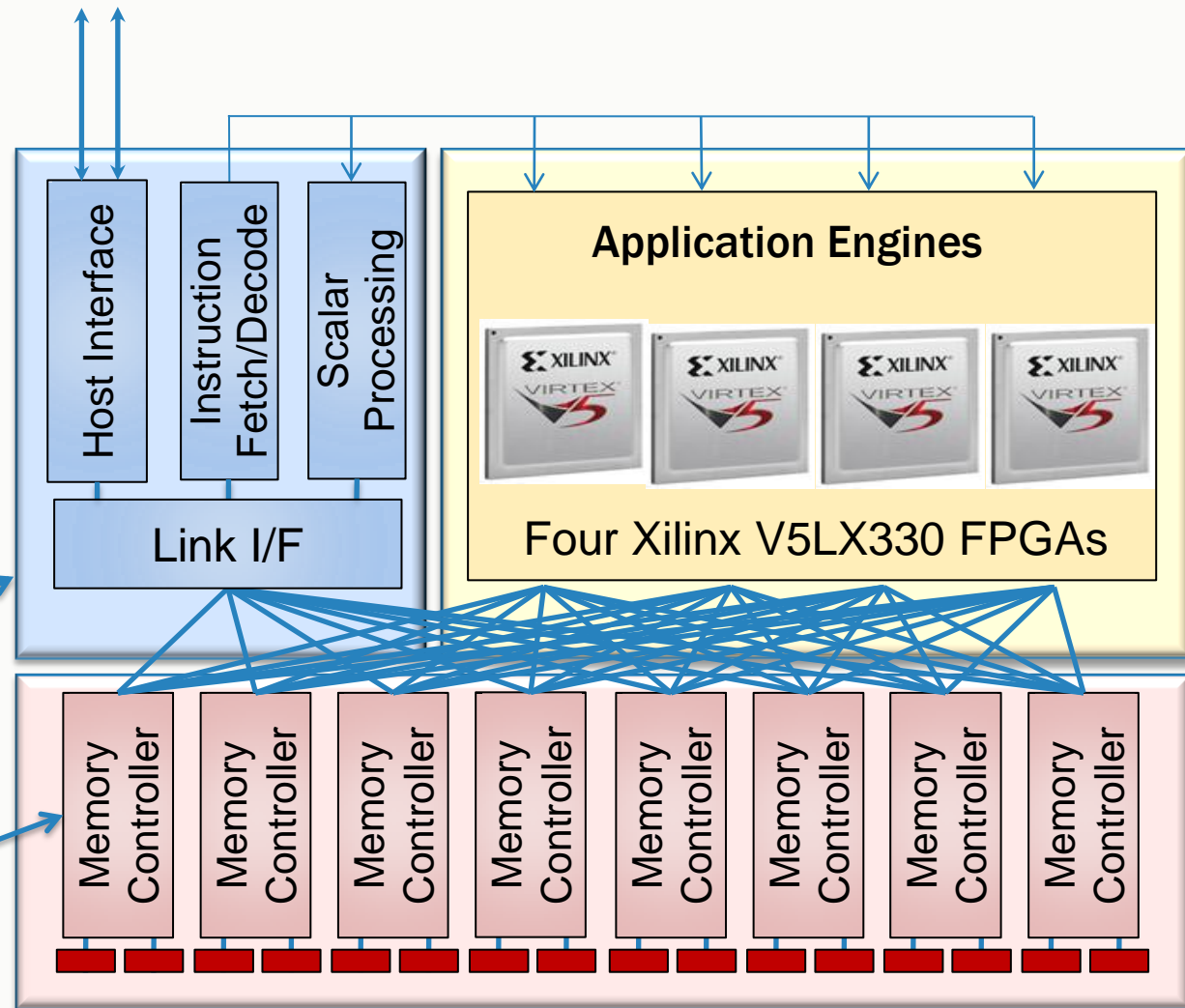


Inside the Coprocessor

*Host interface
and memory
controllers
implemented
by coprocessor
infrastructure*

Implemented
with Xilinx
V5LX110 FPGAs

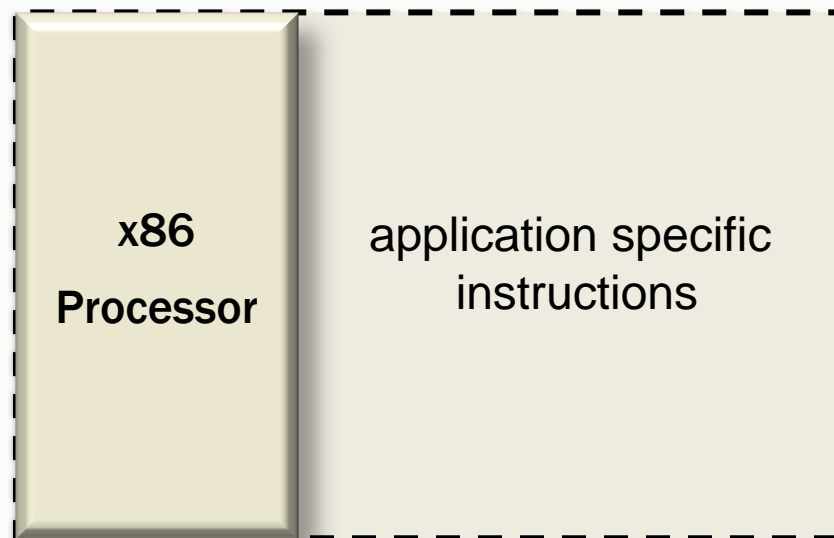
Implemented
with Xilinx
V5LX155 FPGAs



16 DDR2 memory channels
Standard or Scatter-Gather DIMMs
80GB/sec throughput

What Is a Personality?

a “personality” is a reloadable set of instructions that augment an x86



same view of memory as the x86

appear as natural extensions

applicable to a class of applications or specific to a particular code

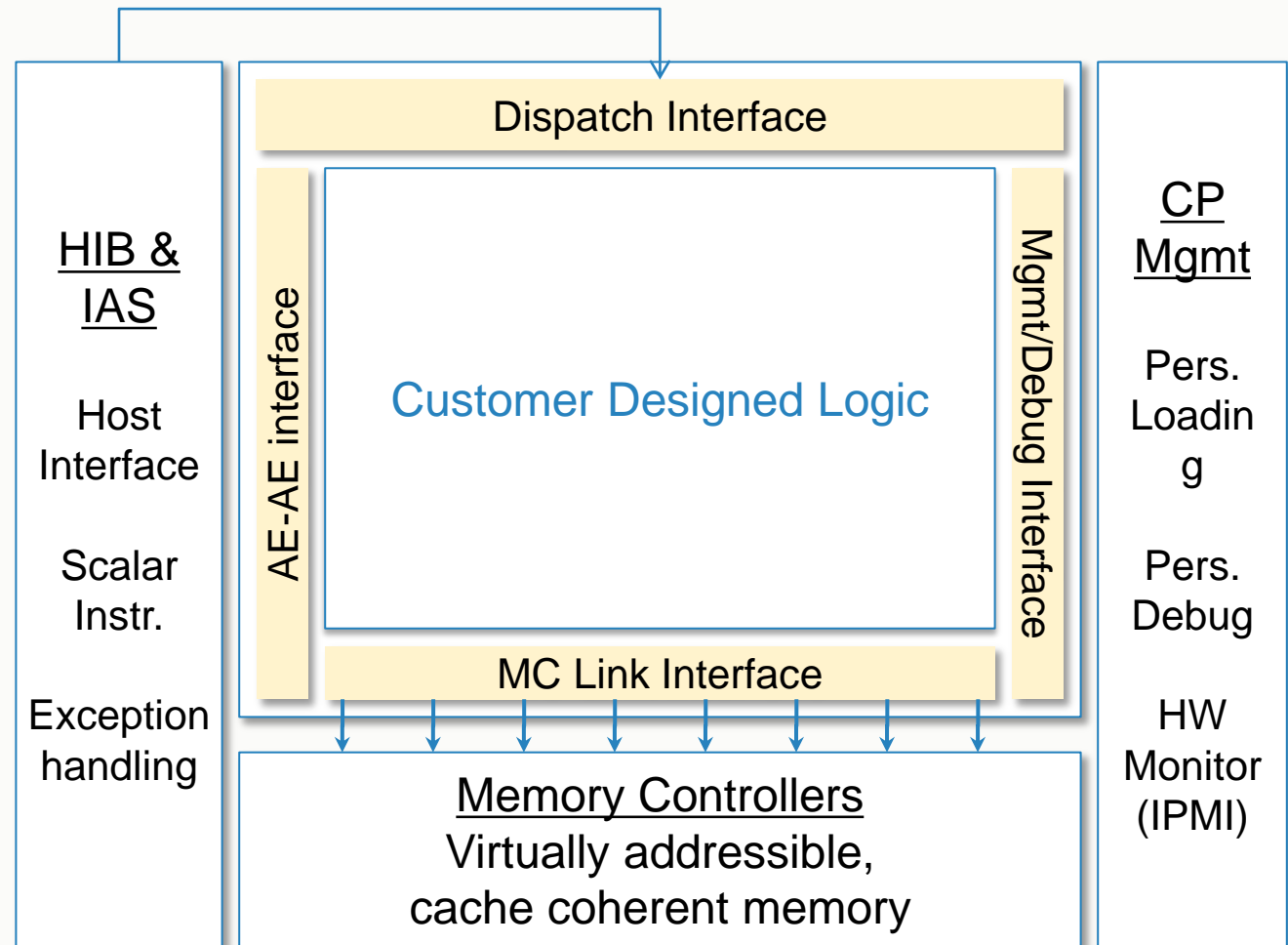
execute on a Convey Coprocessor and take advantage of its highly parallel architecture

Custom Personalities

- **Personality Development Kit**
 - logic libraries implement interfaces to coprocessor infrastructure
 - System simulation environment for debugging
 - Management tools package bit files produced with Xilinx toolset into personalities
- **Architected instruction interface**
 - transfer data to/from AE
 - control custom AE logic

Personality Development Kit (PDK)

- Customer designed logic in Convey infrastructure
- Executes as instructions within an x86-64 address space
- Allows designers to concentrate on Intellectual Property, not housekeeping



Personality Development Kit

- **The PDK is a framework and tools that enable developers to customize the architecture to a particular application or algorithm**
- **Creates great opportunities...**
 - Great potential for acceleration
- **And challenges**
 - Customizing the architecture means FPGA development
 - FPGA development is difficult and time consuming

Personality Development Kit

- **Focus on user productivity**
 - Flexible, configurable infrastructure
 - Simulation and debug tools
 - Support resources
- **Where is the time really spent?**
 - Learning curve
 - RTL coding
 - Verification and debug
 - Timing closure



PDK INSTRUCTION SET ARCHITECTURE

PDK ISA

- **The PDK instruction set architecture allows the user to define both machine state and instructions**
- **Machine state consists of 4 control registers implemented in the dispatch interface, and a user-defined number of AEG registers**
 - AEG = Application Engine General
- **Two kinds of instructions**
 - Move to/from an AE register
 - Custom instructions defined by the user

Convey Coprocessor Instruction Set

Single Precision Vector Personality

Custom PDK Personality

Canonical Instruction Set

- Program Flow Instructions
- Compare & Condition code instructions
- Scalar byte, word, double, quad load/store
- Scalar single, double, integer arithmetic
- Scalar Logical and shift instructions
- Scalar convert
- Scalar move
- Status & control instructions

Custom personality extensions

- Move operand to/from AE
- Custom instructions
- AE status & control instructions

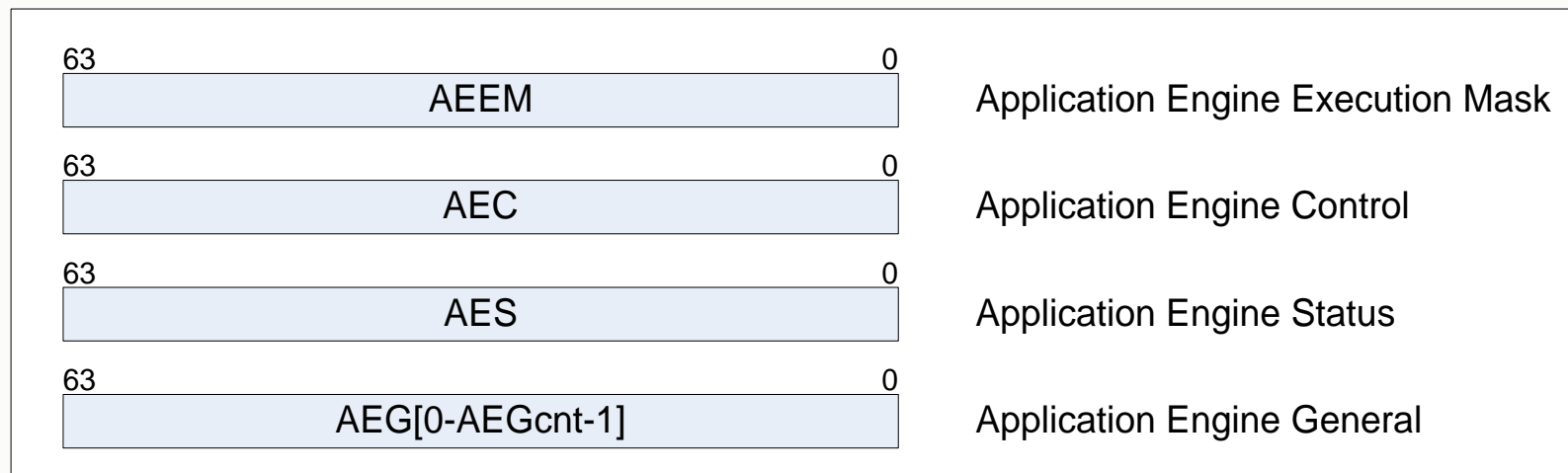
Single Precision Vector Extensions

- Vector single & complex/single arithmetic
- Vector integer arithmetic
- Vector bit & logical operations
- Vector data type conversions
- Vector single, complex/single & integer load/store
- Vector strided & indexed load/store
- Vector partitioning
- Vector under mask, compress, and expand
- Vector reductions (min, max, sum, product)

All personalities include the canonical instruction set, which executes on the scalar processor

PDK Supported Machine State

- **AEEM, AEC, AES and AEGcnt registers implemented in the dispatch interface**
- **AEG registers implemented in custom personality**



Execution Model

- Instructions are dispatched if the AE is enabled in the Application Engine Execution Mask register (AEEM)

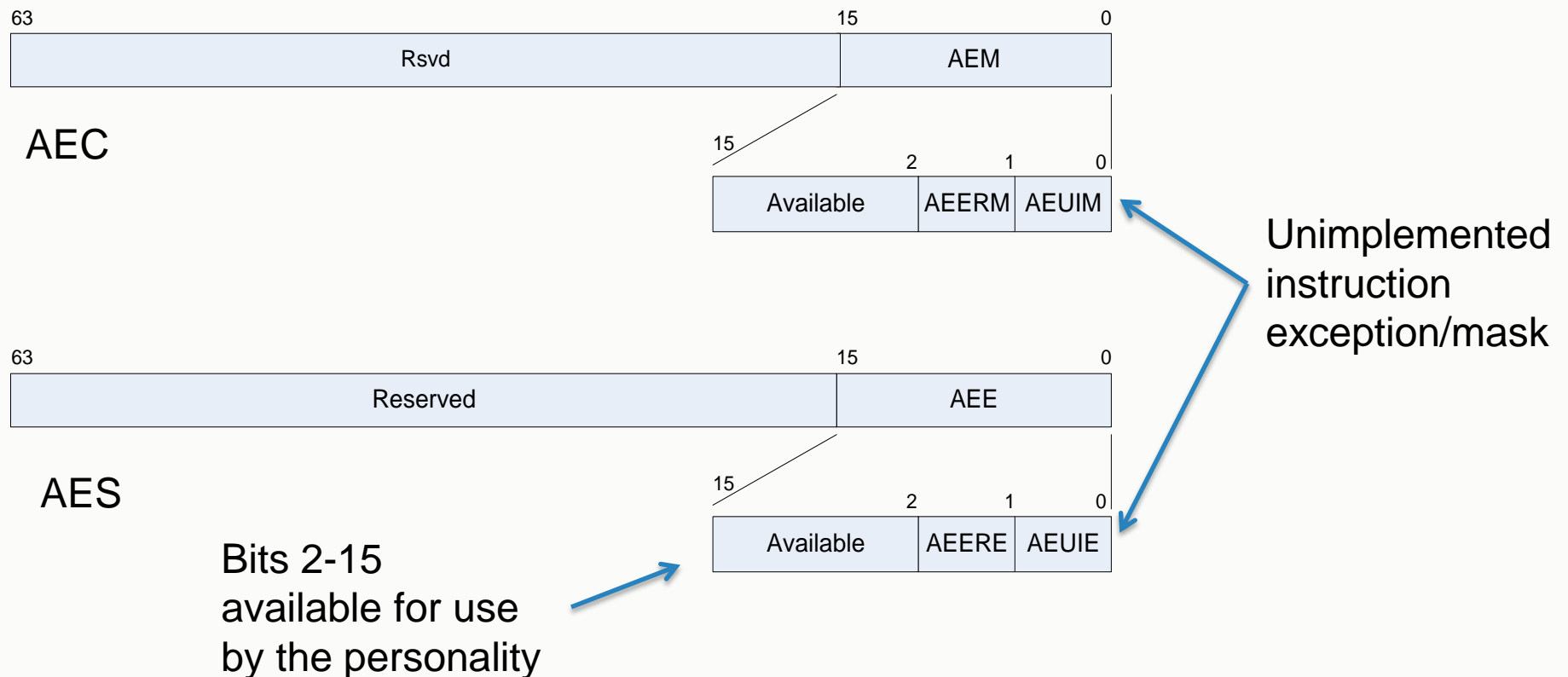


```
mov $0xf, %AEEM    # select AEs 0-3
caep00 $0           # execute custom instruction
```

```
for (aeld = 0; aeld < 4; aeld += 1) {
    if (AEEM.IEEM<aeld>) {
        if ((inst.m == 1) || (aeld == inst.ae))
            <Execute instruction>
    }
}
```


Exceptions

- Exceptions are events that stop the flow of instructions on the coprocessor
- Cause traps if not masked in the AEC register



Application Engine General (AEG)

- **AEGs are 64-bit registers defined by the custom personality**
- **AEGcnt –max AEG index defined by the personality**
- **Used for context save and restore**
 - All PDK personalities use the same context save and restore routines



Custom AE Instructions

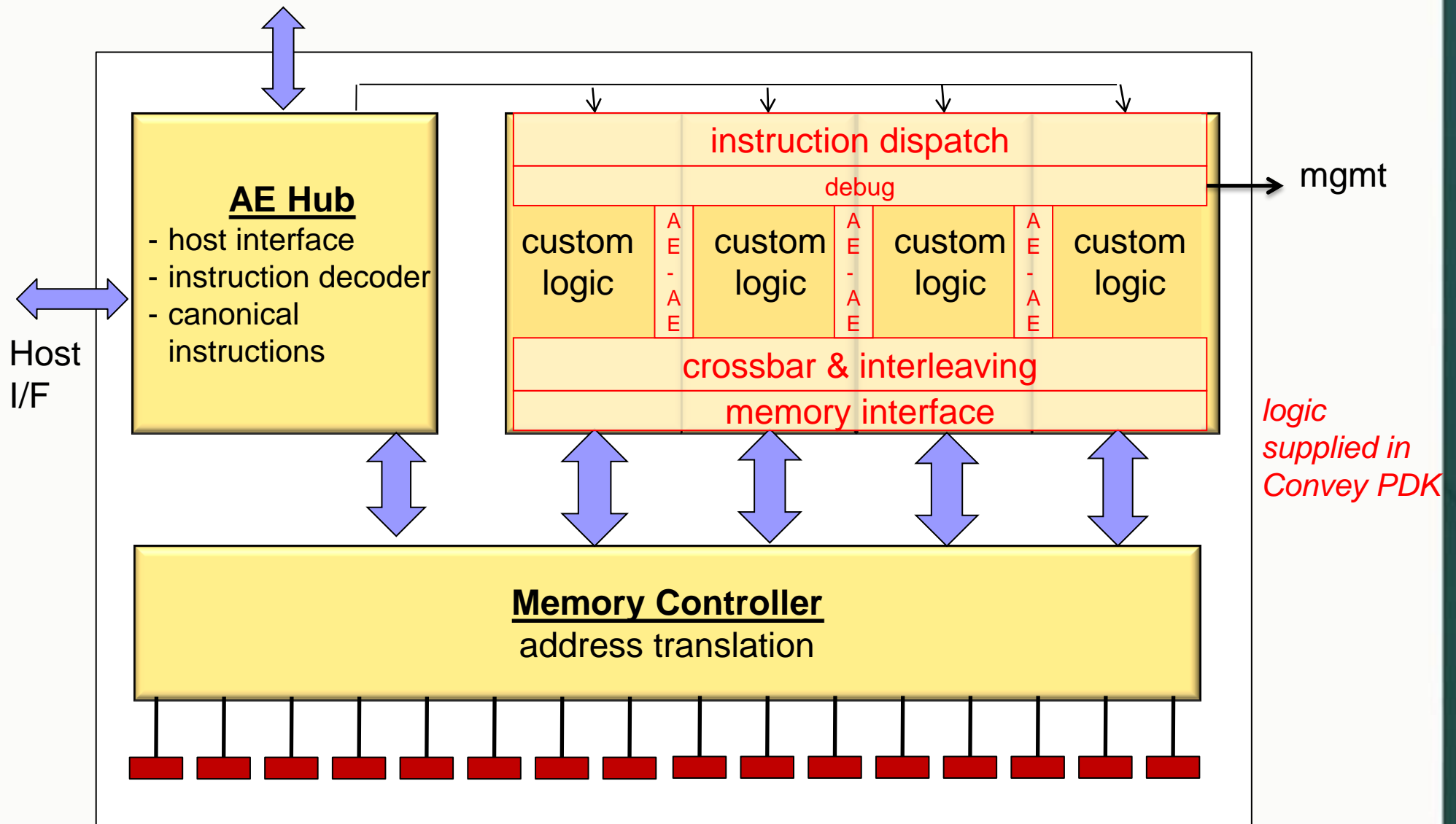
- **Custom instructions are defined by the personality**
 - CAEP00 (opcode 0x20)
 - ...
 - CAEP1F (opcode 0x3f)
- **Two type of custom instructions:**
 - Masked instructions are sent to all AEs
 - Directed instructions are sent if the 'ae' field of the instruction matches the AE index (and AE is enabled in the AEEM register)



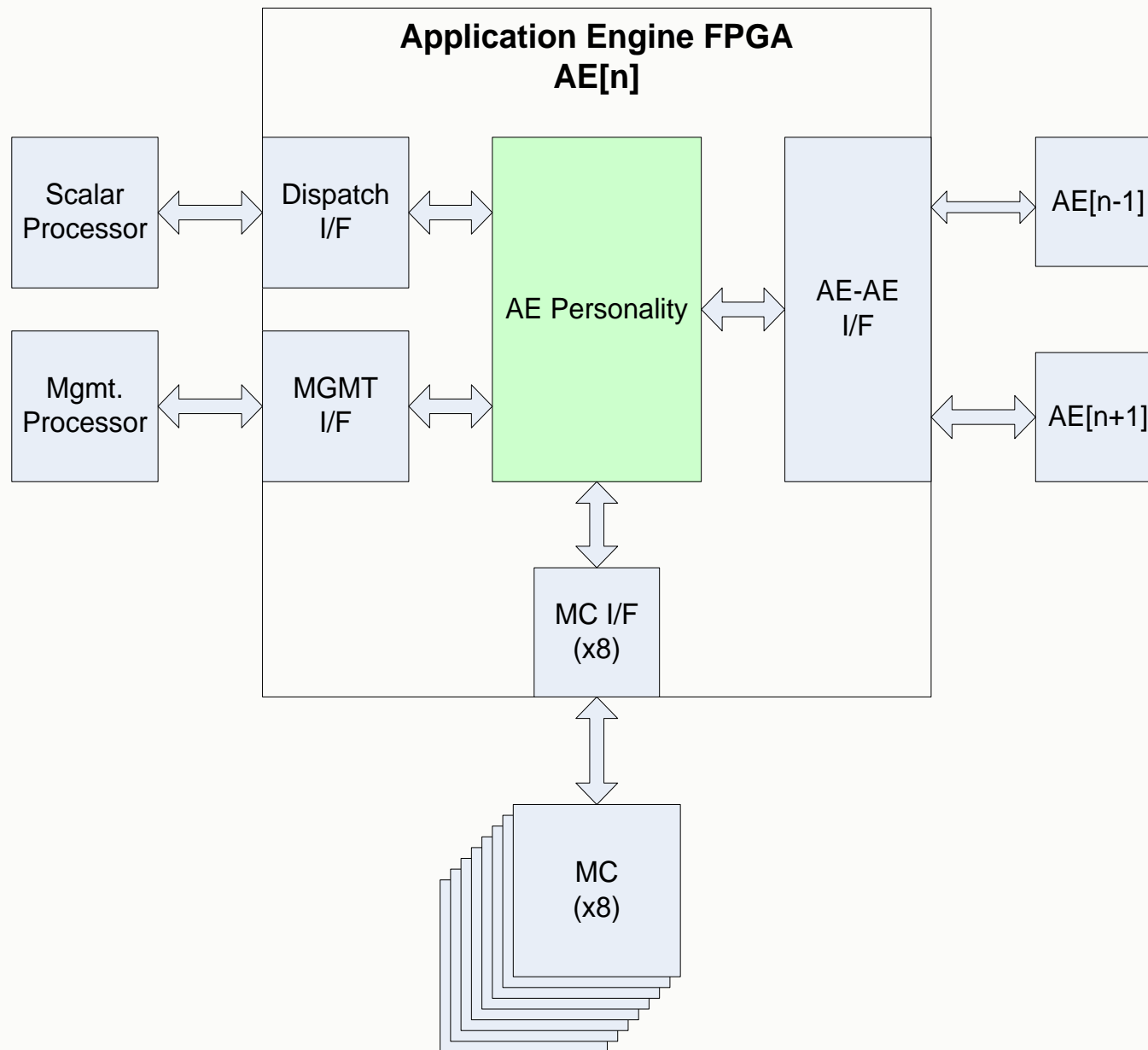


HARDWARE INTERFACES

Coprocessor Diagram

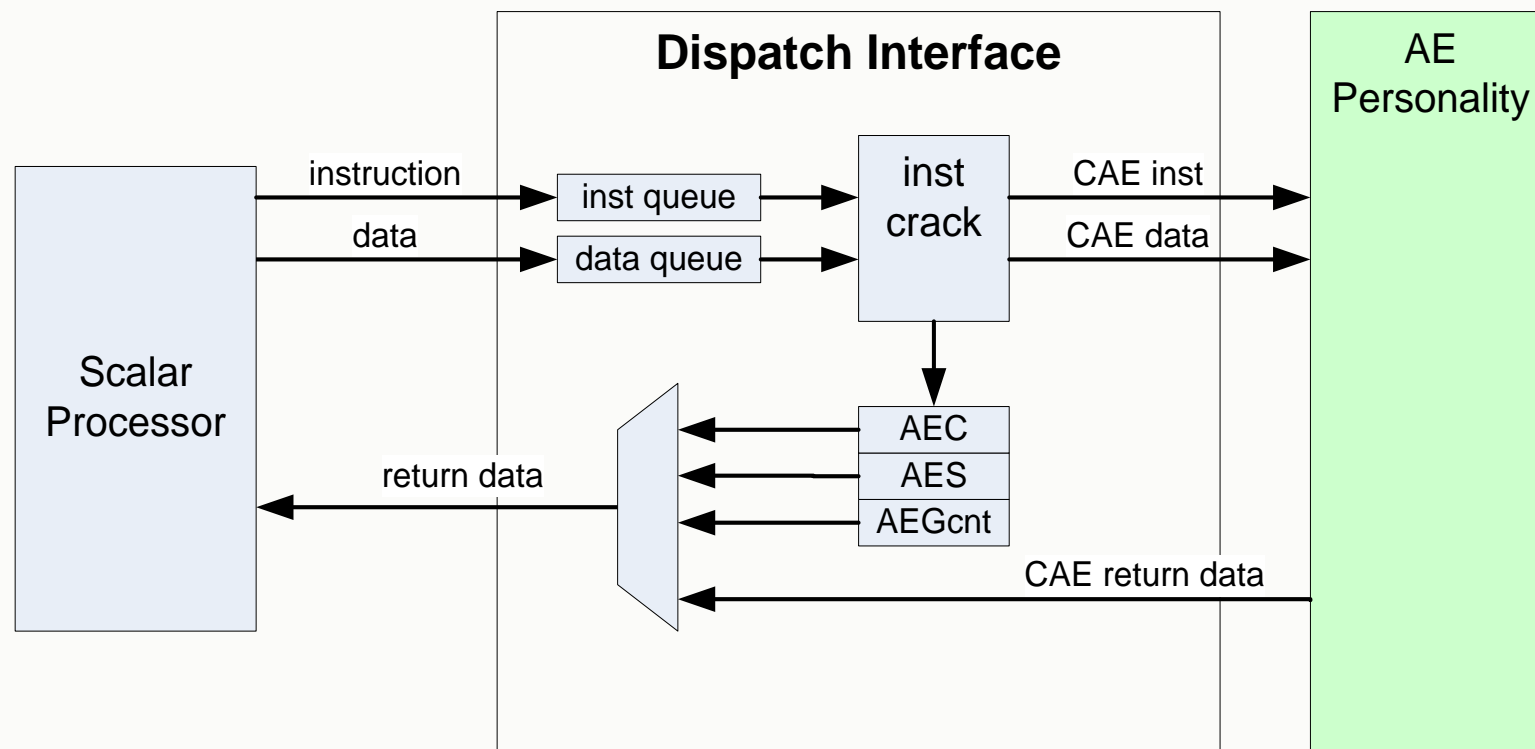


AE FPGA Hardware Interfaces



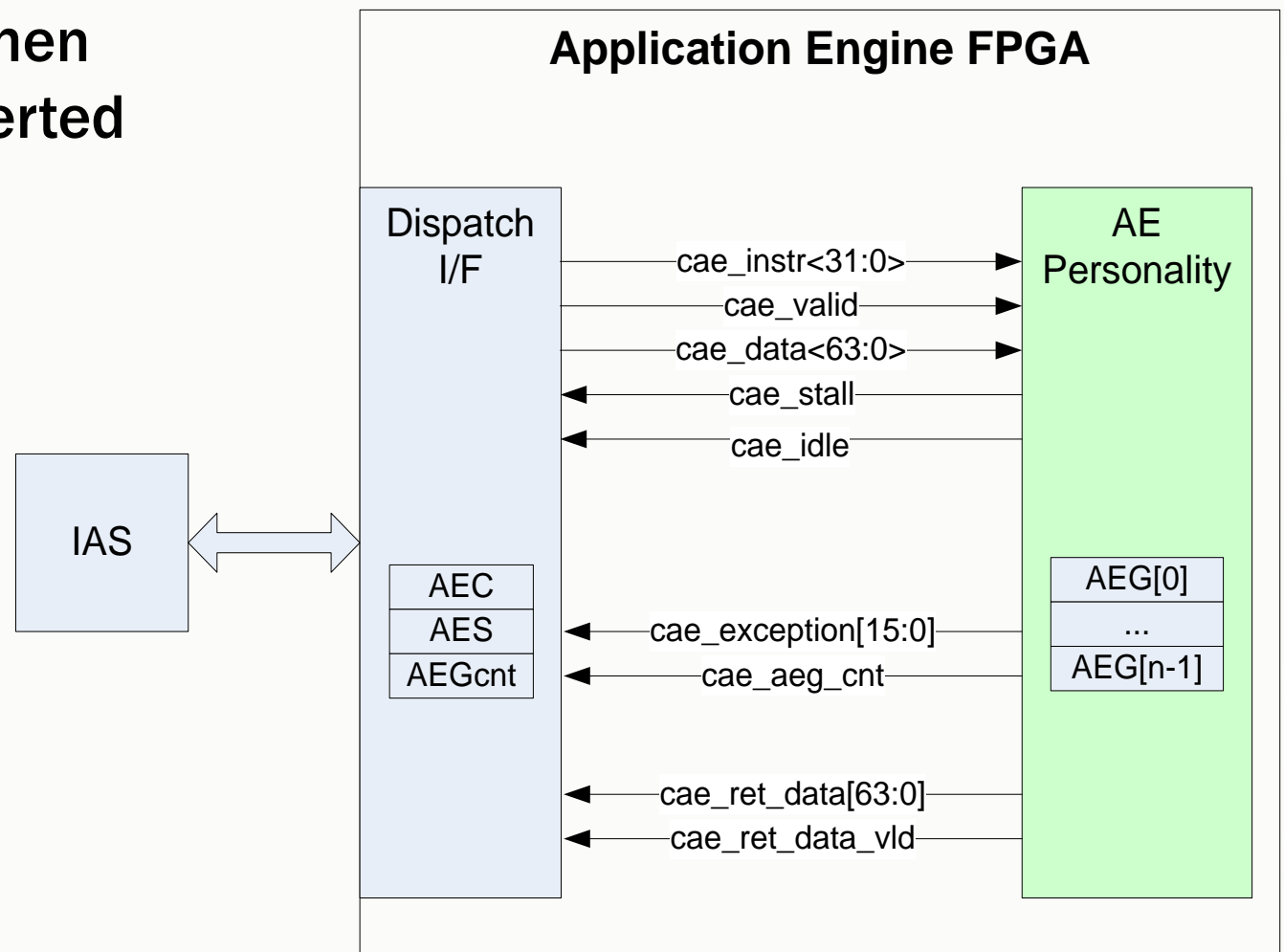
Dispatch Interface

- Instruction crack/decode
- AEEM, AEC, AES and AEGcnt register logic
- Unimplemented instruction checking
- Exception handling



Dispatch Interface

- New instruction (and associated scalar data, if any) is valid when `cae_valid` is asserted
- Personality can stall dispatch of new instructions by asserting `cae_stall`
- Personality must assert `cae_idle` when not busy

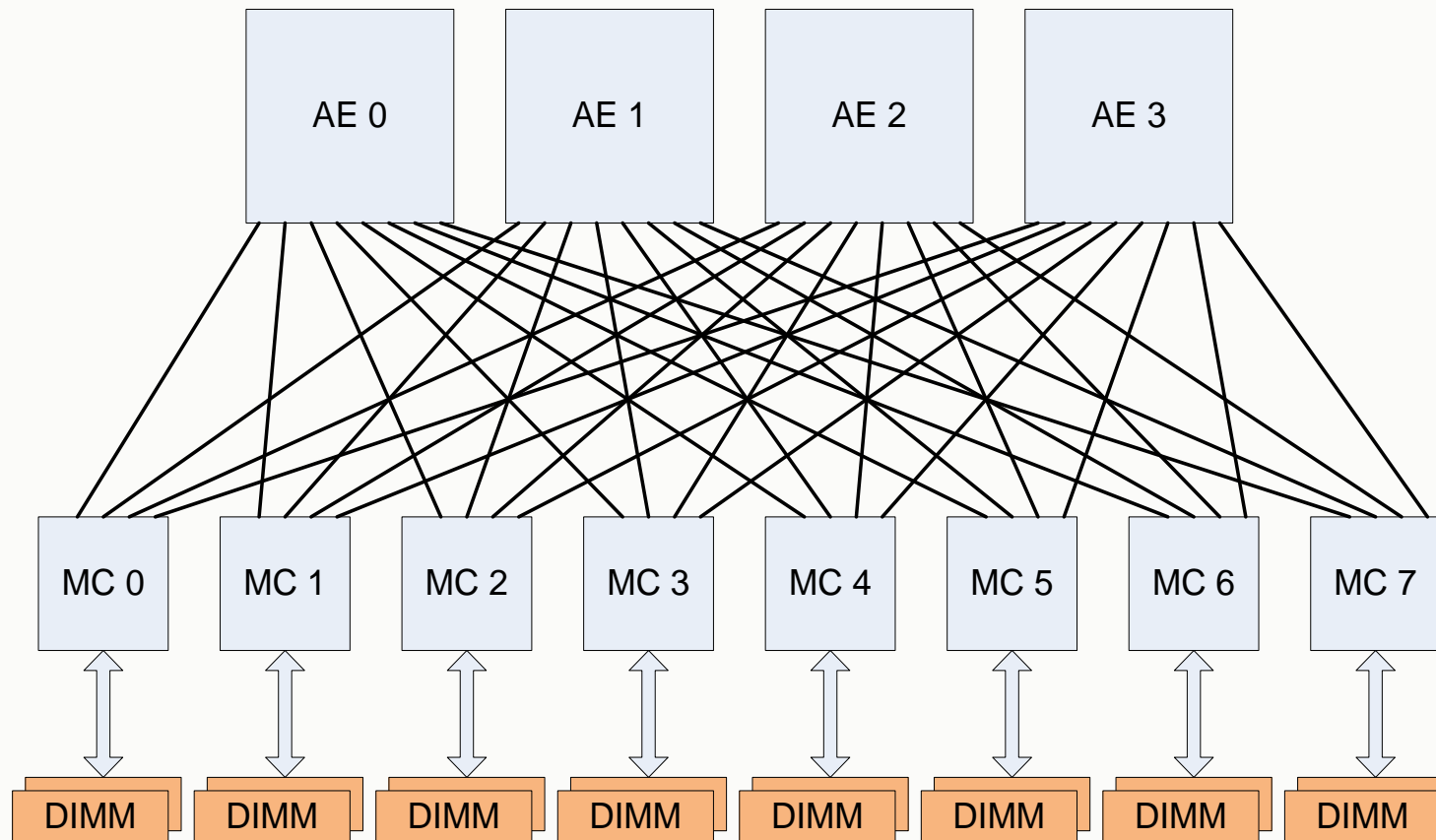


MC Interface

- **Each AE contains 8 MC interfaces, each connected to a Memory Controller FPGA**
 - Each MC connects to 1/8 of coprocessor memory
- **Optional MC crossbar routes memory requests to the appropriate MC based on the virtual address**
 - If the crossbar is not used, requests must be sent to the appropriate MC interface (see interleave)

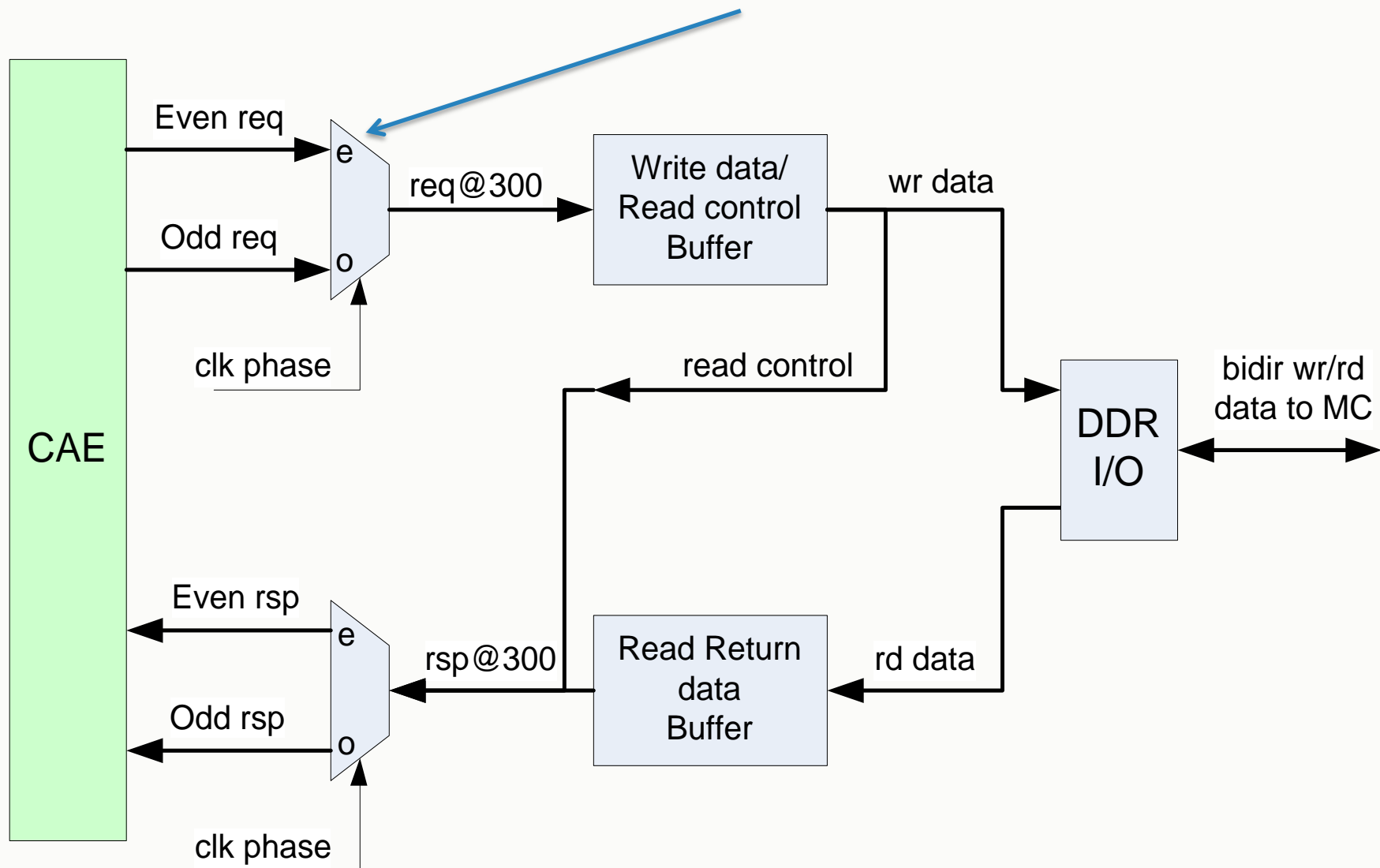
MC Interface

- All AEs are directly connected to all MCs
- Aggregate peak bandwidth is 76.8GB/s:
 - $4 \text{ AEs} * 8 \text{ MCs} * 300\text{MHz} * 8 \text{ bytes}$



MC Interface

300MHz AE-MC links are divided into “Even” and “odd” 150MHz memory interfaces



MC Interface

- **MC request data bus is used for both reads and writes:**
 - 64 bits of data on writes
 - 32 bits of read control data on reads

Request

mc_req_ld
mc_req_st
mc_req_vadr[47:0]
mc_req_size[1:0]
mc_req_wrd_rdctl[63:0]

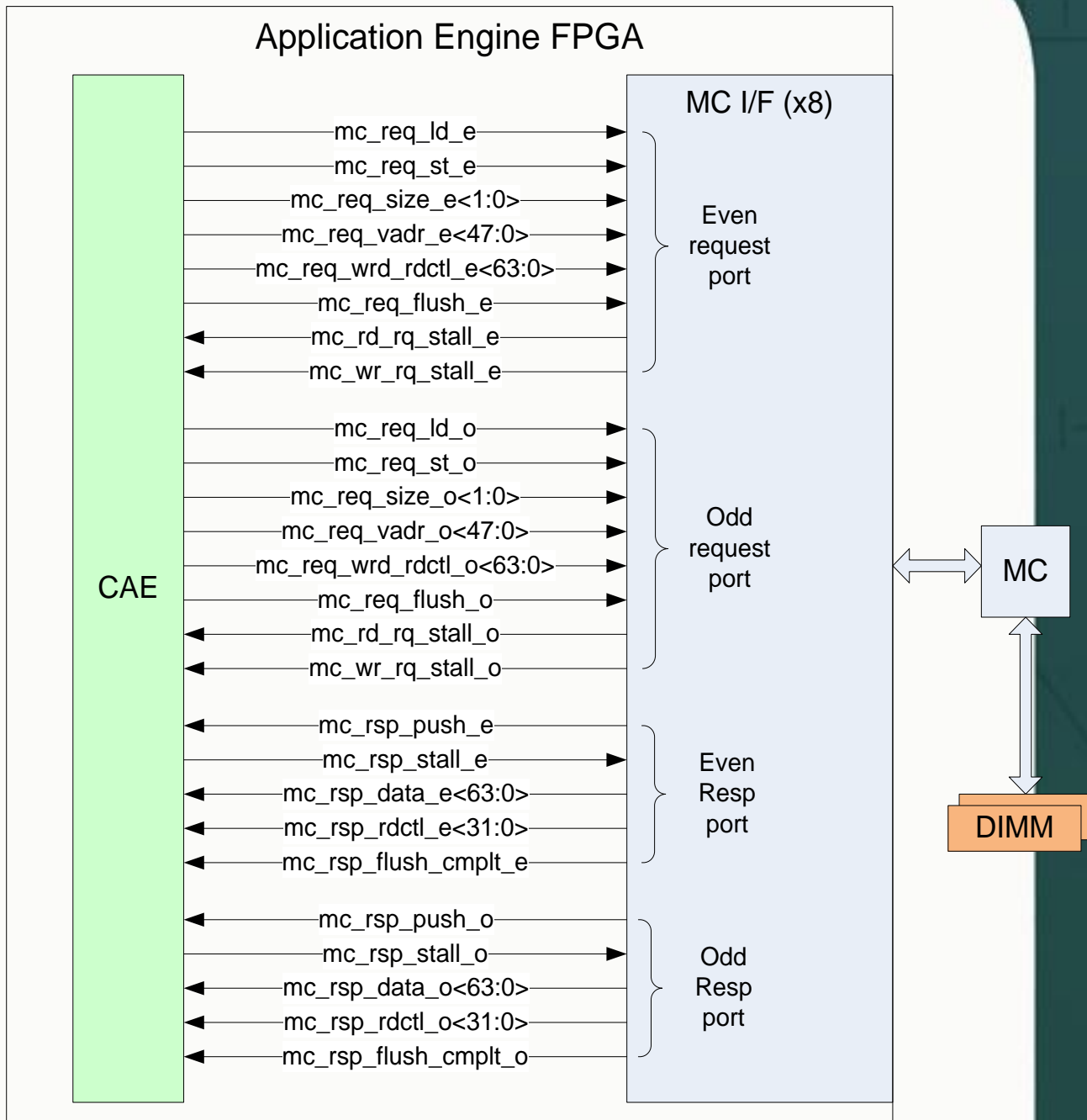
Response

mc_rsp_push
mc_rsp_data[63:0]
mc_rsp_rdctl[31:0]

Read control returned with response

MC Interface

- Load/store can be sent on even and odd ports every 150MHz cycle (unless stall is asserted)
- Read responses are returned on the even and odd response ports
- Write flushes can be used to synchronize data in memory



Memory Ordering

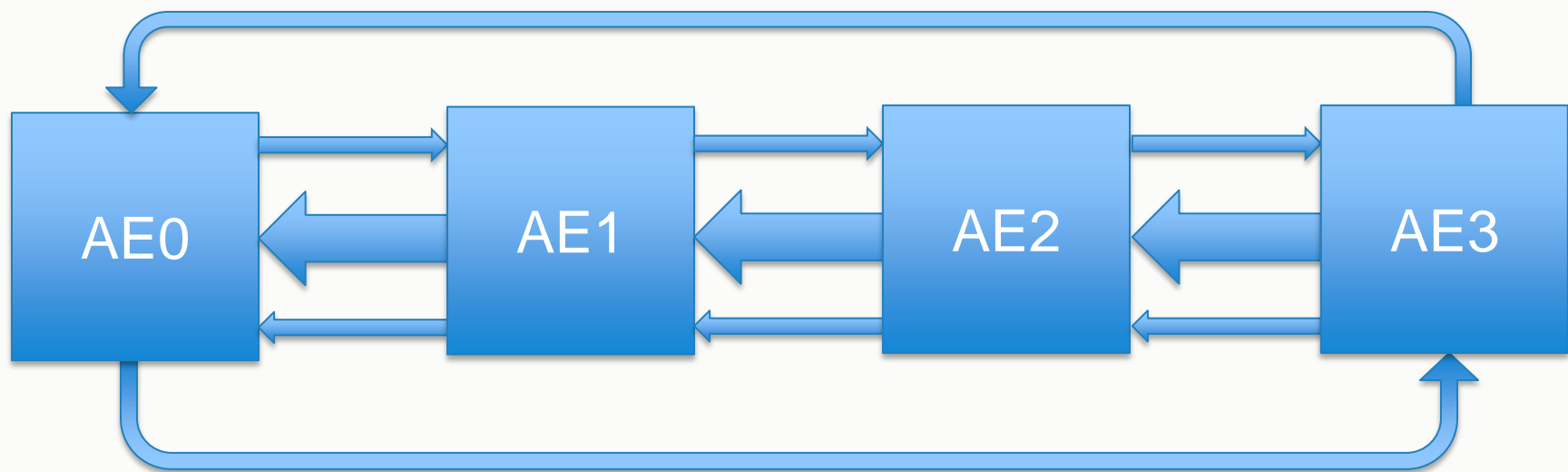
- **The coprocessor supports a weakly ordered memory model**
 - Coprocessor memory reads and writes can proceed independently to memory following different interconnect paths to maximize memory bandwidth
- **Options for memory ordering**
 - Fence instructions can be used to order memory requests but propagate through all AEs and all MCs
 - Optional read ordering module in AE returns reads in order of requests
 - Optional strong-ordering module in the AE tracks all outstanding requests and stalls when a hazard is detected
 - MC interface write flush signals can be used to ensure all outstanding writes have completed

MC Interface – Write Flush

- **Stores to memory are unordered, so it is possible for a store indicating “data valid” can pass a store of the related data**
- **The write flush signals can be used to ensure all outstanding writes have completed:**
 - `mc_req_flush_*` signals the flush
 - `mc_rsp_flush_cmplt_*` indicates the flush is complete

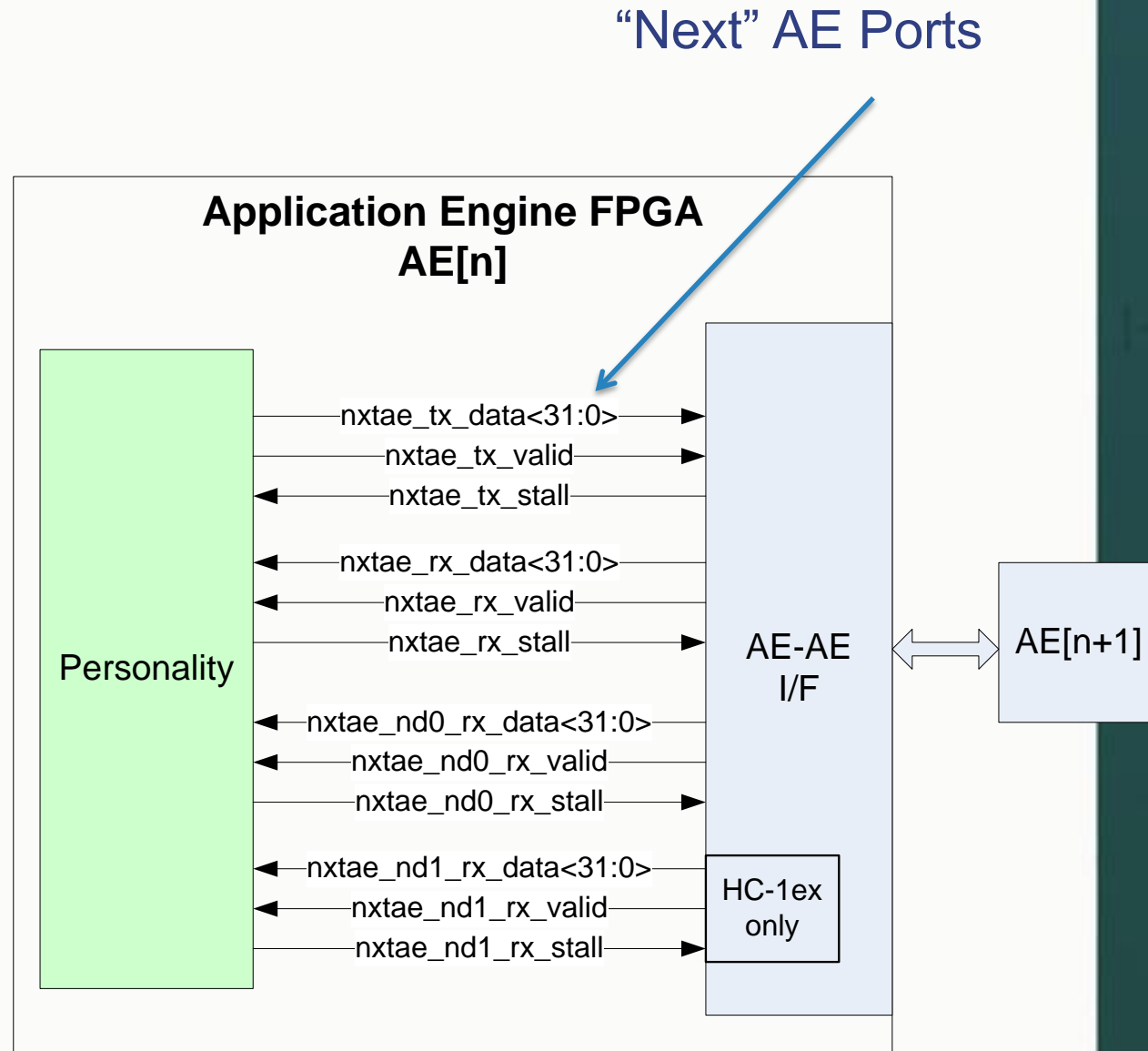
AE-to-AE Interface

- **Two “narrow” links connect the AEs in a ring, one in each direction**
 - 8 bits at 300MHz QDR → 32 bits at 150MHz
- **One link (two in HC-1ex) connects neighboring AEs in one direction**
 - 16 bits at 300MHz QDR → 68 bits at 150MHz



AE-to-AE Interface

- Allows direct data transfer between AEs
- Interface is enabled with a makefile variable (ports are removed if interface is not enabled)
- Parity/CRC and flow control handled in the interface

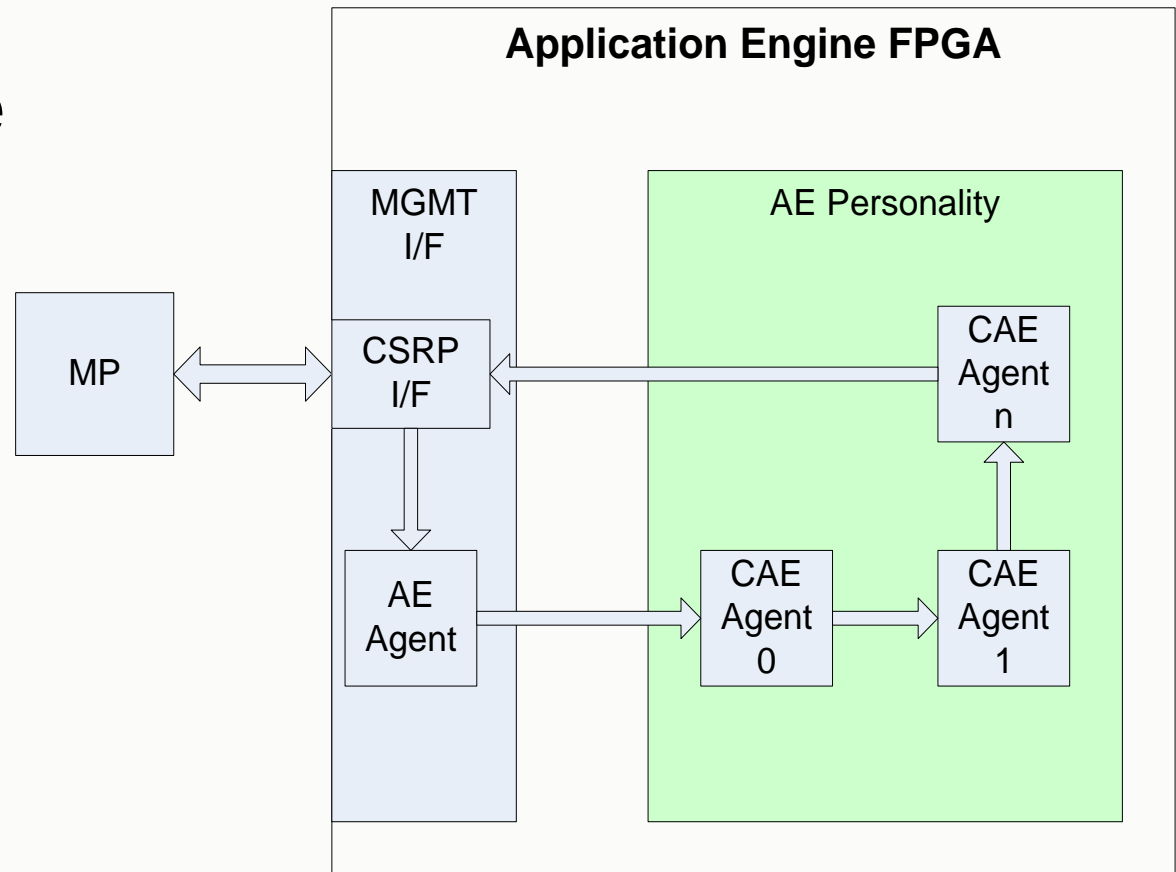


Management/Debug Interface

- **Connection from the Management Processor (MP) to the AE FPGAs**
- **Connects to CSR ring inside the AE, allows the MP to monitor and configure FPGA**
- **Can be used by the personality for debug**
 - Personality state
 - Performance counters
- **Can be accessed even if the AE is hung**

Management/Debug Interface

- CSR Agents are optional, but the ring must be completed
- The module `cae_csr.v` in the sample personality can be used as a starting point



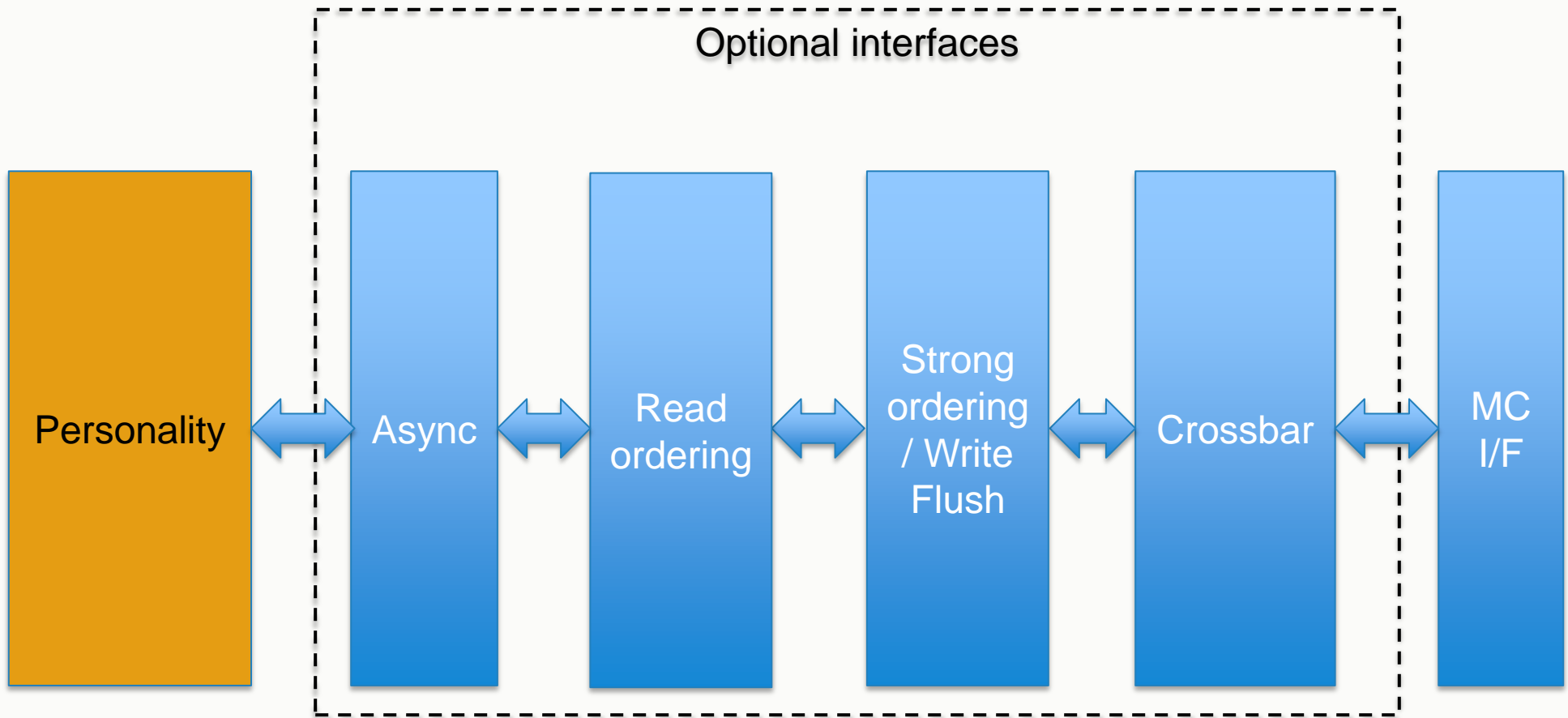
Clocks

- **Dispatch and memory interfaces to custom personality use 150MHz clock**
- **These clocks are generated by a PLL in the AE and can be used by custom personality**
 - 150MHz, 300MHz, 75MHz
- **Custom personalities can generate other clocks by instantiating another PLL or DCM**
 - Asynchronous interfaces automatically instantiated based on a clock ratio parameter

Configurable Infrastructure

- **The PDK includes an infrastructure of configurable components**
- **All of these are enabled using variables in the build environment:**
 - Asynchronous interface to personality
 - Read ordering
 - Strong ordering
 - Memory Crossbar
- **Well qualified and designed to ease timing closure**

Memory Interface Connectivity

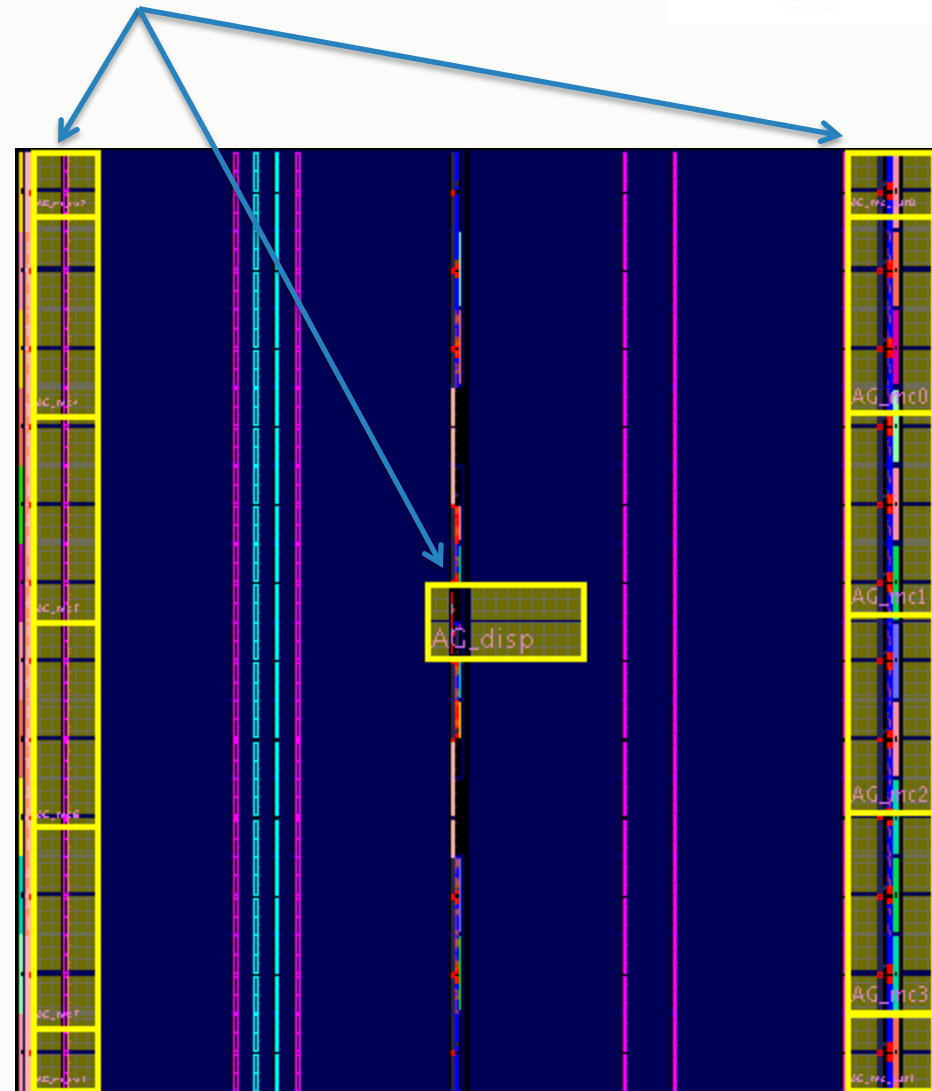


HC-1 FPGA Resources



PDK Infrastructure

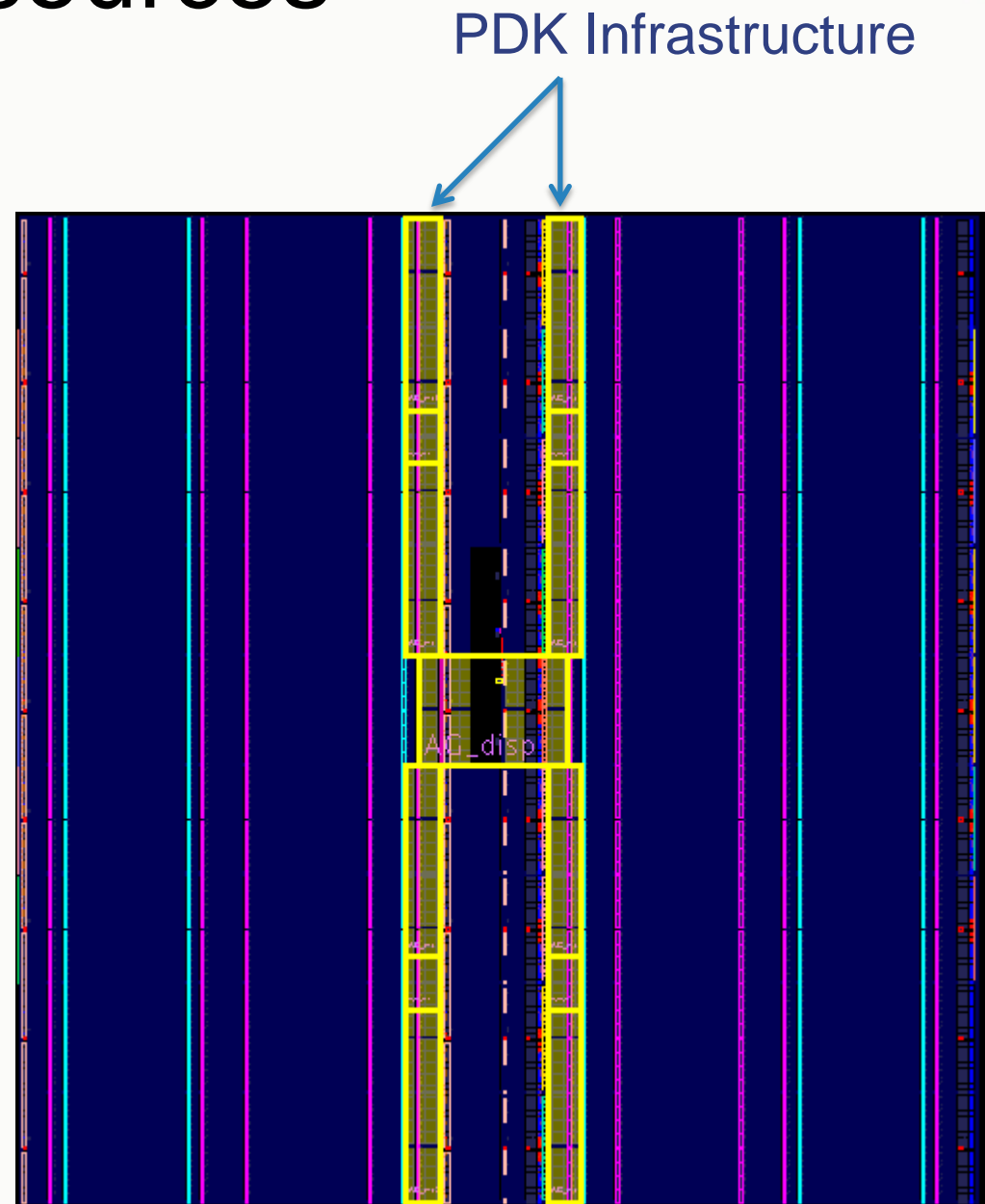
- AEs are implemented in Xilinx Virtex 5 LX330 FPGAs
- Required Convey interfaces use about 11% of the available slices and 25% of available block RAMs



Virtex 5 LX330

HC-1^{ex} FPGA Resources

- HC1^{ex} AEs are implemented in Xilinx Virtex 6 LX760 FPGAs
- Required Convey interfaces use about 6% of the available slices and 12% of available block RAMs



Virtex 6 LX760

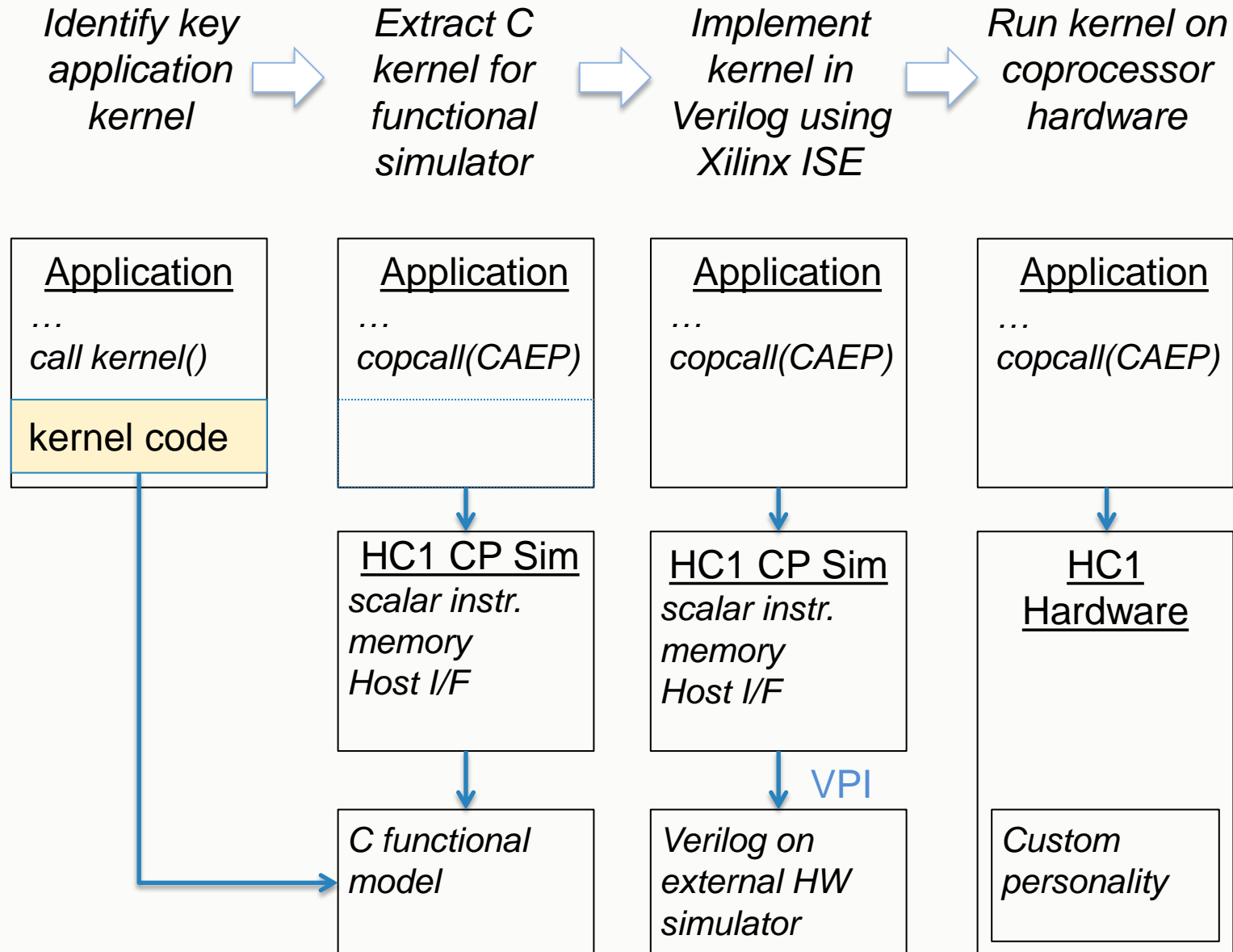


SIMULATION ENVIRONMENT

PDK Simulation Environment

- **The PDK simulation environment is designed to enable a stepwise approach to personality development**
- **The Convey Architecture Simulator allows the developer to use application to drive software or hardware simulation**
- **Allows the user to debug at a higher level at abstraction**

Convey PDK Design Flow

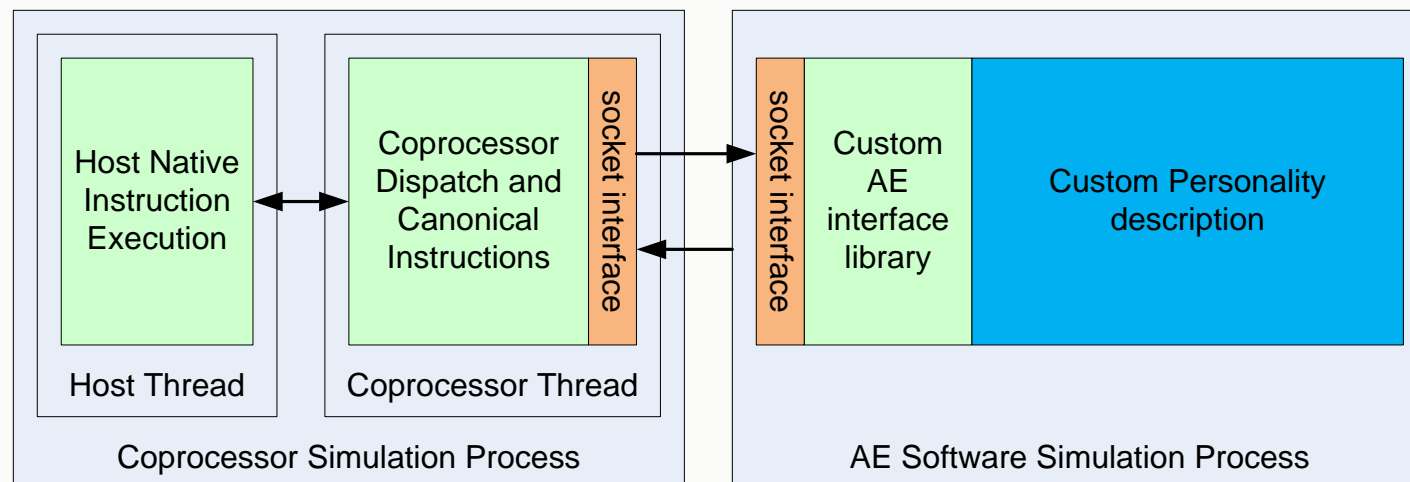


Coprocessor Architecture Simulator

- **Architecture simulator allows debug of coprocessor application prior to running on hardware**
- **Simulator is started when the application is run**
 - CNY_SIM_THREAD = libcpSimLib2.so
- **For custom personalities, the simulator executes the custom AE software model or Verilog simulation**
 - CNY_CAE_EMULATOR = <exe>
 - Default is /opt/convey/personalities/<sig>/caeemulator

AE Software Simulation

- Custom AE software model connects to the architecture simulator through a socket interface
- AE software model interfaces are provided by Convey
 - Instruction dispatch, memory load and store
- Personality developer implements software model of the custom personality in C++

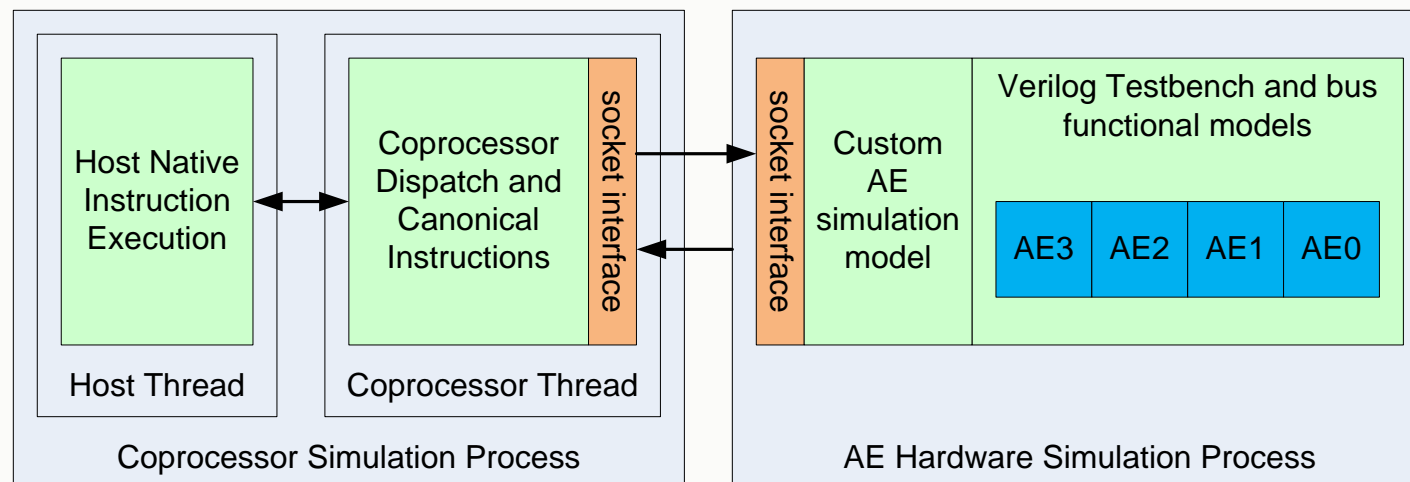


Application Engine Software Model

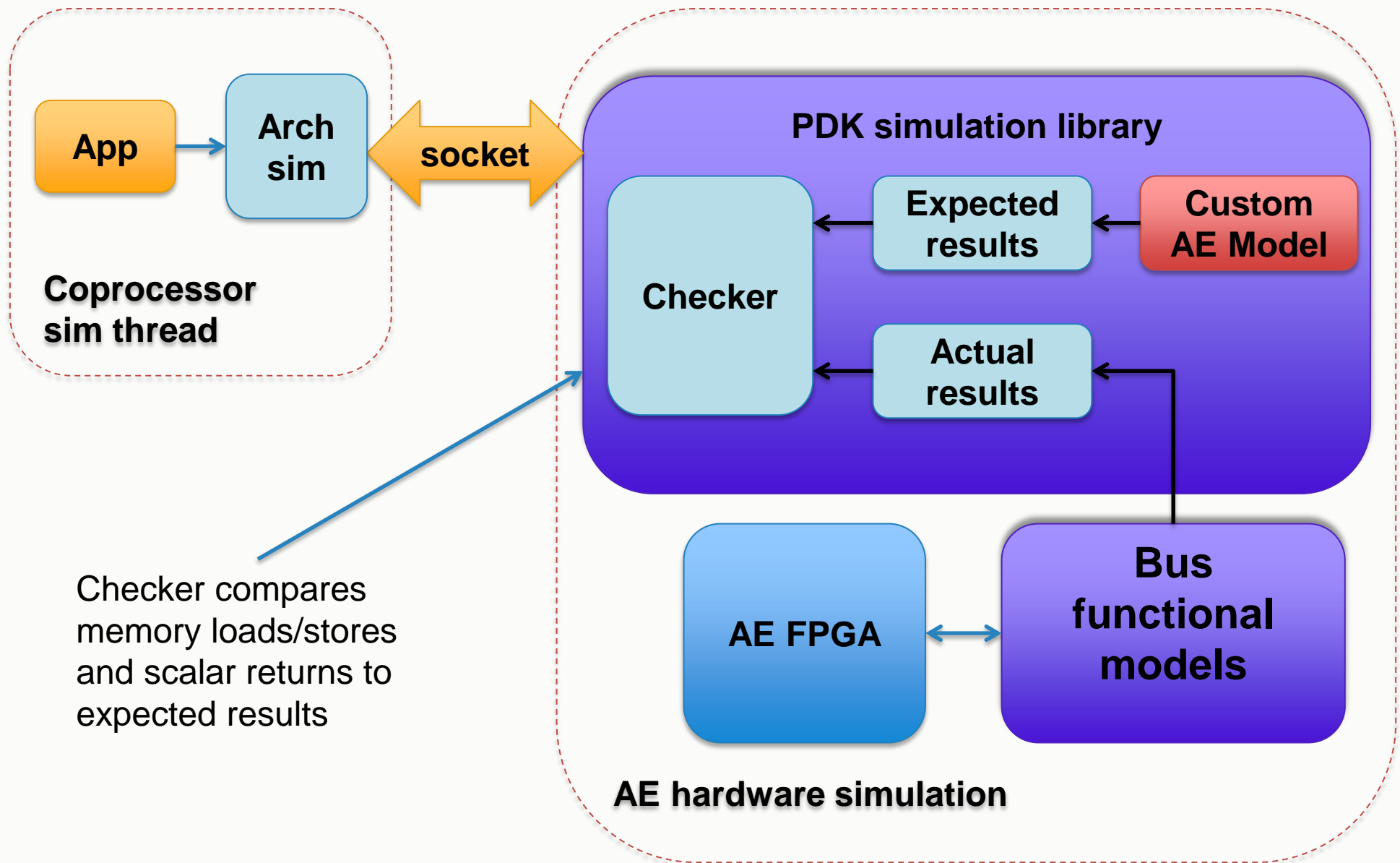
- **AE model must implement these functions**
 - InitPers: provides initial state (and AEG count)
 - CaepInst: implements custom instructions
- **Callable functions**
 - AeMemLoad: load request
 - AeMemStore: store request
 - ReadAeg: reads a 64-bit AEG register
 - WriteAeg: writes a 64-bit AEG register
 - SetException: sets AE exception bit in AES register
 - SetAegCnt: defines AEG max for personality

Hardware Simulation

- The PDK hardware simulation environment also uses the coprocessor simulator and custom AE software model
- VPI (Verilog Procedural Interface) allows the arch simulator to drive AE Verilog model



Hardware Simulation





PDK Tool Flow

PDK Tool Flow

- **The PDK includes libraries and makefiles to be used with third-party tools**
- **These tools are required by the PDK but not included**
 - Xilinx ISE
 - HDL Simulator (VCS and Modelsim supported)
 - Xilinx Chipscope is strongly recommended!
 - Xilinx PlanAhead can be very useful in physical design

PDK Installation

- **PDK is installed in /opt/convey/pdk/<rev>**
 - Rev is a dated revision of PDK
- **Environment variables point to PDK:**
CNY_PDK= /opt/convey/pdk
CNY_PDK_REV = <rev>
CNY_PDK_PLATFORM = (hc-1 | hc-1ex)

PDK Project

- **FPGA projects have this structure by default**

pdk_project/

Makefile.include – project settings, points to PDK makefile

phys/ - Xilinx physical implementation, contains constraints files

sim/ - Simulation directory, including AE software model, configuration files for hardware simulation

verilog/ - RTL to be synthesized into FPGA

PDK Makefiles

- **Project Makefile.include allows project settings to be kept in one place and used by sim and phys makefiles**

```
# Project settings
```

```
# PDK variables
```

```
CNY_PDK = /opt/convey/pdk
```

```
CNY_PDK_REV = 2011_11_22
```

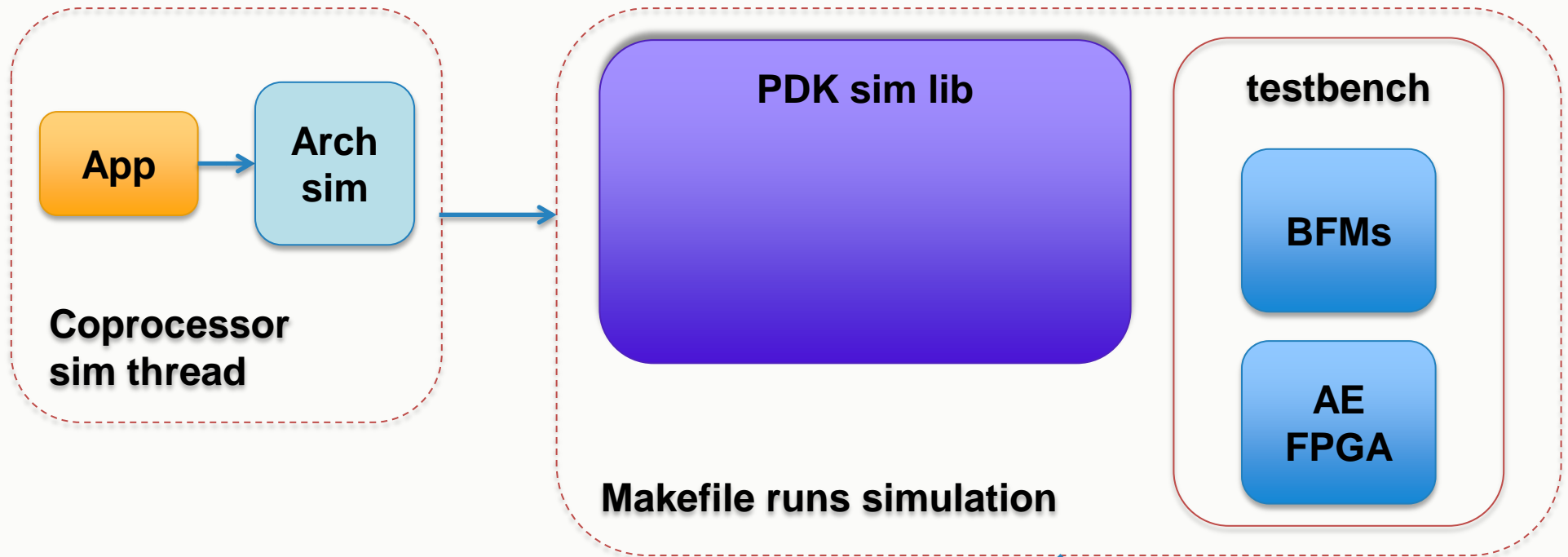
```
CNY_PDK_PLATFORM = hc-1ex
```

```
# Include Convey Makefile Template
```

```
include
```

```
$(CNY_PDK)/$(CNY_PDK_REV)/$(CNY_PDK_PLATFORM)/lib/MakefileInclude.cnypdk
```

Hardware Simulation



Hardware sim is started by running the application

→ `CNY_CAE_EMULATOR = <sim_script>`

```
# CNY_PDK_SIM_SEED = 11223344  
  
# Include Convey Makefile Template  
include ../Makefile.include
```

AE Physical Build

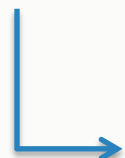
- To implement the AE using Xilinx ISE, simply run 'make' in the phys directory
- User can add source and ucf files from the project makefile

```
# optional user constraints
UCF_FILES += smpl_pers.ucf

# Include Convey Makefile Template
include ../Makefile.include
```

AE Physical Build

- **PDK build flow automatically runs synthesis, place and route, timing analysis and bitgen**
- **After the build is run successfully, use ‘make release’ to package the bitfile**
 - generates a release directory at the project level
 - generates cae_fpga.tgz and a project archive

 /opt/convey/personalities/4.1.1.1.0/ae_fpga.tgz



DEBUGGING A CUSTOM PERSONALITY

Design for Debug

- **Visibility is often the most challenging part of debugging an FPGA**
- **PDK provides 3 ways to get visibility into FPGA**
 - AEG registers defined as status can be read by the application (or GDB)
 - CSR Agents can provide management processor visibility into internal FPGA state (even if the application is hung)
 - Chipscope logic analyzer

GDB

- **Provides access to machine state**
 - AEG registers in the AE
- **GDB description file allows user to specify how registers are displayed**
- **Debug host application as well as all coprocessor state**
- **GDB pauses on an instruction boundary**
 - Not effective if app is hung

#CMD	Name	Mask	AEG	
	format			
REG	ad1	0x0f	0	u64
REG	ad2	0x0f	1	u64
REG	ad3	0x0f	2	u64
REG	size	0x0f	3	u64
REG_R	sum	0x0f	10	25 u64

GDB

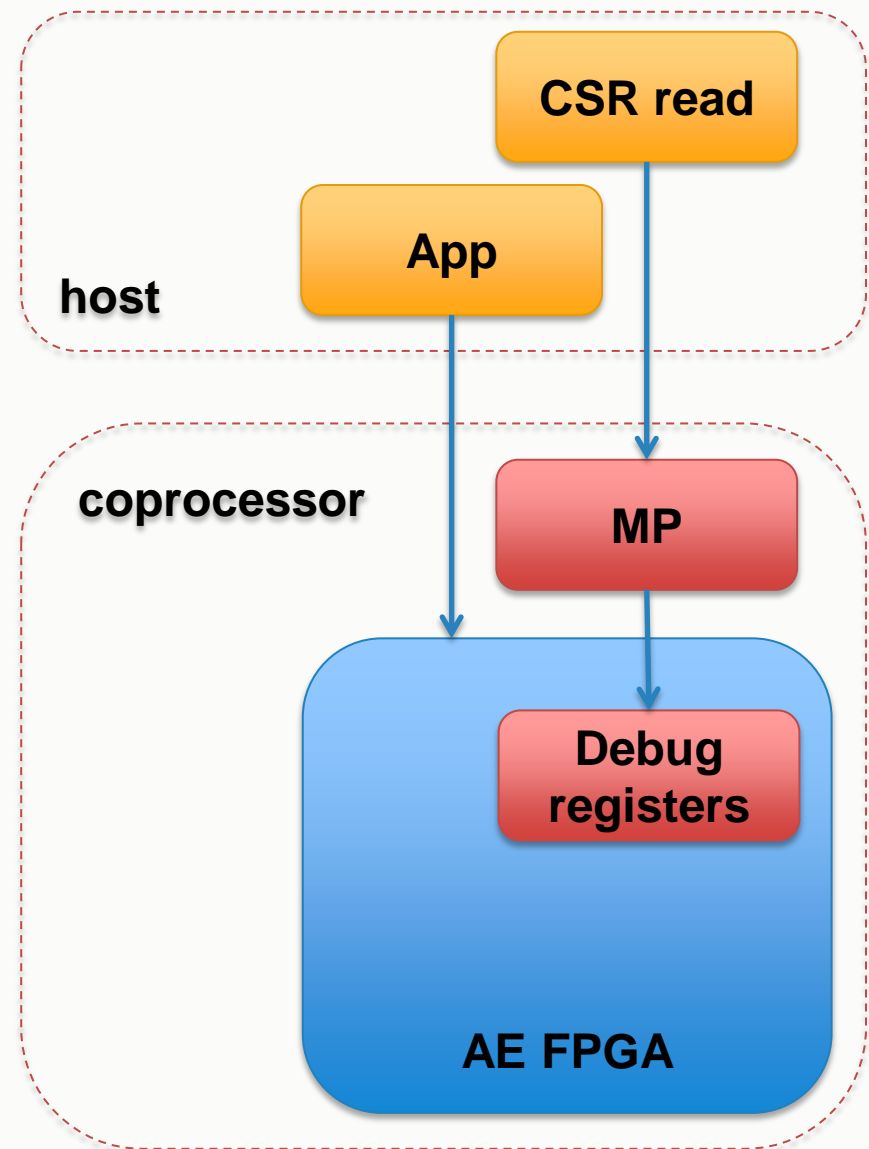
- **To run GDB, run Convey's GDB with full path**
 - `/opt/convey/bin/gdb`
- **Convey GDB gives access to host app as well as all coprocessor state**
- **For help with Convey GDB, type**
 - `info cny_help`

GDB

- GDB uses a GDB description file to display registers
- By default, it looks for 'gdbregisters' in `/opt/convey/personalities/<sig>/`
- During development, description file can be manually loaded in GDB
- GDB pauses on an instruction boundary

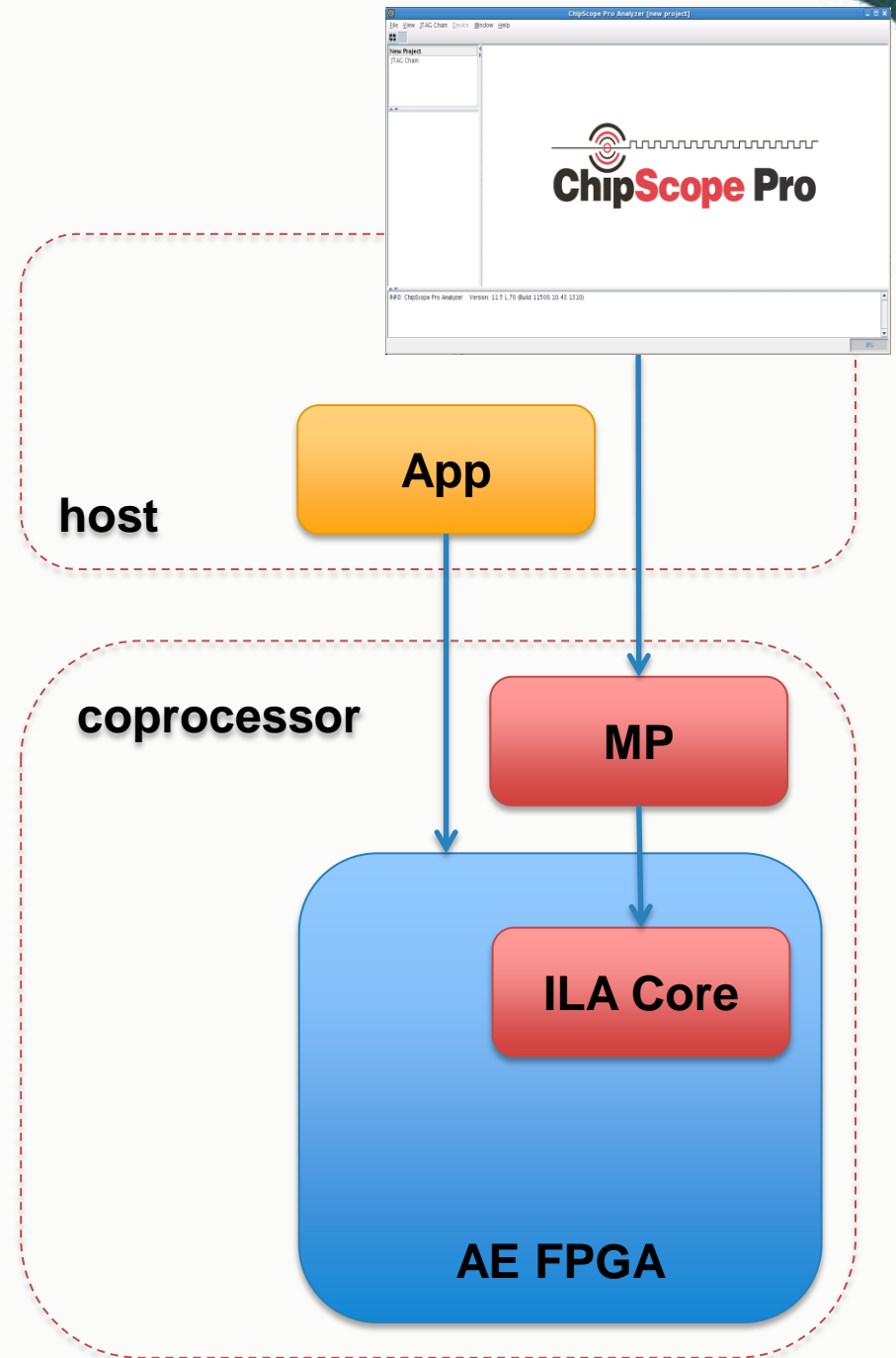
CSR Access through MP

- **Registers connected to CSR rings can still be read through the Management Processor**
 - Works even if the AE is hung



Xilinx Chipscope

- **Chipscope Pro Analyzer** can provide greater visibility into the **FPGA**
 - Logic analyzer enables complex triggering and buffering of lots of data
- **Remote Chipscope** through the **MP** allows debugging a design without physically connecting cables



Inserting a Chipscope core

- **Chipscope core can be inserted using Core Inserter**
 - run inserter (or inserter.sh for older ISE versions) in the phys directory
- **Use cae_fpga.ngc as the input netlist**
- **Change output netlist to cae_fpga.ngo**
- **Once the core is inserted, type 'make' in the phys directory to rerun from ngdbuild step**

Starting the Chipscope server

- **Remote Chipscope capability can be used to debug a design without physically connecting cables**
- **On the HC-1 host start the Chipscope server as root**
 - `/opt/convey/sbin/mpchipscope start`

Running the Chipscope analyzer

- **On a development system, run the Chipscope analyzer**
 - analyzer.sh
- **Click on the “JTAG Chain” pulldown and select “Open Plug-in”**
- **In the Plug-in Parameters box, enter**
`'xilinx_xvc host=[host IP]:2542 disableversioncheck=true'`

Running the Chipscope analyzer

- A connection is established to the JTAG chain, and a list of devices appears
- If your design contains analyzer cores, you should see 4 cores available
- AE FPGAs must be loaded in order for chipscope to connect

Flush the MP cache

Add the image

Load the image

```
/opt/convey/sbin/mpcache -f  
/opt/convey/sbin/mpcache --add -S 4.1.1.1.0  
/opt/convey/sbin/mpcache --load -S 4.1.1.1.0
```

Best Practices for Debug

- **Debug application and model fully before debugging verilog**
 - AE software model must accurately represent the AE
- **Implement exceptions for illegal operations or invalid input data**
- **Use debug registers for visibility (counters, state, overflow/underflow, etc.)**
- **Use watchdog timers for long instructions**

Coprocessor Logs

- **The /var/log/messages file on the host contains useful debug information from both the host and the MP**
 - Which personality was loaded
 - Segmentation faults from the AEs
 - Hardware errors on the coprocessor



SAMPLE PERSONALITY

Sample Personality

- **Sample personality is provided with PDK to use as a starting point**
- **Simple Vector Add personality to illustrate how to connect to all interfaces**

```
for (i=0; i<length; i++)  
{  
    a3[i] = a1[i] + a2[i];  
    sum += a3[i];  
}  
return sum;
```

Sample Personality

- **Sample personality machine state**
 - Array addresses and length sent to AEGs 0-3
 - Sums returned in AEGs 30-33

AEG Index	Register Name	Description
0	MA1	Memory Address 1
1	MA2	Memory Address 2
2	MA3	Memory Address 3
3	LEN	Length – number of add operations/elements in each memory array
30-33	SAE[3:0]	Application Engine Sums
40	STL	Total Sum (of all participating AEs)

Sample Application

- **Get the signature for the personality**

```
cny_get_signature ("pdk", &sig, &sig2);
```

- Note that signature “pdk” is for the sample
- Customer creates a new one (64000+)

- **Use copcall to dispatch routine**

```
act_sum = l_copcall_fmt(sig, cpVadd, "AAAAA",  
                        a1, a2, a3, size, ae_sum);
```

Copcall to Dispatch Routine

```
sum = l_copcall_fmt(sig, cpVadd, "AAAAA", a1, a2, a3, size, ae_sum);
```

Operands passed
through scalar registers
starting with A8

Result returned in A8

cpVadd:

```
mov %a8, $0, %aeg  
mov %a9, $1, %aeg  
mov %a10, $2, %aeg  
mov %a11, $3, %aeg  
caep00 $0
```

```
...  
mov.ae0 %aeg, $30, %a8  
rtn
```

To AE
Dispatch


Sample Application

**Write arguments to
AEG registers and
execute custom
instruction...**

cpVadd:

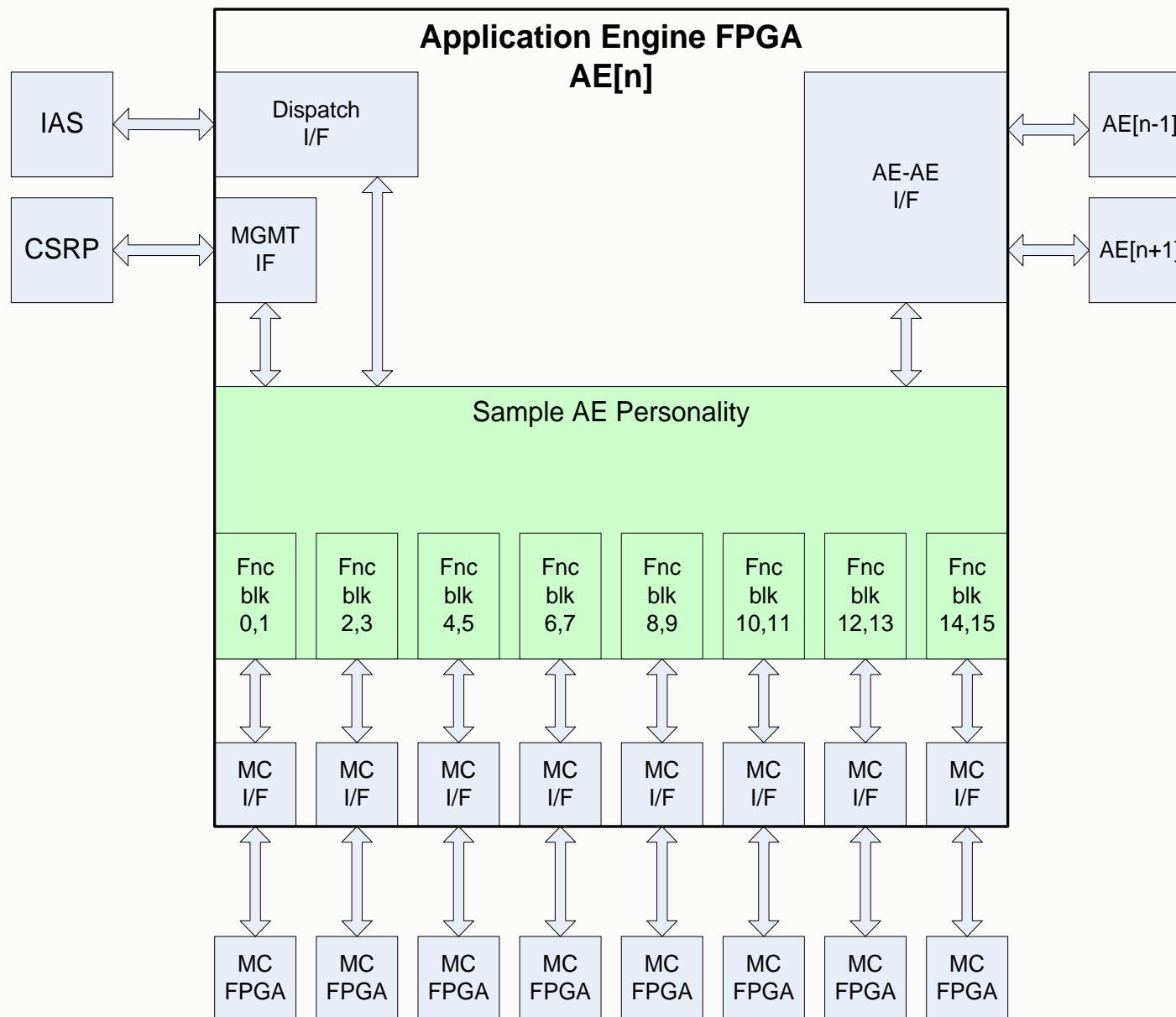
```
mov %a8, $0, %aeg  
mov %a9, $1, %aeg  
mov %a10, $2, %aeg  
mov %a11, $3, %aeg  
caep00 $0
```

**...Then store sums
from each AE to
memory**



```
mov.ae0 %aeg, $30, %a16  
mov.ae1 %aeg, $31, %a17  
mov.ae2 %aeg, $32, %a18  
mov.ae3 %aeg, $33, %a19  
st.uq %a16, $0(%a12)  
st.uq %a17, $8(%a12)  
st.uq %a18, $16(%a12)  
st.uq %a19, $24(%a12)  
mov.ae0 %aeg, $30, %a8  
rtn
```

Sample Personality Block Diagram



Running the Sample Personality

- **Copy sample personality to a working directory**

`/opt/convey/pdk/2011_11_22/hc-1/examples/cae_vadd`

- **The Makefile.include points to the correct version of PDK and PLATFORM**

Simulating the Sample Personality

- **Run the sample application using the 'run' script in the SampleAppVadd directory**
 - To compile the application, run 'make'
 - To run using the software model:
 - ./run
 - To run a Verilog simulation
 - ./run -vsim
 - Requires CNY_PDK_HDLSIM = (Mentor|Synopsys), set in user or site Makefile

Running the Sample Application

- **A working FPGA image for the sample application is installed on the Convey system**
 - `opt/convey/personalities/4.1.1.1.0/ae_fpga.tgz`
- **To run the sample app using the coprocessor, use the 'runcp' script in the SampleAppVadd directory**
 - `./runcp`

AE Physical Build

- To Build the AE, run 'make' in the cae_vadd/phys directory
- This uses a templated Makefile in /opt/convey/pdk, generates a bitfile for the personality
- Use 'make release' to package the bitfile into a tgz file

Flushing the MP Cache

- **AE images are cached by the Management Processor (MP)**
- **When a routine is dispatched to the coprocessor, the MP uses a cached AE image if available**
- **If the image is changed, force a reload (as root) with**
 - `/opt/convey/sbin/mpcache -f`

Host messages file

- **The /var/log/messages file on the host contains useful debug information from both the host and the MP**
 - Which personality was loaded
 - Segmentation faults from the AEs
 - Hardware errors on the coprocessor

Cnyinfo command

- **The cninfo command on the host can be used to gather information about the coprocessor:**
 - Memory interleave scheme
 - Amount of memory
 - Active personality signature
 - hardware/software feature versions

Setting Interleave Boot Option

- **Convey Scatter/Gather DIMMs support two interleave schemes: binary and 31-31**
- **To change the default interleave, add or change an entry in the `/etc/grub.conf` file**
 - `convey=interleave=3131`
 - `convey=interleave=binary`
- **Change the default grub entry if necessary**

Checking Interleave from App

- **An application can determine which scheme is enabled before dispatching to the coprocessor**

```
// check interleave is binary
if (cny_cp_interleave() == CNY_MI_3131) {
    printf("ERROR - interleave set to 3131, this
    personality requires binary interleave\n");
    exit (1);
}
```



PRODUCTIVITY

Third Party Tools for Accelerated Personality Development

- **C-to-VHDL synthesis tools**

- speed conversion of C functional prototype to hardware design
- connect to PDK standard interfaces



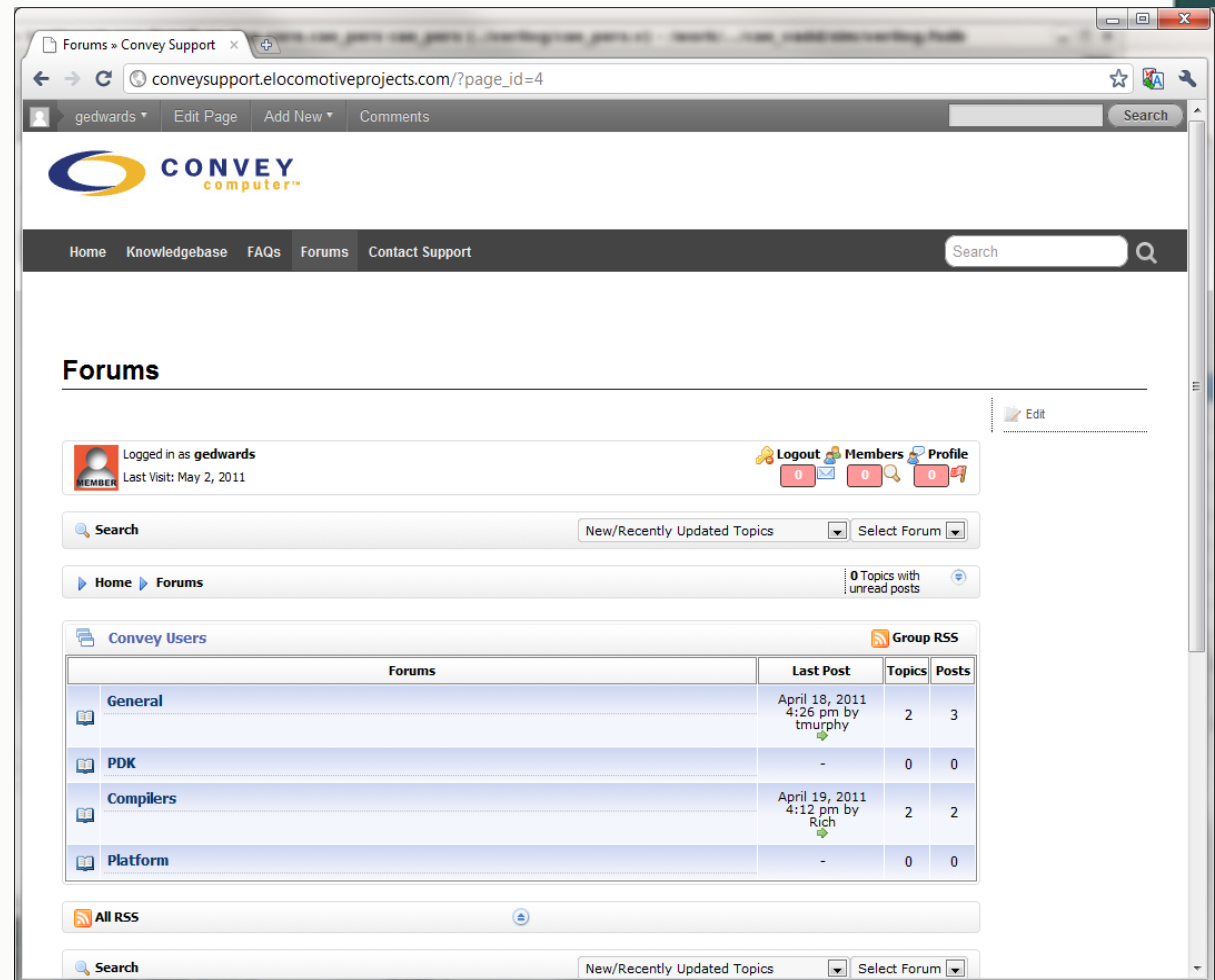
- **Functionality,
productivity,
performance**

High-Level Synthesis Tool Strategy

- **Strategy is to enable any high-level synthesis tools to be used on top of the PDK**
 - Based on customer demand
- **To support that, we focus on**
 - Flexible, easy-to-use infrastructure to build on
 - Generic interfaces exposed to the user or high-level tool
 - Easing timing so the user can focus on functionality

Support Website

- Support site has many resources for developers
 - Documentation
 - Knowledgebase
 - Users Forum
 - Downloads
 - Whitepapers



THE WORLD'S FIRST HYBRID-CORE COMPUTER.

