Designing, Verifying and Building an Advanced L2 Cache Subsystem using SystemC

Presentor: Steve Frank - Panève

Authors:

Thomas Tessier - Panève

Dan Ringoen - Panève

Hai Lin - Panève

Eileen Hickey - Panève

Steven Anderson - Forte Design Systems

Panève

FORTE DESIGN SYSTEMS

Agenda

Who is Panève?

Rhino Architecture and L2 Cache Focus for this presentation

Why SystemC?

Forte Leverage

Design Success

Q & A

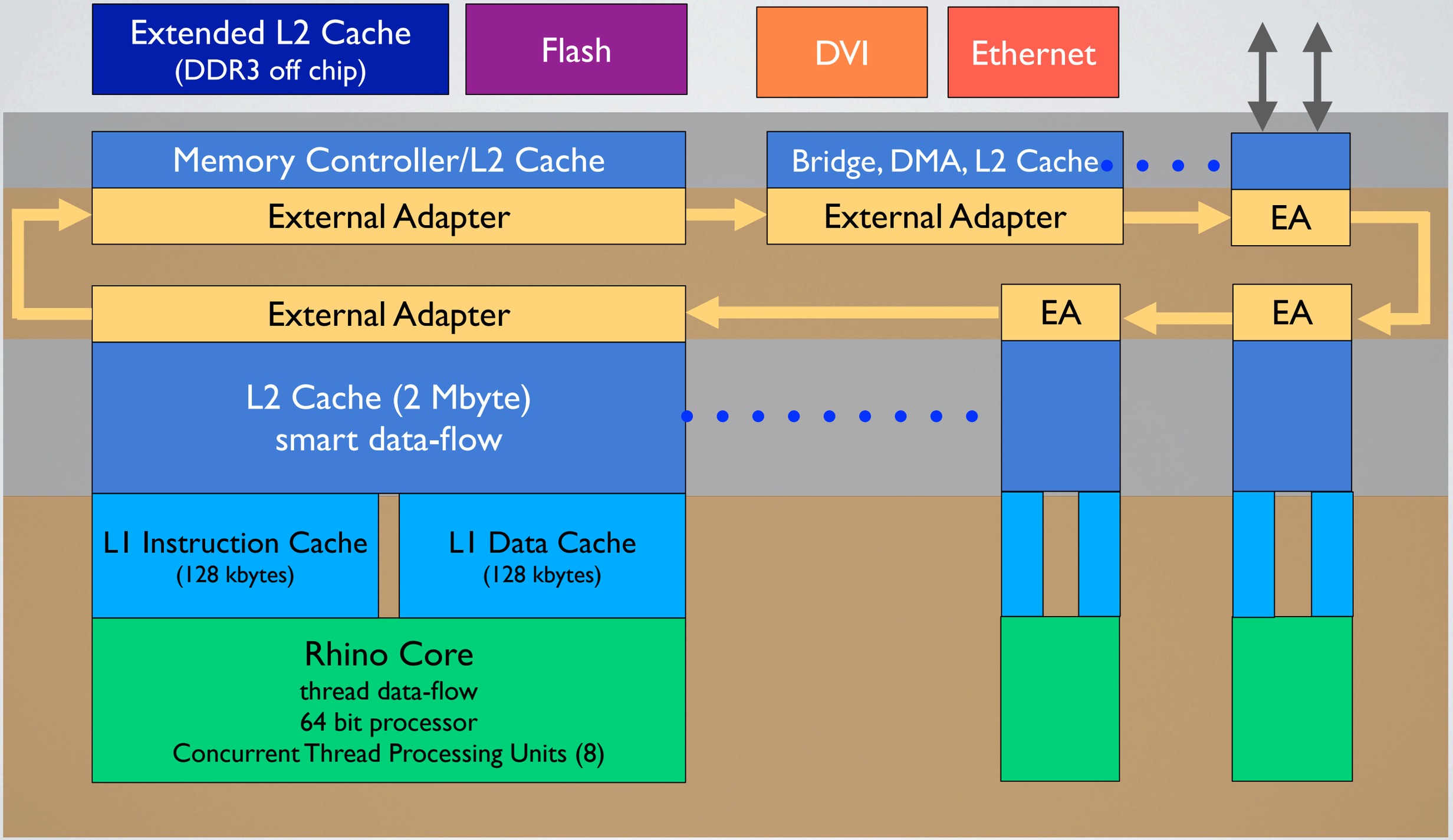
Panève

Targeting to be the leading provider of general purpose software-driven processing solutions for embedded devices

Architecture breakthrough allows fundamental shift from heterogeneous (multiple hard-wired units) to a single, software- driven SoC

Smart Data-Flow Caching enabling software transparent parallel processing

Rhino SOC Architecture



Focus on L2 Cache

Core innovation, manage complexity and verification

Entire system designed in SystemC

Design challenges were met using:

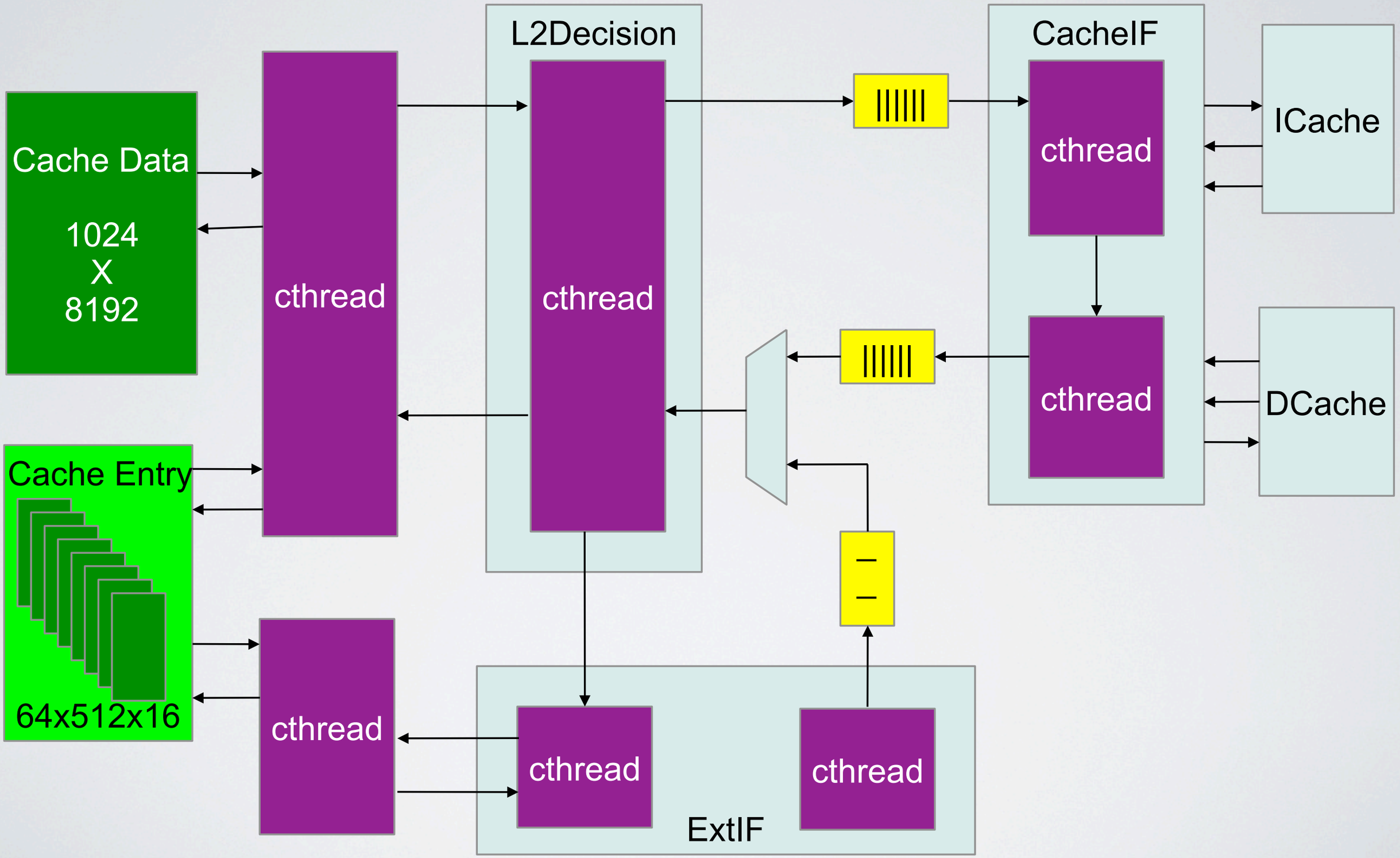
Point-to-Point (P2P) Interconnect

Compiled Memories

Clock Domain Crossing (CDC) interconnect

Templates and transactional organization

L2 Cache Block Diagram



Using SystemC for Hardware Design

SystemC can be used as your design source when coupled with a high level "Cynthesizer" tool

Full parameterized design

Use of templates makes it easy to describe parameterized functions.

Hybrid behavioral, transactional, cycle accurate design

A few lines of SystemC can create a lot of hardware, Caches are highly repetitive

SystemC as the Testbench as well means everyone is working in one language

Complex Classes

Data rarely travels alone, it has flags and states

Cache normally has a "tag" information with it

SystemC struct/class is the perfect approach

Forte manages data structures using helper methods
cynw_interrupt

Complex Classes

```
struct ext_header_s {
    ext_requestor_s requestor;
    ext_cmd_s cmd;
    block_address_s address;
    block_cstate_s cstate;
    bool modified;

    ext_header_s() {}
```

```
inline void init() {
    requestor = 0;
    cmd = 0;
    address.init();
    cstate.init();
    modified = false;
}
```

Other Standard methods
like: Assignment (=),
Compare (==) and
sc_trace are required

Complex Classes

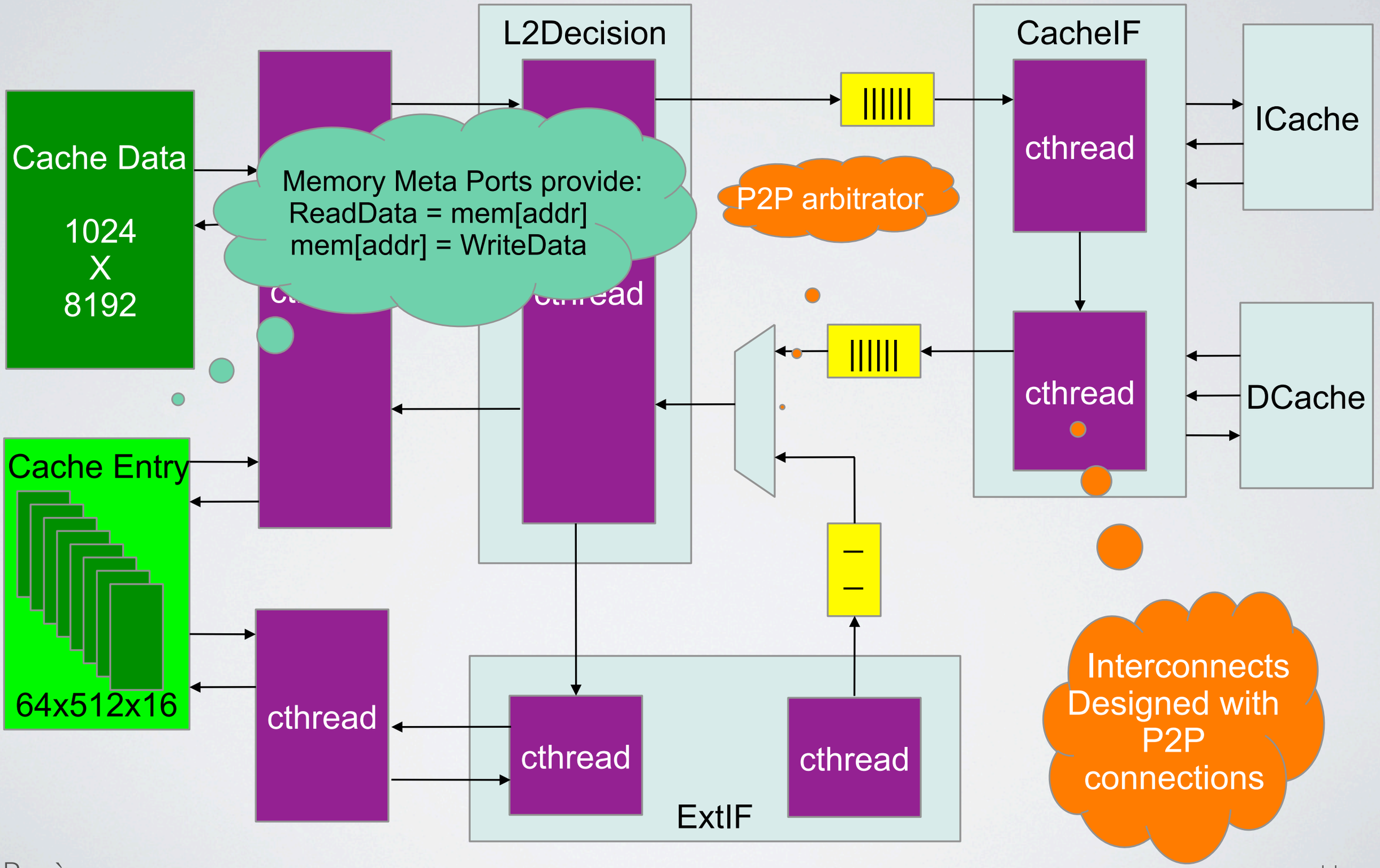
```
Inline void operator=(const
 a2c_packet_s& other) {
   requestor =
 other.requestor;
   cmd = other.cmd;
   address =
 other.address;
   cstate = other.cstate;
   modified =
other.modified
}
```

```
inline operator
  c2a_packet_s( ) const {
  c2a_packet_s retVal;
  retVal.requestor =
  requestor;
  retVal.cmd = cmd;
  retVal.address =
  address;
  retVal.cstate = cstate;
  retVal.modifed =
  modified;
  return retVal;
```

Assignment to other packet type

Casting to other packet type

P2P Interconnect Usage



Panève Usage of P2P

Use P2P at module ports to move complex structured data

Allowed what-if by replacing a P2P interface with a P2P FIFO, no change in design code to test

Testbenches were developed quickly as the complex class often had helper methods to develop data so the TB only need to deal with the flow.

Testbench Usage of P2P

Testbenches often exploited help applications in structures to create data

Testbench connectivity was just the P2P port and clock, so development was very quick

Only needed to consider control flow not data generation in most cases

Could be used with STL and SCV to exploit smart pointers and scoreboards

Forte Compiled Memories

Memory compiler of different widths and depths

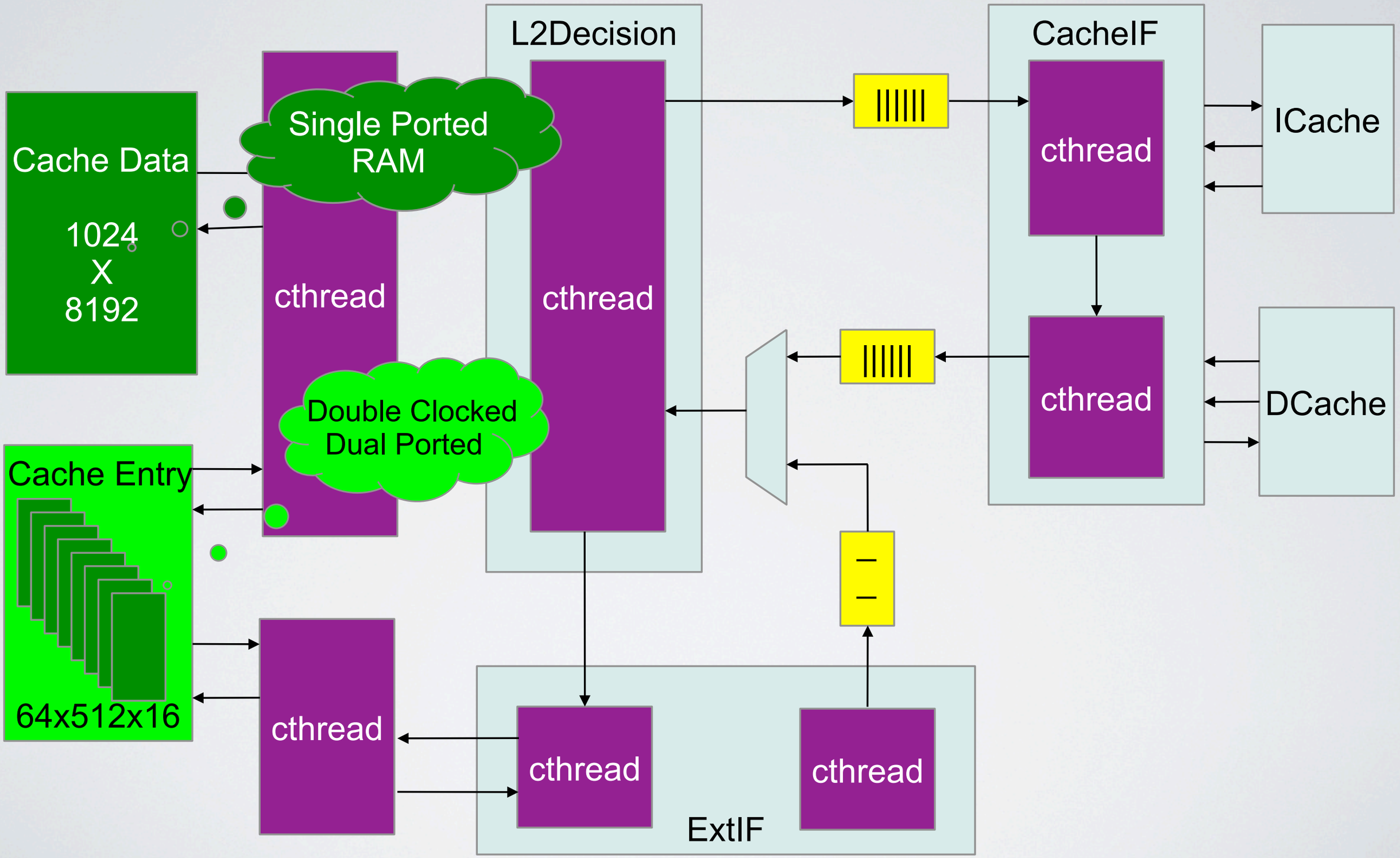
Allowed technology specific pin names which made mapping automatic

Allowed a vendor specific Verilog simulation model

Specialized Forte Memory wrappers could make a dual port out of single port by running the memory at 2x clock

Dual ported FIFO memories can also be built

Compiled Memory Usage



Why Compiled Memories

Forte requires one to map array usage to physical memory, which is why we use the memory compiler

The compiled memories have shared ports that can make them look like the memory is multiple ported

Forte provides the ability to run the memory at double and half speed from the compiler

Forte has interfaces for 2x and 4x memory to multiple the number of ports

Simple get() and put() methods to access memory as well as normal MEM[ADDR] form.

Clock Domain Crossing

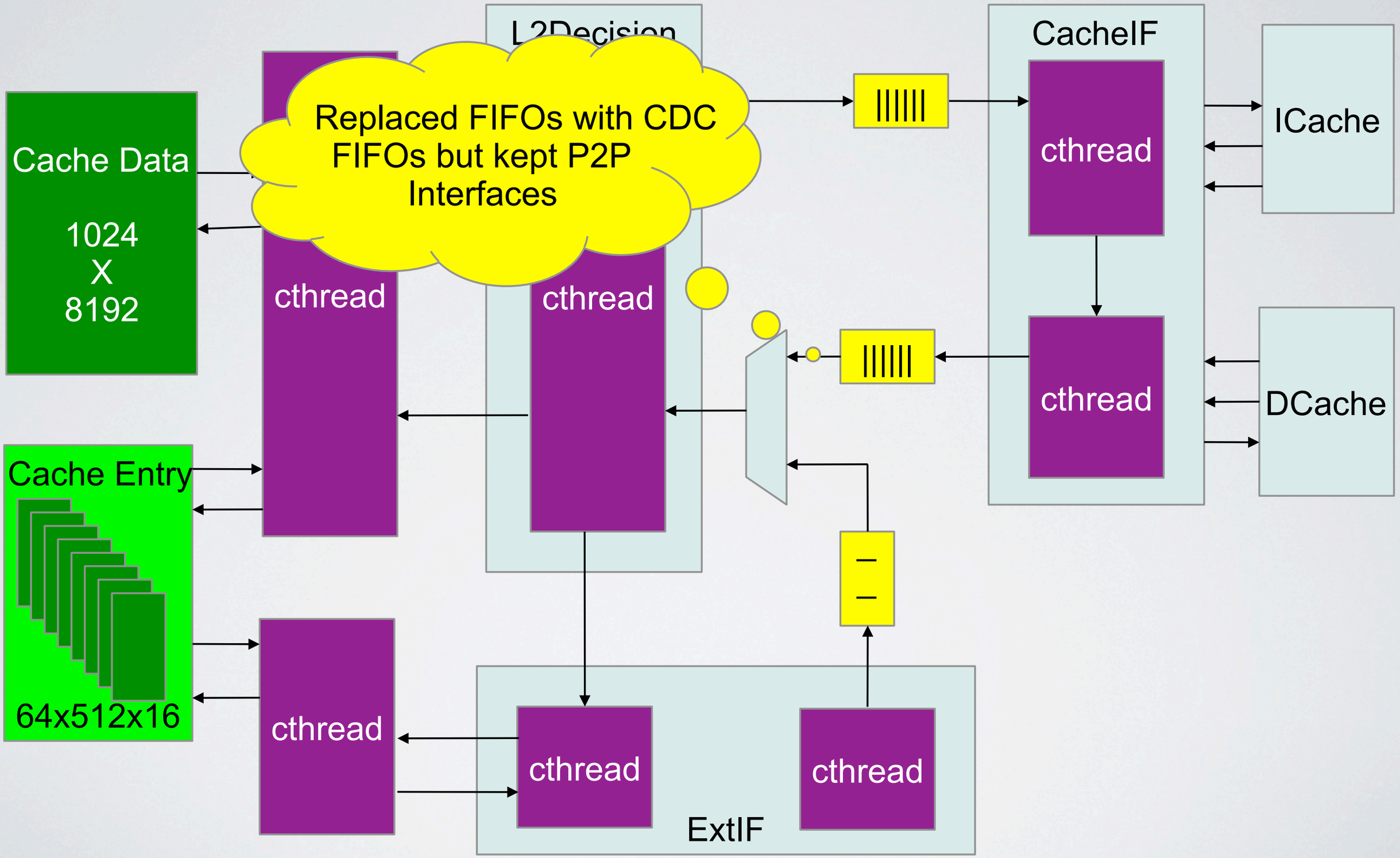
Forte has develop a set of IP for dealing with clock domain crossing

It looks exactly like the P2P from the standpoint of the usage model

Replaces P2P where-ever you need to cross clock domains

Can be simple register based or FIFO based

Clock Domain Crossing Usage



Suggested Improvements

Generalized SystemC improvement of template support was added in Forte release 4.2 with our encouragement

Forte Behavioral memories don't necessary map the foundry memory behavior so careful analysis is required when picking the attributes

Arrays of P2P are somewhat limited and we would like to exploit there power

Success

Using P2P allowed us to focus on our design not on interfaces

Using CDC allowed us to focus on how to use the data and not if we managed the clock domain crossing correctly

Testbenches were relatively easy to develop as a P2P port is driven via put() and received via get()

Our paper points to a 100-to-1 code reduction

Our paper points to a 10-to-1 test bench development reduction