ACKNOWLEDGEMENT

By utilizing this website and/or documentation, I hereby acknowledge as follows:

Effective October 1, 2012, QUALCOMM Incorporated completed a corporate reorganization in which the assets of certain of its businesses and groups, as well as the stock of certain of its direct and indirect subsidiaries, were contributed to Qualcomm Technologies, Inc. (QTI), a wholly-owned subsidiary of QUALCOMM Incorporated that was created for purposes of the reorganization.

Qualcomm Technology Licensing (QTL), the Company's patent licensing business, continues to be operated by QUALCOMM Incorporated, which continues to own the vast majority of the Company's patent portfolio. Substantially all of the Company's products and services businesses, including QCT, as well as substantially all of the Company's engineering, research and development functions, are now operated by QTI and its direct and indirect subsidiaries[1]. Neither QTI nor any of its subsidiaries has any right, power or authority to grant any licenses or other rights under or to any patents owned by QUALCOMM Incorporated.

No use of this website and/or documentation, including but not limited to the downloading of any software, programs, manuals or other materials of any kind or nature whatsoever, and no purchase or use of any products or services, grants any licenses or other rights, of any kind or nature whatsoever, under or to any patents owned by QUALCOMM Incorporated or any of its subsidiaries. A separate patent license or other similar patent-related agreement from QUALCOMM Incorporated is needed to make, have made, use, sell, import and dispose of any products or services that would infringe any patent owned by QUALCOMM Incorporated in the absence of the grant by QUALCOMM Incorporated of a patent license or other applicable rights under such patent.

Any copyright notice referencing QUALCOMM Incorporated, Qualcomm Incorporated, QUALCOMM Inc., Qualcomm Inc., Qualcomm or similar designation, and which is associated with any of the products or services businesses or the engineering, research or development groups which are now operated by QTI and its direct and indirect subsidiaries, should properly reference, and shall be read to reference, QTI.

[1] The products and services businesses, and the engineering, research and development groups, which are now operated by QTI and its subsidiaries include, but are not limited to, QCT, Qualcomm Mobile & Computing (QMC), Qualcomm Atheros (QCA), Qualcomm Internet Services (QIS), Qualcomm Government Technologies (QGOV), Corporate Research & Development, Qualcomm Corporate Engineering Services (QCES), Office of the Chief Technology Officer (OCTO), Office of the Chief Scientist (OCS), Corporate Technical Advisory Group, Global Market Development (GMD), Global Business Operations (GBO), Qualcomm Ventures, Qualcomm Life (QLife), Quest, Qualcomm Labs (QLabs), Snaptracs/QCS, Firethorn, Qualcomm MEMS Technologies (QMT), Pixtronix, Qualcomm Innovation Center (QuIC), Qualcomm iSkoot, Qualcomm Poole and Xiam.

Node Power Architecture (NPA) Overview

80-N1089-1 A

QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Revision History

Version	Date	Description
A	Apr 2010	Initial release

Contents

- Architecture
- NPA Extensions
- NPA Usage
- References
- Questions?

Architecture

Node Power Architecture (NPA)

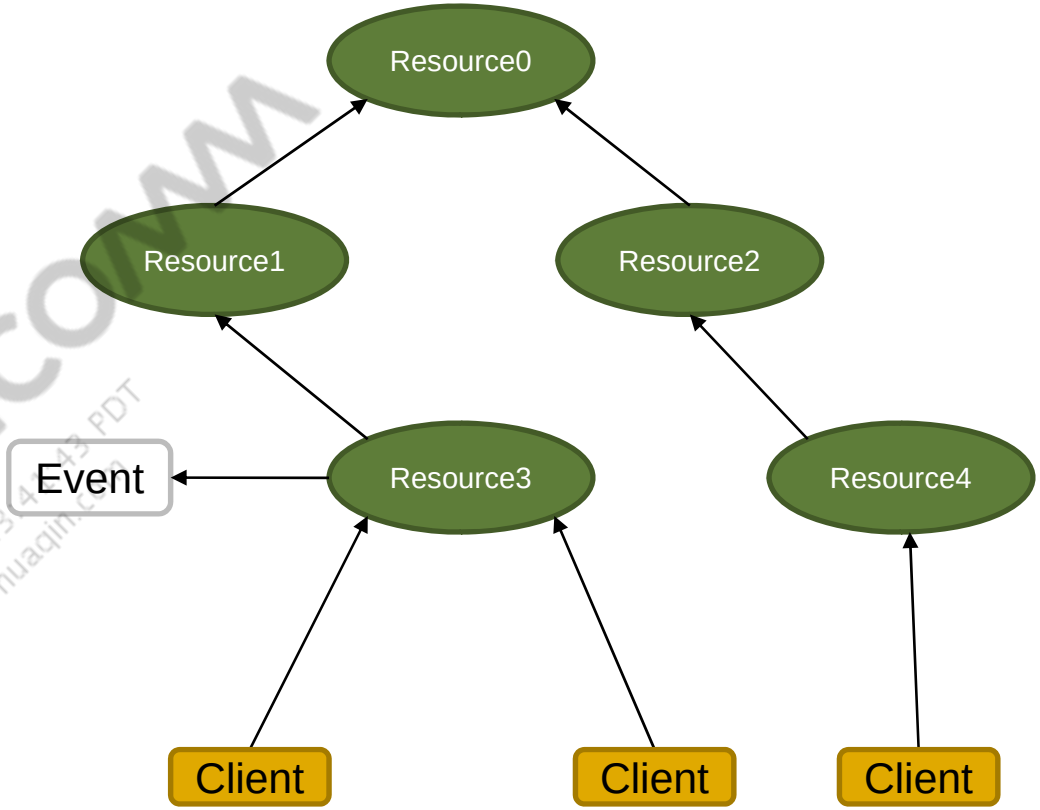
- Goal is finer-grained, more concurrency-aware resource management
  - Increase the ability of tasks to issue requests for their needs
    - Richer resource semantics
      - » Directly request what client needs (CPU processing), not side effect (CPU clock)
  - Maximize power decisions that can be made via resource requests
    - Optimize resource power consumption while meeting concurrent client requirements
    - Dependent resource requirements can be handled without client interaction
    - Optimize idle power management decisions
  - Improve visibility into resource management for:
    - Development/debugging
    - Target power optimization
    - Identification of resource oversubscription

NPA (cont.)

- NPA is the software framework developed to support these goals
  - Implements common resource management functionality
    - Client registration
      - » Memory management
    - Concurrency management
      - » Locking
      - » Standard request aggregation library (minimum, maximum, summary, etc.)
    - Event notification
    - Dependency management
      - » Enforces order of initialization
    - Logging
  - Provides common client interface
  - Allows construction of target-specific resource graph

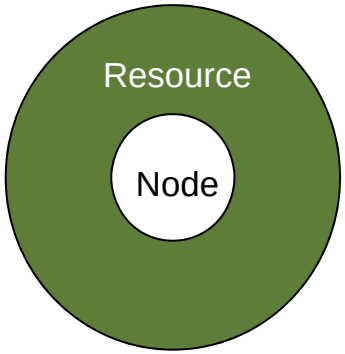
Resource Graph

- Resource graph is constructed at initialization time
- Clients and event handlers can be added dynamically
- Client requests are issued against resources
- Resource aggregates multiple concurrent requests to compute new resource state
- Resource updates/cancels dependency requests as needed to satisfy new resource state



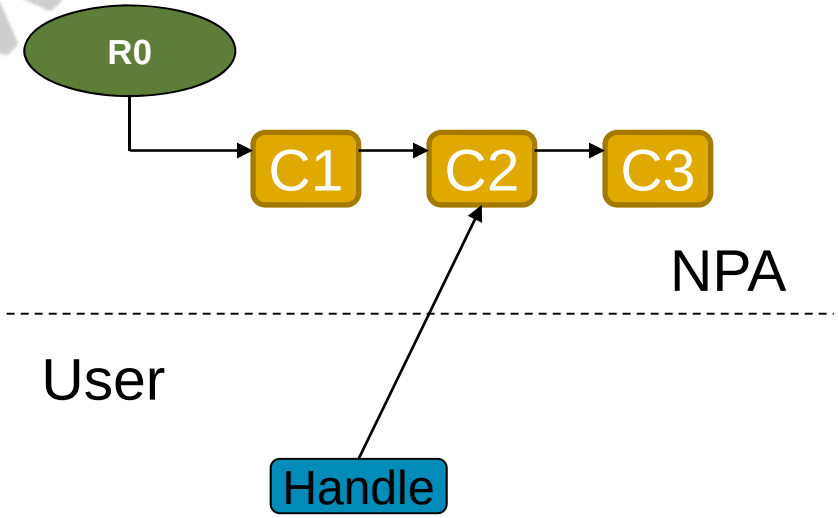
Resources

- Resources are the entities that clients issue work requests against.
  - All resource requests and updates are logged.
- The resource is responsible for meeting the various client requests and optimizing power consumption.
  - The function responsible for determining the correct level to meet the client requests is user-defined.
    - A standard library of common functions is provided.
  - The function responsible for updating the hardware and other resource dependencies is user-defined.
  - The framework guarantees atomicity.
- Resources define their own units (MIPS, MHz, MBps, etc.).
  - Work requests are made in these units.
- Resources are identified by name.
  - There is no compile-time linkage between the client and the resource.
  - The string lookup is resolved at client creation time; no request time overhead.
  - There can be multiple names (aliases) for the same resource.



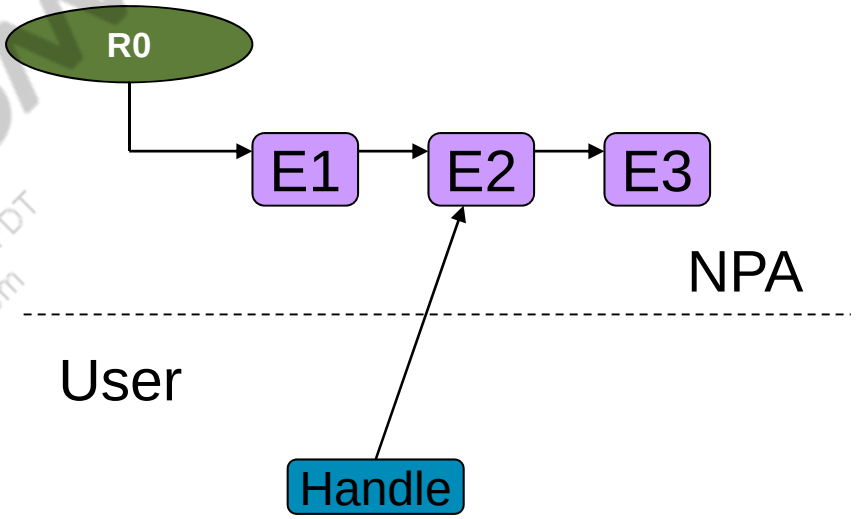
Clients

- Clients are used to issue work requests to resources
  - Clients may be synchronous or asynchronous
    - Sync clients do not return until request is completed
    - Async clients return immediately and invoke user callback when request is completed
  - Work requests may be richer than simply resource level
    - Required
    - Impulse
    - Isochronous
    - Other work models defined as necessary
  - Clients are identified by name
    - Client name used for logging purposes
- Clients are accessed via opaque handle
- Client requests can be derived from hardware feedback (CPU idle time, load monitoring, etc.) as well as software applications



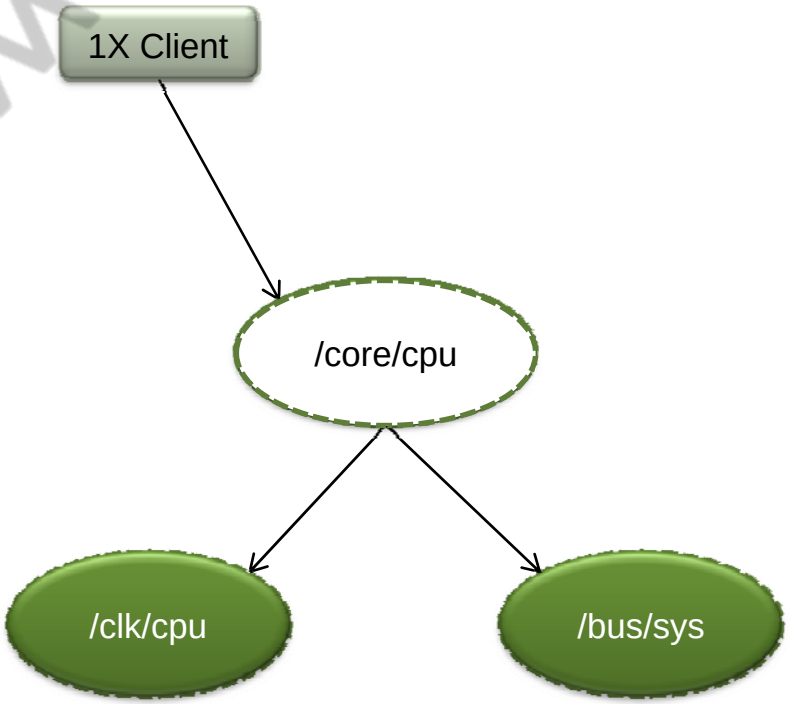
Events

- Events are the mechanism by which resources notify users of resource state changes
- Anyone may register an event
- Events are accessed via opaque handle
- Events can be triggered by:
  - High/low watermark transitions – Watermarks expressed as headroom
  - Any change in resource state – For profiling purposes
- Events can be used to monitor resource load or trigger other actions off resource state change
  - Reduce workload when resource is oversubscribed
  - Disable optional functions when resource goes away



Dependency Management (Animated)

- NPA provides mechanism for expressing/managing dependencies
  - Resources explicitly list which other resources they depend upon
  - Clients/others nonresources can register for event (callback/signal) when resource they require exists
  - Resources will only be created and initialized when all of their dependencies exist
  - External users can receive event when required resources become available
  - Definition and creation is logged



Aliases

- Resources can be accessed via multiple names/aliases
  - Aliases can be used to provide target independence
    - » On SCMM – /bus/arbiter → /bus/ahb/arbiter
    - » On MSM7x30 – /bus/arbiter → /bus/axi/arbiter
    - – Bus architecture and bus driver changed, but client interface remained consistent
  - Aliases can be used to provide alternate semantics on top of common driver
    - – Resource can determine what resource name the client used for creation
    - – Resource can use that information to enhance operation
      - » Can provide backwards compatibility to clients using legacy name
      - » Can provide enhanced operation to requests made via particular names, e.g., /bus/arbiter/apps could be aliased to /bus/arbiter; clients requests made via /bus/arbiter/apps alias get automatically enabled/disabled when apps power collapse occurs
  - Aliases can be created by resource author, users, or target team
  - Aliases are resolved at creation time
    - – No additional runtime overhead incurred by alias

Logging

- Each public NPA call is logged to common log
- Log uses names (strings)
  - Human readable
  - String expansion done in separate thread
- Logging done using ULog
  - Not NPA-specific
  - Being adopted by other projects
    - DCVS
    - MMPM
    - AMPPPS
- Available log tools
  - BREW®/Windows Mobile® app to write log to EFS
  - CMM scripts to extract logs from halted processor/core image
  - Coming soon – Diag access

```
0x00D4C3D9:  npa_define_resource
    (resource: "/clk/ahb") (initial_state: 0)
0x00D4C3DB:  end npa_define_resource
    (resource: "/clk/ahb") (initial_state: 0)
0x00DF355C:  npa_create_sync_client
    (resource: "/clk/ahb") (client:"foo")
        (type: NPA_REQUEST_REQUIRED)
0x00DF355D:  end npa_create_sync_client
    (resource: "/clk/ahb") (client:"foo")
        (type: NPA_REQUEST_REQUIRED)
0x00DF3565:  npa_issue_required_request
    (client: "foo") (request: 1)
    (resource: "/clk/ahb")
0x00DF3565:  end npa_request
    (client: "foo") (resource: "/clk/ahb")
        (request_state: 100) (active_state:
    100)
0x00DF3566:  npa_complete_request
    (client: "foo") (resource: "/clk/ahb")
0x00DF3566:  end npa_request
    (client: "foo") (resource: "/clk/ahb")
        (request_state: 0) (active_state: 0)
```

NPA Data Dump

- Retrieve entire NPA Graph state via Trace32
  - Including all resource states and all client requests
  - Has proven very useful for debugging power and performance issues
- Example
  - Why is the CPU currently running at maximum?
  - Because USB is requesting maximum

```
0x75D2C3: npa_resource (name: /core/cpu)
    (units: MIPS)  (resource max: 480)
    (active max: 480)
    (active state: 480) (active headroom:
    0)
    (request state: 480)
0x75D2C3:   npa_client (name:
    GPS_CPU_CLIENT)
    (type: NPA_CLIENT_REQUIRED) (request:
    0)
0x75D2C3:   npa_client (name:
    modem/cdma/1x/srch)     (type:
    NPA_CLIENT_REQUIRED) (request: 280)
0x75D2C3:   npa_client (name:
    /core/wiredconnectivity/hsusb)   (type:
    NPA_CLIENT_REQUIRED) (request: 480)
0x75D2C3:   npa_client (name: HDR)
    (type: NPA_CLIENT_REQUIRED) (request:
    0)
0x75D2C3:   npa_client (name: audio)
    (type: NPA_CLIENT_REQUIRED) (request:
    0)
0x75D2C3:   npa_client (name: CPU Dynamics
    Timer)
    (type: NPA_CLIENT_RESERVED2) (request:
    0)
0x75D2C3:   end npa_resource
```

Memory Management

- NPA in detail –NPA structures are created dynamically at runtime
  - Fixed-size memory pool allocator used to prevent fragmentation, minimize allocation costs
    - Pools are initialized with user-allocated memory
    - Pools can optionally be extended via heap allocations
      - Heap support not required
    - Memory added to pools are never freed

NPA Extensions

NPA Extensions

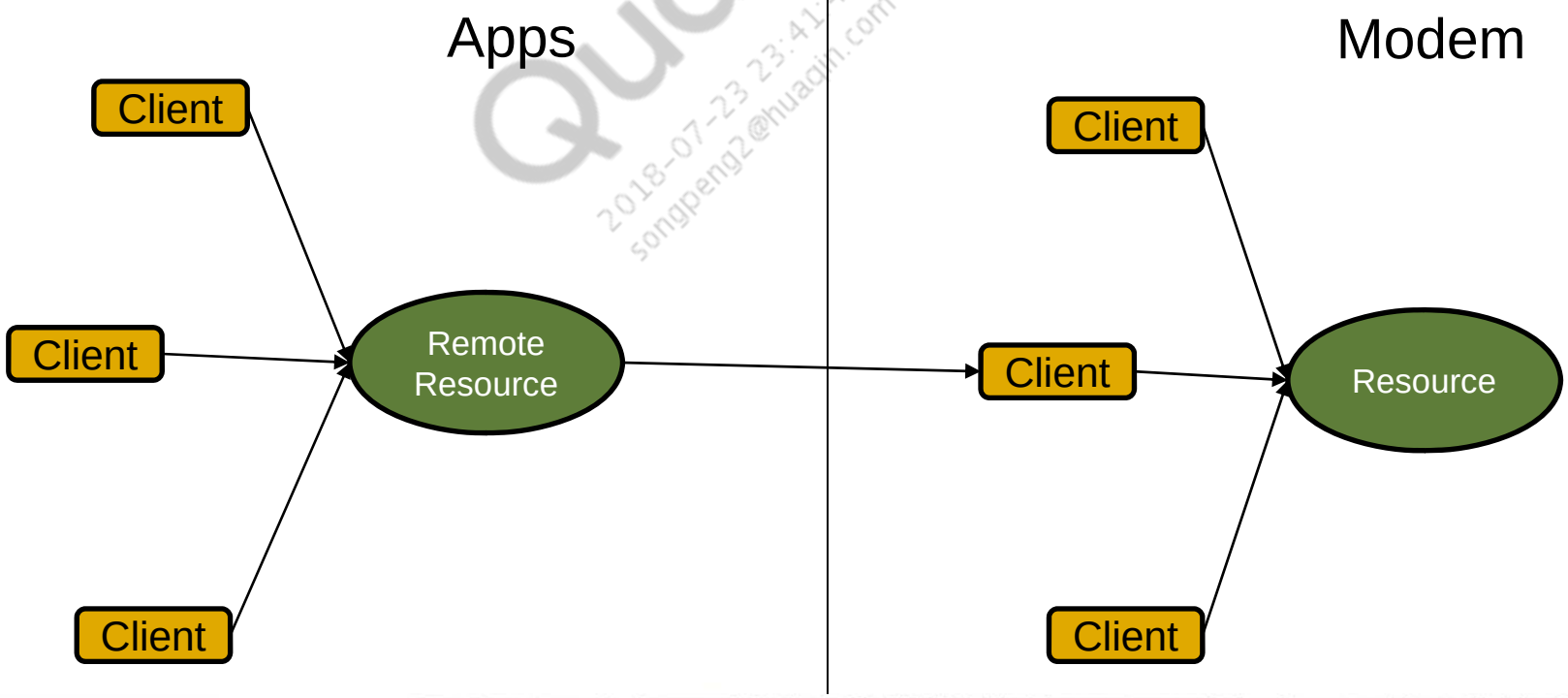
- The NPA framework itself is fairly thin and general purpose
  - Design intent is for users to extend for particular use case – Framework structures can be viewed as base classes
- There have been opportunities to leverage common use cases into additional libraries built on top of NPA
  - NPA remoting – Provide ability to issue requests to nonlocal resources
    - Modem on MSM7x30
    - RPM on MSM8660™
  - User space access – Provide ability for user space to make resource requests to selected resources without resource author having to write user space driver
- Also, there are opportunities to leverage the NPA data structures and request information to extend functionality
  - NPA sleep
  - Topology independent bus requests
  - Automated startup sequencing

NPA Remoting

- Clients always interact with resources that are local – Some resources are actually controlled on remote processor
- NPA remoting is a library of update and driver functions that handle forwarding requests to NPA resources on remote processor
  - NPA remoting is transport-independent
    - Multiple transports and transport versions can be supported simultaneously
      - RPC and QMI could coexist
      - Resource author's choice as to which transport to use
    - Remoting not exposed to clients
      - All access appears local
      - Client code does not need to change if resource control is moved on future target
  - NPA remoting implemented outside of framework
    - Intended as option for resource authors, but not required
    - Resource authors always retain option to use alternate mechanism

NPA Remoting (cont.)

- Clients always interact with a local resource
- Resources always take input via clients
- A resource may, as part of its update and/or driver, issue a request via a client on another processor
  - Remote request mechanism is transport agnostic – Currently, RPC is implemented



Remote Exception Handling

- Distributed requests introduce the possibility for exceptions, e. g., modem restart
- Framework/remote protocol identifies exception – Lock all resources
- NPA client/resource structures maintain state – Requests can be reissued transparently to users

NPA Sleep

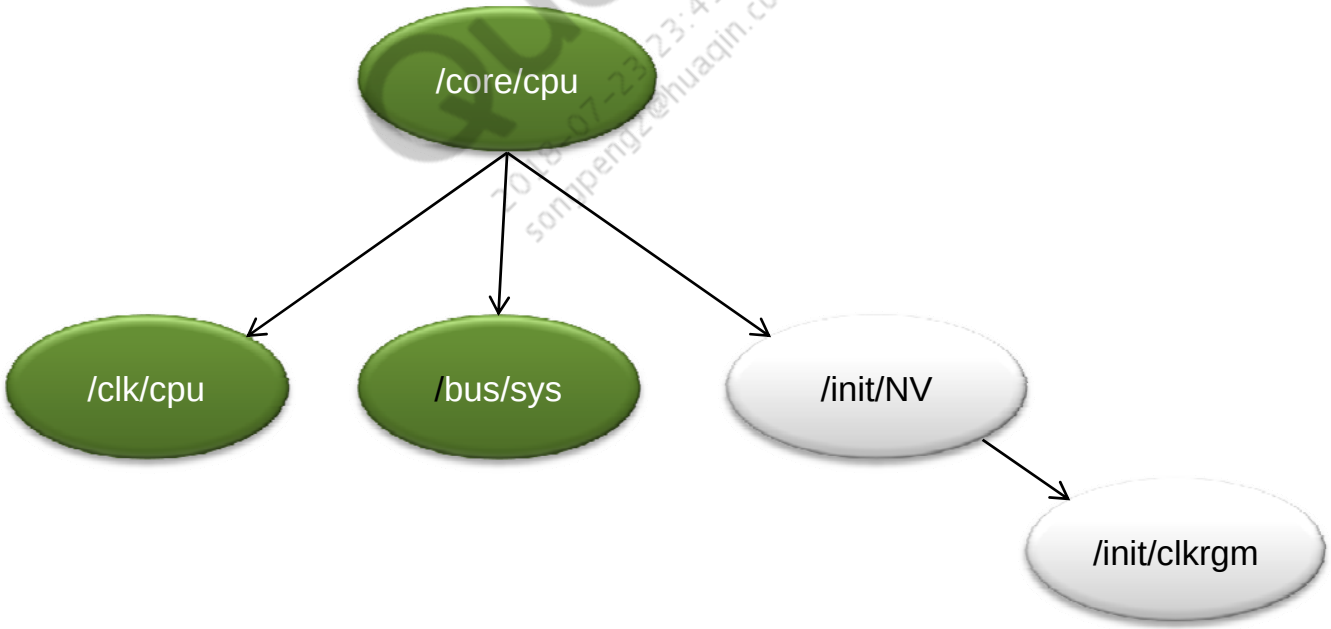
- Clients issue requests for resources they need and cancel requests when the resource is no longer in use
- Typically, when resource is unused, it can be disabled
  - However, not true for resources, CPU is also using memory, VDD, XO
- Those resources cannot be shut down until CPU goes idle
  - These resources are called Low Power Resources (LPR)
- Sleep process will use the LPR states to determine which low power modes are enabled/disabled and dynamically choose the best set based on required latency and expected sleep time

Topology-Independent Bus Requests

- Under development
- Goal is for client request to specify bus endpoints without regard to underlying bus topology
- Bus resource leverages connectivity information in NPA resource graph to determine route ability
- Clients become isolated from the particular topology on a given target
  - Aliases can also be used to remap endpoints

System Startup

- Subsystems could define logical /init resources to allow other subsystems to express initialization dependencies and allow NPA to properly sequence system startup
  - No requirement that subsystem support an NPA-style request interface
- Subsystems could in turn leverage NPA dependency management and initialization sequencing for their own dependencies



NPA Usage

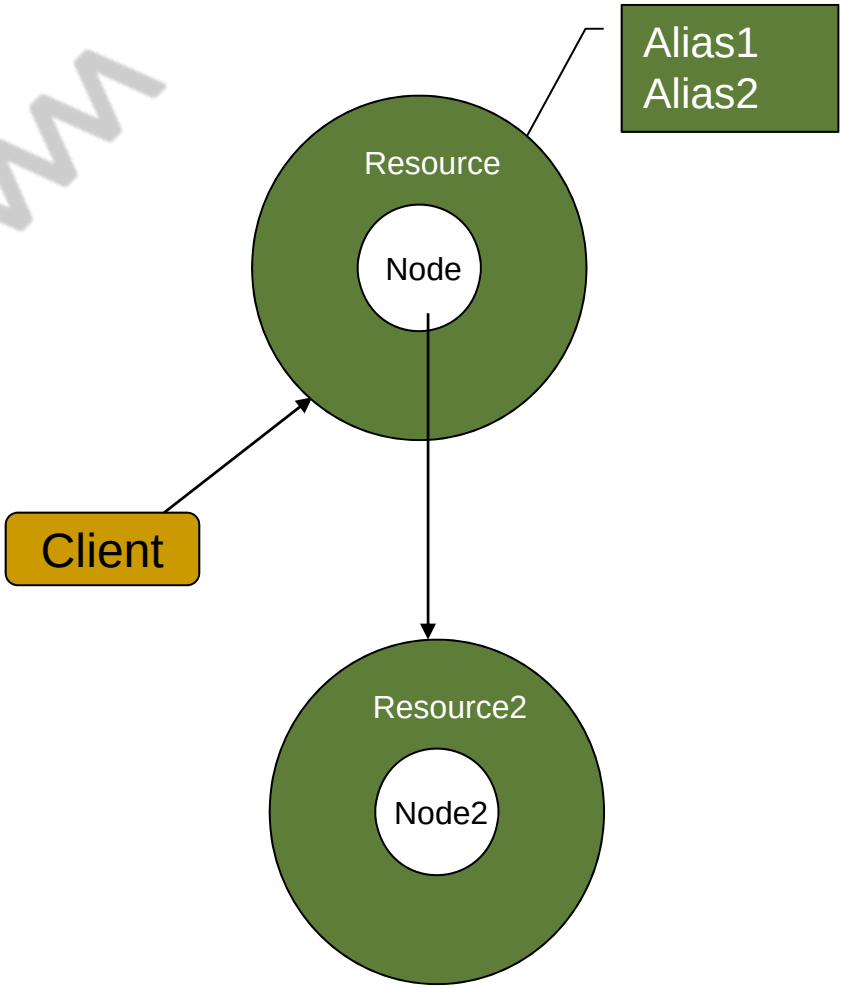
Client Creation and Use

- Clients are created with the create client function; takes the arguments:
  - Resource name
  - Client name
  - Client work model
- Returns a handle to the resource
  Note: Client must check return value.
- Issues work request
  Note: Client request API may vary with client work model
- Cancel work request

```
/* create a handle to the /core/cpu resource
    */
npa_resource_handle cpu_hdl =
    npa_create_sync_client(
    "/core/cpu",          /* resource name */
    "client_name",        /* client name   */
    NPA_CLIENT_REQUIRED /* work model    */
);

/* Check to see handle was created */
if ( cpu_hdl == NULL )
{
  ERR_FATAL( "resource unavailable" );
}

/* Issue a work request of 100 MIPS to the
    CPU */
npa_issue_required_request( cpu_hdl, 100 );

/* cancel work request */
npa_cancel_request( cpu_hdl );
```

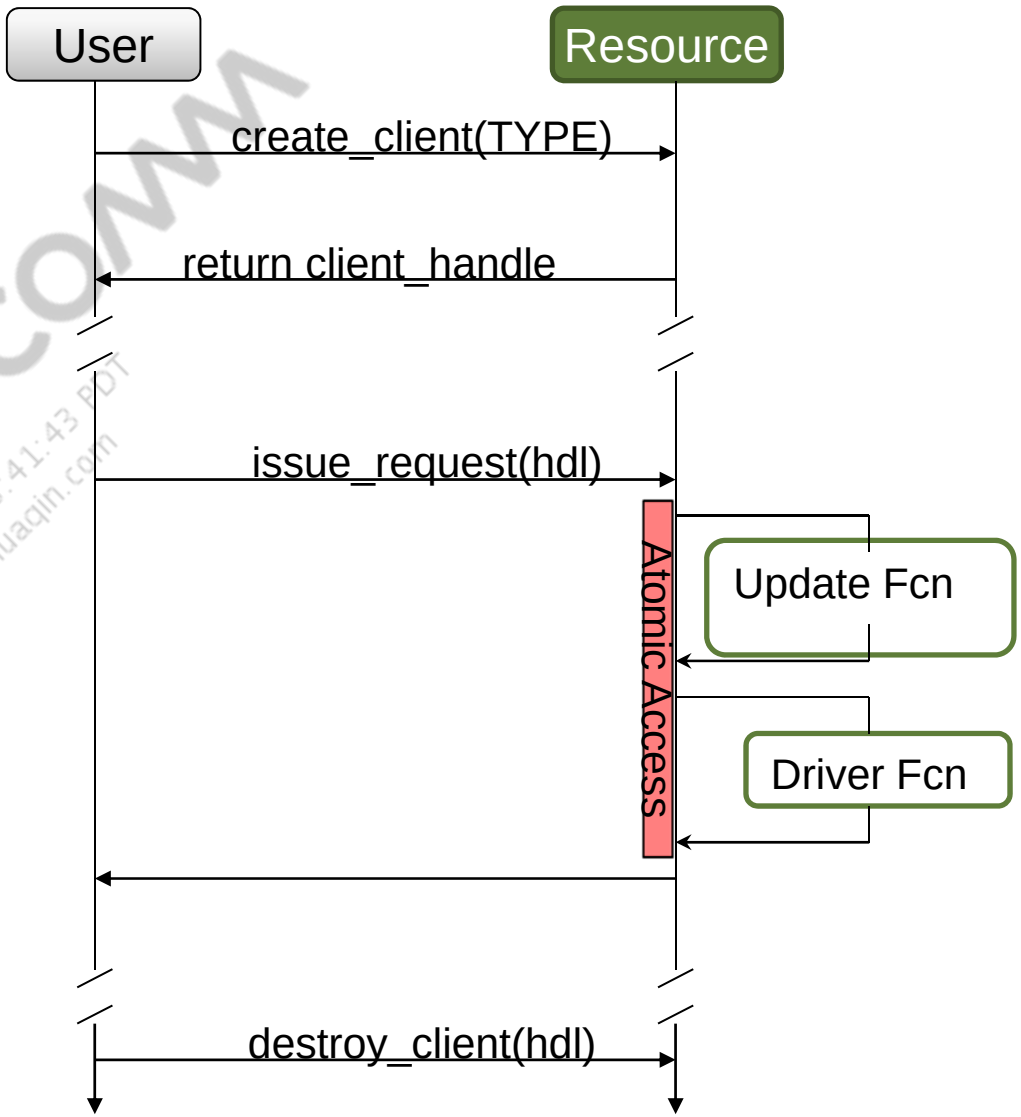
NPA Node Definition and Creation

- Resources are identified by name
  - A resource can have one or more aliases
    - Aliased resources are transparent to the client
- Users register clients with resources to issue requests to the resource
  - Client registration happens at runtime
  - A resource can support multiple client request types
    - Each client type is a potentially different interface
    - Some default client types are defined
- Nodes can have clients of other resources
  - These are the node dependencies



NPA Interact

- Users create clients of a particular type and receive a client handle
- Users use that handle to issue requests to the resource
  - Requests are processed atomically in two stages
    - Update function – Aggregate the new request with any outstanding client requests to determine new resource state
    - Driver function – Apply new state to the resource
- When user's need for the resource finishes, the user cancels the request
- User can destroy the client when it no longer needs to issue requests



Node Definition and Creation

- Nodes are defined via a set of structures
  - Dependency array
    - Dependency name
    - Client type
  - Resource definition array
    - Resource name
    - Maximum value
    - Resource attributes
    - Plugin – Typically from library
    - User data

```
/* Dependency array */
static npa_resource_dependency
  core_cpu_deps[] =
{
  {"/clk/cpu", NPA_CLIENT_REQUIRED},
   {"/bus/ahb", NPA_CLIENT_REQUIRED}
};
/* Resource array */
static npa_resource_definition
    core_cpu_resources[] =
{
  {
    "/core/cpu",            /* Name */
    "MIPS",                 /* Units */
    512,                    /* Max State */
    &npa_sum_plugin,        /* Plugin */
    NPA_RESOURCE_DEFAULT,   /* Attributes */
    NULL,                   /* User Data */
    NULL                    /* Query Function
    */
  }
};
```

Node Definition and Creation (cont.)

- Nodes are defined via a set of structures
  - Node structure
    - Name
    - Driver function
    - Attributes
    - User data
      - User data field is void *
      - Used to hold arbitrary data
    - Dependency array
    - Resource array
      - NPA_ARRAY macro computes array size
      - NPA_EMPTY_ARRAY macro can be used for empty array

```
/* Node definition */
static npa_node_definition
   core_cpu_node =
{
  "/node/core/cpu",     /* Name */
  core_cpu_driver_fcn, /* Driver
   Function */
  NPA_NODE_DEFAULT,     /* Node
   Attributes */
  NULL,                 /* Node User
   Data */
  NPA_ARRAY(core_cpu_deps),  /*
   Dependencies */
  NPA_ARRAY(core_cpu_resources) /*
   Resources */
};
```

Node Definition and Creation (cont.)

- Node creation function
  - npa_define_node( );
- Takes as arguments:
  - Node definition structure
  - Initial resource state array
  - Optional node creation event
- When all node dependencies exist:
  - Creates resources
  - Creates clients for each dependency
  - Invokes driver function with initial values
  - Adds resources to graph
  - Triggers node creation event

```
/* add node to graph with initial value
    of 128 */
npa_resource_state init_state = 128;

npa_define_node(
    &core_cpu_defn,
    &init_state,
    NULL );

/* Node Creation Function will return
    immediately without creating anything
    if node dependencies are not present.
   In this case, creation will be
    deferred until all the dependencies
    are present.
   This means nodes can be defined in any
    order.
   Node Creation Event can be used to
    synchronize on actual node creation.
*/
```

NPA Node Customization

- A node author can customize the following resource interactions
  - Resource state change – Driver function
  - Client request aggregation – Update function
  - Client creation/destruction
  - Client request type
  - User queries
- Typically, node author only needs to define the driver function

Driver Function

- Driver function
  - Takes resource and desired state as arguments – Framework clips state to resource maximum
  - Issues requests to dependencies
    - Typically a function of state
    - Dependencies are indexed from the dependency array that was part of the resource definition
      - NPA_DEPENDENCY () is a helper macro for indexing array
  - Issues request to hardware
    - May not be necessary (i.e., a logical node)
  - Returns actual state set

Note: Typically, this is the only function a node creator will have to write.

```
/* Core Cpu Driver Function
   This is a logical node – no direct HW.
      It computes state in MIPS and
    derives:
     1) frequency request to CPU clock
     2) BW request to bus */
static npa_resource_state
    core_cpu_driver_fcn
  ( npa_resource       *resource,
      npa_client          *client
    npa_resource_state  state )
{
  /* issue request requests */
    npa_issue_required_request(
      NPA_DEPENDENCY( resource, 0 ),
    mips2freq(state) );
  npa_issue_required_request(
    NPA_DEPENDENCY( resource, 1 ),
    mips2bw(state) );

  return freq2mips(
    npa_get_state(NPA_DEPENDENCY(resource, 0
    ));
}
```

Resource Initialization

- At resource creation time
  - Driver function is invoked with an NPA_CLIENT_ INITIALIZE client
  - Request state is the initial state as passed in to npa_define_node()

```
/* Example Driver Function */
static npa_resource_state driver_fcn
  ( npa_resource        *resource,
     npa_client           *client
    npa_resource_state  state )
{
    if ( client->type ==
    NPA_CLIENT_INITIALIZE )
    {
      // Perform initialization
      // Init(state)
    }
    else
    {
      // Do normal processing
    }
}
```

Standard Plug-ins

- The following standard plug-ins are defined in npa_resource.h
  - Binary plug-in – Request state is on (maximum) or off
  - Max plug-in – Request state is maximum of concurrent requests
  - Min plug-in – Request state is minimum of concurrent requests
  - Sum plug-in – Request state is sum of concurrent requests
  - Identity plug-in – Request state is the client's request
  - Always-on plug-in – Request state is always maximum, irrespective of client requests

```
npa_resource_plugin npa_binary_plugin;
npa_resource_plugin npa_max_plugin;
npa_resource_plugin npa_min_plugin;
npa_resource_plugin npa_sum_plugin;
npa_resource_plugin npa_identity_plugin;
npa_resource_plugin npa_always_on_plugin;
```

References

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1

Questions?

https://support.cdmatech.com