Software Development Kit – P4 and C Development Toolchain

Dataplane Acceleration Developer Day (DXDD)
Nov. 2016

[redacted]

Agenda

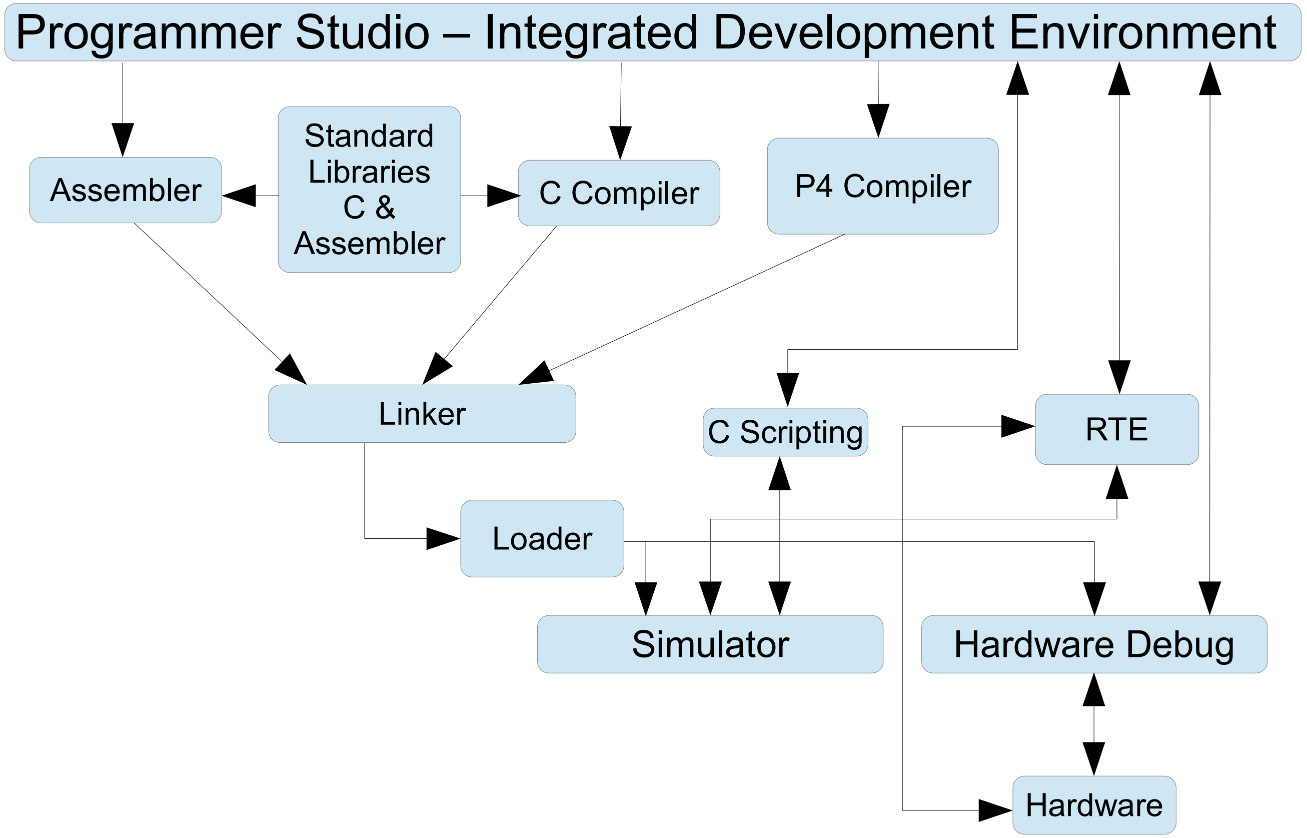
- Software Development Kit (SDK) Overview
- Toolchain Theory of Operation
- Debugging using Simulator and Hardware (SmartNIC)

Software Development Kit (SDK) Overview

- Integrated Development Environment, running on Windows (natively, in a VM, or in WINE)
- Complete package for SmartNIC application development: edit, build, debug, optimize …
- Supports data plane programming using P4 and C
- Supports multiple development platforms
  - ✓ Cycle accurate simulator for SmartNIC's Network Flow Processor
  - ✓ Enables remote debugging of Agilio SmartNICs in Linux servers
- Includes documentation (in PDF and HTML formats)

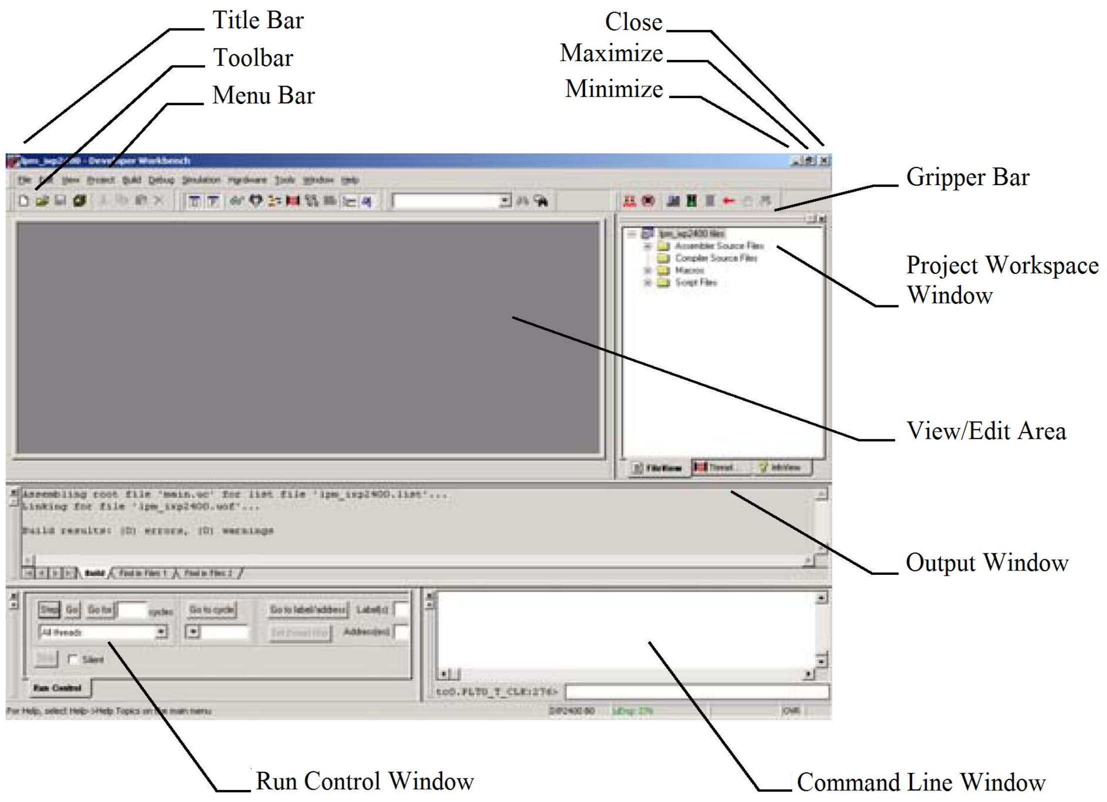
SDK Components

- Programmer Studio (GUI)
- Assembler
- C Compiler
- P4 Compiler
- Linker/Loader
- Simulator
- C Scripting - Cling
- Standard Library
- Run Time Environment



Programmer Studio IDE Components

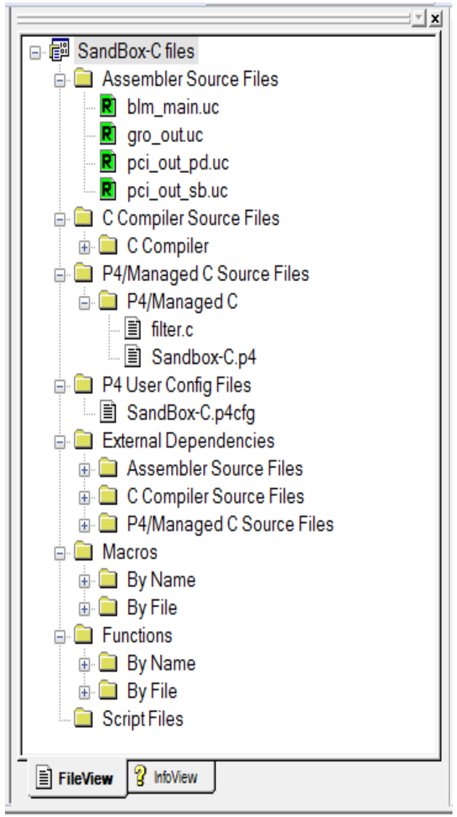
- Integrated Development Environment (IDE)
- Ability to manage ongoing development by organizing settings and files into projects
- Project types
  - C (Standard / Debug Only)
  - P4
- Two sets of toolbar and docking configurations:
  - Debug Mode
  - Build/Edit Mode



```
Assembling root file 'main.uc' for list file 'ipm_ixp2400.list'...
Linking for file 'ipm_ixp2400.uof'...

Build results: (0) errors, (0) warnings
```

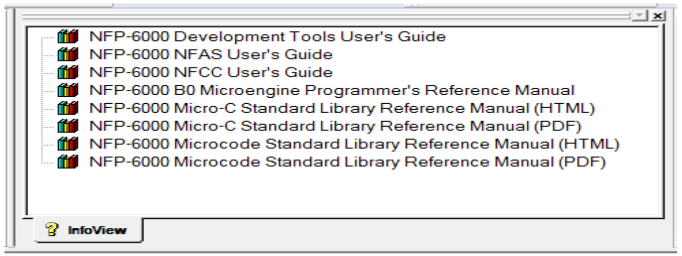
Programmer Studio Views

- The Project Workspace is a dockable window where you access and modify project files, view threads during debugging, and view documentation in PDF and HTML format, using tabs:
  - FileView
  - ThreadView
  - InfoView



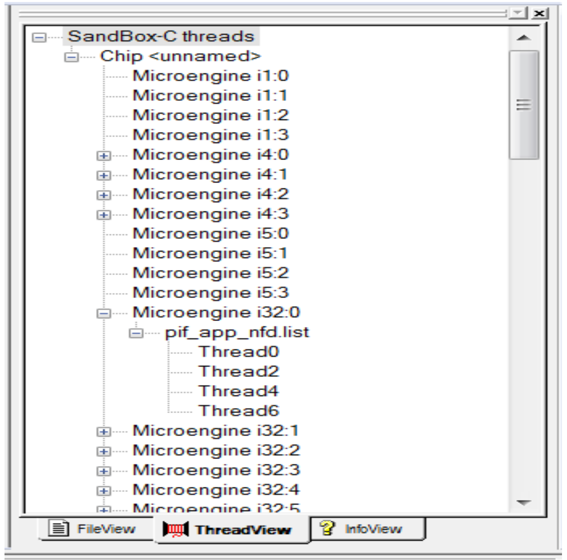
```
SandBox-C files
  Assembler Source Files
    blm_main.uc
    gro_out.uc
    pci_out_pd.uc
    pci_out_sb.uc
  C Compiler Source Files
    C Compiler
  P4/Managed C Source Files
    P4/Managed C
      filter.c
      Sandbox-C.p4
  P4 User Config Files
    SandBox-C.p4cfg
  External Dependencies
    Assembler Source Files
    C Compiler Source Files
    P4/Managed C Source Files
  Macros
    By Name
    By File
  Functions
    By Name
    By File
  Script Files
```

FileView



```
NFP-6000 Development Tools User's Guide
NFP-6000 NFAS User's Guide
NFP-6000 NFCC User's Guide
NFP-6000 B0 Microengine Programmer's Reference Manual
NFP-6000 Micro-C Standard Library Reference Manual (HTML)
NFP-6000 Micro-C Standard Library Reference Manual (PDF)
NFP-6000 Microcode Standard Library Reference Manual (HTML)
NFP-6000 Microcode Standard Library Reference Manual (PDF)
```

InfoView



```
SandBox-C threads
  Chip <unnamed>
    Microengine i1:0
    Microengine i1:1
    Microengine i1:2
    Microengine i1:3
    Microengine i4:0
    Microengine i4:1
    Microengine i4:2
    Microengine i4:3
    Microengine i5:0
    Microengine i5:1
    Microengine i5:2
    Microengine i5:3
    Microengine i32:0
      pif_app_nfd.list
        Thread0
        Thread2
        Thread4
        Thread6
    Microengine i32:1
    Microengine i32:2
    Microengine i32:3
    Microengine i32:4
    Microengine i32:5
```

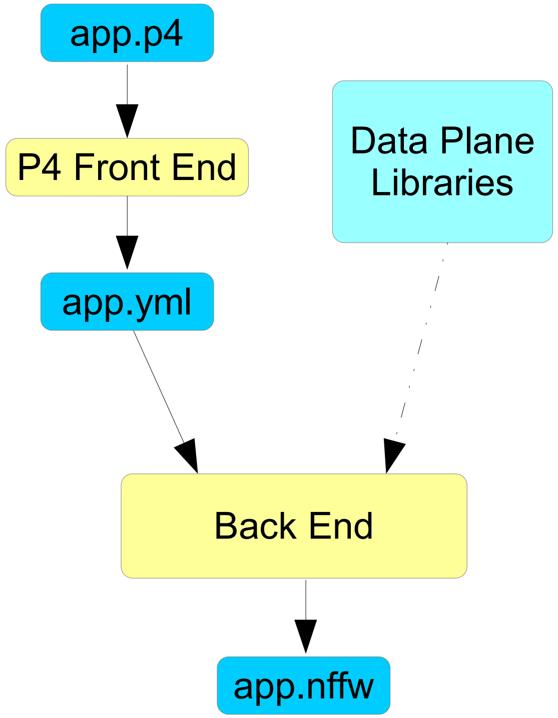
ThreadView

C Compiler Details

- Accepts standard C, augmented with pragmas and specifiers, e.g. __declspec() to explicity specify memory types (DRAM vs. on-chip memory) and properties (e.g. thread local or global).
- Accepts in-line assembly via __asm{ } statement.
- Optimizes program in “whole program mode”, in-lining functions and specializing data types based on the context in which they are used.
- Generates a .list file (effectively a binary code object file) for each microengine (flow processing core).

P4 Compilation Details

- Front end passes take P4 source and compiles it to Intermediate Representation (IR) in YAML format
  - Languages other than P4 can be supported in future
  - IR standardized at opensourcesdn.org
- IR can be displayed as graphs (parser, ingress control flow, egress control flow)
- Back end passes compile IR to firmware (native code for SmartNIC – ELF file)
- Code leverages Data Plane Libraries provided by Netronome for microflow caching, packet classification, PCIe and network I/O, packet re-ordering etc.



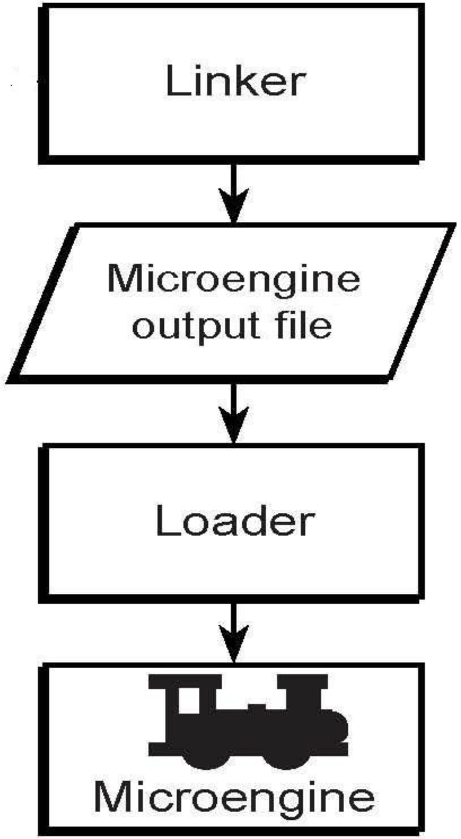
Standard Libraries / Components

- C libraries are supplied to enable convenient access to the Network Flow Processor's features, for example packet I/O, buffer allocation/freeing, and function accelerators (e.g. ring put/get, statistics, load balancing, hashing, metering, individual lookup operations, etc.)
- Larger Standard Components deliver functionality like packet reordering, PCIe NIC functionality, flow caching, algorithmic classification, etc.

Linker / Loader Details

Loader operations:

- Read NFFW file (object file) headers
- Verify NFFW file (object file) is valid for target
- Perform relocation and resolve symbols (including import variables)
- Attempt clean interruption/stopping of hardware engines to be loaded, or reset the islands
- Set and verify CSRs
- Load memory sections (excluding code sections)
- Load initialization code sections and execute them
- Load code sections
- Trigger “new firmware” event on host



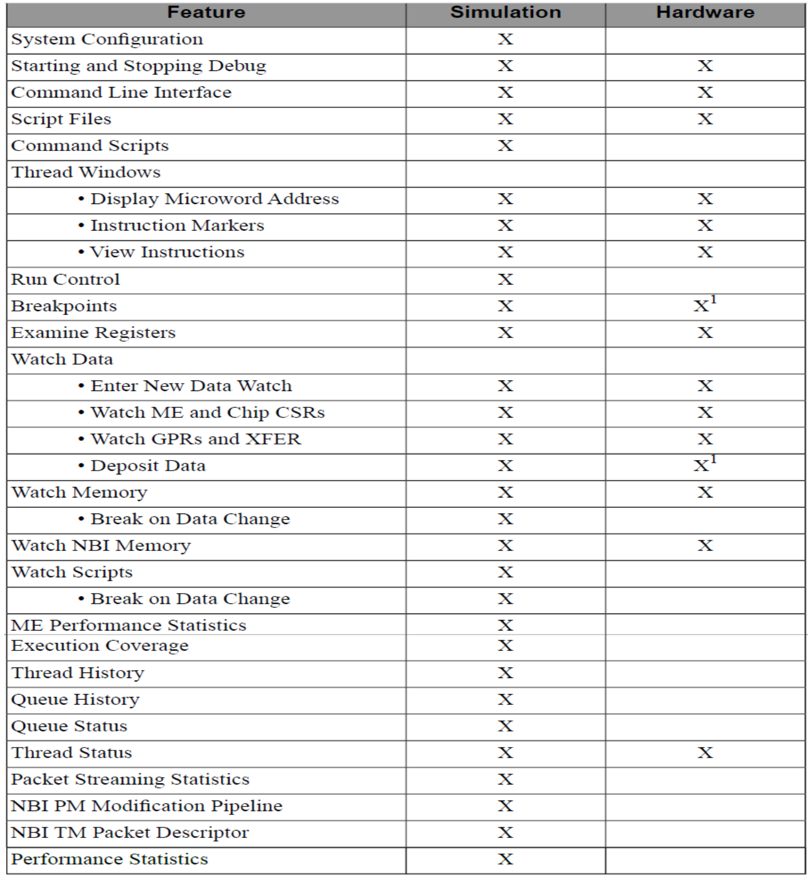
Debugger Support

The Programmer Studio IDE supports debugging in three different configurations:

- **Local simulation (default):** Programmer Studio and the Network Flow Processor simulator both run on the same Microsoft Windows platform.
- **Remote simulation:** Programmer Studio runs on Windows, communicating over a network with a separate Network Flow Processor simulator process (running on Windows or Linux).
- **Hardware:** Programmer Studio runs on Windows, communicating over a network with a Linux server containing a SmartNIC.

Debugging Features – Simulation / Hardware

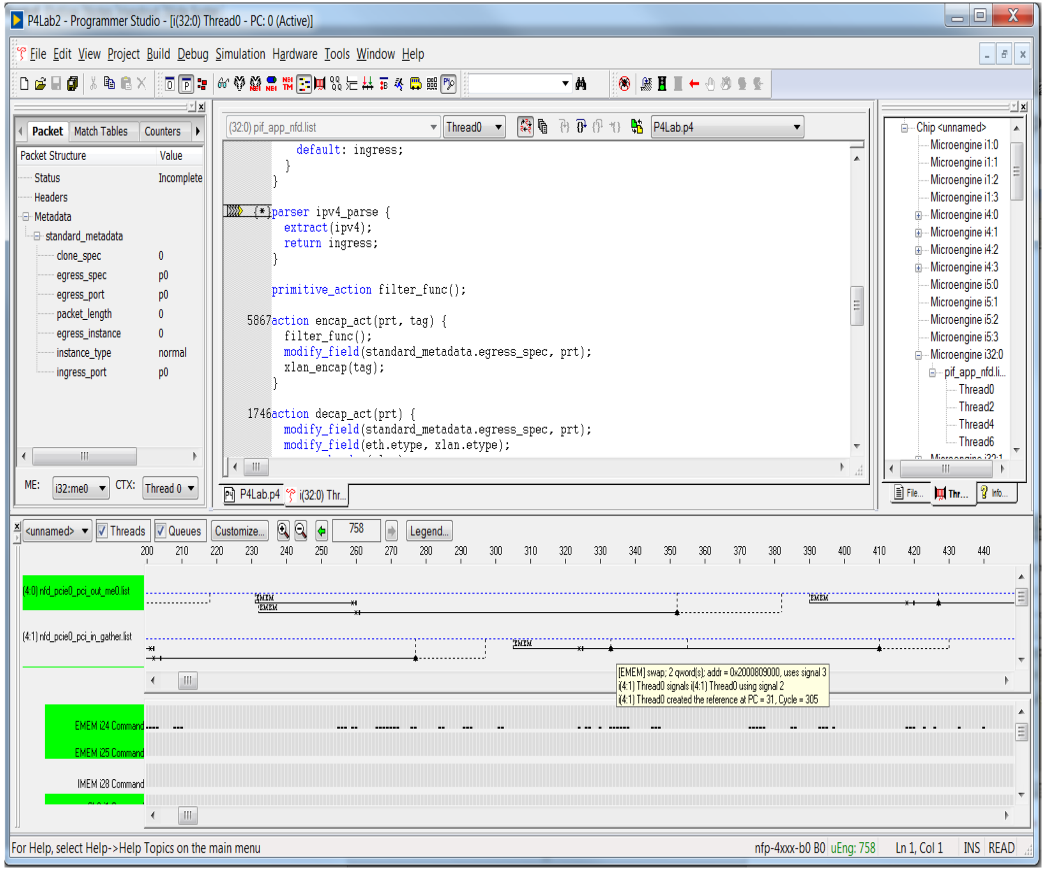
- More debugging features available when simulating NFP than running application on hardware
  - Execution stage marking in thread windows
  - Code execution coverage
  - Command execution history
- Running application is faster on hardware than on simulator



Feature	Simulation	Hardware
System Configuration	X	
Starting and Stopping Debug	X	X
Command Line Interface	X	X
Script Files	X	X
Command Scripts	X	
Thread Windows		
• Display Microword Address	X	X
• Instruction Markers	X	X
• View Instructions	X	X
Run Control	X	
Breakpoints	X	X[1]
Examine Registers	X	X
Watch Data		
• Enter New Data Watch	X	X
• Watch ME and Chip CSRs	X	X
• Watch GPRs and XFER	X	X
• Deposit Data	X	X[1]
Watch Memory	X	X
• Break on Data Change	X	
Watch NBI Memory	X	X
Watch Scripts	X	
• Break on Data Change	X	
ME Performance Statistics	X	
Execution Coverage	X	
Thread History	X	
Queue History	X	
Queue Status	X	
Thread Status	X	X
Packet Streaming Statistics	X	
NBI PM Modification Pipeline	X	
NBI TM Packet Descriptor	X	
Performance Statistics	X	

Network Flow Processor Simulator

- Provides cycle-accurate simulation for all data-plane chip functionality
- Advanced simulation, profiling, and debugging capabilities within IDE
- Rapid prototyping and intuitive optimization of user applications
- Support for parallel software and hardware engineering efforts

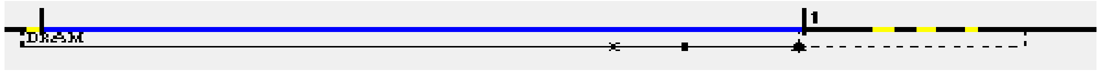


Packet Structure	Value
Status	Incomplete
Headers	
Metadata	
standard_metadata	
clone_spec	0
egress_spec	p0
egress_port	p0
packet_length	0
egress_instance	0
instance_type	normal
ingress_port	p0

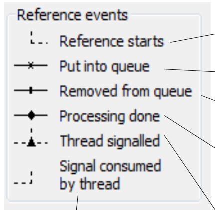
```
    default: ingress;
  }
}

{*}parser ipv4_parse {
  extract(ipv4);
  return ingress;
}

primitive_action filter_func();

5867action encap_act(prt, tag) {
  filter_func();
  modify_field(standard_metadata.egress_spec, prt);
  xlan_encap(tag);
}

1746action decap_act(prt) {
  modify_field(standard_metadata.egress_spec, prt);
  modify_field(eth.etype, xlan.etype);
```

```
[EMEM] swap; 2 qword(s); addr = 0x2000809000, uses signal 3
i(4:1) Thread0 signals i(4:1) Thread0 using signal 2
i(4:1) Thread0 created the reference at PC = 31, Cycle = 305
```

Simulator – History Collection



- Thread history – tracks references that are generated by execution of instructions
- Queue history – tracks commands issued on internal buses (to/from function accelerators, memory, or I/O peripherals)
- History data is collected from:
  - Event bus
  - Local CPP (Command Push Pull) bus
  - DSF CPP (Distributed Switch Fabric CPP)
- Benefits
  - High level view of microengine execution
  - Quickly and easily locate performance bottlenecks and application bugs



This is when ME request bus access via cmd_req and granted bus access after arbitration

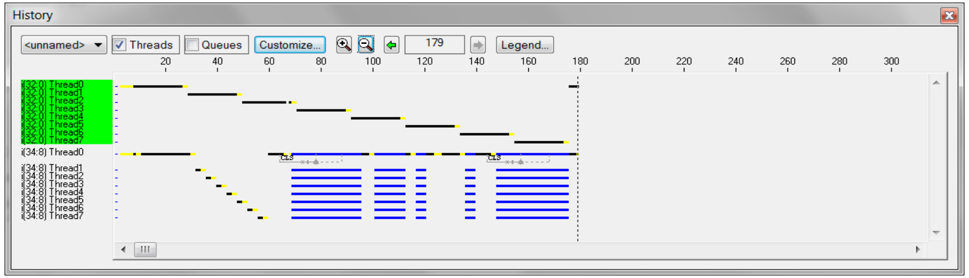
This is when ME en-queues the command into ME command FIFO

This is when command de-queued from ME command FIFO and put on the DSF CPP or routed to an internal target (at the IMB level)

This is when command received in targeted island IMB and need to be addressed by the targeted island. At this point command put into target island command FIFO

This is when target island processed the command and response put back on the DSF CPP and on its way to master island. At this point command is de-queued from target command FIFO.

This is when signal received by master island (ME). At this point ME consumed the signal



Questions