



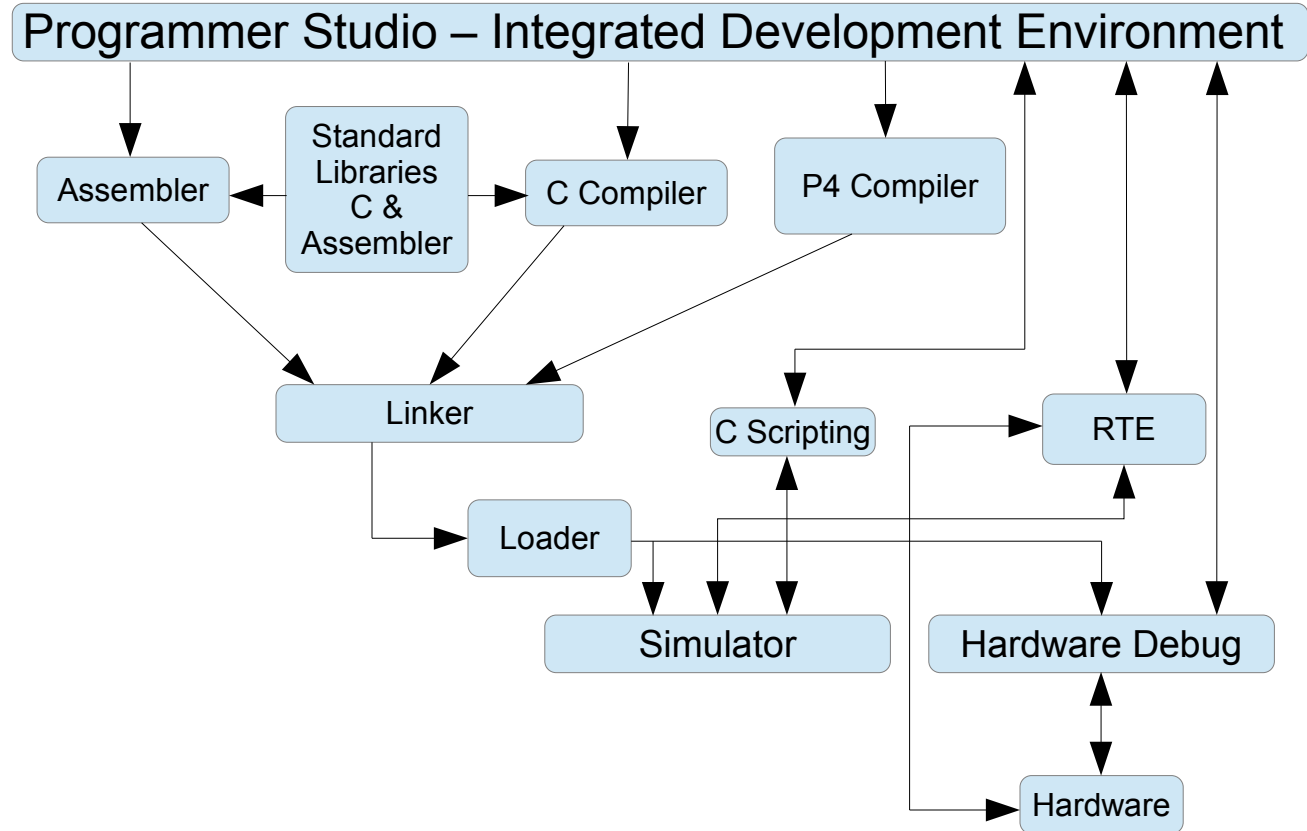
# Software Development Kit – P4 and C Development Toolchain

Dataplane Acceleration Developer Day (DXDD)  
Nov. 2016

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

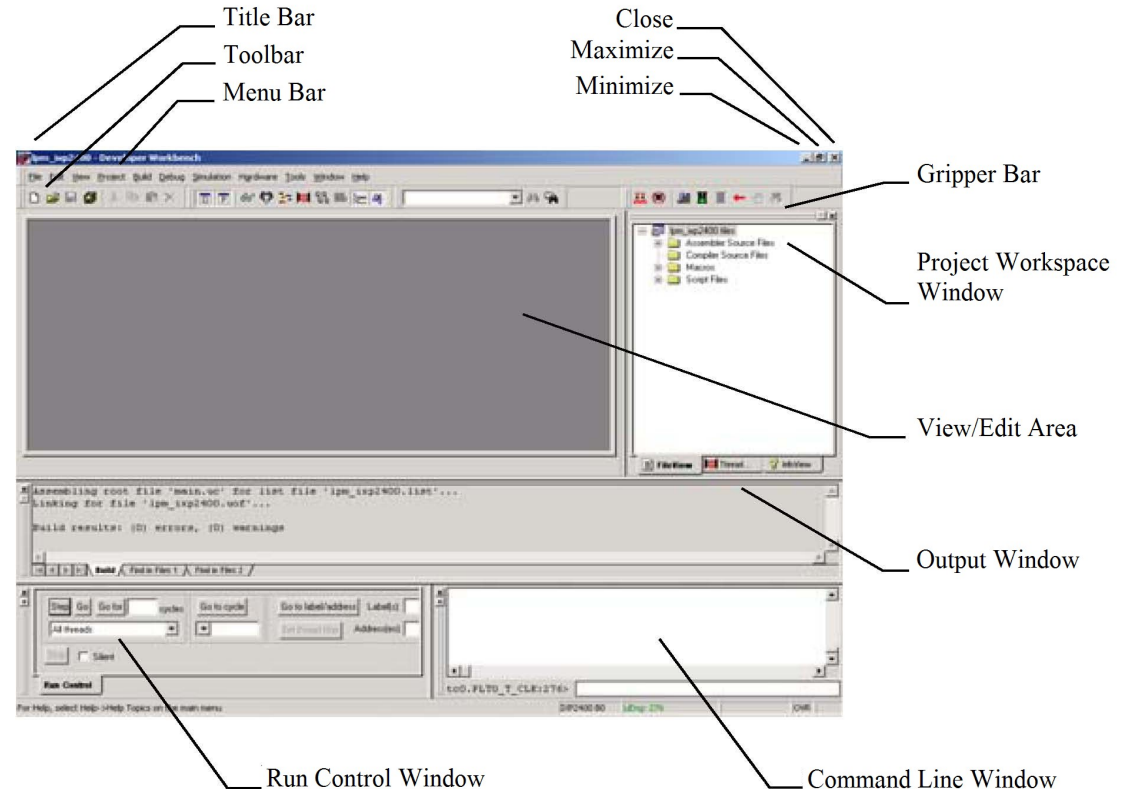
- Integrated Development Environment, running on Windows (natively, in a VM, or in WINE)
- Complete package for SmartNIC application development: edit, build, debug, optimize ...
- Supports data plane programming using P4 and C
- Supports multiple development platforms
  - ✓ Cycle accurate simulator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

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



# Programmer Studio IDE Components

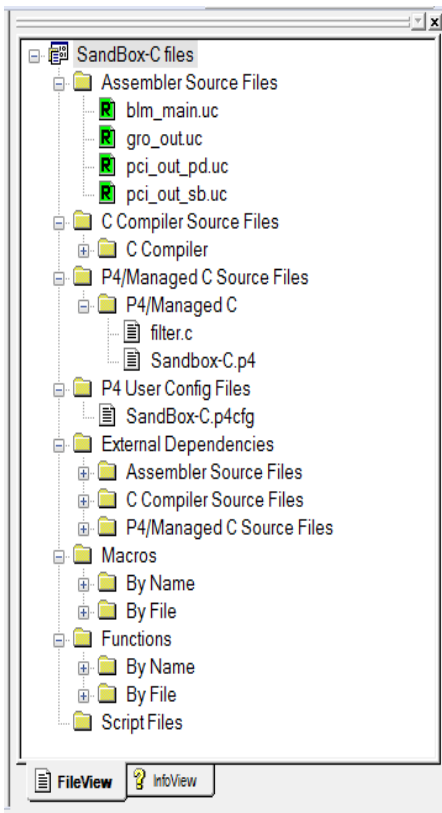
- Integrated Development Environment (IDE)
- Ability to manage ongoing development by organizing settings and files into projects
- Project types
  - C (Standard / Debug Only)
  - P4
- Two sets of toolbar and docking configurations:
  - Debug Mode
  - Build/Edit Mode



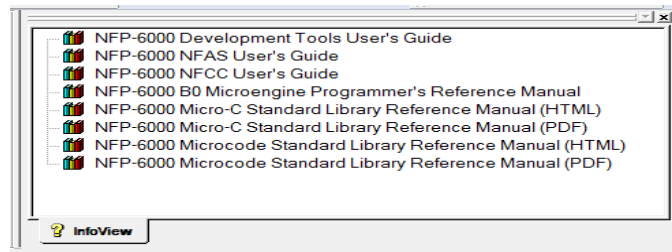
# Programmer Studio Views

- 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:

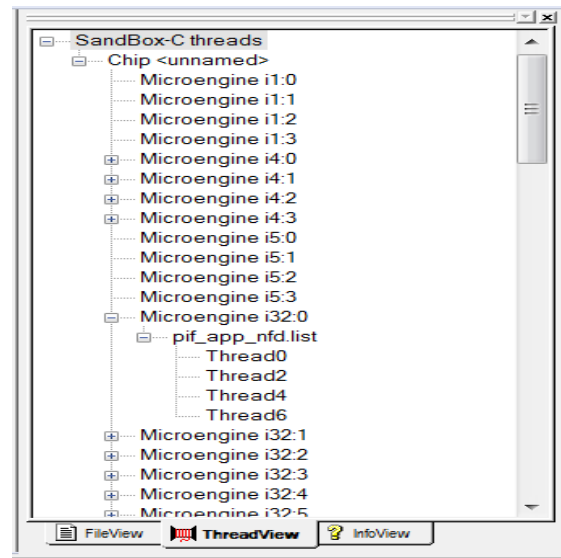
- FileView
- ThreadView
- InfoView



FileView



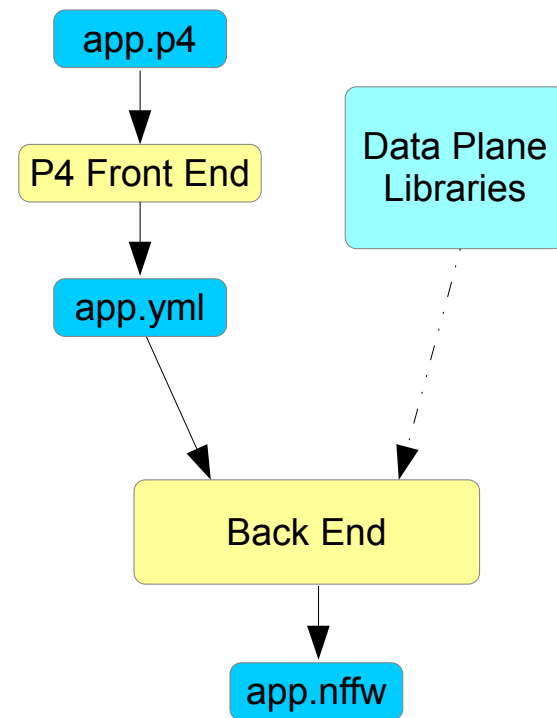
InfoView



ThreadView

- Accepts standard C, augmented with pragmas and specifiers, e.g. `__declspec()` to explicitly 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).

- 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](http://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.

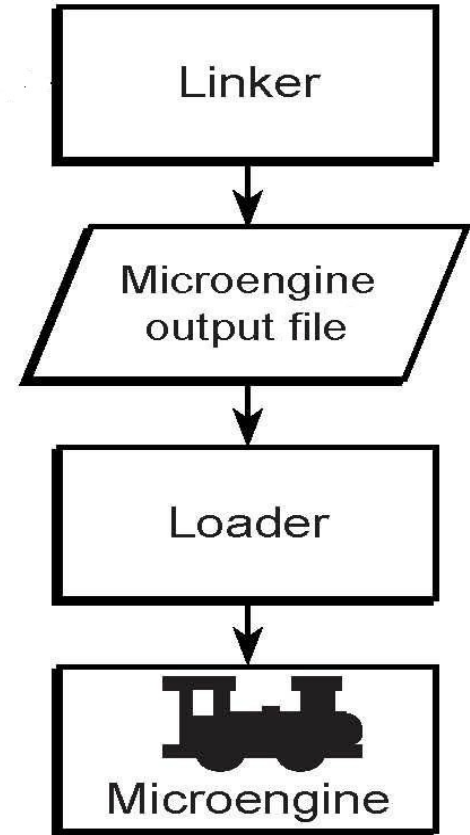




- C libraries are supplied to enable convenient access to the Network Flow Processor's features, for example packet I/O, buffer allocation/freeing, and function accelerators (e.g. ring put/get, statistics, load balancing, hashing, metering, individual lookup operations, etc.)
- Larger Standard Components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

## 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



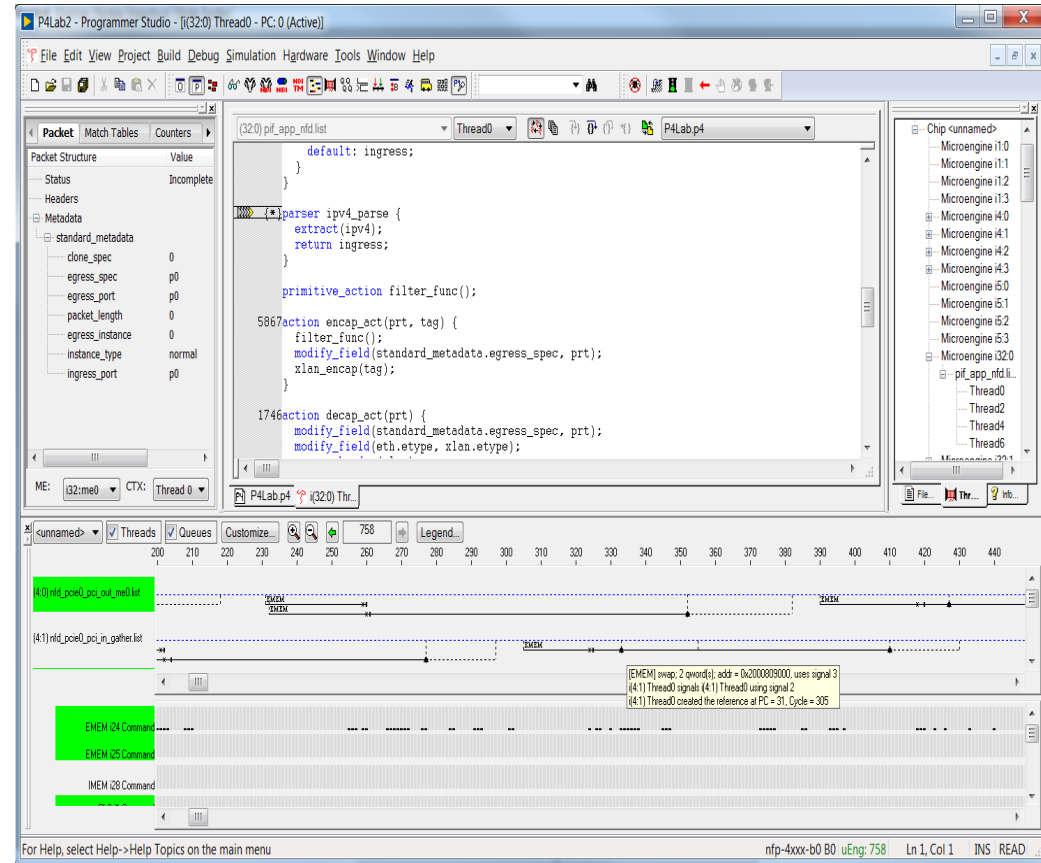
The Programmer Studio IDE supports debugging in three different configurations:

- **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.

- 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 <sup>1</sup>
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 <sup>1</sup>
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	

- 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



The screenshot displays the P4Lab - Programmer Studio interface. The top window shows the source code for a packet parser and actions. The code includes a default ingress action, a parser for IPv4, and actions for encapsulation and decapsulation.

```
default: ingress;
}

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

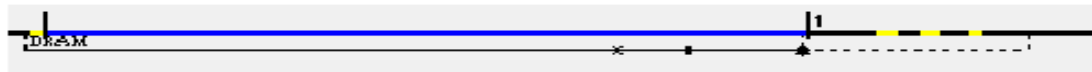
primitive_action filter_func();

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

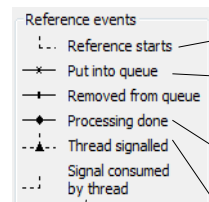
1746 action decap_act(prt) {
    modify_field(standard_metadata.egress_spec, prt);
    modify_field(eth.etype, xlan.etype);
}
```

The bottom window shows a timing diagram with threads and queues. A legend indicates that the diagram shows 758 cycles. A tooltip provides details about a memory swap operation: [EMEM] swap: 2 qword(s); addr = 0x2000809000; uses signal 3; [4:1] Thread0 signals [4:1] Thread0 using signal 2; [4:1] Thread0 created the reference at PC = 31; Cycle = 305.

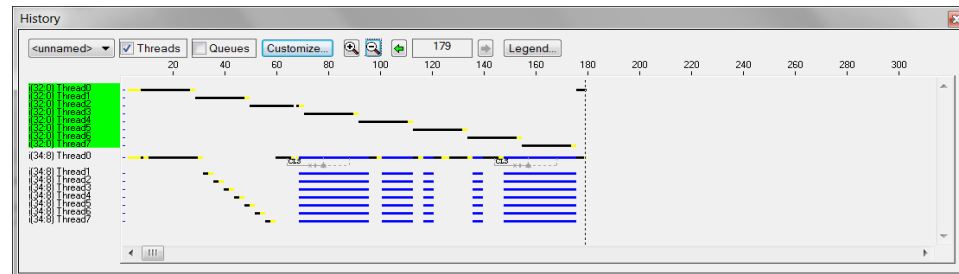
# Simulator – History Collection

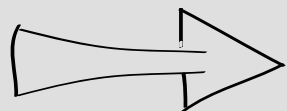


- Thread history – tracks references that are generated by execution of instructions
- Queue history – tracks commands issued on internal buses (to/from function accelerators, memory, or I/O peripherals)
- History data is collected from:
  - Event bus
  - Local CPP (Command Push Pull) bus
  - DSF CPP (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