基于FPGA的P4研究

报告人：厉俊男
国防科学技术大学

主要内容

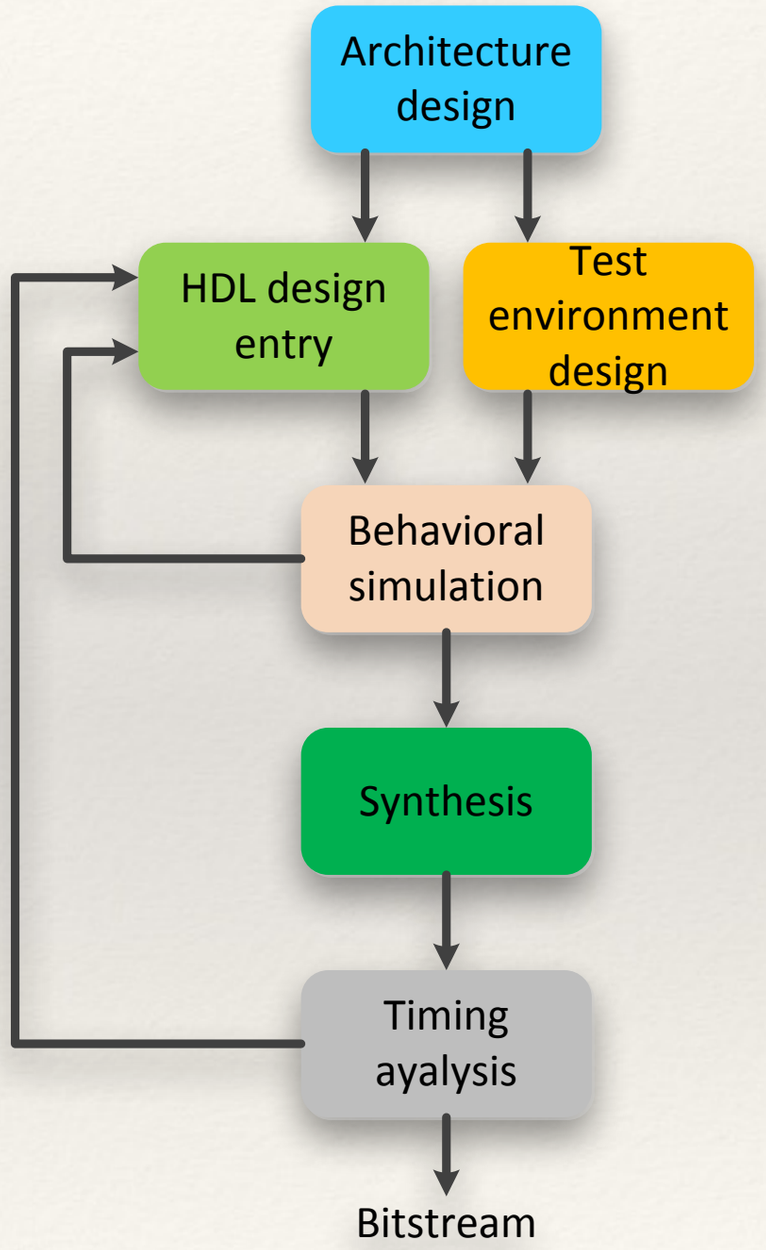
- 研究背景
  - FPGA开发模式
  - 现有FPGA支持P4的开发模式
  - 现有FPGA支持P4方案存在的不足
- 我们的方案
  - P4FAST

一、研究背景

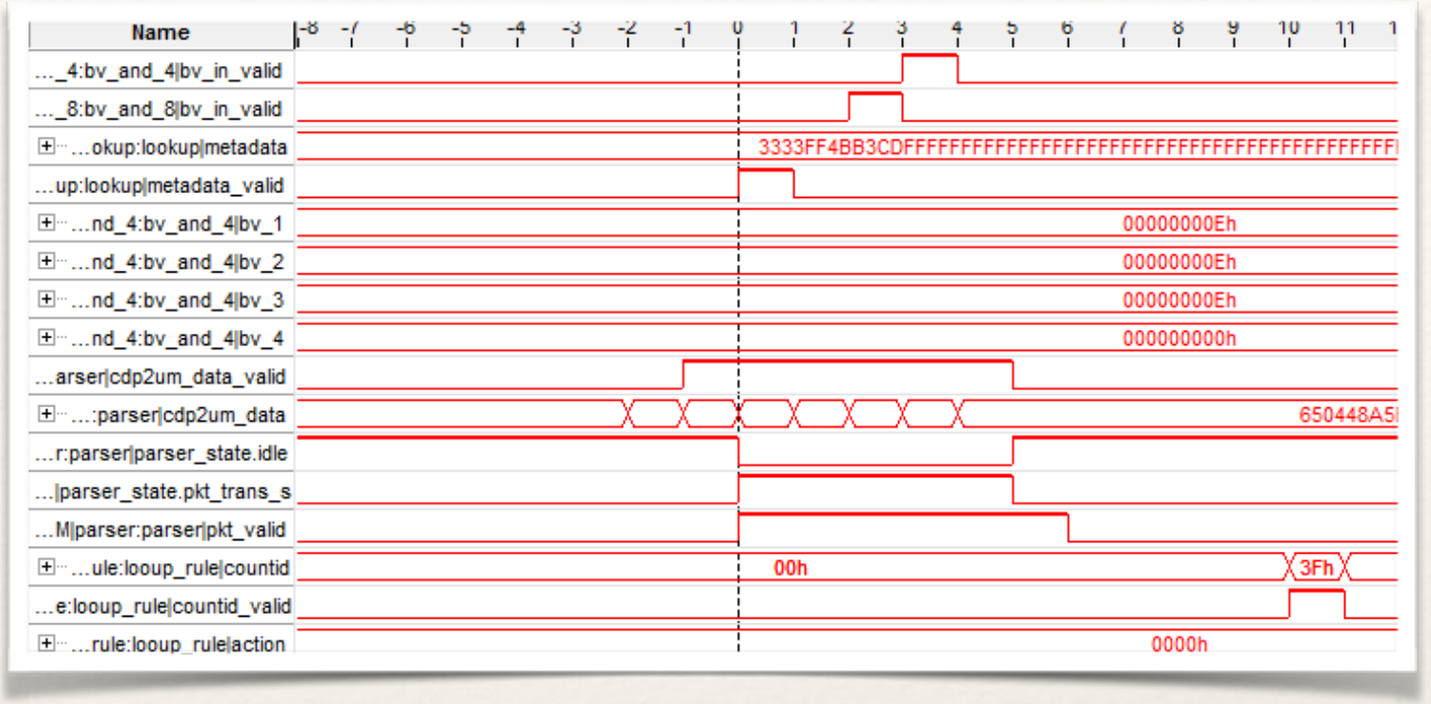
传统FPGA开发

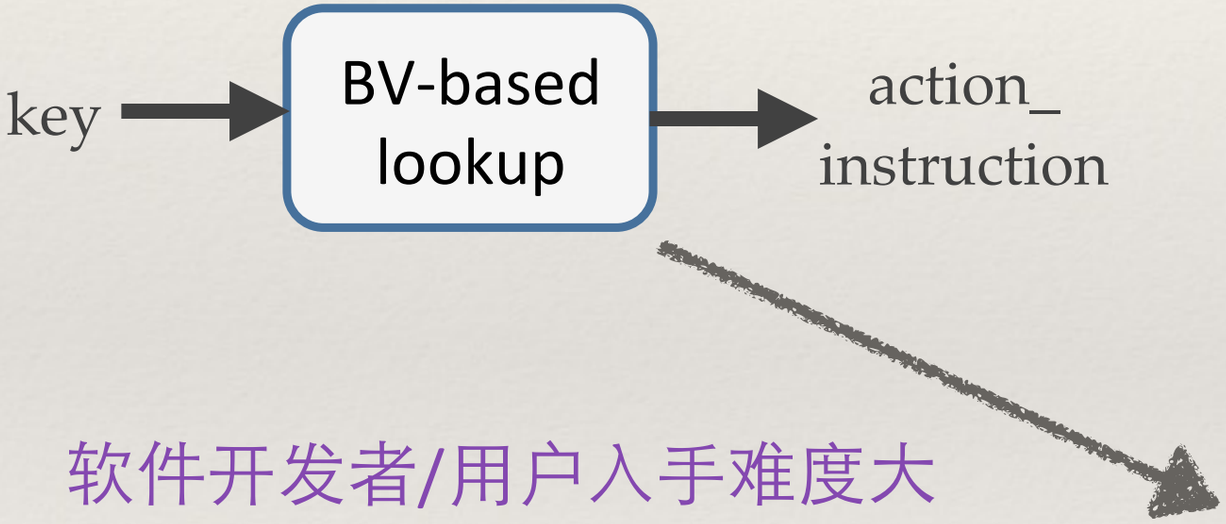
- 需要具备丰富的芯片设计经验
- 能够使用硬件描述语言
  （Verilog、VHDL等）
- 需要设计与开发测试环境
  - 验证功能的正确性以及性能的评价

- 传统FPGA开发流程：
  - 整体架构的开发
  - 功能模块、测试环境的设计
  - 功能模拟
  - 整个工程的仿真测试
  - 工程的综合
  - 时序分析



- 硬件调试困难
  - 需要对信号进行分析
  - 工程综合时间开销大
  - 受硬件资源限制
    - 每次能够分析的信号数量受限





软件开发者/用户入手难度大

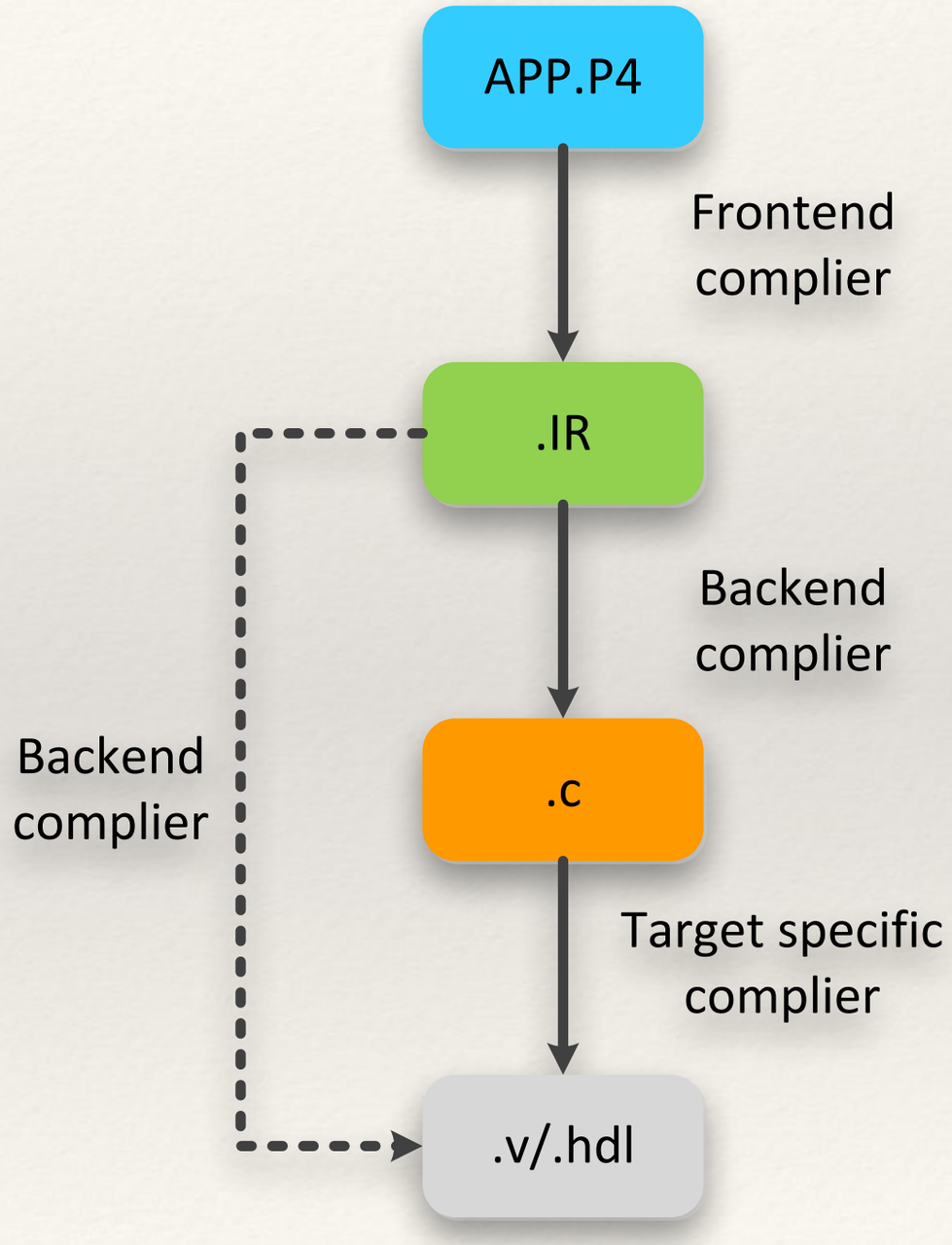
事实上，用户并不关心底层硬件的实现方式

```
always @ (posedge clk)begin
    if(bv_valid == 1'b1)begin
        bv_out_valid <= 1'b1;
        if(bv[range_end-1:0])begin
            bv_out <= bv;
            count_out <= count;
          end
        else begin
            bv_out <= bv >> range_end;
            count_out <= count + range_end;
          end
    end
    else begin
        bv_out_valid <= 1'b0;
        bv_out <= {width{1'b0}};
        count_out <= {width_count{1'b0}};
      end
end
```

P4架构下FPGA开发

- P4架构下FPGA开发流程
  - 前端编译：P4 → HLIR
  - 后端编译：HLIR → C语言
  - 平台相关编译：C语言/HLIR → HDL
    (Hardware Description Language)

- 目前基于FPGA实现的后端编译器
- Cornell: P4FPGA
  - P4 → HLIR → Bluespec → Verilog
- Xilinx SDNet
  - P4 → (HLIR →) PX → HDL.
- Netronome OpenNFP:
  - P4 → HLIR → C /C++ → HDL



P4FPGA

- P4FPGA
  - 目标：为了使P4的编译能够适配大量的FPGA平台
    - 使用高级语言描述数据平面处理功能
    - 减少硬件调试时间（Debug）
    - 实现跨平台使用

- P4FPGA 技术要点
  - 处理流程：
    P4 → IR → Bluespec → FPGA
  - Bluespec作为实现语言
  - P4FPGA的处理流程采用
    Makefile的方式实现
  - 整合Vivado和Quartus工具链

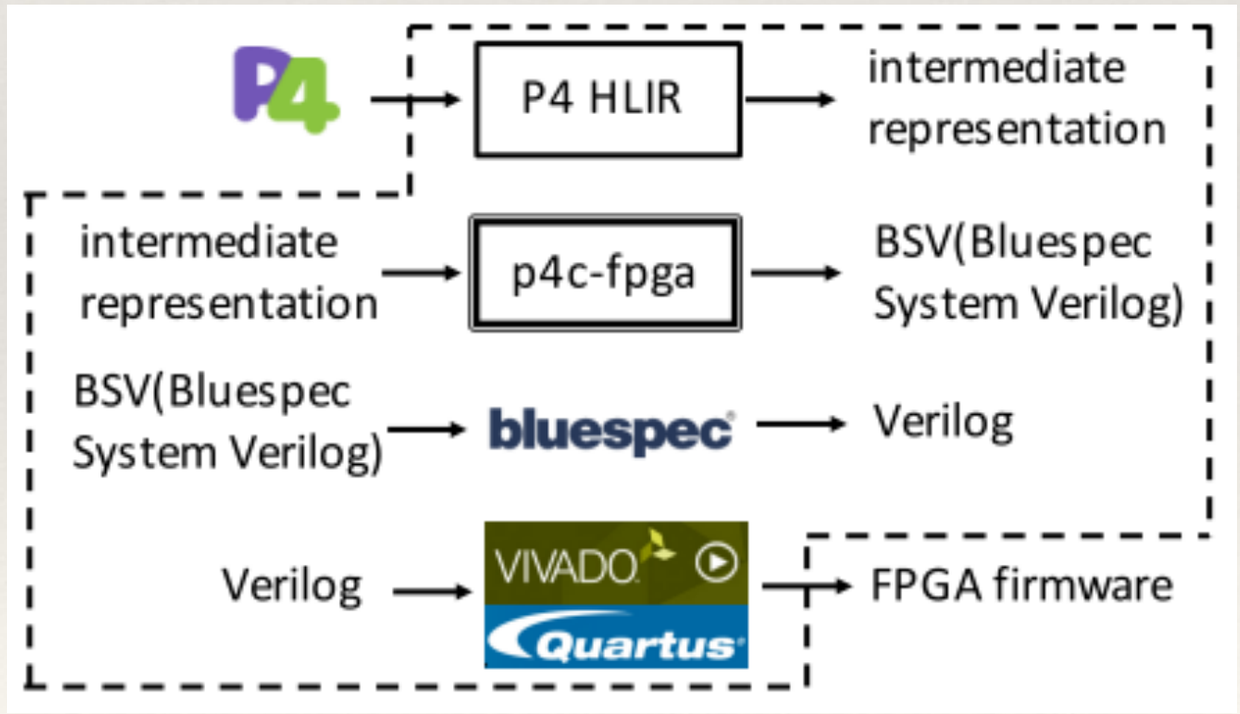


Fig. from P4FPGA Towards an Open Source P4 Backend for FPGA, P4'15 workshop

SDNet

- SDNet基于Xilinx FPGA平台实现了可编程SDN交换机
  - SDNet采用面向对象的模块化说明语言PX，不描述实现细节
  - 支持并预置了更通用的解析、编辑、查找等IP引擎
  - 允许用户自定义处理引擎模块
  - 允许不同引擎基于数据流的组合

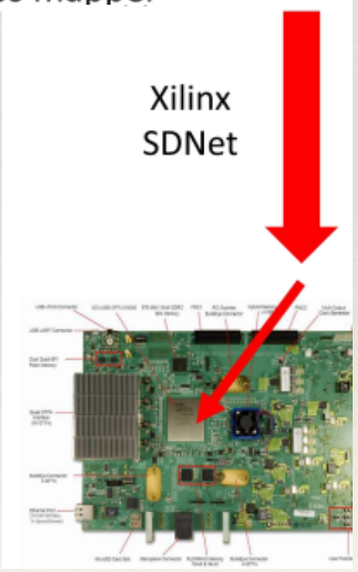


Figure from P4 for an FPGA target, P4'15 workshop

openNFP

- Netronome OpenNFP
  - 开发了Netronome SDK 6.0，实现P4/C语言转化为HDL的功能
  - 底层硬件使用服务器适配卡

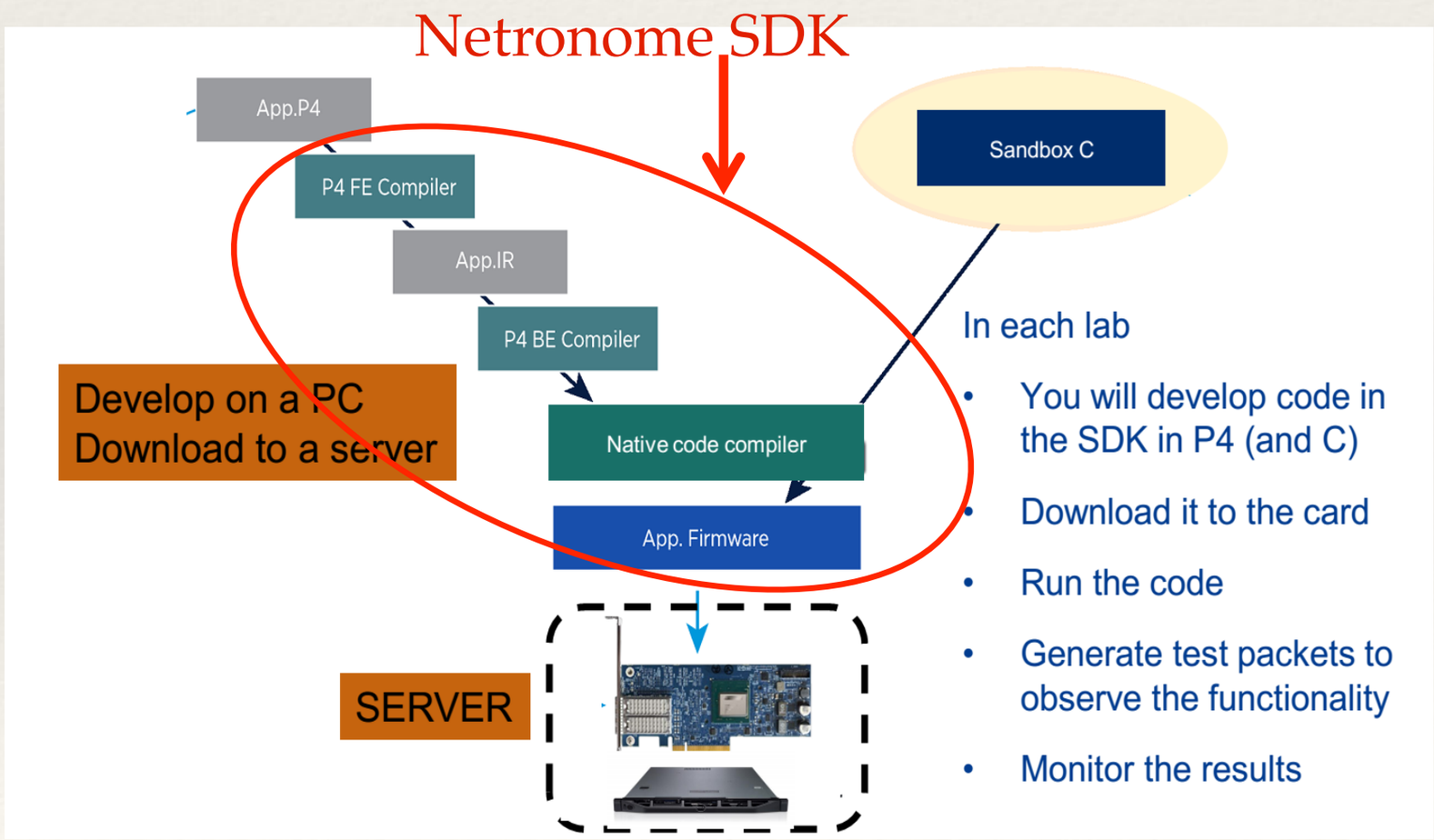


Figure from OpenNFP: open Network Function Processing, P4'15 workshop

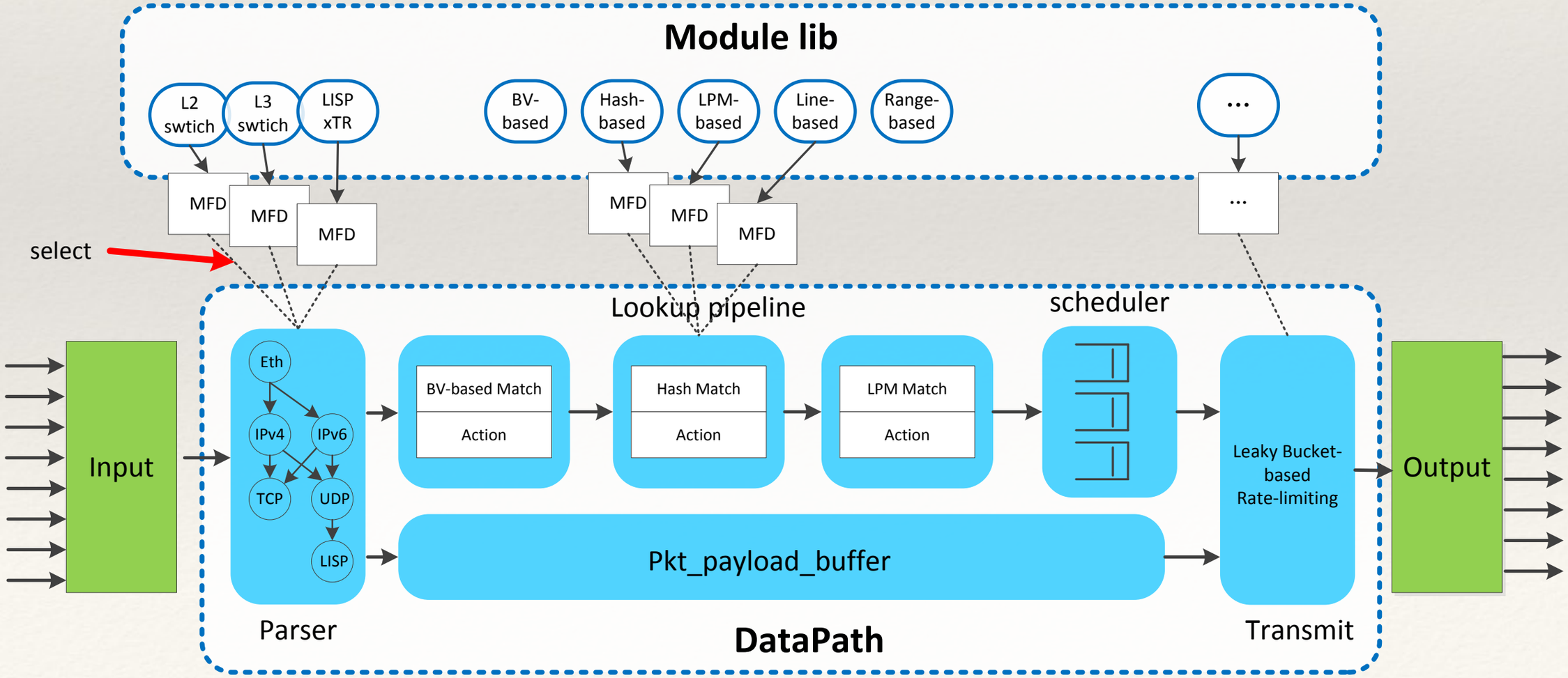
现有FPGA支持P4存在的不足

- 现有FPGA支持P4方案存在的不足：
  - P4语言转化为硬件语言比较困难
    - 高级语言与底层硬件语言存在“代沟”，高级语言描述能力强，很多处理功能、算法直接转化为硬件语言困难（有状态的报文处理，流调度，查找算法等）
  - 性能不高
    - FPGA开发最大的优势是并行度高，后端编译器实现模块内，模块间的并行性开发难度大，同时需要处理数据相关等问题
  - 时钟频率不高
    - FPGA的处理速度受最低时钟频率的限制（水桶短板），大量的阻塞式赋值，又会带来报文处理延时的增加
  - 资源的利用率不高
    - P4编程对寄存器的使用透明，硬件资源的利用率相对于硬件开发较低

二、我们的方案

基于FAST平台实现P4

- 我们的方案——P4FAST:
  - 利用FAST开源的优势，提供丰富的功能模块库，允许不同引擎基于数据流功能进行组合
    - 模块化的设计，有利于模块的重用
    - 每一个功能模块都有对应的MFD（Modular Function Description）描述
    - 采用参数化方式，减少硬件资源的浪费，避免模块重新设计



模块功能描述（MFD）

- MFD (Modular Function Description)
  - 以match、action模块为例
    - Description用于描述该模块的功能
    - Parameter则是可以用于模块的配置（e.g. 关键字位宽、表项数量）

```
match bv_based{
    description{
        lookup_type = BV_BASED;
        //lookup_type = HASH_BASED: 1;
        // hash_parameter == 1 represent mod operation;
        width_key_set_valid = 1;
        //"1"represent this module support set width_key;
        width_key_max = 1024;
        width_key_min = 8;
        num_entry_set valid = 1;
        //"1"represent this module support set num_entry;
        num_entry_max = 8192;
        num_entry_min = 256;
    }
    parameter{
        parameter_width_key;
        parameter_num_entry;
        parameter_width_action_instruction;
    }

}
```

```
action{
    description{
        drop:1;
        send_egress_port:1;
        send_cpu:1;
        //delay_send:1;
        set_next_table:1;
        reserved:4;
        index_action_data:8;
    }
}
action_data{
    description{
        port:8;
        //ip_dst:32;
        //mac_dst:48;
        next_table_id:8;
    }
}
```

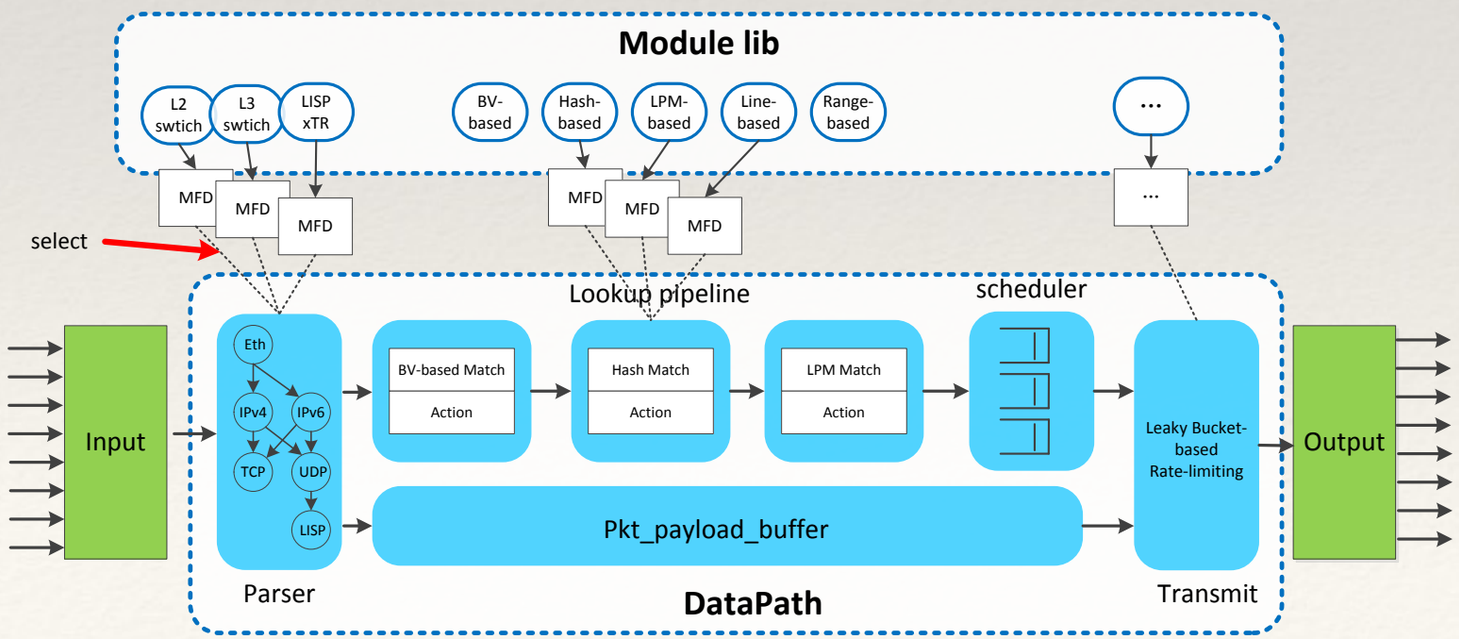
模块参数化配置

- 以BV-based查找模块为例
  - 在顶层模块中定义相应的变量参数
    - 输入关键字的长度
    - 表项的数量
  - “动态”的（根据用户的需求）定制相应大小的表项空间
  - 类似于IP核的功能

```
// __adjust_parameter
parameter       width_key    = 288; // widht_key = width_search_key * num_search;
parameter       width_ruleid= 6; // rule_id of matched entry
parameter       width_ram    = 64;
parameter       depth_ram    = 512;
parameter       stride       = 9; // width_bit_lookup_once
parameter       width_search_key = 27; // width_search_key = num_lookup_bit * stride; key of search_engine;
parameter       width_bv     = 64;
parameter       num_search   = 4; // number of search_engine; each has some lookup_bits;
parameter       num_lookup_bit = 8;;
//  localbus_addr_parameter
// localbus is used to connect MM and UM
parameter       bit_search   = 22;
parameter       bit_lookup_bit  = 19;
parameter       bit_ram      = 17;
parameter       bit_bv       = 8;
// __adjust_end
```

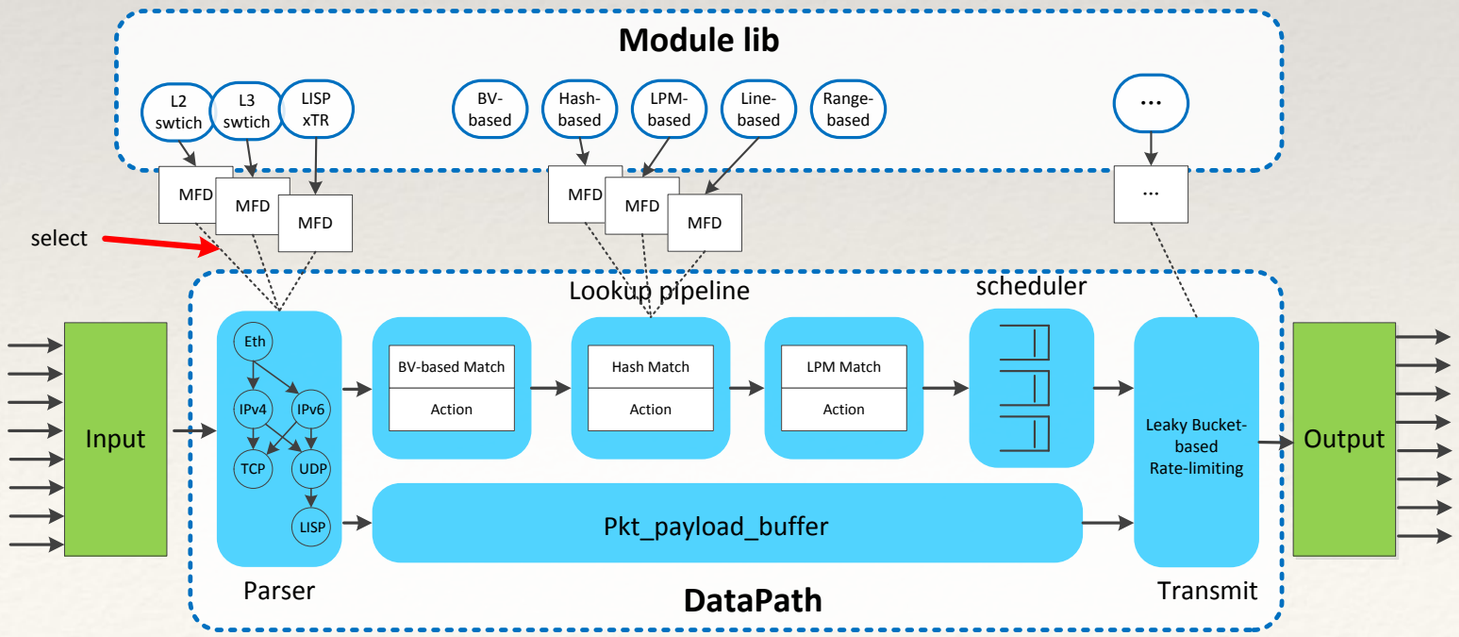
基于FAST平台实现P4

- 我们的方案——P4FAST:
  - 利用FAST开源的优势，提供丰富的功能模块库，允许不同引擎基于数据流功能进行组合
  - 采用软硬件协同开发的模式
    - 基于verilog/vhdl设计数据平面功能模块
    - 基于C/C++/python设计控制器与数据平面通信API
    - 基于P4从模块库中选择、组合相应的FAST流水线



基于FAST平台实现P4

- 我们的方案——P4FAST:
  - 利用FAST开源的优势，提供丰富的功能模块库，允许不同引擎基于数据流功能进行组合
  - 采用软硬件协同开发的模式
    - 用户和开发者分离的架构
      - 允许开发者自定义处理引擎模块
      - 用户根据需求组合流水线



P4FAST开发流程（1/2）

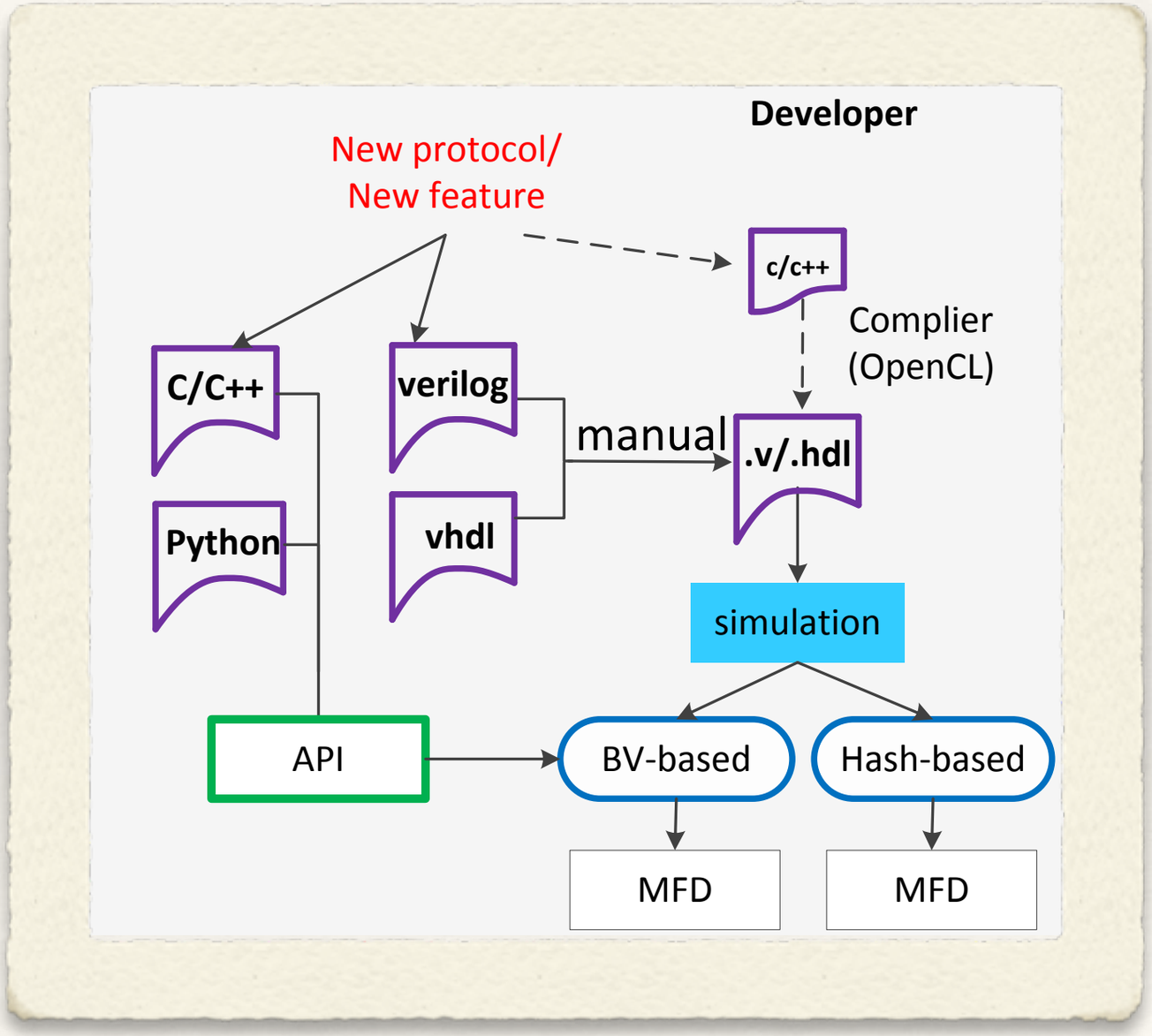
- 功能模块开发流程（开发者模式）

A. 根据新协议、新需求，基于HDL实现相应的功能模块

B. 模块仿真，综合，时序分析

C. 生成相应的模块功能描述（MFD），供用户选择

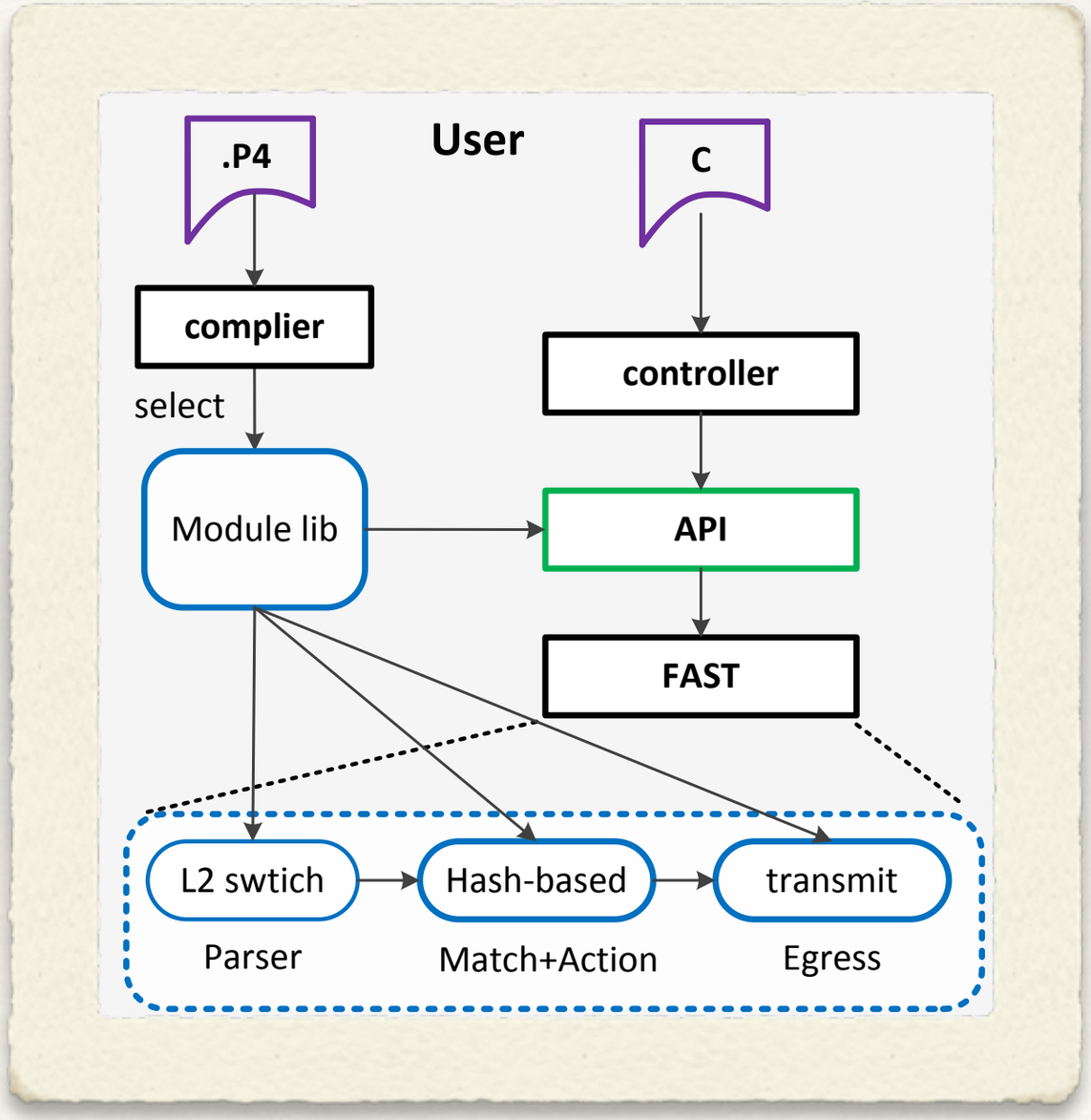
D. 软件开发者配合生成访问硬件模块的API接口

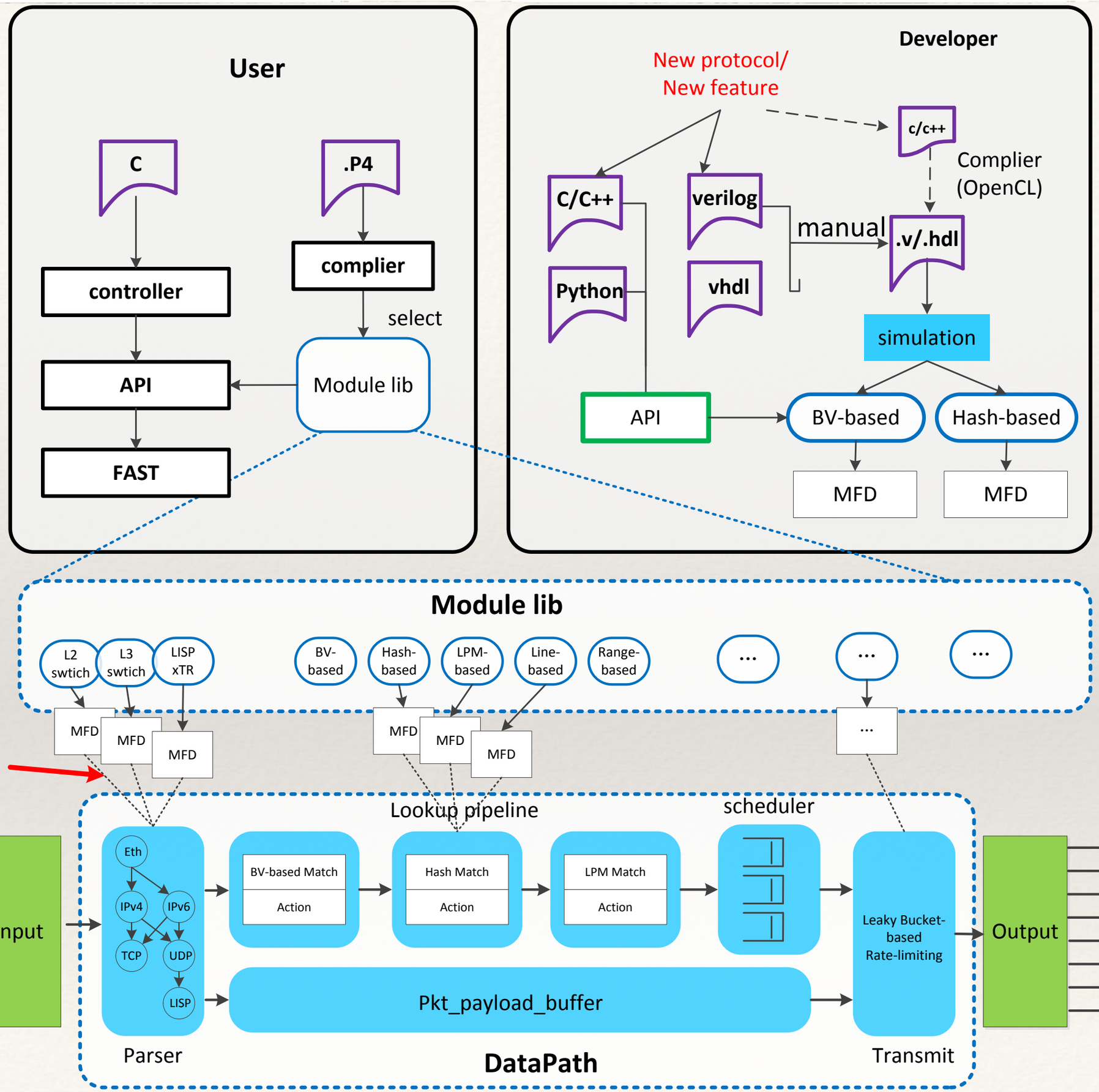


P4FAST开发流程（2/2）

- 功能模块开发流程（用户模式）

A. 根据P4描述，选择合适模块组合成FAST流水线（编译器）
B. 根据所选择模块，形成相应的API接口，控制器向数据平面下发流表
C. 将FAST处理流水线下发至FPGA，综合测试
D. 实际应用场景测试




User
C
.P4
controller
complier
select
API
Module lib
FAST
Developer
New protocol/
New feature
c/c++
Complier
(OpenCL)
C/C++
verilog
manual
.v/.hdl
Python
vhdl
simulation
API
BV-based
Hash-based
MFD
MFD
Module lib
L2 swtich
L3 swtich
LISP xTR
BV-based
Hash-based
LPM-based
Line-based
Range-based
...
...
...
MFD
MFD
MFD
MFD
MFD
MFD
...
select
Lookup pipeline
scheduler
Input
Eth
IPv4
IPv6
TCP
UDP
LISP
BV-based Match
Action
Hash Match
Action
LPM Match
Action
Leaky Bucket-based Rate-limiting
Output
Pkt_payload_buffer
Parser
DataPath
Transmit

下一步工作

- 编译器的研究与开发
  - 编译P4程序，根据需求如何高效的从模块库中选择合适的模块，并将其组合成FAST流水线
    - 存在多个备选模块的情况（优先级评价）
    - 存在近似匹配的情况（相似度评价）
  - 编译P4程序，生成一些简单的无状态处理模块
    - 简单的协议解析功能
    - 报文封装、转发功能
- 开发数据平面的并行性
  - 目前数据平面只支持单一的流水线，在下一步工作中可以考虑增加多条流水线
    - 涉及报文流在不同流水线中的调度

实现方式对比

实现方式	高级秒速语言	功能模块实现方式	开发模式	编译时间	编译器开发难度	功能模块开发周期	资源利用率	处理性能	开源情况
P4FPGA	P4	P4描述，经编译器产生verilog代码							开源
SDNet	P4/PX	P4/PX描述，经SDNet编译产生硬件代码	软件开发（+硬件优化）	hours	难度较大	相对较短	相对较低	并行性开发相对困难	未开源
openNFP	P4	P4描述，经目标相关编译器生成硬件代码							
P4FAST	P4	基于HDL硬件开发功能模块库，用户P4描述，选择组合数据平面流水线	软件+硬件开发	minutes	难度较小	相对较长	相对较高	相对较高	开源

“谢谢。”