THE CLICK2NETFPGA TOOLCHAIN

Teemu Rinta-aho, Mika Karlstedt, Madhav P. Desai

USENIX ATC ‘12, Boston, MA, 13th of June, 2012

CLICK2NETFPGA

- We have explored the possibilities of High Level Synthesis (HLS) in the packet processing domain
  - HLS is transforming software into hardware
- Using a number of open source components, some new code (and some glue), we have created ***a prototype* toolchain** that allows
  - Defining Click configurations using existing and new elements,
  - Writing new Click elements in C++, and
  - Compiling them to hardware, to be run on NetFPGA
- The main components are Click, NetFPGA, LLVM, Click2LLVM and AHIR

RELATED WORK

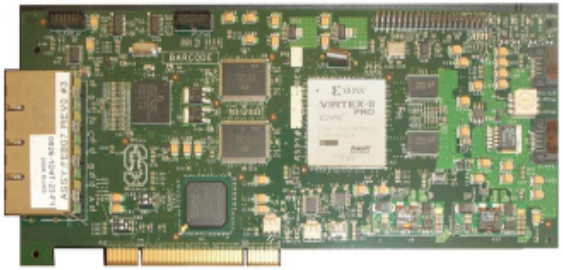
- There are several academic and commercial HLS tools available: Trident, LegUp, AutoESL, Catapult C, C2S, ...
- They either hardware accelerate certain parts of a software program, or completely synthesise only smaller units
- The Click2NetFPGA Toolchain is a “system-to-system” compiler
  - More specifically designed for a specific source and a target system
  - .. but modifying or extending the source system does not require knowledge of the target system
  - .. and modifying packet processing code does not require understanding hardware design or programming in Verilog or VHDL

CLICK MODULAR ROUTER

- A software platform (C++) for building different kinds of packet-processing nodes / functions, or “routers”
- Define a software router from packet processing elements
- Runs in Linux/BSD user space or in Linux kernel
- 100+ existing elements, e.g. ARPResponder, Classifier
- Easy to add new elements
- Easy to build custom routers

NETFPGA

- A PCI network interface card with an FPGA
  - 4 x 1 Gbps Ethernet interface
- Line-rate, flexible, and open platform
- For research and classrooms
- More than 1,000 NetFPGA systems deployed
- A few open-source, Verilog-based reference designs
- Harder to modify or add new modules (for an average network developer/ researcher) than in Click



LLVM

- An open source compiler, from UIUC
- A set of tools and optimizers
- Easy to write new compiler passes
- Easy to write new backends (and, maybe, frontends)
- Represents intermediate code in SSA (Single Static Assignment) form
  - An abstract, assembler-like form, with unlimited registers
- Outperforms GCC in many (but not all) ways
- Can perform global optimizations (after linking)

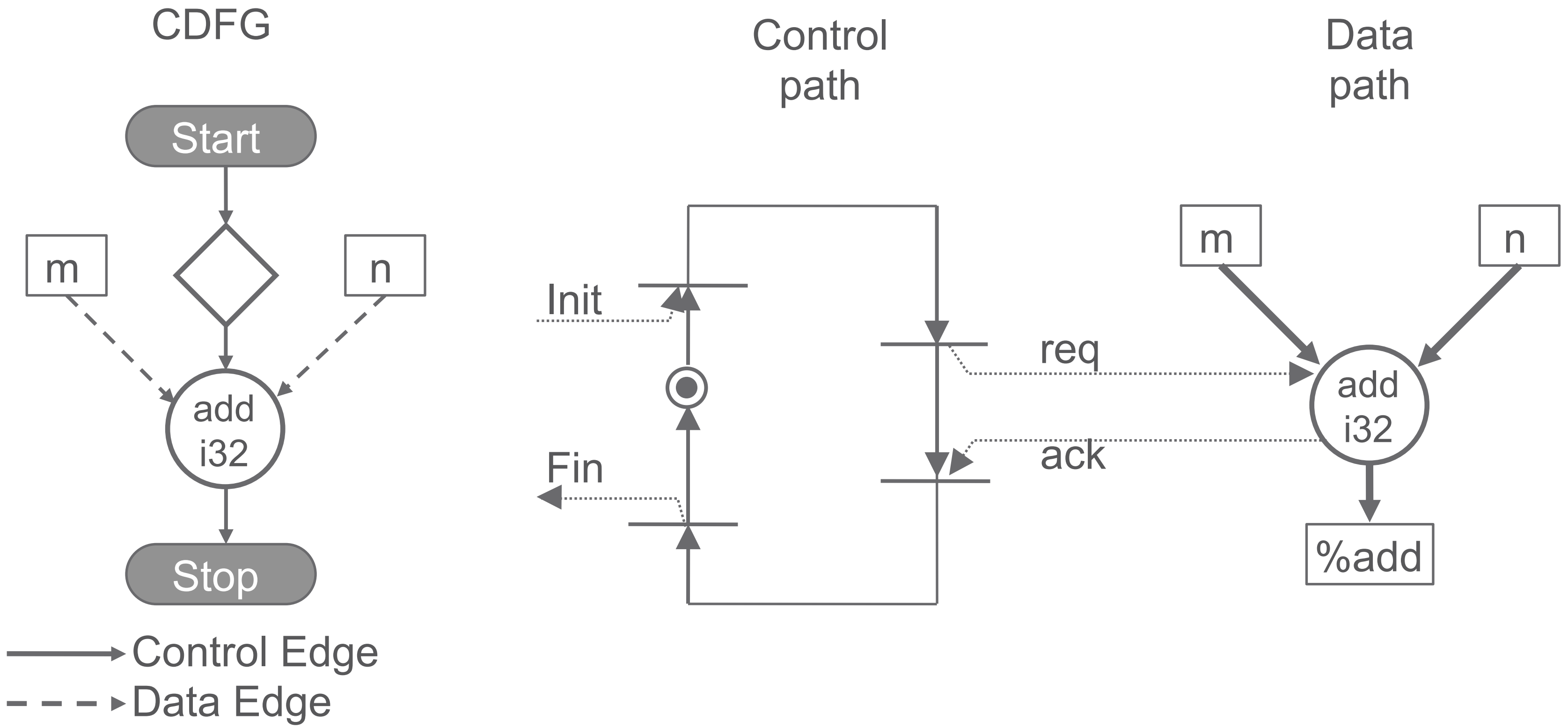
AHIR

- LLVM backend for generating VHDL
  - Open source, from IIT Bombay
  - Factorises the system into control, data, and storage
    - Supports scalable optimisations and analyses
  - Current limitations: no recursion or function pointers, otherwise full C
  - Generates a VHDL module out of each LLVM IR function
- Design = Set of modules with I/O channels
  - I/O through a simple VHDL “library”, resembling Unix pipes

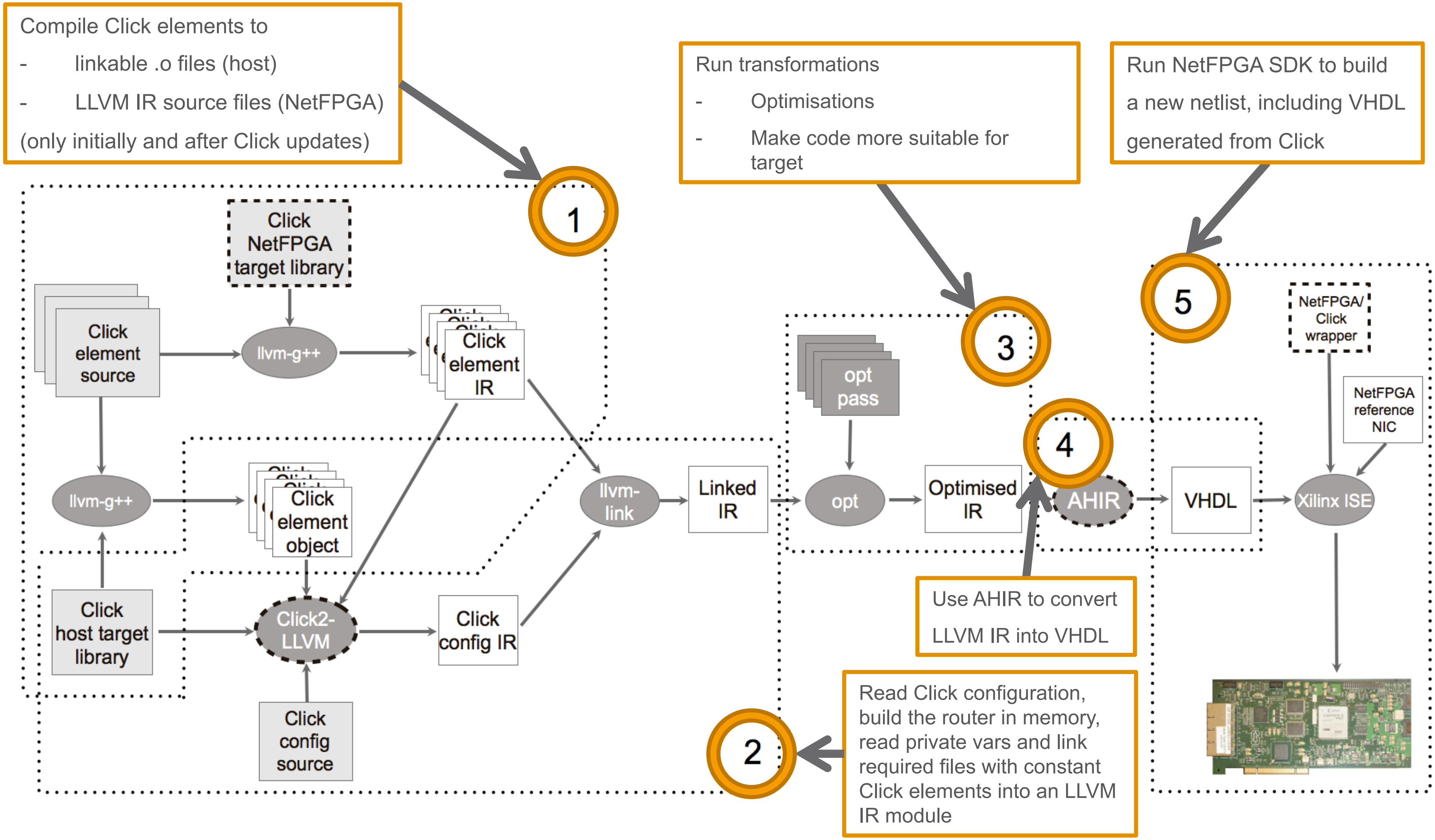
AN AHIR EXAMPLE

CONVERTING AN LLVM INSTRUCTION TO VHDL

- A simple example of **addition** instruction
- C code: **`int d = m + n;`**
- Equivalent LLVM IR: **`%d = add i32 %m, %n`**



IMPLEMENTATION



Compile Click elements to

- linkable .o files (host)
- LLVM IR source files (NetFPGA)

(only initially and after Click updates)

Run transformations

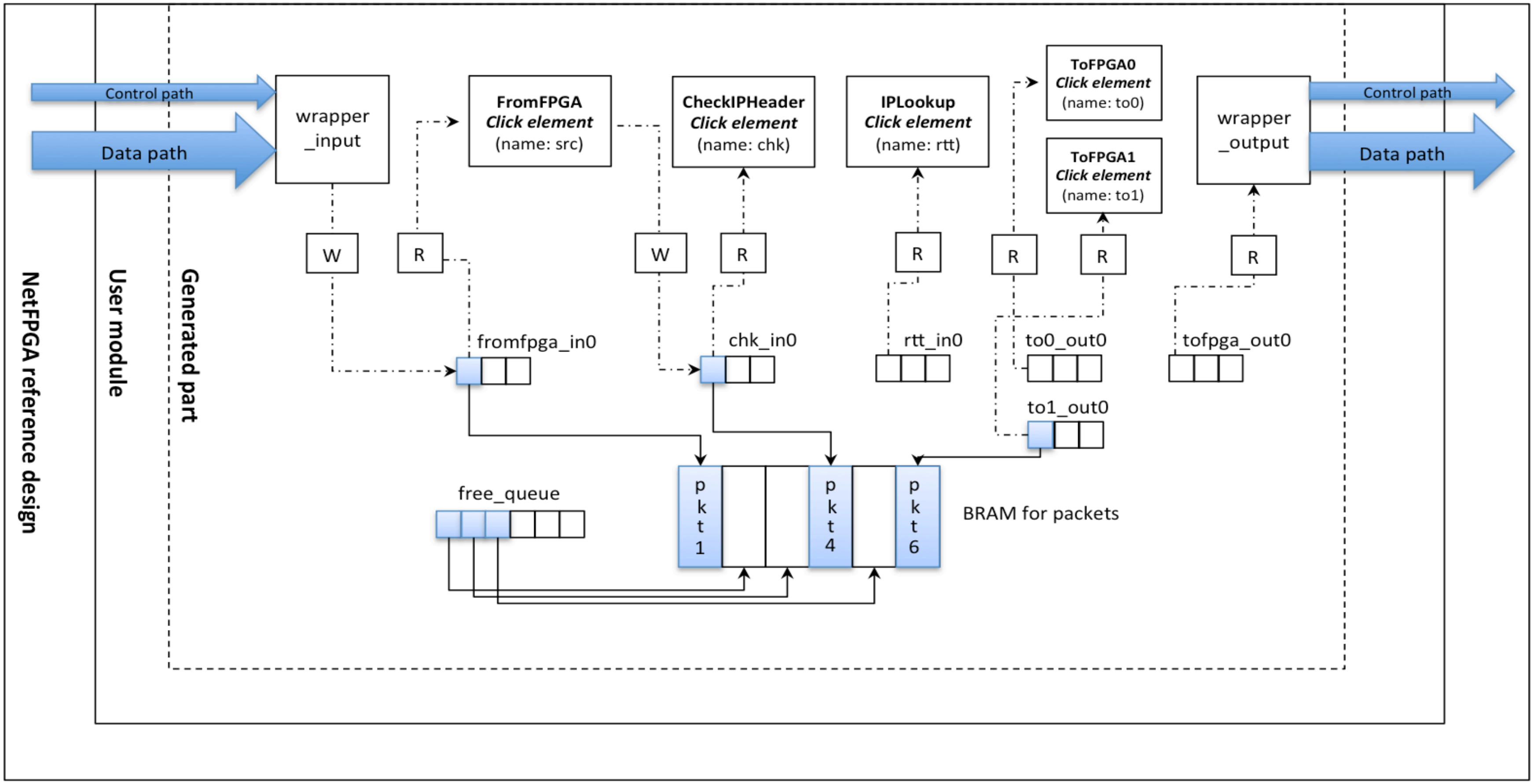
- Optimisations
- Make code more suitable for target

Run NetFPGA SDK to build a new netlist, including VHDL generated from Click

Use AHIR to convert LLVM IR into VHDL

Read Click configuration, build the router in memory, read private vars and link required files with constant Click elements into an LLVM IR module

RESULTING HARDWARE



EVALUATION

- Modifying a Click-based hardware router requires only modification of the Click element's C++ code, running make and waiting for the hardware synthesis to complete
- Current prototype can reach 1/3 of the line speed (1 Gbps)
  - Translation between NetFPGA and Click data models on input and output is the major bottleneck
- We could reach the same packet processing performance but we would use more (than available) FPGA resources
  - For example, in one configuration, replicating the CheckIPHeader element gave an 18% improvement with 10% increase in resource usage
  - Also adding double banked memory gives a total of 31% improvement with a total of 19% increase in resource usage

FUTURE WORK

- Redesigning memory/packet-I/O model
  - 64-bit instead of 8-bit memory transfer functions
  - Initiating Click processing before a complete packet is received
- Using memory outside the FPGA (e.g. DRAM)
- Finding more ways to add parallelism
- Using optimized code templates in LLVM -> VHDL transformation
- Dividing Click code to run partly on host CPU and partly as hardware
  - Get back Click live reconfigurability
  - Save FPGA resources for time critical processing tasks

CONCLUSION

- We have shown that writing a toolchain that transforms a complex software system into a hardware system ***is possible***
- However, more work is required in order to develop a toolchain that
  - Creates a hardware system which runs ***more efficiently*** than the original software system and ***doesn't require more hardware resources*** than a hand-written hardware design
  - Supports ***all software features*** (e.g. recursive constructs, system reconfigurability)

Thank you!

Questions?

ERICSSON