C2RTL: A High-level Synthesis System for IP Lookup and Packet Classification

MD Iftakharul Islam, Javed I Khan

Department of Computer Science
Kent State University
Kent, OH, USA.

IEEE HPSR, 2021

Outline

IP lookup and packet classification

- Two of the most fundamental operations of a router.
- IP lookup $\Longrightarrow$ longest prefix match

Table: An example forwarding table

Prefix	Next-hop
131.123.252.42/32	1
169.254.0.0/16	2
169.254.192.0/18	3

- Packet classification $\Longrightarrow$ exact/prefix match to find action

Table: An example packet classifier

Rule	Src. IP	Dest. IP	Src. Port	Dest. Port	Protocol	Action
r1	01100$*$	01100$*$	111	111	80	enqueue
r2	11010$*$	$*$	10$*$	11$*$	22	drop
r2	11$*$	11011$*$	10$*$	11$*$	22	modify
r4	$*$	$*$	$*$	$*$	*	forward

- Implemented in TCAM or Pipelined ASIC.
- ASIC generally executes **Trie** based IP lookup and packet classification algorithms in hardware.

ASIC Design

- ASIC is developed register-transfer (RTL) level Verilog or VHDL $\Longrightarrow$ extremely complex, requires huge effort
  - Need to use circuit level abstractions
  - Need to keep track of path latency to schedule operations at right time.
- This calls for designing ASIC in a higher level.

High-level Synthesis

- High-level Synthesis (HLS) is a design methodology where pipelined ASIC is developed using high-level languages such as C or SystemC.
- HLS generates corresponding Verilog or VHDL RTL from C or SystemC code.
  - Higher design productivity and lower complexity
  - Shorter simulation cycle.
- HLS has not been adopted in routing/switching chips, to the best of our knowledge.

C2RTL High-level Synthesis

- Primarily designed for IP lookup or packet classification, but we would like to extend it for other data plane functions as well.
- C2RTL takes an IP lookup or packet classification algorithm in C as input and generates corresponding Verilog RTL.
- Implemented an a GCC plugin
- We made the source code publicly available `https://tamimcse.github.io/c2rtl`.

Existing HLS tools

- Current HLS tools focus on C or SystemC based system design.
  - Mentor’s Catapult
  - Cadence’s Stratus
  - NEC’s CyberWorkBench
  - Synopsys’ Synphony
  - SWSL **[ANCS 2013]** (designed for IP lookup)
- Switch compiler for programmable ASIC.
  - LEAP **[ANCS 2012]**, Domino **[SIGCOMM 2016]**
- HLS tools for FPGA.
  - Xilinx’s Vivado
  - Bambu **[FPL 2013]**
  - LegUp **[FPGA 2011]**

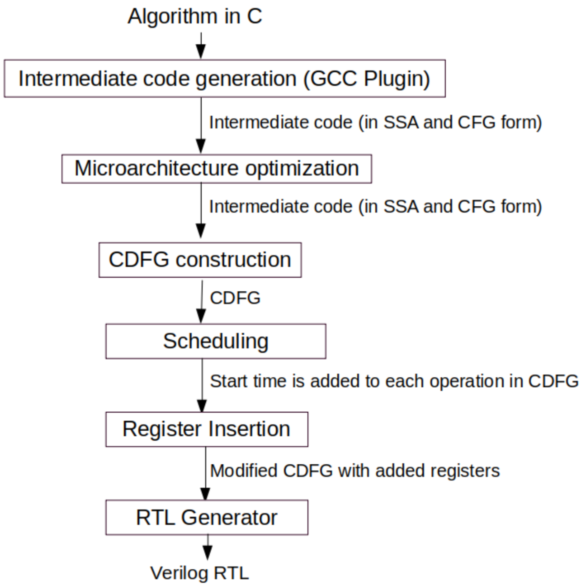
Programming convention in C2RTL

- An input program in C2RTL is implemented in C language, but with several restrictions.

Table: programming restrictions in C2RTL

No loop (while, for, do-while)
No unstructured control flow (goto, break, continue)
No ternary operation
No dynamic memory allocation
No global variables
No structure
No switch
No function call
Each branch has to have a separate return statement

C2RTL design flow


Algorithm in C
Intermediate code generation (GCC Plugin)
Intermediate code (in SSA and CFG form)
Microarchitecture optimization
Intermediate code (in SSA and CFG form)
CDFG construction
CDFG
Scheduling
Start time is added to each operation in CDFG
Register Insertion
Modified CDFG with added registers
RTL Generator
Verilog RTL

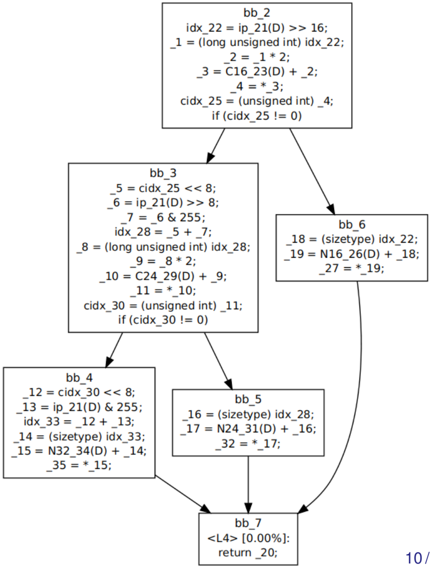
Intermediate code generation by GCC

```c
#include <stdint.h>

uint8_t sail(uint32_t ip, uint8_t N16[100
  uint16_t C16[100], uint8_t N24[100],
  uint16_t C24[100], uint8_t N32[100]) {

  unsigned int idx, cidx;

  idx = ip >> 16;
  cidx = C16[idx];
  if (cidx) {
    idx = (cidx << 8) + ((ip >> 8) & 255)
    cidx = C24[idx];
    if (cidx) {
      idx = (cidx << 8) + (ip  & 255);
      return N32[idx];
    } else {
      return N24[idx];
    }
  } else {
    return N16[idx];
  }
}
```



```
bb_2
idx_22 = ip_21(D) >> 16;
_1 = (long unsigned int) idx_22;
_2 = _1 * 2;
_3 = C16_23(D) + _2;
_4 = *_3;
cidx_25 = (unsigned int) _4;
if (cidx_25 != 0)
```

```
bb_3
_5 = cidx_25 << 8;
_6 = ip_21(D) >> 8;
_7 = _6 & 255;
idx_28 = _5 + _7;
_8 = (long unsigned int) idx_28;
_9 = _8 * 2;
_10 = C24_29(D) + _9;
_11 = *_10;
cidx_30 = (unsigned int) _11;
if (cidx_30 != 0)
```

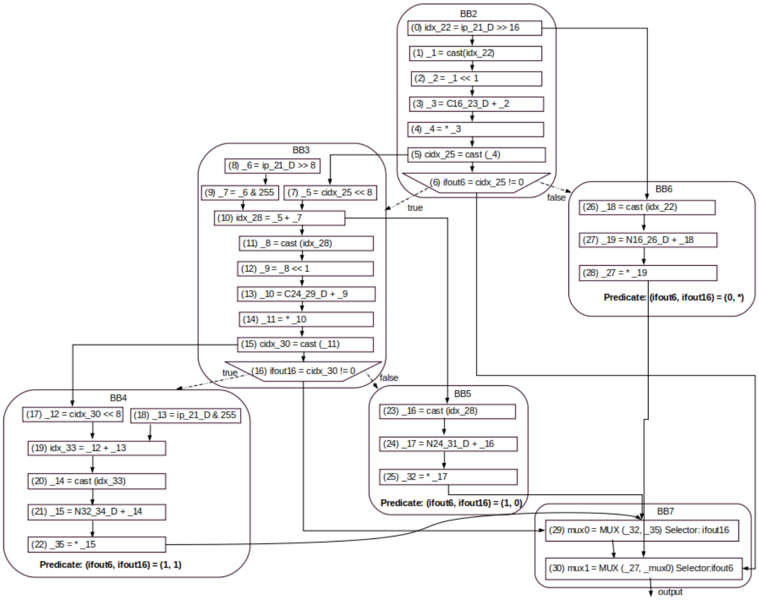
```
bb_6
_18 = (sizetype) idx_22;
_19 = N16_26(D) + _18;
_27 = *_19;
```

```
bb_4
_12 = cidx_30 << 8;
_13 = ip_21(D) & 255;
idx_33 = _12 + _13;
_14 = (sizetype) idx_33;
_15 = N32_34(D) + _14;
_35 = *_15;
```

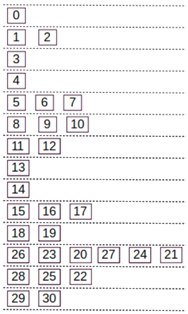
```
bb_5
_16 = (sizetype) idx_28;
_17 = N24_31(D) + _16;
_32 = *_17;
```

```
bb_7
<L4> [0.00%]:
return _20;
```

Control and Data flow graph (CDFG)



Scheduling and Register Insertion



- As Late as Possible (ALAP) scheduling **[TCAD 1991].**
- We obtained the latency of operations from Bambu's **[FPL 2013]** 45 nm Nandgate Open Cell library characterization.
- We insert register when the result of an operation crosses cycle boundary.

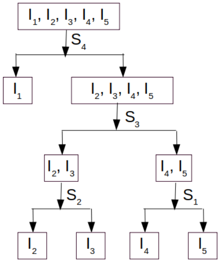
RTL Generation

- We implement each operation using a Verilog module from a component library obtained from Bambu.
- We implement each arrays using register file and SRAM.

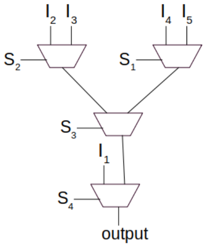
MUX tree generation

Table: An example predicates for a SSA ϕ operation

	Selectors			
Inputs	S_1	S_2	S_3	S_4
I_1	$*$	$*$	$*$	0
I_2	$*$	1	1	1
I_3	$*$	0	1	1
I_4	0	$*$	0	1
I_5	1	$*$	0	1



(a) Predicates splitting tree



(b) Corresponding MUX tree

Evaluation

- We evaluate C2RTL by implementing SAIL **[SIGCOMM 2014]**, Poptrie **[SIGCOMM 2015]** and CP-Trie **[HPSR 2021]** based IPv6 lookup and TabTree **[ANCS 2019]** based packet classification.
- We evaluate the resulting Verilog using Icarus Verilog simulator. It shows that the generated Verilog is functionally correct.

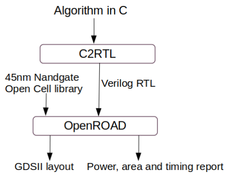

Algorithm in C
C2RTL
45nm Nandgate Open Cell library
Verilog RTL
OpenROAD
GDSII layout
Power, area and timing report

Figure: C code to physical chip layout generation

- We use OpenROAD to generate physical chip layout from the Verilog.

Evaluation in ASIC

	Poptrie	SAIL	CP-Trie	TabTree
Clock speed	1 GHz	1 GHz	1 GHz	1 GHz
Internal Power	76.5 mW	0.722 mW	64.6 mW	0.033 mW
Switching Power	24.4 mW	0.229 mW	22.2 mW	0.0054 mW
Leakage Power	1.15 mW	0.0108 mW	0.926 mW	0.00061 mW
Total Power	102.05 mW	0.961 mW	87.726 mW	0.0391 mW
Area	0.0658 mm^2	0.00061 mm^2	0.0523 mm^2	0.000034 mm^2

Thank You