

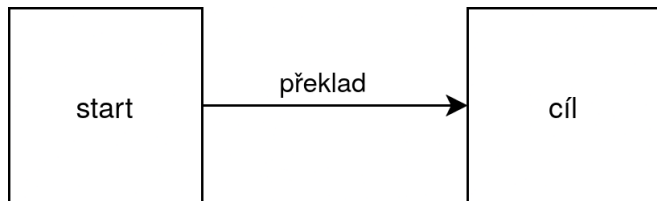
# Zprávy ze zákopů hardwarových IR

Emil J / widlarizer 2025-03-12

# Předem

- ▶ syntézou se myslí logická syntéza, tj. minimalizace logiky a její mapování na výrobitelné/dostupné logické bloky
- ▶ IR je intermediate representation, vnitřní reprezentace

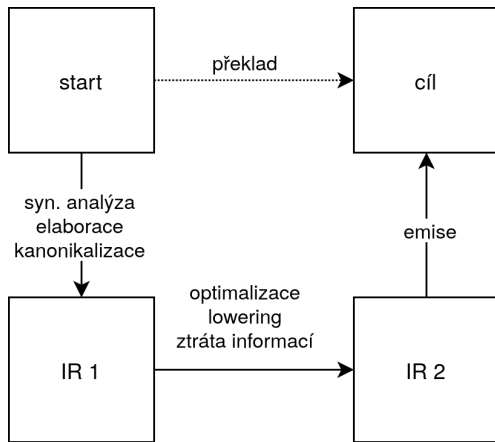
# Kapitola 1: Zadání



Obrázek 1: Naivní pohled

- ▶ src: HDL, HLS
- ▶ dst: RTL, netlist LUT pro FPGA, standardních buněk pro ASIC, C++ simulační model, SMT verifikační model, Redstone, Factorio?

# Kapitola 1: Zadání



Obrázek 2: Možná implementace - 2 IR

## Jak to vypadá pro Verilog?

```
1 wire [7:0] x;  
2 wire mux, a, b;  
3 assign mux = x[0] ? a : b;
```

- ▶ modelujeme `wire` a operace nad nima (hradla, buňky)
- ▶ každý spojení hodnot může řezat a slepovat
- ▶ `wire` může mít libovolný počet zdrojů hodnot (budičů signálu)
- ▶ instance modulu nevyžaduje viditelnost deklarace rozhraní modulu

# Počkat, není to jenom kompilátor?

- ▶ můžeme psát hardware třeba v LLVM IR?

# Můžeme psát hardware třeba v LLVM IR?

Operace mají arbitrární šířky

- ▶ docela detail ale ok

# Můžeme psát hardware třeba v LLVM IR?

## Tok řízení

- ▶ program je posloupnost instrukcí
- ▶ LLVM IR tohle zachovává
- ▶ HDL mají imperativní paradigma jen pro praktičnost
- ▶ neexistuje “instruction pointer”
- ▶ “elaborací” se eliminuje tok řízení na čistě tok dat

```
1  always @(posedge clk)
2      if(s) begin
3          x = a;
4      end else begin
5          x = b;
6      end
```

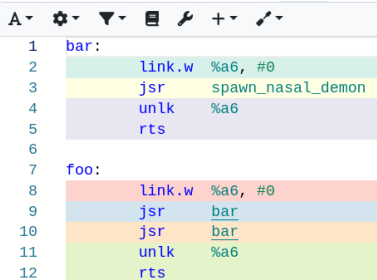
```
1  always @(posedge clk)
2      x <= s ? a : b;
```



# Můžeme psát hardware třeba v LLVM IR?

## Moduly nejsou funkce

```
1  extern void spawn_nasal_demon();
2  void bar() {
3      spawn_nasal_demon();
4      return;
5  }
6  void foo() {
7      bar();
8      bar();
9  }
```



```
A ▾ ⚙ ▾ ▾ 🗒 🛠 + ▾ 🖋 ▾
1  bar:
2      link.w  %a6, #0
3      jsr     spawn_nasal_demon
4      unlk    %a6
5      rts
6
7  foo:
8      link.w  %a6, #0
9      jsr     bar
10     jsr     bar
11     unlk    %a6
12     rts
```

Obrázek 3: Funkce

# Můžeme psát hardware třeba v LLVM IR?

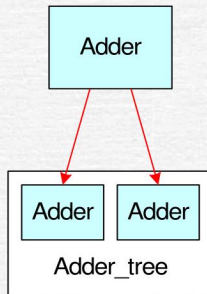
Moduly nejsou funkce

## Module Instantiation

```
module adder(out,in1,in2);  
  output out;  
  input  in1,in2,sel;  
  
  assign out=in1 + in2;  
endmodule
```

instance  
example

```
module adder_tree(out0,out1,in1,in2,in3,in4);  
  output out0,out1;  
  input  in1,in2,in3,in4;  
  
  adder add_0(out0,in1,in2);  
  adder add_1(out1,in3,in4);  
endmodule
```



Obrázek 4: Moduly

## Kapitola 2: Otevřené IR pro hardware

# Verilog

- ▶ otevřený standard
- ▶ nejednoznačný
- ▶ prostě ne

## Yosys / RTLIL

- ▶ IR pro Yosys
- ▶ silně postavený na Verilogu
- ▶ pro celou syntézu, ne jen “interchange format” mezi nástroji

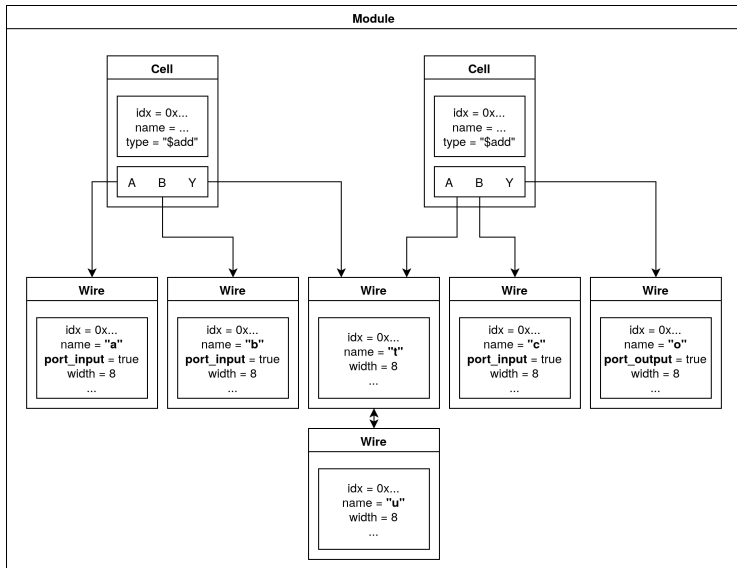
## Yosys / RTLIL

- ▶ akademický experiment který se vymkl z rukou
- ▶ jádro: 2013-2014
- ▶ první open source verilogový frontend
- ▶ první FOSS end-to-end flow pro FPGA (+nextpnr)
- ▶ open source čipy (+OpenROAD)
- ▶ formální verifikace s FOSS SMT solvery
- ▶ prověřený, populární
- ▶ C++17/C++11

## Syntetizuje moduly jako funkce

- ▶ hranice modulu je arbitrární bariéra proti optimalizacím
- ▶ pokud se nepoužije `flatten`
- ▶ `abc` ale zabije src atributy
- ▶ => s `flatten` nezbude žádný “debug info”

# Implementace RTLIL



Obrázek 5:  $t = a + b$ ;  $u = t$ ;  $o = t + c$



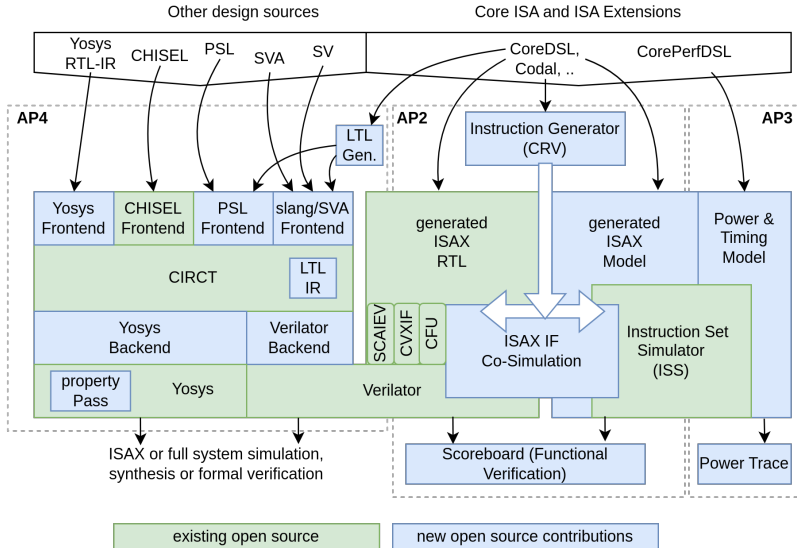
## prjunnamed / Unnamed IR

- ▶ bleeding-edge, vývoj začal koncem ledna letos
- ▶ zatím nic pro hardwaráře
- ▶ masivní zlepšení ve výkonu, paměťové spotřebě, dev experience
- ▶ Rust
- ▶ silná inspirace LLVM a silnými zkušenostmi s Yosys
- ▶ cíle zatím omezeny na FPGA
- ▶ fundamentální omezení
  - ▶ bez `abc`
  - ▶ interpretace `x/undef` pro jednoznačnost
  - ▶ vyjádření časování
- ▶ nehierarchická reprezentace
  - ▶ hierarchie je jen metadata
  - ▶ optimalizace napříč moduly
- ▶ koncepty odprototypovány v Amaranth HDL

- ▶ MLIR navazuje na LLVM styl pro širší spektrum domén
- ▶ GPU, akcelerátory, potenciálně hardware
- ▶ hezky formální a konzistentní framework
- ▶ tablegen na definici tablegenu
- ▶ Chisel - FIRRTL - Yosys
- ▶ nevěřím overhead pro syntézu

- ▶ goto <https://llvm.org/devmtg/2021-11/slides/2021-RepresentingConcurrencywithGraphRegionsinMLIR.pdf>
- ▶ goto godbolt - doma >.>

## ► Projekt Open Source Verification of Instruction Set Extensions



# Otázky?

- ▶ hráli jste si s MLIR?
- ▶ chcete slyšet něco o syntéze?
- ▶ jak to vidíte s večerí?
- ▶ goto [@the\\_art\\_of\\_giving\\_up@mastodon.social](https://mstdn.social/@the_art_of_giving_up)