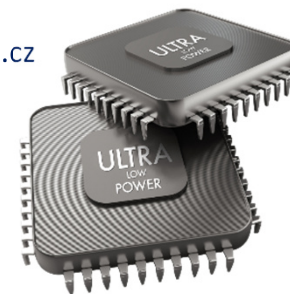
ASICentrum A COMPANY OF THE SWATCH GROUP

em microelectronic A COMPANY OF THE SWATCH GROUP

www.asicentrum.cz

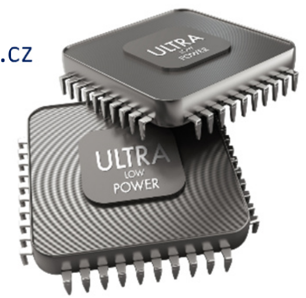


Simulace číslicových obvodů

ASICentrum, s.r.o.

Jakub Šťastný, Ph.D.

jakub.stastny@asicentrum.cz



Nejčastější činnost při návrhu číslicového systému

Návrh číslicového systému:

- modelování na různé úrovni abstrakce
- od specifikace
- po integrovaný obvod

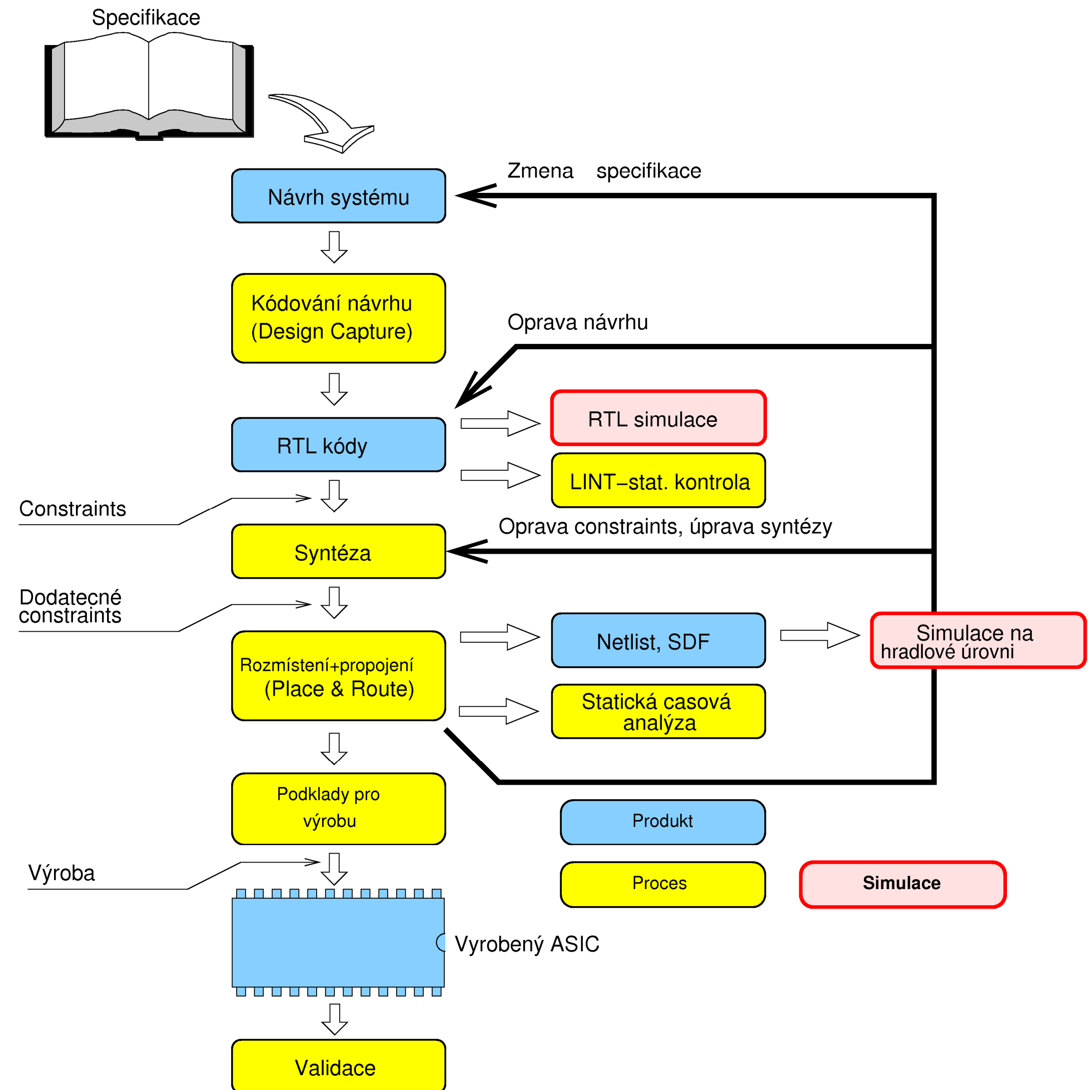
Nejčastější činnost: verifikace

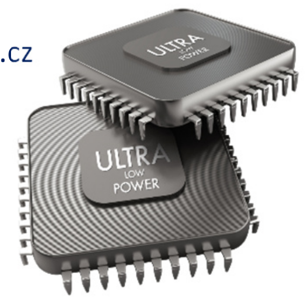
- ověřujeme, že návrh je bez chyb
- zajímá nás shoda se specifikací
- až 80% času strávíme verifikací a tedy často i simulacemi

Simulacemi trávíme většinu času.

Dnešní program:

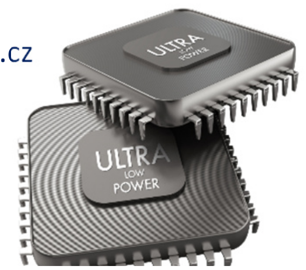
1. úrovně abstrakce
2. simulace na hradlové úrovni
3. kdy je „dosimulováno“?
4. náhodná čísla a simulace
5. náměty k dalšímu studiu





Úrovně abstrakce

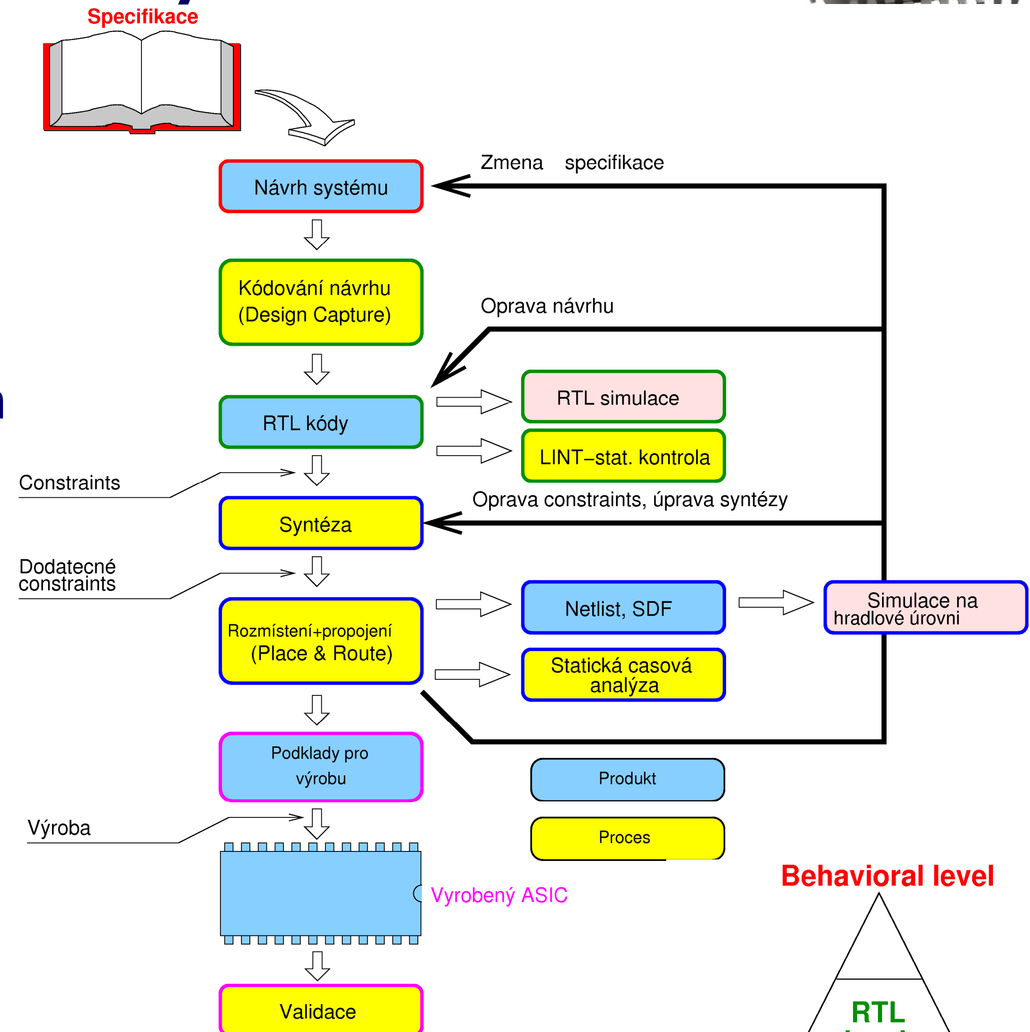
Abstrakce (z lat. *abs-trahere*, odtáhnout, odvléci, oddělit) ve filozofii označuje buď důležitý moment procesu poznání při přechodu od smyslového k racionálnímu poznání, nebo hotový výsledek tohoto procesu. Proces abstrakce spočívá obecně v tom, že existuje řada analytických aktů myšlení, jimiž je zpracováván konkrétní smyslový materiál a při nichž se odhlíží od určitých znaků, vlastností a vztahů daného předmětu. Jiné znaky, vlastnosti a vztahy jsou naopak vyčleňovány jako podstatné. [Wikipedia, https://cs.wikipedia.org/wiki/Abstrakce]

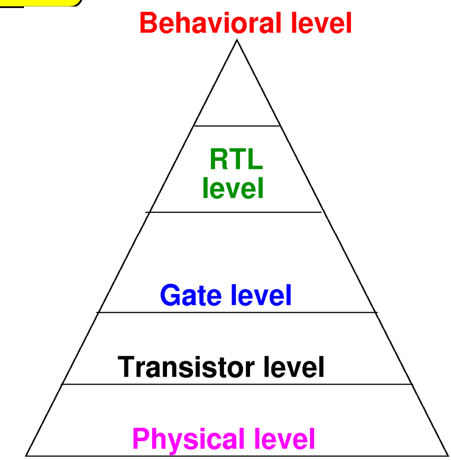


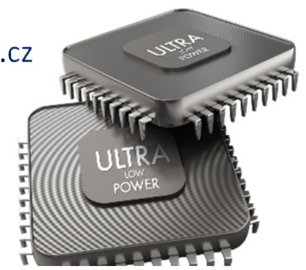
Modelování číslicového systému

- **Proces návrhu**
  - **tvoříme postupně se zpřesňující model**
  - **převádíme popis (model) návrhu mezi úrovněmi abstrakce**
- **Čím vyšší úroveň abstrakce, tím**
  - **potřeba větší rezervy**
  - **a méně detailů**
- **Přechod na nižší úroveň abstrakce**
  - **detaily přidává**
  - **zvyšuje složitost modelu**
  - **náročnější implementace**
  - **ovšem model je blíže realitě**
  - **pokud možno automatický**
    - **vs ruční**

K čemu potřebuji abstraktnější modely? Proč všechno nedělat na tranzistorové úrovni?





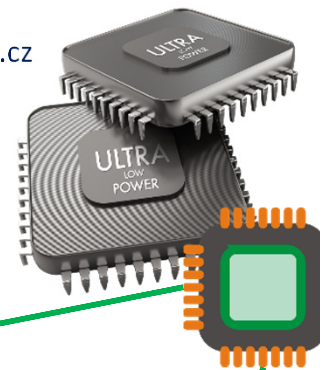


Přechody a trasovatelnost

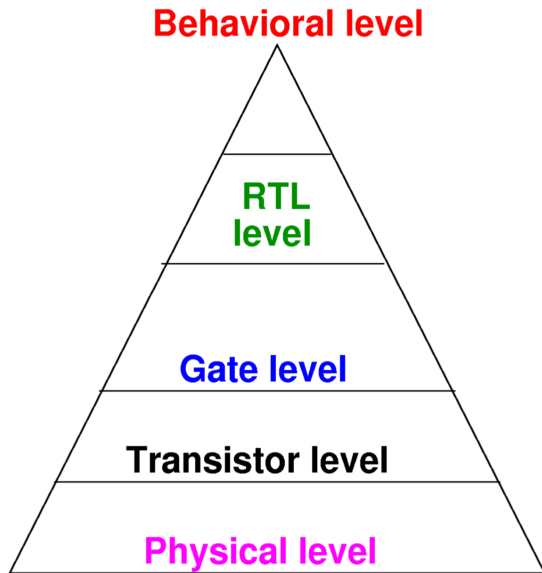
- **Modely na vyšší úrovni abstrakce**
  - **se snáze implementují**
  - **simulace běží rychleji**
  - **někdy je používáme jako modely pro verifikaci modelů na nižší úrovni abstrakce**
- **Abstraktnější model**
  - **nezachycuje řadu detailů**
  - **některé typy chyb pak nelze vůbec detekovat**
  - **ale rychlejší běh simulace umožňuje najít zase jiné druhy chyb...**
  - **příklad: akcelerátor pro rozpoznávání řeči**

Specifikace
Návrh systému
Zmena specifikace
Kódování návrhu (Design Capture)
Oprava návrhu
RTL kódy
RTL simulace
LINT–stat. kontrola
Constraints
Syntéza
Oprava constraints, úprava syntézy
Dodatecné constraints
Rozmístení+propojení (Place & Route)
Netlist, SDF
Simulace na hradlové úrovni
Statická casová analýza
Podklady pro výrobu
Produkt
Proces
Výroba
Vyrobený ASIC
Validace

Behavioral level
RTL level
Gate level
Transistor level
Physical level



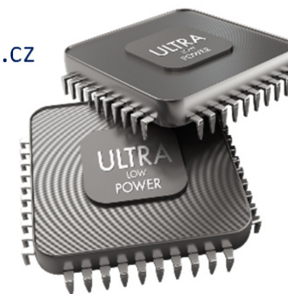
Hierarchie návrhu a úrovně abstrakce



tb_top
testbench top
"model vseho, co je na cipu"
Zakaznicky obvod
strukturní popis
vdd_ext
vdd_low_en
regulátor
vdd
reset_gen
por
clk_gen
osc_en
osc
tepl. senzor
ADC
vstupní rozhraní
výstupní rozhraní
vdd vdd2
Digitální jádro
dig_top
DUT
Design Under Test
stimuli_gen
osc
=
OK/FAIL
DUT
model
(prediktor)

Analogové bloky – návrh na tranzistorové úrovni (… layout), nebo RNM modely pro verifikaci (behav.)

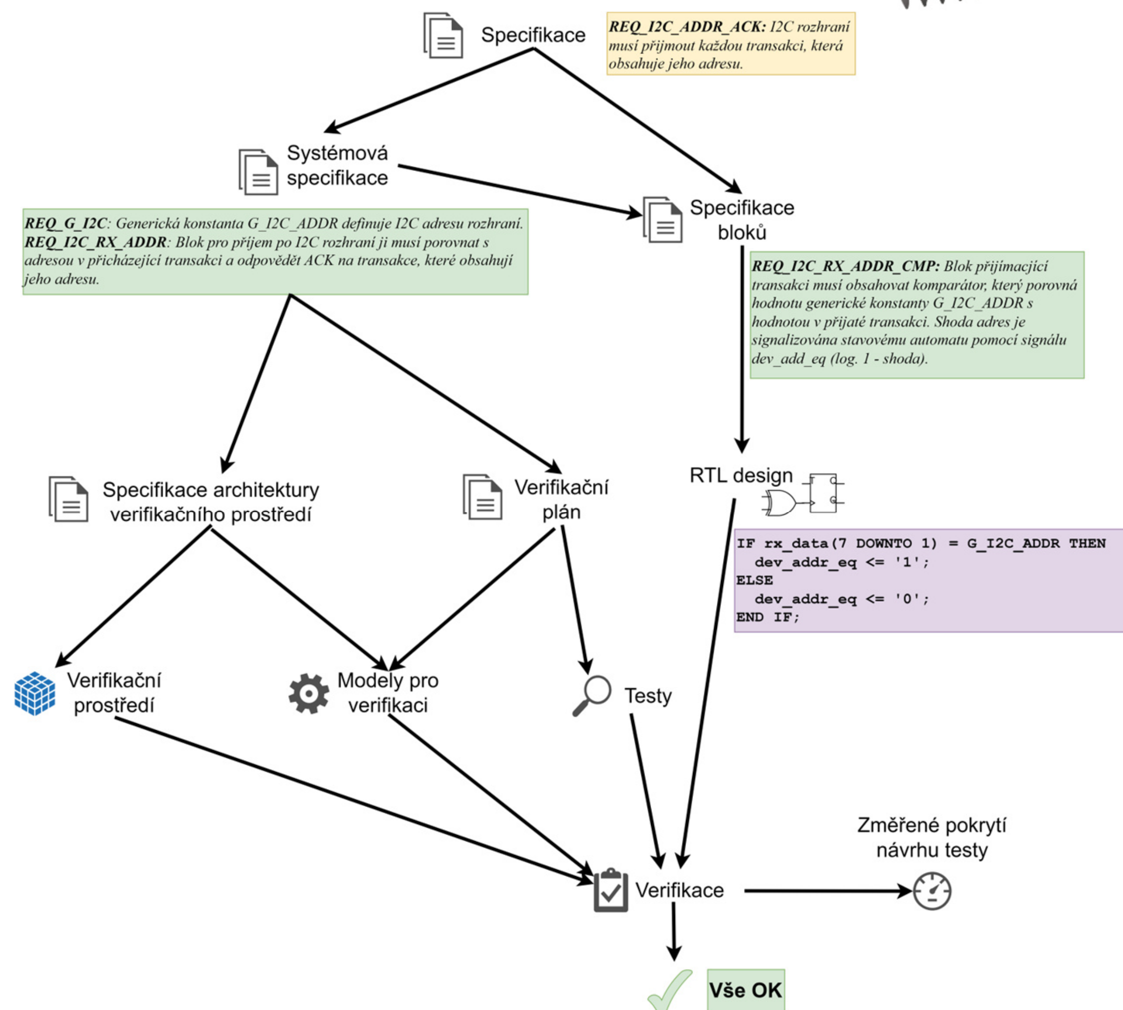
Vyšší úrovně abstrakce jsou vhodné pro modelování – ověření správné funkce.

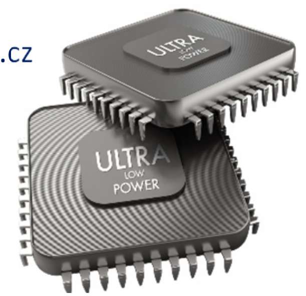


Vzájemné propojení modelů

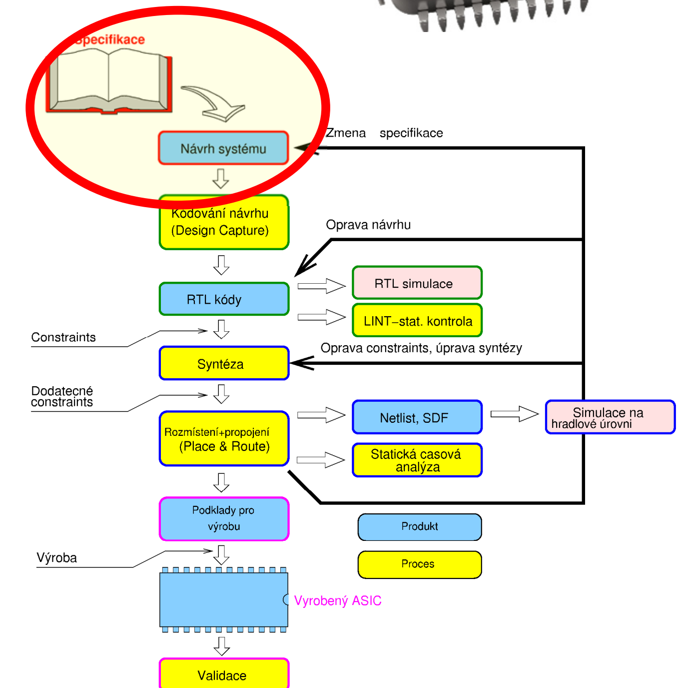
- **Trasovatelnost**
- **Každý požadavek ze specifikace...**
- **musí být někde implementován**
- **musí být někde/nějak verifikován**
- **vzájemné propojení artefaktů a jejich částí**

Změna jednoho modelu se musí promítnou v odpovídajících částech ostatních modelů.





Behaviorální úroveň

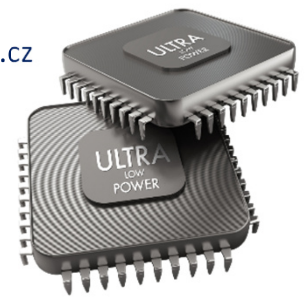


Příklad: popis obvodu pro sečtení čtyř čísel:

`e = a + b + c + d`

Je to jeden řádek kódu.

- **Popis chováním**
  - **algoritmus v programovacím jazyce**
  - **programujeme**
  - **jeden řádek kódu = i tisíce hradel**
  - **př. dělička je implementována operátorem dělení**
- **Obvykle**
  - **nepracujeme s hodinami**
  - **neužíváme stavové automaty**
  - **volně nakládáme s pamětí**
- **Lze odladit správnou funkci algoritmů**
  - **např. algoritmy číslicového zpracování signálů**
  - **model platformy, tzv. virtuální platforma**
- **Jazyky - C++, Matlab, SystemVerilog nebo VHDL**
- **Používané datové typy**
  - **log. úrovně (pokud vůbec) modelovány jako true/false**
  - **časté použití složitých datových typů**



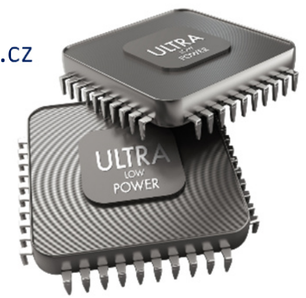
RTL úroveň (Register Transfer Level)

4 bity a b c d
a_in b_in c_in d_in
5 bitu
a_b c_d
clk res
6 bitu
e

e = a + b + c + d

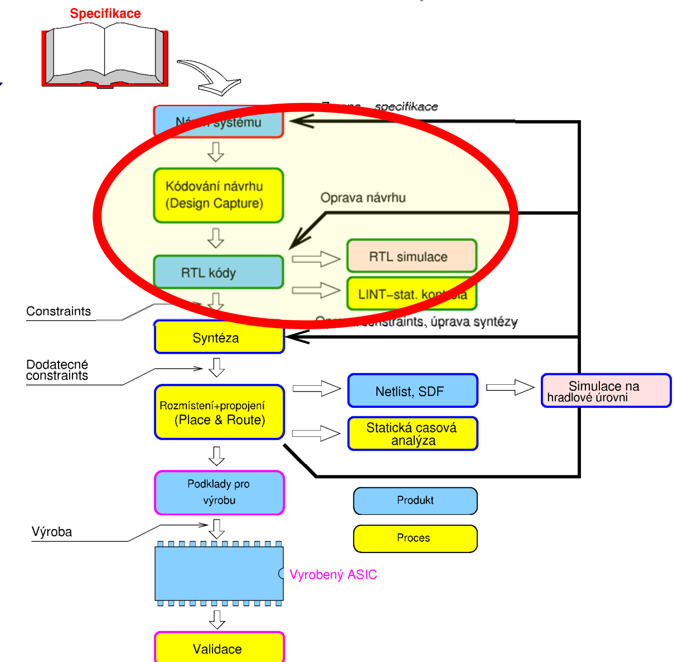
```
 SIGNAL a : unsigned (3 DOWNTO 0);
 SIGNAL a_b : unsigned (4 DOWNTO 0);
…
  add : PROCESS (clk, res)
  BEGIN
    IF (res='1') THEN
      a_in <= 0; b_in <= 0;
      c_in <= 0; d_in <= 0;
      a_b  <= 0; c_d <= 0;
    ELSIF (rising_edge(clk)) THEN
      a_in <= a; b_in <= b;
      c_in <= c; d_in <= d;
      a_b  <= a+b;
      c_d  <= c+d;
      e    <= a_b+c_d;
    END IF;
  END PROCESS add;
```

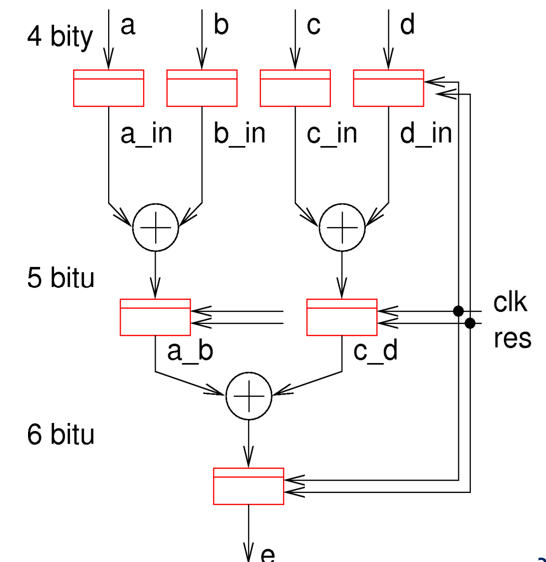
Jedno sčítání čtyř čísel teď zaplní celý slajd :-).
(a to jsem to ještě zjednodušil)

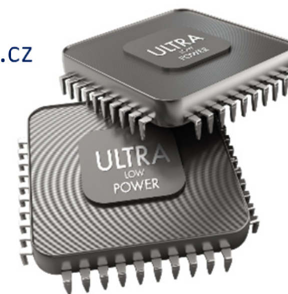


RTL úroveň (Register Transfer Level)

- **Popisuje *mikroarchitekturu* navrhovaného systému**
  - **jako registry propojené kombinačními bloky**
  - **jeden řádek kódu = desítky až stovky hradel**
  - **př. dělička je implementována jako sekvenční obvod**
- **Lze odladit**
  - **implementaci mikroarchitektury systému**
  - **ověříme, zda je systém funkčně správně**
  - **zda obvod reaguje jak jsme si přáli**
- **Simulace nerespektuje reálná zpoždění**
  - **ale je velmi rychlá**
- **Nelze odladit**
  - **správnost interního časování**
    ... zpoždění na kombinačních prvcích, apod.
- **Používáme a popisujeme**
  - **hodiny a reset**
  - **stavové automaty**
  - **kombinační operátory**
  - **strukturní popis**
  - **už nemáme rádi velké paměti**
- **Obvykle VHDL, Verilog nebo SystemVerilog**
- **Používané datové typy**
  - **logické úrovně modelovány jako 1,0 (XZHLWU-)**
  - **občas použití složených typů (record, array)**





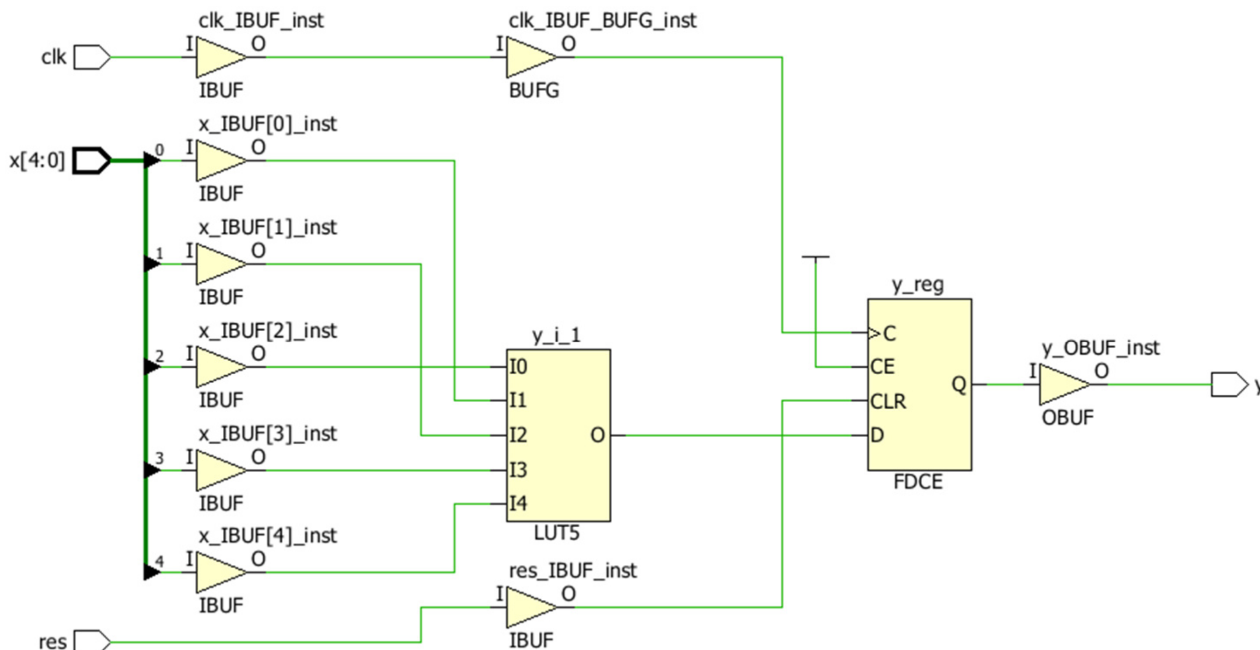


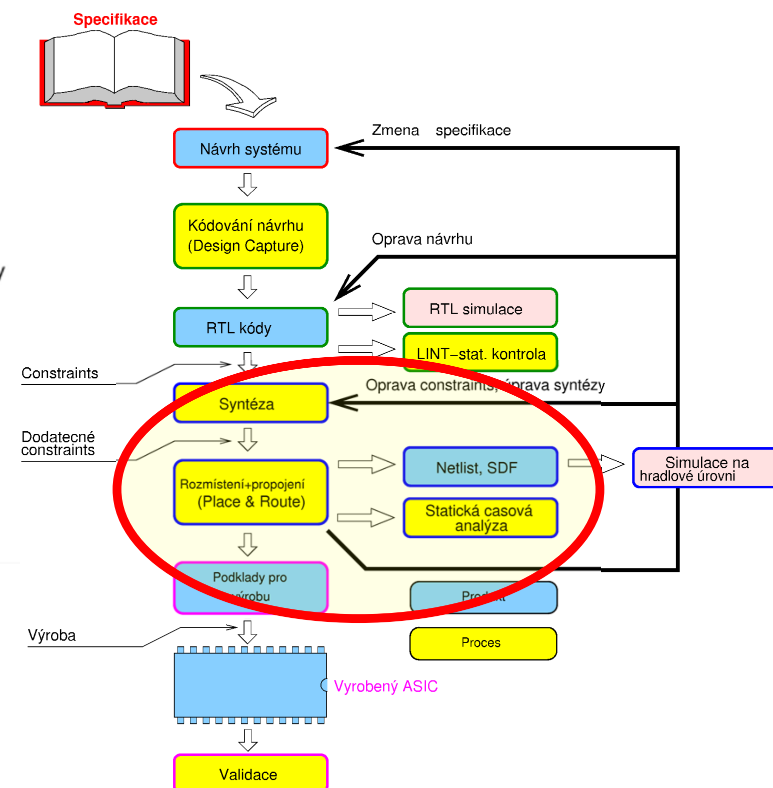
Hradlová úroveň (Gate Level)

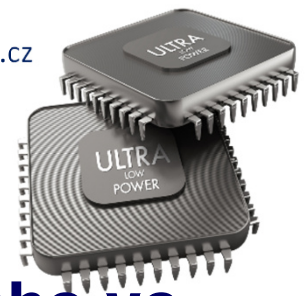
Jednoduchý obvod, který realizuje

`y = x(0) OR x(1) OR x(2) OR x(3) OR x(4)`

zaplní i několik slajdů.

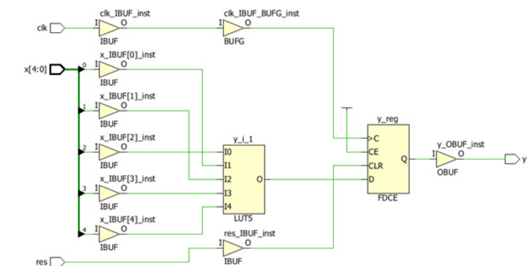


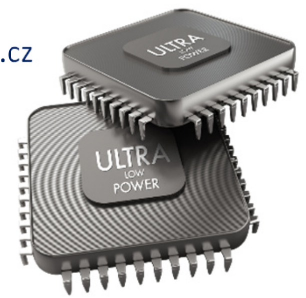




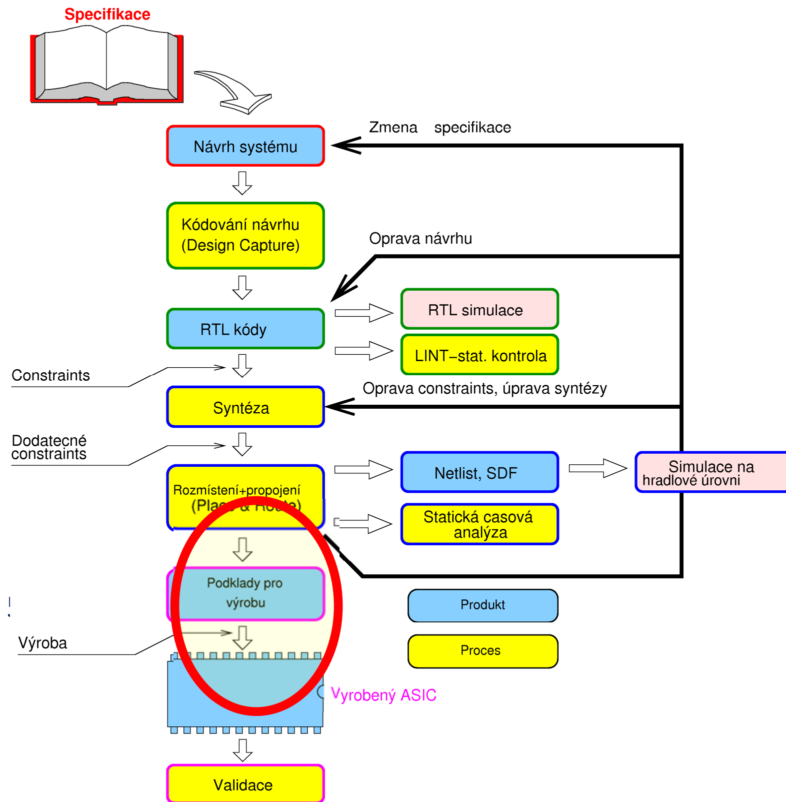
Hradlová úroveň (Gate Level)

- **Simulujeme finální podobu obvodu implementovaného ve zvolené technologii**
  - **práce s elementárními logickými buňkami (NAND, …)**
- **Návrh je popsán jako**
  - **schéma obvodu ve zdrojovém kódu (*netlist*)**
  - **před simulací anotován informacemi o zpoždění na obvodových prvcích a spojích (*back annotation*)**
  - **obvykle nepíšeme „ručně“, výstup syntézy z RTL a PnR**
  - **Obvykle Verilog netlist nebo VHDL netlist + SDF**
  - **několik řádků kódu = i jen jedno hradlo**
  - **př. dělička implementována**
    - **jako sekvenční obvod realizující dělení**
    - **včetně všech hradel, bufferů, atd.**
    - **včetně detailního hodinového stromu**
- **Modelujeme reálná zpoždění na prvcích, detailní simulace**
- **Lze odladit detailní časování obvodu**
- **Používané datové typy**
  - **jen logické úrovně**





Tranzistorová úroveň (transistor level)



- **Analogové schéma obvodu**

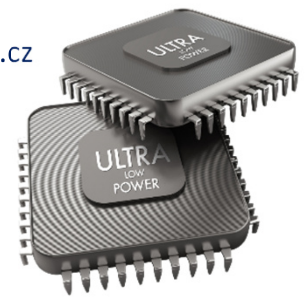
Simulátor pracuje s obvodovými veličinami (U-I)

- **řeší soustavy diferenciálních rovnic**
- **→ vyšší přesnost simulace**
- **významné zpomalení jejího běhu**
- **místo logické 0/1 – celá škála analogových hodnot**

Používáme analogové simulátory

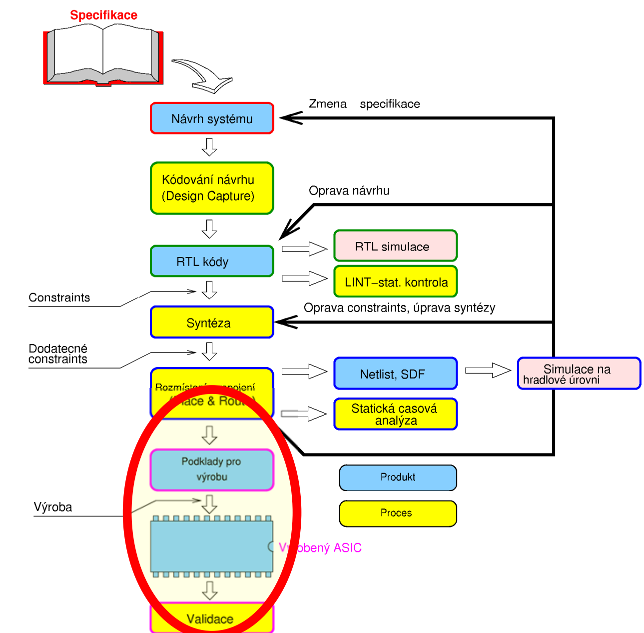
Lze odladit např.

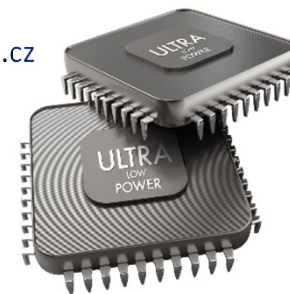
- **chyby v rozhraních mezi analogovými a číslicovými bloky v systému**
- **chování „za číslicovou abstrakcí"**



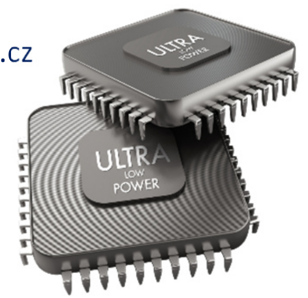
Fyzická úroveň

- **Podklady pro výrobu masek**
- **GDS formát**
  - **Graphics Database System**
- **GDS obsahuje popis layoutu v geometrické podobě**

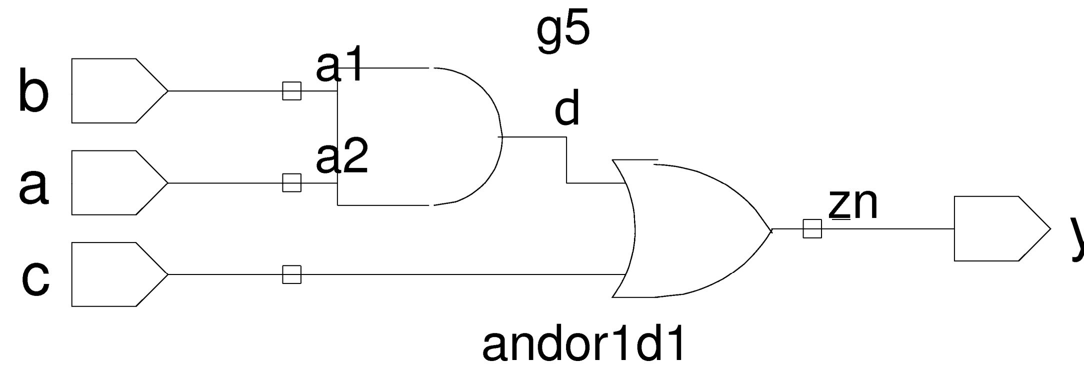




Událostmi řízená simulace



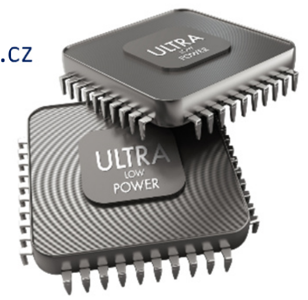
Událostmi řízená simulace - malé cvičení



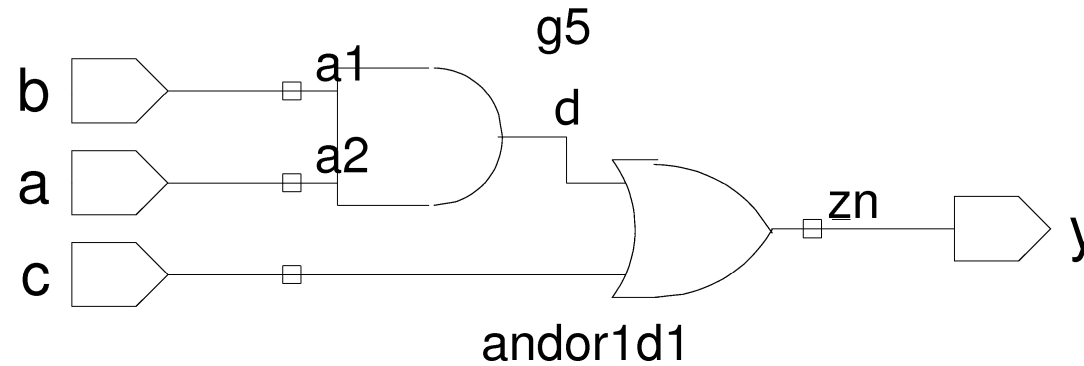
drobné cvičení ve VHDL:

```
log: PROCESS (a,b,c)
BEGIN
  d <= a AND b;
  y <= d OR c;
END PROCESS log;
```

Vlnky si namalujeme na tabuli….



Jak je to možné?



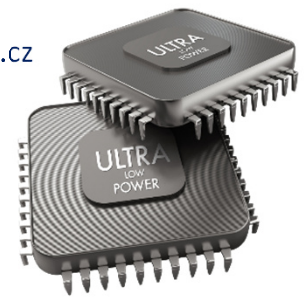
log: **PROCESS** (a_t,b_t,c_t)

BEGIN

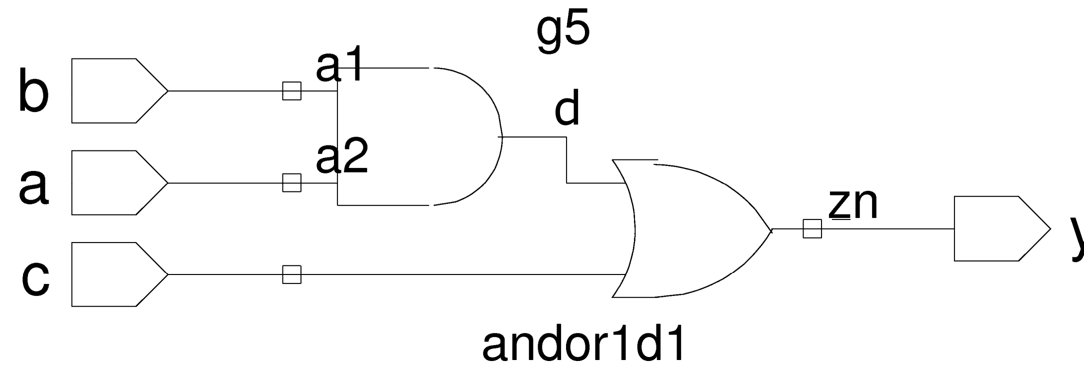
  d_{t+1} <= a_t **AND** b_t;

  y_{t+1} <= d_t **OR** c_t;

END PROCESS log;

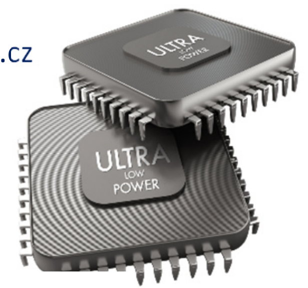


Správné řešení - 1

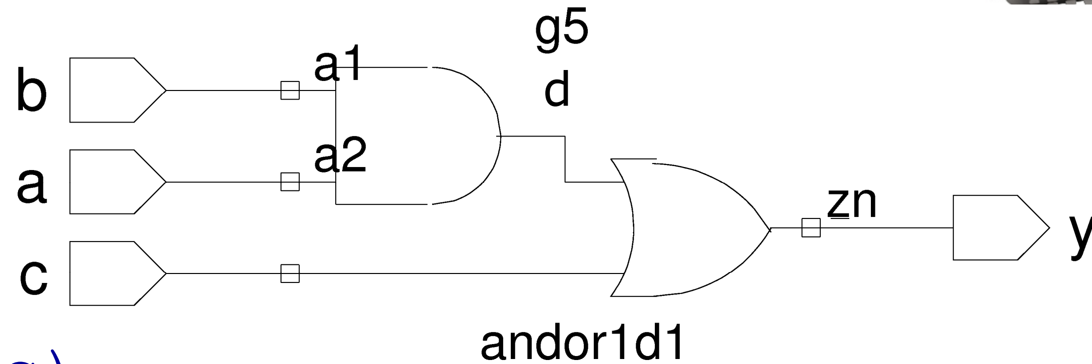



```
log: PROCESS (a_t,b_t,c_t,d_t)
BEGIN
  d_t+1 <= a_t AND b_t;
  y_t+1 <= d_t OR c_t;
END PROCESS log;
```

1. Napadají vás ještě jiná řešení?
2. Jak na **takovou** chybu přijít?



Správná řešení - 2



Proměnná vs signál?

```
log: PROCESS (a,b,c)
  VARIABLE d: std_logic;
BEGIN
  d := a AND b;
  y <= d OR c;
END PROCESS log;
```

také lze ☺

```
y <= c OR (a AND b);
```

drobné cvičení ve VHDL:

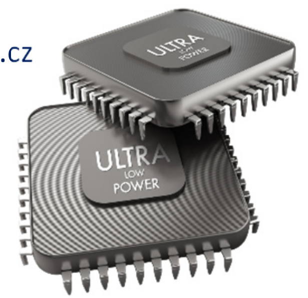
```
log: PROCESS (a,b,c)
BEGIN
 d <= a AND b;
 y <= d OR c;
END PROCESS log;
```

Dá se na to nějak přijít? .. je to vážná chyba!

vcom -check_synthesis:

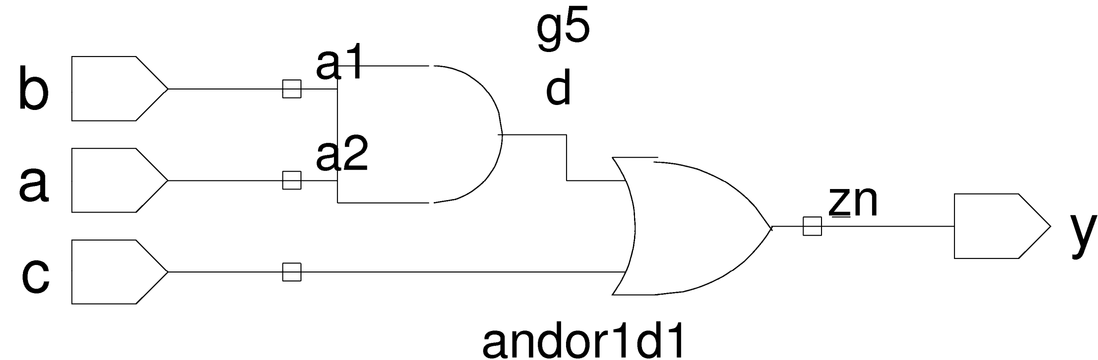
*# ** Warning:...: (vcom-1400) Synthesis Warning: Signal "d" is read in the process but is not in the sensitivity list.*

Je to „jenom warning...“: Varování neignorujeme!

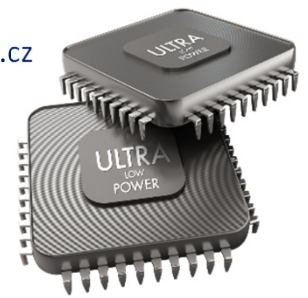


Simulace řízená událostmi

```
log: PROCESS (a,b,c,d)
BEGIN
 d <= a AND b;
 y <= d OR c;
END PROCESS log;
```



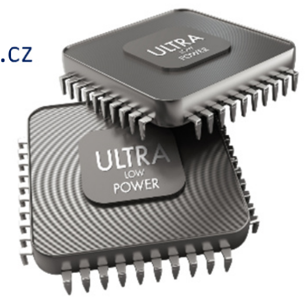
- **Bavíme se o „šíření událostí"**
- **Tzv. event-driven simulation**
- **Rychlá, simulátor počítá jen to, co se mění**
- **Jádro simulátoru je tzv. fronta událostí**
  - **vkládáme nové, když se „něco mění"**
  - **vybíráme, když vyhodnocujeme další krok**
- **Událost = (uzel, čas kdy se má měnit, hodnota na kterou se má měnit)**



Událostmi řízená simulace

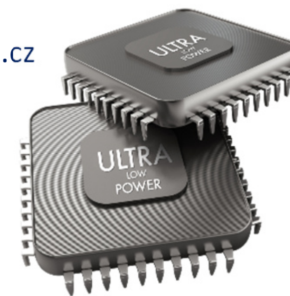
Zjednodušený simulační algoritmus:

1. **Elaborace návrhu: načtení už zkompilovaného návrhu, kompletace hierarchie, expanze generických konstrukcí, kontrola**
2. **Start simulace: uzlům v obvodu přiřazeny a dopočteny počáteční stavy, naplánovány události**
3. ***t_c = t_n***
4. **Vyjmeme událost z fronty a zpracujeme ji; každý proces, který má ve svém citlivostním seznamu signál na kterém se vyskytla událost, je vykonán. Vzniknou další události a jsou vloženy do fronty událostí.**
5. **Čas příštího simulačního cyklu *t_n* se nastaví na nejbližší z**
   - **kdy je další budič ve frontě událostí je aktivní**
   - **čas ve kterém se probudí nějaký proces v návrhu (uspaný např. pomocí příkazu WAIT).**
     Pokud *t_n = t_c*, pak je další simulační cyklus tzv. delta cyklus.
   - **TIME'HIGH**
6. **Simulátor pokračuje krokem 4.**

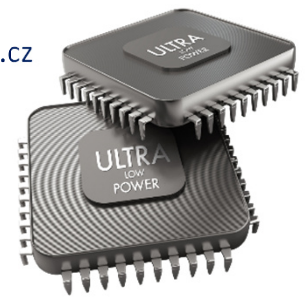


Implikace pro praxi

- **Doba běhu simulátoru roste**
  - **s počtem událostí, které jsou modelovány.**
    - **s velikostí návrhu**
    - **s počtem hodinových cyklů, které synchronní číslicový obvod zpracovává a**
- **Simulace se zpomaluje směrem k nižším úrovním abstrakce, kde je třeba modelovat více prvků detailněji.**
- **Simulace je ukončena (a jak na to…)**
  - **když je fronta událostí prázdná**
    - **přestat generovat hodiny pro obvod**
    - **přestat stimulovat jeho vstupy.**
  - **také lze**
    ```
    VHDL  ASSERT (false) REPORT „konec simulace“
    SEVERITY FAILURE;
    ```
  - **nebo VHDL 2008 `stop` a `finish`**
- **Události lze ukládat a zkoumat zpětně (pomocí wave file).**



Simulace na hradlové úrovni

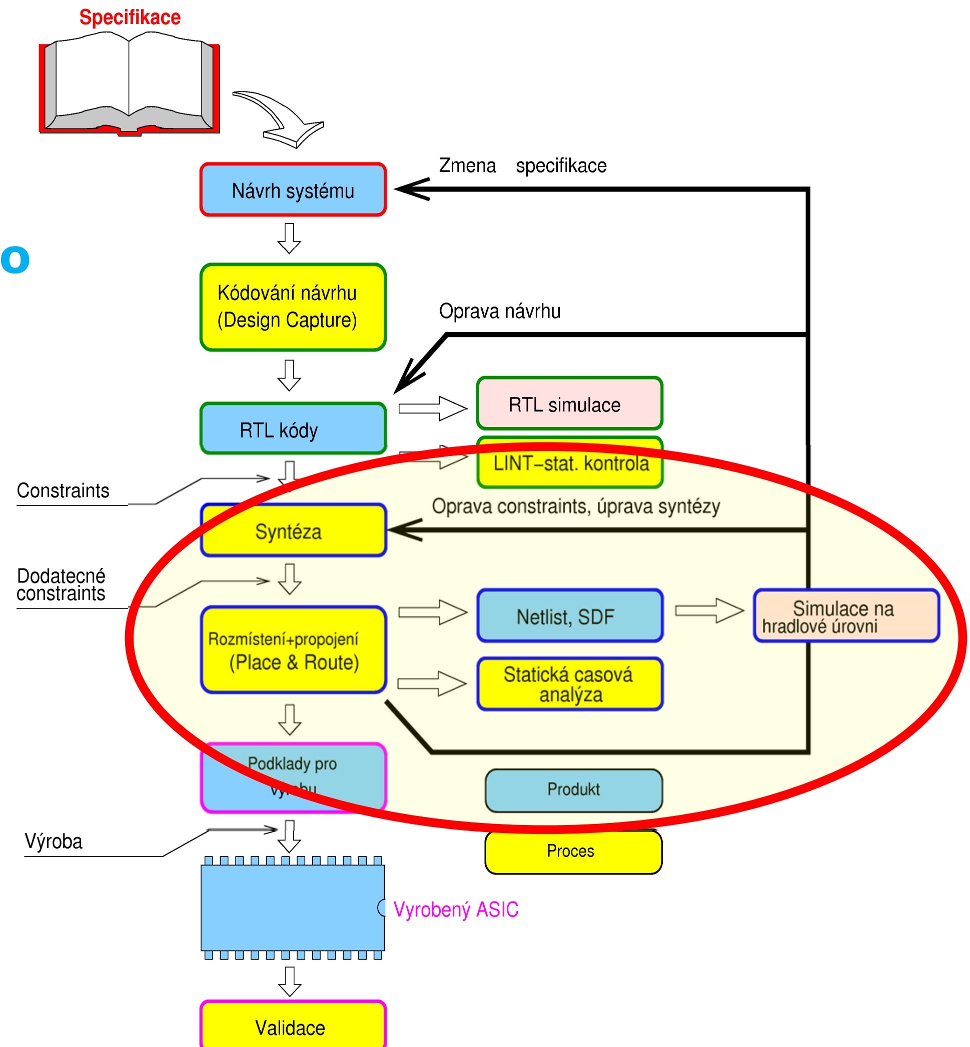


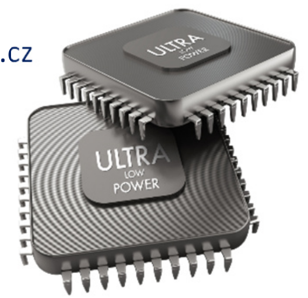
Hradlová úroveň (Gate Level)

Z nástrojů, které má digitální návrhář k dispozici, simulace na hradlové úrovni nejlépe odráží reálné chování číslicového návrhu.

Jak se k ní dostaneme:

- **Model na hradlové úrovni nepíšeme ručně**
- **Výstup**
    - **syntézy z RTL kódu**
    - **následuje rozmístění a propojení**
- **"Ruční" práce na této úrovni**
    - **př. ECO změny návrhu**





Motivace: proč simulaci na hradlové úrovni?

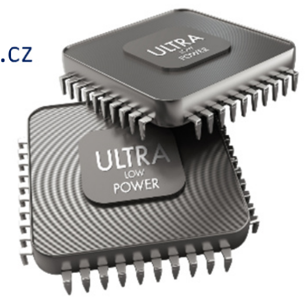
Co získáme?

- **vidíme reálná zpoždění na obvodových prvcích:**
- **simulátor během simulace kontroluje dodržení časových parametrů klopných obvodů**
  - **můžeme odladit problémy, které na RTL úrovni nevidíme**
  - **známe časové parametry klopných obvodů?**
- **potvrdíme výstupy statické časové analýzy**
  - **víme, co je STA?**
- **zverifikujeme správné chování návrhu**
  - **po resetu**
  - **po připojení napájecího napětí**
- **můžeme odhadnout dynamickou spotřebu elektrické energie**
- **návrhář má lepší pocit z návrhu: jasně vidím, že „to funguje“.**
- **… a další, viz citace v doporučené literatuře**

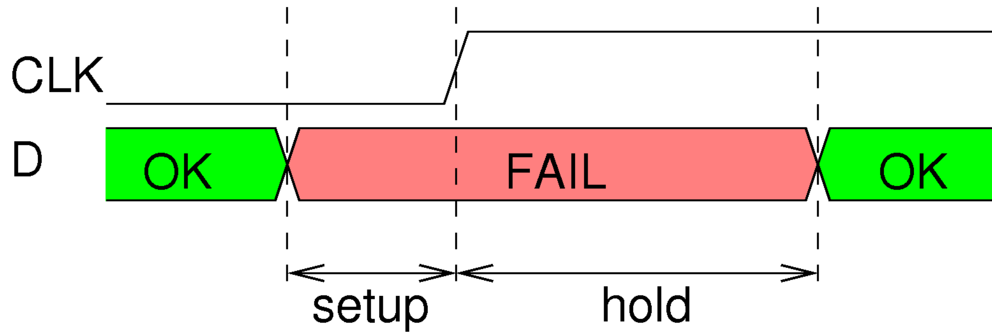
Simulace je ovšem jen tak dobrá, jak dobré jsou stimuly!

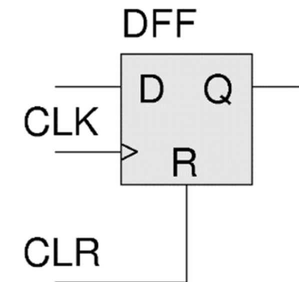
Nevýhody:

- **Nastavit a rozběhnout simulaci na hradlové úrovni dá práci**
  - **musíme odladit verifikační prostředí**
  - **simulátor běží pomaleji, než na RTL úrovni**

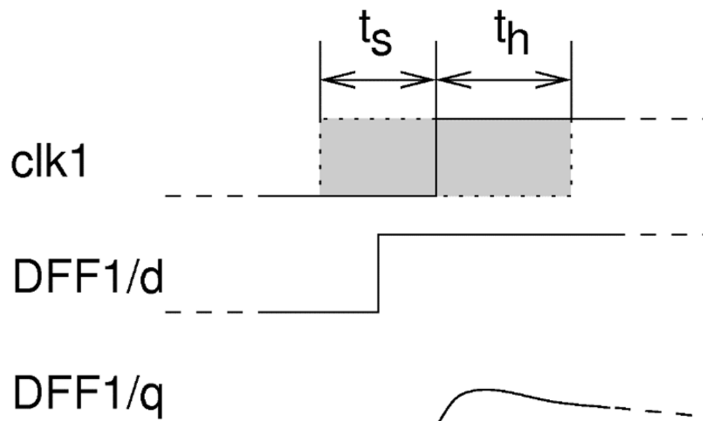


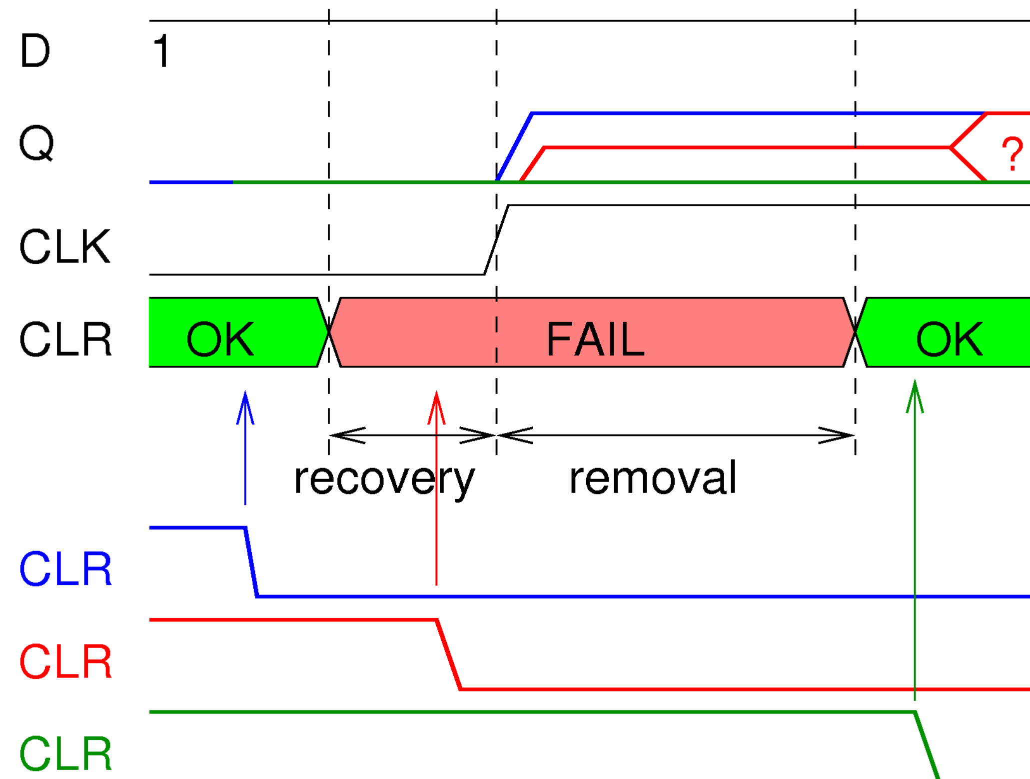
Automatické kontroly časových parametrů klopných obvodů....?

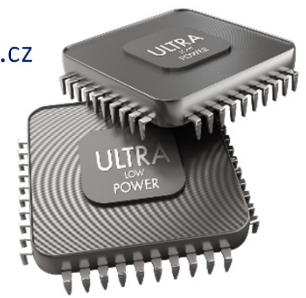




Co když porušíme ...?







Vstupy pro simulaci a její výstup

SDF = Standard Delay File

Netlist
Schéma zapojení v hradlech → Kompilace

SDF soubor
Zpoždění na buňkách, spojích a časové parametry klopných obvodů pro kontrolu za běhu simulace → Kompilace

Tech. knihovna
Modely všech buněk („hradel“) v příslušné technologii (např. TSMC, 180 nm) → Kompilace

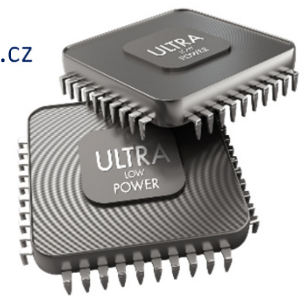
Testbench → Kompilace

Kompilace → **Elaborace** → **Simulace** → Log simulátoru

T-file
Seznam vypnutých kontrol časových parametrů - typicky např v synchronizátoru na 1. DFF předstih. → Simulace

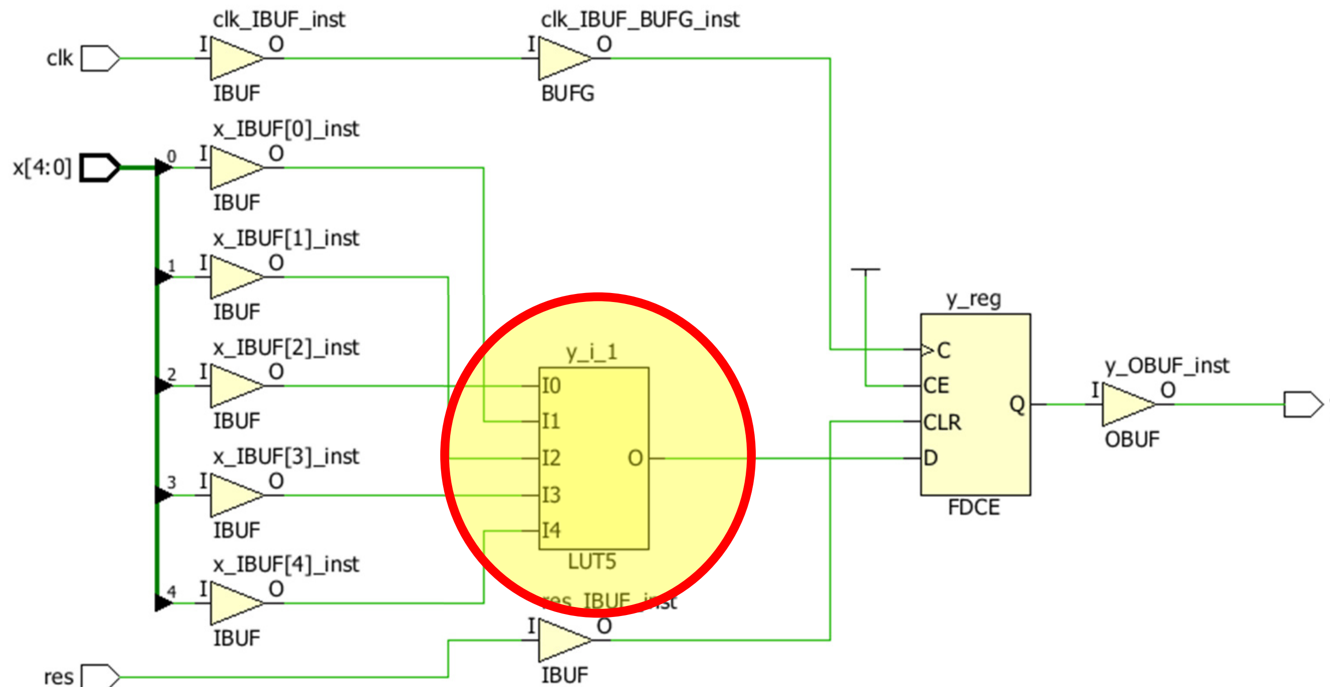
Log simulátoru:

```
SYSTEM_SETUP: RC_OSC_FREQ=4.73934MHz    Time: 0 ns
INFO: STARTING simulation.    Time: 0 ns
--------------------------------------------------
Generate reset: power-off + 800 ns + power-up    Time: 0 ns
--------------------------------------------------
Warning! Timing violation
         $setuphold<setup>( negedge CPN &&& SI_DEFCHK:2780 PS, negedge SI:2480 PS,  0.94 : 940 PS,  0.00 : 0 FS));
         File: /work/demo/wrk/regression/src/stdcells/dff2_res.v, line = 24641
         Scope: :I_DUT_TOP.IDIGITAL_TOP.IDIG_TOP.i_test_top.i_res_sync.res_sync_reg_1_
         Time: 2780 PS
```

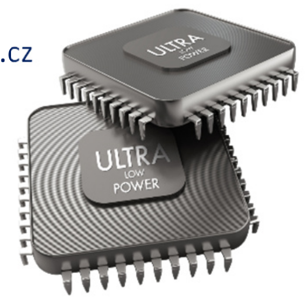


Hradlová úroveň - příklad netlistu

y = x(0) OR x(1) OR x(2) OR x(3) OR x(4);



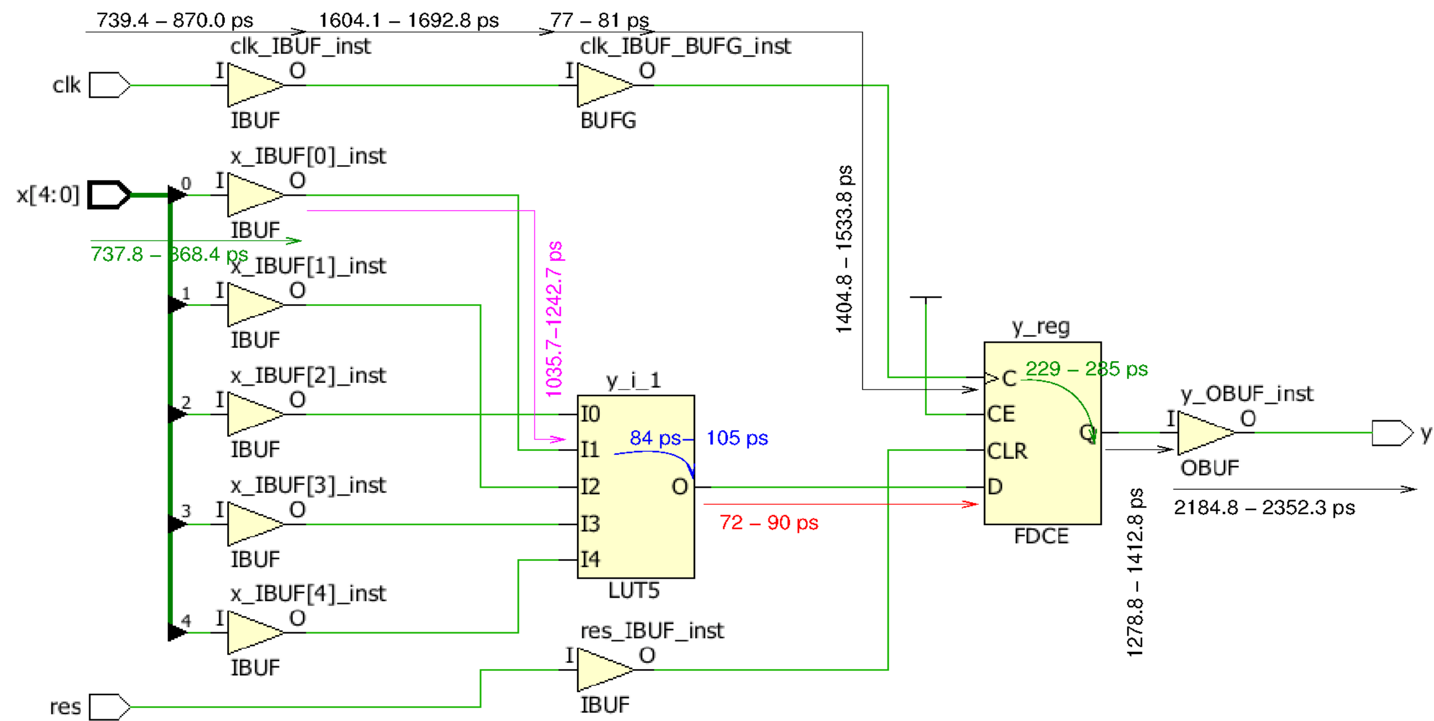

```
IBUF \x_IBUF[0]_inst
      (.I(x[0]),
       .O(x_IBUF[0]));
... vynechané podobné instance
OBUF y_OBUF_inst
      (.I(y_OBUF),
       .O(y));
LUT5 #(
   .INIT(32'hFFFFFFFE))
   y_i_1
      (.I0(x_IBUF[2]),
       .I1(x_IBUF[0]),
       .I2(x_IBUF[1]),
       .I3(x_IBUF[3]),
       .I4(x_IBUF[4]),
       .O(y_d));
```



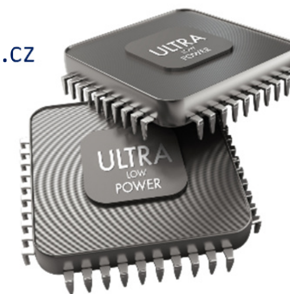
Hradlová úroveň – anotace zpoždění (SDF)

```
//Popis zpoždění registru
(CELL
 (CELLTYPE "FDCE")
 (INSTANCE y_reg)
 (DELAY                              (MIN:TYP:MAX)
  (ABSOLUTE                          0→1, 1→0
   (IOPATH C Q (229.0:285.0:285.0) (229.0:285.0:285.0)
  )
 )
)
```

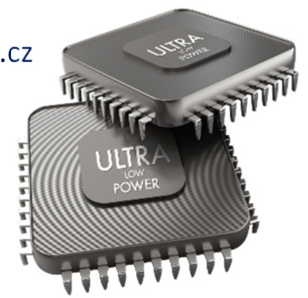
```
(CELL
  (CELLTYPE "BUFG")
  (INSTANCE clk_IBUF_BUFG_inst)
  (DELAY
    (PATHPULSE (50.0))
    (ABSOLUTE
      (IOPATH I O (77.0:81.0:81.0) (77.0:81.0:81.0))
. . .
```




```
//Anotace zpoždění na spojích
(CELL
    (CELLTYPE "or_block")
    (INSTANCE )
    (DELAY
      (ABSOLUTE
      (INTERCONNECT x_IBUF\[0\]_inst/O y_i_1/I1 (1035.7:1242.7:1242.7) (1035.7:1242.7:1242.7))
      (INTERCONNECT y_i_1/O y_reg/D (72.0:90.0:90.0) (72.0:90.0:90.0))
```



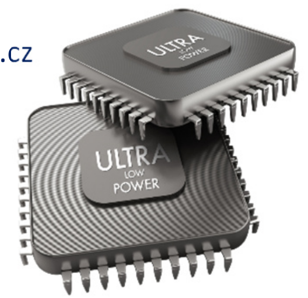
Jak poznáme, že jsme simulovali už dost?



Motivace

- Verifikace může představovat až 80% práce na projektu
- Zjednodušený postup při verifikaci:
  1. sestavíme seznam požadavků na návrh
  2. sestavíme verifikační plán
     - na každý jeden/více požadavků je jeden testcase
     - testů mohou být i stovky
     - testy jsou „selfchecking“ – OK/FAIL
  3. verifikátoři implementují model systému
  4. verifikátoři píší testy
- Potřebujeme vědět, kdy je hotovo…
  - co myslíte, kdy je hotovo?

ID	Requirement	test_spi_01	test_spi_02	rest_reg_map
SPI.1	All polarities and phases shall be supported by the interface, selection shall be done via the **spi_pol** and **spi_pha** registers.	Covered	covered	covered
SPI.2	User data registers shall be read/write.			covered
	….			
INT.1	Register **int_en** shall enable interrupt.			covered

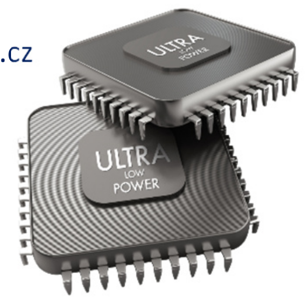


Jak určit, že je verifikace hotová?

Potřebujeme měřit – potřebujeme metriku!

Metriky pro monitorování postupu verifikace

- No. of tests written, passing
  - kolik procent testů je napsáno?
- Requirement coverage
  - kolik procent požadavků z design specifikace je pokryto verifikací (simulacemi a code review)?
- Code coverage (statement, block coverage)
  - kolik procent řádků RTL kódu bylo vykonáno v simulacích?
- FSM coverage
  - které stavy stavových automatů byly navštíveny během simulace?
  - které přechody – ač platné – nebyly nikdy provedeny?
- Expression coverage
  - které části logických výrazů nikdy nezpůsobily, že byl celý výraz platný/neplatný



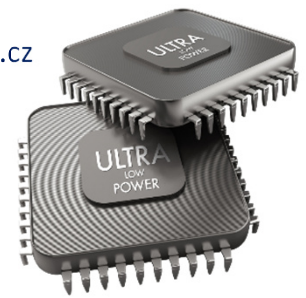
Jak určit, že je verifikace hotová?

- Toggle coverage
  - kolik procent spojů v návrhu bylo překlopeno z 1 do 0 a zpět?
- Code, FSM, Expression, Toggle coverage jsou „implementační“ metriky, nepokrývají funkci, ale strukturu

- Functional coverage
  - kolik procent funkcí návrhu bylo simulací vykonáno?
  - ruční implementace

- Nedostatek času – standardní stav, postup podle priorit požadavků na systém:
  1. most critical/time consuming first
     - ❖ může být depresivní
     - ❖ největší rizika za námi už na začátku
     - ❖ management může být nešťastný
  2. the easiest first, the most critical/time consuming second
     - ❖ návrh si osaháme před tím, než se vrhneme na rizika
     - ❖ přináší menší depresi
     - ❖ je potřeba se nenechat ukolébat dobrým startem

Při akutním nedostatku času – *spray and pray* postup podle pokrytí bloků



Příklad reportování pokrytí v simulátoru

Top Level Summary

Instance name: tb_top.DUT.I_DIG_TOP
Type name: design.dig_top(struct)

Coverage Summary Report, Instance-Based

Block Average	Block Covered	Branch Average	Branch Covered	Statement Average	Statement Covered	Expression Average	Expression Covered	Toggle Average	Toggle Covered	Fsm Average	Fsm Covered	FSM State Average	FSM State Covered
99.10%	98.31% (23220/23620 /1380)	99.05%	98.03% (19899/20299/134)	99.41%	99.87% (16549/16570 /1462)	97.00%	91.47% (39032/42670 /6290)	79.86%	92.02% (59101/64226 /3616)	100.00%	100.00% (147/147/28)	100.00%	100.00% (58/58/7

Block	Branch	Statement	Expression	Toggle	Fsm	Fsm Covered	FSM State	FSM Transition	FSM Arc	name
96.55% (56/58/7)	96.15% (50/52)	100.00% (21/21/7)	94.42% (203/215/18)	97.05% (3484/3590/121)	n/a	n/a	n/a	n/a	n/a	Self

Coverage of immediate sub-instances:

Block Average	Block Covered	Branch Average	Branch Covered	Statement Average ▲	Statement Covered	Expression Average	Expression Covered	Toggle Average	Toggle Covered	Fsm Average	Fsm Covered	FSM State Average	FSM Cov
98.13%	98.13% (263/268)	98.13%	98.13% (262/267)	76.92%	76.92% (10/13)	97.05%	97.05% (329/339)	98.10%	98.10% (517/527/4)	n/a	n/a	n/a	n/a
98.30%	96.28% (233/242/216)	98.09%	95.71% (201/210/7)	98.29%	98.18% (162/165/224)	97.81%	92.13% (480/521/743)	76.56%	78.53% (611/778/141)	n/a	n/a	n/a	n/a
98.07%	96.32% (183/190/15)	97.83%	95.65% (154/161/3)	99.10%	97.65% (166/170/14)	98.13%	96.14% (498/518/57)	89.58%	91.67% (649/708/27)	n/a	n/a	n/a	n/a
99.73%	99.73% (376/377/121)	99.68%	99.70% (330/331/20)	99.65%	99.73% (368/369/135)	99.60%	98.79% (408/413/311)	98.77%	98.82% (753/762/55)	100.00%	100.00% (78/78/9)	100.00%	100.
99.49%	98.78% (729/738/337)	99.44%	98.60% (633/642/8)	99.66%	98.91% (543/549/368)	99.03%	97.73% (1335/1366/923)	95.00%	85.42% (2525/2956/101)	n/a	n/a	n/a	n/a
99.20%	98.97% (863/872/85)	99.12%	98.86% (781/790)	99.85%	99.53% (856/860/87)	94.99%	94.30% (1059/1123/314)	95.35%	93.47% (959/1026/82)	n/a	n/a	n/a	n/a
75.00%	96.09% (8673/9026)	75.00%	95.48% (7458/7811)	100.00%	100.00% (5976/5976)	72.99%	83.38% (13482/16170 /1812)	97.36%	92.28% (16615/18005/877)	100.00%	100.00% (11/11)	100.00%	100.
95.00%	93.33% (14/15)	93.75%	91.67% (11/12)	100.00%	100.00% (11/11)	92.31%	87.88% (29/33)	96.43%	93.62% (44/47)	n/a	n/a	n/a	n/a
99.54%	98.44% (63/64/3)	99.40%	97.62% (41/42/3)	100.00%	100.00% (71/71/3)	92.37%	94.49% (120/127)	99.56%	98.61% (640/649/16)	n/a	n/a	n/a	n/a
99.75%	99.78% (455/456/23)	99.72%	99.75% (400/401/11)	100.00%	100.00% (335/335/19)	98.67%	98.16% (906/923/172)	68.76%	89.53% (1060/1184/62)	100.00%	100.00% (31/31/1)	100.00%	100.
99.99%	99.98% (10075/10077/125)	99.99%	99.98% (8638/8640/58)	100.00%	100.00% (6899/6899/122)	99.28%	96.35% (17356/18013/726)	98.22%	91.24% (23877/26170 /1763)	100.00%	100.00% (13/13/9)	100.00%	100.
100.00%	100.00% (8/8/2)	100.00%	100.00% (6/6)	100.00%	100.00% (4/4/1)	100.00%	100.00% (17/17)	83.05%	83.05% (49/59)	n/a	n/a	n/a	n/a
100.00%	100.00% (7/7/6)	100.00%	100.00% (6/6)	100.00%	100.00% (5/5/6)	96.43%	94.44% (17/18/25)	85.91%	94.17% (113/120/23)	n/a	n/a	n/a	n/a

Coverage Summary Report, Instance-Based

Top Level Summary

Instance name: tb_top.DUT.I_DIG_TOP.i_scan_tst_wrp
Type name:
File name: /[illegible]

Příklad reportování pokrytí v simulátoru

Coverage Summary Report, Instance-Based

Block	Branch	Statement	Expression	Toggle	Fsm	Fsm Covered	FSM State	FSM Transition	FSM Arc	name
98.13% (263/268)	98.13% (262/267)	76.92% (10/13)	97.05% (329/339)	98.10% (517/527/4)	n/a	n/a	n/a	n/a	n/a	i_scan_ts

Uncovered Block Detail Report, Instance Based

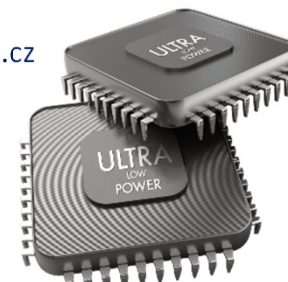
Instance name: tb_top.DUT.I_DIG_TOP.i_scan_tst_wrp
Type name:
File name: /
Number of uncovered blocks: 5 of 268
Number of unreachable blocks: 0
Number of uncovered branches: 5 of 267
Number of unreachable branches: 0
Number of uncovered statements: 3 of 13
Number of unreachable statements: 0

```
Count  Block #Stmt Line  Kind                  Origin Source Code
-----------------------------------------------------------------------------------------
0      182   1     731   a case item of  *     729    WHEN "0010" => test_sdo_d <= core_i2c_hs_detected_i;
0      188   1     737   a case item of  *     729    WHEN "1000" => test_sdo_d <= tst_clk_fifo;
0      190   1     739   a case item of  *     729    WHEN "1010" => test_sdo_d <= tst_otp_clkout;
0      199   0     759   when else       *     756    '1'                   WHEN (tst_otp_rw_ext = '1') OR (tst_ext_fifo_clk = '1' AND tst_ext_sdo_c
0      204   0     770   when else       *     767    '0'                   WHEN (tst_otp_rw_ext = '1') OR (tst_ext_fifo_clk = '1' AND tst_ext_sdo_c

(*) indicates a branch
```

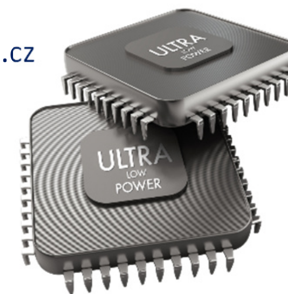
Uncovered Expression Detail Report, Instance Based

Instance name: tb_top.DUT.I_DIG_TOP.i_scan_tst_wrp
Type name:
File name: /
Number of uncovered expressions: 10 of 339
Number of unreachable expressions: 0

```
index  | grade          | line   | expression
-----------------------------------------------------------------------------------------
109.1  | 60.00% (3/5)   | 759    | ((TST_OTP_RW_EXT = '1') or ((TST_EXT_FIFO_CLK = '1') and (TST_EXT_SDO_CLK = '1'))) or ((TST_EXT_SYS_CLK = '1') and (
109.4  | 50.00% (1/2)   | 759    | (TST_EXT_SDO_CLK = '1')
109.5  | 50.00% (1/2)   | 759    | (TST_EXT_SYS_CLK = '1')
109.6  | 50.00% (1/2)   | 759    | (TST_EXT_SDO_CLK = '1')
113.1  | 60.00% (3/5)   | 770    | ((TST_OTP_RW_EXT = '1') or ((TST_EXT_FIFO_CLK = '1') and (TST_EXT_SDO_CLK = '1'))) or ((TST_EXT_SYS_CLK = '1') and (
113.4  | 50.00% (1/2)   | 770    | (TST_EXT_SDO_CLK = '1')
113.5  | 50.00% (1/2)   | 770    | (TST_EXT_SYS_CLK = '1')
113.6  | 50.00% (1/2)   | 770    | (TST_EXT_SDO_CLK = '1')

index: 109.1 grade: 60.00% (3/5) line: 759 source: '1'          WHEN (tst_otp_rw_ext = '1') OR (tst_ext_fifo_clk = '1' AND tst_ext_sdo_clk

((TST_OTP_RW_EXT = '1') or ((TST_EXT_FIFO_CLK = '1') and (TST_EXT_SDO_CLK = '1'))) or ((TST_EXT_SYS_CLK = '1') and (TST_EXT_SDO_CLK = '1'))

 <---------1----------->    <-----------2----------->    <----------3---------->    <----------4---------->

index     | hit   | rval | <1> <2> <3> <4>
--------------------------------------------
109.1.4   | 0     | 0    | 0   0   -   0
109.1.5   | 0     | 0    | 0   -   0   -
```

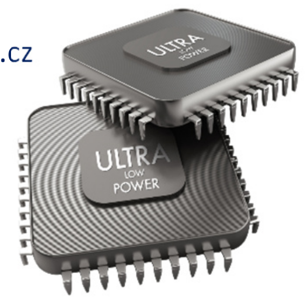


Příklad sledování postupu práce na projektu



Jak si ušetřit při práci s verifikací?

Ukázkový příklad:
verifikuji návrh pulzně-šířkového modulátoru



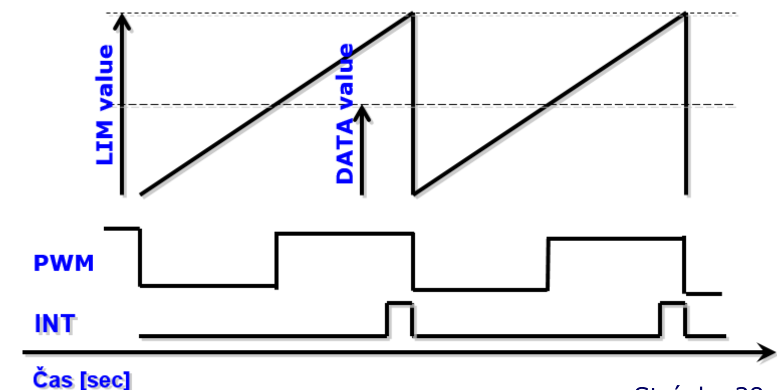
Jak verifikovat – deterministický přístup

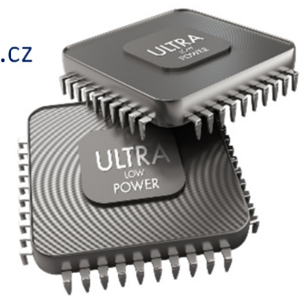
Příklad scénáře pro test:

1. Nahraji do PWM DATA registru hodnotu 0x25, do registru LIM s limitem hodnotu 0xAA
2. Po 0x24 pulzů hodin kontroluji zda je výstup PWM v log. 0, průběžně kontroluji zda je INT výstup v log. 0
3. Po dalším pulzu zkontroluji zda je PWM v log. 1, INT v log. 0
4. Po (0xAA-0x26) pulzech zkontroluji zda se výstup INT překlopí do log. 1 na jednu periodu hodin, do té doby musí být v log. 0
5. atd…

Nevýhody tradičního přístupu

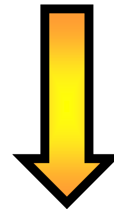
- časově náročné vymýšlení stimulů
- verifikátor má tendenci systém používat "jedním způsobem"
- má "klapky na očích"
- implementace scénářů podle plánu je časově náročná



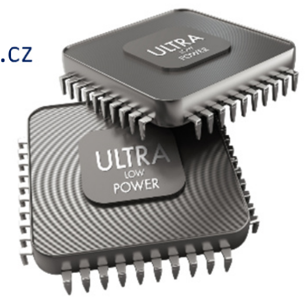


Nejde to jinak?

- **Lépe by bylo nechat si vygenerovat náhodný scénář**
  - **klidně i víckrát, opakování je „zadarmo“**
- **Jenže když generuji náhodný scénář, jak zjistím, co se zverifikovalo?**
  - **co myslíte….?**
- **Pak jen rychle dopsat kousek deterministického testu na obtížně dosažitelný kód**

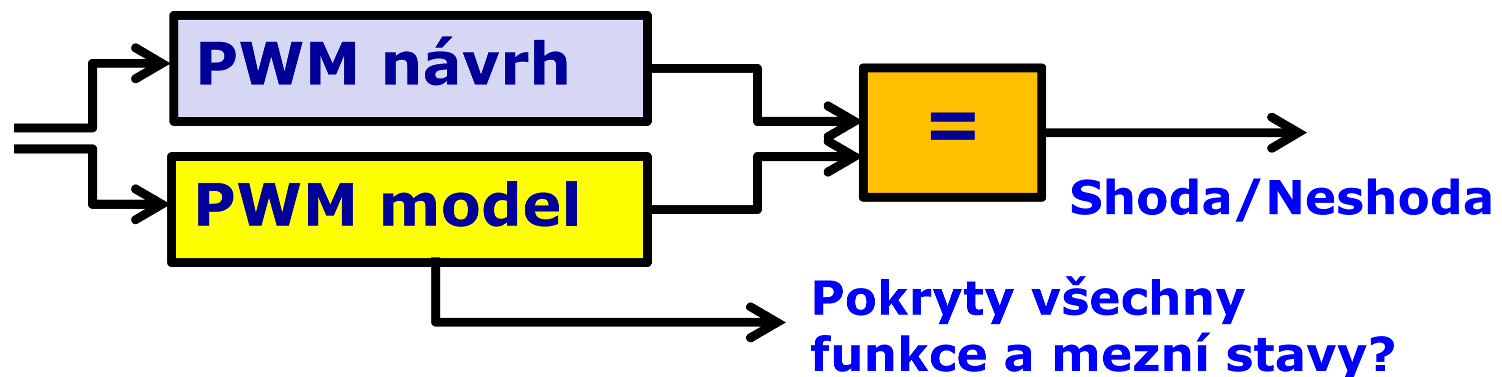


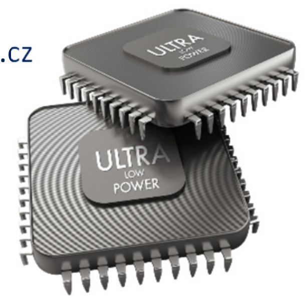
Constrained Random Verification



Constrained Random Verification – jak na to

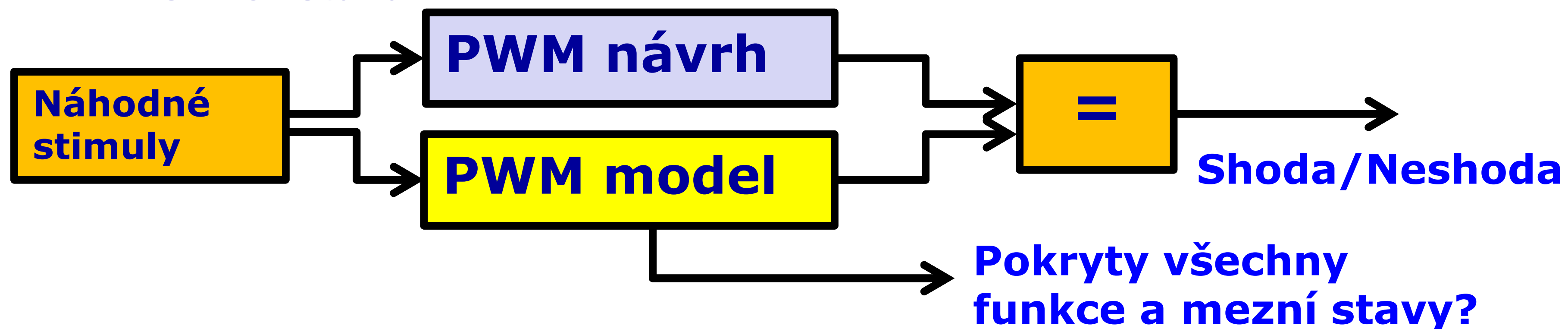
1. **Napíšeme model chování PWM bloku**
   - model bude monitorovat vstupy do PWM bloku
   - podle nich si nastaví vnitřní stav
   - bude generovat výstup PWM a INT s rozlišením jedné periody hodin
   - v modelu budeme sledovat
     - zda se do PWM někdy nahraje hodnota 0x00 a 0xFF
     - zda se do PWM něco zapisuje a něco z něco čte
2. **Napíšeme komparátor výstupů návrhu PWM modulátoru a modelu PWM modulátoru**

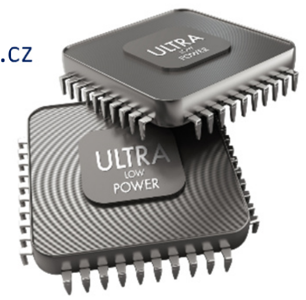




Constrained Random Verification – jak na to

1. **Napíšeme jednoduchý test**
   - **ve smyčce 128 krát**
     1. **počká náhodný počet hodinových cyklů (min. 512, max 1024)**
     2. **s pravděpodobností 0,5 zapíše na adresu mimo adresní rozsah registrové mapy náhodnou hodnotu, jinak zapíše do náhodně vybraného registru náhodnou hodnotu**
2. **Spustíme, zkontrolujeme, zda došlo k mezním stavům, zápisu a čtení všech registrů**
3. **Pokud nejsou pokryty všechny stavy**
   - **prodloužíme test – více smyček, nebo ho spustíme víckrát**
   - **upravíme generování náhodných čísel**
   - **na závěr ručně připíšeme stimuly pro pokrytí obtížně dosažitelných mezních stavů**





Constrained Random Verification

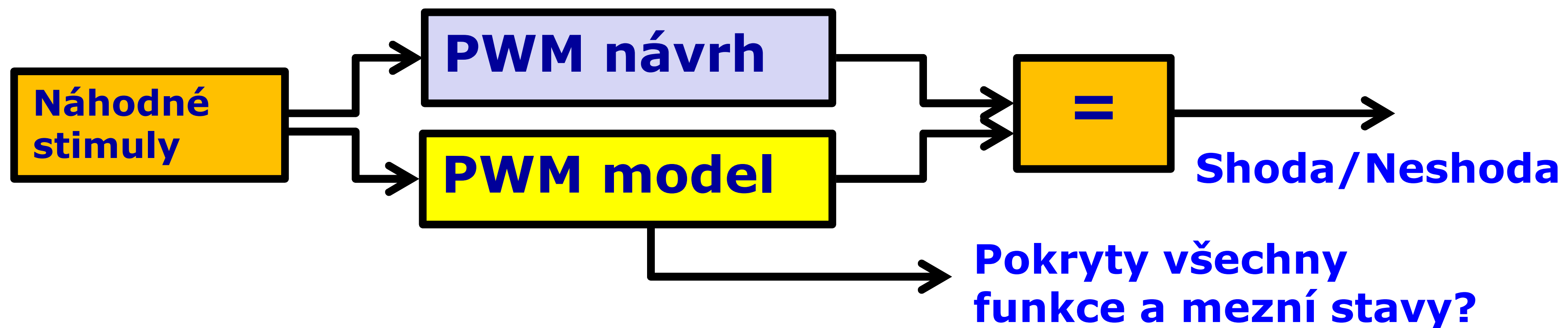
Na tabuli namalovat graf - srovnání pracností

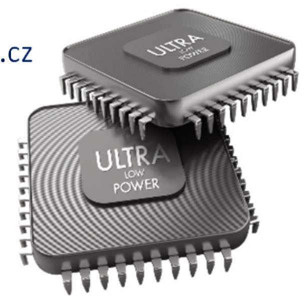
Nevýhody

- složitější simulační prostředí
- větší počáteční investice

Výhody

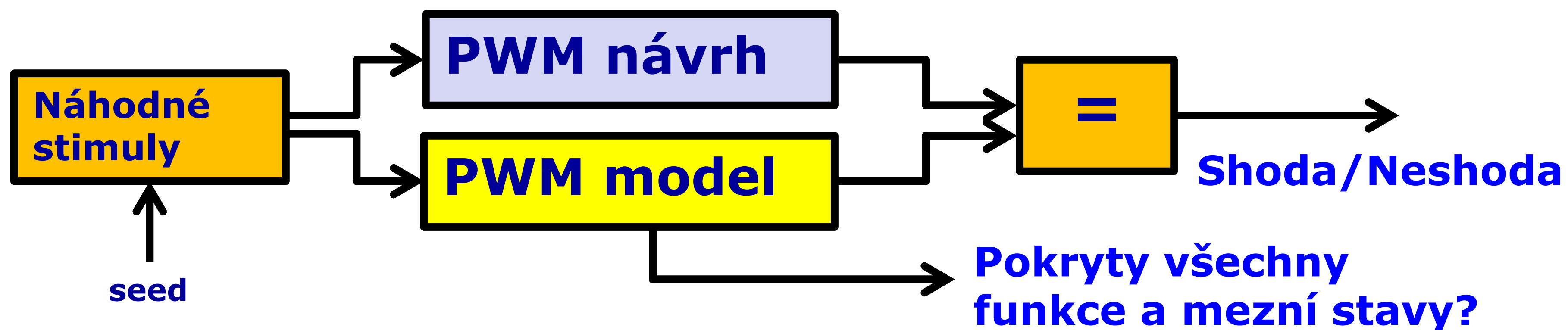
- není třeba vymýšlet stimuly,
  - řeší problém "klapek na očích"
- obětuji strojový čas a ušetřím svůj čas
  - každý běh je trochu jiný
  - pokud budu často pouštět regresi,
    - probádám větší stavový prostor
    - a můžu potkat nečekané okrajové případy (corner cases)
- pro větší návrhy jednoznačně lepší postup

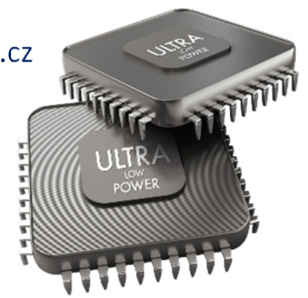




Constrained Random Verification: na co si dát pozor

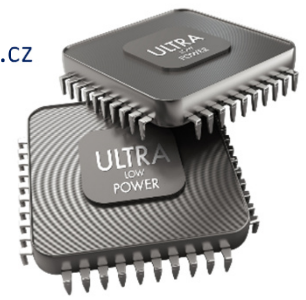
- **Test běží pokaždé jinak….**
  - **nastavení semínka (seed) pro generátor náhodných čísel na začátku**
  - **nevadí, protože pokrytí funkcí je zajištěno**
  - **ve spojení s regresním testováním je to výhoda**
  - **pozor také na modifikace kódu – jedno generování čísla navíc… (SVN revize)**
- **Ve VHDL je generování náhodných čísel s danými vlastnostmi složitější**
  - ***příklad: s pravděpodobností 0.5 zapiš na adresu mimo adresní rozsah registrové mapy náhodnou hodnotu, jinak zapiš na náhodně vybranou adresu náhodnou hodnotu***
  - **ano, to je**
  - **používejte SystemVerilog pro testy a verifikační prostředí, ten pro to má nativní podporu**





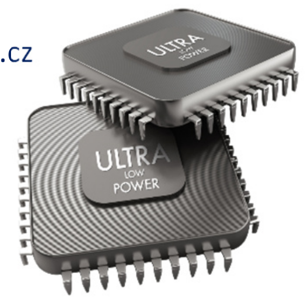
Použité zkratky

- **ABV** – **Assertion-Based Verification**
- **FSM** – **Finite State Machine**
- **INT** – **INTerrupt**
- **IP** – **Intellectual Propery**
- **PSL** – **Property Specification Language**
- **PWL** – **Pulse Width Modulator**
- **RTL** – **Register Transfer Level**
- **SDF** – **Standard Delay File**
- **STA** – **Statical Timing Analysis**
- **UVM** – **Universal Verification Methodology**



Náměty k dalšímu studiu

- Webové stránky
  http://minimizedlogic.sweb.cz
- Současné trendy v návrhu a verifikaci
  - Wilson verification study [online, vid. 23. dubna 2019]
  - https://blogs.sw.siemens.com/verificationhorizons/2022/10/10/prologue-the-2022-wilson-research-group-functional-verification-study/
  - https://blogs.sw.siemens.com/verificationhorizons/
- Detailněji a do větší hloubky o simulaci
  - Šťastný, Jakub. Simulace číslicových obvodů: seriál pro DPS Elektronika od A do Z, dostupný na http://minimizedlogic.sweb.cz
- Zajímavý článek o modelování navrhovaných systémů
  - B. Hailpern and P. Tarr, "Model-driven development: The good, the bad, and the ugly," in *IBM Systems Journal*, vol. 45, no. 3, pp. 451-461, 2006.
  - https://ieeexplore.ieee.org/document/5386628
- Tipy na knihy na další studium
  - Janick Bergeron, Writing Testbenches: Functional Verification of HDL Models. Springer, 2nd edition, 2003
  - Jakub Šťastný. FPGA prakticky.Praha: BEN-technická literatura, 2011.200 s.
- SDF formát - standard
  - IEEE 1497-2001 IEEE Standard for Standard Delay Format (SDF) for the Electronic Design Process [vid. 23. dubna 2019]
    Dostupné z
    http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=972829



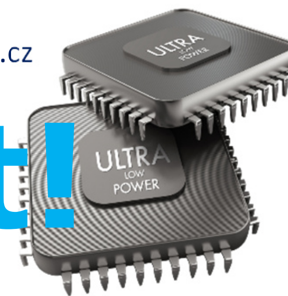
Další doporučená (a použitá) literatura

- DeepChip server, www.deepchip.com, o simulacích na hradlové úrovni
  - Dan Joyce's 16 bug types only found with gate-level simulation [online, vid. 23. dubna 2019]
    - http://www.deepchip.com/items/0569-01.html
  - Dan Joyce's 29 cost-effective gate-level simulation tips [online, vid. 23. dubna 2019]
    - https://deepchip.com/items/0569-02.html
    - https://deepchip.com/items/0569-03.html
    - https://deepchip.com/items/0569-04.html
- Mentor Graphics, ModelSim Reference Manual. Dostupné po instalaci nástroje v adresáři modelsim_installation_path\docs\pdfdocs\xxx_sim_ref.pdf, kde xxx je jméno simulátoru.

ASICentrum A COMPANY OF THE SWATCH GROUP

em microelectronic A COMPANY OF THE SWATCH GROUP

www.asicentrum.cz



Děkuji za pozornost!

Stále hledáme nové kolegy:

www.asicentrum.cz

a kanceláře máme kousek odsud ☺

ASICentrum s.r.o., Purkyňova, Brn

ASICentrum s.r.o.

5.0 ★★★★★ (1)

Electronic parts supplier

Directions Save Nearby Send to your phone Share

Purkyňova 648/125, 612 00 Brno-Medlánky

6HMF+62 Brno

asicentrum.cz

511 156 213

Claim this business

Suggest an edit

Air Technology S.r.o
ASICentrum s.r.o.
Cook Point CEITEC
CAD CAM SYSTEMS sro
Purkyňova
Kolejní
Barber shop
Kadeřnictví Palačák
Fit Centrum Machina
Parking lot
Caffe Bar Piccolo
Grocery store
Potraviny "U Žida"
Vysoké učení technické v Brně...
Yacht Club
Technologický park
Technologický Park Brno as