

# Od algoritmu k číslicovému obvodu

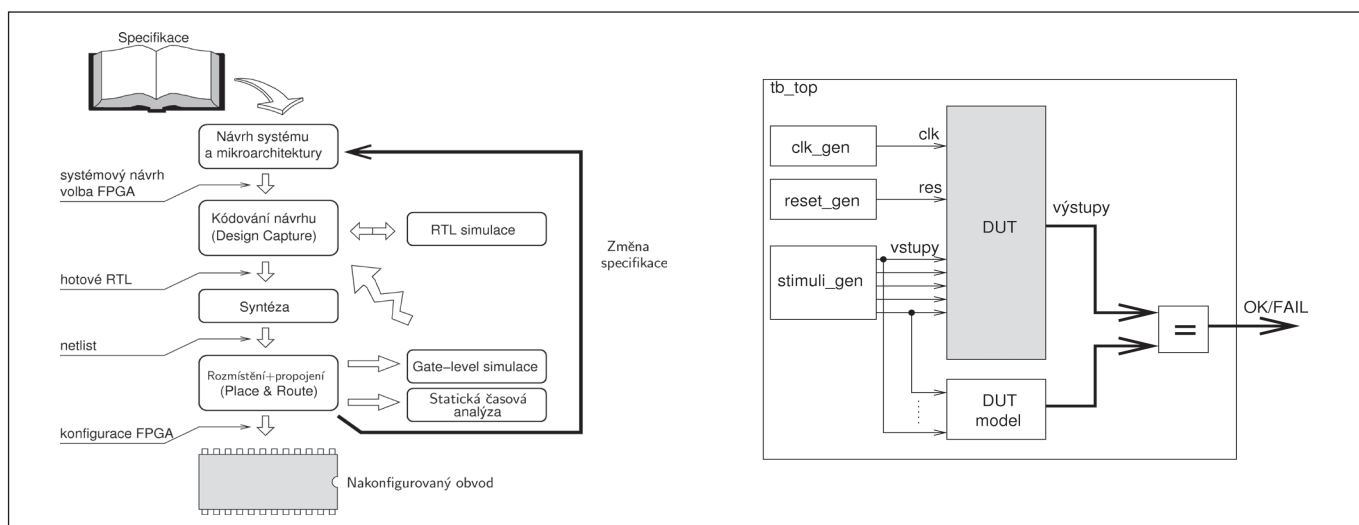
## Úvod

Když Gordon Moore v roce 1965 napsal [1] „The future of integrated electronics is the future of electronics itself. The advantages of integration will bring about a proliferation of electronics, pushing this science into many new areas.“, zdaleka netušil, kam polovodičový průmysl v ná-

specifikace po zahájení prodeje funkčního systému na trhu (tzv. Time To Market, TTM), se odpovídajícím způsobem neprodukuje. Zpoždění s uvedením produktu na trh přitom může znamenat značné finanční ztráty. To vede k potřebě pracovat rychleji a efektivněji a představuje další významnou hrozbu platnosti Mooreova zákona.

**Ing. Jakub Šťastný, Ph.D.**  
**ASICentrum, s. r. o.**  
**FPGA Laboratoř, FEL ČVUT**

úrovni abstrakce. Díky tomu jsou například paměti, které budou v reálném číslicovém systému implementovány pomocí paměťových maker, v modelu realizová-



Obr. 1 Návrhový cyklus číslicového systému na FPGA a základní postup při verifikaci obvodu [2] (DUT=Design Under Test).

sledujícím půlstoletí dospěje. Hypotéza, kterou ve svém článku vyslovil (v původním znění [1] „The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase.“) se mezitím (po malé korekci) stala zákonem a nic na tom nemění ani fakt, že se neustále setkáváme se spekulacemi o tom, kdy přestane platit. Nejčastěji udávanou příčinou pro konec platnosti Mooreova zákona je maximální dosažitelná hustota integrace; tranzistory není možné zmenšovat do nekonečna. Hustota integrace ale není jedinou potenciální překážkou pokračování exponenciálního zesložitování mikroelektronických systémů. Každý čip je dílem lidských rukou a rostoucí integrace a množství funkcí na čipu znamená také rostoucí množství práce (effort, úsilí měřené v člověkoměsíci). Přitom čas na návrh, od vzniku

## Návrhový cyklus číslicového systému

Návrhový cyklus číslicového systému je znázorněn na obr. 1 vlevo. Prvním krokem návrhu musí být vždy specifikace funkce (více viz [2], strana 7). Specifikace je často ve formě obsažného dokumentu popisujícího detailně chování systému; tento dokument je posléze schválen všemi zúčastněnými stranami. V některých případech se lze setkat i s tzv. spustitelnou specifikací – modelem navrhovaného systému, který může být implementován v řadě jazyků (například C, C++, VHDL, Matlab). Zatímco funkce reálného číslicového systému je synchronizována hodinami a systém má definovanou latenci a propustnost, model je implementací samotných algoritmů bez ohledu na hodinové cykly a další reálná omezení. Je tedy implementován na algoritmické, případně transakční

ny jako jednoduchá pole hodnot. Registry reálného obvodu mohou být modelovány pomocí proměnných používaných algoritmem, aritmetické operace jsou typicky implementovány pomocí pohyblivé řádové čárky. Výhodou modelu je spustitelnost – můžeme mu předložit libovolná vstupní data, na kterých jsou vykonány operace požadované od finálního systému a získat příslušné výstupy jak by je měl produkovat reálný systém.

Model je posléze validován zákazníkem/zadavatelem projektu. Na jeho základě je definována mikroarchitektura obvodu. Ve fázi systémového a mikroarchitektonického návrhu fixujeme strukturu budoucího obvodu, mimo jiné provádíme převod polí a proměnných na paměťová makra a registry, definujeme množství a typy užitých funkčních jednotek (například aritmetických operátorů), propustnost a latenci systému, maximální frekvenci systémových hodin

a v neposlední řadě i spotřebu elektrické energie obvody. Model tedy slouží jako podklad pro implementaci RTL (Register Transfer Level, [2], strana 19) kódu obvodu ve fázi kódování návrhu. Mezi RTL kódem a modelem je ovšem zásadní rozdíl: model říká co se má dělat, RTL kód potom jak se to má dělat. RTL kód je implementován typicky v jazyce VHDL nebo Verilog. Tradiční přístup k mikroarchitektonickému návrhu představuje ruční práce, fáze je poměrně časově náročná.

Dalším nezbytným krokem po vlastní implementaci RTL je verifikace navrženého obvodu – ověření správnosti návrhu. Navržený obvod je testován pomocí série simulací, je kontrolováno jeho správné chování proti původnímu modelu (viz [3] a *obr. 1*, pravá část) a postupně jsou odstraňovány jednotlivé chyby. Jedním typem chyb jsou chyby, které vznikají během vlastní konverze modelu na RTL úroveň; takové lze většinou snadno najít a rychle odstranit. Obrnou proti nim je precizně pracující návrhář a pečlivé provádění důkladných simulací navrhovaného obvodu. Jinou kategorií chyb jsou koncepční chyby ve fázi definice mikroarchitektury obvodu, takové ale skýtají riziko kompletního přepracování celého systému a potenciální časovou ztrátu až mnoha člověkoměsíců práce. Hororový scénář potom představuje změna modelu systému v důsledku změny požadavků zákazníka, nejlépe před vlastním koncem projektu. Potom hrozí potřeba kompletního přepracování mikroarchitektonického návrhu i RTL kódu s obrovskými časovými ztrátami.

## Úzká hrdla klasického návrhu

Všechny tyto kroky (systémový a mikroarchitektonický návrh, kódování na RTL úrovni a verifikace) jsou velmi časově náročné a chceme-li dosáhnout zrychlení návrhového cyklu, musíme zrychlit jejich provádění. Zrychlení můžeme dosáhnout více způsoby: například fázi kódování RTL návrhu lze zkrátit užitím již dříve navržených a odladěných bloků – jader (IP cores, Intellectual Property, viz např. [4]). Tím se lze dobrat i ke zkrácení následné fáze verifikace, protože už

jednou odladěná jádra většinou nebudou vyžadovat takové úsilí strávené hledáním a laděním chyb. Projektoví manažeři dobrodružnějších povah mohou také vynechat část verifikačních simulací testujících některé důvěryhodnější funkce těchto jader (i když vlastní zkušenosti autora bohužel spíše ukazují, že co se nezverifikovalo, nebude fungovat a že ani profesionálně udělaná jádra nejsou bez chyb). Dalšího urychlení verifikace lze dosáhnout použitím formální verifikace, inteligentnějších verifikačních technik (například randomizované verifikace proti modelu) a programovacích jazyků s podporou pro tyto techniky (například SystemVerilog).

Nicméně i po aplikaci všech uvedených opatření stále zůstává v návrhovém cyklu (viz *obr. 1*) časově náročná fáze ručního mikroarchitektonického návrhu.

## Syntéza z algoritmu

V tomto okamžiku je zajímavé se zamyslet nad paralelami mezi návrhem hardware a programováním software. Vlastní proces konverze software do spustitelného binárního souboru je dnes extrémně jednoduchý. Prostým stiskem tlačítka v integrovaném návrhovém prostředí (IDE, Integrated Development Environment) získáme po krátkém čekání spustitelný kód – a je hotovo. Proč není něco podobného možné při konverzi modelu systému např. v C++ do RTL kódu?

Skutečně, tato myšlenka už mnoho let není horkou novinkou a existují komerční i akademické nástroje, které tento proces dokáží automatizovat. Jedná se nicméně o algoritmicky velmi složitou úlohu a tak je třeba počítat s jistými omezeními. Proces konverze se nazývá „syntézou z algoritmu“, případně „syntézou na vyšší úrovni“; my se budeme dále držet zkratky HLS (High Level Synthesis).

Předně, mezi návrhem hardware a programováním software je mnoho rozdílů. Programujeme-li software, výstupem je spustitelný soubor, kód optimalizujeme typicky na rychlost běhu, jen výjimečně velikost (embedded aplikace) a většinou ani není třeba se trápit s omezeními, která by například bránila používat mnoho proměnných, či velká pomocná pole. Častou praktikou je například

ukládání velkého množství předpočítaných hodnot pro urychlení dalšího běhu programu. Oproti tomu při návrhu hardware je výstupem polovodičový čip nebo konfigurovaný FPGA obvod, a musíme tedy mít na mysli omezení daná fyzikálními zákony i při vlastním psaní RTL kódu. Vlastní postup i jen samotné „konverze“ RTL kódu do podoby polovodičové struktury je netriviální a rozhodně ho nelze provést stiskem tlačítka „compile“ jako u programování. Obvykle si také nemůžeme dovolit operovat s velkými pomocnými paměťmi a optimalizaci výsledného produktu provádíme na datovou propustnost, latenci zpracování, spotřebu elektrické energie, maximální dosažitelnou hodinovou frekvenci a občas i na další (z pohledu programátora poněkud obskurní) parametry – například propojitelnost (routability) nebo využití plochy čipu. Z těchto důvodů také na úrovni kódování často rozlišujeme mezi návrhem a programováním.

Z uvedeného srovnání vyplývá, že proces přímé konverze algoritmu do RTL kódu bude automatizovatelný jen s řadou omezení. To nejmarkantnější je, že nelze konvertovat kód zapsaný libovolným způsobem; i v tomto případě je třeba se seznámit se syntetizovatelnou podmnožinou použitého programovacího jazyka a stále přemýšlet nad vlastní implementací.

Abychom čtenáři proces konverze algoritmu na RTL úroveň více přiblížili, připravili jsme jednoduchý demonstrační příklad – implementaci bloku pro výpočet druhé odmocniny technikou půlení intervalu na FPGA obvod. Použití HLS samozřejmě není omezeno jen na FPGA. FPGA platforma byla zvolena z didaktických důvodů.

Ve *výpisu 1* je zdrojový kód modelu bloku pro výpočet druhé odmocniny; vstupem je 16 bitů široké celé číslo a bez znaménka, výstupem  $x$ , pro které platí, že  $x^2 \leq a$ . Ve *výpisu 2* je pak možné nalézt model přepsaný do podoby odpovídající zadání – vstupem bloku je 16 bitů široké číslo a bez znaménka (datový typ `ac_int<16,false>`), tělo algoritmu je zkráceno jen na 8 iterací – v každé iteraci se interval obsahující výsledek zkrátí na polovinu a výsledek se tak zpřesní o jeden bit. Výsledek je ovšem osmibitový, bez znaménka (datový typ `ac_int<8,false>`),

protože je odmocninou ze 16 bitového čísla, proto jen 8 opakování *for* cyklu. Konečně výpis 3 prezentuje jednoduché testovací prostředí, které ukazuje, že uprave-

nový kód byl načten do HLS nástroje a byly nastaveny základní parametry řídicí proces syntézy: z didaktických důvodů jsme zvolili implementaci do ob-

RTL kódu (Generate RTL). Výstupem nástroje byl přehledný Ganttův diagram plánování operací, viz obr. 2. V diagramu vidíme, v jaké sekvenci budou prováděny jednotlivé operace ze zdrojového kódu (ve výpisu 2 jsou v komentářích uvedeny názvy operací), v jakém cyklu hodin (každý rámeček „Cx“ je jeden hodinový cyklus obvodu) a dále jak na sobě jednotlivé operace závisí (např. u násobení vidíme, že je závislé na výsledku průměrování a jeho výsledek je dále potřebný pro operaci komparace realizovanou pomocí sčítačky).

Vygenerovaný RTL kód posléze můžeme načíst do simulátoru a ověřit jeho správnou funkci verifikací, viz obr. 3. Funguje-li dle očekávání, použijeme nástroje pro implementaci na FPGA obvodech (např. ISE fy Xilinx) a vygenerujeme přímo konfiguraci programovatelného obvodu. Celý proces konverze C++ do RTL kódu trval jen minuty a v tomto jednoduchém případě jsou výsledky srovnatelné s ruční prací, srovnej [2], kapitola 7, kde je stejný blok implementován ručně. Zájemce o hlubší pochopení práce HLS nástroje odkazujeme na stejný zdroj, kde je popsán postup ruční syntézy datové cesty; HLS nástroj pracuje velmi podobným způsobem. Více detailů o HLS pak lze nalézt v [6].

Opravdový půvab HLS vynikne, pokud budeme potřebovat změnit specifikaci systému: například zákazník přijde se změnou požadavku – z nových skutečností vyplývá, že  $f_{clkmax} = 100 \text{ MHz}$ , ne 20 MHz. Na použitém FPGA obvodu toto není triviální požadavek, obvod totiž neobsahuje hardwarové násobičky a zvýšení pracovní frekvence znamená nutnost použít pipelinovanou násobičku, místo čistě kombinačního obvodu. To s sebou nese nezanedbatelné množství práce, protože kromě změny násobičky je ješ-

### Výpis 1: Model bloku pro výpočet druhé odmocniny

```
unsigned model_square_root (unsigned a)
{
    unsigned sq_hi = 255;
    unsigned sq_lo = 0;
    unsigned sq_tmp = 0;
    for (int cycle=0; cycle<16; cycle++)
    {
        sq_tmp = (sq_hi+sq_lo)/2;
        if ( (sq_tmp*sq_tmp) > a) {
            sq_hi = sq_tmp;
        } else {
            sq_lo = sq_tmp;
        }
    }
    return sq_lo;
}
```

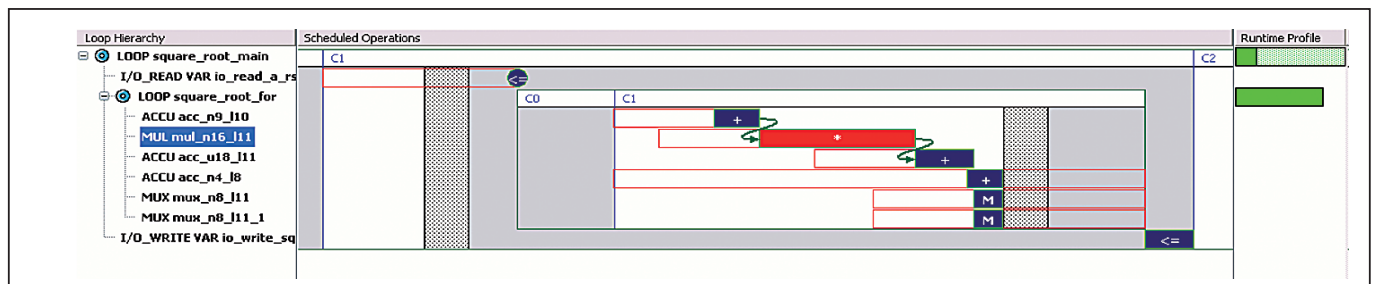
### Výpis 2: Zdrojový kód algoritmu v „syntetizovatelné podmnožině“ C++

```
#include <ac_int.h>
ac_int<8,false> square_root (ac_int<16,false> a)
{
    ac_int<8,false> sq_hi = 255;
    ac_int<8,false> sq_lo = 0;
    ac_int<8,false> sq_tmp = 0;
    for (int cycle=0; cycle<8; cycle++) //++ acc_n4_18
    {
        sq_tmp = (sq_hi+sq_lo)/2; //+ acc_n9_110
        if ( (sq_tmp*sq_tmp) > a) { // * mul_n16_111
            sq_hi = sq_tmp; //> acc_u18_111
        } else {
            sq_lo = sq_tmp;
        }
    }
    return sq_lo;
}
```

ný algoritmus funguje stejně jako původní model.

Pro syntézu ukázkového příkladu byl použit HLS nástroj Catapult C [5]. Zdro-

vodu Spartan 2E s maximální pracovní frekvencí  $f_{clkmax} = 20 \text{ MHz}$ . Dalším krokem už bylo spuštění operace plánování zdrojů (Schedule) a následného generování



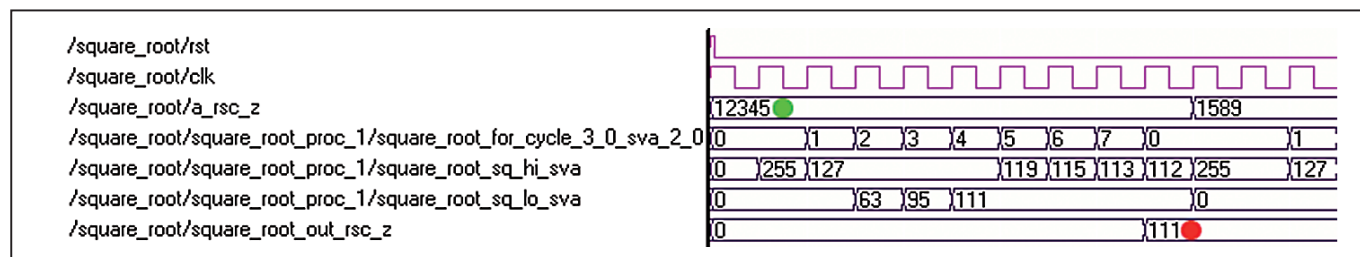
Obr. 2 Plánování operací.

tě třeba změnit i řídicí automat obvodu. Používáme-li HLS nástroje, stačí jen přenastavit pracovní frekvenci na 100 MHz a opětovně spustit celý proces syntézy;

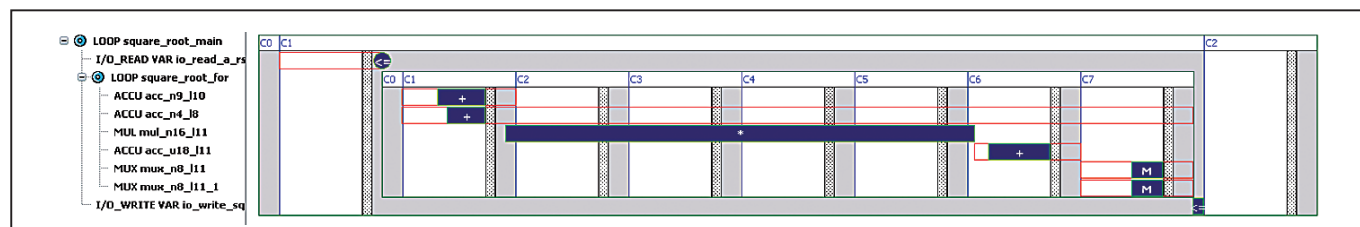
pověď na takové otázky je při ruční implementaci RTL kódu velmi obtížné.

Nespornou výhodou je dále skutečnost, že odladěný model – abstraktní

zkušený návrhář implementující pečlivě vyladěný systém. Proto použití HLS není vhodné např. pro implementaci obvodů typu RFID, kde je nutné počítat



Obr. 3 Demonstrace funkce obvodu v simulátoru ModelSim. Zelenou tečkou je vyznačena počáteční hodnota a červenou tečkou finální výstup obvodu (1112=12321). Všimněte si signálů, které jsou připojeny na výstup registrů implementujících jednotlivé proměnné algoritmu – *sq\_hi*, *sq\_lo* a *cycle*.



Obr. 4 Plánování operací pro pracovní frekvenci 100 MHz

po zhruba minutě práce nástroje dostaneme nové RTL kódy a Ganttův diagram na obr. 4. Vidíme, že násobení bylo nyní automaticky rozděleno do čtyř hodinových cyklů. Problém je tedy vyřešen.

## Výhody a nevýhody HLS

HLS přístup přináší nesporně zkrácení návrhového cyklu; konverze modelu na RTL kód je rychlá a práce, která by jinak trvala i mnoho týdnů, je provedena doslova za desítky minut. Rychlost a pohodlí generování RTL kódu s sebou přináší i doposud netušené možnosti rychlého experimentování s mikroarchitekturou a provádění tzv. what-if (co-kdyby?) analýzy. Jsme tak schopni rychle odpovědět na otázky typu „Co se stane, když budu chtít vyšší  $f_{clkmax}$ ?“ nebo „K čemu dojde, když budu chtít, aby celý výpočet byl implementovaný pomocí proudového zpracování s latencí 5 cyklů?“ atd. Díky tomu lze během jediného dne prozkoumat řadu alternativ a zvolit nejvhodnější mikroarchitekturu systému. Můžeme tak řešit potenciální problémy s časováním obvodu, plochou a příkonem mnohem dříve, než při použití standardního přístupu. Získat okamžitou od-

algoritmus – zůstává hlavním zdrojovým kódem pro implementaci na RTL úrovni. Jeho změna v důsledku nových požadavků zákazníka tak místo na přepisování RTL kódu vede jen na elegantní přegenerování RTL pomocí HLS nástroje. Celý proces je navíc automatický a tedy bez náhodných chyb vnesených návrháři.

V neposlední řadě se rapidně zjednodušuje i znovupoužití jader. Blok implementovaný v C++ můžeme vysyntetizovat s různými parametry mikroarchitektury (např. minimalizovanou plochou, spotřebou nebo latencí) podle požadavků aplikace. To je něco, co je ve světě znovupoužívání RTL jader nemožné bez přepracování tzv. „znovupoužitého“ bloku, čímž se ztrácí všechny výhody opětovného použití. HLS zjednodušuje i přenos bloků mezi technologickými knihovnami pro návrh ASIC obvodů, případně FPGA platformami. RTL kód ve skutečnosti nikdy není zcela technologicky nezávislý, na úrovni C++ je ovšem technologická nezávislost vynucena.

Všechny uvedené výhody jsou ovšem vyváženy mnohými negativy. V první řadě HLS nástroje nejsou schopny dosáhnout kvality výsledku, jakou dosáhne

s každým klopícím hradlem z důvodů maximálního omezení spotřeby elektrické energie obvodem. Použití HLS syntézy s sebou také přináší závislost na nástroji jednoho zvoleného výrobce, zatímco RTL syntéza je průmyslový standard a při přechodu mezi nástroji obvykle nenastávají dramatické situace. HLS nástroje ani nepatří k nejlevnějším. A v neposlední řadě je třeba se naučit nový návrhový jazyk: HLS syntéza vyžaduje, aby byl syntetizovatelný C++ kód zapsán specifickým způsobem, při jeho psaní je stále nutno přemýšlet nad tím, jak bude vypadat výsledná polovodičová struktura. To spolu s faktem, že (na rozdíl od nástrojů pro programování) zde stále neexistuje řešení pracující na jeden „triviální“ stisk tlačítka s sebou nese jednoduché poselství: bylo by vážnou chybou se domnívat, že používání HLS zcela nahradí zkušenosti návrhářů.

Za dnešního stavu věci je tak HLS nejvhodnější pro realizaci větších jader implementujících složité výpočetní algoritmy (např. komprese a dekomprese videa, digitální rádiové aplikace atd.) bez kritických nároků na spotřebu elektrické energie obvodem a plochu zabranou čipem. Tato jádra pak mohou být složena dohro-

mady se zbytkem systému implementovaného pomocí standardního postupu. Právě proto, aby byla možná snadná integrace produktu navrženého pomocí HLS s existujícími bloky, je výstupem HLS nástroje RTL kód. Použití HLS nástroje

tom byli „zasaženi“ příchodem assertions do běžné praxe, randomizovanou verifikací a dalšími technikami automatizace verifikace. V tomto kontextu je HLS jen logickým důsledkem vývoje polovodičového průmyslu. A jako u předcho-

*Electronics*. vol. 38, no. 9, April 19 1965

[2] *Jakub Štátný. FPGA prakticky, BEN Praha 2011. <http://shop.ben.cz/cz/121316-fpga-prakticky.aspx> (kontrolováno 17. 11. 2011)*

### Výpis 3: Verifikační prostředí pro algoritmus výpočtu druhé odmocniny

```
void tb_run (void)
{
    int fi_match = 0;
    int fi_mismatch = 0;
    for (int i = 0; i < 64; i++)
    {
        int test_num = rand() & 0xFFFF;
        int dut_out = square_root(test_num);
        int cpp_out = model_square_root(test_num);
        cout << "Input:" << test_num << " DUT output:" << dut_out <<
            " SQR output:" << cpp_out << "\n";
        fi_match += (dut_out == cpp_out);
        fi_mismatch += (dut_out != cpp_out);
    }
    cout << "Matches: " << fi_match << " Mismatches: " << fi_mismatch << "\n";
}
```

tedy typicky neodstraňuje potřebu použití standardní RTL syntézy a stále vyžaduje expertizu s tím spojenou.

## Závěr

Polovodičový průmysl, zejména metodologie návrhu číslicových systémů, prochází neustálou evolucí směřující k vyšší produktivitě práce návrhářů, a když ne ke snížení, tak aspoň k zachování TTM. Opravdoví pamětníci si mohou vzpomenout na „pravěké“ techniky plně ručního návrhu, zavedení prvních CAD nástrojů a posléze revoluční příchod RTL syntézy. Mladší návrháři po-

zích evolučních kroků, i zde je tichý příslib řádového zvýšení produktivity práce za předpokladu správného užití nástrojů; zda se HLS rozšíří tak jako RTL syntéza, ukáže čas.

## Poděkování

Tato práce byla podpořena grantem GAČR 102-11-1795: Novel selective transforms for non stationary signal processing. [stastnj1@fel.cvut.cz](mailto:stastnj1@fel.cvut.cz)

## Literatura:

[1] *Gordon Moore. Cramming more components onto integrated circuits.*

[3] *Lukáš Ručkový, Tomáš Kubec, Jakub Štátný. Z08-2: Návrh verifikačního prostředí pro FPGA bloky. FEL ČVUT v Praze, Katedra teorie obvodů. <http://amber.feld.cvut.cz/fpga/publications> (kontrolováno 17. 11. 2011)*

[4] *Design & Reuse. <http://www.design-reuse.com/> (kontrolováno 17. 11. 2011)*

[5] *Mentor Graphics. Catapult C. <http://www.mentor.com/esl/catapult/overview> (kontrolováno 17. 11. 2011)*

[6] *Michael Fingeroff. HLS Blue Book. <http://www.hlsbluebook.com> (kontrolováno 17. 11. 2011)*

## Nový standard IPC-2152

Nový standard IPC-2152 pro určení přípustné velikosti proudu v plošných spojích při návrhu DPS (*Standard for Determining Current-Carrying Capacity in Printed Board Design*) vychází ze základní konfigurace, která představuje konzervativní metodu pro určení potřebné velikosti plošných spojů pro přenos určité velikosti proudu. Nové grafy, které tento standard zahrnuje, jsou založené na testech prováděných na plošných spojích na desce bez měděných ploch, a to jak za normálních podmínek, tak i ve vakuu. V základní konfiguraci jsou uvedeny hodnoty tloušťky desky, materiál, šířka a tloušťka plošného spoje a dále pak různé variace těchto proměnných.

Standard IPC-2152 může být použit k vytvoření pravidel pro individuální návrh plošných spojů s ohledem na přenos proudu. Doporučení uvedená v tomto standardu lze přizpůsobit jakékoli technologii desky plošných spojů.

[www.ipc.org](http://www.ipc.org)