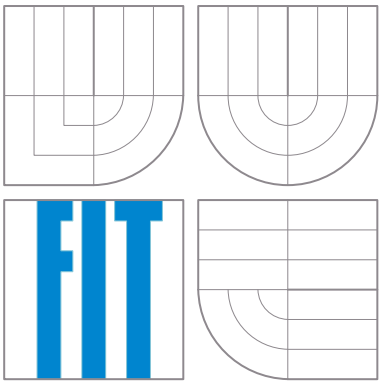
Vysoké učení technické v Brně
Fakulta informačních technologií



Pokročilé číslicové systémy

Verifikace číslicových systémů využitím metod a nástrojů formální verifikace

cvičení

2013

Tento materiál vznikl za podpory Fondu rozvoje vysokých škol (projekt FR1086/2013/G1).

Obsah

1 Úvod

Složitost současných číslicových obvodů neustále roste. Důsledkom toho je, že ověřování jejich korektnosti formou verifikace a testování trvá neúměrně dlouho a stává se skutečnou výzvou pro návrháře.

Pro verifikaci a testování číslicových obvodů existuje v současnosti velké množství přístupů. I když je většina z nich založena na RTL simulaci, jako např. funkční verifikace, stále více se uplatňují i techniky a nástroje formální verifikace. Důvod je ten, že funkční verifikace nedokazuje koreknost systému. Není totiž možné dokázat, že funkční verifikace generuje všechny vstupy pro verifikovanou jednotku, které by pokryly 100 %-ní chování obvodu a odhalili přítomnost všech chyb.

Jako výhodná strategie pro ověřování korektnosti obvodů se proto jeví kombinace obou přístupů - funkční i formální verifikace.

Přístupy založené na formální verifikaci přinášejí rozšířené možnosti pro analýzu chování obvodu založenou na definování tzv. formálních tvrzení (angl. *assertions*), které toto chování popisují. Formální tvrzení jsou výroky v temporální logice, které umožňují zachytit a kontrolovat chování systému v čase. Použité nástroje pak buď provádějí kontrolu formálních tvrzení pomocí úplného proskoumávání stavového prostoru (model checking), nebo v spojení s funkční verifikací kontrolují dodržování platnosti zavedených formálních tvrzení v průběhu simulace *Assertion-Based Verification — ABV*. Funkční verifikací se ale v tomto cvičení zabývat nebudeme.

Formální tvrzení jsou známá i z prostředí vývoje softwarových produktů. Navzdory tomu, že způsob vývoje číslicových obvodů se stále více podobá vývoji softwarových produktů, jedna vlastnost zůstává pro oblast číslicových obvodů neměnná — čas. Jazyky pro specifikaci chování číslicových obvodů (angl. *Hardware Description Languages — HDL*) umožňují zachycení časových vlastností a popis přesného chování v čase. To standardní procedurální jazyky (např. C, C++, Java, ale zcela ani Verilog) neumožňují.

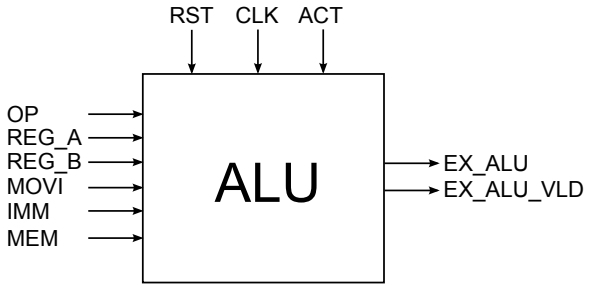
Mezi první jazyky umožňující zachycení temporálních vlastností patří **Sugar** a **OVA** (Open VERA Assertions). Tyto jazyky byly časem rozšiřovány a byly přijaty jako standard společností Accelera. Z jazyku Sugar se stal později jazyk PSL (*Properties Specification Language*), který spolu s jazykem OVA posloužil jako základ pro standardizaci jazyka SVA (*SystemVerilog Assertions*). Jazyk SVA je součástí samotného standardu SystemVerilog (IEEE 1800-2009), obsahuje konstrukty z jazyka PSL a OVA. Jazyk PSL je samostatným jazykem, který je možno použít v jiných HDL jazycích a je také samostatným standardem (IEEE 1850-2005).

V tomto cvičení se zaměříme na aplikaci formálních tvrzení (assertions), budeme se zabývat jazykem **SystemVerilog Assertions** (SVA). Budou vysvětleny základy automatické formální analýzy a použití technik formální verifikace pro potřeby testovaní a odhalování chyb v číslicových systémech pomocí formálních tvrzení. Všechny cviční příklady budou aplikovány na **aritmeticko-logickou jednotku** (ALU), ve které se uplatní jak kontroly uvnitř dané jednotky, tak kontroly rozhraní. Budou demonstrovány výhody aplikace postupů formální verifikace využitím specializovaného jazyka, možnosti vytvoření základního prostředí pro verifikaci a využití sady nástrojů Questa od společnosti Mentor Graphics.

2 Popis verifikované jednotky ALU

Na obrázku 1 je znázorněno rozhraní verifikované jednotky ALU. Toto rozhraní je tvořeno následujícími signály:

- řídicí:
  - `CLK [in]`: hodinový signal; všechny vstupy a výstupy ALU jsou aktivní na náběžné hraně tohoto signálu,
  - `RST [in]`: synchronní reset,
  - `ACT [in]`: aktivační vstup, hodnota `'1'` signalizuje požadavek na výpočet,
- datové:
  - `OP(3:0) [in]`: volba operace ALU (viz Tabulka 1),
  - `REG_A(DATA_WIDTH-1:0) [in]`: vstup z registru jenž obsahuje první operand,
  - `MOVI(1:0) [in]`: vybírá druhý operand (viz Tabulka 2),
  - `REG_B(DATA_WIDTH-1:0) [in]`: vstup z registru jenž obsahuje druhý operand (`MOVI = "00"`),
  - `MEM(DATA_WIDTH-1:0) [in]`: vstup z paměti jenž obsahuje druhý operand (`MOVI = "01"`),
  - `IMM(DATA_WIDTH-1:0) [in]`: přímo zadaný druhý operand (`MOVI = "10"`),
- výstupy:
  - `EX_ALU_VLD [out]`: signál značící platnost výsledku (může být již tentýž takt jako je `ACT = '1'`),
  - `EX_ALU(15:0) [out]`: výsledek operace (hodnota je platná jen když `EX_ALU_VLD = '1'`); pro operace mimo násobení je výsledek předán v jednom taktu, u násobení je výsledek předán ve dvou taktech (v obou je signál `EX_ALU_VLD` nastaven do `'1'`, první je předána méně významná část výsledku).



Obrázek 1: Rozhraní ALU jednotky.

Tabulka 1: Význam hodnot vstupního signálu `OP` komponenty ALU.

Hodnota	Význam
0x0	OUT = IN1 + IN2
0x1	OUT = IN1 - IN2
0x2	OUT = IN1 * IN2
0x3	OUT = IN2 >> 1
0x4	OUT = IN2 << 1
0x5	OUT = IN2 ROR 1
0x6	OUT = IN2 ROL 1
0x7	OUT = ~IN2
0x8	OUT = IN1 & IN2
0x9	OUT = IN1 | IN2
0xA	OUT = IN1 ^ IN2
0xB	OUT = ~(IN1 & IN2)
0xC	OUT = ~(IN1 | IN2)
0xD	OUT = ~(IN1 ^ IN2
0xE	OUT = IN2 + 1
0xF	OUT = IN2 - 1

Tabulka 2: Význam hodnot vstupního signálu `MOVI` komponenty ALU.

Hodnota	Význam
0x0	REG_B
0x1	MEM
0x2	IMM
0x3	<RESERVED>

3 Nástroje Questa společnosti Mentor Graphics

3.1 Automatická formální analýza pomocí nástroje Questa Autocheck

Nástroj Questa Autocheck umožňuje realizovat automatickou formální analýzu zdrojových souborů verifikované jednotky. Tato kontrola probíhá bez zásahů ze strany návrháře, nevyžaduje žádné definice testů tak, jak je známe z oblasti funkční verifikace. Využívá množinu předem definovaných formálních tvrzení v podobe knižnic anebo uživatelsky definovaných formálních tvrzení. Cílem této analýzy je detekce bežně se vyskytujících problemů a chyb, které vznikají během návrhu obvodů. Typické příklady jsou aritmetické chyby, přetečení, propojení komponent, vznik nežádoucích registrů, chyby v konečných automatech, chybějící signály v sensitivity listu procesů, atd.

3.2 Formální verifikace pomocí nástroje Questa Formal

V tomto případě jsou formální tvrzení definovány většinou uživatelem a vkládány přímo do zdrojového kódu návrhu číslicového obvodu anebo zapsány v externím souboru a připojeny k verifikaci. Nástroj Questa Formal je postaven na principu Bounded Model Checking, kdy ověřuje

platnost jednotlivých tvrzení proskoumáváním stavového prostoru verifikované jednotky. V případě, že nástroj objeví porušení sledované podmínky, poskytne protipříklad v podobě konkrétních vstupů a stavů, které vedou k chybě. Jinak poskytne důkaz o korektnosti obvodu.

4 Úlohy k řešení

V následujícíh úlohách prozkoumáme základní možnosti využití nástrojů Questa Autocheck a Questa Formal pro odhalení chyb v implementaci jednotky ALU.

Úloha 1 — příprava prostředí pro automatickou formální analýzu v nástroji Questa Autocheck.

Pro automatickou statickou analýzu je potřebné vytvořit pracovní knihovnu pro práci se zdrojovými soubory, jejich překlad a načtení vytvořených komponent.

Následující příkazy vytvoří pracovní knihovnu "work" pro práci s ALU (implicitní, tak jak jej známe z práce s VHDL soubory, je knihovna `work`). Následně je potřeba nadefinovat mapování pracovní knihovny s adresářem `.work`.

```
# vlib work
# vmap work work
```

V dalším kroku vytvoříme seznam souborů "vhdl_file_list" pro překladač.

```
# find ./src/ -type f -name *.vhd -print > vhdl_file_list
```

V tomto okamžiku můžeme realizovat překlad zdrojových souborů pomocí příkazu "vcom":

```
# vcom -f vhdl_file_list
```

Úloha 2 — spuštění statické analýzy zdrojového kódu.

V této chvíli máme připraveny přeložené zdrojové soubory komponent ALU pro statickou analýzu nástrojem Autocheck. Povšimněte si připravených skriptů pro tento nástroj ve složce `autocheck_files`:

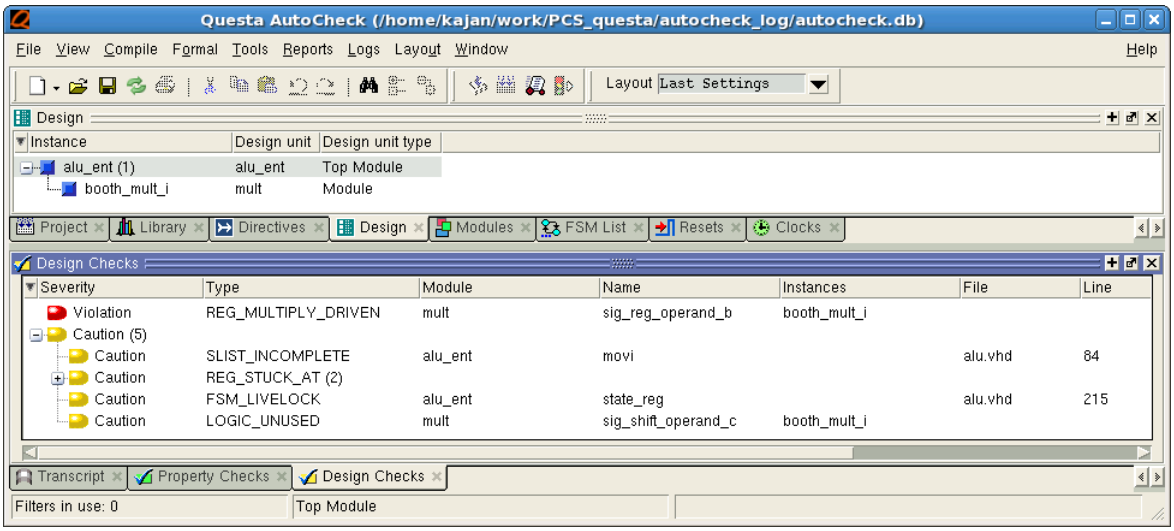
```
|-- autocheck_files
    |-- autocheck_run.do           // spouštěcí skript
    |-- autocheck_directives.tcl   // direktivy pro analýzu
    `-- autocheck_waivers.tcl      // ignorované direktivy pro analýzu
```

Výsledkem statické analýzy bude nová složka `autocheck_log`, která bude obsahovat soubory s výsledky analýzy.

Samotnou statickou analýzu spustíme příkazem:

```
# qautocheck -do ./autocheck_files/autocheck_run.do
```

V zobrazeném GUI v záložce Design Checks se objeví seznam **porušení vestavěných formálních tvrzení** (*Violations*) a seznam **várování** (*Cautions*) pro přeložený obvod (obrázek 2).



Obrázek 2: Prostředí Questa AutoCheck.

Následující podúlohy se zaměřují na opravu nalezených chyb. Pro úpravu zdrojových souborů použijte raději externí editor. Všimejte si také jednotlivých řádků v nástroji Autocheck, kde nástroj zobrazuje množství informací umožňujících lokalizaci a opravu chyb.

1. Porušení `REG_MULTIPLY_DRIVEN` signalizuje pro daný Registr/Latch, že zápis do něj je prováděn z více vstupních signálů. Odhalte místo chyby a opravte ji (pravým tlačítkem myši `Show` $\longrightarrow$ `Source`). Po opravě pomocí `vcom` opětovně přeložte zdrojové soubory a spusťte znovu statickou analýzu.

2. Odhalte původ varování `SLIST_INCOMPLETE`. Můžete si opět zobrazit zdrojový kód. Zjištěný nedostatek v zdrojovém kódu daného obvodu sice nemá vliv na jeho chování v HW po syntéze, nicméně může způsobit nepříjemné chyby vedoucí k nevysvětlitelnému chování obvodu během simulace.

3. Odhalte původ varování `FSM_LIVELOCK`. Je možné názorně zobrazit přechody stavového automatu (`Show` $\longrightarrow$ `FSM`). V tomto případě se jedná o možnost přechodu ze stavu, do kterého už není definován přechod zpět. Konkrétně se jedná o počáteční stav, který umožňuje zpracování dalších operací. Opravte chybu v zdrojovém kódu, aby bylo možné spracovat další operace.

4. Odhalte původ varování `UNUSED_LOGIC`, který signalizuje nepoužitou logiku ve zdrojovém kódu. V tomto případě vhodně zkombinujte náhled do zdrojového kódu spolu s náhledem schémy obvodu (`Show` $\longrightarrow$ `Schematic`). Kontextovou nabídku pro tyto účely zobrazíte pravým tlačíkem myši, např. pro určitý signál. Najděte nevyužitou logiku, která je ve zdrojovém souboru definována.

Po opravě všech nedostatků nebude záložka Design Checks obsahovat žádné položky.

Nápověda: Význam jednotlivých porušení a varování je možno zobrazit přímo v nástroji Autocheck při práci s ním.

Úloha 3 — statická formální verifikace v nástroji Questa Formal.

V této úloze si vyzkoušíme možnosti nástroje Questa Formal. Pro verifikovaný obvod budeme definovat formální tvrzení, jejichž platnost bude nástroj během procesu verifikace sledovat.

Pro vytvářená formální tvrzení bude použit jazyk SVA a jednotlivá tvrzení budou definována v externích souborech. V tomto případě budeme pracovat se soubory ve složce `./formal_files`:

```
|-- formal_files
    |-- assertions.sv           // externí soubor s formálními tvrzeními
    |-- formal_directives.tcl   // direktivy pro formální verifikaci
    |-- init.seq                // inicializační skript
    `-- qformal_run.do          // spouštěcí skript
```

V následující části bude uveden seznam formálních tvrzení, které je potřeba vytvořit a zapsat do souboru `assertions.sv`. Uvedená tvrzení vytvářejte **postupně**! Pro každé formální tvrzení postupujte dle pokynů níže, experimentujte s nástrojem Questa Formal a sledujte výpisy z formální verifikace.

Tvrzení:

1. Vstupní signál `MOVI` nemůže nikdy nabývat hodnoty `3`, která je definována jako nepovolená (viz Tabulka 2).
2. Operace násobení trvá definovaný počet taktů — vytvořte tvrzení, které zachycuje vlastnost požadavku na provedení operaci násobení. V 1. taktu hodinového signálu musí být definována operace násobení (OP), ALU musí být aktivní (ACT) a připravena počítat (ALU_RDY). Dál po 1 taktu hodin jednotka `ALU` zneplatní signál `ALU_RDY`, což signalizuje probíhající operaci násobení a nemožnost přijímat požadavky na další aritmeticko-logické operace. Nakonec, po 9 taktech hodin, bude operace násobení dokončena a signál ALU_RDY bude mít opět aktivní hodnotu.
3. Žádný z datových vstupů nemůže mít nedefinovanou hodnotu (`'X'`) v případě požadavku na operaci `ALU`.
4. Výsledkem aritmetických operací jsou platné hodnoty v požadovaném rozsahu, jako příklad možnost uvést operací sčítaní (`ADD`), kdy součet hodnot vstupních operandů nepřesahuje hodnotu 255.

Příklad:

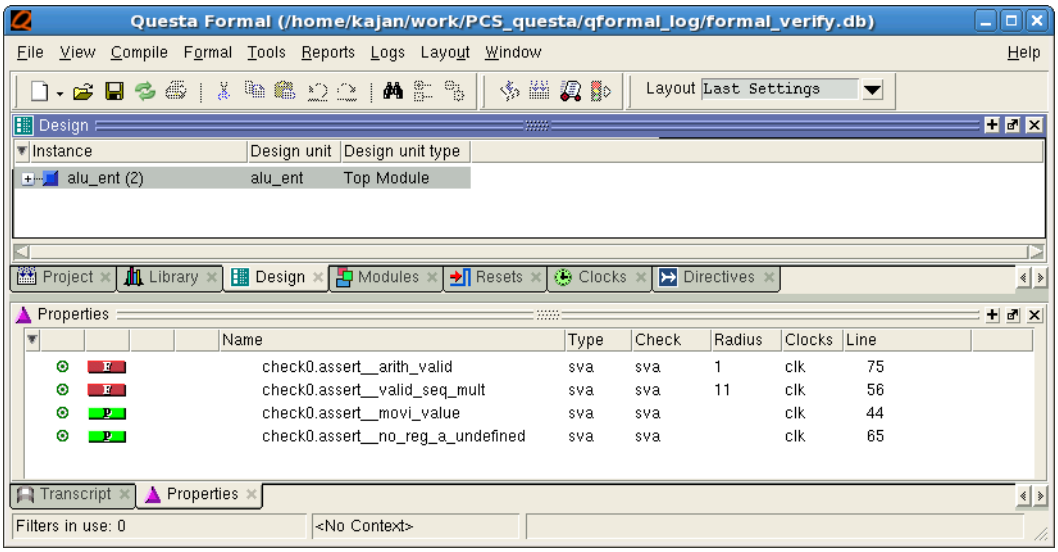
```
property valid_signals;
  @(posedge clk) disable iff (rst)
    a |-> b;
endproperty

assert property (valid_signals)
  else $error("Signals are not valid!\n");
```

Uvedené tvrzení umožňuje kontrolovat platnost signálů `a` a `b` a to za následujících podmínek: kontrola se provádí s každou nástupnou hranou uvedeného hodinového signálu (`posedge clk`), zároveň se ruší při aktivní hodnotě resetovacího signálu (`disable iff rst`). Během kontroly se sleduje následující podmínka: když má signál `a` platnou hodnotu, tak i signál `b` má platnou hodnotu.

Protože tvrzení samo o sobě neumožňuje verifikovat popsané chování nebo sbírat statistiky pokrytí, pro tyto účely slouží direktivy kontroly (`assert`). Uvedené direktivy pak umožňují kontrolovat splnění podmínek daných tvrzením a provádět definovanou akci v případě jeho splnění nebo nesplnění. V ukázkovém příkladu je zachycen případ nesplňení podmínky formálního tvrzení a použití systémového volání jazyka SystemVerilog (`$error`) pro výpis chyby za běhu verifikace.

Nápověda: Jako pomůcku k vytváření formálních tvrzení použijte vstupní prezentaci, dokumentaci od Mentor Graphics pro nástroj Questa Formal (`Assertions Quick Reference`) nebo některou z kníh, které jsou po celý čas k dispozici k nahlédnutí.



Obrázek 3: Prostředí Questa Formal se zobrazenými tvrzeními.

Překlad zdrojových souborů s formálními tvrzení a samotnou formální verifikaci spustíme pomocí skriptu:

```
# ./do_verify.sh
```

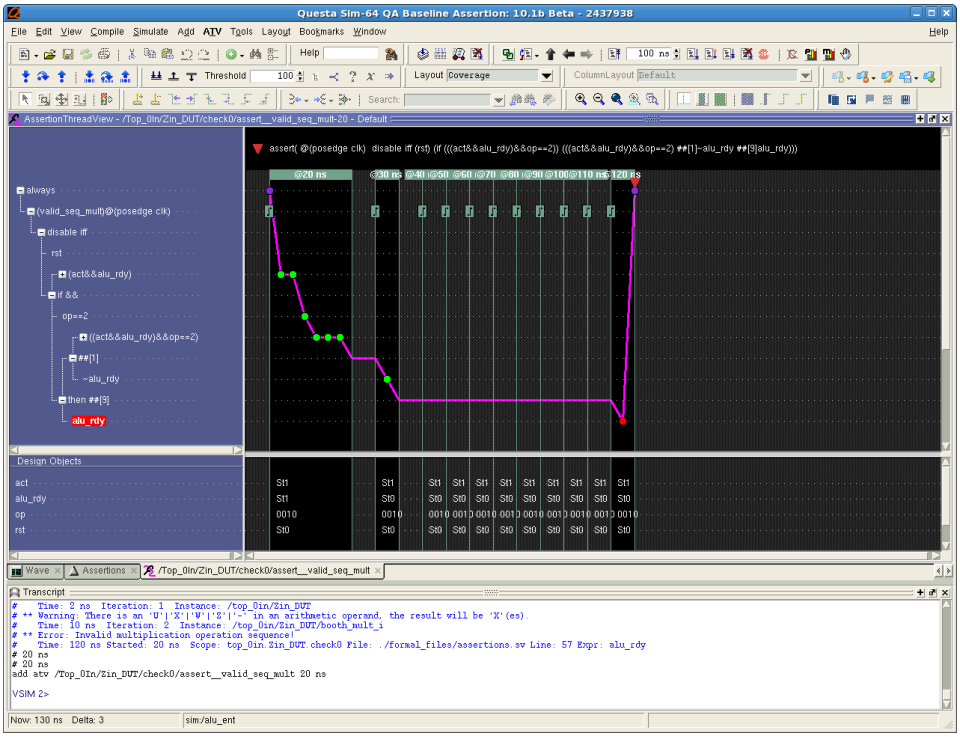
Výsledkem formální analýzy bude nová složka `qformal_log`, která bude obsahovat soubory s výsledky analýzy. Projděte informativně výstupní textové soubory z procesu formální verifikace a seznamte se s jejich obsahem.

Uživatelské rozhraní nástroje Questa Formal spustíme pomocí skriptu:

```
# ./run_gui.sh
```

Tip: Nahlédněte do uvedených skriptů, aby ste se seznámili s parametry jednotlivých nástrojů a s jakými vstupními soubory pracují.

V uživatelském rozhraní nástroje Questa Formal (obrázek 3) by všechna tvrzení měly být v této fázi označené jako *Firing*. To znamená, že pro uvedené tvrzení existuje protipříklad, kdy tvrzení není splněno. To může být způsobeno následujícími problémy: může se jednat o chybu ve verifikované komponentě, nesprávnou specifikaci samotného tvrzení, anebo tvrzení ve svojí definicí očekává vstupní hodnotu, která není povolena. Pro bližší informace použijte dostupnou dokumentaci.



Obrázek 4: Prostředí Questa Formal — Assertion Thread View.

Všimnete si možnosti zobrazení vyhodnocení formálního tvrzení v samostatném okně (`Show` ⟶ `Assertion Thread View`), které zachytává porušení některé z definovaných podmínek v přesném časovém sledu (obrázek 4).

Pro další práci s nástrojem Questa Formal provedeme další nastavení inicializačního skriptu

a direktiv pro formální analýzu.

Pro úpravy inicializačního skriptu "init_seq" pro další formální analýzu proveďte následující:

- počáteční hodnoty všech vstupních signálů mají mít hodnotu 0,
- všechny nadefinované registry mají být inicializovány na hodnotu 0,
- vytvořte referenci na hodinový signál definován v souboru s direktivami pro formální analýzu,
- nadefinujte resetovací sekvenci celého obvodu v počáteční fázi formální analýzy.

Po modifikaci inicializačího skriptu proveďte celou formální analýzu znovu, spusťe uživatelské rozhraní a sledujte změny.

V poslední fázi provedeme úpravy direktiv pro formální verifikaci. Ty budou sestaveny z posloupnosti úprav, cílem kterých je dosažení stavu *Proven* u tří formálních tvrzení (formální tvrzení pro aritmetickou operaci je možno buď odstranit nebo upravit zdrojový kód jednotky `ALU`):

- nastavte periodu hodin pro hodinový signál `CLK`,
- signál `MOVI` nemůže nabývat hodnot `'11'` (binárně),
- signál `ACT` musí být vždy aktivní.

Nápověda: Uvedené úpravy se týkají nastavení omezujících podmínek pro dané signály.