

## Teoretický úvod – laboratorní úloha č. 7

# Realizace stavového automatu v jazyce VHDL

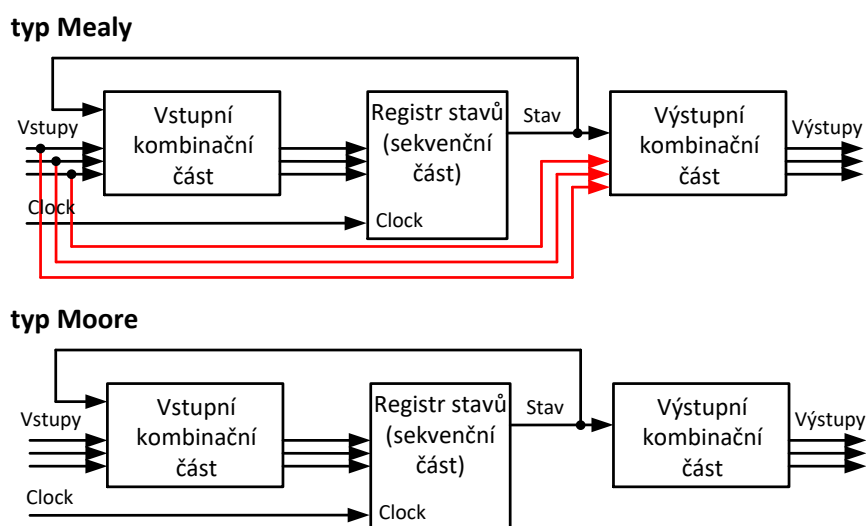
## 1. Stavové automaty typu Mealy a Moore

Stavové automaty jsou sekvenční obvody, které se mohou v jednom okamžiku nacházet vždy jen v jednom stavu z definované množiny konečného počtu stavů, proto se někdy též označují jako konečné stavové automaty<sup>1</sup>. Obvod pak podle definovaných podmínek a na základě vnitřního či vnějšího buzení přechází postupně z jednoho stavu do následného, což označujeme jako tzv. přechody.

V této laboratorní úloze se budeme zabývat pouze automaty, jejichž sekvenční část je synchronizována pomocí hodinového vstupu `Clock` a tyto automaty tak přecházejí mezi stavy synchronně. Výstup obvodu je pak závislý buď na vzájemné kombinaci buzení (vstupů obvodu) a stavu automatu, takový automat označujeme jako typ Mealy, nebo je závislý pouze na vnitřním stavu automatu a pak jej označujeme jako typ Moore. Obr. č. 1 schematicky ukazuje oba typy.

Každý stavový automat obsahuje kombinační část (kombinační obvod) a sekvenční část (sekvenční obvod), která slouží pro uchování informace o stavu obvodu a jeho přechodu do následného stavu. Kombinační obvody sestávají z elementárních hradel, sekvenční část pak z klopných obvodů, typicky z D či J-K. Každý stavový automat můžeme popsat pomocí tzv. výstupních funkcí a přechodových funkcí, takový popis tak vede na soustavu Booleovských rovnic.

Je výhodné graficky popisovat stavové automaty pomocí tzv. přechodových diagramů<sup>2</sup>, ve kterých znázorníme jednotlivé stavy automatu jako uzly grafu a hrany pak tvoří přechody mezi nimi se zapsáním těchto přechodových podmínek a patřičných výstupů do grafu. Jinou možnost představují tzv. přechodové tabulky, v nichž popíšeme chování automatu při procházení jednotlivých stavů v závislosti na vstupech a generování výstupu obvodu dle typu automatu.

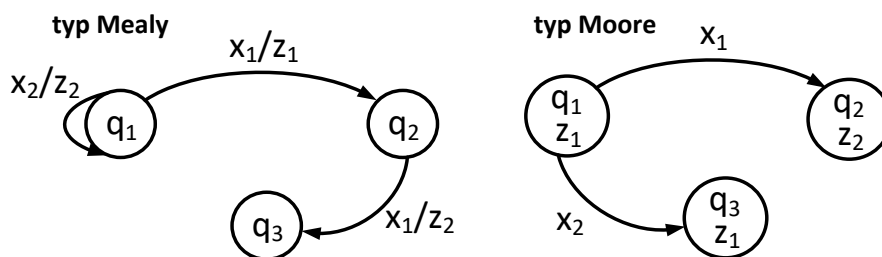


Obr. č. 1: Schematická ilustrace rozdílu mezi automaty typu Mealy a Moore.

<sup>1</sup> V anglické literatuře se používá zkratka FSM – Finite State Machine.

<sup>2</sup> Někdy se také označují jako stavové diagramy.

Obr. č. 2 ilustruje rozdíl mezi automatem typu Mealy a Moore na příkladu části přechodového diagramu.



Obr. č. 2: Ukázka části přechodového diagramu automatu typu Mealy a Moore.

Obr. č. 2 ukazuje, že výstup  $Z$  obvodu typu Mealy je generován při přechodu z jednoho stavu  $q$  do druhého v případě vstupní podmínky  $x$ , zatímco u automatu typu Moore je vygenerován až po přechodu do nového stavu  $q$ .

Obr. č. 1 rovněž dokumentuje, že kombinace vstupů se promítne u Mealyho automatu okamžitě do výstupu obvodu, zatímco v případě Mooreova typu tato vstupní kombinace ovlivní pouze přechod do nového stavu a teprve při dalším přechodu se dostane na výstup. Každý automat Mealyho typu lze vždy převést na typ Moore, opačný převod je možný za určitých podmínek (více o automatech v přednáškách).

## 2. Stavové automaty v jazyce VHDL, deklarace vlastního datového typu, kódování stavů

V jazyce VHDL se používá prakticky výlučně behaviorální popis chování automatu pro jeho realizaci a implementaci, odpadá proto řada kroků při ručním návrhu automatů (tabulky přechodů, kódování stavů, minimalizace a realizace přechodových funkcí apod.); ty provede automaticky proces syntézy. Důležitý je samozřejmě výběr typu automatu, Mealy či Moore, pro realizaci daného zadání, neboť oba typy automatů odlišuje jejich chování a výhody či nevýhody, jak bylo uvedeno na přednášce.

Hlavním rozdílem z hlediska jejich realizace v jazyce VHDL je, že Mealyho automat reaguje rychleji na změny na svém vstupu, zatímco u Mooreova automatu se daná změna promítne až po přechodu do nového stavu, který je řízen pomocí hran hodinového signálu `Clock`. Na druhou stranu je u Mealyho automatu generován výstup asynchronně, neboť je dán kombinačním výstupem vstupu a stavu, zatímco v případě Mooreova automatu je výstup vždy synchronní, odpadá tedy případný problém s nesynchronním výstupem obvodu.

S ohledem na behaviorální realizaci v jazyce VHDL lze obvykle vytvořit nejružnější varianty výsledného kódu, je však potřeba se řídit doporučeným postupem návrhu a doporučenou strukturou VHDL kódu tak, aby syntezátor provedl optimální mapování navrženého automatu do zvoleného programovatelného pole. Syntezátor totiž obvykle hledá při procesu syntézy standardizované bloky v zapsaném kódu jazyka VHDL, které pak zpracovává daným postupem.

Doporučené předpřipravené kódy pro realizaci konečného stavového automatu v prostředí programu Quartus v okně editoru VHDL nalezneme po kliknutí na ikonu s názvem *Insert Template* (případně v menu *Edit* → *Insert Template...*), kde v nabídce okna *Insert Template* rozklikneme postupně nabídku „VHDL – Full Designs – State Machines“, kde se nachází doporučený popis v jazyce VHDL jak pro čtyřstavový automat typu Mealy (Four-State Mealy State Machine), tak čtyřstavový automat typu Moore (Four-State Moore State Machine), na tyto kódy se podíváme v další kapitole, a další 2 typy stavových automatů.

V předchozích úlohách a na přednáškách jsme se zmínili, že v jazyce VHDL lze vytvářet vlastní datové typy a sub-typy a to buď výčtem, nebo omezením standardních datových typů. Pro realizaci stavového automatu je výhodné vytvořit vlastní výčtový datový typ, který bude obsahovat množinu možných stavů automatu, a jeho přiřazením do pomocného signálu můžeme jednoduše ovládat procházení jednotlivých stavů při přechodech obvodu. Deklaraci zapisujeme v úvodu architektury mezi klíčová slova `architecture` a `begin` a vlastní syntaxe je následující:


```
type {nazev_typu} is ({stav1},{stav2},{stav3},...{stavn});
signal {nazev_signalu} : {nazev_typu};
```

V souvislosti s deklarací stavů automatu se stručně zmíníme i o jejich kódování. Při ručním návrhu stavového automatu je jedním z důležitých kroků tzv. kódování vnitřních stavů automatu, při němž přiřazujeme každému možnému stavu jeho reprezentaci pomocí některého binárního kódu. Toto kódování může výrazně ovlivnit celkové výsledné parametry realizace navrženého automatu, např. počet potřebných klopných obvodů pro realizaci sekvenční (registrové) části, spotřebu obvodu, celkové výstupní zpoždění apod.

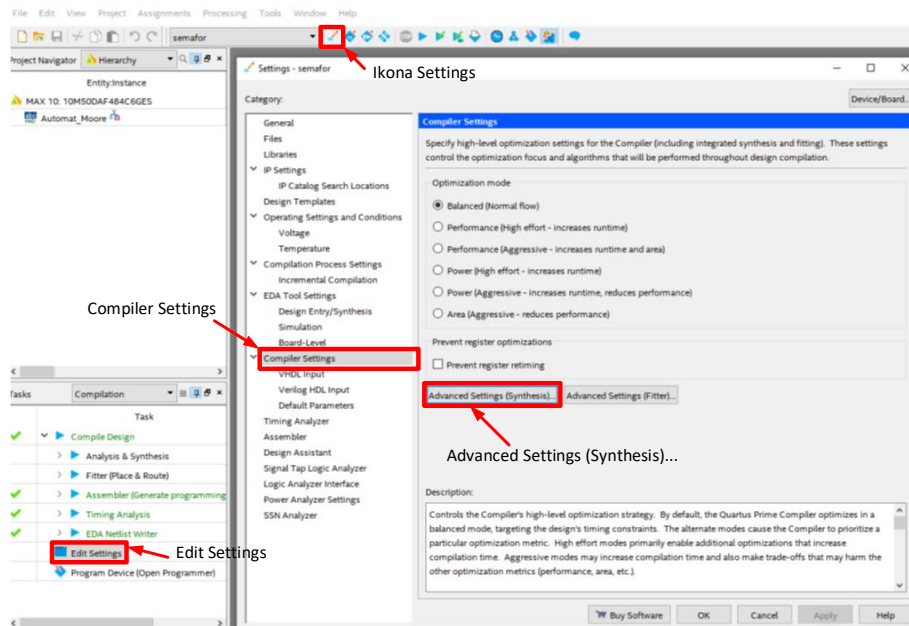
Pro toto kódování vnitřních stavů lze využít řadu kódů, každý z nich má své specifické výhody a nevýhody. Pro implementaci do hradlových polí se nejčastěji používají kódy 1 z  $n$  (tzv. one-hot) či binární kód (tzv. sequential). Kód 1 z  $n$  můžeme jednoduše popsat jako poziční kód, kdy kódové slovo obsahuje vždy přesně jednu hodnotu logická 1 na dané pozici, zatímco zbývající bitové pozice tvoří logické 0. Nevýhodou tohoto kódu je velký počet nepracovních stavů (bitové pozice logických 0, pro  $n$  bitů stavů je potřeba  $n$  klopných obvodů), avšak jeho dekódování je jednoduché pomocí klopných obvodů (kterých je ale potřeba větší množství), což přináší výhodu při realizaci automatu v obvodech FPGA, které obsahují obvykle velké množství klopných obvodů D.

Binární kód (tzv. sequential) je naproti tomu jednoduchý, vyžaduje menší počet stavových bitů (klopných obvodů, pro  $n$  bitů stavů je potřeba  $\log_2(n)$  klopných obvodů) a obvykle mnohem menší počet nepracovních stavů. Pro jeho dekódování je však potřeba větší počet logických hradel (pro větší počet termů), a proto je vhodnější pro programovatelné obvody typu CPLD. V jazyce VHDL lze pomocí tzv. atributů ovlivnit výsledek syntézy a implementace, při návrhu a realizaci stavových automatů můžeme využít této syntaxe:

```
attribute ENUM_ENCODING : string;
attribute ENUM_ENCODING of {nazev_typu} : type is "kod stavu1 kod stavu 2
... kod stavu n"; -- napr. 00 01 10 11
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of {nazev_signalu} : signal is
"{nazev_kodovani_stavu}"; -- one-hot, sequential, gray, auto...
```

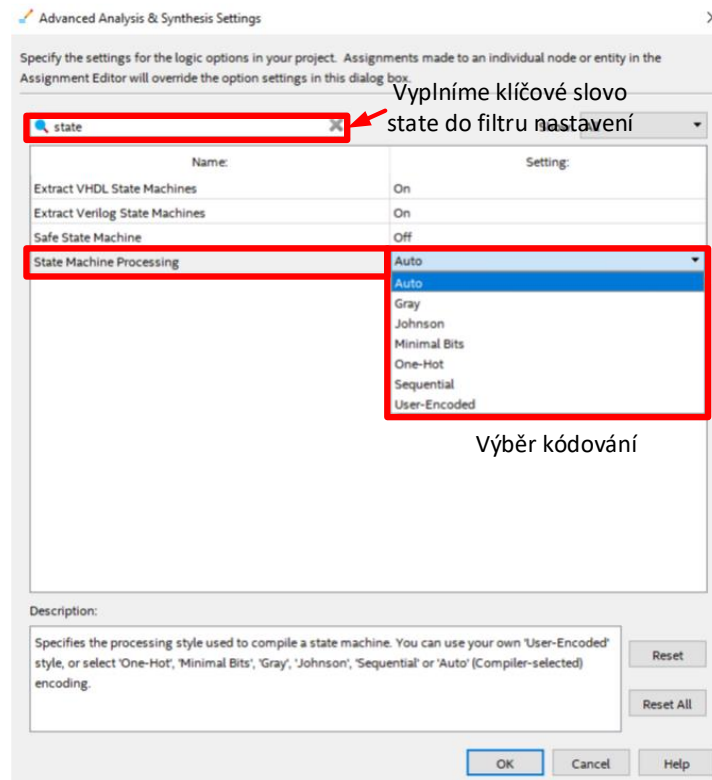
Polím `nazev_typu` a `nazev_signalu` odpovídají deklarace uvedené výše. Prvním atributem `ENUM_ENCODING` ovlivníme přiřazení kódů jednotlivým stavům, atributem `FSM_ENCODING` pak způsob zakódování těchto stavů. Alternativně můžeme v programu Quartus v levém dolním okně „Tasks“ dvojklíkem levým tlačítkem otevřít položku *Edit Settings* (případně v hlavní liště ikon kliknout na ikonu  *Settings*) a v otevřeném okně *Settings* najdeme položku *Compiler Settings*. Zde v okně vpravo pak klikneme na tlačítko *Advanced Settings (Synthesis)*...

Obr. č. 3 ilustruje tuto situaci.



Obr. č. 3: Okno s nastavením Syntezátoru a jeho parametrů.

V nově otevřeném okně *Advanced Analysis & Synthesis Settings* vyplníme v okénku pro vyhledávání v levém horním rohu výraz „state“ pro vyfiltrování nastavení týkajících se pouze stavových automatů. Ze 4 vyhledaných položek nás pak bude zajímat poslední s názvem *State Machine Processing*, která umožňuje vybrat a změnit způsob kódování vnitřních stavů automatu. Obr. č. 4 ukazuje situaci po kliknutí v pravém sloupečku u této položky, kdy dojde k rozbalení roletové nabídky možných způsobů kódování; můžeme zvolit z možností: *Auto*, *Gray*, *Johnson*, *Minimal Bits*, *One-Hot*, *Sequential*, *User-Encoded*.



Obr. č. 4: Nastavení způsobu kódování vnitřních stavů automatu.

Nyní můžeme již přejít k vlastní realizaci stavového automatu v jazyce VHDL. Ta je obvykle řešena rozdělením do 3 samostatných podmínkových konstrukcí. První představuje v sekvenčním prostředí proces synchronizace přechodů celého automatu pomocí detekce vzestupné či sestupné hrany hodinového vstupu Clock, obvyklý je rovněž vstup Reset (synchronní či asynchronní).

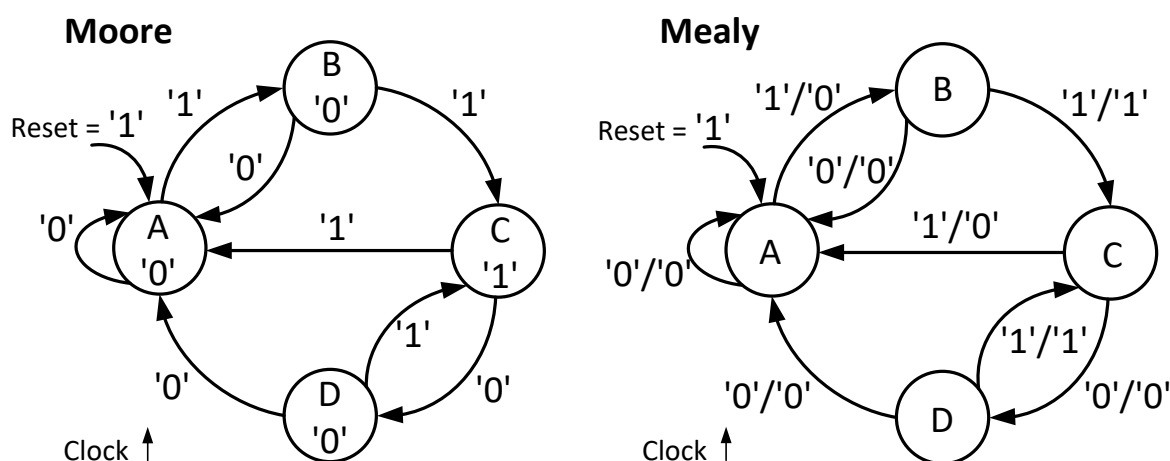
Druhá podmínková část řeší výstupy automatu; tu lze vytvořit jak v sekvenčním, tak i paralelním prostředí. V této části se liší implementace automatu typu Mealy a Moore, zatímco pro automat Mooreova typu vyhodnocujeme na výstup obvodu v podmínkách pouze stav obvodu, v případě Mealyho typu testujeme v podmínkách stav i vstup obvodu.

Poslední podmínkovou část je opět možné zapsat v paralelním nebo sekvenčním prostředí a tato podmínková konstrukce řeší přechody obvodu, tedy z jakého současného stavu lze přejít do jakého následného při jaké podmínce. Vše si ukážeme v další kapitole na příkladu.

### 3. Ukázka návrhu stavového automatu v jazyce VHDL

V této kapitole provedeme ukázkový návrh stavového automatu a porovnání realizace typu Moore a Mealy. Obr. č. 5 a slovní popis pro nás budou výchozí.

Stavový automat má stavy A, B, C, D, hodinový vstup Clock (na jehož vzestupné hrany probíhají přechody automatu), vstup Reset (při resetování se automat synchronně vrací do stavu A) a vstupní port Vstup, výstupem je port Vystup.



Obr. č. 5: Přechodové diagramy pro ukázkový stavový automat pro realizaci typem Moore i typem Mealy.

Realizujeme automat nejprve typem Moore.

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Automat_Moore is
port (Clock,Reset,Vstup : in std_logic;
      Vystup : out std_logic);
end Automat_Moore;
architecture Behavioral of Automat_Moore is
type stavy is (A,B,C,D);
signal soucasny_stav,novy_stav : stavy;
begin

process (Clock)
begin
if Clock='1' and Clock'event then
    if Reset='1' then
        soucasny_stav<=A;
    else
        soucasny_stav<=novy_stav;
    end if;
end if;
end process;

process (soucasny_stav)
begin
Vystup<='0';
case (soucasny_stav) is
    when C =>
        Vystup<='1';
    when others =>
        Vystup<='0';
end case;
end process;

process (soucasny_stav,Vstup)
begin
novy_stav<=soucasny_stav;
case (soucasny_stav) is
    when A =>
        if Vstup='1' then
            novy_stav<=B;
        else
            novy_stav<=A;
        end if;
    when B =>
        if Vstup='1' then
            novy_stav<=C;
        else
            novy_stav<=A;
        end if;

```

```

when C =>
  if Vstup='1' then
    novy_stav<=A;
  else
    novy_stav<=D;
  end if;
when others =>
  if Vstup='1' then
    novy_stav<=C;
  else
    novy_stav<=A;
  end if;
end case;
end process;
end Behavioral;

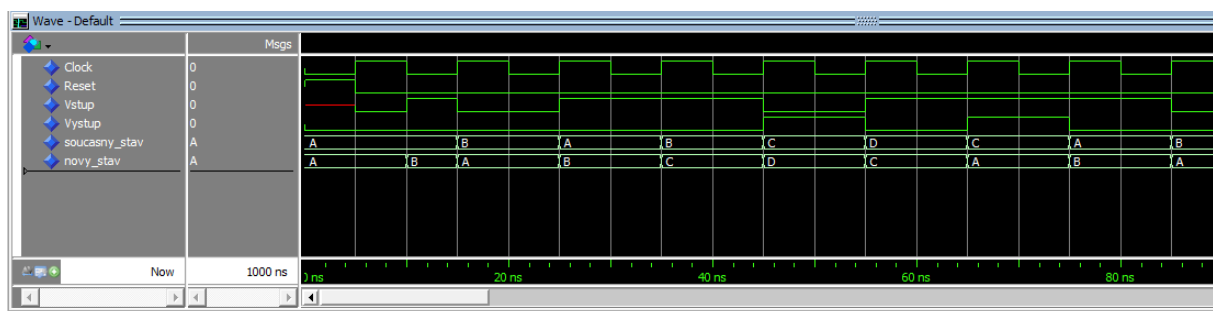
```

V entitě deklarujeme použité porty, dále datový typ `stav`, který obsahuje výčet jednotlivých stavů, A, B, C, D. Pro řízení přechodů automatu a uložení aktuálního a následného stavu deklarujeme dvojici signálů `soucasny_stav`, `novy_stav`. První proces je pouze synchronizace na vzestupné hrany vstupního taktu `Clock` a v případě aktivování vstupu `Reset` přechází automat do stavu A.

Druhý proces řídí výstup automatu. Ten v případě automatu typu Moore závisí pouze na stavu obvodu, proto citlivostní seznam procesu obsahuje jen signál `soucasny_stav`. Jako první je použito přiřazení logické 0 do portu `Vystup`, toto ošetření je v jazyce VHDL vhodné, aby byla předdefinována hodnota ještě před vlastním provedením příkazu `case`, což zabrání případným nechtěným vznikům latchů a problémů s logickými hazardy (více viz přednášky). `Vystup` má být logická 1 pouze ve stavu C.

Poslední proces zajišťuje přechody mezi stavy na základě hodnot `soucasny_stav` a `Vstup`. V jeho úvodu je opět před-definice hodnoty, jako v předchozím procesu.

Obr. č. 5 nám pak pomůže popsat jednotlivé možnosti stavů, z kterého a kam a za jaké podmínky může automat přejít. Provedeme nyní simulaci navrženého automatu na úrovni RTL, abychom ověřili správnost jeho funkce. Obr. č. 6 ukazuje její výsledek.



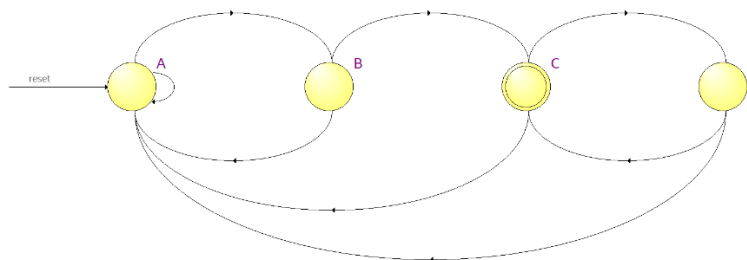
Obr. č. 6: Výsledek RTL simulace navrženého automatu typu Moore.

Na začátku simulace byl nastaven vstup `Reset = '1'` po dobu 5 ns a poté bylo resetování ukončeno. Po dalších 5 ns byl vstup `Vstup` nastaven do hodnoty logické 1 po dobu 5 ns a při příchodu vzestupné hrany hodinového vstupu `Clock` v čase simulace 15 ns tak automat přešel do stavu B. Protože při další vzestupné hraně hodinového vstupu `Clock` byl nastaven `Vstup = '0'`, přešel automat zpět do stavu A. Obr. č. 5 dokumentuje tuto situaci.

Poté byl od času 25 ns do 45 ns nastaven `Vstup` na hodnotu '1', proto automat při vzestupné hraně hodinového vstupu `Clock` v čase 35 ns nejprve přešel do stavu B a následně v čase 45 ns do stavu C, kdy se na jeho výstupním portu `Vystup` objevila hodnota '1'. Vstup byl mezitím nastaven zpět na logickou 0 a automat tak ze stavu C přešel v čase 55 ns do stavu D. V čase 65 ns se opět při vzestupné hraně hodinového vstupu `Clock` nacházel vstupní port `Vstup` ve stavu logické 1, proto se automat zpět ze stavu D vrátil do stavu C a na výstupu se opět objevila hodnota '1', a protože byl vstup nadále ve stavu logická 1, přešel automat v čase 75 ns do stavu A.

V programu Quartus si rovněž můžeme prohlédnout a zkontrolovat přechodový (stavový) diagram automatu tak, jak jej navrhl syntežátor na základě námi provedeného popisu. Obr. č. 5 (výchozí) můžeme konfrontovat s diagramem (zda odpovídá zadanému popisu i diagramu).

Zobrazení diagramu provedeme volbou v menu *Tools* → *Netlist Viewers* → *State Machine Viewer*. Obr. č. 7 ukazuje, jak je v otevřeném okně *State Machine Viewer* vykreslen přechodový (stavový) diagram automatu. Ve spodní části okna lze zobrazit tabulku přechodů (Transitions) a kódování vnitřních stavů (Encoding).



Obr. č. 7: Přechodový (stavový) diagram navrženého automatu typu Moore vygenerovaný syntežátorem.

Porovnáním obou diagramů je zřejmé, že námi vytvořený popis automatu v jazyce VHDL byl proveden správně a oba diagramy jsou totožné.



Obr. č. 5 bude nyní předlohou pro realizaci automatu typu Mealy jazyce VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Automat_Mealy is
    -- stejná deklarace portu jako v případě typu Moore
end Automat_Mealy;
architecture Behavioral of Automat_Mealy is
    type stavy is (A,B,C,D);
    signal soucasny_stav,novy_stav : stavy;
begin

    process (Clock)
    begin
        -- tento proces je zcela stejný jako v případě typu Moore
    end process;

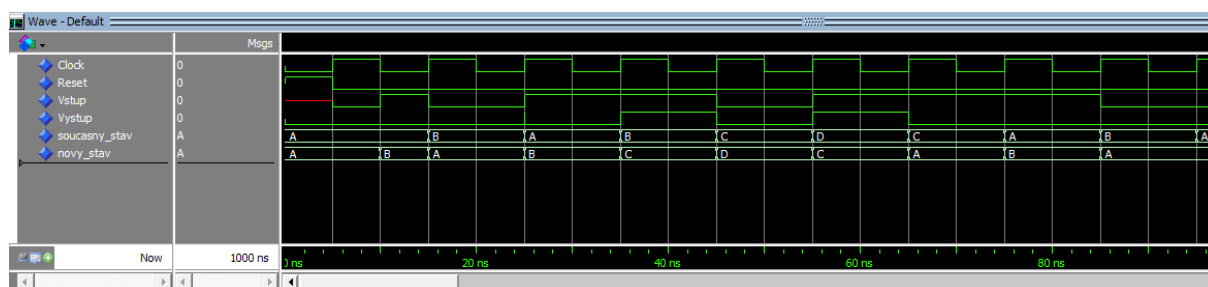
    process (soucasny_stav,Vstup)
    begin
        Vystup<='0';
        case (soucasny_stav) is
            when B|D =>
                if Vstup='1' then
                    Vystup<='1';
                else
                    Vystup<='0';
                end if;
            when others =>
                Vystup<='0';
        end case;
    end process;

    process (soucasny_stav,Vstup)
    begin
        -- tento proces je zcela stejný jako v případě typu Moore
    end process;
end Behavioral;
```

Jedinou odlišností od předchozího VHDL kódu automatu typu Moore je prostřední proces, který definuje výstupy automatu. V případě typu Mealy se v citlivostním seznamu procesu nachází jak signál `soucasny_stav`, tak i vstupní port `Vstup`, neboť v případě automatu typu Mealy závisí výstup obvodu na obou těchto hodnotách.

Po prvotním ošetření (před-definování) následuje vlastní podmínka typu `case`, kdy, jak z přechodového diagramu na Obr. č. 5 vyplývá, je `Vystup` roven logické 1 v případě, že automat přechází ze stavu B do stavu C (když je ve stavu B a `Vstup = '1'`) a pokud automat přechází ze stavu D do stavu C (když je ve stavu D a `Vstup = '1'`).

Bylo by samozřejmě možné v podmínkách `case` definovat obě možnosti zvlášť, ale pomocí znaku `|` (pipy) ve významu „nebo“ v podmínkách VHDL je můžeme sloučit. Nyní provedeme opět RTL simulaci automatu Mealy se stejně definovanými budícími vstupy, jako v případě předchozí simulace pro typ Moore.

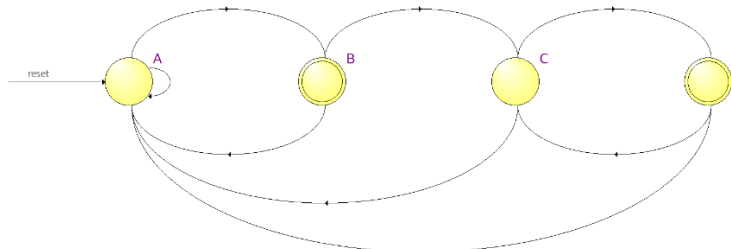


Obr. č. 8: RTL simulace automatu typu Mealy.

Obr. č. 8 ukazuje rozdíl při realizaci typem Mealy oproti typu Moore: je zřejmé, že výstup obvodu Vystup je v případě typu Mealy nastaven již v okamžiku přechodu a nikoliv až po přechodu do nového stavu, jako v případě typu Moore.

U obvodu typu Moore se projeví dodatečné čekání na vzestupnou hranu hodinového vstupu `Clock`, aby byl aktualizován výstup obvodu, u obvodu typu Mealy se toto děje nesynchronně bez nutného čekání. Na druhou stranu však v případě použití automatu typu Mealy je nutné počítat s nesynchronním výstupem a jeho vlivem na další části obvodu, a proto bývá v určitých situacích bezpečnější použití automatu typu Moore, jehož výstup je vždy synchronní.

Obr. č. 9 představuje vygenerovaný přechodový diagram vytvořený syntezátorem (použijeme opět postup jako v případě předchozího automatu typu Moore).

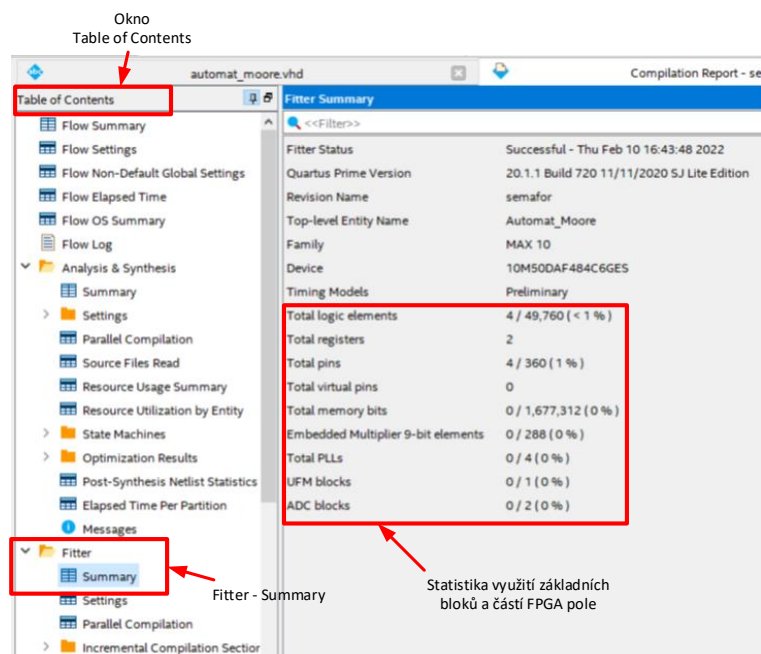


Obr. č. 9: Přechodový (stavový) diagram navrženého automatu typu Mealy vygenerovaný syntezátorem.

Z vygenerovaného diagramu je patrné, že je Vystup roven logické 1 ve stavech B a D. Avšak vzhledem k tomu, že automat je typu Mealy, víme, že k nastavení hodnoty Vystup na logickou 1 dochází při přechodu ze stavu B do stavu C a ze stavu D do stavu C.

Na příkladu automatu typu Moore ilustrujme nyní rozdíl při jeho implementaci, pokud zvolíme kódování jeho vnitřních stavů pomocí binárního kódu (sequential) a kódu 1 z n (one-hot). Pomocí jednoho z obou možných postupů popsaných v předchozí kapitole 2 nastavíme postupně oba způsoby kódování. Pro každý z nich pak provedeme syntézu navrženého automatu pomocí kompilace celého projektu.

Dále klikneme v levé části hlavního okna programu Quartus do podokna s názvem *Table of Contents*. Obr. č. 10 zobrazuje statistiku využití základních bloků a stavebních prvků FPGA pole a informace o celém projektu (po rozkliknutí položky *Fitter* → *Summary*).



Obr. č. 10: Základní informace a statistiky projektu.

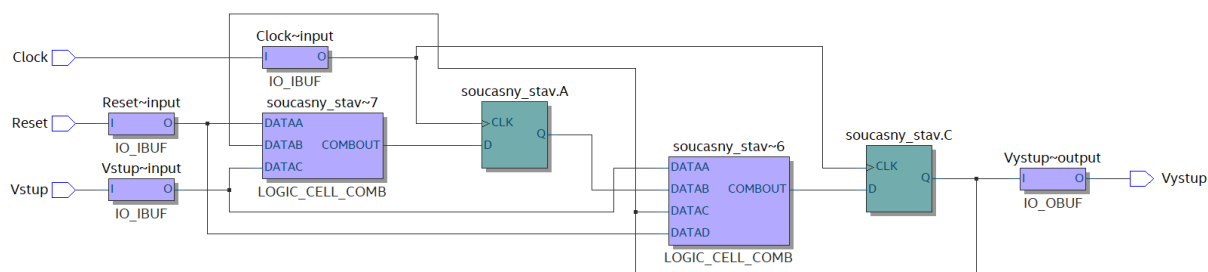
V tomto projektu nás bude zajímat především počet obsazených logických elementů LE (Logic Element)<sup>3</sup>, každý obsahuje jednu 4vstupovou LUT tabulku (Look-up Table) a 1 programovatelný registr sloužící jako klopový obvod typu D.

Tab. č. 1 přehledně ukazuje porovnání počtu alokovaných základních prvků FPGA pole pro implementaci vytvořeného automatu při použití binárního kódu a kódu 1 z n pro kódování vnitřních stavů ukázkového automatu.

Tab. č. 1: Porovnání počtu obsazených prvků pro různé typy kódování.

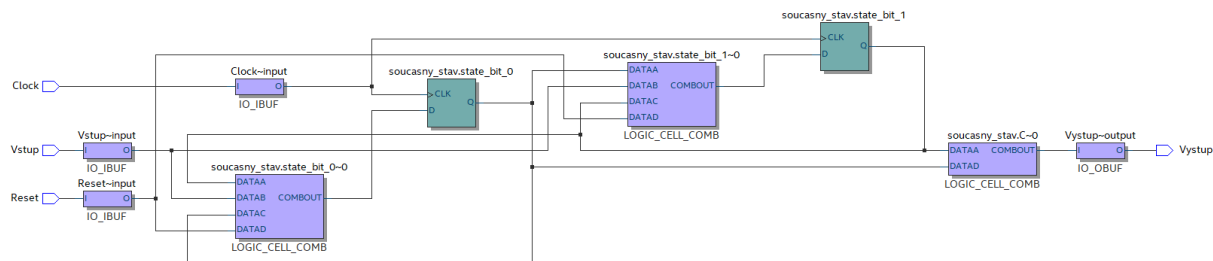
Typ kódování vnitřních stavů / počet obsazených prvků	Binární (sequential)	1 z n (one-hot)
Počet logických elementů LE	4	3
Počet registrů (klop. obvodů D)	2	2

Porovnání realizace automatu při výběru různého kódování můžeme provést rovněž ze schématu pro implementaci do přípravku DE10-Lite na úrovni bloků FPGA pole (tzv. technologické schéma) po jejich namapování (Post-Mapping), které jsme již používali v laboratorní úloze č. 3. Obr. č. 11 představuje schéma realizace navrženého automatu typu Moore při použití kódování vnitřních stavů pomocí kódu 1 z n. Obr. č. 12 představuje technologické schéma při použití binárního kódování.



Obr. č. 11: Technologické schéma implementace automatu při použití kódu 1 z n.

<sup>3</sup> Podrobněji jsme si stavbu FPGA MAX10 použitého na přípravku DE10-Lite popisovali na přednášce. Logický element LE je nejmenší logickou jednotkou FPGA pole, 16 LE je sdruženo v 1 logickém bloku LAB (Logic Array Block).



Obr. č. 12: Technologické schéma implementace automatu při použití binárního kódu.

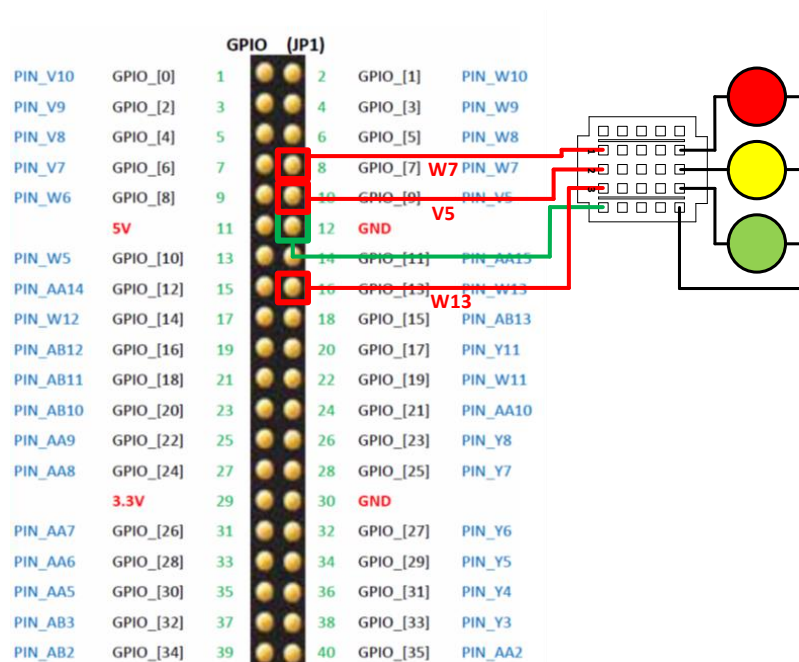
Z porovnání vyplývá, že pro realizaci automatu při volbě kódování sequential (binární kód) je potřeba trojice LUT tabulek (3 kombinační obvody), zatímco při volbě kódování 1 z n nám stačí pouze 2 LUT (2 kombinační obvody). Kliknutím pravým tlačítkem na daný kombinační obvod ve schématu lze v položce *Properties* zobrazit zapojení elementárních hradel daného kombinačního obvodu a také jeho pravdivostní tabulku a výstupní funkci.

## 4. Popis chování semaforu, připojení k přípravku DE10-Lite

Hlavním úkolem v této laboratorní úloze je provést návrh a realizaci stavového automatu, který simuluje chování silničního semaforu v křižovatce, a provést jeho implementaci do kitu DE10-Lite a ověřit jeho funkčnost díky přípravku semaforu připojeného do tohoto kitu. V této kapitole je slovně popsána funkce semaforu, na základě tohoto popisu nakreslíte přechodový diagram automatu a realizujete jej v jazyce VHDL.

Semafor obsahuje 3 barevné LED diody – červenou, žlutou a zelenou. Semafor se může nacházet ve 4 různých stavech – *cervena* (svítí pouze červená LED), *cervena\_zluta* (svítí dvě LED, červená a žlutá), *zelená* (svítí pouze zelená LED), *zluta* (svítí pouze žlutá LED). Přechody semaforu (automatu) se odehrávají synchronně na vzestupné hrany hodinového vstupu *Clock*. Na začátku je semafor ve stavu *cervena*, po 4 sekundách přejde do stavu *cervena\_zluta*, po 1 sekundě přejde do stavu *zelená*, po 4 sekundách přejde do stavu *zluta* a odtud po 1 sekundě zpět do stavu *cervena*. Semafor obsahuje vstup *Reset*, při jehož aktivování se semafor dostane okamžitě do stavu *cervena*. Dále obsahuje semafor vstup *Rizeni*, který zajistí, že semafor přechází pouze mezi stavy *cervena* a *zelená* navzájem vždy po 4 sekundách, stavy *cervena\_zluta* a *zluta* jsou vynechány.

Zkontrolujte připojení přípravku semaforu (3 LED diod) k přípravku DE10-Lite. Obr. č. 13 naznačuje připojení přípravku DE10-Lite ke svítivým diodám LED: 1 společná zem (GND) a 3 vodiče k anodám jednotlivých diod.



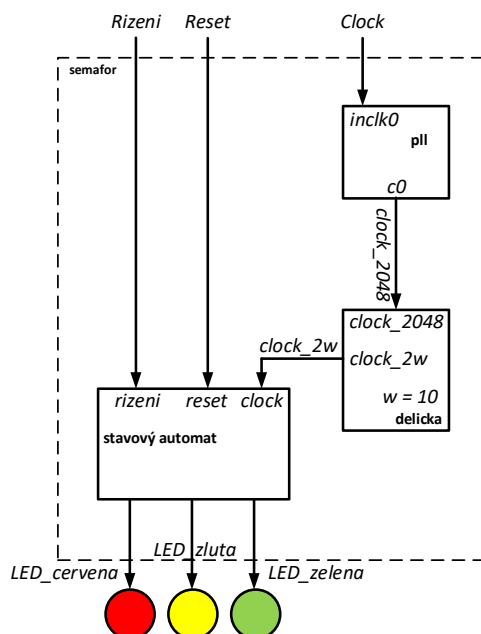
Obr. č. 13: Připojení přípravku semaforu (3 LEDky) ke kitu DE10-Lite.

## 5. Realizace děličky kmitočtu 1 Hz pro ovládání semaforu

Jak vyplývá z předchozího popisu činnosti semaforu (v kapitole 4), budeme pro přechody mezi jeho stavy potřebovat dvojici dob čekání, 1 sekundu a 4 sekundy. Za základ zvolíme hodinový signál s periodou 1 sekunda (1 Hz), dobu čekání 4 sekundy pak z něho dodatečně odvodíme jednoduše tak, že budeme čítat 4 jeho vzestupné hrany. Pro realizaci základního taktu 1 Hz využijeme PLL fázový závěs a děličku frekvence typu  $2^W$  z laboratorní úlohy č. 6. Pomocí fázového závěsu vytvoříme hodinový signál s frekvencí 2048 Hz ze zdrojového kmitočtu 10 MHz a z něho následně vhodně navrženou děličkou typu  $2^W$  vytvoříme konečný hodinový signál s frekvencí 1 Hz přesně tak, jak jsme postupovali v úloze č. 6. Jak bylo naznačeno výše, pro dobu čekání 4 sekundy pak jen vytvoříme pomocný čítač, který bude čítat vzestupné hrany hodinového signálu 1 Hz a při načítání 4 jeho vzestupných hran bude výstupem čítače doba čekání 4 sekundy (odpovídá frekvenci 0,25 Hz).

## 6. Realizace stavového automatu a celého projektu semaforu v jazyce VHDL

V této kapitole si nastíníme možnost realizace stavového automatu – semaforu v jazyce VHDL na základě popisu jeho činnosti v kapitole 4. Obr. č. 14 vhodně uvádí celkové řešení semaforu.



Obr. č. 14: Zjednodušené blokové schéma navrhovaného semaforu.

Výslednou entitu semaforu vytvoříme ze 3 bloků – komponenty pll (PLL fázový závěs použitý jako preddělička kmitočtu), komponenty delicka (dělička frekvence ve druhém stupni pro vytvoření požadovaného hodinového taktu 1 Hz) a konečně z vlastního stavového automatu pro ovládání semaforu. Dvě komponenty děliček kmitočtů, pll a delicka, převezmeme v podstatě bez jakékoliv úpravy z laboratorní úlohy č. 6. Vytvořené komponenty pll a delicka pak jen využijeme v rámci entity semafor a provedeme jejich správné namapování.

Možných realizací a zápisů VHDL kódu výsledné entity semafor je celá řada, pro inspiraci můžeme uvést základní rysy jedné z možných; uvedený VHDL kód je však potřeba doplnit (viz domácí přípravu).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity semafor is
port (clock,reset,rizeni : in std_logic;
      LED_cervena,LED_zelena,LED_zluta : out std_logic);
end semafor;

architecture Behavioral of semafor is
type state_type is (cervena,cervena_zluta,zelena,zluta);
signal state : state_type;
signal clock_2048,clock_2w : std_logic;

component delicka is
...
end component;
```

```

component pll is
...
end component;

begin

delicka_1: pll
port map (...);

delicka_2: delicka
generic map(...)
port map (...);

process(reset,clock_2w)
variable count : integer range 0 to 2 := 0;
begin
if reset='0' then
    state <= cervena;
    LED_cervena <='1';
    LED_zluta <='0';
    LED_zelena <='0';
elsif clock_2w='1' and clock_2w'event then
    case state is
        when cervena =>
            ...
            if count=2 then
                count := 0;
                if rizeni='0' then
                    ...
                else
                    ...
                end if;
            when cervena_zluta =>
                ...
            when zelena =>
                ...
            when zluta =>
                ...
            when others =>
                ...
        end case;
    end if;
end process;

end Behavioral;

```

Obr. č. 14 v úvodní deklaraci portů entity `semafor` uvádí porty. Následuje část architektury, ve které je vytvořen vlastní datový typ `state_type` obsahující všechny 4 možné stavy semaforu, viz jeho popis v kapitole 4, a dále deklarace obou komponent, `pll` a `delicka`. Také jsou zde deklarovány všechny 3 signály, signál `state` obsahuje vnitřní stav automatu, dvojice signálů `clock_2048` a `clock_2w` slouží pro propojení komponent obou děliček frekvence.

V těle architektury je nejprve provedeno mapování komponent `pll` a `delicka`. Vlastní chování semaforu je obsaženo v jednom procesu. Ten obsahuje v citlivostním seznamu vstupy `reset` (asynchronní nulování)<sup>4</sup> a hodinový signál `clock_2w` o frekvenci 1 Hz.

V procesu je deklarována pomocná proměnná `count` pro čítání vzestupných hran hodinového signálu `clock_2w` v rozsahu 0 až 2, která slouží pro realizaci doby čekání 4 sekund (viz dále). První podmínkou je provedení asynchronního nulování, uvnitř části detekce vzestupné hrany hodinového vstupu se nachází podmínka typu `case` pro rozhodování a vyhodnocování stavu semaforu.

Její poslední částí je ošetření všech ostatních možností pomocí klíčového slova `when others`. V každém z možných stavů semaforu je pak dále provedeno přiřazení výstupů (rozsvícení LED pro daný stav pomocí výstupů `LED_cervena`, `LED_zluta`, `LED_zelena`) a provedení přechodu do následného stavu dle popisu činnosti semaforu. V případě stavů `cervena` a `zelena` jsou dále vnořeny podmínky pro vyhodnocení doby čekání (4 sekundy) a stavu vstupu `řízení`.

---

<sup>4</sup> Protože pro vstup `reset` využijeme tlačítko, které má invertované charakteristiky (viz předchozí Laboratorní úloha č. 5) a při stisknutí je jeho hodnota logická 0, je podmínka pro jeho aktivaci rovna `reset='0'`. Pozor rovněž při mapování portů frekvenční děličky `delicka_2`, kde je nutné rovněž zohlednit invertovanou hodnotu tlačítka `Reset`.