STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

Tux-Man pro FPGA

Martin přívozník
Liberecký kraj

Liberec 13.03.2020

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

Tux-Man pro FPGA

Tux-Man for FPGA

Autoři: Martin Přívozník
Škola: Střední průmyslová škola strojní a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3, příspěvková organizace
Kraj: Liberecký
Konzultant: Ing. Vladimír Prokeš, Ing. Petr Socha

Liberec 13.03.2020

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Liberci dne 13.03.2020 ……

Martin Přívozník

Poděkování

Děkuji mému učiteli Ing. Vladimíru Prokešovi za vedení mé práce a věcné připomínky, které mě vedly správným směrem. V neposlední řadě děkuji svému garantovi praktické části Ing. Petrovi Sochovi, který při práci se mnou prokázal nevídanou trpělivost a při průvodu praktickou částí maturitní práce byl schopný zachovat klid i ve chvíle, kdy jsem si myslel, že to není možné. Z jeho přednášek jsem si hodně odnesl.

Dále děkuji panu učiteli Miroslavu Machovi, jehož hodiny číslicové techniky v druhém ročníku mi základ potřebný k začátku práce.

Anotace

Tématem práce je návrh arkádové hry na RTL úrovni a její implementace na programovatelném hradlovém poli, tj. FPGA. Uživatelský vstup bude zajištěn pomocí PS/2 klávesnice a výstup prostřednictvím VGA. K implementaci je využit jazyk VHDL.

Klíčová slova

RTL, programovatelné hradlové pole, FPGA, PS/2, VGA, VHDL

Annotation

The topic of this thesis is designing an arcade game on an RTL level and its implementation on a programmable gate array, i.e. FPGA. User input is provided by PS/2 keyboard and output is shown using VGA. Language called VHDL is used for implementation.

Keywords

RTL, programmable gate array, FPGA, PS/2, VGA, VHDL

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

např.	například
angl.	anglicky
tj.	to je
zkr.	zkratka
tzv.	takzvaný, takzvaně
ASIC	Application-specific integrated circuit
FPGA	Field Programmable Gate Array
RTL	Register Transfer Level
HDL	Hardware Description Language
FSM	Finite State machine
LUT	LookUp Table
ROM	Read-Only Memory
RAM	Random-Access Memory
VGA	Video Graphics Array
VHDL	Very High Speed Integrated Circuit Hardware Description Language
ZS	Zakázaný Stav
USB	Universal Serial Bus

Úvod

Počítačové systémy jsou v dnešní době nenahraditelnou součástí našich životů. Počítače můžeme nalézt naprosto všude, ať už se jedná o domácí počítače, přenosné počítače, automatizované systémy, nebo dokonce i o jednoduché kalkulačky. Dá se již říci, že svým způsobem řídí způsob, kterým žijeme. S rozvojem oboru počítačových systémů přichází čím dál tím více prostředků pro zjednodušení práce s nimi a minimalizaci nákladů. Nezbytnou součástí ve světě počítačů jsou vestavěné systémy, neboli jednoúčelové počítače. Narozdíl od univerzálních počítačů se většinou využívají pro konkrétní úlohu. Díky tomu mohou být jednodušší, optimalizovanější a mnohem levnější, než univerzální počítače. K vytvoření takového číslicového systému je nutné rozumět číslicovému návrhu, který je poté možné implementovat ve formě ASIC (angl. Application Specific Integrated Circuit, česky zákaznický integrovaný obvod) nebo například ve formě FPGA (angl. Field Programmable Gate Array, česky programovatelné hradlové pole). ASIC je digitální specializovaný obvod a je využíván za předpokladu, že tento obvod nebude dále potřebovat změnu ve funkčnosti a bude se vyrábět v sérii. V případě implementace do FPGA se počítá s tím, že FPGA je možné přeprogramovat a je vybráno v případě, kdy obvod bude vyžadovat změny v budoucnu, nebo již nebude třeba po vykonání malého počtu úloh. Především je také výrazně levnější než ASIC.

Příkladem využití počítačů jsou arkádové hry, které jsou zábavou pro celý svět již téměř od doby vynalezení počítačů. Jednou z těchto her je Pac-Man, který je známou hrou již od roku 1980. Pac-Man se stal symbolem pro mnoho dalších aplikací, jako jsou i televizní seriály a populární písně. Dodnes se jedná o velice populární hru.

Má práce se bude zabývat návrhem a implementací číslicového logického obvodu realizujícího klona Pac-Mana. Pro implementaci bude využita forma FPGA. Cílem práce je procvičení číslicového návrhu a realizace v programovatelném hradlovém poli. Značná část práce se tedy bude skládat z teorie potřebné k návrhu a implementaci.

V kapitole 1 se budu věnovat teorii potřebné pro návrh a implementaci číslicového logického obvodu pro FPGA. V kapitole 2 rozeberu postup návrhu číslicového obvodu na základě teoretických znalostí z předchozí kapitoly. Teorii využiji také k implementaci obvodu na vybrané FPGA v kapitole 3. Na konci v kapitole 4 rozeberu postup testování práce.

Kapitola 1

Analýza

Tato kapitola obsahuje stručný souhrn znalostí a informací potřebných pro následný návrh a implementaci. V sekci 1.1 je vysvětlen číslicový obvod a postup jeho návrhu. V sekci 1.2 stručně vysvětluji programovatelné hradlové pole a dále vybranou vývojovou desku. Sekce 1.3 se zabývá použitými komunikačními rozhraními, které zajišťují uživatelský vstup a výstup. Na závěr kapitoly, v sekci 1.4 vysvětluji princip a funkčnost hry, jenž je vzorem pro můj návrh.

1.1 Číslicový návrh

V této sekci se věnuji tomu, co je číslicový obvod a jak jej navrhnout jak ve schématu, tak v jazyce popisujícím hardware. V podsekci 1.1.1 rozeberu logické funkce, prostředky jejich popisu a realizace pomocí logických hradel. Podsekce 1.1.2 je zaměřená na návrh sekvenčních obvodů a synchronních sekvenčních automatů (FSM), na což naváže podsekce 1.1.3, ve které vysvěluji princip hodinových domén a plně sekvenčního návrhu. V podsekci 1.1.4 stručně ukážu, jak převést schéma číslicového obvodu do kódu v jazyce popisujícím hardware (Hardware Description Language, HDL), v mém případě do jazyka Very High Speed Integrated Circuit Hardware Description Language (VHDL).

1.1.1 Kombinační obvody

Booleovská funkce

Booleovská funkce je funkce N vstupů a M výstupů nad množinou $\{0, 1\}$. V případě, kdy má funkce více jak jeden výstup, lze ji rozdělit na M funkcí s jedním výstupem. Uvážíme-li Booleovu algebru, platí pro operace sčítání a násobení pravidla uvedená v tabulce 1.1. Operace se dvěma vstupními hodnotami nazýváme binární operace. Některé binární operace, přestože často používají stejná značení + a * jako v algebře reálných čísel, mají v Booleově algebře stejnou prioritu a jiný význam (žádná operace nemá přednost) [2]. Pro logický součet a logický součin platí základní pravidla v tabulce 1.2. Příkladem re-

$a + b = b + a$	$a * b = b * a$	(komutativita)
$a + (b + c) = (a + b) + c$	$a * (b * c) = (a * b) * c$	(asociativita)
$a + (b * c) = (a + b) * (a + c)$	$a * (b + c) = (a * b) + (a * c)$	(distributivita)
$a + 0 = a$	$a * 1 = a$	(neutralita 0 a 1)
$a + \overline{a} = 1$	$a * \overline{a} = 0$	(vlastnosti negace)

Tabulka 1.1: Axiomy a vztahy Booleovy algebry.[1]

de Morgan	$\overline{(a + b)} = \overline{a} * \overline{b}$	$\overline{(a * b)} = \overline{a} + \overline{b}$
idempotence	$a + a = a$	$a * a = a$

Tabulka 1.2: Základní pravidla Booleovy algebry.[2]

prezentace Booleovské funkce je pravdivostní tabulka 1.3, kde in_1 a in_2 jsou vstupní hodnoty a *out* je výstupní hodnota. Pravdivostní tabulka obsahuje vždy N^2 řádků, aby reprezentovala výstupní hodnotu pro všechny možné kombinace vstupních hodnot. Další možností je Booleovská formule .[2]. K vyjádření formule a k popisu booleovské funkce používáme nejčastěji základní funkce uvedené v tabulce 1.4

Kombinační obvod

Kombinační logický obvod, je takový obvod, ve kterém jsou výstupní hodnoty dány pouze aktuální kombinací vstupních proměnných. Neobsahuje žádnou paměť předchozích stavů. Jedinou vyjímkou je krátký časový interval, za který logický člen (AND, NAND, OR, NOR ...) vyhodnotí výstup na základě vstupních hodnot. Tento časový interval může být zanedbatelný v případě krátkých datových cest. V případě dlouhé datové cesty může být potřeba na tento časový interval brát zřetel a zvážit optimálnější řešení.

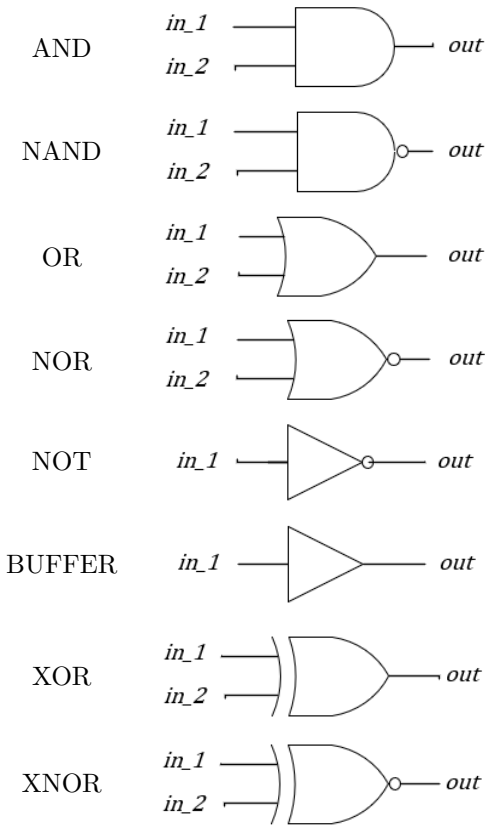
Platí, že u číslicových obvodů každá proměnná v operaci nabývá hodnotu jednoho tzv. bitu. Bit je základní jednotkou dat a může nabývat hodnot 0, nebo 1. Reprezentací číslicového obvodu je schéma číslicového obvodu, kde každá z funkcí je reprezentována tzv. schématickou značkou. Schématické značky mohou být různé, dokud z nich jasně vyplívá, jakou funkci zastupují. Schématické značky jsou propojené signály, které představují jednotlivé bity. Pro minimalizaci je možné několik signálů (bitů) zakreslit jediným konektorem, pokud je označený počtem bitů, které reprezentuje. Nejčastěji používané normy značení

in_1	in_2	*out*
0	0	$f(0,0)$
0	1	$f(0,1)$
1	0	$f(1,0)$
1	1	$f(1,1)$

Tabulka 1.3: Pravdivostní tabulka.

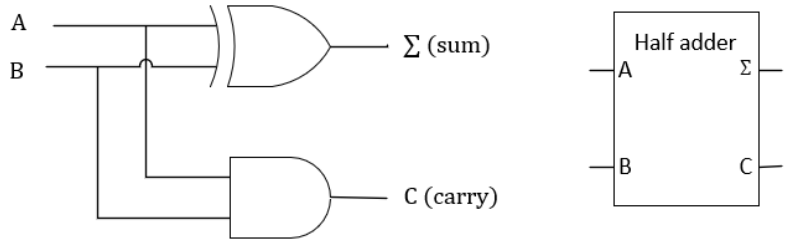
Název	Pravdivostní tabulka	Formule
AND (logický součin)	in_1 in_2 out 0 0 0 0 1 0 1 0 0 1 1 1	$out = in_1 * in_2$
NAND (negovaný logický součin)	in_1 in_2 out 0 0 1 0 1 1 1 0 1 1 1 0	$out = \overline{in_1 * in_2}$
OR (logický součet)	in_1 in_2 out 0 0 0 0 1 1 1 0 1 1 1 1	$out = in_1 + in_2$
NOR (negovaný logický součet)	in_1 in_2 out 0 0 1 0 1 0 1 0 0 1 1 0	$out = \overline{in_1 + in_2}$
NOT (logická negace)	in_1 out 0 1 1 0	$out = \overline{in_1}$
BUFFER (opakovač)	in_1 out 0 0 1 1	$out = in_1$
XOR (nonekvivalence)	in_1 in_2 out 0 0 0 0 1 1 1 0 1 1 1 0	$out = in_1 \oplus in_2$
XNOR (ekvivalence)	in_1 in_2 out 0 0 1 0 1 0 1 0 0 1 1 1	$out = \overline{in_1 \oplus in_2}$

Tabulka 1.4: Tabulka nejpoužívanějších základních logických funkcí.



Tabulka 1.5: Schématické značky některých nejpoužívanějších základních logických funkcí (americká norma ANSI).

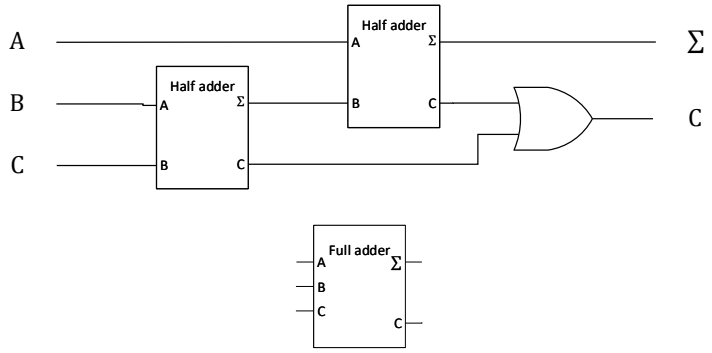
jsou evropská a americká. Příklady schématických značek pro nejpoužívanější logické funkce jsou uvedené v tabulce 1.5. Pro usnadnění práce můžeme využívat logických bloků, které plní danou funkci. Opět platí, že ze značení logických bloků ve schématu musí plně vyplívat, jakou funkci zastupují. Logický blok, který má definovanou funkci může být použit schématu. Při návrhu číslicových obvodů využíváme hierarchie, kde jsou pro každý logický blok popsány vstupy i výstupy a v případě, kdy se nejedná o základní logické bloky, tak je popsána i funkce (formule, pravdivostní tabulka, nebo schéma bloku) a vhodně přiřazena ke schématu. Jedním ze základních logických bloků je poloviční sčítačka, viz obrázek 1.1. Poloviční sčítačka umí sečíst dvě jednobitová čísla a vygenerovat bit do vyššího řádu (carry) podle pravdivostní tabulky 1.6. Zřetězením dvou polovičních sčítaček a přenesením pomocí funkce OR získáme poté kompletní sčítačku, viz obrázek 1.2.[2] Kompletní sčítačka umí sečíst jednobitová čísla, vygenerovat bit do vyššího řádu a přijmout bit z nižšího, sčítá tedy tři bity. Sčítá počet jedniček na vstupech.



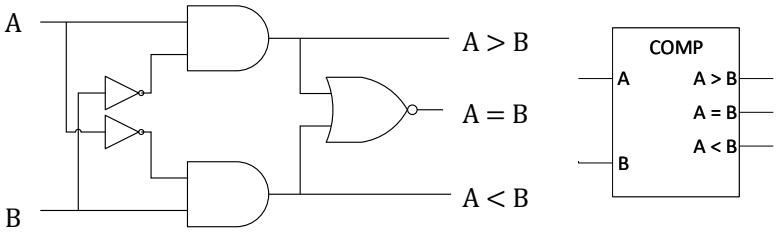
Obrázek 1.1: Schéma poloviční sčítačky a příklad jejího značení ve schématu.

A	B	C	$\sum$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

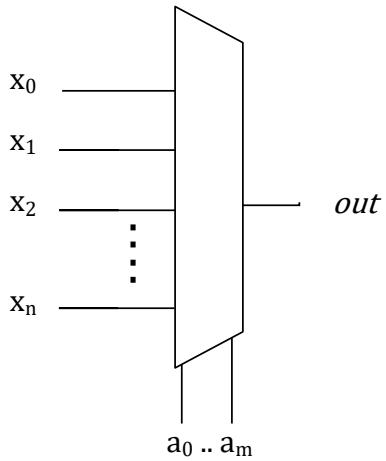
Tabulka 1.6: Pravdivostní tabulka poloviční sčítačky.



Obrázek 1.2: Schéma kompletní sčítačky a příklad jejího značení ve schématu.



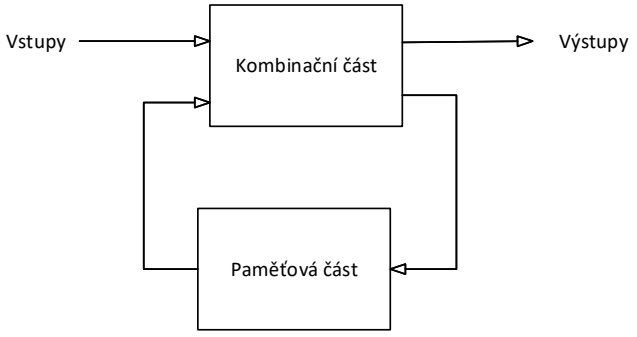
Obrázek 1.3: Schéma komparátoru a příklad jeho značení ve schématu.



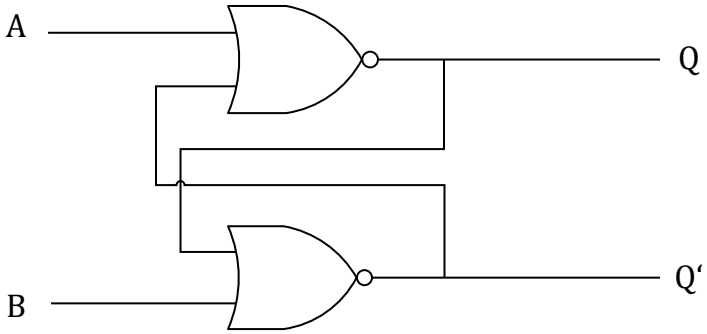
Obrázek 1.4: Schématická značka multiplexoru.

Pro porovnávání dvou hodnot a vyhodnocení jejich nerovnosti používáme tzv. komparátor, viz obrázek 1.3. Dalším důležitým základním logickým blokem je multiplexor, viz obrázek 1.4. Multiplexor na základě řídícího vstupu/řídících vstupů (a) přivádí na výstup (out) jeden ze vstupních signálů (x).

Číslicový obvod lze realizovat například z integrovaných obvodů, v nichž bývají hradla realizována z několika tranzistorů. Logické hodnoty představuje napětí přivedené na obvod. Logická 1 bývá reprezentována napětím kladným a logická 0 napětím nulovým.



Obrázek 1.5: Huffmanův model sekvenčního obvodu.



Obrázek 1.6: Schéma RS klopného obvodu s využitím logických hradel NOR.

1.1.2 Sekvenční obvody

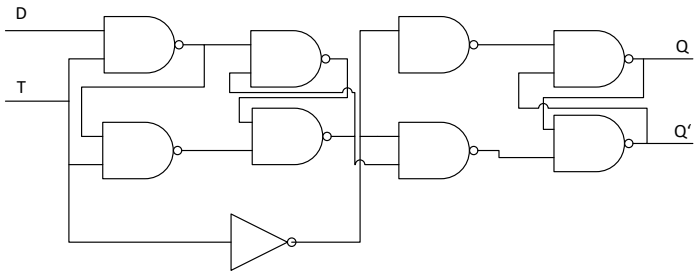
Sekvenční logický obvod

Sekvenční logický obvod je takový obvod, ve kterém výstupní hodnoty nejsou dány pouze aktuální kombinací vstupních proměnných, ale zároveň jeho vnitřním stavem. Sekvenční obvod si tedy musí zapamatovat předchozí hodnoty pomocí paměti, která bývá realizována pomocí zpětné vazby [2]. Sekvenční obvod se dělí na dvě části - kombinační a paměťovou, kde paměťová část je tvořena logickým obvodem, ve kterém bývá zavedena zpětná vazba a kombinační část bývá tvořena kombinačním obvodem. Obecné schéma sekvenčního obvodu je na obrázku 1.5.

Příkladem logického obvodu se zpětnou vazbou může být tzv. RS klopný obvod, viz obrázek 1.6. RS klopný obvod překlápí mezi dvěma mezními hodnotami. Pravdivostní tabulka RS klopného obvodu se může dělit na tři části - zakázaný stav (ZS), překlápěcí část a paměť, viz tabulka 1.7. Pokud se tedy

A	B	Q	Q'
0	0	Z.S.	
0	1	0	1
1	0	1	0
1	1	Paměť	

Tabulka 1.7: Pravdivostní tabulka RS klopného obvodu s využitím NOR.



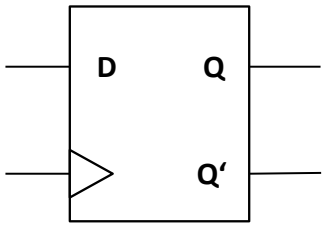
Obrázek 1.7: Schéma D klopného obvodu.

změní vstupní hodnoty z $[0, 1]$, nebo $[1, 0]$ na $[1, 1]$, na výstupu bude předchozí hodnota, dokud neproběhne další změna vstupních hodnot. Jedná se tedy o tzv. hladinový klopný obvod (angl. latch). Častěji využívaným klopným obvodem je tzv. D klopný obvod, viz obrázek 1.7. Výhodou D klopného obvodu je fakt, že narozdíl od RS klopného obvodu nemá zakázaný stav, viz pravdivostní tabulka 1.8. Signál T u D klopného obvodu se dá považovat za tzv. povolovací signál. Při náběžné hraně na T (změna stavu z logické 0 na logickou 1) se na výstup Q zapíše aktuální hodnota na signálu D. V tomto případě se tedy jedná o hranový klopný obvod. Příklad značení D klopného obvodu je na obrázku 1.8. Vstupní signál označený trojúhelníkem je signál T. Trojúhelník značí, že obvod mění výstupní hodnotu v reakci na náběžnou hranu. Tento vstup se jinak nazývá hodinovým vstupem. Pokud má číslicový obvod takový vstup jedná se o sekvenční obvod. V případě hladinového obvodu by ve značení mohl být místo trojúhelníku čtverec.

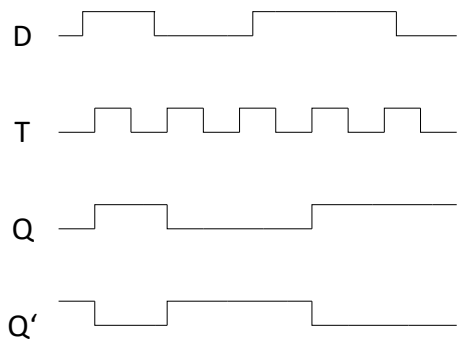
Hodnoty na signálech se dají popsat tzv. časovým diagramem. Časový diagram popisuje, jaké logické hodnoty nabývá daný signál v jaký čas. Příklad časového diagramu je na obrázku 1.9.

T	D	Q	Q'
⌐¬	x	x	$\overline{x}$
Jinak		Paměť	

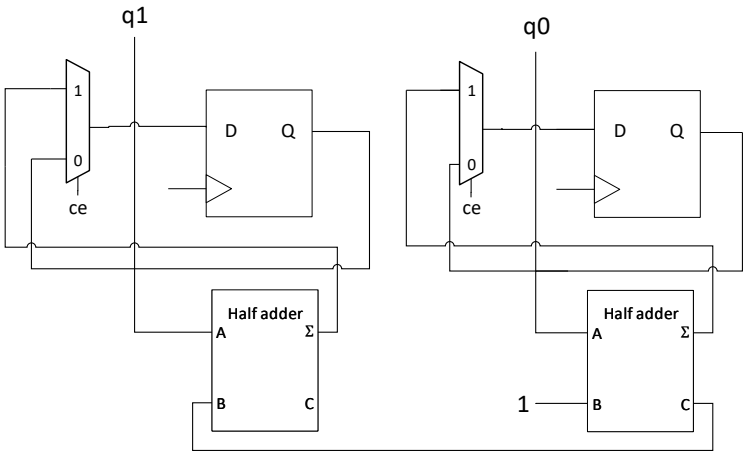
Tabulka 1.8: Pravdivostní tabulka D klopného obvodu.



Obrázek 1.8: Příklad značení D klopného obvodu ve schématu.



Obrázek 1.9: Časový diagram D klopného obvodu.

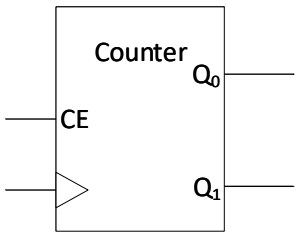


Obrázek 1.10: Příklad schématu dvoubitového čítače reagujícího na náběžnou hranu.

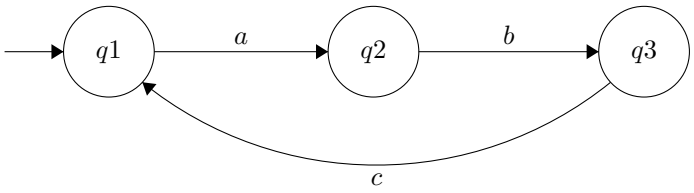
Příkladem sekvenčního logického bloku je čítač. Čítač je takový sekvenční logický obvod, který uchovává v ,,paměti"informaci o tom, kolikrát zaznamenal změnu stavu (dle návrhu náběžnou, nebo sestupnou hrana) na hodinovém signálu. Příklad schématu čítače je na obrázku 1.10. Může být ve schématu značen, jako na obrázku 1.11.

Konečný stavový automat

Konečný stavový automat (angl. Finite State Machine - FSM) M je šestice M=(Q, T, D, δ, λ, q_0), kde [3]:



Obrázek 1.11: Příklad schématické značky dvoubitového čítače reagujícího na náběžnou hranu.



Obrázek 1.12: Příklad stavového diagramu konečného stavového automatu.

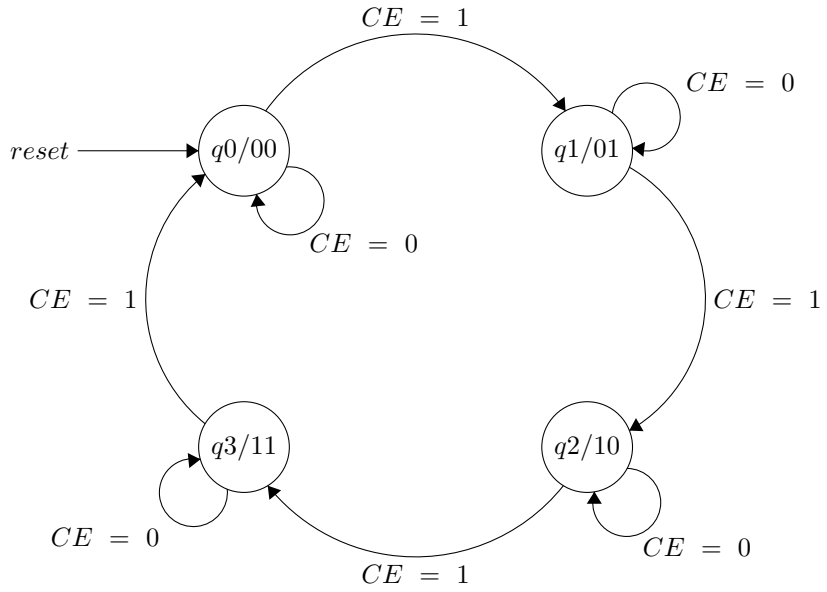
- Q je konečná množina vnitřních stavů
- T je konečná množina vstupních symbolů (signálů)
- D je konečná množina výstupních symbolů (signálů)
- δ je zobrazení z Q x T do Q (stav x vstup do stav) nazývané přechodová funkce
- λ je zobrazení z Q x T do D (Mealy, ze stavu a vstupu do výstupu) nebo Q do D (Moore, jen ze stavu do výstupu)
- q_0 je počáteční stav

Má definované stavy, mezi kterými za daných podmínek přepíná a mění svůj výstup. FSM provádí vždy právě jeden přechod s každou náběžnou hranou hodin. Musí popisovat přechodovou funkci (podmínky pro změnu stavu a do kterého), výstupní funkci (jaký signál v danou chvíli udává na výstup) a dále musí mít definovaný počátečný stav. Příklad stavového diagramu FSM je na obrázku 1.12, kde a, b, c jsou podmínky pro změnu stavu a $q1, q2, q3$ jsou stavy FSM. Počáteční stav je značen šipkou, která nevychází z žádného stavu, v tomto případě $q1$. Rozeznáváme dva typy FSM na základě definičního oboru výstupní funkce [2, 3]. Prvním typem je typ Mealy (angl. často input-based), který mění svůj výstup na základě aktuálního stavu a změn na vstupních signálech. Druhý je typ Moore (angl. často state-based), jehož výstupní funkce je závislá pouze na aktuálním stavu (každý stav má definovaný výstup). Příkladem stavového diagramu čítače modulo 4 s povolujícím vstupem je obrázek 1.13

1.1.3 Synchronní návrh

Pravidla návrhu synchronního obvodu

Synchronní obvod je takový obvod, ve kterém všechny sekvenční části obvodu mají na svůj hodinový vstup přivedený stejný signál, aby měnily svůj stav ve stejný čas. Tento signál můžeme nazývat společné hodiny. Společné hodiny mění svůj stav na dané frekvenci (nejčastěji dle krystalu v obvodu). Část obvodu se společným hodinovým signálem nazýváme hodinovou doménou [4]. Pro návrh synchronního obvodu platí několik pravidel, které je nutné dodržet

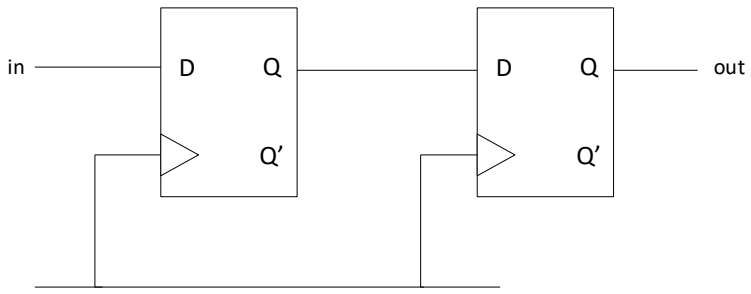


Obrázek 1.13: Příklad stavového diagramu (typ Moore) čítače modulo 4 se vstupem count enable.

pro zajištění správné funkčnosti. Společné hodiny nesmí být hradlované (signál musí být přiveden přímo k hodinovým vstupům sekvenčních obvodů). V případě hradlování společných hodin může docházet k zákmitům a časovým rezervám v jednotlivých částech obvodu. Dále je nutné, aby obvod obsahoval synchronní reset. Po synchronním resetu obvodu změní stav celý obvod společně a nedojde k časovým nesrovnalostem [4].

Při návrhu se nemusí vždy pracovat pouze s jednou hodinovou doménou. Univerzální řešení přechodu mezi hodinovými doménami je dvoubranové FIFO se dvěma hodinovými vstupy.Pro zamezení nestabilit můžeme využít tzv. synchronizér. Synchronizér se skládá ze dvou D klopných obvodů zapojených v sérii, které zajistí podmínky pro korektní funkci ostatních klopných obvodů v cílové časové doméně. Vstupní hodnota D klopného obvodu nesmí být změněna krátce před příchodem náběžné hrany na hodinovém vstupu (tj. předstih). Zároveň nesmí být změněna těsně po příchodu náběžné hrany na hodinovém vstupu (tj. přesah) [4]. V případě porušení těchto podmínek se může stát, že klopný obvod bude mít abnormálně zpožděnou, nebo kolísavou odezvu. Příklad synchronizéru je na obrázku 1.14.

Datové signály jsou řízené řídícími signály. Protokol řídících signálů musí být navržen tak, aby se měnil vždy jen jeden. Příkladem řídících signálů mohou být signály "strobe"nebo "ack". Úkolem tohoto protokolu je indikace, zda jsou data aktuální. V případě tzv. jednosměrného handshake budou data aktuální po dobu, kdy signál "strobe"nabývá logické $'1'$. U tzv. oboucestného handshake



Obrázek 1.14: Příklad návrhu synchronizéru.

jsou data platná a signál strobe bude nabývat logické ′1′, dokud nepřijde od protistrany signál ACK.

1.1.4 Jazyk VHDL

Charakteristika jazyka VHDL

VHDL (VHSIC (Very-High-Speed Integrated Circuit, česky velmi rychlý integrovaný obvod) Hardware Description Language) je programovací jazyk sloužící pro popis hardware (tj. Hardware Description Language, zkr. HDL, česky jazyk popisující hardware). Používá se pro simulaci a návrh digitálních integrovaných obvodů, např. pro FPGA (Field of Programmable Gate Array, česky programovatelná hradlová pole). VHDL je silně typovaný jazyk a umožňuje popsat obvod na hradlové, RTL (Register Transfer Level, česky přenos na úrovni registrů) i algoritmické úrovni [5].

Datové typy

VHDL je silně typovaný jazyk, při operacích je tedy nutné, aby se typy shodovaly. Jedny z důležitých základních datových typů jsou: bit, integer, std_logic a std_logic_vector.

- Array - pole prvků jiného typu
- Bit - Výčtový typ (’0’, ’1’)
- Integer - Celé číslo
- Std_logic - Popis signálu (’U’,’X’, ’0’, ‘1’, ’Z’, ’W’, ’L’, ’H’, ’-’)
- Std_logic_vector - Pole std_logic
- Signed - Bitový vektor, který nepopisuje stav signálu, ale pouze číselnou hodnotu

- Unsigned - Signed, který může obsahovat pouze kladné hodnoty

Základní konstrukce

Konstrukce jazyka VHDL se dělí na několik částí [5]. Jednou z nich je tzv. entita. Entita popisuje pouze rozhraní, nikoliv chování nebo vnitřní strukturu návrhu. Obsahuje tedy definici vstupních a výstupních portů. Příklad návrhu entity multiplexoru se dvěma datovými vstupy A, B, jedním řídícím vstupem SEL a výstupem Y:

```
entity MULTIPLEXER is
port(   A, B, SEL       : in bit;
        Y               : out bit
        );
end MULTIPLEXER;
```

Další částí je architektura. Architektura obsahuje chování a vnitřní strukturu entity. Architektura tedy musí být definována uvnitř entity a nelze, aby fungovala samostatně. Entita může obsahovat více architektur. Ta se poté dělí na dvě části. Deklarační a sekci paralelních příkazů. Deklarační část se nachází před klíčovým slovem begin a je vyhrazena pro deklaraci signálů, konstant, nebo typů. Součástí sekce paralelních příkazů může být instancování komponent, behaviorální popis (tj. popis chování), nebo tzv. procesy. Sekce paralelních příkazů se nachází v uvnitř architektury (tj. část za klíčovým slovem begin). Příklad architektury výše uvedeného multiplexoru:

```
architecture MUX_BODY of MULTIPLEXER is

signal SELNON, ASEL, BSEL : std_logic;

begin

SELNON <= not SEL;
ASEL <= A and SELNON;
BSEL <= B and SEL;
Y <= ASEL or BSEL;

end MUX_BODY;
```

Proměnná označená klíčovým slovem signal značí stejnojmenný signál v obvodu. Klíčové slovo *std_logic* říká, že se jedná o jednobitový signál. V případě vícebitového signálu by bylo využito klíčové slovo *std_logic_vector*. Signál si pamatuje svou hodnotu, dokud není změněna dalším přiřazením. V části paralelních příkazů operátor <= zastupuje operátor přiřazení. Hodnota z pravé strany je tedy po vyhodnocení logické operace přiřazena signálu na levé straně. Všechny logické operace v tomto případě jsou vyhodnocovány paralelně. Tato metoda se většinou využívá pro popis chování kombinačních obvodů. V případě popisu sekvenčního obvodu je nejčastěji využíván tzv. proces. Proces je součástí

architektury a přestože popisuje sekvenční obvod, tak běží souběžně. V sekvenčním popisu je možné krom operátorů přiřazení a logických operací využívat také podmínky, jako je např. *if*, nebo *case*. V případě psaní syntetizovatelného kódu proces obsahuje citlivostní seznam a opět popis chování. Citlivostní seznam je seznam signálů, na který sekvenční obvod reaguje (u sekvenčních obvodů hodinový vstup). Příklad procesu, který popisuje D klopný obvod, kde D je vstupní hodnota, Q je výstupní, *clk* jsou hodiny a *reset* je synchronní reset:

```
D_FLIP_FLOP : process (clk)
begin
        if(clk'event and clk='1') then
                if(reset = '1') then
                        Q <= '0';
                else
                        Q <= D;
                end if;
        end if;
end process;
```

Pokud na hodinách obvod zaznamená náběžnou hranu a není aktivní synchronní reset, tak na signál Q zapíše aktuální hodnotu na signálu D.

Celý obvod je možné popsat v rámci jedné entity, pro přehlednost je možné ale využívat komponenty. Při návrhu většího obvodu je vhodné logicky oddělit části obvodů od sebe a vytvořit z nich entity, které je možné poté propojit v nadřazené entitě pomocí klíčového slova component a namapování vstupních a výstupních signálů jednotlivých entit pomocí příkazu port map. Při mapování je poté možné pouze přiřadit vstupní a výstupní signály komponenty k signálům v nadřazené entitě [5].

Popis FSM

Jazyk VHDL nezná pojem automatu a existuje mnoho způsobů, jak jej popsat ve VHDL. Často používaný popis a zároveň takový, který usnadní práci logické syntéze programu je metoda se třemi procesy [6]. Prvním procesem je přechodová funkce a jedná se o kombinační proces, který na základě vstupních hodnot vyhodnocuje, jaký bude příští stav FSM. Vstupy přechodové funkce jsou stavy a vstupy FSM a jejím výstupem je příští stav. Druhý proces je registr stavu, který je sekvenčním procesem. Jeho vstupem je příští stav a výstupem je nový aktuální stav. Příklad registru stavu, kde initial_state je počáteční stav automatu, CURRENT_STATE je aktuální stav a NEXT_STATE je následující stav:

```
CLKP : process (clk)
begin
        if (clk'event and clk ='1') then
                if (reset = '1') then
                        CURRENT_STATE <= initial_state;
                else
```

```
                    CURRENT_STATE <= NEXT_STATE;
                end if;
        end if;
end process;
```

Poslední proces je kombinační proces zvaný výstupní funkce, který na základě aktuálního stavu a v případě návrhu Mealyho FSM i vstupních hodnot vyhodnocuje výstup FSM. Výstupní funkce má jako vstupy stav a dle návrhu i vstup FSM. Výstupem je výstup FSM [6]. Příklad přechodové a výstupní funkce FSM s funkcí čítače modulo 4, kde T_STATE je výčet stavů čítače, TRANSP je přechodová funkce a OUTP je výstupní funkce (Mealyho návrh):

```
type T_STATE is (Q1, Q2, Q3, Q4);
signal CURRENT_STATE, NEXT_STATE : T_STATE;
begin
TRANSP : process (CURRENT_STATE, ce)
begin
        case CURRENT_STATE is
                when Q1 => if ce='1' then
                                NEXT_STATE <= Q2;
                           else
                                NEXT_STATE <= Q1;
                           end if;
                when Q2 => if ce='1' then
                                NEXT_STATE <= Q3;
                           else
                                NEXT_STATE <= Q2;
                           end if;
                when Q3 => if ce='1' then
                                NEXT_STATE <= Q4;
                           else
                                NEXT_STATE <= Q3;
                           end if;
                when Q4 => if ce='1' then
                                NEXT_STATE <= Q1;
                           else
                                NEXT_STATE <= Q4;
                           end if;

        end case;
end process;

OUTP : process (CURRENT_STATE, ce)
begin
        case CURRENT_STATE is
                when Q1 => if ce='1' then
                                q <= "01";
```

```
                    else
                        q <="00";
                end if;
            when Q2 => if ce='1' then
                        q <= "10";
                    else
                        q <="01";
                end if;
            when Q3 => if ce='1' then
                        q <= "11";
                    else
                        q <="10";
                end if;
            when Q4 => if ce='1' then
                        q <= "00";
                    else
                        q <="11";
                end if;
        end case;
end process;
```

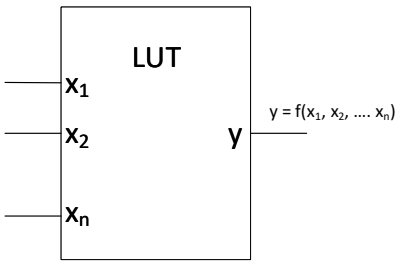
1.2 Programovatelná hradlová pole FPGA

Tato sekce se zabývá tím, co jsou programovatelná hradlová pole (Field of programmable gate array, FPGA) a jak probíhá práce s těmito hradlovými poli na vývojové desce. V podsekci 1.2.1 popisuji části FPGA, které jsou použity pro implementaci číslicového obvodu. Podsekce 1.2.2 stručně vysvětluje kroky nezbytné pro implementaci samotného číslicového obvodu na základě jeho popisu VHDL kódem. V podsekcích 1.2.3 a 1.2.4 je stručně popsána použitá vývojová deska a vývojové prostředí, které se váže k FPGA, které deska obsahuje.

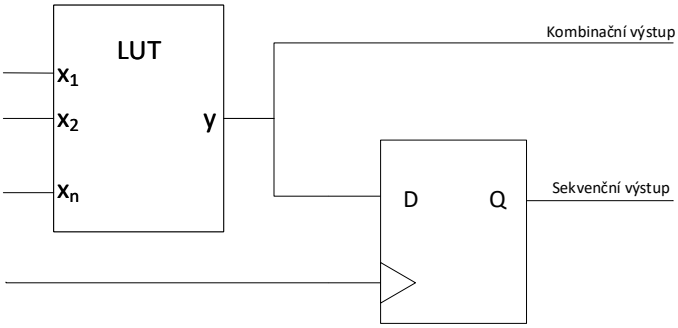
1.2.1 Dostupné prostředky

Základní stavební prvky

Technologie FPGA využívá tří základních stavebních prvků. Jedním z nich je propojení. Propojení tvoří síť vodičů, které jsou propojeny MOS tranzistory nebo tzv. antipojistkami (nevodivý prvek, který je možné vnějším působením napětí prorazit a poté přejde do vodivého stavu). Dle nastavení buňky v konfigurační paměti jsou vodiče pomocí tranzistorů spojeny. Stejně jako u logických hradel pro tyto spínací prvky platí, že jejich odpor působí přídavné zpoždění. Při konfiguraci je tedy potřeba, aby FPGA vytvořilo specializovaný rozvod hodinového signálu, který zajistí, že signály budou správně načasovány [7]. Hodinový signál mívá stromovou topologii k zajištění co největší časové přesnosti v celém obvodu.



Obrázek 1.15: Příklad zakreslení LUT.



Obrázek 1.16: Příklad základního bloku FPGA.

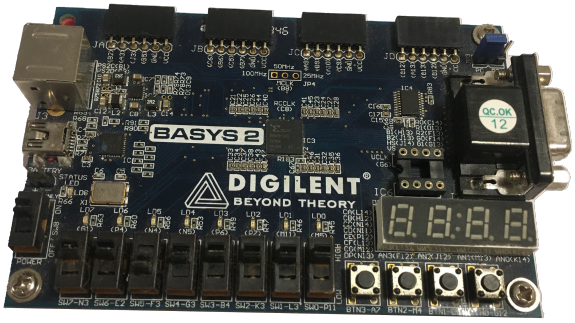
Druhým základním stavebním prvkem jsou tzv. Look-up Tables (zkr. LUT). Pomocí programového propojení lze nakonfigurovat síť kombinačních prvků. FPGA využije části konfigurační paměti jako pravdivostní tabulky k implementaci logické funkce. LUT může být zakreslen jako na obrázku 1.15.

Posledním stavebním prvkem jsou registry. Typicky ke každé logické funkci přísluší jeden registr. V FPGA převládají hranové D klopné obvody, které mají typicky vstup pro povolení hodin [7]. Základní blok může vypadat jako na obrázku 1.16.

1.2.2 Logická syntéza

Kroky logické syntézy

Logická syntéza je proces, při kterém je popis hardware převeden do schématu, který užívá konkrétní dostupné prostředky daného FPGA. Proto se dá říct, že k dosažení nejlepších výsledků je potřeba navrhovat obvod s ohledem na cílovou technologii. Logická syntéza se skládá z několika kroků. Mezi důležité kroky patří syntéza, mapování a ,,place and route" [8]. Úkolem syntézy je rozpoznání cha-



Obrázek 1.17: Digilent Basys 2.

rakteristických bloků použitých v popisu hardware a vytvoření tzv. netlistu (tj. schéma z logických hradel). Nástroj musí umět rozpoznat aritmetické operátory, paměti, čítače, FSM, multiplexory a další bloky. V rámci syntézy probíhá také proces předoptimalizace, který např. vymaže nevyužité bloky v popisu a vysokoúrovňové datové typy, jako je například integer (označení celočíselného 32-bitového datového typu) přeloží do bitové reprezentace. Mapování řeší realizaci obvodu pomocí dostupných bloků FPGA. Jedná se o překlad z netlistu na dostupné stavební bloky. Úkolem place and route je napojit přeložené stavební bloky na ty, které obsahuje konkrétní čip a rozvést spoje v čipu tak, aby správně fungovalo např. časování obvodu. Po provedení těchto kroků je vytvořen tzv. bitstream, který je možné nahrát do paměti konkrétního FPGA.

1.2.3 Vývojová deska Digilent Basys 2

Specifikace

Vývojová deska Digilent Basys 2 [9] viz obrázek 1.17. je platforma pro návrh a implementaci logických obvodů. Deska obsahuje Spartan-3E FPGA od firmy Xilinx. Může pracovat s vestavěným oscilátorem, který lze nastavit na frekvenci 25, 50, nebo $100MHz$, nebo má připravený socket pro druhý externí oscilátor. Využívá XCF02 Flash ROM (Read Only Memory), ve které ukládá FPGA konfikurace na dobu neurčitou. Pracuje také s tzv. Pmods (analogové a digitální I/O (vstupní i výstupní) moduly, které dovolují například převod z analogového signálu na digitální a naopak). Má tři vestavěné stabilizátory napětí ($1.2V$, $2.5V$, $3,3V$), které dovolují využívat $3.5V - 5.5V$ externí napájecí zařízení [9]. Typicky

využívá napájení z USB, přes který může mít FPGA také naprogramováno, ale deska obsahuje také konektor pro baterii. Vstupní napájení je vedeno přes tzv. power switch. Deska musí být nakonfigurována pro vykonávání dané činosti. K vygenerování bitstreamu pro Spartan-3E FPGA je nutné použít vývojové prostřední od firmy Xilinx. Software od firmy Digilent zvaný Adept poté může být použit ke konfiguraci FPGA přes USB [9].

Dostupné prostředky

Vývojová deska obsahuje čtyři tlačítka, osm přepínačů, které lze využít jako vstupy obvodu. Jako výstup lze využít osmi LED diod, nebo sedmisegmentového displeje. Sedmisegmentový displej na desce obsahuje čtyři číslice, které jsou připojeny na společné anodě (tzn. jsou aktivní při logické $'0'$). Krom těchto prvků obsahuje deska také PS/2 a VGA konektor (tříbitvé D/A převodníky pro červenou a zelenou barvu a dvoubitový převodník pro modrou) a konektory pro přídavné moduly.

1.2.4 Vývojové prostředí Xilinx ISE

Uživatelské rozhraní

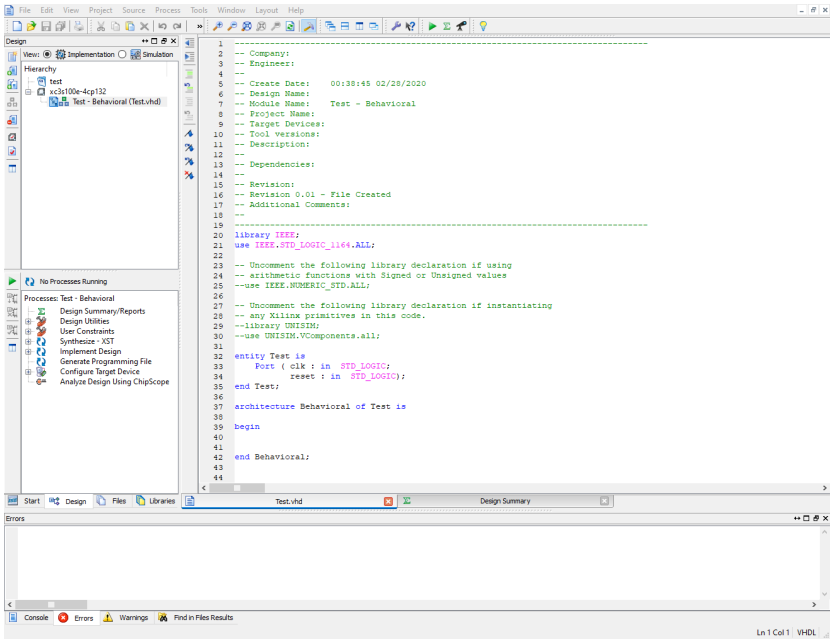
Xilinx ISE je softwarový nástroj od firmy Xilinx pro syntézu a analýzu HDL návrhů [10]. Umožňuje syntézu a simulaci návrhu, vytváření návrhů v jazycích VHDL nebo Verilog a konfiguraci FPGA (FPGA od firmy Xilinx, nikoliv od jiné). Primárním rozhraním je Project Navigator, ve kterém lze editovat zdrojový kód, sledovat výstupní konzoli a probíhající kroky syntézy viz obrázek 1.18. Okno hierarchie zobrazuje návrhové soubory (moduly), jejichž závislosti mají stromovou strukturu. Okno procesů popisuje operace, které bude ISE provádět na aktivním modulu (často syntéza, mapování, ...). Pro simulaci návrhu lze využít ISIM nebo ModelSim logického simulátoru, jejichž programy musí být také psané v HDL. Tyto simulátory zvládají primárně tyto typy simulací: Logické ověření (kontrola výstupů modulu), ověření časování a simulace post-place and route (ověření chování po naprogramování FPGA) [10].

1.3 Rozhraní

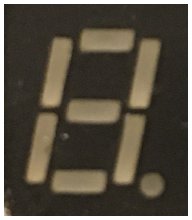
Tato sekce popisuje komunikační rozhraní použitá v návrhu a implementaci, která zajišťují uživatelský vstup a výstup. V podsekci 1.3.1 vysvětluji rozhraní 7-segmentového displeje. Sekce 1.3.2 a 1.3.3 popisují protokoly PS/2 a VGA.

1.3.1 7-segmentový displej

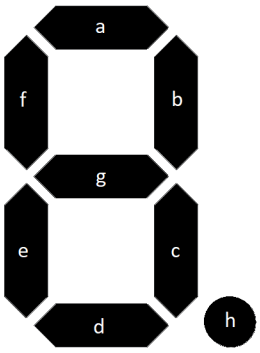
Sedmisegmentový displej se skládá ze sedmi uspořádaných LED diod viz obrázek 1.19. Většina sedmisegmentových displejů obsahuje také osmou LED diodu, kterou využívá jako tečku. Jednotlivé segmenty je možné rozsvítit přivedením daného signálu (při zapojení na společnou katodu logickou 1 a při zapojení na společnou



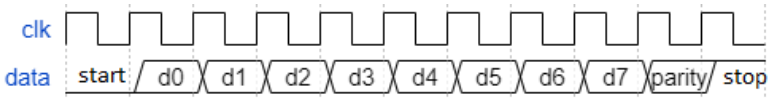
Obrázek 1.18: Základní pohled na Xilinx ISE Project Navigator.



Obrázek 1.19: Sedmisegmentový displej.



Obrázek 1.20: Příklad zapojení segmentů na sedmisegmentovém displeji.



Obrázek 1.21: Časový diagram PS/2 přenosu.

anodu na logickou 0). Častý způsob zapojení segmentů je na obrázku 1.20. Kombinacemi rozvícených segmentů lze vytvořit obrazy až 128 znaků.

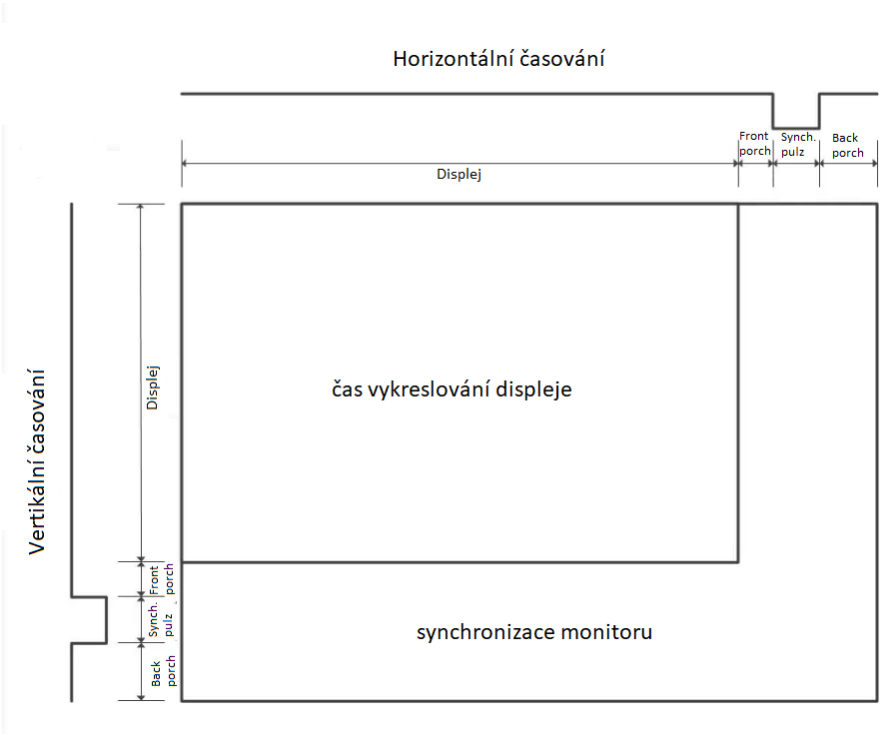
1.3.2 PS/2

Charakteristika

Rozhraní PS/2 využívá dvoudrátové sériové komunikace (data a hodiny) většinou pro ovládání myši, nebo klávesnice. Při přenosu dat využívá jedenáctibitové slovo, které obsahuje tzv. start bit, stop bit, paritní bit a osmibitovou hodnotu (1 bajt), která představuje přenesenou informaci [11]. Pokud nejsou přenášena data, signály rozhraní PS/2 jsou v klidovém stavu v logické 1. Při přenosu datový kabel sériově vysílá informace na frekvenci hodinového signálu. Časový diagram PS/2 přenosu je na obrázku 1.21.

PS/2 klávesnice

PS/2 klávesnice využívá skenovací kódy ke komunikaci se zařízením. Každá z kláves má přiřazený kód, který je odeslán při stisknutí, nebo puštění klávesy. V případě, že se jedná o speciální klávesu, nebo o puštění klávesy, tak může klávesnice přenést více jedenáctibitových slov v sérii. Při stisknutí speciální klávesy bývá odeslána informace $'11100000'$ (tj. $'E0'$ v hexadecimální soustavě) a po ní kód příslušné klávesy. Při puštění klávesy je odeslána hodnota $F0$. Například při puštění šipky směřující doleva (kód $6B$) jsou v sérii odeslány informace $E0, F0, 6B$.



Obrázek 1.22: Časový diagram VGA.

1.3.3 VGA

Charakteristika

VGA je standard pro řadič zobrazující grafický výstup [11]. Využívá paralelního přenosu k předávání informací řadiči grafického výstupu. Přenáší synchronizační signály (HS - Horizontální synchronizace, VS - Vertikální synchronizace) a signály RGB (tj. signály definující vykreslovanou barvu). HS a VS dle své frekvence nastavují rozlišení monitoru a umožňují tedy časovou synchronizaci s monitorem, který vykresluje pixel po pixelu. Po příchodu synchronizačního signálu je třeba čekat daný čas, než se monitor sesynchronizuje (tj. časy back porch a front porch, viz obrázek 1.22). RGB jsou analogové signály a hloubka vykreslované barvy je dána příslušným napětím (často $0 - 7V$) [11].

1.4 Hra Pac-Man

Pac-Man je plošinová arkádová hra, ve které jde o ovládání žluté kuličky s ústy (Pac-Mana) v bludišti vyplněném tečkami, které má za úkol sníst. Krom Pac-Mana se v bludišti nachází také čtyři duchové, kteří se snaží Pac-Mana

dohonit před tím, než stihne sníst všechny tečky v bludišti. Pokud některý z nich Pac-Mana chytí, tak je hráči odebrán jeden ze tří životů a hra začíná odznova uprostřed bludiště bez teček, které jsou již snězeny a duchové jsou vráceni zpět do tzv. domečku. Aby se Pac-Man mohl bránit duchům, tak se na mapě nachází čtyři větší tečky, po jejichž snězení může sníst i duchy, které se následně na chvíli zastaví v domečku a čekají na vyprchání efektu. Hra může být ovládána čtyřmi tlačítky, kde každé nastaví Pac-Manovi nový směr (dle tlačítka nahoru, dolu, doleva, doprava).

Kapitola 2

Návrh hry

V této kapitole se zabývám návrhem obvodu pro finální implementaci. V sekci 2.1 rozeberu návrh hry jako celek. Sekce 2.2 rozebírá návrh obvodu pro ovládání periferií (PS/2, VGA, sedmisegmentový displej). V sekci 2.3 vysvětluji návrh vnitřní logiky samotné hry a v sekci 2.4 poté propisování textur na výstup a inicializaci základních hodnot.

2.1 Specifikace hry

V této sekci rozebírám konkrétní prvky hry a vysětluji návrhy pro jejich řešení. V podsekci 2.1.1 popisuji, jak má hra fungovat jako celek. Podsekce 2.1.2 vysvětluje optimální návrh herního pole, na kterém se hra bude odehrávat a podsekce 2.1.3 popisuje chování postav na navrženém herním poli.

2.1.1 Cíl a chování hry

Cílem práce je navrhnout obvod realizující klon známé 2D arkádové hry a implementovat jej na vývojové desce Digilent Basys 2. Hráč bude moci hru ovládat pomocí šipek na PS/2 klávesnici a stav hry bude moci sledovat na VGA monitoru. Na začátku hry se hráčova postava (Tux-Man) objeví v bludišti, ve kterém se budou nacházet také čtyři duchové. Hráč bude muset pomocí šipek na klávesnici ovládat Tux-Mana a sbírat body rozložené na mapě zatímco nesmí přijít do styku s duchy. Body budou zobrazovány v hexadecimální soustavě na sedmisegmentový displej dostupný na vývojové desce. Tux-Man má tři životy na to, aby posbíral všechny body v bludišti. Životy bude možné sledovat na LED diodách na vývojové desce. Pokud hráče duchové třikrát chytí, tak bude hra zastavena a bude nutné resetovat obvod pro novou hru. Ve hře bude jedna hratelná úroveň, kde se duchové budou pohybovat stejnou rychlostí, jako Tux-Man.

2.1.2 Herní mapa

Herní mapa bude navržena jako mřížka. Navrhuji rozdělit rozlišení VGA na matici, kde každá buňka má rozlišení 16×16 pixelů. Z těchto buněk bude možné jednoduše vytvořit herní mapu s rozlišením 21×19 buněk dle originální mapy. Výhodou tohoto řešení je možnost jednoduchého uchovávání informací o jednotlivých políčkách mapy v paměti a přistupování k těmto informacím dle souřadnic x a y. Každý z herních prvků bude mít své souřadnice, které bude možné mezi sebou porovnávat.

2.1.3 Postavy a jejich chování

Každá z postav bude mít nastavený směr dle vnitřního signálu (u Tux-Mana ovládáno šipkami na klávesnici, u duchů řízeno automatem), na základě kterého se bude posouvat o jednu souřadnici v pravidelném časovém intervalu. Těsně před posunem postavy proběhne kontrola, co se nachází na souřadnicích před danou postavou dle nastaveného směru. Pokud se na souřadnicích před postavou nic nenachází, tak je posunuta. V případě Tux-Mana, pokud se bude jednat o bod, tak se daný bod smaže z paměti a Tux-Man je přesunut na jeho pozici a pokud se bude jednat o zeď, tak jeho souřadnice nebudou změněny. Duchové body budou ignorovat a pokud se před nimi bude nacházet zeď, tak pomocí generátoru náhodných čísel vyhodnotí nový směr. Duchové budou tedy náhodně měnit směr pokaždé, když dojdou ke zdi.

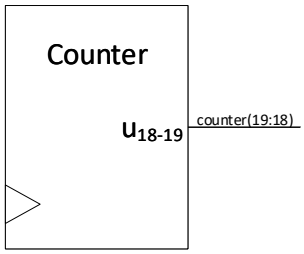
2.2 Periferie

V této sekci rozebírám návrhy číslicových logických obvodů pro komunikaci s periferiemi. V podsekci 2.2.1 popisuji návrh pro ovládání sedmisegmentového displeje. Podsekce 2.2.2 popisuje navržený obvod pro zpracování signálu z PS/2 klávesnice. a v sekci 2.2.3 poté vysvětluji obvod generující signály pro ovládání VGA rozhraní.

2.2.1 Sedmisegmentový displej

Analýza

Pro počítání bodů na sedmisegmentový displej jsem zvolil hexadecimální soustavu, každá číslice tedy bude mít čtyřbitový vstup. Na sedmisegmentovém displeji vývojové desky Digilent Basys 2 je možné na všech čtyřech číslicích zobrazovat v jednu chvíli pouze jednu hodnotu, ale každá z číslic má vlastní povolovací vstup (při logické $'0'$ svítí a při logické $'1'$ nesvítí) . Navrhuji tedy postupně přepínat aktivní číslice tak, aby byla v jednu chvíli aktivní pouze jedna a zároveň přepínat hodnotu, která je zobrazena na displeji. Pokud tak bude provedeno na frekvenci vyšší, než $30Hz$, tak lidské oko nebude schopno zachytit problikávání displejů a bude vytvořena iluze plynulého obrazu zobrazujícího jinou hodnotu na každé číslici. Problémem tohoto řešení je obnovovací



Obrázek 2.1: Čítač s vyvedeným osmnáctým a devatenáctým bitem.

frekvence sedmisegmentového displeje (při problikávání číslice na vysoké frekvenci nemusí vestavěné LED diody stíhat měnit svůj stav). Pokusím se tedy návrhem co nejvíce přiblížit minimální frekvenci nerozeznatelné lidským okem (zhruba třicet snímků za vteřinu).

Návrh obvodu

K docílení chtěné frekvence navrhuji využít čítač, který čítá v reakci na náběžnou hranu hodinového signálu ($50MHz$), jehož x-tý bit bude této frekvence dosahovat. Uvážíme-li čtyřčíslicový displej, frekvenci hodinového signálu $50MHz$ a pro každou číslici chtěnou frekvenci zhruba $30Hz$, tak bude platit následující:

$$(50 * 10^6)/2^x = 4 * 30$$

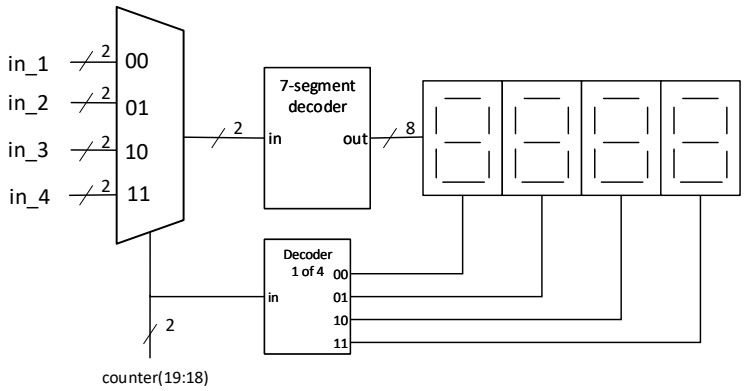
$$\log_2((50 * 10^6)/120) = x$$

$$x = 18.67$$

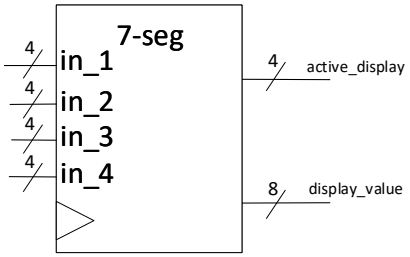
Po zaokrouhlení na celé číslo dolů vyjde osmnáctý bit čítače, který se nejvíce přibližuje chtěné frekvenci. Z důvodu přepínání mezi čtyřmi číslicemi potřebujeme z čítače dva bity. Čítač tedy může vypadat jako na obrázku 2.1. Úkolem tohoto čítače bude paralelně přepínat aktivní vstupy a aktivní číslice na sedmisegmentovém displeji. K přepínání vstupů využiji multiplexor a k přepínání aktivní číslice dekodér 1 z 4. Ke správnému propsání číslice na displej musí být hodnota dekódována dekodérem na sedmisegmentový displej (pro přehlednost dle ASCII tabulky). Celý obvod může vypadat jako na obrázku 2.2. Modul budu dále značit jako na obrázku 2.3.

2.2.2 PS/2

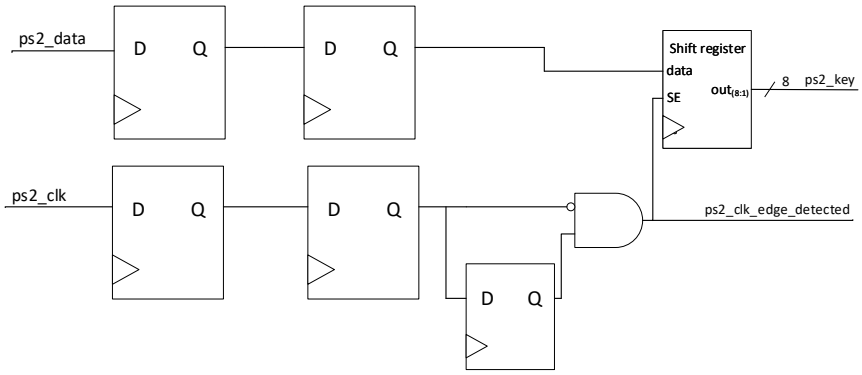
Cílem je navrhnout modul, který zpracuje příchozí signál z klávesnice a na výstup bude posílat kód stisknuté klávesy, indikovat zda se jedná o speciální klávesu a zda je klávesa aktuální (strobe). Pro hru nebude nutné sledovat puštění kláves, ale pouze jejich stisknutí. Tento modul lze pro usnadnění navrhnout tak,



Obrázek 2.2: Připojení sedmisegmentového displeje na čítač.



Obrázek 2.3: Značení modulu pro ovládání sedmisegmentového displeje.



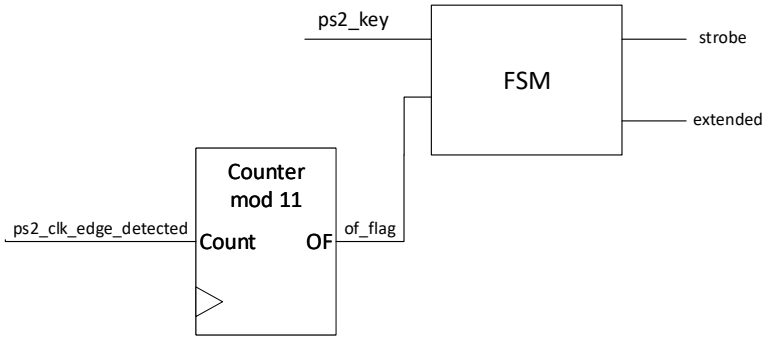
Obrázek 2.4: Návrh obvodu pro zpracování příchozích dat z klávesnice PS/2.

aby přehlížel puštění klávesy. Kvůli o mnoho vyšší frekvenci oscilátoru na desce než je frekvence hodinového signálu PS/2 navrhuji využít synchronizér na oba signály PS/2 a pomocí detektoru sestupné hrany dle protokolu povolovat zápis dat do registrů. Pro uložení jedenáctibitového slova využiji jedenáctibitového posuvného registru s povolovacím vstupem (pokud je aktivní povolovací vstup, tak s náběžnou hranou hodinového vstupu všechny hodnoty předá o řád výš a zapíše vstupní hodnotu na nejnižší bit). Připojením detektoru sestupné hrany z hodinového signálu PS/2 na povolovací vstup posuvného registru docílíme zápisu jedenáctibitového slova do paměti. Obvod může být zakreslen jako na obrázku 2.4. Výstupem druhého až devátého bitu posuvného registru (v obrázku jako *ps2_key*) je informace o klávese.

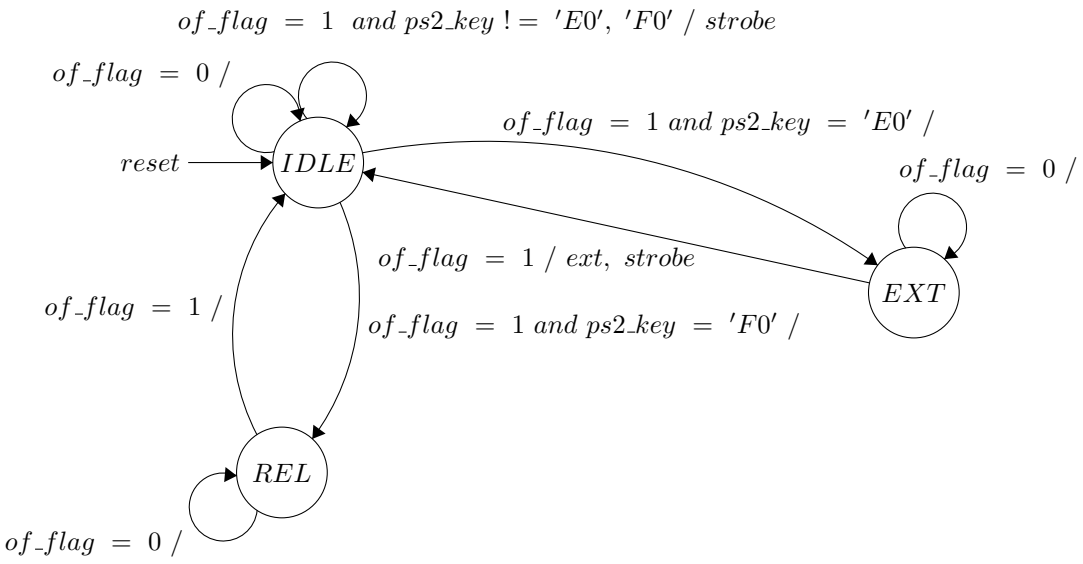
Pro detekci zda je klávesa aktuální nebo se jedná o speciální klávesu nebo o puštění klávesy jsem se rozhodl využít konečný stavový automat (FSM), který bude měnit hodnoty výstupů *strobe* a *extended* (speciální klávesa) na základě příchozích kláves. Pro kontrolu je potřeba, aby automat věděl, kdy je v registru hodnota klávesy aktuální. Výstupu indikujícího, že je informace v posuvném registru aktuální lze docílit připojením signálu *ps2_clk_edge_detected* k čítači modulo 11. Ze signálu přetečení čítače lze poté vyčíst, zda jsou všechna data načtena. Návrh obvodu je na obrázku 2.5, kde OF je indikátor přetečení čítače. FSM lze poté navrhnout se třemi stavy. Rozhodl jsem se využít Mealyho návrh FSM, viz obrázek 2.6. Modul je možné dále značit, jako na obrázku 2.7.

2.2.3 VGA

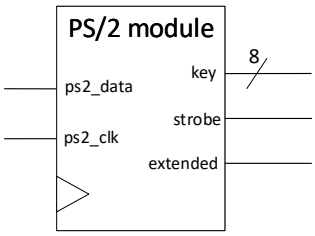
Cílem je navrhnout modul pro VGA, který bude generovat potřebné signály pro správnou funkci VGA monitoru (horizontální a vertikální synchronizace) a na výstup udávat informace o tom, který pixel (dle souřadnic) je právě vykreslován. Jednotlivé barvy bude měnit část obvodu spravující textury dle stavu hry. Pro práci využiji rozlišení 640 × 480 pixelů s frekvencí vykreslování pixelů



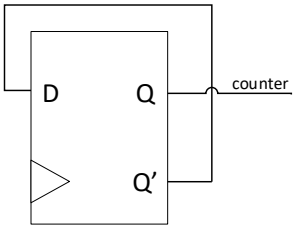
Obrázek 2.5: Návrh obvodu pro detekci speciálních kláves a aktivity.



Obrázek 2.6: Stavový diagram FSM pro indikaci zda je klávesa aktuální, speciální, či se jedná o puštění klávesy.



Obrázek 2.7: Značení modulu pro PS/2.

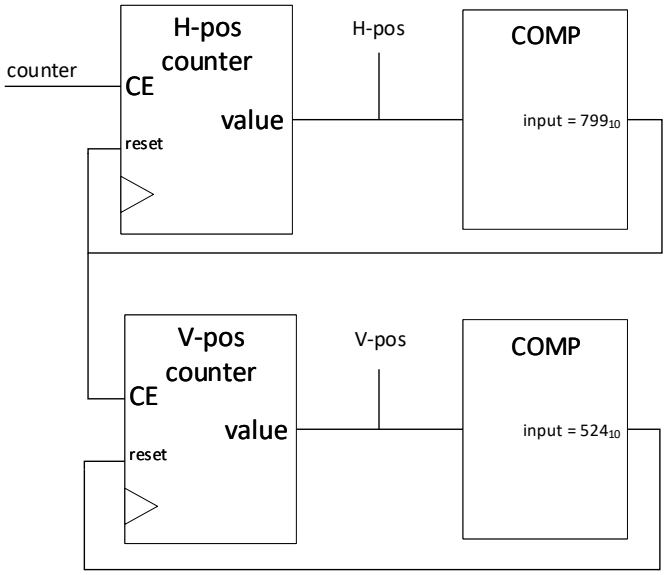


Obrázek 2.8: Dělička dvěma.

25.175MHz. Specifikace tohoto VGA budou dle protokolu [12]:

- Horizontal front porch: 16 taktů
- Horizontální synchronizační pulz: 96 taktů
- Horizontal back porch: 48 taktů
- Vertical front porch: 37 taktů
- Vertikální synchronizační pulz: 2 takty
- Vertical back porch: 60 taktů

K dosažení požadované frekvence (25.175MHz) navrhuji využít toho, že hodinový signál na vývojové desce má frekvenci 50MHz. Využiji jeden D klopný obvod k vydělení frekvence dvěma, viz obrázek 2.8. Signál counter je nyní možné připojit na povolovací signál dvou čítačů modulovaných dle specifikací a číst z nich hodnotu aktuálních souřadnic, viz obrázek 2.9, kde výstup obvodu H-pos je x-ová souřadnice a V-pos je y-ová. Pro resetování čítačů ve chvíli, kdy má souřadnice přejít na nulu využívám komparátor, ve kterém porovnávám hodnotu čítače s hodnotou uvedenou v protokolu VGA. Synchronizační signály budou vzhledem k předchozímu návrhu vysílány tak, aby byly pixely vykreslovány



Obrázek 2.9: Obvod pro výpočet souřadnic vykreslovaného pixelu.

z levé strany do pravé a od shora dolu. K tomu poslouží výstupy H-pos a V-pos a jednoduchý komparátor, který vyhodnotí, zda se hodnota na souřadnicích nachází v rozmezí určeném pro vyslání synchronizačního signálu (v pixelech horizontální signál $656 - 736$ a vertikální $490 - 492$), viz obrázek 2.10. Modul budu dále označovat jako na obrázku 2.11.

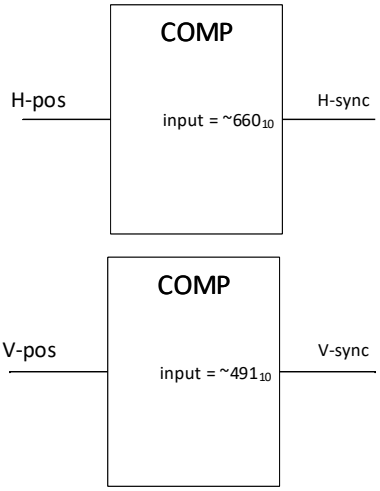
Kvůli kolísavosti vestavěného oscilátoru může vzniknout také malé kolísání obrazu. K zamezení by bylo možné dokoupit oscilátor pro VGA, se kterým by obvod mohl pracovat

2.3 Logika hry

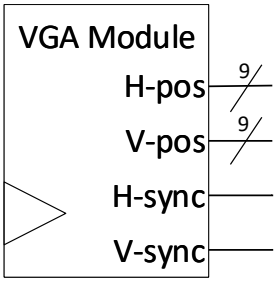
Tato sekce popisuje návrh obvodu pro vnitřní logiku hry. V podsekci 2.3.1 vysvětluji způsob navržení herní mapy. Sekce 2.3.3 popisuje návrh obvodu pro ovládání postavy na mapě a sekce 2.3.4 se zabývá návrhem pro samovolný pohyb duchů po mapě.

2.3.1 Herní mapa jako mřížka

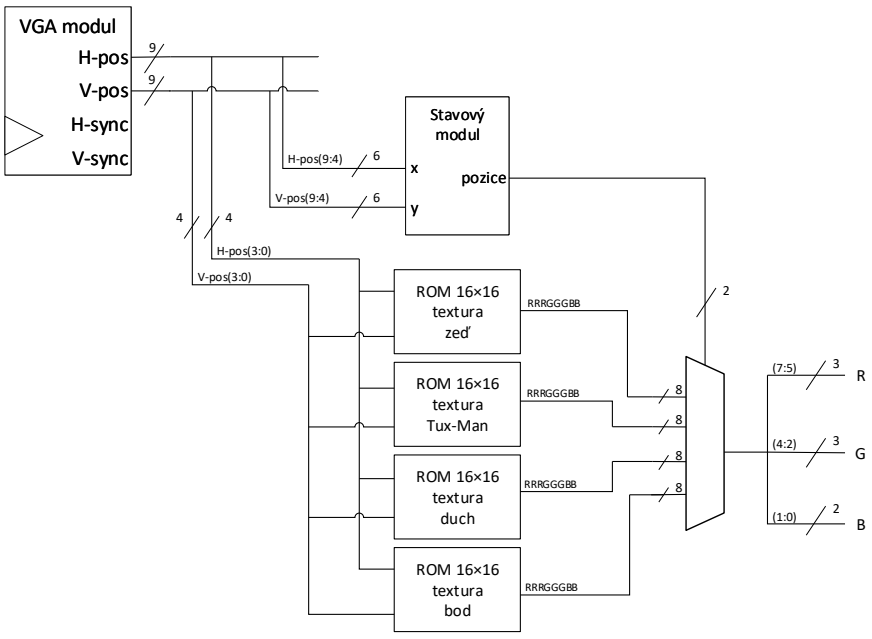
Pro možnost pohodlné práce s prvky v mřížce doporučuji navrhnout modul, který bude měnit a kontrolovat stav hry (pozice jednotlivých prvků, jako jsou duchové, Tux-Man, zdi, body) na základě prvků v 16×16 buňkách této mřížky.



Obrázek 2.10: Komparátory využité ke generování synchronizačních pulzů VGA.

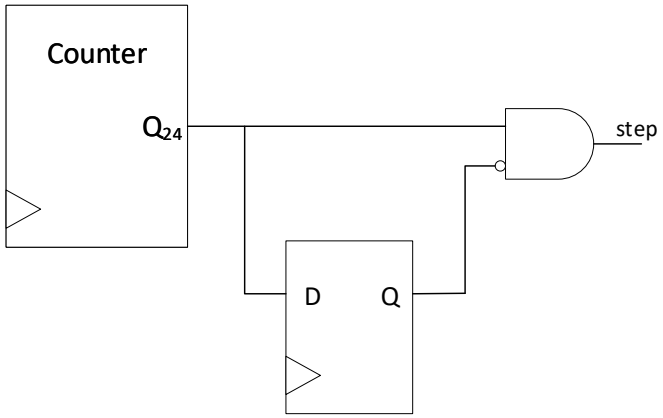


Obrázek 2.11: Značení modulu VGA.



Obrázek 2.12: Hrubý návrh obvodu pro vykreslování textur do mřížky.

Tento modul prozatím nazvu stavový modul. Souřadnice 16×16 buňky získáme ze čtvrtého až devátého bitu signálu H-Pos a V-Pos z VGA modulu. Nultý až třetí bit těchto signálů lze využít k vykreslování textur na základě souřadnic. Navrhuji tedy texturu pro každý herní prvek uložit do matice paměťových buňek 16×16 (textury se nebudou přepisovat, můžeme tuto paměť tedy nazývat Read-Only Memory, česky paměť pouze ke čtení, zkr. ROM). Výstupem ROM s texturami budou aktuální hodnoty RGB s hloubkami dle možností, které poskytuje vývojová deska pro implementaci (tři bity pro hloubku červené barvy, tři bity pro hloubku zelené a dva bity pro hloubku modré). Pokud stavový modul bude navržen tak, aby jeho výstupy byly informace o pozici jednotlivých prvků na základě časování VGA (např. pokud VGA právě vykresluje pixel na souřadnicích [1,1], kde se nachází zeď, tak signál stavového modulu, který indikuje přítomnost zdi bude dosahovat logické $'1'$), tak lze těmito výstupy řídit multiplexor, který dle těchto řídících signálů udá na výstupy R,G a B požadované barvy (dle vybrané textury). Hrubý návrh obvodu pro výběr textur je na obrázku 2.12. Součásti návrhu stavového modulu budu vysvětlovat v podsekcích 2.3.2, 2.3.3 a 2.3.4.



Obrázek 2.13: Návrh generátoru signálu indikujícího herní krok.

2.3.2 Pohyb postav po mapě

Herní krok

Navrhuji vybrat frekvenci pro pohyb postav okolo $2Hz$ (posun o buňku dvakrát za vteřinu), aby se hra dala stíhat, ale zároveň neprobíhala pomalu. Podobně jako u sedmisegmentového displeje v podsekci 2.2.1 v sekci 2.2 navrhuji využít čítač ke zpomalení hodinového signálu a získání chtěné frekvence. Za předpokladu, že chceme frekvenci blízkou $2Hz$, bude výpočet bitu čítače vypadat takto:

$$(50 * 10^6)/2^x = 2$$

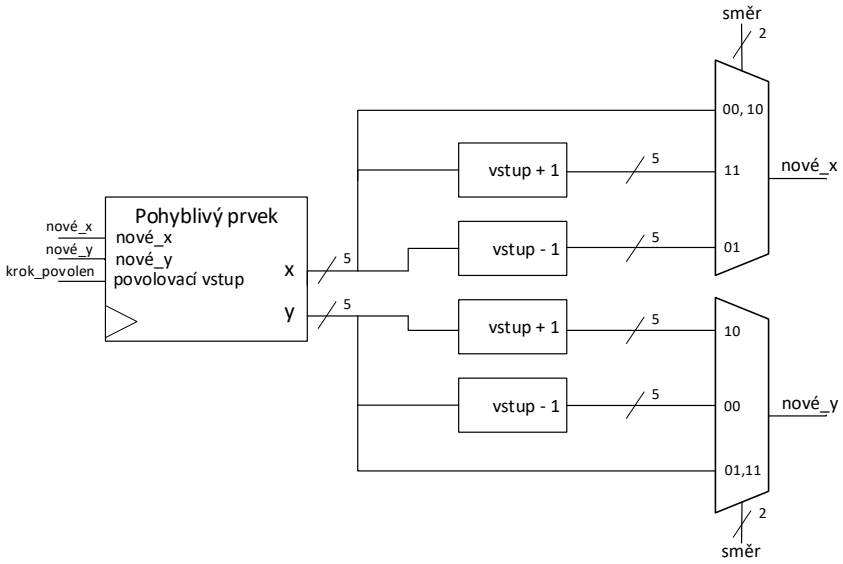
$$\log_2((50 * 10^6)/2) = x$$

$$x = 24.58$$

Hodnotu zaokrouhlím dolů, aby hra nebyla moc pomalá. S dvacátým čtvrtým bitem čítače vyjde frekvence dle výpočtu na $2.99Hz$. Herní krok navrhuji signalizovat v jednom taktu, abych předešel časovým komplikacím v pozdější fázi návrhu. Využiji tedy detektor náběžné hrany, abych docílil jednotaktového signálu zhruba třikrát do vteřiny. Tento signál nazvu step, viz obrázek 2.13.

Posun pohyblivých prvků s ohledem na zdi

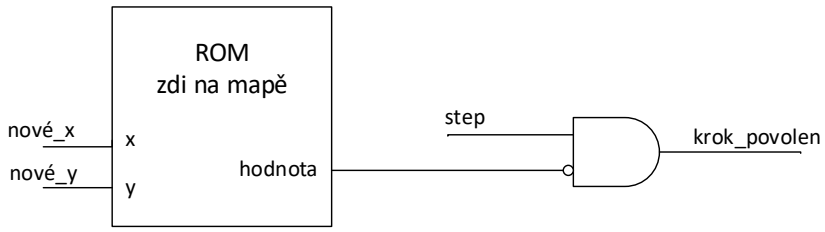
Každý z herních prvků (Tux-Man, duchové, body a zdi) bude mít v paměti uložené své souřadnice. Pro zdi a body navrhuji, z důvodu, že se na mapě nenachází pouze jednou a že nejsou pohyblivé, vytvořit matici paměťových buněk (u zdí se bude jednat o ROM a u bodů z důvodu, že se budou mazat z mapy při přejetí Tux-Manem, se bude jednat o Random-Access memory, zkratka RAM)



Obrázek 2.14: Obecný návrh obvodu pro vyhodnocení následujících souřadnic pohyblivých prvků.

a k nim přistupovat pomocí souřadnic. Pro Tux-Mana a každého z duchů navrhuji ukládat informace o souřadnicích ve vícebitových D klopných obvodech. Při změně souřadnic a příchodu herního kroku bude možné povolit zápis nových souřadnic. Modul pro každý pohyblivý prvek se tedy bude chovat jako velký D klopný obvod s povolovacím vstupem. Protože se postavy budou pohybovat pouze v mřížce v souřadnicích 21×19, tak je možné, aby souřadnice x a y byly pětibitové signály.

Každý z pohyblivých prvků bude mít vlastní dvoubitový signál, který bude určovat jeho aktuální směr ("00" bude reprezentovat směr nahoru, "01" doleva, "10" dolů a "11" doprava). Návrh pro ovládání směrového signálu budu popisovat v podsekcích 2.3.3 a 2.3.4. Pomocí signálu, který vybírá aktuální směr je možné předpovědět souřadnice pro další krok, viz obrázek 2.14, kde signál *smer* reprezentuje vybraný směr a *nove_x*, *nove_y* jsou nové souřadnice a *krok_povolen* je signál, který bude aktivní, pokud proběhne herní krok a zároveň před pohyblivým prvkem nebude stát zeď. Se signály nabývajícími hodnoty předpokládaných souřadnic je nyní možné z ROM s informacemi o zdech vyčíst, zda se na daných souřadnicích nachází zeď, resp. zda se tam pohyblivý prvek může pohnout. Výstup této ROM bude logická $'1'$ pokud se zde zeď nachází a logická $'0'$ pokud nikoliv. Pokud tedy proběhne herní krok a na předpokládaných souřadnicích se nebude nacházet zeď, tak je možné pohyblivému prvku povolit postup na nové souřadnice (signál *krok_povolen*). Návrh pro generování signálu



Obrázek 2.15: Návrh obvodu pro generování signálu, který povoluje přepsání souřadnic.

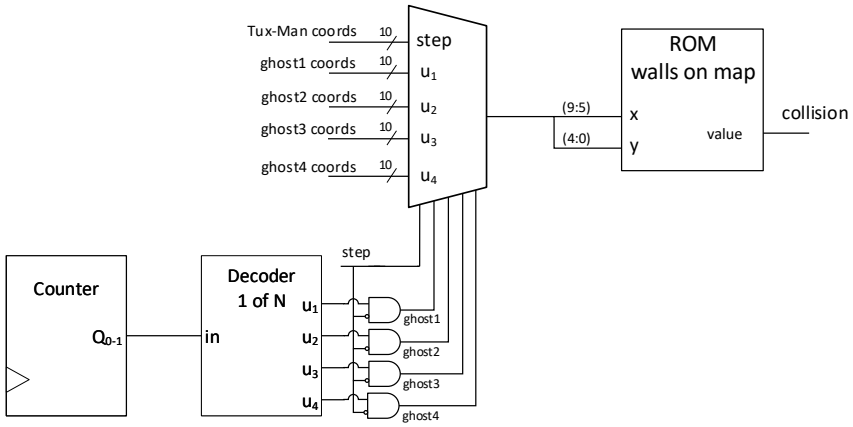
krok_povolen je na obrázku 2.15.

Optimalizace

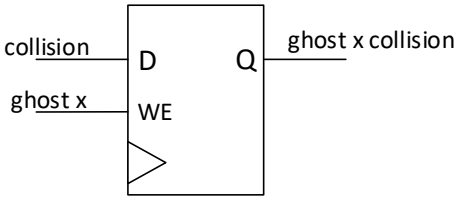
Kdyby se výše uvedený obvod implementoval pro každý pohyblivý prvek ve hře, tak by byla nutná inicializace paměti pro každý tento prvek (resp. každý z duchů a Tux-Man by měli svou vlastní paměť s informacemi o zdech na mapě). Důvodem je, že z jedné paměti může v jednu chvíli číst pouze jeden prvek, viz podsekce3.3.2 v kapitole Implementace. Navrhuji tedy vytvořit si vlastní tzv. arbitrovaný přístup pro souřadnice jednotlivých pohyblivých prvků ve hře a každému z nich dát prioritu přístupu do paměti ve chvíli, kdy to potřebuje. Tux-Man bude přístup potřebovat pouze při herním kroku, aby obvod dokázal vyhodnotit, zda je možné ho posunout, nebo ne. Pro usnadnění práce s duchy doporučuji duchům střídat prioritu přístupu k paměti po celou dobu, pokud zrovna neprobíhá herní krok (jestli se budou moci posunout a kam bude vyhodnoceno dříve, než ve chvíli herního kroku). Arbitraci sběrnice je možné navrhnout pomocí čítače, dekodéru a multiplexoru, viz obrázek 2.16, kde ROM je paměť s informacemi o zdech na mapě, decoder $1ofN$ je dekodér jedna z N, vstupní signály multiplexoru jsou sjednocené souřadnice jednotlivých pohyblivých prvků a $ghost1 - ghost4$ je pojmenování duchů. V tuto chvíli by každý z duchů měnil směr na základě stejného signálu *collision*. Navrhuji tedy jim za pomocí D klopných obvodů zapsat informaci *collision* pouze ve chvíli, kdy na ně přijde řada. Tímto získáme signál s informací o kolizi každého ducha. U Tux-Mana nebude potřeba zapisovat informaci do paměti z důvodu, že tuto informaci využije pouze v jednom hodinovém taktu při příchodu herního kroku. Obvod pro uchování informace o kolizi ducha může vypadat jako na obrázku 2.17, kde x je číslo ducha.

Sbírání bodů

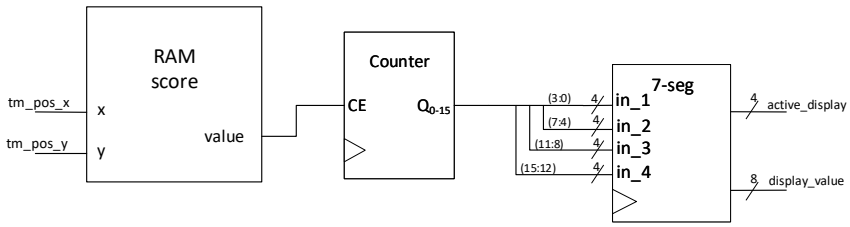
Jediná postava, která bude moci sbírat body bude Tux-Man. Navrhuji připojit souřadnice Tux-Mana na inicializovanou RAM se souřadnicemi bodů. Každá paměťová buňka v paměti bude nabývat logické $'1'$, pokud se na ni nachází



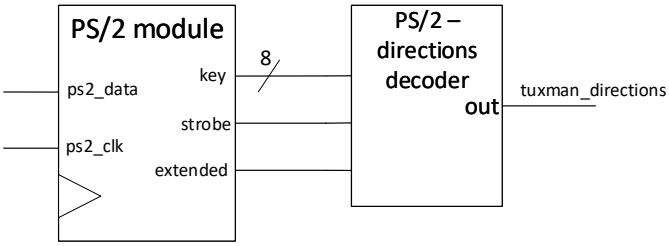
Obrázek 2.16: Návrh obvodu pro optimalizaci přístupu do paměti se zdmi.



Obrázek 2.17: Návrh obvodu pro uchování informace o buňce před duchem.



Obrázek 2.18: Návrh obvodu pro počítání bodů sebraných Tux-Manem.

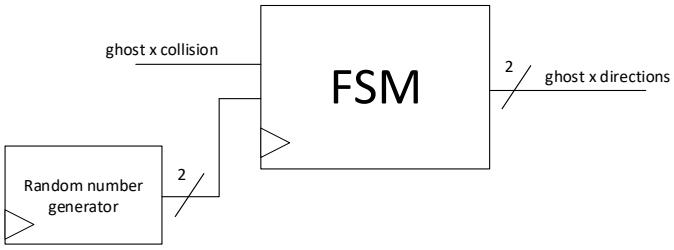


Obrázek 2.19: Obvod pro ovládání Tux-Mana.

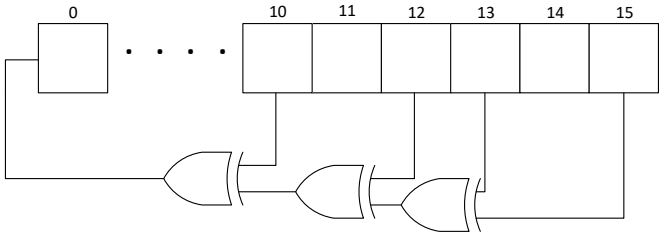
bod a logické $'0'$ pokud nikoliv. Pokud v této RAM na souřadnicích Tux-Mana vynulujeme hodnotu ve chvíli, kdy na tyto souřadnice Tux-Man stoupne, tak se na výstupu paměti na jeden takt zobrazí informace o tom, zda se na buňce nachází bod, nebo ne. Tímto signálem je možné čítat počet bodů, které hráč nasbíral. Výstup tohoto čítače je možné připojit rovnou na sedmisegmentový displej. Návrh obvodu pro sbírání bodů je na obrázku 2.18, kde signály *tm_pos_x*, *tm_pos_y* jsou souřadnice Tux-Mana a RAM score je paměť s informacemi o bodech, která obsahuje vnitřní implementaci pro vynulování paměťové buňky při čtení z dané souřadnice.

2.3.3 Ovládání Tux-Mana

Pro ovládání Tux-Mana pomocí PS/2 modulu navrhuji využít dekodér, který přeloží vybranou klávesu na směrový signál definovaný v podsekci 2.3.2. Tux-Manův směrový signál budu nazývat *tuxman_directions*. Návrh obvodu pro ovládání Tux-Mana je na obrázku 2.19, kde dekodér udává při aktivním signálu *strobe*, *extended* a daným kódem klávesy příslušný směr ("00" při hodnotě $'75'$, "01" při hodnotě $'6B'$, "10" při hodnotě $'72'$, "11" při hodnotě $'74'$).



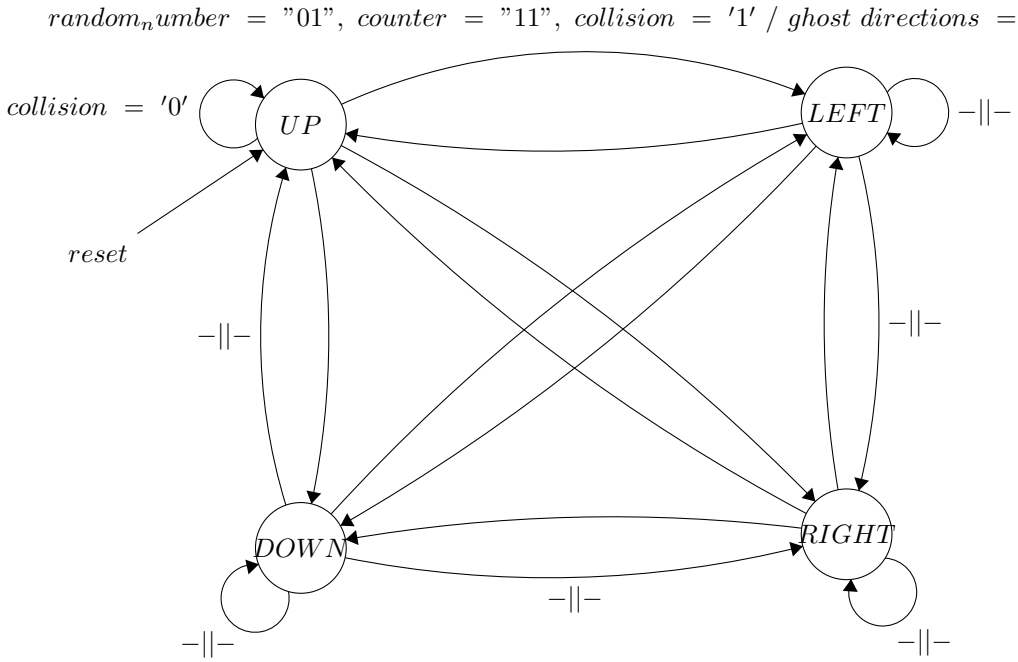
Obrázek 2.20: Hrubý návrh obvodu pro ovládání duchů.



Obrázek 2.21: Náčrt generátoru náhodných čísel.

2.3.4 Ovládání pohybu duchů

Pro ovládání pohybu každého ducha navrhuji využít FSM. Dle sekce 2.1 může být FSM navržen tak, že duch bude mít nastaven jeden směr tak dlouho, dokud se před ním nebude nacházet zeď. Při posunu ke zdi navrhuji využít generátoru náhodných čísel a generovat dvoubitová náhodná čísla do té doby, dokud nedají dohromady směr, kterým není vůči duchovi zeď a poté může znovu čekat, dokud duch nedojde ke zdi. Vzhledem k tomu, že frekvence posunu postav je velmi nízká, tak je velmi malá pravděpodobnost (méně než tisícina procenta), že duch nestihne vyhodnotit další směr a bude posunut do zdi. Hrubý návrh obvodu pro ovládání duchů je na obrázku 2.20, kde Random number generator je generátor náhodných čísel, x je číslo ducha a ghost x directions je směr, kterým duch míří. Generátor náhodných čísel bude vnitřně implementován jako posuvný registr, který s každým taktem vyhodnotí pomocí logických hradel XOR novou hodnotu, kterou vloží na nejnižší bit. Generátor si lze představit jako na obrázku 2.21 [13] Tento generátor náhodných se nazývá LFSM (angl. Linear-feedback shift register, česky Posuvný registr s lineární zpětnou vazbou). FSM pro nastavování směru lze navrhnout se čtyřmi stavy, kde každý stav reprezentuje směr, kudy duch může jít. Pokud kolizní signál ducha nabývá logické $'0'$, tak není třeba měnit směr. Pokud bude nabývat logické $'1'$, tak FSM přejde do nového stavu dle hodnoty na generátoru náhodných čísel. Je třeba ale brát v po-



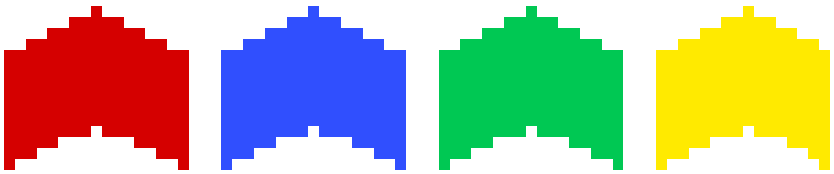
Obrázek 2.22: Stavový diagram FSM pro vyhodnocení nového směru ducha.

taz, že kvůli sdílení ROM, která uchovává informace o zdech je nutné čekat čtyři takty na aktuální data. Toto je možné ošetřit připojením dvoubitového čítače a měnit stav automatu pouze tehdy, když tento čítač nabyde hodnoty "11". Tento čítač bude tedy dalším vstupem FSM. Výstupem tohoto FSM musí být vždy směr dle stavu, ve kterém se nachází, aby bylo možné z paměti přečíst, zda se v novém směru nachází zeď. Hrubý stavový diagram FSM je na obrázku 2.22 Z důvodu obsáhlejšího návrhu se v obrázku nachází pouze několik ukázkových přechodových podmínek, ze kterých ostatní vyplívají. Počátečný stav je směr nahoru, aby duchové na začátku vyjeli ze svého hnízda dle návrhu mapy, viz podsekce 2.4.

2.4 Inicializace pamětí

2.4.1 Textury

Textury budou navrženy v matici paměťových buněk 16×16, kde každá buňka bude nabývat osmibitové hodnoty, viz sekce 2.3. Inspirací budou tedy kresby 16×16 pixelů. Duchy navrhuji inicializovat jako na obrázku 2.24. Tux-Man bude vypadat jako žluté kolečko, viz obrázek 2.23. Zdi navrhuji udělat jako jednoduchý modrý čtverec a body jako menší bílé kolečko, viz obrázek 2.25.



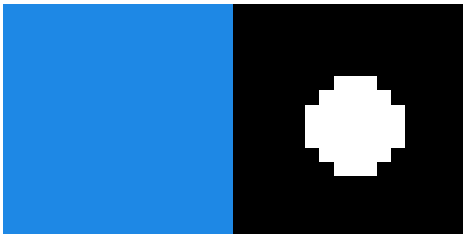
Obrázek 2.23: Návrh textur duchů.

Obrázek 2.24: Návrh textury Tux-Mana.

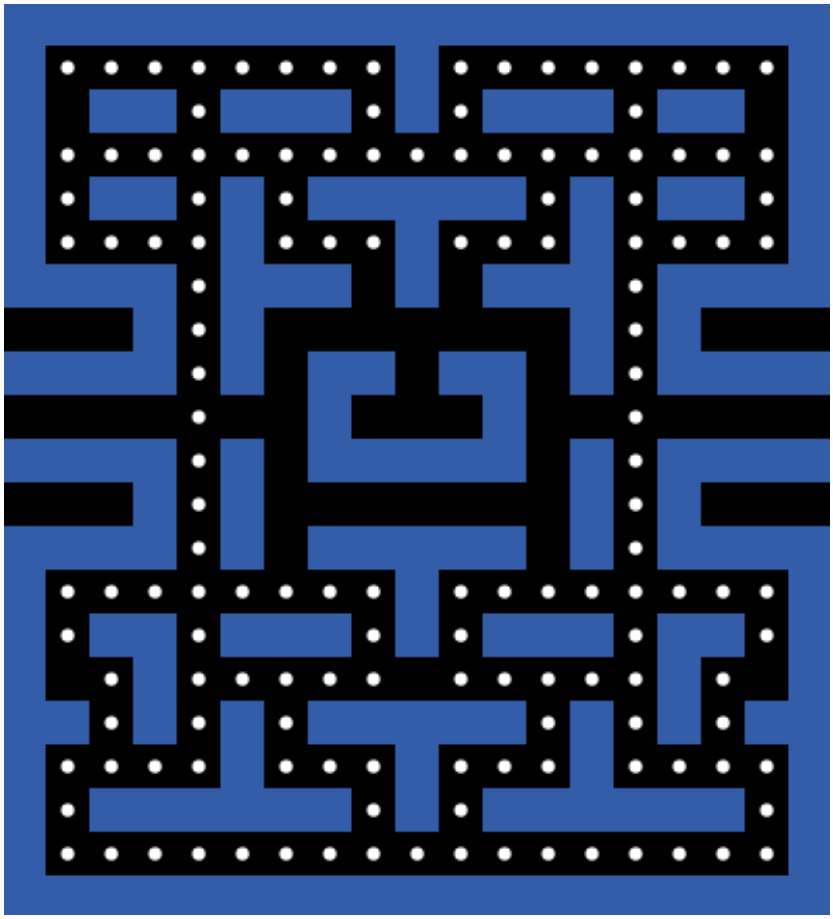
Pokud nebude na místě žádná z těchto textur, tak navrhuji vykreslovat pouze černou barvu na pozadí.

2.4.2 Základní pozice prvků hry

Základní pozice prvků budou v paměti uloženy stejným způsobem jako textury, viz podsekce **??** s tím rozdílem, že tyto paměti budou uchovávat logickou $'0'$, nebo logickou $'1'$. Logická $'0'$ reprezentuje nepřítomnost prvku, logická $'1'$ reprezentuje přítomnost. Např. paměti pro zdi a body budou uchovávat po inicializaci data v matici stejně, jako je na obrázku 2.26, kde modré pole je zeď (v paměti se zdmi logická $'1'$), pole s bílou tečkou je bod a černé pole je prázdné v obou pamětech jako logická $'0'$. Základní pozice Tux-Mana bude na souřadnicích v mřížce $[9; 15]$ a základní pozice všech čtyř duchů $[9; 9]$.



Obrázek 2.25: Návrh textury pro zeď a bod na mapě.



Obrázek 2.26: Mapa hry.

Kapitola 3

Implementace

Tato kapitola se zabývá implementací navržených číslicových obvodů na vývojovou desku Digilent Basys 2 v jazyce VHDL.

3.1 Úvod do implementace

V této sekci popíšu očekávání od implementační části a dostupné prostředky, které využiji pro práci. Podsekce 3.1.1 popisuje očekávaný postup práce a v podsekci 3.1.2 jsou popsány dostupné prostředky pro implementaci.

3.1.1 Cíle implementace

Cílem je v jazyce VHDL popsat obvod, který bude sesyntetizovatelný a výsledné propojení obvodů v FPGA bude funkcí přesně odpovídat navrženému obvodu. Strukturu popisu se pokusím navrhnout tak, aby výsledné moduly pro jednotlivé úlohy měly stejně pojmenované vstupy a výstupy a obecně práci co nejvíce sjednotit s návrhem. Postupovat v popisu obvodů budu stejně jako ve fázi návrhu, viz kapitola 2 s případnými poznámkami k postupu a testování.

3.1.2 Prostředky pro implementaci

Pro implementaci využiji vývojovou desku Digilent Basys 2. Vývojová deska obsahuje vestavěné porty PS/2 a VGA, které jsou vhodné pro moji práci. Dále lze využít i vestavěný sedmisegmentový displej a LED diody pro počítání bodů a indikaci zbývajících životů hráče. K naprogramovanému FPGA na vývojové desce půjde připojit jakýkoliv funkční VGA monitor a jakákoliv funkční PS/2 klávesnice a správná funkce zůstane zachována.

Kód VHDL budu psát ve vývojovém prostředí Xilinx ISE, které podporuje syntézu kódu a vytvoření tzv. bitstreamu (informace, která je uložena do paměti FPGA) pro FPGA Xilinx Spartan-3E. Pro nahrání bitstreamu do FPGA využiji software Digilent Adept.

3.2 Periferie

V této sekci popíšu postup popisu jednotlivých modulů pro propojení periferií na základě návrhu. V podsekci 3.2.1 popíšu implementaci návrhu modulu pro sedmisegmentový displej, v podsekci 3.2.2 implementaci modulu pro PS/2 a podsekce 3.2.3 vysvětlí implementaci modulu pro VGA.

3.2.1 Sedmisegmentový displej

Dle podsekce 2.2.1 je cílem vytvořit entitu, která bude mít čtyři čtyřbitové vstupní hodnoty, jednu výstupní osmibitovou a jednu výstupní čtyřbitovou. Pojmenujme tuto entitu Segment_Display. Aby měla entita stejnou funkci jako výsledný návrh segmentového displeje, bude vypadat takto:

```
entity Segment_Display is
    Port ( in_1 : in  STD_LOGIC_VECTOR (3 downto 0);
           in_2 : in  STD_LOGIC_VECTOR (3 downto 0);
           in_3 : in  STD_LOGIC_VECTOR (3 downto 0);
           in_4 : in  STD_LOGIC_VECTOR (3 downto 0);
           active_display : out  STD_LOGIC_VECTOR (3 downto 0);
           display_value : out  STD_LOGIC_VECTOR (7 downto 0);
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC);
end Segment_Display;
```

popis vnitřního zapojení je dle schématu. Modul jsem testoval tak, že jsem připojil spodní dva bity všech vstupních hodnot na všech osm přepínačů na desce a testoval, zda jde vidět správný výstup. Lidským okem bohužel ale bylo možné vidět problikávání displejů na frekvenci $30Hz$. Bohužel bylo možné lehce sledovat i vyšší frekvence, takže k uspokojivému výsledku jsem se dostal až po připojení šestnáctého a sedmnáctého bitu navrženého čítače ($763Hz$).

3.2.2 PS/2

Cílová entita dle podsekce 2.2.2 bude vypadat následovně a pojmenuji ji PS2:

```
entity PS2 is
    Port ( ps2_data : in  STD_LOGIC;
           ps2_clk : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           key : out  STD_LOGIC_VECTOR (7 downto 0);
           strobe : out  STD_LOGIC;
           extended : out  STD_LOGIC);
end PS2;
```

Vnitřní zapojení jsem rozdělil na dvě části, jimiž jsou datová cesta a kontrolér. Kontrolér je popis konečného stavového automatu (FSM) a datová cesta je zbytek obvodu. Tyto dvě části spojuji pomocí komponent ve vrchní entitě. Obvod je popsán stejně jako ve schématu. Pro popis posuvného registru jsem využil signál, kterému přiřazuji nové hodnoty pomocí operátoru bitového posunu, viz proces níže, který reprezentuje posuvný registr, kde signál shift_register_output je signál posuvného registru, ps2_edge_detected je signál detekující sestupnou hranu dle návrhu a ps2_data_synced je aktuální výstup synchronizéru pro datový signál:

```
shift_register_holding_key : process(clk)
begin
if(clk'event and clk='1') then -- rising edge on CLK
        if(reset = '1') then -- reset
                shift_register_output <= (others => '0');
        else
                if ps2_clk_edge_detected = '1' then
                        shift_register_output <=
                         ps2_data_synced &
                        shift_register_output(10 downto 1);
                end if;
        end if;
end if;
end process;
```

Modul jsem testoval pomocí propojení se sedmisegmentovým displejem, kde bylo cílem zobrazovat v hexadecimální soustavě kód stisknuté klávesy po jejím stisknutí. Zde proběhlo testování bez problému.

3.2.3 VGA

Entita VGA bude vypadat dle podsekce 2.2.3 takto:

```
entity VGA is
    Port (      clk : in  STD_LOGIC;
                reset : in  STD_LOGIC;
                VSync : out  STD_LOGIC;
                HSync : out  STD_LOGIC;
                HPos_out : out STD_LOGIC_VECTOR(9 downto 0);
                VPos_out : out STD_LOGIC_VECTOR(9 downto 0)
        );
end VGA;
```

Pro pohodlnou práci v tomto modulu jsem se rozhodl porovnávat hodnoty čítačů v komparátorech s konstantami, vytvořil jsem si tedy konstantu pro každou hodnotu týkající se časování dle specifikace VGA, viz:

```
constant hva : integer := 640;
```

```
constant hfp : integer := 16;
constant hsp : integer := 96;
constant hbp  : integer := 48;

constant vva : integer := 480;
constant vfp : integer := 10;
constant vsp : integer := 2;
constant vbp : integer := 33;
```

Později při testování, když monitor hlásil neplatný vstup, jsem došel k výsledku, že pokud se aktuální vykreslovaná část nenachází ve viditelné části monitoru, tak je nutné vykreslovat černou barvu, resp. logické $'0'$ na pinech RGB. Přidal jsem do obvodu tedy další komparátor, jehož vstupem jsou aktuální vykreslované souřadnice H-Pos a V-Pos a výstupem je signál, který jsem pojmenoval active. active nabývá logické $'1'$ pokud jsou aktuální H-Pos a V-Pos ve viditelné zóně, viz následující proces:

```
Activity_indicator : process(HPOS, VPOS)
begin
        if (HPOS < hva) and (VPOS < vva) then
                active <= '1';
else
                active <= '0';
end if;
end process;
```

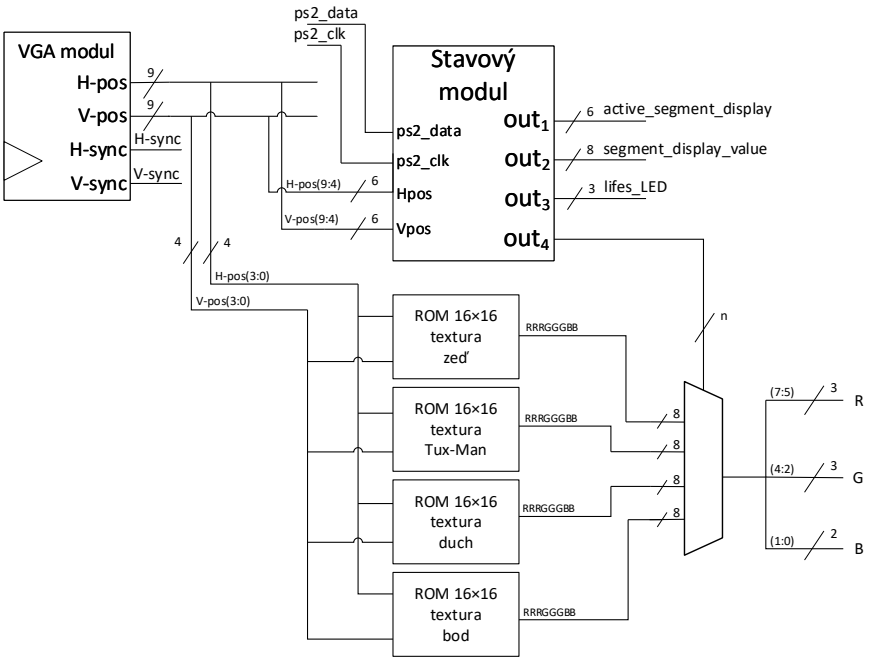
Výstupy RGB musejí nyní být řízeny multiplexorem, který na základě vstupu active vykresluje buďto vybranou barvu, nebo černou v případě, že active nabývá logické $'0'$. Pro testování jsem na monitor pomocí komparátorů nakreslil bílé okraje. Vše nyní fungovalo, jak mělo.

3.3 Logika hry

3.3.1 Celkové propojení

Popis logiky hry probíhal jako v návrhu v sekci 2.3. Kvůli menšímu rozsahu bylo možné uchovat stavový modul jako monolitickou část vrchní entity, která doposud propojovala periferie. Po lehkém rozšíření schématu z návrhové fáze může tato vrchní entita nyní vypadat jako na obrázku 3.1. Jediným neznámým signálem zde je lifes_LED, který je může být připojen na LED diody na vývojové desce a pomocí dekodéru $1zN$ zobrazovat zbývající životy hráče a n jsou signály o pozici každého prvku ve hře (multiplexor pro vyhodnocování signálů RGB bude mít více podmínek z důvodu otáčení postav a vykreslování černé barvy, pokud se aktuální vykreslovaný pixel nenalézá ve viditelné části VGA). Vrchní entita bude tedy vypadat následovně:

```
entity TuxManGame is
```



Obrázek 3.1: Rozšířený návrh vrchní entity.

TuxManGame Project Status			
Project File:	TuxManFPGA.xise	**Parser Errors:**	No Errors
Module Name:	TuxManGame	**Implementation State:**	Mapped
Target Device:	xc3s100e-4cp132	**• Errors:**	X 2 Errors (2 new)
Product Version:	ISE 12.4	**• Warnings:**	46 Warnings (4 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	289	1,920	15%	
Number of 4 input LUTs	1,999	1,920	104%	OVERMAPPED
Number of occupied Slices	1,038	960	108%	OVERMAPPED
Number of Slices containing only related logic	1,038	1,038	100%	
Number of Slices containing unrelated logic	0	1,038	0%	
Total Number of 4 input LUTs	2,064	1,920	107%	OVERMAPPED
Number used as logic	1,999			
Number used as a route-thru	65			
Number of bonded IOBs	26	83	31%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	4.61			

Obrázek 3.2: Shrnutí Xilinx ISE s upozorněním na nedostatek prvků v FPGA.

```
    Port ( ps2_clk : in STD_LOGIC;
           ps2_data : in STD_LOGIC;
           R,G : out STD_LOGIC_VECTOR(2 downto 0);
           B : out STD_LOGIC_VECTOR(1 downto 0);
           HSync : out STD_LOGIC;
           VSync : out STD_LOGIC;
           active_segment_display : out STD_LOGIC_VECTOR (3 downto 0);
           segment_display_value : out STD_LOGIC_VECTOR (7 downto 0);
           lifes_LED : out STD_LOGIC_VECTOR (2 downto 0);
           reset : in STD_LOGIC;
           clk : in STD_LOGIC);
end TuxManGame;
```

3.3.2 Využití dostupných prostředků a problémy implementace

Při prvotním návrhu, viz 2.3.2, kde si každý z pohyblivých prvků inicializoval vlastní paměť jsem se setkal s problémem, kde v FPGA nebylo dostatek místa pro tyto paměti. Při pokusu o implementaci s tímto řešením byla od Xilinx ISE vypsána chybová hláška a ve shrnutí, viz obrázek 3.2, byl vyhlášen nedostatek LUTů. Po optimalizaci pomocí arbitrace sběrnice se problém již nenaskytoval. Aktuální shrnutí využitých dostupných prostředků je na obrázku 3.3

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	375	1,920	19%	
Number of 4 input LUTs	1,348	1,920	70%	
Number of occupied Slices	921	960	95%	
Number of Slices containing only related logic	921	921	100%	
Number of Slices containing unrelated logic	0	921	0%	
Total Number of 4 input LUTs	1,410	1,920	73%	
Number used as logic	1,348			
Number used as a route-thru	62			
Number of bonded IOBs	29	83	34%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	4.10			

Obrázek 3.3: Aktuální shrnutí využitých dostupných prostředků.

Kapitola 4

Testování

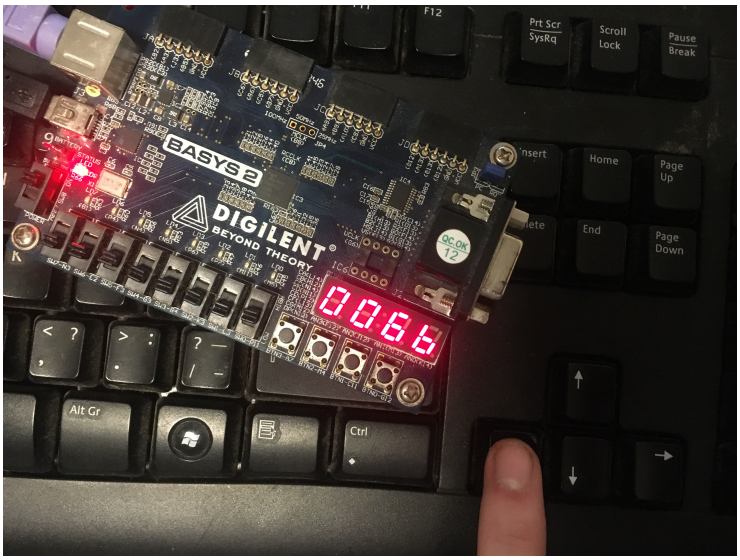
Moduly jsem testoval postupně pokaždé, když jsem dokončil jejich implementaci. První byl modul pro práci se sedmisegmentovým displejem, který jsem otestoval tak, že jsem na spodní dva bity každého ze vstupů připojil přepínače z vývojové desky Digilent Basys 2 (pro čtyři číslice sedmisegmentového displeje osm přepínačů). Sedmisegmentový displej správně zobrazoval příslušné hodnoty.

Druhým modulem byl PS/2 modul, jenž jsem testoval za pomocí sedmisegmentového displeje. cílem bylo, aby se na sedmisegmentovém displeji při stisknutí klávesy zobrazil příslušný kód klávesy, což bylo úspěšné, viz obrázek 4.1.

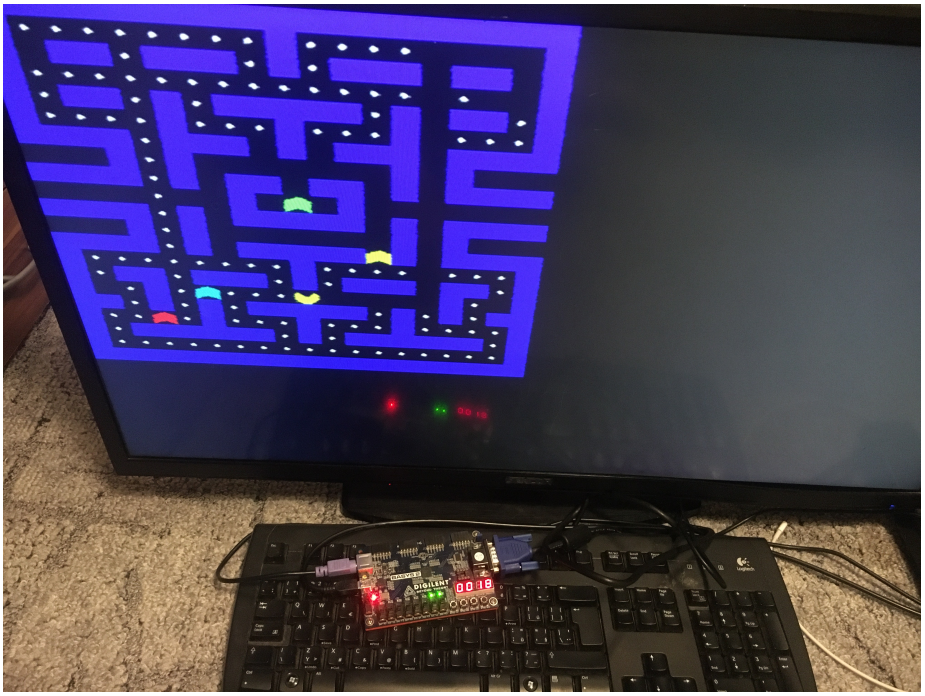
Dalším modulem byl VGA, jehož testování probíhalo samostatně. Po správné optimalizaci synchronizačních signálů jsem pomocí komparátorů vykreslil bílý rámeček na VGA monitor a sledoval, zda má časování nějaké nesrovnalosti. Když testování VGA proběhlo bez problému, tak jsem začal s implementací hry samotné.

Hru jsem dále testoval postupným hraním hry s každou novou částí. Prvním úkolem bylo vytvořit herní mapu, na které by se dalo hrát. Poté jsem do ní vložil první postavu, která šla ovládat PS/2 klávesnicí a zastavila se před zdmi na mapě. Když tato část fungovala, tak jsem přidal na mapu body, které by mohla postava sbírat. Poslední částí byly postavy řízené automatem, jejichž testování znovu probíhalo postupným hraním.

Toto postupné testování bylo velmi přívětivé a umožnil mi ho hierarchický návrh, který jsem při postupu dodržoval. Považuji ho za velkou výhodu při postupu návrhu a implementace logických číslicových obvodů. Když hra byla kompletní a všechny prvky implementované, tak jsem strávil v kuse dvě hodiny hraním a nenarazil jsem na žádnou závadu. Hra je plně hratelná dle návrhu. Ukázka hry je na obrázku 4.2.



Obrázek 4.1: Testování modulu pro PS/2.



Obrázek 4.2: Testování hry.

Závěr

Cílem práce bylo vytvořit klon známé arkádové hry pomocí číslicového návrhu a realizací na programovatelném hradlovém poli.

První a nejdůležitější součástí práce bylo naučit se základní principy číslicového návrhu. Po nastudování kombinačních, sekvenčních obvodů a synchronního návrhu ve formě přednášek konzultanta odborné práce jsem vytvořil první číslicové návrhy, které obsahovaly návrh pro obsluhu periferií, které jsem využíval k odborné práci.

S postupem práce jsem se doučoval nejen nové logické bloky, které byly potřebné pro návrh, ale také jsem musel využít protokolů sedmisegmentového displeje, PS/2 a VGA. Pro testování návrhů pro připojení periferií a možnost další implementace odborné práce jsem se naučil ovládat jazyk VHDL a k přepisu návrhu do tohoto jazyka popisujícího hardware bylo z důvodu optimalizace a uspokojivých výsledků nedílnou součástí pochopení logické syntézy pro vybrané FPGA (Xilinx Spartan-3e).

Po úspěšném otestování modulů obsluhujících periferie jsem znalosti z těchto návrhů uplatnil pro návrh samotné logiky hry. Postup práce probíhal dle očekávání a výstup každé jednotlivé části byl po otestování a případné úpravě uspokojivý. Konečným výstupem je tedy funkční číslicový logický obvod, který má funkčnost, jakou by měl mít podle jeho návrhu.

Dá se tedy říci, že všechny cíle práce byly splněny. Výslednou implementaci jsem testoval ovládáním dle její funkce. Po několika testovacích pokusech číslicový logický obvod pořád vykazoval korektní činnost.

Během práce jsem využíval verzovacího systému a celý projekt uvolňuji pod MIT licencí jako open-source a je k dispozici na stránkách GitHub [14].

Literatura

[1] G. Boole, *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities.* Dover Publications, 1854, ISBN: 0486600289.

[2] doc. Ing. Hana Kubátová CSc., *Struktura a architektura počítačů s řešenými příklady*, 2nd ed. Thákurova 1, 160 41, Praha 6: České vysoké učení technické v Praze, 2018, ISBN: 9788001064108.

[3] B. MELICHAR, *Jazyky a překlady.* České vysoké učení technické v Praze, ISBN: 80-01-01511-4.

[4] Bečvář, Daněk, Schmidt, Novotný, "Přechod mezi hodinovými doménami," Přednáška předmětu BI-PNO, 2006.

[5] Martin Novotný, "VHDL - Processes, signals and data types," Přednáška předmětu BI-PNO, 2006.

[6] Bečvář, Daněk, Schmidt, Novotný, "VHDL II - Automaty, typy, generický popis," Přednáška předmětu BI-PNO, 2006.

[7] ——, "Technologie návrhu a realizace číslicových obvodů," Přednáška předmětu BI-PNO, 2006.

[8] ——, "Struktura FPGA, RTL Syntéza, optimalizace," Přednáška předmětu BI-PNO, 2006.

[9] *Basys 2^{TM} FPGA Board Reference Manual*, Digilent.

[10] *ISE Design Suite Software Manuals and Help*, Xilinx.

[11] B. Govindarajalu, *IBM PC AND CLONES.* Mc Graw Hill India, ISBN: 9780070483118.

[12] "VGA," VESA, Beaverton, OR 97003, Standard, 1987.

[13] A. Klein, *Linear Feedback Shift Registers.* Springer, ISBN: 978-1-4471-5079-4.

[14] M. Přívozník, "Tuxman-fpga," https://github.com/MartinPrivoznik/TuxMan-FPGA, 2019.

Přílohy

TuxMan.rar	Zdrojové kódy a bitstream
Images.rar	obrázky