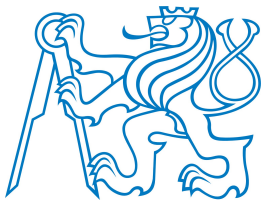
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra elektrických pohonů a trakce



Techniky subharmonické PWM

The subharmonic PWM Techniques

Diplomová práce

Studijní program: Elektrotechnika, energetika a management

Studijní obor: Elektrické stroje, přístroje a pohony

Vedoucí práce: Ing. Pavel Kobrle

Tomáš Košťál

Praha 2014

Oficiální zadání

Prohlášení

Prohlašuji, že jsem svoji diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW, apod.) uvedené v přiloženém seznamu.

V Praze dne 2. ledna 2014 .....

podpis

Poděkování

Děkuji zejména vedoucímu této bakalářské práce panu Ing. Pavlu Kobrlemu za mnoho cenných podnětů a rad při jejím zpracování.

Dále bych rád pak poděkoval panu Ing. Miroslavu Lvovi za zapůjčení vybavení.

Anotace

Tato práce shrnuje základní problematiku modulačních technik subharmonických šířkové pulzních modulací. Dále rozebírá problematiku virtuálního procesoru Microblaze. V praktické části popisuje realizaci modulátoru pro dvojčinný jednofázový, šestipulzní trojfázový tříúrovňový a pětiúrovňový měnič s ovládacícm rozhraním VGA a PS/2 pomocí programovatelného hradlového pole a procesoru Microblaze.

Annotation

This thesis summarizes the basics of subharmonic pulse width modulation techniques. Furthermore, it deals with Microblaze soft-core processor system. In the practical part of the thesis, realization of modulator for two level one phase, two level three phase, three level and five level multilevel converter, with VGA and PS/2 interface, within programmable gate array and Microblaze processor system.

Obsah

Seznam použitých zkratek

PWM – Pulse Width Modulation

OŘ – Obdélníkové řízení

DDS – Direct Digital Synthetisator

NCO – Numerically controlled Oscillator

LUT – Lookup Table

XPS – Xilinx Platform Studio

BRAM – Block Random Access Memory

XST – Xilinx Synthesis Tool

FPGA – Field Programmable Gate Array

UUT – Unit Under Test

BSB – Base systém Builder

XPS – Xilinx Platform Studio

HDL – Hardware Description Language

VHDL – VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

LSB – Least Significant Bit

MSB – Most Significant Bit

VGA – Video Graphics Array

TFT – Thin Film Transistor

ELF – Executable and Linkable Format

CB-PWM – Carrier Based Pulse Width Modulation

PSCPWM – Phase-shifted carrier PWM

1 Úvod

Elektrická energie provází v moderní společnosti člověka takřka na každém kroku. Ať už prostřednictvím domácích spotřebičů, osvětlení, dopravních prostředků, lékařské či sdělovací a výpočetní techniky, současný životní styl by byl bez ní nemyslitelný. Toto široké rozšíření využívání elektrické energie je dáno jejími nespornými výhodami, mezi něž patří například možnost ji transformovat na libovolnou jinou formu energie, relativně snadný a spolehlivý přenos, možnost výroby z různých jiných forem primární energie nebo čistota v místě užití.

Široká škála aplikací ovšem vedla k potřebě, transformovat parametry elektrické energe (například napětí, kmitočet, počet fází) na hodnoty vhodné pro daný spotřebič. Každá transformace energie s sebou ale nevyhnutelně přináší i ztráty. Vývoj výkonové elektroniky v posledních desetiletích umožnil konstruovat stále účinnější měniče energie a nakládat tak s energií mnohem hospodárněji, což je v současné době klíčová otázka pro udržitelný rozvoj společnosti.

Měniče elektrické energie jsou dnes založeny na výkonových spínacích součástkách, jejichž charakteristickou vlastností jsou dva stavy: plně propustný a plně uzavřený. Součástka sama není schopna žádnou transformaci energie provádět, pokud není řízeno, jakým způsobem má být spínána. A právě řízení je jednou z klíčových vlastností účinnější transformace elektrické energie moderními měniči.

Tato práce se proto zaměří na řízení měničů pomocí šířkově pulzní modulace, na které je díky jejím vlastnostem postavena současná výkonová elektronika. Zejména budou probrány techniky užívající subharmonickou šířkově pulzní modulaci, se zaměřením na její provedení pro víceúrovňové měniče, které jsou v současnosti slibně se rozvíjející, perspektivní technologií. Dále budou popsány možnosti programovatelných hradlových polí, jakožto rychlé a přesné platformy pro realizaci modulačních technik. Popis nevynechá ani problematiku virtuálních procesorů, která s hradlovými poli úzce souvisí. Bude popsán virtuální mikroprocesor Microblaze, který pro svá hradlová pole dodává firma Xilinx.

V další části bude popsána realizace modulátoru pro víceúrovňový měnič, která bude využívat popisu součástí pomocí HDL a jejich implementaci v hradlovém poli s procesorovým systémem Microblaze. K demonstrování širokých možností takové platformy bude provedeno uživatelské rozhraní modulátoru pomocí klasického PC monitoru a klávesnice s rozhraním PS/2. Realizace bude provedena pomocí hradlového pole Spartan3AN osazeného na vývojové desce Spartan3AN Starter Kit od společnosti Digilent.

2 Modulace pro polovodičové měniče

Shrnout vyčerpávajícím způsobem problematiku modulací pro polovodičové měniče je úkol velmi náročný, protože různé modulace ve výkonové elektronice jsou předmětem značného zájmu mnoha výzkumníků a pracovišť. Díky tomuto překotnému vývoji se zatím neustálila ani terminologie a to ani u poměrně základních pojmů. Zejména v anglofonní literatuře se vyskytuje bezpočet různých výrazů pro totéž nebo se naopak používá jeden výraz pro více jevů. Vzniklo a stále vzniká velké množství písemných materiálů zabývajících se touto problematikou, které přinášejí nové a nové postupy či jejich zlepšení a tak dále znepřehledňují situaci, což konstatují i renomovaní autoři, kteří se v obsáhlých monografiích snaží tuto problematiku shrnout (např. [1]).

2.1 Co je to modulace

Modulace je proces, kterým je měněna některá vlastnost (či více vlastností) nosného signálu pomocí signálu modulačního, který obsahuje nějakou informaci. Tento proces se používá k přenesení informace v prostředí, kde nelze informaci přenést samotným modulačním signálem. S modulacemi nejrůznějších druhů se setkáváme nejen ve sdělovací technice, ale i ve výkonové elektronice, kde se používá k přenášení informace o žádaném průběhu napětí či proudu z řídících obvodů na výkonové spínací prvky. Nyní definujme některé důležité pojmy, týkající se problematiky modulací:

modulační signál – signál, který chceme namodulovat na nosný signál

nosný signál – signál, který modulujeme modulačním signálem

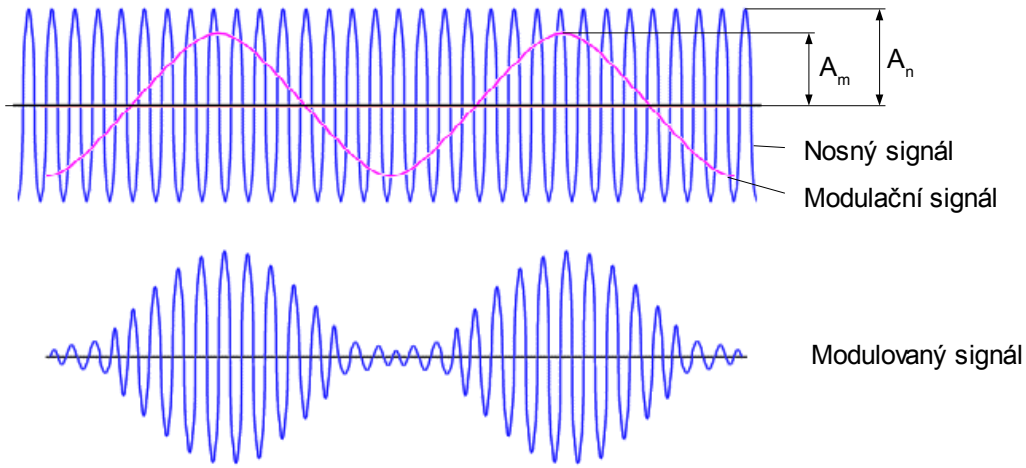
modulovaný signál – výsledný signál po procesu modulace

modulační index, hloubka modulace – ohledně těchto termínů panuje nejednoznačnost, viz kapitolu 2.2.1.

Uvedená vysvětlení mohou vyvolávat dojem cyklické definice, proto nejnázornějším způsobem bude ilustrovat tyto pojmy obrázkem, zpravidla se pro tyto účely používá amlitudová modulace viz obr.2.1.

2.1.1 Modulační index, hloubka modulace

Ohledně těchto dvou termínů panuje mezi autory, zabývajícími se problematikou modulací pro výkonovou elektroniku, značná nejednotnost. V některých pracích se používá termín *modulační index* (modulation index), zatímco v jiných *hloubka modulace* (modulation depth). Oba termíny jsou však různými autory různě definovány, někdy jako synonymum, jindy zcela jinak.



Obr. 2.1: Schematické zobrazení amplitudové modulace

Obecně se tvrdí, že velikost modulačního indexu (resp. hloubky modulace) udává, jak moc se mění velikost modulované veličiny oproti její nemodulované velikosti; tj. poměr modulované a nemodulované veličiny. Právě to ovšem dává prostor k nejednoznačnosti, neboť veličinu, ke které budeme modulační index vztahovat, si můžeme zvolit libovolně. V případě dopředných schémat pulzně šířkových modulací (viz dále) se utvořily dvě názorové větve. Obě v souvislosti s modulačním indexem (či hloubkou modulace) uvažují velikost amplitudy výstupního napětí, ovšem rozcházejí se ve volbě vztažné veličiny.

Prvním způsobem je vztažení velikosti amplitudy základní harmonické výstupního napětí k napětí, dosažitelnému při obdélníkovém řízení (OŘ, v anglické literatuře se užívá termín square wave inverter voltage). Tento způsob uvádí např. [2]. Označíme-li takto definovaný modulační index jako m_i, pak pro OŘ bude jeho hodnota $m_i = 1$. Pro asynchronní pulzně šířkovou modulaci (viz dále) pak bude nejvyšší dosažitelná hodnota

$$m_i = \frac{\frac{\sqrt{3}}{2}}{\frac{2\cdot\sqrt{3}}{\pi}} = \frac{\pi}{4} = 0{,}785 \tag{2.1}$$

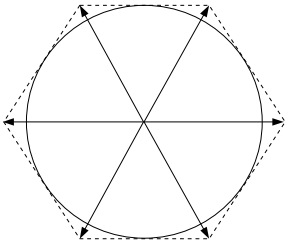
kde hodnota ve jmenovateli je velikost amplitudy základní harmonické výstupního napětí při OŘ

$$U_{OŘ} = \frac{2\cdot\sqrt{3}}{\pi} U_d \tag{2.2}$$

kde U_d je napětí na filtru daného měniče (předp. šestipulzní dvojhladinové zapojení napěťového střídače) a hodnota v čitateli U_P je maximální hodnota vygenerovatelného napětí při použití asynchronní PWM.

$$U_{\mathrm{P}} = \frac{\sqrt{3}}{2} U_{\mathrm{d}} \tag{2.3}$$

Ta odpovídá poloměru kružnice, vepsané do šestiúhelníku tvořeného aktivními stavy OŘ, jak ukazuje obr.2.2.



Obr. 2.2: K modulačnímu indexu dle [2]

Druhým způsobem je vztažení amplitudy modulačního signálu k amplitudě signálu nosného. Modulační index je takto definován například v [3] a [5]. Stejně je tento termín použit i v [1] s poznámkou, že jako synonymum je užíván i pojem *hloubka modulace* (*modulation depth*). V [6] je použit pojem amplitudový modulační index. [4] stejnou definici používá pro pojem hloubka modulace.

Aby nedocházelo k záměně, budu v této práci dále používat termín *hloubka modulace* m_{h} ve významu, v jakém je definována v [4] (s termínem hloubka modulace ve významu, jak je modulační index definován v [2] jsem se v literatuře nesetkal, toto označení by tak záměně mělo předejít zcela jistě), tedy

$$m_{\mathrm{h}} = \frac{A_m}{A_n} \tag{2.4}$$

kde A_{m} (v literatuře často jako A_{r} označující termín *reference signal*) je amplituda modulačního signálu a A_{n} (často jako A_{c} podle *carrier signal*) amplituda signálu nosného (obr. 2.2).

2.1.2 Synchronní a asynchronní modulace

Jako *synchronní* označujeme takovou modulaci, kdy modulační algoritmus je nějakým způsobem svázán s výstupním kmitočtem. U *asynchronní* modulace je spínací kmitočet konstantní, tedy asynchronní ke generovanému výstupnímu kmitočtu. Výhodou asynchronní modulace je stále stejné využití spínací schopnosti výkonových prvků v celém rozsahu generovaného kmitočtu. Za nevýhodu lze považovat měnící se počet sepnutí na jednu periodu výstupního průběhu. Při nízkém výstupním kmitočtu připadá na jednu periodu výstupního průběhu mnohem vyšší počet sepnutí než při vysokém výstupním kmitočtu. Může to vést až k tomu, že při vyšším výstupním kmitočtu již nebude moci na periodu výstupního průběhu připadnout dostatečný počet sepnutí,

aby nedošlo k výraznému zkreslení výstupního průběhu. Tyto nepříznivé jevy asynchronní modulace při vyšších kmitočtech odstraňuje modulace synchronní. Za další výhodu synchronní modulace je považován snazší přechod do obdélníkového řízení (z něj tato modulace v podstatě vznikne vkládáním mezer) při nejvyšších výstupních kmitočtech. Za nevýhody synchronních modulací je považován složitější přechod mezi různými kmitočty a také nevyužití spínacích vlastností součástek při nižších frekvencích [2].

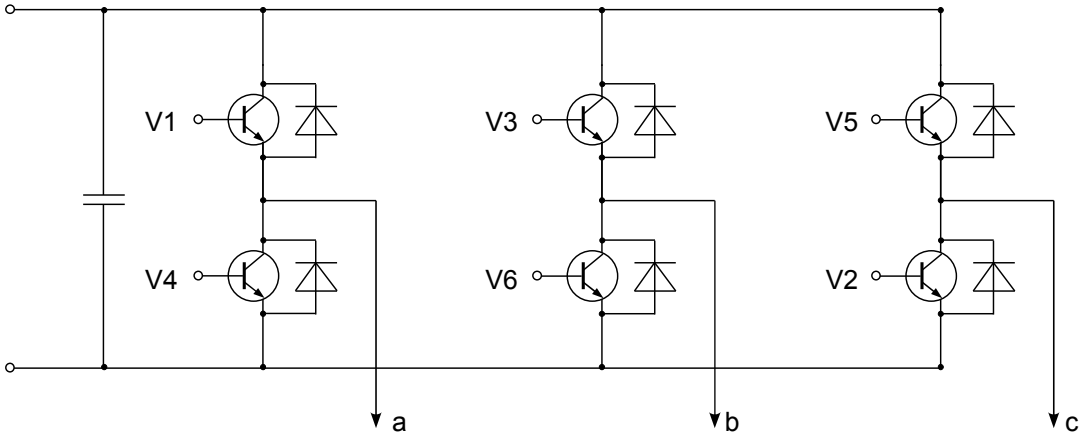
2.2 Dělení modulací pro výkonové měniče

V [2] je zavedeno následující dělení modulací pro výkonové měniče:

- Dopředná schémata (Feedforward Schemes) – modulační algoritmus pracuje bez zpětné vazby, je naprogramován dopředu.
  - PWM s nosnou vlnou (Carrier-Based PWM) – nevýhodou těchto modulací je vyjádřený dominantní kmitočet v harmonické skladbě generovaného výstupu, daný spínacím kmitočtem u asynchronních modulací a násobkem generovaného kmitočtu u synchronizovaných modulací Rozlišují se tři podtypy:
    - Suboscilační metoda (Suboscillation nemo též Subharmonic Method)
    - Modulace prostorového vektoru (Space Vector Modulation – SVM)
    - Synchronizovaná nosná modulace (Synchronized Carrier Modulation)
  - PWM bez nosné vlny (Carrierless PWM) – důvodem použití je snaha o potlačení dominantních kmitočtů v harmonické skladbě
- Zpětnovazebná schémata (Feedback PWM-Control) – zadaná hodnota je porovnávána se skutečnou, výsledek tohoto porovnání přímo ovlivňuje spínací pulzy a regulovaná veličina se tak pohybuje v zadaném tolerančním pásmu. Mezi nejčastější druhy patří:
  - Hysterezní proudová regulace
  - Suboscilační proudová regulace
  - Regulace proudového prostorového vektoru
  - Proudová regulace v souřadnicích pole
  - Metoda lookup table
  - přímá regulace toku či momentu

2.2.1 Obdélníkové řízení

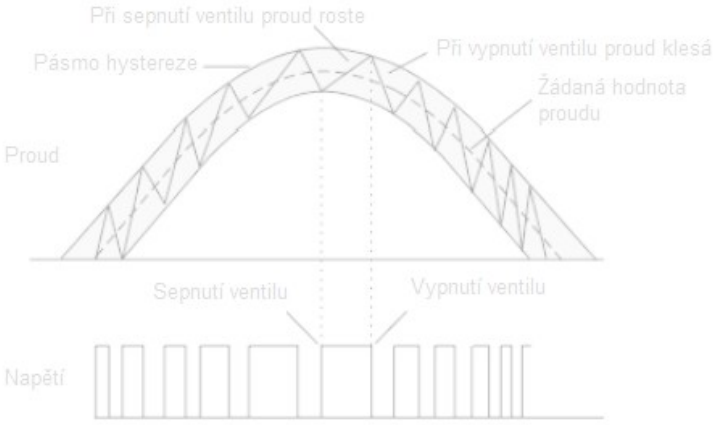
Obdélníkové řízení (OŘ) je způsob řízení, který se používá například pro dvouhladinové šestipulzní zapojení napěťového střídače (obr. 2.3). Jedná se o nejjednodušší spínací algoritmus, spínání je velmi jednoduché a tak nelze měnit amplitudu základní harmonické výstupního napětí. Více o obdélníkovém řízení lze nalézt například v [2]nebo [19].



Obr. 2.3: Dvojhladinový šestipulzní střídač

2.2.2 Pulzně šířková modulace

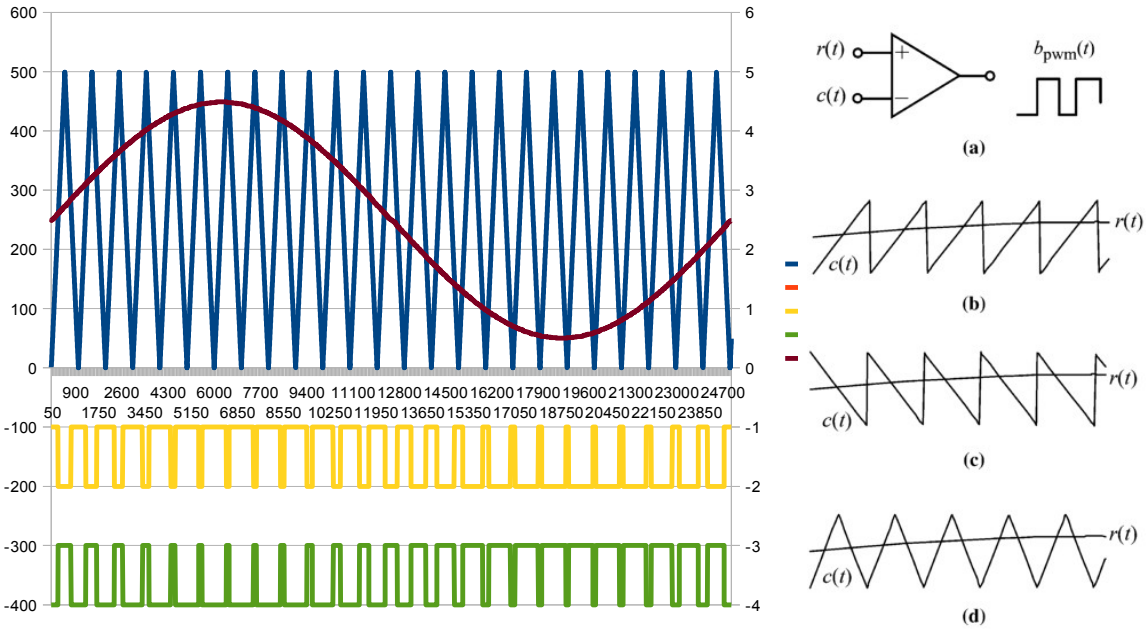
V současné době představuje pulzně šířková modulace nejčastější způsob řízení napěťových střídačů, umožňující současnou změnu výstupního kmitočtu a základní harmonické výstupního napětí. [2] Pulzně šířková modulace vzniká vkládáním mezer do obdélníkového řízení. Na obr. 2.4 je znázorněn základní princip. Je patrné, že průběh proudu obsahuje mnohem vyšší podíl základní harmonické než u OŘ. Realizace PWM předpokládá použití vypínatelných součástek (IGBT, IGCT, GTO dříve též klasické tyristory s vhodnými komutačními obvody).Pulzně šířková modulace je pro řízení dnešních spínacích (zejména výkonových tranzistorů IGBT) součástek naprosto ideální. Výkonový tranzistor je totiž konstruován tak, že má dva stavy s minimálními ztrátami. V plně otevřeném stavu je na tranzistoru minimální úbytek napětí (je na něm pouze saturační napětí), přes plně vypnutý tranzistor naopak neprochází téměř žádný proud. Přechody mezi těmito dvěma stavy se snažíme minimalizovat. Těmto charakteristickým vlastnostem skvěle odpovídá právě pulzně šířková modulace, která je dvoustavová a přechod mezi oběma stavy trvá v ideálním případě nulový čas. PWM tedy umožňuje přenášet signál v dvouhodnotové modulaci, což vedlo k tomu, že je dnes jednou ze základních technik, na kterých je postavena současná výkonová elektrotechnika. Termínem střída (zatěžovatel) označujeme poměrnou dobu setrvání signálu v horní úrovni v rámci jedné periody.



Obr. 2.4: Princip pulzně šířkové modulace

2.3 Subharmonická PWM

Jak již bylo napsáno v kapitole 2, existuje v terminologii mnoho nejednotností. To se týká především názvů různých modulačních technik. Pro účely této práce tedy budeme chápat *subharmonickou šířkově pulzní modulaci* jako modulaci vzniklou porovnáváním nosného signálu (trojúhelníkovitého či pilovitého) s modulačním signálem (nejčastěji sinusového tvaru). Tyto charakteristiky jsou vystiženy například v označení „*Sinusoidal PWM*“ [5] (též zkratkou SPWM). Dalšími názvy, které se pro tento druh modulací užívají jsou: „*suboscillation PWM*“ (suboscilační PWM) či „*triangulation PWM*“. Název „*Subharmonic PWM*“ používá mimo jiné [6].



Obr. 2.5: Základní princip PWM, a) princip analogové realizace, b)trailing edge, c)leading edge, d)double edge

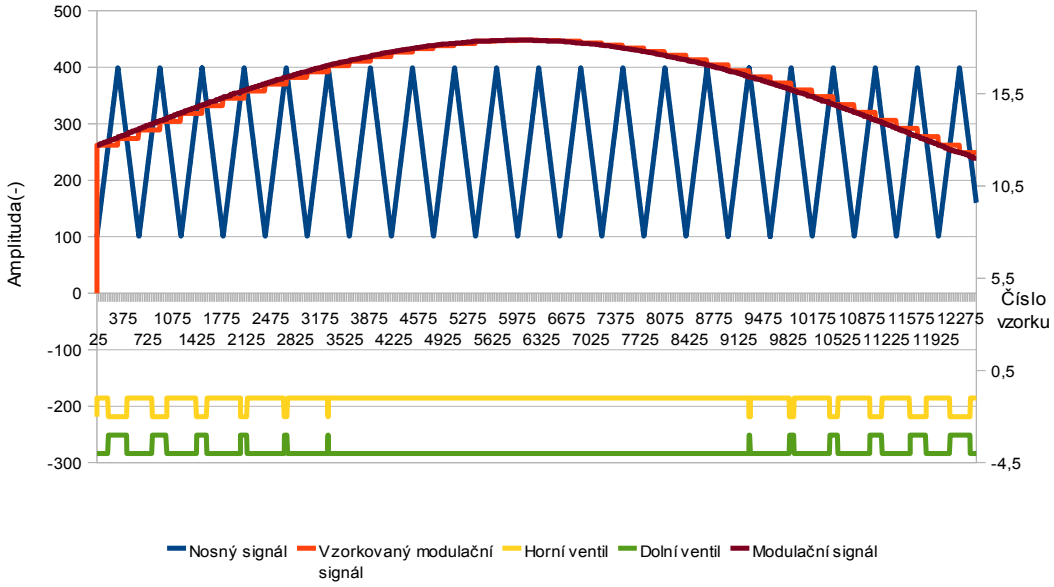
Základní princip subharmonické PWM je na obr. 2.5. Takovou modulaci můžeme zapsat vztahem:

$$b_{\text{pwm}}(t) = \text{sgn}[\ r(t) - c(t)\ ] \tag{2.5}$$

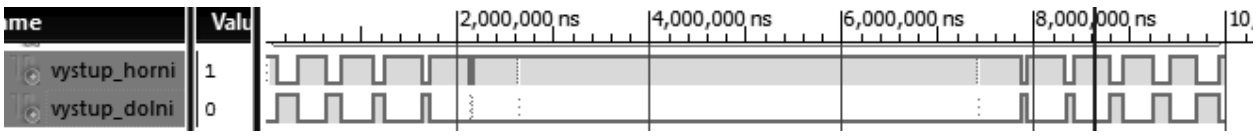
kde *r(t)* je modulační signál (*reference*) a *c(t)* signál nosný (*carrier*)

2.3.1 Přemodulování

K přemodulování dochází v případě, že hloubka modulace m_h se blíží jedné či $m_h > 1$. Na obr. 2.6 je zobrazen extrémní případ přemodulování, kdy amplituda modulačního signálu je výrazně vyšší než amplituda nosného. V takovém případě se nosný signál s modulačním nikdy nepotkají a nedojde tak vygenerování některých spínacích pulzů. Někdy se tento jev využívá záměrně k dosažení vyšší amplitudy výstupního napětí, ovšem je třeba počítat s tím, že tento postup výstupní signál také zkresluje. K podobnému problému může dojít i v případě, že m_h je blízký jedné. Pak délka vygenerovaného spínacího pulzu může být kratší než zadaná ochranná doba a pulz se tedy nevygeneruje.Tomuto stavu je blízký průběh na obr. 2.16. Obr. 2.7 ukazuje tutéž simulaci v zobrazení přímo v ISIM.



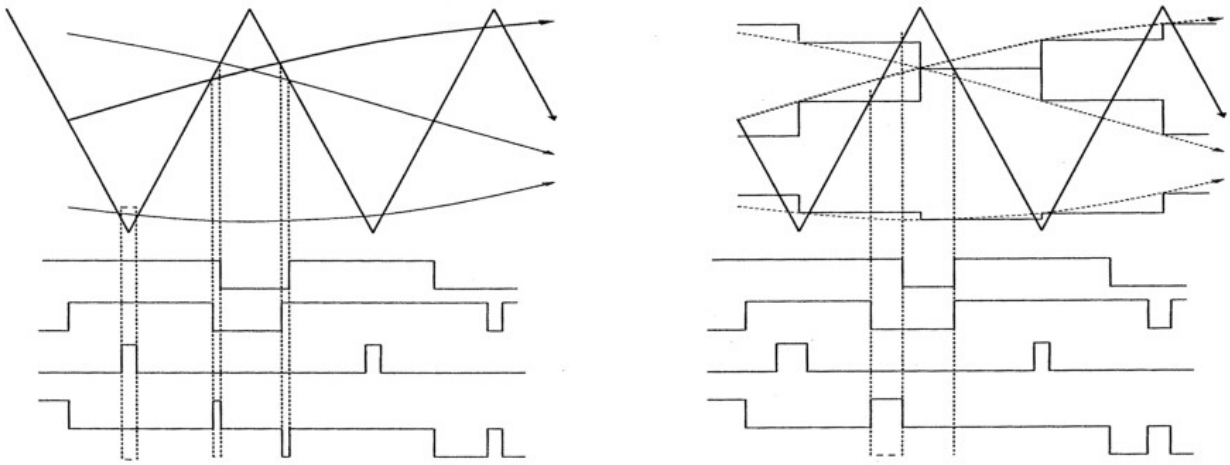
Obr. 2.6: Přemodulování



Obr. 2.7: Přemodulování - ISIM

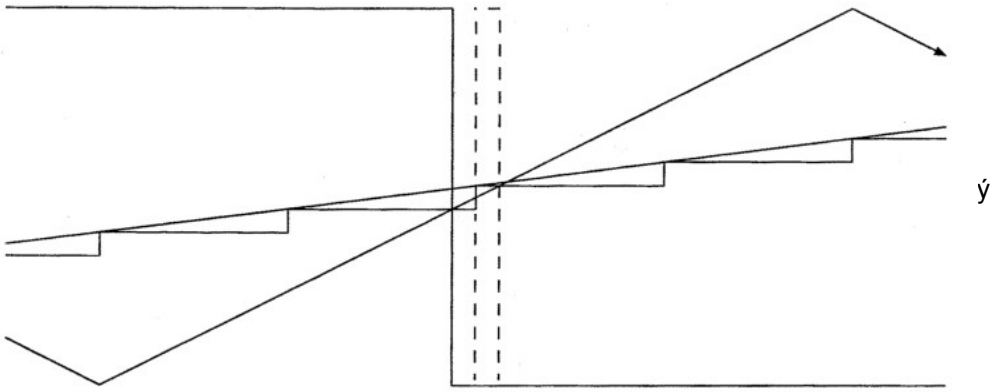
2.3.2 Naturally sampled PWM, Regularly sampled PWM

V anglické literatuře označuje termín *Naturally sampled PWM v* podstatě analogovou realizaci principu na obr. 2.5., tedy že spínací pulz je vygenerován okamžitě při protnutí modulačního a nosného signálu. Termínem *Regularly sampled PWM* (též *uniformly sampled*, např. [4]) je označována realizace číslicová. Předpokládá totiž kvantování v amplitudě a vzorkování v periodě, tedy realizaci pomocí diskrétních hodnot. Zejména u modulačního signálu bývá krok amplitudy a periody velký, protože tyto hodnoty musí být zpravidla uloženy v tabulce, jejíž velikost je omezena dostupnou pamětí. Princip je ukázán na obr. 2.8.



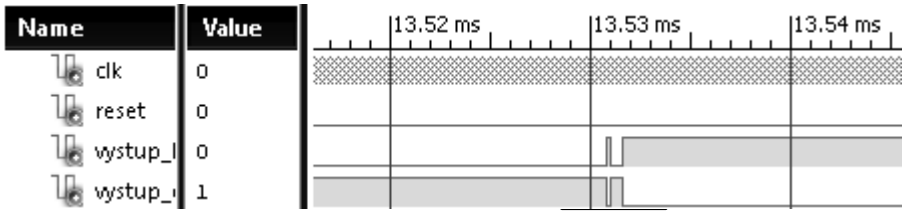
Obr. 2.8:Natural Sampling vlevo a Regular sampling vpravo. Vpravo je rovněž patrné zpoždění pulzu, kterým je pravidelné vzorkování provázeno [4]

Číslicová realizace s sebou přináší některé problémy, vyplývající z její diskrétní povahy. Možný problém ukazuje obr. 2.9. Zde se hodnota modulačního signálu skokově změní krátce po okamžiku, kdy amplituda nosného signálu již její velikost přesáhla a došlo tak k pokynu zařadit na výstup mezeru. Po zvětšení amplitudy modulačního signálu bylo ovšem dosaženo stavu, který odpovídá zařazení pulzu. V krátké době na to ale amplituda nosného signálu amplitudu modulační opět překonává, tady se má zařadit na výstup mezera. Takto vygenerovaný krátký



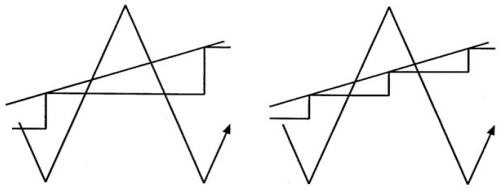
Obr. 2.9: Vícenásobné sepnutí při nedodržení vzorkování v úvrati nosného signálu

pulz, pokud je delší než nastavená ochranná doba a skutečně se tak dostane na výstup, nejenže zhoršuje harmonickou skladbu výstupního průběhu, ale také zbytečně namáhá spínací součástku (spínací ztráty).



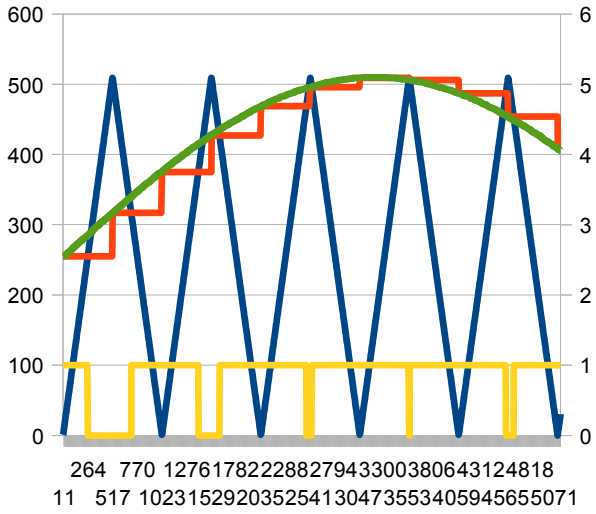
Obr. 2.10: Vícenásobné sepnutí zachycené při simulaci v ISIM

Jako ochrana proti tomuto jevu se zavádí vzorkování modulačního signálu pouze v úvrati nosného signálu (myšleno v okamžiku, kdy trojúhelník mění směr a pila se nuluje) a hodnota z tohoto bodu se ponechá pro komparaci po celou dobu dobu periody nosného. U trojúhelníkovitého signálu lze toto *vzorkování* provádět i každou půlperiodu (tedy v horní i dolní úvrati), takové modulaci se pak říká *„asymmetrical double edge regularly sampled PWM“* [14]. Slovo „*asymmetrical*“ odráží skutečnost, že „vzorek“ není u této modulace symetrický po celou dobu periody podle svislé osy trojúhelníku (obr. 2.11 vpravo). Pokud se vzorkuje jen v dolní úvrati (nebo naopak jen v horní úvrati, což je ekvivalentní [1]) hovoříme o „*symmetrical double edge regularly sampled PWM“* (obr. 2.11 vlevo).

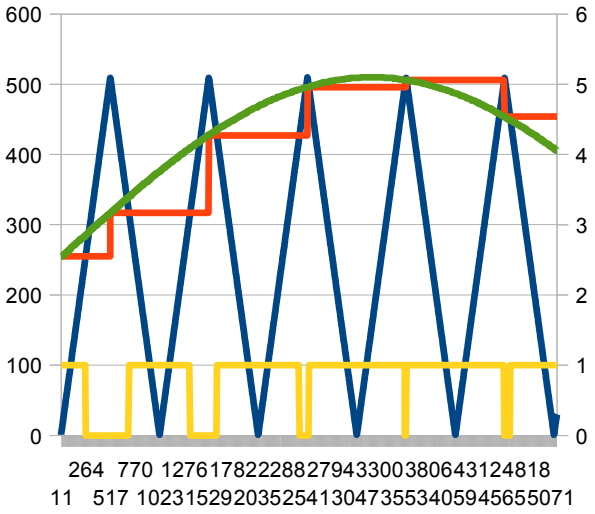


Obr. 2.11: Symmetrical a asymmetrical double edge PWM [4]

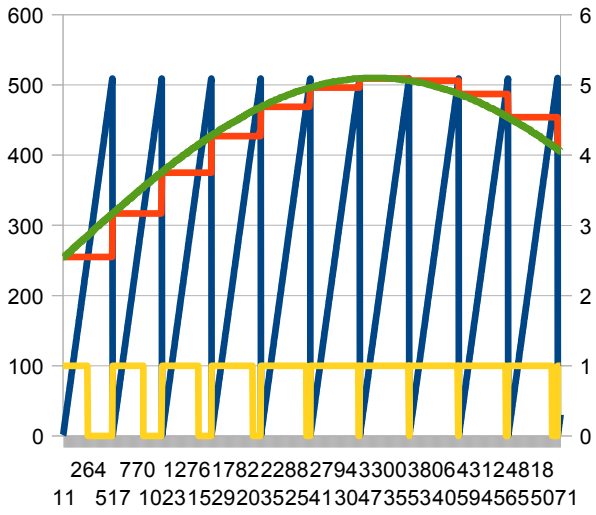
Je-li nosným signálem pila (obr. 2.5b) (*sawtooth*) nebo obrácená pila (obr. 2.5c) (*inverse sawtooth*), hovoříme o „*single edge regularly sampled PWM*“. Pro rozlišení se doplňují buď přízviskem „*sawtooth*“ a „*inverse sawtooth*“, nebo se jim také říká „*leading-edge regularly sampled PWM*“ (pokud je signálem pila) a „*leading-edge regularly sampled PWM*“ (signálem obrácená pila). To je odvozeno od skutečnosti, že kolmá hrana pilovitého signálu se protíná s modulačním signálem vždy na začátku (konci) své periody, a jedna hrana spínacího pulzu se tak i při změně střídy neposunuje (obr. 2.14 a 2.15). To tyto dvě modulace odlišuje od „*double edge*“ modulací, kde se posunují obě hrany pulzu (odtud označení double-edge). U „*symmetrical*“ je spínací pulz navíc centrovaný kolem středu periody (obr. 2.13), což přináší jisté výhody v oblasti harmonické skladby. U „*asymmetrical*“ k centrování nedochází, pulz může být jinak na každou stranu od středu (jak je patrno na obr. 2.12).



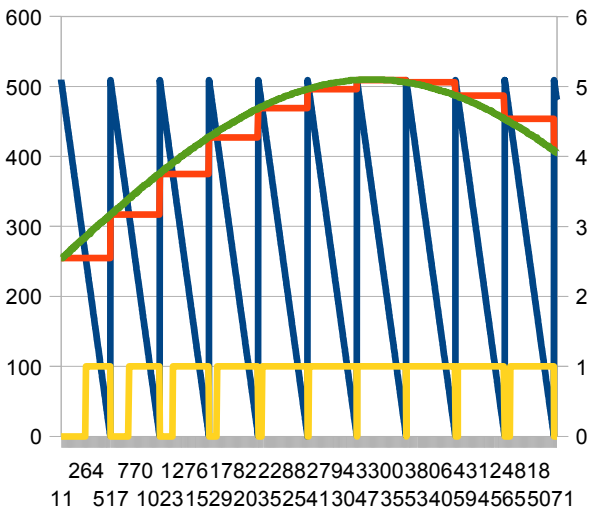
Obr. 2.12: Double edge asymmetrical



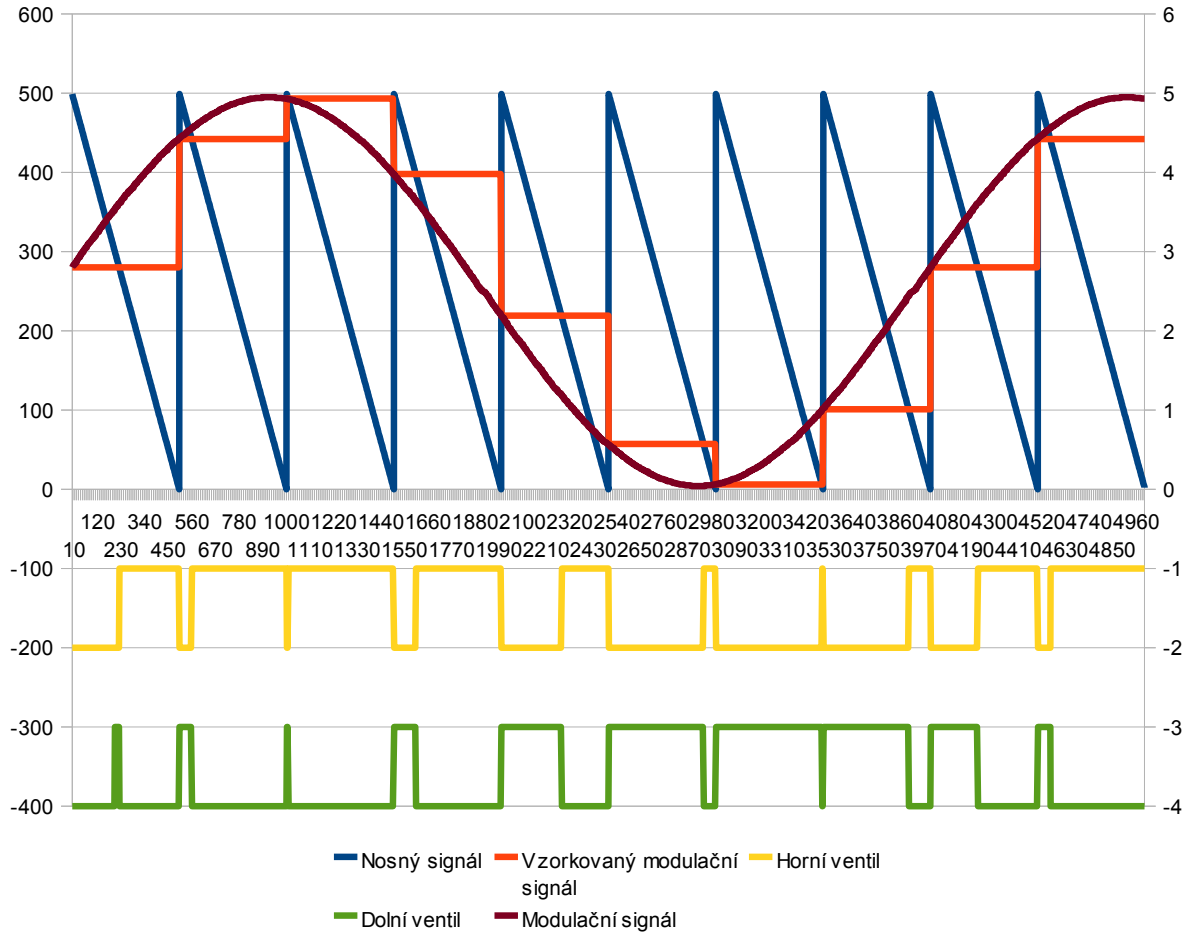
Obr. 2.13: Double edge symmetrical



Obr. 2.14: Sigle edge – rising edge (sawtooth)



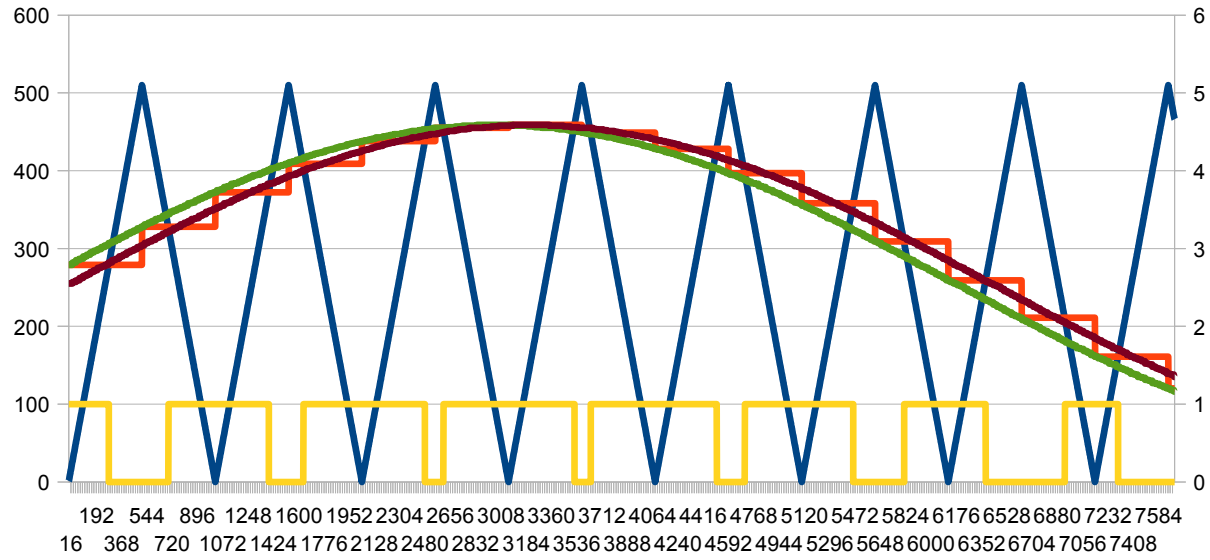
Obr. 2.15: Trailing edge (inverse sawtooth)



Obr. 2.16: Průběh jedné fáze při nízkém kmitočtu nosného signálu (leading edge).

2.3.3 Phase advancing

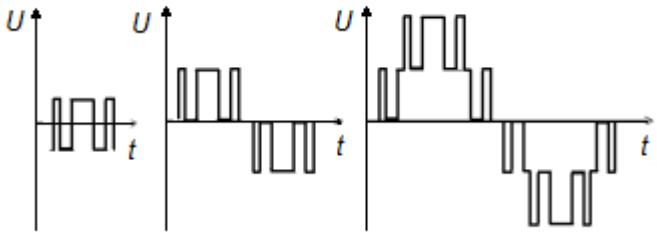
Při pozorném zkoumání průběhů musíme dospět ke zjištění, že opatření proti generování nepatřičných pulzů vedlo vedlo k fázovému posunu modulačního průběhu. Střední hodnota navzorkovaného průběhu je jiná než signál, z nějž bylo vzorkováno. Tento problém se řeší dopředným posuvem modulačního signálu (obr. 2.17). Střední hodnota navzorkovaného pak odpovídá žádané hodnotě modulačního, což se projeví i na výstupním průběhu. Tomuto postupu se říká „*phase advanced reference*“ nebo též „*phase advancing*“ [1]. Posun je třeba provést u „*asymmetrical double edge*“ PWM o čtvrtinu periody nosného signálu, u ostatních o polovinu [1], což vyplývá z povahy časové délky vzorku („*asymmetrical*“ vzorkuje při stejném kmitočtu dvojnásobně často než ostatní techniky).



Obr. 2.17: Phase advancing

2.4 CB – PWM pro víceúrovňové měniče

Víceúrovňové měniče elegantně řeší některé problémy výkonové elektroniky, jako například rozdělení napětí na jednotlivých součástkách při vysokonapěťových aplikacích. Jejich další velkou výhodou je oproti „klasickým“ dvouúrovňovým měničům příznivější harmonické složení výstupního napětí. To je dáno právě možností připojit na výstup více napěťových hladin a ne pouze dvě (kladný a záporný pól filtru) jako u „klasických“. Víceúrovňové měniče si v poslední době nacházejí stále častěji cestu i do nižších výkonových oblastí. Zvlášť oblíben je tříúrovňový měnič s diodami, který získává třetí úroveň připojením středu filtru (*NPC – Neutral Pint Clamped*) sestaveného ze dvou kondenzátorů. [2]



Obr. 2.18: Fázové napětí měničů (zleva): dvouúrovňového, tříúrovňového a pětiúrovňového

Mezi nejrozšířenější topologie víceúrovňových měničů patří kaskádové zapojení můstků (*Cascaded H-bridge*), víceúrovňový měnič s diodami (*Diode Clamped Multilevel Inverter*) a víceúrovňový měnič s plovoucími kondenzátory (Flying Capacitor Multilevel Inverter) [1]. Na obr. 2.22 je znázorněno schema pětiúrovňového měniče s plovoucími kondenzátory. Měnič tohoto typu je vyvíjen na naší katedře, proto je logické se zaměřit směrem, který by mohl být

uplatněn pro řízení takovéhoto měniče. Více o víceúrovňových měničích se lze dozvědět například v [1].

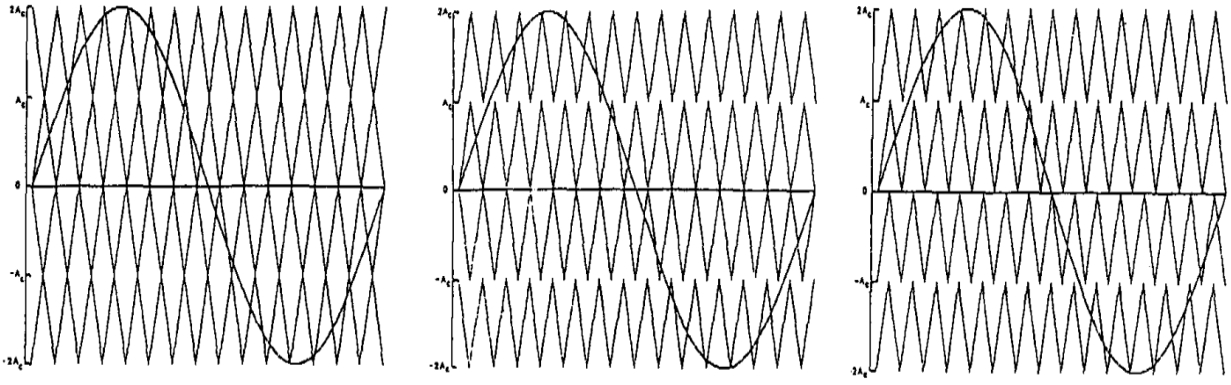
K řízení víceúrovňových měničů se velmi často používají techniky CB-PWM, tzv. „*multi-carrier PWM*“. Vycházejí se z předpokladu, že pro *n* úrovňový měnič je třeba *n-1* nosných signálů. Počet modulačních signálů závisí na počtu fází. To platí i pro měnič dvouúrovňový, kde jsou dvě úrovně napětí a jeden nosný signál.

Nosné signály mají buď zmenšenou amplitudu (n-1 krát) a vůči modulačnímu signálu se řadí do pásem „na sebe“, což první publikoval [30] nebo se nosné signály fázově posouvají. [1] U první možnosti rozlišujeme tři základní schemata:

Altenative phase opposition disposition (APOD) - nosné signály v sousedních pásmech jsou vůči sobě posunuty o 180°

Phase oppositopn disposition (POD) - nosné signály nad nulovou úrovní modulačního signálu jsou vůči těm pod nulvou úrovní posznuty o 180°

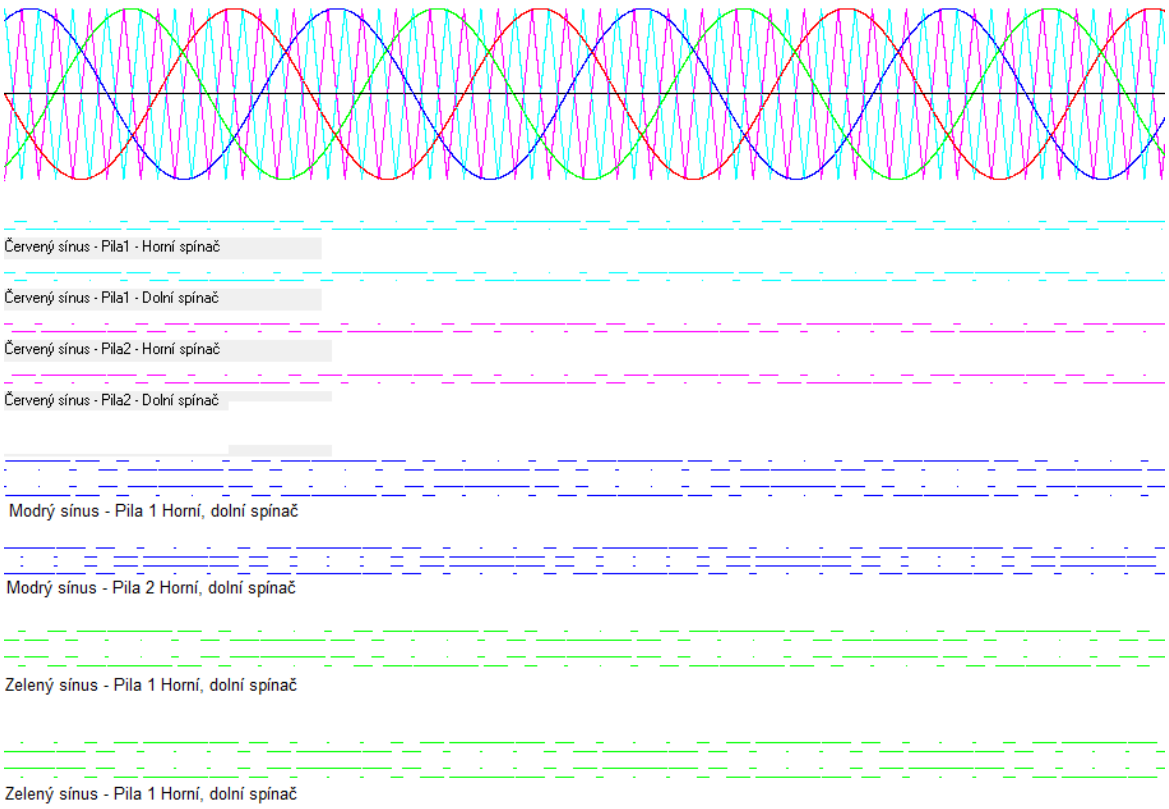
Phase disposition (PD) - všechny nosné signály jsou ve fázi



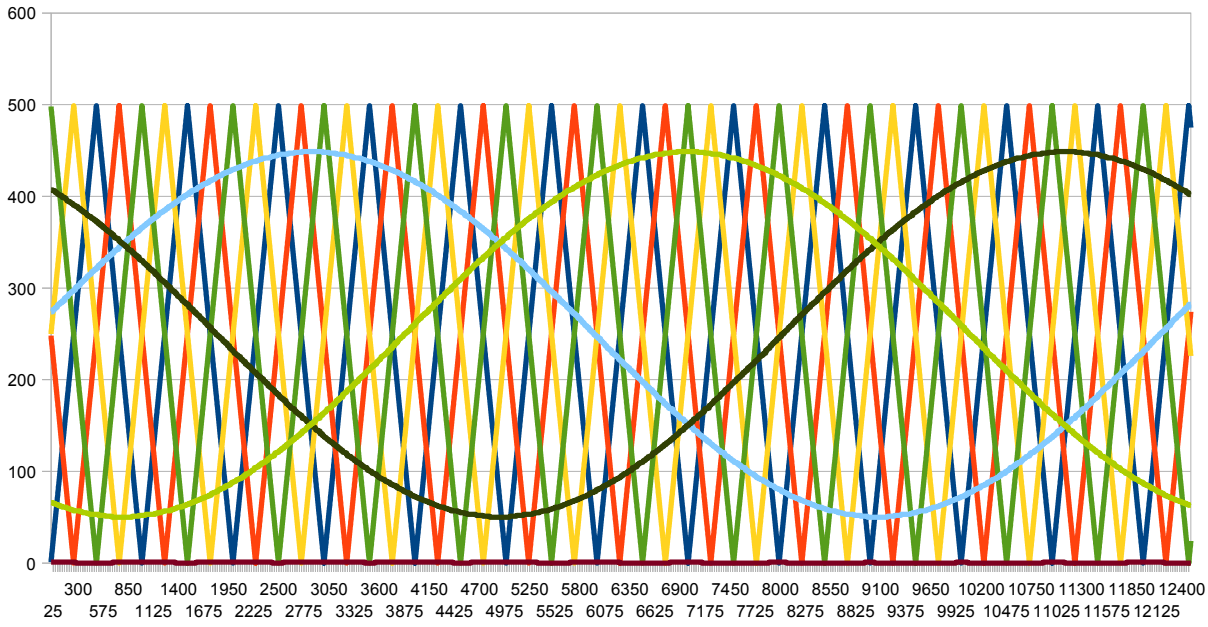
Obr. 2.19: Zleva_ APOD, POD a PD

U druhé možnosti, fázově posunutých nosných signálech s plnou amplitudou, hovoříme o „*Phase-shifted carrier PWM*“ (*PSCPWM*). Pro účely vývoje jsem v prostředí Microsoft Visual Basic 6.0 sestavil animovanou vizualizaci průběhu nosných a modulačních signálů pro tříúrovňový měnič (obr. 2.20). Obr. 2.21 ukazuje průběhy pro měnič pětiúrovňový.

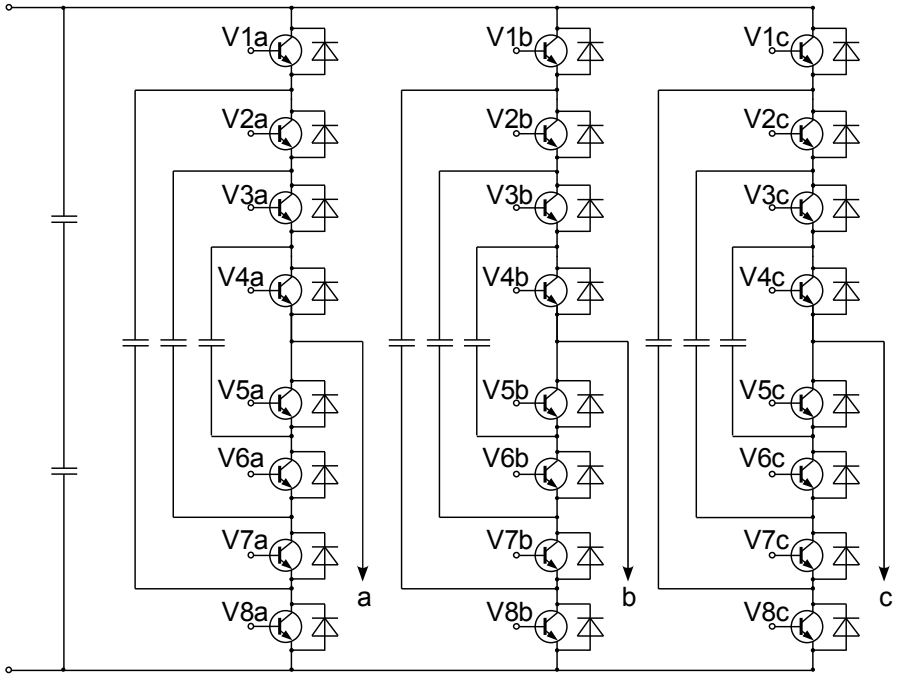
Pro spojení s ventily měniče u PSCPWM technik platí, že čím posunutější nosný signál, tím dále od středu (vývodu fáze) jedné větve měniče se nacházejí [6].



Obr. 2.20: Vizualizace průběhů nosných a modulačních signálů pro trojúrovňový měnič včetně pulzů pro ventily



Obr. 2.21: Průběhy modulačních a nosných signálů pro pětiúrovňový měnič, technika double edge regularly sampled PWM



Obr. 2.22: Pětiúrovňový měnič s plovoucími kondenzátory

2.5 Kritéria hodnocení modulací

V [2] jsou pro hodnocení jednotlivých modulačních technik uvedena následující kritéria: spínací podmínky, provozní podmínky a optimalizace modulací. Hodnotí dále nejen samotný princip modulace, ale i vlastnosti dané jejím technickým řešením v praxi.

2.5.1 Spínací podmínky

„Použitý modulační algoritmus musí zajistit především bezpečnou funkci výkonového polovodičového měniče. Základní bezpečnostní kritérium je zachování minimální doby sepnutí a minimální doby vypnutí“ [2]. Splnění těchto podmínek musí zajistit v první řadě právě modulační algoritmus, ačkoli měniče jsou zpravidla vybaveny ochrannými prvky, které by nebezpečné stavy nedovolily realizovat. Dalším potřebným prvkem je dodržení ochranné doby mezi vypnutím a sepnutím ventilu v jedné fázi.

Minimální dobu sepnutí je třeba dodržet kvůli fyzikálním vlastnostem součástek. Po sepnutí nepřechází součástka do vodivého stavu v nulovém čase, ale postupně. Například u GTO tyristorů se po spínacím impulzu rozšiřuje ve struktuře vodivá oblast. Pokud by pokyn k vypnutí přišel před plným rozšířením této vodivé plochy, uzavíral by se vypínací proud menší plochou než za normálních okolností a součástka by byla tepelně přetěžována.

Minimální dobu vypnutí je třeba dodržovat z důvodů, aby se spínací součástka plně dostala do vypnutého stavu. Oblast prostorového náboje v součástce se plně odplaví až po uplynutí určitého času.

Spínací kmitočet je třeba volit nejen s ohledem na vlastnosti použitých spínacích součástek, ale také na možné problémy s elektromagnetickou kompatibilitou. Obecný trend směřuje ke zvyšování spínacích frekvencí, protože to vede k menšímu zvlnění výstupního proudu. Spínací kmitočty výkonových prvků se dnes pohybují řádově kolem stovek Hz u velkých GTO tyristorů a nad 1 kHz u IGBT tranzistorů.

2.5.2 Provozní podmínky

Mezi hlavní provozní podmínky na které je při hodnocení modulací brán zřetel zahrnuje [2] harmonickou skladbu proudu, zvlnění momentu (při použití v pohonu) a dobu odezvy na regulační zásah

Harmonická skladba proudu je velmi častým srovnávacím kritériem v mnoha pracích, které se problematikou modulací pro výkonové měniče zabývají [1],[5],[6],[14]. Často sledovaným cílem je dosáhnout co nejvyšší podíl základní harmonické ve výstupním průběhu a minimilizovat podíl ostatních harmonickcých. Některá modulační schémata naproti tomu záměrně vkládají do výstupního průběhu některé vyšší harmonické (nejčastěji třetí harmonickou, například za účelem dosažení vyšší amplitudy výstupního napětí).

Zvlnění momentu sledujeme za účelem dosažení příznivých mechanických vlastností pohonu, je-li modulace použita k řízení měniče pro pohon (může být též použita například pro aktivní filtr). V případě použití asynchronního stroje vyplývá například požadavek na nesinusový proud, protože žádaným stavem je průběh statorového magnetického toku po šestiúhelníku [2].

Doba odezvy na regulační zásah spočívá ve schopnosti rychlé reakce modulační techniky na změnu zadaných parametrů. K častým problémům se zpožděnou reakcí dochází například tehdy, je-li modulační technika realizována pomocí programu v řídícím mikropočítači. Vliv má jednak taktování procesoru a dále doba potřebná k vykonání všech potřebných instrukcí vedoucích ke změně stavu. Tyto problémy se řeší používáním speciálních periferií jako HSO (*High Speed Output*), EPA (*Event Processor Array*), EV (*Event manager module*) založené zpravidla na principu naprogramování událostí s časem vykonání do určitého seznamu, který se dále vykonává bez zásahu řídícího programu. Takováto řešení ovšem zase narážejí na problém dostatečně rychlé změny takto předprogramovaného seznamu akcí. Nejelegantnějším řešením realizace modulačních technik je použití hardwarové implementace pomocí FPGA, které výše zmíněnými problémy netrpí.

2.5.3 Optimalizace modulací

Z hlediska optimalizace modulací uvádí [2] zaměření na eliminaci obsahu vyšších harmonických složek v generovaném napětí či snížení jejich celkového obsahu, což úzce souvisí se spínacím

kmitočtem. Dále klade důraz na zvlnění výstupního průběhu, které by mělo být rovnoměrné bez výrazných špiček, protože na tyto špičky je nutné dimenzovat součástky měniče, což zbytečně zvyšuje jejich cenu. Příznivé situace lze dosáhnout off-line provedeným předvýpočtem optimálních spínacích subcyklů, které se aplikují podle okamžitého provozního stavu. Modulace lze tedy hodnotit i podle toho, jak snadno lze takové řešení realizovat.

2.6 Porovnání subharmonických PWM

Mnoho autorů zakládá porovnávání jednotlivých modulačních technik na harmonické skladbě [1], [5], [14]. Jak konstatuje [1] je situace v různých způsobech výpočtů krajně nepřehledná a její podrobnější rozbor by přesahoval rámec této práce. Nejčastějším faktorem, který se pro srovnání harmonické skladby používá, je *harmonické zkreslení THD* (*Total Harmonic Distortion*) a jeho poddruhy [1]. Princip tkví v porovnávání množství žádané harmonické (nejčastěji základní) a ostatních harmonických složek proudu. Vychází se z předpokladu, že každou periodickou funkci *v(t)* je možné popsat Fourrierovým rozvojem jako

$$v(t) = V_0 + V_1 \cos \omega_1 t + V_2 \cos \omega_2 t + V_3 \cos \omega_3 t + \ldots \tag{2.6}$$

potom

$$THD = \sqrt{\left(\frac{2V_0}{V_1}\right)^2 + \sum_{n=2,3,\ldots}^{\infty} \left(\frac{2V_n}{V_1}\right)^2} \tag{2.7}$$

Podíváme-li z hlediska harmonické skladby na výše probrané číslicově provedené subharmonické modulační techniky, vychází nejlépe „*symmetrical double edge PWM*“. Tato modulace je charakterizována modulovanými pulsy, centrovanými vzhledem k PWM periodě. Výhoda „*symmetrical PWM*“ oproti „*asymmetrical PWM*“ je, že „*symmetrical PWM*“ má 2 neaktivní oblasti se stejnou dobou trvání - na začátku a konci každé PWM periody. Tato symetrie způsobuje menší generaci vyšších harmonických.

Z hlediska doby odezvy na regulační zásah je nejvýhodnější „*asymmetrical PWM*“, protože při daném spínacím kmitočtu (kmitočtu nosného signálu) provádí vzorkování modulačního signálu dvakrát častěji.

Z pohledu spínacích dob vycházejí při stejném spínacím kmitočtu příznivěji double edge modulace, protože v nosný signál je v oblasti úvrati otevřenější. Single edge modulace mají výhodu v jednoušším provedení (v číslicové podobě jsou realizovatelné jednodušším čítačem).

3 Procesor microblaze

3.1 Programovatelná hradlová pole

Programovatelná hradlová pole – FPGA (Field Programmable Gate Array) jsou číslicové integrované obvody s proměnnou konfigurací, kterou lze libovolně naprogramovat. Sousloví „Field Programmable“ v anglickém názvu charakterizuje skutečnost, že FPGA obvod je vyráběn jako univerzální a pro konkrétní aplikaci je upraven (naprogramován) až u zákazníka (v terénu, v poli). Tyto obvody jsou mezičlánkem mezi klasickými mikropočítači a zákaznickými integrovanými obvody (ASIC – *Application-Specific Integrated Circuit*). Nacházejí uplatnění především v případech, kdy klasické mikropočítače nedostačují svými vlastnostmi, především rychlostí a periferiemi, a kdy by se výroba zákaznického obvodu ekonomicky nevyplatila. Jedná se tedy především o výrobu v menších sériích a vývoj prototypů.

Vnitřní struktura hradlového pole se skládá z programovatelných bloků, programovatelných matic spojů , vstupně-výstupních bloků a blokových pamětí RAM (označované jako BRAM). Propojováním těchto bloků lze vytvářet téměř libovolné logické funkce.

Hlavním prvkem progamovatelného bloku je tzv. *Lookup Table* (LUT). Ta obsahuje definici žádané logické funkce v podobě pravdivostní tabulky (odtud termín LUT) uložené v paměti. Tato pravdivostní tabulka popisuje chování obvodu, kterým by byla žádaná logická funkce realizována pomocí hradel. Tabulka je adresována hodnotou vstupů, které by odpovídaly vstupům takového obvodu a v odpovídajícím formátu je i její výstup.

Více o FPGA se lze dozvědět například v [12].

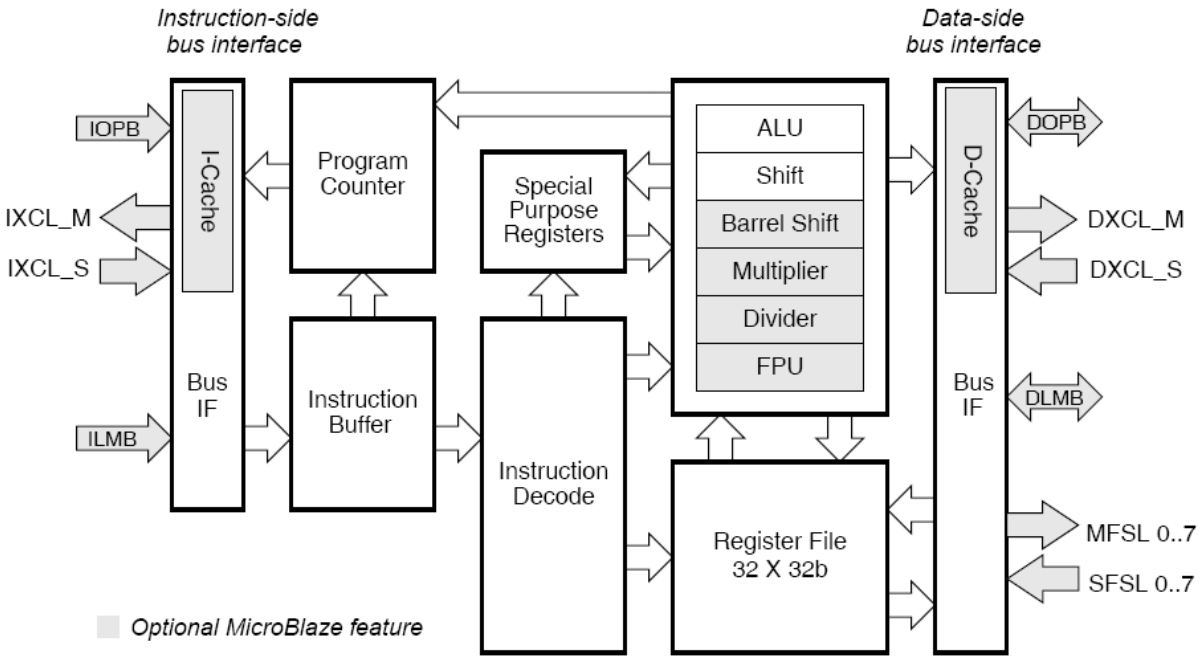
Pro mnoho aplikací ovšem nestačí pouze možnost vytvořit logické obvody vykonávající požadovanou funkci. V mnoha případech vyžaduje daná aplikace i použití procesoru. Velká výhoda hradlových polí tkví v možnosti rozdělit úkoly mezi hardwarové obvody a program procesoru. Na trhu jsou již řešení, kombinující procesorové jádro s hradlovým polem na jednom čipu. Pokud je potřeba použít procesor v hradlovém poli, které není dopněno procesorovým jádrem, přicházejí na řadu tzv. „soft proceory“. Soft procesor (též „soft mikroprocesor“ nebo „softcore mikroprocesor“) je fyzicky neexistující jádro procesoru, které je vytvořeno implementací na hradlovém poli pomocí popisu v některém HDL jazyce (Hardware Description Language). Pro hradlová pole Xilinx rodiny Spartan-3 se používá nejčastěji soft procesor **Microblaze**, jehož implementace je součástí vývojového prostředí Xilinx ISE. Možnosti soft procesorů závisejí pouze na velikosti použitého hradlového pole. V případě FPGA z rodiny Spartan-3 lze použít jedno či dvoujádrový soft procesor Microblaze. Výhody takového řešení

spočívají v daleko efektivnějším přizpůsobení dané aplikaci a v možnosti měnit funkčnost již hotového výrobku daleko výrazněji, než pouhou změnou programu (s možným omezením paměti) jako u klasických mikroprocesorů a řídících mikropočítačů.

V návrhovém systému FPGA můžeme navíc vytvářenému řídícímu mikropočítači přidávat libovolné periferie v libovolném množství nebo i definovat periferie vlastní, což ho činí daleko flexibilnějším. Potřebujeme-li například zařízení s osmi „Compare“ jednotkami, které by se mezi klasickými mikropočítači pravděpodobně nenašlo, stačí je v systému se soft procesorem jednoduše nadeklarovat.

3.2 Architektura procesoru

Soft procesor Microblaze je 32bitový RISC procesor s Harwardskou architekturou, používá tedy oddělenou paměť pro data a instrukce. To odráží i použití dvou sběrnic: DLMB (Data-side Local Memory Bus) a ILMB (Instruction-side Local Memory Bus).



Obr. 3.1: Architektura procesoru Microblaze [8]

Na obr. 3.1 vidíme schema architektury s vyznačením pevných a volitelnách součástí. S výběrem volitelných součástí Microblaze se lze setkat např. při vytváření projektu v XPS (viz 4.7.1). Microblaze je procesor typu RISC (*Reduced Instruction Set Computing*). Tyto procesory se vyznačují jednoduchými strojovými instrukcemi, za účelem dosažení co nejrychlejší práce procesoru. Druhá skupina, CISC (*Complex Instruction Set Computing*) podporují složitější instrukce, na jejichž zpracování ovšem potřebují více času. Tomuto rozdílu bude odpovídat i přeložený program. Program, přeložený pro procesor typu RISC, bude delší než pro typ CISC,

nicméně celková doba potřebná pro vykonání programu bude kratší díky rychle zpracovávaným jednoduchým instrukcím. Charakteristickým rysem procesorů typu RISC je stejná délka všech instrukcí a použití „*pipeliningu*“ (více o pipeliningu v kapitole 3.2.3).

3.2.1 Registry

Všechny registry jsou 32 bitové. Lze je rozdělit na dvě skupiny:

GPR (*General Purpose Register*), tedy registry pro všeobecné použití

SPR (*Special Purpose Register*), tedy speciální registry

Registrů typu GPR je v procesoru 32 a jsou označeny R0-R31. Některé mohou být využity skutečně libovolně, jiné získávají při různých konfiguracích procesoru různé funkce, jako například uložení návratové adresy při přerušení. SPR registrů může být v závislosti na konfiguraci až 18. V této skupině nalezneme registry jako Program Counter (označen PC) nebo registry pro signalizaci výjimek a přerušení. Detailní popis lze nalézt v [8].

3.2.2 Paměť

Protože je Microblaze 32 bitový procesor, může adresovat až 32 bitový paměťový prostor. Pro paměť dat či paměť programu tak může využívat až 4 GiB paměti. Datovou i programovou paměť lze umístit do stejného paměťového prostoru a sdílet tak prostředky. Pro připojení paměti využívá procesor dvou sběrnic, DLMB a ILMB, jak bylo uvedeno v kapitole 3.2.

3.2.3 Pipelining

Termínem „*pipelining*“ (též zřetězené zpracování) je nazývána metoda, která se používá ke zrychlení práce procesoru při zpracovávání instrukcí, které vyžadují několik cyklů. V takovém případě může zůstat část procesoru nevyužita. Tomu *pipelining* zabraňuje rozdělením instrukce na dílčí instrukce, které jsou zpracovány různými částmi procesoru a tak můţe být postupně vykonáváno více instrukcí najednou. [23].

Při načítání instrukce z pomalejší paměti dochází ke zpoždění, samotné načtení instrukce může trvat i několik cyklů. Z tohoto důvodu je v procesoru umístěn načítací buffer, který načítá instrukce s předstihem, čímž nedochází ke snížení účinnosti pipeliningu. V procesoru Microblaze rozlišujeme tří- a pětifázový *pipelining*.

Obr. 3.2: Schema třífázového pipeliningu [23]

Při třífázovém *pipeliningu* se v první fázi instrukce načte, ve druhé fázi se dekóduje a ve třetí vykonává. Pokud je k vykonání instrukce potřeba více cyklů (na obr. 3.2 takovou instrukci představuje *instrukce 2*), musejí ostatní instrukce počkat (*instrukce 3* na obr. 3.2). Během tohoto čekání ovšem načítací buffer načítá další instrukce.

Při pětifázovém *pipeliningu* jsou rozlišovány kromě fáze načtení, dekódování a vykonávání instrukce (shodně s třífázovým) ještě fáze „čtení z paměti“ a „zápis výsledků do paměti“.

Druh použitého pipelingu lze vybrat pomocí parametru C_AREA_OPTIMIZED v MHS souboru projektu s procesorem Microblaze. Hodnota 0 znamená použití pětifázového, hodnota 1 třífázového pipeliningu.

3.2.4 Instrukce

U procesoru Microblaze rozlišujeme dva typy instrukcí. Instrukce typu A a typu B. Instrukce typu A se vyznačují tím, že mají dva zdrojové a jeden cílový registr. Instrukce typu B mají pouze jeden zdrojový a jeden cílový registr. Rozdíl v použití lze demonstrovat například na instrukcích ADD a ADDI pro sčítání. Instrukce ADD je typu A, má tedy dva zdrojové registry (Ra a Rb) a jejich součet uloží do třetího registru (například Rc). Instrukce ADDI je naproti tomu typu B. Proto umí přičíst hodnotu registru zdrojového (např. Rd) k registru cílovému (Re) tak, že výsledek operace se uloží do cílového registru Re.

3.2.5 Přerušení

Mimo klasického přerušení a resetu může procesor Microblaze reagovat ještě na hardwarové výjimky, uživatelské výjimky a na pozastavení běhu programu (break).

Pokud je zaznamenán signál resetu, proběhne vyčištění funkce pipeliningu a Program Couter se nastaví nulovou adresu (0x0).

Přerušení je vyhodnocováno, pokud je dovoleno nastavením *Interrupt Enable* (IE) bitu v registru MSR (Machine Status Register) na hodnotu jedna. Při přerušení je dokončena instrukce, která se nachází ve fázi vykonání, instrukce ve fázi dekódování je nahrazena instrukcí

pro skok na adresu 0x10 (adresa přerušení). Návratová adresa pro přerušení je automaticky načtena z GPR registru R14. Při provádění přerušení je nastaveno ignorování dalších přerušení nastavením IE bitu v MSR registru na hodnotu 0. Po provedení přerušení se IE bit vykonáním instrukce RTID automaticky nastaví na hodnotu 1. [23].

Hardwarová výjimka slouží k zachycování následujících druhů chyb: zakázaná instrukce, chyba instrukční či datové sběrnice procesoru (DLMB či ILMB), podtečení, přetečení či dělelní nulou. Může být vyvolána buď jednotkou FPU (*Floating Point Unit*) či MMU (*Memory Management Unit*). Dojde-li k hardwarové výjimce vyvolané jednotkou FPU, je vymazán *pipelining* a provede se skok na adresu 0x20. K uložení návratové adresy se použije GPR registr R17.

Uživatelská výjimka je vyvolána vložením zvláštní instrukce do programového kódu a je uložena na adrese 0x8. Návratou adresu je doporučeno ukládat do registru R8.

Pozastavení běhu programu (break) lze rozdělit na hardwarové a softwarové. V prvním případě se dokončí vykonávaná instrukce a provede se skok na adresu 0x18. Návratová adresa je uložena do registru R16. Softwarové pozastavení programu je vyvoláno zvláštní instrukcí typu A. Instrukce skoku je uložena v jednom zdrojovém registru. Okamžitá hodnota *čítače instrukcí* (PC) se uloží do cílového registru. Více o této problematice lze nalézt v [8].

3.2.6 Endianita

Pojmem endianita rozumíme formu zápisu čísel v paměti. Rozlišujeme tzv. *big-endian*, kdy nejdůležitější bit (M*SB – Most Significant Bit*) je uložen vpravo, tedy v bitu 0 (respektive nejnižším bitu čísla). Oproti tomu *little-endian* ukládá nejdůležitější bit vlevo, tedy na nejvyšší bit čísla. V oficiálních materiálech je tato problematika popisována zmateně. Např. v [10] se píše, že Microblaze pracuje jako *little-endian*, je-li připojen ke sběrnici AXI a jako *big-endian*, je-li v systému se sběrnicí PLB. Přesto však i se sběrnicí PLB například se softwarovými registry pracuje jako little-endian, nula je na bitu 31 (alespoň verze 8.0, kterou používáme). Tyto zmatky možná budou způsobeny tím, že od některé verze procesoru Microblaze bylo zavedeno něco, co bylo v předchozích verzích řešeno jinak. *Endianitu* lze nastavit pomocí parametru C_ENDIANNESS v MHS souboru, není to však doporučováno.

3.2.7 Podporované sběrnice

Procesor Microblaze ve verzi 8.30 (podzim 2013) podporuje sběrnice AXI a PLB verzi 4.3 (volitelně). Podpora sběrnice OPB (*On-chip Peripheral Bus*) byla ukončena ve verzi 7.20, stejně jako podpora starčí verze 2.3 sběrnice PLB. PLB je ve verzi ISE 12.3 stále ještě hlavní podporovanou sběrnicí, pro kterou jsou uzpůsobeny i dodané periferie. Sběrnice AXI byla

vytvořena zejména s novou produktovou řadou Zynq, která kombinuje hradlové pole s fyzickým procesorovým jádrem ARM. Dle firmy Xilinx tato nová sběrnice lépe reaguje na potřeby spojení FPGA a ARM jádra. Microblaze tuto sběrnici podporuje především z důvodu, kdy by se měl na platformě Zinq realizovat víceprocesorový systém s jádry Microblaze realizovanými pomocí přilehlé FPGA struktury. V následných verzích Xilinx ISE ani dalších vývojových prostředích, které jej nahradily (Vivado) již ale sběrnice PLB není podporována. Rodina hradlových polí typu Spartan 3, a tedy ani Spartan3AN s námi použitou vývojovou deskou mezi podporovanými zařízeními pro AXI sběrnici není, nabídka vytvoření systému s AXI, která je v ISE 12.3 je pro nás tedy jen teoretická.

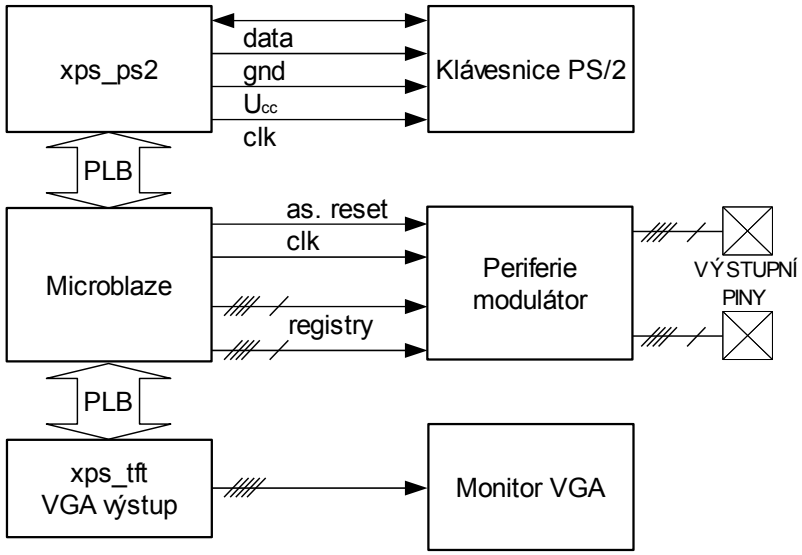
4 Realizace

Mým úkolem bylo pomocí procesoru Microblaze a vývojové desky Digilent Spartan 3AN Starter Kit realizovat alespoň tři různé techniky CB-PWM. K realiaci jsem si vybral tyto čtyři: Symmetrical Double Edge Regularly Sampled PWM (sinus – trojúhelník se symetrickým generováním), Asymmetrical Double Edge Regularly Sampled PWM (sinus – trojúhelník s asymetrickým generováním), Leading-Edge PWM (sinus - pilovitý signál) a Trailing-Edge PWM (sinus – obrácený pilovitý signál). V přesných názvech panuje mezi autory nejednotnost, viz kapitolu 2. Každý z těchto čtyř druhů jsem dále realizoval v podobě pro dvojčinný jednofázový měnič (použití pro můstek i polomost), šestipulní trojfázový střídač, tří a pětiúrovňový trojfázový měnič (uvažován typ s plovoucími kondenzátory). K dalším požadavkům patřilo provést nastavování parametrů jednotlivých modulačních technik pomocí uživatelského rozhraní s klasickým PC monitorem, připojeným pomocí VGA rozhraní a standardní 100/101 klávesovou PC klávesnicí připojenou přes rozhraní PS/2. K realizaci jsem použil softwarový balík Xilinx ISE Design Suite 12.3.

4.1 Navržená struktura

Zadaný úkol se nabízelo rozdělit na dvě části, aby byly plně využity možnosti a filozofie softcore procesoru Microblaze v hradlovém poli. Vlastní modulátor, zajišťující provádění vybraných modulačních technik, musí provádět mnoho operací v přesně daných a velmi krátkých časových intervalech. Proto je vhodné ho realizovat hardwarově, jako periferii procesoru, konkrétně popisem v HDL jazyce a syntézou pro naprogramování do FPGA. Protože pomocí logických obvodů realizovatelných v FPGA je velmi složité či v některých případech přímo nemožné (použitý vývojový software tyto možnosti nepodporoval) realizovat operace s proměnnými typu *real*, je k výpočtu veličin pro modulátor ze zadaných parametrů, stejně jako pro obsluhu uživatelského rozhraní, použit program v procesoru Microblaze. Periferie je k procesoru připojena přes sběrnici PLB v4.6, komunikace je prováděna přes softwarové registry. Dále je do

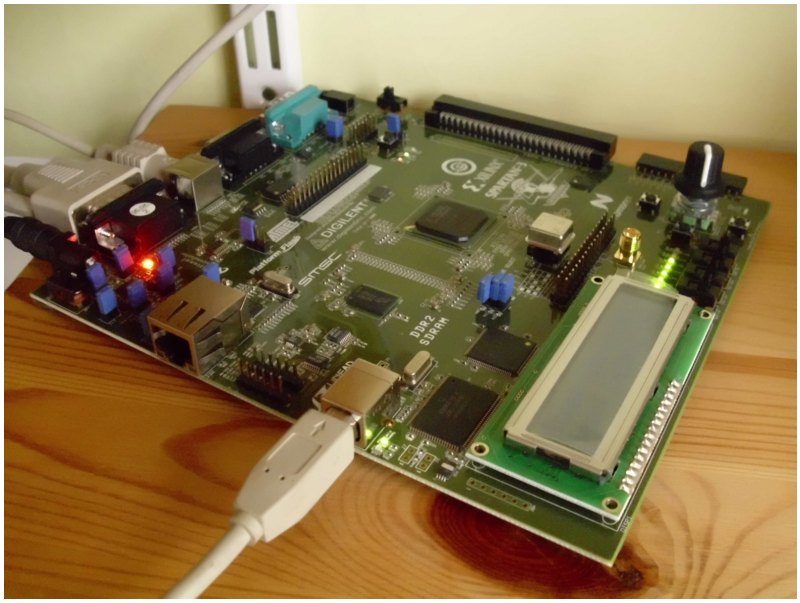
periferie vyveden zvláštní hodinový signál s kmitočtem 50 MHz a signál pro asynchronní reset spolu se sběrnicí. Obsluha klávesnice a monitoru je provedena přes konektory a příslušné součástky na vývojové desce za pomoci dodávaných periferií a ovladačů a ovládacího programu. Navrženou strukturu schematicky zobrazuje obr.4.1.



Obr. 4.1: Navržená struktura

4.2 Vývojová deska

Použitá vývojová deska, plným názvem Spartan 3AN Starter kit (obr. 4.2), obsahuje mimo hradlového pole (konkrétně typ XC3700AN-FGG484) dva konektory Canon 9 pro sériový port (RS232), čtyři spínače (tzv. DIP), čtyři tlačítka, osm LED diod, konektor VGA, znakový LCD displej 2×16 znaků, konektor PS/2 (6 vodičový Mini-DIN), rotační inkrementální spínač se středovým tlačítkem, SPI A/D převodník, SPI D/A převodník, konektor RJ45, 16 Mbit SPI flash paměť, 4 MiB paralelní flash paměť a 512 MiB DDR2 SDRAM paměť. Jedno z tlačítek slouží jako reset, takže nelze použít pro uživatelské účely. Pro vyvedení jiných signálů z desky může posloužit 100 pinový konektor označovaný jako Hirose FX2, dále pak tři pozice pro šestipinové konektory, z nichž dvě jsou standardně osazeny konektorem typu BLS (samice).



Obr. 4.2: Vývojová deska Spartan3AN Starter Kit

4.2.1 Důležitá poznámka pro připojení desky k zařízení

Pokud má být vývojová deska Spartan 3AN Starter kit připojena k nějakému zařízení, se kterým má komunikovat, či které má ovládat, je třeba upozornit na skutečnost, že po zapnutí napájení se na všech externích portech zhruba na sekundu objeví logická jednička. Následně jsou všechny porty opět uvedeny do stavu logické nuly. Není tedy vhodné mít desku připojenou k nějakému ovládanému zařízení, kde by tato vlastnost mohla vadit (typicky právě modulátor výkonového měniče, kdy by výstupní signály z desky měly přímo ovládat budiče spínacích součástek – pokud by měnič nebyl vybaven ochrannými obvody, znamenalo by to zkrat ve všech fázích). Tato vlastnost by šla zřejmě odstranit naprogramováním flash paměti desky, a tím změněním konfigurace výchozího stavu, to jsem ovšem experimentálně neověřoval.

4.3 Vývoj modulátoru

4.3.1 Vývojové prostředí Xilinx ISE

Vývojové prostředí Xilinx ISE Design Tools je soubor aplikací sloužících k všestranné podpoře při návrhu a vývoji zařízení s hradlovými poli firmy Xilinx. Obsahuje součásti pro vývoj HDL popisu logických obvodů, jejich simulaci, ladění, syntézu a nahrání do FPGA. Dále obsahuje součásti pro konfiguraci systémů s virtuálními procesory, podporu pro vytváření periferií k těmto procesorům a prostředí pro vývoj, ladění a nahrávání softwarových aplikací.

Tato práce popisuje postupy ve verzi 12.3, pro niž má katedra zakoupenu licenci. V jiných verzích mohou být postupy zcela odlišné. Firma Xilinx vydává nové verze velmi často (nejdéle

po půl roce) a jednotlivé verze jsou mezi sebou téměř nekompatibilní. Změny se týkají například rozložení ovládacích prvků, nabídek uživatelských menu, ale i tak zásadních věcí, jako třeba jmen funkcí použitých při programování nebo datových sběrnic. To můžeme demonstrovat třeba na příkladu sběrnice OPB (On-chip Peripheral Bus), kterou procesor Microblaze používal do verze 7.30 [8], [9]. Přechod na sběrnici PLB s sebou nesl výměnu všech periferií, protože ty dřívější tehdy neexistující sběrnici PLB nepodporovaly a nová verze procesoru Microblaze zase nepodporovala sběrnici OPB. To s sebou neslo téměř obtížnější práci než vytvořit celý nový projekt. V použité verzi 12.3, která vyšla v září 2010 se poprvé objevuje sběrnice AXI. V současnosti má použitá sběrnice PLB status „deprecated“, tedy není podporovaná. Rovněž vývojové prostředí ISE bylo nahrazeno prostředím Vivado. FPGA Spartan 3AN bylo nahrazeno novějšími modely a již se nevyrábí. Firma Xilinx se rozhodla jít cestou spojení hardwarových ARM procesorových jader s FPGA na jednom čipu (např. rodina výrobků Zynq). Procesor Microblaze je však nadále podporován jako doplňkový softprocesor pro tyto systémy, ovšem pouze se sběrnicí AXI, která byla vyvinuta právě pro přechod k systémům typu Zynq.

Tento překotný vývoj komplikuje rovněž dostupnost jakékoli jiné literatury týkající se konkrétních příkladů aplikací. Firma Xilinx sice na svých stránkách zveřejnila některé ukázkové projekty, ovšem pouze pro některé verze svých vývojových prostředí, mezi nimiž ISE 12.3 často chybí. Jak již bylo řečeno výše, jsou projekty z jiných verzích zpravidla nepoužitelné. Stejně tak vede tento rychlý vývoj k chybám v dokumentaci. Například v [10] se píše, že procesor Microblaze je v systémech se sběrnicí PLB little-endian a se sběrnicí AXI big-endian, v praxi ovšem v systému s PLB zapisuje do registrů systémem big-endian. Odhalování takovýchto chyb může při vývoji podstatným způsobem zdržovat.

Pro spouštění některých součástí (zejména XPS) je z důvodu ověřování licence na serveru potřeba připojení k internetu. To stačí ovšem jen pro okamžik spouštění, dále lze otevírat a editovat projekty i bez připojení. V tomto režimu lze i kompilovat softwarové aplikace pro procesor. Připojení k internetu je dále potřeba při vytváření Netlistu, kdy se při syntéze kontrolují platnosti licence pro jednotlivé dílčí periferie a komponenty systému.

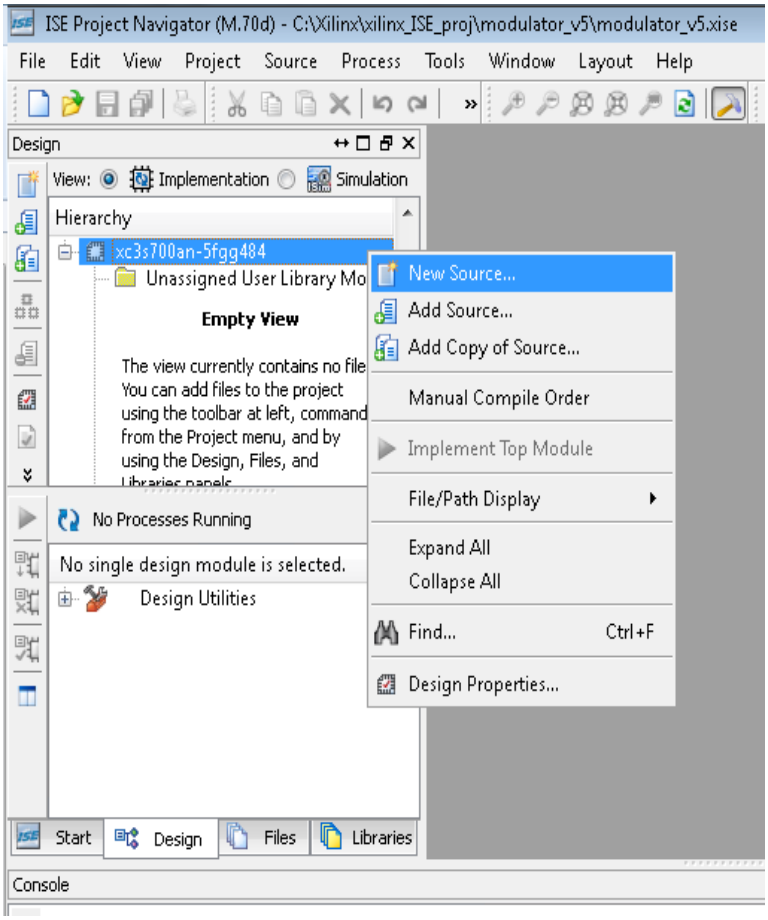
4.3.2 ISE Project Navigator

K vývoji v HDL jazycích (jsou podporovány Verilog a VHDL) je v balíku ISE Design Suite 12.3 určen program ISE Project Navigator. Pokud jsme při instalaci neměnili výchozí konfiguraci, lze spustit přes nabídku Start – ISE Design Suite 12.3 – Design Tools – Project Navigator. Pokud používáme 64-bitový operační systém, je na výběr 32 a 64 bitová verze.

Nový projekt založíme přes *File – New project*. Otevře se průvodce, kde vybereme pro použití s deskou Spartan 3AN Starter Kit typ zařízení (*FPGA Family*) „Spartan 3A and Spartan

3AN“ a jako *Device* „xc3s700AN“. Ostatní položky můžeme ponechat ve výchozím stavu.

Project Navigator pracuje ve dvou základních režimech, které je možné přepínat ve vrchní části levého panelu (obr.4.3). V režimu *implementation* lze provádět syntézu kódu v HDL pro nahrání do hradlového pole (pomocí XST – Xilinx Synthesis Tool), zatímco v režimu *Simulation* se na syntetizovatelnost kódu nehledí a pro účely testování je možno používat libovolné funkce prostředky jazyka. Výchozím simulátorem je Xilinx ISIM.

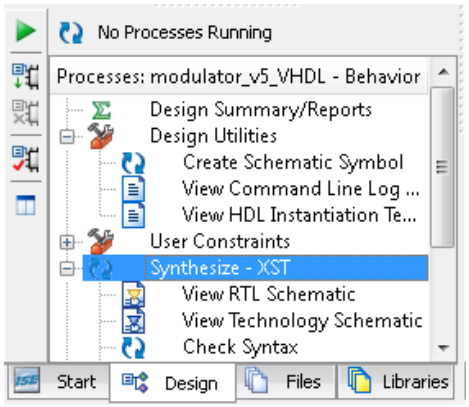


Obr. 4.3: Detail ISE Project Navigator

Do prázdného projektu je potřeba přidat nový zdrojový soubor. To provedeme pomocí volby *New Source* z nabídky, vyvolané pravým tlačítkem myši na nejvyšší položce v seznamu *Hierarchy*. Z následující je možno vybrat *Verilog module* či *VHDL module* zdrojový soubor. Jako použitý HDL jsem si zvolil VHDL. Průvodce dále požaduje vyplnit jméno souboru a nabídne vyplnit vstupní a výstupní porty modulu, jejichž popisy následně vygeneruje. Ty lze ovšem doplnit i později přímo do zdrojového kódu. Funkčnost kódu je vhodné průběžně ověřovat pomocí volby *Synthetize – XST* nebo pomocí *Check Syntax* (4.4). Pokud chceme funkčnost kódu ověřit simulací, přepneme do režimu *Simulation*.

Zde je třeba přidat simulační kód, který bude provádět změny na vstupních portech,

abychom mohli sledovat odezvu. To provedeme obdobně, jako přidání nového kódu v režimu *Implementation* (obr. 4.3), s tím rozdílem, že z nabídky vybereme položku *VHDL TestBench*. Na základě kódu z režimu *Implementation* se vygeneruje soubor, obsahující shodné vstupní a výstupní porty. V něm je dovoleno používat klauzule *wait for <čas>*, což využijeme pro simulaci časových změn na vstupních portech. V přehledu *Hierarchy* je zdrojový soubor z režimu *Implementation* zahrnut pod tento nový soubor jako tzv. UUT (Unit Under Test). Pro účely této práce budu rozlišovat pojem *simulační kód* (kód ve *VHDL TestBench*) a pojem simulovaný kód (*UUT*)



Obr. 4.4: ISE Project Navigator - možnosti

V simulačním kódu je třeba upravit konstantu *clk_period*, tak, aby odpovídala žádanému časování. V našem případě 20 ns (50 MHz hodinový signál). Simulace se spouští pomocí *Simulate Behavioral Model* na kartě *Design – Processes*, což spustí simulační program ISIM (obr. 4.6).

V porovnání s jinými simulátory HDL jazyků má velkou nevýhodu v tom, že neumí zobrazit výstupy v analogovém tvaru (myšleno hodnotu výstupu v podobě vektoru). Klikneme-li pravým tlačítkem na jméno signálu, můžeme v nabídce *Radix* zvolit zobrazení hodnot signálu ve dvojkové, desítkové či šestnáctkové soustavě. Poklepáním na název signálu, který je vektorem zobrazíme průběh na jednotlivých bitech. Pokud chceme mít výstup simulátoru v analogovém zobrazení (což je například při vývoji signálového generátoru velmi názorné), je nejjednodušším způsobem (nebudeme-li brát v úvahu použití jiného simulátoru a jiného vývojového prostředí) nechat vypsat hodnoty do textového souboru a tyto hodnoty zpracovat tabulkovým procesorem (Calc, Excel...) nebo pomocí Matlabu (což je v případě deseti a statisíců hodnot mnohem rychlejší). Do simulačního kódu za tímto účelem přidáme následující příkazy a deklarace:

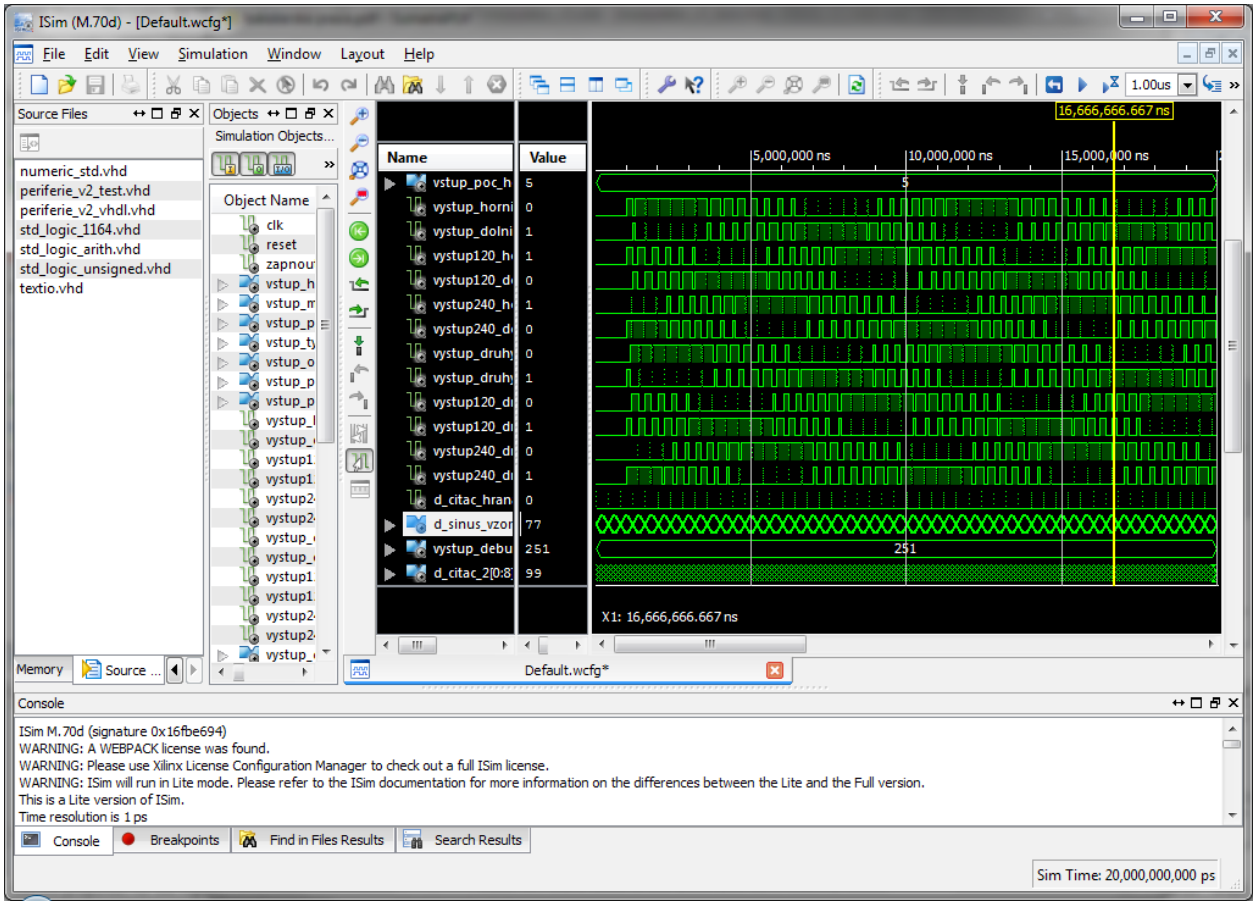
- Do deklarací použitých knihoven přidáme:
```
use ieee.numeric_std.all
use std.textio.all;
```

- Do bloku ENTITY: `generic (log_file: string := "vystup.log");`
- Za hlavičku ARCHITECTURE:
  `file soubor: TEXT open write_mode is log_file;`
- Pak již následuje na stejné úrovni jako proces *stim_proc* proces pro zápis do souboru. Je vhodné jej umístit do samostatného procesu se seznam signálů citlivostní formy (*sensitivity list*), aby zápis do souboru probíhal jen v okamžicích, kdy se signál mění a soubor tak nebyl plný duplicitních hodnot. Do seznamu umístíme buď všechny signály, jejichž signály chceme zapisovat nebo ten, který se mění nejčastěji. Proces může vypadat jako na obr. 4.5.

```
process(d_citac_vystup)
      variable line_number : line;
begin
      write (line_number,
            integer'image(conv_integer(d_citac_vystup))&","&
            integer'image(conv_integer(d_sinus_vystup)));
      writeline(soubor, line_number);
end process;
```

Obr. 4.5: Zdrojový kód procesu pro zápis do souboru

Další nevýhodou je nemožnost zobrazit vnitřní signály simulovaného kódu (*UUT*). To lze nejsnáze vyřešit přidáním výstupního portu v simulovaném kódu a zapisovat do něj shodné hodnoty, jako do sledovaného výstupu nebo tento signál přímo na takový výtupní port namapovat. Není nutné znovu generovat testovací VHDL soubor se simulačním kódem. Stačí odpovídající vstupy a výstupy přidat deklaracemi i do dotyčného simulačního kódu. Nový port přidáme do seznamu portů v klauzuli COMPONENT - PORT. Zde zachováváme orientaci portu jako v kódu simulovaném (UUT) , stačí tedy příslušnou deklaraci zkopírovat ze simulovaného kódu (samozřejmě nesmíme zapomenout upravit potřebné středníky na koncích řádků). Dále je potřeba deklarovat signál stejného typu, jako je port (název shodný být nemusí, ale pro přehlednost se obvykle nechává stejný) a tento signál k portu namapovat. Při dalším spuštění simulace již jeho hodnotu uvidíme mezi ostatními.

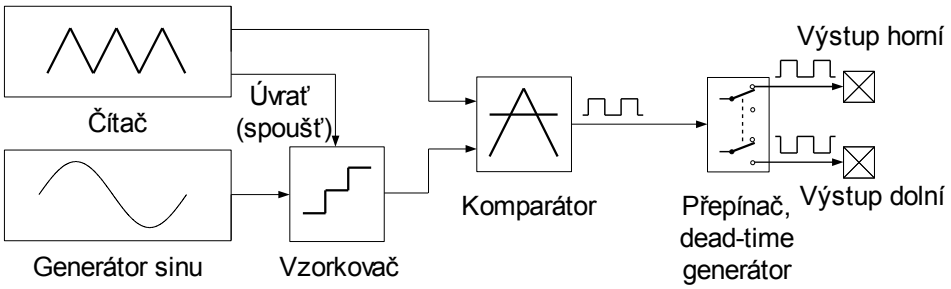


Obr. 4.6: Okno simulátoru ISIM

Nyní se budu věnovat vlastnímu vývoji VHDL kódu pro modulátor.

4.3.3 Návržená struktura modulátoru

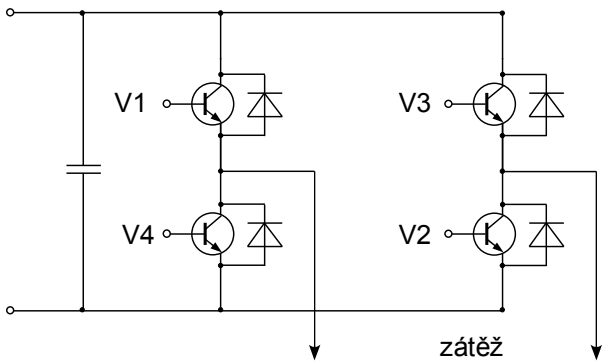
Na základě poznatků uvedených v kapitole 2.3.2 jsem navrhl následující strukturu hardwaru pro realizaci vybraných modulačních technik. Strukturu nejprve popíši na nejjednodušším typu modulátoru, tedy pro dvojčinný jednofázový měnič.



Obr. 4.7: Struktura modulátoru pro dvojčinný jednofázový měnič

Hlavním principem je porovnávání nosného signálu se signálem modulačním. To je realizováno pomocí komparátoru. Nosný signál je vytvářen čítačem, signál modulační

generátorem sinového průběhu. Dle 2.3.2 je při realizaci v číslicové podobě nutné udřovat modulační signál konstantní po celou dobu náběžné či sestupné rampy čítače, což je provedeno modulem, který jsem označil jako *Vzorkovač*. „Vzorkování“ je prováděno v úvratích čítače pomocí jednobitového signálu (náběžnou hranou krátkého pulzu) *Úvrať* (ve VHDL kódu označen jako `citac_hrana`). Signál z komparátoru je veden do modulu *Přepínač*, který zajišťuje správné spínání komplementárních ventilů. Je-li na výstupu z komparátoru zaznamenána změna, vypne se nejdříve ventil, který je právě sepnut a spustí se čítač, který zajišťuje čekání do doby, než uplyne nastavená *ochranná doba* (v praxi též označována termínem *dead-time*, ve zdrojovém reprezentována signálem `vstup_ochranna_doba`). Potom sepne výstup pro druhý ventil. Blok *Přepínač* je tak vybaven tzv. dead-time generátorem. Takovýto modulátor lze použít např. pro můstkový měnič (obr. 2.3), kdy výstup modulátoru „*Výstup horní*“ bude přiveden na ventily V1 a V2 a „*Výstup dolní*“ na V3 a V4. Stejně tak je možné použít uspořádání *polomost*, s kapacitním děličem.



Obr. 4.8: Dvojčinný můstkový měnič

Jako popisný jazyk jsem zvolil VHDL. Jednotlivé bloky jsem realizoval pomocí procesů, které mezi sebou vzájemně komunikují pomocí signálů (*signals*). Zadávání hodnot do modulátoru je provedeno pomocí softwarových registrů z Microblaze (viz 4.7.6). Seznam zadávacích proměnných a jim odpovídajícím registrům lze nalézt v kapitole 4.7.7.

4.3.4 Přímá digitální syntéza signálu

Navržená struktura v zásadě odpovídá myšlence přímé digitální syntézy signálu (*Direct Digital Synthetizer, DDS*) s číslicově řízeným oscilátorem (*Numerically controlled oscilator NCO*) viz např. [24] [29]. Tento postup vychází z užití referenčního signálu o konstantním kmitočtu, kdy je pomocí různých číslicových (někdy též digitálně-analogových či přímo analogových) generována výstupní frekvence jako násobek či podíl referenčního signálu. Ve svém řešení jsem použil čítače, které po přetečení provedou nějakou akci (např. zvětšení hodnoty výstupního signálu o jedna), což je v zásadě princip NCO. Přímé digitální syntéze pak odpovídá například to, že na základě akcí pomocných čítačů je na výstup generátoru sinu zapisována

okamžitému stavu odpovídající hodnota funkce sinus.

4.3.5 Generátor proměnného sinusového průběhu

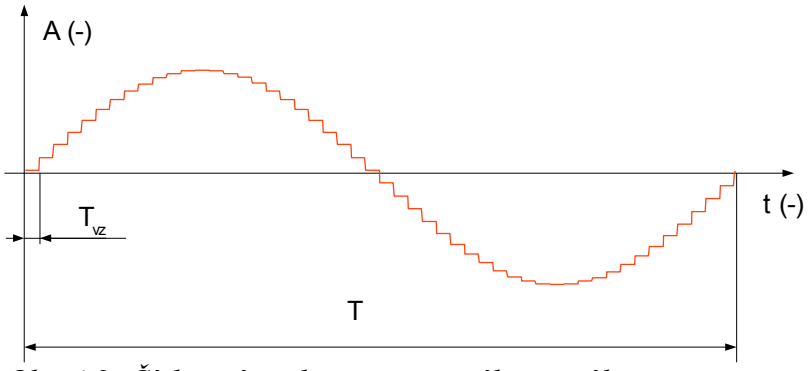
Generovat přesné sinusové průběhy o libovolném kmitočtu je v číslicové technice složité. Máme totiž pevně daný kmitočet hodinového signálu, z něhož lze získat pouze kmitočty odpovídající násobkům a dílům toho hodinového. Pro některé požadované kmitočty má pak perioda velmi dlouhý desetinný rozvoj, takže s použitelným rozlišením nelze takový kmitočet vyrobit zcela přesně a tak se někdy generátor sinu realizuje s výstupními kmitočty v hodnotách po skocích [18] nebo dokonce jen pro pevně zadaný kmitočet [16].

Někoho by mohlo zmást, že jazyk VHDL obsahuje i funkce pro výpočet goniometrických funkcí, tyto však nejsou tzv. syntetizovatelné, tedy je nelze přeložit pro hardware. Mohou tak sloužit pouze pro účely testování, například pro generování zkušebních signálů při simulaci. V hardwaru je nutné vyřešit problém jinak. Jako nejčastější řešení se používá uložení předem vypočtených funkčních hodnot goniometrické funkce v tabulce (Lookup Table, LUT). Takováto tabulka sice nárokuje nemalé množství paměti (úměrně počtu hodnot v ní obsažených), ovšem dále je na hardware méně náročná než řešení výpočtová a tudíž bývá i mnohem rychlejší.

Hodnoty uložené v tabulce mohou být buď v absolutní, nebo v relativní formě (tedy jako přírůstky předchozích hodnot). Druhá forma šetří paměťové nároky (je potřeba uložit číslo s nižším počtem bitů), přidává ale naproti tomu nároky na hardware v podobě sčítání. Dále lze paměť ušetřit tím, že se uloží pouze čtvrtperioda funkce ($0 \div \frac{\pi}{2}$), zbylé hodnoty se dopočítají zdrcadlením.

Navržené řešení generátoru sinového průběhu tedy bude na výstupu vždy po uplynutí konstantní časové hodnoty zobrazovat novou hodnotu amplitudy tak, aby se za dobu uplynutí periody požadovaného signálu vystřídaly uložené hodnoty pro všechny čtyři čtvrtperiody. Tuto jednu hodnotu amplitudy budu nazývat *vzorek*.

Takto vzniklý signál bude díky povaze číslicových systémů schodovitý (obr. 4.9).



Obr. 4.9: Číslivocá realizace sinusového signálu

Z obr. 4.9 Je patrné, že „hladší“ průběh dostaneme, zvětšíme – li počet vzorků za periodu T (vzorkování) a stejně tak počet hodnot amplitudy (kvantování). Počet kvantizačních úrovní se volí různý pro různé účely. Hodnoty v LUT jsem uložil osmibitově, k dispozici je tedy 256 úrovní amplitudy. Zdá se, že je to řešení dostačující, v praxi se používá i méně, např. [17] používá rozlišení sedmibitové. Výstup generátoru sinu jsem pak provedl devítibitový, pro odlišení kladné a záporné půlperiody (výstup je realizován v celých číslech).

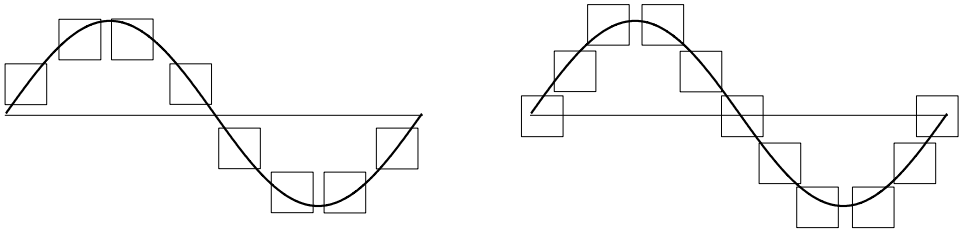
4.3.6 Volba počtu vzorků

Volba počtu vzorků může být složitější, než by se na první pohled mohlo zdát. Časová délka vzorků a jejich počet totiž ovlivňuje možné výstupní kmitočty. Zvolíme-li například počet vzorků na periodu n_T 500 a perioda hodinového signálu T_{clk} bude 20 ns, bude například při délce trvání vzorku 2000 T_{clk} perioda výstupního signálu T:

$$T = n_T \cdot 2000\, T_{clk} = (500 \cdot 2000 \cdot 20)\ \text{ns} = 20\,000\,000\ \text{ns} = 0{,}02\ \text{s} \quad , \tag{4.1}$$

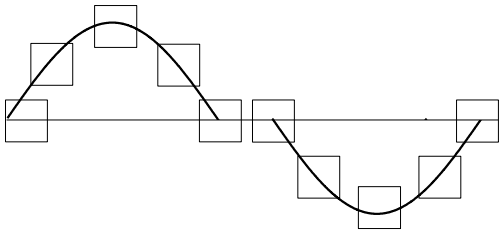
tedy výstupní kmitočet bude 50 Hz. Do volby počtu vzorků pak promlouvá i způsob uložení hodnot v LUT. Zvolíme-li řešení, kdy je v LUT pouze čtvrtina periody, a tuto čtvrtinu periody příslušným způsobem opakovat, musí být zvolený počet vzorků dělitelný čtyřmi.

Drobnou úpravou lze dosáhnout i počtu vzorků sice nedělitelných čtyřmi, ale dělitelnými dvěma. U takových počtů pak vychází na půlperiodu lichý počet vzorků, například pro $n_T = 510$ vyjde na polovinu periody 255 vzorků a tedy na čtvrtinu periody 127,5 vzorku. To je možné obejít tak, že v jedné čtvrtině periody bude sice uloženo 128 vzorků, ale každou 2. a 3. čtvrtinu periody (tedy na konci každé půlperiody) se poslední hodnota nevolá a místo ní se vyvolá opět první hodnota následné čtvrperiody. Průběh pak bude vypadat jako na obr. 4.10 vpravo.



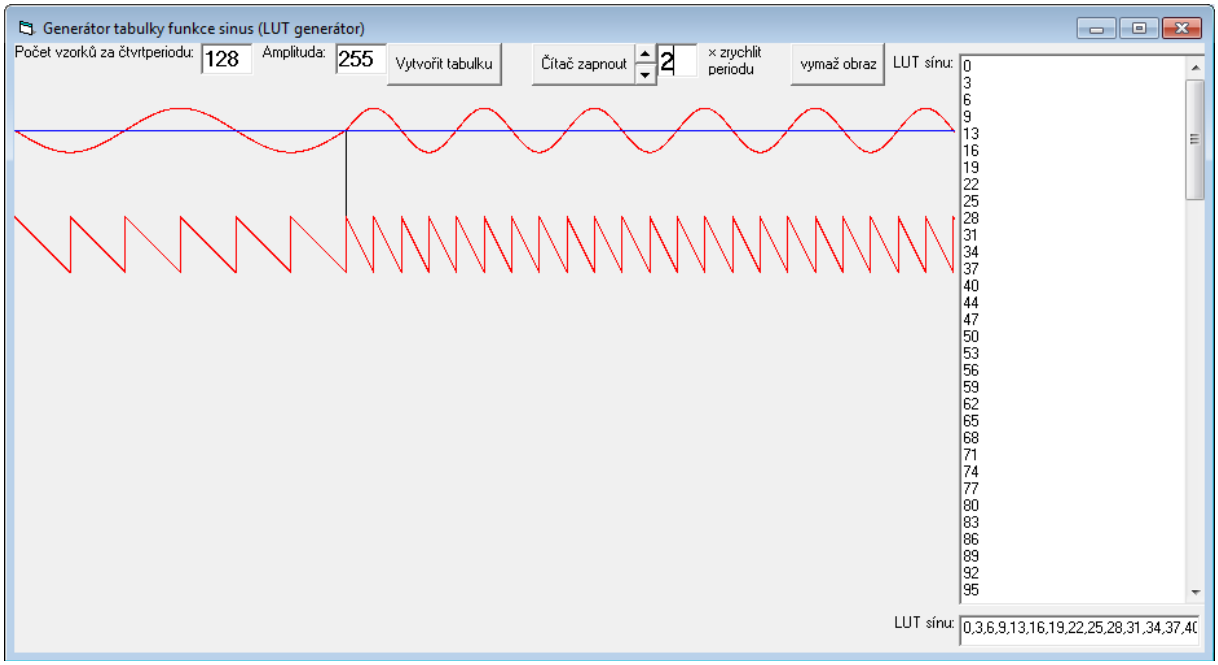
Obr. 4.10: Sudý a lichý počet vzorků sinu za polovinu periody

Poslední hodnota 2. čtvrtperiody je sdílena s první hodnotou 3. čtvrtperiody, tedy na tomto ilustrativním obrázku připadá na polovinu periody jakoby 5 vzorků. Na obr. 4.10 vlevo je průběh pro sudý počet vzorků na čtvrtperiodu. Každou čtvrtinu periody se tedy opakuje stále stejný počet vzorků. Na tomto obrázku si všimneme ještě průchodu osy x (y = 0). Na obr. 4.10 vlevo prochází mezi vzorky nad a pod osou. Je-li systém realizován v celých nezáporných číslech (např. *unsigned byte*) a amplituda bude 249, osa x se nachází mezi hladinami 249 a 250. V první a druhé periodě se hodnota z LUT bude přičítat k číslu 250, ve 3. a 4. odečítat od čísla 249. Pokud bychom jako osu x zvolili pouze jednu úroveň, například 249, signál vůči ose x umístěn nesouměrně, nebo by muselo dojít k situaci, jako na obr. 4.11 a tedy k nespojitému napojení půlvln. V případě lichého počtu vzorků lze jako hodnotu x zvolit jen jedno číslo, jak naznačuje obr. 4.10 vlevo.



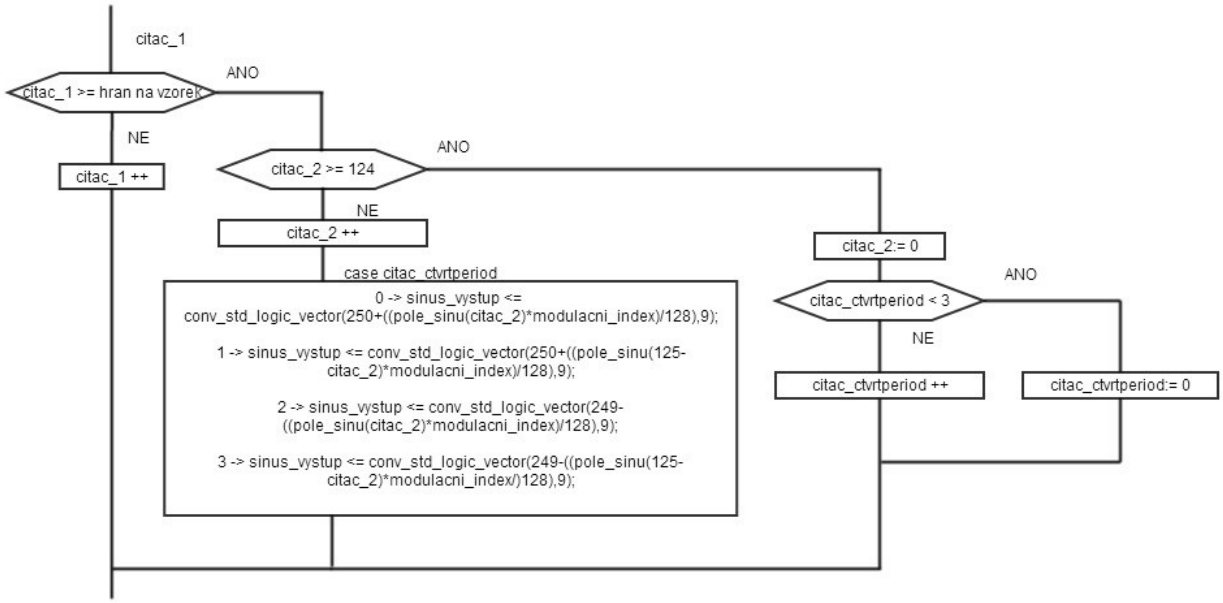
Obr. 4.11: Nevhodné napojení půlperiod při lichém počtu vzorků na půlperiodu

Pro generování LUT o zadaných parametrech (počet vzorků čtvrtiny periody a amplituda) jsem si vytvořil program v prostředí MS Visual Basic 6 (obr. 4.12). Spustitelný program i zdrojový kód jsou součástí přílohy na DVD.



Obr. 4.12: Program pro generování LUT čtvrtiny periody sinu

Vkládat na toto místo zdrojový kód bloku generátoru sinu by bylo nepřehledné, proto ukáži jeho strukturu pouze na stručném vývojovém diagramu (obr. 4.13).



Obr. 4.13: Vývojový diagram generátoru sinu

Dobu trvání jednoho vzorku určuje proměnná `citac_1`, která se zvětšuje při každé náběžné hraně hodinového signálu. Při přetečení zvětší o jedna hodnotu proměnné `citac_2`, kterou se adresuje hodnota v LUT. Podle hodnoty proměnné `citac_ctvrtperiod` se na výstup dostává příslušně upravená hodnota z LUT. Při každém přetečení proměnné `citac_2` je

zvětšena hodnota proměnné `citac_ctvrteperiod`, která udržuje přehled, ve které čtvrtině periody se průběh nachází. Vývojový diagram je pro 504 vzorků a nezobrazuje další části, nutné pro ošetření úvodní fáze (uvolnění resetu) a synchronizaci s čítačem.

Velikost amplitudy výstupního signálu se mění zmenšením základní hodnoty, jak je popsáno v kapitole 4.10. Fázový posun je zajištěn změnou počátečních hodnot řídících čítačů. Případná změněná hodnota amplitudy či periody je načítána při průchodu nulou (vynulování proměnné `citac_ctvrtperiod`)

Vstupními proměnnými jsou:

- `vstup_hran_na_vzorek_sinu` počet period hodinového signálu, po který má být na výstupu jeden vzorek
- `vstup_hloubka_modulace` hodnota ovládající velikost výstupní amplitudy
- `vstup_poc_hod_citac_1` počáteční hodnota čítače 1
- `vstup_poc_hod_citac_2` počáteční hodnota čítače 2

Výstupem je:

- `sinus_vystup` devítibitový výstup

4.3.7 Generátor nosného signálu

Generátor nosného signálu, tedy trojúhelníkového a pilového, vychází z podobných principů, jako generátor sinu. Také zde je na výstup předávána hodnota ve vzorcích a doba jejich zobrazení je řízena pomocným čítačem na základě hodinového signálu. I zde volba počtu vzorků ovlivňuje možné hodnoty výstupních kmitočtů. Proto jsem pro čítač zvolil 1000 vzorků na periodu u trojúhelníkovitých, resp. 500 vzorků na periodu u pilovitých. Potom například pro dobu trvání jednoho vzorku 50 period hodinového signálu dosáhneme u trojúhelníkovitého signálu výstupní periodu $T_{\text{ČΔ}}$:

$$T_{\check{C}\Delta} = n_{\text{TČ}\Delta} \cdot 40\, T_{\text{clk}} = (1000 \cdot 50 \cdot 20)\ \text{ns} = 1\,000\,000\ \text{ns} = 0{,}001\ \text{s} \quad , \tag{4.2}$$

Tomu odpovídá kmitočet 1 kHz. Pilovitý signál o stejném kmitočtu musí mít dvojnásobný počet hodinových pulzů na jeden vzorek, neboť trojúhelníkovitý signál má dvojnásobný počet vzorků na periodu (náběžná a klesající rampa).

Blok obsahuje kód pro signál všech čtyřech realizovaných modulačních technik. Který z nich se bude vykonávat určuje signál `vstup_typ_citace`. Úkolem čítače je také ve správné

okamžiky předat pokyn *Vzorkovači* k načtení nové hodnoty sinu. To je provedeno pomocí pulzu šířky jednoho vzorku, který je v signálu `citac_hrana` vygenerován v příslušné úvrati čítače.

Vstupní signály jsou

- `vstup_pocet_hran_na_inkrement` počet period hodinového signálu, po který má být na výstupu jeden vzorek
- `vstup_typ_citace` určuje druh čítače a tím nosného signálu, nabývá hodnot 0-3

Výstupem jsou signály

- `citac_vystup` devítibitový výstup
- `citac_hrana` jednobitový výstup

4.3.8 Vzorkovač

Z hlediska popisu je vzorkovač velmi jednoduchý. Stačí, aby při náběžné hraně signálu `citac_hrana` převedl hodnotu ze vstupu na výstup a tam ji držel do dalšího pokynu. Za tím účelem je signál `citac_hrana` v seznamu signálů citlivostní formy (*sensitivity list*).

Vstup:

- `sinus_vystup` devítibitový signál

Výstup:

- `sinus_vzorek_komparator` devítibitový výstup

4.3.9 Komparátor

Vstupy:

- `sinus_vzorek_komparator` devítibitový signál
- `citac_vystup` devítibitový signál

Výstup:

- `komparator_vystup` jednobitový výstup

4.3.10 Přepínač

K vytváření ochranné doby mezi vypnutím jednoho a zapnutím druhého ventilu v jedné větvi je

použit pomocný čítač.

Vstup:

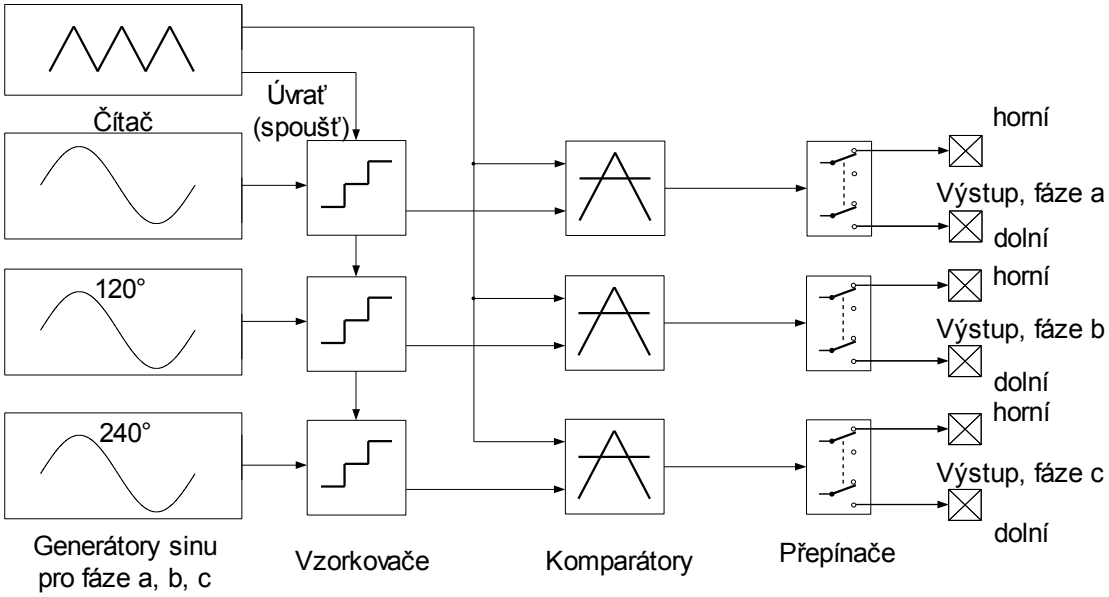
- `komparator_vystup` devítibitový signál

Výstupy:

- `vystup_horni` jednobitový výstup, vyveden na port
- `vystup_dolni` jednobitový výstup, vyveden na port

4.4 Struktura modulátoru pro šestipulzní měnič

Struktura navržená pro šestipulzní měnič (obr. 4.14) vychází ze struktury pro dvojčinný měnič. Nejvýznamnější odlišností je přítomnost tří, fázově posunutých sinových signálů.



Obr. 4.14: Navržená struktura modulátoru pro šestipulzní měnič

Fázový posun lze provést nejsnadněji tak, že na výstup předáme hodnoty z LUT, které jsou o patřičný počet vzorků posunuty. Nabízí se tedy otázka, zda může být generování těchto tří signálů provedeno jedním blokem. Protože jsou v LUT uloženy pouze hodnoty jedné čtvrtiny periody, je řešení složitější než jen posílání tří, o konstantu posunutých, hodnot. Bylo by třeba udržovat další dvě hodnoty příslušných čtvrtin period, čítače i ostatní režie a řešení by bylo nepřehledné. Zvolil jsem proto řešení se třemi identickými bloky generátoru sinu (totožné s řešením popsaném v kapitole 4.3.6), kdy každý začíná s jinou hodnotou proměnných `citac_ctvrtperiod` a `citac_2`. Nutný „*phase advancing*“ fázový posun je dán kmitočtem

nosného signálu, je proto pro všechny tři sinové signály stejný.

Toto řešení zabírá paměť dalšími dvěma LUT, stále je však paměťově méně náročné, než kdyby byla v LUT uložena celá perioda sinu. Takto jsou to pouze tři čtvrtiny. Pro úsporu paměti by bylo možné vytvořit zvláštní blok (například formou samostatného procesu), který by byl adresován jednotlivými sinovými bloky a vracel jim požadovanou hodnotu z LUT. Protože použité hradlové pole poskytuje dostatečné hardwarové možnosti, nebylo nutné optimalizovat množství použité paměti ve vztahu k LUT. Řešení se třemi LUT se používá i v praxi, což popisuje např. [17].

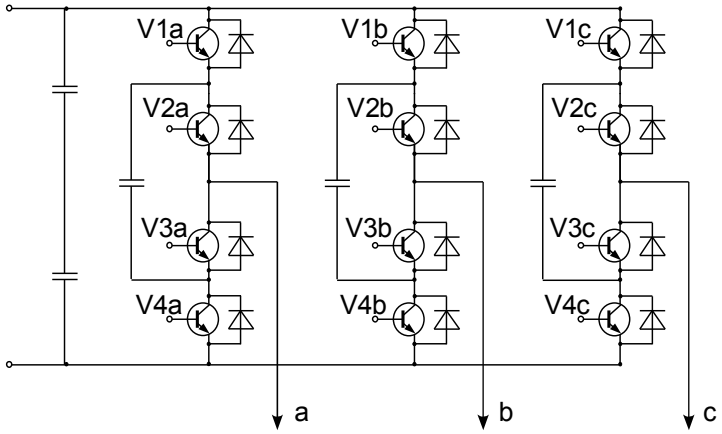
Aby byl fázový posun o třetinu, resp. dvě třetiny snadno realizovatelný, je vhodné vybrat počet vzorků na periodu sinu dělitelný třemi. Zároveň, kvůli uložení čtvrtiny periody do LUT by měl být dělitelný čtyřmi. Hledáme tedy násobky dvanácti. Jako nejvhodnější kandidát se jeví číslo 504, protože nejlépe využívá rozsah devítibitový rozsah (0-511) použitých proměnných.

Ve struktuře je díky přítomnosti tří modulačních signálů zapotřebí tří *vzorkovačů*. Protože se vzorkování vztahuje pouze k jednomu čítači, je možné je sloučit do jednoho bloku se třemi vstupními a třemi výstupními signály. Tři použité *komparátory* však již pracují samostatně. Posledním článkem jsou opět bloky přepínačů, které jsou shodné s provedením u dvojčinného jednofázového měniče. I ovládání modulátoru se provádí pomocí stejných proměnných a registrů, jako v předchozím případě.

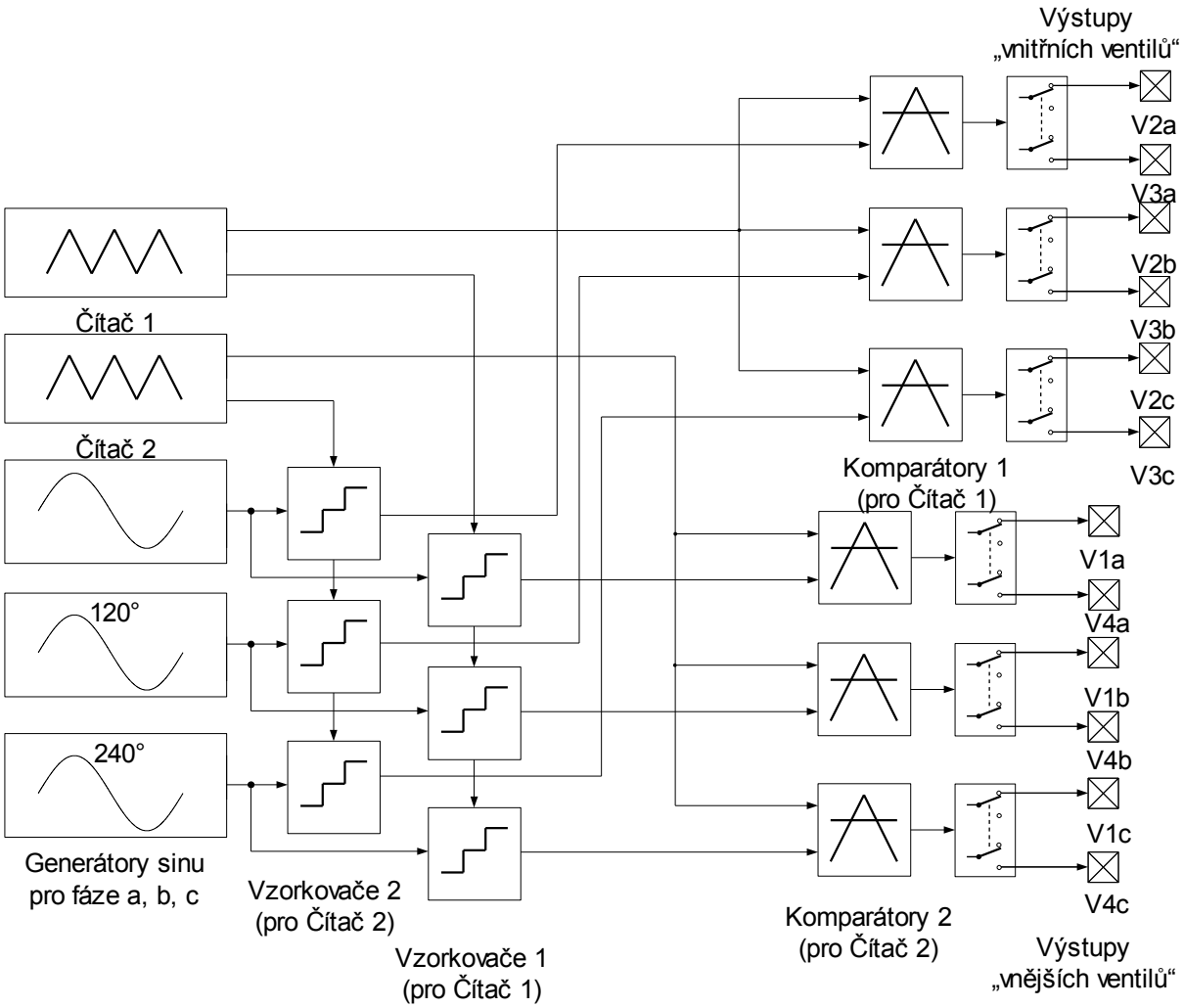
4.5 Struktura modulátoru pro tříúrovňový měnič

Tříúrovňový měnič vyžaduje v každé větvi 4 spínací součástky. Pro realizaci modulační techniky jsem zvolil metodu posunutých nosných signálů (Phase Shifted Carrier Based PWM). V případě dvou nosných signálů jsou tyto vůči sobě posunuty o polovinu své periody. Oba dva mají stejný kmitočet. Konstrukcí víceúrovňových měničů existuje více druhů (viz 2.4), obr. 4.15 ukazuje tříúrovňový měnič v provedení s plovoucími kondenzátory (FCMI – *Floating Capacitors Multilevel Inverter*).

Ve struktuře (obr. 4.16) se proto objeví ještě jeden čítač (generátor nosného signálu). Ten je stejný, jako první čítač, pouze začíná od jiné hodnoty. Vstupní řídící proměnnou obou čítačů je počet period hodinového signálu, po kterou má na výstupu setrvat jeden vzorek. Ta je pro oba čítače stejná a neovlivňuje číslo vzorku na výstupu čítače, pouze jeho trvání. Proto je nastavená hodnota fázového posunu (v číslech vzorků) pevná shodná pro jakýkoli nastavený kmitočet.



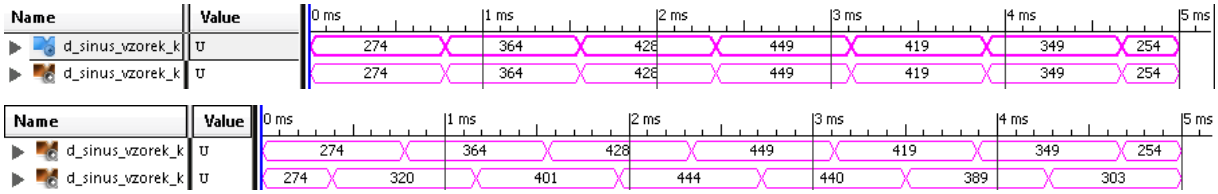
Obr. 4.15: Tříúrovňový měnič s plovoucími kondenzátory



Obr. 4.16: Navržená struktura modulátoru pro tříúrovňový měnič

„*Phase advancing*“ modulačního signálu je také stále stejný, protože čítače mají stále stejnou periodu. Na obr. 4.17 je vidět hlavní rozdíl mezi modulačními technikami *asymmetrical double edge* a *symmetrical double edge*. U „*asymmetrical*“ vycházejí okamžiky vzorkování pro

oba čítače totožně. Posunutí o půl periody, totiž znamená, že ve stejný okamžik nastane horní úvrať jednoho a dolní úvrať druhého čítače a tedy i časy převzetí vzorku. U „*symmetrical*“ a obou druhů *sigle edge* modulačních technik jsou vzorky vůči sobě posunuty o půl periody nosného signálu.

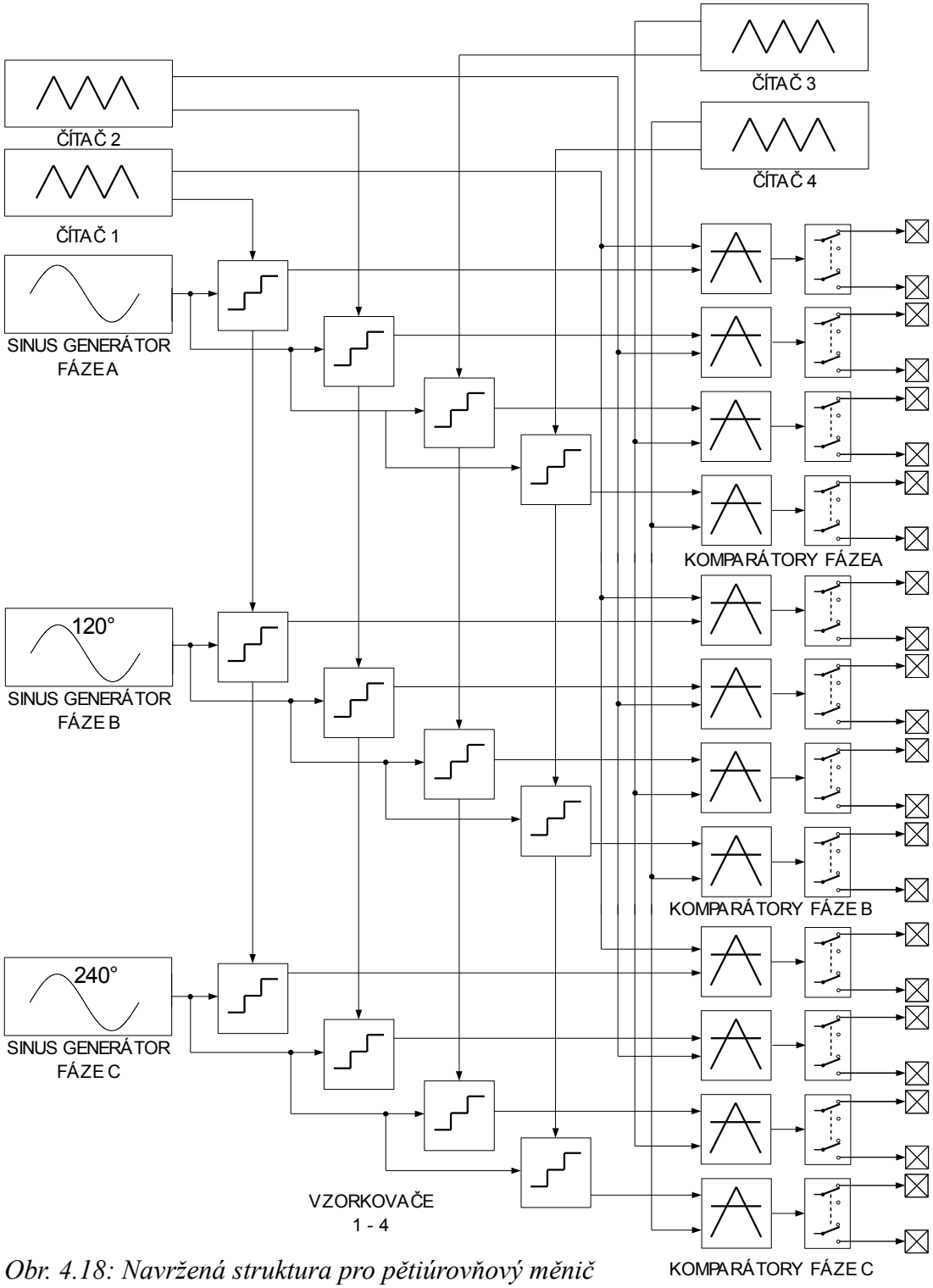


Obr. 4.17: Nahoře: vzorkování jednoho modulačního signálu dvěma čítači při asymmetrical double edge modulace
Dole: vzorkování dvěma čítači při symmetrical double edge

Vzorkování modulačních signálů probíhá pro každý čítač zvlášť, bylo tedy nutné zařadit i další bloky typu „*vzorkovač*“. Podobně jako v případě modulátoru pro šestipulzní měnič jsou ve zdrojovém kódu realizovány všechny tři vzorkovače, připadající na jeden čítač, realizovány jedním procesem. Dva čítače také rozlišují dvě skupiny *komparátorů*, kde jedna skupina porovnává signál z *čítače 1* se třemi modulačními signály (resp. jejich navzorkovanými hodnotami), druhá skupina tyto modulační signály porovnává s *čítačem 2*. Skupina výstupů odvozená od *čítače 1* je připojena k „vnitřním ventilům“, tedy těm, které přímo sousedí s vývodem fáze z měniče (např. *V2a* a *V3a*). Skupina vývodů s nosným signálem od *čítače 2* je zapojena na ventily „vnější“, tedy například *V1a* a *V4a*. Ve zdrojovém kódu jsou všechny přidané bloky (resp. jim odpovídající procesy) označeny přízviskem „*druhy*“.

4.6 Struktura modulátoru pro pětiúrovňový měnič

Struktura pětiúrovňového měniče je extrapolací struktury měniče tříúrovňového. Protože pro měnič s *n* hladinami je třeba *n-1* nosných signálů (viz 2.4), jsou ve struktuře zařazeny čtyři čítače (4 generátory nosného signálu). Tomu odpovídají i čtyři skupiny *vzorkovačů* a *komparátorů*. Ve zdrojovém kódu jsou nové prvky označeny přídomkem „*druhy*“, „*treti*“ a „*ctvrty*“, podle toho, ke kterému z čítačů se vztahují. Čítače jsou vůči sobě posunuty vždy o čtvrtinu periody. Perioda všech čítačů je shodná. Všechny ostatní vlastnosti a řídící proměnné jsou stejné jako u byly u měniče tříúrovňového. Přiřazení vývodů k ventilům odpovídá schématu, že čím posunutější je nosný signál, tím vzdálenější od středu větvě (vývodu fáze) jsou příslušné ventily. Vezmeme-li v úvahu například měnič, jehož schéma je zobrazeno na obr. 2.22, budou vývodům pro fázi a a *čítač 1* odpovídat ventily *V4a* a *V5a, čítači 2* pak ventily *V3a* a *V6a*, *čítači 3 V2a* a *V7a* a konečně *čítači 4 V1a* a *V8a*.



Obr. 4.18: Navržená struktura pro pětiúrovňový měnič

Máme-li vytvořen funkční kód budoucí periferie v ISE, můžeme začít vytvářet procesorový systém. K tomu použijeme z balíku aplikací součást XPS (Xilinx Platform Studio).

4.7 XPS

Xilinx Platform Studio (dále jen XPS) je součástí tzv. EDK (Embedded Development Kit), což je

část aplikací ISE, která, jak název napovídá, má sloužit k vývoji vestavěných systémů. Druhou hlavní součástí EDK je *Software Develpment Kit* (SDK), který má sloužit k vývoji programů vytvářených pro vestavěný systém. Použití SDK se v této práci nebudu věnovat, neboť všechny potřebné úkony při vývoji programů je možné provést i přímo z XPS.

4.7.1 Vytvoření projektu pomocí BSB

Po spuštění XPS je potřeba vytvořit nový projekt či otevřít stávající, na což jsme dotázáni hned v úvodu. Vytvoření nového projektu provedeme pomocí *Base System Builder Wizard* (BSB). Postup je následující:

- Zvolit cílový adresář projektu, jméno projektového souboru (ve výchozím stavu „*system.xmp*“), *Next.*
- Výběr sběrnicového systému, zvolit *PLB system*, *Next.*
- *Create new design, Next.*
- Výběr vývojové desky – Spartan-3AN Starter Kit, *Next.*
- *Single processor system*, *Next.*
- Dostáváme se na kartu s výběrem možností časování a velikosti paměti. *Reference Clock Frequency* je frekvence vystupující do celé desky z bloku oscilátoru (na výběr 133 MHz, což je napsané na použitém oscilátoru a 50 MHz, což je dosaženo děličkou). *System Clock Frequency* je frekvence použitá pro časování procesoru Microblaze (dostupná i pro periferii v podobě sugnálu *Bus2IP_Clk*). *Local Memory* určuje velikost BRAM procesoru. Je dobré, ji nastavit větší, pokud například víme, že budeme chtít v programu používat paměťově náročné funkce, jako např. z knihovny *stdio.h*. Velikost paměti lze později změnit i v *System Assemby View* na kartě *Addresses* (pak je ale pro funkčnost třeba přegenerovat i *Linker Script* programu). Pokud budeme chtít v projektu využívat DDR paměť, nesmíme zvolit frekvenci menší než 125 MHz (v obou kolonkách), protože nižší frekvenci použitá paměť nepodporuje. Na to bychom byli upozorněni na následující kartě, protože však BSB nemá možnost vrátit se v průvodci zpět, bylo by nutné průvodce ukončit a založit nový projekt znovu.
- V dalším okně lze zvolit použité výchozí periferie. Pro svůj projekt jsem použil DDR2_SDRAM, nutnou pro umístění videopaměti monitoru, dále LEDs_8Bit pro účely indikace a pro účely ladění DIPs_4bit a BTNs_4bit (přepínače a tlačítka). *Dlmb_cntlr* a *ilmb_cntlr* jsou standardní neodebiratelné součásti a reprezentují

paměť dat a programu pro Microblaze.

- Dále již ponecháme výchozí nastavení a pomocí *Next* ukončíme průvodce.

4.7.2 Projekt v XPS

Projekt se sestává z rozsáhlé souborové a adresářové struktury. Nejdůležitější popisné soubory jsou pro úpravy přístupné z karty Project v levé části okna XPS.

- MHS (*Microprocessor Hardware Specification*). Tento soubor obsahuje seznam použitých periferií, jejich připojení na sběrnice a vzájemná propojení, stejně jako seznam externích portů. Obsahuje též některé konfigurovatelné parametry periferií (například kmitočty signálů *clock_generator*)
- MSS (*Microprocessor Software Specification*). Obsahuje informace o tom, jaké mají být pro jednotlivé periferie použity ovladače.
- UCF (*User Constraint File*). Soubor obsahující definice vstupů a výstupů vyvedených na fyzické výstupy pouzdra FPGA. Pro připojení jednotlivých periferií na desce je potřeba v dokumentaci desky (pro použitou desku Spartan-3AN Starter Kit v [22]) vyhledat, na které fyzické vývody je co připojeno. Například pro užití klávesnice (v projektu název periferie *xps_ps2_0*) je třeba přidat do UCF následující definice:

```
    Net xps_ps2_0_PS2_1_DATA_pin   LOC = V11 | IOSTANDARD = LVCMOS33 |
DRIVE = 8 | SLEW = SLOW;
    Net xps_ps2_0_PS2_1_CLK_pin   LOC = W12 | IOSTANDARD = LVCMOS33 |
DRIVE = 8 | SLEW = SLOW;
```

Název za klíčovým slovem *Net* musí být shodný s názvem v kategorii *Name* v *External Ports* na kartě *Ports* v *System Assembly View*. LOC udává označení fyzického vývodu pouzdra hradlového pole. *IOSTANDARD* udává druh použitého logiky. LVCMOS33 označuje že se jedná o logický vstup/výstup pro napěťovou úroveň 3,3 V, tedy *Low Voltage CMOS*. Dalšími možnými standardy jsou např LVTTL pro TTL logiku, LVCMOS25, LVCMOS18 (vhodný například pro moderní paměti RAM) a jiné. *DRIVE* udává dovolený výstupní proud v mA, typické používané hodnoty pro periferie jsou 4 a 8. *SLEW* je doba přeběhu při změně úrovně na pinu. Více o možnostech a vlastnostech UCF souboru se lze dočíst například v [28].

4.7.3 Přidání vlastní periferie do systému

Pro zahrnutí periferie s kódem, který byl vyvinut v ISE Project Navigator využijeme průvodce *Create or Import Peripheral* v menu *Hardware*.

- Zobrazí se uvítací obrazovka průvodce, *Next.*
- Pro tvorbu nové periferie vybrat *Create templates for a new peripheral, Next.*

- V dalším kroku vybrat *To an XPS Project* (periferie bude přístupná pouze v daném projektu), *Next.*
- Pojmenovat periferii (nedovolená syntaxe jména či nedovolené znaky jsou okamžitě signalizovány), *Next.*
- Druh připojení – vybrat sběrnici PLB, *Next.*
- Karta *IPIF (IP Interface) Services*. Zde se volí parametry rozhraní mezi periferií a sběrnicí PLB v4.6:
  - *Software reset* umožňuje po zápisu zvláštního slova na zvláštní adresu sběrnice (unikátní pro každou periferii) spustit proceduru restartování dané periferie. Tím je možné restartovat z programu samostatně libovolnou periferii.
  - *Read/Write FIFO* aktivuje režim zápisu jako do paměti FIFO. To je vhodné pro některé paměťové periferie.
  - *Interrupt control* povoluje periferii vyvolat přerušení.
  - *User logic software register* povoluje periferii, respektive uživatelské logice, která ji obsluhuje, mít registry adresovatelné z programu.
  - *User logic memory space* je volba, která dovoluje generovat signály pro adresování více paměťových rozsahů. Hodí se například pro různé paměťové banky.
  - *Include data phase timer* aktivuje možnost jakéhosi „watchdog timeru“ pro paměťovou sběrnici. Pokud je tato možnost zapnuta, má periferie pouze 128 taktovacích cyklů sběrnice na odpověď či reakci, jinak bude operace čtení či zápisu přerušena.
  - *User logic master* vygeneruje příslušné rozhraní, pokud má být periferie v režimu „master“. První tři možnosti jsou podporovány pouze sběrnicí PLB v4.6. Více o těchto možnostech můžeme najít například v nápovědě.

Z Uvedených možností pro periferii modulátoru hodí zaškrtnout pouze *User logic software register* a případně *Software reset*, i když jeho praktické použití je sporné (viz kapitolu4.7.6). Každá zatržená možnost se projeví příslušným způsobem ve vygenerovaných HDL souborech a souborech ovladačů (pro registry a softwarový reset viz kapitolu 4.7.6).

- Na kartě *Slave interface* ponechat výchozí nastavení, *Next.*

- Na kartě *User S/W Register* lze nastavit počet softwarových registrů (*Number of software accssible registers*) periferie. Pro periferii modulátoru nastavit 8 (viz 4.7.7), *Next*
- Na následujících dvou kartách lze ponechat výschozí nastavení.
- Na kartě *Peripheral Implementation* zaškrtnout *Generate template driver files to help you to implement software interface*, což způsobí vygenerování potřebných souborů tak, že již bude jen stačit doplnit popisný Hdl kód periferie. *Next.*
- Pomocí *Finish* dokončíme průvodce.
- Periferie je nyní přístupná na kartě *IP Catalog* (Xilinx používá pro periferie též označení *IP* (Intelligence Property), v různých zdrojích se s nimi lze setkat v zásadě jako se synonymy) pod *Project Local PCores*.

V adresáři projektu, v podadresáři *./drivers/src* by se měla objevit složka odpovídající názvu periferie. Zde se nachází soubor ovladače, jmeno_periferie.h. V něm nalezneme deklarace funkcí pro zápis do registrů a pro vyvolání resetu (viz 4.7.6). Před dalším použitím je potřeba opravit známou chybu, kdy se v tomto souboru používají funkce typu xil_io_in32 a xil_io_out32, které nikde nejsou definované. V celém souboru je tak třeba provést náhradu:

- `xil_io_in32` → `Xio_In32`
- `xil_io_out32` → `Xio_Out32`

Dále by se složka s názvem periferie měla objevit i v adresáři *./Pcore*. Zde se nachází soubory pro popis harwaru periferie. Zde v adresáři *data* je do souboru *jmeno_periferie.mpd* potřeba přidat definice portů, do části uvozené klíčovými slovy *PORT*. Pro modulátor to budou následující:

```
PORT hodiny = "", DIR = I
```

Což je vstupní port hodinového signálu (resp. jeden pin). A potom příslušné výstupy, jak to vyžaduje daná verze modulátoru. Nejjednodušeji tedy

```
PORT vystup_horni = "", DIR = O
PORT vystup_dolni = "", DIR = O
```

Je potřeba zapsat vše zcela bez chyb, protože případná chyba se přenese do dalších soubrů a obtížně se odstraňuje.

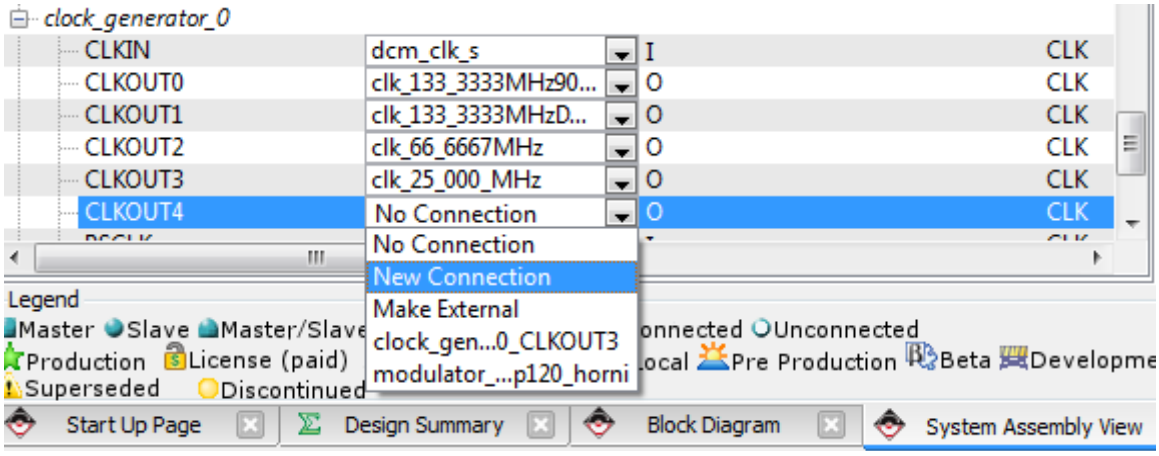
Po tomto úkonu vybereme v XPS možnost Rescan User Repositories z menu Project. To

přenese změny z *MPD* souboru do projektu, například se příslušné porty zobrazí na kartě *Ports* v *System Assembly View*. Změny lze provádět kdykoli, před opětovným nahráním do FPGA je ovšem potřeba vždy vytvořit *Netlist* a *Bitstream*. Tyto změny by syntetizátor měl zaznamenat i bez *Clean Netlist* (viz 4.7.1), lze však doporučit toto preventivně provést.

Po přetažení periferie do *Bus Interfaces* v *System Assembly View* a jejím připojení ke sběrnici je třeba na kartě *Addresses* tamtéž provést *Generate Addresses*. Potom na kartě *Ports* je třeba výstupní porty definovat jako externí. V seznamu najdeme periferii *modulator_0* a u jednotlivých vývodů vybereme z nabídky ve sloupci *Net* možnost *Make External*. Každý výstup, se kterým toto provedeme, se objeví v horní části v seznamu *External Ports*. Název ve sloupci *Name* se ve výchozím stavu jmenuje jako výstup u modulator_0 doplněný on přídavek _pin. Toto jméno můžeme změnit, je však důležité, abychom jej přesně v tomto tvaru použili také v UCF souboru (kapitola 4.7.2).

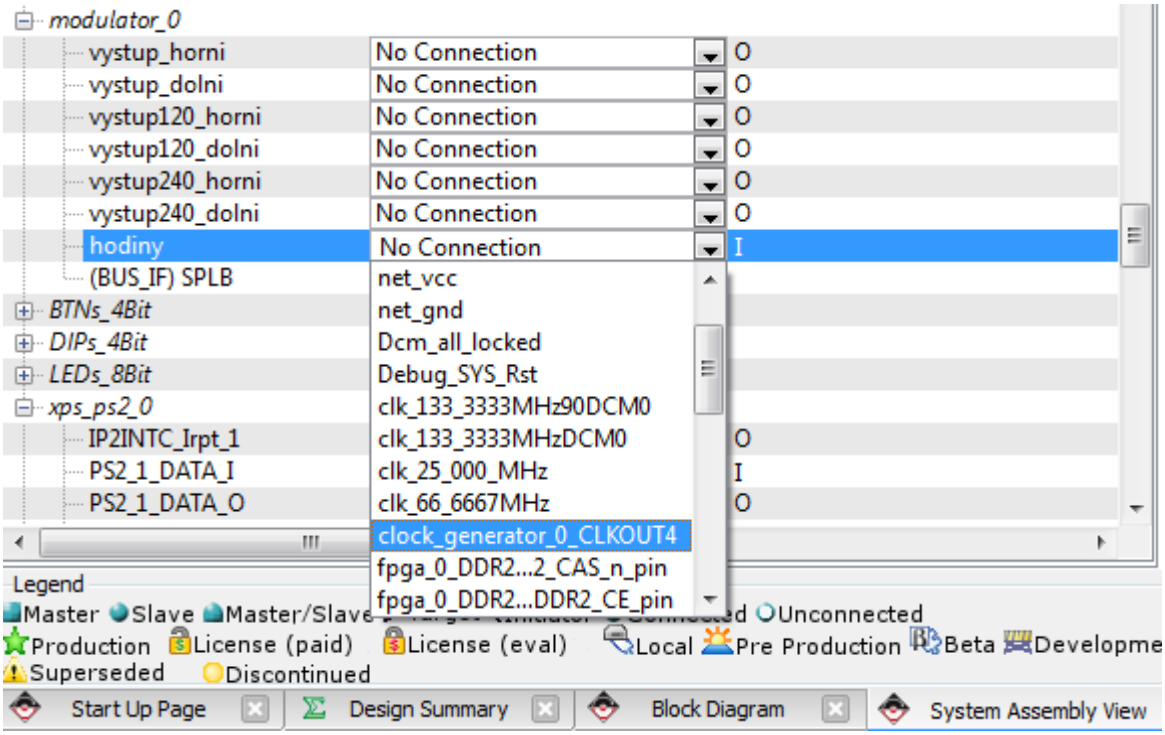
Jiný postup je třeba zvolit pro vstup hodinového signálu do periferie. Nejprve na kartě *Bus interfaces* v *System Assembly View* upravíme vlastnosti periferie *clock_generator* (ta je v projektu vždy již ve výchozím stavu). Dvojklikem se otevře okno *XPS Core Config – clock_generator_0*. V levé části tohoto okna jsou ve sloupci výstupy jednotlivých hodinových signálů. Najdeme první nepoužitý (tedy první, který má v kolonce „Required Frequency (Hz)“ nulu) a zapíšeme do něj 50000000 (50 MHz), pro kteroužto frekvenci je modulátor navržen.

Nově použitý hodinový výstup se potom objeví i na kartě *Ports* (v naše případě jako CLKOUT4). Zatím je u něj napsáno *No Connection*. Vybereme *New Connection* (obr. 4.19).



Obr. 4.19: Nový výstup z periferie clock_generator_0

Takto upravený port není externí, neobjeví se proto v seznamu *External Ports* a není jej tedy třeba ani uvádět v UCF souboru. Vzniklá „síť“ nebo „signál“ je nyní přístupná ve sloupci *Net* pro všechny vstupní porty jako „*clock_generator_0_CLKOUT4*“. U periferie *modulator_0* tak můžeme z nabídky u portu *hodiny* vybrat tento signál (obr. 4.20) a spojení je tak dokončeno.



Obr. 4.20: Spojení výstupu clock_generator_0 a vstupu „hodiny“ periferie modulator_0

4.7.4 Periferie pro obsluhu klávesnice (xPS2)

Přidání periferie obsluhující klávesnici (*XPS PS2 Interface 1.01.b*) je obdobné, jako přidání jiných periferií s vývody vně pouzdra FPGA. V IP katalogu ji najdeme pod kategorií *Communication Low-Speed*. Přidáme periferii do projektu a zapojíme ji na sběrnici PLB. Necháme vygenerovat adresy na kartě *Addresses* a na kartě *Ports* nastavíme vývody PS2_1_DATA a PS2_1_CLK. jako externí pomocí *Make external*. Pak už jen vývody přidáme v UCF souboru. Kam přesně jsou příslušné kontakty procesoru přivedeny, zjistíme v [22]. Pro desku Spartan3AN Starter Kit to bude následující definice:

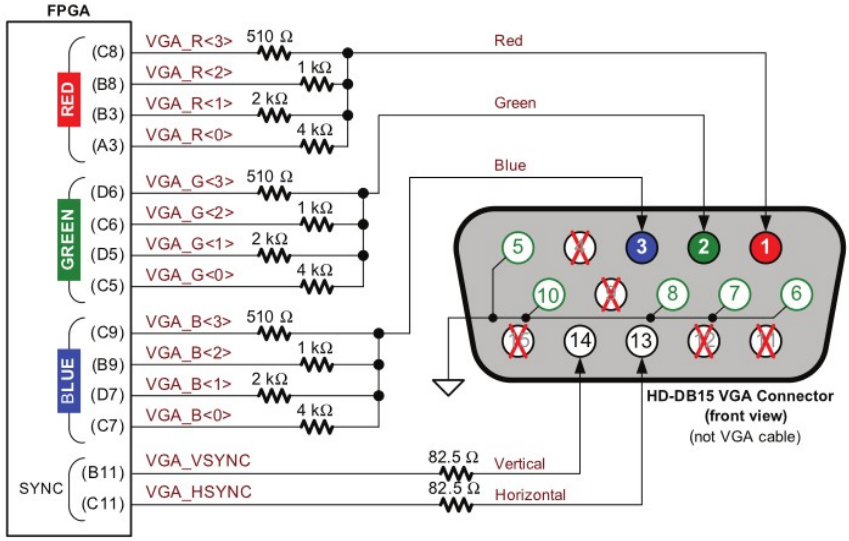
```
    Net xps_ps2_0_PS2_1_DATA_pin   LOC = V11 | IOSTANDARD = LVCMOS33 |
DRIVE = 8 | SLEW = SLOW;
    Net xps_ps2_0_PS2_1_CLK_pin   LOC = W12 | IOSTANDARD = LVCMOS33 |
DRIVE = 8 | SLEW = SLOW;
```

Klávesnice je obsluhována především prostřednictvím přijímaných kódů kláves, které posílá. Zdroj hodinového signálu pro klávesnici (PS2_1_CLK) vytváří periferie sama. Komunikace směrem ke klávesnici je zapotřebí pouze pro rozsvěcení kontrolních LED na klávesnici (*numLock*, *CapsLock*, *ScrollLock*). Ty se musí rozsvítit příkazem poslaným z vnějšku klávesnice, stisk např. klávesy *Caps Lock* způsobí pouze odeslání příslušného kódu klávesy, nikoli rozsvícení příslušné kontroly. Komunikace s klávesnicí je dostatečně popsána v [22]. V programu zajišťuje tento úkon funkce `rozsvit_NUM_LOCK_LED`.

4.7.5 Periferie pro obsluhu monitoru (xTFT)

Zprovoznění periferie xTFT je složitější. Je třeba zapojit do vstupu SYS_TFT_Clk hodinový signál 25 MHz. To se provede obdobně, jako připojení 50 MHz hodinového signálu pro periferii modulátoru (viz kapitolu 4.7.3). Fyzická realizace převodu binární hodnoty barevné složky na analogovou znázorňuje obr. 4.21 Periferie předpokládá šestibitový výstup, na desce Starter Kit je však proveden čtyřbitový výstup. V UCF souboru je třeba i tak definovat pro tyto nezapojené vývody příslušnou napěťovou úroveň, protože ta je ve výchozím stavu jiná, než v ostatních případech používaná úroveň 3,3 V. V případě, že to neprovedeme, ohlásí se při generování bitsreamu chyba „No valid sites“ - autorouter nemá prostor aby provedl spojení těchto nezapojených vývodů jiným napětím s jinou napěťovou úrovní než jsou okolní vývody. Definice vypadá například takto:

```
Net xps_tft_0_TFT_VGA_R_pin<5>    IOSTANDARD = LVCMOS33;
```



Obr. 4.21: Provedení VGA vývodu na desce Spartan 3AN Starter kit [22]

Jak již bylo řečeno, je fyzický výstup čtyřbitový, což znamená 16 možných úrovní každé ze složek R, G, B. Dohromady lze tedy zobrazit 16·16·16 = 4096 barev.

4.7.6 Komunikace přes softwarové registry

Nejsnazším způsobem komunikace s periferií je předávání hodnot přes takzvané uživatelské softwarové registry (*User Software Registers*, dále jen *softwarové registry*). Periferie jich může mít od jednoho do 4096 (zadává se v průvodci vytvořením periferie *Create or Import Peripheral* na kartě *User S/W Register*) a průvodce vytvoří vše potřebné, pro jejich obsluhu, včetně hlavičkových souborů funkcí pro čtení a zápis z programu (volitelně též pro softwarový reset). Jejich standarsní délka je 32 bitů. Z VHDL kódu jsou pak hodnoty těchto registrů přístupné pro čtení i zápis jako signály (signals). Při použití registrů a zároveň systémového resetu ze sběrnice

(*Bus2IP_Reset*) je třeba dávat pozor, protože tento reset se při každém zápisu z programu do libovolného registru nastaví na úroveň logické nuly (tedy se vypne). Tento signál tedy nelze použít jako plnohodnotný reset, který bude např. periferii udržovat ve vypnutém stavu než provedeme její konfiguraci. Jako řešení je možné vyhradit si pro účely resetu jeden bit některého registru, což jsem provedl. Ve VHDL popisu periferie v souboru *user_logic.vhd* jsem vytvořil signál `reset` a do něj namapoval `zapnout_registr(30)` (slv_reg0). Reset periferie se tak provádí pomocí příkazu

```
MODULATOR_mWriteSlaveReg0(XPAR_MODULATOR_0_BASEADDR, 0, 2);
```

Pokud necháme vygenerovaný hlavičkový soubor bez úpravy (po opravení známého bugu s xil_io_in a xil_io_out viz 4.7.3), probíhá zápis do periferie pomocí funkce

```
JMENO_PERIFERIE_mWriteReg(BaseAddress, RegOffset, Data)
```

- *BaseAddress* je adresa periferie v paměťové struktuře. Název konstanty, která tuto adresu zastupuje nalezneme ve vygerovaném souboru xparametrs.h – ovšem pozor, v něm se objeví až po prvním překladu programu (*Build Project*) po vygenerování Netlistu s příslušnou periferií. Ve výchozím stavu je ve tvaru XPAR_JMENO_PERIFERIE_BASEADDR.
- *RegOffset* číslo bitu, od kterého chceme zapisovat (0-31).
- *Data* je vlastní hodnota, kterou do registru chceme zapsat. Délka registru je 32 bitů. Registr je zapisován stylem *Big-Endian*, nula (LSB) je tedy na bitu 31.

Čtení z registru je možné provádět funkcí:

```
JMENO_PERIFERIE_mReadReg(BaseAddress, RegOffset)
```

Významy parametrů jsou stejné jako při četní. Reset (nastavení signálu *Bus2IP_Reset* na '1') se provádí funkcí

```
JMENO_PERIFERIE_mReset(BaseAddress)
```

4.7.7 Obsluha registrů v periferii

Ve vygenerovaném souboru *user_logic.vhd* je kód pro obsluhu sběrnice a softwarových registrů. Hodnoty registrů jsou realizovány pomocí signálů (signals). Standardní názvy jsou slv_reg0 až slv_reg(n-1), podle toho, kolik registrů bylo požadováno. Aby odpovídaly jmenné konvenci signálů, kterou jsem začal používat při vývoji kódu v Project Navigatoru, přejmenoval jsem pro lepší přehlednost tyto výchozí názvy na své. To lze snadno provést nejlépe automatickou

náhradou textu. Bylo by též možno ponechat názvy v původní podobě a vytvořit si vlastní signály a do nich překlápět hodnotu signálů registrových, ovšem nepovažoval jsem to za vhodné řešení i s ohledem na vyšší nároky na hardware. Původní a nové signály korespondují následovně:

```
slv_reg0    →    zapnout_registr
slv_reg1    →    vstup_hran_na_vzorek_sinu
slv_reg2    →    vstup_modulacni_index
slv_reg3    →    vstup_pocet_hran_na_inkrement
slv_reg4    →    vstup_typ_citace
slv_reg5    →    vstup_ochranna_doba_hran
slv_reg6    →    vstup_poc_hod_citac_1
slv_reg7    →    vstup_poc_hod_citac_2
```

`zapnout_registr(31)` je namapován jako signál `zapnout`

`zapnout_registr(30)` je namapován jako signál `reset`

4.7.8 Tvorba Netlistu a bitstreamu

Poslední fází návrhu procesorového systému s procerorem Microblaze v XPS je překlad popisných zdrojových souborů do formy, kterou je možné naprogramovat do FPGA. Funkce *Generate Netlist* (přístupná v menu *Hardware*) provede z popisných souborů syntézu (hlavně souborů popisujících chování periferií v HDL) pomocí XST a dalších součástí. V této fázi se chybovým hlášením projeví případné chyby v popisu periferie. V této fázi je nutné připojení k internetu z důvodu ověření licence.

Dále je před nahráním do FPGA provede vytvoření *bitstreamu*, což je „datový proud“ ve formátu, který je přímo možné nahrát do FPGA. V této fázi probíhá automatické umisťování prvků a trasování spojení mezi nimi ve struktuře hradlového pole. Při tom je využit obsah UCF souboru, popisující fyzické vyvedení signálů na vývody pouzdra hradlového pole.

Pokud je při tvorbě *bitstreamu* ohlášena chyba *Constraint not met <jmeno_signalu>*– znamená to, že „autorouter“ neumí natrasovat spojení z vnitřní struktury na požadovaný vývod popsaný v UCF souboru tak, aby byly splněny parametry pro dovolené zpoždění signálu. Pak je nutné vyzkoušet nějaký jiný vývod (tedy v UCF souboru změnit u příslušného signálu položku LOC).

Tato chyba se též může vyskytnout v trochu jiném formátu a to vypsáním:

```
    ERROR: 1 constraint not met.
    PAR could not meet all timing constraints. A bitstream will not be
generated.
    To disable the PAR timing check:
    1> Disable the "Treat timing closure failure as error" option from
the Project Options dialog in XPS.
    OR
```

```
    2> Type following at the XPS prompt: XPS% xset
enable_par_timing_error 0
```

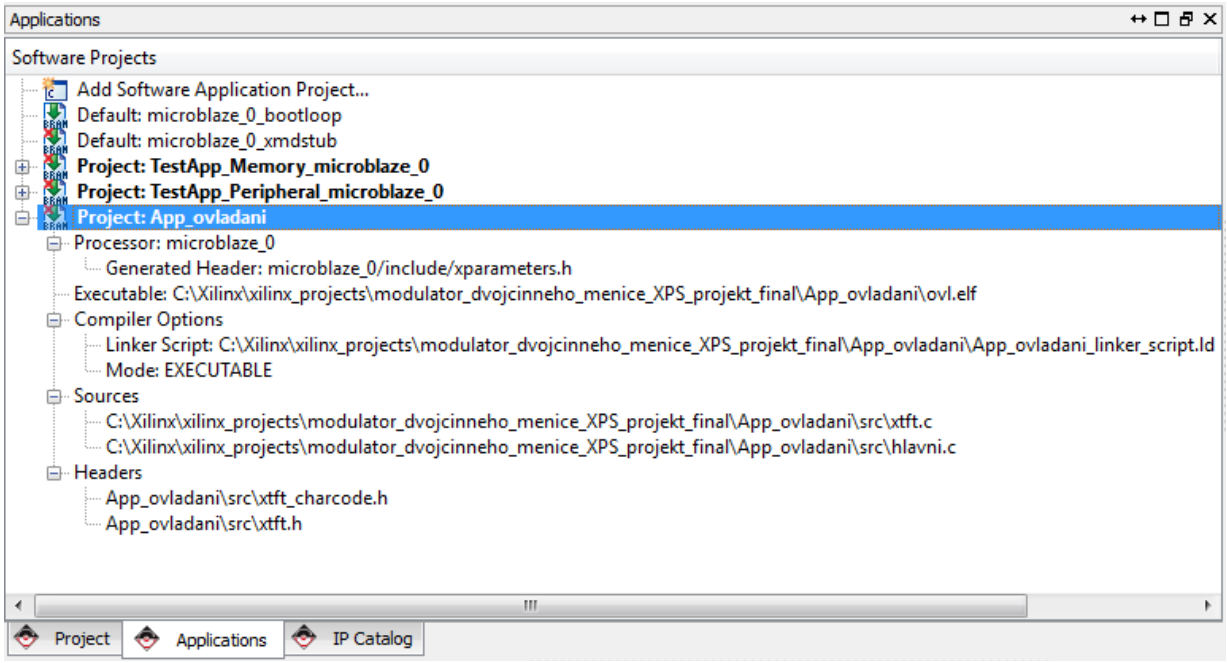
Tuto chybu lze obejít odoznačením položky „*Treat timing closure failure as error*“ na kartě *Design Flow v* Menu *Project – Options.*

Pokud chceme po vytvoření *Netlistu* provést nějaké změny v HDL popisu periferie, je nutné po úpravách vždy znovu vygenerovat *Netlist* i *Bitstream*. Předtím je ale nutné vyvolat z menu *Hardware – Clean Netlist*, dále *Clean Bits* a *Clean Hardware*. Pokud bychom to neprovedli, využijí se při generování různé již syntetizované a napůl zkompilované pomocné mezisoubory a změna v popisu periferie se nikde neprojeví.

4.8 Tvorba ovládacího programu a jeho nahrání

Při práci v XPS (zde v ISE 12.3) setkáme v několika případech s hlášením, že podpora softwarových funkcí v XPS bude v příštích verzích ukončena, přechod do SDK (Software Development Kit) považuji za zbytečné zesložitění. V XPS je výhodou dostupnost všech funkcí pro vývoj celého systému (můžeme editovat všechny systémové soubory návrhu, VHDL kódy periferií, nahrávat hotové návrhy do FPGA, psát programy v jazyce C, kompilovat je a nahrávat do procesoru) z jednoho místa.

Nejsnazším způsobem, jak v XPS vytvořit program pro procesor je upravit zdrojový soubor jedné z automaticky vygenerovaných aplikací. Vytvoření nové aplikace však není složité, stačí poklepat na položku *Add Software Application Project...* na kartě *Applications* (4.22) a zadat jméno. Zdrojový soubor v jazyce C přidáme kliknutím pravým tlačítkem na *Sources* a vybráním možnosti *Add New File...* a *Add Existing File...V* případě zvolení první možnosti následuje dotaz, kam se má nový soubor uložit. Je vhodné si pro přehlednost v adresáři projektu (tedy tam, kde se nachází soubor system.xmp) za tímto účelem vytvořit složku, podobně jako má třeba vygenerovaná aplikace *TestApp_Peripheral_Microblaze_0*. Potom je třeba vygenerovat *Linker Script*. To provedeme kliknutím pravým tlačítkem na jméno naší nové aplikace a vybráním *Generate Linker Script*. Po odkliknutí hlášení, že softwarové funkce budou opuštěny se dostáváme k nastavení. Ve vyvolaném okně je přehled, na jaká paměťová místa se budou jednotlivé bloky programu ukládat. Ve výchozím stavu se vše ukládá do BRAM procesoru (zde pojmenováno ilmb_cntlr_dlmb_cntlr). Používáme-li v projektu další paměť (např. DDR), je možné nastavit i tu, ale toto řešení se mi nepodařilo zprovoznit. Důležité jsou kolonky, dotazující se na jméno a umístění zkompilovaného ELF a vygenerovaného *Linker Scriptu*. Potvrzením se vygeneruje *Linker Script*, což se projeví vypsáním cesty k němu pod položkou *Compiler Options*. Program se zkompiluje pomocí *Build Project* z nabídky vyvolané pravým tlačítkem myši na názvu aplikace. Tím je na zadané cestě vytvořen ELF soubor.



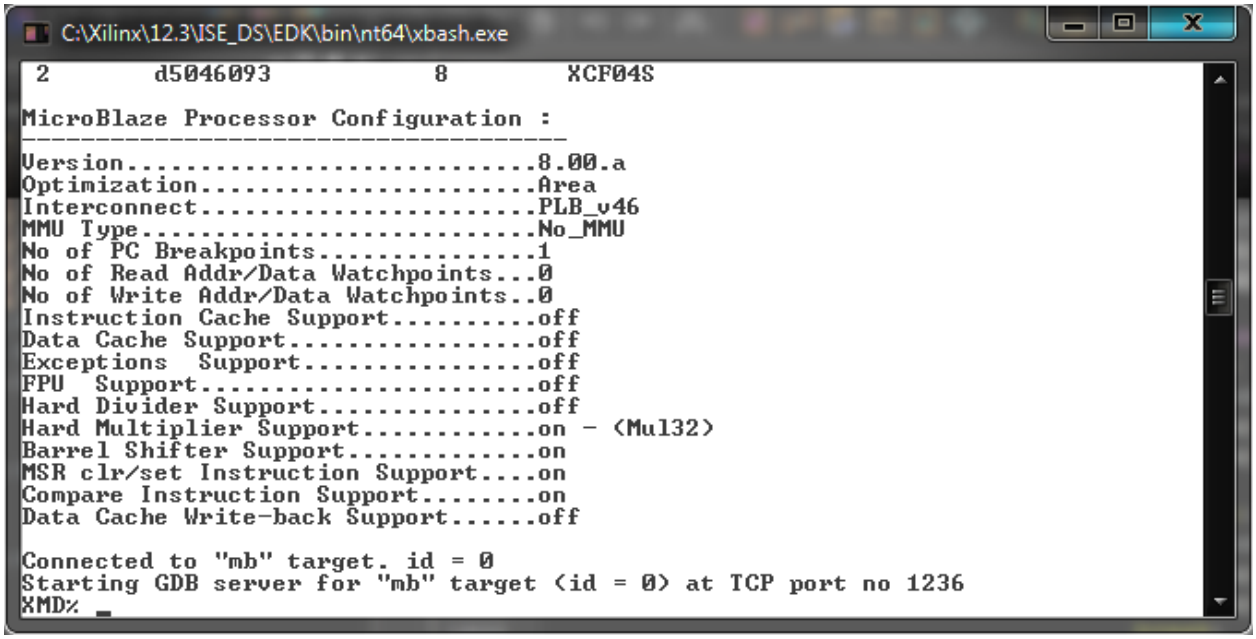
Obr. 4.22: K popisu karty Applications

4.8.1 Nahrání aplikace

Nahrát hotový program do procesoru lze dvěma základními způsoby. Lze vybrat *Mark to Initialize BRAMs* (nabídka pravým tlačítkem myši na název projektu), což zaintegruje data programu přímo do bitstreamu nahrávaného na desku. Takto lze označit pouze jednu aplikaci (tento stav je indikován zmizením červeného křížku u její ikonky v seznamu) a před nahráním bitstreamu je zapotřebí ještě spustit z menu *Device Configuration – Update Bitstream.*

Druhou možností je použití bootloop smyčky pro inicializování paměti a načítání aplikace z externí paměti. To lze provést tak, že v seznamu aplikací se jako *Mark to Initialize BRAMs* označí aplikace Microblaze_0_bootloop, provede se *Update Bitstream* a bitstream se nahraje do desky. Následně se spustí součást EDK zvaná XMD (Xilinx Microprocessor Debugger) přes menu *Debug – Launch XMD*. Spustí se terminálový program (obr. 4.23), který určitou dobu provádí inicializaci. Po proběhnutí inicializace můžeme psát příkazy.

4.8.2 Práce s XMD



Obr. 4.23: Komponenta XMD po inicializaci

Příkazem *PWD* zobrazíme aktuální cestu (výchozí je standardně v adresáři projektu, tedy tam, kde se nachází *system.xmp*), příkazem *DIR* obsah adresáře. Cestu je potřeba změnit do složky, kde se nachází *ELF* soubor, který chceme procesorem spustit. To provádíme příkazem *CD*, poeodbně jako v MS DOS. O úroveň výše de lze dostat použitím standardního „*CD* ..“.

Následně příkazem *DOW jmeno_elfu.elf* stáhneme *ELF* soubor do paměti. Příkazem *CON* se spustí procesor. Ten spustí bootloop smyčku, kterou má nahranou ve své BRAM a pomocí ní nahraje a spustí program z externí paměti (DDR). Zastavení procesoru se děje příkazem *STOP*. Znovu spuštění se provede příkazem CON (pokračování od místa, kde byl program zastaven) či *RUN* (program je spuštěn od začátku – lze požít jako reset).

Je třeba upozornit, že příkaz **STOP vypíná pouze běh procesoru** (program se zastaví na vykonávané instrukci). **Všechny periferie ovšem běží dál**, tento příkaz nevyvolává jejich reset.

Pomocí postupu s XMD je možné aplikace nahrát do desky mnohem rychleji, než pomocí prvního postupu. Pomocí XMD by také mělo být možné, zpětně nahrávat obsah paměti procesoru případně jednotlivých proměnných (příkaz MRD_VAR), to se mi ovšem nepodařilo zprovoznit. Více o možnostech a příkazech v XMD lze nalézt v [26].

V projektech, které jsou součástí přílohy této práce, je *ELF* soubor ve složce App_ovladani pod názvem ovl.elf (ovládání).

4.9 Popis ovládacího programu

Hlavní zdrojový kód programu je obsažen v souboru hlavni.c v adresáři ./App_ovladani/src. Na kartě Applications v EDK jsou dále přilinkovány upravené soubory xtft.c, xtft.h a xtft_charcode.h, rovněž z adresáře ./App_ovladani/src.

Include

xtft.h ovladače VGA výstupu; deklarace provedena zápisem #include "xtft.h", kde uvozovky značí načítání z pracovního adresáře (zde ./App_ovladani/src/) viz kapitolu 4.9.1.

xps2.h ovladač klávesnice

xstatus.h pomocný pro ovladač klávesnice

xgpio.h pro pomocné vstupy a výstupy

xparameters.h obsahuje definice adres a spol. pro přístup k periferiím

modulator.h ovladač modulátoru

xutil.h systémová knihovna

volitelně, budeme-li chtít použít přerušení a časovač:

xtmrctr.h ovladač časovače

xintc_l.h ovladač pro interrupt controller

Definice konstant

```
#define TFT_DEVICE_ID       XPAR_TFT_0_DEVICE_ID
#define PS2_DEVICE_ID       XPAR_PS2_0_DEVICE_ID
```

je generovaná mapa konstant pro identifikátory zařízení VGA monitoru a klávesnice ze souboru xparameters.h (začínají písmeny „XPAR“) pro ovladače periferií

```
#define TFT_FRAME_ADDR      XPAR_MPMC_0_MPMC_HIGHADDR - 0x001FFFFF
```

důležitá konstanta, určuje oblast videopaměti pro ovladač VGA monitoru – adresa je vypočtena jako koncová adresa použité DDR paměti (název konstanty ze souboru *xparameters.h*) minus 2 097 151 B (v hex), což je adresa v paměti DDR, od níž je k dispozici volný prostor 2MB. V dolní části paměti je umístěn program, proto umisťujeme videopaměť na druhou stranu.

Následují definice slovních názvů vybraných barev a několik pomocných konstant uživatelského rozhraní a konstanty omezující rozsahy zadávaných parametrů.

Definice funkcí

Mimo grafických funkcí, popsaných v kapitole 4.9.2, jsou definovány následující funkce:

- *vypsatTextXy* – vypíše text na souřadnice zadanou barvou popředí a pozadí. `TftInstance.ColVal` a `TftInstance.RowVal` zůstávají po použití těchto dvou

funkcích na původních hodnotách, stejně jako nastavená barva (`Xtft_SetColor`).

- *vypsatTextXy_kurzorNechat* – vypíše text na souřadnice zadanými barvami. Barvy `TftInstance.FgColor` a `TftInstance.BgColor`, nechává beze změny. Ovšem nastavení polohy kreslení (ColVal, RowVal) jsou za posledním znakem vypsaného řetězce.
- *vypsatTextXy_kurzorNechat_barvuNechat* – jako předchozí, ale po použití mění i barvy
- *smazatRadek* – vymaže řádku po celé šířce obrazovky barvou pozadí vypsáním mezer
- *rozsvit_NUM_LOCK_LED* – obstará komunikaci s klávesnicí za účelem rozsvícení kontrolky
- *timer_int_handler* – volitelně, pro použití přerušení od časovače
- *TftInicializace* – provede úkoly k nastavení zobrazení
- *Smycka_klavesnice* – obstarává obsluhu klávesnice (včetně inicializace) a menu

4.9.1 Zobrazování textu

Ovladač periferie *xTFT* standardně obsahuje funkce pro výpis textu v sedmibitovém kódování ASCII [20]. Použitý font je bitmapový o rozměrech 8×12 bodů, na obrazovku (periferie xTFT podporuje rozlišení 640×480 bodů) se tedy vejde 80 znaků na řádek ve čtyřiceti řádcích. V souborech ovladačů je ale pravděpodobně nějaká chyba, způsobující, že při zápisu na poslední pozici posledního řádku (pravý dolní roh obrazovky) dojde k zatuhnutí programu. Tvary jednotlivých znaků jsou uloženy v souboru *xtft_charcode.h* ve formě pole dvourozměrných polí (rozměru 8×12) jedniček a nul (jedničky označují body vykreslované barvou popředí, nuly barvou pozadí). Pole je adresováno čísly odpovídajícími kódy vypisovaných písmen v ASCII. Tento soubor se nachází mezi ostatními soubory ovladače periferie xTFT (vISE 12.3 je k dispozici verze xTFT 2.01a, k ní přísluší soubory ovladačů ve složce C:\Xilinx\12.3\ISE_DS\EDK\sw\XilinxProcessorIPLib\drivers\tft_v2_00_a\src, resp. podle toho, kde je ISE nainstalováno), odkud se při kompilování programu kopíruje do ./microblaze_0/include. Mým cílem bylo pro dosažení komfortního zobrazení rozšířit znakovou sadu i pro českou abecedu (malá písmena). Protože zdrojový soubor hlavni.c je kódován ve výchozím kódování operačního systému Windows (pro českou lokalizaci je to *windows-1250*), odpovídají česká písmena v textu souboru hlavni.c pozicím rozšířené ASCII tabulky (8bitové kódování). Pole v souboru *xtft_charcode.h* jsem proto rozšířil na 256 prvků a na pozice odpovídající požadovaným písmenům doplnil jejich obrazy. Zbylé pozice jsou vyplněny nulami, výpis nedefinovaného znaku se tak na obrazovce projeví vypsáním mezery. Znak mimo rozšířenou ASCII nelze vypsat, protože není možné jej uložit již ve zdrojových souborech

programu, nemůže se tedy stát, že by se nějakým znakem pole adresovalo mimo svůj rozsah a došlo tak k vyvolání chyby. Volné pozice je možné doplnit dalšími vlastními znaky (nabízela by se například řecká abeceda) s tím, že ve zdrojových souborech programu je třeba je zapisovat znakem na odpovídající pozici v kódování windows-1250. Například pokud bych doplnil na pozici pole 220 znak „Ω“, musel bych pro jeho vypsání ve zdrojovém kódu programu napsat znak „Ü“. Tabulka znakové sady windows-1250 je přiložena v příloze.

Pro zajištění použití upravené verze souboru xtft_charcode.h, je třeba nakopírovat soubory xtft.c, xtft.h a xtft_charcode.h do pracovního adresáře aplikace (v mém případě ./App_ovladani/src) a přilinkovat je jako zdroje i na kartě Applications v XPS. Přilinkování ve zdrojovém souboru programu se pak provede deklarací:

```
#include "xtft.h"
```

Použití uvozovek zajistí načtení tohoto z pracovního adresáře (zde ./App_ovladani/src) a nikoli ze standardního adresáře pro include (./microblaze_0/include), kam se při kompilování aplikace (prvním po vygenerování *Netlistu*) kopíruje verze z adresáře ovladačů (děje se vždy po přegenerování *Netlistu*). Takto přidaný xtft_charcode.h můžeme editovat přímo v prostředí XPS. Upravený soubor xtft_charcode.h (doplněn o písmena malé české abecedy) je součástí přílohy na DVD.

4.9.2 Grafické funkce

Seznam a podrobný popis funkcí, které jsou součástí ovladačů periferie *xTFT* je popsán v dokumentaci API (lze vyvolat v XPS – System Assembly View kliknutím pravým tlačítkem na řádek periferie – *Driver – View API Documentation*). K ovladači je dodáván i příklad použití (C:\Xilinx\12.3\ISE_DS\EDK\sw\XilinxProcessorIPLib\drivers\tft_v2_00_a\examples\ xtft_example.c, respektive dle konkrétní instalace ISE), kde jsou definovány některé další funkce. Zejména je to funkce `TftWriteString`, která na základě funkce ovladače `Xtft_Write` rozšiřuje funkci pro vypsání jednoho znaku na vypsání řetězce. Dále funkce `TftDrawLine`, která přes složitost svého kódu funguje spolehlivě pouze pro vodorovné a svislé čáry. Čáry šikmé se zobrazují přerušovaně či vůbec.

Na základě těchto funkcí jsem vytvořil další, které usnadňují práci s grafickým rozhraním.

V první fázi vývoje jsem se setkal s problémem výpisu hodnot číselných proměnných na obrazovku. Funkce dodané s ovladačem uměly vykreslovat pouze textové řetězce, číselné proměnné by tedy bylo potřeba převést na text. Použití klasických konstrukcí s funkcemi ze `stdio.h` nebylo možné použít, protože jejich realizace je paměťově náročná a výsledný

program se nevešel do BRAM procesoru. Předtím, než jsem program přesunul do externí paměti DDR, vytvořil jsem jednoduchou funkci `cislovkaNaObraz (u8 i)`, která pomocí funkce `Xtft_Write` a konstrukce *switch* vypisuje na obrazovku znaky 0-9 v závislosti na velikosti parametru. Tuto funkci jsem dále použil v `intNaObraz4 (int i)`, která může zobrazit proměnnou v rozsahu 0-999. Zdrojový kód je jednoduchý:

```
void intNaObraz (int i){
cislovkaNaObraz (i / 100);
i = i - (100 * (i/100));
cislovkaNaObraz (i / 10);
i = i - (10 * (i/10));
cislovkaNaObraz (i);
}
```

Funkce nemá ošetřené vstupní podmínky, ale při překročení rozsahu nezpůsobí chybu, pouze nic nezobrazí. Obdobným způsobem jsem vytvořil funkci `intNaObraz4 (int i)` pro rozsah 0-9999. Zobrazení desetinných čísel pak řeším pomocí dvou proměnných (místa před a za desetinnou čárkou). Funkce `intNaObraz_xy (int i, int x, int y)` a `intNaObraz4_xy (int i, int x, int y)` využívají stejné principy, pouze v parametrech mají navíc polohu, takže již není před použitím třeba volat `XTft_SetPos` . `TftInstance.ColVal` a `TftInstance.RowVal` zůstávají po použití těchto dvou funkcích na původních hodnotách.

Funkce `ilustraceModulace (u16 x, u16 y, u8 Druh)` vykreslí na zadané souřadnice jeden ze čtyř ideových obrázků modulační techniky. Funkce je realizována pomocí tabulky bodů křivky a *for* cyklů přímých čar, vše s relativní adresací.

`UvodniVykresleniMenu (u32 TftDeviceId)` je zodpovědné za vykreslení a vypsání všech prvků grafického uživatelského rozhraní. `VykresliNapovedu (u32 TftDeviceId)` zase za překreslení obrazovky do režimu nápovědy. Parametr *TftDevideId* je ukazatel na tzv. *Tft instance*, což je paměťová struktura zprostředkovávající rozhraní mezi videopamětí, vykreslovanou periferií *xTFT* a programem.

Na závěr je třeba upozornit na skutečnost, že vykreslení čáry na poslední linku obrazovky (souřadnice y = 480) způsobí z neznámého důvodu zatuhnutí programu. Stejně tak zápis znaku na pozici 632,468, jak bylo zmíněno v kapitole 4.9.1.

4.9.3 Obsluha klávesnice a menu

Softwarové vybavení dodávané s vývojovým prostředím obsahuje pouze příklad, který umí zobrazit kód klávesy odeslaný z klávesnice a příklad odeslání dat do klávesnice (myšleno příkazů k rozsvícení kontrolek). Uživatel tak musí obsluhu kláves ošetřit sám. Periferie může

pracovat ve dvou režimech. Buď s přerušením, kdy lze nastavit načítání dat z klávesnice do bufferu a vyvolání přerušení za nastavených okolností (například po načtení určeného počtu kódů) nebo v tzv. „polled mode“, kdy je periferie dotazována z programu a žádné přerušení nevyvolává. To je dle dokumentace k periferii i upřednostňovaný způsob provozu.

Stisknuté klávesy indikuje klávesnice posíláním jednobajtových kódů (tzv. „*scancodes*“). Většina kláves je reprezentována jednobajtově, některé klávesy jsou (označované jako „*extended keys*“) jsou reprezentovány uvozujícím bytem „E0“ a jsou tedy kódovány dvojbajtově (např. klávesy šipek). Dále je rozlišeno stlačení a uvolnění klávesy. Klávesnice posílá kód klávesy (či rozšířené klávesy) po stisknutí a pokud je klávesa držena, posílá její kód následně každých 100 ms [22]. Uvolnění klávesy je reprezentováno bytem „F0“ následovaným kódem klávesy (či rozšířené klávesy, tedy navíc s „E0“). Příklad přijatých dat tedy může vypadat následovně:

- stisknuta šipka nahoru: `E0,75`
- uvolněna šipka nahoru: `E0,F0,75`
- stisknuta klávesa „M“ `3A`
- uvolněna klávesa „M“ `F0,3A`

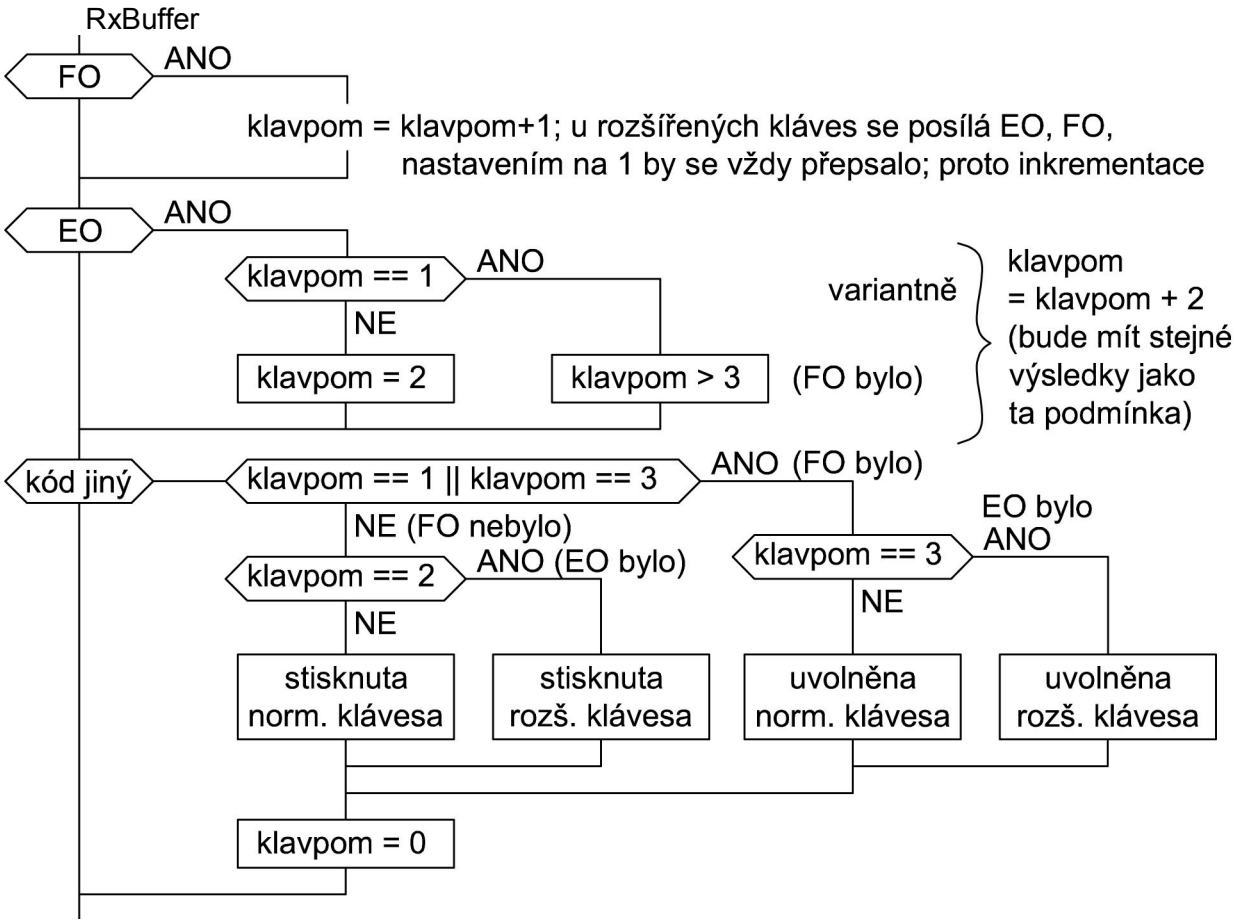
Klávesnice neposílá zvláštní kódy při současném stisknutí *Shift*, *Ctrl* či *Alt* s další klávesou, toto musí rozlišit aplikace.

Z programu je kód klávesy přístupný funkcí

```
u32 XPs2_Recv (XPs2 * InstancePtr, u8 * BufferPtr, u32 NumBytes)
```

kde `Xps2 * InstancePtr` je ukazatel na strukturu ovladače, `u8 * BufferPtr` je ukazatel na proměnnou, do níž se má načtený kód uložit a `u32 NumBytes` je počet bajtů, který se má načíst. V režimu „*polled mode*“ může mít pouze hodnotu 1. Funkce vrací počet načtených bajtů, což v režimu „*polled mode*“ není k užitku. K uložení hodnoty načteného kódu klávesy je v programu použita proměnná `u8 RxBuffer`.

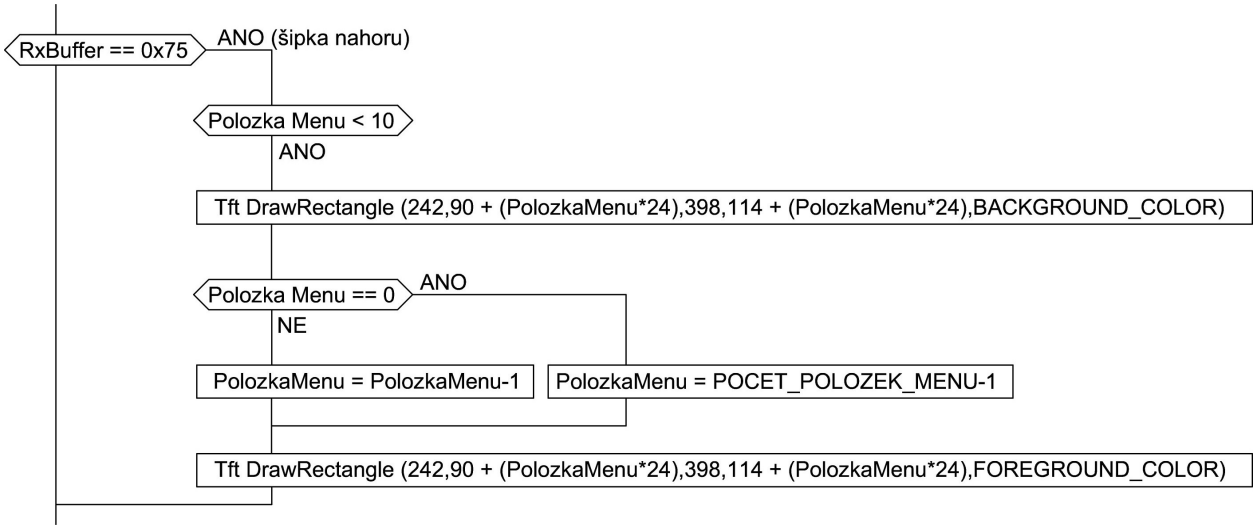
Pro ovládací program jsem navrhl a použil strukturu znázorněnou na obr. 4.24.



Obr. 4.24: Vývojový diagram kódu pro obsluhu klávesnice

Obsluha jednotlivých částí menu pak probíhá v blocích „*uvolněna normální klávesa*“ a „*uvolněna rozšířená klávesa*“. V nich je vyhodnocen kód klávesy a provedeny příkazy, které dané klávese přísluší. Stisk kláves je ignorován, aby nedošlo k vícenásobnému zachycení kódu stejné klávesy a tudíž několikanásobnému nechtěnému vykonání příkazů (viz výše).

Na obr. 4.25 je znázorněna struktura kódu pro obsluhu klávesy „šipka nahoru“ při jejím uvolnění. Klávesy „šipka nahoru“ a „šipka dolů“ řídí posun v menu na obrazovce. Pokud je řídící proměnná *PolozkaMenu* menší než deset, je menu v základní úrovni a posunem obdélníku je vyznačena vybraná položka. Stisknutím enteru se vstoupí do režimu úpravy vybrané položky. Řídící proměnná *PolozkaMenu* je zvětšena o deset, čímž se příkazy v bloku šipek přestanou vykonávat. Ostatní klávesy pak reagují podle toho, která položka menu byla vybrána, tedy například šipky doleva a doprava, backspace nebo klávesy číselného bloku.



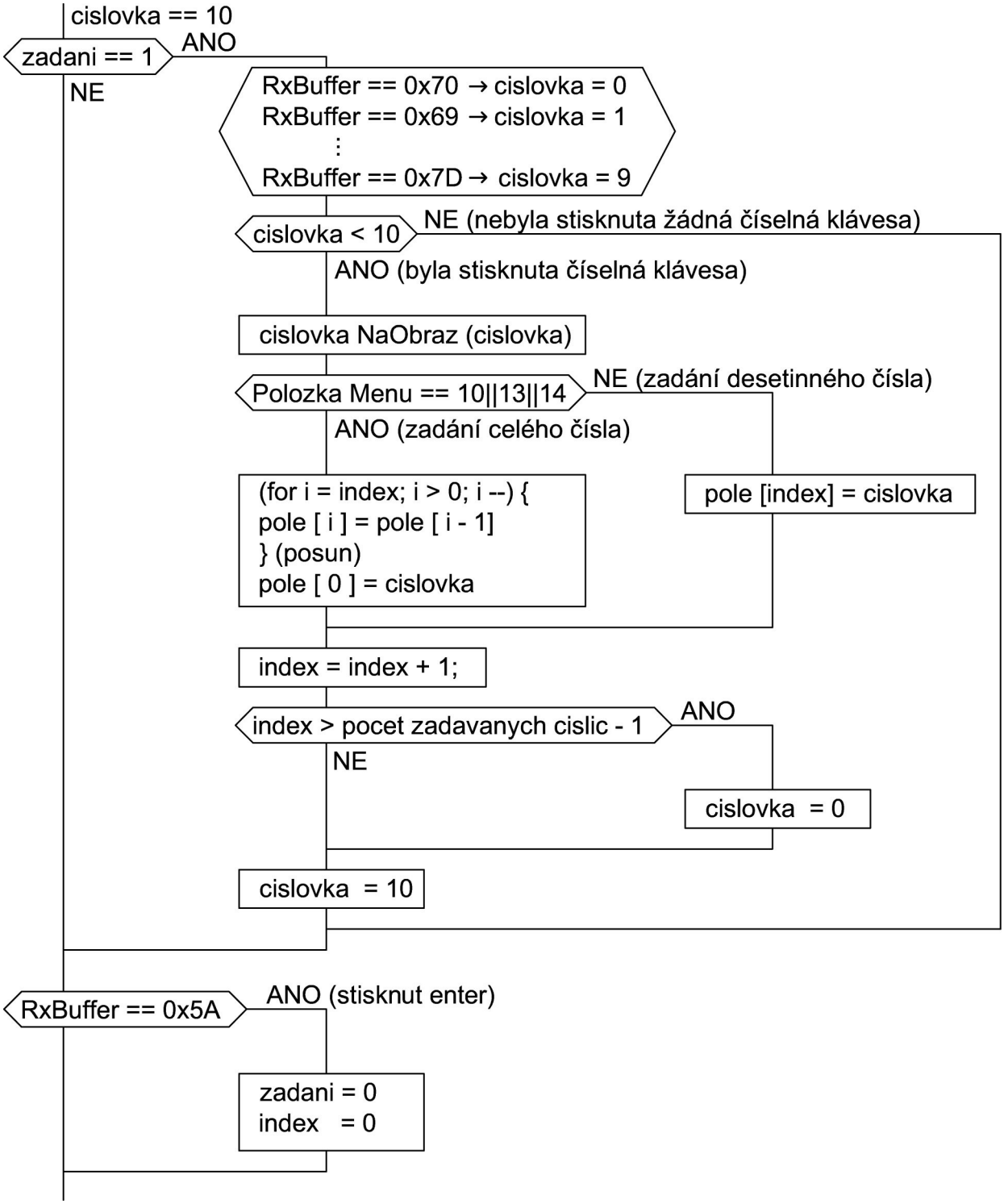
Obr. 4.25: Struktura části kódu pro obsluhu menu

Klávesy číselného bloku jsou zachytávány po nastavení řídící proměnné *zadani* na hodnotu 1. V souvislosti s nimi bylo potřeba vyřešit problém, jak jednotlivé číslovky převést na číselnou proměnnou (např. typu *integer*).

Za tím účelem jsem vytvořil pole, v němž každé místo odpovídá jednotkám, desítkám, stovkám atd., případně desetinnám, setinám... Je nutné rozlišit, zda je zapisováno číslo celé, či desetinný rozvoj. Pokud uživatel napíše v případě desetinného čísla jednu číslovku za desetinnou čárkou, jedná se o desetinu. Pokud napíše dvě číslovky, první bude stále desetina. V případě čísel před desetinnou čárkou je ale situace jiná. Napíšeme-li jednu číslovku, jedná se jednotku. Pokud ale napíšeme dvě číslovky, musí být první zadaná číslovka vyhodnocena jako desítka. Potom je nutné jednotlivé číslovky v poli posunout. Zadávání je ukončeno opět stiskem klávesy *enter*. Zadané číslovky se pak na číselnou proměnnou převedou následujícím příkazem (příklad pro hodnotu *Nova_frekvence*):

```
Nova_frekvence = pole[0] + pole[1]*10 + pole[2]*100;
```

Desetinná čísla reprezentuji v programu čísly celými (viz kapitolu 4.9.2) Vývojový diagram části kódu, která řeší obsluhu zadávání číselných hodnot, ukazuje obr. 4.26.



Obr. 4.26: Struktura kódu zachytávání číslovek

4.10 Výpočty hodnot pro modulátor

Parametry zadané prostřednictvím uživatelského rozhraní musí být uzpůsobeny do formátu, který vyžaduje periferie modulátoru.

Frekvence modulačního signálu (v menu vystupuje jako *Frekvence*) je do modulátoru předávána jako celé číslo (proměnná *vstup_hran_na_vzorek_sinu*), reprezentující počet hodinových cyklů, po který má na výstupu generátoru sinu být jeden vzorek z LUT. Výpočet by se lišil podle zvoleného počtu vzorků na periodu, resp. půlperiodu (viz 4.3.6). Pro počet vzorků, který jsem použil (504 pro šestipulzní, tří a pětiúrovňový měnič, 500 pro dvojčinný jednofázový) bude obecný vztah následující:

$$vstup\ hran\ na\ vzorek\ sinu = \frac{1}{f} \cdot \frac{1}{T_{\mathrm{clk}}} \cdot \frac{1}{n_T} \tag{4.3}$$

kde f je zadaná frekvence modulačního signálu, T_{clk} je perioda hodinového signálu modulátoru a n_{T} je počet vzorků za periodu. Protože T_{clk} i n_{T} je konstantní, je v programu výpočet realizován podílem konstanty a frekvence. Pro 504 vzorků:

$$vstup\ hran\ na\ vzorek\ sinu = \frac{99206{,}35}{f} \tag{4.4}$$

Pro 500 vzorků:

$$vstup\ hran\ na\ vzorek\ sinu = \frac{100\,000}{f} \tag{4.5}$$

Protože uživatel zadává frekvenci v celých číslech, což ne vždy odpovídá celému číslu počtu hodinových period na jeden vzorek, je na obrazovce zobrazena hodnota frekvence, která odpovídá zaokrouhlenému počtu hodinových period.

Hloubka modulace je v periferii modulátoru realizována pomocí násobení základního vzorku z LUT desetinným číslem. Hloubka modulace je předávána jako celé číslo v rozsahu 0-99. Protože ve VHDL není práce s reálnými čísly syntetizovatelná, stejně, jako pomocí XST není sysntetizovatelné dělení jinými čísly, než mocninami dvou, nebylo možné původní návrh

```
sinus_vystup <= conv_std_logic_vector
          (249-(pole_sinu(citac_2)*modulacni_hloubka/100),9);
```

realizovat. Číslo 249 ve výpočtu je hodnota „nuly“ osy *y*. Verzi

```
sinus_vystup <= conv_std_logic_vector
          (249-(pole_sinu(citac_2)*modulacni_hloubka/128),9);
```

už ale syntetizovat možné je (nedělí se 100 ale 128), je však nutné odpovídajícím způsobem upravit i posílanou hodnotu `modulacni_hloubka`. Výpočet je prováděn následovně:

```
vstup_modulacni_hloubka = (u8)(float)(((float)
                    (Hloubka_modulace*128)/(float)1000)+0.5);
```

Přičítání 0,5 ve výpočtu zajistí správné zaokrouhlení na celé číslo (převod na celočíselný

typ proměnné *u8*).

Frekvence nosného signálu je předávána také hodnotou počtu period hodinového signálu. Výpočet probíhá obdobným způsobem, jako u výpočtu frekvence modulačního signálu, jen je potřeba rozlišit výpočet pro trojúhelníkovitý a pilovitý nosný signál. Trojúhelníkovitý signál má dvojnásobný počet vzorků na periodu (náběžná a klesající rampa), proto pro dosažení stejného kmitočtu platí u obou typů platí

$$n_{hpila} = 2 \cdot n_{h\Delta} \tag{4.6}$$

kde n_{hpila} je počet period hodinového signálu na jeden vzorek (ve zdrojových kódech označuji častěji jako inkrement) u pilovitého signálu a $n_{h\Delta}$ trojúhelníkovitého (viz 4.3.7). V programu je výpočet realizován (pro 1000 vzorků na periodu u trojúhelníkovitého signálu) následovně

$$vstup\ \ pocet\,hran\,na\,inkrement = \frac{50\,000}{f} \tag{4.7}$$

I v případě nosné frekvence je hodnota posílaná do modulátoru zaokrouhlena na celá čísla, proto se na obrazovce zobrazuje nikoli zadaná, ale tomu odpovídající přepočtená hodnota.

Ochranná doba je zadávána uživatelem přímo jako násobek počtu period hodinového signálu, proto je možné ji do modulátoru předávat rovnou v zadaném tvaru.

Volba *Modulační techniky* sama o sobě žádný výpočet nepotřebuje, protože je však nezávisle na ní zadávána i frekvence nosného signálu, je potřeba při přechodu z trojúhelníkovitého na pilovitý signál (a obráceně) přepočítat hodnotu proměnné *vstup_pocet_hran_na_inkrement* (vynásobit resp. vydělit dvěma).

Hodnoty jsou do modulátoru předávány jednak při potvrzení volby *Zapnout* dále pak při potvrzení volby Převzít hodnoty (v tom případě se předává pouze hodnota *Hloubky modulace* a *Frekvence modulačního signálu*). V prvním případě je potřeba předat ještě hodnoty pro „*phase advancig*“ - fázový posun modulačního signálu (viz 2.3.3). Ten se odvíjí od zvolené modulační techniky, proto není vhodné jej počítat při zadávání frekvence, ale právě při předávání hodnot do modulátoru, kdy už se modulační technika nemůže měnit.

Pro modulační techniku *Symmetrical Double Edge* (trojúhelník se symetrickým generováním) je potřeba posun o polovinu periody nosného signálu:

$$posun = \frac{n_{h\Delta} \cdot n_{vz1/2T}}{n_{\sin}} \tag{4.8}$$

Kde $n_{vz1/2T}$ je počet vzorků na půlperiodu nosného signálu a $n_{\sin}$ je počet period

hodinového signálu na jeden vzorek sinu (reprezentován proměnnou *vstup_hran_na_vzorek_sinu*). Součin ve jmenovateli tak udává polovinu periody nosného signálu v periodách hodin. Podíl s $n_{\sin}$ (též v periodách hodin) určí, kolik vzorků sinu je za tuto dobu normálně vygenrováno a o tolik je tedy nutné posunout adresování z LUT. Celá část hodnoty *posun* je potom vstupní hodnotou pro čítač 2 generátoru sinu (proměnná *vstup_poc_hod_citac_2*), zlomková část přenásobená tisícem je počáteční hodnotou čítače 1 generátoru sinu (proměnná *vstup_poc_hod_citac_1*).

Aymmetrical Double Edge (trojúhelník s asymetrickým generováním) vyžaduje posun o čtvrtinu periody:

$$posun_{as} = \frac{\frac{n_{h\Delta} \cdot n_{vz1/2T}}{2}}{n_{\sin}} \tag{4.9}$$

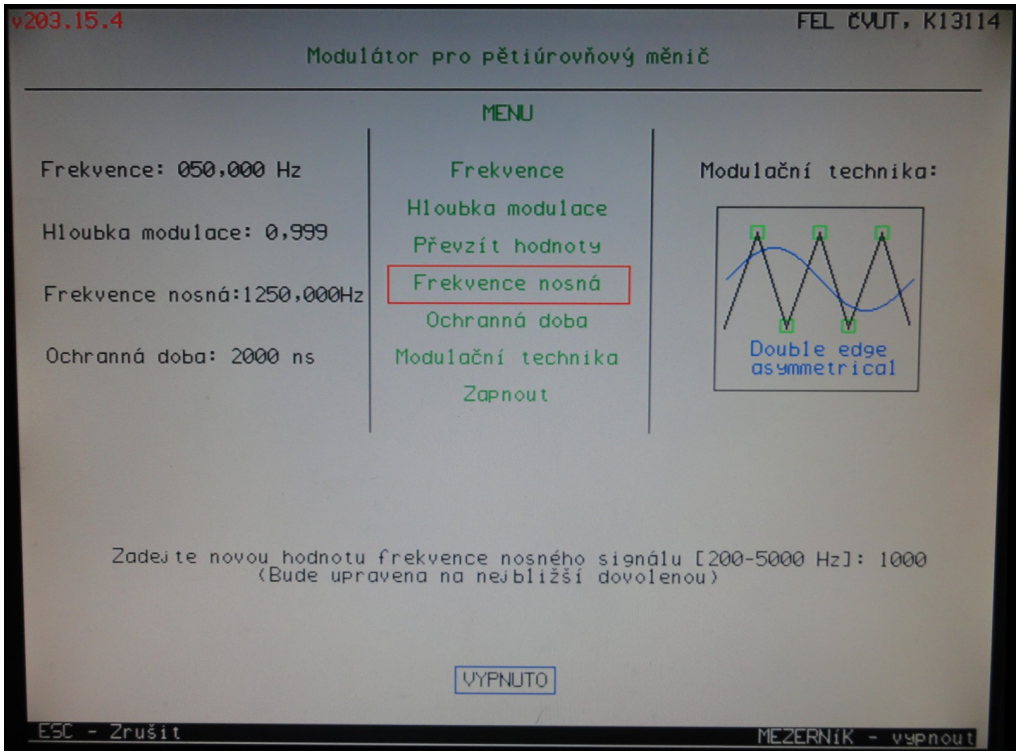
Rising edge a *falling edge* (s pilovitými signály) vyžadují posun o polovinu periody, protože ale mají poloviční počet hodinových period, je výpočet stejný, jako *Aymmetrical Double Edge*. Hodnoty pro oba čítače generátoru sinu se z vypočítaného posunu zjišťují stejně jako v prvním případě.

Po výpočtu všech hodnot se tyto přenesou do periferie pomocí funkcí ovladače (kapitola 4.7.6). Seznam registrů a jim odpovídajících proměnných je uveden v kapitole 4.7.7.

4.11 Popis uživatelského rozhraní

Uživatelské rozhraní je realizováno pomocí klasické 100/101 klávesové PC klávesnice připojené přes rozhraní PS/2 a klasický PC monitor. Zobrazení na monitoru ukazuje obr. 4.27.

Ve střední části obrazovky se nachází menu, umožňující zadávat parametry modulátoru. Vybraná položka je označena červeným obdélníkem, výběr se provádí šipkami nahoru a dolů. K úpravě vybrané položky je třeba stisknout enter, zadanou hodnotu opět potvrdit klávesou enter (podporován je velký enter i enter v číselném bloku klávesnice). Zadávání lze vždy zrušit klávesou escape, což oznamuje i dolní řádek obrazovky. Pro opravu zadání lze využít i klávesu backspace. V levé části obrazovky se zobrazují nastavené hodnoty, v pravé je ideovým obrázkem a popisem udána zvolená modulační technika. Jednotlivé položky menu jsou následující:



Obr. 4.27: Grafické uživatelské rozhraní modulátoru

- Frekvence - zadání frekvence modulačního signálu. Dovolený rozsah zadání (nutno brát ohled na VHDL kód periferie) lze upravit pomocí konstant v programu. Nová hodnota je vypsána šedě a je použita až po potvrzení pomocí položky menu Převzít hodnoty.
- *Hloubka modulace* - zadání frekvence modulačního signálu. Lze zadat v rozsahu 0 - 0,999. Nová hodnota je vypsána šedě a je použita až po potvrzení pomocí položky menu *Převzít hodnoty.*
- *Převzít hodnoty* - překlopí nově nastavené hodnoty Frekvence a Hloubky modulace do modulátoru. Funguje i v zapnutém stavu,kdy ostatní položky měnit nelze. Překlopení hodnot je provedeno potvrzením kontrolního dotazu (ano - ne, výběr se provádí šipkami).
- *Frekvence nosná* - zadání frekvence nosného signálu. Dovolený rozsah zadání (nutno brát ohled na VHDL kód periferie) lze upravit pomocí konstant v programu.
- *Ochranná doba* - nastavení ochranné doby spínacích součástek (dead time) - zadává se v násobcích periody hodinového signálu (20 ns). Dovolený rozsah zadání (nutno brát ohled na VHDL kód periferie) lze upravit pomocí konstant v programu.

- *Modulační technika* - výběr použité modulační techniky. Výběr se provádí šipkami doprava a doleva, potvrzuje se enterem.
- *Zapnout* - po potvrzení kontrolního dotazu (ano - ne, výběr se provádí šipkami) překlopí do modulátoru vypočítané hodnoty řídících proměnných ze zadaných parametrů a dá povel k zapnutí, tedy generování spínacích půlzů na příslušných vývodech vývojové desky. Zapnutý stav je signalizován nápisem zapnuto/vypnuto v dolní části obrazovky a dále rozsvícenými indikačními svítivkami na desce (LD0 - LD6).

Modulátor je možno kdykoli vypnout stiskem klávesy mezerník. V základní úrovni menu je pomocí klávesy F1 možno vyvolat nápovědu, v níž jsou jednotlivé položky ve stručnosti popsány.

4.12 Oživování

Po zdárném vývoji VHDL kódu pro modulátor dvojčinného, šestipulzního trojfázového, tří a pěti úrovňového trojfázového měniče jsem byl nemile překvapen, že kód, který skvěle fungoval v simulátoru ISIM nelze v podobě periferie v EDK syntetizovat. Komponenta XST ohlašovala při generování Netlistu nové a nové chyby, proto bylo potřeba zdrojový kód od základu přeprogramovat. Hlavní překážkou se ukázala přítomnost více než dvou položek v hlavním bloku *if - elsif* jednotlivého procesu. Zdrojový kód typu, který je ilustrativně zobrazen na obr. 4.28 ohlásil chybové hlášení „Signal *jmeno_signalu* cannot be synthesized, bad synchronous description“ (označení signal se ve výpisu používá i pro proměnné – *variables*).

```
citac: process (clk, reset, zapnout)
       variable pomocny_citac: integer range 0 to 1023 :=0;
begin
       if reset = '1' then
            pomocny_citac :=  0;
       elsif rising_edge(zapnout) then
            pomocny_citac :=  1;
       elsif rising_edge(clk)then
            pomocny_citac :=  pomocny_citac + 1;
       end if;
end process;
```

Obr. 4.28: Ilustrativní VHDL kód ohlašující v XST „Signal pomocny_citac cannot be synthesized, bad synchronous description“

Po odmazání např. řádku „*elsif rising_edge(zapnout)*“ již daný proces pomocí XST syntetizovat lze.

Dále není možné pomocí XST v EDK 12.3 pro použité hradlové pole (xc3s700an)

syntetizovat logiku, která by reagovala na náběžnou i sestupnou hranu (atribut signálu 'event), není tedy možné reagovat na pouhou změnu stavu signálu a využívat tak negování jednobitového signálu jako spoušť pro různé děje. Nelze tedy použít čistě jen podmínku „*elsif signal'event then*“, ale pouze „*elsif signal'event and signal='1'*“, což je ekvivalentní užití podmínky „*elsif rising_edge(signal)*“. Z toho důvodu nelze použít ani strukturu „*elsif signal'event and signal='1'*“ a „*elsif signal'event and signal='0'*“. Všechny části zdrojových kódů zobrazené v této práci pocházejí ze syntetizovatelné verze. Při vývoji VHDL kódu v prostředí ISE je tedy vhodné ověřovat vlastnosti kódu v režimu „*implementation*“ pomocí XST syntetizátoru (Synthetize - XST)

Ukázalo se též, že není možné užít pro zamýšlené časování periferie 50 MHz systémové hodiny procesoru (přes signál sběrnice *Bus2IP_clk*), protože toto časování není podporováno DDR pamětí použitou na vývojové desce. Nutnost užití DDR paměti je dána jednak velikostí ovládací aplikace, která se již nevejde do BRAM paměti procesoru, ale především nutností vyhradit zde paměťový prostor o velikosti 2 MB pro potřeby obrazového výstupu (periferie xTFT). Pokud toto časování zvolíme v BSB a zároveň mezi použité periferie přidáme DDR paměť, BSB ohlásí chybu. Protože BSB neobsahuje možnost „zpět“, je potřeba založit nový projekt s jiným časováním (nejlépe doporučených 133 MHz). Problém bylo nutné vyřešit zvláštním hodinovým signálem z periferie *clock_generator.*

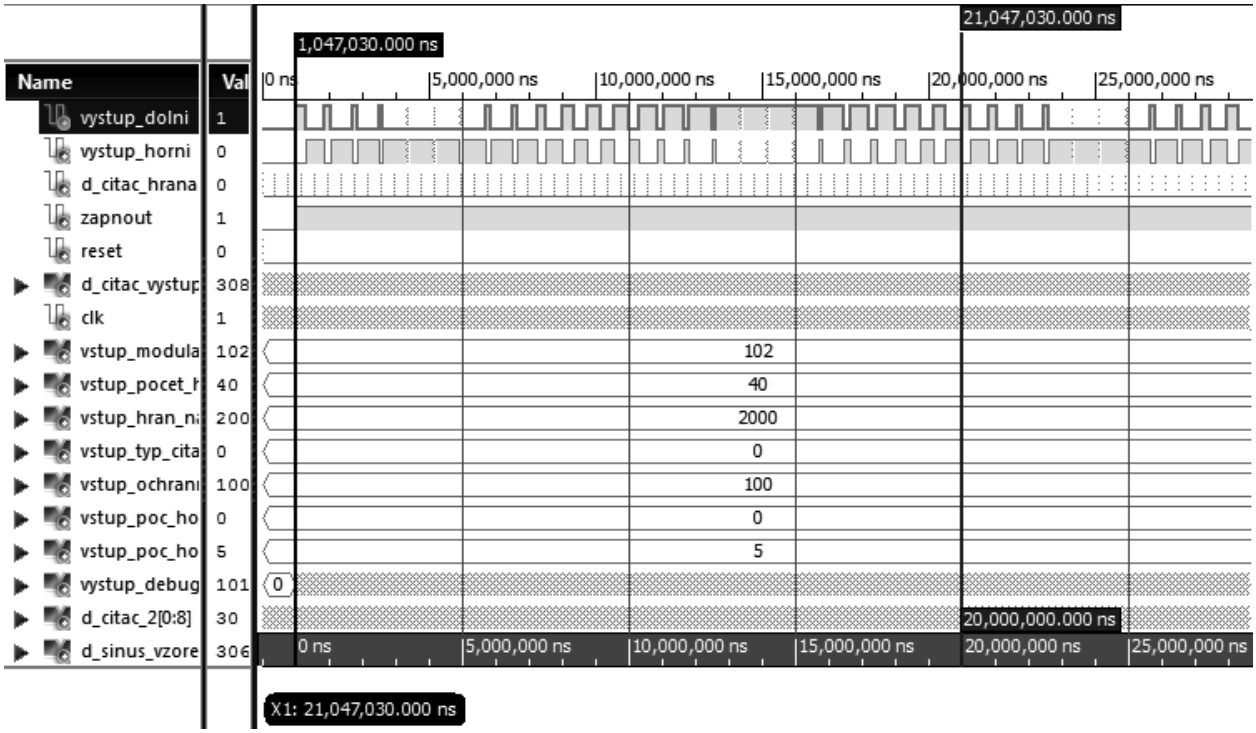
Při oživování se osvědčilo vyvést výstupní piny na nepájivé kontaktní pole a signalizovat nejdříve jejich stavy pomocí svítivek. Pro dignostiku byl dále použit příruční osciloskop Welleman HPS140 a logický analyzátor Asix Sigma.

4.13 Výsledky simulací a měření:

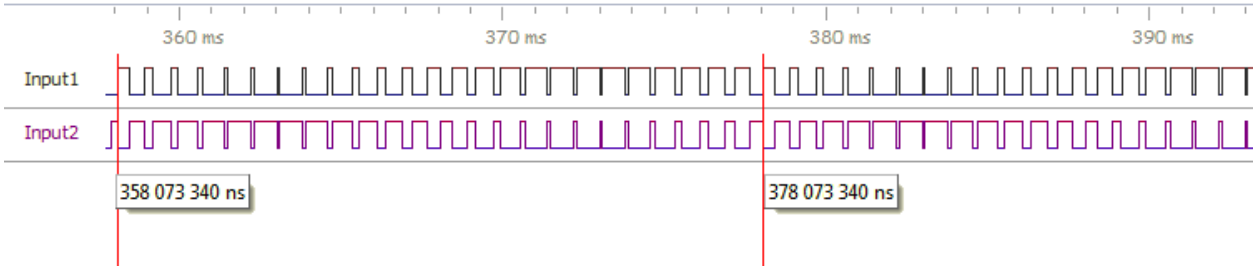
K simulaci využívám program ISIM, který je součástí Xilinx ISE. Ověření skutečných výstupů na vývojové desce provádím pomocí logického analyzátoru Asix Sigma.

Na obr.4.29 vidíme simulaci pro dvojčinný propustný měnič při parametrech 50 Hz, 500 vzorků na periodu sinu, hloubka modulace 0,8, spínací kmitočet 1250 Hz pro *asymmetrical double edge regularly sampled PWM*. Na průbězích vidíme dle předpokladů i centrování pulzů kolem středu periody PWM, což je typická vlastnost *double edge modulací.* Tento střed je vyznačen signálem *d_citac_hrana*, který je spouští pro ovzorkování sinusového průběhu a označuje tedy úvrať čítače. Na obr. 4.30 je průběh totožného kódu se stejným nastavením, ale již v hradlovém poli, změřený logickým analyzátorem. Vidíme, že simulovaný a naměřený průběh si odpovídají a že souhlasí i předpokládané časování. Na zobrazené časové údaje nemají vliv ochranné doby ani u prvního pulzu, neboť jednou z vlastností implementace je i ponechání uplynutí ochranné doby po zapnutí. Na obr. 4.31 vidíme ochrannou dobu, která odpovídá nastaveným dvěma µs.

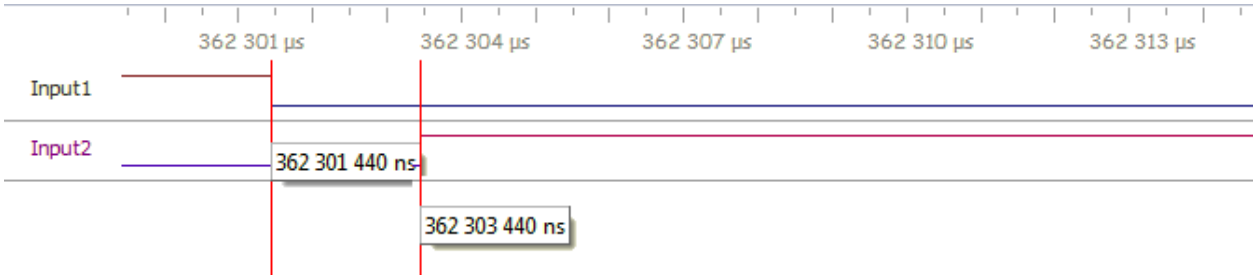
Výsledky simulací a měření:



Obr. 4.29: Modulátor pro dvojčinný jednofázový měnič - průběhy v ISIM, 50 Hz



Obr. 4.30: Průběh z analyzátoru Sigma



Obr. 4.31: Ochranná doba

Další výsledky měření a simulací jsou uvedeny v příloze.

5 Závěr

V první části shrnula tato práce některé poznatky o modulacích pro výkonové měniče a zaměřila se na techniky pulzně šířkové modulace s nosným signálem. Byla popsána kritéria hodnocení modulací a na jejich základě byly porovnány některé vlastnosti vybraných modulačních technik. Na základě těchto teoretických předpokladů byl pomocí hradlového pole a procesorového systému s virtuálním procesorem Microblaze realizován modulátor pro dvojčinný propustný měnič, šestipulzní trojfázový měnič, tříúrovňový trojfázový měnič a pětiúrovňový trojfázový měnič.

K realizaci bylo použito hradlové pole Spartan3AN společnosti Xilinx (konkrétně typ xc3s700an) osazené na vývojové desce Spartan3AN Starter Kit. Modulátor byl realizován formou periferie procesoru Microblaze. Jeho struktura byla popsána v jazyce VHDL ve vývojovém prostředí Xilinx ISE. Funkčnost periferie byla ověřena pomocí simulačního programu ISIM a následně i v hradlovém poli.

K demonstrování širokých možností FPGA bylo pomocí klasického PC monitoru, připojeného přes VGA a standardní počítačové PS/2 klávesnice vytvořeno uživatelské rozhraní, dovolující obsluze přehledně zadávat parametry modulátoru a zobrazovat jeho okamžitý stav.

Jedním z hlavních problémů při realizaci se ukázal rychlý inovační cyklus společnosti Xilinx, který způsoboval těžkosti při získávání odpovídající dokumentace. Společnost Xilinx vydává nejméně každého půl roku nové verze vývojového prostředí, které jsou mezi sebou v mnoha ohledech nekompatibilní. I inovační cyklus hardwaru je velmi rychlý, použité hradlové pole se již například vůbec nevyrábí. To zapříčiňuje nedostatek podkladů pro konkrétní verzi.

Dalším problémem se ukázalo, že simulátor ISIM v pořádku simuluje VHDL kód, který ovšem není přeložitelný (syntetizovatelný) pro hardware. To vedlo v určité fázi realizace k nutnosti přeprogramovat téměř od základu takřka hotový zdrojový kód, což způsobilo značné zdržení.

I přesto se však podařilo realizovat modulátor včetně grafického uživatelského rozhraní dle úvodních předpokladů a ověřit jeho funkčnost nejen pomocí simulace, ale i prakticky pomocí logického analyzátoru.

Seznam použité literatury

[1] HOLMES, D., LIPO, T. Pulse width modulation for power converters: principles and practice. Hoboken, NJ: John Wiley, 2003, xix, 724 p. ISBN 04-712-0814-0

[2] JAVŮREK, J. Regulace moderních elektrických pohonů. První vydání. Praha: Grada Publishing,a.s., 2003, 261 s. ISBN 80-247-0507-9

[3] GOLE, A.M., Course notes in 24.437 Power Electronics, PWM Techniques for Harmonic Reduction in VSC, University of Manitoba, 2000, 8 p. Dostupné z: http://educypedia.karadimov.info/library/spwm.pdf

[4] WALKER, G.R. Digitally-implemented naturally sampled pwm suitable for multilevel converter control. *IEEE Transactions on Power Electronics*. 2003, vol. 18, issue 6, p. 1322-1329. DOI: 10.1109/TPEL.2003.818831. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1243690

[5] BARGE S.A., JAGTAP S.R. Harmonic Analysis of Sinusoidal Pulse Width Modulation. *International Journal of Advanced Electrical and Electronics Engineering*. 2013, vol. 2, issue 5, p.13-16. ISSN: 2278-8948. Dostupné z: http://www.irdindia.in/Journal_IJAEEE/PDF/Vol2_Iss5/3.pdf

[6] URMILA B.,SUBBARAYUDU D. Multilevel Inverters: A Comparative Study of Pulse Width Modulation Techniques. *International Journal of Scientific & Engineering Research*. 2010, vol. 1, issue 3. ISSN: 2229-5518. Dostupné z: http://www.ijser.org/researchpaper%255CMultilevel_Inverters-A_Comparative_Study_of_Pulse_Width_Modulation_Techniques.pdf

[7] http://www.xilinx.com/tools/microblaze.htm

[8] MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 14.1. UG081 (v14.1). Dostupné z: http://www.docshut.com/khpysn/microblaze-ref-guide.html

[9] MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 10.1i. UG081 (v9.0). Dostupné z: www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[10] http://www.xilinx.com/support/answers/41934.htm

[11] PINKER, J., POUPA, M. Číslicové systémy a jazyk VHDL. 1. vyd. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-730-0198-5

[12] WILSON, P. Design recipes fot FPGAs. Amsterdam: Elsevier, c2007, xxii, 289 p. ISBN 978-0-7506-6845-3

[13] HOLMES, D. The significance of zero space vector placement for carrier-based PWM schemes. IEEE Transactions on Industry Applications. vol. 32, issue 5, p. 1122-1129. DOI: 10.1109/28.536874. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=536874

[14] VASCA, F., IANNELLI, L.. Dynamics and control of switched electronic systems: advanced perspectives for modeling, simulation and control of power converters. Springer-Verlag London Limited 2012, xiv, 492 pages. ISBN 978-1-4471-2884-7

[15] VAN DER SPIEGEL, J. VHDL Primer. [online]. [cit. 2013-11-07]. Dostupné z:

http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html

[16] YI-CHU, L., SHAO-WEI, L. Three-phase digital-signal generator sweeps frequency. *EDN Network* [online]. [cit. 2013-11-12]. Dostupné z: http://www.edn.com/design/analog/4363371/Three-phase-digital-signal-generator-sweeps-frequency

[17] ATMEL CORPORATION. *AVR447: Sinusoidal driving of three-phase permanent magnet motor using Atmega48/88/168*. 2006, 26 s. 8010A-AVR-06/06. Dostupné z: http://www.atmel.com/Images/doc8010.pdf

[18] AGGARWAL, Y. FPGA design of a controller or three-phase inverters. *Electronics for you*. 2006, roč. 38, č. 8, s. 116-122. ISSN: 0013-516X Dostupné z: http://electronicsforu.com/electronicsforu/articles/hits.asp?id=1287

[19] PAVELKA, J., ČEŘOVSKÝ, Z., LETTL, J. Výkonová elektronika. 3. přeprac. vyd. Praha: ČVUT, 2007, 227 s. ISBN 978-80-01-03626-6.

[20] http://www.asciitable.com/

[21] http://www.xilinx.com/tools/microblaze.htm

[22] Starter Kit Board User Guide. Ug334 (v1.1). Dostupné z: www.xilinx.com/support/documentation/boards_and_kits/ug334.pdf

[23] PRAŽAN, V. Procesor MicroBlaze. Praha, 2013. Bakalářská práce. České Vysoké Učení Technické v Praze.

[24] http://www.sigenol.cz/theory.php

[25] Portable Formats Specification, Version 1.1, dostupné z: http://flint.cs.yale.edu/cs422/doc/ELF_Format.pdf

[26] AGURTO, C. XILINX MICROPROCESSOR DEBUGER (XMD) REFERENCE GUIDE. Dostupné z: http://www.ece.unm.edu/fpga/carla/Lab_microprocessors/XMD commands REFERENCE V.0.2.pdf

[27] http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/platform_studio/ps_c_bsb_configuring_processor_system_settings.htm

[28] Constraints Guide. UG625 (v11.4). Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/cgd.pdf

[29] SKANDA, V. *Sine Wave Generator Using Numerically Controlled Oscillator Module*, AN1523. Dostupné z: http://ww1.microchip.com/downloads/en/AppNotes/00001523A.pdf

[30] CARRARA, G., GARDELLA, S., MARCHESONI, M., SALUTARI, R., SCIUTTO, G. A new multilevel PWM method: a theoretical analysis. *IEEE Transactions on Power Electronics.* vol. 7, issue 3. DOI: 10.1109/63.145137. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=145137

[30] RICHTA, K., ŠALOUN, P. Programovací jazyk C. Vyd. 1. Praha: ČVUT, Elektrotechnická fakulta, 1998, 253 s. ISBN 80-010-1890-3

Přílohy

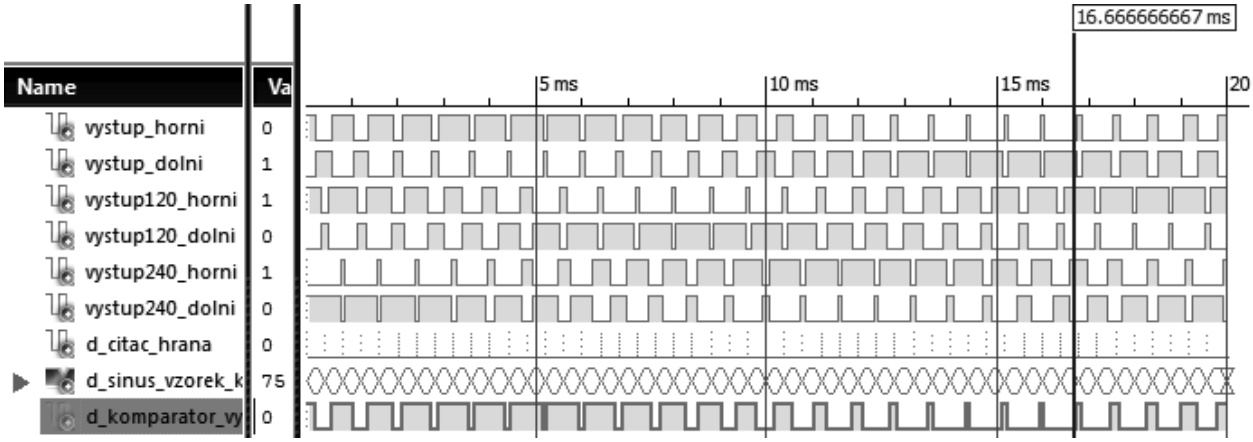
Příloha 1 – Některé další výsledky měření a simulací

Příloha 2 – DVD s podklady k této diplomové práci

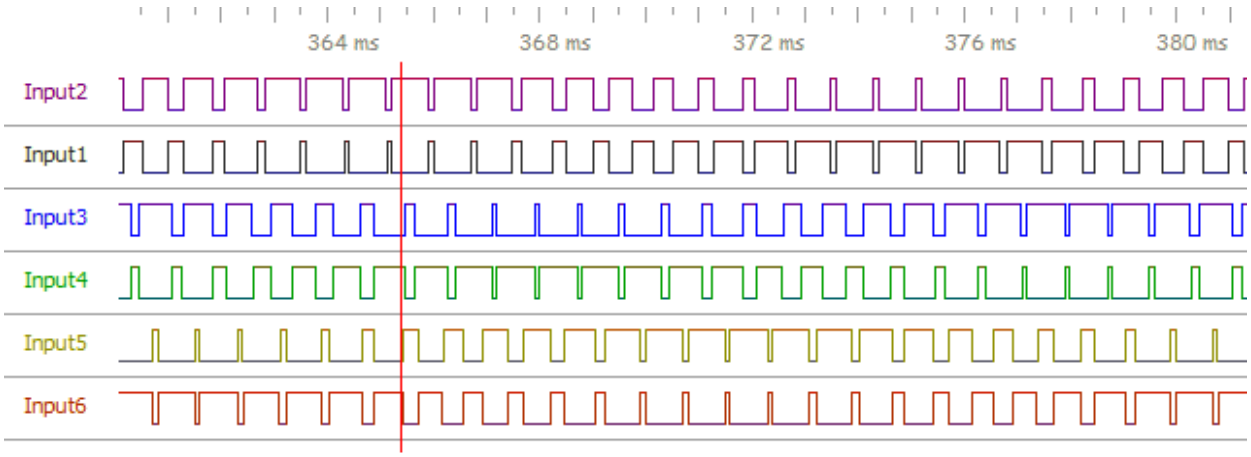
- Elektronická verze této práce ve formátu PDF
- Projekt v XPS modulátor pro dvojčinný propustný jednofázový měnič
- Odpovídající projekt v ISE
- Projekt v XPS modulátor pro šestipulzní trojfázový měnič
- Odpovídající projekt v ISE
- Projekt v XPS modulátor pro tříúrovňový trojfázový měnič
- Odpovídající projekt v ISE
- Projekt v XPS modulátor pro petiúrovňový trojfázový měnič
- Odpovídající projekt v ISE
- Tabulka hodnot dosahovaných frekvencí při různém počtu vzorků v generátoru sinu (ve formátu ODS)
- zdrojový soubor *xtft_charcode.h* s českým rozložením à la windows-1250
- tabulka znakové sady windows-1250 (ve formátu HTML)
- tabulka kódů kláves (ve formátu HTML)
- Počítadlo délky řetězce pro GUI (EXE a projekt ve VB6)
- Vizualizace PWM pro tříúrovňový měnič (EXE a projekt ve VB6)
- Generátor sinového obrázku (EXE a projekt ve VB6)
- Generátor sinus LUT (EXE a projekt ve VB6)

Příloha 1 – Některé další výsledky měření a simulací

Trojfázový trojpulzní měnič, 504 vzorků na periodu sinu, $f = 50{,}0032$ Hz; $m_h = 0{,}8$; ochranná doma 3000 ns, $f_{nosná} = 1250$ Hz, double edge asymmetrical regularly sampled PWM

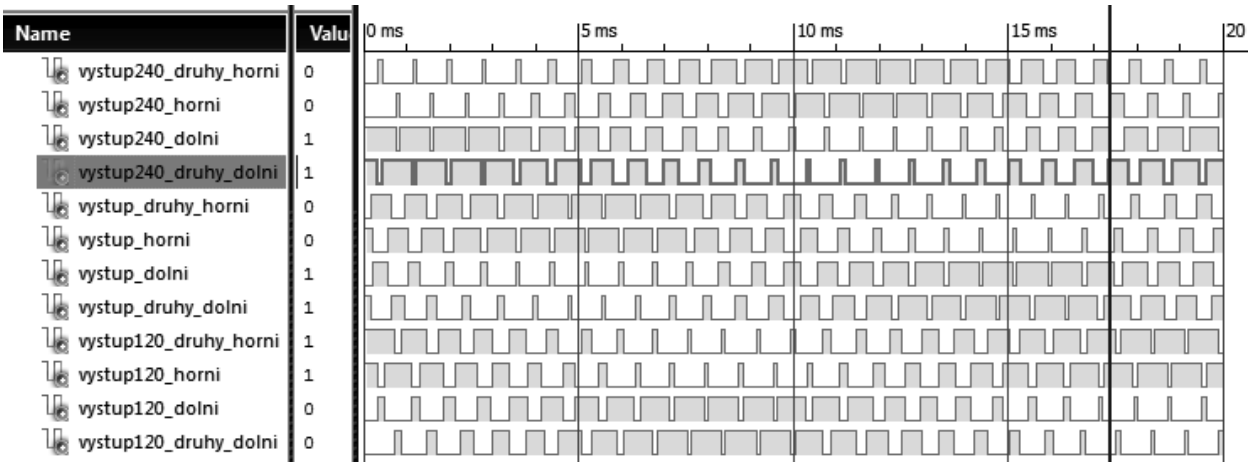


Obr. 5.1: Simulace v ISIM

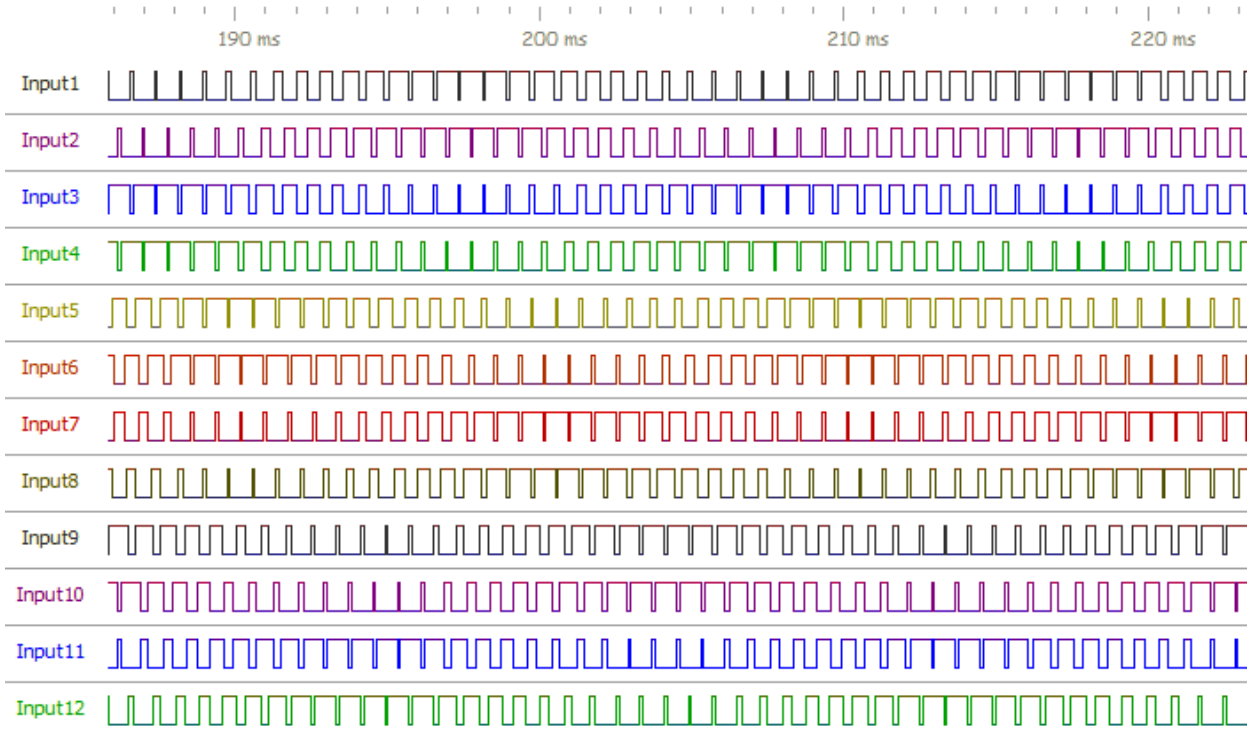


Obr. 5.2: Odpovídající průběh zaznamenaný logickým analyzátorem

Trojfázový tříúrovňový měnič, 504 vzorků na periodu sinu, $f = 50{,}0032$ Hz; $m_h = 0{,}8$; ochranná doma 3000 ns, $f_{nosná} = 1250$ Hz, double edge asymmetrical regularly sampled PWM

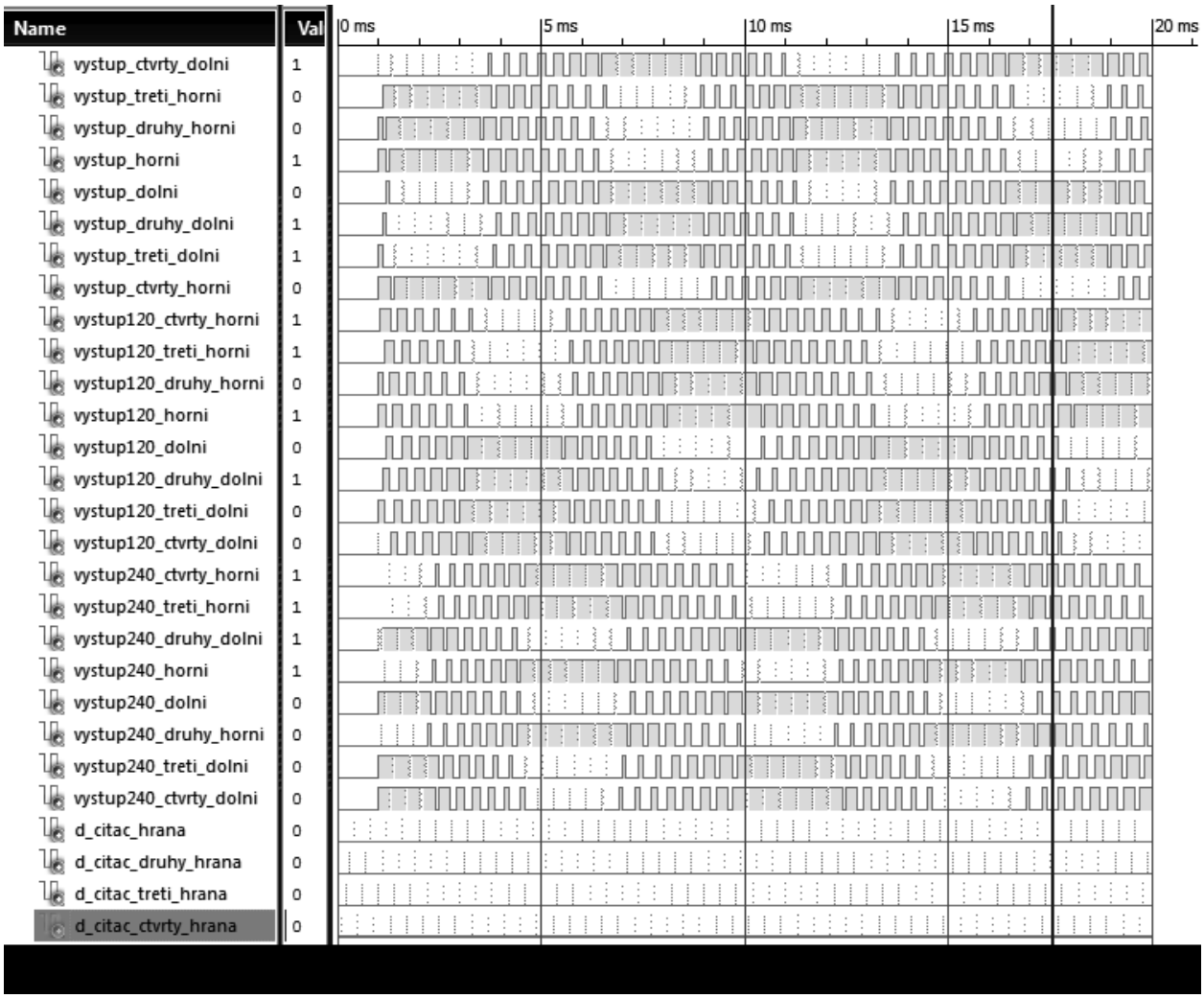


Obr. 5.3: Simulace v ISIM



Obr. 5.4: ss

Trojfázový pětiúrovňový měnič, 504 vzorků na periodu sinu, f = 100,0064 Hz; m_h = 0,8; ochranná doma 3000 ns, $f_{nosná}$ = 1250 Hz, leading-edge regularly sampled PWM.
V dolní části se nacházejí signály „d_citac_hrana“ - ty označují úvratě nosného signálu.Logický analyzátor Sigma má bohužel je 16 vstupů, signály jsem vyvedl na konektor FX2, který není příliš dobře přístupný, tedy se mi bohužel nepodařilo získat průběhy v graficky publikovatelné formě



Obr. 5.5: Simulace v ISIM