VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

IMPLEMENTACE OFDM DEMODULÁTORU V OBVODU FPGA

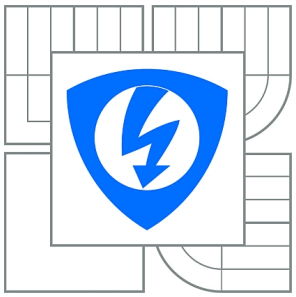
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. PAVEL SOLAR
AUTHOR

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF RADIO ELECTRONICS

IMPLEMENTACE OFDM DEMODULÁTORU V OBVODU FPGA

OFDM DEMODULATOR IMPLEMENTATION IN FPGA

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

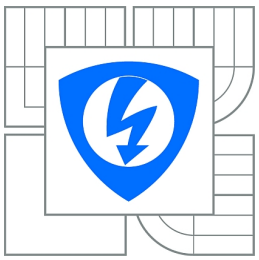
AUTOR PRÁCE — Bc. PAVEL SOLAR

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. ROMAN MARŠÁLEK, Ph.D.

SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Pavel Solar	***ID:***	89873
Ročník:	2	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Implementace OFDM demodulátoru v obvodu FPGA

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s principem modulátoru a demodulátoru OFDM a s principem synchronizace a ekvalizace v OFDM. Seznamte se s prostředím ISE pro práci s FPGA, modulem Coregen a s jazykem VHDL. Vytvořte simulaci OFDM modulátoru a demodulátoru v programu MATLAB. Do simulace zahrňte i blok časové synchronizace a ekvalizace.

Bloky demodulátoru OFDM implementujte do obvodu Virtex II/IV, dle pokynů vedoucího práce. Činnost demodulátoru ověřte simulací v prostředí Xilinx ISE.

DOPORUČENÁ LITERATURA:

[1] PROAKIS, J.G., SALEHI, M. Communication systems engineering, 2/E. Englewood Cliffs: Prentice Hall, 2002.

[2] The designers guide to VHDL <online>. Dostupné na WWW:
http://www.doulos.com/knowhow/vhdl_designers_guide/

Termín zadání:	8.2.2010	***Termín odevzdání:***	21.5.2010

Vedoucí práce: doc. Ing. Roman Maršálek, Ph.D.

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Diplomová práce stručně rozebírá princip OFDM modulace, možnosti synchronizace a odhadu frekvenční charakteristiky kanálu v OFDM. Je vytvořen jednoduchý model OFDM systému v programu MATLAB. Kombinací schématického popisu a popisu v jazyce VHDL je vytvořen ve vývojovém prostředí ISE behaviorální popis OFDM demodulátoru pro implementaci do FPGA.

KLÍČOVÁ SLOVA

OFDM, demodulátor, synchronizace, odhad frekvenční charakteristiky kanálu, ISE, VHDL, FPGA

ABSTRACT

The master's thesis briefly analyses the principle of OFDM modulation, possibilities of the synchronization and channel estimation in OFDM. The simply model of OFDM system is made in MATLAB. Because of the implementation in FPGA is generated the behavioral description of the OFDM demodulator through the combination of the schematics description and the description in the VHDL language. The ISE development environment is used.

KEYWORDS

OFDM, demodulator, synchronization, channel estimation, ISE, VHDL, FPGA

SOLAR, Pavel *Implementace OFDM demodulátoru v obvodu FPGA*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2010. 69 s. Vedoucí práce byl doc. Ing. Roman Maršálek, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Implementace OFDM demodulátoru v obvodu FPGA“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce doc. Ing. Romanu Maršálkovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne .....

.....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

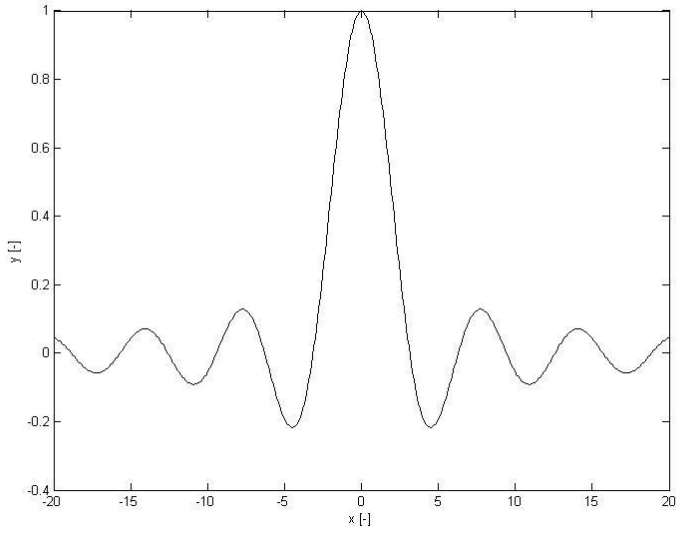
Systémy OFDM (Orthogonal Frequency Division Multiplex) nacházejí širší uplatnění teprve v posledních několika desetiletích, přestože je jejich princip znám již od padesátých let 20. století. Důvodem je nutnost použití DFT (resp. IDFT), které lze realizovat až s použitím moderní elektroniky (FPGA, DSP...) a za pomoci dokonalejších výpočetních algoritmů. OFDM nachází uplatnění především v bezdrátových sítích (W-LAN), nebo pozemním digitálním rozhlasovém vysílání (DAB) a vysílání digitální televize (DVB-T).

Náplní diplomové práce je návrh OFDM demodulátoru a jeho implementace do obvodu FPGA (Field-Programmable Gate Array). V úvodní části je rozebrán princip modulace OFDM, možnosti synchronizace signálu a odhadu frekvenční charakteristiky kanálu. Je zkonstruován jednoduchý model OFDM systému na bázi standardu IEEE 802.11a v programu MATLAB, na jehož základě je vytvořen behaviorální popis demodulátoru ve vývojovém prostředí ISE pro implementaci do FPGA.

1 OFDM

1.1 Systémy MCM a OFDM

Systém OFDM [1] patří mezi digitální modulační formáty s více nosnými vlnami MCM (Multicarrier Modulation). Oproti systémům s jednou nosnou vlnou je u MCM s N nosnými vlnami rychlost datového přenosu pro jednu nosnou přibližně N-krát menší. Ve vysílači se sériový datový tok mění na N paralelních větví, které se modulují na jendnotlivé subnosné na kmitočtech f_0 , f_1 , f_2 , ... f_N . V demodulátoru jsou paralelní signály zpracovány zpět v jeden rychlý sériový datový tok. Oproti starším systémům MCM, kdy jsou od sebe pásma jednotlivých kanálů subnosných oddělen ochrannou mezerou, jsou kmitočty subnosných OFDM voleny tak, že vytváří ortogonální soustavu. To znamená, že každá subnosná je na kmitočtu, kde spektra ostatních subnosných procházejí nulou. Je -li totiž signálem obdélníkový průběh, je jeho spektrem funkce $sincx = sinx/x$:



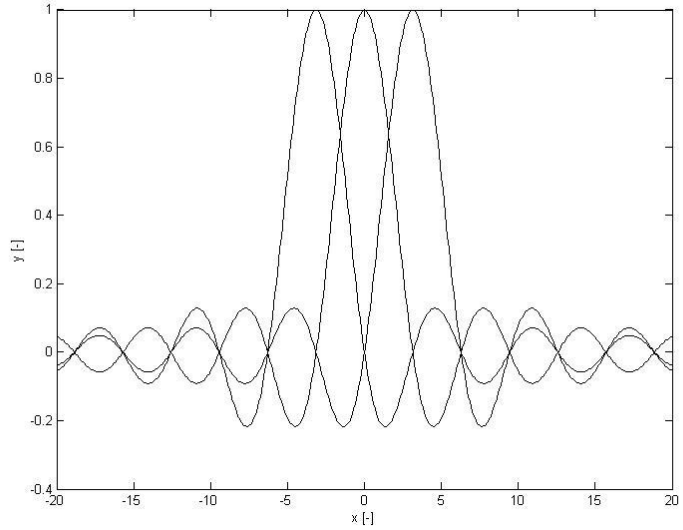
Obr. 1.1: Funkce sinc

Funkce sinc prochází nulou pro všechna $x = n \cdot \pi, \forall n \in Z - \{0\}$. To znamená, že kmitočty subnosných MCM lze volit tak, že pro kmitočet každé subnosné budou všechny ostatní subnosné procházet nulou (Obr. 1.2).

Konkrétně je kmitočet k -té subnosné [1]:

$$f_k = \frac{k}{NT_s} \tag{1.1}$$

kde $NT_s = T_{OFDM}$ je doba trvání užitečné části OFDM symbolu.

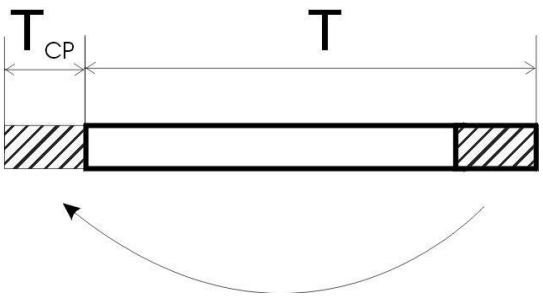


Obr. 1.2: Princip ortogonality

1.2 Vlastnosti OFDM

Systémy OFDM mají velmi dobrou spektrální účinnost (zejména v porovnání s klasickým kmitočtovým multiplexem FDM). Hlavní předností je však velká odolnost proti interferencím vznikajícím mnohacestným šířením. Protože je symbolová rychlost subkanálu mnohonásobně nižší, než symbolová rychlost původního signálu (tj. symbolová perioda Ts je delší), neuplatní se mezisymbolové interference (ISI) vznikající různým zpožděním složek signálu přicházejících k přijímači různými cestami a tedy i s odlišným zpožděním. Další výhodou je také odolnost proti selektivnímu úniku - při ztrátě signálu v úzkém rozsahu kmitočtů lze vhodným kanálovým kódováním zajistit nulovou ztrátu informace. V kanálech s mnohacestným šířením však může docházet kromě interferencí ISI také k interferencím mezi subnosnými vlnami (ICI). Z důvodu zvýšení odolnosti OFDM vůči ICI se na začátek každého symbolu vkládá cyklický prefix (CP), tj. před začátek symbolu se vloží část užitečného signálu z konce tohoto symbolu (Obr. 1.3). CP zvyšuje také odolnost OFDM proti ISI (při sčítání signálu z různým zpožděním se užitečná část symbolu nepřekrývá s užitečnou částí sousedního symbolu - toho lze dosáhnout i pouze pomocí ochranného intervalu, kdy nejsou data vysílána vůbec).

Systém OFDM je však velmi citlivý na kmitočtový offset nosných vln nacházejících se blízko sebe. Stačí jen malá změna kmitočtového odstupu subnosných a systém tak ztrácí ortogonalitu. Další nevýhodou jsou velké nároky kladené na zesilovače zpracovávající signál OFDM. Aby nedocházelo k narušení ortogonality, musí být přesně lineární. Navíc má signál OFDM velkou hodnotu poměru mezi špičkovou

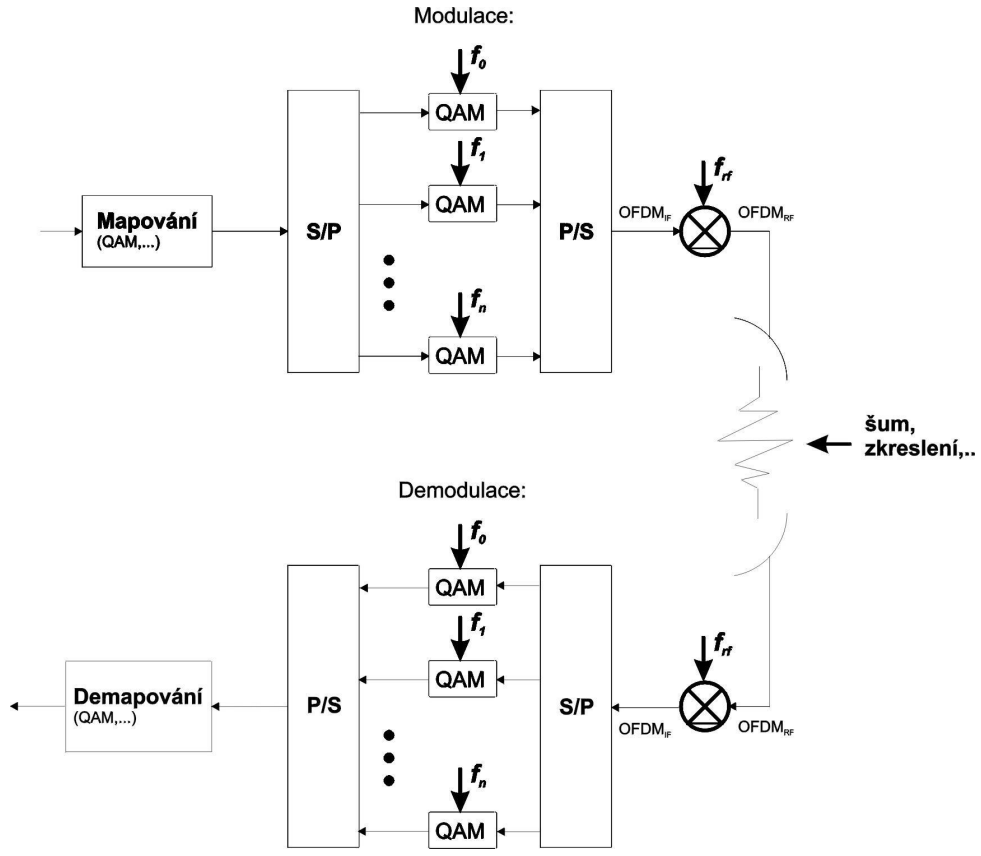


Obr. 1.3: Cyklický prefix

hodnotou výkonu signálu ku jeho střední hodnotě. To znamená, že výkonové zesilovače musí mít velký poměr BO (Back-off). Podstatným problémem OFDM je také obtížná synchronizace.

1.3 Princip OFDM

Asi nejjednodušší bude vysvětlit princip OFDM pomocí schématu (Obr. 1.4) [1],[3]:



Obr. 1.4: Schéma OFDM systému

Vstupní signál přichází nejdříve do kodéru některé z modulací (QAM, QPSK, 16QAM, 64QAM...). Zde se $n = log_2 M$ bitů vstupujících rychlostí f_b sdruží do N kódových skupin, z nichž každá má dobu trvání $T_s = n/f_b$. Každá skupina se kóduje (mapuje) pomocí dané modulace do dvou modulačních složek a_k, b_k, které vytvářejí komplexní datový symbol x_k. Tyto symboly se dále v S/P převodníku rozvádějí do N paralelních větví a v dílčích modulátorech (např. QAM) modulují synfázní a kvadraturní složky subnosných vln. Individuální soustava všech uvažovaných modulovaných subnosných vln pak vytváří OFDM symbol s dobou trvání $T_{OFDM} = NT_s = nN/f_b$. V sumačním členu P/S se všechny takto získané signály sčítají, čímž vzniká OFDM signál v mezifrekvenčním pásmu. Ten je dále transponován do příslušného radiového pásma. Takový signál vchází do přenosového kanálu. Postup zpracování signálu v demodulátoru je přesně opačný (viz Obr. 1.4).

Chceme -li, aby byl OFDM multiplex efektivní, je zapotřebí velké množství nosných. To ale vyžaduje stejně velké množství modulátorů (resp. demodulátorů). Konstrukce takového zařízení by byla velice složitá. Avšak modulaci (demodulaci) velkého počtu ortogonálních subnosných vln lze provést pomocí diskrétní Fourierovy transformace IDFT (DFT) [1]. Při tomto provedení je signál definován softwarově v kmitočtové oblasti jako vzorkovaný digitální signál, jehož diskrétní Fourierovo spektrum existuje pouze při diskrétních kmitočtech. Přitom každá subnosná vlna OFDM odpovídá jednomu elementu tohoto diskrétního spektra. Amplitudy a fáze subnosných jsou určeny vždy příslušnými složkami svých komplexních datových symbolů x_k. Změny těchto symbolů (modulačních stavů) probíhají na všech subnosných synchronně, takže mohou být zpracovány společně, symbol po symbolu. Soustavu modulátorů lze potom nahradit procesem IDFT, který poskytuje sérii vzorků, jež reprezentují uvažovaný signál v časové oblasti.

Podívejme se nyní na systém OFDM z matematického hlediska. Každou subnosnou lze vyjádřit vztahem [1]:

$$\Phi_k(t) = e_k^{j2\pi ft} = \cos(2\pi f_k t) + j\sin(2\pi f_k t) \tag{1.2}$$

kde $f_k = k/NT_s$ je kmitočet k-té subnosné, takže jejich vzájemný kmitočtový odstup je $f_{k+1} - f_k = \Delta f = 1/NT_s$. Jeden symbol OFDM základního pásma sdružuje N modulovaných subnosných a tedy je dán relací [1]:

$$S(t) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{j2\pi f_k t}; \qquad 0 < t < NT_s \tag{1.3}$$

která představuje inverzní diskrétní Fourierovu transformaci (IDFT) konstelačních symbolů x_k. V praxi se používá inverzní rychlá Fourierova transformace (IFFT). V přijímači je postup analogický za pomoci přímé diskrétní Fourierovy transformace (DFT, resp. FFT).

1.4 Synchronizace OFDM

Jak již bylo řečeno výše, jsou systémy OFDM velice citlivé na synchronizaci. Tu je potřeba provádět dvojím způsobem: jednak je to obnova referenční nosné vlny (CR) a jednak časování symbolů (STR).

V přijímači je k zesynchronizování potřeba obnovit nemodulovanou frekvenční nosnou vlnu. Ta však bývá důvodů energetické bilance potlačena. Z toho důvodu jsou v přijímači obvody pro obnovu nosné vlny CR (Carrier Recovery).

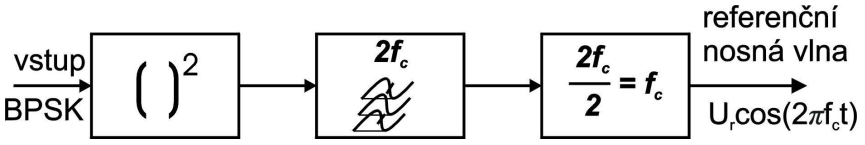
Výstup demodulátoru přijímače musí být vzorkován v přesně stanovených okamžicích [1] $t_m = mT_s + \tau$, kde $T_s = 1/f_s$ je symbolová perioda, τ je nominální časové zpoždění signálu mezi vysílačem a přijímačem a $m = 0; \pm 1; \pm 2; \ldots$. K uvedenému periodickému vzorkování je v přijímači zapotřebí ještě taktovací (hodinový) signál, který se získává v obvodech obnovy časování symbolů STR (Symbol Timing Recovery) označovaných také jako obvody symbolové synchronizace. Ty musí poskytovat informaci nejen o symbolovém kmitočtu f_s, s nímž musí být vzorkován výstup přizpůsobeného filtru, resp. korelátoru demodulátoru, nýbrž musí stanovit i přesné okamžiky vzorkování v rámci symbolových period.

1.4.1 Obnova referenční nosné vlny pomocí pilotního signálu

Existují dva způsoby obnovy referenční nosné vlny pomocí pilotního signálu. Při prvním z nich je spolu s užitečným signálem vysílán pomocný nemodulovaný pilotní signál. Druhý způsob odvozuje referenční nosnou vlnu přímo z modulovaného signálu.

První metoda využívá nepřetržitě vysílaný pilotní signál (těchto signálů může být i více). U modulačních způsobů, jejichž kmitočtové spektrum je ve středu přenášeného pásma nulové, se vystačí s jediným pilotním signálem, který je možné situovat právě do tohoto středu [1] (metoda TCT, resp. TIB). Takové spektrum má například modulace BPSK, u níž je signál vysílače kódován ve formátu Manchester. Není -li střed pásma modulovaného signálu volný, lze použít dva pilotní signály umístěné symetricky kolem středu spektra (DTCT), nebo blízko sebe nad pásmem modulovaného signálu (DTSSB). Požadovaný referenční kmitočet pak lze získat jednoduchou aritmetickou operací s pilotními signály (např. u DTCT je kmitočet nosné rovný aritmetickému průměru pilotních signálů). Nevýhodou této varianty je její energetická náročnost - pilotní signály spotřebují jistou část energie systému.

Druhou variantou je obnova referenční nosné vlny s asistencí pilotního signálu (PSAM). Při této metodě se formou časového multiplexu střídají symboly nesoucí užitečnou informaci s tzv. pilotními symboly. Výhodou této metody je, že na rozdíl



Obr. 1.5: Obnova referenční nosné vlny pomocí kvadrátoru

od předchozí varianty nedochází ke změnám spektra přenášeného signálu, ani k fluktuacím modulační obálky (může způsobovat problémy ve výkonových stupních vysílače). Nevýhodou je naopak snížení maximální přenosové rychlosti. Kromě toho v podmínkách velmi rychlých úniků nemusí krátký vzorek pilotního signálu přesně reprezentovat referenční nosnou vlnu po celou dobu následujících datových symbolů, takže obnovená nosná vlna už není po celou dobu elementárního časového rámce již dokonale koherentní.

1.4.2 Metody obnovy referenční nosné vlny bez pilotního signálu

U signálů s oběma postranními pásmy a potlačenou nosnou vlnou (modulace MPSK, MQAM...) stačí k obnovení nosné vlny pouze vhodný algoritmus zpracovávající přímo užitečný signál. Zvýší se tak energetická účinnost systému (odpadají výkonové ztráty nutné k vyslání pilotního signálu).

První z možností obnovy referenční nosné vlny bez pilotního signálu je použití Umocňujících smyček (Squaring loops) [1]. Jejich základem je nelineární dvojbran, na jehož vstup se přivádí vzorek vstupního signálu MPSK a na výstupu se odebírá jeho M-tá mocnina. Modulovaný signál MPSK je určen obecně vztahem

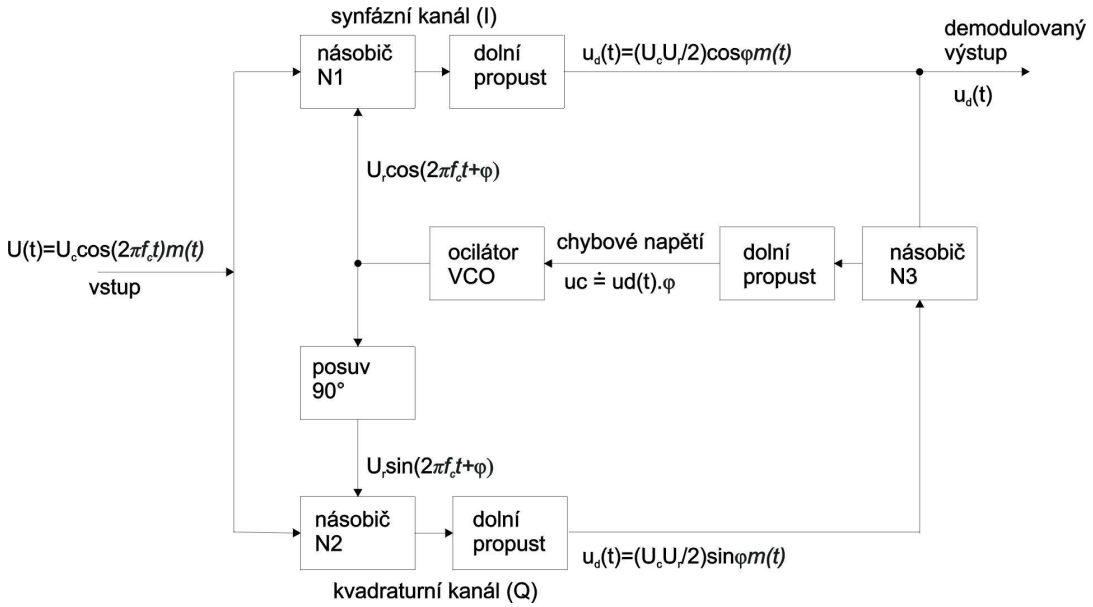
$$s_{MPSK}(t) = \sqrt{\frac{2E_s}{T_s}} \cos\left[2\pi f_c t + (i-1)\frac{2\pi}{M}\right], \quad proO \leq t \leq T_s,\ i = 1, 2, ..., M \quad (1.4)$$

kde $E_s = (log_2 M)E_b$ je energie na jeden symbol a $T_s = (log_2 M)T_b$ je symbolová perioda. Má -li nosná vlna kmitočet f_c, je v "umocněném" signálu obsažena mj. i složka s kmitočtem Mf_c. Tu lze vyčlenit z výstupního spektra dvojbranu následujícím pásmovým filtrem, za nímž je vhodné zařadit sledovací filtr s fázovým závěsem PLL. Vlivem umocnění je z této složky odstraněno fázové klíčování (neboť $\pi(m-1) = 0 \, mod \, 2\pi$), takže průchodem následujícím děličem kmitočtu s dělícím poměrem M se získá nemodulovaná nosná vlna s kmitočtem f_c. Příklad této koncepce pro modulaci BPSK je schéma na Obr.1.5.

Obnovená nosná vlna však vykazuje fázovou nejistotu $2\pi/M$ a nemůže být proto přímo využita jako reference, neboť případné odchylce od správné fáze produkuje

demodulátor chybný signál. Daný problém lze odstranit přenosem malého vzorku nosné vlny, který je využíván ke korekci případných fázových chyb regenerované nosné. Jinou možností synchronizace je využití redundance, záměrně vkládané do přenášeného signálu za účelem rámcové synchronizace, nebo protichybového kódování. Dalším řešením je aplikace Millerova kódování (zpožděné modulace), které je imunní vůči některým fázovým nejistotám. Potřebu obnovy nosné vlny lze zcela odstranit použitím diferenciálního kódování datového signálu před modulátorem vysílače a diferenciálního dekódování v přijímači.

Další možností obnovy referenční nosné vlny je použití Costasovy smyčky (Costas Loop) [1], která může být použita k regeneraci nosné vlny nebo přímo k vlastní demodulaci signálu BPSK (Obr. 1.2).

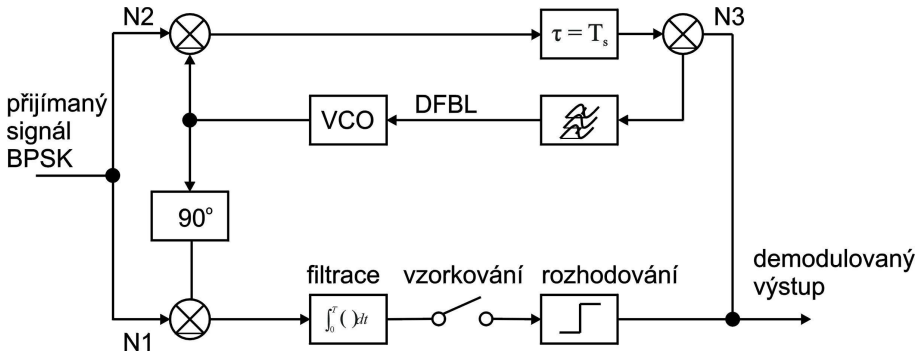


Obr. 1.6: Costasova smyčka

Toto zapojení lze použít pro libovolnou modulaci MPSK nebo MQAM. Avšak i zde vzniká fázová nejistota $2\pi/M$ pro smyčku M-tého řádu. To lze řešit již výše uvedeným způsobem.

Jako poslední uvedeme možnost obnovení referenční nosné pomocí smyčky s rozhodovací zpětnou vazbou DFBL (Decision Feedback Loop) [1]. Příklad pro BPSK je na Obr.1.7.

Přijímaný signál je v násobičích N1 a N2 násoben kvadraturními nosnými vlnami odvozenými z výstupu oscilátoru VCO. Výstupem násobiče N1 je po filtraci, realizované korelátorem přizpůsobeným k datovému symbolu s periodou T_s, vzorkování a rozhodování již užitečný výstupní signál. Ten je také zaváděn do násobiče N3, kde se násobí výstupem násobiče N2 zpožděným o symbolovou (bitovou) periodu



Obr. 1.7: Smyčka s rozhodovací zpětnou vazbou DFBL

T_s. Výstupem tohoto násobiče se po filtraci dolní propustí dolaďuje oscilátor VCO, který při správné funkci celého systému generuje koherentní nosnou vlnu. Uvedené zapojení lze opět zobecnit pro M-stavové modulace. Avšak i u této smyčky vzniká fázová nejistota $2\pi/M$, takže i zde je nutné aplikovat jednu z výše uvedených metod řešení tohoto problému. U tohoto řešení jsou však vlastnosti smyčky z hlediska přesnosti "závěsu" regenerované nosné na nominální nosnou znatelně lepší, než u prvých dvou variant.

1.4.3 Obnova časování symbolů

Prvním nejjednodušším způsobem obnovy časování symbolů je stejně jako u obnovy nosné použití multiplexního vysílání časovacího (hodinového) signálu. Tato metoda je však, jak již bylo uvedeno výše, energeticky neefektivní.

Další možností časové synchronizace je využití známé struktury OFDM signálu, tj. toho, že každý symbol obsahuje CP [6]. Lze použít prostou korelaci vzorků signálu vzdálených K (počet vzorků trvání OFDM symbolu bez CP). Sofistikovanější způsob je však nalezení maxima funkce

$$\tau_{est} = argmax_m \left\{ \left| \sum_{l=m}^{m+L_{CP}-1} r[l]r^*[l+K] \right| - \frac{SNR}{SNR+1} \cdot \frac{1}{2} \sum_{l=m}^{m+L_{CP}-1} \left(|r[l]|^2 + |r[l+K]|^2 \right) \right\} \tag{1.5}$$

kde τ je časový offset, r je m-tý vzorek přijatého signálu a L_{cp} je délka CP. Metoda je však neúčinná v případě mnohacestného šíření signálu.

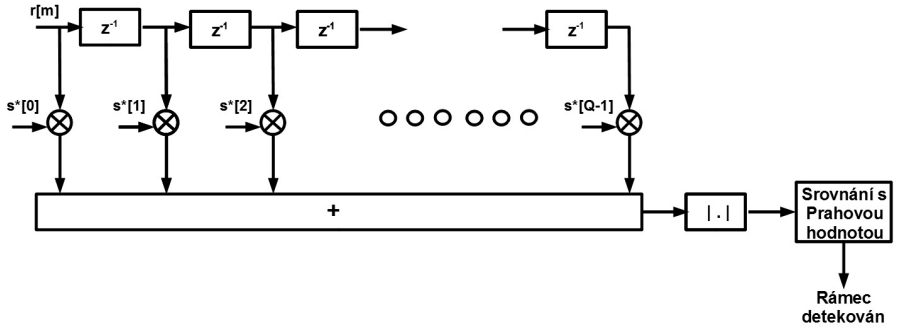
1.4.4 Časová synchronizace s využitím preamble

Jiný způsob časové synchronizace je vyslání známé synchronizační sekvence a to zpravidla na začátku každého vyslaného rámce [6]. Synchronizační preambli využívá např. standard IEEE 802.11a (wifi), který definuje preambli složenou ze dvou

částí – v první části jsou vyslány 2 OFDM symboly sloužící k časové synchronizaci rámce, druhá část preamble pak obsahuje jeden dvojnásobně dlouhý symbol sloužící k odhadu kanálu (viz níže). Je -li pak v přijímači synchronizační sekvence známa, je nejjednodušší možností vedoucí k nalezení počátku rámce výpočet vzájemné korelace (cross correlation) přijatého signálu se synchronizační sekvencí. Korelační funkce nabývá maxima právě pro počátek rámce:

$$\tau_{est} = argmax_m \left| \sum_{Q-1}^{l=0} s^*[l] r[m+l] \right| \tag{1.6}$$

kde s je synchronizační sekvence o délce Q. Rovnici lze interpretovat zapojením na Obr.1.8:



Obr. 1.8: Časová synchronizace pomocí korelace preamble

Modifikací jednoduché korelace vzniká Schmidl-Coxův algoritmus [6]. Prostá korelace je doplněna váhováním podle výkonu signálu. Synchronizační sekvenci tvoří dva shodné OFDM symboly délky N:

$$P(m) = \sum_{l=0}^{N-1} r^*[m+l] r[m+l+N] \quad R(m) = \sum_{l-0}^{N-1} |r[m+l+N]|^2 \tag{1.7}$$

Časový offset lze pak získat:

$$\tau_{est} = argmax_m \left\{ \frac{|P(m)|^2}{R^2(m)} \right\} \tag{1.8}$$

Výhodou je, že výstupem algoritmu je také odhad časového offsetu:

$$\epsilon_{est} = \frac{1}{2\pi} arg \left\{ P(\tau_{est}) \right\} \tag{1.9}$$

1.5 Odhad frekvenční charakteristiky kanálu

Při průchodu signálu kanálem není signál ovlivněn pouze aditivním šumem w (W), ale i impulsní (frekvenční) charakteristikou kanálu h (H):

$$r[m] = \sum_{l=0}^{L-1} h[m,l]s[m-l] + w[m] \tag{1.10}$$

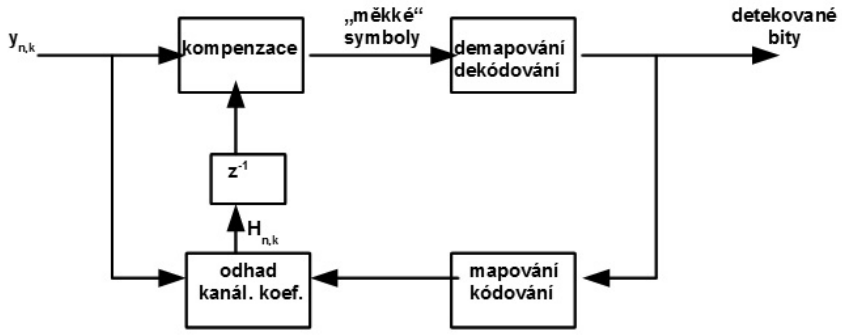
$$R[m,k] = S[m,k] \cdot H[m,k] + W[m,k] \tag{1.11}$$

kde $r(R)$ je přijatý signál v časové (frekvenční) oblasti, $s(S)$ je vyslaný signál v časové (frekvenční) oblasti, $w(W)$ je aditivní šum v časové (frekvenční) oblasti a $h(H)$ je impulsní (frekvenční) charakteristika kanálu v čase m a na frekvenci k.

Zhoršení kvality signálu způsobené přičtením šumu je již nevratné, ale deformaci signálu způsobenou mnohacestným šířením lze do značné míry kompenzovat, zjistíme -li frekvenční (resp. impulsní) charakteristiku kanálu.

1.5.1 Odhad frekvenční charakteristiky kanálu bez vysílání redundantních dat

Dekódované bity na výstupu demodulátoru jsou zpátky modulovány [6]. Takto vzniklý signál je srovnáván se signálem vstupním, čímž můžeme zjistit koeficienty frekvenční charakteristiky kanálu $H_{n,k}$ (pro n-tý symbol a k-tou subnosnou). Ty pak mohu použít pro kompenzaci vlivu kanálu na následující vzorek (Obr.1.9):



Obr. 1.9: Odhad frekvenční charakteristiky kanálu bez treninkové sekvence

1.5.2 Odhad frekvenční charakteristiky kanálu pomocí pilotních nosných

Na určitých subnosných jsou v daných časových intervalech (nebo i kontinuálně) vysílána známá data. Pak lze koeficienty frekvenční charakteristiky kanálu pro n-tý symbol a k-tou subnosnou určit prostým podělením přijatých y a vyslaných dat x (před IFFT resp. po FFT):

$$H_{n,k,est} = \frac{y_{n,k}}{x_{n,k}} \tag{1.12}$$

Schéma pro umístění pilotů v čase a v prostoru může mít nejrůznější podobu v závislosti na předpokládaných vlastnostech kanálu.

1.5.3 Odhad frekvenční charakteristiky kanálu s využitím preamble

Předpokládáme -li, že je frekvenční charakteristika kanálu konstantní nebo se mění dostatečně pomalu, postačí určit koeficienty frekvenční charakteristiky vždy jednou na začátku rámce vysláním známé preamble. Jedná se v podstatě o použití pilotních nosných na všech frekvencích. Například standard IEEE 802.11a používá dva shodné tréninkové OFDM symboly obsahující pouze $+1$ a -1. Při implementaci rovnice (1.13) lze tedy získat koeficienty frekvenční charakteristiky kanálu pouhým bitovým posuvem přijatých vzorků preamble. Výsledná frekvenční charakteristika se získá zprůměrováním obou výsledků získaných z prvního a druhého tréninkového symbolu v preambli:

$$H_{k,est} = \frac{1}{2}\left(H_{1,k} + H_{2,k}\right) \tag{1.13}$$

Podobného výsledku lze dosáhnout odhadem impulsní charakteristiky kanálu v časové oblasti. Přepíšeme -li rovnici (1.10) do maticové podoby [6]:

$$r = Sh + w \tag{1.14}$$

kde S je tzv. Toeplitzova matice (vektory vyslaných dat jsou "poskládány" na diagonále nulové matice), vytvořená z tréninkových dat, lze impulsní charakteristiku kanálu odhadnout jako

$$h_{est} = \left(S^H S\right)^{-1} S^H r \tag{1.15}$$

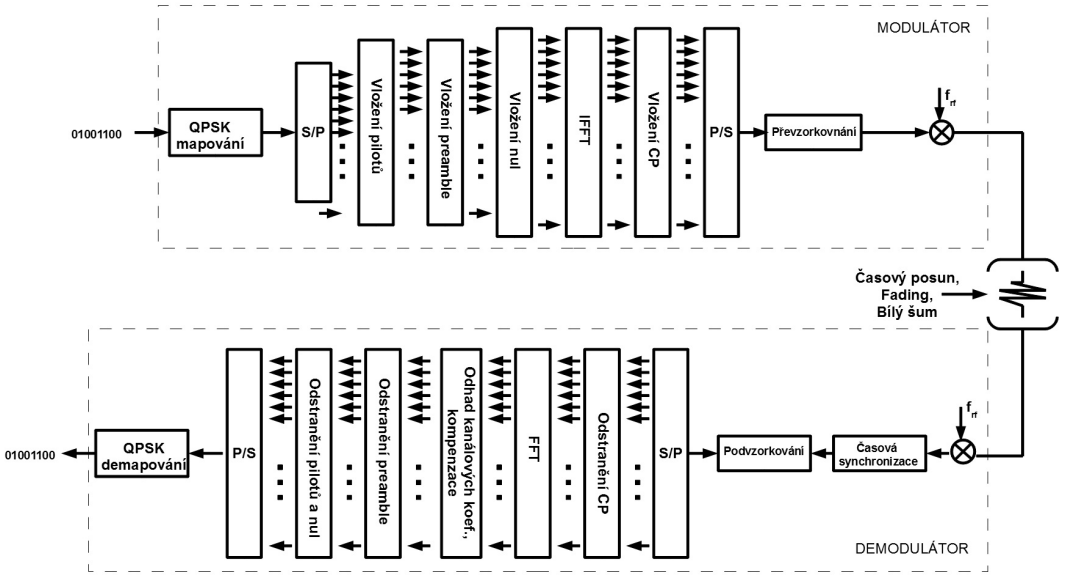
kde S^H je tzv. Hermitovská matice (hermitian matrix) Toeplitzovy matice.

2 SIMULACE OFDM V PROG. MATLAB

2.1 Popis modelu

Před samotnou implementací demodulátoru do obvodu FPGA je vhodné navrhnout jeho model v některém z programů, kde může být vyzkoušena principielní funkčnost bez závislosti na cílovém hardwaru. Jako vhodné prostředí pro takovou simulaci se jeví program MATLAB, v němž byl model OFDM modulátoru vytvořen.

Pro ověření funkčnosti demodulátoru je nutné i sestrojení modelu modulátoru, který zajistí vytvoření dat pro demodulaci. Systém vychází ze standardu IEEE 802.11a, přičemž jsou modelovány pouze jeho základní vlastnosti (počet nosných, délka symbolu, CP, preamble, ...). Schéma modelu je zobrazeno na Obr.2.1. Zdrojový kód (m-file) je přiložen v příloze A.

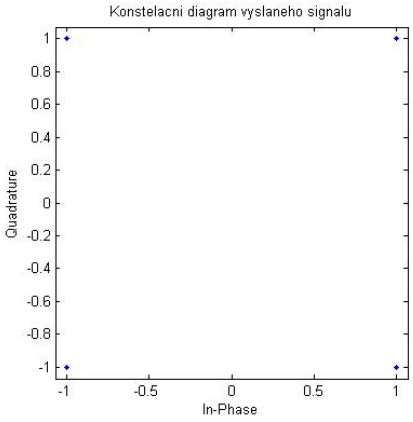


Obr. 2.1: Schéma modelu OFDM systému v MATLABu

2.2 OFDM modulátor

Vstupní signál *sig* je definován jako náhodná binární posloupnost délky n. Ten je následně namapován do symbolů odpovídající zvolené modulaci (použita QPSK) (Obr.2.2).

Namapovaný signál *maps* je dále v S/P převodníku rozdělen do $N_{sd} = 48$ paralelních větví, jejichž počet odpovídá počtu nosných nesoucí užitečnou informaci. V následujícím bloku jsou do signálu vloženy piloty podle schématu standardu IEEE



Obr. 2.2: Konstelační diagram QPSK modulovaného signálu

802.11a, resp. HIPERLAN [9], [10], [11]. Budeme -li nosné číslovat od $l = -26$ do $l = 26$, jsou piloty vloženy na subnosné číslo $-21, -7, 7$ a 21. Ostatní nosné (-26 až -22, -20 až $-8, -6$ až $-1, 1$ až $6, 8$ až $20, 22$ až 26) jsou využity pro užitečná data. Na pilotní nosné je modulována pseudonáhodná sekvence generovaná polynomem

$$S(x) = X^7 + X^4 + 1 \tag{2.1}$$

je -li výchozím stavem 1111111 a všechny "1" jsou následně přepsány na "-1" a všechny "0" jsou přepsány na "1". Každý prvek sekvence je použit pro jeden OFDM symbol. Pseudonáhodná sekvence se opakuje s periodou $T = 127$.

V dalším bloku je na začátek signálu vložena preamble, jejíž struktura opět kopíruje standard IEEE 802.11a. Je složena ze dvou stejně dlouhých částí délky 2 OFDM symbolů (jeden OFDM symbol tvoří 80 vzorků a trvá $4\mu s$). První část preamble se skládá z 10 opakování krátké tréninkové sekvence (STS – short training sequence). Tato část preamble slouží k detekci signálu, zjištění výkonové úrovně, časové synchronizaci a hrubé frekvenční synchronizaci. Krátká tréninková sekvence vysílá data na 12 subnosných:

$$\begin{aligned} S_{-26,26} = \sqrt{52/(2 \cdot 12)} \times \{&0, 0, 1 + \mathrm{j}, 0, 0, 0, -1 - \mathrm{j}, 0, 0, 0, 1 + \mathrm{j}, 0, 0, 0, \\ &-1 - \mathrm{j}, 0, 0, 0, -1 - \mathrm{j}, 0, 0, 0, 1 + \mathrm{j}, 0, 0, 0, 0, 0, 0, 0, -1 - \mathrm{j}, 0, 0, 0, \\ &-1 - \mathrm{j}, 0, 0, 0, 1 + \mathrm{j}, 0, 0, 0, 1 + \mathrm{j}, 0, 0, 0, 1 + \mathrm{j}, 0, 0, 0, 1 + \mathrm{j}, 0, 0\} \end{aligned} \tag{2.2}$$

Periodicity s $T_{IFFT}/4 = 0,8\mu s$ je dosaženo aplikací 64-bodové IFFT na $S_{-26,26}$. Druhá část preamble se skládá ze dvou opakování dlouhé tréninkové sekvence (LTS – long traning sequence), před kterým je vložen jeden dvojnásobně dlouhý CP. LTS je

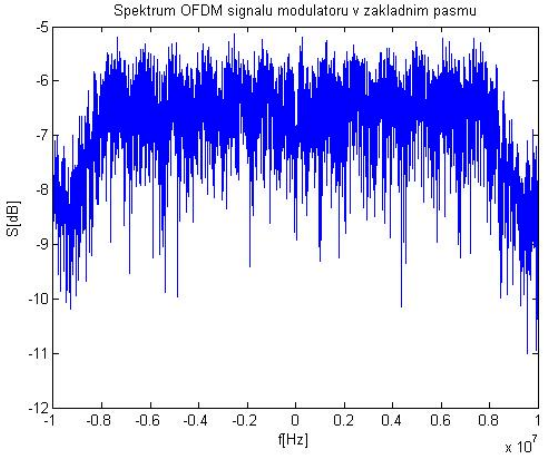
ve frekvenční oblasti složena z 53 subnosných. Vytvoření LTS lze dosáhnout přímou aplikací IFFT na sekvenci

$$\begin{aligned} L_{-26,26} = \{&1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, \\ &-1, 1, -1, 1, 1, 1, 1, 0, 1, -1, -1, 1, 1, -1, 1, -1, 1, -1, \\ &-1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1\} \end{aligned} \tag{2.3}$$

Tuto druhou část preamble lze v demodulátoru využít pro odhad kanálových koeficientů a přesnou frekvenční synchronizaci.

Následuje blok 64-prvkové IFFT, před nímž jsou do signálu vloženy nuly (doprostřed a na kraj spektra - vzniká signál *with_guards*).

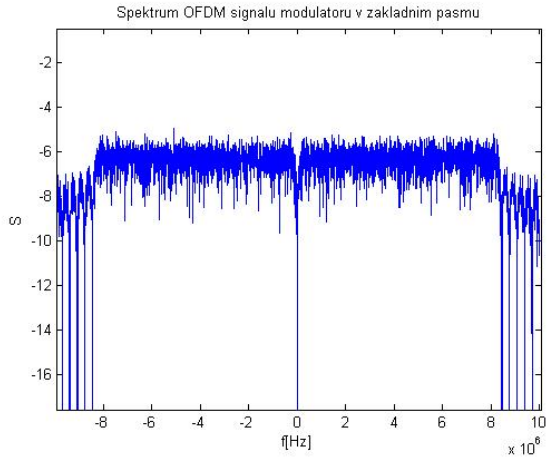
Poté je do signálu vložen CP - každý OFDM symbol trvající 64 vzorků je rozšířen o 16-ti vzorkový CP. Jelikož požadujeme, aby druhá část preamble složená ze dvou OFDM symbolů měla pouze jeden dvojnásobně dlouhý CP, musí být ještě tato část signálu dodatečně upravena. Takto vzniklý signál již splňuje naše požadavky na OFDM - jeho spektrum je zobrazeno na Obr.2.3.



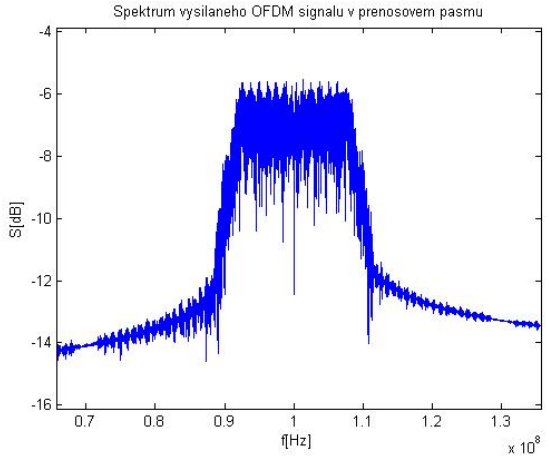
Obr. 2.3: Spektrum OFDM signálu modulátoru v základním pásmu

Oproti předpokladu nemá spektrum hladký průběh se zřetelným poklesem výkonu na nulovém kmitočtu (SS složka). Experimentálně bylo ověřeno, že toto nežádoucí zkreslení způsobuje vložení CP. Spektrum signálu v základním pásmu bez vloženého CP je vidět na Obr.2.4. V praxi je pravděpodobně signál po vložení CP ještě vyfiltrován.

Před vysláním do přenosového kanálu je signál ještě namodulován na nosnou $f_c = 100MHz$. Výsledné spektrum OFDM signálu vycházejícího z modulátoru je pak zobrazeno na Obr.2.5.



Obr. 2.4: Spektrum OFDM signálu v základním pásmu bez vložení CP



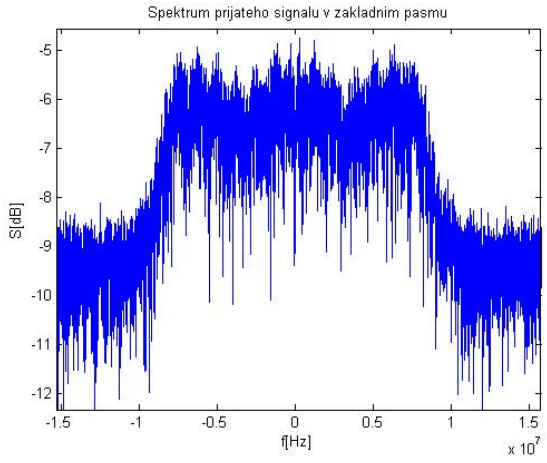
Obr. 2.5: Spektrum OFDM signálu modulátoru v přenosovém pásmu

2.3 Průchod kanálem

Signál vysílaný modulátorem přichází na vstup demodulátoru zkreslený. Deformace signálu reálným přenosovým prostředím je reprezentována časovým zpožděním, modelem ricianovského kanálu a přidáním bílého šumu. Kvůli jednoduchosti je časový posun nahrazen cyklickým posunutím - počet vzorků signálu zůstává stejný. Velikost zpoždění signálu je nadefinována na začátku zdrojového kódu (v tomto případě 4321 vzorků, tedy $4321 \cdot 2,5ns = 10,8\mu s$). Zpoždění jednotlivých cest v modelu ricianovského kanálu jsou voleny tak, aby jejich rozdíl nebyl větší než délka CP (konkrétně je nadefinováno 6 cest se zpožděním 0, 50, 100, 150, 200 a 500 ns s přenosem -3, 0, -0,5, -2, -4 a -6 dB). k-faktor riceanovského kanálu je zvolen $k = 1$. Nakonec je v kanálu k signálu přičten qaussovský bílý šum ($SNR = 15dB$).

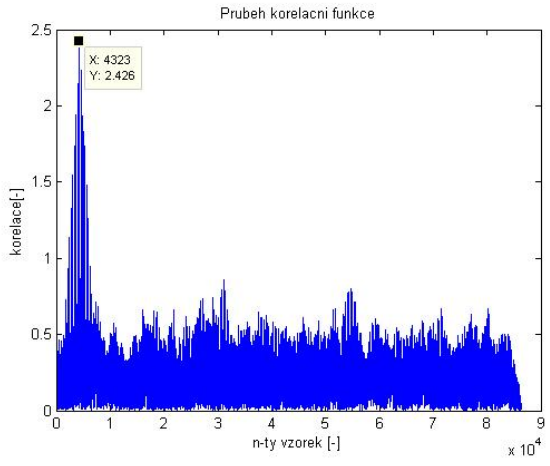
2.4 OFDM demodulátor

Na vstupu demodulátoru je signál nejdříve přesunut z přenosového do základního pásma. Spektrum přijatého signálu v základním pásmu je na Obr.2.6. Je patrné, že signál je poznamenán průchodem kanálem (srovnej s Obr.2.3).



Obr. 2.6: Spektrum přijatého signálu v základním pásmu

Následuje blok časové synchronizace - je provedena vzájemná korelace přijatého signálu se synchronizační sekvencí reprezentovanou první částí preamble. Podle principu vysvětleného výše (rov. 1.6), (Obr.1.8) nabývá korelační funkce maxima právě pro počátek rámce. Průběh korelační funkce je patrný na Obr.2.7.



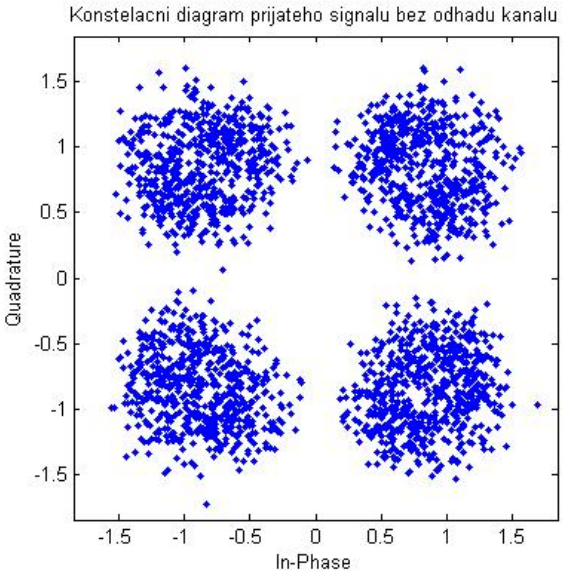
Obr. 2.7: Průběh korelační funkce sloužící k časové synchronizaci

Jelikož jsou na vstup modulátoru přiváděna náhodně generovaná data, nepatrně se liší výsledky jednotlivých simulací. Odchylka odhadnutého počátku rámce od jeho

skutečné pozice se nikdy neliší více než o 10 vzorků, což odpovídá polovině doby vzorkovací periody v základním pásmu. Časová synchronizace je tedy plně funkční.

Po podvzorkování signálu (v modulátoru byla kvůli přesunu signálu do přenosového pásma zvýšena vzorkovací frekvence z 20 na 400 MHz) jsou všem OFDM symbolům odstraněny CP a sériový tok dat je S/P převodníkem změněn na paralelní. Následuje 64-bodová FFT.

Poté jsou pomocí metody popsané rovnicí (1.13) odhadnuty koeficienty frekvenční charakteristiky kanálu, jejíž vliv na přijatý signál je následně kompenzován. Účinnost kompenzace po odhadu frekvenční charakteristiky je patrná z Obr.2.8 (konstelační diagram přijatého signálu před kompenzací) a Obr.2.9 (konstelační diagram přijatého signálu po kompenzaci odhadnutými koeficienty frekvenční charakteristiky kanálu).

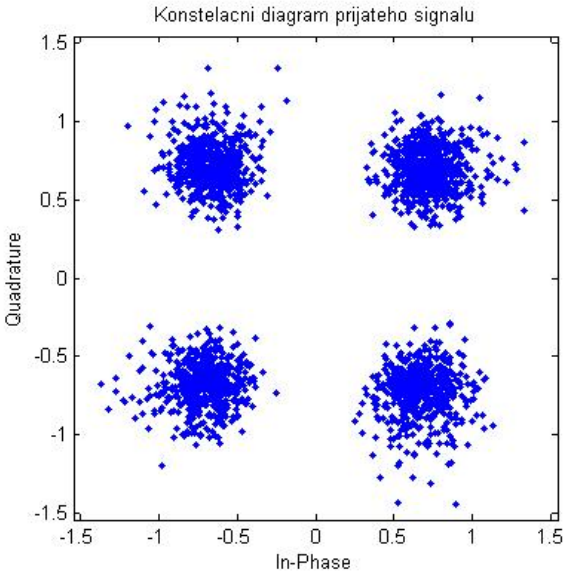


Obr. 2.8: Konstelační diagram přijatého signálu bez kompenzace

Obnovený signál již zcela neodpovídá vyslanému signálu - jednak není odstraněn aditivní šum, jednak vnáší šum chybu i do odhadu frekvenční charakteristiky kanálu.

Po kompenzaci frekvenční charakteristiky kanálu je signál po odstranění preamble, pilotních a nulových subnosných převeden v P/S převodníkem do sériové formy.

Posledním blokem demodulátoru je QPSK dekodér, na jehož výstupu jsou již data v bitové formě. Jak je již patrné i z konstelačního diagramu přijatého signálu na Obr.2.9, je výsledná chybovost BER pro stávající nastavení kanálu nulová.



Obr. 2.9: Konstelační diagram přijatého signálu po kompenzaci odhadnutými koeficienty frekvenční charakteristiky kanálu

3 FPGA

Obvody FPGA (Field-Programmable Gate Array) [5] jsou v zásadě tvořeny polem konfigurovatelných logických bloků (CLB). Ty se dále dělí na menší části nazývané podle výrobce (řezy, logické buňky,...). Logické buňky obsahují struktury pro vytvoření kombinačních funkcí a klopné obvody. K propojení bloků CLB slouží programovatelná propojovací struktura (PI). Pole bloků CLB je obklopeno vstupně-výstupními bloky (IOB). Obvykle také obsahují klopné obvody, nikoli však kombinační logiku. Jsou připojeny k vývodům obvodu FPGA a jejich úkolem je propojení vnějších signálů se signály v poli bloků CLB. Kromě logických buněk, které jsou základními univerzálními strukturními prvky, obsahují bloky CLB ještě další specializované prvky, které slouží ke snadnějšímu a efektivnímu vytvoření často používaných zapojení. Jsou to především multiplexory používané ve funkci spínačů při propojování logických buněk do celků pro vytvoření složitějších kombinačních funkcí. Dále mají obvody FPGA vytvořeny struktury pro použití většího množství hodinových signálů, speciální struktury pro generování rychlého přenosového signálu u sčítaček a další přídavné prvky. Rovněž je zde variabilita pouzder a možnost volby několika velikostních variant obvodů určité řady v témže pouzdru. V poli bloků CLB jsou také rozmístěny další bloky. Typicky bývají na čipu bloky pro zpracování a úpravu hodinových signálů, hardwarové násobičky, blokové paměti RAM/ROM, struktury pro rychlou sériovou komunikaci (multigigabitové přijímače/vysílače) a řada dalších bloků, u moderních obvodů FPGA i celé procesory – např. procesorová jádra PowerPC u novějších variant obvodů Virtex firmy Xilinx.

Obvody FPGA jsou právě jedním z elektronických prvků, které umožnily masové rozšíření systémů využívajících OFDM díky možnosti implementace DFT do součástky velmi malých rozměrů, přičemž nedochází k podstatnému zpoždění signálu. Obvody FPGA jsou však již tak složitá zařízení, že pro jejich naprogramování je nutné používat složité sofistikované systémy. Jedním z takových nástrojů je např. prostředí ISE fy. Xilinx.

3.1 Jazyk VHDL

Jazyk VHDL (VHSIC hardware description language) patří do rodiny programovacích jazyků HDL (hardware description language). Je akceptován jako ověřený standard IEEE. Používá se pro návrh a simulaci digitálních integrovaných obvodů. VHDL má prostředky pro popis paralelismu, konektivity a explicitní vyjádření času. Používá se jak pro simulaci obvodů, tak i pro popis integrovaných obvodů, které se mají vyrábět.

3.1.1 Základní vlastnosti jazyka VHDL

Nejdůležitější vlastnosti jazyka VHDL jsou [7]:

- Je to otevřený standard (open standard). K jeho použití pro sestavení návrhových systémů není třeba licence jeho vlastníka, jako je tomu u jiných jazyků HDL (například u jazyka ABEL). To je jeden z důvodů, proč je tento jazyk v návrhových systémech často používán.

- Umožňuje pracovat na návrhu, aniž je předtím zvolen cílový obvod. Ten může být zvolen až v době, kdy jsou známy definitivní požadavky na prostředí, v němž má navrhovaná konstrukce pracovat, a je možno cílový obvod měnit podle potřeby při zachování popisu v jazyku HDL, přičemž může být zvolen obvod PLD nebo FPGA. (Device-independent design.)

- Je možno provést simulaci navrženého obvodu na základě téhož zdrojového textu, který pak bude použit pro syntézu a implementaci v cílovém obvodu. Zdrojový text je možno zpracovávat v různých simulátorech a v syntetizérech různých výrobců. Odsimulovaný text může být použit v dalších projektech s různými cílovými obvody, což je umožněno tím, že jazyk VHDL podporuje hierarchickou strukturu projektů. Této vlastnosti jazyka se říká přenositelnost (portability) kódu.

- V případě úspěšného zavedení výrobku na trh lze popis konstrukce v jazyku VHDL použít jako podklad pro její implementaci do obvodů ASIC vhodných pro velké série.

3.1.2 Struktura modelu v jazyku VHDL

Model vytvořený v jazyku VHDL má dvě základní části [7]: deklaraci entity (entity declaration) a popis (tělo) architektury (architecture description, body). Deklarace entity popisuje vstupy a výstupy konstrukce, popis architektury definuje její funkci.

Mezery, tabulátory a znaky nového řádku slouží jako oddělovače, nemají další význam a lze je používat k rozčlenění textu tak, aby byl přehledný. Vyhrazená (klíčová) slova (reserved words) jsou psána velkými písmeny. Jsou to slova, jejichž význam je stanoven v definici jazyka VHDL. Význam některých dalších vícepísmenových slov je definován dalšími standardy, které rozvíjejí základní definici jazyka VHDL užitými knihovnami. Identifikátory (uživatelské symboly) slouží k označení signálů, proměnných a podobných objektů. Komentáře (comments) v textu VHDL začínají dvěma pomlčkami ($--$) a končí vždy na konci řádku.

Jazyk VHDL je velice složitý a sofistikovaný nástroj. Popsat dokonale všechny jeho možnosti a způsob vytváření modelu v jazyku VHDL není v možnostech této práce. Téma by zajisté vystačilo na samostatnou knihu. Dobrým zdrojem pro osvojení si základních principů při programování ve VHDL jsou např. skripta [7].

3.2 Systém ISE

ISE (Integrated Software Environment) je vývojové prostředí pro FPGA obvody firmy Xilinx. Využívá mimo jiné i jazyk VHDL. Uživatelské rozhraní systému obsahuje několik oken [5]. V jednom z nich nazvaném "Sources in Project" se zobrazují názvy zdrojových položek (a odpovídajících souborů), které jsou zařazeny do projektu. Uživatel klepnutím vybere položku, která bude zpracovávána. K vybrané položce se v dalším okně "Processes for Source" zobrazí procesy, kterými je možno položku zpracovávat. Typickými procesy jsou syntéza, implementace, simulace a vytvoření programovacího souboru.

Syntéza je v podstatě vytvoření "netlistu", tj. zapojení obvodových prvků (logické členy, klopné obvody, registry atd.), tedy vlastně vytvoření schématu zapojení s obvodovými prvky, které jsou obsaženy v předpokládaných cílových obvodech a které vykonává požadovanou funkci. U složitých obvodů FPGA jsou kromě výše uvedených obvodových prvků do netlistu zařazeny prvky specifické pro cílové obvody (obvykle jsou společné pro určitou řadu obvodů, například Spartan-II, Virtex-II a podobně). Pokud tyto specifické prvky nejsou použity, je netlist přenositelný na jiné cílové obvody. Jsou-li použity, je přenositelnost omezena na takové obvody, které tyto prvky obsahují. Jakákoliv konstrukce je v principu realizovatelná bez těchto prvků (s výjimkou velmi speciálních prvků, jako jsou prvky pro úpravu hodinového signálu, pro řízení odběru z napájecího zdroje apod.), jejich použití však může výsledek syntézy výrazně zlepšit a v mnohých případech je tento výsledek bez nich tak složitý, že je prakticky nepoužitelný (příklady: blokové násobičky a další prostředky pro podporu aritmetiky, paměti RAM atd.). Netlist se zapisuje nejčastěji ve formátu EDIF.

Implementace zahrnuje několik postupných kroků, které vyústí do vytvoření popisu propojení cílového obvodu, který je podkladem pro vytvoření tzv. bitstreamu používaného pro konfiguraci FPGA. Důležité jsou zejména kroky označené Mapping a Place-and-Route. Zhruba řečeno, mapování spočívá v přiřazení obvodových prvků použitých ve výsledku syntézy konkrétním prvkům obsaženým v cílovém obvodu – je analogií výběru konkrétních součástek u návrhu desky plošných spojů. Pak následuje rozmístění (placement) těchto prvků, tedy jejich přiřazení odpovídajícím obvodovým strukturám, které jsou v nenaprogramovaném cílovém obvodu vytvořeny. Zdařilá volba rozmístění má velký význam pro provedení následujícího kroku – propojení (routing), opět je vhodné představit si analogii s návrhem desky plošného spoje. Propojením se vytvoří plán výsledné struktury včetně potřebného nakonfigurování programovatelných propojek. Požadovaný stav jednotlivých propojek je pak obsažen v souboru, který je výsledkem implementace (bitstream).

Dalším výsledkem implementace je simulační model, který je vytvořen v jazyku

VHDL, popř. Verilog a který je doplněn údaji o časových parametrech propojených prvků i spojovacích struktur použitého cílového obvodu (tzv. back-annotation). Tento model sice je zapsán v některém z jazyků HDL, ale je pro člověka prakticky nečitelný, protože je to čistě strukturální popis se stovkami vložených komponent i pro poměrně jednoduché zapojení. Je však použitelný pro časovou simulaci, k níž se používá simulátor, jehož vstupním souborem je soubor zapsaný v jazyku HDL, například ModelSim – podobně jako u funkční simulace.

Programování (u FPGA se častěji užívá pojem konfigurace) – v obvodech PLD představuje nastavení programovatelných propojek. U obvodů FPGA s konfigurační informací uchovávanou ve volatilních prvcích (RAM) je možno přenést konfigurační informaci přímo do těchto prvků z počítače, nebo naprogramovat konfigurační paměť (nejčastěji PROM nebo EEPROM), z níž se konfigurační informace natáhne do FPGA po připojení napájecího napětí nebo po přijetí signálu pro zahájení procesu konfigurace. Dnes se prakticky používá téměř výhradně princip programování v systému (ISP – In-System Programming. Některé firmy (např. Actel) vyrábějí také nevolatilní obvody FPGA s jednorázovým, dále již neměnným nakonfigurováním, pro které se samozřejmě konfigurační paměť nepoužívá.

Velmi důležitou součástí vývoje aplikace pro obvody FPGA je simulace. Rozeznáváme několik druhů simulace podle účelu simulace a podle modelu, který se k ní používá. Funkční simulace na úrovni RTL (Register Transfer Level – behaviorální úroveň) se typicky provádí pro ověření syntaxe zapsaného kódu a k ověření, zda model správně funguje, aniž se přihlíží k časovému rozměru. Tato simulace není závislá na architektuře cílového obvodu, pokud se nepoužívají primitivy jako vložené komponenty. Používá se k ní model představovaný zdrojovým textem, tj. textem, kterým konstruktér popsal svou představu o funkci vyvíjeného zařízení, a provádí se před dalším zpracováním tohoto textu systémem – proto se jí někdy také říká simulace před fittingem nebo před propojením (pre-fit nebo pre-route simulation). Tento model neobsahuje údaje o konkrétních vlastnostech cílového obvodu, jako například údaje o zpoždění a podobně, a je proto poměrně jednoduchý. Funkční simulace je proto obvykle rychlá. Časová simulace používá model generovaný systémem při implementaci. V tomto modelu již mohou být zahrnuty skutečné parametry cílového obvodu a respektováno výsledné propojení jeho strukturních prvků, takže touto simulací můžeme získat podrobné a poměrně přesné údaje o časových poměrech – odtud plyne název tohoto druhu simulace. Podle použitého modelu se jí také říká simulace po fittingu či po propojení (post-fit nebo post-route simulation). Simulace post-fit (post-route) modelu bývá časově mnohem náročnější než funkční simulace.

3.2.1 CORE Generator

Nyní se podrobněji podívejme na součásti vývojového prostředí ISE, jež jsou využity při implementaci OFDM demodulátoru. Jedním z nejdůležitějších nástrojů je CORE Generator, pomocí něhož lze do projektu vložit tzv. IP jádro (Intelectual Property Core). IP jádro je funkční blok, vykonávající některou z často používaných funkcí, který lze vložit do ISE projektu jako celek a programátor tak nemusí ztrácet čas programováním dané funkce. IP jádra mají zpravidla řadu nastavitelných parametrů, čímž je lze optimalizovat pro danou aplikaci, ale i tak může být použití jádra částečně nevýhodné, požadujeme -li např. co nejmenší nároky na cílový obvod – některé části jádra totiž mohou zůstat nevyužité, přesto budou do obvodu implementovány. To je však překážkou jen při specifických aplikacích. IP jádra existují jak volně distribuovatelná, tak i placené verze – mnohdy je výhodnější zakoupení již vyvinutého IP jádra, než placení složitého vývoje.

3.2.2 Fast Fourier Transform v6.0

Xilinx®LogiCORE™IP Fast Fourier Transform [8] je IP jádro sloužící k výpočtu rychlé Fourierovy transformace (FFT) používající algoritmus Cooley-Tukey. Při jeho implementaci lze nastavit množství parametrů – Např. je možné použít jak dopřednou, tak i zpětnou FFT, lze zpracovávat data délky $N = 2m$, $m = 3 - 16$, vstupní data mohou být $8 - 34$ bitová a mohou být vyjádřena jak ve formátu s pevnou, tak i s plovoucí čárkou. Výstupní data mohou být v bitově-převráceném (jak vycházejí z FFT algoritmu) pořadí, nebo je lze přijímat (za cenu malého zpoždění) v přirozeném pořadí.

3.2.3 Block Memory Generator v3.3

Xilinx®LogiCORE™IP Block Memory Generator [12] je další IP jádro. IP Block Memory Generator vytváří paměti RAM (ROM) optimalizované jak z hlediska maximálního výkonu (rychlosti), tak i z hlediska minimální spotřeby hardwaru.

3.2.4 Divider Generator v3.0

Posledním IP jádrem použitým při implementaci je Divider Generator v3.0 [13], který vytváří obvod pro celočíselné dělení. Metoda dělení je založena na algoritmu "Radix-2" nebo "High-Radix" (volitelné - viz uživatelská příručka).

4 IMPLEMENTACE DEMODULÁTORU DO FPGA

4.1 model

K implementaci OFDM demodulátoru navrženého v MATLABu je použito výše popsané vývojové prostředí ISE (kap.3.2). Jako vrcholový modul projektu byl zvolen modul v prostředí Schematic.

Jelikož v době návrhu nebyl k dispozici konkrétní funkční přípravek, do něhož by mohl být model implementován, budeme v následujícím předpokládat implementaci do obvodu Virtex4 XC4VSX35 s tím, že na vstup přichází již podvzorkovaný dvanáctibitový signál v základním pásmu.

4.1.1 Top modul

Schéma vrcholového modulu (*Schema_top*) je zobrazeno v příloze B.1. Prostředí Schematicu je zvoleno záměrně pro jeho názornost - lze jednoduše bez předchozích znalostí jazyka VHDL pochopit funkci a zapojení jednotlivých bloků.

Jako generátor dat je využita paměť ROM (kap.3.2.3). Data vygenerovaná OFDM modulátorem modelu v MATLABu (kap.2.2) jsou pomocí konverzního algoritmu převedena do binární formy (dvojkový doplněk Q12.0). Takto jsou potom vložena do inicializačního souboru ROM paměti. Při inkrementaci adresy ROM paměti podle hodinového signálu pak ROM paměť generuje vstupní signál. Jelikož je potřeba generovat I a Q složku vstupního signálu, jsou do schématu vloženy paměťové moduly dva.

Pro generování hodinového signálu s nulovou a 180° fází je použita komponenta $BaseDigitalClockManagerCircuit$, která generuje např. hodinový signál s fází po 90°, hodinový signál s dvojnásobným kmitočtem aj.

4.1.2 Časová synchronizace

Pro časovou synchronizaci jsou použity dva moduly - *correlator* a $Starter\_FFT$ (zdrojové kódy ve VHDL jsou přilozeny v přílohách B.2 a B.3). Je využita metoda popsaná rovnicí (1.6) a Obr.1.8, která byla ověřena simulací v MATLABu.

Vstupní signál (reálná a imaginární složka zvlášť) přichází do korelátoru (*correlator*), kde je ukládán do posuvného registru délky 1920 bitů (délka časové synchronizační sekvence vynásobená šířkou slova vstupního signálu = 160 * 12).

Jednotlivé vzorky v posuvném registru jsou každý hodinový takt násobeny příslušným vzorkem synchronizační sekvence (tedy jeho imaginárním doplňkem). Získáme tak reálnou a imaginární část součinu (*soucin_re* a *soucin_im*) podle vzorců (4.1),(4.2):

$$\Re(a \cdot b) = \Re(a) \cdot \Re(b) - \Im(a) \cdot \Im(b) \tag{4.1}$$

$$\Im(a \cdot b) = \Re(a) \cdot \Im(b) + \Im(a) \cdot \Re(b) \tag{4.2}$$

V následujícím procesu jsou každý hodinový takt takto vzniklé součiny sečteny - dostáváme reálnou a imaginární část součtu ($soucet\_re + j \cdot soucet\_im$).

Posledním bodem korelace je vypočtení absolutní hodnoty čísla $|soucet\_re + j \cdot soucet\_im|$. Pro potřeby korelace však postačí sečíst kvadráty reálné a imaginární složky - odmocnění by znamenalo zbytečnou matematickou operaci navíc. Je pouze potřeba stanovit vhodnou rozhodovací úroveň pro stanovení maxima korelace.

Výstupem modulu *correlator* je zpožděný vstupní signál a signál *start* indikující dosažení maxima korelace - tedy první část synchronizační sekvence na vstupu demodulátoru.

Tímto je ošetřeno nalezení počátku rámce. Je však ještě potřeba zajistit periodické spouštění FFT tak, aby byl odstraněn CP. Tuto úlohu vykonává modul *Starter_FFT* (Příloha B.2). Ten musí kromě cyklického spouštění FFT s periodou 80 hodinových cyklů (délka OFDM symbolu včetně CP) zaručit i správné dekódování druhé části preamble, která má pro dva OFDM symboly jeden dvojnásobně dlouhý CP.

4.1.3 FFT

Modul rychlé Fourierovy transformace je vytvořen pomocí IP jádra popsaného výše (Kap.3.2.2) s parametry uvedenými v Tab.4.1.

4.1.4 Odstranění pilotních a nulových subnosných

Signál na výstupu FFT obsahuje i složky odpovídající nulovým a pilotním subnosným (z celkového počtu 64 subnosných jsou 4 piloty, 12 nulových subnosných a pouze 48 subnosných je využito pro přenos dat). Pro další zpracování je tedy nutné vyfiltrovat pouze data datových subnosných. Tuto funkci vykonává modul *Pilots_and_Guards_removing* (Příloha B.4).

Vstupní data (výstup FFT) jsou ukládány střídavě do dvou posuvných registrů *tmp*0 a *tmp*1 o délce 64 symbolů. Jako první jsou na výstup čtena data z druhé

Tab. 4.1: Nastavení FFT

Parametr	**Nastavení**	**Parametr**	**Nastavení**
Channels	1	Transform Length	64
Target Clock Frequency	20 MHz	Implementation Options	Pipelined, Streaming I/O
Data Format	Fixed Point	Scaling Options	Unscaled
Input Data Width	12	Phase Factor Width	12
Rounding Models	Truncation	Optional Pins	SCLR
Output Ordering	Natural Order	Input Data Timing	No offset
Nr. Of Stages Using Block RAM	0	Reorder Buffer	Block RAM
Complex Multipliers	Use 3-multiplier structure	Butterfly Arithmetic	Use CLB logic

poloviny registru (proto je nutné použít registry dva), přičemž jsou vynechány data nulových a pilotních subnosných.

4.1.5 Odhad frekvenční charakteristiky kanálu, kompenzace

Jednoduché dělení a násobení vedoucí k získání frekvenční charakteristiky kanálu a následné kompenzaci jejích účinků na přijatý signál podle rovnice (1.12) je ve VHDL poněkud složitější - je potřeba provádět matematické operace s komplexními čísly. Funkci estimace a kompenzace zajišťují tři moduly - *Channel_estimation* (Příloha B.4), *divider* a *delay*35 (Příloha B.6).

Uvažujme nyní vyslaná tréninková data TX_{TR} (druhá část preamble před IFFT), která jsou po průchodu kanálem a FFT přijata jako RX_{TR}. Jelikož jsou v tréninkové sekvenci vyslány pouze reálné symboly $+1$ a -1, resp. $+k$ a $-k$ v integerovské reprezentaci čísel, můžeme pro odhadnutou frekvenční charakteristiku kanálu H_{est} (pro danou nosnou) psát:

$$H_{est} = \frac{RX_{TR}}{TX_{TR}} = \frac{RX_{TR}}{\pm k} \tag{4.3}$$

Z přijatých vzorků signálu nesoucího užitečná data RX se pak snažíme zjistit vyslaná data TX_{est} podle rovnice:

$$TX_{est} = \frac{RX}{H_{est}} = \frac{RX}{\frac{RX_{TR}}{TX_{TR}}} = (\pm k)\frac{RX}{RX_{TR}} \tag{4.4}$$

Pro reálnou a imaginární část vyslaného signálu můžeme tedy psát:

$$\Re\{TX_{est}\} = (\pm k)\frac{(\Re\{RX\}\cdot\Re\{RX_{TR}\}) + (\Im\{RX\}\cdot\Im\{RX_{TR}\})}{\Re\{RX_{TR}\}^2 + \Im\{RX_{TR}\}^2} \tag{4.5}$$

$$\Im\{TX_{est}\} = (\pm k)\frac{(\Im\{RX\}\cdot\Re\{RX_{TR}\}) - (\Re\{RX\}\cdot\Im\{RX_{TR}\})}{\Re\{RX_{TR}\}^2 + \Im\{RX_{TR}\}^2} \tag{4.6}$$

Na vstup modulu *Channel_estimation* (příloha B.5) přichází tedy signál pouze z datových subnosných. Nejdříve je do posuvného registru uložena druhá část přijaté preamble (po odstranění nulových a pilotních nosných je symbol dlouhý 48 vzorků). Pro jednoduchost je využit pouze první OFDM symbol tréninkové sekvence LTS (rov. (2.3)).

V dalším kroku je z jednotlivých vzorků přijaté tréninkové sekvence vypočítán jmenovatel rovnic (4.5) a (4.6) a je opět uložen do registru.

Pro každý vzorek užitečného signálu jsou pak vypočteny čitatele rovnic (4.5) a (4.6), které jsou spolu se jmenovately vypočtenými v předchozím kroku zaslány na výstup modulu *Channel_estimation*.

Protože ISE nepodporuje dělení typů *std_logic_vector* navzájem (přestože je dělení ve standardu VHDL definováno), je potřeba pro získání reálné a imaginární části signálu podle rovnic (4.5) a (4.6) použít děličku - modul *divider*, který je vygenerován pomocí IP jádra *DividerGenerator v*3.0 (kap. 3.2.4). Nastavení děličky je zřejmé z Tab.4.2:

Tab. 4.2: Nastavení děličky

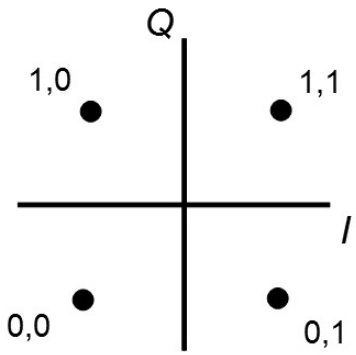
Parametr	**Nastavení**	**Parametr**	**Nastavení**
Algorithm Type	Radix2	Dividend and Quotient Width	32
Divisor Width	20	Remainder Type	Remainder
Operand Sign	Signed	Clocks per Division	1

Na výstupu děličky jsou již užitečná data kompenzovaná odhadnutou frekvenční charakteristikou kanálu. Proces dělení trvá 35 hodinových cyklů - o tuto dobu je potřeba zpozdit signál *data_valid* signalizující platná data na výstupu. Zpoždění vykonává modul *delay*35 (Příloha B.6) realizovaný posuvným registrem.

4.1.6 QPSK dekodér

Posledním blokem zpracovávajícím signál v OFDM demodulátoru je QPSK dekodér reprezentovaný modulem *QPSK_demapper* (viz. příloha B.7). QPSK konstelační diagram je zobrazen na Obr.4.1.

Z něj je zřejmé, že první bit dekódovaného signálu záleží pouze na polaritě Q složky (imaginární části) QPSK signálu a druhý bit záleží na polaritě I složky (reálné části) QPSK signálu. S nástupnou hranou hodinového signálu je tedy na výstup přenesena negace znaménkového bitu vzorku Q signálu a negace znaménkového bitu I signálu je uložena do registru. Ta je na výstup přivedena se sestupnou hranou hodinového signálu.

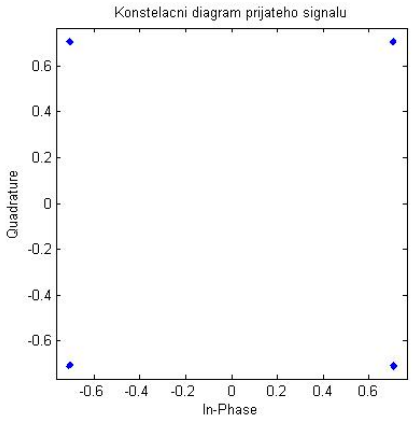


Obr. 4.1: Konstelační diagram QPSK

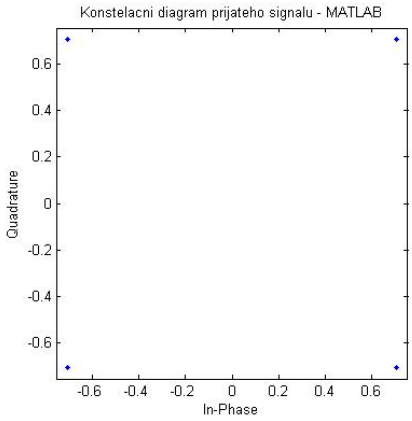
4.2 Behaviorální simulace

4.2.1 Demodulace nezkresleného OFDM signálu

Při behaviorální simulaci byl na vstup demodulátoru přiveden nejdříve nezkreslený OFDM signál. Konstelační diagram demodulovaného signálu prvních dvou OFDM symbolů je zobrazen na Obr.4.2. Srovnání s výsledky simulace v MATLABu je na Obr.4.3.



Obr. 4.2: ISE - Konstelační diagram přijatého signálu bez zkreslení kanálem



Obr. 4.3: MATLAB - Konstelační diagram přijatého signálu bez zkreslení kanálem

Zde se zastavme u metody exportu dat ze simulátoru ISim - v zásadě lze použít metody dvě. První variantou je použití konverzní funkce pro převod dat z formátu *std_logic_vector* do formátu *string* (či slohy obsahující takovou konverzní funkci) a výpis hodnoty sledovaného signálu příkazem *write* při tvorbě testovacího benče.

Méně elegantní, ale jednodušší variantou, je výpis hodnot signálu za použití příkazové konzole simulátoru ISim.

V našem případě byla nakonec použita druhá metoda. Příklad pro výpis Q složky výstupního signálu:

```
run 8250 ns
isim set radix dec
set myvalue [show value qpsk_im_out]
set tempfile [open vystup.txt w]
puts $tempfile $myvalue
run 10 ns
set myvalue [show value qpsk_im_out]
puts $tempfile $myvalue
.
.
.
run 325 ns
set myvalue [show value qpsk_im_out]
puts $tempfile $myvalue
run 10 ns
.
.
.
run 10 ns
set myvalue [show value qpsk_im_out]
puts $tempfile $myvalue
close $tempfile
```

Vzorky jsou v dekadické formě uloženy do souboru *vystup.txt*. Následně je jejich hodnota v MATLABu upravena na rozsah od -1 do $+1$ a je vykreslen konstelační diagram demodulovaného signálu.

Výstupní signály jednotlivých bloků demodulátoru jsou zobrazeny v příloze B.8. Z grafu je patrné zpoždění signálu při průchodu demodulátorem.

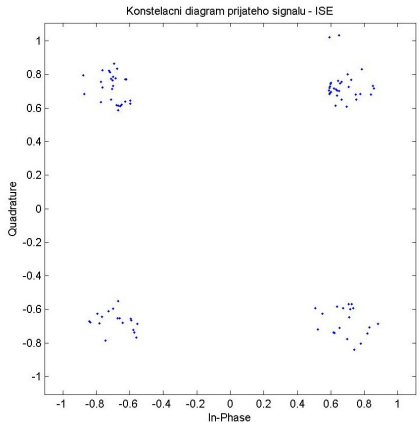
4.2.2 Demodulace OFDM signálu zkresleného riceanovským kanálem

Při demodulaci signálu zkresleného riceanovským kanálem bylo zjištěno, že použití prosté korelace při časové synchronizaci (viz. rovnice(1.6) a Obr.1.8) je nevyhovující. Při simulaci v MATLABu bylo totiž hledáno maximum korelační funkce. Pro implementaci do FPGA by však bylo hledání maxima korelační funkce pro úsek signálu délky celého rámce výpočetně neúnosně náročné. Proto byla zvolena pouze metoda stanovení maxima určením vhodného prahu. Má -li riceanovský kanál parametry uvedené v kapitole 2.3, překračuje korelační funkce práh stanovený pro neporušený signál ve třech okamžicích. Zvýšit rozhodovací práh nelze, protože hodnoty nižších maxim jsou stále vyšší, než maximum korelační funkce pro neporušený signál. Řešením by mohlo být použití Schmidl-Coxova algoritmu (rov.(1.7),(1.8)), kdy je korelace doplněna váhováním podle výkonu signálu, případně zkrácení korelace na jeden STS symbol (korelační funkce by pak měla dosahovat maxima 10x). Z časových důvodů již však nebylo možné hypotézy ověřit. Otázkou zůstává, zda budou reálné hodnoty

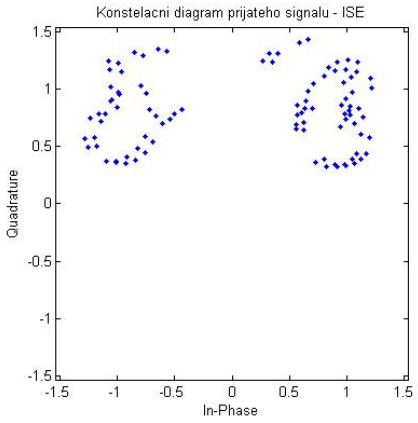
zkreslení po průchodu kanálem dosahovat tak kritických hodnot, jako při simulaci. Pro kabelové vedení bude patrně dostačovat prosté určení maxima korelační funkce prahováním.

V následujícím je použita simulace s posunutým prahem pro určení maxima korelační funkce.

Konstelační diagram dekódovaného signálu po průchodu riceanovským kanálem (6 cest se zpožděním 0, 50, 100, 150 a 500 ns a přenosem -3, 0, 5, -2, -4 a -6 dB, k = 1) je na Obr.4.4. Pro srovnání je na Obr.4.5 zobrazen i konstelační diagram přijatého signálu bez kompenzace odhadnutou frekvenční charakteristikou kanálu.



Obr. 4.4: ISE - Konstelační diagram přijatého signálu po průchodu riceanovským kanálem



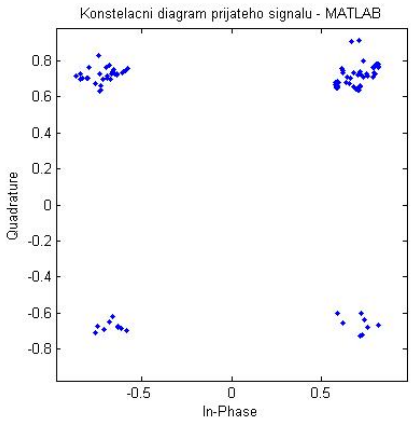
Obr. 4.5: ISE - Konstelační diagram přijatého signálu po průchodu riceanovským kanálem bez kompenzace

Na Obr.4.6 a Obr.4.7 jsou zobrazeny ekvivalentní konstelační diagramy simulace v MATLABu. Rozdílné výsledky simulací jsou patrně způsobeny tím, že na rozdíl od MATLABu je v ISE prováděna časová synchronizace až na vstupním signálu s nižší vzorkovací frekvencí.

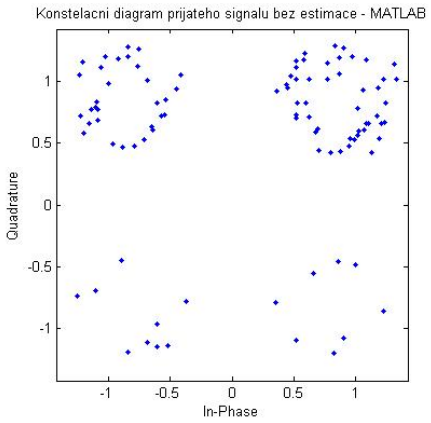
4.2.3 Demodulace OFDM signálu zkresleného riceanovským kanálem s aditivním šumem

V posledním bodu simulace byl na vstup demodulátoru přiveden signál zkreslený výše uvedeným riceanovským kanálem a bílým gaussovským šumem se SNR = 15 dB. Opět je pro srovnání zobrazen konstelační diagram výstupního signálu i konstelační diagram signálu bez kompenzace odhadnutou frekvenční charakteristikou kanálu (Obr.4.8 , Obr.4.9).

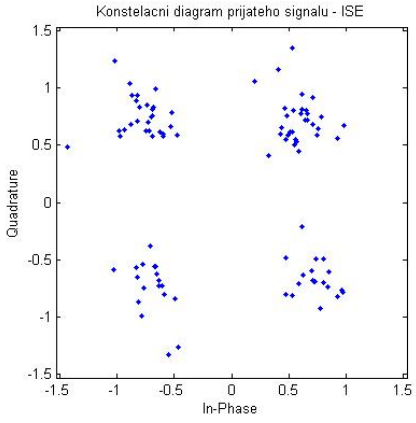
Výsledky totožné simulace v MATLABu jsou na Obr.4.10 a Obr.4.11.



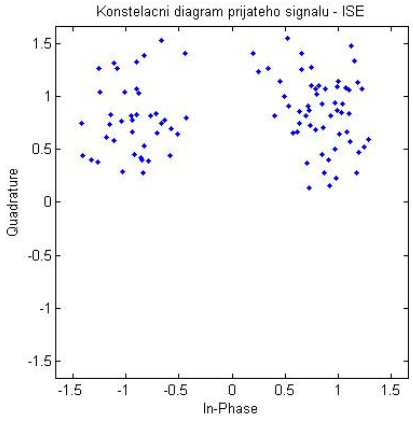
Obr. 4.6: MATLAB - Konstelační diagram přijatého signálu po průchodu riceanovským kanálem



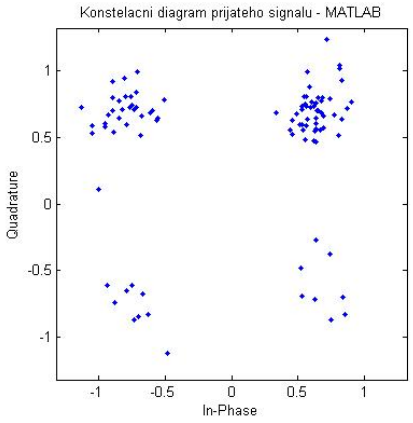
Obr. 4.7: MATLAB - Konstelační diagram přijatého signálu po průchodu riceanovským kanálem bez kompenzace



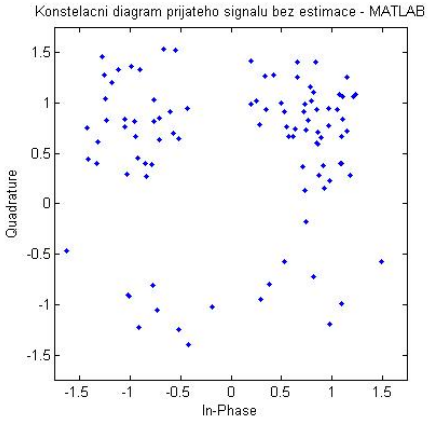
Obr. 4.8: ISE - Konstelační diagram přijatého signálu po průchodu riceanovským kanálem s aditivním šumem



Obr. 4.9: ISE - Konstelační diagram přijatého signálu po průchodu riceanovským kanálem s aditivním šumem bez kompenzace



Obr. 4.10: MATLAB - Konstelační diagram přijatého signálu po průchodu riceanovským kanálem s aditivním šumem



Obr. 4.11: MATLAB - Konstelační diagram přijatého signálu po průchodu riceanovským kanálem s aditivním šumem bez kompenzace

5 ZÁVĚR

V rámci diplomové práce je stručně rozebrán princip OFDM modulace, možnosti synchronizace a odhadu frekvenční charakteristiky kanálu v OFDM.

Byl sestrojen model OFDM systému v MATLABu, jehož funkčnost byla ověřena simulací.

Pode MATLABu byl vytvořen popis OFDM demodulátoru pro implementaci do obvodu FPGA v návrhovém systému ISE, jehož funkčnost byla ověřena behaviorální simulací v simulátoru ISim. Vytvořený popis však není v ISE syntetizovatelný. Implementaci do FPGA se tedy nepodařilo dokončit.

Všechny zdrojové soubory a celý projekt v ISE lze nalézt na přiloženém CD.

LITERATURA

[1] ŽALUD, Václav *Moderní radioelekronika.* 1. vyd. Praha: Nakladatelství BEN, 2000. 653 s. ISBN: 80-86056-47-3 <http://www.boldis.cz/citace/citace.html>.

[2] ŠEBESTA, Vladimír *Teorie sdělování.* 2. vyd. Brno: VUTIUM, 2001. 92 s. ISBN: 80-214-1843-5

[3] CHIUEH, Tzi-Dar; TSAI, Pei-Yun. *OFDM Baseband Receiver Design for Wireless Communications.* 1.vyd. Singapore: John Wiley and Sons, 2007. 258 s. ISBN: 978-0-470-82234-0

[4] MARŠÁLEK, Roman. *Teorie radiové komunikace - přednáška č.10.* [online]. Brno: FEKT VUT v Brně, 3.12.2007 [cit. 28.12.2008] . Dostupný z: <https://krel.feec.vutbr.cz:8009/USR/VYUKA/M_EST/MTRK/prednasky/prednes10/prednes10.pdf>.

[5] KOLOUCH, Jaromír. *Programovatelné logické obvody.* [online]. Brno: FEKT VUT v Brně, 2005 [cit. 30.4.2008]. Dostupný z : <https://krel.feec.vutbr.cz/VYUKA/M_EST/MPLD/Skripta/ProgrLogObv07.pdf>.

[6] MATZ, Gerald. *Wireless OFDM Systems.* Wien: Institut für Nachrichtentechnik und Hochfrequenztechnik TU Wien, přednášky k předmětu U̇bertragungstechnik Vertiefung pro zimní semestr ak.r 2008/2009

[7] KOLOUCH, Jaromír. *Programovatelné logické obvody – počítačové cvičení.* [online]. Brno: FEKT VUT v Brně, 2005 [cit. 30.4.2008]. Dostupný z : <https://www.feec.vutbr.cz/et/skripta/urel/Programovatelne_Log_Obvody_P.pdf>.

[8] *Uživatelská příručka k Xilinx® LogiCORETM IP Fast Fourier Transform.* [online]. [cit. 4.1.2008]. Dostupný z: <http://www.xilinx.com/support/documentation/ip_documentation/xfft_ds260.pdf>.

[9] LIU, Hui, LI, Quoqing. *OFDM-Based Broadband Wireless Networks.* 2. vyd. New Jersey: John Wiley and Sons, 2005. 251 s. ISBN: 978-0-471-72346-2

[10] BAHAI, Ahmad R.S., SALTZBERG, Burton R., ERGEN, Mustafa. *Multi-Carrier Digital Communications: Theory and Applications of OFDM.* 2. vyd. New York: Springer Science+Business Media, 2004. 411 s. ISBN: 0-387-22575-7

[11] European Telecommunications Standards Institute. *ETSI TS 101 475 v 1.3.1 – Broadband Radio Access Networks (BRAN); HIPERLAN Type 2; Physical (PHY) layer.* [online]. [cit.12.2001]. Dostupný z: <http://www.etsi.org>.

[12] *Uživatelská příručka k Xilinx® LogiCORETM IP Block Memory Generator v3.3.* [online]. [cit. 12.2.2010]. Dostupný z: <http://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen_ds512.pdf>.

[13] *Uživatelská příručka k Xilinx® LogiCORETM IP Divider Generator v3.0* [online]. [cit. 12.5.2010]. Dostupný z: <http://www.xilinx.com/support/documentation/ip_documentation/div_gen_ds530.pdf>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

BPSK Binary Phase Shift Keying (binární fázové klíčování)

CLB Configurable Logic Blocks (konfigurovatelný logický blok)

CP Cyclic Prefix (cyklický prefix)

CR Carrier Recovery (obnova referenční nosné vlny)

DAB Digital Audio Broadcasting (digitální rozhlasové vysílání)

DFBL Decision Feedback Loop (smyčka s rozhodovací zpětnou vazbou)

DFT Discrete Fourier Transform (diskrétní Fourierova transformace)

DSP Digital Signal Processor (Digitální signálový procesor)

DAB Digital Video Broadcasting – Terrestrial (pozemní digitální televizní vysílání)

FDM Frequency Division Multiplex (kmitočtový multiplex)

FPGA Field-Programmable Gate Array

ICI Inter-Carrier Interference (interference mezi subnosnými vlnami)

IDFT Inversee Discrete Fourier Transform (Zpětná diskrétní Fourierova transformace)

ISI Inter-Symbol Interference (mezisymbolové interference)

LTS Long Traing Sequence (dlouhá tréninková sekvence)

MCM Multicarrier Modulation (modulace s více nosnými vlnami)

OFDM Orthogonal Fequency Division Multiplex (ortogonální kmitočtový multiplex)

PLD Programmable Logic Device (programovatelný logický obvod)

PSK Phase Shift Keying (fázové klíčování)

QAM Quadrature Amplitude Modulation (kvadraturní amplitudová modulace)

QPSK Quadrature Phase Shift Keying (kvadraturní fázové klíčování)

STR Symbol Timing Recovery (Obnova časování symbolů)

STS	Short Traing Sequence (krátká tréninková sekvence)
W-LAN	VHSIC hardware description language - programovací jazyk HDL
W-LAN	Wireless Local Area Network (lokální bezdrátová síť)
BER	Bit Error Ration (bitová chybovost)
e	eulerovo číslo
E_b	energie signálu na jeden bit
E_s	energie signálu na jeden symbol
f_b	bitová rychlost
f	vzorkovací kmitočet
f_k	kmitočet k-té subnosné
f_{vz}	vzorkovací kmitočet
$H(\omega)$	frekvenční charakteristika kanálu
$h(t)$	impulsní charakteristika kanálu
K	délka OFDM symbolu bez CP [vzorků]
L_{cp}	délka CP [vzorků]
M	počet stavů dané modulace
m	diskrétní čas (m-tý vzorek signálu)
r_m	m-tý vzorek přijatého signálu
s	synchronizační sekvence
S	Toeplitzova matice
SNR	Signal to Noise Ratio (poměr výkonu signálu k výkonu šumu)
t	čas
T_b	bitová perioda
t_m	čas m-tého vzorku
T_{OFDM}	doba trvání OFDM symbolu

T_s	vzorkovací perioda, symbolová perioda
w	bílý šum
x_k	namapovaný vstupní signál k-té subnosné před IDFT
y_k	signál k-té subnosné demodulátoru po DFT
Z	Množina celých čísel
Φ_k	k-tá subnosná
Δf	kmitočtový osdtup subnosných
τ	časové zpoždění signálu mezi vysílačem a přijímačem, časový offset
τ_{est}	odhadovaný časový offset

SEZNAM PŘÍLOH

A ZDROJOVÝ KÓD (M-FILE) SIMULACE V MATLABU

```
close all
    clear
    clc

    N_sd = 48;          %Pocet subnosnych nesoucich data
    N_sp = 4;           %pocet pilotnich subnosnych
    N_st = 52;          %celkovy pocet subnosnych
    T_u = 64;           %Delka trvani uzitecne casti symbolu [vzorku]
    T_cp = 16;          %Delka trvani CP [vzorku]
    T_symb = 80;        %Delka trvani celeho symbolu [vzorku]
    f_s = 20e6;         %Vzorkovaci kmitocet [Hz]
    n = 100*N_sd;       %Pocet vstupnich datovych bitu
    f_c = 100e6;        %Kmitocet 0-te subnosne v RF pasmu
    f_s_rf = 400e6;     %Vzorkovaci kmitocet v RF [Hz]
    T_s_rf = 1/f_s_rf;  %Vzorkovaci perioda v RF [s]

    % Parametry kanalu:
    time_shift = 4321;  %Velikost zpozdeni signalu pri pruchodu kanalem [vzorku v RF]
    f_d = 0;            %Maximalni dopplerovsky posun [Hz]
    k = 1;              %Koeficient k Ricianovskeho kanalu [-]
    tau = [0 0.5e-7 1e-7 1.5e-7 2e-7 5e-7];   %Vektor zpozdeni jednotlivych cest [s]
    pdb = [-3 0 -0.5 -2 -4 -6];      %Vektor prenosu jednotlivych cest [dB]
    snr = 15;           %SNR po pridani bileho gausovskeho sumu [dB]



    %% ----- Vstupni signal -----
    sig = randint(n,1);    % generovani nahodneho signalu

    %% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%% M O D U L A T O R %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %% ----- QPSK Mapovani -----
    % Konverze signalu do 2-bitovych dekadickych cisel %
    sig_symb = bi2de(reshape(sig,2,length(sig)/2).','left-msb');

    % QPSK kodovani %
    maps = pskmod(sig_symb,4,5*pi/4,'gray');



    %% ----- S/P prevodnik -----
    after_SP = reshape(maps,N_sd,length(maps)/N_sd);



    %% ----- Vlozeni pilotnich subnosnych -----
    % Pseudo-nahodna seqence pro pilotni subnosne (generovana podle S(x) = X7 + X4 + 1)
    p = [1,1,1,1,-1,-1,-1,1,-1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,1,1,-1,1,1,1,1,1,1,-1,1,1,1,1,-1,1,1,-1,1,1,-1,1,-1,1,1,-1,
        -1,1,1,1,1,1,-1,-1,1,1,-1,-1,1,-1,1,-1,1,1,1,-1,-1,1,-1,-1,-1,1,-1,-1,1,-1,1,1,1,1,1,-1,1,-1,1,1,-1,1,-1,1,-1,
        1,1,1,-1,-1,1,-1,-1,-1,1,1,1,1,-1,-1,1,-1,-1,-1,1,-1,-1];

    p_long = [];
    for n_th_p = 1:fix(length(after_SP(1,:))/length(p))
    p_long = [p_long,p];
    end
    if mod(length(after_SP(1,:)),length(p))
       p_long = [p_long,p(1:mod(length(after_SP(1,:)),length(p)))];
    end

    % Vlozeni pilotu
    with_pilots = [after_SP(1:5,:);p_long;after_SP(6:18,:);p_long;after_SP(19:24,:);zeros(1,length(p_long));after_SP(25:30,:);
              p_long;after_SP(31:43,:);p_long;after_SP(44:48,:)];



    %% ----- Vlozeni Preamble -----
    % Kvuli jednoduchosti je preamble vlozena pouze na zacatek signalu - v
    % realu se vklada na zacatek kazdeho ramce.
```

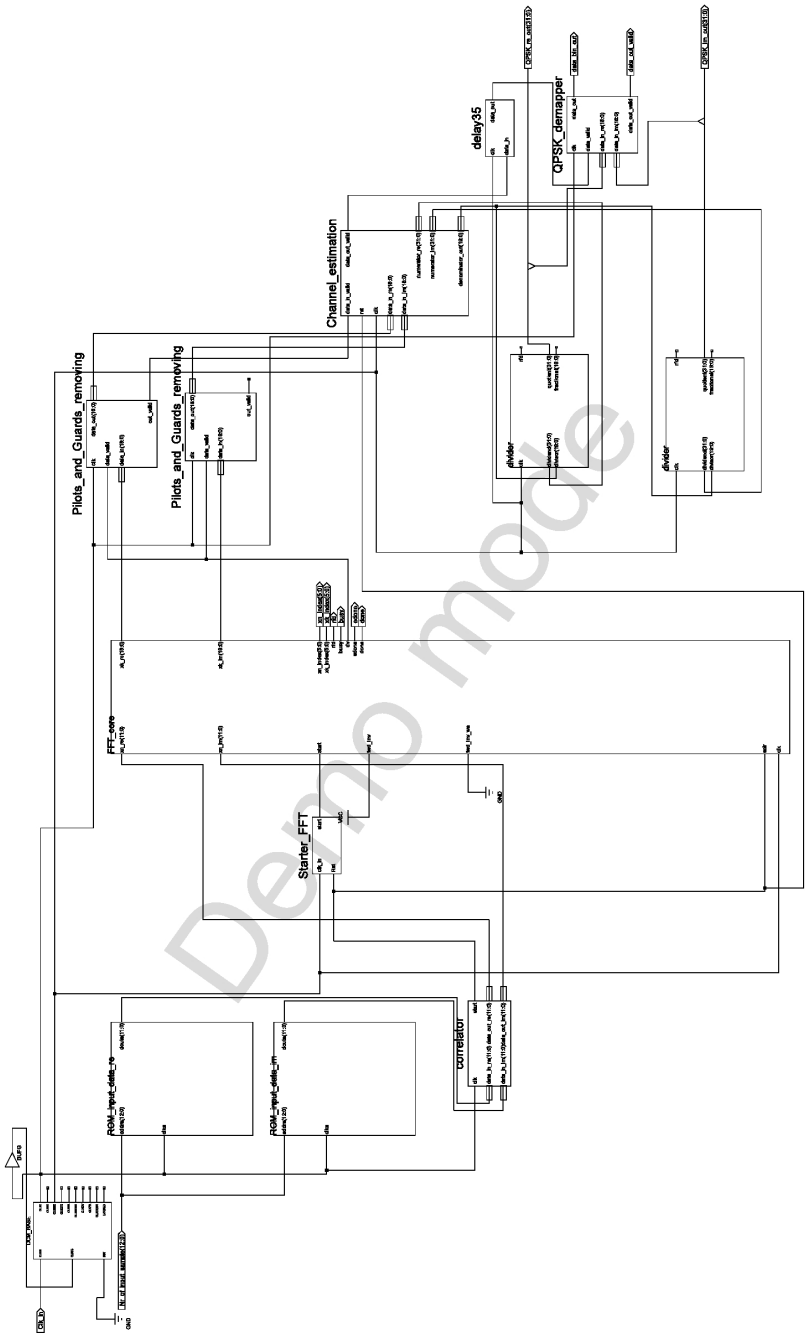
```
% Kratka treninkova sekvence
STS = transpose(sqrt(13/6) * [0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,0,0,0,1+j,
0,0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0]);

% Dlouha treninkova sekvence
LTS = transpose([1,1,-1,-1,1,1,-1,1,1,1,1,1,1,1,-1,-1,1,1,-1,1,-1,1,1,1,1,1,0,1,-1,-1,1,1,-1,1,-1,1,1,-1,-1,-1,-1,-1,1,1,1,-1,-1,1,-1,-1,1,-1,1,1,1,1]);

with_TS = [STS,STS,LTS,LTS,with_pilots];


%% ----- Vlozeni nul -----
with_guards = [with_TS;zeros(11,length(p_long)+4)];


%% ----- IFFT -----
before_IFFT = [with_guards(27:64,:);with_guards(1:26,:)];
after_IFFT = ifft(before_IFFT);


%% ----- Vlozeni CP -----
with_CP = [after_IFFT(1:16,:);after_IFFT];


%% ----- P/S prevodnik -----
after_PS = reshape(with_CP,1,numel(with_CP));

% Uprava Preamble (po 10 STS nasleduji 2LTS s dvojnasobnym CP)
with_preamble = [after_PS(1:160),after_PS(289:320),after_PS(177:240),after_PS(257:320),after_PS(321:end)];

        % Spektrum OFDM signalu modulatoru v zakladnim pasmu
        OFDM_spectrum = abs(fftshift(fft(with_preamble(321:end))/length(with_preamble(321:end))));
        f_vect =(-f_s/2):(f_s/length(with_preamble(321:end))):(f_s/2)-(f_s/length(with_preamble(321:end)));
        figure();
        plot(f_vect,log(OFDM_spectrum));
        xlabel('f[Hz]')
        ylabel('S[dB]')
        title('Spektrum OFDM signalu modulatoru v zakladnim pasmu');


%% ----- Modulace signalu do RF pasma -----
OFDM_resample = resample(with_preamble,f_s_rf,f_s);
   % Prevzorkovani signalu (vzorkovaci frekvence musi byt minimalne
   % dvakrat vyssi, nez maximalni frekvence modulovaneho signalu v RF
   % pasmu)

OFDM_rf = [];
for n_th_sample = 1 : length(OFDM_resample)
   OFDM_rf = [OFDM_rf, sqrt(2)*real(OFDM_resample(n_th_sample) * exp(j*2*pi*f_c*(n_th_sample-1)*T_s_rf))];
end

        % Spektrum vysilaneho OFDM signalu v prenosvem pasmu
        OFDM_spectrum = abs(fftshift(fft(OFDM_rf)/length(OFDM_rf)));
        f_vect =(-f_s_rf/2):(f_s_rf/length(OFDM_rf)):(f_s_rf/2)-(f_s_rf/length(OFDM_rf));
        figure();
        plot(f_vect,log(OFDM_spectrum));
        xlabel('f[Hz]')
        ylabel('S[dB]')
        title('Spektrum vysilaneho OFDM signalu v prenosovem pasmu');

        % Spektrum prvni casti preamble vysilaneho OFDM signalu v prenosvem pasmu
        OFDM_spectrum = abs(fftshift(fft(OFDM_rf(1:3200))/length(OFDM_rf(1:3200))));
        f_vect =(-f_s_rf/2):(f_s_rf/length(OFDM_rf(1:3200))):(f_s_rf/2)-(f_s_rf/length(OFDM_rf(1:3200)));
        figure();
        plot(f_vect,log(OFDM_spectrum));
        xlabel('f[Hz]')
        ylabel('S[dB]')
        title('Spektrum prvni casti preamble OFDM signalu v prenosovem pasmu');

        % Spektrum druhe casti preamble vzsilaneho OFDM signalu v prenosvem pasmu
        OFDM_spectrum = abs(fftshift(fft(OFDM_rf(3201:6400))/length(OFDM_rf(3201:6400))));
        f_vect =(-f_s_rf/2):(f_s_rf/length(OFDM_rf(3201:6400))):(f_s_rf/2)-(f_s_rf/length(OFDM_rf(3201:6400)));
        figure();
        plot(f_vect,log(OFDM_spectrum));
```

```
        xlabel('f[Hz]')
        ylabel('S[dB]')
        title('Spektrum druhe casti preamble vysilaneho OFDM signalu v prenosovem pasmu');

        % Vysilany signal v casove oblasti (preamble + 2 OFDM symboy)
        time = [0:T_s_rf:(4*T_symb/f_s-T_s_rf)];
        figure();
        plot(time,OFDM_rf(1:(4*T_symb*f_s_rf/f_s)));
        xlabel('t[s]')
        ylabel('S[dB]')
        title('Vysilany signal v casove oblasti (preamble + 2 OFDM symboy)');



%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% P R U C H O D   K A N A L E M %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% ----- Casovy posun -----
% Kvuli jednoduchosti je pouzito cyklicke posunuti
if (time_shift>0)
    time_shifted = [OFDM_rf((length(OFDM_rf)-time_shift+1):length(OFDM_rf)) OFDM_rf(1:(length(OFDM_rf)-time_shift))];
else
    time_shifted = [OFDM_rf(-time_shift+1:length(OFDM_rf)) OFDM_rf(1:-time_shift)];
end



%% ----- Fading -----
chan = ricianchan(1/f_s_rf,f_d,k,tau,pdb);
after_fading = filter(chan,time_shifted);



%% ----- Pridani bileho sumu -----
OFDM_rx = awgn(after_fading,snr,'measured');



%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% D E M O D U L A T O R %%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% ----- Presun signalu do zakladniho pasma -----
OFDM_rx_h = hilbert(OFDM_rx);
OFDM_rx_bb = [];
for n_th_sample = 1 : length(OFDM_rx)
    OFDM_rx_bb = [OFDM_rx_bb, OFDM_rx_h(n_th_sample) * exp(-j*2*pi*f_c*(n_th_sample-1)*T_s_rf)];
end

        % Spektrum prijateho signalu v zakladnim pasmu
        OFDM_spectrum = abs(fftshift(fft(OFDM_rx_bb)/length(OFDM_rx_bb)));
        f_vect =(-f_s_rf/2):(f_s_rf/length(OFDM_rx_bb)):(f_s_rf/2)-(f_s_rf/length(OFDM_rx_bb));
        figure();
        plot(f_vect,log(OFDM_spectrum));
        xlabel('f[Hz]')
        ylabel('S[dB]')
        title('Spektrum prijateho signalu v zakladnim pasmu');



%% ----- Casova synchronizace -----
% Odhad zpozdeni (korelace prijateho signalu s prvni casti preamble)
for n_th_round = 1 : (f_s_rf/f_s)
    correlation(n_th_round,:)= xcorr(OFDM_rx_bb(n_th_round:f_s_rf/f_s:end-f_s_rf/f_s+n_th_round),with_preamble(1:160));
end
correlation_s=reshape(correlation,1,numel(correlation));
[max_value,max_index] = max(abs(correlation_s));
time_shift_est = max_index - length(OFDM_rx_bb) + f_s_rf/f_s - 1

% Posunuti signalu na zacatek preamble
if (time_shift_est>=0)
    time_estimated = [OFDM_rx_bb(time_shift_est+1:length(OFDM_rx_bb)) OFDM_rx_bb(1:time_shift_est)];
    else
    time_estimated = [OFDM_rx_bb((length(OFDM_rx_bb)+time_shift_est+1):length(OFDM_rx_bb))
      OFDM_rx_bb(1:(length(OFDM_rx_bb)+time_shift_est))];
end
```

```
%% ----- Prevzorkovani signalu -----
downsampled = downsample(time_estimated,f_s_rf/f_s);



%% ----- S/P prevodnik  -----
after_SP2 = reshape(downsampled,T_symb,length(downsampled)/T_symb);



%% ----- Odstraneni CP ------
without_CP = [after_SP2(17:80,1:2),[after_SP2(33:80,3);after_SP2(1:16,4)],after_SP2(17:80,4:end)];



%% ----- FFT -----
after_FFT = fft(without_CP);



%% ----- Odhad kanalovych koeficientu ve frekv. oblasti -----
H_1 = after_FFT(:,3)./before_IFFT(:,3);
H_2 = after_FFT(:,4)./before_IFFT(:,3);

H_est = (H_1 + H_2)/2;

for n_th_row = 1 : length(H_est)
    after_frequency_fading_estimation(n_th_row,:)= after_FFT(n_th_row,:) / H_est(n_th_row);
end



%% ----- Odstraneni Preamble -----
without_preamble = after_frequency_fading_estimation(:,5:end);



%% ----- Odstraneni Pilotu a Nul -----
without_G = [without_preamble(39:64,:);without_preamble(2:27,:)];

without_P = [without_G(1:5,:);without_G(7:19,:);without_G(21:32,:);without_G(34:46,:);without_G(48:52,:)];



%% ----- P/S prevodnik -----
serial = reshape(without_P,1,numel(without_P));

            % Konstelacni diagram prijateho signalu
            scatterplot(serial);
            title('Konstelacni diagram prijateho signalu');



            %Konstelacni diagram prijateho signalu bez odhadu kanalu
            without_preamble_c = after_FFT(:,5:end);
            without_G_c = [without_preamble_c(39:64,:);without_preamble_c(2:27,:)];
           without_P_c = [without_G_c(1:5,:);without_G_c(7:19,:);without_G_c(21:32,:);without_G_c(34:46,:);without_G_c(48:52,:)];
            serial_c = reshape(without_P_c,1,numel(without_P_c));
            scatterplot(serial_c);
            title('Konstelacni diagram prijateho signalu bez odhadu kanalu');



%% ----- QPSK demapovani -----
demod = pskdemod(serial,4,5*pi/4,'gray');
rx_bin = reshape(rot90(fliplr(de2bi(demod,'left-msb'))),numel(de2bi(demod)),1);



%% ----- Vypocet BER -----
BER = sum(rx_bin~=sig)/length(sig)
```

B KONFIGURACE FPGA

B.1 Schéma vrcholového modulu



B.2 correlator - zdrojový kód VHDL

```
---- Uncomment the following library declaration if instantiating
     ---- any Xilinx primitives in this code.
     --library UNISIM;
     --use UNISIM.VComponents.all;


     entity correlator is
         Port ( data_in_re : in  STD_LOGIC_VECTOR (11 downto 0);
                data_in_im : in  STD_LOGIC_VECTOR (11 downto 0);
                clk : in  STD_LOGIC;
                data_out_re : out  STD_LOGIC_VECTOR (11 downto 0);
                data_out_im : out  STD_LOGIC_VECTOR (11 downto 0);
           start: out STD_LOGIC := '0');
     end correlator;


     architecture Behavioral of correlator is
       signal tmp_re: std_logic_vector (1919 downto 0) := (others => '0');
       signal tmp_im: std_logic_vector (1919 downto 0) := (others => '0');
       signal soucin_re: std_logic_vector (3839 downto 0) := (others => '0');
       signal soucin_im: std_logic_vector (3839 downto 0) := (others => '0');
       signal soucet_re: std_logic_vector (182 downto 0):= (others => '0');
       signal soucet_im: std_logic_vector (182 downto 0):= (others => '0');
       signal abs_h: std_logic_vector (365 downto 0):= (others => '0');


     begin


       process (clk)  -- ukladani  vstupnich dat do posuvneho registru
       begin
         if (clk'event and clk='1') then
           for i in 1 to 159 loop
             tmp_re((((i+1)*12)-1) downto ((i)*12)) <= tmp_re(((i*12)-1) downto ((i-1)*12));
             tmp_im((((i+1)*12)-1) downto ((i)*12)) <= tmp_im(((i*12)-1) downto ((i-1)*12));
           end loop;
           tmp_re(11 downto 0) <= data_in_re;
           tmp_im(11 downto 0) <= data_in_im;
         end if;
       end process;


       nasobeni: process (clk)    -- nasobeni prichozich vzorku se vzorky synchronizacni sekvence
       constant synch_re: std_logic_vector(1919 downto 0) := X"05eef0fe41240bc124fe4ef005e004f5ffe6000fe6f5f0040
     5eef0fe41240bc124fe4ef005e004f5ffe6000fe6f5f00405eef
     0fe41240bc124fe4ef005e004f5ffe6000fe6f5f00405eef0fe4
     1240bc124fe4ef005e004f5ffe6000fe6f5f00405eef0fe41240
     bc124fe4ef005e004f5ffe6000fe6f5f00405eef0fe41240bc12
     4fe4ef005e004f5ffe6000fe6f5f00405eef0fe41240bc124fe4
     ef005e004f5ffe6000fe6f5f00405eef0fe41240bc124fe4ef00
     5e004f5ffe6000fe6f5f00405eef0fe41240bc124fe4ef005e00
     4f5ffe6000fe6f5f00405eef0fe41240bc124fe4ef005e004f5f
     fe6000fe6f5f004";
       constant synch_im: std_logic_vector(1919 downto 0) := X"fa1ffb0a00190000190a0ffbfa110f01bedbf43edb01b10ff
     a1ffb0a00190000190a0ffbfa110f01bedbf43edb01b10ffa1ff
     b0a00190000190a0ffbfa110f01bedbf43edb01b10ffa1ffb0a0
     0190000190a0ffbfa110f01bedbf43edb01b10ffa1ffb0a00190
     000190a0ffbfa110f01bedbf43edb01b10ffa1ffb0a001900001
     90a0ffbfa110f01bedbf43edb01b10ffa1ffb0a00190000190a0
     ffbfa110f01bedbf43edb01b10ffa1ffb0a00190000190a0ffbf
     a110f01bedbf43edb01b10ffa1ffb0a00190000190a0ffbfa110
     f01bedbf43edb01b10ffa1ffb0a00190000190a0ffbfa110f01b
     edbf43edb01b10f;
       begin
         if (clk'event and clk='0') then
           for i in 0 to 159 loop
             soucin_re(((i*24)+23) downto (i*24)) <= ((tmp_re(((i*12)+11) downto (i*12)) * synch_re(((i*12)+11) downto (i*12))) -
(tmp_im(((i*12)+11) downto (i*12)) * synch_im(((i*12)+11) downto (i*12))));
             soucin_im(((i*24)+23) downto (i*24)) <= ((tmp_re(((i*12)+11) downto (i*12)) * synch_im(((i*12)+11) downto (i*12)))
+ (tmp_im(((i*12)+11) downto (i*12)) * synch_re(((i*12)+11) downto (i*12))));
           end loop;
         end if;
       end process nasobeni;


       scitani: process (clk)    -- soucet jednotlivych soucinu
       variable soucet_re_var: std_logic_vector(182 downto 0) := (others => '0');
       variable soucet_im_var: std_logic_vector(182 downto 0) := (others => '0');
       begin
         soucet_re_var := (others => '0');
```

```
    soucet_im_var := (others => '0');
    if (clk'event and clk='1') then
      for i in 0 to 159 loop
        soucet_re_var := (soucet_re_var + soucin_re(((i*24)+23) downto (i*24)));
        soucet_im_var := (soucet_im_var + soucin_im(((i*24)+23) downto (i*24)));
      end loop;
      soucet_re <= soucet_re_var;
      soucet_im <= soucet_im_var;
    end if;
  end process scitani;

  abs_hodnota: process (clk)              -- Vypocet absolutni hodnoty souctu:
  begin                             -- matematicky spravne by mel byt vysledek odmocnen,
    if (clk'event and clk='0') then         -- ale misto toho staci posunout rozhodovaci uroven
        abs_h <= (soucet_re * soucet_re) + (soucet_im * soucet_im); -- korelatoru
    end if;

  end process abs_hodnota;

  start_proces: process (clk,abs_h)
  begin
    if (clk'event and clk='1') then
      if (abs_h > X"3c0000000000") then
        start <= '1';
      else start <= '0';
      end if;
      data_out_re <= tmp_re(((3*12)+11) downto (3*12));
      data_out_im <= tmp_im(((3*12)+11) downto (3*12));
    end if;
  end process start_proces;



end Behavioral;
```

B.3 Starter_FFT - zdrojový kód VHDL

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


    ---- Uncomment the following library declaration if instantiating
    ---- any Xilinx primitives in this code.
    --library UNISIM;
    --use UNISIM.VComponents.all;


    entity Starter_FFT is
        Port ( clk_in, Rst : in  STD_LOGIC;
               start : out  STD_LOGIC := '0');
    end Starter_FFT;


    architecture Behavioral of Starter_FFT is
    begin
      Pocitadlo: PROCESS (clk_in, Rst)
        VARIABLE i: integer range 0 to 412 := 0;
      BEGIN
        IF Rst = '1' THEN i := 157;
        ELSIF (clk_in'event AND clk_in = '1') THEN
          case i is
            when 0 =>         -- cekani na inicializacni signal                    (rst) od korelatoru
              i := 0;
              start <= '0';
            when 1 to 188 =>     -- prvni cast preamble + CP druhe casti                    preamble
              i := i + 1;
              start <= '0';
            when 189 to 253 =>  -- druha cast preamble (start impuls musi prijit o
              i := i + 1;  --2 periody driv kvuli zpozdeni FFT)
              start <= '1';
            when 254 to 332 =>
              i := i + 1;
              start <= '0';
            when 333 =>
              i := i + 1;
              start <= '1';
            when 334 to 411 =>
              i := i + 1;
              start <= '0';
            when 412 =>
              i := 333;
              start <= '0';
          end case;
        END IF;
      END PROCESS Pocitadlo;

    end Behavioral;
```

B.4 Pilots_and_Guards_removing - zdrojový kód VHDL

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


    entity Pilots_and_Guards_removing is
        Port ( clk : in  STD_LOGIC;
               data_in : in  STD_LOGIC_VECTOR (18 downto 0);
               data_valid : in  STD_LOGIC;
               data_out : out  STD_LOGIC_VECTOR (18 downto 0) := "0000000000000000000";
           out_valid : out STD_LOGIC := '0');
    end Pilots_and_Guards_removing;


    architecture Behavioral of Pilots_and_Guards_removing is
      signal tmp0: std_logic_vector(1215 downto 0);    -- sirka = (64 * sirka vstupniho signalu)
      signal tmp1: std_logic_vector(1215 downto 0);    -- sirka = (64 * sirka vstupniho signalu)
      signal tmp_n : std_logic := '0';
      signal n_th_sample: integer range 100 downto 0 := 0;
    begin


      process (clk,data_valid,tmp_n)  -- ukladani do posuvneho registru c.0
      begin
        if (tmp_n = '0') then
          if (data_valid = '1') then
            if (clk'event and clk='1') then
              for i in 1 to 63 loop
                tmp0((((i+1)*19)-1) downto ((i)*19)) <= tmp0(((i*19)-1) downto ((i-1)*19));
              end loop;
              tmp0(18 downto 0) <= data_in;
            end if;
          end if;
        end if;
      end process;


      process (clk,data_valid,tmp_n)  -- ukladani do posuvneho registru c.1
      begin
        if (tmp_n = '1') then
          if (data_valid = '1') then
            if (clk'event and clk='1') then
              for i in 1 to 63 loop
                tmp1((((i+1)*19)-1) downto ((i)*19)) <= tmp1(((i*19)-1) downto ((i-1)*19));
              end loop;
              tmp1(18 downto 0) <= data_in;
            end if;
          end if;
        end if;
      end process;


      process (clk, data_valid,tmp_n,n_th_sample)  -- cteni dat
        variable n_th_symbol: integer range 1000 downto 0 := 0;
      begin
        if (data_valid'event and data_valid ='1') then
          tmp_n <= not tmp_n;       -- zmena aktivniho posuvneho registru
          n_th_symbol := n_th_symbol + 1;
        end if;


        if n_th_symbol >= 2 then
          if (tmp_n = '1') then     -- cteni dat z posuvneho registru c.0 (vymeni se horni a dolni polovina vzorkuu a vynechaji se
piloty a nuly)
            if (clk'event and clk = '1') then
              case n_th_sample is
                when 0=>
                  data_out <= "0000000000000000000";
                  out_valid <= '0';
                when 1=>
                  data_out <= tmp0(493 downto 475);
                  out_valid <= '1';
                when 2 =>
                  data_out <= tmp0(474 downto 456);
                  out_valid <= '1';
                when 3 =>
                  data_out <= tmp0(455 downto 437);
```

```
  out_valid <= '1';
when 4 =>
  data_out <= tmp0(436 downto 418);
  out_valid <= '1';
when 5 =>
  data_out <= tmp0(417 downto 399);
  out_valid <= '1';
when 6 =>
  data_out <= tmp0(379 downto 361);
  out_valid <= '1';
when 7 =>
  data_out <= tmp0(360 downto 342);
  out_valid <= '1';
when 8 =>
  data_out <= tmp0(341 downto 323);
  out_valid <= '1';
when 9 =>
  data_out <= tmp0(322 downto 304);
  out_valid <= '1';
when 10 =>
  data_out <= tmp0(303 downto 285);
  out_valid <= '1';
when 11 =>
  data_out <= tmp0(284 downto 266);
  out_valid <= '1';
when 12 =>
  data_out <= tmp0(265 downto 247);
  out_valid <= '1';
when 13 =>
  data_out <= tmp0(246 downto 228);
  out_valid <= '1';
when 14 =>
  data_out <= tmp0(227 downto 209);
  out_valid <= '1';
when 15=>
  data_out <= tmp0(208 downto 190);
  out_valid <= '1';
when 16 =>
  data_out <= tmp0(189 downto 171);
  out_valid <= '1';
when 17 =>
  data_out <= tmp0(170 downto 152);
  out_valid <= '1';
when 18 =>
  data_out <= tmp0(151 downto 133);
  out_valid <= '1';
when 19 =>
  data_out <= tmp0(113 downto 95);
  out_valid <= '1';
when 20 =>
  data_out <= tmp0(94 downto 76);
  out_valid <= '1';
when 21 =>
  data_out <= tmp0(75 downto 57);
  out_valid <= '1';
when 22 =>
  data_out <= tmp0(56 downto 38);
  out_valid <= '1';
when 23 =>
  data_out <= tmp0(37 downto 19);
  out_valid <= '1';
when 24 =>
  data_out <= tmp0(18 downto 0);
when 25 =>
  data_out <= tmp0(1196 downto 1178);
  out_valid <= '1';
when 26 =>
  data_out <= tmp0(1177 downto 1159);
  out_valid <= '1';
when 27 =>
  data_out <= tmp0(1158 downto 1140);
  out_valid <= '1';
when 28 =>
  data_out <= tmp0(1139 downto 1121);
  out_valid <= '1';
when 29 =>
  data_out <= tmp0(1120 downto 1102);
  out_valid <= '1';
when 30 =>
```

```
              data_out <= tmp0(1101 downto 1083);
              out_valid <= '1';
            when 31 =>
              data_out <= tmp0(1063 downto 1045);
              out_valid <= '1';
            when 32 =>
              data_out <= tmp0(1044 downto 1026);
              out_valid <= '1';
            when 33 =>
              data_out <= tmp0(1025 downto 1007);
              out_valid <= '1';
            when 34 =>
              data_out <= tmp0(1006 downto 988);
              out_valid <= '1';
            when 35 =>
              data_out <= tmp0(987 downto 969);
              out_valid <= '1';
            when 36 =>
              data_out <= tmp0(968 downto 950);
              out_valid <= '1';
            when 37 =>
              data_out <= tmp0(949 downto 931);
              out_valid <= '1';
            when 38 =>
              data_out <= tmp0(930 downto 912);
              out_valid <= '1';
            when 39 =>
              data_out <= tmp0(911 downto 893);
              out_valid <= '1';
            when 40 =>
              data_out <= tmp0(892 downto 874);
              out_valid <= '1';
            when 41 =>
              data_out <= tmp0(873 downto 855);
              out_valid <= '1';
            when 42 =>
              data_out <= tmp0(854 downto 836);
              out_valid <= '1';
            when 43 =>
              data_out <= tmp0(835 downto 817);
              out_valid <= '1';
            when 44=>
              data_out <= tmp0(797 downto 779);
              out_valid <= '1';
            when 45 =>
              data_out <= tmp0(778 downto 760);
              out_valid <= '1';
            when 46 =>
              data_out <= tmp0(759 downto 741);
              out_valid <= '1';
            when 47 =>
              data_out <= tmp0(740 downto 722);
              out_valid <= '1';
            when 48 =>
              data_out <= tmp0(721 downto 703);
              out_valid <= '1';
            when 49 =>
              data_out <= "0000000000000000000";
              out_valid <= '0';
            when others =>
              out_valid <= '0';
          end case;
        end if;
      end if;


      if (tmp_n = '0') then     -- cteni dat z posuvneho registru c.1 (vymeni se horni a dolni polovina vzorkuu a vynechaji se piloty a nuly)
        if (clk'event and clk = '1') then
          case n_th_sample is
            when 0=>
              data_out <= "0000000000000000000";
              out_valid <= '0';
            when 1=>
              data_out <= tmp1(493 downto 475);
              out_valid <= '1';
            when 2 =>
              data_out <= tmp1(474 downto 456);
              out_valid <= '1';
            when 3 =>
```

```
  data_out <= tmp1(455 downto 437);
  out_valid <= '1';
when 4 =>
  data_out <= tmp1(436 downto 418);
  out_valid <= '1';
when 5 =>
  data_out <= tmp1(417 downto 399);
  out_valid <= '1';
when 6 =>
  data_out <= tmp1(379 downto 361);
  out_valid <= '1';
when 7 =>
  data_out <= tmp1(360 downto 342);
  out_valid <= '1';
when 8 =>
  data_out <= tmp1(341 downto 323);
  out_valid <= '1';
when 9 =>
  data_out <= tmp1(322 downto 304);
  out_valid <= '1';
when 10 =>
  data_out <= tmp1(303 downto 285);
  out_valid <= '1';
when 11 =>
  data_out <= tmp1(284 downto 266);
  out_valid <= '1';
when 12 =>
  data_out <= tmp1(265 downto 247);
  out_valid <= '1';
when 13 =>
  data_out <= tmp1(246 downto 228);
  out_valid <= '1';
when 14 =>
  data_out <= tmp1(227 downto 209);
  out_valid <= '1';
when 15=>
  data_out <= tmp1(208 downto 190);
  out_valid <= '1';
when 16 =>
  data_out <= tmp1(189 downto 171);
  out_valid <= '1';
when 17 =>
  data_out <= tmp1(170 downto 152);
  out_valid <= '1';
when 18 =>
  data_out <= tmp1(151 downto 133);
  out_valid <= '1';
when 19 =>
  data_out <= tmp1(113 downto 95);
  out_valid <= '1';
when 20 =>
  data_out <= tmp1(94 downto 76);
  out_valid <= '1';
when 21 =>
  data_out <= tmp1(75 downto 57);
  out_valid <= '1';
when 22 =>
  data_out <= tmp1(56 downto 38);
  out_valid <= '1';
when 23 =>
  data_out <= tmp1(37 downto 19);
  out_valid <= '1';
when 24 =>
  data_out <= tmp1(18 downto 0);
when 25 =>
  data_out <= tmp1(1196 downto 1178);
  out_valid <= '1';
when 26 =>
  data_out <= tmp1(1177 downto 1159);
  out_valid <= '1';
when 27 =>
  data_out <= tmp1(1158 downto 1140);
  out_valid <= '1';
when 28 =>
  data_out <= tmp1(1139 downto 1121);
  out_valid <= '1';
when 29 =>
  data_out <= tmp1(1120 downto 1102);
  out_valid <= '1';
```

```
            when 30 =>
              data_out <= tmp1(1101 downto 1083);
              out_valid <= '1';
            when 31 =>
              data_out <= tmp1(1063 downto 1045);
              out_valid <= '1';
            when 32 =>
              data_out <= tmp1(1044 downto 1026);
              out_valid <= '1';
            when 33 =>
              data_out <= tmp1(1025 downto 1007);
              out_valid <= '1';
            when 34 =>
              data_out <= tmp1(1006 downto 988);
              out_valid <= '1';
            when 35 =>
              data_out <= tmp1(987 downto 969);
              out_valid <= '1';
            when 36 =>
              data_out <= tmp1(968 downto 950);
              out_valid <= '1';
            when 37 =>
              data_out <= tmp1(949 downto 931);
              out_valid <= '1';
            when 38 =>
              data_out <= tmp1(930 downto 912);
              out_valid <= '1';
            when 39 =>
              data_out <= tmp1(911 downto 893);
              out_valid <= '1';
            when 40 =>
              data_out <= tmp1(892 downto 874);
              out_valid <= '1';
            when 41 =>
              data_out <= tmp1(873 downto 855);
              out_valid <= '1';
            when 42 =>
              data_out <= tmp1(854 downto 836);
              out_valid <= '1';
            when 43 =>
              data_out <= tmp1(835 downto 817);
              out_valid <= '1';
            when 44=>
              data_out <= tmp1(797 downto 779);
              out_valid <= '1';
            when 45 =>
              data_out <= tmp1(778 downto 760);
              out_valid <= '1';
            when 46 =>
              data_out <= tmp1(759 downto 741);
              out_valid <= '1';
            when 47 =>
              data_out <= tmp1(740 downto 722);
              out_valid <= '1';
            when 48 =>
              data_out <= tmp1(721 downto 703);
              out_valid <= '1';
            when 49 =>
              data_out <= "0000000000000000000";
              out_valid <= '0';
            when others =>
              out_valid <= '0';
          end case;
        end if;
      end if;
    end if;

  end process;

  process(clk,data_valid)
  begin
    if (data_valid ='0') then
      n_th_sample <= 0;
    elsif (clk'event and clk = '0') then
      n_th_sample <= n_th_sample + 1;
    end if;
  end process;

end Behavioral;
```

B.5 Channel_estimation - zdrojový kód VHDL

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

    ---- Uncomment the following library declaration if instantiating
    ---- any Xilinx primitives in this code.
    --library UNISIM;
    --use UNISIM.VComponents.all;

    entity Channel_estimation is
        Port ( data_in_re : in  STD_LOGIC_VECTOR (18 downto 0);
               data_in_im : in  STD_LOGIC_VECTOR (18 downto 0);
               data_in_valid : in  STD_LOGIC;
           rst : in  STD_LOGIC;
               clk : in  STD_LOGIC;
               numerator_re : out  STD_LOGIC_VECTOR (31 downto 0);
               numerator_im : out  STD_LOGIC_VECTOR (31 downto 0);
               data_out_valid : out  STD_LOGIC := '0';
           denominator_out: out std_logic_vector (19 downto 0):= (others => '0')
           );



    end Channel_estimation;

    architecture Behavioral of Channel_estimation is
      type state is (s0,s1,s2);
      signal train_rx: state := s0;

      signal koef_re: std_logic_vector (911 downto 0) := (others => '0');
      signal koef_im: std_logic_vector (911 downto 0) := (others => '0');

      constant k: std_logic_vector(11 downto 0) := "011111111111";

      signal sum_re: STD_LOGIC_VECTOR (37 downto 0);
      signal sum_im: STD_LOGIC_VECTOR (37 downto 0);

      signal denominator: std_logic_vector (1823 downto 0):= (others => '0');

    begin



      process (clk,rst,train_rx,data_in_valid)  -- ukladani  vstupnich dat do posuvneho registru
      begin
        if rst = '1' then
          train_rx <= s0;
        elsif (rising_edge(data_in_valid)) then
          case train_rx is
            when s0 => train_rx <= s1;
            when s1 => train_rx <= s2;
            when others => train_rx <= s2;
          end case;
        end if;
        if (train_rx = s1 and data_in_valid = '1') then
          if (clk'event and clk='1') then
            for i in 1 to 47 loop
              koef_re((((i+1)*19)-1) downto ((i)*19)) <= koef_re(((i*19)-1) downto ((i-1)*19));
              koef_im((((i+1)*19)-1) downto ((i)*19)) <= koef_im(((i*19)-1) downto ((i-1)*19));
            end loop;
            koef_re(18 downto 0) <= data_in_re;
            koef_im(18 downto 0) <= data_in_im;
          end if;
        end if;
      end process;



      process (train_rx)
      begin
        if (train_rx'event and train_rx = s2) then
          for i in 0 to 47 loop
            denominator(((i*38)+37) downto (i*38)) <= ((koef_re(((i*19)+18) downto (i*19)) * koef_re(((i*19)+18) downto (i*19)))
+ (koef_im(((i*19)+18) downto (i*19)) * koef_im(((i*19)+18) downto (i*19))));
          end loop;
```

```
      end if;
    end process;


    process(clk,data_in_valid,train_rx,rst)
      variable n_th_sample: integer range 100 downto 0 := 100;
    begin
      if rst = '1' then
        n_th_sample := 100;
      end if;
      if (data_in_valid'event and data_in_valid = '1' and train_rx = s2) then
        n_th_sample := 0;
      end if;
      if (clk'event and clk='1') then
        if (n_th_sample <= 47) then
         sum_re <= (data_in_re * koef_re((911-(n_th_sample*19)) downto (911-(n_th_sample*19)-18))) + (data_in_im * koef_im((911-
(n_th_sample*19)) downto (911-(n_th_sample*19)-18)))     ; --/ denominator((1823-(n_th_sample*38)) downto (1823-(n_th_sample*38)-
37)));


         sum_im <= (data_in_im * koef_re((911-(n_th_sample*19)) downto (911-(n_th_sample*19)-18))) - (data_in_re * koef_im((911-
(n_th_sample*19)) downto (911-(n_th_sample*19)-18)))     ; --/ denominator((1823-(n_th_sample*38)) downto (1823-(n_th_sample*38)-
37)));
        end if;
      end if;
      if (clk'event and clk='0') then
        case n_th_sample is
          when 0 | 1 | 4 | 6 | 8 to 13 | 16 to 18 | 20 to 24 | 27 | 28 | 31 | 37 | 38 | 41 | 44 to 47 =>
            numerator_re <= (k * sum_re(24 downto 4));
            numerator_im <= (k * sum_im(24 downto 4));
            denominator_out <= denominator((1823-(n_th_sample*38)-13) downto (1823-(n_th_sample*38)-32));
            data_out_valid <= '1';
            n_th_sample := n_th_sample + 1;
          when 2 | 3 | 5 | 7 | 14 | 15 | 19 | 25 | 26 | 29 | 30 | 32 to 36 | 39 | 40 | 42 | 43 =>
            numerator_re <= ((-k) * sum_re(24 downto 4));
            numerator_im <= ((-k) * sum_im(24 downto 4));
            denominator_out <= denominator((1823-(n_th_sample*38)-13) downto (1823-(n_th_sample*38)-32));
            data_out_valid <= '1';
            n_th_sample := n_th_sample + 1;
          when others =>
            data_out_valid <= '0';
            numerator_re <= (others => '0');
            numerator_im <= (others => '0');
        end case;


      end if;
    end process;



  end Behavioral;
```

B.6 delay35 - zdrojový kód VHDL

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

    ---- Uncomment the following library declaration if instantiating
    ---- any Xilinx primitives in this code.
    --library UNISIM;
    --use UNISIM.VComponents.all;

    entity delay35 is
        Port ( clk : in  STD_LOGIC;
               data_in : in  STD_LOGIC;
               data_out : out  STD_LOGIC);
    end delay35;

    architecture Behavioral of delay35 is

      signal tmp: std_logic_vector(34 downto 0) := (others => '0');

    begin

      process (clk)  -- ukladani  vstupnich dat do posuvneho registru
      begin
        if (clk'event and clk='1') then
          for i in 1 to 34 loop
            tmp(i) <= tmp(i-1);
          end loop;
          tmp(0) <= data_in;
          data_out <= tmp(34);
        end if;
      end process;

    end Behavioral;
```

B.7 QPSK_demapper - zdrojový kód VHDL

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


    ---- Uncomment the following library declaration if instantiating
    ---- any Xilinx primitives in this code.
    --library UNISIM;
    --use UNISIM.VComponents.all;


    entity QPSK_demapper is
        Port ( data_in_re : in  STD_LOGIC_VECTOR (18 downto 0);
               data_in_im : in  STD_LOGIC_VECTOR (18 downto 0);
               clk : in  STD_LOGIC;
               data_valid : in  STD_LOGIC;
               data_out : out  STD_LOGIC := '0';
          data_out_valid : out STD_LOGIC := '0');
    end QPSK_demapper;


    architecture Behavioral of QPSK_demapper is


      signal tmp : std_logic := '0';
    begin


      process (clk, data_valid)
      begin
        if data_valid = '1' then
          if(clk'event and clk = '1') then
            data_out <= not data_in_im(18);
            tmp <= not data_in_re(18);
            data_out_valid <= '1';
          end if;
          if(clk'event and clk = '0') then
            data_out <= tmp;
            data_out_valid <= '1';
          end if;
        elsif(clk'event and clk = '1') then
          data_out <= '0';
          data_out_valid <= '0';
        end if;
      end process;


    end Behavioral;
```

B.8 Zpoždění signálu při průchodu demodulátorem - behaviorální simulace v ISim