

**Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií**

FIIT-13428-47963

Bc. Maroš Ďuríček

**ZABEZPEČOVACÍ SYSTÉM VIACERÝCH
ČASOVAČOV NA BÁZE PROGRAMOVATEĽNÉHO
OBVODU**

Diplomová práca

Študijný program: Počítačové a komunikačné systémy a siete

Študijný odbor: 9.2.4 Počítačové inžinierstvo

Miesto vypracovania: Ústav počítačových systémov a sietí, FIIT STU Bratislava

Vedúci diplomovej práce: Ing. Mária Pohronská

máj 2012

Podakovanie

Na tomto mieste by som sa chcel poďakovať všetkým, ktorí mi či už poradili alebo ma usmernili pri vypracovávaní tejto práce, najmä však Ing. Márii Pohronskej za jej odbornú pomoc.

ANOTÁCIA

Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLOGIÍ

Študijný program: Počítačové a komunikačné systémy a siete

Autor: Bc. Maroš Ďuríček

Diplomová práca: ZABEZPEČOVACÍ SYSTÉM VIACERÝCH ČASOVAČOV NA BÁZE
PROGRAMOVATEĽNÉHO OBVODU

Vedúci diplomovej práce: Ing. Mária Pohronská

máj 2012

V tejto diplomovej práci sú analyzované bezpečnostné požiadavky na dodržanie doby odozvy systémov reálneho času, opis interných a externých watchdog časovačov a ich použitie v systémoch. Ďalšia časť analyzuje existujúci systém viacerých časovačov na báze FPGA, jeho architektúru a princíp fungovania.

Práca obsahuje originálny návrh zabezpečovacieho systému s viacerými hardvérovými watchdog časovačmi, ktorý komunikuje cez navrhnutý komunikačný protokol cez dve komunikačné USB rozhrania. Výsledkom práce je implementovaný funkčný prototyp systému v programovateľnom obvode Xilinx Spartan-3A.

ANNOTATION

Slovak University of Technology Bratislava

FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Degree Course: Computer and communication systems and networks

Author: Bc. Maroš Ďuríček

Master's Theses: SECURING SYSTEM WITH MULTIPLE FPGA HARDWARE WATCH-
DOGS

Supervisor: Ing. Mária Pohronská

2012, May

This thesis analyzes the security requirements for compliance response time in real-time systems and a description of internal and external watchdog timers and their use in systems. Another part analyzes the existing multiple FPGA watchdog timers system, its architecture and principle of operation.

This thesis contains original design of securing system with multiple hardware watchdog timers which communicates with the designed communication protocol through two USB interfaces. The outcome of this thesis is functional prototype of the system implemented in programmable circuit Xilinx Spartan-3A.

Obsah

1	Úvod	1
1.1	Použité pojmy	2
1.2	Použité skratky	2
2	Analýza problému	3
2.1	Systémy reálneho času s viacerými procesmi	3
2.1.1	Oblasti zabezpečenia	4
2.2	Zabezpečenie požiadaviek na dodržanie doby odozvy	4
2.2.1	Watchdog časovač	4
	Interný watchdog časovač	5
	Externý watchdog časovač	5
	Požiadavky na robustný watchdog časovač	6
2.3	Existujúci systém viacerých časovačov na báze FPGA	6
2.3.1	Princíp fungovania	7
2.3.2	Architektúra systému	7
2.3.3	Zhodnotenie systému	8
2.3.4	Vhodné rozšírenia	8
2.4	Komunikačné rozhrania	9
2.4.1	UART	10
2.4.2	USB	10
2.4.3	PCI Express	11
2.5	Výber zariadenia vhodného pre implementáciu	12
2.5.1	Virtex-5 FX70T	13
2.5.2	Avnet Spartan-3A Evaluation Kit	14
2.6	Výber komunikačného modulu	16
3	Návrh	17
3.1	Požiadavky na systém	17
3.1.1	Výber vhodného komunikačného rozhrania	17
3.2	Architektúra systému	18
3.2.1	Watchdog časovač	19
3.2.2	Zapínač	19
3.2.3	Prioritný kódovač/dekodér	19
3.2.4	Riadiaca jednotka	20

3.2.5	Komunikátor	21
3.2.6	Komunikačný protokol	22
4	Implementácia	25
4.1	Použitý jazyk a vývojové prostredia	25
4.2	Použité bloky z iných zdrojov	25
4.3	Princíp implementácie jednotlivých blokov	26
4.3.1	Blok syntetizovateľného oneskorenia	26
4.3.2	Pripojenie k hosťovskému systému	27
4.4	Overenie implementácie	28
4.4.1	Simulácia	28
4.4.2	Syntéza	28
4.4.3	Ladenie v FPGA obvode	30
4.4.4	Chyby objavené počas ladenia a ich náprava	31
4.5	Implementácia veľkej architektúry	32
4.5.1	Komunikačné rozhranie	33
4.5.2	Výsledky syntézy veľkej architektúry	35
5	Testovanie	37
5.1	Funkčné testovanie	37
5.1.1	Meranie odozvy času v komunikácii	39
6	Zhodnotenie	41
7	Technická dokumentácia	43
7.1	Pripojenie FTDI USB modulu	43
7.2	Rozhrania modulov	45
7.2.1	Rozhranie celého systému	45
7.2.2	Rozhranie bloku viacerých watchdogov	46
7.2.3	Rozhranie watchdog časovača	46
7.2.4	Rozhranie zapínača	47
7.2.5	Rozhranie prioritného kódovača/dekodéra	47
7.2.6	Rozhranie 4-bitového dekodéra	49
7.2.7	Rozhranie deličky hodinového signálu	49
7.2.8	Rozhranie riadiacej jednotky	49
	Rozhranie syntetizovateľného oneskorenia	50
7.2.9	Rozhranie komunikátora	51
7.2.10	Rozhranie použitého modulu UART	52
7.2.11	Rozhranie blokov typu FT245 pre paralelný prenos	52
7.3	Technická príručka	53
7.3.1	Postup pri nahrávaní logiky do obvodu	53
7.3.2	Nastavenie USB modulu	54
	Literatúra	55

A	Schéma architektúry	A-1
B	Kompletné diagramy konečných stavových automatov	B-1
C	Používateľská príručka	C-1
D	Publikovaný článok na medzinárodnej konferencii Kybernetika a Informatika 2012	D-1
E	Odoslaný článok na medzinárodnú konferenciu Applied Electronics 2012	E-1
F	Obsah priloženého média	F-1

1 Úvod

Vnorené ale aj štandardné systémy reálneho času sú čím ďalej, tým viac zložitejšie. Pred niekoľkými rokmi bolo štandardné používať len niekoľko procesov a stačil na zabezpečenie doby odozvy len jeden hardvérový watchdog časovač. V dnešnej dobe vidíme zložité systémy s operačnými systémami, na ktorých sa vykonávajú aj stovky procesov. Ak sa jedná o systém, ktorý potrebuje zvýšenú spoľahlivosť, je potrebné vyriešiť kontrolu jednotlivých spustených procesov iným spôsobom ako doteraz, t.j. s použitím softvérového watchdog časovača.

Zabezpečovanie vnorených systémov patrí medzi dôležitú časť návrhu, či už sa jedná o zabezpečenie procesov reálneho času, integrity pamäti systému alebo vykonávaného kódu. Keďže mnoho systémov využíva len jeden (alebo žiadny) hardvérový watchdog časovač, použitie takéhoto časovača pri viacerých procesoch je neefektívne, pretože viacero softvérových časovačov využíva jeden hardvérový. Ak príde k zlyhaniu jedného z procesov, nedá sa jednoznačne určiť, o ktorý proces sa jedná. Väčšinou sa rešartuje celý systém, čo je v niektorých prípadoch úplne zbytočné.

Ideálny stav by bol, keby každý z procesov spustených v systémoch mal pridelený vlastný hardvérový watchdog časovač. Pri systémoch s viac ako 100 procesmi sa to zdá byť neefektívne, no ak sa watchdog časovače vhodne umiestnia do rýchlej programovateľnej logiky, môžu obslúžiť aj takýto veľký počet procesov. Softvérová záťaž pre kontrolu jednotlivých procesov (softvérový watchdog) by sa znížila, čím by sa zvýšil výkon takýchto systémov. Práve takéto riešenie je predkladané v tejto práci, pričom veľký počet hardvérových watchdog časovačov je realizovaný v rýchlom FPGA obvode.

Cieľom tejto práce je teda navrhnúť a otestovať systém, ktorý bude schopný zabezpečiť systémy s viacerými spustenými procesmi a to práve pomocou hardvérových watchdog časovačov. Navrhovaný systém predpokladá prítomnosť softvérového ovládača na systéme, ktorý bude zabezpečovať.

1.1 Použité pojmy

- Watchdog časovač - špeciálny typ časovača, ktorý deteguje zlyhanie systému (bližšia definícia je v 2.2.1). Niekde je uvedený aj ako watchdog počítadlo
- Hostovský systém, zabezpečovaný systém - systém, ku ktorému je pripojený zabezpečovací systém
- Zabezpečovací systém - systém viacerých hardvérových watchdog časovačov, ktorý je predmetom návrhu tejto práce

1.2 Použité skratky

- FPGA - Field-Programmable Gate Array (pole logických členov programovateľné používateľom)
- USB - Universal Serial Bus - univerzálna sériová zbernica
- UART - Universal Asynchronous Receiver/Transmitter - univerzálny asynchrónny sériový vysielač/prijímač
- RS-232 - štandard pre sériové komunikačné rozhranie - sériový port
- PCIe - PCI Express zbernica
- RT - Real Time - reálny čas
- LCD - Liquid Crystal Display - displej s kvapalnými kryštálmi
- CMOS - Complementary Metal Oxide Semiconductor - dopĺňujúci sa kov-oxid-polovodič
- I2C - Inter-Integrated Circuit sériová zbernica
- SATA - Serial Advanced Technology Attachment - sériová zbernica ATA
- PS/2 - Personal System/2 konektor
- LED - Light-Emitting Diode - luminiscenčná dióda
- FIFO - First in, first out - rad, pri ktorom sa prvé vložené dáta vyberajú najskôr
- VHDL - very-high-speed integrated circuits hardware description language - hardvérový opisný jazyk pre veľmi rýchle integrované obvody

2 Analýza problému

V nasledovnej kapitole sa nachádza analýza problematiky watchdog časovačov, existujúceho systému viacerých časovačov a komunikačných rozhraní, pomocou ktorých bude zabezpečovací systém komunikovať.

Pri hľadaní riešenia v zabezpečení procesov je potrebné porozumieť existujúcim systémom, analyzovať ich nedostatky a navrhnúť vylepšenia, medzi ktoré patrí vyššia spoľahlivosť, rýchlosť reakcie, komunikácia a pod.

2.1 Systémy reálneho času s viacerými procesmi

Na dokončenie práce a doručenie služieb na základe času je potrebný systém reálneho času (RT). Inými slovami, RT systém má prísne požiadavky na časovanie, ktoré musia byť splnené [1].

RT systém je systém, ktorého doba odozvy je ohraničená a teda má zvýšenú spoľahlivosť. RT systém s viacerými procesmi je taký systém, ktorý obsahuje niektorý z mechanizmov prepínania procesov, pretože na ňom môžu byť spustené viaceré procesy súčasne. Logicky vyplýva, že RT systém, na ktorom beží len jeden proces má vyššiu pravdepodobnosť bezchybnej prevádzky ako n-procesový, ktorý má n-krát nižšiu pravdepodobnosť práve kvôli počtu spustených procesov v systéme. Je teda potrebné nejakým spôsobom zvýšiť spoľahlivosť viacprocesového RT systému a práve watchdog časovače dokážu zabrániť ich zlyhaniu.

Spôsob zabezpečenia požiadaviek na dodržanie časovania alebo doby odozvy je v súčasnosti riešený pomocou hardvérových časovačov v spolupráci so softvérovými watchdog časovačmi alebo pomocou jednoduchých obvodov hardvérových watchdog časovačov. Pri viacerých procesoch sa rieši zabezpečenie buď všetkých procesov spoločne bez ohľadu na to, koľko procesov je spustených, alebo sa to rieši inými spôsobmi, ktoré využívajú len jeden obvod.

2.1.1 Oblasti zabezpečenia

Existujú rôzne oblasti zabezpečovania systémov. Napríklad hlavná pamäť počítača môže často zlyhať, preto sa používajú záložné bloky alebo napríklad zabezpečenie paritou, kontrola integrity dát a programu a pod. Iné oblasti sa sústreďujú na zabezpečenie vstupno-výstupných operácií a periférií, zabezpečenie komunikácie rôznymi opravnými kódmi a rôzne ďalšie oblasti ktoré sa venujú šifrovaniu dát a zabezpečenie informácií z pohľadu aplikácií.

Táto práca sa venuje zabezpečeniu požiadaviek na dodržanie doby odozvy systému, pretože každý RT systém musí dodržať svoj stanovený čas doby odozvy najmä ak sa jedná o tzv. *hard-RT*, kde nesmie dôjsť k prekročeniu stanoveného času.

2.2 Zabezpečenie požiadaviek na dodržanie doby odozvy

Systémy reálneho času musia interagovať s externým prostredím tak, aby spĺňali preddefinovaný čas na interakciu ([2], s. 543). Tento čas sa nazýva dobou odozvy systému, pretože reprezentuje čas, ktorý je potrebný na spracovanie požiadavky a následne jej výstup do vonkajšieho prostredia.

V tomto dokumente budeme rozumieť dobu odozvy aj v zmysle prítomnosti systému alebo jeho častí, pretože každá časť systému môže v rôznych časoch mať inú dobu odozvy.

Ak je potrebné zabezpečiť odozvu systému, a teda aj prípad keď systém prestane pracovať, používa sa watchdog časovač uvedený v podkapitole 2.2.1. Pri práci s vnorenými systémami reálneho času je vhodné prikloniť sa k hardvérovému časovaču. Pri zabezpečovaní rôznych iných systémov alebo ich softvérových častí je možné použiť softvérový watchdog časovač.

2.2.1 Watchdog časovač

Watchdog časovač je periféria, ktorá poskytuje možnosť detekcie zlyhania systémového softvéru. Je to v podstate počítadlo, ktoré je periodicky reštartované softvérom na počiatočnú hodnotu tak, že počas normálneho vykonávania operácií toto počítadlo nikdy nedosiahne nulu. Ak by z rôznych dôvodov došlo k takémuto stavu, vygeneruje sa signál pre reštart procesora, nemaskovateľné prerušenie alebo iná systémová udalosť ([2], s. 448). Teda z vyššej definície vyplýva, že watchdog časovač poskytuje mechanizmus, ktorý zaisťuje, že náš softvér pracuje správne. Ak sa použije obsluha prerušenia, môžu sa nahráť informácie o dôvode zlyhania spolu s ich počtom, ktoré môžu byť neskôr vyhodnotené technikom na prípadnú výmenu zariadenia ([3], s. 3-2). Tieto definície watchdog časovačov sú len voľnými definíciami,

pretože takéto zariadenia je možné využiť aj napríklad na zabezpečenie hardvérových častí systémov ako sú obvody, napájanie a iné.

Niektoré z možných chýb v systéme, ktoré môžu nastať a pri použití watchdog časovača je ich možné ošetriť:

- Nekonečný cyklus ako výsledok nesprávneho vyhodnotenia parametrov
- Ukazovateľ do zásobníka obsahuje neplatnú adresu, na ktorej sa nenachádza začiatok inštrukcie
- Stavové registre procesov obsahujú neplatné informácie
- Hodiny procesora nepracujú korektne
 - v prípade, že watchdog časovač má nezávislý hodinový vstup
- Zlyhanie napájania
 - v prípade, že watchdog časovač má nezávislé napájanie
- ...

V tejto práci je potrebné prihliadať na to, že zabezpečovaný systém bude mať viac procesov, ktoré sa budú nezávisle kontrolovať. Z tohto pohľadu môžu vzniknúť aj iné chyby ako napríklad chybné ukončenie procesu z dôvodu nesprávne načítaných parametrov.

Interný watchdog časovač

Väčšina moderných procesorov obsahuje interné watchdog časovače. Väčšinou sa jedná o jeden časovač, ktorý slúži na ochranu aplikácie na najnižšej systémovej úrovni. Podľa [2] má interný časovač niekoľko nevýhod, no na základnú ochranu postačuje.

Jednou z nevýhod je, že niektoré interné časovače realizujú reštart procesora len nastavením pôvodnej hodnoty do programového počítadla (*program counter*), pričom sa spoliehajú, že sa vnútorné periférie zotavia samy. Niektoré časovače namiesto reštartu generujú len nemaskovateľné prerušenie, čo v niektorých prípadoch nezotaví celý systém. Ďalšou nevýhodou môže byť ľahké prepísanie vnútorných stavov časovača jednoduchou vstupno-výstupnou operáciou, ktorá môže nastať pri nekontrolovanom vykonávaní neznámeho chybového kódu. Ten môže dokonca aj zakázať (vypnúť) časovač. Pri správnom reštartovaní systému je potrebné reštartovať aj jednotlivé vonkajšie a vnútorné periférie systému, pretože môžu obsahovať chybné stavové informácie.

Externý watchdog časovač

Externý watchdog časovač prispieva k väčšej ochrane systému, pretože môže fungovať ako samostatná nezávislá jednotka. Pri použití iného hodinového signálu (*clock*), ktorý nie je zdieľaný s procesorom, je možné detegovať aj zlyhanie oscilátora a uviesť systém do bezpečného stavu.

Medzi externé časovače patria aj tzv. časovače s oknom (*Window watchdog*) spomenuté v [2]. Tieto oknové watchdog časovače sa vedia vyhnúť vypršaniu časovača zápisom dvoch rôznych hodnôt (bajtov) za sebou, ale vedia aj ošetriť udalosti, ktoré nielenže prídu neskôr, ale aj príliš skoro. Takýto systém vie dokonca detegovať prídlhé štartovanie systému alebo priskorú reakciu na nejakú udalosť, pred ktorou sa napr. mal vykonávať náročný výpočet.

Požiadavky na robustný watchdog časovač

Podľa literatúry z ([2], s. 389) vieme vlastnosti robustného watchdog časovača zhrnúť nasledovne:

- Nepredpokladať známy stav po reštarte systému generovaným časovačom - to znamená, že ak dôjde k reštartu celého systému, watchdog časovač nesmie predpokladať žiadny známy stav systému, od ktorého bude pokračovať
- Hardvér, ktorý uvedie systém do bezpečného stavu - najmä ak príde k elektrickým skratom, aby nedošlo k poškodeniu systému, ale aby sa systém korektne dostal do bezpečného stavu
- Po vypršaní časovača sa vygeneruje hardvérový reštart - nie iba nemaskovateľné prerušenie
- Reštartujú sa aj vnútorné a vonkajšie periférie - aby nenastali prípady, že v registroch periférie ostanú nekompletné povely, ktoré sa doplnia neznámymi údajmi
- Zaručí sa, že program nemôže preprogramovať watchdog časovač - žiadny používateľský program nesmie mať plný prístup k watchdog časovaču
- Ponechávajú sa záznamy z ladiacich výstupov pri chybách - na neskoršiu diagnostiku
- Použiť *jumper*, ktorý pri vložení prepne časovač do ladiaceho stavu (vypne sa hardvérový reštart) - aby pri testovaní nebol spúšťaný hardvérový reštart
- Použijú sa napríklad svetelné diódy na indikáciu zlyhania - pri každom reštarte systému musí zariadenie indikovať, že k reštartu došlo

2.3 Existujúci systém viacerých časovačov na báze FPGA

Návrh zabezpečovacieho systému bude inšpirovaný z práce od Ing. Pohronskej opísaný v [4]. V tejto časti je opísaný podrobnejšie. Softvérovej časti systému - ovládaču sa venuje vo svojej práci Ing. Abaffy opísanej v [5].

2.3.1 Princíp fungovania

Jedná sa o zabezpečenie systémov s viacerými procesmi tak, že každý proces v systéme má pridelený jeden hardvérový watchdog časovač, ktorý zabezpečuje jeho dobu odozvy. Komunikáciu zabezpečuje ovládač na hosťovskom systéme a prebieha zatiaľ pomocou sériového rozhrania s prepojením RS-232.

Po vyžiadaní hosťovským systémom je pridelený a zapnutý jeden časovač, do ktorého sa vloží počiatočná hodnota prijatá z hosťovského systému pomocou komunikačného rozhrania. Ďalej je prijímaný signál zabezpečujúci reštart časovača. Ak časovač dosiahne nulu, vygeneruje signál pretečenia a systém zistí jeho adresu, ktorú pošle hosťovskému systému s príznakom, že časovač pretiekol. Softvérový ovládač potom zabezpečí vygenerovanie nemaskovateľného prerušenia pre procesor, ktorý zabezpečí buď zastavenie procesu alebo jeho zotavenie (podľa naimplementovaného ovládača).

2.3.2 Architektúra systému

Existuje viacero architektúr, pretože systém je navrhnutý modulárne, aby sa dal jednoducho rozširovať. Architektúra je hierarchická, zatiaľ navrhnutá do dvoch úrovní s 8-bitovou adresnou zbernicou a predpokladá sa použitie 16-bitovej adresy pre 2^{16} počítačiel.

Základnou funkčnou jednotkou je jednoduché počítaadlo (časovač), ktoré obsahuje vstupné hodinové signály pre počítanie a pre nahrávanie hodnoty (*Clk*). Ďalšími vstupmi sú reštart a jednotbitový sériový vstup pre načítanie hodnoty do watchdog časovača (*sLoad*). Výstupom je len jednotbitový signál pretečenia (*overflow*). Takto jednoduché počítačdlá je možno zapájať do viacerých úrovní, aby sa dala architektúra rozširovať. Každé počítaadlo má ešte pridelené hradlo, ktoré určuje, či je počítaadlo zapnuté. Počítačdlá sú navrhnuté ako 16 alebo 24-bitové (podľa potreby), no veľkosť je možné meniť.

Riadiaca jednotka zabezpečuje rozhranie medzi komunikačnou linkou a počítačdlami. S počítačdlami je prepojená pomocou adresnej zbernice a signálmi *sLoad* a *Reset*. Zabezpečuje zapínanie počítačdiel, adresovanie a nahrávanie hodnôt, ako aj ich reštart. Má dva výstupy hodín - rýchlejšie hodiny pre nahratie hodnôt a pomalšie pre počítanie watchdog časovačov (najmenší čas 1 ms je postačujúci). Riadiaca jednotka obsahuje výstupy pre komunikačné ro-

zhranie *UART*, do ktorého paralelne posiela dáta. Vstupy tvoria pretečenie a réžia, pomocou ktorej sa zistí adresa konkrétneho počítačla.

Na zistenie adresy počítačla je potrebný kódovač, ktorý je v tomto prípade prioritný, pretože sa predpokladá pretečenie viacerých počítačiel súčasne. Na najvyššej úrovni je potrebné preniesť signál (dekódovať ho), ktorý zapne len určený dekodér, aby sa nezapisovala súčasne adresa viacerých zariadení na jednu zbernicu.

Posledným prvkom je blok pripojený na sériovú zbernicu, ktorý zabezpečuje protokol pre prenos dát.

2.3.3 Zhodnotenie systému

Systém sa navrhoval ako rozšíriteľný a v tomto zmysle je dobré, že časovače obsahujú len tú logiku, ktorá je pre ne potrebná. Adresovanie časovačov je komplikované, pretože sa zohľadňujú aj možné viacnásobné pretečenia časovačov v jenom čase. Toto musí byť súčasťou systému, aby sa predošlo oneskorenému odoslaniu prerušenia hostovskému systému. Celkovo je adresovanie prispôsobené tak, aby sa dalo v budúcnosti rozširovať bez úprav inej logiky, ale len počtu bitov na zbernici a dekodéroch (a kódovačoch).

Citlivou stránkou je zatiaľ prenos pomocou *UART* sériového rozhrania, ktoré bolo určené na otestovanie systému. Do výsledného produktu je potrebné pridať vhodné komunikačné rozhranie, poprípade aj viacero rozhraní, aby sa zlepšila odozva samotnej komunikácie.

2.3.4 Vhodné rozšírenia

Okrem komunikačného rozhrania sú vhodné aj ďalšie rozšírenia:

- Pridanie prioritných časovačov pre kritické procesy - Každý systém má určité procesy, ktoré je potrebné zabezpečiť s vyššou prioritou ako ostatné. Ak tieto kritické procesy zlyhajú, zlyháva aj celý systém aj s ostatnými procesmi. Ak sa pridajú prioritné časovače, pri ich pretečení budú tieto uprednostňované a systému sa zvýši pravdepodobnosť bezporuchového chodu.
- Výstupné signály na prepojenie so vstupmi pre reštart na hostovskom systéme - Toto rozšírenie je z dôvodu reštartu hostovského systému a prípadne aj jeho periférií, ak to bude potrebné. Ak chceme hostovský systém reštartovať, musí byť priamo spojený s reštartovacími vstupmi a nestačí odoslať správu pomocou komunikačného rozhrania.
- Vstup pre indikáciu reštartu hostovského systému - Ešte pred tým, ako sa systém rozhodne reštartovať hostovský systém, počká dopredu určený čas. V priebehu tohto

času by mohol hosťovský systém sám rozpoznať zlyhanie a reštartovať sa. Signál sa potom preniesie na vstup, ktorý indikuje, že už nie je potrebné vyslať reštart. Rieši sa to samostatným vstupom, pretože nemôžeme predpokladať, že komunikačné rozhrania budú v tom stave funkčné.

- Uchovávanie počtu aktívnych časovačov a časovačov s nulovou hodnotou - Slúži na zistenie, či systém nie je potrebné reštartovať. Ak počet aktívnych časovačov bude rovný počtu aktívnych časovačov na nule, systém počká dopredu určený čas a potom sa vygeneruje signál pre reštart. Inými slovami, systém si kontroluje počet svojich aktívnych watchdog časovačov a ak všetky aktívne časovače neboli reštartované do určitého času (majú nulovú hodnotu), musí sa reštartovať hosťovský systém.
- Prijem správ o korektne ukončených procesoch - Tieto správy sú potrebné na zastavenie prideleného časovača a zníženie počtu aktívnych časovačov na nule. Tým sa zabezpečí korektné uvoľnenie časovača a možnosť použiť ho pri inom procese. Ak by sa tak nestalo, zbytočne by sa posielali hosťovskému systému správy o procesoch, ktoré už dávno nie sú spustené a nebolo by možné identifikovať, kedy je potrebné systém reštartovať. Toto rozšírenie musí byť súčasťou robustného systému.
- Ukladanie záznamov do externej pamäte určených na diagnostiku - Každý reštart sa bude indikovať svetelnými diódami, ale na zvýšenie presnosti určovania spôsobených chýb procesov by bolo vhodné všetky udalosti o reštartoch systému a vyslaných správ pre neodpovedajúce procesy zapisovať do externej pamäte (napr. pamäťová karta, flash pamäť, ...).
- Viacbitové komunikačné slová pre povely ako sú *reset* alebo *load* - Ak niektorý z procesov systému začne generovať neidentifikovateľné povely vplyvom zlyhania, mohol by zavolať aj niektoré z funkcií na reštart prideleného časovača. Aby sa tomuto predišlo, parametrom takýchto funkcií by mali byť dopredu určené viacbitové povely (operačné kódy), ktoré sa zriedkavo vyskytnú pri náhodnom generovaní. Týmto sa zvýši bezpečnosť systému.
- Použitie „oknových“ watchdogov s detekciou príliš skorého reštartu časovača - Jedným z ďalších rozšírení je možnosť využiť „oknové“ watchdog časovače. Tieto detegujú aj priskorý príchod reštartu z hosťovského systému. V takomto prípade si každý watchdog musí uchovávať ďalšie dve časové hodnoty, ktoré následne porovnáva s hodnotou času príchodu reštartu. Toto rozšírenie je vhodné pri *hard* systémoch reálneho času.

2.4 Komunikačné rozhrania

Jedna z dôležitých častí systému je komunikácia medzi obvodom s časovačmi a zabezpečeným systémom. Existujúce riešenie uvedené v kapitole 2.3 používa na komunikáciu UART sériové rozhranie. Ako rozhranie vhodné na otestovanie funkčnosti systému je postačujúce, no reálny systém by mal mať komunikáciu riešenú tak, aby mala čo najmenšiu latenciu a aby sa vedela prípadne po prijatí nekorektných dát aj čiastočne zotaviť. Pri systémoch reálneho času alebo externých watchdog časovačoch nie je vhodné uvažovať o bezdrôtovej komunikácii kvôli nežiaducim vplyvom z okolia.

Na komunikáciu sa používajú dva typy zberníc - sériové a paralelné. Paralelné zbernice sú vhodné na prenosy veľkých objemov dát a ich oneskorenie je v podstate menšie vzhľadom na počet bitov. V súčasnej dobe však prevládajú sériové rozhrania s vysokou prenosovou rýchlosťou pri relatívne nízkom oneskorení. Zbernice USB, PCI Express a Ethernet/LAN sú podľa [6] jedny z najpoužívanějších zberníc pri ovládaní prístrojov (*instrument control*). Ak je potrebné vybrať rozhranie pre zabezpečovací systém, ktorý musí detegovať nízku (alebo vysokú) dobu odozvy, vhodnejšie budú sériové prenosy. Objem dát bude maximálne zopár bajtov. Dáta budú buď nastavovať hodnoty do jednotlivých watchdog časovačov alebo ich reštartovať. Najrozšírenejšie paralelné zbernice by boli pri častých posielaniach reštartu (v jednom čase bude prichádzať len jeden reštart) nevyužité. Preto vybrané komunikačné rozhranie bude práve sériové.

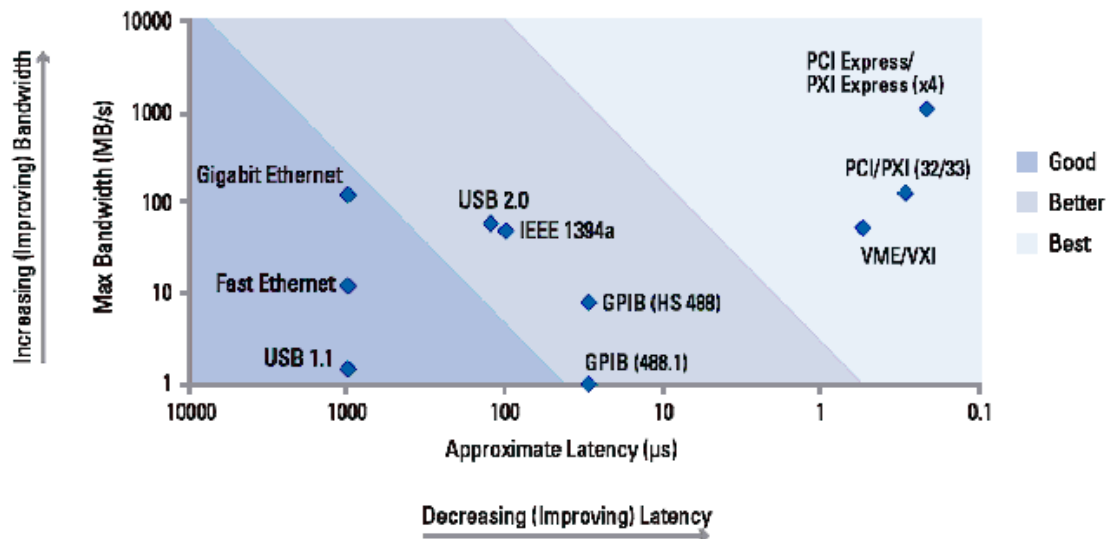
Z pohľadu výberu vhodného rozhrania pre cieľový systém je potrebné zvážiť ešte ďalšie ukazovatele. Jedným z nich je šírka prenosového pásma a oneskorenie. Šírka pásma označuje počet prenesených dát za čas a oneskorenie označuje rýchlosť, za akú sa dáta prenesú po zbernici od vysielača po prijímač. Na obrázku 2.1 je zobrazené porovnanie jednotlivých prístrojových zberníc.

V nasledujúcich podkapitolách sú vybrané dve zbernice - USB a PCI Express, pretože sa jedná o najrozšírenejšie komunikačné vonkajšie zbernice. Prvá časť analyzuje rozhranie UART, ktoré je zatiaľ súčasťou existujúceho systému a môže slúžiť ako záložná komunikačná cesta.

2.4.1 UART

Univerzálny asynchrónny vysielač/prijímač spolu so štandardom RS-232 je komunikačným rozhraním, ktoré používa existujúci systém viacerých časovačov. Komunikáciu bude potrebné najprv otestovať, na čo poslúži práve toto rozhranie z dôvodu jeho jednoduchej obsluhy a implementácie priamo do programovateľného obvodu. Nevýhodou tohto rozhrania je jeho

vysoké oneskorenie, pomalá rýchlosť prenosu nie je až tak dôležitá, pretože systém nebude posielat vysoký objem dát.



Obr. 2.1: Šírka pásma v závislosti od oneskorenia pre prístrojové zbernice. [6]

2.4.2 USB

Univerzálna sériová zbernica (USB) je jednou z najrozšírenejších zberníc na pripájanie periférnych zariadení k počítačovým systémom. Medzi základné vlastnosti USB patria podľa [7] najmä jednoduchosť, flexibilitu, robustnosť, jednoduchý protokol a široké spektrum aplikácií. Viac informácií o tejto zbernici je možné nájsť aj na webovej stránke USB fóra¹

Čo sa týka implementácie, existuje niekoľko implementácií protokolu USB do FPGA dostupných medzi projektami *OpenCores*². Väčšina vývojových zariadení pracujúcich s USB má ale vyhradený samostatný externý obvod, ktorý obsahuje všetku logiku USB hosta aj funkcie. Z pohľadu šetrenia miesta na obvode a spotreby energie je použitie externého obvodu výhodnejšie. USB zbernica je vhodná na použitie aj z pohľadu univerzálnosti, pretože v dnešnej dobe už takmer každý počítačový systém je štandardne vybavený USB zbernicou.

2.4.3 PCI Express

Zbernica PCI Express je nástupcom technológie PCI (*Peripheral Component Interconnect*). Jedná sa síce aj sčasti o spätne kompatibilnú zbernicu, ale s úplne inou architektúrou. Ar-

¹USB fórum. Online dostupné na <http://www.usb.org/developers/docs/> (2011-04-26)

²Projekt OpenCores. Online dostupné na <http://opencores.org/> (2011-04-26)

chitektúra PCI Express vychádza z vrstvovej architektúry počítačových sietí. Každé jedno zariadenie (*Endpoint*) používa samostatnú linku. Pri PCI boli komunikačné „linky“ zdieľané. Dôležitou súčasťou tejto zbernice je paketový prenos buď vstupno-výstupných operácií alebo pamäťových operácií. Na zabezpečenie sa používa cyklický redundantný kód na linkovej vrstve. Táto vrstva deteguje chyby a prípadne si vyžiada opakované poslanie chybného paketu dát. Niečo viac o PCI Express zberniciach je opísané v článku [8].

Implementácia PCI Express protokolu je náročná, ale vývojové dosky, na ktoré sa umiestňujú tieto rozhrania (ako Virtex od firmy Xilinx), sú prispôsobené a majú už niektoré vrstvy implementované. Súčasťou vývojového prostredia od Xilinx je aj generovanie hard jadier niektorých blokov. Medzi ne patria aj PCI Express bloky koncových zariadení. Výhodou je abstrakcia na vyššiu úroveň, čo zabezpečí, že programátor sa nemusí zaoberať implementáciou protokolu, ktorý sa už v súčasnosti využíva a môže sa sústrediť na implementáciu koncovej aplikácie.

2.5 Výber zariadenia vhodného pre implementáciu

K dispozícii na implementáciu boli zariadenia uvedené nižšie. Každé zo zariadení má iný programovateľný obvod a iné periférne zariadenia.

1. *CoolRunner-II CPLD Starter Kit – Featuring the DataGATE Low-Power Advantages* s rozšíreniami *CoolRunner-II Starter Kit Peripheral Module Bundle*
2. *ALTERA DE2-70 Development and Education Board* s rozšíreniami *4.3" LCD Touch Panel Package* a *5 Mega Pixel Digital Camera Package*
3. *Avnet Spartan-3A Evaluation Kit*
4. *Embedded Development HW/SW Kit – Virtex-5 FX70T PowerPC & MicroBlaze Processor Edition*

Základom prvého zariadenia je obvod CoolRunner-II 256 TQ144. Je to menšia vývojová doska obsahujúca aj periférie ako Mini-USB, 4 LED, štvoritý 7-segmentový displej, 4 porty určené pre rozšírenia a iné. Dostupnými rozšíreniami sú analógovo-digitálny a digitálno-analógový prevodník, moduly pre prácu so servomotormi, sériový flash modul, RS-232 sériový port.

Druhé zariadenie od firmy ALTERA má obvod Cyclone II. Vývojová doska je väčšieho rozmeru a obsahuje periférie ako USB 2.0 typ A a B, vstup a výstup pre audio, vstup a výstup pre video, sériové rozhrania RS-232 a infračervený port, PS/2 port pre klávesnicu

alebo myš, 10/100 Ethernet a dva 40-pinové porty pre rozšírenia. Okrem iného obsahuje aj slot pre SD pamäťovú kartu, osem 7-segmentových displejov a veľký 16 x 2 LCD displej. Medzi rozšíreniami je digitálna kamera a 4,3" dotykový displej.

Tretie zariadenie je menšie a obsahuje obvod Spartan-3A. Obsahuje USB-UART most, 4 LED, I2C teplotný senzor a I2C port.

Posledné zariadenie je vývojová doska s obvodom Virtex-5. Obsahuje množstvo periférií, medzi ktorými sú dôležité najmä USB host aj periféria, 10/100/1000 Ethernet, PS/2, SATA, PCIe, AC97 audio, vstup a výstup pre video, RS-232 sériový port, 16 x 32 znakový LCD displej a ďalšie.

Ako zariadenie vhodné pre implementáciu bolo vybrané tretie zariadenie *Avnet Spartan-3A Evaluation Kit*, ktoré obsahuje dostatočne veľký obvod. Pri výbere bolo zvažované aj posledné zariadenie s Virtex-5 obvodom, ktoré obsahuje rýchle zbernice, ale aj veľa zbytočných nevyužitých periférií. Dôležitým faktom je, že posledné zariadenie je prioritne určené na vývoj vnorených systémov s pripravenými jadrami pre rôzne konfigurácie. Použitie tohto zariadenia by mohlo celkový čas na implementáciu predĺžiť.

2.5.1 Virtex-5 FX70T

Vývojová doska Virtex-5 FX70T obsahuje obvod Virtex-5, ktorý je mimoriadne veľký aj na niekoľko stoviek watchdog časovačov.

FPGA obvod na vývojovej doske je z rodiny Virtex-5, konkrétne sa jedná o označenie *XC5VFX70T*. Zhrnutie najdôležitejších vlastností (podľa [11]):

- Platforma FXT pre vysoko výkonné vnorené systémy s pokročilou sériovou komunikáciou.
- FPGA štruktúra s reálnymi 6-vstupovými pravdivostnými tabuľkami (LUT), možnosť 64-bitovej distribuovanej RAM pamäte
- Riadenie hodín - bloky pre nulové oneskorenia pri vyrovnávacích obvodoch, frekvenčná syntéza a hodinový fázový posun.
- Konfigurácia pomocou SPI, rozhrania paralelnej flash pamäte, prístup cez JTAG
- Integrovaný blok koncového zariadenia PCI Express kompatibilný so špecifikáciou 1.1
- RocketIO vysielač s prijímačom pre vysoký prenos dát použitý aj pri PCIe rozhraní od 150 Mb/s do 6,5 Gb/s - integrovaný na čipe
- Obvod je kompatibilný a je možné na ňom testovať RISC architektúru PowerPC mikroprocesora
- 65 nm CMOS technológia

- 1 V napätie jadra
- Monitorovanie stavu systému ako teplota, napájanie a aj používateľské analógové vstupy

Vývojová doska obsahuje zo spodnej časti slot na zapojenie externej DDR2-SODIMM pamäte. Dodávaná je 256 MB SODIMM, je možné použiť aj väčšiu veľkosť. Testovaná je pri frekvencii 400 MHz.

Jeden oscilátor o frekvencii 100 MHz je používaný s napájaním 3,3 V. Dostupný je aj programovateľný generátor hodín pre rôzne periférie a FPGA.

Doska obsahuje 15 luminiscenčných diód, z nich je 8 zelených usporiadaných pre všeobecné použitie, 5 zelených usporiadaných vedľa tlačidiel a 2 červené na signalizáciu chýb, ale môžu sa prekonfigurovať na iné použitie.

Na vrchnej časti vedľa pripojenia PS/2 je umiestnený jeden RS-232 konektor ako host (DCE) určený na komunikáciu FPGA s okolitými zariadeniami. Na prepojenie ako periférie je potrebné použiť null modem. Maximálna modulačná rýchlosť je 115 200 Bd. Nie je použité riadenie toku, FPGA je prepojené len na RX a TX piny.

LCD Displej je dvojriadkový a 16-znakový. Je možné meniť jeho kontrast pomocou potenciometra. Dátové rozhranie LCD je prepojené s FPGA a používa sa len 4-bitový mód.

Doska používa CompactFlash typu I a je možné pomocou tohto rozhrania konfigurovať FPGA obvod cez JTAG port. Na pamäťovej karte je možné uložiť až 8 konfiguračných obrazov. Samozrejme tento port obsahuje aj prepojenie priamo do FPGA, aby bolo možné pamäťovú kartu obsluhovať ako klasickú externú pamäť. Dátová zbernica je zdieľaná s USB kontrolórom.

Zabezpečenie hosta a periférie USB poskytuje samostatný externý USB obvod s dvomi sériovými rozhraniami a interným mikroprocesorom. Jedno je pripojené na hostovský konektor typu A a druhý na typ B.

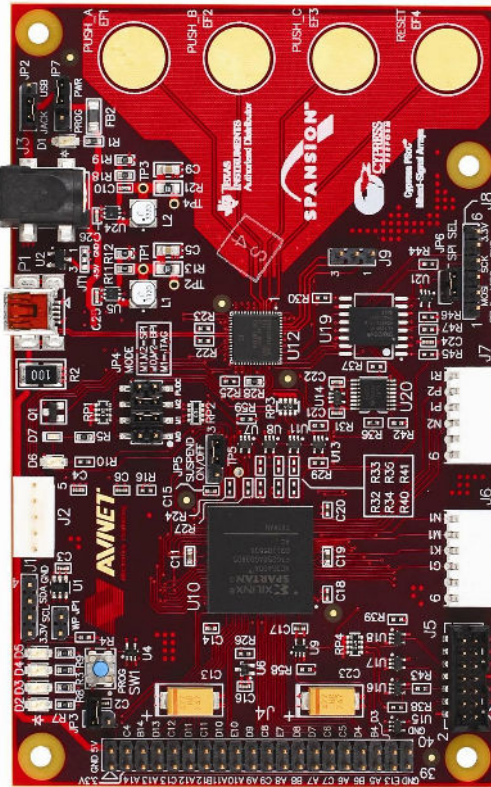
Na spodnej časti je konektor pre jedno-linkové spojenie (1x) cez RocketIO prijímače s vysielacími pre Virtex-5 FPGA integrovaný blok koncového zariadenia PCI Express.

2.5.2 Avnet Spartan-3A Evaluation Kit

Na obrázku 2.2 je fotografia vývojovej dosky vhodnej pre použitie na zabezpečovací systém. Bol vybraný obvod Spartan-3A, ktorý je dostatočne veľký aj na väčší počet watchdog časovačov. Na implementáciu je vhodné použiť vývojové prostredie s použitím VHDL jazyka Xilinx ISE WebPack, ktoré je plne kompatibilné s týmto zariadením.

V ďalších podkapitolách sú opísané najdôležitejšie časti, ktoré sú potrebné pre implementáciu zabezpečovacieho systému. Celkový opis zariadenia je možné nájsť v [9]. FPGA

obvod na vývojovej doske je z rodiny Spartan-3a, konkrétne sa jedná o označenie *XC3S400A-4FTG256C*.



Obr. 2.2: Vývojová doska Avnet Spartan-3A Evaluation Kit. [9]

Zhrnutie najdôležitejších vlastností vývojovej dosky (podľa [9]):

- obvod Xilinx XC3S400A-4FTG256C Spartan-3A FPGA, ktorý je programovateľný cez USB 2.0 port
- 4 LED diódy, ktoré je možné meniť používateľským programom
- 4 tlačidlá konfigurovateľné ako vstup
- I2C teplotný senzor s rozsahom od -55 to $+125^{\circ}\text{C}$ komunikujúci cez I2C zbernicu
- dva 6-pinové vývody pre prípadné rozširujúce moduly
- 20 x 2 používateľský vstupno-výstupný port, všetky porty sú 3,3 V CMOS
- 4 MiB paralelná flash pamäť, priamo prepojená s FPGA obvodom, jedná sa o jeden pamäťový odul 32 Mbit S29GL032N
- 128 Mb SPI flash pamäť určená na konfiguráciu FPGA

- Most USB-UART určený na programovanie a na komunikáciu
- Rozhranie Xilinx JTAG určené na programovanie a ladenie
- 16 MHz oscilátor ako hlavný zdroj hodinového signálu

2.6 Výber komunikačného modulu

Pri použití vývojovej dosky s obvodom Spartan-3A opísanej v kapitole 2.5.2 je potrebné na komunikáciu cez USB vybrať vhodný modul. Použitie modulu so vstavanou logikou celého rozhrania zbernice USB je jednoduchšie ako implementácia tohto rozhrania do FPGA obvodu.

Na komunikáciu pri tejto vývojovej doske pripadá do úvahy sériové rozhranie RS-232 a USB. Signály RS-232 je možné vyviesť z obvodu na všeobecné vstupno-výstupné porty alebo prepojiť ich priamo na UART-USB most, ktorý je súčasťou vývojovej dosky. Externý modul je potrebný ak chceme komunikovať cez USB iným spôsobom ako cez rozhranie UART.

Ako vhodný USB modul bol vybraný *FTDI FT2232H mini module*, ktorý je zobrazený na obr.2.3.



Obr. 2.3: FTDI FT2232H mini module [12]

Tento USB modul je vysokorýchlostný, podporuje USB 2.0 Hi-Speed mód, je priamo napájaný z USB portu a všetku komunikáciu zabezpečuje obvod *FT2232H*. Podporovaný je asynchrónny sériový prenos od 300 *baud* do 12 *Mbaud* pri úrovniach TTL alebo synchrónny prenos do 30 *Mbps* pri JTAG, SPI a I2C. Obvod *FT2232H* obsahuje dve komunikačné rozhrania plne konfigurovateľné na synchrónny alebo asynchrónny prenos, UART, JTAG, SPI, I2C, Bit-bang alebo FIFO typu FT245.

3 Návrh

3.1 Požiadavky na systém

Minimálne tri požiadavky sa kladú na zabezpečovací systém. Prvým je dodržanie doby odozvy samotného zabezpečovacieho systému, druhým je rýchla komunikácia najmä z pohľadu latencie a s použitím kontrolných kódov a tretia je presné časovanie jednotlivých časovačov s minimálnymi odchýlkami.

Medzi používateľské požiadavky patrí veľkosť a prenosnosť zariadenia, použitie štandardných komunikačných rozhraní, ktoré sú bežne dostupné, jednoduché zapojenie a indikácia korektného alebo chybného správania zariadenia.

Požiadavky pre vývojárov softvérových ovládačov sú najmä jednoduchosť komunikačného protokolu a jednoznačná a prehľadná dokumentácia.

Systém musí byť schopný prijímať a odosielať súbežne správy prostredníctvom komunikačného rozhrania a vyhodnocovať ich. Každý z watchdog časovačov bude pri zapnutí systému vo vypnutom stave. Pri pretečení niektorého z watchdog časovačov musí byť zabezpečené vypnutie signálu pretečenia, aby bolo možné rozpoznať ďalšie pretečenia iných watchdog časovačov. Pri pretečení všetkých watchdog časovačov musí systém odoslať varovanie zabezpečovanému systému a o nastavenú hodnotu oneskorenia ho aj hardvérovo reštartovať.

3.1.1 Výber vhodného komunikačného rozhrania

Na základe analyzovaných zberníc v kapitole 2.4 možno uvažovať o použití USB a PCIe rozhraní. Zabezpečovací systém musí využívať komunikáciu tak, aby nedochádzalo k vysokým oneskoreniam, aby bolo možné chybné prijaté dáta opraviť a poprípadе vyžiadať ich opätovné vyslanie.

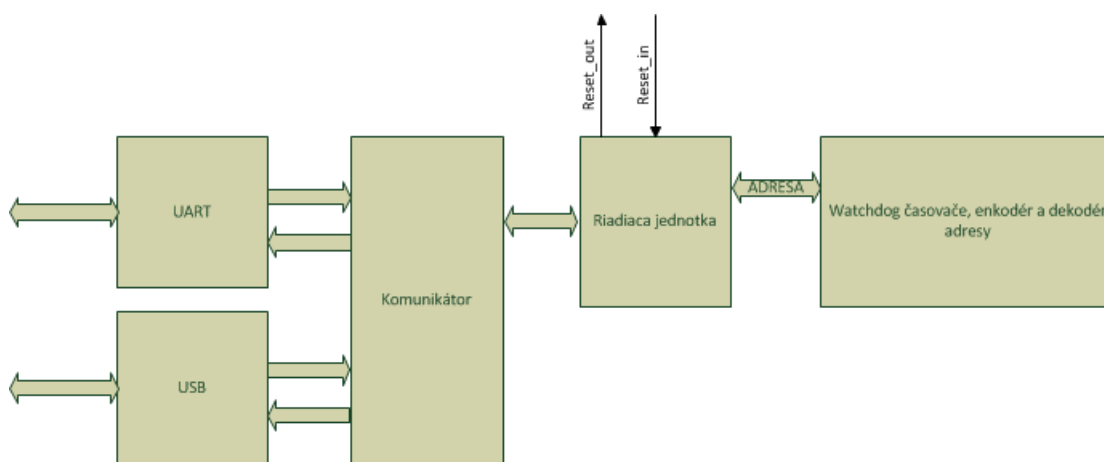
V tejto práci sa bude vychádzať zo systému viacerých časovačov opísaného v kapitole 2.3. Na základe toho, že systém ešte nebol implementovaný v reálnom zariadení, bude sa používať aj sériové rozhranie UART. Toto rozhranie sa môže využívať ako záloha v prípade,

že iné rozhranie nebude funkčné alebo hosťovský systém nebude podporovať iné rozhranie.

Na komunikáciu sa po otestovaní s rozhraním UART použije zbernica USB, nakoľko spĺňa požiadavky na rýchlu komunikáciu s nízkou latenciou. Možným rozšírením systému je použitie zbernice PCI Express, ktorá má zložitejšiu architektúru a najnižšie hodnoty latencie. V tejto práci sa nebude používať zbernica PCI Express.

3.2 Architektúra systému

Architektúra systému vychádza z požiadaviek kladených v kapitole 2.3.4. Bloková schéma je na obrázku 3.1.



Obr. 3.1: Bloková schéma hrubého návrhu

Architektúra je navrhnutá tak, aby ju bolo možné experimentálne overiť a implementovať do programovateľného obvodu. V budúcnosti ju bude možné rozšíriť. Architektúra pozostáva z jedného bloku so sériovým rozhraním UART, komunikátor pripravený na použitie s rôznymi komunikačnými rozhraniami, riadiacu jednotku, adresový dekodér, prioritný kódovač/dekodér a 16 24-bitových watchdog časovačov so zapínačmi (*enabler*). Adresový systém je 8-bitový pre možné rozšírenie.

Komunikátor označuje blok, ktorý podľa dostupnosti použije komunikačnú zbernica s lepšou charakteristikou (USB). Predpokladá sa použitie aj UART rozhrania na komunikáciu so systémami, ktoré nemajú možnosť používať modernejšie zbernice.

Watchdog časovače s riadiacou jednotkou majú usporiadanie podľa pôvodnej architektúry, no riadiaca jednotka neobsahuje žiadne bloky pre komunikáciu, ale iba výstupy neskôr spracované v bloku **komunikátor**, ktorý zabezpečí ich prenos.

Riadiaca jednotka obsahuje najmä informácie o počte používaných časovačov a o počte

časovačov, ktorých hodnoty sú na nule a sú aktívne. Okrem týchto informácií je potrebné mať konfigurovateľnú hodnotu oneskorenia, ktoré sa uplatní pri čakaní pred reštartom hosťovského systému. Ako posledná informácia je hodnota adresy posledného časovača, ktorý patrí medzi prioritné časovače. Predpokladá sa použitie prioritných časovačov od začiatkovej nulovej adresy. Prioritné časovače bude možné kedykoľvek doplniť, nebudú súčasťou implementovaného prototypu, ale len súčasťou komunikačného protokolu. Riadiaca jednotka ako jediná vie, ktorý časovač kedy dosiahol nulu. Mapovanie adresy časovača s číslom spusteného procesu je ale záležitosťou ovládača na hosťovskom systéme.

3.2.1 Watchdog časovač

Samotný časovač je riešený ako jednoduché zariadenie, no je potrebné ho navrhnuť tak, aby spĺňal požiadavky riadiacej jednotky. Tá musí vedieť časovače zapínať prostredníctvom zapínačov, vložiť im počiatočnú hodnotu, reštartovať ich, pozastaviť ich počítanie alebo vypnúť signál pretečenia a uvoľniť ich (vypnúť).

Stavový automat watchdog časovača (obr. 3.2) je obohatený o stav *Pause*, ktorý časovač pozastaví. Pri vypnutí časovača sa tak nespotrebuje veľa energie. Na druhú stranu ak časovač pretečie a odošle sa správa hosťovskému systému, je potrebné časovaču vypnúť signál *overflow*, aby neblokoval ďalšie možné pretečené časovače. V poslednom rade aby bolo možné zistiť, či je časovač pred reštartom v stave počítania alebo v stave pretečenia, časovač má pridaný 1-bitový port pre indikáciu takéhoto stavu. To umožní riadiacej jednotke prostredníctvom zapínača korektne zistiť, či reštartuje pretečený alebo nepretečený časovač.

3.2.2 Zapínač

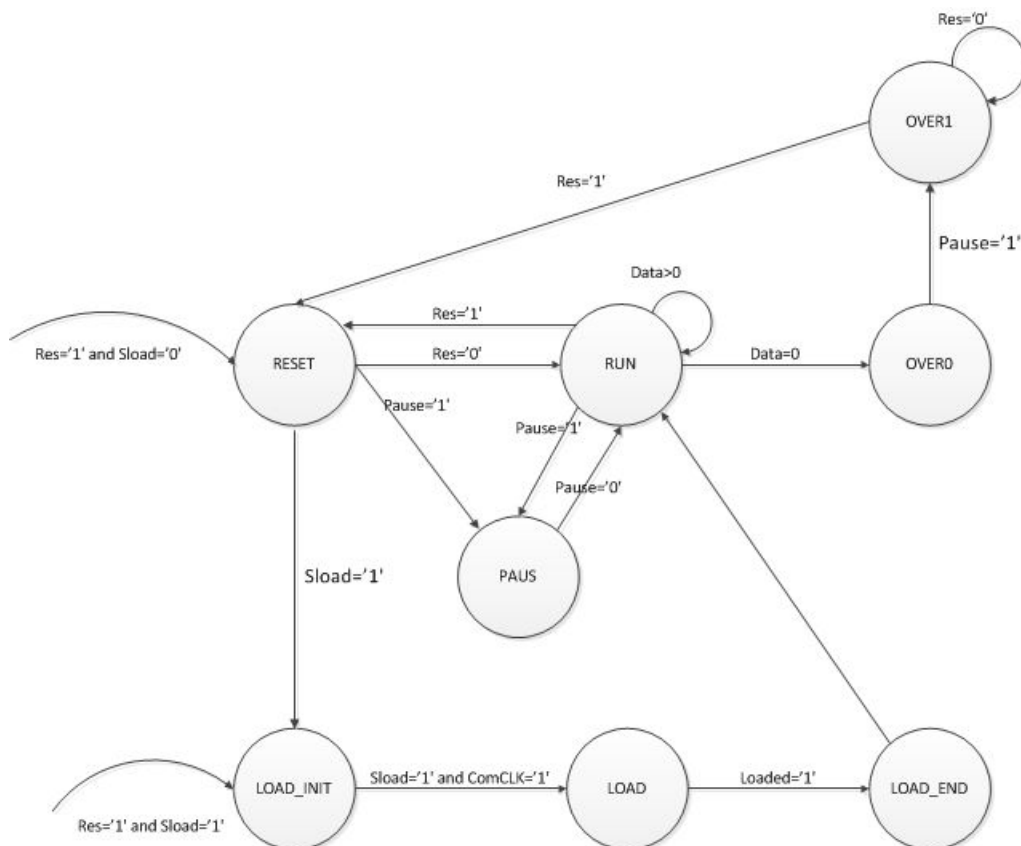
Zapínač je rozšírené rozhranie pre každý watchdog časovač. Má len jednoduchú logiku, podľa vybranej adresy sprístupní riadiacej jednotke vybraný watchdog časovač a umožní meniť jeho stav. Je možné z neho prečítať aj hodnotu o stave pretečenia časovača. Obsahuje jeden register, ktorý uchováva 1-bitovú hodnotu o stave, či je príslušný watchdog časovač aktívny alebo neaktívny.

Zapínač by mohol byť súčasťou watchdog časovača, no ak je oddelený, architektúra je z pohľadu funkčnosti prehľadnejšia.

3.2.3 Prioritný kódovač/dekodér

Časovače s vyššou prioritou majú nižšiu adresu. Pri pretečení niektorého z časovačov je potrebné niektoré správy o pretečení uprednostniť. Tento blok zabezpečí korektné dekodovanie

na adresovú zbernicu. Takisto spätne zakóduje informáciu o pretečení pre hierarchicky nižšie bloky s časovačmi. Tieto bloky sa predpokladajú pri rozšírení systému.



Obr. 3.2: Zjednodušený diagram konečného stavového automatu watchdog časovača

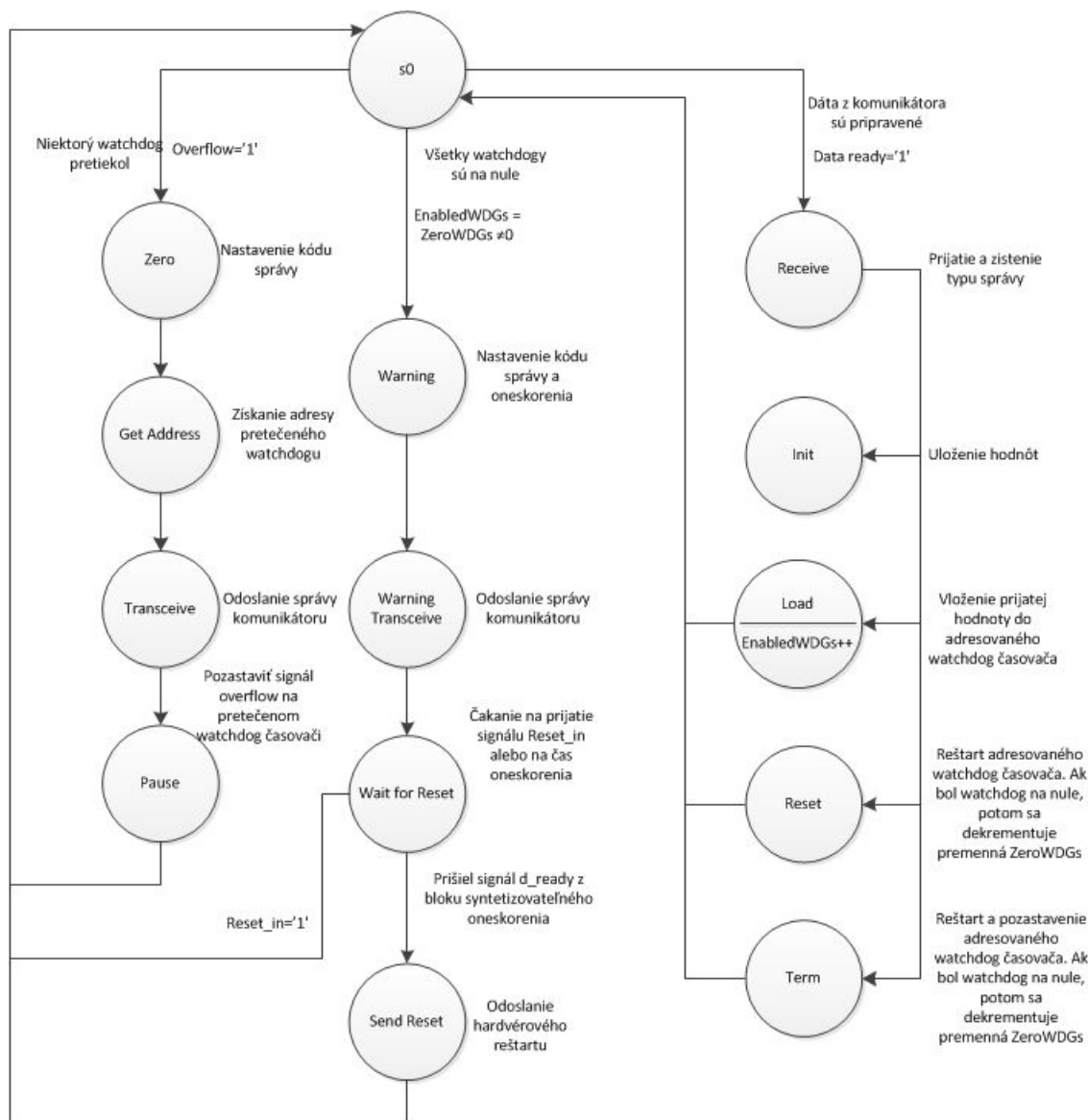
3.2.4 Riadiaca jednotka

Riadiaca jednotka prijíma a odosiela správy komunikátoru. Vykonáva prijaté správy, zapína a ukončuje počítanie časovačov, rozhoduje, či je potrebné hosťovský systém reštartovať, prijíma signály pretečenia od časovačov. Riadiaca jednotka je teda centrálnou jednotkou vykonávajúcou všetky úlohy a je navrhnutá ako rozsiahly stavový automat.

Medzi dôležité vnútorné premenné riadiacej jednotky patria počítadlá zapnutých a pretečených časovačov pri zapnutí nastavené na nulu. Ak sú tieto dve hodnoty ekvivalentné a nie sú rovné nule, je potrebný reštart systému o konfigurovateľnú hodnotu oneskorenia. Zjednodušený diagram konečného stavového automatu je na obr. 3.3.

3.2.5 Komunikátor

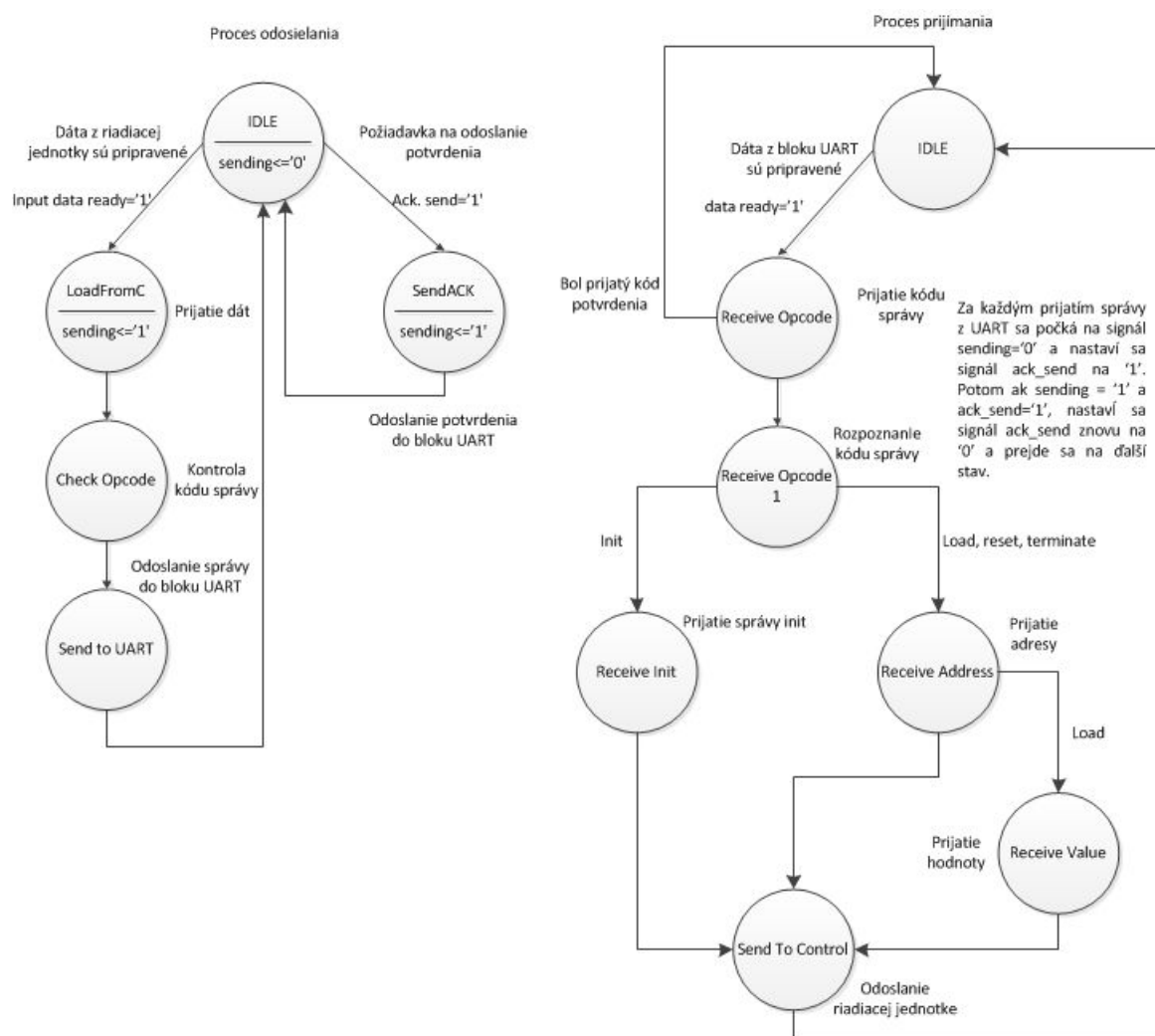
Komunikátor obsluhuje komunikačné rozhrania. musí byť schopný súčasne vyslať aj prijať správy. Ale kvôli obmedzenej veľkosti obvodu sa každé prijaté slovo (8-bitov) potvrdzuje hosťovskému systému, aby nedošlo k strate dát. Pri odosielaných dátach sa potvrdzovanie nevyžaduje, pretože sa predpokladá, že hosťovský systém je inteligentnejší. Potvrdzovanie správ slúži aj ako indikácia prijatia bajtu patriaceho do komunikačného protokolu.



Obr. 3.3: Zjednodušený diagram konečného stavového automatu riadiacej jednotky

Komunikátor prijíma správy, dekóduje ich a skrátenú formu odošle riadiacej jednotke. To platí aj opačne, a teda prijíma správy od riadiacej jednotky, správu zakóduje a odošle

komunikačnému rozhraniu. Kódovaním sa rozumie 8-bitový kód správy zameniť za 3-bitový, ktorý postačuje na pokrytie správ. Pri komunikácii je ale potrebný dlhý kód, aby neprišlo k prípadnej náhodnej správe, ktorá môže oba systémy poškodiť. Rozpoznávanie komunikačného protokolu je navrhnuté ako konečný stavový automat kvôli úspore miesta v obvode. Zjednodušený diagram konečného stavového automatu je na obr. 3.4.



Obr. 3.4: Zjednodušený diagram konečného stavového automatu komunikátora

3.2.6 Komunikačný protokol

Komunikačný protokol musí byť navrhnutý tak, aby bol zrozumiteľný a jednoznačný. Žiadna zo správ nesmie byť zameniteľná, a preto každá zo správ obsahuje 8-bitový kód správy, ktorý ju jednoznačne identifikuje. V obvode sa tento kód už zmení na 3-bitový z dôvodu úspory vo-

dičov a registrov. Protokol je navrhnutý úsporne, pretože čím dlhšie sú komunikačné správy, tým dlhšie trvá potom ich spracovanie v systéme.

Správy prenášané cez komunikačné rozhrania sú:

Smerom do zabezpečovacieho systému:

- Init - nahranie konfiguračných hodnôt: počet prioritných procesov, oneskorenie pre reštart
- Load - obsahuje identifikátor watchdog časovača a hodnotu pre časovač
- Reset - reštart určeného časovača
- Terminate - ukončenie používania daného časovača

Smerom k hostovskému systému:

- Warning - varovná správa pre hostovský systém, že dôjde k jeho reštartu
- Zero - niektorý z časovačov dosiahol nulu (pretiekol), prenesie sa jeho adresa a následne sa predpokladá, že softvérová časť hostovského systému vygeneruje nemaskovateľné prerušenie alebo inak ošetrí tento stav
- ACK - potvrdenie každého prijatého bajtu, ktorý zapadá do komunikačného protokolu

Formát správ je uvedený v tabuľke 3.1.

Tabuľka 3.1: Formát komunikačných správ

Správa	Kód	Prvých 8 <i>b</i>	Zvyšných 24 <i>b</i>
INIT	'0010 1101'	Počet prioritných časovačov 5 <i>b</i> + Oneskorenie 3 <i>b</i>	
LOAD	'0011 1011'	Adresa	Hodnota
RESET	'0001 1010'	Adresa	
TERMINATE	'1111 1001'	Adresa	
WARNING	'0100 1001'	Oneskorenie spodné 3 <i>b</i>	
ZERO	'0101 0010'	Adresa	
ACK	'0110 1110'		

4 Implementácia

4.1 Použitý jazyk a vývojové prostredia

Na implementáciu je použitý hardvérový opisný jazyk *VHDL*, ktorý spĺňa všetky požiadavky na opis. Žiaľ nie všetky jazykové konštrukcie sú syntetizovateľné, čo v niektorých prípadoch môže viesť ku chybám pri syntéze do programovateľného obvodu. Systém môže byť plne simulovateľný na úrovni opisu, ale pri syntéze už simulácia môže mať iné výsledky. Jazyk *VHDL* umožňuje rôzne úrovne opisu hardvéru, v tomto prípade je použitá úroveň opisu správania sa konečných stavových automatov jednotlivých blokov (výnimku tvoria jednoduché bloky ako dekodér a pod.).

Pri implementácii sa ukázalo vhodne použitie kombinácie viacerých vývojových prostredí. Okrem vývojového prostredia určeného najmä pre simuláciu *ModelSim SE 6.4a* sa osvedčilo využitie aj kompletného dizajnérskeho vývojového prostredia *FPGA Advantage 8.1*. Hierarchicky najvyššia entita (kompletný obvod) sa v tomto prostredí jednoducho vytvorí z už vytvorených opisov jednotlivých komponentov pomocou grafického rozhrania. Posledné vývojové prostredie, ktoré sa použilo vrámci implementácie je *Xilinx ISE WebPack 13.4*, určené prioritne k syntéze do konkrétneho programovateľného obvodu.

4.2 Použité bloky z iných zdrojov

Mnoho moderných hardvérových čipov, alebo systémov na čipe obsahuje plno rôznych jadier aj od iných výrobcov. Preto aj táto práca používa jeden blok od iného autora, ktorý je plne funkčný. Jedná sa o komunikačný blok UART (*RS232*) od autora Akram Mashni. Je dostupný pod licenciou LGPL.¹

Tento blok plne vyhovoval požiadavkám systému, ale bol rozšírený o jeden dôležitý signál - *tx_empty*, na základe ktorého sa komunikátor dozvie, že vyrovnávacia pamäť (*buffer*) odosielania je voľná a je možné odoslať ďalší bajt.

¹RS232. http://opencores.org/project,rs232_interface.

4.3 Princíp implementácie jednotlivých blokov

Väčšina konečných stavových automatov jednotlivých blokov je implementovaných podľa určitej konvencie uvedenej nižšie. Keďže procesy musia byť syntetizovateľné, musia sa zohľadniť aj citlivosti procesov na vstupné signály. Ak je signál čítaný v procese, musí byť v zozname citlivosti, teda medzi zátvorkami za kľúčovým slovom *process*.

```

prechodovy_proces : process (CLK, RST, ...) — proces s citlivosťou signalov
begin
  if RST = '1' then — asynchrone vstupy
    ...
  elsif (CLK'event and CLK = '1') then — synchronne vstupy
    case state is — prechod stavov
      when STATE1 =>
        ...
      when others =>
        ...
    end case;
  end if;
end process;

vystupny_proces : process (s, ...) — proces citlivý na stavy
begin
  case state is — vystupy stavov
    when STATE1 =>
      ...
    when others =>
      ...
  end case;
end process;

```

Pri syntéze rozsiahlych automatov treba dbať nie len na celkovú optimálnosť návrhu. Niekedy je potrebné rozšíriť takýto automat o ďalšie stavy, ktoré vykonávajú jednoduchšie operácie, kvôli zjednodušeniu pri syntéze. Napríklad výstupné operácie stavu preložiť na operáciu vykonávajúcu sa pri prechode. Porušia sa pravidlá navrhnutého *Mooreovho* automatu, ale celkové správanie obvodu sa neporuší, ale zjednoduší. Umožní to nastaviť vnútorné premenné počas aktuálneho stavu a nie až na jeho konci.

4.3.1 Blok syntetizovateľného oneskorenia

Riadiaca jednotka musí byť schopná odoslať signál s oneskorením s presnosťou na jednu sekundu. Na úrovni simulácie je možné použiť v jazyku *VHDL* kľúčové slovo **wait**, za ktoré

je možné uviesť hodnotu oneskorenia v časových jednotkách.

Použité zariadenie na implementáciu systému používa hlavný hodinový signál s frekvenciou 16 MHz. Watchdog časovače majú tento komunikačný hodinový signál určený na nahrávanie hodnoty, no počítanie časovačov je pomalšie a používa sa na to delička hodinového signálu, ktorá je súčasťou každého z watchdog časovačov. Samostatná delička hodinového signálu, ktorej výstupom je hodinový signál o frekvencii 1 kHz, sa využíva pri bloku syntetizovateľného oneskorenia.

Blok oneskorenia používa teda hodinový signál s frekvenciou, ktorej perióda je 1 ms. Vstupom je trojbitový vektor symbolizujúci oneskorenie v sekundách a signál *Reset*. Po reštarte bloku signálom *Reset* začína počítanie od nuly po limit daný vstupným vektorom prevedeným na milisekundy, teda vynásobený konštantou 10^3 . Ak sa dosiahne limit, daný čas ubehol a na výstupnom porte *Ready* sa indikuje koniec počítania.

4.3.2 Pripojenie k hostovskému systému

Pripojenie k systému pomocou rozhraní UART a USB je možné viacerými spôsobmi. Prvý spôsob je priamo prepojiť vývody UART modulu na piny PSoC obvodu zabezpečujúceho aj programovanie FPGA obvodu. Ten totiž má premostenie USB-UART, no používa baudovú rýchlosť minimálne 19 200 Bd.

Druhým spôsobom je pripojenie USB modulu *FTDI FT2232H* opísaného v podkapitole 2.6. Keďže tento modul vie komunikovať aj s rozhraním UART, stačí prepojiť signály *Rx* a *Tx* rozhrania na jeho vstupno-výstupné vývody. USB Modul je od výroby nastavený na komunikáciu s rozhraniami UART, takže nie je potrebná ďalšia konfigurácia. Potrebné je len prepojiť ďalšie vývody tak, aby bol modul napájaný zo zbernice USB. Ak bude totiž napájaná aj vývojová doska pomocou USB, je potrebné zabezpečiť aby mali rovnaké úrovne zeme.

Tretí spôsob je pripojenie USB modulu paralelne k rozhraniu asynchrónne FIFO (FT245). V práci je navrhnuté a implementované rozhranie umožňujúce priamo prepojiť USB modul pomocou dvoch kanálov. V tejto konfigurácii je nutné FTDI obvodu prepísať konfiguráciu v pamäti EEPROM softvérom na to určeným. Dva kanály sa používajú na zabezpečenie obojsmernej komunikácie.

Všetky tri zapojenia boli implementované, výsledný produkt obsahuje ale len prvé a tretie zapojenie s možnosťou výberu jedného z nich, aj počas spusteného systému.

4.4 Overenie implementácie

Na overovaní implementácie prototypu je potrebné začať na úrovni simulácie. Obvod bol zostavený a simulovaný najprv so 4 watchdog časovačmi, neskôr so 16. Boli zostavené niekoľké testovacie entity v návrhovom systéme *FPGA Advantage 8.1*, kde je možné vytvoriť testy pomocou vývojového diagramu (*flow chart*).

Samostatne bol testovaný blok watchdog časovača, neskôr aj s blokom zapínača. Testovacia entita pre celý obvod bola navrhnutá čiastočne pomocou vývojového diagramu a čiastočne pomocou prepojenia s blokom UART, keďže toto rozhranie slúži na odosielanie a prijímanie dát. Na obr. 4.1 je vývojový diagram použitej testovacej entity pre simuláciu pre celý obvod.

4.4.1 Simulácia

Výsledky čiastkových simulácií pomohli k odstráneniu viacerých chýb v návrhu. Pri simulácii watchdog časovačov sa objavili chyby v sériovom nahrávaní hodnôt, ktoré boli odstránené. Na simuláciu jednotlivých blokov aj na celkový systém bolo použité prostredie ModelSim 6.4a.

Pri simulácii celkového obvodu sa ustúpilo od pôvodného návrhu komunikátora s jedným procesom na prijímanie a odosielanie dát. Komunikátor bol prerobený na paralelný príjem a odosielanie dát tak, ako je to v návrhu v podkapitole 3.2.5. Ako súčasť komunikátora sa pridalo potvrdzovanie každého prijatého slova od zabezpečeného systému z dôvodu pamäťového obmedzenia obvodu. Výhodou potvrdzovania správ je aj spätná väzba zo zabezpečovacieho systému.

Simulácia bola vykonaná len na obvode so šestnástimi watchdog časovačmi a rozhraním UART. Túto konfiguráciu bolo potrebné implementovať a sfunkčniť na programovateľnom obvode. Potom po rozšírení architektúry už simulácia nebola potrebná, nakoľko bolo overené, že jednotlivé bloky pracujú korektne.

4.4.2 Syntéza

V návrhovom prostredí *Xilinx ISE WebPack 13.4* bol vytvorený projekt za účelom syntézy obvodu. Cieľový obvod je hradlové pole (FPGA) Xilinx Spartan-3A. Po prvotnom spustení syntézy bolo nutné spraviť niekoľko zmien, ktoré sa týkali VHDL opisu na úrovni správania sa, no aj samotnej architektúry. Zmena architektúry sa dotkla všetkých trojstavových hradíel a signálov vnútri obvodu. FPGA Spartan-3A má totiž trojstavové hradlá len na vstupno-výstupných vývodoch. Výstup zo syntézy obsahoval varovné správy, že trojstavová logika sa

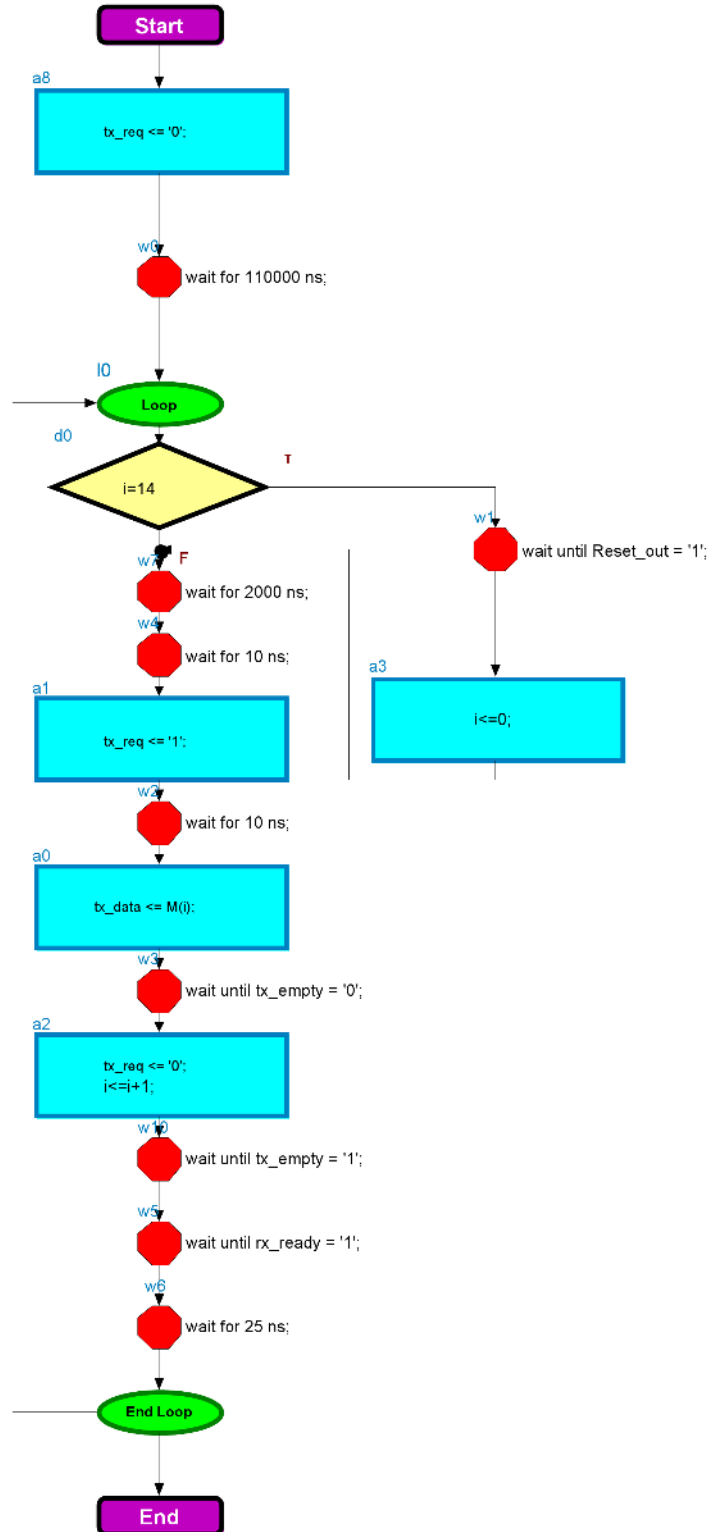
Architecture Declarations

```
type pamat is array (0 to 13) of std_logic_vector(7 downto 0);
```

```
signal M: pamat :=
( "00101101", --init
  "00000001",
  "00111011", --load
  "00000001", --address
  "00000000", -- value
  "00000001",
  "00000000",
  "00111011", --load
  "00000010", --address
  "00011111", -- value
  "00000000",
  "00000000",
  "00011010", -- reset
  "00000001"); --, --address
```

```
signal i: integer := 0;
shared variable ack: bit := '1';
```

```
.....
```



Obr. 4.1: Vývojový diagram testovacej entity pre obvod

nahradiť bežnou logikou a tretí stav bude mať hodnotu logickej jednotky, čo je nežiadúce. Adresová zbernica je teda rozdelená na dve jednosmerné zbernice. Signál *Zero*, ktorý bol pôvodne navrhnutý ako trojstavový, musí byť vedený z každého watchdog časovača (resp. jeho zapínača). Výber medzi signálmi zabezpečí jeden multiplexor riadený pomocou adresy. Ak odstránime tretí stav zo zbernice, musíme definovať hodnotu na zbernici, ktorá bude nahrádzať stav vysokej impedancie, teda nedefinovanú adresu. Adresná zbernica vchádza do dekodéra pre výber watchdog časovača a nemôže byť stále aktívna. V tomto prípade je zvolená najvyššia adresa (0xFF), a teda o jeden watchdog časovač bude menej ako by bolo možné do architektúry umiestniť.

Po týchto a ďalších úpravách zdrojových kódov kvôli syntéze, sa obvod podarilo preložiť až po implementáciu (*Place&Route*) s menším počtom závažnejších *warning* správ. Výsledky syntézy s 15timi watchdog časovačmi sú na obr. 4.2.

Device Utilization Summary				[-]
Logic Utilization	Used	Available	Utilization	Note(s)
Total Number Slice Registers	1,396	7,168	19%	
Number used as Flip Flops	1,200			
Number used as Latches	196			
Number of 4 input LUTs	3,293	7,168	45%	
Number of occupied Slices	2,215	3,584	61%	
Number of Slices containing only related logic	2,215	2,215	100%	
Number of Slices containing unrelated logic	0	2,215	0%	
Total Number of 4 input LUTs	3,372	7,168	47%	
Number used as logic	3,293			
Number used as a route-thru	79			
Number of bonded IOBs	5	195	2%	
IOB Latches	1			
Number of BUFGMUXs	4	24	16%	
Average Fanout of Non-Clock Nets	3.29			

Obr. 4.2: Výsledky syntézy do obvodu Spartan-3A s 15 watchdogmi

Z výsledkov syntézy z obr.4.2 je vidno, že sa využilo len 47% všetkých blokov LUT (*Lookup table*). Obvod je využitý len približne na 50%, a teda v budúcnosti bude možné syntetizovať systém s viacerými časovačmi. Je potrebné ešte dodať, že celkový systém bude komunikovať cez zbernicu USB, čo pridá tiež určitý počet logiky do obvodu.

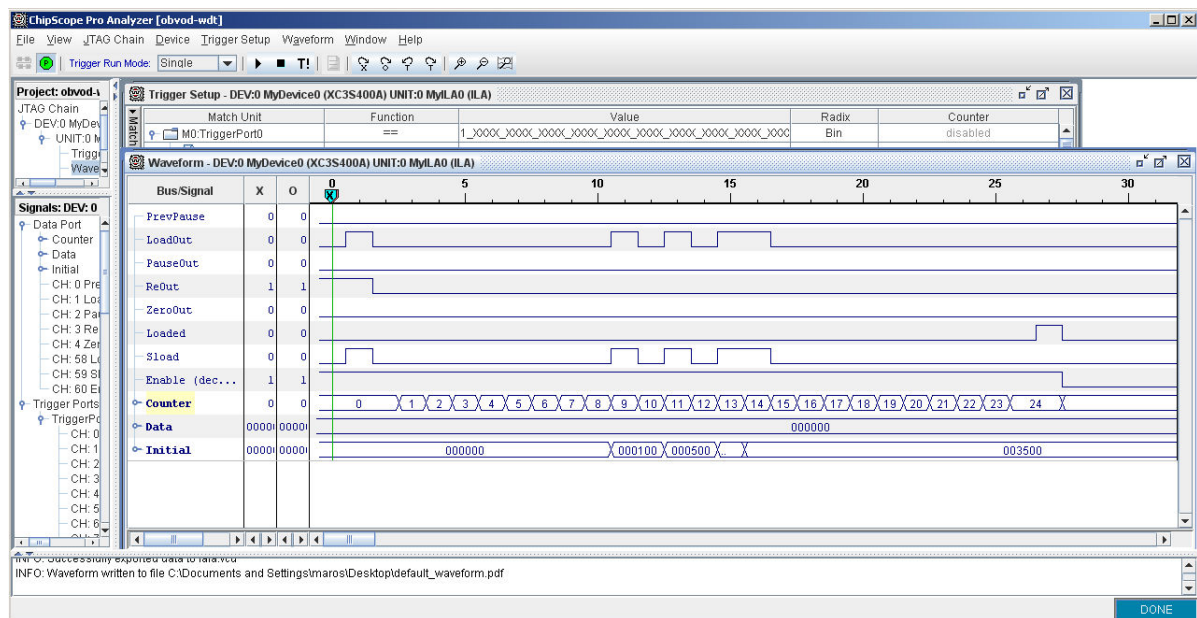
4.4.3 Ladenie v FPGA obvode

Ak sa vyvíja zložitejší logický systém na vyššej úrovni ako je úroveň registrového prenosu, po implementácii do fyzického obvodu sa môžu vyskytnúť chyby. Aj v tomto prípade vzniklo

niekoľko chýb, ktoré sa prejavovali len na fyzickom zariadení.

Po prvom prepojení s hosťovským systémom pomocou USB rozhrania obvod nejavil žiadne známky aktivity, takže bolo nutné použiť ladiace prostredie na odstránenie chýb vznikajúcich vnútri obvodu.

Na ladenie bol používaný program *ChipScope Pro* od firmy Xilinx, na prepojenie sa použil prepojovací JTAG kábel *Xilinx USB Platform Cable II*. Program *ChipScope Pro* je vlastne logický analyzátor. Do vnútra obvodu sa vygeneruje blok so zadanými sledovanými signálmi. Tento blok spolupracuje s analyzátorom *ChiScope Pro* pomocou rozhrania JTAG, a teda je možné sledovať hodnoty signálov vo „vnútri“ obvodu. Aplikácia *ChipScope Pro* nie je súčasťou voľnej licencie a pre tento účel bola použitá skúšobná 30-dňová licencia. Na obrázku 4.3 je obrazovka ladenia obvodu, na ktorom sú zobrazené signály jedného watchdog časovača ešte počas ladenia. Neskôr bol watchdog časovač upravovaný.



Obr. 4.3: Ladenie watchdog časovača pomocou logického analyzátoru ChipScope Pro

4.4.4 Chyby objavené počas ladenia a ich náprava

Ladenie obvodu je potrebné najmä na odstránenie chýb, a preto je potrebné pri ladení zmeniť niektoré časti návrhu, aby bol obvod stabilný a pracoval spoľahlivo.

Jedným z najväčších problémov je to, že jazyk VHDL umožňuje opísať korektne simulovateľný obvod, no pri syntéze sa to prejaví rôznymi varovnými správami, ktoré je potrebné odstrániť. Pri syntéze je nutné sa vyhýbať používaniu záchytných registrov v opise koneč-

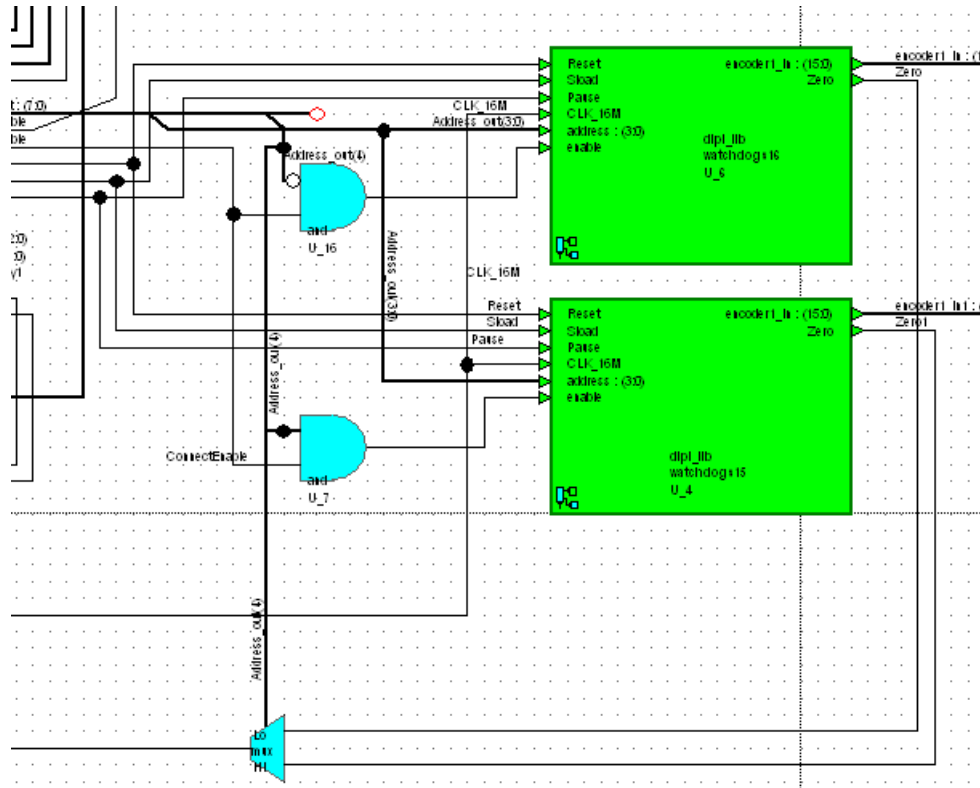
ných stavových automatov. Všetky stavy výstupnej funkcie automatu musia mať **vždy** nastavované **všetky** výstupné porty. V opačnom prípade sa môže stať, že obvod navonok bude pracovať korektne, no pri niektorých vstupných kombináciách sa bude správať neštandardne.

Druhý problém je v hodinovom signále vedenom po celom obvode. V pôvodnom návrhu sa uvažovali dva hodinové signály. Pre zjednodušenie a lepšiu synchronizáciu sa 1 *kHz* hodinový signál pripája len do bloku syntetizovateľného oneskorenia. Watchdog časovače boli upravené tak, aby používali len 16 *MHz* hodinový signál a majú vnútornú premennú, ktorá počíta delený hodinový signál. Je síce pridaný jeden register navyše, no synchronizácia je na vysokej úrovni a po obvode je smerovaný len jeden hodinový signál. To zabezpečí väčšiu stabilitu watchdog časovačov, nakoľko obsahujú len jeden konečný stavový automat a nie pôvodné dva s rôznymi frekvenciami hodinových signálov, ktoré bolo potrebné synchronizovať.

4.5 Implementácia veľkej architektúry

Na zistenie veľkosti obvodu bol urobený experiment, ktorý mal overiť koľko watchdog časovačov sa zmestí do FPGA obvodu Spartan-3A. Príprava veľkej architektúry je riešená tiež v prostredí FPGA Advantage 8.1.

Šestnásť watchdog časovačov spolu s dekodérom bolo umiestnených do jedného vyššieho bloku. Dekodér vyššej úrovne je riešený pomocou dvoch logických členov AND, pretože podľa tabuľky výsledkov syntézy 4.2 vyplýva, že sa do obvodu nezmestí viac ako 32 watchdog časovačov. Podobne hierarchicky zapojené boli aj dekodéry pre signál pretečenia, ale tu bol použitý ten istý blok, nie logické členy. Bolo nutné vyriešiť všetky potrebné prepojenia pomocou multiplexorov vzhľadom na nemožnosť použitia 3-stavových hradiel. Na obrázku 4.4 je znázornenie pripojenia dvoch blokov viacerých časovačov. Druhý blok sa nazýva *watchdogs15*, no v tomto bloku boli počas syntézy postupne odoberané watchdog časovače za účelom nájdenia takého počtu watchdog časovačov, pri ktorom výsledok syntézy nebude obsahovať žiadne chyby alebo varovné správy týkajúce sa využitia všetkých zdrojov obvodu. Totiž pri krajných hodnotách naplnenosti FPGA obvodu výsledok syntézy skončí korektne aj v prípade, že je prekročená hranica vyťaženia obvodu a vo výstupe je len informatívne spomenuté, že je použitých viac ako 100% obvodu. Takýto výsledok je možné implementovať do obvodu, no je veľmi pravdepodobné, že nebude pracovať korektne.



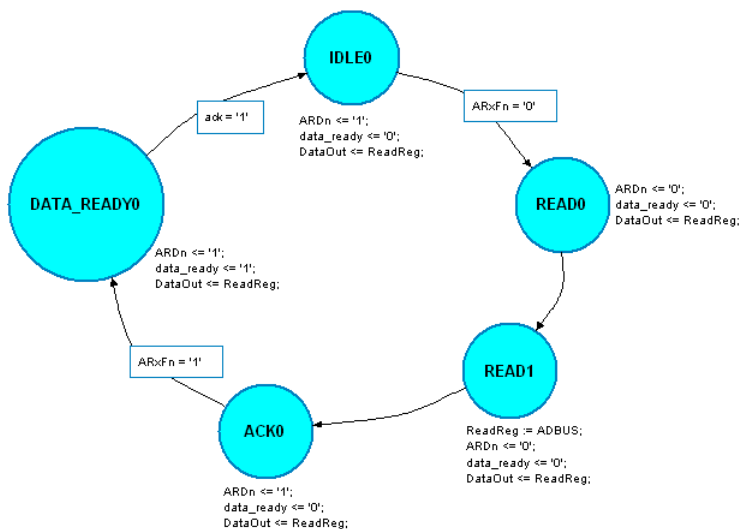
Obr. 4.4: Pripojenie viacerých watchdog časovačov

4.5.1 Komunikačné rozhranie

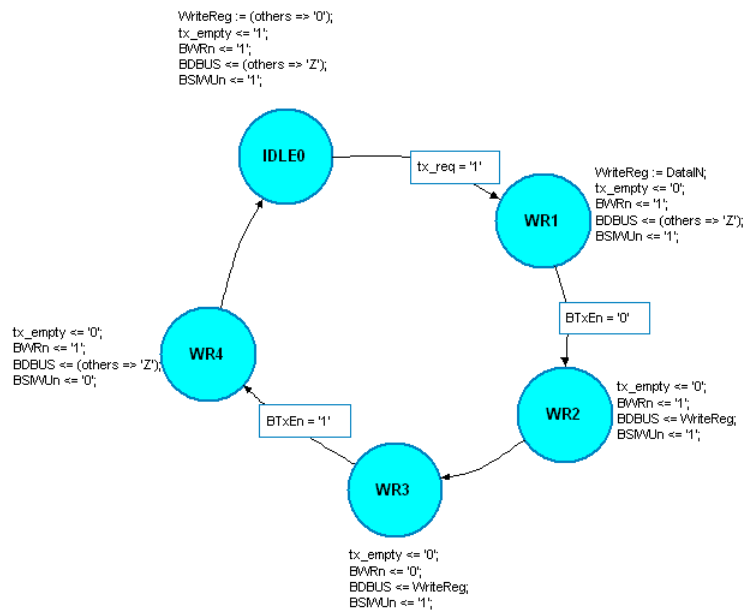
Architektúra používa na základnú komunikáciu sériové rozhranie UART, ale predpokladá sa použitie USB rozhrania PSoC obvodu na vývojovej doske, vzhľadom na to, že nie je nutné pripájať iný modul k vývojovej doske.

Pre priamu komunikáciu pomocou USB bol navrhnutý blok, ktorý komunikuje cez USB paralelným prenosom dát v spôsobe asynchrónneho FIFO prenosu FT245 [13]. Na komunikáciu sú použité dva kanály a každý je použitý jednosmerne z dôvodu možnej kolízie. Tento blok je pripájaný ku komunikátoru, ktorý vie paralelne prijímať aj odosielať správy. Rozhranie je navrhnuté do dvoch samostatných modulov a každý modul je pripojený na jednu vonkajšiu dátovú zbernicu USB kanála. Ku komunikátoru sú tieto bloky pripojené tak, aby bolo možné vybrať komunikáciu medzi UART rozhraním a paralelným FIFO rozhraním pomocou tlačidiel a registra riadiaceho výber pomocou multiplexorov. Rozhrania sú implementované podľa časových priebehov uvedených v dokumentácii obvodu FT2232H [13]. Sú to jednoduché konečné stavové automaty zobrazené na obrázkoch 4.5 a 4.6. Keďže každý z kanálov je obojsmerný, je potrebné ošetriť stavy aj pri prípadnom zápise hodnoty zabezpečovaného systému na kanál, ktorý je určený pre neho na čítanie a opačne. V prípade zapisovacieho

kanála zo strany implementovaného systému je potrebný trojstavový automat zabezpečujúci pri prípadnom zápise na zbernicu aktiváciu signálu pre čítanie. V prípade čítacieho kanála je potrebné nastaviť signál riadiaci zápis na konštantnú hodnotu. Neplnia sa tak vnútorné vyrovnávacie pamäte obvodu FT2232H.

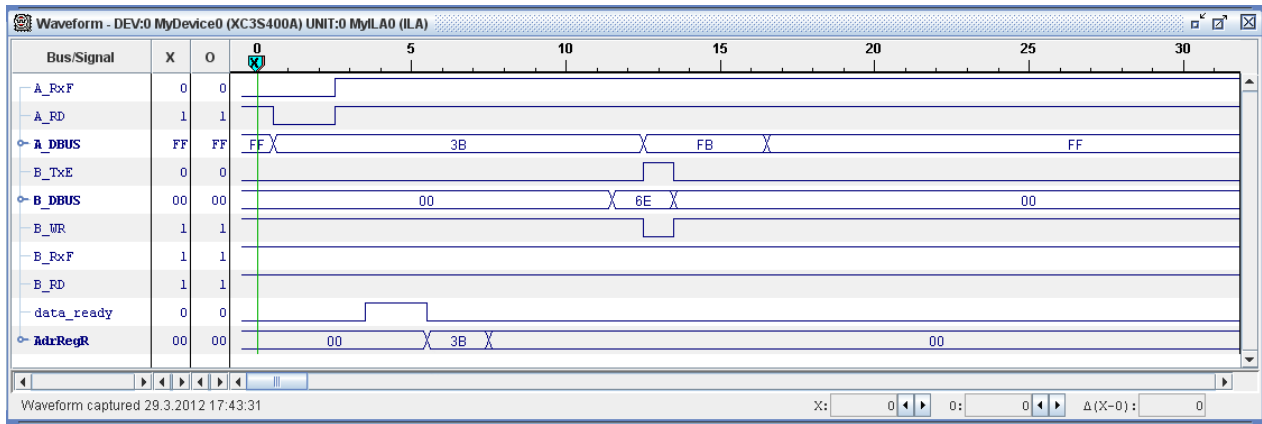


Obr. 4.5: Konečný stavový automat pre paralelné čítanie typu FT245.



Obr. 4.6: Konečný stavový automat pre paralelný zápis typu FT245.

Paralelné komunikačné rozhranie s USB bolo ladené pomocou aplikácie ChipScope Pro (obr.4.7). Po otestovaní tohto bloku bolo nutné pridať medzi riadiace signály ešte jeden signál - *SIWU*, ktorý slúži ako pokyn pre FTDI obvod aby okamžite odoslal dáta hostovi a nečakal, kým sa naplní jeho vyrovnávacia pamäť. Obvod FT2232H a iné obvody obsahujú interný časovač, ktorý čaká na naplnenie vyrovnávacej pamäte a až potom odošle dáta na USB zbernicu. To je ale pri malých dátových prenosoch neželaným javom. Preto sa používa signál *Send Immediately*, teda odoslanie dát ihneď. Zníži to oneskorenie v komunikácii a zabezpečí to, že prijímané a odosielané dáta nebudú skreslené inými dátami z vyrovnávacej pamäte.



Obr. 4.7: Ladenie paralelného komunikačného rozhrania pomocou logického analyzátoru ChipScope Pro

4.5.2 Výsledky syntézy veľkej architektúry

Veľká architektúra obsahuje navyše register pre uchovávanie hodnoty na prepínanie komunikačných rozhraní, komunikačné rozhranie asynchrónne FIFO pre USB, spolu 26 watchdog časovačov (adresy 0x00 - 0x19), spolu s nimi navyše jeden dekodér a dva kódovače pre signál pretečenia a niekoľko multiplexorov. Počas syntézy boli experimentálne manuálne odobierané watchdog časovače tak, aby sa ich do obvodu zmestilo čo najviac. Výsledky syntézy sú na obr. 4.8. Podľa výsledkov je jasné, že maximálny počet watchdog časovačov v danej architektúre je 26, čo je pozitívny výsledok.

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	2,347	7,168	32%	
Number of 4 input LUTs	6,163	7,168	85%	
Number of occupied Slices	3,582	3,584	99%	
Number of Slices containing only related logic	3,582	3,582	100%	
Number of Slices containing unrelated logic	0	3,582	0%	
Total Number of 4 input LUTs	6,622	7,168	92%	
Number used as logic	6,163			
Number used as a route-thru	459			
Number of bonded IOBs	32	195	16%	
Number of BUFGMUXs	1	24	4%	
Number of MULT18X18SIOs	1	20	5%	
Average Fanout of Non-Clock Nets	3.38			

Obr. 4.8: Výsledky syntézy do obvodu Spartan-3A s 26 watchdogmi a dvomi komunikačnými rozhraniami

Na porovnanie bola experimentálne urobená aj syntéza do obvodu Virtex-5 spomínaného v kapitole 2.5.1. Použitých bolo práve 26 watchdog časovačov, ktoré sú limitom pre obvod Spartan-3A. Využitie LUT (lookup-table) je na 44% a na 59% sú okupované jednotlivé časti (*Slices*). Z toho vyplýva, že obvod Virtex-5 je pre túto konkrétnu architektúru po syntéze jeho veľkosťou necelý dvojnásobok obvodu Spartan-3A z pohľadu zdrojov.

5 Testovanie

Testovanie systému prebiehalo v troch fázach. V prvej fáze sa testovalo pripojenie a komunikácia s blokom komunikátora cez komunikačné rozhranie UART. V druhej fáze sa testovala funkčnosť architektúry s 15 watchdog časovačmi a jedným komunikačným rozhraním a nakoniec plná architektúra s 26 watchdog časovačmi a obomi komunikačnými rozhraniami (UART aj USB).

Keďže softvérový ovládač pre hosťovský systém nie je súčasťou tejto práce, supluje ho aplikácia na sériovú komunikáciu. Vybraná bola aplikácia *RealTerm*¹, ktorá umožňuje zadefinovať pokročilé nastavenia pre komunikáciu. Dôležité nastavenie je oneskorenie odosielaných bajtov v komunikácii z dôvodu simulácie prečítania potvrdzovacej správy *ACK* zo zabezpečovacieho systému.

Testovanie komunikátora zahŕňalo najmä otestovanie pripojení komunikačných rozhraní. Komunikátor po každom prijatom bajte spadajúceho do komunikačného protokolu odpovedá správou *ACK*. Týmto spôsobom boli otestované tri rôzne zapojenia komunikačných rozhraní opísané v podkapitole 4.3.2.

5.1 Funkčné testovanie

Na plné otestovanie funkčnosti boli zostavené tri scenáre testovania. Tieto scenáre boli zostavené aby čo najviac zaťažili systém, pretože vtedy vzniká najviac chýb. Na testovanie bola použitá aplikácia *RealTerm* a jednotlivé scenáre boli zapísané v samostatných textových súboroch ako postupnosť bajtov protokolu v hexadecimálnej sústave.

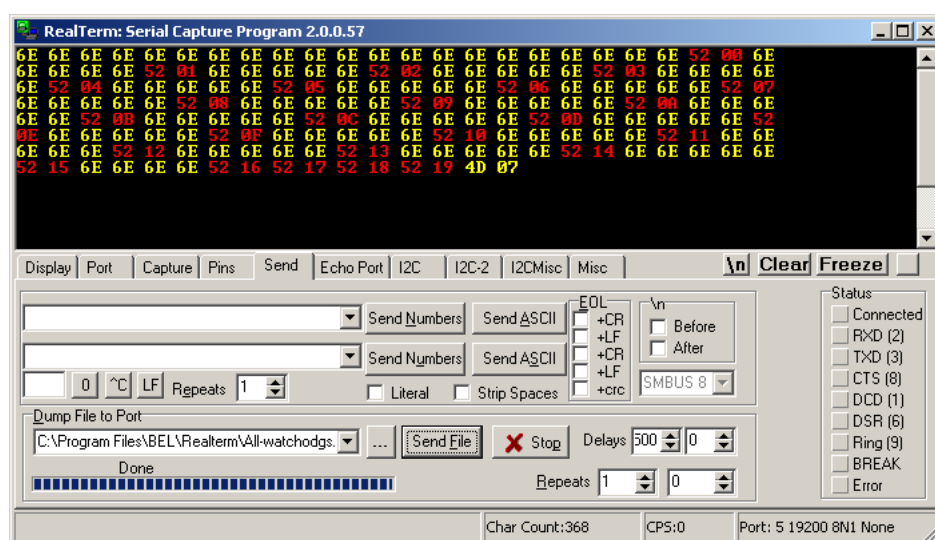
Čas príchodu správ bol meraný bežnými hodinami s presnosťou na 1 sekundu, čo najprv vyhovovalo testu, pretože pri krátkych časových úsekoch sa horšie meria čas. Neskôr, na zvýšenie presnosti merania času, bola použitá aplikácia bežiaca pod linuxovým systémom *jpnevulator*². FTDI modul s USB rozhraním má v linuxovom operačnom systéme vstavané

¹RealTerm. Online dostupná na <http://realterm.sourceforge.net/> (2012-03-08)

²Jackpot Navigator emulator. Online dostupné na <http://jpnevulator.snarl.nl/> (2012-03-08)

ovládače virtuálnych sériových portov. Výstup z tejto aplikácie umožňuje vkladať aj presnejšie merania do výstupov v mikrosekundách.

Prvý scenár testuje nahrávanie nových hodnôt postupne do všetkých watchdog časovačov. Tie by mali po čase pretiecť a odoslať správu o pretečení. Do všetkých watchdog časovačov sa nahrala hodnota reprezentujúca čas 10 sekúnd. Test prebiehal postupne vo všetkých troch zapojeniach komunikačných rozhraní. Výstup z tohto testu je v aplikácii *RealTerm* na obrázku 5.1. Časové oneskorenie medzi bajtami protokolu je nastavené na 500 *ms*. Červenou farbou sú vyznačené prichádzajúce bajty.



Obr. 5.1: Testovanie v aplikácii *RealTerm*

Druhý scenár sa sústredil na testovanie reštartu jedného z watchdog časovačov. Do jedného watchdog časovača sa nahrala hodnota rovná 1 minúte a po 30 sekundách bol watchdog časovač reštartovaný. Pretečenie teda musí prísť až po 1 a pol minúte.

Tretí scenár testoval reštart a uvoľnenie watchdog časovačov. Do troch watchdog časovačov na konci adresného priestoru sa nahráli na začiatku hodnoty rovné 1 minúte. Po určitom časovom oneskorení sa reštartoval prvý watchdog časovač a uvoľnil tretí. V tomto prípade druhý watchdog časovač musí pretiecť najprv, po ňom prvý a tretí ostane v nečinnom stave, pričom sa odošlú správy o hardvérovom reštarte.

Všetky scenáre boli otestované a po odstránení niektorých vzniknutých chýb sa všetky scenáre korektne vykonávajú.

5.1.1 Meranie odozvy času v komunikácii

Výstup z programu *jpnevulator* s rozhraním USB ukázal aj časové údaje prijatia správ. Presnosť sa pohybuje medzi 0 *ms* až 25 *ms*, čo spôsobuje pravdepodobne oneskorenie v ovládačoch na operačnom systéme. Ukážka výstupu programu *jpnevulator* vybraných časových dvojíc, ktoré mali mať podľa testov 10-sekundový rozostup:

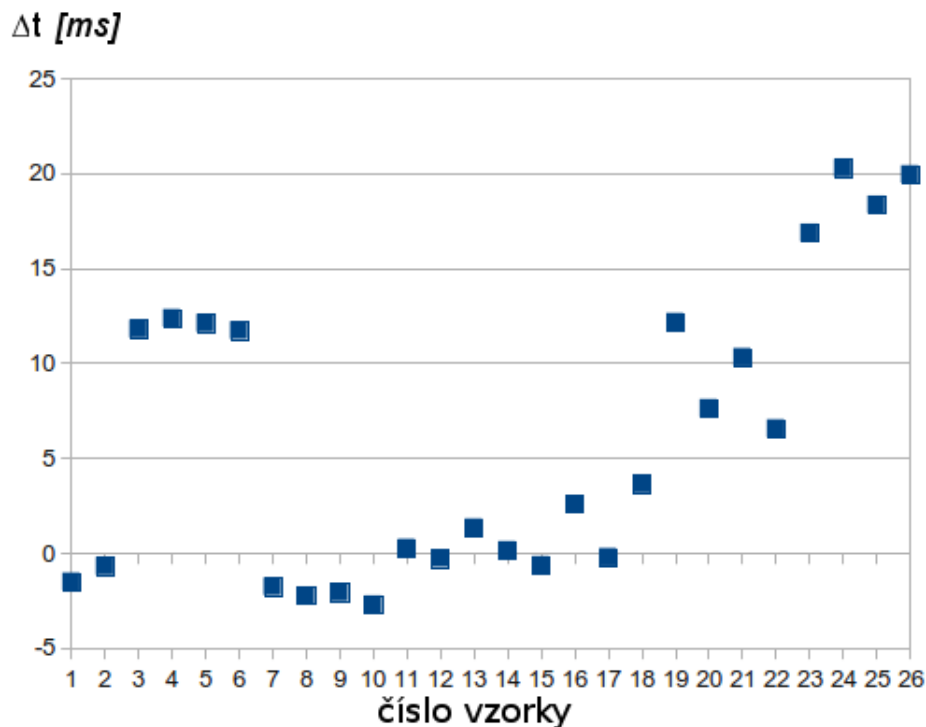
```
...
2012-04-13 21:14:17.751019:
6E n
52 R
0F .
...
2012-04-13 21:14:27.758651:
6E n
52 R
13 .
...
2012-04-13 21:14:30.260335:
6E n
52 R
14 .
...
2012-04-13 21:14:40.278682:
52 R
18 .
...
```

Graf na obrázku 5.2 znázorňuje meranie času odozvy programom *jpnevulator* pri prvom testovacom scenári. Os *y* označuje odozvu milisekundách, os *x* jednotlivé vzorky 26-tich prijatých správ. Záporné hodnoty vyskytujúce sa v grafe v menšom počte znamenajú, že správa prišla o toľko milisekúnd skôr, ako bola očakávaná. Nie je známe, akým spôsobom meria používaný program systémový čas, takže aj tieto hodnoty môžu byť značne skreslené. Počítač, na ktorom boli tieto merania vykonávané je bežným notebookom, ktorý nie je pripravený na vysokú presnosť pri komunikácii.

Podobné meranie boli uskutočnené za rovnakých podmienok na osobnom počítači s operačným systémom Windows XP pomocou programu *COM Port Toolkit*³. Hodnoty odozvy sa

³COM Port Toolkit. Online dostupná 30-dňová verzia na <http://www.compt.ru/> (2012-03-08)

tu pohybovali od 0 *ms* do 17 *ms*, pričom nikdy nebola prijatá záporná odozva. V ovládačoch zariadenia je možné nastaviť oneskorenie časovača na 1 *ms* na FTDI obvode, ktorý čaká na naplnenie vyrovnávacej pamäte. No aj pri takomto nastavení sa meranie nezmenilo a možno usúdiť, že celú komunikáciu spomaľujú ovládače a hosťovský systém, ktorý nie je stavaný na kritické oneskorenie v komunikácii.



Obr. 5.2: Graf časovej odozvy meraný pri prvom scenári testovania

Ak by komunikácia mala oneskorenie medzi navrhnutým systémom a zabezpečeným systémom reálneho času naozaj hodnotu 16 *ms*, nemohli by sa zabezpečiť procesy, ktoré potrebujú kratší čas nastavený vo watchdog časovačoch (pod 16 *ms*).

6 Zhodnotenie

Táto diplomová práca sa zameriava na zabezpečenie požiadaviek na dodržanie doby odozvy systémov reálneho času. V analýze sa nachádzajú časti opisujúce watchdog časovače, ktoré sa používajú práve na dodržiavanie doby odozvy. Keďže už existoval koncept systému s viacerými hardvérovými watchdog časovačmi (princíp je opísaný tiež v analýze), navrhovaný systém, ktorý je výsledkom tejto práce prebral najmä modularitu existujúcej architektúry.

Analyzované boli aj dostupné hardvérové zariadenia, na ktorých by mohol takýto systém byť implementovaný. Rozhodnutie padlo na vývojovú dosku *AvNet Spartan-3A Evaluation kit* najmä z toho dôvodu, že pri zariadení s obvodom Virtex-5 by bolo potrebné používať iné implementačné postupy a nutnosť využívať plnú licenciu vývojového prostredia. Vývojová doska obsahuje aj množstvo nevyužitých periférií a konfigurácia sa nahráva okrem rozhrania JTAG cez pamäťovú kartu. Vývojová doska s týmto obvodom je prioritne určená na vývoj vnorených systémov a uvažovaná bola najmä z dôvodu poskytnutia zbernice PCI Express. Nakoľko navrhované zariadenie by malo byť univerzálne, prišlo sa k záveru, že USB zbernica bude vhodnejšia na komunikáciu. Jediná nevýhoda vo výbere je tá, že obvod Spartan-3A je z pohľadu vnútorných zdrojov menší ako obvod Virtex-5. Je možné do neho umiestniť menší počet watchdog časovačov. Architektúra navrhnutého systému ale umožňuje tento počet kedykoľvek zväčšiť za predpokladu použitia iného programovateľného obvodu.

Architektúra pôvodného konceptu bola vo viacerých bodoch pozmenená, za účelom zlepšenia funkcionality obvodu. Taktiež bol optimalizovaný nízkoúrovňový opis jednotlivých elementov architektúry, s prihliadnutím na obmedzenia pri syntéze FPGA architektúr. Všetky moduly boli nanovo implementované, pridaný bol blok komunikátora, ktorý je pripravený na prácu s viacerými komunikačnými rozhraniami, pretože má 8-bitový paralelný prenos. Jediný modul, ktorý nie je výsledkom tejto práce je modul UART, ktorý je voľne dostupný na stránke *OpenCores*. Bol do neho pridaný len jeden potrebný signál. Po druhom semestri práce sa obvod so šesťnástimi watchdog časovačmi podarilo syntetizovať, ale s veľkým počtom varovaní, ktoré vznikli pri procese *Place&Route*. Počas posledného semestra prebiehala syntéza a implementácia do FPGA obvodu spojená s ladením kvôli vzniknutým chybám. Navrhnutá a implementovaná bola architektúra s viac ako šesťnástimi watchdog

časovačmi spolu s komunikačným rozhraním asynchrónneho paralelného prenosu cez USB modul. V poslednej časti semestra prebiehalo testovanie.

Syntéza a implementácia bola jednou z najviac problematickou časťou, nakoľko jazyk VHDL obsahuje konštrukcie, ktoré syntéza preloží iným spôsobom aj keď sú logicky totožné. Simulácia obvodu bola síce funkčná, no implementácia už v samotnom obvode nepracovala správne. Výsledok testovania spočiatku nebol úspešný a bolo nutné použiť logický analyzátor *Chip Scope Pro* na zistenie, kde sa nachádzajú chyby. Vývojová doska totiž neobsahuje veľa indikačných prvkov, ktoré by sa dali použiť na ladenie.

V konečnom dôsledku bolo nutné zmeniť najmä všetky trojstavové hradlá použité v návrhu za multiplexory, pretože FPGA obvody majú trojstavové hradlá len na vstupno-výstupných portoch. Opisy blokov prešli revíziou a zdrojové kódy museli byť upravené tak, aby nevznikali objavené chyby v logickom analyzátore. Najväčšie chyby boli v oneskorení signálov o hodinový takt alebo nesprávne syntetizované registre.

Po úspešnom otestovaní implementácie takto navrhnutého systému so šestnástimi watchdog časovačmi sa upravila implementácia tak, že sa vložilo do obvodu toľko watchdog časovačov, koľko vyhovovalo syntéze, aby nevznikli chyby z dôvodu využitia viac ako 100% logiky. Na koniec sa implementovalo paralelné rozhranie typu FT245 spolupracujúce s USB modulom *FT2232H mini module* slúžiace pre rýchly USB prenos. Komunikačné rozhrania boli navrhnuté tak, aby sa dali medzi sebou prepínať používateľom pomocou tlačidiel. Vytvorený prototyp je funkčný a pracuje podľa špecifikácie. No na to, aby sa mohol takýto systém nasadiť do prevádzky je nutné znížiť na hosťovskom systéme oneskorenie v komunikácii na minimum. Systém je momentálne schopný komunikácie cez dve komunikačné rozhrania, umožňuje zabezpečiť 26 paralelne spustených procesov vyžadujúcich watchdog časovač. Rozsah hodnôt času jedného watchdog časovača je od $0ms$ do $4\ h\ 39\ min\ 37,215\ s$. Pri pretečení všetkých aktívnych watchdog časovačov je schopný túto situáciu rozpoznať a odoslať po konfigurovateľnom oneskorení hardvérový reštart priamo na vstupy hosťovského systému.

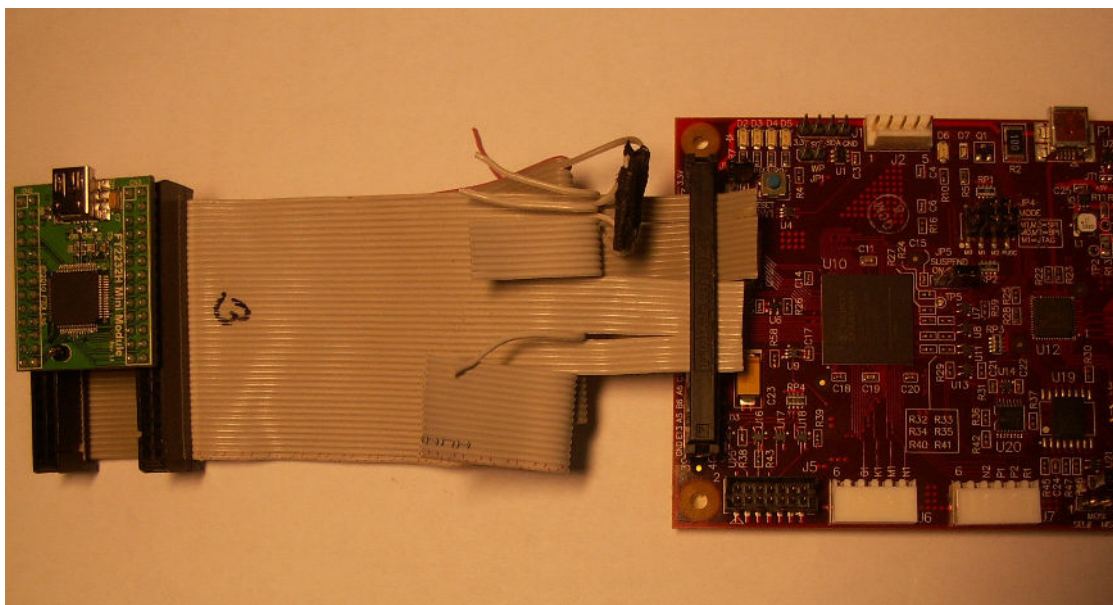
Doplňujúcim rozšírením systému, ktoré by bolo vhodné implementovať, je zápis diagnostických informácií do flash pamäte, aby zodpovedná osoba vedela vyhodnotiť dôvod reštartu niektorého z procesov alebo niektorej z aplikácií na hosťovskom systéme, ku ktorému je tento systém pripojený.

Výstupom práce sú aj dva vedecké články. Článok *A Hardware Reset Implementation in Multiple FPGA Watchdog System* bol publikovaný na medzinárodnej konferencii *Kybernetika a Informatika 2012*. Druhý článok bol odoslaný na medzinárodnú konferenciu *Applied Electronics 2012*.

7 Technická dokumentácia

7.1 Pripojenie FTDI USB modulu

Prepojenie USB modulu s vývojovou doskou je riešené pomocou prepojuvacieho kábla, ktorý má na jednom konci dva konektory určené pre piny CN-2 a CN-3 FTDI USB modulu a druhý koniec má jeden 40-pinový konektor určený pre pripojenie na I/O výstup použitej vývojovej dosky (obr. 7.1).



Obr. 7.1: Prepojenie USB modulu s vývojovou doskou

Pre asynchrónny paralelný prenos dát medzi navrhovaným systémom a FTDI obvodom je vytvorené prepojenie medzi vstupno-výstupnými pinmi a jednotlivé názvy pinov a signálov sú uvedené v tabuľke 7.1.

Tabuľka 7.1: Prepojenie pinov USB modulu a pinov vývojovej dosky

Kanál A			
FT245 Názov	FTDI CN-2 PIN	SPARTAN-3A Názov	SPARTAN-3A I/O PIN
ADBUS 0	7	A14	5
ADBUS 1	10	B14	6
ADBUS 2	9	A13	7
ADBUS 3	12	D13	8
ADBUS 4	14	C13	9
ADBUS 5	13	C12	10
ADBUS 6	16	A12	11
ADBUS 7	15	D11	12
A-RXF	18	B12	13
A-TXE	17	C11	14
A-RD	20	A11	15
A-WR	19	D10	16
A-SIWU	22	A10	17
Kanál B			
FT245 Názov	FTDI CN-3 PIN	SPARTAN-3A Názov	SPARTAN-3A I/O PIN
BDBUS 0	26	C7	29
BDBUS 1	25	C6	30
BDBUS 2	24	A7	27
BDBUS 3	23	D7	28
BDBUS 4	21	D8	26
BDBUS 5	20	E7	24
BDBUS 6	19	B8	25
BDBUS 7	18	C8	22
B-RXF	17	A8	23
B-TXE	16	D9	20
B-RD	15	C9	21
B-WR	14	E10	18
B-SIWU	13	A9	19

7.2 Rozhrania modulov

V tejto časti sú opísané jednotlivé rozhrania všetkých modulov použitých v rámci systému. Ako prvé je uvedené rozhranie celého systému.

7.2.1 Rozhranie celého systému

Rozhranie systému obsahuje hodinový signál, vstup a výstup pre použitie hardvérového reštartu zabezpečovaného systému, vstupy na prepínanie medzi dvomi komunikačnými rozhraniami a vstupy pre komunikáciu, čo sú vstupy rozhrania UART a paralelného prenosu na USB.

ENTITY obvod_new **IS**

PORT(

```

    CLK_16M      : IN      std_logic; — 16 MHz
    Reset_out    : OUT     std_logic;
    Reset_in     : IN      std_logic;
    — vstup na prepnutie kom. rozhrania
    COMSEL       : IN      std_logic;
    — vstup na potvrdenie prepnutia
    COMSEL_GATE  : IN      std_logic;
    — LED indikacia aktívneho rozhrania FIFO
    FIFO         : OUT     std_logic;
    — rozhranie UART
    RxD          : IN      std_logic;
    TxD          : OUT     std_logic;
    — rozhrania kanalu A — len na citanie
    ADBUS        : IN      std_logic_vector ( 7 DOWNTO 0 );
    ARxFn        : IN      std_logic;
    ATxEn        : IN      std_logic;
    ARDn         : OUT     std_logic;
    AWRn         : OUT     std_logic;
    ASIWUn       : OUT     std_logic;
    — rozhrania kanalu B — len na zapis
    BDBUS        : OUT     std_logic_vector ( 7 DOWNTO 0 );
    BRxFn        : IN      std_logic;
    BTxEn        : IN      std_logic;
    BRDn         : OUT     std_logic;
    BWRn         : OUT     std_logic;
    BSIWUn       : OUT     std_logic;

```

);

END obvod_new;

7.2.2 Rozhranie bloku viacerých watchdogov

V tomto bloku sa nachádza jeden dekodér adresy, 16 alebo 15 watchdog časovačov podľa toho, či sa jedná o najvyšší blok. Používa sa 16 *MHz* hodinový signál, signály pre riadenie watchdog časovačov a výstup pre kódovač/dekodér, ktorý sa umiestni mimo blok.

```
ENTITY watchdogs16 IS
  PORT(
    CLK_16M      : IN      std_logic;
    Pause        : IN      std_logic;
    Reset        : IN      std_logic;
    Sload        : IN      std_logic;
    address      : IN      Std_logic_vector (3 DOWNTO 0);
    enable       : IN      Std_logic;
    Zero         : OUT     std_logic;
    encoder1_in  : OUT     std_logic_vector (15 DOWNTO 0)
  );
END watchdogs16 ;
```

7.2.3 Rozhranie watchdog časovača

Watchdog časovač je implementovaný s generickou premennou **WDT_width**, ktorá určuje bitovú šírku časovača. V tomto systéme je nastavená na hodnotu 24. Vnúterná implementácia je určená na prácu so 16 *MHz* hodinovým signálom.

Vstupný port **ComCLK** je hodinovým signálom pri počítaní a pri komunikácii (nahrávaní novej inicializačnej hodnoty). Ostatné vstupné porty majú názov podľa funkcie - **Res** je reštart, **Sload** je sériový vstup pre nahrávanie hodnoty, **Pause** má funkciu pozastavenia časovača alebo vypnutia výstupného signálu **Overflow**. Signál **Zero** indikuje či je časovač na nule. Všetky porty majú šírku jedného bitu.

```
entity WDT_generic is
  generic(
    WDT_width : natural := 24
  );
  port(
    ComCLK:   in  std_logic;
    Res:      in  std_logic;
    Sload:    in  std_logic;
    Pause:    in  std_logic;
    Overflow: out std_logic;
    Zero:     out std_logic
  );
```

```
end WDT_generic;
```

7.2.4 Rozhranie zapínača

Zapínač má všetky príslušné výstupné porty pripojené so vstupmi watchdog časovača okrem výstupného portu *ZeroOut*, ktorý je pripojený na vstup multiplexora, ktorý vyberá podľa adresy z riadiacej jednotky jeden signál pre vstup riadiacej jednotky **ZeroIn**. Pomocou signálu **Enable** sa otvoria výstupné porty pre príslušný watchdog časovač. Všetky vstupy a výstupy majú šírku jedného bitu. Zapínač obsahuje register uchováajúci predchádzajúcu hodnotu *Pause* pre watchdog časovač, preto potrebuje hodinový signál.

```
entity enabler is
  port (
    CLK:      in  std_logic;
    Enable:   in  std_logic;
    LoadIn:  in  std_logic;
    ReIn:     in  std_logic;
    PauseIn:  in  std_logic;
    ZeroIn:   in  std_logic;
    LoadOut: out std_logic;
    ReOut:    out std_logic;
    PauseOut: out std_logic;
    ZeroOut:  out std_logic
  );
end enabler;
```

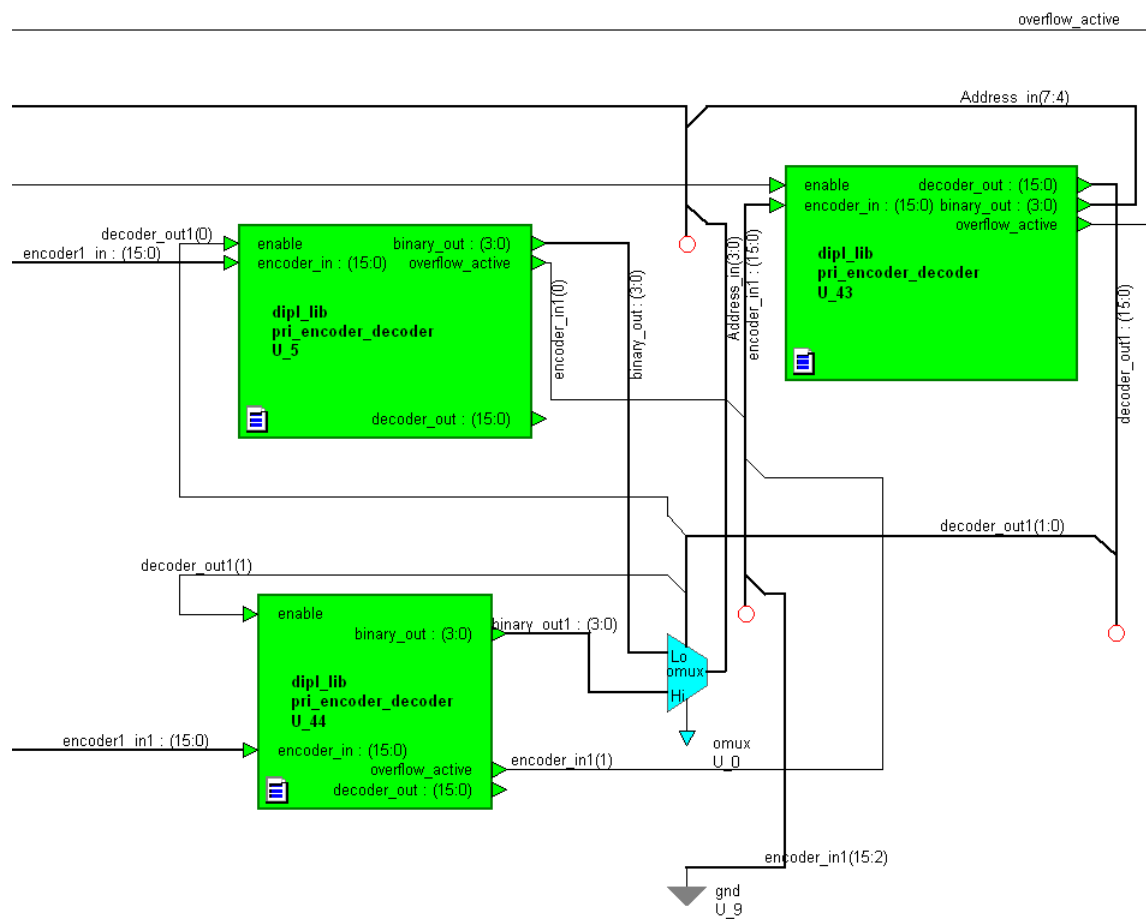
7.2.5 Rozhranie prioritného kódovača/dekodéra

Vstup *enable* zapína príslušný modul. V použitej architektúre sa nachádzajú tri moduly. Obsahuje výstup **decoder_out**, ktorý má zabezpečiť spätnú väzbu pre ďalšie moduly. Výstup **binary_out** je pripojený na adresovú zbernicu vstupujúcu do riadiacej jednotky. Výstup **overflow_active** indikuje pretečenie watchdog časovača.

V navrhnutej architektúre sú tri tieto bloky. Na najvyššej úrovni je výstup *binary_out* pripojený na vrchnú 4-bitovú časť adresnej zbernice smerujúcej do riadiacej jednotky a spodné dva bity výstupu *decoder_out* sú pripojené do rovnakých blokov reprezentujúce 2 bloky 16-tich watchdog časovačov. Výstup *binary_out* z týchto blokov je vyberaný pomocou multiplexora signálom *decoder_out* blokom vyššej úrovne a smeruje na spodnú časť adresovej zbernice vchádzajúcej do riadiacej jednotky. Použitý multiplexor vyberá vstup podľa jedného aktívneho signálu, pretože nikdy nenastane situácia, že sa budú vyberať oba bloky

súčasne. Pre ilustráciu je zapojenie na obrázku 7.2.

```
entity pri_encoder_decoder is
  port (
    enable:          in  std_logic;
    encoder_in:      in  std_logic_vector (15 downto 0);
    decoder_out:     out std_logic_vector (15 downto 0);
    binary_out :     out std_logic_vector (3 downto 0);
    overflow_active: out std_logic
  );
end pri_encoder_decoder;
```



Obr. 7.2: Prepojenie troch kódovačov/dekodérov do kaskády

7.2.6 Rozhranie 4-bitového dekodéra

Na dekódovanie adresy smerom od riadiacej jednotky k watchdog časovačom sa používa jednoduchý 4-bitový adresový dekodér. Signál **enable** zapína dekodér.

```
entity decoder_4 is
    port (
        enable:      in std_logic;
        address_in:  in std_logic_vector (3 downto 0);
        decoder_out: out std_logic_vector (15 downto 0)
    );
end decoder_4;
```

7.2.7 Rozhranie deličky hodinového signálu

Delička hodinového signálu je konfigurovateľná generickou premennou **CLK_DIVISOR**, ktorá určuje deliteľa. V použitej architektúre sa nachádza 16 *MHz* hodinový signál a na počítanie je potrebný signál o frekvencii 1 *kHz*, teda vydelí sa konštantou 16 000. Výstup z deličky vchádza do bloku riadiacej jednotky, kde je pripojený do bloku syntetizovateľného oneskorenia na počítanie času pred hardvérovým reštartovaním zabezpečovaného systému.

```
entity CLK_divider is
    generic (
        CLK_DIVISOR : integer := 16000
    );
    port(
        CLK: in std_logic;
        DivCLK: out std_logic
    );
end CLK_divider;
```

7.2.8 Rozhranie riadiacej jednotky

Prvá skupina vstupno-výstupných portov je určená pre obvod, obsahuje riadiace signály pre obvod, vstupy pre indikáciu stavu a aj hodinový signál určený pre syntetizovateľné oneskorenie, ktoré je súčasťou riadiacej jednotky. Druhá skupina portov je určená pre komunikátor. Obsahuje paralelný prenos medzi riadiacou jednotkou a komunikátorom s potvrdzovacími signálmi **ack** a **data_ready**. Posledná skupina portov je určená pre hardvérový reštart zabezpečovaného systému.

```
entity control is
    port (
```

```

CLK : in std_logic;

-- control signals for the circuit
Address_out: out std_logic_vector(7 downto 0);
Address_in: in std_logic_vector(7 downto 0);
AddressEnable: out std_logic;
ConnectEnable: out std_logic;
Reset: out std_logic;
Sload: out std_logic;
Pause: out std_logic;
ZeroIn: in std_logic;
DivClock: in std_logic; -- divided clock for counting 1kHz
Overflow: in std_logic;

-- parallel communication with the com unit
opc_out: out std_logic_vector(2 downto 0);
adr_out: out std_logic_vector(7 downto 0);
o_ack: in std_logic;
o_data_ready: out std_logic;

opc_in: in std_logic_vector(2 downto 0);
adr_in: in std_logic_vector(7 downto 0);
val_in: in std_logic_vector(23 downto 0);
i_ack: out std_logic; -- acknowledgement to com unit
i_data_ready: in std_logic; -- input data ready

-- signals for hard reset (optional) --
Reset_in: in std_logic;
Reset_out: out std_logic
);
end control;

```

Rozhranie syntetizovateľného oneskorenia

Na vstupný port **CLK_1kHz** musí byť pripojený hodinový signál s frekvenciou 1 kHz, inak nebude oneskorenie pracovať v požadovaných časových jednotkách. Vstup **seconds** je troj-bitový vektor, ktorého hodnota je oneskorenie v sekundách v binárnom bezznamienkovom formáte.

```

entity SynthDelay is -- Synthetizable delay
port(
    CLK_1kHz: in std_logic;
    Reset: in std_logic;
    seconds: in std_logic_vector (2 downto 0);

```

```

        Ready:      out std_logic
    );
end SynthDelay;

```

7.2.9 Rozhranie komunikátora

Komunikátor pracuje s dvomi komunikačnými rozhraniami. Výber je riadený pomocou multiplexorov, pričom výberová hodnota je uložená v registri. Tento register je možné meniť pomocou používateľského vstupu - tlačidiel. Prvý blok portov je určený na komunikáciu s komunikačnými rozhraniami a ďalšie dva sú určené na komunikáciu s riadiacou jednotkou, ktorá má rovnaké porty, len opačnej orientácie. Porty sú pomenované podľa rozhrania UART, ale pri implementácii bloku pre paralelný prenos je implementované rovnaké rozhranie, aby sa nemusel komunikátor meniť.

```

entity com is
    port(
        Clk:          in std_logic;
        — communication with the uart interface
        uart_reset:    out std_logic;
        uart_rdrqn:    out std_logic;
        uart_wrrqn:    out std_logic;
        u_data_out:     out std_logic_vector(7 downto 0);
        u_data_in:      in  std_logic_vector(7 downto 0);
        data_ready:     in  std_logic;
        parity_error:   in  std_logic;
        framing_error:  in  std_logic;
        uart_tbuf_empty: in std_logic;
        uart_tsbuf_empty: in std_logic;

        — parallel communication with the control unit
        opc_out:        out std_logic_vector(2 downto 0);
        adr_out:        out std_logic_vector(7 downto 0);
        val_out:        out std_logic_vector(23 downto 0);
        o_ack:          in  std_logic;
        o_data_ready:   out std_logic;

        opc_in:         in  std_logic_vector(2 downto 0);
        adr_in:         in  std_logic_vector(7 downto 0);
        i_ack:          out std_logic;
        i_data_ready:   in  std_logic
    );
end com;

```

7.2.10 Rozhranie použitého modulu UART

Modul UART bol prebraný z *OpenCores*, jeho rozhranie ostalo nezmenené, bol pridaný len signál `tx_empty`.

```

entity uart is
generic (
    CLK_FREQ: integer := 50;           — Main frequency (MHz)
    SER_FREQ: integer := 9600         — Baud rate (bps)
);
port (
    — Control
    clk           : in    std_logic;
    rst           : in    std_logic;
    — External Interface
    rx            : in    std_logic;
    tx            : out   std_logic;
    — RS232/UART Configuration
    par_en        : in    std_logic;
    — uPC Interface
    tx_req         : in    std_logic;
    tx_end         : out   std_logic;
    tx_empty       : out   std_logic;
    tx_data        : in    std_logic_vector(7 downto 0);
    rx_ready       : out   std_logic;
    rx_data        : out   std_logic_vector(7 downto 0)
);
end uart;

```

7.2.11 Rozhranie blokov typu FT245 pre paralelný prenos

Rozhrania modulov sú z jednej strany rozhraním pre obvod FT2232H v stave asynchrónneho prenosu FT245. Rozhranie pre komunikátor je podobné ako pri rozhraní UART, aby nebolo nutné komunikátor meniť.

```

ENTITY READasync_fifo IS
  PORT(
    CLK           : IN    std_logic;
    RES           : IN    std_logic;
    ADBUS         : IN    std_logic_vector ( 7 DOWNTIO 0 );
    ARxFn         : IN    std_logic;
    ATxEn         : IN    std_logic;
    ARDn          : OUT   std_logic;

```

```

        AWRn      : OUT      std_logic ;
        ASIWUn    : OUT      std_logic ;

        DataOut   : OUT      std_logic_vector ( 7 DOWNTO 0 );
        data_ready : OUT      std_logic ;
        ack       : IN       std_logic

    );
END READasync_fifo ;

ENTITY WRITEasync_fifo IS
    PORT(
        CLK      : IN      std_logic ;
        RES      : IN      std_logic ;
        BRxFn    : IN      std_logic ;
        BTxEEn   : IN      std_logic ;
        BDBUS     : OUT     std_logic_vector ( 7 DOWNTO 0 );
        BRDn     : OUT     std_logic ;
        BWRn     : OUT     std_logic ;
        BSIWUn    : OUT     std_logic ;

        DataIN    : IN      std_logic_vector ( 7 DOWNTO 0 );
        tx_req     : IN      std_logic ;
        tx_empty   : OUT     std_logic

    );
END WRITEasync_fifo ;

```

7.3 Technická príručka

V tejto časti je opísaný postup, akým spôsobom sú použité jednotlivé vývojové prostredia, akým spôsobom prebieha syntéza a implementácia do obvodu.

7.3.1 Postup pri nahrávaní logiky do obvodu

Na systémovej úrovni je použité vývojové prostredie FPGA Advantage 8.1. Projekt používa knižnicu *dipl.lib*. Najvyšší blok, kde je implementovaná veľká architektúra je *obvod_new*. V tomto prostredí je možné upravovať zdrojové kódy blokov a schém zapojení. Na syntézu a *Place&Route* je používané prostredie Xilinx ISE Webpack 13.4. Vytvorený projekt obsahuje kópie zdrojových VHDL kódov z projektu FPGA Advantage. Knižnicu *dipl.lib* je nutné v každom súbore premenovať na knižnicu *work*, tým sa zabezpečí bezproblémová syntéza. Výstupom je binárny súbor - *bitstream*.

Pri použití ladenia pomocou aplikácie *ChipScope Pro* je nutné najprv vložiť alebo vytvoriť nový blok pre tento analyzátor - *ChipScope Definition and Connection File*, ktorý bude súčasťou projektu v *Xilinx ISE*. V ňom sa nastaví všetky signály, ktoré sa budú sledovať. Po implementácii je potrebné zapojiť Xilinx JTAG kábel na JTAG konektor J5 vývojovej dosky. Vývojovú dosku je potrebné napájať externe alebo cez USB zbernicu. Aplikáciou *Xilinx iMPACT* sa nakonfiguruje FPGA pomocou príslušného *bitstream*-u. Potom sa spustí logický analyzátor *ChipScope Pro*. Všetky príkazy je možné spúšťať z okna projektu v *Xilinx ISE*. Analyzátor sa pripojí pomocou tlačidla *connect* a načítajú sa signály. Tie je ale nutné manuálne premenovať podľa poradia v akom boli pridávané počas vytvárania *ChipScope Definition and Connection File*. Ďalej je práca v programe totožná ako v bežnom logickom analyzátoze, t.j. nastaví sa *Trigger* a signály sa načítajú z obvodu cez JTAG.

Na implementáciu do vývojovej dosky Spartan-3A Evaluation kit sa používa aplikácia *AvProg*. Pred prvým spustením ale bolo nutné prepísať firmvér PSoC programátora na verziu 1.1 pomocou aplikácie Cypress PSoC Programmer a hardvérového PSoC Miniprogramátora. Po nahradení je možné pracovať s aplikáciou *AvProg*. Programovanie FPGA je jednoduché, ak chceme programovať SPI flash pamäť, je potrebné prepnúť aplikáciu do SPI módu a vymazať obsah pamäte. Potom je možné nahráť príslušný *bitstream* do pamäte a po reštartovaní vývojovej dosky sa *bitstream* nahrá z flash pamäte do FPGA.

Ak je používaný operačný systém vo virtuálnom prostredí a USB zariadenia sú premostené, nemusí prebehnúť programovanie SPI flash pamäte korektne z dôvodu oneskorenia v komunikácii. V tomo prípade je nutné zapnúť ladiace okno aplikácie *AvProg*. Toto okno je aktivované, keď sa dvakrát klikne na logo *AvNet* a z menu *Help* sa vyberie možnosť *Debug Panel*. Heslo do tohto panelu je 4212¹. Parameter *Poll Limit* je potrebné z hodnoty 1000 zvýšiť na hodnotu 2500, poprípade vyššiu.

7.3.2 Nastavenie USB modulu

USB modul FT2232H obsahuje EEPROM pamäť, ktorú je možné meniť pomocou aplikácie *FT_Prog* dostupnej na adrese výrobcu FTDI. Po pripojení modulu so správnym zapojením napájania z USB zbernice sa spustí *FT_Prog* a dá sa načítať konfigurácia pomocou tlačidla *lupy (Scan and Parse)*. Konfigurácia sa dá jednoducho meniť, takže pri oboch kanáloch sa nastaví použitie FIFO FT245 pre asynchrónny paralelný prenos a ovládače sa nastaví ako virtuálne komunikačné proty (*VCP*). Samozrejme pre budúcnosť je možné pracovať cez priame ovládače, to bude vhodné pre programovanie softvérového ovládača. Nastavenie konfigurácie sa uloží do pamäte pomocou tlačidla *blesku (Program Devices)*.

¹Podľa adresy <http://tristesse.org/AvnetSpartan3AEvaluationKit> (2012-03-08)

Literatúra

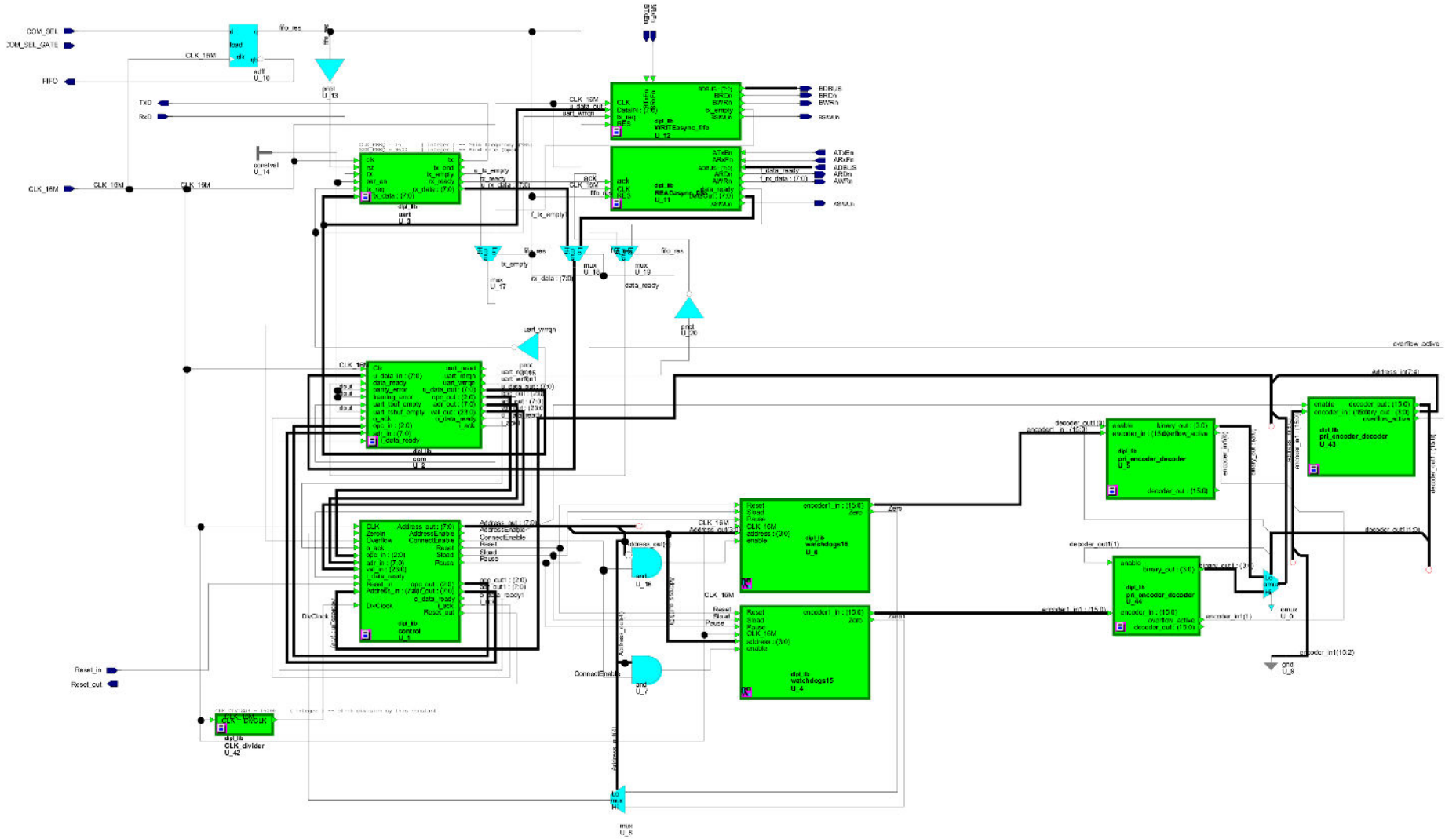
- [1] Lee, Insup and Leung, Joseph Y-T. and Son, Sang H.: "Handbook of Real-Time and Embedded Systems," Chapman & Hall/CRC, (2007). ISBN: 978-1-58488-678-5.
- [2] Labrosse, J., Ganssle, J., Noergaard, T. et al.: Embedded Software: Know It All. Burlington: Elsevier, 2008. 770 s. ISBN 978-0-7506-8583-2.
- [3] Vahid, F., Givargis, T. D.: Embedded System Design: A Unified Hardware/Software Introduction. 1999. 103 s. ISBN 978-0-4713-8678-0.
- [4] Pohronská, M. & Krajčovič, T. (2010), Fault-tolerant embedded systems with multiple fpga implemented watchdogs, in Štefan Kozák, Alena Kozáková, Danica Rosinová, ed., "Proceedings of the International Conference CYBERNETICS AND INFORMATICS", Vydavateľstvo STU, pp. 37.
- [5] Abaffy, J., Krajčovič, T. (2010): Software support for multiple hardware watchdog timers in the Linux OS, 2010 International Conference on Applied Electronics (September 2010).
Online dostupné na: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5599576&isnumber=5599555> (2010-11-24)
- [6] National Instruments. 2006. Instrument Bus Performance – Making Sense of Competing Bus Technologies for Instrument Control. Online dostupné na:
<http://zone.ni.com/devzone/cda/tut/p/id/3509> (2011-03-22)
- [7] Dado, B.: Universal Serial Bus. Učebný text k predmetu Periférne zariadenia. Online dostupné na:
<http://www2.fiit.stuba.sk/~dado/Pz/Cvicenia/Texty/06%20-%20USB.pdf> (2011-04-26)
- [8] Valach, S.: Technologie: Sběrnice PCI Express. 2005. Online dostupné na:
http://www.svethardware.cz/art_doc-DCF855B3ED8092C6C1256FB500796ADF.html (2011-04-26)

- [9] Xilinx Spartan-3A Evaluation Kit User Guide. Online dostupné po bezplatnej registrácii na: www.em.avnet.com/spartan3a-ev1 (2012-04-20).
Dokumentácie sú dostupné aj na: http://weble.upc.es/asig/postgrado/digital/modulo3/curso_Febrero_2011/Avnet%20Spartan3A%20eval%20kit/ (2011-12-03)
- [10] Xilinx: ML507 Board User Guide. Online dostupné na:
http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf
(2011-04-26)
- [11] Xilinx: Virtex-5 Family Overview. Online dostupné na:
http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf (2011-04-26)
- [12] FTDI FT2232H mini module datasheet. Online dostupné na
http://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_FT2232H_Mini_Module.pdf (2011-12-09)
- [13] FTDI FT2232H datasheet. Online dostupné na
http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf
(2011-12-09)

PRÍLOHY

A Schéma architektúry

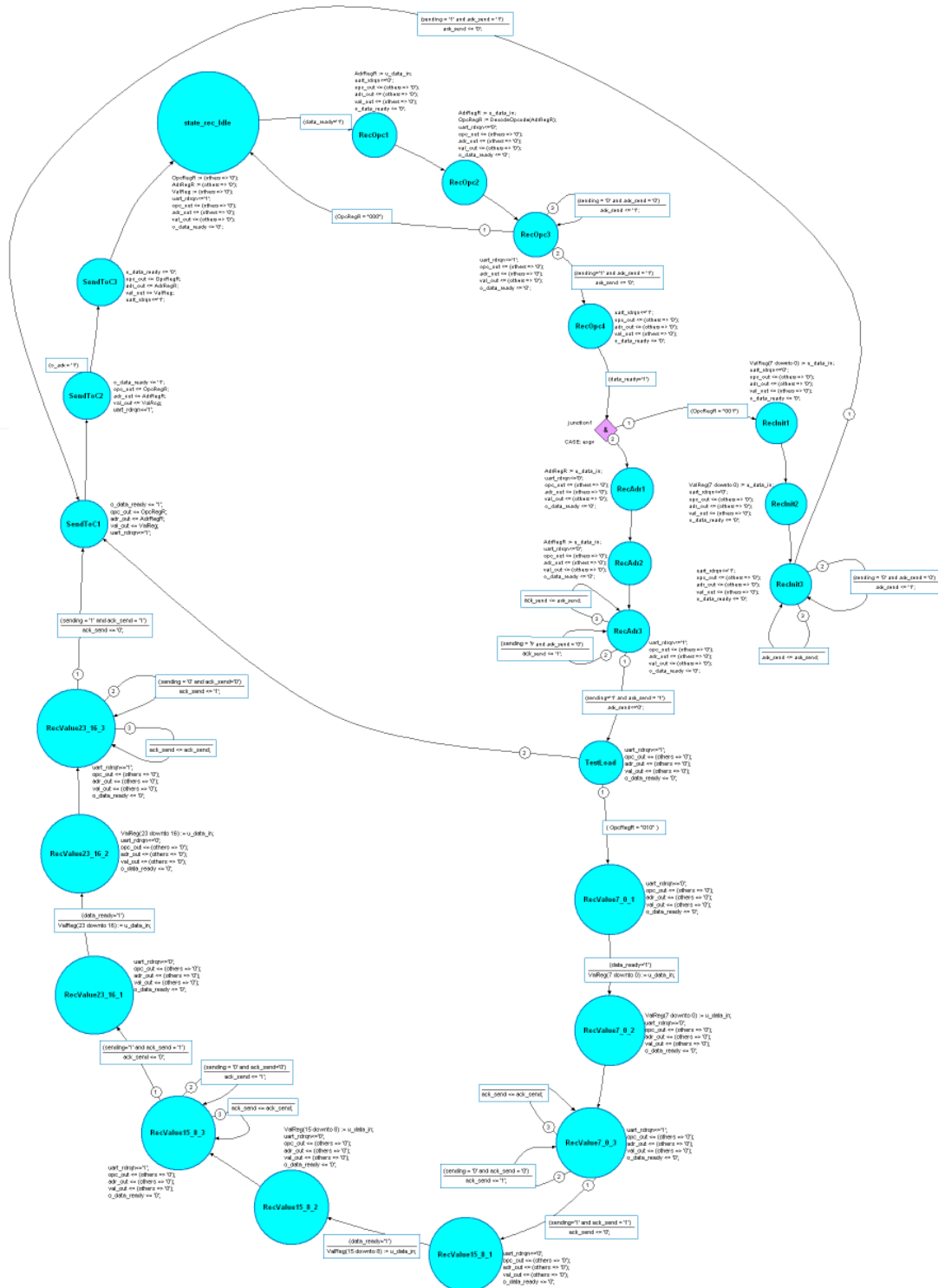
V tejto časti sa nachádza schéma architektúry systému na najvyššej úrovni, pričom watchdog časovačov je možné zapojiť až 31. V zapojení je blok UART, rozhranie pre USB komunikáciu, komunikátor, riadiaca jednotka, bloky watchdog časovačov s dekodérmi, prioritné kódovače-/dekodéry.



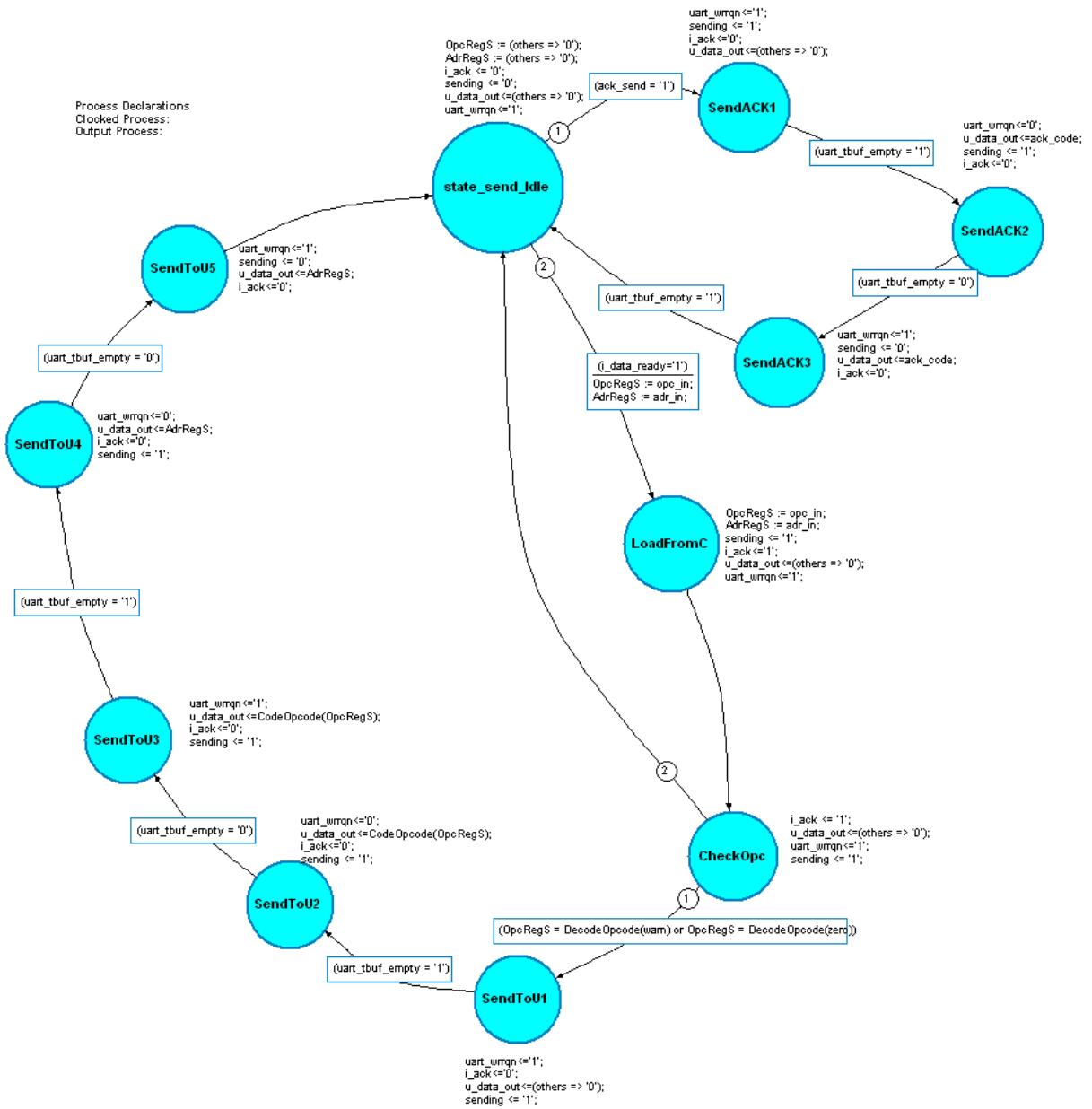
B Kompletné diagramy konečných stavových automatov

V tejto časti sa nachádzajú kompletne diagramy konečných stavových automatov (FSM) bloku komunikátor, riadiaca jednotka, watchdog časovača a paralelný asynchrónny prenos FT245 na komunikáciu s USB. Keďže niektoré z diagramov sú rozsiahle, nachádzajú sa aj na priloženom CD samostatne vo formáte PNG.

B. Kompletné diagramy konečných stavových automatov

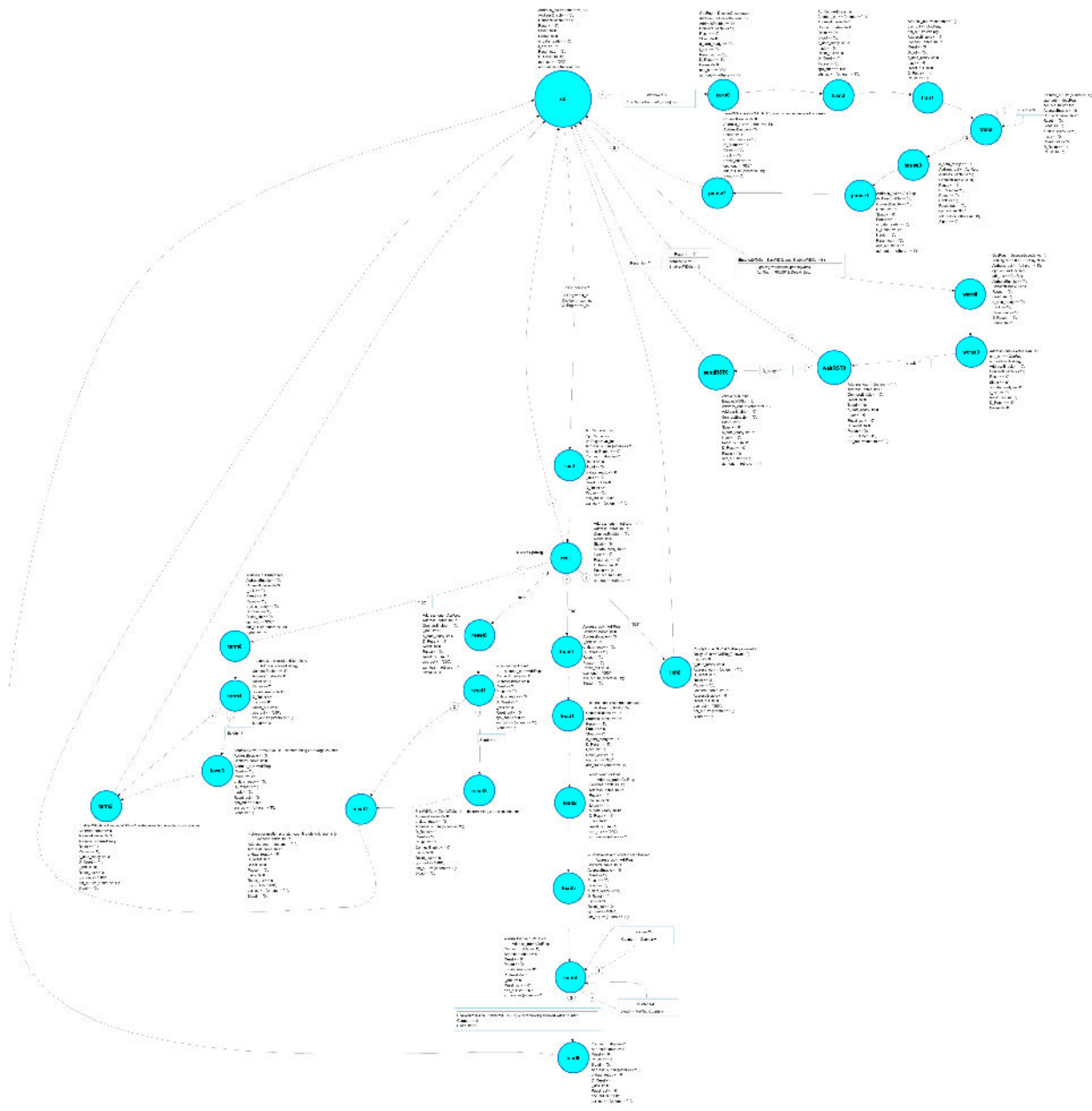


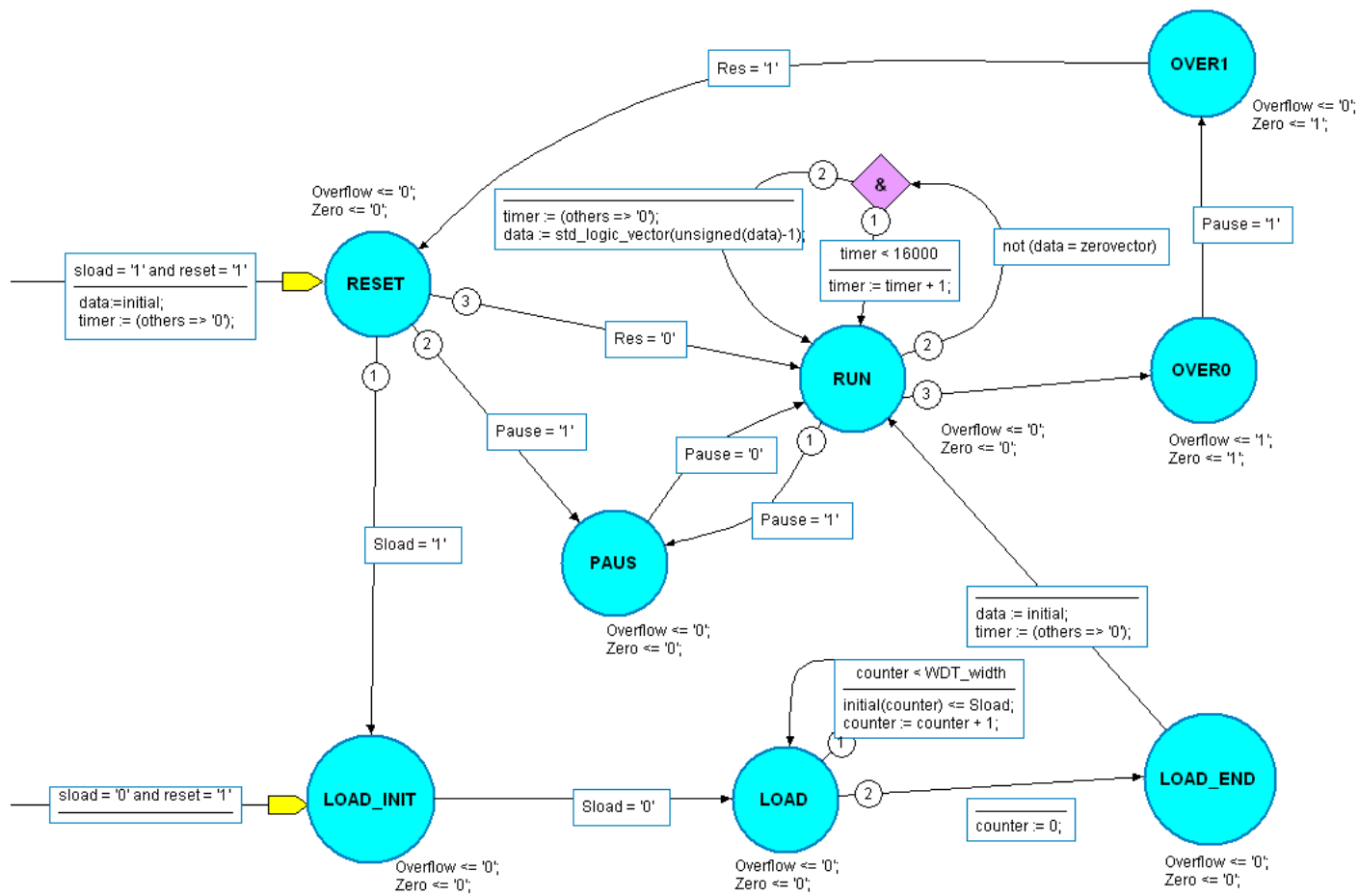
Obr. B.1: FSM Diagram procesu pre prijímanie správ komunikátora



Obr. B.2: FSM Diagram procesu pre odosielanie správ komunikátora

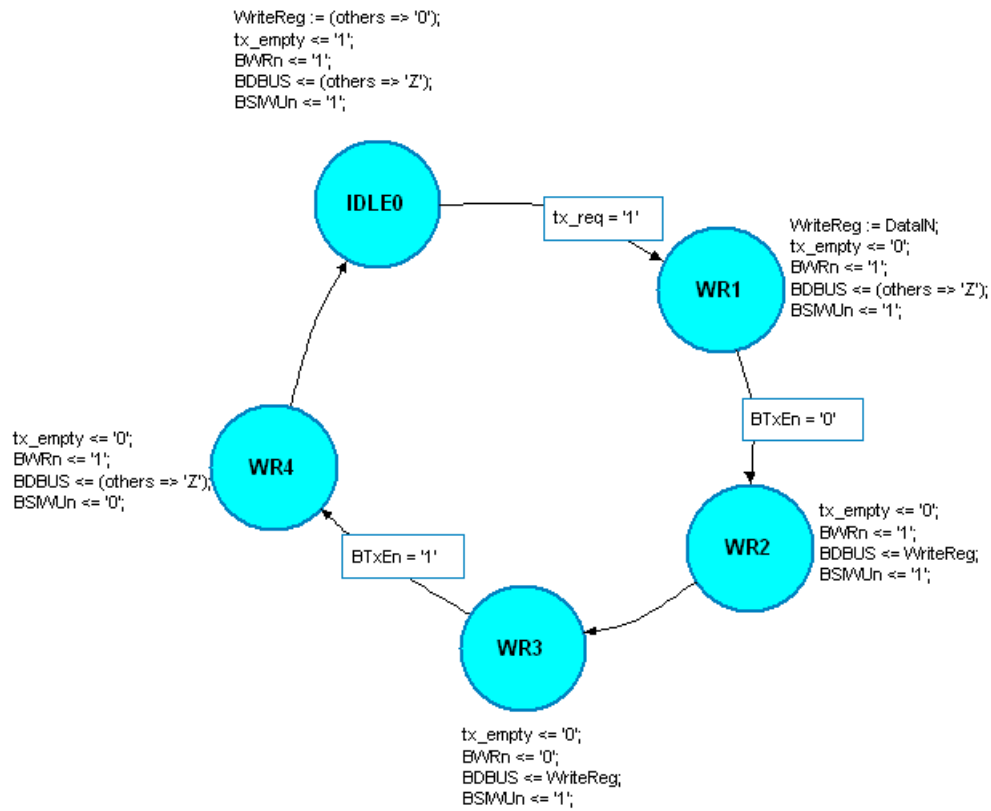
B. Kompletné diagramy konečných stavových automatov



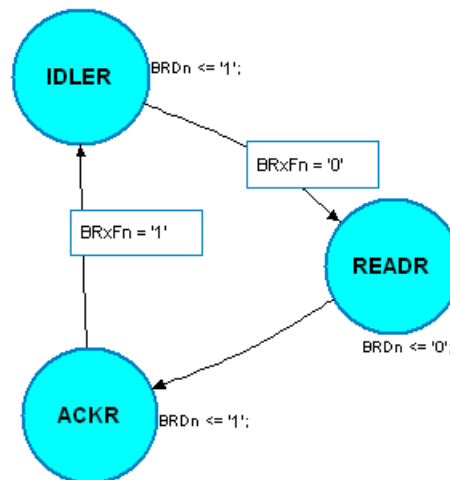


Obr. B.4: FSM Diagram watchdog časovača

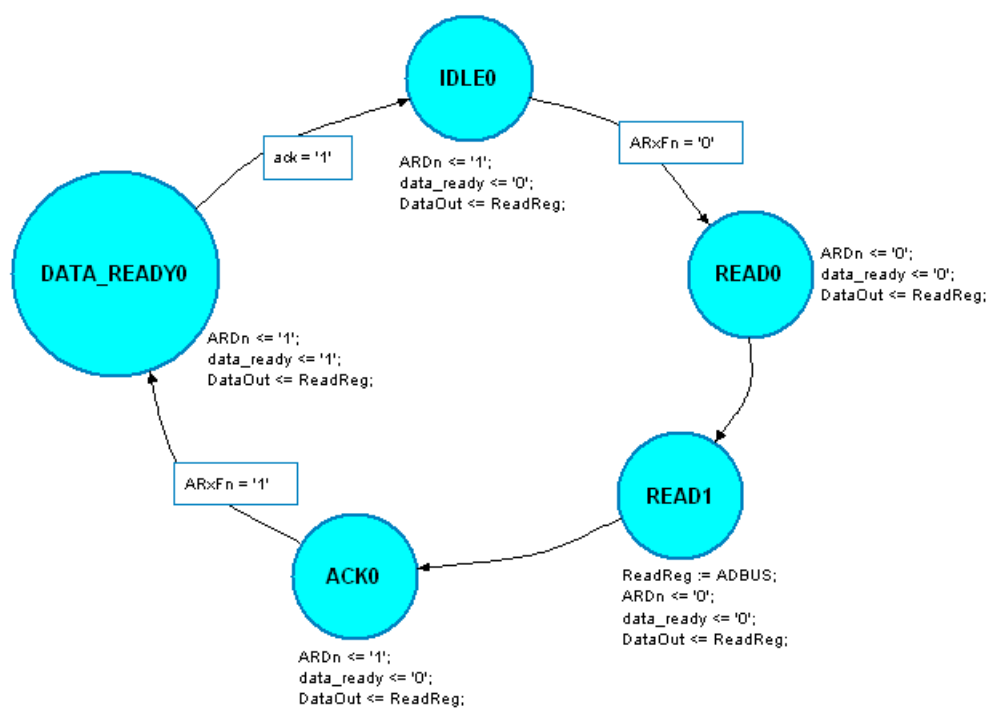
B. Kompletné diagramy konečných stavových automatov



Obr. B.5: FSM Diagram procesu zápisu pre USB rozhranie na zápisu



Obr. B.6: FSM Diagram procesu čítania pre USB rozhranie zápisu



Obr. B.7: FSM Diagram procesu čítania pre USB rozhranie čítania

B. Kompletné diagramy konečných stavových automatov

C Používateľská príručka

V tejto kapitole je opísaný spôsob používania systému z pohľadu používateľa.

Pred spustením systému je nutné zapojiť FTDI modul na konektor J4 vývojovej dosky ako je zobrazené na obrázku 7.1. Vývojová doska a FTDI modul je napájaný pomocou USB zbernice, preto sú potrebné dva USB káble s B-konektorom typu *mini*. Cez USB kábel z FTDI modulu bude prebiehať dvojkanálová komunikácia a cez kábel pripojený k vývojovej doske prebieha komunikácia cez UART. FTDI modul aj vývojová doska potrebujú mať na hosťovskom počítači nainštalované ovládače virtuálneho sériového portu.

Na vývojovej doske sú aktívne tri tlačidlá. Tlačidlo *PUSH_B* slúži ako vstup pre register výberu komunikačného rozhrania a tlačidlo *PUSH_C* ako potvrdenie (hodinový signál) tohto registra. LED dióda D2 svieti ak je aktívne komunikačné rozhranie pre paralelný USB prenos (prvotná hodnota po zapnutí) a ak je zhasnutá, je aktívne rozhranie UART. Hardvérový reštart je simulovaný pomocou LED diódy D5 ako výstup a tlačidlo *PUSH_A* ako vstup pre indikáciu, že sa systém reštartoval.

Komunikácia prebieha pomocou protokolu opísaného v kapitole 3.2.6. Na komunikáciu pomocou aplikácie *RealTerm* je pri aktívnom rozhraní UART potrebné nastaviť sériovú komunikáciu na 19 200 *baud*, 8 dátových bitov, 2 stop bity (ale funguje aj 1 stop bit), žiadne riadenie toku a žiadnu paritu. Komunikácia je obojsmerná vrámci jedného portu a je bajtovo orientovaná. Každý bajt komunikačného slova je potvrdzovaný systémom a až po prijatí potvrdenia (*ACK*) je možné odoslať ďalší bajt.

Pri použití rozhrania paralelného prenosu je jedno aké parametre sériových portov budú nastavené, ovládač ich ignoruje a pracuje priamo s USB. Kanál *A*, ktorý bude mať na hosťovskom systéme pravdepodobne nižšie číslo virtuálneho komunikačného portu, slúži na odosielanie bajtovo-orientovaného komunikačného protokolu. Kanál *B* slúži na prijímanie dát zo zabezpečovacieho systému. Je teda potrebné obsluhovať dva virtuálne komunikačné porty buď cez dve aplikácie *RealTerm* alebo iný program schopný zobrazovať prijaté bajty v hexadecimálnej sústave. Takým je aj napríklad 30-dňová verzia programu *COM Port Toolkit*, ktorý vie zobrazovať aj čas prijatia bajtov v mikrosekundách.

D Publikovaný článok na medzinárodnej konferencii Kybernetika a Informatika 2012

A Hardware Reset Implementation in Multiple FPGA Watchdog System

Maroš Ďuriček*, Mária Pohronská*, Tibor Krajčovič*

**Faculty of Informatics and Information Technologies,
Slovak University of Technology,
Ilkovičova 3, 842 16 Bratislava 4, Slovakia
(e-mail: rozki308@gmail.com, [pohronska]{tkraj}@fiit.stuba.sk)*

Abstract: Security of multi-process embedded systems is of a great concern in majority of embedded and real-time applications. Our work deals with a multiple hardware watchdog timer device that is intended to improve the security and reliability of multi-process systems. The developed architecture is capable of automatically detecting the need of restarting the whole secured system and performing the hardware reset after a configurable delay.

Keywords: watchdog, reconfigurable circuits, fault-tolerant systems, reset, concurrent processes.

1. INTRODUCTION

Embedded and real-time systems are characterised by strict requirements to the safety of their operation. These requirements include guaranteed response time, continuous operation, self-diagnosis and many others. However, a typical embedded system is equipped with only one hardware watchdog timer which secures the whole system with all its concurrent processes running. Using such timer with a number of processes is inefficient as in this case there is only one software driver which serves the hardware timer. In the case of failure in one of these processes, it can be difficult to determine which process has just failed. A multiple watchdog system can solve this problem, but only partially. If the complete system fails, an external interaction is needed to correctly restart the whole system. One of the characteristics of a watchdog system is to “issue a hardware reset on time-out” (Labrosse et al., 2008).

The standard way for securing processes of rich embedded or real-time systems is using a multiple software watchdogs. The software watchdog system has several virtual watchdog timers securing the running processes, while utilizing only one hardware timer for watchdog emulation. A multiple hardware watchdog system provides better security by assigning an independent hardware watchdog to each controlled process, what makes it suitable for usage also in real-time systems.

Our work deals with an existing architecture of multiple hardware watchdog system that has been described in (Pohronská, 2011), adding several improvements including the possibility to physically restart the secured system.

1.1 Related work

Several commercial watchdog extensions exist which can be connected to personal or industrial computers. However, these systems are working only with one or few hardware

watchdog timers. Most commonly used communication interfaces are serial port, peripheral component interconnect (PCI) and keyboard communication port.

We base our work on the experimental multiple hardware watchdog system implemented in FPGA (Pohronská, 2011). This system architecture is modular and easy to extend. It is using the standard serial port (UART) to communicate with the secured system. The basic counter unit is divided into two components. The first component is small 16 or 24-bit watchdog timer, the second component is a logic which enables the timer. The main unit of the system is control unit, which provides interface between the watchdog circuit and UART. A software driver is required on the secured system side.

This work exploits the main ideas of the existing multiple hardware watchdog architecture and adds several major improvements to the system's architecture and functionality. One of these improvements is the reset of the complete secured system.

Section two describes the architecture changes and improvements to the existing multiple watchdog system. In the section three, we deal with the implementation of modules to support the hardware reset function. Section four describes performed experiments and obtained results. Last section concludes the document.

2. IMPROVED ARCHITECTURE OF MULTIPLE HARDWARE WATCHDOG SYSTEM

2.1 Architecture changes

We have made several improvements to the original architecture (Pohronská, 2011). Besides, all source codes of used modules have been optimized. This section deals with the most important improvements.

The original control unit has been divided into two components. First component is the *communicator* which is disposed to communicate not only through UART but also through other interfaces (e.g., USB shown in Fig. 1). Second component is the *control unit* which task is to control the watchdog timers and handle the hardware reset function. This division of these components was necessary since it enables bidirectional communication and adds another main or backup communication interface.

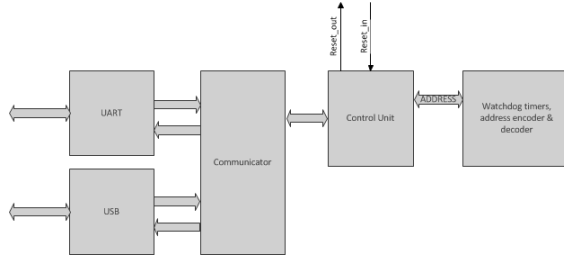


Fig. 1. Extended architecture of multiple watchdog system

The original communication protocol has been extended and now contains six coded messages (Table 1). All messages begin with 8-bit message code and have variable length from 16 to 40 *bits* including the message code. The message codes are seemingly random, but they have been carefully chosen to “ensure a rogue program cannot reprogram watchdog control registers” (Labrosse et al., 2008). One of the added messages is the *terminate* message. This message is necessary for the control unit to recognise enabled from terminated watchdog timers. Every watchdog timer now stops counting when it is not enabled. This is important for power saving, also.

Message *init* holds parameters for configuration of the control unit. The *load* message is used for loading a new 24-bit value to a particular watchdog timer. Message *reset* restarts the particular watchdog. Messages sent to the secured system are *warning* before hardware reset with a delay value and *zero* indicating that a particular watchdog has overflowed.

Table 1. Communication messages

Message	Code	First 8 <i>b</i>	Next 24 <i>b</i>
INIT	0x2D	Priority watchdog quantity, Delay in sec.	
LOAD	0x3B	Address	Value
RESET	0x1A	Address	
TERMINATE	0xF9	Address	
WARNING	0x49	Delay in sec.	
ZERO	0x52	Address	

The circuit’s interface has been extended by hardware reset input and output port. These ports are part of the control unit module and are only one *bit* wide. The hardware reset input port serves for indication that the secured system has already self-restarted. It should be connected to the reset indication

output of the secured system. The hardware reset output port should be connected to the hardware reset switch on the motherboard of the secured system. When necessary, the device is competent to execute hardware reset of the complete secured system.

3. IMPLEMENTATION OF THE HARDWARE RESET

3.1 Principle of the hardware reset feature

When one of the running processes of the secured system accidentally fails, it is recognised by the responsible hardware watchdog timer and the device sends a message with the timer address to the software watchdog driver in the secured system (message *zero*). The system knows the specific process identification number and it is able to reset or kill the failed process.

In the case when all running processes fail and the software watchdog driver fails too (it must have one watchdog timer assigned too), the existing architecture sends messages of all overflowed watchdog timers but does not detect the overall system failure. The main contribution of our work is adding a capability to automatically detect the failure and reset the whole system after a given period of time. This delay is important, because the secured system can have employed other mechanisms of failure detection and can be able to restart itself more safely without assistance of the multiple hardware watchdog system.

The hardware reset output of the multiple watchdog system has to be directly connected to the reset pin of the secured system and it has to have hardware feedback to know if the secured system has been restarted.

3.2 Control unit changes for hardware reset feature

Two variables suit the purpose to detect that all enabled watchdogs are overflowed. First is the counter of enabled watchdogs *EnabledWDGs* that holds the count of running enabled watchdog timers. It increments when the secured system sends message *load* for allocating a new watchdog timer. It decrements when the secured system does not need the watchdog anymore and sends the message *terminate*.

The second variable used is a counter of overflowed watchdogs *ZeroWDGs*. It increments when a watchdog overflows. It decrements when the message *reset* or message *terminate* for the particular watchdog timer is received.

These two counters are initialised to zero when starting. If the counters are equal and nonzero, it indicates that all monitored processes have failed and it is supposed that the system has to be restarted. After detection of such situation, the device waits for a specific time that is configured at initialization of the device, with the message *init*. During this time it expects the external reset indication before restarting the secured system. The granularity of this delay time is in seconds, from 0 to 7 seconds.

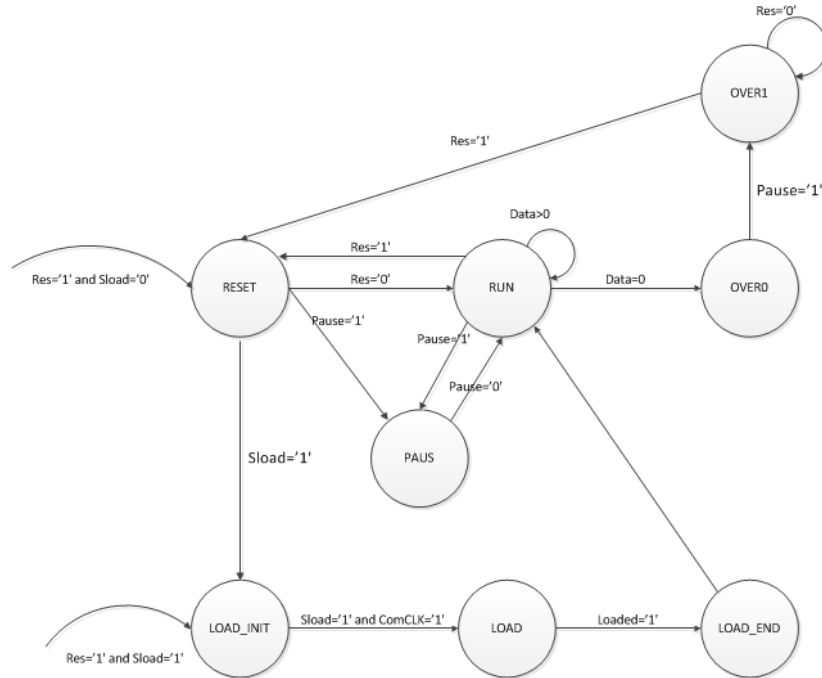


Fig. 2. Simplified finite-state machine diagram of the watchdog timer

The control unit and communicator are interconnected using parallel ports with two handshake signals. The control unit can now send messages to the communicator while the communicator is receiving the messages from the secured system. Each received message from secured system is acknowledged by the communicator, because of the small buffers in the control unit or communicator memory.

3.3 Watchdog timer changes for hardware reset feature

The message *reset* or *terminate* from secured system holds only the address of the watchdog timer so the control unit is not aware if the watchdog is in overflowed state or not. From this reason we have added a feedback signal from each watchdog timer directed to the control unit which indicates that this watchdog timer is in overflowed state. This information is used by the control unit to determine if the *ZeroWDGs* variable should be decremented.

If a watchdog overflows (state *OVER0* in Fig. 2), the control unit sends a message *zero* to the secured system. After that, the particular watchdog timer is stopped by the control unit and switched to the passive state waiting for reset (the *OVER1* state in Fig. 2). In the original architecture, watchdog stayed in the state indicating overflow until it has been reset by the secured system's message, what effectively masked the overflow outputs from other timers. The added watchdog state allows detecting other overflowed timers right after serving the previous ones and sending more *zero* messages of overflowed watchdog timers to the secured system, without a

blocking wait for the system's response. When the message *reset* arrives, the control unit obtains the information whether the watchdog timer was overflowed or not and restarts the watchdog timer. This is necessary for decrementing the *ZeroWDGs* variable.

The simple finite-state machine of the watchdog timer (Fig. 2) has been optimized and extended. We have added the state *PAUS* to switch the timer to the inactive mode. A watchdog is in this state immediately after the system starts or when the message *terminate* is received. During the *OVER1* state the overflow signal is inactive and the watchdog timer is in passive mode.

In the state *LOAD* the watchdog timer performs a special process sensitive to the faster communication clock. Its function is to load the particular initial value faster (16 MHz) than the counting speed (1 kHz).

3.4 Synthesisable delay

For simulation purposes it is possible to use the VHDL *wait* keyword to get a time delay in a process. This is not possible in a synthesised circuit, because the *wait* keyword uses the time units. In this case we are using the clock signal to supply the hardware reset delay in time.

The control unit has the initial value for delay of the hardware reset signal set to 7 seconds. The value is configurable by the message *init*. The delay is specified by 3-bit value in seconds. The synthesisable delay is implemented

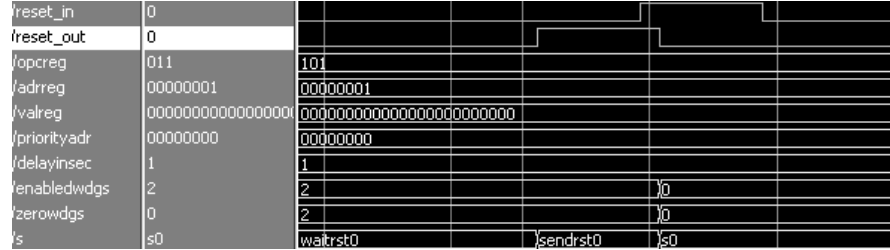


Fig. 3. Hardware reset simulation in ModelSim 6.4

as a counter that operates on 1 kHz clock signal. This delay module is a part of the control unit. One of the input ports of the delay module is a 3-bit vector with the desired delay value. The delay module converts the input to the desired value and counts to this value. When this limit is reached, this module activates signal *ready*.

4. EXPERIMENTAL RESULTS USING SPARTAN-3A CIRCUIT

The design has been implemented in VHDL and synthesized into a Xilinx Spartan-3A device, using the Xilinx ISE Webpack 13.1 software.

The function of the whole system with 16 watchdog timers has been tested and simulated in ModelSim 6.4. Testing entity was created in FPGA Advantage 8.1 using data flow diagram. This entity simulates a system with two watchdog timers and RS232 interface for communication. Aim of the test was to verify the function of watchdog timers, control unit, communication unit and hardware reset feature.

During the simulation we discovered few problems that were eliminated. We had to implement the communicator to support the bidirectional communication as described in section 2.1. The hardware reset has to wait for the response on the *Reset_in* port as shown in Fig. 3 to wait for the system boot.

The test was divided into three parts. Firstly, to load a specified value into two watchdog timers and let them count to zero. Secondly, one of the watchdog timers was reset. Finally, the two watchdogs overflowed and the system had to activate the hardware reset. The last part of the simulation is shown in Fig. 3.

After the simulation we made changes in implementation without changing the functionality, to make the architecture synthesizable. All modules are behaviourally described in VHDL as finite-state machines.

Results of synthesis of this system are shown in Fig. 4. The system was synthesised with 16 watchdog timers for the experimental purposes. As shown in Fig. 4, it is possible to implement architecture with even more watchdog timers in circuit in the future.

Device Utilization Summary				I-1
Logic Utilization	Used	Available	Utilization	Note(s)
Total Number Slice Registers	1,396	7,168	19%	
Number used as Flip Flops	1,200			
Number used as Latches	196			
Number of 4 input LUTs	3,293	7,168	45%	
Number of occupied Slices	2,215	3,584	61%	
Number of Slices containing only related logic	2,215	2,215	100%	
Number of Slices containing unrelated logic	0	2,215	0%	
Total Number of 4 input LUTs	3,372	7,168	47%	
Number used as logic	3,293			
Number used as a route-thru	79			
Number of bonded IOBs	5	195	2%	
IOB Latches	1			
Number of BUFGMUXs	4	24	16%	
Average Fanout of Non-Clock Nets	3.29			

Fig. 4. Device Utilization Summary after synthesis into Xilinx Spartan-3A device

5. CONCLUSIONS AND FUTURE WORK

We have developed a device for securing multi-process real-time systems by providing multiple hardware watchdog timers and functionality of an emergency hardware reset. The work is an iteration of the previous multiple hardware watchdog system, with architectural optimizations and added functionality. The key feature is the function of restarting the secured system, which includes automatic detection of failure and execution of the reset with configurable delay.

In future work, we plan to extend the communication possibilities with USB, as it provides higher speed and lower latency. We also plan to add the support of flash memory for storing the debug and diagnostic information.

ACKNOWLEDGEMENT

This work was partially supported by the Grant No. 1/1105/11 of the Slovak VEGA Grant Agency. This work has been materially supported by the project ITMS 26240120005 supported by the Research 7 Development Operational Programme founded by the ERDF.

REFERENCES

- Pohronská, M.; Krajčovič, T. (2011). FPGA Implementation of Multiple Hardware Watchdog Timers for Enhancing Real-Time Systems Security. In *Eurocon 2011 proceedings*. IEEE.
- Labrosse, J., Ganssle, J., Noergaard, T. et al.: *Embedded Software: Know It All*. Burlington: Elsevier, 2008. 770 s. ISBN 978-0-7506-8583-2.

D. Publikovaný článok na medzinárodnej konferencii Kybernetika a Informatika 2012

**E Odoŝlaný ĉlánok na medzinárodnú
konferenciu Applied Electronics 2012**

Functional Prototype of Multiple Watchdog System Implemented in FPGA

Maroš Ďuriček
Faculty of Informatics and
Information Technologies,
Slovak University of Technology,
Ilkovičova 3,
842 16 Bratislava 4, Slovakia
maros.duricek@vmail.me

Mária Pohronská
Faculty of Informatics and
Information Technologies,
Slovak University of Technology,
Ilkovičova 3,
842 16 Bratislava 4, Slovakia
pohronska@fiit.stuba.sk

Tibor Krajčovič
Faculty of Informatics and
Information Technologies,
Slovak University of Technology,
Ilkovičova 3,
842 16 Bratislava 4, Slovakia
tkraj@fiit.stuba.sk

Abstract – Security of multi-process embedded systems is of a great concern in majority of embedded and real-time applications. Our work deals with a multiple hardware watchdog timer device that is intended to improve the security and reliability of multi-process systems. This paper deals with the implementation and testing of the prototype architecture using an FPGA device Spartan-3A. .

Keywords–watchdog; real-time; fault-tolerant; FPGA

I. INTRODUCTION

Fault-tolerant embedded systems are characterized by strict requirements to the safety of their operation. These requirements include guaranteed response time, continuous operation and self-diagnosis. A typical fault-tolerant or real-time embedded system running critical processes (e.g. automotive or industrial) is usually equipped with only one hardware watchdog timer which secures the whole system with all its concurrent processes running. Using such watchdog to supervise a number of parallel running processes is inefficient, as in this case there is only one software driver which emulates multiple watchdogs and actually operates the hardware watchdog timer. In the case of failure in one of the running processes, it is difficult to determine which process has just failed. And if the software driver is not running correctly, it is almost impossible. If the hardware watchdog fails or if some erroneous application resets the watchdog without knowing, all running processes will be not secured anymore.

A multiple hardware watchdog system with the ability to secure all processes and the whole embedded system independently can significantly increase the reliability of the real-time system. One of the basic characteristics of a watchdog is to “issue a hardware reset on time-out” [1]. The standard way for securing multiple processes of embedded systems is using software watchdogs. The software watchdog provides several virtual watchdog timers or other software mechanisms to securing concurrent processes, while utilizing only one hardware watchdog.

Our work deals with an existing architecture of multiple hardware watchdog system that has been

described in [2], adding several improvements and testing the implemented functional prototype on a Xilinx Spartan-3A device.

We use the main ideas of the existing multiple hardware watchdog architecture and adjust it to fulfill the limits of dedicated programmable hardware (FPGA) for correct synthesis and also to increase the modularity and functionality of the system. We have added architecture improvements such as reset of the complete secured system [3] or ability to communicate by other communication interfaces.

Section two describes the related work, especially the original multiple watchdog system design. In Section three, we are describing the functional overview of the multiple hardware watchdog system. Section four describes the architecture and its components. In the section five, we deal with the synthesis and debugging of the designed prototype. Section six describes performed experiments and obtained results. Last section concludes the document.

II. RELATED WORK

Several commercial watchdog extensions exist which can be connected to personal or industrial computers. However, these systems are working only with one or few hardware watchdog timers. The most commonly used communication interfaces for these purposes are serial port, peripheral component interconnect (PCI) and the keyboard port.

We base our work on the experimental multiple hardware watchdog system implemented in FPGA [2]. This system architecture is modular and easy to extend. It is using the standard serial port (UART) to communicate with the secured system. The basic counter unit is divided into two components. The first component is a small 24-bit watchdog timer; the second component is a logic which enables the timer. The main unit of the system is control unit which provides interface between the watchdog circuits and UART. Each watchdog is individually addressable from the control unit; and the addressing interconnection is cascaded to enable future extensions. The designed system requires a software driver to be used on the secured system side.

III. FUNCTION OVERVIEW

This section introduces the function of the designed system as a black box. In this paper we are using two terms to distinguish two different systems. The *secured system* is the embedded system running multiple processes. The *security system* is our designed system with multiple watchdogs.

Our system provides a number of hardware watchdog timers to secure each running process of an embedded system. Connection between the secured and the security system is established with the USB interface. The secured system is required to have a software driver that is able to communicate with the security system and is responsible for assigning one watchdog to each running process. The security system is able to load a new value to the specific addressable watchdog, to reset it or to correctly terminate it. Each process may have different requirements for the value that is loaded to the watchdog. This value has to be calculated and known to the secured system before the process is started. All provided watchdogs are 24-bit wide and count the time in milliseconds from 0 ms to 4.66 hours. This time interval is suitable for different requirements of the running processes.

When a watchdog timer overflows, the securing system sends an overflow message containing the particular address of the overflowed timer to the secured system. After receiving this message, the secured system should react appropriately while in general it has these possibilities:

- Restarting/restoring the failed process and restarting its watchdog with the same value.
- Restarting/restoring the failed process and loading a new (e.g. higher) value to the watchdog without the need of restarting it.
- Terminating the failed process correctly and free the allocated watchdog.

When more than one watchdog overflows, messages are sent successively.

In the case when all running and secured processes fail and the software watchdog driver (which also should have assigned a watchdog timer) fails too, our system has a capability to automatically detect the failure and reset the whole system after a given period of time. This delay time is configurable by one of the communication protocol messages. The delay is important because the secured system can have employed other mechanisms of failure detection and can be able to restart itself more safely without the assistance of the security system.

The hardware reset output of the multiple watchdog system has to be directly connected to the reset pin of the secured system and it has to have hardware feedback to know if the secured system has been restarted.

IV. IMPROVED SYSTEM ARCHITECTURE

We have made several improvements to the original architecture described in [2]. Only the

modular and hierarchical principle remains. Fig. 1 shows the system-level architecture.

A. Communicator

We are using one communicator block with parallel interface. This block is connected to the actual communication interfaces. It is responsible for decoding the received messages from the secured system, sending the acknowledgement and other messages of the communication protocol. Its function is to mediate between the control unit and the communication interfaces.

The communication protocol is designed to be minimalistic, because time is the critical parameter in this system and communication must be kept as simple as possible. On the other hand, we must ensure that the random code of a potential failed application could not reprogram the device. Tab. I. shows the communication messages of the protocol. Message *init* holds parameters for configuration of the control unit. Configuration parameters are the quantity of the priority watchdogs and the delay time in seconds. The priority watchdogs are watchdogs with higher priority. If these priority watchdogs all fail, the security system does not wait for other overflowing watchdogs and sends the warning and hardware reset. Priority watchdogs are not yet implemented in this prototype. The delay time is the time during the security system waits for the secured system self-reset. The *load* message is used for loading a new 24-bit value to the particular watchdog timer. Message *reset* restarts the particular watchdog. Messages sent to the secured system are *warning* before hardware reset with a delay value and *zero* indicating that a particular watchdog has overflowed. Each received byte of a message from secured system is acknowledged by the *ack* message, because of the small buffers in the control unit or communicator memory. This message indicates the security system response, too. If the security system does not answer any *ack* message to the messages sending from the secured system, this indicates that the multiple watchdog system has failed.

TABLE I. COMMUNICATION MESSAGES

Message	Operation Code	First 8 bits	Next 24 bits
INIT	0x2D	Priority watchdog quantity (5 b), Delay in sec. (3 b)	-
LOAD	0x3B	Address	Value (lower bytes first)
RESET	0x1A	Address	-
TERMINATE	0xF9	Address	-
WARNING	0x49	Delay in sec. (lower 3 b)	-
ZERO	0x52	Address	-
ACK	0x6E	-	-

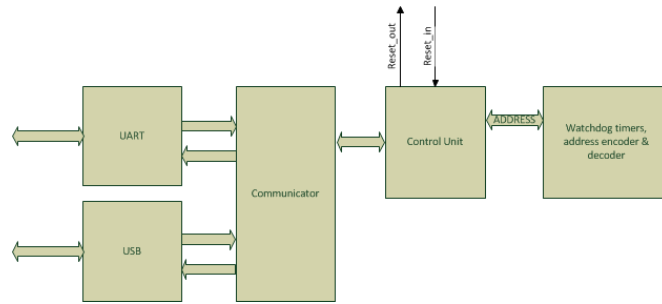


Figure 1. System architecture

B. Control unit

Control unit receives and generates the messages to communicator, controls all watchdogs, detects whether the secured system has failed and activates the hardware reset of the secured system when needed. The failure of the system requiring hardware reset is detected by two registers that compare the amount of enabled watchdogs and overflowed and not yet restarted watchdogs. If these two counters are equal and not zero, it is assumed that the secured system has failed.

The address bus utilized for addressing the watchdogs has been divided into two separate buses. The address output bus from the control unit is used when addressing a watchdog and e.g. loading the new value to it. The address input bus is used to read the address of the overflowed watchdog.

C. Watchdog

Each watchdog timer has only one clock input to communicate with the control unit. The loading value is sent from the control unit serially. An internal clock divisor contained in each watchdog timer is used for counting the time from the loaded value. The counting frequency is 1 kHz. It is simpler to have only one clock routed in FPGA to many blocks. A watchdog is interfaced to the control unit by the block *enabler* that enables the particular watchdog according to the address bus. This block has one register for saving the pausing value.

Watchdog timer has five basic states:

- *Running* state – the timer is counting from its loaded value.
- *Reset* state – the timer is being reset from its current value to the initial value.
- *Paused* state – indicates that the timer is not used.
- *Loading* state – the timer is loading the value from the control unit.
- *Overflow* state – indicates that the timer has overflowed.

The watchdog has one more signal (called *zero*) that is connected to the enabler. From each enabler is the signal *zero* connected to a multiplexer. The control unit can select and read one signal according to the

watchdog address. This signal is indicating that the specific watchdog has overflowed. When more than one watchdog overflows, the overflow signal has to fall back to inactive level to allow other watchdogs indicate their overflow state. Fig. 2 shows the connection of signal *zero*.

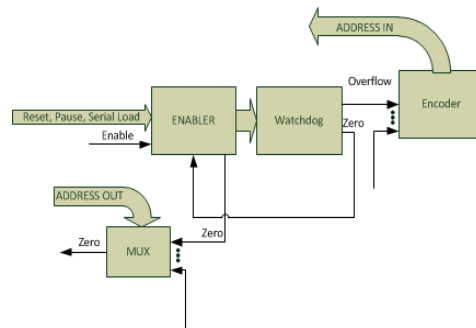


Figure 2. Signal *zero* connection

V. ARCHITECTURE SYNTHESIS AND DEBUGGING

The design has been implemented in VHDL and synthesized into a Xilinx Spartan-3A device, using the Xilinx ISE Webpack 13.4 software. Implementation of the system level was done in FPGA Advantage 8.1. We are using Avnet Spartan-3A Evaluation kit for the implementation. This device is equipped with PSoC which can translate UART signals from the FPGA to USB.

We started to synthesize smaller architecture with 15 watchdogs to inspect the possibilities of the FPGA circuit. Device utilization with this configuration was about 50% of the device. The present configuration has one UART interface, two interfaces communicating with USB circuit, one block with 16 watchdogs and second with 10 watchdogs. Firstly, the synthesis was started with optimistic 32 watchdogs in the device. Finally, we ended with 26 watchdog timers which have been Placed&Routed to the device, utilizing it almost completely. In each step we have manually removed one watchdog timer from the design. Device utilization after the Place&Route process is shown in Fig. 3.

We have used the Xilinx Platform cable and Xilinx ChipScope Pro software for debugging. One system

can be described in many ways in VHDL that are logically the same. But two descriptions of the same system can be synthesized differently. The pre-synthesis simulation is in this case not authoritative. Using the internal logic analyzer is a way to correctly view the inside signals states in the FPGA circuit. Before using the automatically generated block of ChipScope Pro, the clock signals have to be constrained in the user constraint file. Without these constraints, the system can be unstable and have incorrect timing after removing the block.

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	2,347	7,168	32%
Number of 4 input LUTs	6,163	7,168	85%
Number of occupied Slices	3,582	3,584	99%
Number of Slices containing only related logic	3,582	3,582	100%
Number of Slices containing unrelated logic	0	3,582	0%
Total Number of 4 input LUTs	6,622	7,168	92%
Number used as logic	6,163		
Number used as a route-thru	459		
Number of bonded IOBs	32	195	16%
Number of BUFMUXs	1	24	4%
Number of MULT18X18SIOs	1	20	5%
Average Fanout of Non-Clock Nets	3.38		

Figure 3. Device utilization of Spartan-3A

VI. TESTING

Two prototypes with different configurations and communication interfaces were subjects of the testing process. Firstly, we were testing the prototype with 15 watchdog timers and UART communication with 9600 baud rate. The UART interface was connected to the FDTI mini module FT2232H configured in UART mode. This was the first prototype where we have run the debugging process in order to remove the possible unexpected errors. Second configuration is system with 26 watchdogs. Communication interface UART is connected to the PSoC with 19200 baud rate. The second interface is connected through the I/O pins parallel to the FTDI module configured as asynchronous FIFO. In this configuration, we had to firstly debug the communication units connected to USB module. We were testing the system with two different communication interfaces that can be switched by user button on the device. In both cases the secured system was simulated on a common PC with Windows OS. The simulation was controlled manually with virtual com ports connected to serial terminal software able to send hexadecimal codes. In this purpose we were using the terminal RealTerm which is able to send data contained in files with configurable delay between bytes needed to simulate reading the *ack* messages.

We have proposed three scenarios of testing. These scenarios test the most critical aspects of the designed system at full load. In all scenarios we are using time values bigger than one second for easier time measuring. The testing was oriented to verify the work of watchdogs, control unit and correct communication. In the first scenario we were testing loading values and counting of the watchdog timers. We loaded 10 second values to all watchdogs successively. The system should send overflow messages in correct time. Fig. 4 shows this scenario testing in RealTerm application

with UART communication interface. The overflow messages are highlighted in red color. Delay between sending bytes is 500 ms. All watchdogs overflowed from the lowest to the highest address and the last message warns the system that it will be restarted in 7 seconds. The reset signal is for testing purposes only indicated by a LED on the device. In the second scenario we are testing the watchdog reset. We load a 1 minute value and after 30 seconds we send one reset message. In this time the timer should be once reset correctly and then overflow in one minute, the overflow message should thus arrive after 1 minute and 30 seconds. Third scenario is intended to test the correct termination of watchdog counting. We load 1 minute values in three watchdogs in the last addresses. Next, we reset the first watchdog and terminate the second. In this case the third watchdog should overflow as first, and the first watchdog as second. The second watchdog has to be in inactive state.

All scenarios worked correctly in both prototypes and both communication interfaces. We were measuring the time by ordinary clock in the precision of 1 second and all messages arrived in correct time.

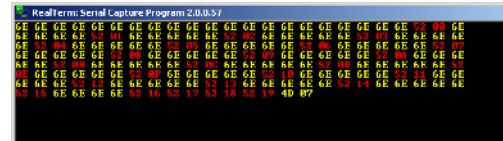


Figure 4. Testing in RealTerm application

VII. CONCLUSION AND FUTURE WORK

We have built the prototype of multiple hardware watchdog system. This system is after several tests fully functional and ready to use. It is able to secure at this time 26 running processes with different time value loaded into each watchdog. A software driver that is able to communicate to our system has to be installed on the secured system. This software driver should be fast, reliable and have abilities to control running processes on the operating system. In future work, we plan to reduce the communication latency. We also plan to synthesize the architecture with more watchdogs to bigger FPGA circuits. Possible system upgrades are adding the support of flash memory for storing the debug and diagnostic information, providing additional warning tools, e.g. sound warnings or mobile communication interfaces to warn the responsible personnel directly.

REFERENCES

- [1] Labrosse, J., Ganssle, J., Noergaard, T. et al.: Embedded Software: Know It All. Burlington: Elsevier, 2008. 770 s. ISBN 978-0-7506-8583-2.
- [2] Pohronská, M.; Krajčovič, T. (2011). FPGA Implementation of Multiple Hardware Watchdog Timers for Enhancing Real-Time Systems Security. In *Eurocon 2011 proceedings*. IEEE.
- [3] Ďuriček, M.; Pohronská, M.; Krajčovič, T.: A Hardware Reset Implementation in Multiple FPGA Watchdog System. In *International Conference of Cybernetics and Informatics Proceedings 2012*, STU Press.

E. Odoslaný článok na medzinárodnú konferenciu Applied Electronics 2012

F Obsah priloženého média

Priložené médium má nasledovnú štruktúru:

```
\
\dokument
\Avnet-Spartan-3A
\FPGA-Adv
\Xilinx-13.4
\seriove terminaly
\FTDI mini module
```

- Adresár **dokument** obsahuje elektronickú formu tohto dokumentu.
- Adresár **Avnet-Spartan-3A** obsahuje dokumentáciu a nástroje k vývojovej doske Avnet-Spartan-3A Evaluation Kit.
- Adresár **FPGA-Adv** obsahuje projekt s implementáciou pre návrhové prostredie FPGA Advantage 8.1.
- Adresár **Xilinx-13.4** obsahuje projekt s implementáciou pre návrhové prostredie Xilinx 13.4.
- Adresár **seriove terminaly** obsahuje inštalácie aplikácií pre sériovú komunikáciu použitých pri testovaní.
- Adresár **FTDI mini module** obsahuje dokumentáciu k použitému USB modulu a k obvodu *FT2232H* a aplikáciu *FT_Prog* pre konfiguráciu modulu.