VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

UNIVERZÁLNÍ PROGRAMÁTOR OBVODŮ S ROZHRANÍM JTAG

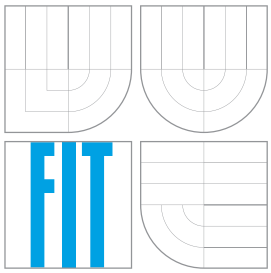
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ BARTEK

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

UNIVERZÁLNÍ PROGRAMÁTOR OBVODŮ S ROZHRANÍM JTAG

VERSATILE PROGRAMMER OF CIRCUITS WITH JTAG INTERFACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — LUKÁŠ BARTEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÁCLAV ŠIMEK
SUPERVISOR

BRNO 2011

Zadání diplomové práce/6463/2010/xbarte06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Bartek Lukáš, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Univerzální programátor obvodů s rozhraním JTAG**
Versatile Programmer of Components with JTAG Interface
Kategorie: Vestavěné systémy

Pokyny:

1. Prostudujte možnosti programování a testování elektronických součástek. Pozornost věnujte především rozhraní JTAG.
2. Seznamte se s principy komunikace po sériové sběrnici USB. Proveďte krátké srovnání dostupných USB řadičů.
3. Navrhněte blokové schéma systému implementujícího JTAG programátor na bázi FPGA a řadiče USB. Zvolte vhodné součástky pro jeho realizaci.
4. Vytvořte schéma zapojení programátoru a v návrhovém systému Eagle nebo KiCAD zrealizujte desku plošných spojů.
5. Naprogramujte jednoduchou obslužnou aplikaci pro PC a vytvořte potřebný firmware (pro FPGA i řadič USB sběrnice).
6. Prostudujte projekt OpenOCD. Pokuste se tento projekt upravit tak, aby byl kompatibilní s vámi navrženým systémem.
7. Funkčnost navrženého programátoru ověřte naprogramováním FPGA obvodu nebo MCU dle pokynů vedoucího.
8. Diskutujte možnosti dalšího rozšíření.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění bodů 1-3 zadání.

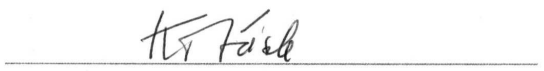
Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Šimek Václav, Ing.**, UPSY FIT VUT
Datum zadání: 20. září 2010
Datum odevzdání: 25. května 2011

L.S.



doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Cílem této diplomové práce je návrh a implementace univerzálního programátoru s rozhraním JTAG. Práce se zabývá hardwarovou i softwarovou částí programátoru. Teoretická část uvádí aktuální stav v používání standardů pro programování a testování elektronických součástek s důrazem na popis implementace JTAG. Další část popisuje programování obvodů ARM a FPGA přes JTAG rozhraní. V praktické části je vysvětleno, jak použít dostupný software pro programování těchto obvodů. Výsledným produktem této práce je vlastní programátor, skládající se z hardwarové části a obslužného softwaru. Na závěr jsou shrnuty možnosti pro budoucí vývoj a vylepšení vlastností.

Abstract

This master's thesis deals with designing and implementation of universal programmer with JTAG interface. The project consists of a hardware and software part. Theoretical part discusses actual state in using the standards for programming and testing electronic devices, with special emphasis on JTAG implementation. Next part deals with programming ARM and FPGA devices through JTAG. The programming of this devices using available software is described in the practical part of this document. Final product of this work is the programmer itself. The programmer consists of the hardware and supplement software. At the end of this thesis there is a conclusion about possible improvements and development in the future.

Klíčová slova

ARM, bitstream, FPGA, FTDI, FT2232, hraniční test, IEEE 1149.1, JTAG, MPSSE, OpenOCD, paměť flash, paměť NAND, SVF, USB

Keywords

ARM, bitstream, boundary scan, flash memory, FPGA, FTDI, FT2232, IEEE 1149.1, JTAG, MPSSE, NAND memory, OpenOCD, SVF, USB

Citace

Lukáš Bartek: Univerzální programátor obvodů
s rozhraním JTAG, diplomová práce, Brno, FIT VUT v Brně, 2011

Univerzální programátor obvodů s rozhraním JTAG

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Václava Šimka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Bartek
20. května 2011

Poděkování

Děkuji panu Ing. Václavu Šimkovi za konzultace, připomínky a návrhy ke zpracovávanému tématu.

Obsah

Kapitola 1

Úvod

S rozvojem schopností mikrokontrolerů se zvyšovala i náročnost vývoje aplikací na ně nasazovaných. První typy mikrokontrolerů byly programovány jednoúčelovými programátory, které vždy vyžadovaly přenos součástky z patice v aplikaci do patice v programátoru a po naprogramování zase zpět. To samozřejmě vývoj zdržovalo a vytvářelo dodatečné náklady při přechodu na jiný typ mikrokontroléru - bylo třeba nové vybavení pro vývoj. Výrobci reagovali vyvinutím součástek programovatelných přímo v zapojení, bez přenášení součástky mezi programátorem a aplikací. Každý z výrobců si vyvinul vlastní standard, který podporují jeho součástky. Tato situace trvá dodnes a není ideální. Souběžně však probíhal vývoj rozhraní pro testování součástek JTAG. Jakmile se toto rozhraní stalo dostatečně vyspělé a hlavně podporované většinou výrobců, ukázala se možnost využít jej mimo testování také k programování a debugování. Komplikovanější součástky předních výrobců již jsou tímto rozhraním vybaveny, nicméně programovací funkce nejsou ještě tak standardizovány jako testovací.

Tato práce se zabývá možností vytvoření univerzálního programátoru pro různé součástky s rozhraním JTAG od různých výrobců, přičemž by bylo uživateli poskytnuto jednotné rozhraní umožňující nahrát předem vytvořený soubor s konfigurací.

V kapitole 2 je přehled možností programování součástek od různých výrobců. Každý výrobce má obvykle svůj vlastní protokol a ty jsou zde stručně představeny.

V kapitole 3 je vysvětlena podstata JTAG rozhraní z pohledu implementace v hardwaru součástky a jsou popsány prvky tohoto rozhraní. Speciální pozornost je věnována možnosti použití JTAG pro programování pamětí součástek.

V kapitole 4 jsou popsány datové formáty, které se používají pro uložení JTAG příkazů na vyšší úrovni a také k popisu rozhraní součástky vybavené JTAG podporou.

V kapitole 5 jsou konkrétní ukázky použití JTAG. Jsou zde popsány adaptéry pro připojení součástky k počítači a software, používaný pro komunikaci po JTAG.

V kapitole 6 je stručný teoretický úvod do principu fungování USB sběrnice. Dále jsou zde rozebrány třídy USB řadičů a vysvětlena spolupráce mezi USB řadičem na straně hardwaru a operačním systémem.

V kapitole 7 jsou rozebrány možnosti realizace navrhovaného programátoru. Je zde uve-

den výběr součástek a blokové schéma možného i zvoleného zapojení.

V kapitole 8 uvedu použití čipu FT2232 firmy FTDI pro vytvoření převodníku USB na JTAG. Prakticky je zde ukázána instalace ovladače do operačního systému a programová obsluha čipu.

V kapitole 9 je uveden způsob komunikace po JTAG a programování pro obvody FPGA.

V kapitole 10 je uveden způsob komunikace po JTAG a programování pro obvody s jádrem ARM.

V kapitole 11 je prakticky popsán způsob nahrání konfigurace do FPGA pole pomocí navrženého programátoru a navrženého software.

V kapitole 12 se prakticky řeší způsob nahrání programu do mikrokontroléru s jádrem ARM za použití navrženého programátoru a programu OpenOCD.

V kapitole 13 je popis funkcí navrženého softwaru a použitý postup při jeho návrhu.

Kapitola 2

Možnosti programování a testování součástek

Každý z výrobců programovatelných obvodů vyvinul vlastní rozhraní umožňující programování čipu v zapojení (In-Circuit Programming). Tím byl vyřešen problém s nutností přemisťování čipu při vývoji. Dalším problémem však bylo ladění vyvíjené aplikace přímo v zapojení na DPS (desce plošných spojů). Dříve používané hardwarové emulátory procesorů (In-Circuit Emulator) byly příliš nákladné pro větší rozšíření. Nyní jsou i ty nahrazovány vestavěnou debugovací jednotkou přímo na čipu, která umožňuje real-time komunikaci mezi programem běžícím na procesoru a nadřazeným počítačem se spuštěným debugovacím prostředím.

2.1 Programovací a debugovací rozhraní

Tato práce je zaměřena na použití standardního rozhraní IEEE1149.1, nazývaného též JTAG, podle organizace, jenž ho standardizovala.[8] Kromě tohoto rozhraní existují i další rozhraní pro programování a ladění. Tato rozhraní však jsou proprietárními řešeními jednotlivých výrobců a jsou tedy dostupná jen v součástkách daného výrobce. Byla většinou zavedena dříve, než došlo k rozšíření JTAG a hlavním cílem bylo usnadnění programování. Stále se však používají, často jsou jednodušší než JTAG a tím i levnější. Spousta jednoduchých součástek by totiž ani nevyužila všechny možnosti poskytované JTAG.

ISP firmy Atmel

In-System Programming je sériové rozhraní, propojující řídící počítač a programovaný mikrokontrolér pomocí čtyř vodičů. Tři vodiče jsou standardní rozhraní SPI (Serial Peripheral Interface) a poslední vodič je RESET, sloužící pro inicializaci programované součástky. Přenos dat je synchronní, řízený hodinami na vodiči TCK. Data tečou od programátoru do součástky vodičem MOSI (Master Out Slave In) a ze součástky do programátoru vodičem MISO (Master In Slave Out). Součástí standardu ISP je identifikátor součástky, který je uložen v interní paměti součástky a lze ho programátorem přečíst. [5]

BDM firmy Freescale

Background Debug Mode je alternativa k tradičnímu ladění s použitím In-Circuit emulátorů. Oproti nim je však výrazně levnější. Obvod pro podporu ladění je totiž vestavěn

přímo do čipu součástky. Skládá se z Background Debug Controlleru (BDC) a debugovacího modulu (DBG). S okolím BDM komunikuje pomocí 6 pinového konektoru, na němž jsou využity 4 pozice (BKGD pin, RESET pin, GND pin, Ucc pin). Příkazy pro BDM se dělí do dvou skupin. Active Background Mode příkazy slouží pro přístup do registrů mikrokontroléru a jejich modifikaci. Při těchto operacích na mikrokontroléru nemůže běžet žádná další aplikace. Lze také zapnout a ovládat krokování CPU instrukcí. Non-Intrusive příkazy umožňují sledovat běžící aplikaci a vypisovat registry CPU. Také umožňují přístup do řídících registrů BDM obvodu. Řídící registry jsou celkem dva. Jeden slouží jako stavový a kontrolní registr, druhý pak k uložení adresy na níž má dojít k přerušení činnosti programu (breakpoint registr).
BDM je funkčně podobný JTAG, je však jednodušší co do podporovaných instrukcí a neumožňuje řetězení více součástek za sebe. [10]

BSL firmy Texas Instruments

Bootstrap Loader umožňuje komunikaci s pamětí vestavěnou v mikrokontrolérech řady MSP430, a to jak RAM paměťí, tak i flash paměťí. Na součástce existuje BSL a současně i JTAG rozhraní a nijak se neovlivňují. Nemohou však být používána současně. Pro zpřístupnění BSL musí být na určené piny aplikována výrobcem udaná posloupnost signálů. Komunikace probíhá přes UART (Universal Asynchronous Receiver Transmitter) protokol s nadřízeným počítačem. Jsou dostupné 2 sady příkazů, podle nutnosti zadávat či nezadávat před provedením heslo. Bez hesla je možné pouze provést kompletní výmaz paměti. Se zadáním hesla je možné číst a zapisovat bloky dat z paměti a do paměti, změnit PC (Program Counter) a spustit program od zadané adresy v paměti. [25]

ICSP firmy Microchip

In-Circuit Serial Programming je z uvedených rozhraní nejstarší. Slouží pouze pro programování flash pamětí mikrokontolérů PIC. Rozhraní je složeno z 5 pinů, obsahujících také napájení pro programované zařízení. Nevýhodou je malá rychlost programování, již pro 16Kb flash trvá operace cca 4 minuty. U nových součástek se ICSP nepoužívá. [20]

2.2 Vestavěné testování

BIST

Built-In Self Test (BIST) je technika spočívající v přidání dalších obvodů na čip součástky, které umožní provádět autonomní testování správné funkčnosti, např. při každém zapnutí napájení. Motivace zavedení je podobná jako u JTAG, tedy omezení používání ATE (Automated Test Equipment) stanic pro testování správné funkce obvodu. ATE jsou pro současné obvody s vysokou integrací velmi drahé na vývoj a dalším problémem je nedostupnost pinů na součástkách pro měřící hroty. BIST může být aplikována nejen na součástku, ale i celou DPS, potažmo i kompletní zařízení. Výhodou oproti ATE je také možnost testovat kdykoliv za provozu, nikoliv jen u výrobce. Další předností je rychlost testování. Naopak nevýhodou je možnost selhání samotného testovacího obvodu, kdy obvod hlásí chybu na bezchybném zařízení.

Existuje několik specializovaných verzí BIST, jako MBIST (Memory BIST), sloužící k testování pamětí nebo ABIST (Analog BIST) sloužící k testování analogových obvodů.

Kapitola 3

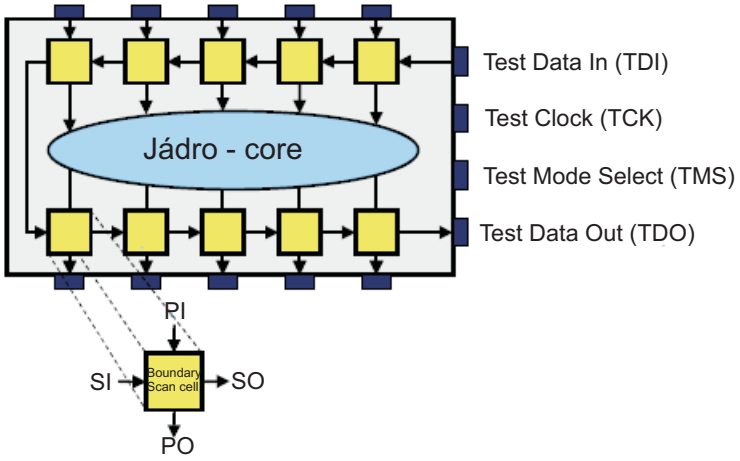
JTAG - specifikace a architektura

Přibližně do poloviny 70.let se při výrobě a osazování desek plošných spojů (DPS) používala technika „bed-of-nails“ pro testování kvality spojů na DPS a funkčnosti a správnosti zapojení elektronických součástek. Tento test se skládal ze dvou částí: Test propojení (power-off test) a test funkčnosti (power-on test). Power-off testuje integritu fyzických spojů mezi měřícími body na DPS. V této fázi se testují přerušené spoje a nežádoucí můstky mezi vodiči. Testování se provádí měřením odporu. Power-on test přikládá elektrické signály na zvolené body (vstupy součástek) a testuje odezvu (výstupy součástek). Tím lze zjistit přítomnost, orientaci a správné propojení osazených součástek. Uvedené testování závisí na fyzické dostupnosti měřících bodů na DPS. Tento přístup byl snadný ze strany pájecích bodů. Se zavedením technologie SMT (Surface Mount Technology) a z toho plynoucího osazování součástek na obě strany DPS, společně se zvýšenou hustotou osazené plochy na desce, se neúměrně zvyšují požadavky na přesnost zaměření sondy na měřící bod na DPS. Zavedení vícevrstvých DPS je další výraznou komplikací. Tato situace vyústila v polovině 80. let k vytvoření skupiny JETAG (Joint European Test Action Group), která byla složena ze zástupců významných evropských výrobců součástek. Cílem této skupiny bylo vytvoření standardu pro testování součástek. Později se do skupiny přidali výrobci z USA a skupina se přejmenovala na JTAG (Joint Test Action Group). [4]

3.1 IEEE 1149.1 - Boundary Scan

Každý vstupní a výstupní signál, tj. pin součástky, je obalen víceúčelovým paměťovým elementem nazývaným boundary-scan buňka. Vstupní buňky se nazývají input-cell, výstupní output-cell. Sada všech input-cell a output-cell je zapojena do série a vznikne posuvný registr, viz obrázek 3.1. Paralelní operace načtení, nazývaná capture, způsobí načtení hodnot ze vstupních pinů do input-cell a signály z jádra (core), tj. vlastní logiky součástky, budou zapsány do output-cell. Paralelní operace nazvaná update způsobí přenesení hodnoty z output-cell do výstupních pinů součástky a hodnoty uložené v input-cells budou načteny do core.

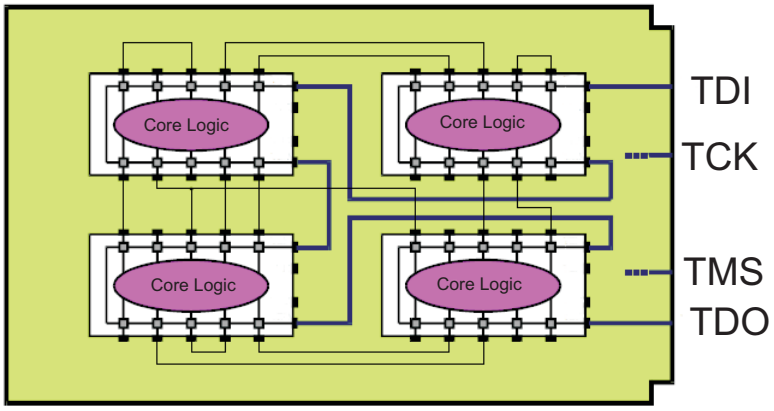
Data uložená v boundary scan buňkách mohou být díky zapojení jako posuvný registr sériově posouvána po obvodu součástky přes všechny buňky. Vstupním pinem je pin TDI (Test Data In) a výstupním je pin TDO (Test Data Out). Zdroj hodin pro registr a celkově pro řídící logiku je přiveden pinem TCK (Test Clock). Činnost prováděná interní testovací logikou je řízena změnou úrovní na řídícím pinu TMS (Test Mode Select). [24]



Obrázek 3.1: Rozmístění Boundary-scan registrů v pouzdře součástky. Podle [4]

Použití boundary scan registrů

Na úrovni součástky boundary scan registry nijak nezasahují do funkčnosti jádra (core). Jinak řečeno, možnost testování není ovlivněna aktuální činností jádra. Více součástek na desce podporujících JTAG může mít sériově řazený výstup TDO na TDI vstup následující součástky. Tím vznikne takzvaný testovací řetězec (boundary scan chain), viz obrázek 3.2. Řídicí vstup TMS a zdroj hodin TCK jsou připojeny ke všem testovaným součástkám paralelně.



Obrázek 3.2: Testovací řetězec na DPS. Podle [4]

Boundary scan (BS) řetězec byl původně určen na testování správné orientace, osazení a propojení jednotlivých čipů na desce. Použití BS jako prostředek pro test funkčnosti se objevilo až dodatečně. Součástky byly původně vkládány do DPS už změřené, to však se zaváděním složitých ASIC a FPGA obvodů začalo být zdlouhavé a drahé. Začal se tedy hledat způsob využití boundary scan registrů pro ověření funkčnosti součástky, tedy hlavně jádra. Součástka bývá nejčastěji nefunkční z důvodu poškození:

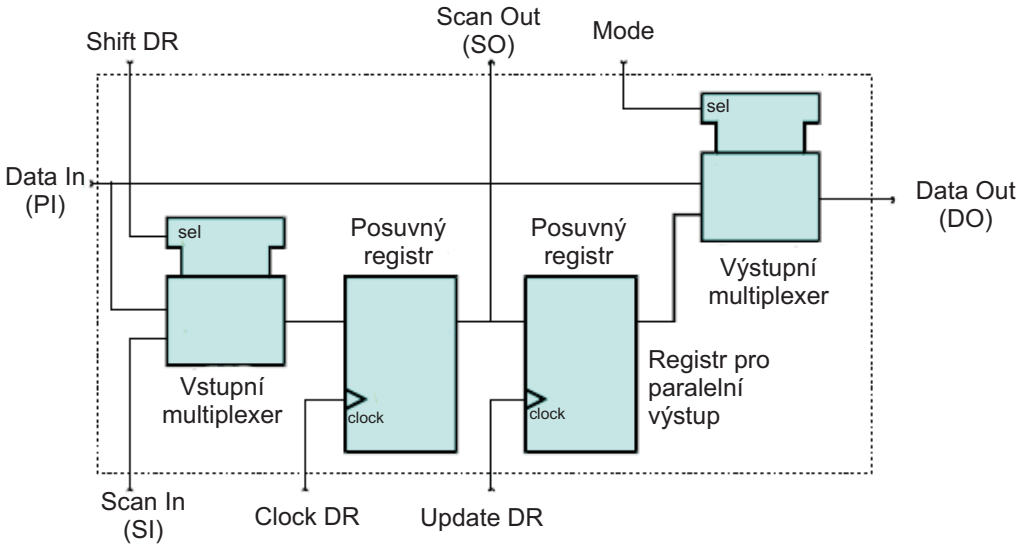
- elektrostatickým výbojem při manipulaci

- nadměrným teplem při pájení
- mechanickým poškozením (nárazem, pádem)

Při těchto poškozeních dojde obvykle ke změnám v těsném okolí součástky a v jejích vstupních obvodech. Samotné core nebývá poškozeno, aniž by současně nebyly poškozeny vstupní obvody. Stačí tedy testovat vstupní obvody (které právě obsahují boundary scan registry) a případně jednoduše zjistit funkci core. (Jde o jednoduché volání jádra typu „Jsi tam?“). Testování vstupních obvodů a core se nazývá Intest. Testování spojů mezi součástkami se nazývá Extest. [4]

Struktura boundary scan buňky

Každá buňka může pracovat ve 4 režimech: normal, update, capture a serial shift. Fyzicky může být buňka složena z klopných obvodů typu D a multiplexoru, viz obrázek 3.3. Nicméně norma IEEE 1149.1 neudává strukturu buňky, pouze její funkci. Při režimu normal jsou data ze vstupu Data_In přenášena přímo na výstup Data_Out. Při režimu Update je signál z Data_In směřován do registru a jeho hodnota je zachycena při hraně TCK. Během serial shift módu je Scan_Out jednoho registru poslán na Scan_In následujícího registru. Módy capture a serial shift nijak neovlivňují data vstupující a vystupující z/do součástky přes piny. To umožňuje real time (za běhu) provádět monitorování funkce obvodu. Proto je časování boundary scan samostatné a nezávislé na časování jádra. [4]



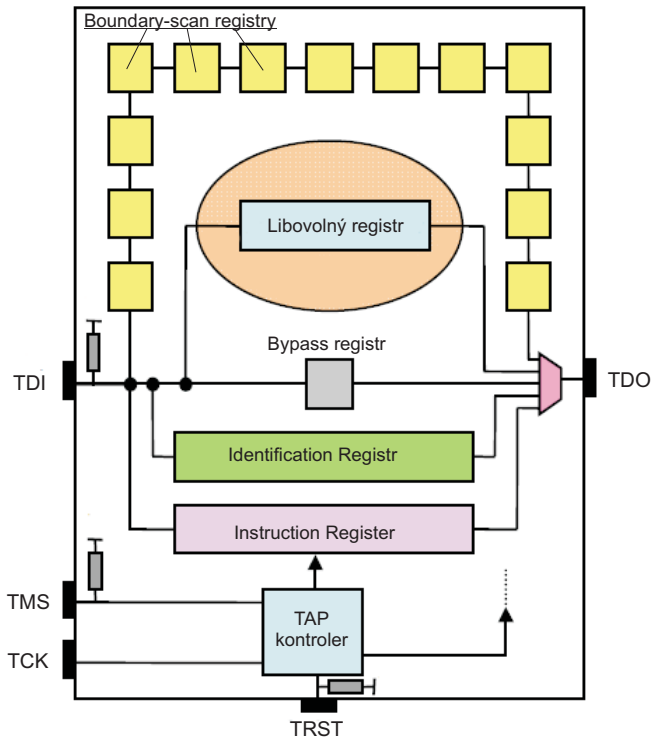
Obrázek 3.3: Struktura jedné boundary-cell buňky. Podle [4]

Architektura součástky s JTAG

JTAG zařízení se skládá z těchto částí (viz obrázek 3.4) :

- 4 vyhrazené piny TDI (Test Data In), TMS (Test Mode Select), TCK (Test Clock), TDO (Test Data Out). Dále pak volitelný pin TRST (TAP Reset). Tyto piny jsou společně označovány jako TAP (Test Access Port).

- boundary scan buňky na vstupních a výstupních pinech součástky
- konečný automat pro řízení testovací logiky
- IR registr uchovávající aktuální prováděnou instrukci
- 1 bitový bypass registr
- obvykle IDCODE registr, obsahující identifikační číslo součástky uložené výrobcem
- další volitelné registry závisející na výrobci



Obrázek 3.4: Rozmístění JTAG komponent v pouzdru součástky. Podle [4]

V jednom okamžiku může být pouze jeden registr propojen mezi TDI a TDO (např. Bypass, IR, ...). Zvolený registr je vybrán na základě dekódování instrukce nahrané do IR. Některé instrukce jsou povinné dle normy (např. Extest), jiné volitelné (např. IDCode). [4]

Instrukční registr

Skládá se z posuvného registru, který může být zapojen mezi TDI a TDO a dále z paměti uchovávající poslední instrukci. Mezi těmito dvěma částmi může být dekódovací logika v závislosti na šířce registru a počtu podporovaných instrukcí. Řídící signál přichází do IR z TAP kontroléru a určuje mód zpracování dat. Dalším řídícím příkazem může být načtení dat z IR registru do paměti IR (update). IR musí být nejméně dvoubitový, aby mohl obsáhnout 4 povinné instrukce (Bypass, Sample, Preload, Extest). [24]

Instrukce IR

Celkem 4 instrukce jsou povinné podle IEEE 1149.1. Bypass instrukce je určena samými jedničkami, nezávisle na délce IR. Po načtení tato instrukce způsobí vložení Bypass registru mezi piny TDI a TDO. Podle definice by paměť IR měla po resetu obsahovat instrukci Bypass. Pouze v případě, že součástka podporuje instrukci IDCode, by měla být výchozí instrukce tato. Obě instrukce Sample a Preload zařazují mezi TDI a TDO boundary-scan registry. Instrukce Sample nastaví buňky na zachycení dat vstupujících do součástky. Operace Preload nastaví zadané hodnoty na piny vystupující ze součástky. Kódy pro tyto operace nejsou normou určeny. Instrukce Extest připraví boundary-scan buňky na test konektivity mezi součástkami. Dále existuje množství volitelných instrukcí, jako: [8]

- Intest: připraví boundary-scan buňky na test core (self-test)
- IDCode: připojí registr ID s uloženým kódem součástky mezi TDI a TDO
- Runbist: inicializuje vestavěnou samotestovací (self-test) rutinu, která otestuje součástku podle postupu určeného výrobcem a do registru mezi TDI a TDO uloží výsledek (pass/fail).
- Clamp: Nastaví výstupní piny do zvolené úrovně (0,1) a poté zařadí mezi TDI a TDO Bypass registr. Slouží pro nastavení bezpečných hodnot na výstupech při testování ostatních součástek (např. odpojení od společné sběrnice).
- Highz: Podobné jako Clamp, ale všechny výstupy jsou odpojeny (nastaveny do stavu vysoké impedance, nebo jako vstup pokud není podpora stavu vysoké impedance).

Každá operace může mít více než 1 kód a neznámé operace by měly být interpretovány jako Bypass. Výrobce může přidat (a obvykle to dělá) vlastní instrukce s vlastními instrukčními kódy. [24]

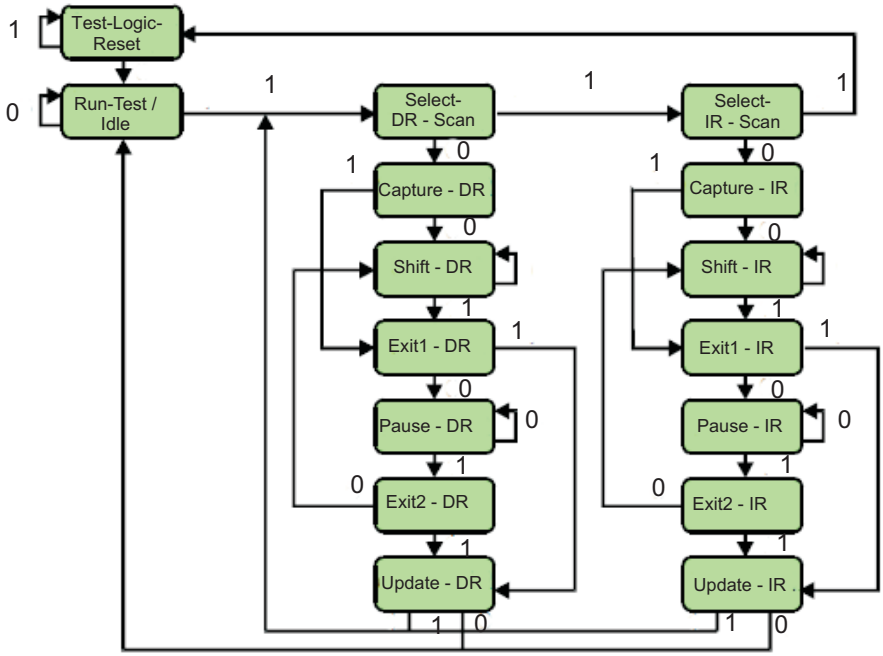
TAP kontroler

TAP kontroler představuje vlastní řídící jádro implementace JTAG. Jedná se o stavový automat, pevně daný zapojením logiky obvodu. Jeho vstupem jsou řídící signály JTAG sběrnice, tedy TMS a TCK. Jeho výstupem jsou vnitřní řídící signály, které slouží ke spínání multiplexorů. Tyto multiplexory poté připojují jednotlivé JTAG registry mezi vývody TDI a TDO. TAP také dodává řídící impulsy do posuvných registrů. Na obrázku 3.5 jsou znázorněny přechody mezi stavy TAP automatu. Výchozí stav po resetu či připojení napájení je Test-Reset Logic. V tomto stavu neprobíhá žádná aktivita. Pro přechod do dalších stavů je třeba přivádět impulsy na vstup TCK a dále logické úrovně 0 nebo 1 na vstup TMS. Právě tyto úrovně určují přechod mezi stavy automatu. Na obrázku jsou vedle hran pro každý přechod uvedeny stavy, které musí být přítomny na vstupu TMS. Při příchodu hodinového impulsu na TCK je podle aktuálního stavu TMS proveden odpovídající přechod. Zajímavě byl vyřešen požadavek na přechod do počátečního stavu Test-Reset Logic. Namísto dalšího signálu stačí držet TMS na úrovni 1 po dobu nejméně 5 impulsů TCK. Tato vlastnost je zřejmá z diagramu přechodů. (Přesto tento signál pro reset TAPu bývá někdy vyveden. Je značen TRST.) Stavový automat má dvě souměrné větve. První se týká obsluhy datového registru, druhá pak instrukčního registru. Datových registrů je na součástce více, v danou chvíli je však pouze jeden zvolen jako aktivní. Naopak instrukční registr

je pouze jeden. Pomocí instrukcí do něj zapisovaných se mimo jiné vybírá aktivní datový registr.

V obou větvích jsou hlavními instrukcemi Capture-DR (resp. Capture-IR) a Update-DR (resp. Update-IR). Ve stavu Capture-xR je možné do daného registru zapisovat bity, vsouvané do zařízení po vodiči TDI. Současně se na výstup TDO vysouvají bity doposud uložené v příslušném registru. Zde tedy probíhá vlastní výměna dat mezi čipem a okolím. Stav Update-xR způsobí, že změny zapsané do registru se vezmou v úvahu. Během zápisu jsou totiž data ukládána do pomocného registru a teprve příkazem Update-xR se skutečně promítnou do stavu součástky.

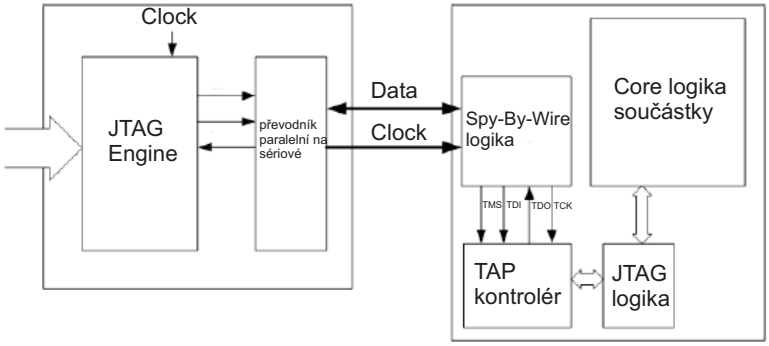
Stavy Capture-DR (resp. Capture-IR) mají různý význam v závislosti na vybraném registru. Při vybraném Boundary-Scan registru tak dojde k načtení hodnot na pinech součástky, zatímco například při vybraném registru IDCode dojde k načtení pevně uložené hodnoty, udávající identifikátor výrobce a součástky. [24]



Obrázek 3.5: Model TAP kontroleru jako stavový automat. Podle [4]

Spy-By-Wire, sériový JTAG

Spy-By-Wire je sériová varianta protokolu JTAG vyvinutá firmou Texas Instruments. Protokol je stejný jako u JTAG, pouze se pro spojení používají jen 2 vodiče. Jeden vodič (SBWTCK) slouží pro přenos hodinového signálu, na druhý vodič (SBWTDIO) jsou multiplexována data z TMS, TDI a TDO. Na obou koncích spojení je tedy multiplexer/ demultiplexer. Nevýhoda je trojnásobné zpomalení přenosu vůči standardnímu JTAG. Výhodou je úspora pinů na pouzdře součástky. Princip činnosti je na obr. 3.6. [23]



Obrázek 3.6: Převod JTAG na Spy-By-Wire. Podle [23]

3.2 IEEE 1532 - In-Circuit Configuration

Po zavedení JTAG Boundary Scan začali výrobci konfigurovatelných součástek uvažovat o využití možností JTAG také pro programování již osazených prvků. Toto programování je možné pomocí přístupu k dodatečně (každým výrobcem) přidaným datovým a řídícím registrům. Tímto způsobem tak lze programovat i součástky odpovídající pouze standardu IEEE 1149.1. Problémem se však opět stala rozdílná implementace programování u různých výrobců, protože touto oblastí se norma IEEE 1149.1 nezabývá. To způsobuje, že JTAG v současné podobě podle normy IEEE 1149.1 nemá standardní přístup k programování. Řešením v blízké budoucnosti by mělo být používání normy IEEE 1532, speciálně vyvinuté pro standardizaci procesu programování a všech ostatních operací spočívajících v manipulaci s vestavěnou pamětí (konfigurace, rekonfigurace, zpětné čtení, verifikace, mazání, zabezpečení obsahu).

Pro podporu IEEE 1532 je požadován standardní BSDL soubor. Oproti IEEE 1149.1 však v něm přibude další sekce (tzv. BSDL rozšíření). BSDL soubor je nedílnou součástí součástky s podporou IEEE 1532 a je dodáván výrobcem. Další změnou je přidání souboru typu ISC, který obsahuje vlastní konfigurační data. ISC je vytvořen návrhovým softwarem pro ASIC či FPGA obvody v poslední fázi návrhu.

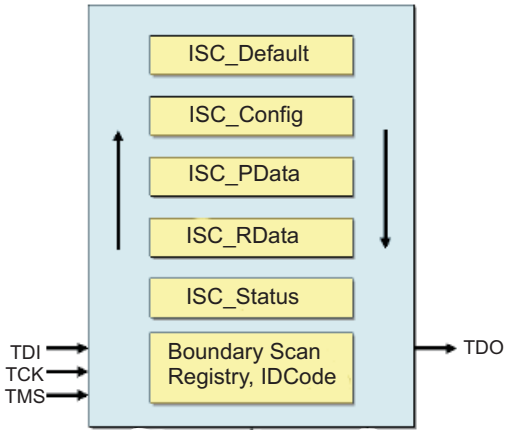
Tyto dva soubory přijdou na vstup 1532Playeru, který provede programování pomocí standardní sběrnice JTAG. Konkrétním příkladem 1532Playeru je J Drive od firmy Xilinx. Xilinx podporuje IEEE 1532 ve všech svých FPGA počínaje Spartan-3. Podpora ostatních výrobců není zatím velká, postupně se však zvedá. [3]

Architektura

Standard IEEE 1532 vychází z IEEE 1149.1 a rozširuje jej. Přidává další povinné a volitelné interní registry společně s novými instrukcemi pro programování. Na obrázku 3.7 je znázorněna základní architektura. Byly přidány 2 interní řídící signály ISC_Enabled a ISC_Done. Předpona ISC_ znamená In-System Configuration. ISC_Enabled indikuje, zda je zařízení připraveno k programování. ISC_Done indikuje výsledek programování, tedy zda bylo úspěšné nebo ne.

Následující registry byly přidány jako povinné:

- ISC_Default - pasivní registr pouze pro vyhovění normě IEEE1149.1. Obvykle se namísto něj použije BYPASS registr



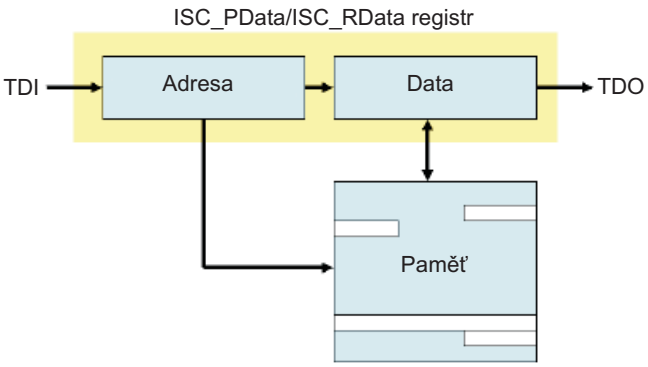
Obrázek 3.7: Registry architektury IEEE 1532. Podle [4]

- ISC_PData - obsahuje adresu paměti nebo registru pro zápis
- ISC_RData - obsahuje přečtená data nebo zapisovaná data

Následující registry byly přidány jako volitelné:

- ISC_Config - uložení parametrů použitých při konfiguraci
- ISC_Status - 2 bitový registr, indikující stav právě prováděné instrukce (programování probíhá, programování ukončeno, chyba)
- ISC_Sector - určuje vybraný paměťový sektor
- ISC_Info - obsahuje výrobcem přednastavené hodnoty nebo aktuální stav
- ISC_Inc - umožňuje inkrementaci adres během programování

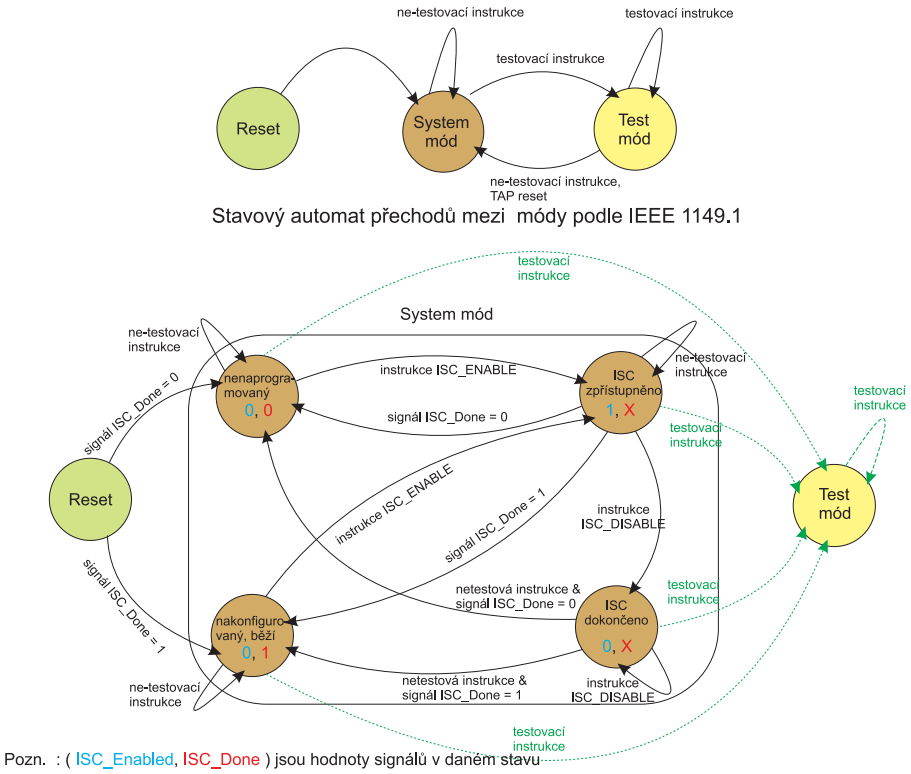
Registrový pár adresa/data umožňuje v jednom kroku provést zápis či čtení ze zadané adresy v paměti, viz obrázek 3.8. Registry ISC_PData a ISC_RData jsou v tomto kroku zřetězeny za sebou a zapojeny mezi piny TDI a TDO, viz obrázek 3.8. [3]



Obrázek 3.8: Registrový pár adresa/data. Podle [4]

Současně s přidáním nových registrů byl rozšířen i repertoár instrukcí o nové, které tyto registry používají. Povinné instrukce z IEEE 1149.1 jsou všechny zachovány, navíc se staly povinnými IDCODE a USERCODE. Instrukce pro práci s pamětí jsou ISC_ENABLE (povolí ISC prostředí, nastaví signál ISC_Enabled na 1), ISC_DISABLE (zakáže ISC prostředí, nastaví signál ISC_Enabled na 0), ISC_PROGRAM (přistoupí k paměťovému místu použitím registru ISC_PData). Volitelně jsou podporovány instrukce například pro mazání bloků paměti či pro nastavení zabezpečení uložených dat. Je také povolené přidávat specifické instrukce jednotlivými výrobci.

Přestože struktura TAP automatu je pro oba standardy stejná, interně se řídící logika může nacházet v rozdílných stavech (módech). V případě IEEE 1149.1 existují pouze dva módy, dělené podle druhu vykonávané instrukce. Systémový mód (zpracování ne-testovacích instrukcí) a testovací mód (zpracování instrukcí pro testování). Dle normy IEEE 1532 se módy také dělí na systémový a testovací, přičemž systémový má další čtyři podstavy, kterými se prochází v závislosti na průběhu programování. Test mode je stejný jako u IEEE 1149.1 a zahrnuje používání všech původních testovacích instrukcí. Přechody mezi stavy při programování jsou řízeny instrukcemi ISC_ENABLE, ISC_DISABLE a vnitřními signály ISC_Enabled a ISC_Done (obrázek 3.9).



Obrázek 3.9: Operační módy řídící logiky. Podle [16]

Kapitola 4

JTAG - datové formáty

Pro podporu JTAG byly vytvořeny standardizované formáty pro ukládání dat pro testování a programování součástek. Ovládání JTAG sběrnice přístupem na jednotlivé piny je vhodné pouze pro ladění, pro praktické nasazení při výrobě či testování je nepoužitelné. Proto byly vytvořeny příkazy na vyšší úrovni a datové formáty pro jejich uložení.

4.1 Boundary Scan Description Language - BSDL

BSDL je jazyk pro popis implementace IEEE 1149.1 v součástce. Je podmnožinou jazyka VHDL. Hlavním cílem BSDL je umožnit automatizaci testovací procedury. Vývojové nástroje mohou komunikovat s TAP kontrolerem a přistupovat do jednotlivých registrů a na I/O rozhraní součástky, k čemuž potřebují znát její architekturu. BSDL soubory bývají poskytovány přímo výrobci příslušné součástky. Deskriptor součástky v BSDL je složen z těchto elementů: [4]

- popis entity (typ součástky)
- generické parametry (výchozí hodnoty, např. typ pouzdra)
- popis logického rozhraní (názvy rozhraní a přiřazení směru dat)
- mapování pinů (přirazení fyzických pinů signálům)
- identifikace pinů TAP (rozložení na pinech součástky)
- popis instrukčního registru (podporované instrukce a jejich bitové reprezentace)
- popis Boundary-Scan registru (přirazení směru toku dat na pinech)

Popis součástky v jazyce BSDL je v textovém ASCII souboru, obvykle s koncovkou .bsd. Tento soubor je požadován pro testery a programátory součástek. Nejdůležitější částí je přirazení kódů k JTAG instrukcím. Následuje ukázka z BSDL souboru pro Spartan XC3S50:

```
attribute INSTRUCTION_OPCODE of XC3S50 : entity is
        "EXTEST    (000000)," &
        "SAMPLE    (000001)," &
        "INTEST    (000111)," &
        "USERCODE  (001000)," &
        "IDCODE    (001001)," &
        "HIGHZ     (001010)," &
        "JPROGRAM  (001011)," &
        "JSTART    (001100)," &
        "JSHUTDOWN (001101)," &
        "BYPASS    (111111)," ;
```

4.2 Serial Vector Format - SVF

SVF je standard pro zápis testovacích postupů, určujících stimuly, dat pro tyto stimuly a očekávaných reakcí pro testování zařízení vyhovujících normě IEEE 1149.1. Jde o standardní textový soubor v kódování ASCII. SVF se používá během celého životního cyklu zařízení, od počáteční verifikace návrhu až po servisní operace a diagnostiku. Boundary-Scan testování je prováděno sekvencí signálů přiváděných na TAP rozhraní. SVF obsahuje příkazy pro změnu stavu TAP automatu. Pro přechod do jiného stavu se nepoužívá sekvence signálů nutná pro přechod do nového stavu, ale pouze uvedení názvu tohoto stavu. Posloupnost signálů nutná pro přechod do nového stavu je již generována zpracovávajícím programem (SVF Player). [2] SVF formát je case sensitive (záleží na velikosti písmen), každý řádek je ukončen středníkem a řádky začínající výkřičníkem (!) nebo dvěma lomítky (//) jsou komentáře.

Základní příkazy SVF jsou SIR, SDR, RUNTEST a STATE.

SIR (Scan Instruction Register)
`SIR length TDI (tdi) SMARK (smask) [TDO (tdo) MARK (mask)]`
length: udává počet bitů, zapisovaných do IR registru ve stavu Shift-IR
tdi: bitová posloupnost (vzor), vkládaná do IR
smask: bitová maska, udávající významové bity zapisované hodnoty
tdo: udává očekávaný výstup z TDO (vysouvané bity z IR)
mask: bitová maska, udávající významové bity čtené hodnoty

SDR (Scan Data Register)
`SDR length TDI (tdi) SMARK (smask) [TDO (tdo) MARK (mask)]`
- význam parametrů stejný jako u SIR, pouze se pracuje s aktuálně zvoleným DR (datovým registrem).

RUNTEST
`RUNTEST run_count TCK`
run_count: počet cyklů, po které TAP automat zůstává ve stavu Run-Test. Používá se jako čekací smyčka v algoritmech přístupu k JTAG sběrnici (např. při programování přes JTAG). Doba čekání mezi operacemi je dána časovým diagramem výrobce zažízení.

STATE
`STATE tap_state`
tap_state: udává nový stav TAP automatu

Na následujícím příkladě, viz tabulku 4.1, je zřetelně vidět výhoda pro uživatele, plynoucí z používání zápisu přístupu na JTAG ve formátu SVF, namísto přístupu na JTAG s využitím nízkoúrovňových operací. Příklad spočívá v získání identifikátoru JTAG zařízení z registru IDCODE. (Instrukční registr je zde 8 bitový a instrukce pro připojení IDCODE registru je 0xFEh. Získaný IDCode není v příkladu uveden celý, pro svou velkou délku. Princip je však zřejmý z prvních bitů a posledního získaného bitu).

SVF soubory jsou používané pro programování a testování FPGA, EEPROM a CPLD zařízení, kde je vhodné odstínit uživatele od nízkoúrovňového přístupu na sběrnici.

4.3 Xilinx Serial Vector Format - XSVF

XSVF je na rozdíl od SVF binárním formátem, sloužícím však ke stejnému účelu jako SVF. XSVF byl vytvořen firmou Xilinx za účelem zmenšení velikosti souboru a zrychlení parsování. Příkazy jsou tvořeny 1 bytovými čísly, za nimiž následuje jeden či více parametrů. Standardní SVF je možné převádět na XSVF pomocí utility svf2xsvf od Xilinxu. [30] Opačný převod tato utilita neumí, nicméně obecně je možný.

Základní příkazy SVF jsou XSIR, XSDR, XRUNTEST a XSTATE. Jejich význam je stejný, jako odpovídající příkazy SVF formátu.
XSVF jsou především určeny pro použití ve vestavěných systémech, kdy je konfigurační soubor uložen přímo na desce zařízení v EEPROM paměti. Pak mohou být konfigurační data snadno zpracovávána mikroprocesorem. Naopak se tento formát nehodí pro použití přímo uživatelem.

4.4 Standard Test And Programming Language - STAPL

STAPL byl vyvinut v 90. letech k překonání omezení formátu SVF. Vznikl rozšířením proprietárního formátu JAM firmy Altera o řízení toku programu jazykem typu BASIC. STAPL byl standardizován institucí JEDEC. [24] Základní struktura STAPL programu je složena z:

- sekce NOTE (obsahuje textové řetězce obsahující informační texty)
- sekce ACTION (prováděná činnost, např. smaž, verifikuj, programuj)
- sekce PROCEDURE a DATA (příkazy a data pro prováděnou činnost)
- sekce CRC (záznam obsahující Cyclic Redundance Code celého souboru)

STAPL Composer a Player

STAPL program je vytvářen pomocí nástroje obecně nazývaného Composer. Může jít o samostatnou utilitu nebo součást integrovaného vývojového prostředí. STAPL program může být prováděn pomocí interpretu nebo může být zkompilován pro zrychlení zpracování. Obě možnosti musí podporovat nástroj zvaný STAPL Player. Player umí provádět tyto činnosti: [24]

Krok	Aktuální stav TAP	Další stav TAP	TDI	TMS	Poznámka
1.1	TLR	RTI	X	0	stav TAP Reset
1.2	RTI	Select-DR-Scan	X	1	
1.3	Select-DR-Scan	Select-IR-Scan	X	1	
1.4	Select-IR-Scan	Capture-IR	X	0	
1.5	Capture-IR	Shift-IR	X	0	
1.6	Shift-IR	Shift-IR	0	0	Nasunutí LSB bitu d0
1.7	Shift-IR	Shift-IR	1	0	Nasunutí d1
1.8	Shift-IR	Shift-IR	1	0	Nasunutí d2
1.9	Shift-IR	Shift-IR	1	0	Nasunutí d3
1.10	Shift-IR	Shift-IR	1	0	Nasunutí d4
1.11	Shift-IR	Shift-IR	1	0	Nasunutí d5
1.12	Shift-IR	Shift-IR	1	0	Nasunutí d6
1.13	Shift-IR	Exit1-IR	1	1	Nasunutí d7 při přechodu do stavu Exit1-IR
1.14	Exit1-IR	Update-IR	X	1	IDCODE instrukce (0xFE) je nyní zpracována
1.15	Update-IR	Select-DR-Scan	X	1	IDCODE registr je nyní připojen mezi TDI a TDO
1.16	Select-DR-Scan	Capture-DR	X	0	
1.17	Capture-DR	Shift-DR	X	0	
1.18	Shift-DR	Shift-DR	0	0	Vysunutí prvního bitu IDCODE na TDO
1.19	Shift-DR	Shift-DR	0	0	Vysunutí druhého bitu IDCODE na TDO
1.20	Shift-DR	Shift-DR	0	0	opakovat 29 krát
1.21	Shift-DR	Exit1-DR	0	1	Vysunutí posledního bitu IDCODE na TDO při přechodu do stavu Exit1-DR
1.22	Exit1-DR	Update-DR	X	1	
1.23	Update-DR	RTI	X	0	Návrat do stavu Run-Test-Idle

Tabulka 4.1: Posloupnost nízkoúrovňových instrukcí pro získání IDCODE

Krok	SVF Syntaxe	Popis činnosti
2.1	SIR 8 TDI (fe) SMASK (ff);	Přesun instrukce IDCODE do IR
2.2	SDR 32 TDI (00000000) TDO (f9604093) SMASK (ffffffff) TDO (f9604093) MASK (0fffffff)	Přesun IDCODE z Data Registru na TDO

Tabulka 4.2: Posloupnost SVF instrukcí pro získání IDCODE

- spustit STAPL soubor
- zpracovat sekci ACTION a PROCEDURE
- vhodně zobrazit data se sekce NOTE
- ověřit CRC součet
- přistupovat k fyzickým signálům na TAP kontroléru
- vytvářet real-time zpoždění pro správné časování signálů
- přistupovat k dalším signálům TAP kontroléru (mimo standardních, např. k TRST)

4.5 Hierarchical Scan Description Language - HSDL

HSDL byl navržen firmou Texas Instruments jako doplněk k BSDL a používá stejnou podmnožinu jazyka VHDL. HSDL pokračuje tam, kde BSDL končí - tedy na hranici mezi součástkou a DPS. HSDL popisuje, jak jsou jednotlivé součástky, podporující IEEE1149.1 propojeny na úrovni celé DPS. Obsahuje tedy popis propojovací sběrnice mezi obvody. [24] Hlavním důvodem vzniku bylo umožnění víceúrovňového pohledu na testované zařízení. Pokud použijeme BSDL, vzniká pro více součástek v testovacím řetězci problém s rozlišením, která data patří odpovídající součástce. Do testovacího řetězce vstupuje proud bitů a jiný proud bitů z něj vystupuje. HSDL umožňuje přehledné zobrazení a testování vložených testovacích řetězců. Příkladem může být základní deska elektronického zařízení, na níž se vyskytují obvody vybavené JTAG rozhraním. Ty jsou propojeny do testovacího řetězce. Na této základní desce pak je konektor pro rozšiřující desku. Do tohoto konektoru je připojena deska, která také obsahuje součástky podporující JTAG rozhraní. HSDL rozlišuje tuto hierarchie JTAG skenovacích řetězců, zatímco při použití BSDL vidíme pouze jednorozměrný řetězec všech JTAG součástek.

Kapitola 5

Programování součástek přes JTAG sběrnici

5.1 Adaptéry pro připojení JTAG zařízení

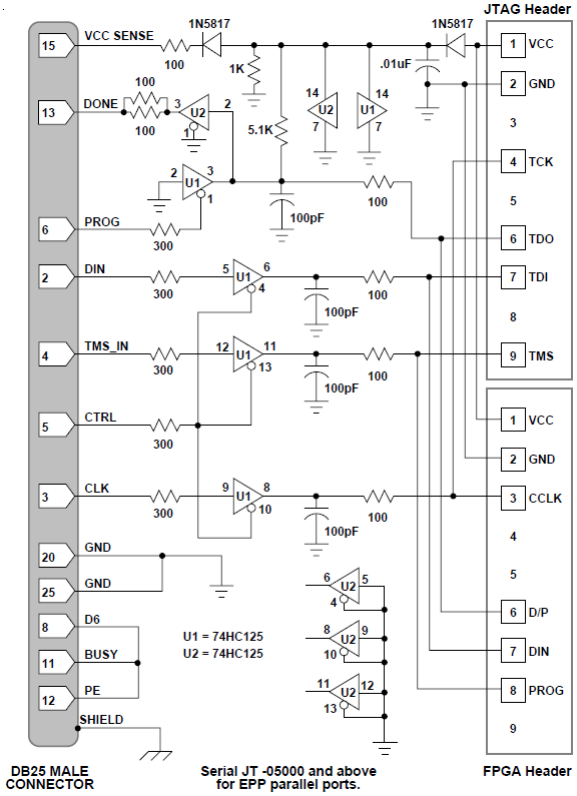
Uvažujme připojení DPS obsahující rozhraní JTAG k počítači typu PC. Vlastní propojující adaptér slouží především jako převodník napěťových úrovní mezi počítačem a DPS. První varianty těchto adaptérů byly připojovány přes paralelní port. S nástupem USB sběrnice se přešlo na USB adaptéry a to především z důvodu rychlosti a také nedostupnosti paralelního portu na nových počítačích.

Existuje spousta variant zapojení adaptéru, přičemž jedinou standardní částí je rozhraní směrem k zařízení, tj. výstupy TDI, TDO, TCK a TMS. Napěťové úrovně však mohou být různé, lišící se podle výrobce JTAG součástky. K ovládání adaptéru je nutné použít ovládací program, který převádí standardní JTAG povely na signály portu obsluhující elektroniku adaptéru. Kvůli možnosti určité univerzálnosti se některá zapojení adaptérů ustálila jako standard. Jejich zapojení je dostupné a při jejich použití máme k dipozici všechny programy, které deklarují shodu s tímto adaptérem. Tyto standardní adaptéry jsou však bohužel dostupné pouze pro paralelní port. Pro USB existuje také množství adaptérů, jejich zapojení však obvykle není k dispozici.

Pro paralelní rozhraní jsou jako standardy brány např. adaptéry Macraigor Wiggler JTAG Cable, Xilinx DLC5 JTAG Parallel Cable a Altera ByteBlaster. Jsou i další, nicméně všechny jsou založeny na stejném principu, který je zřejmý z obr. 5.1, na němž je adaptér Xilinx DLC5 JTAG Parallel Cable. Vlastní elektronika je tvořena obvodem 74HC125, což je 8 násobný oddělovač sběrnice. Jiné adaptéry používají stejný (či podobný) obvod, ale mají jinak propojené piny na paralelním portu, případně další malé změny v zapojení.

Uvedené adaptéry poskytují pouze standardní povinné signály JTAG rozhraní. Existují však výrobci, kteří na piny svých součástek vyvádějí i další signály, jako je TCLK, TRST, SRST či TST. Takové adaptéry pak jsou specifické pro danou řadu součástek. Na obrázku 5.2 je uveden adaptér pro mikrokontroléry MSP430 od firmy Olimex, doplněný o rozšiřující signály pro tuto architekturu. Je však zřejmé, že vlastní jádro zapojení je stejné jako u univerzálních adaptérů - je zde použit obvod 74HC244, funčně obdobný 74HC125. Ze schématu je vidět, že adaptér obsahuje rozšiřující piny.

U USB adaptérů převládá použití proprietárních zapojení. Každá firma dodává vlastní adaptér spolu s odpovídajícím softwarem. Příkladem je Platform USB Cable od firmy Xilinx. Jediné standardy v USB adaptérech jsou v současnosti vztaženy k použití obvodu



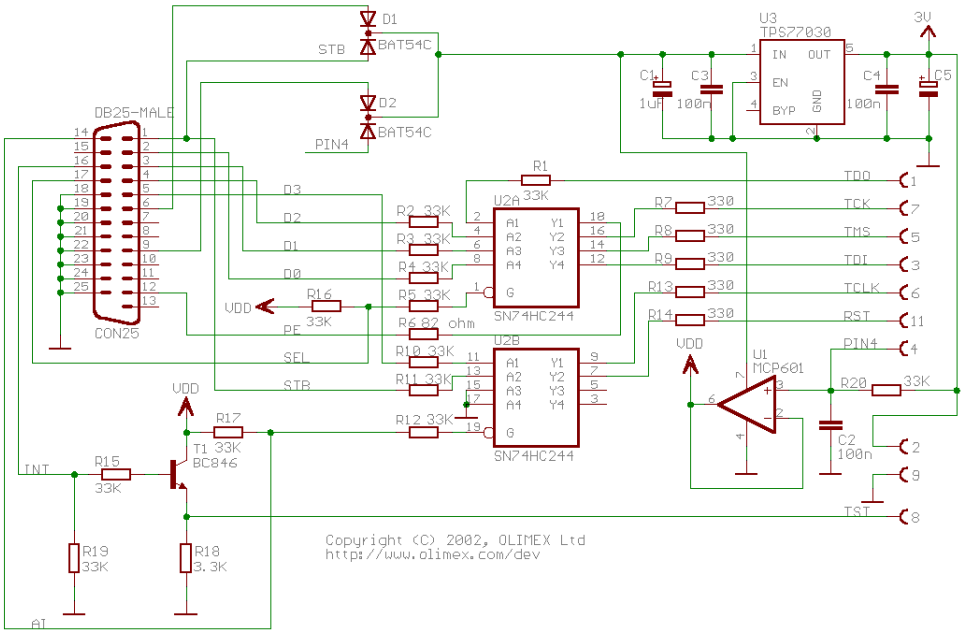
Obrázek 5.1: Adaptér Xilinx DLC5 JTAG Parallel Cable. [28]

FT2232, což je převodník typu USB vestavěné zařízení (USB na sériový či paralelní port). Zapojení takového adaptéru je na obr. 5.3. Na tomto zapojení jsou postaveny adaptéry Amonteg JTAGkey, Olimex ARM-USB-JTAX, Olimex ARM-USB-Tiny, Turtelizer a další. Vzhledem k tomu, že se jedná o de fakto jediný „standard“ pro USB adaptéry, budu uvažovat právě toto vzorové řešení v další části práce. [28]

5.2 Software pro komunikaci po sběrnici JTAG

Pro testování součástek a DPS existuje množství propracovaných komerčních aplikací. Zde se však zaměřím na volně dostupné programy typu Open-source a freeware. Obecně neexistuje univerzální adaptér JTAG pro jakýkoliv obvod od jakéhokoliv výrobce. Budu se zabývat řešeními pro součástky Xilinx FPGA Spartan a ARM. Z volně dostupných programů pro tyto součástky jsou tyto zajímavé svou kvalitou: (v závorce podporované architektury)

- ianjtag inAccess Networks's JTAG tools [ARM]
  (http://www.inaccessnetworks.com/projects/ianjtag/)
- JTAGER [ARM]
  (http://jtager.sourceforge.net/)
- JTAG Tools a jeho nástupce UrJTAG [Xilinx Spartan 3]
  (http://openwince.sourceforge.net/jtag/)



Obrázek 5.2: Adaptér Olimex MSP430-JTAG. [21]

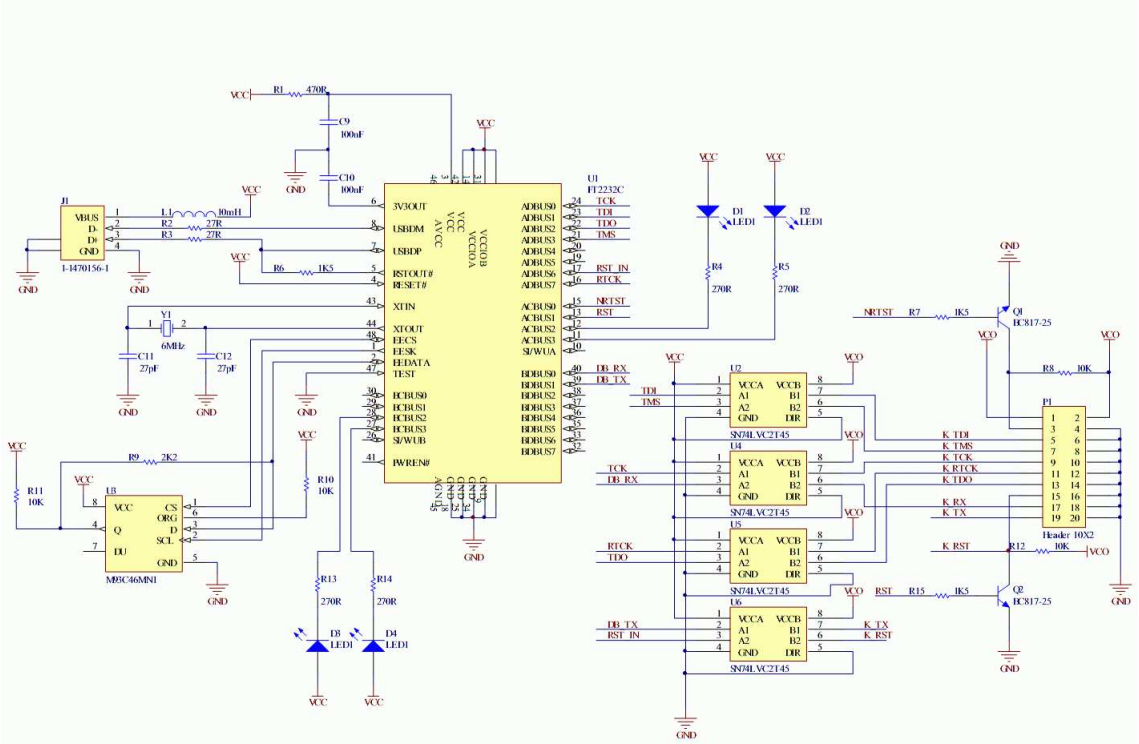
- J-Drive [Xilinx Spartan 3]
  (http://www.xilinx.com/support/documentation/)
- xc3sprog [Xilinx Spartan 3]
  (http://sourceforge.net/projects/xc3sprog/)
- OpenOCD [ARM]
  (http://openocd.berlios.de/web/)

Vyjímečným řešením v oblasti Open-source je projekt OpenOCD, který je cílen především na možnosti debugování procesoru ARM za běhu, ale umožňuje také ostatní komunikaci přes JTAG, včetně programování. O tomto programu bude pojednávat samostatná kapitola, protože v navrhovaném programátoru je žádoucí podpora ze strany OpenOCD.

Při popisu příkladu schopností programovacího softwaru se později zaměřím na UrJTAG, protože představuje velmi komplexní řešení s podporou součástek od výrobců Atmel, Altera, Lattice, Xilinx a dalších.

5.3 Projekt OpenOCD

Projekt započal jako diplomová práce Dominika Ratha. Posléze pokračoval a stále je intenzivně vyvíjen jako Open-source projekt. Je zaměřen na ladění, programování a boundary scan testování mikrokontrolérů a mikroprocesorů s rozhraním JTAG. Podporuje obvody s jádrem ARM7, ARM9, XScale, Cortex-M3 a flash paměti NOR a NAND. Komunikace s čipem může probíhat přes USB nebo paralelní port, což je dáno širokou podporou adaptérů různých výrobců. Společnou vlastností podporovaných USB adaptérů je použití obvodu FTDI FT2232. To je hlavní důvod pro volbu čipu kompatibilního s FT2232 v navrhovaném programátoru, viz kapitola Návrh zapojení programátoru JTAG.



Obrázek 5.3: Adaptér USB používající FT2232. [17]

OpenOCD rozlišuje 3 moduly, ze kterých je složen vývojový řetězec: interface (adaptér pro programování), board (vývojová deska, na níž je umístěn programovaný čip) a target (samotný čip). Pro každý z těchto modulů je použit samostatný konfigurační soubor s nastaveními specifickými pro tento modul. Součástí projektu jsou předpřipravené konfigurační soubory pro spoustu běžných adaptérů, desek a čipů. OpenOCD může komunikovat s GDB (GNU Project Debugger) serverem, což umožňuje podporu ladění pomocí standardního uživatelského rozhraní (samozřejmě jsou zde určitá omezení daná cílovou architekturou, např. malý počet breakpointů).

Projekt OpenOCD se od svého vzniku neustále rozvíjí a má ambice se stát hlavní Open-source vývojovou platformou nejen pro ARM ale i obecně pro JTAG obvody. [26]

5.4 Projekt UrJtag

UrJtag je softwarový balík umožňující práci s hardwarem disponujícím JTAG rozhraním. Má otevřenou a modulární architekturu pro snadná rozšíření např. o tester osazených DPS, či programátor FLASH pamětí. [27] UrJtag může běžet na operačních systémech Windows a Linux (Unix obecně). Podporuje adaptéry pro paralelní port a USB adaptéry postavené na čipu FT2232. Proto jsem ho zvolil pro ukázkový příklad praktického použití JTAG. Po spuštění programu příkazem „urjtag“ program komunikuje pomocí příkazové řádky začínající promptem jtag>. Prvním krokem je konfigurace používaného adaptéru, zde označovaného jako kabel. Uvedený příklad připojí adaptér typu Wiggler na paralelní port LPT1.

```
jtag> cable wiggler ppdev /dev/parport0
Initializing wiggler JTAG Cable on parallel port at 0x378
```

Dalším krokem bude detekce připojeného JTAG zařízení (součástky). Zařízení je rozpoznáno na základě čtení obsahu jeho registru IDCODE a přiřazení odpovídajícího textového popisu z databáze programu. Je pochopitelně nutné tuto databázi předem naplnit těmito dvojicemi údajů (název součástky - IDCODE). Uvedený příklad ukazuje detekci procesoru Intel PXA250.

```
jtag> detect
IR length: 5
Chain length: 1
Device Id: 01011001001001100100000000010011
  Manufacturer: Intel
  Part:         PXA250
  Stepping:     C0
```

Při sériovém spojení více součástek na sběrnici JTAG je vhodné zobrazit si jejich přehled, včetně aktuálně prováděné instrukce a vybraného datového registru. Příklad ukazuje výpis pro jedinou součástku v řetězci.

```
jtag> print chain
 No. Manufacturer Part   Stepping Instruction Register
-------------------------------------------------------
   0 Intel        PXA250 C0       BYPASS      BR
```

Nyní si můžeme vypsat obsah boundary scan registru, tj. logické hodnoty na jednotlivých pinech procesoru. Přiřazení získaného bitového řetězce příslušným pinům je třeba získat ze souboru BSDL od výrobce součástky. Příklad ukazuje výpis stavu pinů procesoru PXA250. Instrukcí SAMPLE/PRELOAD je získán stav pinů. Tento stav je uložen v buňkách boundary scan registru. Obsah registru je poté instrukcí dr vypsán na výstup.

```
jtag> instruction SAMPLE/PRELOAD
jtag> shift ir
jtag> shift dr
jtag> dr
100011001000001000011001011111111111111111001101110...
```

Existují další nízkoúrovňové příkazy umožňující přímé řízení TAP kontroléru a zápis hodnot na piny JTAG sběrnice. Pro praktické využití je však zajímavější podpora zpracování SVF souborů.

```
jtag> svf <SVF soubor>
jtag> instruction BYPASS
jtag> shift ir
```

Použití instrukce BYPASS je pouze pro zajištění připojení definovaného registru po dokončení zpracování SVF souboru.

Velmi zajímavou částí UrJtagu z hlediska této práce je podpora programování FLASH pamětí, i když velmi omezená co do množství podporovaných typů.

Kapitola 6

Sběrnice USB

Záměrem této práce je návrh programátoru pro JTAG rozhraní. Tento programátor bude připojen k PC, na němž poběží obslužný program. Do nedávné doby byla komunikace řešena převážně přes paralelní port (i dnes jsou stále k dostání adaptéry JTAG na paralelní port). To přináší zjednodušení při tvorbě programového vybavení, nicméně omezuje rychlost přenosu a hlavně neumožňuje použití na nových PC, kde již paralelní port není přítomen. Proto bude navrhovaný programátor komunikovat přes USB sběrnici. Dále se zaměřím na popis funkce USB sběrnice, koncových zařízení a volbu vhodného řadiče (čipu).

6.1 Standardy USB

První verze specifikace USB (Universal Serial Bus) byla uveřejněna v roce 1996 pod číslem 1.0. Následovala verze 1.1, která navíc přidala jeden nový typ přenosu (Interrupt OUT), jinak je shodná s verzí 1.0. Obvykle se tyto dvě verze označují společně jako USB 1.x. Podporované rychlosti přenosu jsou označovány jako Low Speed a Full Speed. Požadavky na zvýšení rychlosti přenosu způsobily vznik další verze 2.0. Ta již podporuje 3 typy přenosové rychlosti: Low Speed, Full Speed (stejné jako u USB 1.x) a High Speed (480Mbps). USB 1.x je zpětně kompatibilní s USB 2.0, pouze se nevyužije nejvyšší rychlost. V roce 2008 byla uvolněna specifikace USB 3.0. Ta k původní definici 2.0 přidává další dva vodiče umožňující přenos až 5Gbs. Tento mód se nazývá Super Speed. Verze 3.0 zůstává opět zpětně kompatibilní s USB 2.0. [7]

Typy přenosů na USB

S každým zařízením může počítač komunikovat až čtyřmi různými typy přenosů. Povinně podporovaný je však pouze první z nich, řídící.

Řídící přenos (Controll Transfer) - Vysoká priorita přenosu a automatická kontrola chyb. Tento přenos je používán při počáteční enumeraci, ale může být používán i poté. Pro řídící přenos je vždy garantovaná určitá část přenosového spektra.

Přenos při přerušení (Interrupt Transfer) - Vhodné pro zařízení posílající pravidelně malé množství dat, jako pákové ovladače, myši, klávesnice. Přerušení není vyvoláváno zařízením, ale počítačem, který se v předem nastavitelných intervalech dotazuje na stav zařízení.

Hromadný přenos (Bulk Transfer) - Používané pro přenos velkých množství dat, která nejsou časově kritická, ale nesmí být porušená. Rychlost závisí na aktuálním vytížení sběrnice. Vhodné např. pro tiskárny, externí disky, skenery.

Izochronní přenos (Isochronous Transfer) - Přenos velkého množství dat s přesně daným zpožděním. Neprovádí se kontrola přenášených dat na chyby, proto je tento přenos vhodný pro zařízení kde je důležitější rychlost, než bezchybnost (typicky multimediální data).

6.2 Podpora zařízení USB na straně operačního systému

Ve standardu USB je obsažena podpora enumerace. Jde o automatické rozpoznání připojeného zařízení operačním systémem (OS) a nahrání příslušného ovladače. Proces enumerace je přímo součástí OS a uživatel, ovladač ani obslužný program se ho nijak neúčastní. Po připojení je zařízení rozpoznáno podle změny napěťových úrovní na datových vodičích portu. Existuje několik kombinací těchto úrovní, podle nichž se rozlišuje maximální podporovaná rychlost zařízení. OS v rámci enumerace dotazuje zařízení na informace, jenž zařízení obsahuje ve formě tzv. deskriptorů. Hlavním údajem jsou zde číslo výrobce VID (Vendor Identification) a číslo zařízení PID (Product Identification). Na základě těchto údajů OS vyhledá nebo si vyžádá od uživatele odpovídající ovladač a ten následně zavede. Přístup na zařízení pak probíhá voláním API konkrétního OS a funkcí z knihoven ovladače. [7]

6.3 Řadiče USB na straně zařízení

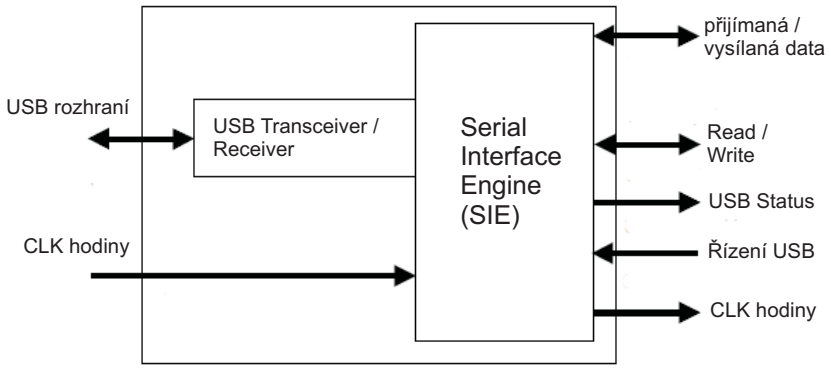
Každé zařízení připojitelné ke sběrnici USB musí mít určitou inteligenci umožňující komunikaci s hubem. Tyto vlastnosti jsou zajišťovány mikrokontrolérem nebo speciálním obvodem typu ASIC (Application Specific Integrated Circuit). Řadič musí obsahovat registry, buffery a vstupně-výstupní rozhraní, aby byl schopen provést enumeraci a poté přijímat a vysílat data.

Řadič typu USB transceiver

V nejjednodušším případě je USB řadič složen jen z fyzického rozhraní pro dekódování a kódování signálů pro sběrnici a jednotky SIE (Serial Interface Engine). Tyto komponenty jsou umístěny do společného pouzdra a výsledná součástka se nazývá USB transceiver. Toto řešení je nejstarší a vyžaduje další obvody pro komunikaci s periferním zařízením. Dnes se již samostatně nepoužívá, nicméně je součástí komplexnějších čipů. Výstupem z SIE jsou posloupnosti bitů přenášených po USB, včetně hlaviček, kontrolních dat apod. Další zpracování obvykle zajišťuje mikrokontrolér nebo ASIC obvod. Na obrázku 6.1 je příklad typické architektury transceiveru. SIE jednotka je složena ze dvou částí, fyzické vrsty (PHY) a vrstvy přístupu k médiu (MAC). [7]

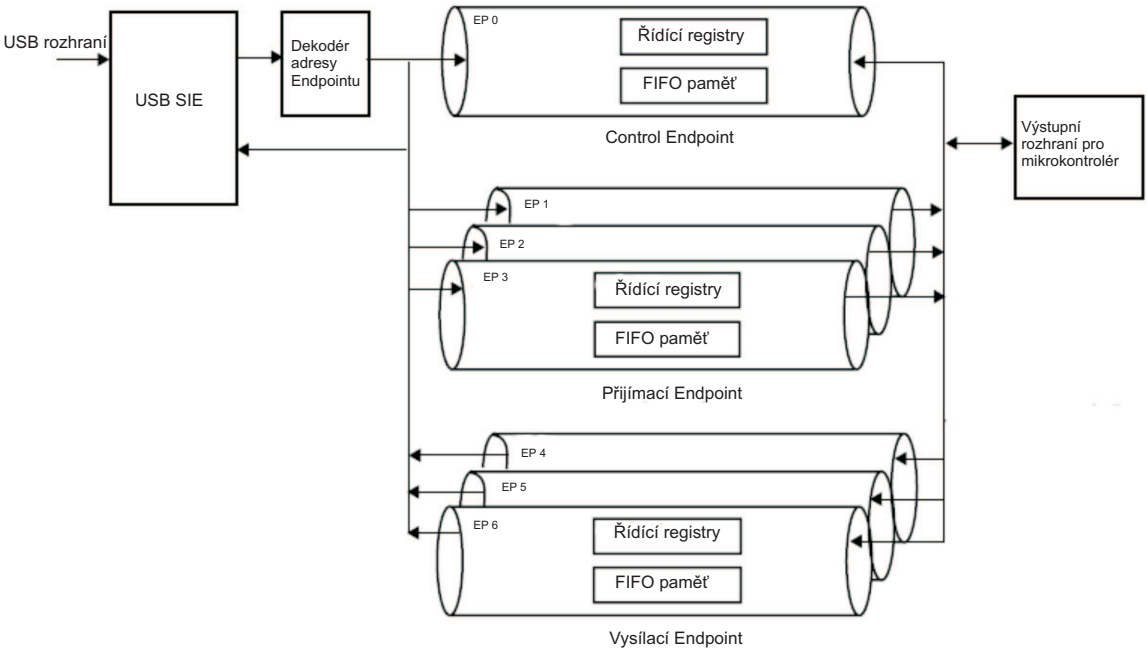
Řadič typu USB peripheral

USB peripheral přidává k USB transceiveru správu koncových bodů (Endpoints). Každý z těchto endpointů obsahuje vyrovnávací paměť typu FIFO. Výstup z těchto koncových



Obrázek 6.1: Architektura USB transceiver. Podle [7]

bodů je vyveden na paralelní rozhraní společně s řídícími signály pro výběr daného endpointu. Počet endpointů závisí na konkrétním zařízení, minimální počet je jeden, maximální 31. Paralelní rozhraní endpointů může být dále vyvedeno na mikrokontrolér pro vlastní zpracování dat. Mikrokontrolér však nemusí být součástí USB peripheral. Na obrázku 6.2 je znázorněno rozšíření USB peripheral oproti USB transceiver. [7]



Obrázek 6.2: Architektura USB peripheral. Podle [7]

Řadič typu USB device

USB device představuje kompletní řešení vzniklé složením předchozích dvou řešení a přidáním mikrokontroléru do jednoho pouzdra. Jedná se o nejpoužívanější řešení. Obvykle se primárně vybírá vlastní jádro mikrokontroléru, přičemž USB je pouze jednou z mnoha periferií, které se na čipu nacházejí. (Dalšími jsou AD převodník, časovač, real-time obvod

a pod.). Obsluha USB rozhraní probíhá zápisem a čtením do registrů mikrokontroléru a pomocí obsluhy přerušení, jenž jsou generována při událostech na USB sběrnici. [7]

Řadič typu USB vestavěné zařízení

Zde je místo obecného mikrokontroléru přímo vestavěna řídící jednotka, která plní specifickou funkci pro nějaké typické periferní zažízení. Dané zařízení může být samozřejmě implementováno i mikrokontrolérem, ale pro často používaná zařízení se takto ušetří čas vývojáře. Navíc je možné zařízení dobře optimalizovat. Jako USB vestavěná zařízení se nejčastěji vyrábějí USB-Serial Port převodník, USB-Ethernet, USB-Smart Card čtečka, USB-IDE disk a podobně. Velkou výhodou je podpora od výrobce čipu, sestávající se z ovladače pro operační systém a aplikačních poznámek usnadňujících tvorbu zapojení. [7]

USB řadič jako IP Core pro FPGA

USB řadič některé z výše uvedených skupin může být také implementován v rekonfigurovatelném hardwaru. Je možné vytvořit si vlastní zapojení a popsat ho v jazycích jako VHDL nebo Verilog. V případě USB je však návrh komplikovaný potřebou dodržovat přesné časování, dále také nutností zpracovat analogový diferenciální signál ze sběrnice USB. Lepší řešení je použít již hotové a ověřené zapojení, které se dodává v podobě IP (Intelectulal Property) Core. IP Core je logická jednotka, složená z popisu obvodu v některém syntetizovatelném jazyce. Tuto jednotku pak pouze přidáme ke svému designu FPGA obvodu a napojíme se na vnější rozhraní jednotky. Existují implementace USB 1.0, 1.1, 2.0, 3.0 i USB On-The-Go. Jedná se však o komerční implementace. Volně dostupné implementace jsou s omezeními a pro nižší verze USB.

6.4 Rychlost přenosu a latence

Při použití sběrnice USB pro přenos dat mezi počítačem a JTAG adaptérem potřebujeme dodržet správné časování dat. Nutností je rovněž bezchybnost přenosu. V úvahu tak padá použití přenosového režimu Interrupt nebo Bulk. Režim Interrupt nemusí být ideální u vysokých rychlostí přenosu. Operační systémy, které nejsou navrženy jako real-time nezaručují dodržení rychlosti přepínání mezi aplikacemi. V prostředí Windows je např. udávaná použitelná mez 1 ms. [7] Rychlejší požadavky na Interrupt transfer nebude operační systém schopen obsloužit. Proto vyspělejší USB řadiče používají bufferování na straně aplikace i čipu, a přenos pak probíhá nárazově přes Bulk transfer. Při odesílání do USB řadiče jsou data shromažďována v bufferu aplikace až do maximální velikosti paketu pro Bulk transfer, poté jsou všechna odeslána jedním přenosem. Je ovšem možné odeslat data i před naplněním bufferu, klesá však efektivita, protože část paketu je nevyužita. Při odesílání z USB řadiče do aplikace v počítači se opět čeká na naplnění bufferu. Současně však běží časovač, který při svém vypršení způsobí odeslání bufferu, i když není zcela naplněn, případně je i prázdný. Tento časovač je možné uživatelsky nastavit. Uvedený postup komunikace s použitím Bulk přenosů je použit například u USB řadiče FT2232, kterým se budu zabývat v dalších kapitolách.

Kapitola 7

Návrh zapojení programátoru JTAG

Navrhovaný programátor by měl být univerzální ve smyslu podpory více platforem. Přestože JTAG je standard, ve skutečnosti předpisuje implementaci pouze základních příkazů vhodných pro oživování a testování zařízení. Programování vestavěné paměti je specifické podle výrobce. Zaměřím se na architektury Xilinx FPGA Spartan 3 a ARM AT91SAM920 z důvodu dostupnosti hardwaru (vývojových kitů) pro testování. Na straně programování (a ladění) architektury ARM bylo vykonáno hodně práce v projektu OpenOCD. Protože cílem OpenOCD je poskytnout kompletní vývojové prostředí Open Source pro platformu ARM, bude i vyvíjený programátor s OpenOCD kompatibilní. Pro zbývající platformy podobné široce rozšířené prostředí není, nicméně existuje velmi propracovaný projekt UrJtag, který již byl popsán dříve.

Jako základ, z nějž budu vycházet je kompatibilita s čipem FT2232 firmy FTDI. Nad tímto základem je možné postavit některé z těchto tří zapojení: pouze FT2232 využívající interní podporu JTAG protokolu, FT2232 jako obecný převodník na paralelní rozhraní společně s FPGA polem, a konečně čip Vinculum, který k původní podpoře FT2232 přidává vestavěný mikroprocesor s vlastním real-time operačním systémem. Nyní rozeberu jednotlivé možnosti podrobně.

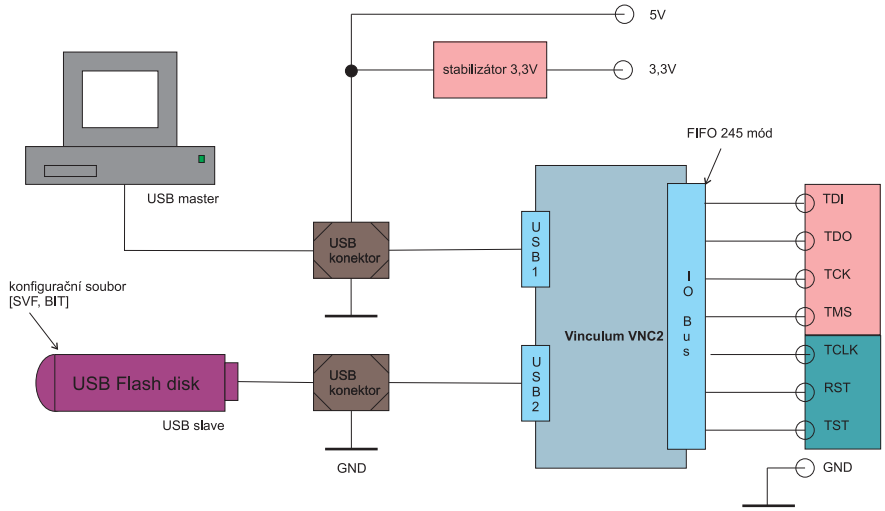
7.1 FTDI Vinculum

Vinculum je čip od firmy FTDI spojující vlastnosti převodníku FT2232 a řadiče typu USB device. Existují dvě verze - VNC1L a VNC2. VNC2 má další nové vlastnosti oproti VNC1L a v dalším textu budu uvažovat tuto variantu. VNC2 oproti FT2232 disponuje dvěmi rozhraními USB 2.0 full speed a vestavěným procesorem. Díky tomuto spojení by se dala řídící logika programátoru implementovat v jednom čipu společně s převodníkem USB. VNC2 však není pouhé spojení USB řadiče a mikrokontroléru.

VNC2 obsahuje jako hlavní komponentu 16 bitový procesor harvardské architektury doplněný pamětí flash o kapacitě 256kB a pamětí RAM o kapacitě 16kB. Na čipu je také vestavěný debugovací obvod, což je výrazná výhoda při ladění aplikace. Dvě rozhraní USB mohou být zapojena jako host a nebo jako device, a to v jakékoliv kombinaci. K čipu tak lze připojit až dvě standardní USB periferie, jako je klávesnice, IDE disk, tiskárna apod. Stejně tak může být čip připojen ke dvěma počítačům a tvořit tak jejich spojení. Na čipu je dále k dispozici široký výběr komunikačních standardů (UART, SPI, FIFO 245 parallel,

...), které mohou být připojeny na piny čipu podle přání uživatele s použitím vestavěného multiplexoru. Na procesoru běží řídící aplikace - firmware. Ten je tvořen real time operačním systémem VOS (Vinculum Operating System), což je preemptivní víceúlohový OS. Aplikace pro VOS je možné psát v (modifikovaném) jazyce C a assembleru. K dispozici jsou VOS ovladače pro standardní USB zařízení, např. IDE disk se souborovým systémem FAT. FTDI k obvodu zdarma poskytuje grafické vývojové prostředí pro kompletní vývoj aplikace, zahrnující překlad, linkování, ladění, komunikaci s čipem a debugování za běhu aplikace. Několik standardních firmware je také k dispozici již hotových, takže pro nenáročné aplikace není třeba vyvíjet žádný software.

Použití čipu Vinculum pro navrhovaný programátor by bylo výhodné ze dvou důvodů. Vinculum vychází z obvodu FT2232, takže by bylo možné dosáhnout zpětné kompatibility s programátory vybavenými tímto obvodem. Dále je zde výhodou značné zmenšení rozměrů a spotřeby programátoru, protože vše potřebné je v jediném pouzdře (s vyjímkou několika pomocných obvodů). Řídící program běžící přímo na čipu umožňuje autonomní činnost i bez připojení k počítači. Tak je možné vytvořit i přenosný programátor, který bude konfigurační data načítat např. s USB flash disku, viz obrázek 7.1. K tomuto využití by bylo třeba doplnit ještě jednoduché uživatelské rozhraní (klávesnice, displey,...).



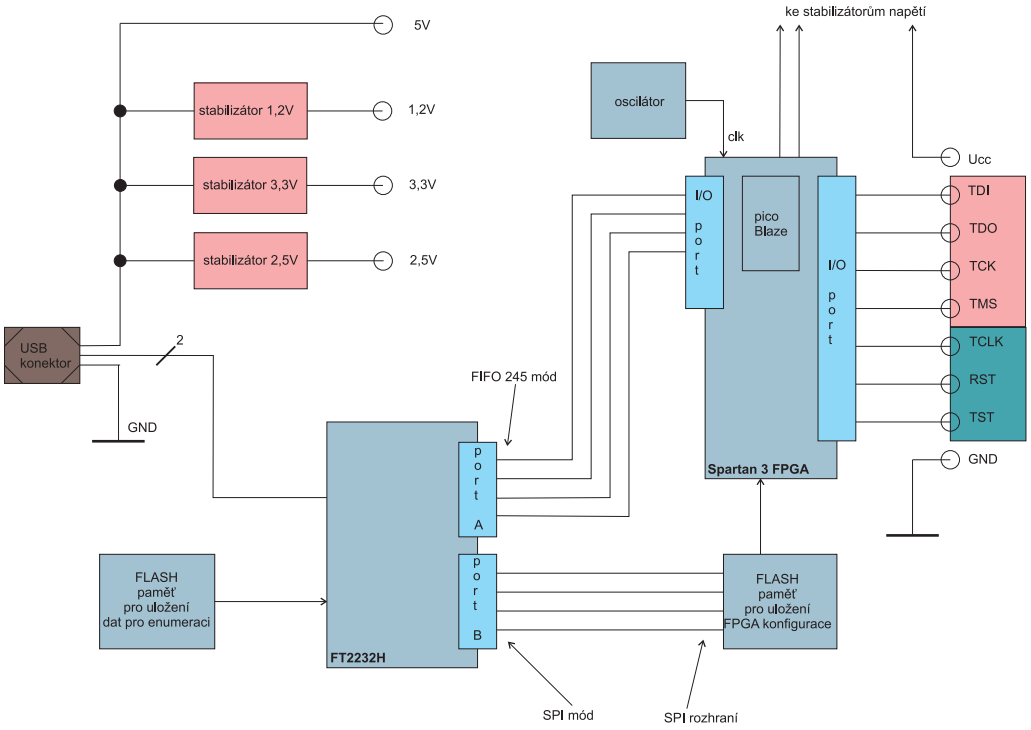
Obrázek 7.1: Blokové schéma programátoru při použití VNC2

Zásadním důvodem proti použití Vinculum VNC2 je chybějící podpora příkazového procesoru MPSSE. Ten je zcela zásadní při zjednodušení ovládání časování protokolů, včetně JTAG. Je zde ovšem k dispozici klasický FIFO 245 mód, který lze naprogramovat pro jakýkoliv protokol. Návrh protokolu JTAG na této nízké úrovni však představuje značně úsilí. Je však možné, že bude brzy výrobcem implementován MPSSE jako firmware běžící v mikrokontroléru VNC2. Tím by bylo dosaženo téměř 100% kompatibility s čipy FT2232. Zatím však výrobce tuto možnost neuvádí. [18]

7.2 FTDI FT2232 a FPGA pole

Možnost zapojení FPGA programovatelného pole dává největší univerzálnost a možnosti použití programátoru. Také je ovšem nejnáročnější po materiální i rozměrové stránce. FPGA může obsahovat algoritmus (prováděný soft-procesorem, např. picoBlaze) pro řízení libo-

volného množství protokolů, navíc je možná snadná rekonfigurace pro případný update. FPGA hradlové pole umožňuje snadnou modifikovatelnost softwaru, a to nejen při vývoji, ale hlavně při používání. Lze provádět update pomocí uživatelského rozhraní, takže uživateli stačí dodat pouze nový konfigurační soubor.



Obrázek 7.2: Blokové schéma programátoru při použití FPGA

Na blokovám schématu (obrázek 7.2) je použit jako převodník USB - paralelní port již dříve uvedený obvod FT2232. Jsou využity oba kanály, kanál A slouží pro komunikaci mezi řídícím počítačem a FPGA polem a kanál B pak pro nahrání konfigurace při inicializaci FPGA. V tomto případě však není FT2232 plně využitý. Funkcionalita MPSSE je vlastně implementována a značně rozšířena přímo v FPGA. Při použití FPGA pole je nutné uložit do něj konfiguraci. Při každém odpojení napětí je totiž konfigurace ztracena. Pro uložení a nahrání existuje více možností, jejichž vhodnost je daná na základě nároků na cenu zařízení a rychlost konfigurace. Konfigurace může být uložena v paměti typu FLASH, která je obvykle v externím pouzdru. Je také možné provádět konfiguraci přímo přes řídící aplikaci, při každém připojení programátoru k počítači a vypustit tak FLASH paměť.

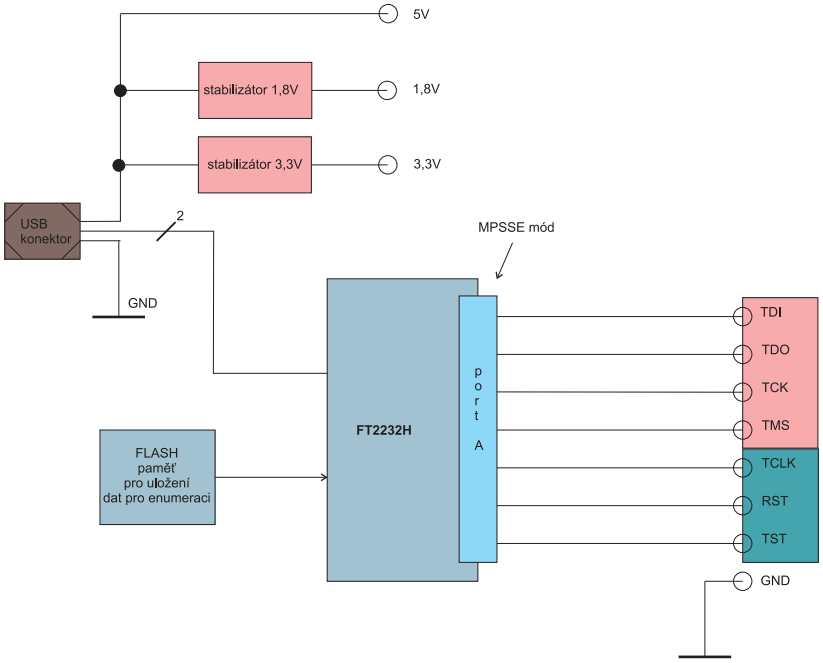
Toto zapojení je více vhodné pro univerzální programátory, kde se počítá s možností rozšiřování a vylepšování podporovaných protokolů a programovacích algoritmů.

7.3 FTDI FT2232 v módu JTAG

Obvody řady FT2232 obsahují příkazový procesor MPSSE (Multi Protocol Synchronous Serial Engine), který lze nakonfigurovat do režimu různých rozhraní (SPI, I2C, FIFO, JTAG). Zajímavá je zde možnost JTAG, která umožňuje připojit přímo na piny čipu sběrnici JTAG. Ovládání je řešeno přes volání příkazů z knihovny DLL FTCJTAG.dll, která tvoří nadstavbu nad nízkoúrovňovým přístupem k čipu. Tato knihovna je dodávána přímo výrobcem čipu

společně s dokumentací. Knihovna však tvoří pouze obálku nad příkazy pro MPSSE. Ty je možné také použít a dosáhnout tak specifické změny protokolu nebo vytvoření vlastního.

MPSSE příkazy jsou tvořeny posloupnostmi bytů, které obsahují klíčová slova kódující požadovanou funkci, následovaná příslušným počtem parametrů. Je zde možnost zapisovat data v režimu little-endian a big-endian. Výrobce poskytuje dokumentaci a také ukázkové zdrojové kódy příkladů, z nichž lze funkci příkazů vysledovat. Pro standardní použití se však lze přímé obsluze MPSSE vyhnout a vystačit s dodávanými DLL knihovnami.



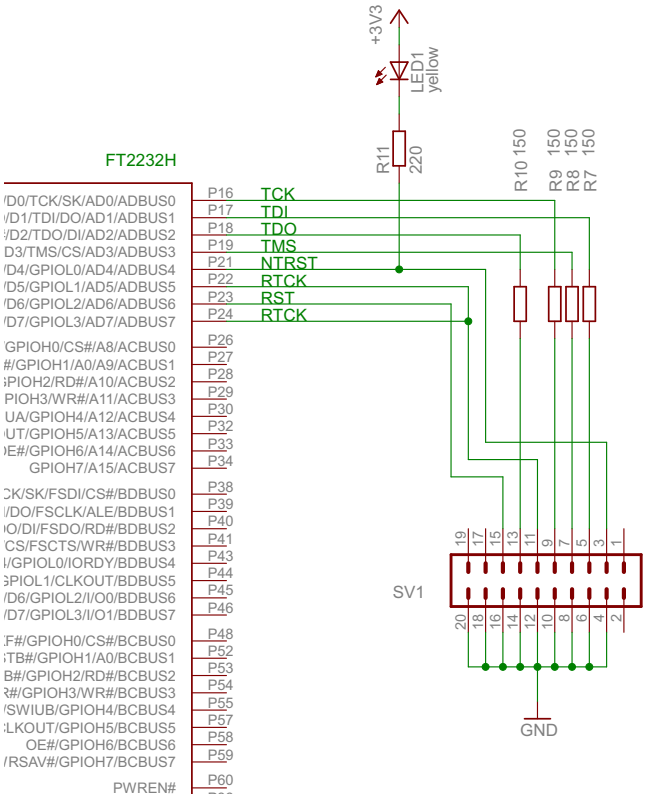
Obrázek 7.3: Blokové schéma programátoru při použití MPSSE módu

Na obrázku 7.3 je blokové schéma v minimální podobě. Je zřejmé, že z výše uvedených zapojení je nejjednodušší, přičemž však poskytuju požadovanou funkcionalitu.

Toto zapojení jsem proto zvolil pro navrhovaný programátor. Podle katalogových údajů jsem navrhl schéma zapojení, které je uvedeno v příloze Schéma zapojení. Oproti originálnímu zapojení jsem použil ekvivalentní součástky, které byly snadno dostupné v ČR. Modul jsem navrhl a postavil ve dvou variantách. Varianta 1 byla především pro testovací účely a má vyvedeny všechny I/O piny na kolíkový konektor s roztečí pro umístění do nepájivého pole. Zapojení jsem poté upravil, aby bylo kompaktnější. Na modul jsem přímo umístil konektor pro připojení k JTAG rozhraní programované desky a důkladně jsem odstínil celý modul potažením nevyužitých oblastí na desce měděnou plochou spojenou s potenciálem země. V příloze Osazení DPS a seznam součástek je nákres desky plošného spoje této 2. varianty včetně osazení součástkami. (Kompletní podklady pro výrobu obou modulů z programu Eagle jsou uloženy na doprovodném CD-ROM). Modul je napájen pouze z USB sběrnice 5V. Nelze z něj ale odebírat napájecí proud pro programované zařízení z důvodu možného překročení mezního proudu stabilizátoru napětí na desce. Programované zařízení je k modulu připojeno pomocí konektoru MLW20, zapojení pinů odpovídá variantě ARM JTAG (viz přílohu Konektory pro JTAG). Zařízení musí mít vlastní napájecí zdroj.

Hardwarová konfigurace adaptéru je kompatibilní se schématem USBJTAG-1, které navrhl Hubert Hoegl a je uvedené v diplomové práci Dominica Ratha. [26] Úprava mého

testovacího modulu na toto zapojení spočívala v zapojení pomocných signálů (NTRST, RST, RTCK), přidání 20 pinového konektoru a doplnění rezistorů pro přizpůsobení napěťových úrovní. Rezistory v signálech TDI, TDO, TCK a TMS umožňují propojení se vstupy obvodu FPGA, pracujícími s napětím 2.5V. Modul programátoru používá napětí IO pinů 3.3V. Výsledné schéma výstupní části modulu je na obrázku 7.4.



Obrázek 7.4: Napojení programátoru na JTAG sběrnici

V následujících kapitolách se tedy budu věnovat použití vytvořeného modulu, resp. konkrétnímu čipu FT2232H, na němž je modul založen. Pro testovací a srovnávací účely jsem také používal programovací adaptér ARM-USB-TINY [21], který je osazen starší variantou čipu FT2232C. Protože právě tato varianta je nejvíce rozšířená, navržený software bude fungovat s oběma typy. Podpora obou typů není bohužel triviální záležitost, protože funkce FTDI ovladače pro jejich ovládání jsou rozdílné (jiné názvy funkcí a parametry).

Kapitola 8

Práce s obvody FT2232

Firma FTDI se specializuje na vývoj a výrobu čipů, které slouží jako převodníky mezi USB sběrnicí a některým sériovým či paralelním protokolem. Tyto protokoly jsou standardně používané a vývojářům známé, odpadá tím tedy nutnost řešit implementační detaily USB protokolu. Velkou oblibu si získaly především čipy série FT232, k čemuž přispěla i velká podpora ze strany výrobce. Ten poskytuje bezplatně ovladače pro operační systémy Windows a Linux a také podrobnou dokumentaci a aplikační poznámky. [15] První obvod z této série je FT232, který poskytuje převodník USB na sériový port, podporující všechny signály dle standardu RS232. Vývody je také možné používat v tzv. režimu bitbang, což umožňuje samostatné řízení jednotlivých pinů výstupu, tedy určitou variantu k obvyklému paralelnímu portu. Tento režim je možné využít pro implementaci vlastního či standardního protokolu, a to včetně JTAG. Nevýhodou je ale nízká rychlost přenosu a nezaručitelná doba odezvy. To plyne z použití okamžitého přenosu načtených/zapisovaných dat mezi počítačem a čipem přes USB sběrnici bez využití bufferů. Dalším vyráběným čipem je FT2232, který v sobě obsahuje „dvakrát“ čip FT232 a navíc přidává další vlastnosti. Hlavní novinkou je zavedení módu MPSSE, který dokáže vytvářet interně (hardwarově) správné časování pro nejrozšířenější protokoly (SPI, JTAG, I2C, ...). Problém latence USB sběrnice řeší použitím vlastních vyrovnávacích bufferů pro čtení i zápis. [12]

8.1 Módy obvodu FT2232

Po úspěšné enumeraci se obvod nachází v módu RS232 UART, tedy může být ihned použit jako sériový port. (Toto výchozí nastavení lze změnit zápisem do konfigurační FLASH paměti připojené k obvodu, ale tímto způsobem lze zvolit jen některé módy). Programově pomocí API je pak možné změnit mód obvodu libovolně na některou z těchto variant: [12]

1. Asynchronous Bit Bang
2. Synchronous Bit Bang
3. MPSSE
4. MCU Host Bus Emulation Mode
5. Fast Opto-Isolated Serial Mode
6. CBUS Bit Bang Mode

7. Single Channel Synchronous 245 FIFO Mode

Pro komunikaci pomocí API musí být samozřejmě nainstalovány odpovídající ovladače do operačního systému.

Pro účely programátoru JTAG je velmi vhodný mód MPSSE (Multi-Protocol Synchronous Serial Engine). Komunikaci v tomto módu řídí příkazový procesor, vestavěný na čipu FT2232, kterému se zasílají příkazy v podobě bytových kódů s parametry. Procesor pak provede vlastní čtení či zápis na pinech, přičemž dodržuje správné časování. [14] Určitou nevýhodou pro programátora je nutnost komunikace s příkazovým procesorem pomocí hexadecimálních čísel - kódů operací. Proto firma FTDI vytvořila nadstavbu pro jednotlivé protokoly, včetně JTAG protokolu. Tato nadstavba je implementována pomocí DLL knihovny, která zapouzdřuje nízkoúrovňovou komunikaci s příkazovým procesorem pomocí volání funkcí. [11]

Uvažujme častý příkaz IDCODE při komunikaci s JTAG zařízením. Tento příkaz vypíše řetězec hexadecimálních číslic z IDCODE registru součástky, které jednoznačně identifikují tuto součástku a jejího výrobce. Při použití přímého bitového přístupu je třeba postupovat takto (zkrácená verze kódu):

```
// Reset kontroleru
ftStatus = FT_SetBitMode(ftHandle, 0x0, 0x00);

// Povolit MPSSE mode
ftStatus = FT_SetBitMode(ftHandle, 0x0, 0x02);

// Navigage TMS přes stavy: Test-Logic-Reset -> Run-Test-Idle ->
// Select-DR-Scan -> Select-IR-Scan -> Capture-IR-Scan -> Shift-IR
// Přechody TMS:                      TMS=1          TMS=0
//     TMS=1             TMS=1              TMS=0          TMS=0

// Pošli data na TMS, nečti data na vstupu TDI
byOutputBuffer[dwNumBytesToSend++] = 0x4A;

// Počet CLK pulsů na CLK vodiči (počet přechodů - 1)
byOutputBuffer[dwNumBytesToSend++] = 0x05;

// Data posílaná na TMS, LSB first. Vzor dat je "00001101"
byOutputBuffer[dwNumBytesToSend++] = 0x0D;

// Odeslat příkaz do bufferu
ftStatus = FT_Write(ftHandle, byOutputBuffer, dwNumBytesToSend,
&dwNumBytesSent)


// Následují další přechody mezi stavy TAP automatu
......

// Načtení 4 bytů IDcode ze součástky
ftStatus = FT_Read(ftHandle, &byInputBuffer, dwNumBytesToRead,
&dwNumBytesRead);

// Výpis načtených dat
printf("IDCODE je 0x %x %x %x %x", byInputBuffer[3], byInputBuffer[2],
 byInputBuffer[1], byInputBuffer[0]);
```

Při použití MPSSE s obalovací knihovnou FTCJTAG.DLL, lze stejnou operaci provést takto (zkrácená verze kódu):

```
// Zápis ID instrukce, tj. "001001" pro Spartana3
WriteDataBuffer[0] = 0x09;

// Odeslat příkaz do bufferu součástky
JTAG_Write(ftHandle, true, 6, &WriteDataBuffer, 1, RUN_TEST_IDLE_STATE);

// Přečte data ze součástky
JTAG_Read(ftHandle, false, 32, &ReadDataBuffer, &dwNumBytesReturned,
RUN_TEST_IDLE_STATE);

// Výpis načtených dat
printf("IDCODE je 0x %x %x %x %x",ReadDataBuffer[3],ReadDataBuffer[2],
ReadDataBuffer[1],ReadDataBuffer[0]);
```

Na první pohled je vidět zvýšení přehlednosti zápisu při použití DLL knihovny. Hlavní výhodou je, že není třeba určovat a řídit přechody mezi stavy TAP automatu. Komunikace se zjednoduší na příkazy pro zápis a čtení. Tato možnost je však k dispozici pouze pro FTDI čipy podporující MPSSE režim, tedy řada FT2232 a FT4232. Je zajímavé, že novější typ USB řadičů od FTDI (Vinculum) režim MPSSE nepodporuje. [15]

8.2 Ovladače pro čip FT2232

Firma FTDI poskytuje ke svým čipům volně dostupné ovladače, ovšem pouze v binární podobě. Ovladače jsou dostupné pro Linux, Windows a MacOS. Ovladače jsou výrobcem děleny na dvě skupiny, D2XX Drivers a VCP Drivers. VCP Drivers umožní pohled na připojené FTDI zařízení jako na sériový port (pro operační systém vznikne nové hardwarové zařízení sériový port). Na takto vytvořený virtuální sériový port je možné přistupovat standardními prostředky z API operačního systému. D2XX Drivers umožňují přímý přístup na FTDI zařízení. V tomto režimu je možné vytvořit na pinech FTDI součástky jakýkoliv sériový či paralelní protokol dle přání uživatele. Některé nejobvyklejší přenosové protokoly jsou přímo podporovány hardwarem čipu (MPSSE režim). Pro použití v JTAG programátoru je nutné využívat služby ovladačů D2XX Drivers. [13]

Nevýhodou originálních ovladačů od výrobce je nedostupnost zdrojových kódů a omezující licenční podmínky. Vznikl proto Open-source projekt libftdi, který poskytuje podobnou funkcionalitu jako D2XX Drivers. [18] Navíc jsou deklarace funkcí velmi podobné originálním, takže případný přechod mezi FTDI a Open-source verzí ovladačů nečiní velký problém. Libftdi vyžaduje pro svoji činnost knihovnu libusb (knihovna pro přístup aplikací v uživatelském režimu na USB rozhraní). Zde vzniká problém při požadavku současně používat originální ovladače od FTDI a Open-source variantu libusb s knihovnou libftdi. Současné používání je myšleno jako instalace obou ovladačů paralelně na jednom operačním systému. Následující příklad se vztahuje k operačnímu sytému Windows, na němž jsem vývoj prioritně prováděl. Při připojení adaptéru s FTDI čipem na USB sběrnici lze po enumeraci zvolit ovladač, který Windows nahrají do systémového adresáře a trvale asociují s detekovaným zařízením. Pokud chceme změnit ovladač (z libusb na ftd2xx či naopak) je nutné původní odinstalovat a při následném připojení zařízení nechat nahrát druhý ovladač. Pokud budeme používat programy používající komunikaci s čipen FT2232 přes libftdi a ftd2xx, museli bychom vždy provést přeinstalaci ovladače. To je však pro praktické použití nere-

álné. Autoři knihovny libusb ale poskytují možnost instalovat libusb driver ke stávajícímu ovladači jako tzv. Filter Driver. [19] Tím je výše uvedený problém vyřešen. Postup instalace Filter Driveru pro FT2232 čip je následující:

1. Po prvotním připojení čipu nainstalujeme originální ovladače od výrobce (FTDI)
2. Ze stránek projektu libusb-win32 [19] stáhneme balíček pro Windows
3. Po rozbalení balíčku spustíme aplikaci install-filter-win.exe
4. V nabídce vybereme postupně všechna zařízení, pro která chceme filtr nainstalovat. Tato zařízení musí být připojena k USB sběrnici a po enumeraci. V případě FT2232 se tedy bude jednat o 2 převodníky USB Serial Converter (A a B kanál).
5. Po instalaci lze ověřit, že ve vlastnostech zařízení ve Správci zařízení přibyl v seznamu „Podrobnosti o ovladači“ soubor libusb0.sys

Po této instalaci lze spouštět aplikace přistupující k čipu s využitím libftdi i ftd2xx zcela transparentně z pohledu uživatele. Lze tedy spustit OpenOCD, který z licenčních důvodů nemůže být předkompilován s FTDI ovladači (používá proto libftdi).

Kapitola 9

Obvody FPGA

FPGA obvody musí být naprogramovány (nakonfigurovány) na požadované zapojení vždy při připojení napájení nebo po resetu. Jejich konfigurace je uložena v interní paměti typu SRAM a je po odpojení napájení ztracena. Pro konfiguraci moderních FPGA obvodů je dostupných několik módů, které lze obecně rozdělit na: [31]

- master serial mode
- master parallel mode
- slave serial mode
- slave parallel mode
- JTAG mode

Který z těchto módů bude použit při inicializaci FPGA je určeno nastavením logických úrovní na MODE pinech (M:0 - M:2). Tato kombinace pinů určuje způsob zavedení konfigurace při startu FPGA. Během normální činnosti je však možné kdykoliv provést novou konfiguraci pomocí JTAG rozhraní, nezávisle na nastavení pinů M:0 - M:2. Kromě uvedených módů existují i další, v závislosti na výrobci. Dále však budu uvažovat jen variantu konfigurace přes JTAG (JTAG mode).

Ještě než můžeme FPGA nakonfigurovat, musíme získat konfigurační soubor - řetězec binárních dat, který bude nahrán do FPGA a způsobí jeho propojení do požadované logické struktury. Tento řetězec se nazývá bitstream a je generován v poslední fázi tvorby softwaru pro FPGA. Formát bitstreamu je závislý na konkrétním výrobci a konkrétním FPGA čipu. Pro jeho generování je nutné použít program od daného výrobce, protože specifikace bitstreamu nejsou zveřejňovány ani standardizovány. Při konfiguraci FPGA přes JTAG rozhraní máme k dipozici pouze 4 standardní piny (TDI, TDO, TCK, TMS). FPGA má však vyvedeny na piny další signály, které se mohou podílet na konfiguraci nebo signalizovat její průběh. Zde již není možné popisovat obecně, proto se následující údaje vztahují k FPGA Xilinx Spartan3, který jsem používal pro praktickou činnost. Mimo již uvedených pinů M:0 - M:2 jsou použity piny PROG_B, INIT_B, DONE, CCLK a DIN (nebo TDI). Průběh konfigurace s vazbou na hodnoty těchto pinů je zjednodušeně následující: [31]

1. Systém je resetován připojením napájení, přivedením úrovně Low na pin PROG_B nebo instrukcí JPROGRAM vykonanou v JTAG TAP.

2. Po resetu je rozpoznán mód konfigurace podle úrovní na pinech M:0 - M:2. Pokud je nastaven Master Serial mód, začne FPGA generovat hodinový signál na pinu CCLK. Na tento pin je připojen hodinový vstup externí paměťi ROM s konfiguračním bitstreamem. Data z paměti jsou přiváděna na pin DIN FPGA. Pokud je zvolen mód JTAG jsou data očekávána na vstupu TDI JTAG. Hodinový signál pak není generován, použije se TCK signál z JTAG.

3. Data přicházející na datový vstup jsou porovnávána na přítomnost tzv. synchronizačního slova, které udává počátek bitstreamu. Pro Spartan je tato posloupnost FFFF FFFF AA99 5566. Bezprostředně pak následují vlastní konfigurační data, která jsou ukládána do interní SRAM paměti. Délka těchto dat je vždy stejná pro daný čip, nezávisle na složitosti návrhu. V bitstreamu jsou uložena i další konfigurační nastavení, inicializační hodnoty pinů, identifikační číslo FPGA, pro nějž je tento bitstream určen, CRC součet a podobně.

4. Po nahrání bitstreamu je spočten jeho kontrolní součet CRC a porovnán s udávanou hodnotou. Pokud se neshodují, je nastaven pin INIT_B na hodnotu Low. Pokud je CRC správný, je provedena interní sekvence zavedení konfigurace, nazývaná Startup sekvence. Následně je nastaven pin DONE na High a v FPGA běží nahraná konfigurace.

9.1 Konfigurační mód JTAG

Jestliže během inicializace je detekován konfigurační mód JTAG (dle nastavení pinů M:0 - M:2), FPGA bude čekat na konfiguraci přes JTAG port. Nyní je třeba začít komunikovat s TAP kontrolérem JTAG na FPGA. Nelze začít přímo posílat konfigurační bitsream, ale nejdříve je nutné zadat instrukce, které způsobí připojení interní logiky na JTAG. Tyto instrukce jsou však mimo standard IEEE 1149.1, takže se liší mezi výrobci i mezi jednotlivými čipy stejného výrobce. Pro Spartan3 jsou to tyto instrukce:

- JPROGRAM: Provede reset FPGA (ekvivalentní funkce s pinem PROG_B)
- CFG_IN: Povolí přístup ke konfigurační sběrnici (pro zápis konfigurace)
- JSTART: Zavede a spustí uloženou konfiguraci

Po zadání instrukce CFG_IN je možné odeslat celý konfigurační řetězec přes TDI pin JTAG. Po jeho nahrání následuje zadání instrukce JSTART, která způsobí provední inicializační Startup sekvence. Pro provádění Startup je nutné přivádět hodinové impulsy na TCK pin JTAG (pro Spartan konkrétně $\geq$ 12 impulsů). Tuto rozdílnost oproti Master Serial módu je třeba ošetřit i při generování samotného bitstreamu. Ve vlastnostech vytvářeného bitstreamu se musí explicitně nastavit jako vstup hodin při Startup sekvenci pin JTAG:TCK. [28]

V tabulce 9.1 je přehledně uvedena posloupnost zadávání instrukcí a dat při programování bitstreamem přes JTAG rozhraní do Spartan2 podle aplikační poznámky [29]. Symbol X ve sloupci vstupních dat znamená libovolnou hodnotu.

krok	popis činnosti	TDI data	TMS data	počet impulsů TCK
1	Po zapnutí připoj TMS k log. 1 a přiveď minimálně 5 pulsů na TCK. Tím TAP přejde do stavu Test-Logic-Reset	X	1	5
2	Přesuň se do stavu Run-Test-Idle	X	0	1
3	Přesuň se do stavu SELECT-IR	X	1	2
4	Přesuň se do stavu SHIFT-IR	X	0	2
5	Zadej CFG_IN instrukci	0101	0	4
6	Zadej poslední bit CFG_IN instrukce "0", při přechodu do stavu UPDATE-IR	0	1	1
7	Přesuň se do stavu SELECT-DR	X	1	2
8	Přesuň se do stavu SHIFT-DR	X	0	2
9	Pošli konfigurační bitstream (bitN [MSB] je prvním odeslaným bitem bitstreamu)	bitN … bit1	0	počet bitů bitstreamu - 1
10	Pošli poslední bit bitstreamu, při přechodu do stavu EXIT-DR	bit0	1	1
11	Přesuň se do stavu UPDATE-DR	X	1	1
12	Přesuň se do stavu SELECT-IR	X	1	2
13	Přesuň se do stavu SHIFT-IR	X	0	2
14	Zadej JSTART instrukci (inicializace konfigurace)	1100	0	4
15	Zadej poslední bit JSTART instrukce "0", při přechodu do stavu UPDATE-DR	0	1	1
16	Přesuň se do stavu SELECT-DR	X	1	2
17	Přesuň se do stavu SHIFT-DR a aplikuj min. 12 impulsů na TCK (STARTUP sekvence)	X	0	≥ 12
18	Přesuň se do stavu UPDATE-DR	X	1	2
19	Přesuň se do stavu Run-Test-Idle (Konfigurace nyní běží.)	X	0	1

Tabulka 9.1: Posloupnost instrukcí pro nahrání bitstreamu do Spartan

Kapitola 10

Obvody s jádrem ARM

ARM sám o sobě není konkrétní mikrokontrolér nebo mikroprocesor. Jedná se o návrh procesorového jádra od společnosti ARM Limited. Tento návrh je prodáván jako IP (Intelectual Property) konkrétním výrobcům polovodičových součástek. Ti k jádru přidají další potřebné či vhodné jednotky a společně s jádrem je zapouzdří a prodávají jako fyzický čip. ARM jádro má několik dostupných verzí (ARM1 - ARM11, StrongARM, XScale, Cortex), které jsou vyráběny různými výrobci (např. Atmel, Philips, TI). V současné době jsou jádra nižší verze než 7 považována za zastaralá. ARM je RISC architektura s následujícími hlavními vlastnosti: [1]

- 32 bitová vnitřní architektura
- 26 bitová adresová sběrnice
- 25 vnitřních 32 bitových registrů
- přístup do paměti pouze instrukcemi Load/Store
- 3 různé instrukční sady (ARM, Thumb a Jazelle)

V dalším textu budu uvažovat konkrétní realizaci čipu, obvod AT91SAM9260 od firmy Atmel. Tento čip (osazený na vývojové desce Olimex SAM9-L9260) jsem použil pro praktickou činnost.

10.1 Uložení vykonávaného programu

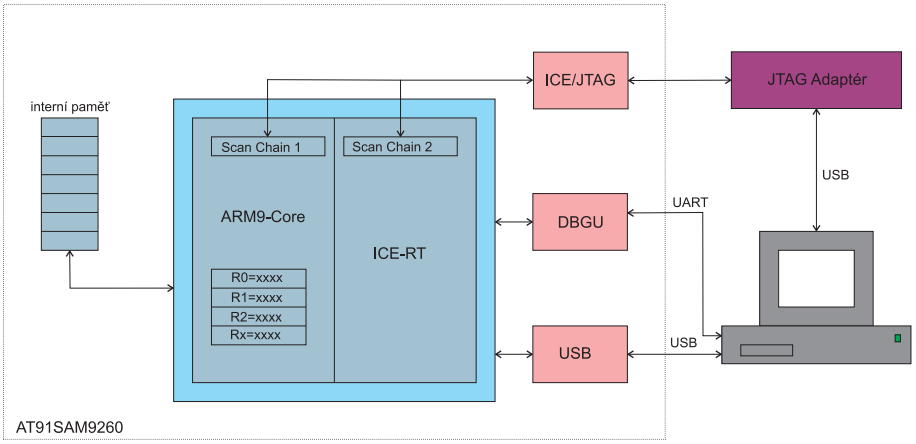
Kvůli maximální univerzálnosti jsou možnosti uložení programu pro ARM9260 velké. Je možné načíst program z SPI flash nebo NAND flash paměti. Většina mikrokontrolérů s jádrem ARM obsahuje i interní flash paměť na čipu. (V AT91SAM9260 ale není). [6]

Postup při bootování čipu je:

1. Inicializace Debug jednotky a USB rozhraní.
2. Spuštění DataFlash Boot programu, který vyhledává platnou hlavičku v externí SPI flash paměti. Je-li nalezena, program je načten do interní SRAM a spuštěn.
3. Není-li program nalezen, je dále prohledávána programem NANDFlash Boot externí NAND Flash na platnou hlavičku. Je-li nalezena, program je načten do interní SRAM a spušťen.

4. Není-li ani zde zaveditelný program nalezen, je spuštěn monitor SAM-BA a čeká na spojení na USB portu nebo na DBGU sériovém portu.

V počáteční fázi předpokládejme, že v žádné paměti není nahrán program a dojde tedy ke spuštění SAM-BA monitoru. Tento program je uložen již od výrobce v části ROM interní paměti. Výrobce udává existenci 2 různých verzí, ovšem ovládací příkazy zůstávají stejné.



Obrázek 10.1: Možnosti programování a ladění ARM

K nahrání programu nyní můžeme použít standardní sériový terminál, USB kabel ve spojení s aplikací SAM-BA nebo JTAG adaptér. Možnosti jsou přehledně znázorněny na obrázku 10.1. Z jednotek na čipu jsou nyní podstatné ARM9-Core (vlastní ALU a registry) a ICE-RT (In Circuit Emulator). Periferní jednotky ICE/JTAG, DBGU a USB převádí komunikaci na standardní protokoly, tj. JTAG, sériový UART a USB. ICE-RT umožňuje přístup na interní sběrnice (datová, adresová) a tím i přístup do interní paměti. Použitím jeho služeb je tak možné paměť programovat. To je vhodné využít jednak při zápisu programu do interní Flash nebo SRAM, dále pak při běhu programu pro debugování - vyčítání a modifikování hodnot v paměti. Přes ICE-RT je také možné přistupovat do celého adresového prostoru, tedy i do externí paměti (na obrázku není nakreslena).

Použití sériové linky je nejjednodušší možnost z hlediska zapojení. Uživatelské rozhraní je však velmi primitivní. Komunikaci řídí SAM-BA monitor, který se po inicializaci čipu ohlásí zasláním zprávy. Monitor poskytuje příkazy uvedené v tabulce 10.1. [6]

Výpis programování souboru na určené místo paměti a následné spuštění:

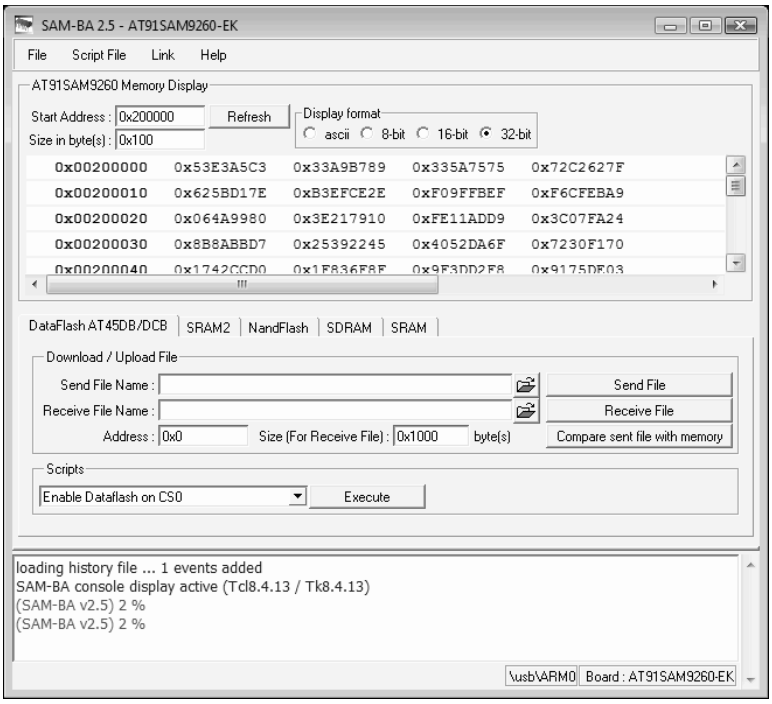
```
RomBOOT                 // Inicializační zpráva od SAM-BA monitoru
>V#                     // Příkaz pro zjištění verze SAM-BA
v1.4 Mar 02 2006 15:19:40
>S200000,#              // Příkaz pro nahrání souboru na adresu 200000h
CCCCCCCCCCCC            // Probíhá přenos souboru protokolem XModem ...
>G200200,#              // Příkaz pro spuštění programu od adresy 200200h
```

Uživatelsky přívětivější variantou je použití programu SAM-BA firmy Atmel, který poskytuje další možnosti, i když pracuje se stejnou množinou dostupných příkazů vykonatelných monitorem SAM-BA na čipu. Principem činnosti je emulace sériového portu přes USB. Je však nutné použít USB ovladače od firmy Atmel (atm6124). Ovladače jsou dodávány

Příkaz	Akce	Argument(y)
O	zapiš byte	Adresa, Hodnota#
o	čti byte	Adresa,#
H	zapiš half word	Adresa, Hodnota#
h	čti half word	Adresa,#
W	zapiš word	Adresa, Hodnota#
w	čti word	Adresa,#
S	pošli soubor	Adresa,#
R	přijmi soubor	Adresa, PočetBytů#
G	spusť program	Adresa#
V	zobraz verzi monitoru	-

Tabulka 10.1: Příkazy pro Boot Program SAM-BA

včetně zdrojových kódů. Na obrázku 10.2 je GUI aplikace SAM-BA. Aplikaci SAM-BA je možné volat i v dávkovém režimu s parametrem, udávajícím konfigurační soubor.



Obrázek 10.2: Uživatelské rozhraní SAM-BA

Použití JTAG

Z hlediska zaměření této práce je nejzajímavější možnost programování paměti pomocí rozhraní JTAG. JTAG je i v ARM obvodech prioritně určeno na testování a nastavování všech datových IO pinů, což v případě AT91SAM9260 vytváří boundary scan registr délky 484 bitů. JTAG je však možné použít i pro přístup a zápis na libovolnou adresu v adresovém prostoru a dále pak pro čtení registrů v jádře. To dává možnost programování pamětí a ladění aplikací běžících na jádře.

Pro služby ladění a programování jsou do jádra přidány dva JTAG registry, značené v dokumentaci jádra jako Scan Chain 1 a Scan Chain 2, viz obrázek 10.1.

Scan Chain 1 je napojen na jádro ARM9. Je používán pro čtení a zápis dat z registrů při ladění běžící aplikace. Tento registr zpřístupňuje sběrnice RDATA a WDATA, které obsahují data, jenž jsou přečtena z paměti, resp. určena pro zápis do paměti za běhu aplikace na jádře. Takto je lze modifikovat přes JTAG sběrnici. Součástí tohoto registru je také kopie právě prováděné instrukce INSTR. Scan Chain 1 se připojí mezi vývody TDI a TDO zadáním příkazu výběru do SCAN_N registru. Délka Scan Chain 1 je 67 bitů a obsahuje tyto informace:

INSTR[31:0], SYSPEED, WPTANDBKPT, unused bit, R(W)DATA[31:0] (SYSPEED a WPTANDBKPT jsou kontrolní bity.)

Scan Chain 2 je napojen na funkční jednotku zvanou Embedded ICE-RT a jeho obsah je dostupný standardně přes rozhraní JTAG. Po provedení odpovídajícího příkazu (zapsání příkazu výběru do SCAN_N registru) je Scan Chain 2 připojen mezi vývody TDI a TDO TAP kontroléru. Tento registr tedy zpřístupňuje služby ICE-RT jednotky a tím umožňuje přístup do adresového prostoru, tedy i programování. [1]

Konfigurace a použití programu OpenOCD

OpenOCD běží jako démon (služba), který komunikuje s JTAG adaptérem a zároveň naslouchá na portu TCP (výchozí hodnota 4444) . Při své inicializaci spustí také server gdbserver, který rovněž čeká na spojení na TCP portu (výchozí hodnota 3333). Gdbserver je používán při ladění běžící aplikace na mikrokontroléru a pro účely programování není používán. Při spuštění OpenOCD je nutné zadat inicializační parametry nebo jméno souboru, v němž jsou tyto parametry uloženy. Pro inicializaci se příkazy dělí dle terminologie OpenOCD na 3 skupiny: [26]

- interface (vlastnosti adaptéru)
- board (vlastnosti vývojové desky)
- target (vlastnosti procesoru)

Pro každou skupinu existují předpřipravené konfigurační soubory, které pokrývají velké množství dostupných adaptérů, vývojových desek a ARM čipů. Zadání první skupiny **interface** je povinné. Zde se udává použité zapojení adaptéru a vybírá vhodný ovladač. Také se definují TAP rozhraní na připojené desce, délka registru IR a provede se základní inicializace systému (reset TAP adaptéru, reset mikroprocesoru, reset všech obvodů, ...). V konfiguračním souboru pro **target** se zadávají údaje pro vlastní ARM čip, tedy jeho typ, endianita, dostupné oblasti interních pamětí SRAM a FLASH, pozice čipu v testovacím řetězci JTAG a pod. V konfiguračním souboru pro **board** jsou oblasti a typy externích pamětí a především posloupnost zápisů inicializačních instrukcí pro periférie v paměťovém prostoru. Tím se provede inicializace mikrokontroléru a jeho periférií (nastavení hodin, watchdogu, pamětí, IO pinů a pod.). Tato část je hardwarově závislá na konkrétním čipu a periférií ch na vývojové desce. Její správné odladění je také nejnáročnější ze všech částí konfigurace. Konkrétní případ použití OpenOCD a postup inicializace a programování je uveden v kapitole Programování ARM.

Kapitola 11

Programování FPGA Xilinx

11.1 Konfigurace z .bit a .bin souboru

Nástroj pro vytvoření konfiguračního řetězce bitgen od Xilinxu dokáže vytvořit různé varianty bitstreamu pro stejnou konfiguraci. Volba inicializačních možností závisí na konkrétním zapojení FPGA obvodu na desce a na požadovaném způsobu konfigurace FPGA. Jako výstup generuje program bitgen od Xilinxu konfigurační soubory BIN, BIT, SVF či XSVF. [31] Pokud požadujeme pouze samotný binární bitstream, zvolíme formát BIN nebo BIT. Bin formát obsahuje pouze konfigurační data, zatímco bit formát přidává na začátek souboru hlavičku s informacemi o tomto souboru. Hlavičku však lze odstranit a zbylá část je zcela shodná s bin formátem. Z hlavičky lze extrahovat originální název projektu, cílovou architekturu, datum a čas vytvoření a délku vlastního bitstreamu, který následuje za hlavičkou. Tyto informace jsou uloženy jako textové řetězce, takže není problém s jejich extrakcí. To je vhodné pro rychlou identifikaci neznámého souboru. Následující výpis ukazuje příklad získatelných informací z konkrétního bit konfiguračního souboru: [22]

```
Design Name: wires.ncd
Device: 3s50pq208
Date: 2010/12/06
Time: 21:31:37
Bitstream Length: 439264 bits
```

Získané údaje jsou výsledkem parsování hlavičky, jejíž hexa výpis následuje. V tabulce 11.1 je detailně vysvětlen význam jednotlivých položek v hlavičce.

```
0009 0FF0 0FF0 0FF0 0FF0 0000 0161 000A "................"
7769 7265 732E 6E63 6400 6200 0A33 7335 "wires.ncd....3s5"
3070 7132 3038 0063 000B 3230 3131 2F30 "0pq208....2011/0"
322F 3238 0064 0009 3135 3A35 303A 3239 "2/28....15:50:29"
0065 0000 D67C FFFF FFFF AA99 5566      "................"
```

11.2 Konfigurace z SVF souboru

Při použití formátu SVF dojde k zapouzdření bitstreamu a všech nutných instrukcí pro nahrání bitstreamu do jediného souboru. SVF soubor musí být vytvořen stejným progra-

0009, = délka hlavičky 9 Bytů
0FF0, 0FF0, 0FF0, 0FF0, 00 = hlavička (synchronizační značka)
00, 0161, = začátek sekce A (0x61 ASCII)
000A, = délka řetězce jména konfigurace, vč. ukončovací 0x00
7769, 7265, 732E, 6E63, 6400, = řetězec "wires.ncd0"
62 = začátek sekce B (0x62 ASCII)
00, 0A = délka řetězce jména čipu pro tuto konfiguraci, vč. ukončovací 0x00
33, 7335, 3070, 7132, 3038, 00 = řetězec "3s50pq2080"
63, = začátek sekce C (0x63 ASCII)
000B, = délka řetězce datumu vytvoření, vč. ukončovací 0x00
3230, 3131, 2F30, 322F, 3238, 00 = řetězec "2011/02/280"
64, = začátek sekce D (0x64 ASCII)
0009, = délka řetězce času vytvoření, vč. ukončovací 0x00
3135, 3A35, 303A, 3239, 00 = řetězec "15:50:290"
65, = začátek sekce E (0x65 ASCII)
0000, D67C, = délka vlastního bitstreamu
FFFF, FFFF, AA99, 5566 = začátek bitstreamu (synchronizační značka)

Tabulka 11.1: Parsování hlavičky souboru .bit

mem jako BIT, protože použité instrukce jsou závislé na výrobci FPGA. Při prozkoumání SVF souboru tak nalezneme instrukce JPROGRAM, CFG_IN a JSTART, jak je uvedeno v kapitole Obvody FPGA.

SVF Player xsvflib

Jakmile máme vytvořen SVF soubor, lze ho spustit jakýmkoliv softwarem, který jej dokáže interpretovat, tzv. SVF Player. Jednou z volně dostupných implementací SVF Playeru je knihovna xsvflib, kterou v navrhovaném programu využívám pro parsování SVF souborů. [32] Knihovna je napsána v jazyce C a není závislá na žádných dalších knihovnách, a to ani na standardních C knihovnách. To usnadňuje její přenositelnost na různé platformy včetně 16 bitových mikrokontrolérů. Knihovna obsahuje 3 samostatné části (SVF parser, XSVF parser, JTAG skener), které je možné volitelně zapnout při kompilaci knihovny.

Pro implementaci obslužného programu používajícího přeloženou knihovnu libxsvf je klíčová struktura libxsvf_host. Do této struktury se uloží ukazatele na uživatelem vytvořené funkce, které slouží pro ovládání konkrétního JTAG adaptéru (v mém případě FT2232 čipu). Tímto způsobem je zajištěna nezávislost knihovny na hardwaru. Struktura libxsvf_host obsahuje tyto položky:

```
struct libxsvf_host {
        int  (*setup)();
        int  (*shutdown)();
        void (*udelay)();
        int  (*getbyte)();
        int  (*sync)();
        int  (*pulse_tck)();
        void (*pulse_sck)();
        void (*set_trst)();
        int  (*set_frequency)();
        void (*report_tapstate)();
        void (*report_device)();
        void (*report_status)();
        void (*report_error)();
        void*(*realloc)();
        enum libxsvf_tap_state tap_state;
        void *user_data;
};
```

Z těchto funkcí jsou některé volitelné (rozšiřující). Pro základní použití je nutné implementovat tyto:

int setup(struct libxsvf_host *h)
Slouží k inicializaci adaptéru při spuštění programu.

int shutdown(struct libxsvf_host *h)
Slouží k deaktivaci adaptéru při ukončování programu. Je garantováno, že po volání této funkce nedojde k dalšímu přístupu na hardware.

void udelay(struct libxsvf_host *h, long usecs, int tms, long num_tck)
Zpozdí další zpracování programu o daný počet mikrosekund. Pokud je parametr num_tck nenulový, proběhne následně zadaný počet hodinových impulsů na TCK se vstupem TMS nastaveným dle parametru tms.

int getbyte(struct libxsvf_host *h)
Načte další byte ze vstupního SVF/XSVF souboru.

void pulse_tck(struct libxsvf_host *h, int tms, int tdi, int tdo, int rmask, int sync)
Toto je klíčová funkce knihovny, provádí tyto úkoly:

1. Nastaví TMS na hodnotu dle parametru tms
2. Nastaví TDI na hodnotu dle parametru tdi
3. Vytvoří hodinový impuls (1-0-1) na výstupu TCK.
4. Zkontroluje výstup z TDO pinu, zda souhlasí s očekávaným stavem, daným parametrem tdo

int set_frequency(struct libxsvf_host *h, int v)
Nastaví frekvenci JTAG sběrnice na hodnotu v [Hz].

void *realloc(struct libxsvf_host *h, void *ptr, int size, enum libxsvf_mem which)
Funkce pro dynamickou alokaci paměti. Na systémech se standardní C knihovnou se zde použije funkce realloc().

Jakmile je struktura libxsvf_host vytvořena a naplněna odkazy na uživatelem definované funkce, je zavolána vstupní funkce knihovny libxsvf, které se jako argument předá struktura libxsvf_host a mód činnosti (SVF, XSVF, JTAGSCAN). Nyní proběhne vlastní zápis SVF souboru na JTAG sběrnici a je zobrazena zpráva o úspěchu.

Součástí projektu libxsvf jsou i ukázkové kódy pro použití knihovny s čipem FT2232H a ovladači libftdi pro tento čip. Při použití mého programátoru dokáže ukázkový program správně detekovat FPGA obvod podle IDCODE, při pokusu o interpretaci SVF souboru však nastane chyba a SVF soubor není do cílového FPGA správně nahrán. Chyba je způsobena nesprávnou bitovou posloupností při komunikaci směrem z FPGA zpět do FT2232 na pinu TDO. Příčinu chyby jsem nezjistil, je možné, že ji způsobuje rušení při přenosu mezi vodiči programátoru, což vede k pozměnění některého z přenášených bitů a tedy výsledek lišící se od očekávaného. Následuje výpis ukázkového programu xsvftool-ft2232h. [32] Testováno s FPGA XC3S400.

```
# ~/libxsvf$ ./xsvftool-ft2232h -c
idcode=0x0141c093, revision=0x0, part=0x141c, manufactor=0x049

# ~/libxsvf$ ./xsvftool-ft2232h -s 3slc_led.svf
[svf.c:330] TDO mismatch.
Error while playing SVF file ‘3slc_led.svf’.
```

Kapitola 12

Programování ARM

Navržený programátor je kompatibilní se softwarem OpenOCD. Vyvíjet vlastní program pro komunikaci s čipy ARM je v současnosti zbytečné úsilí, mnohem vhodnější je zaměřit se na spolupráci s OpenOCD, který se stává Open-source standardem pro tuto architekturu. I přes poměrně rychlý vývoj a množství vykonané práce zůstává OpenOCD poměrně uživatelsky nepřívětivý z hlediska konfigurace vývojového řetězce (JTAG adaptér + vývojová deska + ARM obvod). Jakmile se však povede nastavit správnou konfiguraci, poskytuje OpenOCD velmi bohaté možnosti.

Pro testování jsem používal vývojovou desku od firmy Olimex SAM9-L9260 s čipem ARM AT91SAM9260, JTAG adaptér ARM-USB-TINY [21] a samozřejmě mnou navržený adaptér. ARM-USB-TINY je osazen obvodem FT2232C, můj adaptér FT2232H. Ačkoliv jsem předpokládal zpětnou kompatibilitu FT2232H, není tomu tak. Hlavním rozdílem u verze H je zvětšení maximální rychlosti z 6 MHz na 30 MHz a přidání podpory pro adaptive clocking. Také ovladače pro operační systém jsou rozdílné.

Adaptive Clocking

Kmitočty, na nichž pracují JTAG a vlastní ARM jádro obvodu jsou z podstaty JTAG nezávislé. Tedy jádro běží na jiné frekvenci (asynchronně) než je frekvence TCK na JTAG portu. Tato vlastnost je vhodná pro základní použití JTAG jako boundary scan. Pokud však chceme JTAG použít pro komunikaci s jádrem (při debugování), potřebujeme obě frekvenční domény synchronizovat. ARM jádro může běžet na rychlosti 32KHz nebo i několik set MHz. Pokud bychom adaptive clocking (AC) nepoužili, musí JTAG běžet trvale na nejnižší z těchto frekvencí. ARM čipy AC signál tedy mají obvykle vyveden na pouzdro a na desce je připojen na konektor společně s JTAG signály. Adaptive Clocking signál je značen RTCK a jeho směr je od ARM jádra k JTAG adaptéru. Adaptér nastaví hodnotu High na výstupu TCK a čeká na potvrzení od ARM na signálu RTCK. Jakmile ho obdrží, změní úroveň TCK na Low a tím dojde k vytvoření kompletního impulsu na TCK. Poté opět adaptér čeká na potvzení od ARM na vstupu RTCK a nastaví TCK na High. Tímto způsobem je vytváření hodinových impulsů na TCK řízeno ARM jádrem. Pro využití AC musí tuto vlastnost podporovat cílová platforma, JTAG adaptér i ovládací software. [26]

Při použití čipu od FTDI je nutné použít variantu FT2232H nebo FT4232H. Ostatní varianty AC nepodporují. OpenOCD detekuje typ připojeného čipu a podle toho zapne či vypne použití AC. Při počátečním návrhu programátoru jsem použil jako základ schéma adaptéru usbjtag-1 [26], který však používá čip FT2232C. Tento čip AC nepodporuje, komunikace tedy běží na nejnižší společné frekvenci, což je 33KHz. Při použití FT2232H je

nutné pro povolení AC provést hardwarovou úpravu schématu usbjtag-1. Konkrétně jde o propojení pinu RTCK JTAG konektoru (pin číslo 11) na pin čipu FT2232H GPIOL3 (pin číslo 24). Zapojení je zřejmé ze schématu v Příloze C. Bez tohoto propojení neproběhne mezi FT2232H a ARM jádrem žádná komunikace a tedy ani nebude detekován připojený ARM!

Funkce AC je dostupná jen při použití příkazového procesoru MPSSE a implicitně je vypnuta. Pro její aktivaci je nutné zaslat příkaz „0x96“ pomocí funkce FTDI_Write(). [12] Volání tohoto příkazu je obsaženo ve zdrojových kódech OpenOCD a je automaticky provedeno při detekci čipu FT2232H nebo FT4232H.

Pokud chceme v OpenOCD povolit AC, použijeme v konfiguračním souboru volbu **jtag_rclk** [**freq**]. Na místo parametru freq se doplní frekvence JTAG, která se použije jako implicitní, pokud nějaký prvek vývojového řetězce nepodporuje AC. Pokud je AC podporováno, bude se frekvence měnit dynamicky podle aktuálního kmitočtu jádra ARM. Výhoda AC se projeví především u zápisu či čtení obsahů pamětí, což je běžná činnost při programovování. Při ladění není použití AC tak významné.

12.1 Úložiště pro program

Při programování obvodů ARM osazených na (vývojové) desce, bývají k dispozici tyto paměti:

1. interní flash paměť
2. externí flash paměť typu NOR
3. externí flash paměť typu NAND
4. externí flash paměť typu SPI (Serial Peripheral Interface)

Paměti NAND a NOR používají ke své adresaci a přenosu dat paralelní sběrnice. Na úrovni hardwaru se přístup k nim liší, je však standardizován použitím příkazového procesoru umístěného přímo na čipu paměti. Nejpoužívanějším standardem je CFI (Common Flash Interface). Paměti SPI používají sériový přístup jak pro adresu a data, tak i pro příkazy. Příkazy pro řízení paměti zde nejsou standardizovány. OpenOCD tedy podporuje pouze paralelní paměti NAND a NOR. [26] Příkazové procesory na čipu paměti musí být před přístupem do paměti inicializovány sekvencí příkazů. Ty jsou pro nejběžnější vývojové desky uvedeny ve vzorových konfiguračních souborech dodaných s OpenOCD. Dále se jimi nebudu zabývat, protože přímo nesouvisejí s JTAG. Z hlediska připojení programátoru jsou v OpenOCD podstatné následující parametry, které se uvedou do konfiguračního souboru a jsou načteny při spuštění OpenOCD:

```
# Interface, určuje použitý čip na programátoru.
# Jednotlivé členy této rodiny (C,H,...) si již OpenOCD identifikuje sám.
interface ft2232

# Popisný řetězec FTDI čipu, lze jej změnit zápisem do EEPROM paměti
# připojené k FT2232
ft2232_device_desc "Dual RS232-HS A"

# Rozložení vývodů, tj. propojení pinů čipu na piny JTAG konektoru
ft2232_layout "usbjtag"

# VID a PID použitého čipu
ft2232_vid_pid 0x0403 0x6010

# Zpoždění mezi resetem desky a začátkem komunikace po JTAG
jtag_nsrst_delay 200
jtag_ntrst_delay 200

# Načtení dalšího souboru udávajícího vlastnosti desky,
# periférií a čipu ARM
source [find board/olimex_sam9_l9260.cfg]
```

Flash paměť NAND

Jako ukázkový příklad programování po JTAG uvedu posloupnost kroků nutnou k vymazání a nahrání binárního souboru do paměti NAND Flash. Použil jsem binární soubor dodávaný firmou Olimex k vývojové desce SAM9-L9260. Jde o „nandflash_at91sam9260ek.bin“, který obsahuje obraz OS Linux zkompilovaného pro uvedenou desku.

1. Připojíme se k běžící instanci OpenOCD

   ```
   telnet localhost 4444
   ```

2. Zastavíme procesor na čipu ARM

   ```
   > halt
   target state: halted
   target halted in ARM state due to debug-request, current mode: Supervisor
   cpsr: 0x80000013 pc: 0xc0192dbc
   MMU: enabled, D-Cache: enabled, I-Cache: enabled
   ```

3. Ověříme přítomnost NAND flash a vypíšeme její parametry

```
> nand probe 0
NAND flash device 'NAND 512MiB 3,3V 8-bit' found

> nand list
#0: NAND 512MiB 3,3V 8-bit (Samsung) pagesize: 2048, buswidth: 8,
blocksize: 131072, blocks: 4096
>
```

4. Vymažeme paměť. To je v případě NAND pamětí nutné před každým programováním. Během mazání se vypisují vadné bloky paměti. (Vadné bloky jsou běžné v NAND pamětech).

```
> nand erase 0
bad block: 0
bad block: 10
bad block: 32
....
```

5. V posledním kroku zapíšeme nový obsah do paměti NAND.

```
> nand write 0 nandflash_at91sam9260ek.bin
```

12.2 Přímý přístup do adresového prostoru přes JTAG

Pomocí OpenOCD lze zapisovat na libovolné místo v paměťovém prostoru procesoru ARM. To se využívá především pro programování pamětí, jak je uvedeno v předchozím odstavci. Další využití je také debugování a inicializace periferií. Paměti typu NAND, NOR i SPI poskytují vlastní pohled na uložená data pomocí (obvykle) standardního rozhraní. Před započetím komunikace je třeba do jejich řadiče odeslat inicializační posloupnost. Tato inicializace musí být samozřejmě provedena i před programováním nových dat, proto ji OpenOCD také podporuje. Zápis do adresových prostorů je velmi jednoduchý, použije se příkaz **mww** [**addr**] [**word**] pro zápis a **mdw** [**addr**] [**word**] pro čtení 32 bitového slova. [26] Tyto příkazy zapíšeme do konfiguračního souboru OpenOCD a budou vykonány po spuštění.

Pomocí příkazů mww a mdw lze tedy zapisovat přímo do registrů na procesoru a v periferiích. Jako ukázku možností uvedu program pro střídavé blikání dvou LED diod umístěných na vývojové desce Olimex SAM9-L9260. Dioda PWR_LED (žlutá) je připojena na pin 9 portu PIOA, dioda STAT (zelená) je připojena na pin 6 portu PIOA. Bázová adresa portu PIOA v mikrokontroléru AT91SAM9260 je 0xFFFFF400.

```
# Použité piny portu PIOA:
# ------------------------------------------------------
# |... | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
# ------------------------------------------------------
#        ^              ^
#        |(512)         |(64)

# Zakáže možnost signalizace přerušení na linkách P9 a P6
mww 0xFFFFF444 576

# Nastaví linky P9 a P6 jako výstupní
mww 0xFFFFF410 576

# Zápis na výstupní piny: zelená LED nesvítí, žlutá LED svítí
mww 0xFFFFF430 576

# Zpoždění 500ms
sleep 500

# Zápis na výstupní piny: zelená LED svítí, žlutá LED nesvítí
mww 0xFFFFF434 512
```

Kapitola 13

Ovládací program

Ovládací program pro demonstraci použití navrženého programátoru je vytvořen v prostředí MS Visual Studio 2005. Během vývoje programátoru vzniklo několik samostatných jednoúčelových programů, jejichž některé výstupy byly použity v předchozích kapitolách. Pro sjednocení ovládání jsem všechny moduly spojil do jednoho programu se standardním ovládáním a společným konfiguračním souborem. Ovládací program komunikuje jen s FPGA obvody Xilinx a to řady Spartan3. Pro ostatní výrobce by bylo nutné upravit kódy instrukcí JTAG. Toto je možné řešit parsováním BSDL souborů od jednotlivých výrobců. V těchto souborech jsou všechny potřebné údaje pro komunikaci se součástkou. Navržený program však toto nepodporuje.

Jednotlivé moduly programu slouží k těmto činnostem:

- identifikace vlastností připojeného FTDI čipu na adaptéru
- načtení identifikačního řetězce IDCODE z JTAG součástky
- programování FPGA Xilinx pomocí BIT datastreamu
- programování FPGA Xilinx pomocí SVF souboru

V následujících odstavcích jsou uvedeny jen výstupy aplikace pro každou operaci společně s vysvětlujícím textem. Kompletní zdrojové kódy v jazyce C s podrobným komentářem a spustitelná aplikace jsou uloženy na doprovodném CD-ROM.

Program komunikuje s čipem na programátoru s použitím ovladačů ftd2xx od firmy FTDI, je tedy nutné je před použitím nainstalovat. Pro parsování konfiguračního souboru se použije knihovna libconfig a pro parsování příkazové řádky knihovna argtable. Obě jsou přiloženy k programu a nepotřebují další instalaci. Pro nastavení počátečních hodnot je použit konfigurační soubor config.txt. Nastavuje se v něm typ čipu na programátoru, kmitočet sběrnice JTAG a počáteční hodnoty a směr toku dat univerálních IO pinů na čipu. Na výpisu je vidět použité nastavení. Pokud údaje chybí nebo jsou nesprávné, použije se výchozí nastavení.

```
# Konfigurační soubor pro programátor jtagprog

# Typ připojeného čipu na adaptéru (FT2232, FT2232H)
FTDIChip = "FT2232H";

# Frekvence sběrnice JTAG při komunikaci s programátorem (v Hz)
Frequency = 800000;

# Konfigurace směru univerzálních IO pinů (IN, OUT)
Direction = {
        LowIOPin1 = "IN";
        LowIOPin2 = "IN";
  ....
};

# Konfigurace počátečního stavu univerzálních IO pinu (LOW, HIGH)
# Volba se projeví jen u pinů nastavených jako Direction = OUT
State = {
        LowIOPin1 = "LOW";
        LowIOPin2 = "LOW";
  ....
};
```

13.1 Menu programu

Název programu je jtagProg.exe. Po spuštění je mu třeba zadat parametr podle požadované činnosti. Pro vypsání nápovědy se zadá parametr –help:

```
jtagProg.exe --help
Pouziti: jtagprog [-h] [--idd] [--idm] [--inf] [--prb] [--prs] [<soubor>]
Tento program slouzi ke komunikaci se soucastkou pres JTAG sbernici.
Umoznuje vypis informaci o soucastce, o programovacim adapteru
a programovani FPGA obvodu soubory typu BIT a SVF
  --idd                     IDCODE s pouzitim FTCJTAG.DLL
  --idm                     IDCODE s pouzitim MPSSE
  --inf                     informace o cipu na programatoru
  --prb                     programuj z .bit souboru <soubor>
  --prs                     programuj z .svf souboru <soubor>
  <soubor>                  bit/svf soubor
  -h, --help                vytiskne tuto napovedu
```

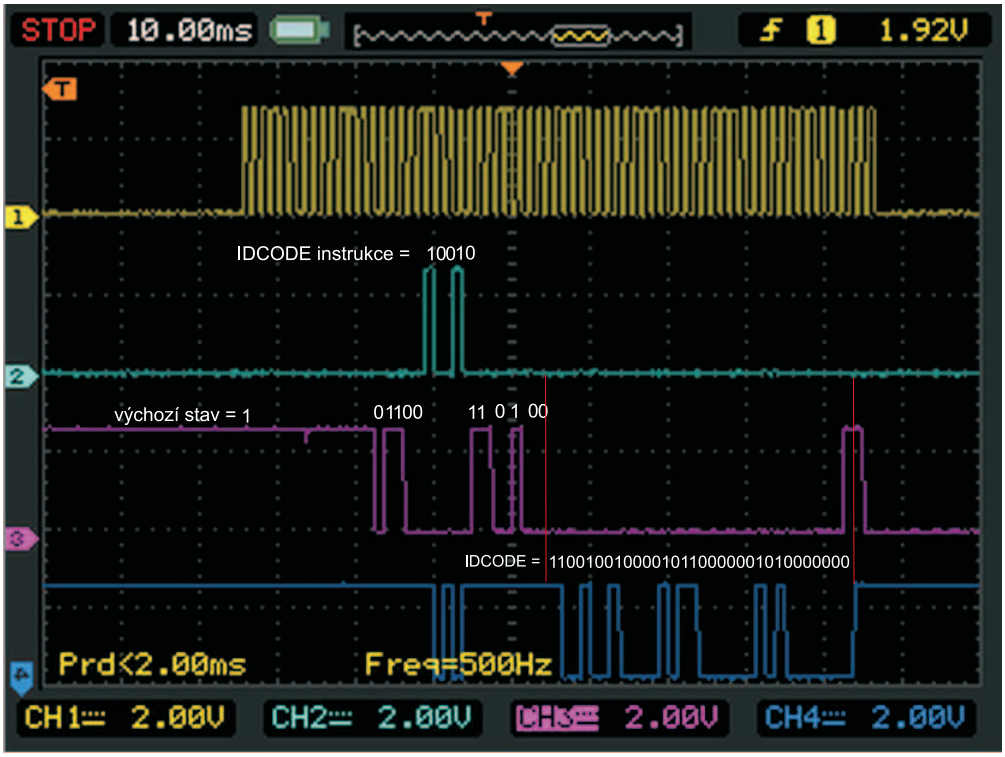
13.2 Identifikace připojené součástky

jtagprog –idd, jtagprog –idm
Identifikace spočívá ve vyčtení obsahu registru IDCODE. Ten obsahuje identifikační řetězec, který jednoznačně určuje výrobce a typ součástky. Pro vyčtení jsou použity dvě varianty. První použije pro komunikaci se součástkou přímo binární kódy příkazového procesoru MPSSE (parametr –idm). Druhá varianta využije funkce poskytované knihovnou FTCJTAG.DLL (parametr –idd). Výsledek je však stejný pro obě varianty:

```
jtagProg.exe --idd
FTDIChip typ: FT2232H
JTAG frekvence: 500
IDCODE s pouzitim FTCJTAG.DLL
Aktualni kmitocet JTAG je: 499 Hz
Zarizeni FTDI2232H inicializovano.

IDCODE = 1 41 c0 93

---------------------------------
jtagProg.exe --idm
FTDIChip typ: FT2232H
JTAG frekvence: 500
IDCODE s pouzitim MPSSE...
Nastavuji MPSSE...
MPSEE ready.

IDCODE = 1 41 c0 93
```

Použití varianty s knihovnou FTCJTAG.DLL je poměrně jednoduché. Pomocí knihovní funkce JTAG_Write zadáme instrukční kód operace IDCODE a poté použitím instrukce JTAG_Read přečteme výsledek z JTAG datového registru součástky. Celá nízkoúrovňová komunikace je tak skryta. To je výhodné z hlediska odolnosti proti chybám a snadnosti použití. Pokud však chceme manipulovat s JTAG TAP na nejnižší úrovni, je nutné použít přímo příkazy FTDI MPSSE. Ty také interně používá knihovna FTCJTAG.DLL. Při ladění komunikace přes MPSSE je nutné použít osciloskop či logický analyzátor, protože každý jednotlivý bit a jeho umístění je podstatné a žádná jiná metoda ladění není dostupná. Na obrázku 13.1 je oscilogram zobrazující komunikaci mezi čipem FT2232H na mém adaptéru a FPGA obvodem Spartan3 XC3S50 na FitKitu. Signál číslo 1 znázorňuje hodinový signál na TCK. Signál číslo 2 znázorňuje vstupní data na pinu TDI. Signál číslo 3 znázorňuje průběh na pinu TMS. A konečně signál číslo 4 zachycuje výstupní data na pinu TDO.

Protože jsem nenalezl v dokumentaci od FTDI přehledné vysvětlení průběhů a jejich závislostí, budu oscilogram podrobně interpretovat. Na pin TCK je přiváděn hodinový signál, zde se konkrétně jedná o 500Hz. Jeho nastavení je závislé na hodnotě v konfiguračním souboru programátoru. Při připojení hodinového signálu na TCK musí být řídící vstup TMS nastaven na hodnotu High a držen v ní alespoň po dobu 5 taktů. Tím dojde k přechodu TAP automatu do stavu Test-Logic-Reset i když počáteční hodnota byla libovolný stav. Tato inicializace je naprosto nezbytná, jinak se TAP nachází v nedefinovaném stavu a další



Obrázek 13.1: Průběh impulsů na JTAG sběrnici

instrukce se budou provádět chybně. Nyní se přesuneme ze stavu Test-Logic-Reset do stavu Shift-IR. To provedeme posloupností „01100“ na TMS pinu. (Všechny možné přechody mezi stavy TAP a jim příslušející logické hodnoty na vstupu TMS jsou znázorněny na obrázku 3.5). Ve stavu Shift-IR přivedeme na vstup TDI instrukci IDCODE (v případě Spartan 3 je to „10010“). Následně pomocí instrukcí na TMS pinu projdeme cestu v TAP automatu: Exit-IR, Update-IR, Select-DR-Scan, Capture-DR, Shift-DR. Nyní s každým přivedeným TCK impulsem bude na výstup TDO vysunut jeden bit z IDCODE Spartana3 (32 bitové číslo). Poslední impuls „1“ na TMS značí konec čtení dat z datového registru. Tím je operace čtení IDCODE ukončena, obdrželi jsme výsledek „11001001000010110000001010000000“, který odpovídá katalogové hodnotě IDCODE.

13.3 Informace o čipu FT2232 na adaptéru

jtagprog –inf
Funkce zobrazí informace o připojeném čipu, které dokáže zjistit ovladač od FTDI.

```
jtagProg.exe --inf
FTDIChip typ: FT2232H
JTAG frekvence: 500
informace o cipu na programatoru
Hledam pripojene FTDI zarizeni...
Pocet nalezenych zarizeni: 2. (na cipu FT2232 a FT2232H jsou dve zarizeni)
Informace o pripojenem FTDI cipu, port 0:
----------------------------------------
Flags value =           2
Type value =            6
Device ID =             67330064
Device location ID =    785
Device serial number =  A
Device description =    Dual RS232-HS A

Informace o pripojenem FTDI cipu, port 1:
----------------------------------------
Flags value =           2
Type value =            6
Device ID =             67330064
Device location ID =    786
Device serial number =  B
Device description =    Dual RS232-HS B
```

13.4 Programování ze souboru .bit

jtagprog –prb
Nejdříve je provedeno parsování hlavičky a současně výpis obsažených informací na výstup. Poté je bitstream ležící za hlavičkou odeslán do FPGA. Rychlost zápisu závisí na nastavené rychlosti JTAG sběrnice, která je uvedena v konfiguračním souboru. Pro dobu programování řádově v sekundách musí být rychlost minimálně 0.5 MHz. Maximální dostupná rychlost je 6MHz pro FT2232D a 30MHz pro FT2232H. Použitelná rychlost však závisí na maximální rychlosti připojeného JTAG zařízení (obvykle maximálně jednotky MHz).

```
jtagProg.exe --prb counter.bit
FTDIChip typ: FT2232H
JTAG frekvence: 500000
programovani ze souboru bit...
Aktualni kmitocet JTAG je: 491803 Hz
Zarizeni FTDI2232H inicializovano.

Design Name:    fpga.ncd
Device:         3s50pq208
Date:           2011/03/03
Time:           15:01:10
Stream Length:  439264 bits

Programovani dokonceno.
```

Modul pro programování je napsán za použití FTDI knihovny FTCJTAG.DLL, což umožňuje snadnou komunikaci s JTAG pomocí operací read a write. Při konfiguraci FPGA přes JTAG je nutné mít předem vytvořený bit soubor a znát posloupnost instrukcí daného obvodu pro nahrání a spuštění konfigurace. Tuto posloupnost jsem nenašel v žádných firemních materiálech Xilinxu. Velmi stručná poznámka o JTAG programování bitstreamem je v aplikační poznámce XAPP188 [29]. Je zde ukázka posloupnosti JTAG instrukcí a přiváděných dat. Tato posloupnost je však vztažena jen na Xilinx Spartan2 obvody. Pro Spartan3 je k dispozici uživatelská příručka UG332 [31]. Zde se již posloupnost instrukcí neuvádí, navíc přibyly další instrukce pro programování. Na základě informací z těchto zdrojů a studiem struktury SVF souborů vytvořených programem bitgen od Xilinxu, jsem dospěl k následující posloupnosti pro nahrání a spuštění bitstreamu:

- zápis instrukce "idcode"
- přečtení odpovědi z datového registru součástky
- zápis instrukce "bypass"
- zápis instrukce "jprogram"
- zápis instrukce "cfg_in"
- čekací cyklus spočívající v zaslání 14000 hodinových pulsů na TCK vstup
- zápis instrukce "cfg_in"
- zápis 95 bitů nulových dat do datového registru
- **zápis vlastního bitstreamu**
- zápis instrukce "jstart"
- spuštění Startup sekvence spočívající v zaslání 12 hodinových pulsů na TCK vstup
- zápis instrukce "bypass"
- zápis instrukce "bypass"

- zápis instrukce "jstart"
- spuštění Startup sekvence spočívající v zaslání 12 hodinových pulsů na TCK vstup
- zápis instrukce "bypass"

Provedením těchto instrukcí se provede reset obvodu, dojde k nahrání bitstreamu a nakonec FPGA signalizuje dokončení konfigurace nastavením pinu DONE na hodnotu High (viz kapitola Obvody FPGA). Ke spuštění konfigurace však při praktickém testování nedojde. Zde je problém zjistit, kde nastala v průběhu konfigurace chyba. Celý proces konfigurace je totiž velmi komplexní [31] a jeho znalost je nutná k úspěšnému oživení programátoru. Navíc jej výrobce (Xilinx) mění i mezi čipy stejné rodiny, čímž nepřímo vyžaduje používání jím dodávaného proprietárního software nebo použití vyšších standardních formátů jako je SVF.

13.5 Programování ze souboru .svf

jtagprog –prs
Pro vlastní parsování jsem použil Open-source knihovnu libxsvf. Její popis je uveden v kapitole Programování FPGA Xilinx. Knihovna se snaží být platformově nezávislá, je tedy napsána v jazyce C a nepoužívá žádné další knihovny. Při přenesení na konkrétní platformu je tedy třeba dopsat implemetace funkcí pro komunikaci se zvoleným adaptérem - ty jsou již ovšem závislé na konkrétním adaptéru a na platformě. Implemetaci jsem doplnil pro použití ovladačů FTDI ftd2xx na Windows. Celou strukturu zdrojových kódů pro libxsvf jsem pak exportoval do podoby DLL knihovny (pro dynamické linkování při spuštění) a LIB knihovny (pro statické linkování při překladu), pojmenovaných xsvf.dll a xsvf.lib. Na přiloženém CDROM je projekt v MS Visual Studiu, nazvaný xsvf, z nějž jsem tyto knihovny vygeneroval.

Pro testování po konverzi na platformu Windows jsem použil stejný hardware jako v ukázkovém příkladě (kapitola Programování FPGA Xilinx). Bohužel se zde projevila stejná chyba, signalizující neplatný výstup z FPGA směrem zpět do FT2232H. Vzhledem k tomu, že chyba se projevila při použití jak ukázkového příkladu na platformě Linux, tak i v konvertovaném programu na platformě Windows, může se jednat o hardwarovou chybu programátoru (nejspíše rušení) nebo o chybu v návrhu samotné knihovny.

13.6 Přehled zdrojových souborů projektu

Pro jednodušší orientaci v souborech projektu jtagProg jsou v následující tabulce uvedeny vytvořené zdrojové soubory a jejich zodpovědnost za vykonávané funkce.

main.c	parsování příkazového řádku, volání jednotlivých modulů
hwinfo.c	metody pro výpis informací o čipu na adaptéru a o programované součástce
ftdiinit.c	nízkoúrovňová inicializace čipů FT2232 a FT2232H
config.c	nastavení hardwaru podle nastavení v config.txt
bitprog.c	programování FPGA z BIT souboru (bitstreamu)
svfprog.c	programování FPGA z SVF souboru
error.c	obsluha chybových hlášení
config.txt	konfigurační soubor pro nastavení vlastností adaptéru

Kapitola 14

Závěr

Cílem této práce byl návrh univerzálního programátoru pro rozhraní JTAG. Po prozkoumání literatury jsem však zjistil, že implementace podpory JTAG programování je rozdílná u každého výrobce a dosažení univerzálnosti je téměř nemožné. Omezil jsem se tedy na platformy ARM a FPGA, které jsem mohl prakticky testovat. Výsledný produkt splňuje většinu bodů zadání, není však v podobě vhodné pro běžné používání.

V textu je uvedeno praktické použití JTAG na platformách ARM a FPGA s popisem nízkoúrovňových detailů, které obvykle uživateli zůstávají skryty, pokud používá hotové softwarové a hardwarové nástroje. Jako největší problém při práci na projektu se ukázalo oživování hardwarového zapojení, které si i při relativní jednoduchosti vyžádalo velké množství času. Z hlediska hardwarového zde uvedené zapojení není nijak inovativní, jediné zajímavé rozšíření oproti obdobným řešením je podpora Adaptive Clocking, což je výhoda pro programování ARM obvodů. Pro tento účel lze navržený programátor používat, je kompatibilní také s perspektivním projektem OpenOCD. Dále je v práci rozebrán postup programování bitstreamu přes JTAG, včetně pokusu o programovou implementaci. Programování bitstreamu se zdá být docela komplikovaný problém, protože dokumentace od výrobce je slabá a nezbývá než experimentovat. Zatím jsem nedosáhl žádaného výsledku, je zde tedy prostor pro další vylepšování. Přínosem je popis předpokládaného algoritmu a jeho implemetace, i když zatím nekompletní. Dalším výstupem je převedení implementace SVF parseru xsvflib na platformu Windows s podporou pro ovladače od FTDI.

Vytvořený programátor byl testován v projektu OpenOCD za použití platformy ARM a dále na nízkoúrovňovou komunikaci s FPGA obvody. Oba tyto testy byly úspěšné. Programování FPGA je zatím neúspěšné, není zřejmé zda je chyba v návrhu hardwaru nebo softwaru.

Na tuto práci by bylo možné navázat podrobným postupem vedoucím k řešení programování bitstreamu do FPGA přes JTAG. Jako hardware by však bylo vhodné použít již ověřených zapojení programovacích adaptérů z jiných projektů, aby při ladění nevznikaly pochybnosti zda je problém v hardwarové či softwarové části.

Pri dalším návrhu programátoru by bylo vhodné použít rozšiřující se standard IEEE 1532, který je oproti IEEE 1149.1 skutečně standardem pro programování přes JTAG. Tím by bylo dosaženo opravdu univerzálního programátoru napříč platformami. Za zvážení stojí i možnost použití čipu Vinculum.

Literatura

[1] ARM Limited: ARM9EJ-S Technical Reference Manual. 2002, elektronická publikace.

[2] Asset InterTech: Serial Vector Format. 1999, elektronická publikace.

[3] Asset InterTech: IEEE 1532 Concurrent In-System Configuration. 2008, informační brožura.

[4] Asset InterTech: Boundary-Scan Tutorial. 2009, elektronická publikace.

[5] Atmel: AVR910: In-System Programming. 2008, aplikační poznámka.

[6] Atmel Corporation: AT91 ARM Thumb Microcontrollers. AT91SAM9260. 2008, katalogový list.

[7] Axelson Jan: *USB Complete. The developer's guide.* Lakeview Research LLC, 2009, ISBN 978-1-931448-08-6.

[8] Bleeker Harry, Eijnden Peter, Jong Frans: *Boundary-Scan Test. A practical approach.* Kluwer Academic Publisher, 1993, ISBN 0-7923-9265-5.

[9] JTAG Pinouts. 2011, [online] [cit. 2011-05-10].
URL http://www.jtagtest.com/pinouts/

[10] Freescale Semiconductor: Introduction to HCS08 Backgrounded Debug Mode. 2006, aplikační poznámka.

[11] FTDI Limited: AN110 Programmer's Guide for High Speed FTCJTAG DLL. 2009, aplikační poznámka.

[12] FTDI Limited: FT2232H Dual High Speed USB to Mulptipurpose UART/FIFO IC. 2010, katalogový list.

[13] FTDI Limited: Software Application Development D2XX Programmer's Guide. 2010, elektronická publikace.

[14] FTDI Limited: AN108 Command Processor for MPSSE and MCU Host Bus. 2011, aplikační poznámka.

[15] FTDI Products. 2011, [online] [cit. 2011-05-10].
URL http://www.ftdichip.com/FTProducts.htm

[16] Jacobson G. Neil: IEEE Std P1532 - A New Standard for 1149.1-based In System Configuration. 1999, firemní prezentace IEEE.

[17] Kramara JTAG+ adapter. 2011, [online] [cit. 2011-05-10].
URL http://www.kramara.com/

[18] libFTDI - FTDI USB driver with bitbang mode. 2011, [online] [cit. 2011-05-10].
URL http://www.intra2net.com/en/developer/libftdi/index.php

[19] libusb-win32: A port of the USB library libusb. 2011, [online] [cit. 2011-05-10].
URL http://sourceforge.net/projects/libusb-win32/

[20] Microchip: In-Circuit Serial Programming Guide. 2003, aplikační poznámka.

[21] Olimex JTAG adapters. 2011, [online] [cit. 2011-05-10].
URL http://www.olimex.com/dev/index.html

[22] Xilinx BIT bitstream files. 2011, [online] [cit. 2011-05-10].
URL http://www.pldtool.com/pdf/fmt_xilinxbit.pdf

[23] Schauer Stefan: SpyBiWire. 2005, firemní prezentace Texas Instruments.

[24] Texas Instruments: IEEE Std. 1149.1 (JTAG) Testability Primer. 1997, elektronická publikace.

[25] Texas Instruments: MSP430 Programming Via the Bootstrap Loader. 2010, aplikační poznámka.

[26] The OpenOCD Project: OpenOCD User's Guide. 2010, elektronická publikace.

[27] UrJTAG - Universal JTAG library. 2011, [online] [cit. 2011-05-10].
URL http://urjtag.org/

[28] Xilinx: JTAG Programmer Guide. 2000, elektronická publikace.

[29] Xilinx: XAPP188 Configuration and Readback of Spartan2. 2002, aplikační poznámka.

[30] Xilinx: SVF and XSVF File Formats for Xilinx Devices. 2009, aplikační poznámka.

[31] Xilinx: UG332 Spartan3 Generation Configuration User Guide. 2009, elektronická publikace.

[32] LibXSVF - A library for implementing SVF and XSVF JTAG players. 2011, [online] [cit. 2011-05-10].
URL http://www.clifford.at/libxsvf/

Příloha A

Adaptéry založené na FT2232

USB převodník od firmy FTDI FT2232 je v současné době nejpoužívanějším obvodem při návrhu JTAG i jiných adaptérů. V následujícím přehledu jsou uvedeny běžně dostupné JTAG adaptéry společně s jejich vlastností. Právě na základě tohoto srovnání vznikly požadavky na vytvářený programátor.

Turtelizer

- podrobnosti na webu: http://www.ethernut.de/en/hardware/turtelizer/
- napěťový rozsah připojeného zařízení: 1,65 - 5,5V
- výstup na 10 pinový konektor ARM (viz Příloha B)
- obsahuje převodník na RS232 standard
- podporován SW OpenOCD, Yagarto, UrJtag
- volně dostupné zapojení
- volně dostupný návrh DPS

Amontec JTAGkey

- podrobnosti na webu: http://www.amontec.com/jtagkey.shtml
- napěťový rozsah připojeného zařízení: 1,4 - 5V
- výstup na 20 pinový konektor ARM (viz Příloha B)
- vyvedeny rozšiřující signály (TRST, SRST)
- podporován SW OpenOCD, UrJtag
- dodáván proprietární SW umožňující interpretovat SVF soubory
- dodávaný SW podporuje přístup k čipům Altera, Atmel, Cypress, Lattice, Xilinx
- SW pro Windows a Linux

OOCDLinks

- podrobnosti na webu: http://www.joernonline.de/contrexx2/cms/index.php?page=126
- napěťový rozsah připojeného zařízení: 3 - 5V
- jde o modul na DPS s rozměry pouze 39 x 32mm
- vznikl jako reakce na projekt OpenOCD
- podporován SW OpenOCD, UrJtag
- volně dostupné zapojení
- volně dostupný návrh DPS
- obsahuje navíc UART rozhraní

Olimex ARM-USB-OCD

- podrobnosti na webu: http://www.olimex.com/dev/arm-usb-ocd.html
- napěťový rozsah připojeného zařízení: 2 - 5V
- výstup na 20 pinový konektor ARM (viz Příloha B)
- obsahuje převodník na RS232 standard
- podporován SW OpenOCD, UrJtag, Yagarto
- proprietální zapojení, nicméně velmi používaný adaptér

Kramara USB JTAG Adapter

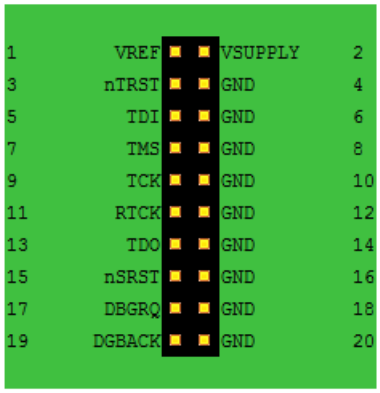
- podrobnosti na webu: http://kramara.com/?q=node/15
- napěťový rozsah připojeného zařízení: 1,65 - 5V
- výstup na 20 pinový konektor ARM (viz Příloha B)
- využívá JTAG mód čipu FT2232, velmi jednoduché zapojení
- volně dostupné zapojení
- podporován SW OpenOCD, IAR, CrossWork, UrJtag

Příloha B

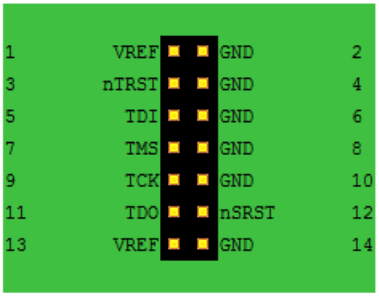
Konektory pro JTAG

Konektor pro JTAG není standardizován, proto každý výrobce používá vlastní. Nicméně alespoň v rámci jednoho výrobce jsou konektory obvykle stejně zapojené. Na obrázku je nákres nejobvyklejších zapojení.
Navržený programátor má konektor zapojen jako ARM JTAG header pinout.

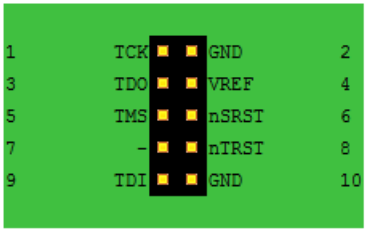
ARM JTAG header pinout



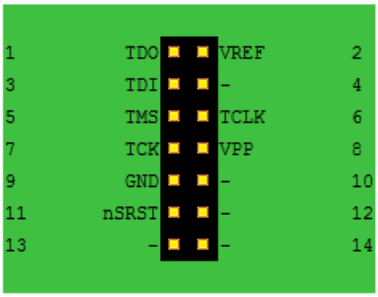
ARM14 JTAG header pinout



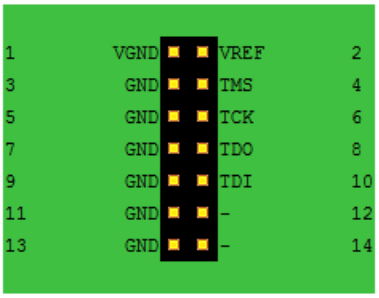
AVR JTAG header pinout



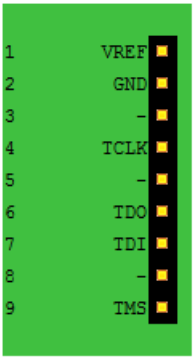
Texas Instruments MSP430



Xilinx Parallel IV 14pin JTAG



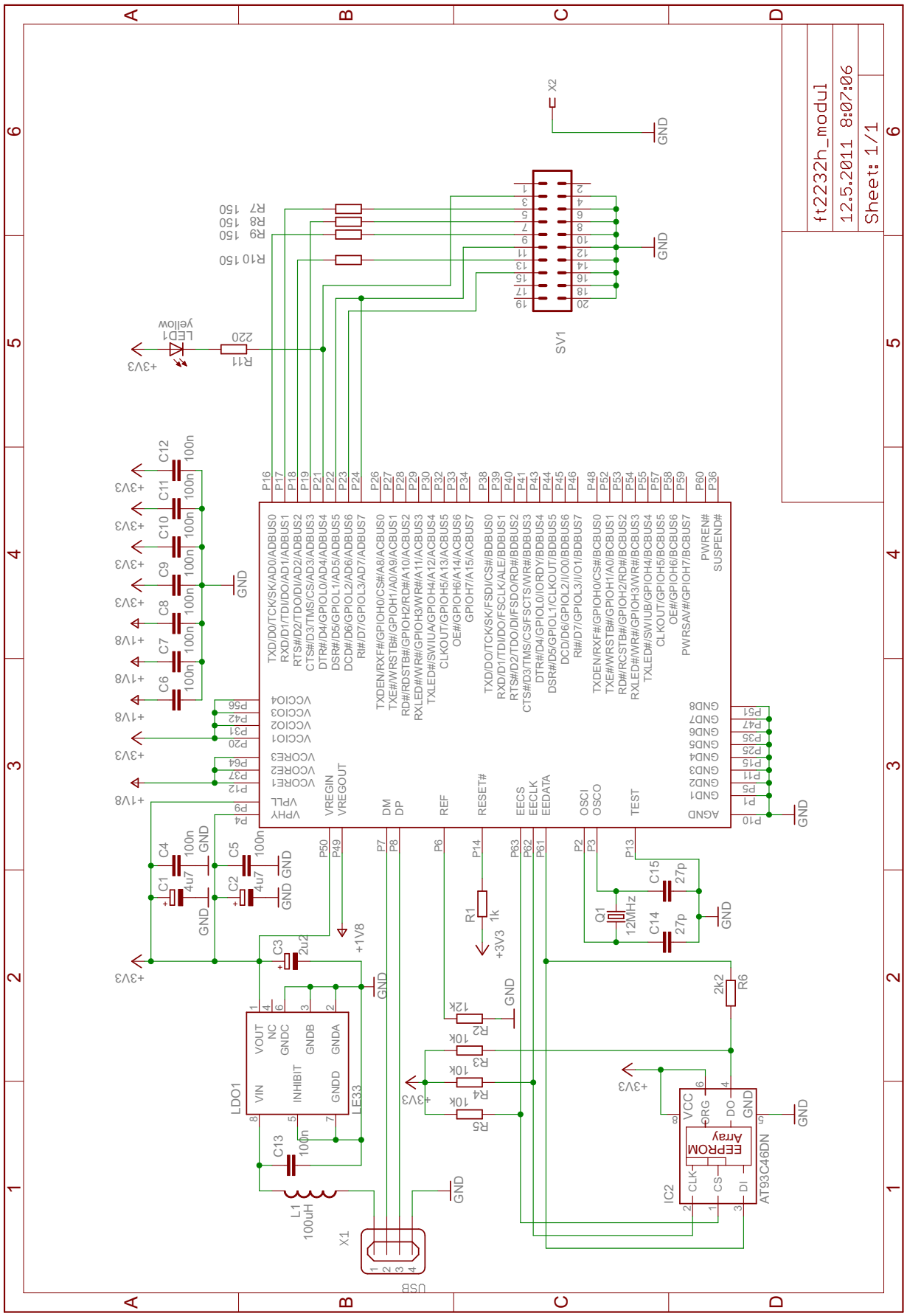
Xilinx Parallel III and IV 9pin JTAG



Obrázek B.1: Konektory JTAG podle výrobců. [9]

Příloha C

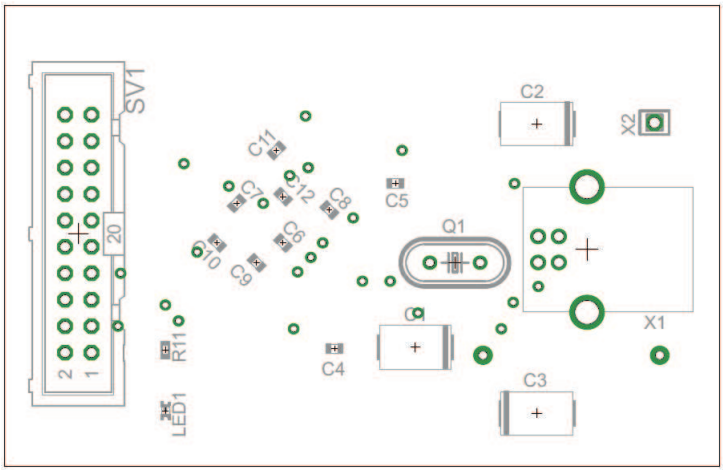
Schéma zapojení



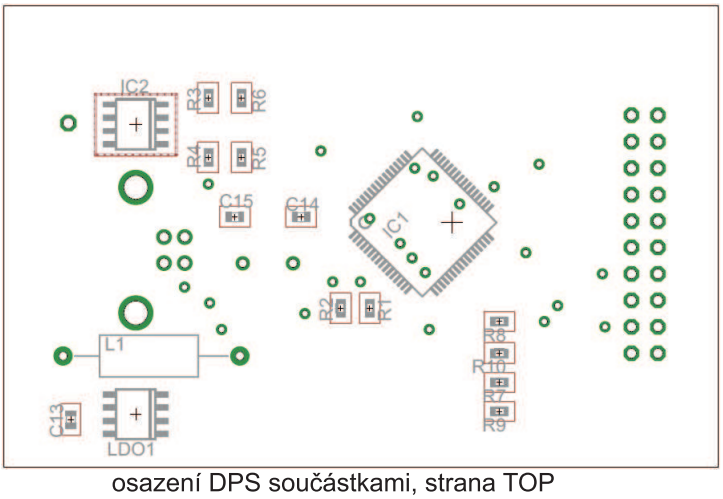
Obrázek C.1: Schéma navrženého programátoru

Příloha D

Osazení DPS a seznam součástek



osazení DPS součástkami, strana BOTTOM



osazení DPS součástkami, strana TOP

Obrázek D.1: Osazovací schéma DPS

Název	Hodnota	Pouzdro	Poznámka
C1	4u7	D/7343-31W	
C2	4u7	D/7343-31W	
C3	2u2	D/7343-31W	
C4	100n	C0603	
C5	100n	C0603	
C6	100n	C0603	
C7	100n	C0603	
C8	100n	C0603	
C9	100n	C0603	
C10	100n	C0603	
C11	100n	C0603	
C12	100n	C0603	
C13	100n	C0603	
C14	27p	C0603	
C15	27p	C0603	
IC1	FT2232H	LQFP64	
IC2	AT93C46DN	SO-08	Serial EEPROM
L1	100uH	TL	tlumivka
LDO1	LE33	SO-08	napěťový regulátor
LED1	yellow	0603	žlutá LED
Q1	12MHz	HC49U-V	krystal
R1	1k	R0603	
R2	12k	R0603	
R3	10k	R0603	
R4	10k	R0603	
R5	10k	R0603	
R6	2k2	R0603	
R7	150	R0603	
R8	150	R0603	
R9	150	R0603	
R10	150	R0603	
R11	220	R0603	
SV1		ML20	konektor 2x10 pinů
X1		MINI-USB-4P	MINI USB 4 pólový konektor
X2			samostatný pin - špička

Tabulka D.1: Seznam součástek JTAG programátoru

Příloha E

Obsah přiloženého CD-ROM

```
|
|---jtagProg.pdf (text práce ve formátu PDF)
|
|---software
|   |---OpenOCD (ARM programátor a debugger)
|   |  |---bin  (program a hlavní konfigurační soubor)
|   |  |---board (konfigurační soubor pro vývojovou desku)
|   |  |---drivers
|   |  |---interface
|   |  |---source (konfigurační soubor pro programovací adaptér)
|   |  |---target (konfigurační soubor pro čip)
|   |
|   |---xsvf (MS Visual Studio projekt knihovny pro parsování SVF)
|   |  |---xsvf (vlastní knivovna xsvf)
|   |  |   |---Release
|   |  |   |---Debug
|   |  |
|   |  |---parser (zkušební program pro testování xsvf knihovny)
|   |      |---Release
|   |      |---Debug
|   |
|   |
|   |---jtagProg (MS Visual Studio projekt výsledného programu)
|       |---Release
|       |---Debug
|
|---hardware
    |---vyvojovy_modul (návrh schématu a DPS v Eagle)
    |---programator (návrh schématu a DPS v Eagle)
```