Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Analyzátor rozhraní JTAG

Plzeň 2013 Lukáš Výškrabka

Zadání

Zde bude vloženo originální zadání.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 27. dubna 2013

Lukáš Výškrabka

Abstrakt

Tato práce poskytuje informace o rozhraní JTAG, jeho vlastnostech a s ním spojené Boundary-Scan architektuře. Popisuje, proč bylo toto rozhraní vytvořeno, jaká vznikla norma a kdo se na jejím vzniku podílel. Jedna z částí se zabývá použitím tohoto rozhraní – kde se využívá a jaké registry a instrukce vyžaduje. Jedním z cílů této práce je návrh a implementace vlastního JTAG analyzátoru. Návrh je simulován v simulátoru ModelSim a následně implementován v programovatelném hradlovém poli (FPGA).

Abstract

This work provides information about JTAG interface, its properties and Boundary-Scan architecture. It describes why the interface was created and who participated in it. One of the parts deals with usage of this interface – where it is used and which registers and instructions are required. This work also shows how it is possible to design and implement own JTAG analyser. Design is simulated by ModelSim simulator and then implemented in field programmable gate array (FPGA).

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

JTAG je standardní sériové rozhraní definované normou *IEEE Std 1149.1*. Bylo navržené zejména pro testování číslicových obvodů, ale používá se i pro jiné účely, jako např. pro programování mikrokontrolerů nebo pro ladění programů vytvořených pro tato zařízení. V dnešní době, v době chytrých mobilních telefonů, je JTAG rozhraní používáno pro servis poškozených (tzv. hard-bricked) mobilních telefonů, běžících na platformě Android. V tomto případě JTAG může sloužit pro nahrání původního *firmware*.

Analyzátor je nástroj pro vývojáře zařízení, která s tímto rozhraním pracují. Při jejich vývoji je třeba i toto rozhraní odladit. Existují aplikace, kde je tohoto rozhraní využíváno např. pro programování mikrokontroléru jiným zařízením (např. také mikrokontrolérem). V tomto případě se těžko obejdeme bez nějakého ladícího nástroje. Tímto nástrojem může být logický analyzátor, a nebo lépe – analyzátor přímo daného rozhraní (v našem případě JTAG). Analyzátorem rozhraní JTAG můžeme monitorovat komunikaci mezi zařízeními a tím snáze odhalit možné nedostatky, popř. chyby.

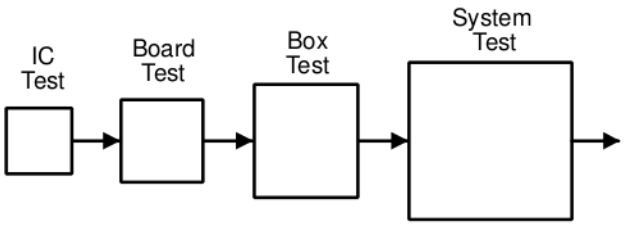
Naším úkolem je prostudovat vlastnosti tohoto rozhraní a navrhnout jeho analyzátor.

2 Úvod do testování

Design for test (DFT), také známo jako *design for testability*, je proces, který zahrnuje pravidla a techniky návrhu lehce testovatelných produktů. DFT slouží k minimalizaci času na vývoj a ke snižování výrobních nákladů. Výrobce, který na trh s elektrotechnikou dodává špatně odladěné výrobky, nemůže být konkurenceschopný. Náklady na nalezení a identifikování chyb použitím tradičních testovacích metod rostou se složitostí výrobku. V důsledku této složitosti se někteří výrobci mohou rozhodnout pro méně důsledné testování jakožto rychlé alternativy k úplnému odladění. Tímto však hazardují s jejich důvěryhodností na trhu. Čas na uvedení výrobku na trh je zde důležitým parametrem. Společnosti, které produkují nové kvalitní výrobky s krátkou dobou vývoje, mají konkurenční výhodu. Návrh testovatelnosti do systému může hrát tedy důležitou úlohu při zavádění nové technologie. [1]

Testování obvodů má dvě hlavní části: řízení a pozorování. Pro testování nějakého systému je nezbytné nastavit tento systém do známého stavu, dodat vstupní data a pozorovat, zda-li se systém chová tak, jak má. Pokud řízení nebo sledování není možno provádět, nelze ověřit správnou funkčnost systému.

Při vývoji produktu dochází k testování na několika úrovních.



Obrázek 2.1: Průběh testování
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

Schopnost znovu použít již dříve vyvinutá testovací data a použití levnějších testovacích prostředků znamená větší výnosy z méně nákladného produktu. Boundary-Scan testování v kombinaci s běžnou testovací sběrnicí a testovacím protokolem přináší tyto výhody:

1. standardní a nákladově efektivní řešení tradičních testovacích problémů
2. nové aplikace

3 Průmyslový standard – IEEE Std 1149.1-1990 (JTAG)

V roce 1985 vznikla skupina složená z klíčových elektronických výrobců, která se připojila k *Joint Test Action Group (JTAG)* a která usilovala o dosažení dohody, jenž by stanovila princip konstrukce integrovaných obvodů. JTAG měl přes 200 členů z celého světa, včetně těch největších výrobců polovodičů a elektroniky. Tato skupina se sešla, aby vytvořila řešení k testování obvodů a aby toto řešení bylo podporováno jako průmyslový standard. Bylo označeno jako IEEE Std 1149.1-1990, IEEE Standard Test Acces Port and Boundary-Scan Architecture. [1]

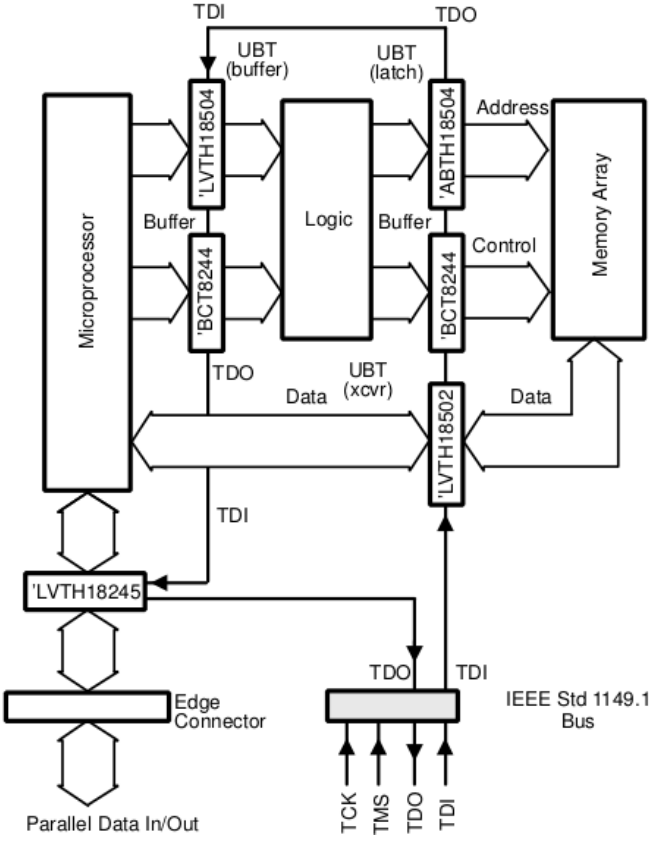
IEEE Std 1149.1 umožňuje sériově načíst testovací instrukce a data do zařízení a následné přečtení výsledků vykonaného testu. Cílem bylo, aby metodika testování byla použitelná i v případě, že se na jedné desce vyskytovaly komponenty od různých výrobců. Každé IEEE Std 1149.1 kompatibilní zařízení má 4 piny: dva na řízení, jeden na vstupní a jeden na výstupní data. Komunikace, tj. přenos instrukcí, testovacích dat a výsledků testu, probíhá sériově. Aby zařízení bylo kompatibilní s tímto standardem, musí podporovat určité základní funkce. Výhodou ale je, že IEEE Std 1149.1 umožňuje návrhářům přidat si vlastní funkce, které budou odpovídat jejich vlastním speciálním požadavkům. [1]

Obecně bývá prvním krokem při testování obvodu sériové nahrání instrukce pro operaci, která má být provedena. Testovací logika je navržena tak, že sériový posuv instrukce neovlivní ty bloky obvodu, jejichž činnost je ovládána touto instrukcí. Vliv instrukce se projeví až po dokončení posuvu (po jejím úplném načtením do obvodu). Jakmile je jednou instrukce načtena, vybraný obvod je nakonfigurovaný k odezvě. Nicméně v některých případech je potřeba do vybraného obvodu načíst příslušná data, aby výsledek testu byl smysluplný. Data jsou do obvodu načítána stejně jako instrukce. Po provedení instrukce jsou pak data sériově vysouvána ven z obvodu. [1]

4 Boundary-Scan architektura a IEEE Std 1149.1

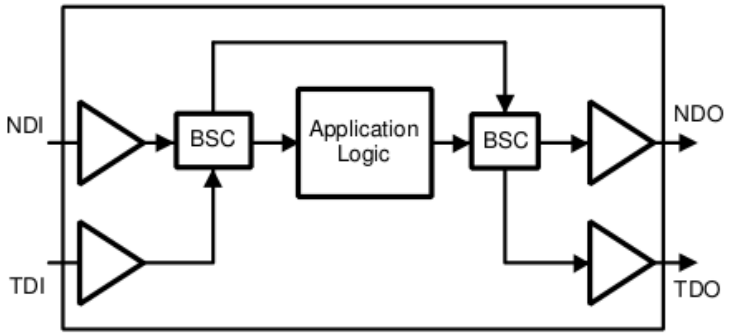
4.1 Boundary-Scan architektura

Testovací logika integrovaného obvodu se skládá z tzv. řadiče TAP (Test Acces Port), Boundary-Scan buněk a několika funkčních registrů. Boundary-Scan je rozhraní mezi integrovaným obvodem a plošným spojem. Je to metodika, která umožňuje dodržet zásady řízení testované jednotky (controllability) a její pozorovatelnost (observability) užitím programového řízení. Základní myšlenka spočívá ve vložení jednoho článku posuvného registru (klopného obvodu) mezi funkční bloky integrovaného obvodu a jeho vývody. [1]



Obrázek 4.1: Příklad Boundary-Scan testování.
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

Na každém vývodu bude tedy umístěn tento základní stavební prvek, který se nazývá Boundary-Scany buňka, neboli Boundary-Scan cell (BSC). Jednoduchý příklad, jak může vypadat Boundary-Scan testování, je vidět na obrázku 4.2.



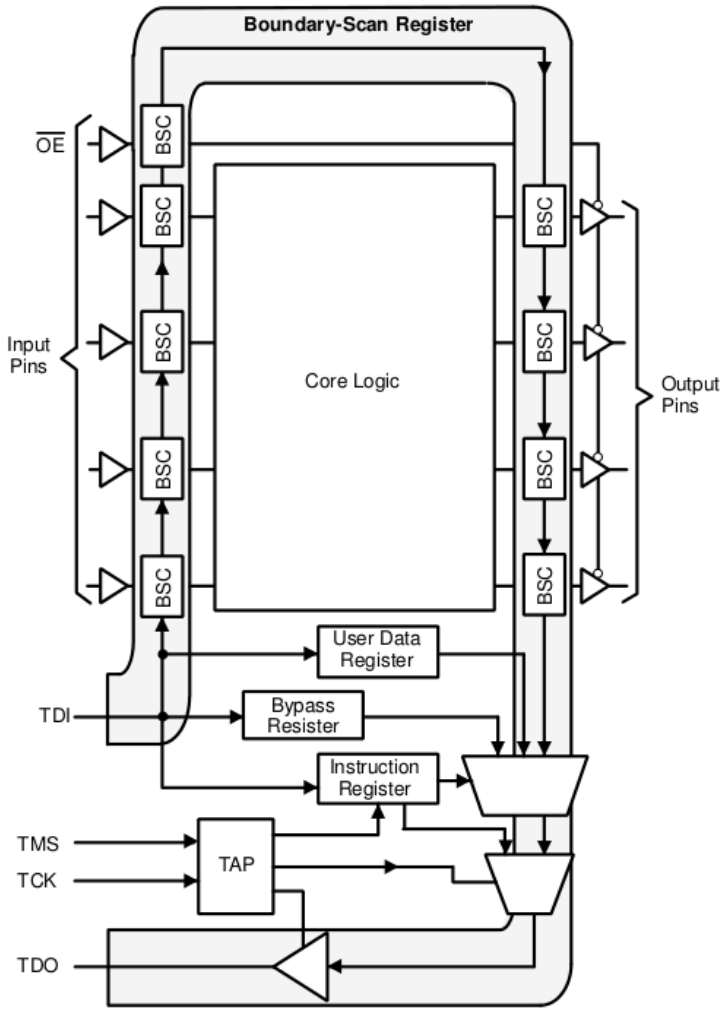
Obrázek 4.2: Boundary-Scan příklad.
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

Při normální práci integrovaného obvodu signály volně projdou každou BSC – a to z normálního datového vstupu (NDI), přes testovaný obvod (Application Logic), na normální datový výstup (NDO). Nicméně, pokud je obvod v testovacím režimu, můžeme na každou BSC nasunout testovací data signálem TDI a signálem TDO výsledky testu z obvodu vysouvat.

4.2 Testovací rozhraní

IEEE Std 1149.1 architektura je vidět na obrázku 4.3. Integrovaný obvod s Boundary-Scan rozhraním obsahuje řadu vnitřních registrů. A to Instrukční registr (Instruction register), Bypass registr, Boundary-Scan registr a nepovinný uživatelský datový registr (user data register).

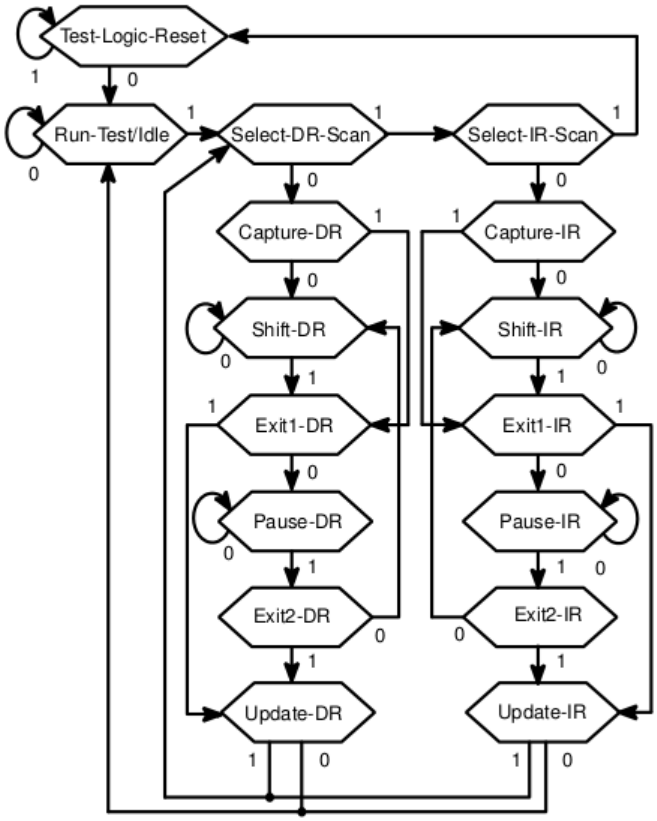
Celé testovací rozhraní je řízeno řadičem TAP (test acces port) – jedná se o konečný automat, který pracuje podle daného přechodového diagramu. Boundary-Scan registr je tvořen z jednotlivých Boundary-Scan buněk. Instrukční a datové registry jsou od sebe navzájem odděleny. Architektura je navržena tak, aby všechny byly přístupny pomocí signálů TDI a TDO. To, který registr bude použit, je řízeno řadičem TAP.



Obrázek 4.3: Boundary-Scan architektura.
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

4.3 Test Access Port (TAP)

Řadič TAP je řízen dvěma signály, a to: test clock (TCK) a test mode select (TMS). Tyto signály určují, zda-li se bude pracovat s datovým nebo instrukčním registrem. Jedná se o konečný automat pracující podle stavového diagramu na obrázku 4.4. IEEE Std 1149.1 používá obě hrany signálu TCK. Signály TMS a TDI jsou vzorkovány při náběžné hraně hodinového signálu, zatímco signál TDO při sestupné hraně.



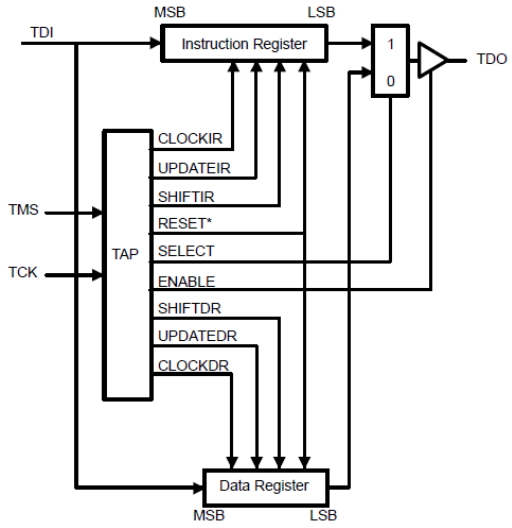
Obrázek 4.4: Stavový diagram řadiče TAP.
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

Stavový diagram se skládá ze šesti stabilních stavů: Test-Logic-Reset, Run-Test/Idle, Shift-DR, Pause-DE, Shift-IR a Pause-IR. Tento protokol je navržen tak, že pokud je signál TMS roven logické 1, existuje pouze jeden stabilní stav a to stav Test-Logic-Reset. To znamená, že vyresetování testovací logiky může být dosaženo aplikací log. 1 na signál TMS po dobu minimálně pěti náběžných hran hodinového signálu.

Při spuštění nebo normální činnosti integrovaného obvodu je řadič TAP uveden do stavu Test-Logic-Reset. V tomto stavu testovací logika nebrání normálnímu fungování integrovaného obvodu. Při příchodu log. 0 na signál TMS se řadič dostává do následujícího stavu – Run-Test/Idle. Funkce jednotlivých stavů budou dále postupně popisovány.

4.4 Registry IEEE Std 1149.1

Stavový diagram řadiče TAP je symetrický. To, zda-li bude použit instrukční registr nebo jeden z datových registrů, je dáno aktuálním stavem řadiče.



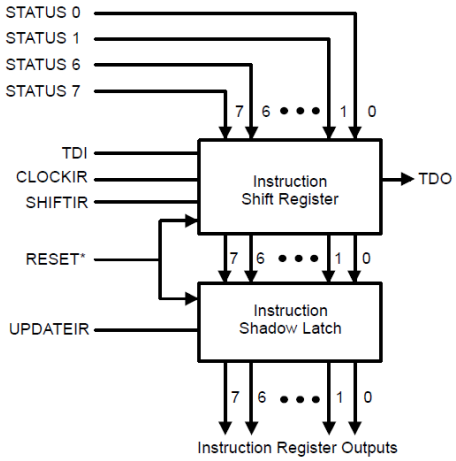
Obrázek 4.5: Řízení registrů řadičem TAP.
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

4.4.1 Instrukční registr (povinný)

Instrukční registr je zodpovědný za výběr datového registru, který má být použit při načítání dat a je k němu přistupováno právě tehdy, pokud se TAP nachází v jednom ze stavů pracujících s instrukčním registrem (pravá větev stavového diagramu). Na základě načtené instrukce se zvolí datový registr, který bude použit při následném načítání dat. Při načítání instrukce do registru signál SELECT z řadiče TAP přivede výstup z instruknčího registru na vývod TDO. [1]

Instrukční registr se skládá z posuvného a z vyrovnávacího registru. Posuvný registr je umístěn mezi piny TDI a TDO laděného integrovaného obvodu a slouží pro sériové nahrání instrukce. Průběh načítání instrukce je řízen řadičem TAP. Instrukce je nasouvána do posuvného registru ve stavu `SHIFTIR` při náběžné hraně signálu TCK[1] (a vysouvána při doběžné hraně TCK). Vstupy STATUS 0 - 7 (viz obr. 4.6) jsou uživatelsky definované –

[1]Signál TMS je roven log. 0 pro setrvání v aktuálním stavu.



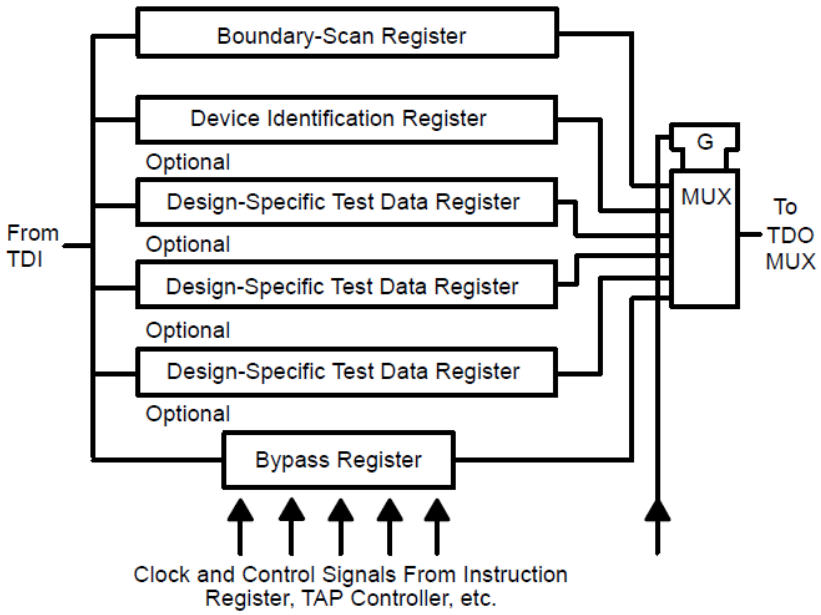
Obrázek 4.6: Architektura instrukčního registru.
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

kromě dvou nejméně významných bitů, které jsou vždy 01. Minimální velikost instrukčního registru je 2. [1]

Vyrovnávací registr obsahuje skupinu klopných obvodů – jeden klopný obvod pro jeden bit instrukčního posuvného registru. Při načítání instrukce do posuvného registru zůstává vyrovnávací registr v původním stavu. Na konci načítání dochází k uložení nových dat z posuvného registru do vyrovnávacího registru (stav `UPDATEIR`). Pokud je aktivován vstup RESET, vyrovnávací registr je nastaven na hodnotu BYPASS instrukce (nebo IDCODE instrukce, pokud je podporována). To zařídí normální funkčnost zařízení a výchozí výběr bypass registru. [1]

4.4.2 Datové registry

IEEE Std 1149.1 vyžaduje dva datové registry – a to Boundary-Scan registr a Bypass registr. Třetí registr je volitelný a slouží pro identifikaci zařízení. Architektura může obsahovat i další uživatelem definované datové registry. Tyto registry jsou paralelně umístěny mezi vstupy TDI a TDO a to, který registr bude použit je dáno aktuálně načtenou instrukcí. Posun dat je řízen řadičem TAP a probíhá ve stavu `SHIFTDR`. V průběhu nasouvání dat do datového registru výstup SELECT z řadiče TAP přivede výstup z vybraného datového registru na výstup TDO. Pokud je jeden registr vybrán, ostatní registry se nemění a zůstávají ve stejném stavu. [1]



Obrázek 4.7: Architektura datových registrů.
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

Boundary-Scan registr

Boundary-Scan registr (BSR) se skládá ze skupiny Boundary-Scan buněk (BSC), které jsou umístěny na V/V pinech. Tvoří tak rozhraní mezi laděným obvodem a plošným spojem. Násun dat probíhá do posuvného registru a až po jeho dokončení se provede zápis do vyrovnávacího registru (stav `UPDATEDR`). Více informací o Boundary-Scan architektuře lze nalézt v kapitole 4.

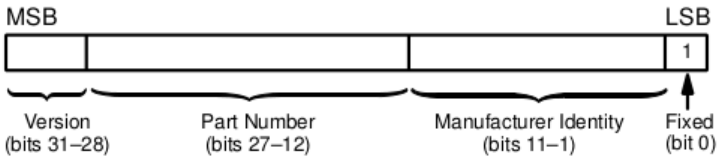
Bypass registr (povinný)

Bypass registr obsahuje jeden bit – jeden klopný obvod. Pokud je registr vybrán, datová cesta mezi TDI a TDO obsahuje pouze jeden bit (umožňuje zkrácení datové cesty o délku registrů těch zařízení, které nejsou potřeba k aktuálně prováděnému testu). Registr je vybrán právě tehdy, když je v instrukčním registru načtena bypass instrukce. [1]

Registr pro identifikaci zařízení (volitelný)

Tento registr bývá v anglické literatuře označován jako *Device Identification Register*. Je to volitelný registr sloužící k identifikaci výrobce zařízení. Obsa-

huje specifické informace o daném zařízení. Na obrázku 4.8 je vidět struktura tohoto registru. Jeho jednotlivé bity mohou být vysouvány ven z registru po jeho vybrání – tj. instrukcí idcode. [1]



Obrázek 4.8: Struktura registru pro identifikaci zařízení.
Zdroj: IEEE Standard Test Access Port and Boundary-Scan Architecture

Identifikační kódy výrobců jsou zapsány, uchovávány a upravovány společností EIA/JEDEC. Jakákoliv firma zde může být na žádost registrována.

4.4.3 Souhrn registrů

Registr	**Délka**	**Hodnota**	**Instrukce**
Instrukční	minimálně 2 bity	X..01	-
Boundary-Scan	podle obvodu	podle obvodu	Extest Sample/Preload Intest Clamp HighZ
Bypass	1 bit	0	Bypass
Device ID	32 bitů	X..1	Idcode Usercode
Uživatelský	podle obvodu	podle obvodu	podle obvodu

Tabulka 4.1: Souhrn registrů

4.5 Instrukce IEEE Std 1149.1

IEEE Std 1149.1 definuje devět testovacích instrukcí. Z toho tři jsou povinné a šest je nepovinných. Níže si tyto instrukce stručně popíšeme.

4.5.1 Povinné

Bypass instrukce

Tato instrukce využívá pouze BYPASS registr, který se připojí do datové cesty mezi vývody TDI a TDO. Umožňuje přesun dat mezi TDI a TDO aniž by se změnila funkčnost daného obvodu. Může být použita například, když se na jedné desce nachází více komponent a my potřebujeme testovat jen některé z nich. Do komponent, které nepotřebujeme testovat, se nahraje instrukce Bypass a data tak budou rovnou procházet do dalšího obvodu. Bez použití této instrukce by data musela projít všemi obvody a tím by se doba testování zbytečně prodloužila. Při resetu bývá nastavována instrukce Bypass (popř. Idcode, je-li definována) jako výchozí. Bitový kód instrukce je definovaný normou na samé jedničky. [1]

Sample/preload instrukce

Umožňuje vybrat a připojit BOUNDARY-SCAN registr mezi vývody TDI a TDO. Před začátkem čtení z registru (ve stavu `Capture-DR`) jsou hodnoty všech signálů na vstupních/výstupních pinech sejmuty a uloženy do Boundary-Scan registru. Tato instrukce tedy slouží k přečtení hodnot na výstupních pinech nebo k přednastavení (preload) testovacích dat do Boundary-Scan registru před použitím instrukce Extest. V průběhu této instrukce se nemění funkčnost komponenty. Bitový kód této instrukce je definovaný výrobcem. [1]

Extest instrukce

Instrukce přepne obvod do externího Boundary-Test režimu – systémová logika je vyřazena z činnosti. Využívá a připojuje mezi vývody TDI a TDO registr BOUNDARY-SCAN. Používá se k testování propojení mezi komponentami. Bity Boundary-Scan registru na výstupních pinech komponenty jsou použity jako budiče testovacích hodnot, zatímco bity Boundary-Scan registru na vstupních pinech jsou použity k zaznamenávání výsledků. Hodnoty z V/V pinů jsou čteny ve stavu `Capture-DR` a zapisovány na V/V piny ve stavu `Update-DR`. Bitový kód instrukce je definovaný normou na samé nuly. [1]

4.5.2 Volitelné

Intest instrukce

Instrukce Intest přepne obvod do interního Boundary-Test režimu. Stejně jako instrukce Extest využívá BOUNDARY-SCAN registr. Slouží k provedení testů systémové logiky: stav všech vstupních pinů komponenty je určen daty[2] nasunutými v Boundary-Scan registru a následně ve stavu `Capture-DR` jsou pak do Boundary-Scan registru uloženy i hodnoty na výstupních pinech systémové logiky. Bitový kód této instrukce je definovaný výrobcem. [1]

Runbist instrukce

Instrukce Runbist[3] uvede obvod do režimu vnitřního testu, mezi vývody TDI a TDO připojí nějaký z uživatelských registrů a spustí vestavěný test. Test je prováděn ve stavu `Run-Test/Idle` řadiče TAP. V průběhu této instrukce jsou vstupy a výstupy komponenty řízeny testem. Po vykonání testu jsou výsledky uloženy v nějakém uživatelském registru. Bitový kód této instrukce je definovaný výrobcem. [1]

Clamp instrukce

Instrukce Clamp nastaví výstupy komponenty na hodnotu obsahující Boundary-Scan registr a připojí jednobitový BYPASS registr mezi vývody TDI a TDO. Před načtením této instrukce může být přednastaven obsah Boundary-Scan registru instrukcí Sample/preload. V průběhu instrukce mohou být data nasouvána přes bypass registr z pinu TDI na TDO, aniž by ovlivnila funkčnost komponenty. Bitový kód této instrukce je definovaný výrobcem. [1]

Highz Instrukce

Tato instrukce nastaví všechny výstupy do stavu vysoké impedance. Bitový kód této instrukce je definovaný výrobcem. [1]

Idcode instrukce

Instrukce Idcode připojí registr pro IDENTIFIKACI ZAŘÍZENÍ mezi vývody TDI a TDO. Registr pro identifikaci zařízení je 32 bitový registr obsahující různé informace o zařízení a výrobci (viz obrázek 4.8). Registr může být také

[2]Testovací data mohou být do Boundary-Scan registru načtena výše popsanou instrukcí Sample/preload.

[3]RUNBIST = RUN Build-In Self Test

přístupný ihned po zapnutí zařízení, po resetu zařízení nebo také ve stavu `Run-Test/Idle`. Bitový kód této instrukce je definovaný výrobcem. [1]

4.5.3 Souhrn instrukcí

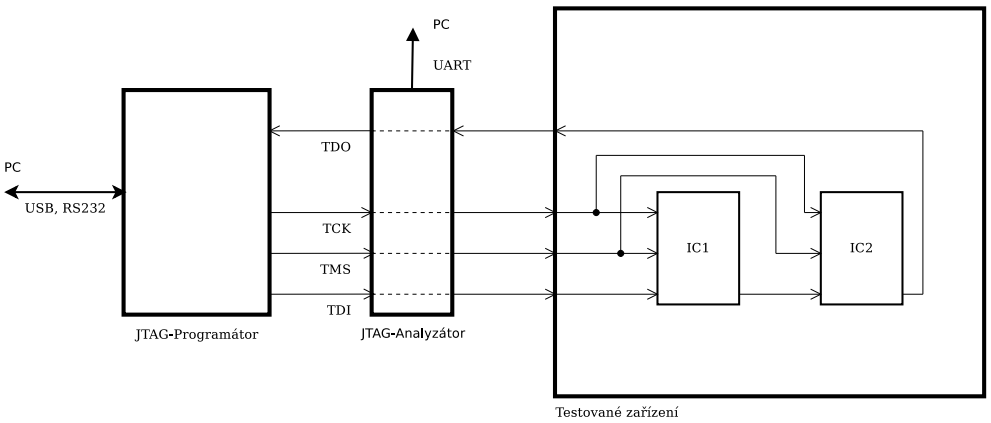
Instrukce	**Kód**	**Režim**	**Používaný registr**
Extest	0..0	Test	Boundary-Scan
Sample/preload	podle obvodu	Normalní	Boundary-Scan
Bypass	1..1	Normální	Bypass
Intest	podle obvodu	Test	Boundary-Scan
Runbist	podle obvodu	Test	podle obvodu
Idcode	podle obvodu	Normální	Device ID
Usercode	podle obvodu	Normální	Device ID
Clamp	podle obvodu	Test	Bypass
HighZ	podle obvodu	Test	Bypass
uživatelská	podle obvodu	podle obvodu	podle obvodu

Tabulka 4.2: Souhrn instrukcí

Poznámka: *Do resp. z registrů je zapisováno resp. čteno sériově. Vstupem TDI jsou data nasouvána do registru a vstupem TDO jsou vysouvána ven. Pro některé instrukce je využíván také paralelní vstup a výstup registrů. V tomto případě je do registru paralelně zapsáno ve stavu Capture-DR resp. Capture-IR.*

5 Analýza problému

5.1 Požadavky na vlastnosti analyzátoru



Obrázek 5.1: Naše situace

Na obrázku č. 5.1 je vidět velmi zjednodušeně naše situace: JTAG programátor, analyzátor a testované (popř. programované) zařízení. JTAG programátor komunikuje s PC např. přes rozhraní USB nebo RS232 a pomocí signálů TDI, TMS a TCK ovládá testované zařízení. Do testovaného zařízení jsou nasouvány data pomocí signálu TDI, ty procházejí všemi integrovanými obvody, které testované zařízení obsahuje, a poté jsou vysouvány signálem TDO zpět do JTAG programátoru. Naším úkolem je tedy vytvořit blok *Analyzátor*, který bude zachycovat komunikaci mezi JTAG programátorem a testovaným zařízením. Analyzátor bude sloužit k monitorování komunikace – nikoliv k simulaci tohoto rozhraní. Zachycená data je pak nutno někde zobrazit. V prvních verzích analyzátoru je budeme posílat po seriovém rozhraní do PC. Požadavky na vlastnosti analyzátoru:

1. Možnost zaznamenat zapsaná data – analyzátor by měl zaznamenat kam a jaká data byla zapsána.

2. Funkčnost v reálném čase – přijatý signál by měl dekódovat a s minimálním zpožděním přivést na výstup analyzátoru (tzn. na vstup testovaného zařízení).

3. Velká vnitřní paměť – komunikace probíhá ve velké rychlosti, tudíž vnitřní paměť (buffer) musí mít dostatečnou velikost, aby byla schopna

pojmout veškerá data (do té doby, dokud nebudou odeslána přes sériový kanál do PC).

4. Možnost zaznamenat časovou značku – k přijatým datům musí být schopný přidat i časovou značku.

5. Možnost nastavovat vlastnosti rozhraní JTAG pro více různých zařízení.

6. Tvarovací obvody – na V/V pinech analyzátoru by měly být umístěny tvarovací obvody pro eliminaci rušivého signálu (např. Schmittův klopný obvod).

7. Příjmač dat ze sériového portu pro PC – k analyzátoru by měl být vytvořen příjmač dat ze sériového portu (např. v jazyce C/C++) pro přehledné monitorování komunikace.

5.2 Volba vhodné architektury

Další úvaha musí směřovat k volbě vhodné architektury. Budu se rozhodovat mezi dvěmi hlavními možnostmi – a to mezi programovatelným hradlovým polem a mikroprocesorem. Programování pomocí JTAG rozhraní probíhá na vysokých frekvencích (může být až 100 MHz – maximální frekvence však není normou stanovena), které by nemusely být rozpoznány na vstupech mikroprocesoru. Z těchto důvodů by bylo lepší zvolit použití hradlového pole, které je velmi rychlé a s těmito frekvencemi nebude mít problém. Na druhou stranu použitím mikroprocesoru by v budoucnu bylo snažší např. zobrazování zachycené komunikace na LCD displeji nebo ovládání analyzátoru vnějšími vstupy. To lze ale vyřešit nahráním nějakého *soft procesoru (např. MicroBlaze)* do hradlového pole. Tento procesor lze programovat v programovacím jazyce C/C++, tudíž v budoucnu by nebyl problém implementovat nějaká další rozšíření, pro které by byl jazyk pro popis HW (např. VHDL[1]) nedostačující.

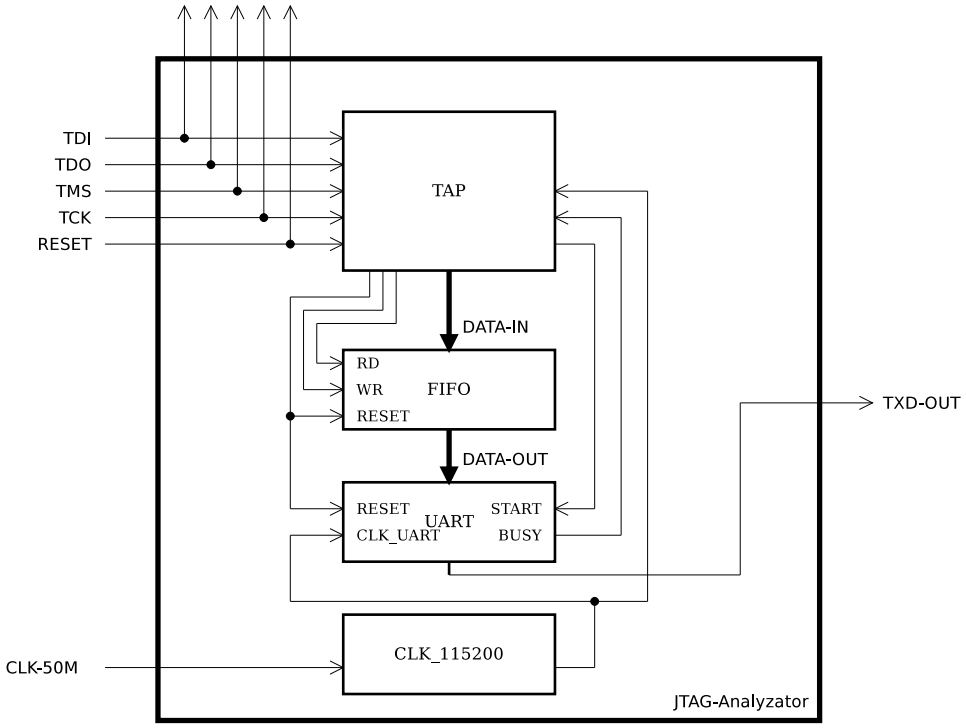
Z výše uvedených důvodů tedy použiji programovatelné hradlové pole (FPGA[2]) od firmy XILINX, které je dostupné ve školních laboratořích a pro které existuje již výše zmíněný soft procesor MicroBlaze.

[1] Very-high-speed-integrated-circuits Hardware Description Language
[2] Field-Programmable Gate Array

6 Popis implementace

6.1 Implementace analyzátoru v FPGA

V předchozí kapitole jsme na základě uvedených argumentů zvolili, že analyzátor implementujeme v programovatelném hradlovém poli. Vzhledem k dostupnosti hardwaru ve školních laboratořích budu analyzátor navrhovat pro desku XST-3S1000 od společnosti Xess, která obsahuje obvod FPGA nesoucí označení Spartan-3 XC3S1000. Pro testování našeho analyzátoru se z počátku omezíme na procesor MSP430 od firmy Texas Instruments využívající JTAG pro programování flash paměti. Tento JTAG má specifické vlastnosti, které se částečně neshodují s normou IEEE Std 1149.1 popsanou v kapitole č. 4. Při návrhu, bude ale kladen důraz na možnost modifikace analyzátoru i pro jiná zařízení využívající JTAG. Toho může být docíleno používáním generických parametrů a konstant v jazyce VHDL.



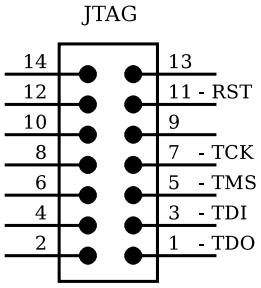
Obrázek 6.1: Komponenty analyzátoru

Obrázek 6.1 Na obrázku jsou vidět všechny komponenty analyzátoru a základní používané signály. Do analyzátoru nám vstupují signály z JTAG

konektoru, a to: *TDI*, *TDO*, *TMS*, *TCK* a nepovinný signál *RESET*. Vstup pojmenovaný jako *clk-50M* je hodinový signál o frekvenci 50MHz získaný z desky XSA-3S1000. Z tohoto signálu později získáme pomocí děličky frekvence hodinový signál pro řízení sériového kanálu. Jediným výstupem je signál *txd-out*, vyvedený do sériového portu. Základem celého zařízení je *řadič TAP*, který je obsažen ve všech zařízeních podporující JTAG. Tento řadič bude ukládat data ze vstupu *TDI* a také bude řídit všechny ostatní komponenty, tj. *FIFO* a *UART*. Pomocí signálu *WR* budou přes vektor *DIN* zapisovány do bufferu (komponenta FIFO) jednotlivé události (např. zápis hodnoty 0x41 do IR) a pomocí signálu *RD* z něj bude přes vektor *DOUT* čteno. Jednotlivé položky z bufferu pak budou přiváděny na vstup komponenty *UART* a odvysílány sériovým kanálem (tj. výstupem txd-out) do PC, kde bude spuštěn náš příjmač, který bude přijatá data ve vhodném tvaru vypisovat na obrazovku. Na obrázku je také vidět, že z analyzátoru vystupují i signály *TDI*, *TDO*, *TMS*, *TCK* a *RESET*. To je pro případ, že nebudeme mít k dispozici kabel s odbočkou. V tom případě můžeme zařadit analyzátor mezi JTAG programátor a programované zařízení jako je tomu na obrázku 5.1.

6.1.1 Vlastnosti rozhraní JTAG procesoru MSP430

Konektor není součástí normy IEEE Std 1149.1, a proto na obrázku 6.2 uvádím ten, který je použit u desky s procesorem MSP430. Na obrázku jsou popsány pouze JTAG signály.



Obrázek 6.2: Námi používaný JTAG konektor

Jak jsem již zmínil, JTAG MSP430 má některé specifické vlastnosti. Obsahuje 8 bitový instrukční registr. Při načítání instrukce do IR je z TDO vysouvána verze JTAG rozhraní implementovaného v cílovém zařízení. Dále obsahuje klasický bypass registr a tři 16 bitové datové registry: *JTAG Memory address bus register (dále jen MAB)*, *JTAG Memory data bus register (dále jen MDB)* a *JTAG Control signal register (dále jen CS)*.

Tyto datové registry mohou být vybrány následujícími instrukcemi, které jsou používány pro programování flash paměti:

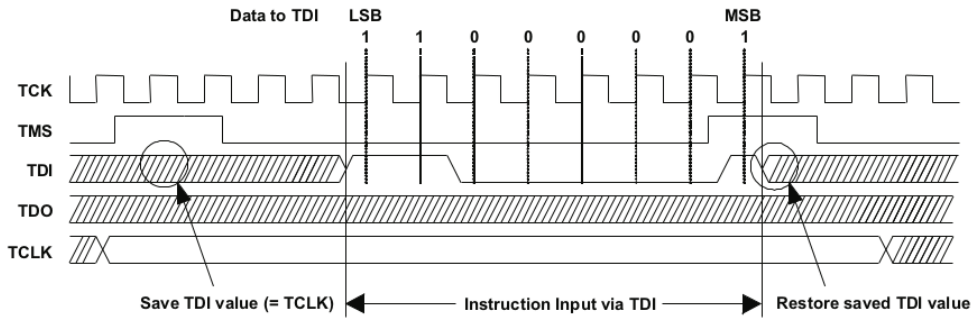
Název instrukce	**8 bitová hodnota**
Řízení MAB	
IR_ADDR_16BIT	0x83
IR_ADDR_CAPTURE	0x84
Řízení MDB	
IR_DATA_TO_ADDR	0x85
IR_DATA_16BIT	0x41
IR_DATA_QUICK	0x43
IR_BYPASS	0xFF
Řízení CPU	
IR_CNTRL_SIG_16BIT	0x13
IR_CNTRL_SIG_CAPTURE	0x14
IR_CNTRL_SIG_RELEASE	0x15
Ověření paměti pomocí PSA (Pseudo Signature Analysis)	
IR_DATA_PSA	0x44
IR_SHIFT_OUT_PSA	0x46
JTAG Acces Security Fuse Programming	
IR_Prepare_Blow	0x22
IR_Ex_Blow	0x24
JTAG Mailbox System	
IR_JMB_EXCHANGE	0x61

Tabulka 6.1: Instrukce pro přístup k paměti používané u MSP430.
Zdroj: MSP430 Memory Programming:User's Guide.

K čemu slouží jednotlivé instrukce a registry není předmětem této práce. Tyto informace lze získat z datasheetu k MSP430 – *MSP430 Memory Programming User's Guide* – který lze nalézt na stránkách[1] firmy Texas Instruments. [2]

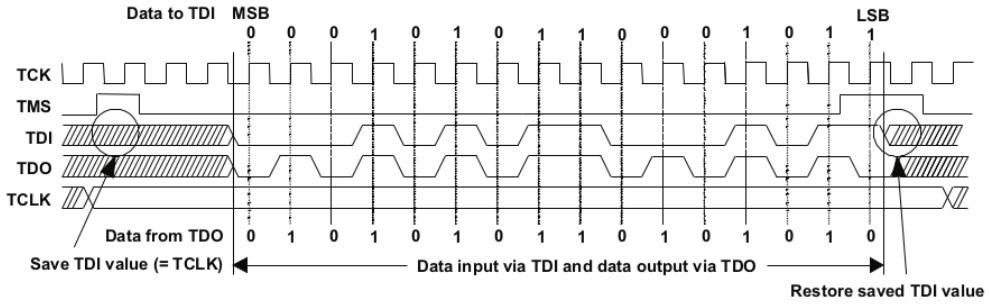
Zápis do IR Na obrázku 6.3 je vidět průběh načítání instrukce do instrukčního registru. Jednotlivé bity instrukce jsou zachytávány ze signálu TDI při náběžné hraně TCK. Přenos začíná vždy *nejméně významným bitem – LSB*. [2]

[1] http://www.ti.com/



Obrázek 6.3: Ukázka načítání instrukce IR_ADDR_16BIT (0x83) do IR.
Zdroj: MSP430 Memory Programming:User's Guide.

Zápis do DR Data do některého z datových registrů jsou nahrávána obdobně. Každý bit je zachycován ze vstupu TDI při náběžné hraně TCK. Ve stejný čas jsou z výstupu TDO vysouvána poslední nahraná a uložená data. Další bit je na signál TDO připraven při sestupné hraně hodinového signálu TCK. Přenos začíná vždy *nejvíce významným bitem – MSB*. [2]



Obrázek 6.4: Ukázka načítání dat do DR (TDI = 0x158B, TDO = 0x55AA).
Zdroj: MSP430 Memory Programming:User's Guide.

6.1.2 Popis jednotlivých komponent

Dále si popíšeme jednotlivé komponenty, jejich funkce, a jak je lze popsat v jazyce VHDL. Tyto komponenty budou reprezentovány entitami. Pro popis funkce entity může být použit *strukturální* nebo *behaviorální* popis. Pokud entita bude obsahovat nějaký parametr, který závisí na cílovém zařízení, bude tento parametr definován jako generický (generic) nebo jako konstanta (constant). U některých komponent bude uveden snímek z programu ModelSim, který byl použit pro simulaci a odladění jednotlivých entit.

Komponenta CLK_115200

```
entity clk_115200 is
port(
 clk_50M: in std_logic;
 reset: in std_logic;
 clk_115200: out std_logic
);
end entity clk_115200;
```

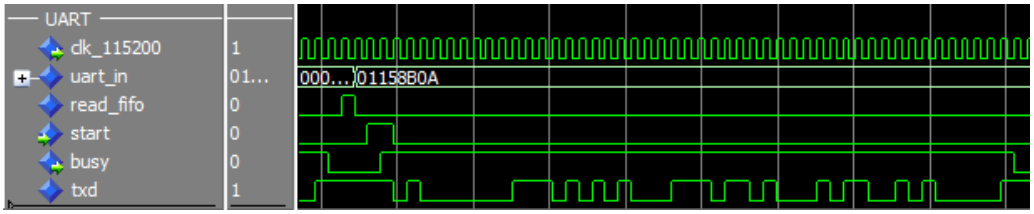
Tato komponenta slouží pro odvození hodinového signálu pro sériový kanál. Jedná se o jednoduchou děličku frekvence za použití jednoho čítače. Do komponenty je na vstup `clk_50M` přiveden hodinový signál o frekvenci 50MHz získaný z oscilátoru, který je na naší desce přiveden na pin `P8`. Na výstupu `clk_115200` pak máme hodinový signál o frekvenci 115200Hz, který bude sloužit pro řízení sériového portu komponentou *TAP*. Tento výstup bude také přiveden na vstup komponenty *UART*, kde bude sloužit k modulaci signálu vysílaného ze sériového portu. Vstup reset slouží k nastavení komponenty do počátečního stavu.

Komponenta UART

```
entity uart is
generic(
 pocet_bitu: positive := 32;
);
port(
 clk_tx: in std_logic;
 reset: in std_logic;
 data: in std_logic_vector((pocet_bitu - 1) downto 0);
 start: in std_logic;
 txd: out std_logic;
 busy: out std_logic
);
end entity uart;
```

Komponenta zajišťující odvysílání jedné položky z komp. *FIFO*. Jedná se o jednoduchý konečný automat, který pomocí výstupu `txd` odvysílá jeden byte, a to ve tvaru: start bit, 8 datových bitů, 1 paritní bit a 1 stop bit. Toto se opakuje, dokud není odvysílaná celá položka (např. 32 bitů). Na vstup `clk_tx` je přiveden hodinový signál generovaný komponentou `clk_115200`. Vysílání

dat ze vstupního vektoru `data` je započato náběžnou hranou na vstupu `start`. Pokud je vysílání aktivní, výstup `busy` se nachází ve stavu log. 1. Vysílaný signál je vyveden na výstup `txd` a tento výstup je přiveden do sériového portu. Generický parametr `pocet_bitu` udává šířků jedné položky nacházející se ve frontě (tzn. šířku vstupu `data`; šířka výstupu `dout` z komp. `FIFO` je rovna `šířka_vstupu_data - 8`; oddělovací znak není ve frontě uložen).



Obrázek 6.5: Ukázka odvysílání jedné položky z fronty včetně jejího načtení

Komponenta FIFO

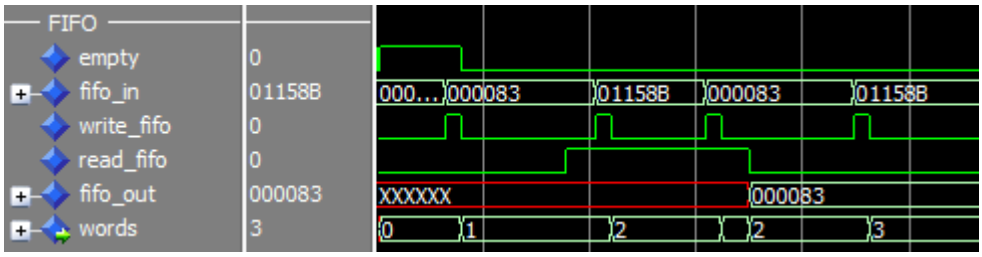
```
entity fifo is
generic(
 sirka: positive := 8;
 polozek: positive := 8;
 addr_width: positive := 3  -- sirka adresy
);
port(
 reset: in std_logic;
 wr: in std_logic;
 rd: in std_logic;
 din: in std_logic_vector((sirka - 1) downto 0);
 dout: out std_logic_vector((sirka - 1) downto 0);
 full: out std_logic := '0';
 empty: out std_logic :='1';
 half: out std_logic := '0';
 words: out std_logic_vector(addr_width downto 0)
);
end entity fifo;
```

Komponenta `FIFO` slouží k ukládání událostí, které nastaly (např. zápis do IR hodnoty 0x41, zápis do MAB hodnoty 0x158B ap.). Jedná se o klasickou datovou strukturu – frontu – kde se položky vybírají ze začátku fronty a ukládají na její konec. Na vstupní vektor `din` jsou přivedena vstupní data

a sestupnou hranou na vstupu `wr` je proveden zápis. Naopak čtení probíhá sestupnou hranou na vstupu `rd` a vybraná položka je umístěna na výstupní vektor `dout`. Komponenta také obsahuje výstupní signály informující o zaplněnosti fronty.

1. Signál `empty` – ve stavu log. 1 pokud je fronta **prázdná**.
2. Signál `half` – ve stavu log. 1 pokud je fronta **z poloviční části zaplněna**.
3. Signál `full` – ve stavu log. 1 pokud je fronta **plná**.

Tato komponenta obsahuje tři generické parametry. Parametr `sirka` slouží k určení šířky jedné položky ve frontě, parametr `polozek` určuje velikost fronty a parametr `addr_width` určuje počet bitů potřebný pro adresování těchto položek.



Obrázek 6.6: Ukázka zápisu resp. výběru položky z fronty.

Formát uložených dat Nyní si popíšeme v jakém tvaru budou data uložena. Naším cílem je zaznamenat jaká hodnota a kam, do jakého registru, byla zapsána. Dále by data od sebe měla být také oddělena nějakým oddělovacím znakem. Tento znak bude sloužit také jako synchronizační – pro případnou ztrátu *bytu* při vysílání. V našem konkretním případě nám na identifikace cílového registru postačí tři bity (jelikož MSP430 využívá pět registrů). Protože ale chceme mít jednotlivé části zarovnané na *byte*, použijeme na část `CÍL` 8 bitů. Šířka části `DATA` bude mít šířku největšího DR, které dané zařízení obsahuje. V našem případě MSP430 obsahuje tři 16 bitové dat. registry, a tak šířka části `DATA` bude 16 bitů. Oddělovací znak by měl mít takovou hodnotu, která se nebude vyskytovat v předešlých částech. Nicméně, oddělovací znak nebudeme do fronty ukládat (neboť by byl u každé položky stejný) a bude přidán až na vstupu do komponenty `UART`. Celkem tedy položka ve frontě bude široká 24 bitů.

cíl[8b]	data[16b]

Velikost fronty Komunikace na rozhraní JTAG probíhá ve vysokých rychlostech (u MSP430 je rychlost načítání dat 4 MHz) a maximální rychlost vysílání po sériovém kanálu je 115200 b/s. Jelikož je vysílání pomalejší než ukládání, je nutné, aby fronta měla dostatečnou velikost pro snížení pravděpodobnosti jejího zahlcení. Pro zvýšení přenosové rychlosti lze využít USB převodník.

Komponenta TAP

```
entity tap is
port(
  TDI: in std_logic;
  TDO: in std_logic;
  TMS: in std_logic;
  TCK: in std_logic;
  JTAG_RESET: in std_logic;
  txd_out: out std_logic;
  clk_50M: in std_logic
);
end entity tap;
```

Komponenta `TAP` slouží k řízení celého zařízení. Nejdůležitější částí této komponenty je již známý stavový automat neboli řadič TAP (viz obr. 4.4). Hlavičku této *entity* si nyní popisovat nebudeme, neboť její vstupy a výstupy byly popsány v úvodu této kapitoly. Popíšeme si ale, některé vnitřní signály a konstanty.

```
  constant IRLength: integer := 8;
  constant DRLength: integer := 16;
  constant dr_count: integer := 5;
  constant odd_znak: std_logic_vector(7 downto 0) := x"0A";
```

Konstanty Tyto konstanty lze měnit v závilosti na cílovém zařízení. Konstanta `IRLength` resp. `DRLength` určuje šířku instrukčního resp. největšího datového registru. Parametr `dr_count` určuje celkový počet registrů a parametr `odd_znak` určuje hodnotu oddělovacího znaku. Na základě těchto parametrů jsou dále definovány rozměry všech potřebných registrů.

```
  signal instruction:std_logic_vector((IRLength - 1) downto 0);
```

```
  signal data: std_logic_vector((DRLength - 1) downto 0);
  signal write_fifo: std_logic := '0';
  signal read_fifo: std_logic := '0';
  signal start: std_logic := '0';
```

Vnitřní signály Vnitřní signály slouží k propojení jednotlivých komponent nebo k uchování informace (klopný obvod, registr). Na výpisu výše jsem na ukázku vybral některé z použitých signálů. Signál `instruction` slouží pro uložení aktuálně načtené instrukce. Obdobně signál `data` slouží pro uložení načtených dat. Hodnota aktuálně načtené instrukce udává DR, do kterého následně budou v cílovém zařízení nasouvána data. Celé načtení instrukce do IR probíhá tak, že ve stavu `Capture-IR` je vybrán instrukční registr (signál `instruction`) do datové cesty mezi TDI a TDO. Ve stavu `Shift-IR` je pak instrukce nasouvána do IR a ve stavu `Update-IR` je následně uložena do fronty (komponenty `FIFO`). Uložení události do fronty je zařízeno tak, že ve stavu `Update-IR` je vytvořena sestupná hrana na signálu `write_fifo`. Je nutné, aby dostatečně včas před sestupnou hranou zápisového signálu byla dodána data na vstup komponenty `FIFO` – tj. hodnota instrukce doplněná o číslo registru (v případě IR to je 0). Uložení informace o načtených datech probíhá podobně. Časové průběhy signálů komponenty `FIFO` jsou vidět na obr. 6.6. Komponenta `TAP` kromě řadiče TAP obsahuje ještě jeden stavový automat. A to automat pro ovládání vysílání po sériovém kanálu. Tento automat je řízen hodinovým signálem `clk_115200` a slouží ke generování signálů `read_fifo` a `start`. Princip funkce je takový, že pokud komponenta `UART` nevysílá (signál `busy` se nachází ve stavu log. 0) a fronta není prázdná (signál `empty` se nachází ve stavu log. 0), automat vytvoří sestupnou hranu na signálu `read_fifo` pro načtení položky z fronty a následně uvede signál `start` do stavu log. 1. Tato náběžná hrana *odstartuje* vysílání položky přivedené na vstupu komponenty. Časové průběhy signálů komponenty `UART` jsou vidět na obr. 6.5.

Komponenta JTAG_ANALYZ

```
entity JTAG_analyz is
port(
  TDI_in: in std_logic;
  TDO_in: in std_logic;
  TMS_in: in std_logic;
  TCK_in: in std_logic;
  JTAG_RESET_in: in std_logic;
```

```
  TDI_out: out std_logic;
  TDO_out: out std_logic;
  TMS_out: out std_logic;
  TCK_out: out std_logic;
  JTAG_RESET_out: out std_logic;

  txd_out: out std_logic;
  clk_50M: in std_logic
);
end entity JTAG_analyz;
```

Tato komponenta pouze propojuje vstupy `X_in` s výstupy `Y_out` a připojuje je ke komponentě `TAP`:

```
radic: tap port map(TDI_in, TDO_in, TMS_in, TCK_in
                    JTAG_RESET_in, txd_out, clk_50M);
```

6.2 Implementace přijímače v PC

Nyní si popíšeme jak vytvořit jednoduchou konzolovou aplikaci v jazyce C++ pro příjem dat ze sériového portu. Bez použití nějaké multiplatformní knihovny je práce se sériovým portem závislá na operačním systému. Vzhledem k tomu, že analyzátor byl simulovaný v programu ModelSim pod systémem Windows a že pro ověření jeho funkčnosti bylo použito IDE pro MSP430 také pod systémem Windows, tak i tuto aplikaci budeme psát pro systém Windows. Se sériovým portem se pracuje podobně jako se souborem. Pro otevření použijeme funkci `CreateFile`.

```
HANDLE hComm = CreateFile(TEXT(PORT_NAME),
                          GENERIC_READ | GENERIC_WRITE,
                          0,
                          0,
                          OPEN_EXISTING,
                          FILE_FLAG_OVERLAPPED,
                          0);
```

Pokud otevření daného portu proběhne úspěšně, provedeme jeho nastavení pomocí struktury `DCB`.

```
DCB dcb;
memset(&dcb,0,sizeof(dcb));
dcb.DCBlength = sizeof(dcb);
```

```
dcb.BaudRate = CBR_115200;
dcb.fBinary = 1;
dcb.Parity = EVENPARITY;
dcb.StopBits = ONESTOPBIT;
dcb.ByteSize = 8;
SetCommState(hComm,&dcb);
```

Následně můžeme z otevřeného portu číst přijatá data funkcí `ReadFile`.

```
ReadFile(hComm, lpBuf, READ_BUF_SIZE, &dwRead, &osReader);
```

Po ukončení čtení dat je vhodné port uzavřít pomocí funkce `CloseHandle`.

```
CloseHandle(hComm);
```

Číst budeme po 4 bytech do proměnné `lpBuf`. Ze sekce 6.1.2 víme, jaký je formát přijatých/vysílaných dat, tak je můžeme ve vhodném tvaru vypsat na obrazovku a do souboru. V případě, že byla zapsána nedefinovaná instrukce, tak její hodnotu zaznamenáme. Jak je vidět ve výpisu přiloženého v příloze B.1, při programování procesoru MSP430 jsou použity mimo jiné i instrukce, které nejsou popsány v datasheetu k MSP430.

Výpisy z aplikace jsou přiloženy v příloze B. Nachází se zde ukázka výpisu získaného z aplikace pri debugování procesoru MSP430 a také seznam nedefinovaných instrukcí v datasheetu. Kompletní zdrojové kódy analyzátoru (VHDL) a příjmače (C++) jsou uloženy na přiloženém CD disku.

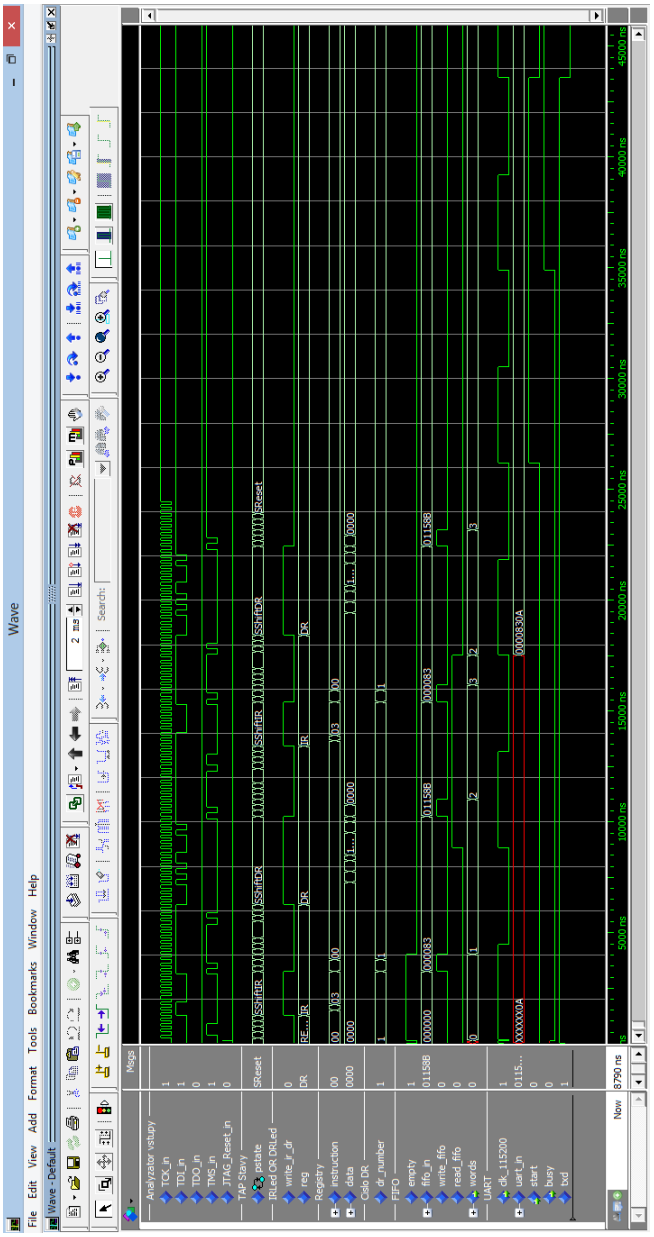
7 Závěr

Rozhraní JTAG je v dnešních době velmi často používané, ať už pro testování obvodů nebo pro programování mikroprocesorů. Cílem práce bylo popsat vlastnosti tohoto rozhraní a na základě nabytých znalostí vytvořit analyzátor.

Práce je rozdělena do čtyř částí. V první části bylo vysvětleno proč skupina JTAG usilovana o dosažení dohody, která by stanovovala princip konstrukce integrovaných obvodů. V druhé části byl čtenář podrobně seznámen s vlastnostmi rozhraní. S tím, jaké obsahuje registry, které instrukce je využívají a k čemu jednotlivé instrukce jsou určeny. V další části jsme provedli analýzu, prostudovali možné architektury a tu, o které jsme byli přesvědčeni, že je nejvhodnější, jsme vybrali. V poslední části byly popsány jednotlivé komponenty námi navrženého analyzátoru.

JTAG analyzátor jsme úspěšně zrealizovali a ověřili jeho funkčnost. Díky tomuto analyzátoru můžeme zaznamenávat jaká data a do jakých registrů byla zapisována. Analyzátor lze modifikovat pro libovolná JTAG kompatibilní zařízení. Mezi možné vylepšení by patřilo zejména použití USB převodníku pro dosažení vyšší rychlosti vysílání dat do PC. Dalším vylepšením by mohlo být např. použití LCD displeje. Tím bychom se obešli bez vysílání dat do PC. Toto rozhraní využívají velké světové firmy, a tak si myslím, že vytvoření analyzátoru bude přínosem.

A Ukázka simulace



Obrázek A.1: Ukázka simulace analyzátoru v programu ModelSim 10.1c.

Na levé straně snímku obrazovky jsou vidět názvy signálů a v prostřední části vidíme jejich časové průběhy.

B Ukázky výpisů

B.1 Debugování procesoru MSP430

```
--------------------------------------------------------
---        ZAPADOCESKA UNIVERZITA V~PLZNI             ---
---  KATEDRA INFORMATIKY A~VYPOCETNI TECHNIKY         ---
--- BAKALARSKA PRACE - ANALYZATOR ROZHRANI JTAG       ---
---          Lukas Vyskrabka - A10B0390P             ---
--------------------------------------------------------
Oteviram COM1...
Seriovy port otevren.
Seriovy port nastaven na: 115200 b/s, 8 datovych bitu, suda
   parita, 1 stop bit.
Ctu z "COM1"...

Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x1501
Zapis do IR:                    0x14    <=> IR_CNTRL_SIG_CAPTURE
Zapis do CS:                    0x0000
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0C01
Zapis do CS:                    0x0401
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0501
Zapis do IR:                    0x0C    <=> !NEZNAMA INSTRUKCE!
Zapis do neznameho registru:    0x0E
Zapis do IR:                    0x14    <=> IR_CNTRL_SIG_CAPTURE
Zapis do CS:                    0x0000
Zapis do IR:                    0x83    <=> IR_ADDR_16BIT
Zapis do MAB:                   0x0015
Zapis do IR:                    0x42    <=> !NEZNAMA INSTRUKCE!
Zapis do neznameho registru:    0x00
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0500
Zapis do IR:                    0x83    <=> IR_ADDR_16BIT
Zapis do MAB:                   0x0015
Zapis do IR:                    0x85    <=> IR_DATA_TO_ADDR
Zapis do MDB:                   0x5A80
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0501
Zapis do IR:                    0x84    <=> IR_ADDR_CAPTURE
Zapis do MAB:                   0x0000
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0500
Zapis do IR:                    0x83    <=> IR_ADDR_16BIT
Zapis do MAB:                   0x0011
```

```
Zapis do IR:                    0x85    <=> IR_DATA_TO_ADDR
Zapis do MDB:                   0x960A
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0501
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0500
Zapis do IR:                    0x83    <=> IR_ADDR_16BIT
Zapis do MAB:                   0x0011
Zapis do IR:                    0x85    <=> IR_DATA_TO_ADDR
Zapis do MDB:                   0x0001
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0501
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0500
Zapis do IR:                    0x83    <=> IR_ADDR_16BIT
Zapis do MAB:                   0x0011
Zapis do IR:                    0x85    <=> IR_DATA_TO_ADDR
Zapis do MDB:                   0x968A
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0501
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0500
Zapis do IR:                    0x83    <=> IR_ADDR_16BIT
Zapis do MAB:                   0x0011
Zapis do IR:                    0x85    <=> IR_DATA_TO_ADDR
Zapis do MDB:                   0x0001
Zapis do IR:                    0x13    <=> IR_CNTRL_SIG_16BIT
Zapis do CS:                    0x0501
Zapis do IR:                    0x0D    <=> !NEZNAMA INSTRUKCE!
Zapis do neznameho registru:    0x00
Zapis do neznameho registru:    0x00
Zapis do neznameho registru:    0x00
Zapis do neznameho registru:    0x00
```

Výpis B.1: Ukázka výpisu z příjmače získaného při debugování procesoru MSP430 pomocí IAR Embedded Workbench. Nachází se zde i datasheetem nepopsané instrukce. Název souboru: `log.txt`

B.2 Debugování procesoru MSP430 – neznámé instrukce

```
-------------------------------------------------------------
-------------------------------------------------------------
---          ZAPADOCESKA UNIVERZITA V~PLZNI                ---
---    KATEDRA INFORMATIKY A~VYPOCETNI TECHNIKY            ---
--- BAKALARSKA PRACE - ANALYZATOR ROZHRANI JTAG             ---
---          Lukas Vyskrabka - A10B0390P                   ---
-------------------------------------------------------------
-------------------------------------------------------------
Seznam pouzitych, ale v datasheetu nepopsanych instrukci:

- 0x0D
- 0x17
- 0x87
- 0x0C
- 0x42
- 0x0B
```

Výpis B.2: Kompletní výpis souboru s datasheetem nepopsanými instrukcemi získaný při debugování procesoru MSP430 pomocí IAR Embedded Workbench. Název souboru: `nezname_instrukce.txt`

C Uživatelská dokumentace

C.1 Modifikace analyzátoru pro jiná zařízení

Analyzátor byl navržen s ohledem na to, aby byl snadno modifikovatelný pro různá zařízení. Parametry JTAG rozhraní jsou v různých JTAG kompatibilních zařízeních různé. Každé takové zařízení může mít např. různě široký instrukční registr i datové registry. Datových registrů navíc může mít i různý počet. Proto jsou tyto parametry v analyzátoru definované jako generické nebo konstanty. Nyní si tyto parametry uvedeme.

Komponenta	**Soubor**	**Parametr**	
		generický	**konstanta**
UART	uart.vhdl	`pocet_bitu`	-
FIFO	fifo.vhdl	`sirka`	-
		`polozek`	-
		`addr_width`	-
TAP	tap.vhdl	-	`IRLength`
		-	`DRLength`
		-	`dr_count`
		-	`odd_znak`

Tabulka C.1: Souhrn parametrů zavislých na zařízení

- Parametr `pocet_bitu` – určuje velikost vstupního vektoru a tím i kolik bytů bude odvysíláno v rámci jedné položky.
- Parametr `sirka` – šířka jedné položky ve frontě. Tato hodnota se rovná hodnotě *pocet_bitu* − *dlka_oddlovacho_znaku* (oddělovací znak se do fronty neukládá, je přidán až na vstup komponenty UART).
- Parametr `polozek` – udává velikost fronty, tj. kolik může obsahovat maximálně položek.
- Parametr `addr_width` – počet bitů potřebných pro adresování všech položek ve frontě ($addr\_width = \log_2 polozek$).
- Parametr `IRLength` – určuje délku instrukčního registru.
- Parametr `DRLength` – hodnota je rovna délce nejdelšího registru v zařízení.

- Parametr `odd_znak` – udává hodnotu oddělovacího znaku.

Dále je nutné v komponentě TAP nastavit pořadí bitů při načítání dat nebo instrukce. Normou není definované který z bitů (LSB, MSB) je přenášen jako první.

C.2 Dokumentace k přijímači ze sériového portu

C.2.1 Modifikace pro jiná zařízení

Jak analyzátor, tak i k němu naprogramovaný přijímač ze sériového portu byl naprogramován s ohledem na to, aby jej bylo možno modifikovat pro jiná zařízení. Nyní si tedy popíšeme, ve kterých souborech se zdrojovými kódy lze měnit různé parametry.

V souboru *msp430.h* lze definovat všechny registry – tzn. jejich názvy a čísla. V souboru *msp430.cpp* se pak nachází funkce `recieved`, která vypisuje přijatá data ve vhodném formátu. Zde lze výpis upravit i pro modifikované registry z hlavičkového souboru. Nastavení sériového portu a názvů výstupních souborů se nachází v souboru *serialport_reader.cpp*, odkud je také volána funkce `recieved` s parametry: počet přijatých bytů, přijatá data, ukazatel na jeden výstupní soubor a ukazatel na druhý výstupní soubor, ve kterém se nachází nedefinované, ale použité instrukce.

C.2.2 Překlad

Ve složce se zdrojovými kódy je přiložen soubor `Makefile`, a proto je program možno přeložit nástrojem `make`, který slouží pro automatický překlad zdrojových kódů. Překlad tedy provedeme příkazem `make`:

```
C:\bakalarska_prace\> make
```

C.2.3 Spuštění

Po úspěšném přeložení lze program spustit z příkazového řádku napsáním názvu spustitelného souboru. Ukázky běhu programu lze nelézt v příloze B.

```
C:\bakalarska_prace\> reader.exe
```

Literatura

[1] *IEEE Standard Test Access Port and Boundary-Scan Architecture,* IEEE Std 1149.1-1990.
Dostupné přes IEEE (1-800-678-IEEE) nebo na CD-ROM od firmy Texas Instruments.

[2] *MSP430 Memory Programming:User's Guide.* Texas Instruments, 2010.
Dostupné z: `http://www.ti.com/lit/ug/slau320i/slau320i.pdf`.