FAKULTA
ELEKTROTECHNICKÁ
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

2015

Pracoviště: Regionální inovační centrum elektrotechniky,
Fakulta elektrotechnická

Výzkumná zpráva č.: 22190-010-2015

Přenos dat z číslicového regulátoru MLC interface v reálném čase

Druh úkolu: vědecko-výzkumný
Řešitelé: Ing. Tomáš Košan, Ph.D.
Vedoucí úkolu: Prof. Ing. Zdeněk Peroutka, Ph.D.
Počet stran: 27
Datum: Červenec 2015
Revize: 1

Tato práce vznikla s podporou projektů SGS-2015-038 a TE02000103 CIDAM

Anotace

Tato práce popisuje jak realizovat přenos dat z MLC interface do PC v reálném čase, respektive v každém cyklu regulační smyčky. V práci jsou rozebrány možné přístupy k tomuto problému, navržena řešení a postupy pro spolehlivý přenos dat. V rámci této práce vznikla také softwarová podpora pro přijímání dat nadřazenou jednotkou (PC).

Abstract

This paper describes how to implement transfer of data from the MLC interface to the PC in real time, respectively, in each cycle of the control loop. The paper discussed possible approaches to this problem, suggested solutions and procedures for reliable data transfer. A part of this work is also support software for receiving data by superior unit (PC).

Seznam symbolů a zkratek

SCI	Serial Communication Interface
DSP	Digital Signal Processor
JTAG	Joint Test Action Group
FPGA	Field Programmable Gate Array

Obsah

Seznam tabulek

Revize	Změny
1	První vydání

1 Úvod

Při ladění pokročilých algoritmů řízení pohonů narazíme na problém přenosu dat algoritmu z řídící jednotky (v našem případě MLC interface) do nadřazeného PC. Data generovaná při běhu algoritmu jsou důležitá především pro precizní odladění funkce navrhovaného algoritmu. Spolu s MLC interface jsou obecně použitelné dva přístupy k získání požadovaných dat:

1. real-time ukládání dat do externí SRAM dostupné na MCU modulu
2. real-time přenos dat do nadřazeného PC

Zásadní nevýhoda metody 1 spočívá v offline zpracování nasnímaných dat a jejich omezeném množství, tato práce se proto zaměří na metodu č.2.

Pro real-time přenos dat z MLC interface do PC lze využít integrovaný převodník USB-SCI. Hardware je navržen tak, že je možné využít tento převodník jak z MCU, tak z FPGA. Dále je možné využít externí převodník USB-SCI, např. ten jenž je součástí JTAG emulátoru JTAGv5.

2 Možnosti propojení MCL interface s PC

V této sekci se zaměříme na rozbor možností propojení MLC interface s nadřazeným PC a zhodnotíme vlastnosti, především přenosovou rychlost jednotlivých řešení.

MLC interface na sobě nese převodník USB-SCI s galvanickým oddělením a přenosovou rychlostí až 3 MBaud založený na obvodu FT232RL. V závislosti na osazeném typu galvanického oddělovače lze teoreticky dosáhnout rychlostí 1 MBaud pro ADuM 1201 ARZ nebo 3 MBaud pro ADuM 1201 BRZ. Dále je možné tento USB-SCI převodník vyřadit a použít externí (pouze u MLC interface v0.2 a vyšší). Externí převodník USB-SCI je součástí JTAG emulátoru JTAGv5.

2.1 Přenosová rychlost SCI linky

U sériového přenosu dat, resp. u přenosu dat po SCI je nutno zohlednit počet přenášených dat (bytů) a četnost přenosů za sekundu. Vzhledem k přítomnosti start a stop bitů u tohoto typu přenosu je výsledný datový tok menší než reálná rychlost přenosu jednotlivých bitů. V celé této práci se předpokládá přenosový formát 8N2, tj. 1 start bit, 8 datových bitů, dva stop bity, žádná parita. Doba přenosu t_t pro N bytů je pak dána rovnicí (1), kde BR je baudrate/bitrate přenosové linky. V našem případě datový paket obsahuje hlavičku (znak STX 0x02) a patičku (znak ETX 0x03) (Obr. 1), pro zvýšení spolehlivosti přenosu jednotlivých paketů dat. Proto

0x02	$DATA_0$	$DATA_1$	...	$DATA_N$	0x03

Obr. 1: Struktura datového paketu posílaného po SCI lince

je ve vzorci (1) $N + 2$. S takto sestaveným paketem, počítá i dekódovací aplikace běžící v nadřazeném PC, viz kapitola 5. Pro výpočet maximálního počtu bytů přenositelných za jednu iteraci regulační smyčky lze použít rovnici (2).

$$t_t = \frac{11 \cdot (N + 2)}{BR} \tag{1}$$

$$N = \frac{t_s \cdot BR}{11} - 2 \tag{2}$$

V případě použití SCI linky pro real-time přenos dat z regulačního algoritmu, je doba t_t zásadní pro zjištění, zda je vůbec možné v jednom kroku regulace přenést do nadřazené jednotky požadovaná data. Doba t_t musí být menší než vzorkovací perioda t_s regulační smyčky. Alternativně je možné data posílat jen jednou za několik iterací regulační smyčky.

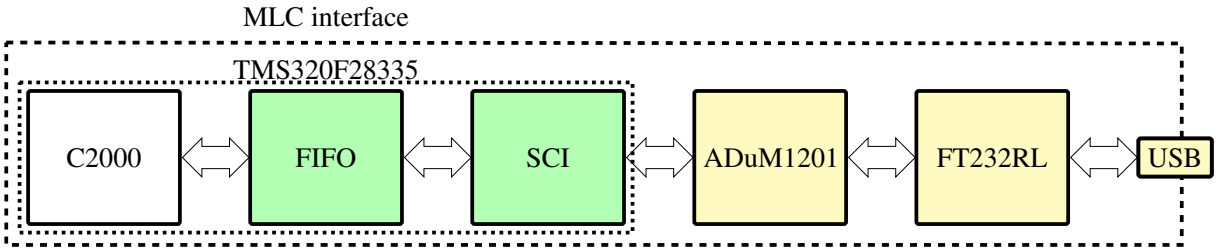
2.2 Integrovaná USB-SCI sériová linka ovládaná z MCU

Nejjednodušší variantou real-time přenosu dat je využití SCI jednotky z MCU. U této metody se využijí funkce z MLC_interface_lib pro nastavení komunikace a přenos dat. Konkrétně u TMS320F28335 je omezena přenosová rychlost na maximálních 1171875 b.s^{-1}. Omezení je dáno limitovanými možnostmi nastavení baudrate generátoru u MCU i FT232RL. Toto řešení umožní přenést (dle rovnice 1) např. 8 hodnot typu *(u)int_16t* za 169 μs.

Z pohledu implementace do MCU je postup následovný:

1. nastavíme baudrate a inicializujeme SCI linku voláním funkce MLC_SCI_init(*BR*)
2. odešleme 16x znak SOH (0x01) pomocí funkce MLC_SCI_send_char(...), toto nadřazené jednotce indikuje start přenosu dat
3. složíme datový paket dle Obr. 1, je třeba si uvědomit, že data se posílají po bytech, tj. např. uint_16t se posílá nadvakrát a to nejdříve nejnižší byte, podrobněji viz kapitola 3
4. odešleme jednotlivé byty paketu pomocí funkce MLC_SCI_send_char(...) (viz Výp. 2)
5. opakujeme body 3. a 4.

SCI periferie má implicitně zapnutu FIFO paměť a lze do ní zapsat až 16 B. Jakmile je plná, funkce MLC_SCI_send_char(...) počká na uvolnění místa ve FIFO a po uvolnění alespoň



Obr. 2: Přenosová trasa SCI linky ovládané z MCU

jednoho byte zapíše do FIFO nový. Blokové schéma přenosové cesty je naznačeno na Obr. 2. Příklad funkce *main()* je uveden na Výp. 1. Uvedený kód zajistí inicializaci samotného mikrokontroléru a hardware na MLC interface. A příklad odeslání paketu je na Výp. 2.

```
#include <stdio.h>
#include "DSP2833x_Device.h"
#include "DSP2833x_Examples.h"
#include "MLC_drv.h"
#include "MLC_ext_module.h"

int main()
{
        int i;
        // basic init of PLL etc.
        InitSysCtrl();
        // Disable CPU interrupts
        DINT;
        DRTM;
        // Initialize the PIE control registers to their default state.
        // This function is found in the DSP2833x_PieCtrl.c file.
        InitPieCtrl();
        // Clear all CPU interrupt flags:
        IER = 0x0000;
        IFR = 0x0000;
        // Initialize the PIE vector table with pointers to the shell
            Interrupt
        // Service Routines (ISR).
        // This function is found in DSP2833x_PieVect.c.
        InitPieVectTable();
        // setup MLC hardware
        // this will enable XINTF and another pins used by MLC interface
        MLC_init();
```

```
        EINT;   // Enable Global interrupt INTM
        ERTM;   // Enable Global realtime interrupt DBGM

        // setup SCI
        MLC_SCI_init(1171875);

        // send initial data
        for (i=0;i<16;i++){
                MLC_SCI_send_char(0x01);
        }
        // endless loop
        while(1);
}
```

Výp. 1: Základní inicializace MLC interface a SCI linky v MCU

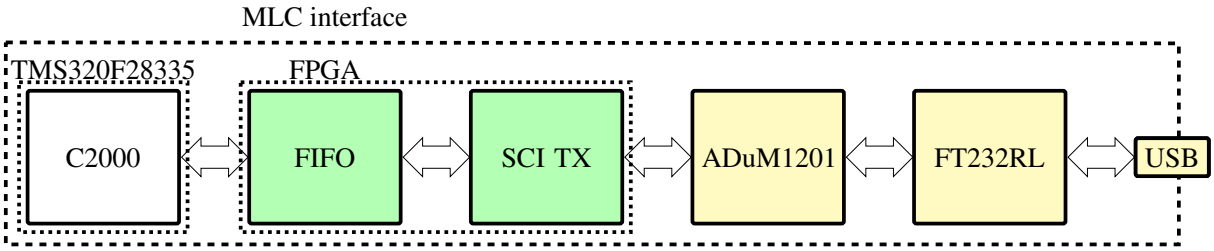
```
MLC_SCI_send_char(0x02);
MLC_SCI_send_char(DATA_0);
...
MLC_SCI_send_char(DATA_N);
MLC_SCI_send_char(0x03);
```

Výp. 2: Odesílání dat pomocí SCI linky v MCU

2.3 Integrovaná USB-SCI sériová linka ovládaná z FPGA

Lepší situace nastává v případě využití speciální entity v FPGA, která zajišťuje odeslání dat přes SCI linku a má lepší možnosti časování, lze tak dosáhnout vyšších přenosových rychlostí, než u varianty z kapitoly 2.2. Dosažitelná rychlost je zde limitována již jen použitým FT232RL na 3 Mbps. Navržená entita je složena ze dvou částí, vstupní FIFO paměti, které lze parametrizovat její velikost, a samotného vysílače dat, blokové schéma viz Obr. 3. Přijímací část nebyla implementována, protože pro daný účel není potřeba. Pro správnou funkci je potřeba FPGA připojit k SCI lince přendáním zkratovací propojky H26 (sektor C4 v kapitole 9) do polohy označené *FPGA+DSP*.

Entita je mapována do paměti MCU a vystačí si s jednou adresou, pro demo firmware je v MLC_interface_lib definováno makro FPGA_SCI_DATA. Paměťové místo je 16 bitů široké, zde je však použito pouze spodních 8 bitů pro zápis dat k odeslání. V případě, že z této adresy čteme, spodní byte je nulový a vrchní indikuje stav entity. MLC_interface_lib definuje příslušné



Obr. 3: Přenosová trasa SCI linky ovládané z FPGA

D15 - D8	D7	D6	D5	D4	D3	D2	D1	D0
X	D_{bit7}	D_{bit6}	D_{bit5}	D_{bit4}	D_{bit3}	D_{bit2}	D_{bit1}	D_{bit0}

Obr. 4: Struktura registru pro zápis do SCI entity

D15 - D10	D9	D8	D7 - D0
0	FPGA_SCI_FIFO_EMPTY	FPGA_SCI_BUSY	0

Obr. 5: Struktura registru pro čtení z SCI entity

masky pro stavové bity, konkrétní popis je na Obr. 4 a 5.

Stavové příznaky registru FPGA_SCI_DATA mají následující význam:

- FPGA_SCI_FIFO_EMPTY - indikuje log. 1, že je FIFO entity zcela prázdná
- FPGA_SCI_BUSY - indikuje úrovní log. 1, že entita vysílá data a zároveň je FIFO paměť plná

Pokud jsou do FIFO zapsána data v případě, kdy je bit FPGA_SCI_BUSY v log. 1, tak jsou tato data zahozena.

Z pohledu MCU se přístup k SCI lince poněkud mění, nevyužíváme již funkcí MLC_SCI_..., ale maker pro přístup do paměti MLC_READ() či MLC_WRITE(). Také není potřeba volat inicializační funkci MLC_SCI_init(). Obecný postup je:

1. inicializujeme MLC interface
2. odešleme 16x znak STH (0x01) pomocí funkce MLC_WRITE(...), tím indikujeme počátek přenosu dat
3. složíme datový paket dle Obr.1
4. odešleme jednotlivé byty paketu pomocí funkce MLC_WRITE(...) (viz Výp. 4)
5. opakujeme body 3. a 4.

Příklad kódu pro funkci *main()* je ukázán na Výp. 3 a příklad odesílání paketu je pak na Výp. 4.

```c
#include <stdio.h>
#include "DSP2833x_Device.h"
#include "DSP2833x_Examples.h"
#include "MLC_drv.h"
#include "MLC_ext_module.h"

int main()
{
        int i;
        // basic init of PLL etc.
        InitSysCtrl();
        // Disable CPU interrupts
        DINT;
        DRTM;
        // Initialize the PIE control registers to their default state.
        // This function is found in the DSP2833x_PieCtrl.c file.
        InitPieCtrl();
        // Clear all CPU interrupt flags:
        IER = 0x0000;
        IFR = 0x0000;
        // Initialize the PIE vector table with pointers to the shell
           Interrupt
        // Service Routines (ISR).
        // This function is found in DSP2833x_PieVect.c.
        InitPieVectTable();
        // setup MLC hardware
        // this will enable XINTF and another pins used by MLC interface
        MLC_init();

        EINT;   // Enable Global interrupt INTM
        ERTM;   // Enable Global realtime interrupt DBGM

        // setup baudrate of FPGA entity
        MLC_WRITE(FPGA_CTRL_REG, FPGA_SCI_12MBDPS);

        // send initial data
        for (i=0;i<16;i++){
                MLC_WRITE(FPGA_SCI_DATA, 0x01);
        }
        // endless loop
```

```
        while(1);
}
```

Výp. 3: Základní inicializace MLC interface a SCI linky v FPGA

```
// check if FIFO is empty
while ((MLC_READ(FPGA_SCI_DATA) & FPGA_SCI_FIFO_EMPTY) !=
   FPGA_SCI_FIFO_EMPTY);
// check if we can transmit another char
while ((MLC_READ(FPGA_SCI_DATA) & FPGA_SCI_BUSY) == FPGA_SCI_BUSY);
MLC_WRITE(FPGA_SCI_DATA, 0x02);
// check if we can transmit another char
while ((MLC_READ(FPGA_SCI_DATA) & FPGA_SCI_BUSY) == FPGA_SCI_BUSY);
MLC_WRITE(FPGA_SCI_DATA, DATA_0);
...
// check if we can transmit another char
while ((MLC_READ(FPGA_SCI_DATA) & FPGA_SCI_BUSY) == FPGA_SCI_BUSY);
MLC_WRITE(FPGA_SCI_DATA, DATA_N);
// check if we can transmit another char
while ((MLC_READ(FPGA_SCI_DATA) & FPGA_SCI_BUSY) == FPGA_SCI_BUSY);
MLC_WRITE(FPGA_SCI_DATA, 0x03);
```

Výp. 4: Odesílání dat pomocí SCI linky v FPGA

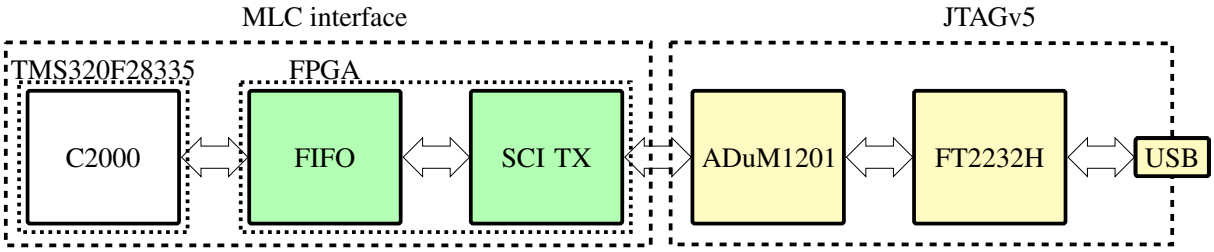
2.4 Externí USB-SCI sériová linka ovládaná z FPGA

Nejvyšší přenosové rychlosti lze dosáhnout použitím externího převodníku USB-SCI. Ten je např. součástí JTAG emulátoru JTAGv5. Použitý obvod FT2232H podporuje přenosové rychlosti až 12 Mbps. Většina JTAGv5 emulátorů má osazeny ADuM1201BRZ, které podporují až 25 Mbps, což umožňuje využít maximální přenosovou rychlost FT2232H. Propojení této datové linky naznačuje Obr. 6.

Co se týče stylu ovládání, ten se shoduje s postupem uvedeným v kapitole 2.3 a tak se nemění ani programová obsluha, jenž tedy odpovídá kódů uvedeným ve Výp. 3 a 4.

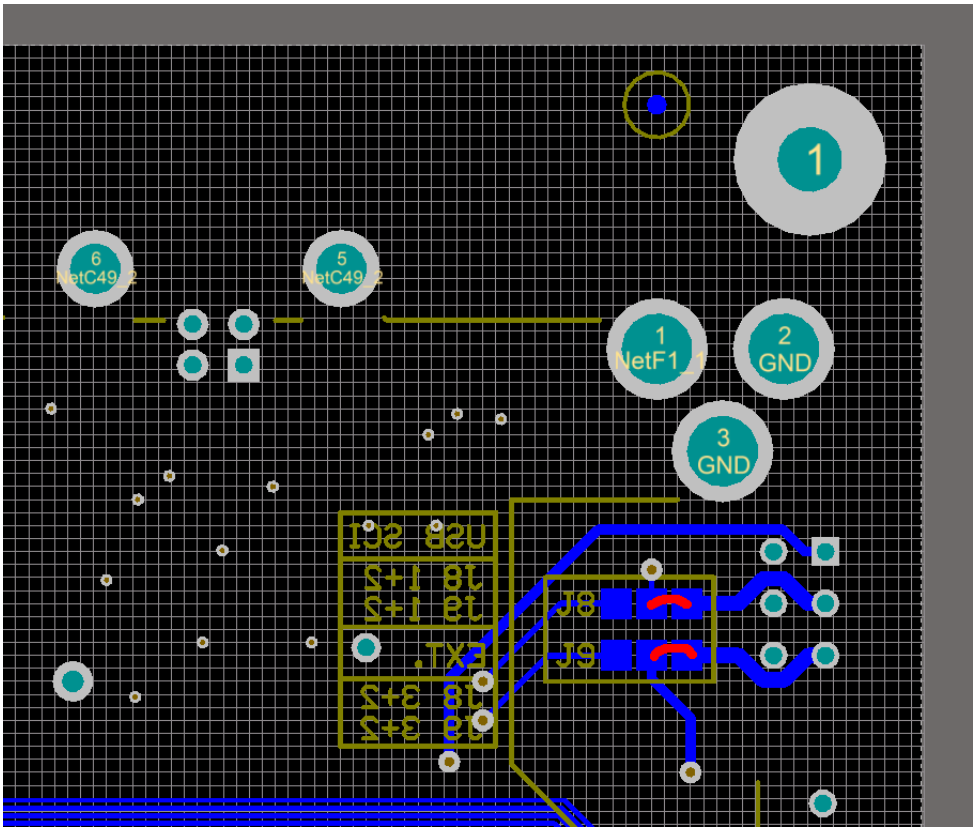
2.4.1 Nastavení MLC interface pro použití externího převodníku USB-SCI

Používání externího USB-SCI převodníku vyžaduje několik zásahů do nastavení hardwaru MLC interface. Zaprvé je nutné použít MLC interface verze 0.2 a vyšší, dále je třeba správně proletovat propojky J8 a J9, jak je naznačeno červenými obloučky na Obr. 7. Propojky J8 a J9



Obr. 6: Přenosová trasa SCI linky ovládané z FPGA využívající JTAGv5

se nacházejí na spodní straně DPS MLC interface v oblasti A6 (viz kapitola 9). JTAGv5 se pak propojí speciálním šestivodičovým kabelem s MLC interface, konkrétně konektorem H34 v sektoru A6. **Je důrazně doporučeno u konektoru H34 odštípnout pin 1, označený jako 3V3 !**



Obr. 7: Proletování propojek J8 a J9 pro použití externího USB-SCI převodníku u MLCv0.2

Dále je potřeba napojit FPGA na SCI linku. Toho se docílí přendáním zkratovací propojky H26 (sektor C4 v kapitole 9) do polohy označené *FPGA+DSP*.

3 Formátování datových typů v MCU pro odeslání přes SCI linku

Aby bylo možno přes osmibitovou SCI linku odesílat podporované typy dat (viz Tab.2, přepínač *-d*), je potřeba patřičné předzpracování v MCU.

V případě hodnot ve formátu UINT8 nebo INT8 není potřeba žádné dodatečné zpracování a bez ohledu na znaménkovost se data k odeslání zapisují postupy ukázanými na Výp. 2 nebo 4.

Při odesílání typů UINT16 a INT16 je jen třeba rozdělit odesílaná data na jednotlivé byte a postupně je odeslat. První se posílá horní byte a následuje spodní byte, postup naznačuje Výp. 5.

```
int16_t data;
...
MLC_SCI_send(data);
MLC_SCI_send(data >> 8);
...
MLC_WRITE(FPGA_SCI_DATA, data);
MLC_WRITE(FPGA_SCI_DATA, data >> 8);
```

Výp. 5: Odesílání UINT16/INT16 dat pomocí SCI linky

Nejsložitější situace nastává pro typ FLOAT, ten je potřeba beze ztráty informace převést na sadu bytů. Toho lze dosáhnout dvěma způsoby:

1. pomocí ukazatelového přetypování
2. využitím unionu

Postup 1, je naznačen níže, jsou možné dvě varianty, jedna s přetypováním za běhu (Výp. 6) a druhá s jednorázovým přetypováním (Výp. 7). Opět platí, že posíláme jako první byte ten nejvíce významný. Obdobně postupujeme u metody s využitím unionu (Výp. 8).

Z pohledu náročnosti převodu dat jsou si ekvivalentní metody s unionem (Výp. 8) a přetypování floatu za běhu (Výp. 6) (obě metody zaberou cca 15 B). Metoda s ukazatelem *p_data* je nejnáročnější a zabere 23 B (Výp. 7). Samotná výpočetní náročnost nebyla změřena. Výpisy kódu ukazují využití SCI entity v FPGA, pokud by byla používána SCI linka v MCU, MLC_WRITE(...) by se nahradila voláním funkce MLC_SCI_send_char(...).

```
float data;
uint32_t* p_data;
...
MLC_WRITE(FPGA_SCI_DATA, *(uint32_t*)&data);
MLC_WRITE(FPGA_SCI_DATA, *(uint32_t*)&data >> 8);
MLC_WRITE(FPGA_SCI_DATA, *(uint32_t*)&data >> 16);
MLC_WRITE(FPGA_SCI_DATA, *(uint32_t*)&data >> 24);
```

Výp. 6: Odesílání FLOAT dat pomocí SCI linky, metoda s ukazatelem

```
float data;
uint32_t* p_data;
...
p_data = (uint32_t*)&data;
MLC_WRITE(FPGA_SCI_DATA, *p_data);
MLC_WRITE(FPGA_SCI_DATA, *p_data >> 8);
MLC_WRITE(FPGA_SCI_DATA, *p_data >> 16);
MLC_WRITE(FPGA_SCI_DATA, *p_data >> 24);
```

Výp. 7: Odesílání FLOAT dat pomocí SCI linky, metoda s ukazatelem

```
union data_mix{
        float f;
        uint32_t i;
}

union data_mix data;
...
data.f = 1.25689845;
MLC_WRITE(FPGA_SCI_DATA, data.i);
MLC_WRITE(FPGA_SCI_DATA, data.i >> 8);
MLC_WRITE(FPGA_SCI_DATA, data.i >> 16);
MLC_WRITE(FPGA_SCI_DATA, data.i >> 24);
```

Výp. 8: Odesílání FLOAT dat pomocí SCI linky, metoda s union

4 Firmware pro FPGA

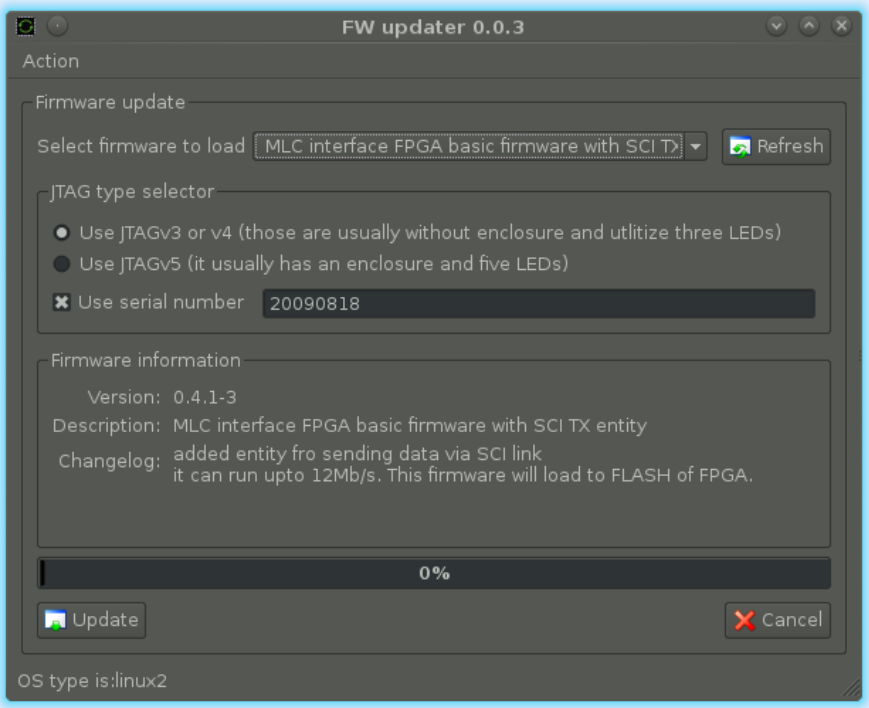
Použití SCI vysílače v FPGA vyžaduje mít v FPGA nahrán příslušný firmware. Doporučený postup nahrání je pomocí programu *MLC firmware updater*. Tento program lze získat z [1],

adresář *Software_PC/FW_updater*. Aplikace se instaluje spuštěním setup.exe. Po dokončení instalace a spuštění programu lze vybrat firmware pro nahrání do FPGA (Obr. 8).

Jsou podporovány JTAG emulátory v3, 4 a 5. Ty je nutno přes redukci ze 14 na 10 pinů připojit k programovacímu konektoru FPGA (sektor A3, konektor H11 - FPGA JTAG, viz. Kap. 9). Pozor nezapojovat do konektoru označeného *Prog. EPCS*.

Ze seznamu dostupných firmware si vybereme položku *MLC interface FPGA basic firmware with SCI entity* (Obr. 8). Také je nutné vybrat správný typ JTAG emulátoru a je vhodné zadat i jeho sériové číslo. Kliknutím na tlačítko *Update* se spustí nahrání firmware do konfigurační FLASH FPGA. Přenos firmware trvá cca 2 minuty, po jeho dokončení je vhodné vypnout a zapnout MLC interface.

Správně nahraný firmware se ohlásí po zapnutí na displeji textem *MLC test T.Kosan 2012/15*. Následně vypíše na displej i verzi firmware: 100.0.



Obr. 8: Aplikace FW updater

4.1 Možnosti SCI entity v FPGA

Navržená entita v FPGA má nastavitelnou rychlost odesílání, implicitně používá formát odesílaného byte 8N2 a baudrate 12 Mb.s^{-1}. Hlavičkový soubor MLC_drv.h definuje konstanty pro

nastavení podporovaných rychlostí přenosu dat dle tabulky Tab. 1. Tyto konstanty se zapisují do registru FPGA_CTRL_REG, jak je ukázáno na Výp. 9. Je potřeba upozornit, že registr FPGA_CTRL_REG využívají pro svou konfiguraci i další entity v FPGA a postup na Výp. 1 je přepíše na nuly. Postupy jak se tomuto problému vyhnout by měly být zřejmé.

Tab. 1: Konstanty pro nastavení rychlosti přenosu SCI entity v FPGA

Konstanta	**Rychlost**
FPGA_SCI_12MBDPS	12 $Mb.s^{-1}$
FPGA_SCI_6MBDPS	6 $Mb.s^{-1}$
FPGA_SCI_3MBDPS	3 $Mb.s^{-1}$
FPGA_SCI_1MBDPS	1 $Mb.s^{-1}$
FPGA_SCI_912KBDPS	912 $kb.s^{-1}$

```
MLC_WRITE(FPGA_CTRL_REG, FPGA_SCI_12MBDPS); // set 12Mbps
...
MLC_WRITE(FPGA_CTRL_REG, FPGA_SCI_3MBDPS);  // set 3Mbps
```

Výp. 9: Nastavení rychlosti SCI entity v FPGA

5 Programové vybavení pro PC

V této kapitole je popsáno programové vybavení pro příjem, dekódování a ukládání dat na straně nadřazené jednotky. Vznikly dvě aplikace, skript v Pythonu *serial2data.py* a program v jazyce C *s2d_v4*. Oba dva generují shodná výstupní data, hlavním rozdíl je ve výpočetní náročnosti, kdy skript v Pythonu nezvládá příjem dat s velmi krátkými prodlevami mezi jednotlivými pakety. Nadruhou stranu poskytuje o něco vyšší komfort ovládání a případné změny v kódu nevyžadují rekompilaci.

Obě aplikace jsou multiplatformní, podporovány jsou systémy Linux a Windows.

5.1 Program serial2data.py

Pro správnou funkcionalitu skriptu *serial2data.py* je potřeba interpret Pythonu verze 2.7 [2] s nainstalovaným rozšířením pro sériovou linku *pyserial* [3]. Skript samotný je dostupný na [1], adresář serial2data/src.

Tento skript zajišťuje:

1. synchronizaci s během algoritmu v MCU na MLC interface, dle postupu popsaného v kapitolách 2.2, 2.3

Tab. 2: Tabulka přepínačů skriptu serial2data.py

Přepínač	**Funkce**
-f, --format	formát dat pro dekódování, podporované typy jsou H → UINT16, h → INT16 a f → FLOAT, jednotlivé typy lze libovolně kombinovat
-p, --port	port použitý pro přenos dat, pro Linux většinou /dev/ttyUSBx, ve Windows COMx
-b, --baudrate	požadovaná rychlost přenosu dat
-h, --help	vypíše nápovědu k programu
-v, --verbose	zapne více informativních výpisů programu (debug)
-o, --output	soubor do kterého se přijatá a dekódovaná data zapíší
-c, --columns	čárkou oddělený seznam hlaviček jednotlivých datových sloupců
-u, --units	čárkou oddělený seznam jednotek pro jednotlivé datové sloupce

2. příjem paketů dat a jejich rozkódování
3. ukládání přijatých rozkódovaných dat do souboru na disku ve formátu CSV s oddělovačem čárka
4. podporuje opakovaný běh s ukládáním dat do souborů s časovými značkami

Skript je napsaný pro příkazový řádek a umožňuje nastavit základní vlastnosti ukládaných dat. Nápovědu k používání lze zobrazit pomocí spuštění s parametrem *-h* nebo *--help*. Seznam přepínačů se stručným popisem viz Tab. 2. Skript má ošetřeno pokud nezadáme některé parametry, nebo jich zadáme moc (typicky délka popisku sloupců a jednotky), a má nastaveny defaultní hodnoty pro přepínače, které jsou nutné pro základní funkci. Tučně zvýrazněné parametry jsou povinné, zde minimálně *-f*.

5.1.1 Ovládání skriptu serial2data.py

Skript serial2data.py je určen pro spouštění v příkazové řádce. Po spuštění skript čeká na inicializační sadu dat (16x ASCII znak STH - 0x01). Čekání lze kdykoli přerušit stiskem klávesy q nebo Q a potvrzením pomocí stisku Enter. Skript informuje o základních parametrech, např. jak budou vypadat hlavičky v souboru a kam se zapisují informace o špatně přijatých paketech (log file).

```
xxx@xxx:/$ python serial2data.py -b 12000000 -p /dev/ttyUSB1 -f ffff
serial2data.py v0.3
Baud rate: 12000000
Using log file: /tmp/s2d_err.log
```

```
Wrn: header is too short, it will be extended.
New header: CH 0,CH 1,CH 2,CH 3
Wrn: header is too short, it will be extended.
New header: -,-,-,-
Inf: waiting for sync data ... press q + enter to quit.
```

Pokud skript obdrží platný začátek přenosu dat, začíná samotný příjem paketů s daty:

```
Sync ok, fetching data ... to interrupt press q and enter.
Creating a new file xxx.dat
```

Jakmile dojde k vypršení časového limitu příjmu dat (typicky když je program v MCU zastaven) nebo když uživatel běh skriptu přeruší pomocí stisku q + enter, skript dá uživateli na výběr tři možné volby jak dále pokračovat:

```
Data receive timeout.
Run again with a new output file ? Choose one option and press Enter:
y/Y (create new file, old data will be saved as x-2015-06-15_13:08:37.txt)
n/N (new data will be appended to existing file)
q/Q (quit)):
```

Stiskem y/Y se stará data zkopírují do nového souboru s časovou značkou a program se vrátí do módu příjmu synchronizace, po úspěšné synchronizaci se přijatá data ukládají do prázdného souboru s původním názvem. Stiskem n/N se skript také přepne do módu počáteční synchronizace, ale nově přijatá data se přidají ke stávajícím na konec souboru. Volbou klávesy q/Q se skript ukončí.

5.2 Program s2d_v4

Narozdíl od skriptu serial2data.py je aplikace s2d_v4 napsána v jazyce C, což vede na vyšší spolehlivost přenosu dat (nižší nároky na výkon procesoru). Aplikace je k dispozici pro dvě hlavní platformy Linux a Windows. Obě verze sdílejí shodný zdrojový kód a přeložené binární soubory lze najít na [1], adresář Software_PC/s2d_v4/build_lin, resp. build_win. Program se neinstaluje, uživatel si jej pouze zkopíruje do vhodného adresáře, a to včetně případných .dll či .so knihoven.

Tento program zajišťuje:

1. synchronizaci s během algoritmu v MCU na MLC interface, dle postupu popsaného v kapitolách 2.2, 2.3
2. příjem paketů dat a jejich rozkódování
3. ukládání přijatých rozkódovaných dat do souboru na disku ve formátu CSV s oddělovačem čárka
4. umožňuje vypsat seznam portů dostupných v systému

5.2.1 Ovládání programu s2d_v4

Seznam přepínačů pro program s2d_v4 je shrnut v Tab. 3. Narozdíl od verze v Pythonu, nejsou podporovány dlouhé formy parametrů (--units atd.). Jinak je význam přepínačů a jejich parametrů shodný u obou verzí aplikace. Přibyl pouze přepínač -l. Tučně zvýrazněné parametry jsou povinné, zde minimálně *-f*.

Tab. 3: Tabulka přepínačů skriptu serial2data.py

Přepínač	**Funkce**
-f	formát dat pro dekódování, podporované typy jsou H → UINT16, h → INT16 a f → FLOAT, jednotlivé typy lze libovolně kombinovat
-p	port použitý pro přenos dat, pro Linux většinou /dev/ttyUSBx, ve Windows COMx
-b	požadovaná rychlost přenosu dat
-h	vypíše nápovědu k programu
-v	zapne více informativních výpisů programu (debug)
-o	soubor do kterého se přijatá a dekódovaná data zapíší
-c	čárkou oddělený seznam hlaviček jednotlivých datových sloupců
-u	čárkou oddělený seznam jednotek pro jednotlivé datové sloupce
-l	vypíše seznam sériových portů a ukončí se

5.3 Formát výstupních souborů

Obě výše zmiňované aplikace generují dva soubory, samotný datový soubor se jménem dle zadání uživatele a dále logovací soubor, kde jsou reportovány špatně přijaté pakety dat.

Skript generuje datový výstupní soubor ve formátu CSV s čárkou jako oddělovačem. Jeden paket dat je oddělen znakem nového řádku (\n). Pro lepší možnost archivace nasnímaných dat, je soubor uveden hlavičkou s datem a časem vytvoření, druhý řádek obsahuje názvy jednotlivých sloupců dat a na třetím řádku jsou uvedeny jednotky pro datové sloupce. Druhý

a třetí řádek může využít KST (viz [4]) pro automatické popsání zobrazovaných grafů. Od čtvrtého řádku dále následují data v textové formě. První sloupec je index přijatých dat, od druhého sloupce jsou zapisována přijatá dekódovaná data. Příklad začátku souboru je uveden níže.

```
# Created by s2d at: Fri Jun 26 13:17:58 2015
IDX,CH0,CH1,CH2,CH3
-,-,-,-,-
1,0,9.88434,0,0
2,1,9.88403,0.000305176,-0.000305176
```

Formát logovacího souboru je uveden níže, jde jen o informaci o špatně přijatých datových paketech, uživatel je pak schopen zjistit kde má v nasnímaných datech mezeru. Do samotného datového souboru jsou cíleně vloženy nulové hodnoty. Soubor s daty logu se vytváří v adresáři skriptu (uživatel zde tedy musí mít právo zápisu) a je při každém novém spuštění přepsán.

```
Failed to receive data at index: 175398.
Failed to receive data at index: 175455.
```

5.4 Příklady použití aplikací serial2data.py a s2d_v4

Nejjednodušší možné použití spočívá v zadání portu, rychlosti přenosu a formátu datového paketu, např.:

```
python serial2data.py -b 12000000 -p /dev/ttyUSB0 -f hHfH
```

```
s2d_v4.exe -b 12000000 -p /dev/ttyUSB0 -f hHfH
```

Zde je požadována rychlost přenosu 12 Mbit.s^{-1}, použitý sériový port je /dev/ttyUSB0 a formát paketu je kombinovaný int16_t, uint_16t, float a uint16_t. Programový kód pro paket je uveden níže. Je potřeba datový paket sestavit tak, aby odpovídal zadanému formátu, jinak aplikace není schopna data přijmout.

```
float data3;
uint16_t data2, data4;
int16_t data1;
...
```

```
MLC_WRITE(FPGA_SCI_DATA, 0x02);

MLC_WRITE(FPGA_SCI_DATA, data1);
MLC_WRITE(FPGA_SCI_DATA, (data1 >> 8));

MLC_WRITE(FPGA_SCI_DATA, data2);
MLC_WRITE(FPGA_SCI_DATA, data2 >> 8);
MLC_WRITE(FPGA_SCI_DATA, data2 >> 16);
MLC_WRITE(FPGA_SCI_DATA, data2 >> 24);

MLC_WRITE(FPGA_SCI_DATA, data3);
MLC_WRITE(FPGA_SCI_DATA, (data3 >> 8));

MLC_WRITE(FPGA_SCI_DATA, data4);
MLC_WRITE(FPGA_SCI_DATA, (data4 >> 8));

MLC_WRITE(FPGA_SCI_DATA, 0x03);
```

Výp. 10: Sestavení paketu s formátem hHfH

Je-li třeba definovat i hlavičky a jednotky pro jednotlivé sloupce, je možné je předat jako parametr programu. Např. z MCU odesíláme sadu čtyř hodnot v plovoucí řádové čárce a konkrétně se jedná o U_dc, i_a, i_b, ω. Parametrem *-c* předáme skriptu názvy jednotlivých sloupců a jednotky předáme parametru *-u*. Začátek výsledného souboru viz níže.

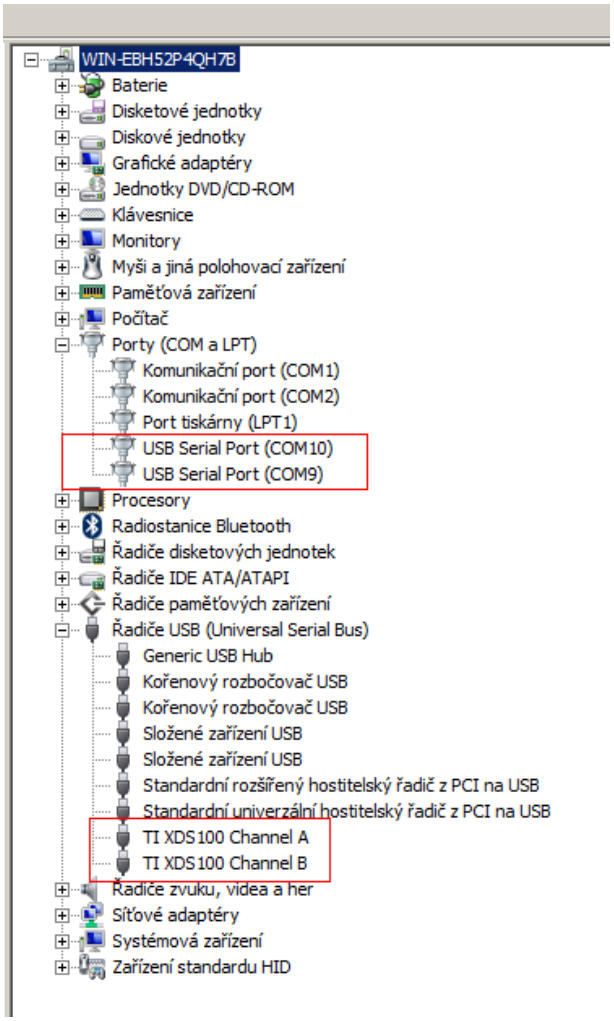
```
s2d_v4.exe -b 12000000 -p /dev/ttyUSB0 -f ffff -c U_dc,i_a,i_b,omega
-u V,A,A,ot/min

# Created by s2d at: Tue Jun 30 12:41:58 2015
IDX,U_dc,i_a,i_b,omega
-,V,A,A,ot/min
1,0,9.89075,0.000305176,0
```

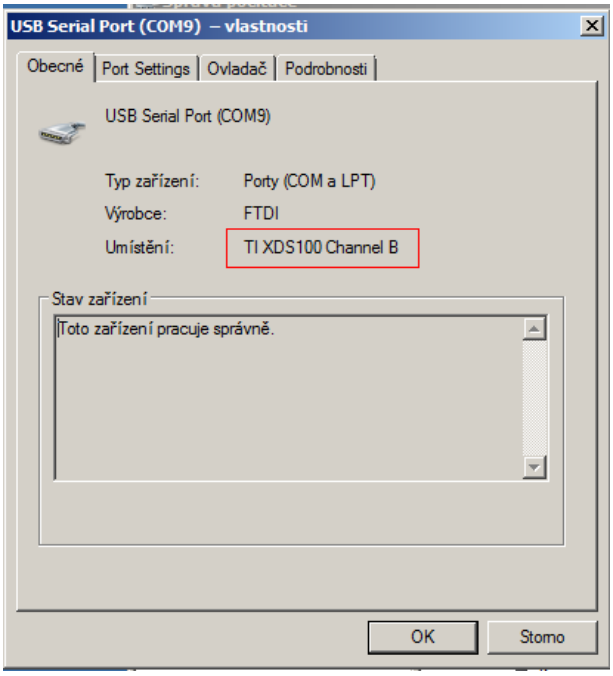
6 Konfigurace JTAGv5 ve Windows

Pro operační systém Windows je potřeba mít správně nainstalovány ovladače pro XDS100v2. Ty jsou součástí instalace Code Composer Studia v4 a vyšší. Správně nainstalovaný JTAGv5

se ve správci zařízení objeví jako TI XDS100 Channel A a B (Obr.9). Pro použití obou aplikací je třeba znát číslo sériového portu. Z vlastností COM portů ve správci zařízení lze toto číslo jednoduše zjistit zobrazením vlastností jednotlivých portů. Ten který má ve vlastnostech umístění TI XDS100 Channel B je třeba předat skriptu pomocí parametru -p/--port.



Obr. 9: Správně nainstalovaný JTAGv5 ve Windows



Obr. 10: Vlastnosti SCI komunikační linky JTAGv5

7 Spolehlivost přenosu dat

Použitý "protokol" přenosu dat se vyznačuje maximální jednoduchostí, která neklade vysoké výpočetní nároky na mikrokontrolér. Toto je však vykoupeno sníženou spolehlivostí přenosu dat, která závisí na několika faktorech:

1. rychlosti přenosu

2. vytížení procesoru nadřazené jednotky
3. počtu přenášených bytů v jednom paketu
4. periodě odesílání paketů

Rychlost přenosu dat se negativně projevuje na odolnosti proti rušení a zároveň určuje dobu za jakou se přeplní FIFO paměť v USB-SCI převodníku. Pro 4 kB FIFO a peridou zasílání 1B dat 30 μs, dojde k přeplnění již za 0,12 s. To úzce souvisí se zatížením procesoru nadřazené jednotky (PC), kdy pokud bude hodně výpočetně zatížený nebo bude čekat na V/V operaci (typicky pevný disk), může být doba 0,12 s nedostatečná pro načtení dat z FIFO do aplikace, resp. pokud aplikaci bude trvat zpracování dat déle než je perioda jejich odesílání, tak FIFO také přeteče.

Dále se na spolehlivosti nepříznivě projevuje velikost přenášených dat, čím více dat v jednom paketu, tím vyšší je šance, že bude přijat špatně a data se zahodí. Následná resynchronizace pak způsobí zahození dalších datových paketů.

Tabulka (Tab. 4) níže shrnuje několik provedených měření pro obě aplikace, při rychlosti 12 Mb.s^{-1} pro různé velikosti datových paketů. Pakety se posílaly s nejmenším možným zpožděním za sebou.

Pokud je mezi pakety vložena mezera, lze dosáhnout větší spolehlivosti příjmu dat. Obecně lze doporučit neposílat pakety častěji než po 30 μs. Naměřené výsledky vlivu periody odesílání paketů na chybovost přenosu dat zobrazuje Tab. 5.

Hodnoty v obou tabulkách je nutno brát pouze jako srovnávací a ne absolutní hodnoty.

Tab. 4: Vliv velikosti paketu dat na spolehlivost přenosu

Paket	Chybovost s2d_v4	Chybovost serial2data.py
64 B	0,0038 %	0,0389 %
32 B	0,0029 %	0,0297 %
16 B	0,00059 %	0,031 %
8 B	0,00049 %	0,005 %

Tab. 5: Vliv periody odesílání dat na spolehlivost přenosu pro data packet 64 B

Perioda	Chybovost s2d_v4	Chybovost serial2data.py
30 μs	0,00848 %	0,039 %
50 μs	0,001 %	0,019 %
100 μs	0,000297 %	0,00099 %

8 Závěr

Tato práce popisuje několik možných řešení přenosu dat z MLC interface do PC v reálném čase. Každé z řešení má své pro a proti, využití pouze možností MCU je jednoduché, ale zároveň nejpomalejší. Naopak použitím FPGA lze dosáhnout vysokých rychlostí přenosu, ale musíme se starat o firmware v FPGA.

Pro základní implementaci přenosu dat práce ukazuje jak převést proměnné mikrokontroléru na bajtovou reprezentaci, kterou je pak možné odeslat po sériové lince.

Pro nadřazenou jednotku (PC) pak jsou dostupné dvě implementace příjmu dat, jedna formou skriptu v jazyce Python a druhá implementovaná v čistém C. Výhodou skriptu v Pythonu je flexibilita a jednoduchá rozšiřitelnost bez nutnosti kompilace. Tento komfort je vykoupen menší spolehlivostí přenosu dat na vyšších rychlostech. Implementace v jazyce C má malou chybovost přenosu avšak je složitější na úpravy. Výstup z obou programů je možné jednoduše zobrazovat pomocí aplikace třetí strany KST ([4]).

Navzdory jednoduchému použitému přenosovému protokolu je spolehlivost přenosu dat dostatečná i při vysokých přenosových rychlostech.

Literatura

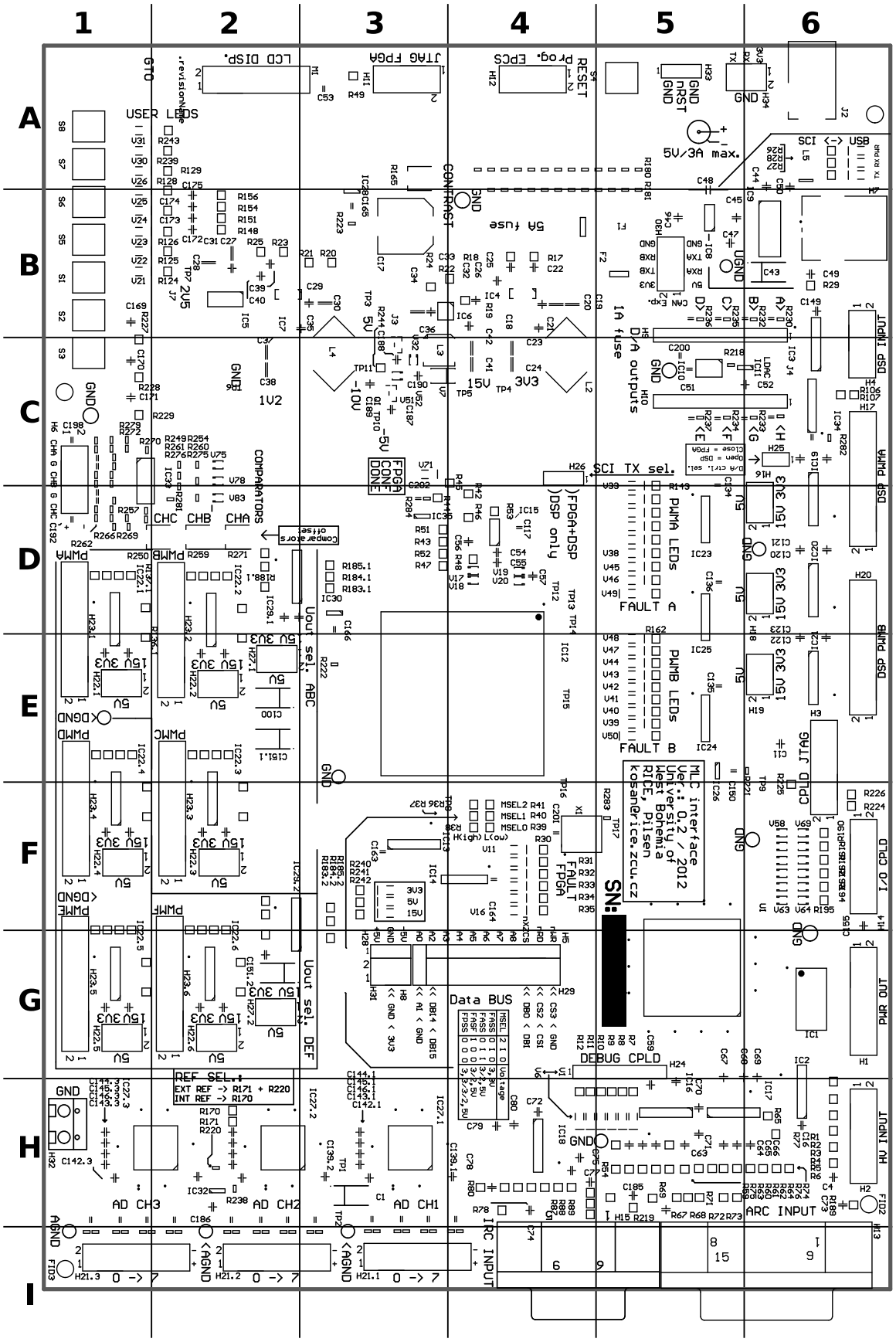
[1] *MLC interface - projektové SVN*. [cit. 14.6.2015]. Dostupný z WWW: ⟨https://riceproject.fel.zcu.cz/svn/projects/2012_eZDSP_kit_interface/13_Project_outputs⟩.

[2] *Python 2.7 Release*. [cit. 14.6.2015]. Dostupný z WWW: ⟨https://www.python.org/download/releases/2.7⟩.

[3] *Welcome to pySerial's documentation*. [cit. 14.6.2015]. Dostupný z WWW: ⟨http://pyserial.sourceforge.net⟩.

[4] *Kst - Visualize your data*. [cit. 14.6.2015]. Dostupný z WWW: ⟨https://kst-plot.kde.org⟩.

9 Příloha A


1
2
3
4
5
6
A
B
C
D
E
F
G
H
I
USER LEDS
LCD DISP.
JTAG FPGA
Prog. EPCS
RESET
CONTRAST
5A fuse
1A fuse
DA outputs
SCI <-> USB
5V/3A max.
DSP INPUT
DSP PWMA
DSP PWMB
I/O CPLD
PWR OUT
HV INPUT
ARC INPUT
IRC INPUT
COMPARATORS
FPGA CONF DONE
FPGA+DSP
DSP only
SCI TX sel.
PWMA LEDs
PWMB LEDs
FAULT A
FAULT B
CPLD JTAG
DEBUG CPLD
FAULT FPGA
Data BUS
MLC interface
Ver.: 0.2 / 2012
University of West Bohemia
RICE, Plzen
kosan@rice.zcu.cz
SN:
REF SEL.:
EXT REF -> R171 + R220
INT REF -> R170
Uout sel. ABC
Uout sel. DEF
AD CH3
AD CH2
AD CH1
ARC INPUT