Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Bakalářská práce

Příklady využití vícejádrového mikroprocesoru XCORE

Autor práce: Roman Severa
Vedoucí práce: Ing. Petr Weissar, Ph.D.

Plzeň 2017

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Roman SEVERA**
Osobní číslo:	**E13B0220P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Název tématu:	**Příklady využití vícejádrového mikroprocesoru XCORE**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

Připravte sadu ukázkových příkladů s vícejádrovým mikroprocesorem xCORE firmy XMOS.

1. Prozkoumejte rozdíly oproti "klasickým" procesorových architekturám.
2. Popište způsob tvorby aplikací.
3. Připravte vzorové aplikace.
4. Vytvořené aplikace ověřte na HW, např. na "XMOS StartKitu".

Rozsah grafických prací:	**podle doporučení vedoucího**
Rozsah kvalifikační práce:	**30 - 40 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí bakalářské práce:	**Ing. Petr Weissar, Ph.D.** Katedra aplikované elektroniky a telekomunikací
Datum zadání bakalářské práce:	**14. října 2016**
Termín odevzdání bakalářské práce:	**8. června 2017**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 14. října 2016

Abstrakt

V práci je vidět rozdíl vícejádrového mikrokontroléru XMOS oproti klasické jednoprocesorové architektuře. Nejprve je velice stručně popsán způsob fungování jednoprocesorové architektury. Dále je popsána poněkud důkladněji architektura XMOS. Cílem práce bylo prozkoumat základní rozdíly vícejádrové architektury a poskytnout tak základní hardwarové informace o této architektuře a zároveň zjistit a popsat způsob tvorby aplikací. Veškeré informace byly získány z firemních dokumentací. Dokumentace firmy XMOS je poněkud obsáhlá a a proto jsou v práci vybrány pouze nejdůležitější vlastnosti. Práce čtenáři poskytne základní informace o platformě XMOS a základní vědomosti co se týče tvorby aplikací pro tuto platformu. Jsou zde vytvořeny jednoduché příklady. Čtenář musí znát programovací jazyk C/C++ a základní informace co se týče mikroprocesorové techniky.

Klíčová slova

XMOS, XCORE, vícejádrový mikrokontrolér, xC

Abstract

Severa, Roman. *Usage of Multicore microprocessor XCORE* [*Příklady využití vícejádrového mikroprocesoru XCORE*]. Pilsen, 2017. Bachelor thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Applied Electronics and Telecommunications. Supervisor: Petr Weissar

---

In the work we can see the difference between the multi-core XMOS microcontroller and classic one-processor architecture. First of all, there is a short description of how the single processor architecture works. Then the XMOS architecture is described somewhat more closely. The aim of the thesis was to investigate the basic differences of multicore architecture and to provide basic hardware information about this architecture and also to find and describe the way of creating applications. All information was obtained from company documentation. XMOS's documentation is somewhat extensive and there are the most important features selected in the thesis. The work will provide the reader with basic information about the XMOS platform and basic knowledge about how to create applications for this platform. Simple examples are created here. The reader has to know the C / C ++ programming language and basic information about the microprocessor technique.

Keywords

XMOS, XCORE, multicore microcontroller, xC

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 7. června 2017

Roman Severa

.....

Podpis

Obsah

Seznam obrázků

Seznam symbolů a zkratek

NVIC	Nested Vectored Interrupt Controller. Řadič přerušení.
GPIO	General-purpose input/output. Vývody pro obecné účely
I/O	Input/Output. Vstupní/Výstupní
ADC	Analog to digital converter. Analogově digitální převodník.
PWM	Pulse Width Modulation. Pulsně šířková modulace.
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
SPI	Serial Peripheral Interface. Sériové rozhraní.
USART	Universal Synchronous/Asynchronous Receiver/Transmitter. Sériové rozhraní.
PLL	Phase-Locked Loop. Fázový závěs.
DSP	Digital Signal Processing.
XS1	Řada a název vstupně výstupních portů v architektuře xCORE.
SERDES	Obvod pro serializaci nebo de-serializaci dat.
xC	Extensions to C. Rozšířený jazyk C

1

Úvod

Nejprve je stručně popsán jednoprocesorový mikrokontrolér, aby byl vidět jasný rozdíl mezi jednočipovým provedením a vícejádrovým mikrokontrolérem. Dále je popsána architektura xCORE. Následně jsou popsány XS1 porty, které jsou důležitou součástí architektury xCORE. Potom je stručně popsán hardware mikrokontroléru StartKit. Kapitola Aplikace pro XMOS zařízení obsahuje stručný náhled do problematiky tvorby aplikací a je v ní popsáno, jak využít již hotové příklady přímo v programovacím prostředí. Dále jsou v kapitole Vývoj aplikací, popsány základní konstrukce jazyka xC, které jsou důležité pro pochopení pracování celé architektury. Nakonec je stručně popsán vývoj vlastní aplikace s využitím připravených knihoven pro mikrokontrolér XMOS StartKit.

Práce vznikla za účelem seznámení se s vícejádrovou architekturou XMOS, prozkoumat a popsat její hardwarové řešení, způsob tvorby aplikací. Práce poskytuje náhled do architektury a jsou v ní popsáné základní řešení tvorby aplikací.

Obsah poskytuje pouze náhled do problematiky mikrokontrolérů xCORE a může sloužit jako stručný návod pro orientaci v této architektuře. Veškerý obsah je získán z firemních dokumentací a vlastních poznatků.

Snaha byla o vytvoření základních informací co se týče architektury xCORE a poskytnout tak jasný pohled do problematiky seřazený tak, aby dával jednoduchý návod k této architektuře a snazší orientaci ve firemní dokumentaci XMOS, která je na první pohled dosti nepřehledná.

2

Jednojádrový mikrokontrolér

Jednojádrový mikrokontrolér je dnes velice rozšířené zařízení. Vyskytuje se v mnoha podobách a provedení. Probereme si základní vlastnosti jednočipového provedení. Jako příklad vezmeme mikrořadič Nucleo STMf411RE.

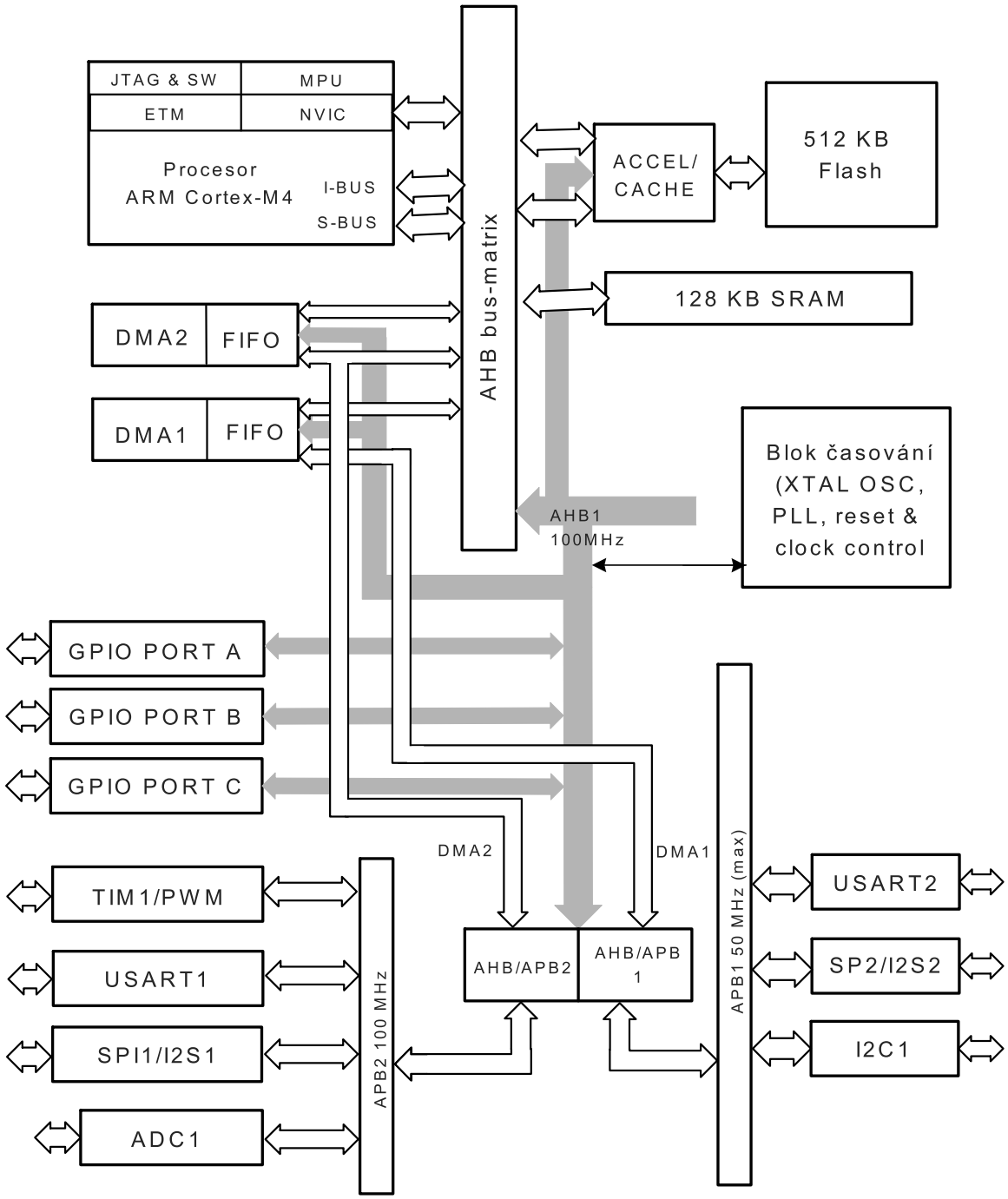
2.1 Mikrokontrolér Nucleo STMf411RE

Nucleo STMf411RE vyrábí společnost STMicroelectronic, která je jednou z předních výrobců mikroprocesorové techniky. Veškeré informace o STM lze naleznout na internetových stránkách www.st.com.

2.1.1 Popis a blokové schéma mikrokontroléru

Základ mikrokontroléru je 32 bitový procesor ARM Cortex-M4, který přes soustavu sběrnic řídí periferní obvody. Procesor čte instrukce z vyrovnávací paměti CACHE, do které jsou data dodávána z paměti FLASH, také může číst a zapisovat data do paměti SRAM. Ovládá I/O porty, z kterých zpracovává data. Jádro procesoru má zabudovaný časovač Systick, který je připojen přímo na systémový zdroj hodin, lze jej použít pro obecné časování. Nastavení a řízení obsluhy přerušení realizuje blok NVIC, který je taktéž součástí jádra ARM. Maximální taktovací frekvence procesoru je 100 MHz, avšak mikrokontrolér disponuje děličkami kmitočtu a lze tedy provozovat i nižší frekvence.

GPIO porty mohou pracovat v několika režimech. Buď jako vstupní, nebo výstupní digitální brány, vstup ADC převodníku, nebo mohou pracovat v alternativním režimu například jako výstup PWM. Mikrokontrolér má i další periferní obvody, které lze různě programově nastavovat a řídit. Časovače, které lze použít v různých režimech, obvody I2C, I2S, SPI, USART. Také je možné k mikroprocesoru připojit externí paměti, a další periferní obvody. Některé periferní obvody mohou pracovat i při zastavení hlavní nekonečné smyčky while. V programu se provedou potřebná nastavení a jednotka pak pracuje nezávisle. Jádro může vykonávat jiné úlohy a různé události se mohou vykonávat v přerušení. Zjednodušené blokové schéma je na obrázku 2.1.



Obr. 2.1: Zjednodušený blokový diagram mikrokontroléru Nucleo STMf411RE: Základní části mikrokontroléru (DMA – přímý přístup do paměťi; ACCEL/CACHE – zrychlená vyrovnávací paměť; GPIOx – vstupně výstupní porty; TIMx – časovač; ADC – analogově digitální převodník; AHB/APB – sběrnice;SRAM – paměť; USART,SPI,I2C,I2S – periferní obvody pro komunikaci; FIFO – vyrovnávací paměť)

2.1.2 Časování a používané registry

Základní hodinový takt vychází z High-Speed zdrojů. Vnitřní HSI má základní takt 16 MHz, který se nastavuje vždy po restartování. Pokud se program snaží přepnout na vnější krystal HSE a ten není k dispozici, zůstává HSI a nastaví se příznak chyby. HSE má takt 4 až 26 MHz. Z HSI či HSE se pak přímo nebo přes blok PLL získává SystemCoreClock, což je základní takt. Blok PLL slouží k násobení nebo dělení základního taktu téměř kterýmkoliv číslem. To umožňuje jemně ladit výsledný takt. Ze SystemCoreClock se získává základní takt pro periferní obvody přes sběrnice AHB1, AHB2 a APB1, APB2. Každá sběrnice má vlastní děličku kmitočtu pro nastavení požadované taktovací frekvence pro periferní obvody.

Mikroprocesor využívá několik desítek, až stovek registrů. Blok RCC registrů slouží pro povolení a restartování hodin periferního obvodu. Pro GPIO porty jsou registry MODER, OTYPER, OSPEEDR, ODR. MODER nastavuje režim, OTYPER typ výstupu, OSPEEDR rychlost hrany a ODR je řídící registr pro hodnotu výstupu GPIO. Každý periferní obvod má různé registry. Převážně jsou to registry pro nastavení různých režimů periferního obvodu.

2.1.3 Psaní kódu pro Nucleo

Nucleo STM mikrokontroléry se mohou programovat ve vývojovém prostředí Atollic TrueSTUDIO for ARM. Programovací jazyk, ve kterém lze psát aplikace je C/C++. Při psaní programu, prostředí napovídá možné využitelné části kódu. Jsou zde vytvořeny struktury a ukazatele, které lze využít. To velmi zjednodušuje vytváření aplikací. Důležitá dokumentace je Datasheet, Reference manual a Programming manual. Ukázka kódu pro blikání led diodou, která je připojena na GPIOA port 5 je na obrázku.2.2.

```
1     #include "stm32f4xx.h"                          //vložení potřebného hlavičkového souboru
2
3     int main(void)                                  // hlavní funkce main
4     {
5       int i = 0;                                    // pomocná proměná
6       if (!(RCC->AHB1ENR & RCC_AHB1ENR_GPIOAEN))    // pokud nejsou povoleny hodiny periferie
7        {
8           RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;      // povolit hodiny periferie
9           RCC->AHB1RSTR |= RCC_AHB1RSTR_GPIOARST;   // reset periferie
10          RCC->AHB1ENR &= ~RCC_AHB1RSTR_GPIOARST;   // konec resetu periferie
11       }
12          GPIOA->MODER |= 0x01 << (5 * 2);          //nastavení režimu jako výstup
13          GPIOA->OTYPER &= ~(1 << 5);               // nastavení typu výstupu push-pull
14          GPIOA->OSPEEDR |= 0x03 << (5 * 2);        // nastavení rychlosti hrany na High-Speed
15      while (1)                                     // nekonečná smyčka while
16      {
17          GPIOA->ODR ^= 1 << 5;                     // střídaní hodnoty výstupu operací XOR
18          for(i = 0; i < 500000; i++);              // čekací cyklus
19      }
20    }
```

Obr. 2.2: Kód pro blikání LED diodou.

3

Vícejádrový mikrokontrolér xCORE

XCORE mikrokontroléry vyrábí společnost XMOS. Veškerá důležitá dokumentace lze naleznout na internetových stránkách www.xmos.com.

3.1 O mikrokontroléru xCORE

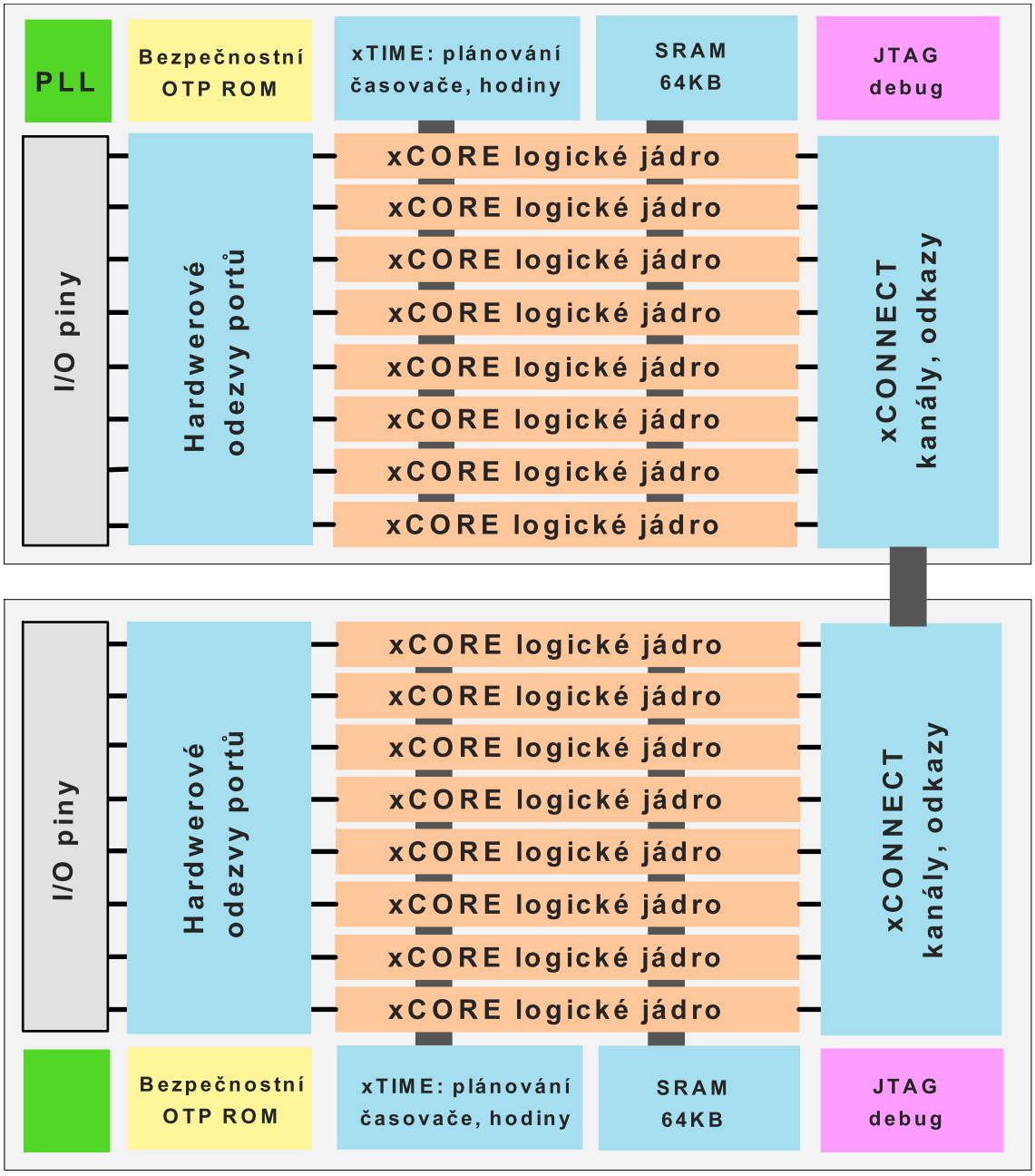
xCORE představuje novou třídu mikrokontrolerů, která se skládá z více procesorových jader, velice flexibilních vstupně-výstupních pinů a unikátní deterministické architektury. Díky těmto vlastnostem lze xCORE velmi snadno použít k vytváření vlastních elektronických vestavěných systému téměř kteréhokoliv typu.Díky unikátnímu časovaní xTIME lze provozovat více úloh v reálném čase. Na každém jádře může probíhat jiná úloha a mohou mezi sebou komunikovat přes xCONNECT. Mikrokontroléry podporují programovací jazyky C/C++, avšak hlavní výhodou je podpora jazyku xC, který firma XMOS vytvořila speciálně pro xCORE. Díky implementaci xC lze velice snadno programovat I/O porty, časování a především komunikaci mezi jádry procesoru. Periferie lze definovat softwarově s využitím xSOFTip. Přímo ve vývojovém prostředí lze přes internet, stáhnout bloky kódu v xC, které můžete použít ve své vlastní aplikaci.

3.1.1 Architektura xCORE

XCORE vícejádrové mikrokontroléry se vyrábí v několika provedení. Základ je více výpočetních 32 bitových jader, která spolu mohou komunikovat přes xCONNECT. Jádra mají společně rozvedené časování xTIME. Pomocí xTIME lze přiřadit potřebnou funkčnost portům. Porty mohou být ovládány kterýmkoliv jádrem procesoru. Jádra mají společnou paměť SRAM a zabezpečovací OTP ROM. Programování probíhá přes rozhraní JTAG debug.

3.1.1.1 DSP, bezpečnost, časování

Procesor xCORE poskytuje obsáhlý rozsah nativních DSP operátorů a instrukcí. To umožňuje integrovat vysokorychlostní digitální zpracování signálů pro audio signál, zpracování



Obr. 3.1: Architektura xCORE mikrokontrolérů

videa, transformace signálu v průmyslu a řízení.

Zabezpečovací blok v xCORE, který pracuje společně s logickými jádry procesoru, umožňuje nastavení a implementování bezpečnostních algoritmů, aby byla zajištěna bezpečnost vámi psaného kódu.

Deterministická architektura procesorů xCORE umožňuje extrémně nízkou latenci a odezvu I/O portů, která je až 100 krát menší než u běžných procesorů.

3.1.2 Jak xCORE funguje

XCORE je 32 bitový RISC procesor, který je složen v bloku. Každý blok má více logických procesorových jader xCORE. V řadě XS1 má každý blok až 8 jader. Zařízení jsou k dispozici s 1, 2 nebo čtyřmi bloky, kde může být až 32 logických jader. XCORE obsahuje podporu pro dlouhou aritmetiku, spoustu dalších DSP operátorů a možnosti zabezpečení kódu. Každý xCORE blok má jednotný paměťový systém, který sdílí všechna jádra v bloku jak pro program, tak i data. Vícenásobná jádra umí sdílet stejný program v paměti a mohou procházet vlastněná data mezi nimi. Procesor xCORE nepoužívá vyrovnávací paměť CACHE, aby vykonávání programu bylo kompletně časově deterministické. Pokud jádro čeká na data, xTIME hardwarový plánovač předá zdroj výkonu dalšímu jádru, což umožňuje efektivní využívání a úsporu energie.

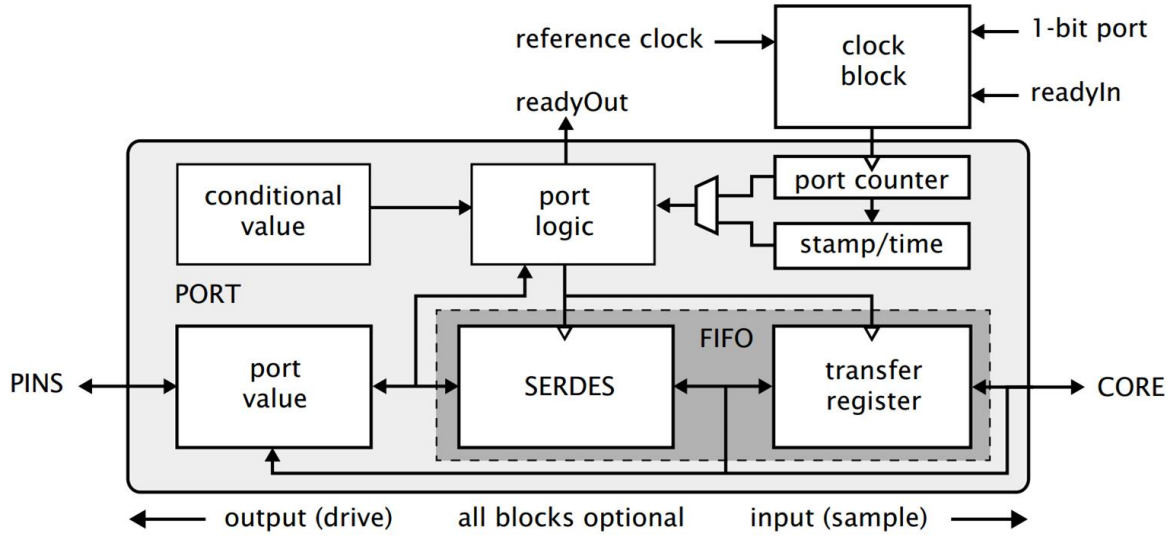
3.1.2.1 Vstupně-výstupní a logické porty

Programovatelné I/O porty dávají xCORE jedinečné flexibilní vlastnosti. Architektura má umístěné řízení portů v těsné blízkosti jader. Data jsou přenášena přímo mezi logickým jádrem, registry a I/O porty. Tím se vyhne využívání paměti a eliminuje se latence. Porty umí sériová data převést na paralelní a naopak. To umožňuje procesoru vysokorychlostní zpracovávání datových proudů. Umí kontrolovat příchod dat, přesně řídit časování a která data se mají poslat, nebo přijmout z pinu. Tato úroveň kontroly vysokého výkonu umožňuje komplexní I/O periferie, které jsou implementovány a sestavovány pomocí softwaru. XMOS poskytuje kompletní řadu softwarových periferních funkcí nazývané xSOFTip.

4

XS1 porty

Port vytváří spojení mezi blokem xCORE a jedním nebo více fyzickými piny. Definuje rozhraní mezi hardwarem připojeným k xCORE mikrokontroléru a softwarem běžícím na zařízení. Logický port umí nastavovat pinům hodnoty high nebo low, může ukázat hodnotu na pinech a volitelně čekat na konkretní stav. porty jsou přístupné pomocí vyhrazených funkcí. Data jsou přenášena mezi piny a jádrem pomocí paměti FIFO, která obsahuje SERDES a přenosový(transfer) registr. To poskytuje možnost serializovaných a bufferovaných dat. Blokové schéma XS1 portu je na obrázku 4.2.



Obr. 4.1: Blokový diagram XS1 portu. (Převzato z Introduction to XS1 ports.pdf)

4.1 Čítač - (Port counter)

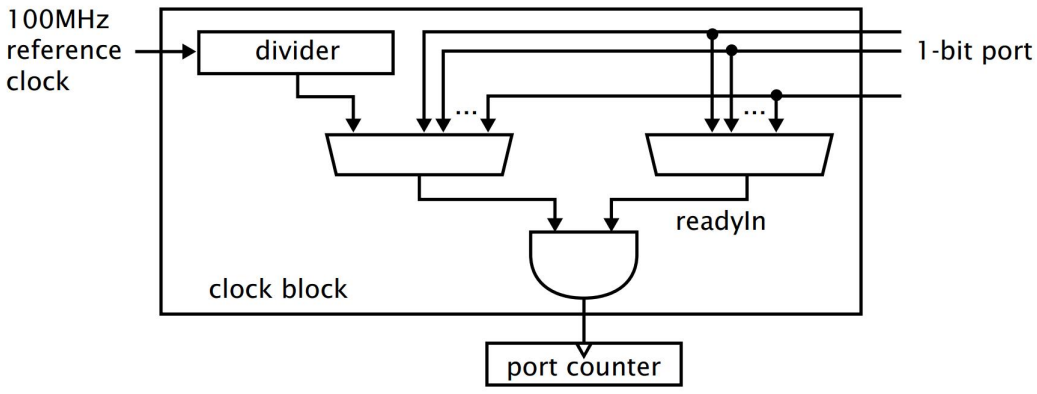
Každý port má čítač(counter), lze ho použít ke kontrolování času, ve kterém jsou data přesouvána mezi vyznačeným portem a přenosovým registrem. Počítadlo je 16-ti bitové a za každý cyklus hodin, s klesající hranou signálu inkrementuje svou hodnotu. Hodnoty

čítače lze vyčíst a zjistit, zda byla data obdržena, nebo pozdržet I/O až do určitého času v budoucnu. Hodnota counteru se automaticky ukládá do timestamp registru, jakmile jsou data přesouvána z, nebo do transfer registru. Tato hodnota je k dispozici v aplikaci, přičemž poskytuje přesnou kontrolu doby odezvy. Jako port counter v aplikaci s hodiny, pouze měří čas na vstupu hodin, pokud má signál sestupnou hranu v pravidelných intervalech. Pokud jsou hodiny nepravidelné port counter není měřítkem času.

4.2 Blok hodin, strobe signal

Mnoho I/O operací vyžaduje data, která mají být vzorkována a řízena při specifických hranách hodin. XCORE zařízení zahrnují sadu programovatelných hodin(clock block), které lze využít k řízení rychlosti s jakou mají porty pracovat. Každý xCORE blok má šest bloků hodin. XS1˙CLKBLK˙REF jsou referenční hodiny hlavního bloku, které běží na výchozí frekvenci 100 MHz. XS1˙CLKBLK˙1 až 5 lze nastavit na různé frekvence. Blok hodin může využívat jednobitový port jako zdroj hodin, umožňující externí aplikaci časování vstupních a výstupních rozhraní. Blok může vzít referenční hodiny a rozdělit hlavní frekvenci osmibitovým děličem(divider).

V mnoha případech jsou I/O signály doprovázeny strobe signálem. Porty xCORE mohou vkládat a interpretovat strobe signál(známé jako readyIn a readyOut) generovaný externími zdroji. Porty mohou generovat strobe signál, který doprovází výstupní data.



Obr. 4.2: Blokový diagram časování. (Převzato z Introduction to XS1 ports.pdf)

4.3 Možné režimy a nastavení portu

4.3.1 Nebufferovaný přenos dat

Pokud je port používán v režimu bez vyrovnávací paměti, signalizační operace na pinu je synchronizována s prováděním instrukcí v jádře. Máme několik způsobů použití.

4.3.1.1 Časové porty (Clocked ports)

Definice následující: Výstup způsobuje vedení dat s následující sestupnou hranou hodin portu. Výstup dat se mění synchronně s hodinami portu. Vstup vrací data z předcházející hrany hodin portu a je vzorkován synchronně s hodinami portu.

4.3.1.2 Časované porty (Timed ports)

Je-li vstupní nebo výstupní port nakonfigurován tak, aby čekal na port counter, port čeká dokud čítač nedosáhne požadované hodnoty a poté data přesune do transfer registru. Pokud je port použit pro výstupní operaci a transfer registr je plný, přenos je pozdržen dokud se registr neuvolní. Čas portu se nemění dokud nejsou vyřízena čekající data na výstupu.

4.3.1.3 Porty s časovým označením (Timestamped ports)

Hodnota v registru port counteru, při kterém jsou data přenášena z nebo do transfer registru, je zachytávána v timestamp registru.

4.3.1.4 Podmíněný vstup (Conditional input)

Podmínka lze nastavit na vstupní port, což způsobí že port počká až bude podmínka splněna. Po splnění zadané podmínky se port chová jako časovaný vstup. Podmínky zahrnují:

- piny schodné - (pinseq): hodnota pinů musí odpovídat zadané hodnotě
- piny neschodné - (pinsneq): hodnota na pinech nesmí odpovídat zadané hodnotě

Pokud je nastavena podmínka, port porovnává hodnotu na pinech se zadanou hodnotou(conditional value). Je-li podmínka splněna je nastavena časová značka(timestamp) a port je připraven pro vstup. Je možno použít časovaní s podmínkou, kde port čeká na zadanou hodnotu port counteru a splnění podmínky, předtím než data budou vstupovat. Podmínku nelze využít pro výstupní port.

4.3.2 Bufferovaný přenos dat

Porty mohou být pozdrženy ve vyrovnávací paměti FIFO dokud jádro není připraveno pro vstupní nebo výstupní data. To dovoluje jádru vykonávat jiné instrukce během této doby. Využitím FIFO, může jediné jádro provádět I/O operace na více pinech paralelně. Porty používané jako blok hodin a strobe signál nemohou být ukládány do vyrovnávací paměti.

4.3.2.1 Bufferovaný vstup

Definice: Z každou náběžnou hranou se data přesouvají z FIFO do transfer registru. Počínaje nejméně významným bitem LSB(nibblem, bytem, nebo 16-ti bitovou entitou). Pokud je transfer registr plný, data jsou vyřazena, aby se uvolnilo místo pro naposledy vzorkovanou hodnotu. Čas v časovém vstupu(timed input) představuje čas v budoucnosti, kdy se spustí data na vstup, to způsobí, že jádro odmítne všechna data z vyrovnávací paměti před provedením vstupu. Podmíněný vstup(A conditional input) přesune data do přenosového registru při náběžné hraně hodin, když je podmínka splněna. Poté tento stav vymaže. Hodnota čítače(port counteru), když jsou data přesouvána z transfer registru, je zaznamenávána v timestamp registru. Je-li několik bufferovaných vstupních portů poháněno ze stejných hodin, mohou představovat jeden vstupní port, za předpokladu, že jádro je schopno zpracovat data ze všech portů, během každého hodinového cyklu.

4.3.2.2 Bufferovaný výstup

Definice: Port přenáší data z transfer registru při každé sestupné hraně hodin. Jádro je blokováno pokud se pokusí o výstup, zatímco je transfer registr plný. Pokud se FIFO vyprázdní, výstupní port bude pokračovat v přenášení své poslední hodnoty. Časový výstup(timed output) způsobí, že port počká až do zadaného času a potom se přenesou data z transfer registru do paměti FIFO. Hodnota čítače(port counter) je zaznamenávána v timestamp registru, když jsou data přesouvána z transfer registru. Podmíněný(conditional) bufferovaný výstup není k dispozici. Je-li několik bufferovaných výstupních portů poháněno ze stejného bloku hodin, mohou se zdát, že fungují jako jediný výstupní port. Za předpokladu, že jádro je schopno dodávat nová data všem portům s každým hodinovým cyklem.

4.3.3 Ostatní režimy

Více o režimech portů lze naleznout v dokumentu Introduction to XS1 port.pdf.

4.3.3.1 Serializovaný přenos dat a Strobing

SERDES lze použít k serializaci (výstupu) nebo de-serializaci (vstupních) dat, což redukuje počet instrukcí požadovaných pro vstupní či výstupní data. Počet bitů v transfer registru a SERDES určují šířku převodů mezi jádrem a portem. Je to násobek šířky portu (počet pinů).

Architektura xCORE poskytuje podporu pro strobing, kde jsou data doprovázena samostatným platným signálem. Až dva piny lze použít ke strobe datům(readyIn a readyOut signály) poskytující čtyři strobe módy.

4.3.3.2 Obousměrný mód (Bidirectional ports)

Port může být naprogramován v obousměrném režimu. V obousměrném režimu není k dispozici žádná vyrovnávací paměť.

4.3.4 Hardware port pin-out

Každý xCORE blok(tile) obnáší kombinaci 1, 2, 4, 8, 16, 32 bitových portů, uspořádání závisí na zvoleném zařízení. Celkový počet možných GPIO portů daleko převyšuje počet pinů skutečně propojených na desce plošného spoje. Proto jsou porty a xCONNECT odkazy(Links) multiplexovány a jsou definované precedence pro překrývání portů a existujících odkazů(Links). V případech kdy je k dispozici pouze několik pinů z širokého portu, lze je požít jako GPIO piny, ale další základní šířka předepsaných portů není k dispozici.

4.3.4.1 priorita portu

Mapování portů na piny je znázorněno na obr. 4.3. Tabulka obsahuje seznam některých pinů, ke kterým lze připojit porty. Odkazy(Links) a porty na levé straně mají přednost před porty na pravé straně. Každý port je označen jeho šířkou a písmenem, které rozlišuje více portů stejné šířky. Bity portu jsou označeny čísly 0 až 31. Odkazy(Links) jsou identifikovány jedním písmenem A-D. Dráty spojení jsou identifikovány pomocí nadpisu číslice 0-4. Port nebo xCONNECT odkaz(link), který je skutečně připojen k pinu je určen pomocí softwaru. Pokud je povolen odkaz(link), má tento odkaz přístup k pinům, kolíky všech podřízených portů jsou zakázány. Je-li povolen port, přepíše všechny porty s vyššími šířkami, které sdílejí jeho piny.

4.3.5 Identifikace portů, konfigurace portů a hodin

Každý port je architektonicky reprezentován identifikátorem, nazývaným zdrojem identifikace(resource identifier). Hlavičkový soubor xs1.h obsahuje všechna mapování. Příklad mapování je ukázán na obr. 4.4.

Porty musí být pojmenovány xC identifikátorem a deklarovány, zda mají být data sériová nebo bufferovaná, předtím než je lze použít v kódu pro I/O operace. Funkce pro konfiguraci portů jsou poskytovány v hlavičkovém souboru xs1.h. Bloky hodin lze nakonfigurovat, aby generovaly hodiny na základě referenčních hodin, nebo lze použít vstupní pin jako zdroj hodin. Když jsou nakonfigurovány hodiny, musí se explicitně spustit v aplikaci. Pokud se spustí hodiny pro všechny porty stejné, mají porty stejnou hodnotu čítače(counteru). Funkce pro konfiguraci hodin jsou uvedeny v hlavičkovém souboru xs1.h. Porty mají několik režimů, které musí být nastaveny, než-li je port nakonfigurován. Funkce pro konfiguraci režimů portů jsou uvedeny v xs1.h. Operace s porty lze řídit sadou definovaných funkcí v xs1.h.

	⇐ Nejvyšší		Priorita		Nejnižší ⇒	
Pin	link	1-bit ports	4-bit ports	8-bit ports	16-bit ports	32-bit port
XnD00		$1A$				
XnD01	A^4 out	$1B$				
XnD02	A^3 out		$4A^0$	$8A^0$	$16A^0$	$32A^{20}$
XnD03	A^2 out		$4A^1$	$8A^1$	$16A^1$	$32A^{21}$
XnD04	A^1 out		$4B^0$	$8A^2$	$16A^2$	$32A^{22}$
XnD05	A^0 out		$4B^1$	$8A^3$	$16A^3$	$32A^{23}$
XnD06	A^0 in		$4B^2$	$8A^4$	$16A^4$	$32A^{24}$
XnD07	A^1 in		$4B^3$	$8A^5$	$16A^5$	$32A^{25}$
XnD08	A^2 in		$4A^2$	$8A^6$	$16A^6$	$32A^{26}$
XnD09	A^3 in		$4A^3$	$8A^7$	$16A^7$	$32A^{27}$
XnD10	A^4 in	$1C$				
XnD11		$1D$				
XnD12		$1E$				
XnD13	B^4 out	$1F$				
XnD14	B^3 out		$4C^0$	$8B^0$	$16A^8$	$32A^{28}$
XnD15	B^2 out		$4C^1$	$8B^1$	$16A^9$	$32A^{29}$
XnD16	B^1 out		$4D^0$	$8B^2$	$16A^{10}$	
XnD17	B^0 out		$4D^1$	$8B^3$	$16A^{11}$	
XnD18	B^0 in		$4D^2$	$8B^4$	$16A^{12}$	
XnD19	B^1 in		$4D^3$	$8B^5$	$16A^{13}$	
XnD20	B^2 in		$4C^2$	$8B^6$	$16A^{14}$	$32A^{30}$
XnD21	B^3 in		$4C^3$	$8B^7$	$16A^{15}$	$32A^{31}$
XnD22	B^4 in	$1G$				
XnD23		$1H$				

Obr. 4.3: Ukázka mapovaní portů. (Převzato z Introduction to XS1 ports.pdf)

Port name	Resource identifier
XS1_PORT_32A	0x200000
XS1_PORT_16A	0x100000
XS1_PORT_16B	0x100100
XS1_PORT_8A	0x80000
XS1_PORT_8B	0x80100
XS1_PORT_8C	0x80200
XS1_PORT_8D	0x80300
XS1_PORT_4A	0x40000
XS1_PORT_4B	0x40100
XS1_PORT_4C	0x40200

Obr. 4.4: Ukázka identifikátorů. (Převzato z Introduction to XS1 ports.pdf)

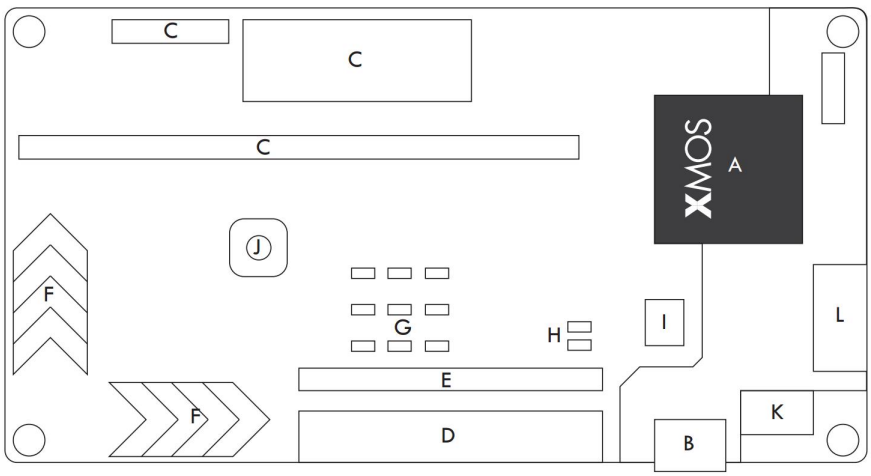
5

XMOS StartKit

XMOS StartKit je konfigurovatelná vývojová deska s xCORE vícejádrovým mikroprocesorem. Umožňuje konfigurovat software podle potřeby a poskytuje mnoho pokročilých funkcí. Mikrokontrolér obsahuje osm 32 bitových logických jader, které lze programovat. Startkit je ideální platformou pro různé účely od robotiky a ovládání pohybu, až po digitální audio.

5.1 Blokové schéma rozmístění komponentů mikrokontroléru

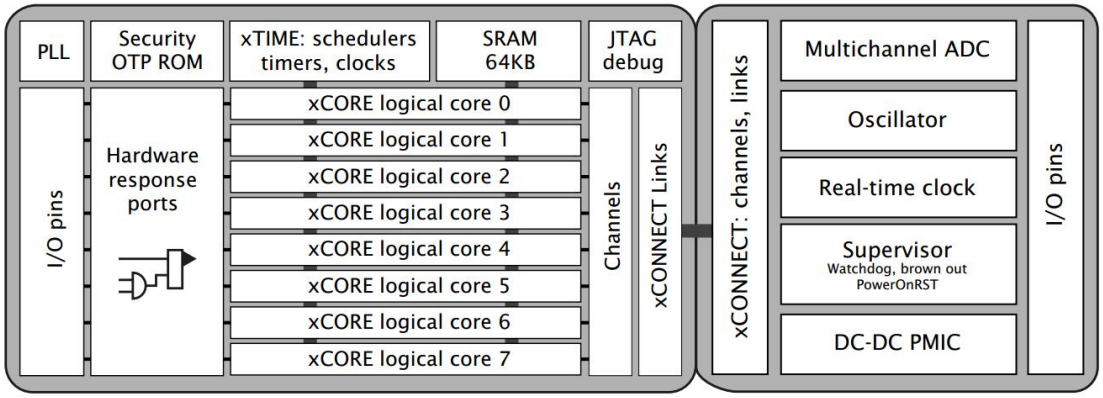
Na obrázku je vidět přesné rozmístění komponentů mikrokontroléru.



Obr. 5.1: Blokový diagram StartKitu: A – xCORE mikrokontrolér s integrovaným debugger ; B – Micro USB konektor pro debugger/JTAG ; C – PCIe slot pro sliceCARD nebo 1x24 GPIO ; D – 2x13 GPIO kompatibilní s Raspberry Pi ; E – 1x13 header poskytující dva XMOS Links ; F – dvakrát čtyři dotykové zóny ; G – 3x3 zelené LED ; H – dvě zelené LED ; I – SPI Flash ; J – tlačítko ; K – analogové vstupy ; L – 24 MHz Oscilátor (převzato - XMOS StartKit.pdf)

5.2 Popis mikrokontroléru

StartKit je založen na dvou blocích(tile) xCORE zařízení(xCORE-Analog A8-DEV). Blok 0 je věnován integrovanému ladění(debugger) a USB PHY. Blok 1 je programovatelný a poskytuje 8 logických jader s výpočetní rychlostí 500 MIPS. Všechny digitální I/O v bloku 1 jsou vyvedeny na piny, což umožňuje velkou kombinaci periferií, které mají být implementovány do desky StartKit. Zařízení xCORE-Analog A8-DEV je dostupné pouze jako součást StartKitu a není proto samostatně dokumentované. Pokud se má StartKit použít pro komerční účely, může pomoci XS1-A8A-64-FB96 Datasheet. Nebo pro použití StartKitu jako cílové platformy, může být užitečný XS1-U16A-128-FB217 Datasheet. Blokové schéma zařízení, které je možné programovat je na 5.2.



Obr. 5.2: Blokový diagram XMOS zařízení (xCORE). (Převzato z XS1-A8A-64-FB96 Datasheet.pdf)

5.2.1 Integrovaný debugger

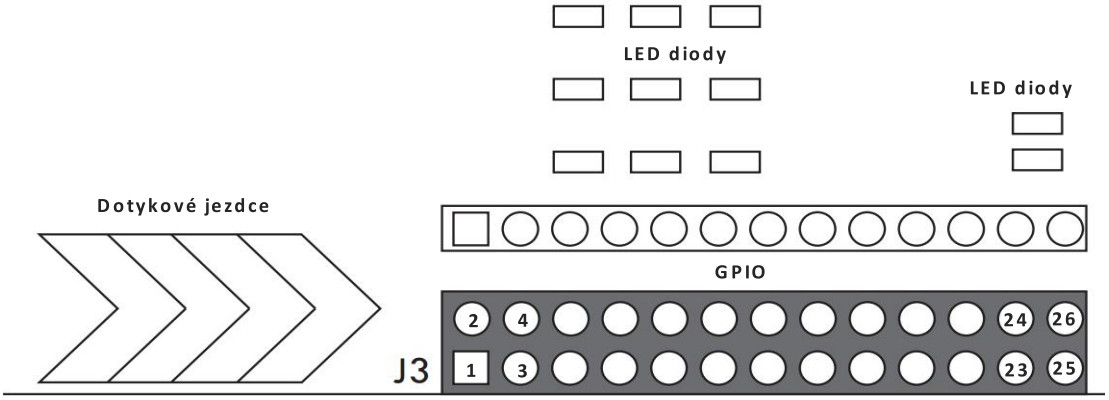
Integrovaný debugger a související součásti jsou umístěny na jednom konci desky. Debugger je přístupný pomocí micro-USB připojeného k hostitelskému počítači. To umožňuje nástrojům xTIMEcomposer ladit aplikaci běžící na zařízení.

5.2.2 Popis periferií

Periferie mají definované porty, k nim příslušející piny a umístění na desce.

5.2.2.1 GPIO

Na desce je 2x13 pinů GPIO připojených na kombinaci 1-bitových a 32-bitových portů. Porty jsou kompatibilní s Raspberry Pi, nebo mohou být alternativně použity jako uživatelem konfigurovatelné GPIO. Pokud se použije připojení Raspberry Pi, nejsou k dispozici LED diody a tlačítko. Připojení je vidět na obrázku 5.3.

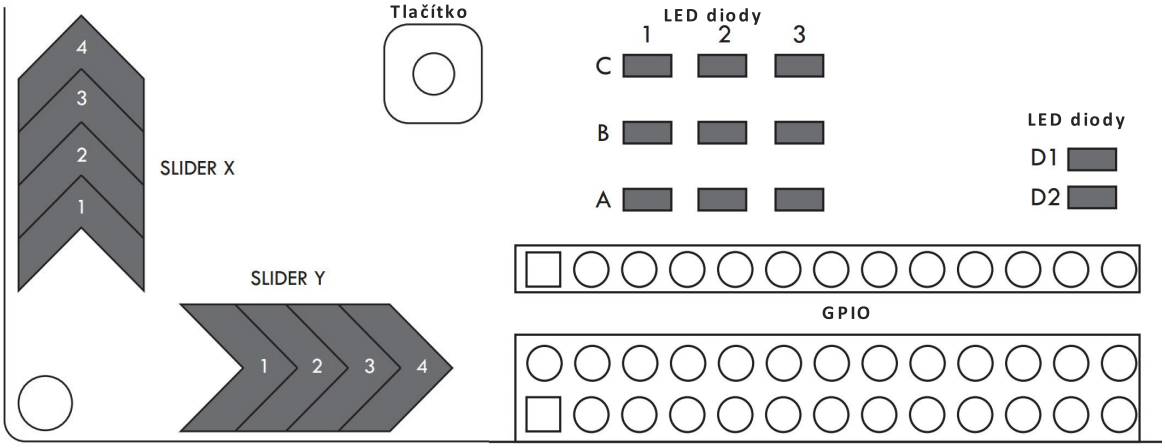


Port	Pin	Header IO		Pin	Port
	NC	1	2	NC	
P32A0	X0D49	3	4	NC	
P32A19	X0D70	5	6	GND	
P32A18	X0D69	7	8	X0D68	P32A17
	GND	9	10	X0D63	P32A12
P32A10	X0D61	11	12	X0D62	P32A11
P32A9	X0D58	13	14	GND	
P32A8	X0D57	15	16	X0D56	P32A7
	NC	17	18	NC	
P1A0	X0D0	19	20	GND	
P1D0	X0D11	21	22	NC	
P1C0	X0D10	23	24	X0D51	P32A2
	GND	25	26	X0D50	P32A1

Obr. 5.3: Popis portů (xCORE) StartKitu. (Převzato z StartKit Architecture.pdf)

5.2.2.2 Dotykové jezdce, led diody a tlačítko

StartKit podporuje dvakrát čtyři kapacitní dotykové zóny(senzory), 3x3 zelené LED diody, dvě vedlejší LED diody a jedno tlačítko. Dotykové senzory jsou připojené na 4-bitové porty. Každá LED dioda je připojena na odlišný pin. 3x3 diody jsou mapovány na 32-bitový port a dvě vedlejší diody jsou na 1-bitovém portu. Tlačítko je mapováno na jeden bit z 32-bitového portu. Led a tlačítko jsou aktivní v logické nule. Ukázka na obrázku 5.4.



Port	Pin	Slider
P4A1	X0D2	X1
P4A2	X0D3	X2
P4A3	X0D8	X3
P4A4	X0D9	X4
P4B1	X0D4	Y1
P4B2	X0D5	Y2
P4B3	X0D6	Y3
P4B4	X0D7	Y4

Port	Pin	LED
P32A19	X0D70	A1
P32A18	X0D69	A2
P32A17	X0D68	A3
P32A12	X0D63	B1
P32A11	X0D62	B2
P32A10	X0D61	B3
P32A9	X0D58	C1
P32A8	X0D57	C2
P32A7	X0D56	C3

Port	Pin	Processor
P1A0	X0D0	LED-D1
P1D0	X0D11	LED-D2

Port	Pin	Processor
P32A0	X0D49	BUTTON

Obr. 5.4: Popis portů (xCORE) StartKitu. (Převzato z StartKit Architecture.pdf)

6

Aplikace pro XMOS zařízení

K vytváření aplikací slouží vývojové prostředí xTIMEComposer Studio(Community). Studio lze stáhnout z internetové stránky www.xmos.com. Je vhodné mít stálý přístup k internetovému připojení. Studio dokáže komunikovat s širokou podporou a velice jednoduše lze importovat již hotové příklady. To velice usnadňuje začátečníkům s touto platformou ve vývoji vlastních aplikací. Více o studiu lze naleznout v dokumentu studio.pdf.

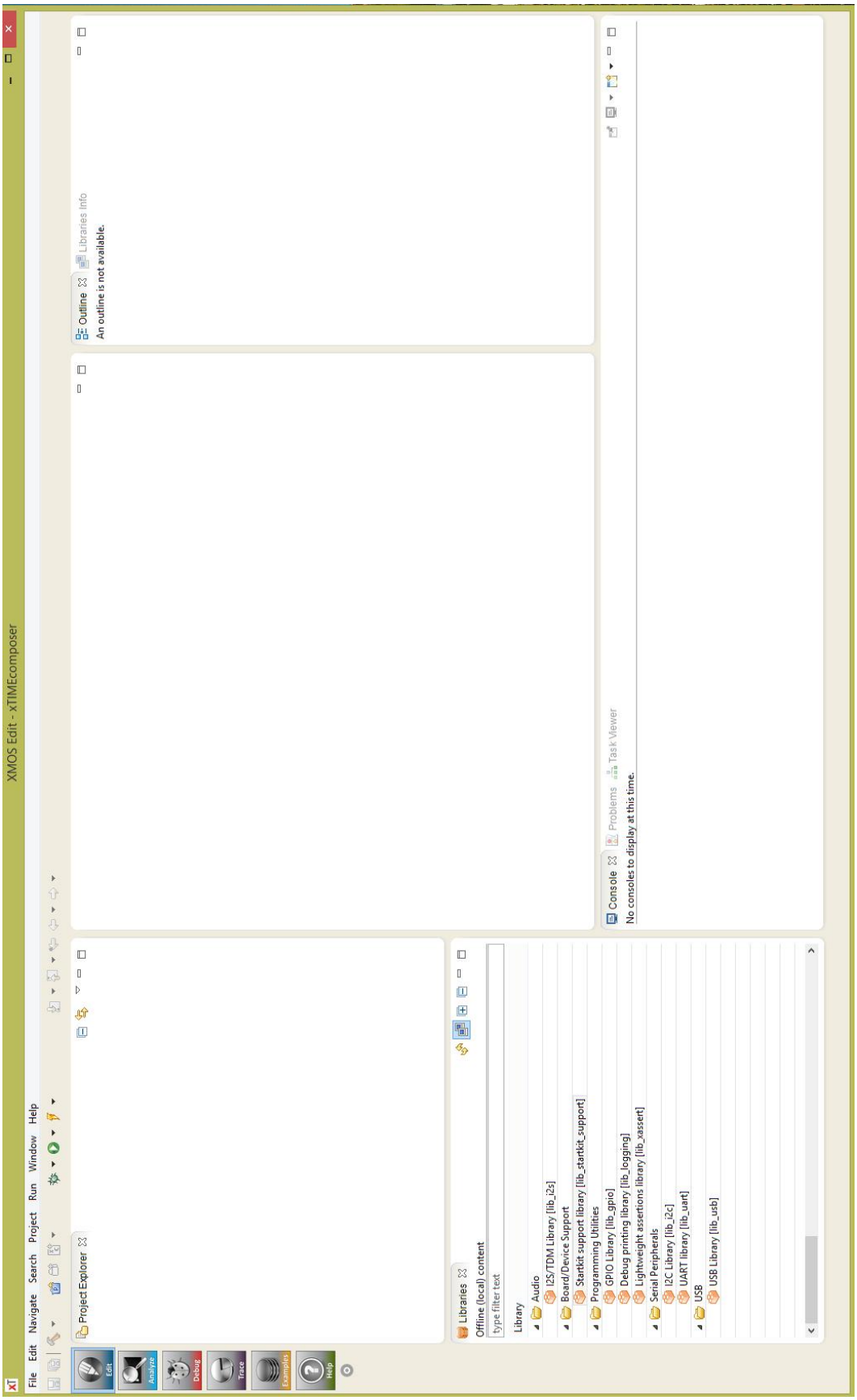
6.1 Instalace XTIMEcomposer studia a první pohled

Instalace vývojového prostředí na operačním systému Windows probíhá stejně jako většina programů .exe. Nejprve je však zapotřebí mít v počítači nainstalováno Java SE 32-bit. S 64 bitovou verzí studio ve verzi 14.2.4 nelze spustit. Se studiem se nainstaluje i dokumentace a speciální xTIME příkazový řádek. Po dokončené instalaci a prvním spuštění xTIME Composer studio je zapotřebí zvolit jméno a cílovou pracovní složku takzvaný workspace. Po vytvoření workspace se spustí studio s úvodní obrazovkou XMOS Welcome. Pro vývoj je zapotřebí kliknout na okénko Edit v levém horním rohu. Okno je vidět na obrázku 6.1

6.2 Nabídka na levé straně a důležitá tlačítka

XTIMEComposer studio má různé pohledy a přepínaní mezi nimi je na levé straně.

- Edit - hlavní okno pro psaní aplikací a orientaci v projektu
- Analyze - možnost analyzovat kód a časovaní
- Debug - debugger perspektiva pro krokování programu
- Trace - osciloskop pro sledování průběhů signálů
- Examples - příklady, které je možné vyzkoušet
- Help - podpora komunity XCore Exchange



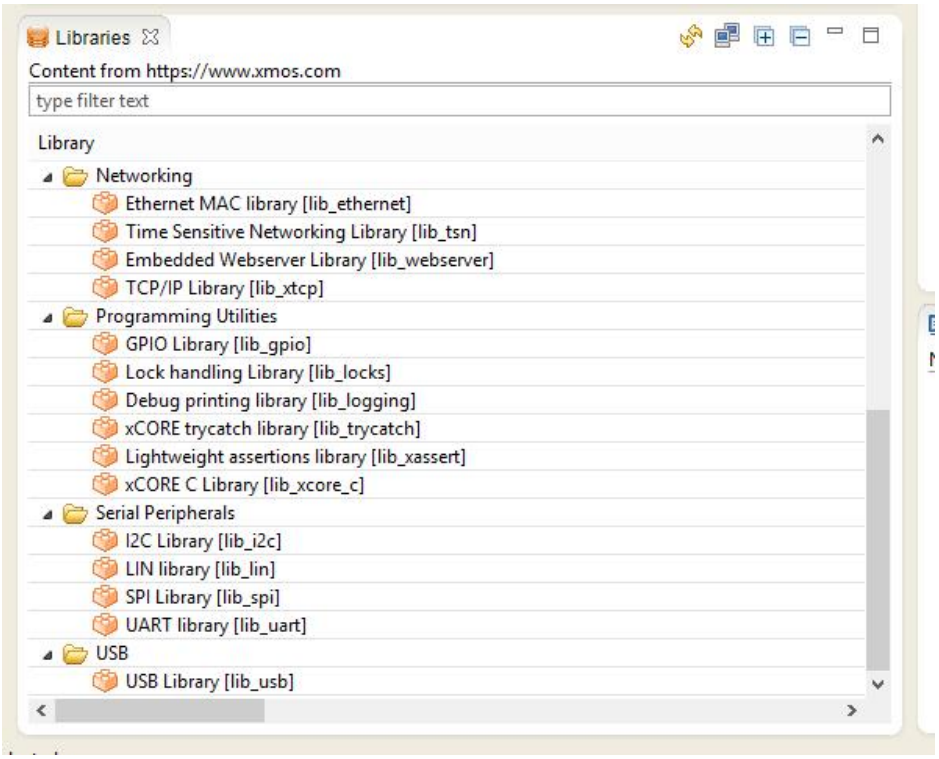
Obr. 6.1: Pohled do studia

V horní liště jsou tři tlačítka pro testování a nahrání programu do xCORE zařízení. Tlačítko Debug slouží ke spuštění aplikace v režimu ladění. Launch nahraje aplikaci do vnitřní paměti a spustí. Aplikace zůstává v mikrokontroléru. Run slouží pouze ke spuštění aplikace na zařízení bez nutnosti nahrát program do vnitřní paměti. Po odpojení a opětovném připojení USB zůstává v mikrokontroléru původní program nahraný do paměti přes Launch. Při používání těchto tlačítek je nutné orientovat se v daném projektu, který musí být nastavený jako aktuální.

6.3 Vložení ukázkových příkladů

6.3.1 Vložení knihoven

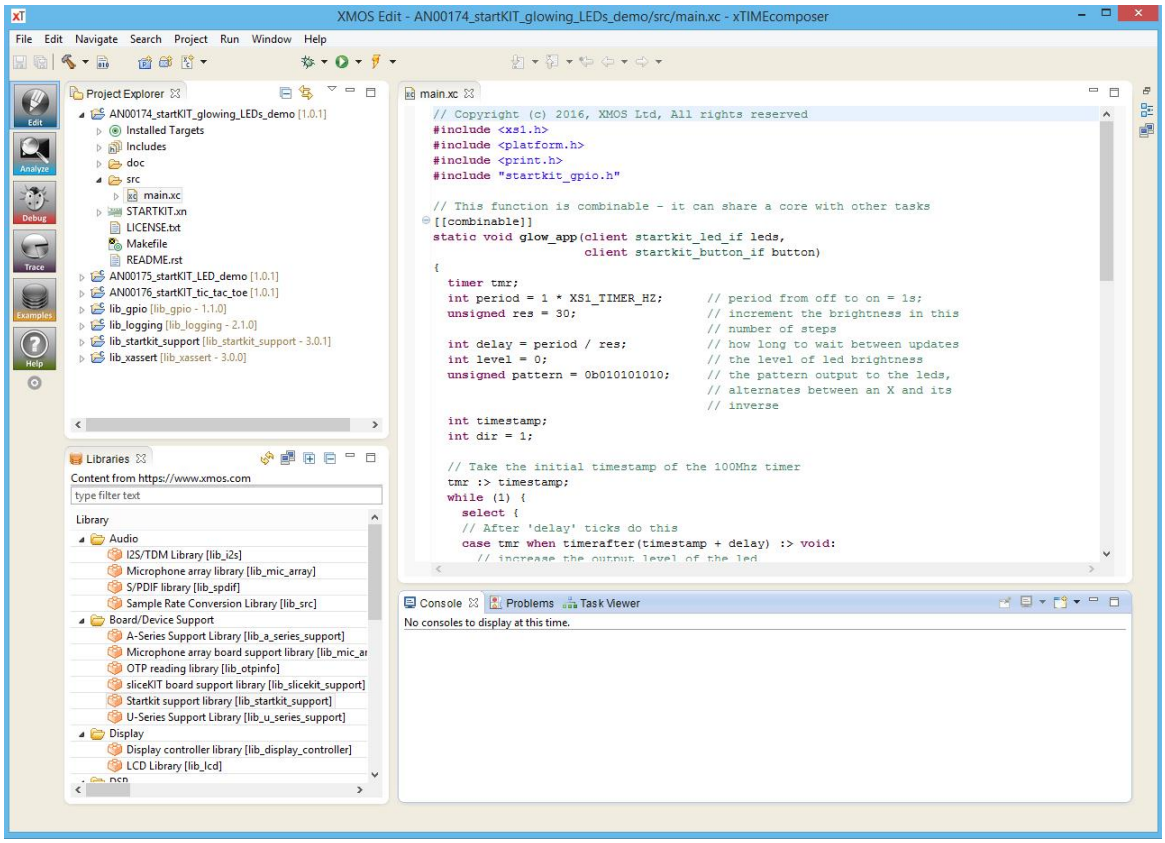
Před vložením ukázkových příkladů je nutné stáhnout knihovny, které jsou v nich používané. V režimu Edit je zobrazeno okno Libraries, ze kterého je možné stáhnout požadované knihovny. Ukázka okna na obrázku 6.2. Pro XMOS StartKit a jeho možné vzorové aplikace jsou důležité knihovny GPIO Library, Debug printing library, Lightweight assertions library a Startkit support library. Pro vložení knihoven do workspace stačí dvojklik levým tlačítkem myši na danou knihovnu a potvrzení že opravdu chceme knihovnu vložit do pracovního prostředí. Následně je možné vložit ukázkové příklady.



Obr. 6.2: Ukázka knihoven

6.3.2 Vložení hotových aplikací

Pro vložení již hotových příkladů je nutné kliknout na Examples. V pohledu Examples je možno vybrat z různých složek, pro různé platformy, několik ukázkových příkladů. Pro mikrokontrolér StartKit jsou zde tři ukázkové příklady, které lze okamžitě vyzkoušet. Je nutné je naimportovat do pracovního prostředí. V pohledu Edit, v okně Project Explorer je vidět složka programu. Nejdůležitější je složka src, ve které je schovaný kód aplikace s hlavní funkcí main. K dispozici je pár jednoduchých příkladů a jeden složitější. Ukázka na obrázku 6.3.



Obr. 6.3: Ukázka xTIMEcomposer studia s vloženými ukázkovými příklady

6.3.3 Spuštění aplikace

Nejprve je nutné připojit desku StartKitu přes microUSB do počítače a chvíli počkat než se nainstalují drivery. Kabel USB by měl být dlouhý maximálně 3 metry. Po připojení mikrokontroléru k PC, už jen stačí zvolit způsob spuštění. Může se přímo nahrát program do paměti kontroléru pomocí Launch, nebo jen spustit přes Run. Dále je nutné v následujícím okně zvolit xCORE Application. Objeví se okno Select Device, kde se zvolí XMOS startKIT, potvrdí se OK a tímto je aplikace spuštěna. Program se zastaví tlačítkem Terminate v okně Console.

7

Vývoj aplikací

Zařízení XMOS se mohou programovat v jazyce C/C++, avšak důležitým nástrojem je jazyk xC. Díky jazyku xC je programovaní portů, časovačů, a komunikace mezi jádry poměrně jednoduchá. Jazyk xC obsahuje veškeré možnosti jazyka C/C++ a spoustu vymožeností navíc. Probereme pár základních vlastností. Informace o jazyku xC lze naleznout v XMOS Programming Guide a Programming XC on XMOS.

Před psaním vlastní aplikace, je zapotřebí na začátek souboru s vlastním kódem vložit potřebné hlavičkové soubory. V hlavičkových souborech jsou deklarovány funkce, které pomáhají implementaci jazyka xC. Jsou to především hlavičkové soubory xs1.h, platform.h a timer.h. Mohou být dobře užitečné i soubory print.h nebo stdio.h.

Při vložení hlavičkového souboru stdio.h lze využívat klasické funkce jazyka C, jako například prinf() nebo scanf(). Při implementaci těchto funkcí, komunikuje mikrokontrolér s XTIME konzolí. XTIME konzole je součástí XTIMEComposer studia, ale lze jí spustit i samostatně. XTIME konzole je poměrně velikou výhodu, právě kvůli snadné implementaci funkcí z jazyka C ji lze velice snadno použít k naprogramování uživatelskému rozhraní.

7.1 Některé vlastnosti jazyka xC

Zde je několik základních vlastnosti jazyka xC. Jazyk xC je poměrně rozsáhlý a obsahuje spoustu vymožeností. Vyjmenujeme ty nejzákladnější.

7.1.1 Paralelní chod

Hlavní vymožeností jazyka xC je možnost spuštění paralelního chodu úloh pomocí příkazu par.

```
# include <stdio .h>
void hw( unsigned n) {
        printf (" Paralelni uloha cislo: %u\n", n);
}
```

```
int main () {
        par {
        hw (0) ;
        hw (1) ;
        hw (3) ;
        }
return 0;
}
```

Výše uvedený kód spustí tři paralelní úlohy. Úlohy jsou stejné a každá vypíše do konzole text "Paralelni uloha cislo: 3", dále 1 a 0. Příklad rozšíříme v dalším textu.

7.1.2 Přístup k I/O : blikání led

```
# include <platform.h>
# include <xs1.h>
# include <timer.h>

port p = XS1_PORT_1A ;

int main () {
        while (1) {
        p <: 0;
        delay_milliseconds (200) ;
        p <: 1;
        delay_milliseconds (200) ;
        }
return 0;
}
```

Ve výše uvedeném kódu jsou nejprve vloženy potřebné hlavičkové soubory. Dále je typem port definován port p, na který je připojena led dioda LED-D1. Ve smyčce while(1) je nejprve pomocí operátoru šipka vlevo a dvojtečka zapsána nula na port p. Dále se čeká 200 ms. Potom se zapíše na port p hodnota 1 a znova se čeká. Tímto je naprogramováno jednoduché blikání led diody.

Funkce delay je definovaná v souboru timer.h a dá se využít v mnoha případech. Velice zjednodušuje časovaní. Slovo port se klasicky využívá k definici portu.

7.1.3 Vytvoření více úloh paralelně

Úlohy jsou většinou spuštěny v nekonečné smyčce while a jsou definovány stejně jako funkce v jazyce C. Mohou obsahovat parametry, ale nemají návratovou hodnotu. Začínají

tedy typem void. Ukázka jednoduchého paralelního chodu dvou úloh, kde v každé úloze bliká jiná led dioda je ukázán níže.

```
out port led1_port = XS1_PORT_1A;
out port led2_port = XS1_PORT_1D;

void task1(void){ while(1) {       // úloha jedna
               delay_milliseconds(200);
               led1_port <: 1;
               delay_milliseconds(200);
               led1_port <: 0;
   }
}
void task2(void){ while(1) {       // úloha dvě
            delay_milliseconds(400);
            led2_port <: 1;
            delay_milliseconds(400);
            led2_port <: 0;
   }
}
int main () {  // hlavní funkce main
    par{           // paralelní spuštění úloh
     task1();
     task2();
       }
    return 0;
}
```

Out port znamená, že je port definován jako výstupní. Takto lze definovat i vstupní port jako in port. Ve výše uvedeném kódu chybí vložení hlavičkových souborů.

V rozsáhlejších úlohách je nejprve vytvořena inicializace periferií, časovaní, proměnných a pak následuje smyčka while.

7.1.4 Události

Ve výše uvedeném kódu je jádro zbytečně zaneprázdněno čekáním. To je poněkud neefektivní a proto je lepší využít události s pomocí časovače. Je zapotřebí vytvořit časovač a pak reagovat na události časovače pomocí příkazu select. Select má konstrukci jako switch v jazyce C. V jednotlivých větvích case se pak odehrávají události. Následující funkce je označena jako combinable. To znamená, že může sdílet jádro s dalšími úlohami označenými combinable, které zpracovávají další možné události.

```
port led1_port = XS1_PORT_1A;
port led2_port = XS1_PORT_1D;
[[combinable]]
void svit_led_task(port p, int delay_in_ms ) { // úloha pro blikání led
    timer tmr;                  // deklarace časovače
    unsigned t;                         // deklarace pomocné proměně pro čas
    const int delay_ticks = delay_in_ms * 100000;
     // převedení ms na takt hodin 100MHz
    unsigned val = 0; // pomocná proměnná pro svit led
    tmr :> t;         // přečíst počáteční hodnotu časovače a uložit do t
    while (1) {
        select {  // příkaz select, v něm se odehrávají události
        case tmr when timerafter (t + delay_ticks ) :> void :
        // při dopočítání do stanovené hodnoty provede následující kód
                p <: val ;  // zápis na led
                val = ~val;  // změna hodnoty pro led
                t += delay_ticks;
                // nastavení dalšího času při kterém nastane událost
                break ;
            }
        }
}
int main () {  // hlavní funkce
    par{                                                    // paralelně
     svit_led_task(led1_port,100); // volání úlohy
     svit_led_task(led2_port,200);
       }
    return 0;
}
```

Výše uvedený kód už tolik nezatěžuje procesor čekáním. Událost při které je změněna hodnota led diody se odehrává vždy při dopočítání do stanovené hodnoty. Příkaz select reaguje na události časovače, může ale obsahovat i další větvě case.

7.2 Komunikace mezi úlohamy

Úlohy komunikují prostřednictvím explicitních transakcí mezi nimi. Každá úloha může komunikovat s kteroukoliv jinou, bez ohledu na to na kterém zařízení a jádře běží. Kompilátor implementuje transakce co nejúčinnějším způsobem, využitím základního komunikačního hardwaru. Veškerá komunikace se provádí spojením bod k bodu(point-to-point) mezi úlohami. Tyto spojení jsou explicitně v programu.

7.2.1 Kanály a rozhraní (Channels and interfaces)

XC poskytuje tři metody komunikace mezi úlohami. Rozhraní(Interfaces), kanály(channels) a proudící kanály(streaming channels). Kanály poskytují nejjednodušší komunikaci mezi úlohami. Umožňují synchronní přenos dat bez typu mezi úlohami. Streaming channels umožňují asynchronní komunikaci mezi úlohami. Využívají libovolné vyrovnávací paměti v komunikační matici hardwaru. Množství vyrovnávací paměti závisí na hardwaru, typicky je to jedno nebo dvě slova dat. Kanály obojího typu jsou nejnižší úrovní abstrakce komunikačního hardwaru dostupného v xC, ale lze jej použít k implementaci poměrné účinné komunikace mezi jádry, avšak bez kontroly typu a nelze je použít ke komunikaci úloh na stejném jádře.

Rozhraní(Interfaces) poskytují metodu pro provádění typových transakcí mezi úlohami. To umožňuje program s více transakčními funkcemi mezi úlohami. Rozhraní mohou komunikovat mezi úlohami na stejném jádře. Také umožňují upozornění na asynchronní signalizaci mezi úlohami během jinak synchronní komunikace.

7.2.2 Připojení rozhraní

Rozhraní poskytuje nejvíce strukturovanou a flexibilní metodu propojení mezi úlohami. Rozhraní definuje typ transakcí, které mohou nastat mezi úkoly a data, která sou nimi předána. Například následující deklarace rozhraní definuje dva typy transakcí.

```
interface my_interface {
void fA( int x, int y);
void fB( float x);
};
```

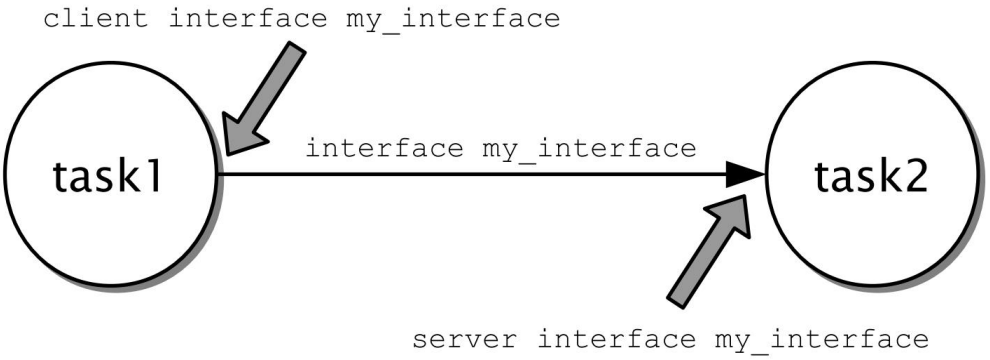
Typy transakcí jsou definovány jako funkce v C, mohou mít jakýkoliv typ argumentů. Argumenty definují jaká data budou předávána mezi úlohami. Připojení rozhraní mezi dvě úlohy se skládá ze tří částí. Připojení samotné, konec klienta a konec serveru. Možno vidět na obrázku 7.1. XC poskytuje typy ke každé části:

- připojení rozhraní(interface) je typ - interface T
- klientský konec(client end) je typ - client interface T
- koncový server(sever end) je typ - server interface T

Kde T je typ rozhraní.

Klientský konec připojení může být předán do úlohy jako parametr. Úloha která má přístup ke klientu, může iniciovat transakce pomocí následující syntaxe. Podobně jako volání funkce:

```
void task1(client interface my_interface i)
{
```



Obr. 7.1: Naznačení propojení úloh

```
// 'i' je klientský konec připojení
// komunikace s ostatními konci připojení
i.fA (5, 10) ;
}
```

Koncový server může být předán do úlohy a tato úloha může počkat na transakce požitím konstrukce select. Select čeká, dokud nebude transakce zahájena z druhé strany.

```
void task2 ( server interface my_interface i)
{
// čekání na některou z transakcí 'i '.
select {
case i.fA( int x, int y):
printf (" Received fA: %d, %d\n", x, y);
break ;
case i.fB( float x):
printf (" Received fB: %f\n", x);
break ;
}
}
```

Select může zpracovávat několik typů transakcí z několika různých zdrojů, pomocí specifických rozhraní. Úlohy můžeme propojit deklarováním instance interface a předáním jako argument pro obě úlohy.

```
int main(void)
{
interface my_interface i;
par {
task1 (i);
```

```
task2 (i);
}
return 0;
}
```

Výše uvedený kód vypíše do konzole Received fA: 5, 10

Pouze jedna úloha může využívat koncový server připojení a pouze jedna úloha může používat klientský konec. Pokud více než jedna úloha používá některý z konců, způsobí to chybu při překladu kompilátorem.

7.2.3 Kanály (Channels)

Kanály poskytují primitivní komunikaci mezi úlohami. Nedefinují typ transakce. Dvě úlohy se dají propojit přes kanál použitím deklarace chan.

```
void task1 ( chanend c) {
        c <: 5;
 }

void task2 ( chanend c) {
        select {
        case c :> int i:
        printintln (i);
        break ;
 }
}

chan c;
par {
task1 (c);
task2 (c);
}
```

S kanály se využívají speciální operátory šipka dvojtečka, které se používají pro odeslání nebo přijímaní dat. Výše uvedený kód pošle číslo 5 přes kanál.

8

Vlastní vývoj aplikací pro Startkit

Při vytváření vlastní aplikace pro StartKit, se může postupovat následujícím způsobem.

8.1 Založení nového projektu

Nový projekt se zakládá následovně. V horní liště v záložce file je nutno vybrat new a dále xTIMEComposer Project. Vyskočí okno kde je nutné zadat název projektu a vybrat platformu, která se bude používat. V našem případě vybereme hardware XMOS StartKit. Potvrdí se a projekt je založen. Bude velice užitečné, když si do svého projektu implementujeme již připravenou podporu pro StartKit, která se dá stáhnout z knihoven přímo ve studiu.

8.2 Vložení souborů do vlastního projektu

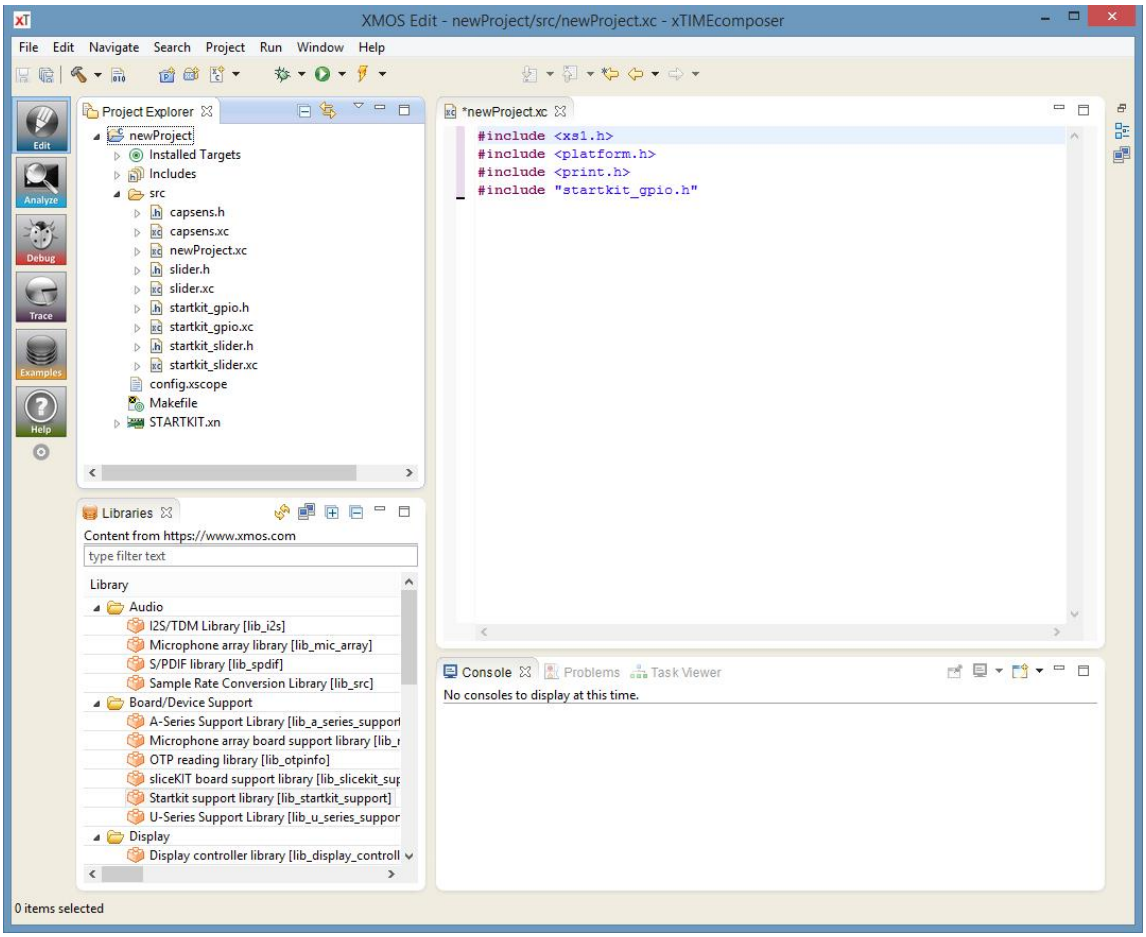
Nejprve se stáhne knihovna startkit support. Abychom mohli knihovnu v klidu využívat a zároveň nepřekážela v project Exploreru, je možné všechny zdrojové soubory zkopírovat do vlastního projektu do složky src. Potom je možné knihovnu smazat. Do souboru, kam přijde hlavní kód vložíme potřebné hlavičkové soubory. Výsledek je vidět na obrázku8.1.

8.3 Psaní aplikace

V takto připraveném projektu můžeme využívat StartKit support library. Knihovna má definované funkce a rozhraní, které mají schopnost přístupu k led diodám, tlačítku a kapacitním senzorům. Proto tvorba aplikace, která využívá tyto periferie není příliš složitá.

8.4 Ukázkový Příklad

Příklad na rozsvícení všech 3X3 led diod. Pomocí tlačítka se dají led diody vypnout a po opětovném stisknutí se rozsvítí na plný jas. Pomocí dotykových jezdců lze nastavit dva



Obr. 8.1: Pohled na připravený projekt s možností využití knihovny StartKit support

různé nižší jasy. Každý jezdec na jeden jas.

8.4.1 Kód

```
#include <xs1.h>
#include <platform.h>
#include <print.h>
#include "startkit_gpio.h"
[[combinable]]
void app(client startkit_led_if leds,  // klientská úloha
         client startkit_button_if button,
         client slider_if i_slider_x,
         client slider_if i_slider_y)
{
  timer tmr;
  int delay = 100000;
  unsigned diody = 0b111111111; //lze zapsat číslo binárně
  int timestamp;
  unsigned svetlo = 0xFFFF;
  tmr :> timestamp;
  while (1) {
    select {
    case i_slider_x.changed_state(): // změna na jezdci
        sliderstate state = i_slider_x.get_slider_state();
        if(state == PRESSING|| PRESSED|| LEFTING|| RIGHTING)
            svetlo = 0x0e00;
        break;
    case i_slider_y.changed_state(): // změna na jezdci
           sliderstate state2 = i_slider_y.get_slider_state();
           if(state2 == PRESSING|| PRESSED|| LEFTING|| RIGHTING)
               svetlo = 0x4fff;
           break;
    case tmr when timerafter(timestamp + delay) :> void:
      leds.set_multiple(diody,svetlo); // aktualizace diod
      timestamp += delay;
      break;
    case button.changed(): // změna tlačítka
      if (button.get_value() == BUTTON_DOWN) {
        diody = ~diody;  // zapnout/vypnout diody
        svetlo = 0xffff; // plný jas
      }
```

```
      break;
        }
    }
}
startkit_gpio_ports gpio_ports =     // definice portu
  {XS1_PORT_32A, XS1_PORT_4B, XS1_PORT_4A, XS1_CLKBLK_1};

int main()
{
  startkit_led_if i_led;  // deklarace rozhraní (interface)
  startkit_button_if i_button;
  slider_if i_slider_x, i_slider_y;
  par {
    on tile[0]: startkit_gpio_driver(i_led,i_button, // řízení
    i_slider_x,
    i_slider_y,
    gpio_ports);
    on tile[0]: app(i_led,i_button,i_slider_x,i_slider_y);//ovládání
  }
  return 0;
}
```

8.4.2 Vysvětlení kódu

Úloha app má čtyři parametry client. To znamená, že bude očekávat změny na tlačítku, jezdcích a aktualizovat hodnotu jasu pro led. Pomocí rozhraní klienti odesílají změny do koncového serveru, který ovládá příslušné výstupy(například změní jas led). Server je naprogramovaný v knihovně startkit support. Stejně tak jsou v knihovně definovaná rozhraní, která využíváme v psaní úlohy klienta. V knihovně StartKitu je také naprogramovaná PWM pro devítici led diod. Hodnotu jasu určuje proměnná svetlo.

9

Závěr

Jednoprocesorový mikrokontrolér je stále velice užitečný, je vhodný pro méně náročné aplikace z hlediska výkonu. Zvládá však i náročnější aplikace, vždy záleží na zvoleném typu a architektuře. Pro náročnější aplikace, kdy je zapotřebí kontrolovat mnoho vstupních signálů v čase, může být jednoprocesorový mikrokontrolér nedostačující z hlediska časování. Při zpracování dat v reálném čase obvykle odskakuje na obsluhu přerušení a přerušeních může přicházet velice mnoho. Přerušení se vykonává s určitou prioritou a nikdy nemůžou být vykonána dvě přerušení naráz.

Výcejádový mikrokontrolér xCORE je z hlediska architektury úplně jiný, než jednoprocesorová provedení. Neobsahuje žádné rozsáhlé sběrnice a nemá žádnou vyrovnávací paměť cache. Paměť je společná jak pro data tak program a běží v paměti SRAM. Paměť SRAM sdílí nejméně 8 procesorových jader, kde každé jádro zvládá zpracovávat data ze vstupních pinů naráz. Jádra spolu komunikují přes spojovací matici xCONNECT. To činí architekturu xCORE velice vhodnou pro aplikace v reálném čase.

Při vytváření aplikací se může narazit na mnoho nepříjemných problémů. Kompiler a a spouštění aplikace v prostředí xTIMEComposer studio není úplně dokonalé, čas od času to chce při spouštění desku StartKitu odpojit a znovu připojit k PC. Také to chce občas čistit projekt aby kompiler nehlásil chyby. Provázání stažených knihoven se staženými příklady, nebo s vytvářenou aplikací, může být poněkud matoucí. Vždy však funguje, vložit všechny zdrojové soubory do složky src.

Programování mikrokontroléru v jazyce xC je velice efektivní a skrývá mnoho výhod. Pomocí jazyka xC, je programovaní komunikace jader mezi sebou a konfigurace portů poměrně jednoduchá. Skrývá však mnoho konstrukcí a složitostí, které vyžadují praxi a mnoho trpělivosti. Je zapotřebí naučit se programování rozhraní, konfigurace portů, provázání komunikace mezi jádry, nastavení časovačů, předávání proměnných a další. Zkušený vývojář však může využít již připravené hotové příklady a lehce je implementovat do svého projektu.

Veškeré příklady a kódy zmíněné v této práci byly otestovány na hardwaru StartKit. Odkazy na důležitou dokumentaci k architektuře xCORE a programovací software jsou v příloze.

Literatura

[1] Introduction-to-XS1-ports˙X2738B.pdf

[2] XS1-A8A-64-FB96-Datasheet˙1.4.pdf

[3] startKIT-Hardware-Manual˙1.3.pdf

[4] XMOS-Programming-Guide-˙documentation˙F.pdf

[5] Programming-XC-on-XMOS-Devices˙1.pdf

[6] lib˙startkit˙support-[userguide]˙3.0.1rc1.pdf

Příloha A

Odkazy na firemní dokumentaci

Veškerou dokumentaci lze naleznout v odkazu níže:
https://www.xmos.com/support/tools