Česká zemědělská univerzita v Praze
Technická fakulta
katedra elektrotechniky a automatizace

BAKALÁŘSKÁ PRÁCE

Uživatelské rozhraní pro ovládání mikropočítačového systému pomocí PC

Autor: Michal Bůžek

Vedoucí práce: doc. Ing. Stanislava Papežová, CSc.

Praha, 2012

Prohlášení

Prohlašuji, že jsem práci na téma „Uživatelské rozhraní pro ovládání mikropočítačového systému pomocí PC" vypracoval samostatně a použil jsem podklady uvedené v seznamu literatury.

Nemám žádný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 9. 4. 2012 .....

Poděkování

Děkuji především za odborné konzultace a velmi vstřícný přístup vedoucí mé bakalářské práce docentce Ing. Stanislavě Papežové CSc.

Abstract

This thesis is focused on developping a graphical user interface for easy creation and editing weekly values of the curve, with a possibility to backup values into a file and also to transmit values between computer and microcomputer system connected to it. Finally it deals with the design of microcomputer system itself. The result of this work is working computer application, that can succesfully communicate with the microcomputer system.

keywords: microcomputer, data transfer, USB, UART

Abstrakt

Tato práce se věnuje tvorbě grafického uživatelského rozhraní pro jednoduchou tvorbu a úpravy týdenní křivky hodnot s možností takto vytvořenou křivku zálohovat do souboru a také ji přenášet mezi počítačem a k němu připojeným mikropočítačovým systémem. Nakonec se práce zabývá i návrhem mikropočítačového systému. Výsledkem práce je funkční počítačová aplikace úspěšně komunikující s mikropočítačem.

Klíčová slova: mikropočítač, přenos dat, USB, UART

Obsah

1 Úvod

1.1 Motivace

Ještě před třiceti lety bychom kolem sebe obtížně hledali elektronické zařízení obsahující mikroprocesor. V dnešní době budeme naopak obtížně hledat elektroniku, která žádný procesor neobsahuje. Že se budou mikropočítače používat i v zařízeních na jedno použití a jejich ceny budou tak nízké, že se vyplatí použít mikroprocesor i v elementárních zapojeních, kdy to není nutné si v době jejich vzniku dokázal představit asi málokdo.

Vývoj šel neskutečně dopředu. Když jsem se poprvé setkal s mikrokontroléry, stál jeden tolik, co dnes stojí celý vývojový kit i s dvěma mikrokontroléry. Od malička se zajímám o elektroniku a tak mi tyto chytré elektrosoučátky vždy imponovaly. Lákalo mě se o nich dozvědět víc a proniknout tak do tajů jejich programování a zapojení. Proto, když jsem zjistil, že se dnes dá pořídit vývojový kit se dvěma 16-ti bitovými mikrokontroléry za cenu do 100 Kč, musel jsem to mít.

Chtěl jsem toto zařízení použít pro vývoj své bakalářské práce. Proto jsem si vybral téma bakalářské práce „Uživatelské rozhraní pro ovládání mikropočítačového systému pomocí PC". Problematika komunikace mikrokontroléru s velkým počítačem mi přišla zajímavá. Ačkoliv v zadání práce stojí „návrh programového vybavení mikrokontroléru na bázi 8051", po dohodě s vedoucím práce jsem se rozhodl použít jinou architekturu mikrokontroléru. Celou 3. kapitolu jsem věnoval shrnutí současných typů mikrokontrolérů, jejich programátorů a periferií potřebných pro mnou zamýšlený projekt.

1.2 Cíl a metodika práce

Tato práce se zabývá v první řadě tvorbou počítačové aplikace od počátečního zhodnocení a výběru vhodné vývojové platformy, přes návrh, až po samotnou realizaci. Aplikace by měla umožňovat tvorbu, úpravy, ukládání a otevírání (ze souboru) křivky hodnot v čase kalendářního týdne. Tato aplikace má dále umožňovat křivku odeslat do mikropočítačového zařízení a obráceně také přečíst křivku uloženou v mikrokontroléru. Jedno z možných použití může být tvorba regulačních hodnot pro precizní termostat s teplotou řízenou podle týdenní křivky.

V druhé řadě se práce zabývá volbou vhodného řešení na poli mikrokontrolérů a následně také návrhem programového vybavení mikrokontroléru. Návrh směřuje k zprovoznění všech funkčních bloků potřebných k vytvoření termostatu řízeného týdenní křivkou získávanou přenosem z osobního počítače.

Cílem práce je vytvořit funkční počítačovou aplikaci umožňující tvorbu týdenní křivky a komunikující s mikrokontrolérem pomocí USB rozhraní.

1.3 Struktura práce

Ve 2. kapitole se z počátku věnuji zhodnocení problematiky vývoje multiplatformních aplikací pro osobní počítače a výběru vývojového prostředí a programovacího jazyka. Poté se věnuji návrhu aplikace samotné a následují ukázky řešení konkrétních problémů se kterými jsem se potkal při realizaci.

3. kapitola je celá teoretická a věnuje se mikrokontrolérům a podpůrným obvodům. Jsou zde porovnány jednotlivé druhy mikorkontrolérů a jejich vývojové platformy. Poté se zde věnuji problematice komunikace po USB rozhraní, převodníkům a dalším důležitým zařízením pro vývoj křivkou řízeného termostatu.

4. kapitola se věnuje návrhu programového vybavení mikropočítače s krátkými ukázkami kódu.

2 Programové vybavení PC

2.1 Operační systémy

Každý moderní počítač potřebuje pro svojí činnost operační systém. Operační systém je software, který se stará o obsluhu periferií[1], správu paměti, běh uživatelských programů[2] a samozřejmě o uživatelské rozhraní. Snahou bývá, aby bylo uživatelské rozhraní co nejlépe pochopitelné pro běžného uživatele.

První počítače neměly operační systém. Na počátku 60. let 20. století dodávali výrobci počítačů propracované nástroje pro řízení dávkového zpracování spouštěných programů. První operační systémy byly dodávány k sálovým počítačům (mainframe). V roce 1967 byl firmou IBM vydán operační systém MFT, který podporoval v omezené míře multitasking. Od roku 1964 byl vyvíjen *Multics*, který však Bellovy laboratoře přestaly vyvíjet v roce 1969, kdy byl v těchto laboratořích vytvořen první Unix. [7]

2.1.1 Linux, Mac OS X, Windows

Operačních systémů je nepřeberné množství, v současné době jsou nejvíce rozšířené systémy pro osobní počítače na bázi *Unix*u (*Linux*, *Mac OS X*, *Android*) a *Windows* od společnosti Microsoft.

Unixové operační systémy mají rozdílné API od systémů *Windows*, tudíž nelze jednoduše[3] spouštět program psaný pro *Linux* na *Windows* a naopak. Aby to nebylo jednoduché, různé unixové systémy nemají jednotné API. To znamená, že není možné spustit bez úprav program psaný pro *Mac OS X* na *Linux*u. Dokonce není zaručeno, že program bude bez problémů fungovat na různých verzích stejného OS.

2.1.2 Grafické rozhraní

Dříve bylo uživatelské rozhraní čistě textové (konzole). Konzolové uživatelské rozhraní přijímá od uživatele textové příkazy a dává mu zpětnou vazbu pomocí výpisů do konzole. Tento systém je jednoduchý z hlediska složitosti programu a systémových nároků na hardware. Není však jednoduchý na pochopení uživatelem, který si musí pamatovat množství příkazů a jejich syntaxi.

Proto se již od 70. let pracovalo na vytvoření grafického uživatelského rozhraní (GUI). Dnes najdeme grafické rozhraní snad na každém osobním počítači. Grafické rozhraní osobního počítače se zpravidla snaží být co nejvíce intuitivní pro jednoduché používání běžným uživatelem. Ideál je, když jej dokáže bez potíží ovládat i člověk bez předchozích zkušeností s ním.

Zatímco unixová konzole je díky *POSIX* standardům na všech unixových systémech stejná, konzole v *MS Windows* se od dob *MS DOS*u prakticky nezměnila, u grafických rozhraní je situace podstatně složitější.

[1] externí zařízení jako tiskárny, klávesnice atp
[2] musí vytvořit pro procesy stabilní aplikační rozhraní (API) a přidělovat jim systémové zdroje
[3] existují emulátory, jako wine pro Linux, či Cygwin pro Windows pro tyto účely

2.1.3 X Window System

Vznikl na MIT jako snaha o sjednocení GUI prostředí v Unixových systémech. Od začátku byl navržen jako nezávislý na konkrétní platformě.

V současnosti se na Linuxových systémech používá implementace *X.Org. X Window System* obsahuje grafickou knihovnu *Xlib* pro programování grafických. Tu však v současné době používá velmi málo aplikací.

Většina aplikací místo toho používá widgety z různých nadstavbových frameworků (*GTK+*, *Qt*, *Motif* atd.). Díky tomu GUI není ani na Linuxových systémech jednotné, jak bylo původně zamýšleno.

2.1.4 Cococa

Cococa je API operačního systému *Mac OS X* od společnosti Apple. *Cococa* používá GUI *Aqua*. *Mac OS X* obsahuje i rozhraní *Carbon*[4] pro zajištění zpětné kompatibility se starými aplikacemi pro Mac OS a také *X Window System* pro spouštění Linuxových aplikací. V praxi to však nefunguje u všech Linuxových aplikací, kvůli chybějícím knihovnám.

2.1.5 Windows API

Operační systém *Windows* má jen jedno grafické rozhraní s jedním API, to je z hlediska programátora značné zjednodušení, avšak uživatel ztrácí možnost svobodné volby rozhraní, jakou má např. v prostředí Linuxu.

2.2 Tvorba multiplatformních aplikací

Na trhu s operačními systémy pro dominuje systém *Windows* od společnosti Microsoft, avšak v poslední době začal jeho podíl klesat.[4] Může za to rostoucí obliba počítačů Apple a rychlý růst podílu mobilních zařízení v poslední době.

Na osobních počítačích se nejčastěji používají operační systémy *Windows*, *Mac OS X* a *Linux*. Při tvorbě aplikací pro osobní počítače je proto vhodné je již od počátku navrhovat tak, aby je bylo možné provozovat nezávisle na operačním systému. Jak by se mohlo zdát po přečtení minulé kapitoly, vytvořit aplikaci, která by fungovala na všech třech platformách (*Windows*, *Linux*, *Mac OS X*) by znamenalo napsat ji třikrát a pokaždé pro jiné API.

Naštěstí je situace o mnoho jednodušší, díky multiplatformním řešením. A není jich málo. Já se zmíním o těch nejznámějších.

[4] díky Carbonu je možné spouštět aplikace psané pro *Power PC* i na dnes používané platformě *Intel*

2.2.1 Java

Nejznámější je asi platforma Java. Její hlavní výhodou je její portabilita. To znamená, že Java běží úplně stejně na různém hardwaru a různých operačních systémech.

To je zajištěno tím, že programy v Javě se kompilují do Java bytekódu místo toho, aby se kompilovaly přímo pro konkrétní platformu a hardware. Zkompilovaný bytekód se poté spouští na konkrétním hardwaru a OS pomocí *Java Virtual Machine*, která překládá instrukce pro konkrétní hardware a operační systém.

Nevýhodou je rychlost, která je díky zpracovávání bytekódu nižší a vzhled aplikace, který je typický pro aplikace psané v Javě, není však plně integrován do grafického prostředí operačního systému a díky tomu také vypadá jinak než ostatní aplikace. Ačkoliv tento problém by měly řešit témata *Swing*u.

2.2.2 GTK+, QT

Grafické knihovny známé z Linuxu se dají používat i na Windows, vzhled aplikací však není nativní a nejsou plně integrovány[5] do prostředí Windows, do Windows se instalují další grafické knihovny a jejich použití není úplně pro začátečníky. Výhodou však je, že je opensource a zdarma. Příklad aplikace v *GTK+* je *Gimp*, pro který byl původně celý GTK (Gimp ToolKit) napsaný.

2.2.3 wxWidgets

Od roku 2004 se takto jmenuje původně projekt *wxWindows*. Je to opensource projekt a jeho velkou výhodou je, že zajišťuje nativní vzhled ve všech prostředích, která podporuje (Windows, Linux, Mac OS X) Nevýhoda je, že tvorba grafického rozhraní se odehrává psaním kódu, což je nejen zdlouhavé, ale i náročné na naučení.

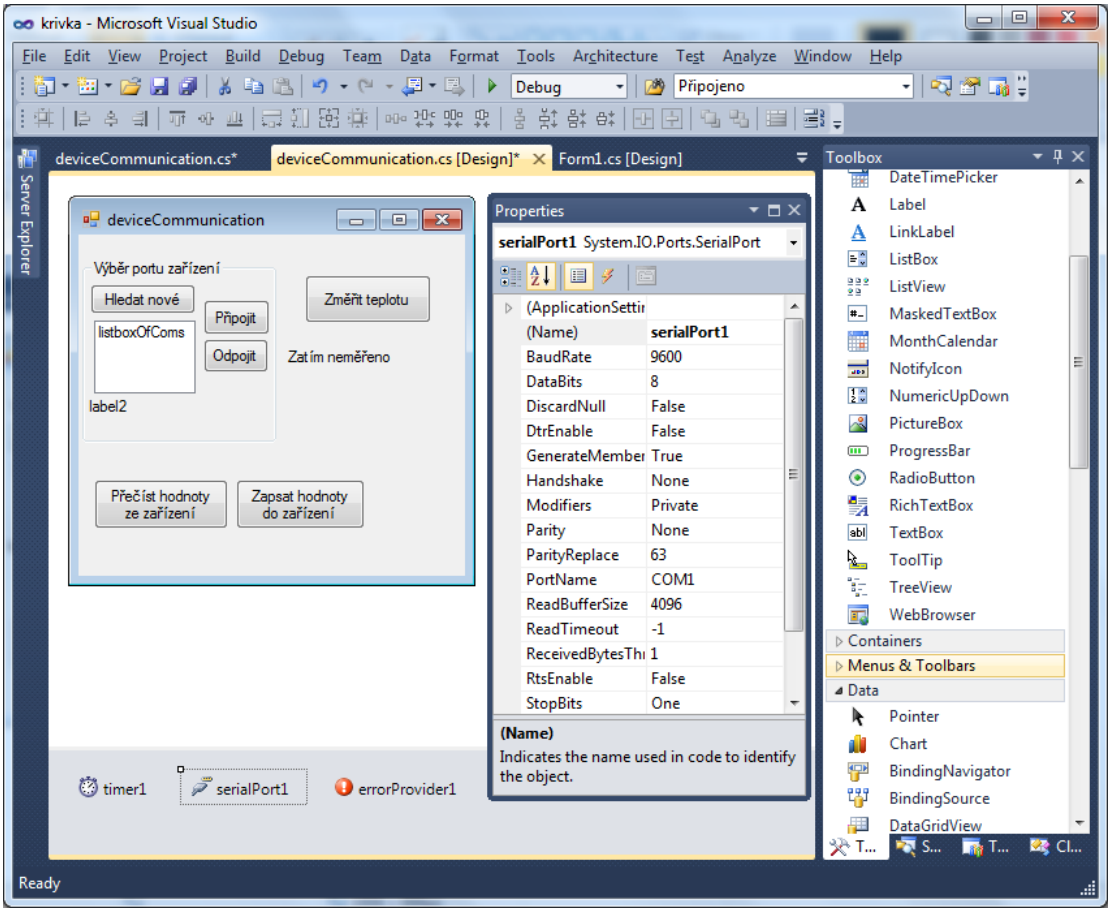
2.2.4 .NET Framework

.NET Framework je platforma společnosti MICROSOFT. Stejně jako *Java* se nekompiluje přímo do strojového kódu, nýbrž do jazykové mezivrstvy tvořené kódem jazyka *Microsoft Intermediate Language* (MSIL). Až z mezikódu MSIL se překládá do strojového kódu na konkrétním hardware a platformě.

Do MSIL se může kompilovat z široké škály programovacích jazyků, jako jsou např. C++ .NET, C# .NET, J# .NET, či Visual Basic .NET. Díky společnému běhovému prostředí (*Common Language Runtime*) není problém tvořit projekt v týmu, kde každý programátor používá jiný jazyk, ba dokonce je možné převádět kód z jednoho jazyka do jiného.[10]

Aplikace psané v .NET Frameworku jsou plně integrovány do prostředí MICROSOFT *Windows* a díky Opensource implementaci .NET Frameworku *Mono* je možné aplikace přenést i na jiné platformy, jako je *Linux*, *Mac OS X*, či *Android*. *Mono* podporuje různé grafické toolkity (*Windows.Forms*, *GTK#*,

[5] např. používá vlastní (jiné) dialogy pro práci se soubory



Obrázek 1: Tvorba uživatelského rozhraní ve Visual Studiu

Qyoto, *MonoMac*), díky čemuž může výsledná aplikace vypadat na každé podporované platformě nativně.

.NET Framework se neomezuje jen na tvorbu aplikací pro desktop, je v něm možné tvořit třeba i rozsáhlé webové projekty, je to vskutku robustní systém.

Velkou výhodou při programování v .NET Frameworku je použití vývojového prostředí Microsoft *Visual Studio*, které poskytuje velice propracovaný designer pro návrh grafického rozhraní, jednoduše a efektivně. viz *obr. 1*, vynikající nástroje pro debugování[6] a také velmi rozsáhlou dokumentaci s ukázkami kódu.

2.3 Výběr vývojového prostředí

Pro vývoj uživatelského rozhraní na ovládání mikropočítače jsem zvolil Microsoft *Visual Studio* (*obr. 1*) a programovací jazyk C# .NET.

Požadoval jsem tvorbu multiplatformní aplikace (.NET) v objektovém programovacím jazyce s Céčkovou syntaxí (C#) s dostatečným množstvím užitečných knihoven a jednoduchou (vizuální) tvorbou grafické části aplikace (Visual Studio).

[6] hledání a odstraňování chyb v programu

S ohledem na majoritní platformu MICROSOFT *Windows* jsem se rozhodl pro grafickou část zvolit toolkit *Windows.Forms*, který nabízí plnou integraci s prostředím MICROSOFT *Windows*. Projekt *mono* jej také podporuje a je možné v něm psané aplikace díky tomu spustit i na jiných platformách, ovšem se vzhledem Windows aplikace. Pro nativní vzhled na jiných platformách, by se musela grafická část upravit pro jiný toolkit (např. *GTK#* pro *Linux* a *MonoMac* pro *Mac OS X*), zatímco aplikační logika by se přepisovat nemusela ani v tomto případě.

I když jsem jinak zastánce opensource řešení. V tomto případě jsem dal přednost komerčnímu produktu právě pro svou dotaženost v oblasti jednoduché tvorby aplikací s množstvím nástrojů značně ulehčujícím práci.

Při mém rozhodování mezi platformou Java a C# mi pomohl mimo jiné i článek „Java vs. C# - který jazyk zvolit?" na www.zive.cz. [11]

2.4 Návrh uživatelského rozhraní

2.4.1 Požadavky

Uživatelské rozhraní by mělo umožňovat tvorbu a úpravy teplotní křivky. Mělo by mít možnost křivku přenést do mikropočítače a naopak nahrát křivku z mikropočítače do PC a tu následně zobrazit. Zobrazenou křivku v programu by mělo být možné uložit a také vyvolat z uloženého souboru.

2.4.2 Vzhled a rozmístění prvků

Program by měl být co nejvíce intuitivní a jeho ovládání by mělo být dostatečně jednoduché, aby umožňovalo pohodlné zadávání hodnot.

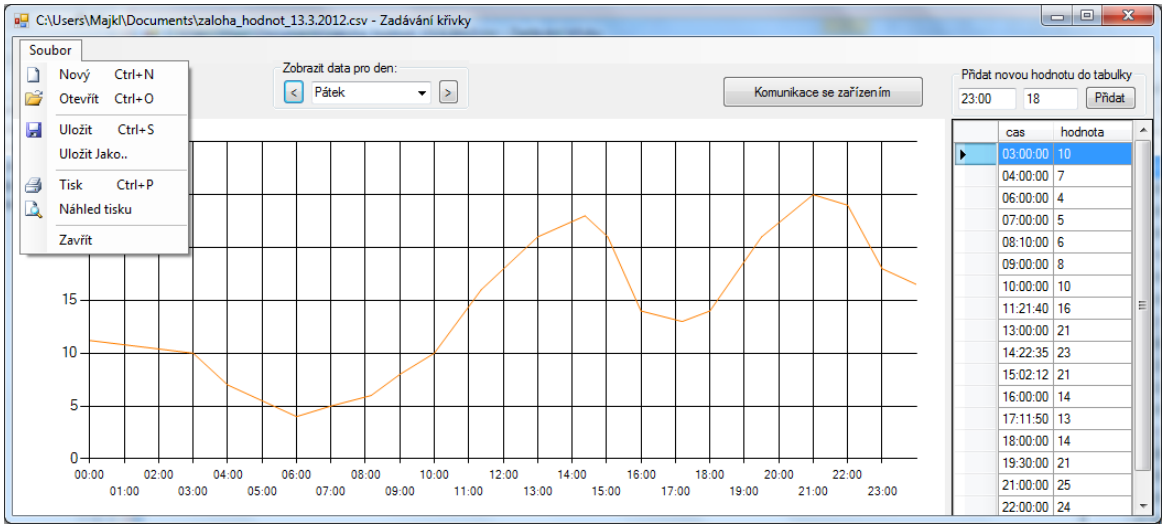
2.4.3 Hlavní okno

Na *obr. 2* je vidět jak vypadá hlavní okno programu.

Menu *Soubor* umožňuje ukládat/načítat křivku do/ze souboru v počítači. Nahoře se vybírá den v týdnu, pro který se bude zobrazovat křivka s hodnotami. Vlevo pod výběrem dne je vyobrazena samotná křivka a vpravo jednotlivé hodnoty křivky pro zvolený den.

Graf křivky navazuje na poslední hodnotu z předešlého dne a pokračuje na první hodnotu ze dne následujícího. Tím je zajištěno zobrazení části křivky před první a za poslední hodnotou grafu. Začátek křivky v Pondělí navazuje plynule na konec křivky v Neděli.

Při změnách v tabulce hodnot se automaticky překresluje graf křivky. Změny se provádějí dvojklikem na políčko, které chceme změnit. Přidávání nových hodnot se provádí formulářem nad tabulkou a mazání označeného záznamu je uskutečněno po stisku klávesy *delete*.



Obrázek 2: Hlavní okno programu

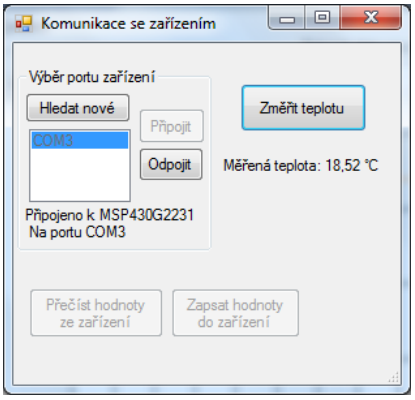
Mezi dny v týdnu se dá přepínat buď změnou dne v padacím menu, nebo pomocí šipek umístěných po stranách padacího menu.

Současná verze programu obsahuje i tlačítko *Generovat náhodné hodnoty*, které naplní všech sedm dní v týdnu náhodnými hodnotami v náhodných časech. Toto tlačítko sloužilo k ulehčení práce při testování aplikace v průběhu její tvorby a může být odstraněno.

Tlačítko *Komunikace se zařízením* otevře nové okno určené pro komunikaci s mikropočítačem.

2.4.4 Okno komunikace se zařízením

Na *obr. 3* je okno určené pro vlastní komunikaci s mikropočítačem.

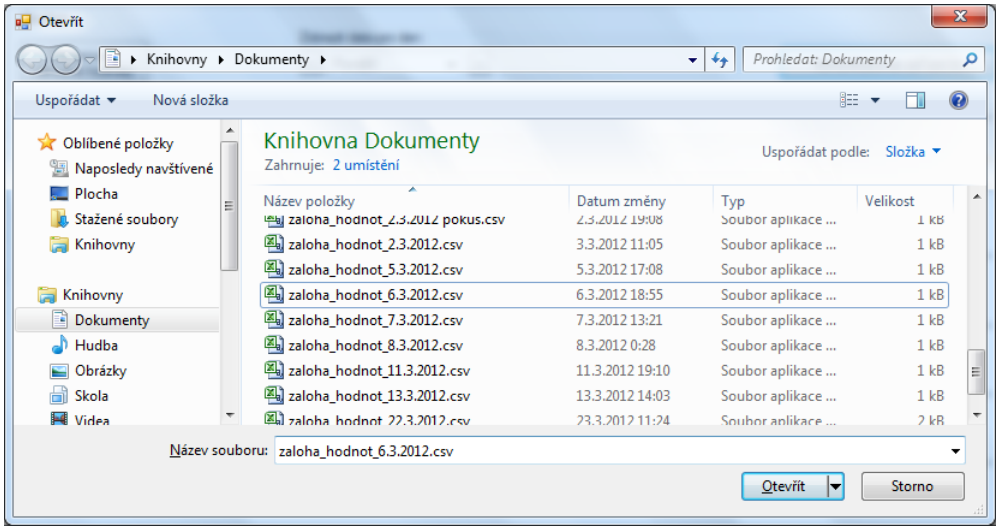


Obrázek 3: Okno Komunikace se zařízením

Tlačítko *Hledat nové* vyhledá všechny sériové porty v počítači (zařízení se připojuje přes virtuální sériový port) a zobrazí je. Pod tímto tlačítkem je seznam všech vyhledaných portů, kde si uživatel může jeden z nich vybrat. K zařízení na vybraném portu se poté může uživatel připojit stisknutím tlačítka *Připojit*.

Po připojení se tlačítko *Připojit* stane neaktivní a nelze na něj klikat a naopak aktivními se stanou tlačítka *Odpojit*, *Změřit teplotu* a pokud to zařízení podporuje, tak i tlačítka *Přečíst hodnoty ze zařízení* a *Zapsat hodnoty do zařízení*. Po připojení se také zobrazí k jakému zařízení je program připojen a na jakém portu. V současné době se je název zařízení typ použitého mikrokontroléru[7], na *obr. 3* se

[7] v případě komerčního využití by bylo vhodné místo názvu procesoru zobrazovat marketingový název zařízení



Obrázek 5: Dialog pro otevírání souboru

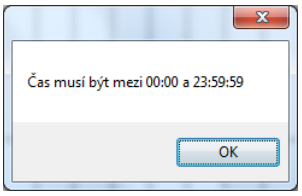
jedná o procesor *MSP430G2231* od Texas Instruments, který bohužel není natolik vybaven, aby mohl podporovat regulaci teploty podle křivky a proto se nabízí pouze tlačítko *Změřit teplotu*.

Po kliknutí na tlačítko *Změřit teplotu* se zobrazí teplota změřena mikrokontrolérem. Tlačítka Zapsat/-Přečíst hodnoty mají za úkol křivku hodnot buď uložit do mikrokontroléru, nebo přečíst z mikrokontroléru, který by podle ní měl řídit teplotu.

2.4.5 Dialogy

Pro otevírání a ukládání souborů se díky použití *Windows.Forms* toolkitu používají standardní filedialogy z Windows viz. *obr. 5*. Na jiných platformách by při použití odpovídajícího grafického toolkitu vypadaly dialogy také nativně.

Chyby, jako například špatný formát vkládaného času jsou uživateli zobrazovány ve vyskakujícím dialogovém okně, které je vidět na *obr. 4*.

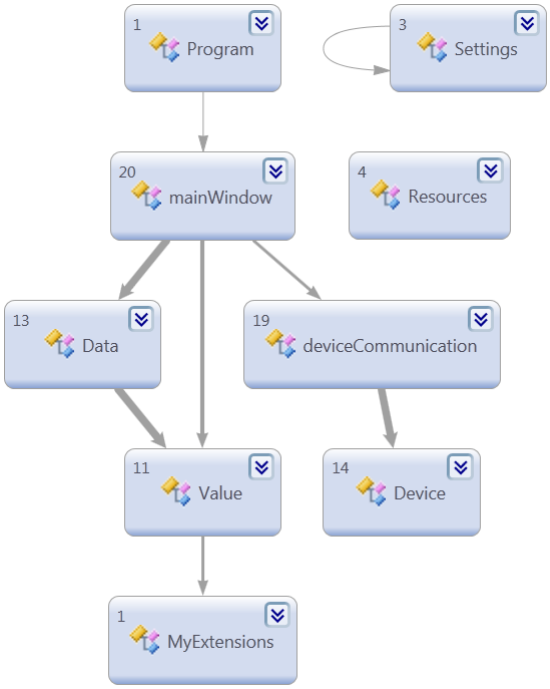


Obrázek 4: Chybové okno

2.5 Realizace

2.5.1 Objektový návrh

Na *obr. 6* jsou vidět třídy a vztahy mezi nimi. Třída *mainWindow* znázorňuje hlavní okno programu (*obr. 2*), třída *deviceCommunication* znázorňuje okno komunikace se zařízením (*obr. 3*). Třída *Data* se má starat o vše spojené s daty, ať už jejich uložení, tak i jejich třídění, či ukládání a načítání ze souboru. Třída *Hodnota* v sobě uchovává čas a hodnotu a je součástí kolekce *values* ve třídě *Data*, kde se všechny hodnoty uchovávají.



Obrázek 6: Vztahy mezi objekty

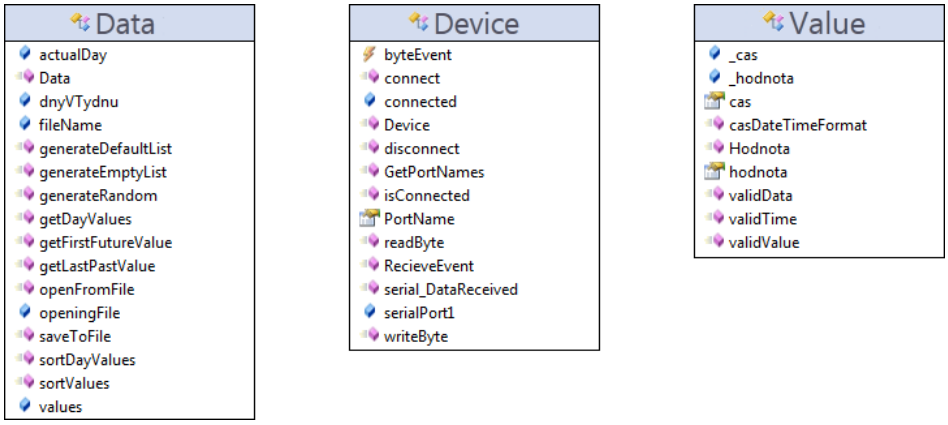
Ve skutečnosti se v projektu používá mnohem víc objektů, jako např. *chart* (znázorňující graf), *serialPort* (znázorňující sériový port), či *dataGrid* (zodpovědý za tabulku hodnot), ty jsou však přímo součástí .NET frameworku a v obrázku jsou znázorněny jen třídy mnou definované. Zde je vidět v praxi jak .NET Framework usnadňuje rutinní práci a umožňuje programátorovi se více soustředit na skutečné jádro problému.

2.5.2 Datové operace

O všechnu manipulaci s daty se stará třída *Data*, její struktura je znázorněna na *obr. 7*. Nejdůležitější je kolekce *values*, v té jsou uloženy všechny hodnoty ze všech dnů. Je to kolekce kolekcí hodnot. Každý den má svojí kolekci hodnot a tyto kolekce jsou sdruženy v kolekci *values*. Pole *dnyVTydnu* obsahuje názvy dnů v týdnu pro jejich zobrazení. Metoda *getDayValues* vrací kolekci hodnot pro den zadaný v parametru metody.

Metoda *sortValues* seřadí všechny hodnoty ve všech dnech pomocí metody *sortDayValues*, která s využitím delegování seřadí hodnoty podle času v daném dni. Třídění probíhá po každé editaci, přidání nové hodnoty, či načtení ze souboru. Jednak kvůli přehlednosti, jednak kvůli kontinuitě křivky v grafu. Aby bylo zobrazení grafu kompletní, včetně návaznosti na minulý a následující den, získají se poslední hodnota z předchozího dne a první hodnota ze dne následujícího pomocí metod *getFirstFutureValue* a *getLastPastValue* a tyto se přidají ke grafu, čímž se zajistí kontinuální křivka.

Každý den musí obsahovat alespoň jednu hodnotu, proto se místo zobrazení prázdné tabulky vygeneruje defaultní kolekce metodou *generateDefaultList* a její obsah se poté zobrazí. Každý den v této kolekci obsahuje hodnotu 22 v 7:00 ráno, je poté již ja uživateli či ji ponechá, či změní. Tím je také



Obrázek 7: Struktura tříd Data, Device a Value

zajištěno, že se nenarazí na problém, že den neobsahuje žádné hodnoty a není kde vzít předchozí, či následující hodnotu.

Třída *Value* (*obr. 7*) reprezentuje jednotlivou hodnotu. Každá instance třídy *Value* obsahuje čas a hodnotu samotnou. Tyto instance jsou pak uloženy do seznamu *Data.values*. Třída *Value* má několik přetížených konstruktorů, každý pro různý formát času a hodnoty (string, či TimeSpan / string, či Integer). Pro převod času ze *string*u do *TimeSpan*u je s výhodou použita technologie *Extension Methods*, která je součástí .NET frameworku od verze 3.5 (C# verze 3.0). Ve třídě *MyExtensions* je nadefinovaná „Extension" *toTimeSpan(this String str)*, která funguje podobně jako metoda *toString()*, akorát převádí řetězec ve formátu "00:00", či "00:00:00" na typ *TimeSpan*.

Třída *Data* se stará také o ukládání a nahrávání dat ze souboru. Z hlediska čistě objektového by tuto činnost mohla dělat nová třída *File*, avšak třída *Data* toho nemá tolik na starosti aby nemohla obsluhovat i ukládání a načítání souborů, koneckonců je to také operace s daty. Proto k těmto účelům obsahuje třída *Data* metody *saveToFile* a *openFromFile*.

Formát souboru pro ukládání dat jsem zvolil léty osvědčený **.csv*. Je to klasický textový soubor, jehož struktura vypadá následovně:

```
1 0;03:00:00;19
2 0;04:00:00;14
3 0;12:00:00;11
```

Každý záznam je na novém řádku a jednotlivé položky jsou odděleny středníkem. V našem případě je první hodnota číslo dne v týdnu (0-6), kde číslo 0 znamená pondělí a 6 neděli. Druhá položka je časový údaj a třetí položka je hodnota, například teploty pro daný den a čas. Formát *csv* je možné otevřít i obyčejným textovým editorem, či tabulkovým procesorem (Microsoft Excel, OpenOffice Calc, atp.), což mi připadá výhodné. Soubor hodnot tak může zobrazit, či upravit i člověk, který nemá nainstalovanou speciální obslužnou aplikaci.

Metoda *saveToFile* ze zobrazených hodnot vytvoří tento *csv* soubor a metoda *openFromFile* naopak tento soubor rozparsuje a vytvoří ze získaných dat kolekci ve formátu kolekce *Data.values*. Následující ukázka části kódu metody *openFormFile* by měla dostatečně vysvětlit jakým způsobem se tak děje:

Výpis 1: Načítání dat ze souboru

```
Data.values = generateEmptyList(); //nejdříve vygenerujeme prázdný seznam
bool formatError = false; //pro případ, že by některé řádky měly špatný formát
//dokud je co číst tak načítáme
while (tr.Peek() >= 0)
{
  readData = tr.ReadLine().Split(new Char[] { ';' });
  //pokud nejsou na řádku tři hodnoty, je to špatně
  if (readData.Length == 3)
  {
    //první hodnota je den v týdnu
    actualDay=Convert.ToInt32(readData[0]);
    try
    {
      //druhá čas a třetí hodnota, třdída Value si při vkládání hodnot
          zkontroluje jejich validitu
      Value value = new Value(readData[1], readData[2]);
      //pokud nevznikla vyjímka zapsat hodnoty do seznamu
      Data.values[actualDay].Add(value);
    }
    catch
    {
      //vznikla vyjímka - alespoň jeden řádek souboru měl špatný formát
      formatError = true;
    }
  }
  else
    formatError=true
}
//na výskyt špatně formátovaných řádků uživatele upozorníme, zbytek hodnot se
    použije
if (formatError)
  MessageBox.Show("Soubor obsahuje chybné údaje, které nebudou načteny (
      například více hodnot pro jeden čas)");
```

2.5.3 Zobrazení dat

O zobrazení dat se stará třída *mainWindow*. V této třídě jsou definovány všechny obslužné funkce událostí hlavního okna. Například při stisku (přesněji až při uvolnění) klávesy v zadávacím poli *tbHodnota (pole pro zadávání nové hodnoty)* se zkontroluje zda stisknutá klávesa byla klávesa *Enter* a pokud ano, vyvolá se událost kliknutí na tlačítko *Přidat* a také se přesune kurzor do zadávacího pole pro čas, aby bylo možné pohodlně zadat další hodnotu. Tímto je zajištěno jednoduché přidávání hodnot pomocí klávesy *Enter*. Tato jednoduchá obslužná funkce je k vidění na následující ukázce kódu.

Obdobným způsobem vypadají i obslužné funkce všech jiných událostí, jako je např. kliknutí na tlačítko, či změna hodnoty v tabulce.

Výpis 2: Obsluha událostí

```
private void tbHodnota_KeyUp(object sender, KeyEventArgs e)
{
  if (e.KeyCode == Keys.Enter)
  {
    btAddValue.PerformClick();
    tbCas.Focus();
  }
}
```

Pro zobrazování dat je důležitá metoda *viewDay(int actualDay)*, při jejímž zavolání se aktualizuje tabulka hodnot (*dataGrid*) a zavolá se metoda *populateChart(actualDay)*, která má na starosti překreslení grafu dle aktuálních hodnot ve třídě *Data*. Metoda *viewDay* se volá po každé změně v kolekci *Data.values* (tj. mazání, přidávání, editace záznamu, změna dne, načítání nových hodnot, atd.). Jádro metody populateChart funguje následovně:

Výpis 3: Zobrazování grafu

```
//Datum aktuálního dne, nezáleží na něm, záleží až na časech k němu přičtených
DateTime baseDate = new DateTime(1988, 01, 01, 00, 00, 00);
//Získáme poslední hodnotu z předchozího dne
Value lastPastValue = data.getLastPastValue(day);
//Datum pro hodnotu z předchozího dne musí být den před aktuálním dnem
DateTime lastDate = baseDate.AddDays(-1);
//k "včerejšímu" datu přičteme čas posledního záznamu
DateTime chartTime = lastDate.Add(lastPastValue._cas);
//tento záznam bude první hodnotou grafu (není vidět, ale vede na něj spojnice)
series.Points.AddXY(chartTime, lastPastValue._hodnota);
//naplníme graf hodnotami z aktuálního dne
foreach (Value value in dayValues)
{
  chartTime = baseDate.Add(value._cas);
  series.Points.AddXY(chartTime, value._hodnota);
}
//přidáme hodnotu z následujícího dne obdobným způsobem jako tu z minulého dne
Value firstFutureValue = data.getFirstFutureValue(day);
DateTime nextDate = baseDate.AddDays(+1);
chartTime = nextDate.Add(firstFutureValue._cas);
series.Points.AddXY(chartTime, firstFutureValue._hodnota);
```

Aktuální den, neboli den, který se zrovna zobrazuje uživateli se získává jako vlastnost *SelectedIndex* objektu *dayCombo*, který reprezentuje ComboBox výběru dne.

2.5.4 Komunikace s mikrokontrolérem

Pro komunikaci s mikrokontrolérem je určeno okno *Komunikace se zařízením*, které reprezentuje třída *deviceCommunication*. Ta krom metod zajišťujících obsluhu grafických prvků obsahuje také metodu

dataReceived, která je volána ze třídy *Device* (její struktura na *obr. 7*) ve chvíli, kdy sériový port přijme data. Uvnitř této metody se poté, podle přijatého příznaku, rozhoduje co je to za data a co se s nimi má dělat. Konkrétně se opět jedná o ovladač události, který se zaregistruje při úspěšném připojení k zařízení. Tato registrace ovladače události vypadá následovně:

```
1  device.byteEvent += new Device.recieveByteEvent(dataReceived);
```

Funkce *dataRecieved* je díky tomu volána z jiného vlákna (threadu) než v jakém vlákně probíhá veškeré zobrazování formulářových prvků. Pokud bychom chtěli změnit obsah nějakého prvku formuláře (např. výpis stavové informace do *label*u) z funkce *dataRecieved*, dostali bychom chybu, že nelze měnit cokoliv v jednom threadu z threadu jiného. Je to tak záměrně. Kdyby nějaký thread něco změnil v jiném threadu, jak by se o tom ten první dozvěděl? Toto by mohlo vést k obtížně dohledatelným chybám, proto je to v .NET frameworku zakázáno.

Neznamená to ale, že to nejde udělat. Pro tyto účely se používá *Control.Invoke Method*. Tato metoda spustí delegovanou funkci na threadu, který má kontrolu nad žádaným prvkem. Tak je zajištěna bezpečná mezi-vláknová komunikace. Jak to vypadá v kódu metody *deviceCommunication.dataRecieved* je patrné z krátké ukázky kódu.

Výpis 4: Control.Invoke Method

```
1  void dataReceived(object sender, ushort value)
2  { //provádí se invoke nad celým formulářem (this)
3    this.BeginInvoke(new EventHandler(delegate{
4      //mode se používá pro uchování informace co se má dít
5      switch (mode)
6      { //defaulně je MODE.NONE, neočekáváme žádná data
7        case MODE.NONE:
8        break;
9  //HANDSHAKE znamená, že uživatel klikl na tlačítko připojit a mikrokontroléru
       byl odeslán požadavek na identifikaci a na ten MCU odpoví své identifikační
        číslo podle modelu
10       case MODE.HANDSHAKE:
11 //pokud se přijatý identifikátor shoduje s hodnotou pro daný typ, je úspěšně
       připojeno.
12         if (value == MSP430G2231)
13         {
14           status.Text=("Připojeno k MSP430G2231 \n Na portu " + device.PortName
                 );
15 //connected nastavuje aktivní/ neaktivní talčítka atp.
16           connected(true);
17 //tento model umí jen měřit teplotu, tak aktivujeme tlačítko pro měření teploty
18           btMeasureTemp.Enabled = true;
19           mode = MODE.NONE;
20         }
21     }
22   }));
23 }
```

Třída *Device* se poté stará o vše spojené s mikrokontrolérem. Jsou to operace, jako připojování, přijímání dat, odesílání dat, detekce chyb atp.

Při připojování se uvnitř metody *connect* zaregistruje událost přijmutí dat na sériový port.

Výpis 5: Registrace události *SerialDataRecieved*

```
serialPort1 = new SerialPort(selectedCom);
serialPort1.DataReceived +=
new SerialDataReceivedEventHandler(serialPort1_DataReceived);
```

Tato událost spouští obslužnou funkci *serialPort1_DataRecieved,* kde se zprostředkuje událost *recievedByteEvent* přístupná z třídy *deviceCommunication.*

Výpis 6: Příjmání dat přes sériový port

```
//definice události
public delegate void recieveByteEvent(object sender, ushort data);
public event recieveByteEvent byteEvent;
protected virtual void RecieveEvent(ushort data)
{
  if (byteEvent != null)
    byteEvent(this, data);
}

void serial_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
  int numToRead = serialPort1.BytesToRead;
  //komunikujeme s 16ti bitovým mikroprocesorem po dvou 8-mi bitových slovech,
      případný lichý počet bajtů upravíme na sudý
  if (numToRead % 2 != 0)
    numToRead--;
  //načteme data do pole serialBuf
  byte[] serialBuf = new byte[numToRead];
  serialPort1.Read(serialBuf, 0, numToRead);

  ushort temp = 0;
  //načteme dvoubajtové hodnoty, spojíme je do jedné proměnné a vyvoláme
      událost pro zpracování dat
  for (int i = 0; i < numToRead; i = i + 2)
  {
    temp = (ushort)serialBuf[i + 1];
    temp <<= 8;
    temp |= (ushort)serialBuf[i];
    RecieveEvent(temp);
  }
}
```

Při úspěšném načtení hodnot z mikrokontroléru se aktualizuje křivka v hlavním okně na křivku načtenou z mikrokontroléru. V případě ukládání je uživatel pouze informován o úspěšném zapsání hodnot do paměti mikrokontroléru.

3 Mikrokontroléry

3.1 Souhrn vybraných typů

Na trhu s mikrokontroléry dnes figuruje mnoho firem. V této kapitole se pokusím shrnout vlastnosti a výhody různých typů od různých výrobců. Převážně se jedná o 8-mi bitové mikrokontroléry, ale najdou se i 16-ti, či 32-bitové.

3.1.1 8051

mikrokontrolér 8051 pochází z roku 1980, kdy byl vyvinut společností INTEL a je vývojově procesorem relativně starým. U návrhářů však dosáhl takové obliby, že i v současné době se řada výrobců orientuje na výrobu procesorů založených na jádře procesoru 8051, které je rozšířeno o více či méně dalších periferií. Například firma PHILIPS vyrábí 24 různých typů těchto procesorů majících společné jádro, ke kterému jsou připojeny některé z periferií jako je:

- paměť programu o velikosti 2kB až 32kB, kterou lze programovat jenom jednou (provedení OTP) nebo několikrát (provedení EPROM)[8]
- paměť EEPROM pro uchování konstant
- rozšířená vnitřní paměť RAM na 256b
- 8 nebo 10-bitový A/D převodník obvykle s osmikanálovým analogovým multiplexerem
- rozšířené vstupně/výstupní vývody
- komparační a záchytný systém
- I^2C sběrnice

Výrobci nabízejí procesory od základního hodinového kmitočtu 12MHz až po 33MHz ve standardním pouzdře DIL, přes provedení PLCC až po malá pouzdra PQFP [9].

Když firma ATMEL viděla, jaký úspěch na trhu zaznamenal INTEL se svojí rodinou 80C5x, uvedla kolem roku 1993 svoji inovaci tohoto oblíbeného mikrokontroléru. Tou inovací bylo použití paměti Flash jako programové paměti namísto do té doby používané paměti EPROM. Díky velmi dobře zvládnuté technologii Flash slavila firma ATMEL velké úspěchy s tímto mikrokontrolérem. To vedlo firmu k uvedení vlastních „odvozenin“ základního typu. Tak vznikly oblíbené mikrokontroléry rodiny 89Cxx a 89Sxx [8].

Výše uvedené informace již nejsou zcela aktuální. Dnes se dá pořídit mikrokontrolér s jádrem 8051 běžící na frekvenci až 100 Mhz, nebo vybavený USB portem, avšak stále je to stará architektura z

[8] Procesory s programovou pamětí EPROM musely mít drahé keramické pouzdro s okénkem pro mazání paměti UV zářením. Existovaly také levnější verze procesoru s laciným plastovým pouzdrem, ovšem díky absenci mazacího okénka, již nebylo možné program v nich měnit. Tyto procesory s laciným pouzdrem se používaly pro velké série po dokonalém odladění programu.

dob, kdy se nepočítalo s programováním MCU ve vyšších jazycích než v Assembleru. Jistě, že je to s množstvím paměti, které dnes tyto procesory nabízejí, možné, ale kompilace kódu v jazyce C pro tuto architekturu nikdy nebude optimalizována tak dobře, jako pro jednu z moderních architektur, které jsou pro programování v jazyce C přímo navrhovány.

3.1.2 AVR

Velké oblibě se dnes těší mikrokontroléry s architekturou AVR od společnosti ATMEL.

Na začátku 90.let minulého století se skupina norských návrhářů spolu s programátory rozhodla navrhnout novou strukturu mikrokontroléru tak, aby struktura tohoto mikrokontroléru vyhovovala překladačům vyšších programovacích jazyků, zejména široce používaného jazyka C. Výsledkem snažení této skupiny bylo optimalizované jádro nové řady mikrontrolérů s harvardskou architekturou nesoucí hlavní charakteristiky mikrokontrolérů s redukovanou instrukční sadou (RISC).

Na rozdíl od jádra mikropočítače 8051 má tato architektura 16bitová instrukční slova. Zvětšení šířky instrukčního slova na jednu stranu zvětšilo požadavky na velikost paměti, na druhou stranu však umožnilo zrychlit načtení mnoha instrukcí, neboť kromě několika výjimek, vystačí instrukce s jedním slovem, tj. mikrokontrolér je dokáže načíst během jednoho hodinového cyklu. Druhým viditelným rozdílem je propojení ALU s tzv. polem 32 pracovních registrů. Tato organizace ALU spolu se 16 bitovým instrukčním slovem umožnila návrhářům snížit počet hodinových taktů potřebných na provedení téměř všech instrukcí na pouhé dva takty, načtení+dekódování a vykonání. Díky tomu že instrukce vystačí s jedním slovem, to umožnilo implementaci jednoduchého překrývání zmíněných dvou fází. Počet hodinových taktů potřebných na vykonání instrukce typu registr-registr se tímto snížil na pouhý 1 hodinový takt. Srovnáme-li toto se základním instrukčním cyklem řady 80C51, vidíme, že jádro nové řady mikrokontrolérů AVR dokáže poskytnout 12x vyšší výpočetní výkon při shodném hodinovém taktu. Vylepšení jdoucí ruku v ruce překladačům jazyka C jsou např. existence tří data pointerů (X, Y, Z) a několika způsobů adresování [8].

Původní řada AT90 se již nevyrábí, nyní jsou v nabídce řady:

- ATtiny
  - Pro jednoduché aplikace (paměť flash 0,5 - 8 kB)
- ATmega
  - Lépe vybavené, více integrovaných zařízení, rozšířená sada instrukcí (paměť flash 4 - 256 kB)
  - existují i speciální verze pro specifické použití, jako USB kontrolér, LCD kontrolér, či s CAN rozhraním
- ATxmega
  - Vlastnosti zvyšující výkon jako DMA, "Event System" a podpora kryptografie. (paměť flash 16 - 384 kB)
- FPSLIC™ (AVR s FPGA)

  - Hradlové pole s 5 000 až 40 000 hradly
  - Jádro AVR může běžet až na 50 Mhz

- 32 bit AVR
  - integrovaný SIMD a DSP pro zpracování videa a zvuku
  - konkurence procesorů s ARM architekturou

3.1.3 PIC

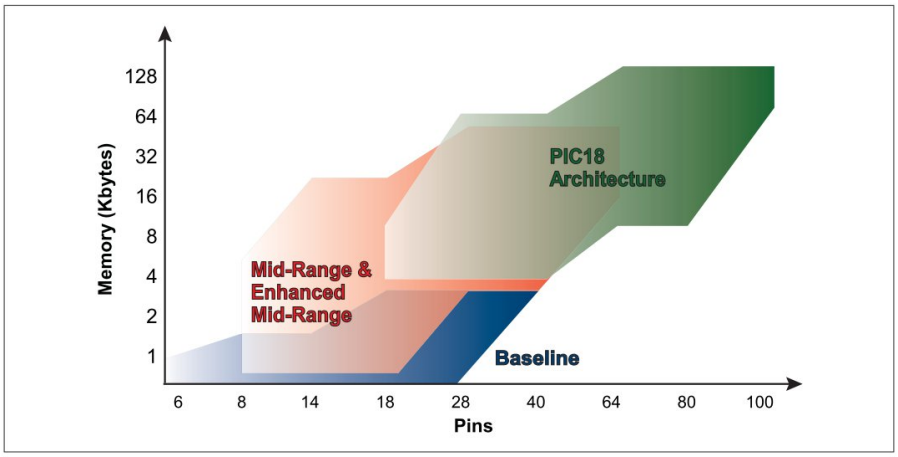
Asi největším konkurentem společnosti ATMEL na trhu s mikrokontroléry je společnost MICROCHIP se svými mikrokontroléry PIC. Struktura mikrokontrolérů PIC je podobná. Jsou to také RISC mikrokontroléry s harvardskou architekturou se širokým spektrem nabízených periferií. Mikrokontroléry PIC se vyrábí v 8-mi bitovém, 16-ti bitovém, či 32 bitovém provedení. V nabídce najdeme mikrokontroléry s podporou rozhraní jako jsou: UART, I^2C, SPI, USB, CAN, PSP a mnoho dalších.

Osmibitové mikrokontroléry PIC se nabízí ve čtyřech verzích architektury jádra:

- Baseline
- Mid-Range
- Enhanced Mid-Range
- PIC18 (dříve též označovaná high-end)

a v typových řadách: PIC10, PIC12, PIC14, PIC16, PIC17, PIC18

Tyto se opět liší svým výkonem a vybavením, každá řada je navrhována pro jiné použití. Poměrně přehledně je to vidět z následujícího grafu, kde na ose X je znázorněn počet pinů a na ose Y velikost Flash paměti.



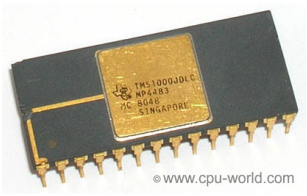
Obrázek 8: Nabídka 8 bit mikrokontrolérů PIC [2]

V nabídce jsou i 16-ti bitové (PIC24F, PIC24H, dsPIC30F a dsPIC33F) a 32-bitové mikrokontroléry PIC32.

3.1.4 MSP430

Zatímco předchozí mikrokontroléry jsou si více, či méně podobné, procesory s jádrem MSP430 od TEXAS INSTRUMENTS se již liší poměrně značně. Na rozdíl od všech výše jmenovaných s Harvardskou architekturou, MSP430 jsou mikrokontroléry s Von Neumanovou architekturou (stejná paměť pro data i program). Tyto 16-ti bitové mikrokontroléry jsou navrženy pro levné a především nízkoodběrové embbed zařízení.

Mikrokontroléry MSP430 nejsou tolik rozšířeny jako mikrokontroléry PIC, či AVR. Mohlo by se zdát, že společnost TEXAS INSTRUMENTS je na trhu s mikrokontroléry nováčkem, avšak opak je pravdou. Vzpomeňme si na první kalkulačky. Byla to právě tato firma, která v roce 1974 vyvinula první mikrokontrolér na světě (TMS1000), který pak poháněl ony slavné kalkulačky Texas Instruments. Možná si říkáte, že první mikroprocesor byl přeci *Intel 4004* (rok 1971) a máte jistě pravdu, avšak to byl mikroprocesor, který na rozdíl od mikrokontroléru neobsahuje paměť programu a další periferie, tyto obvody jsou vně mikroprocesoru a on sám provádí jen výpočty. To jen na vysvětlenou.



Obrázek 9: TMS1000

MSP je zkratka *Mixed Signal Procesor*. Mikrokontroléry MSP430 byly uvedeny na trh již v roce 1992, takže ani tato rodina není žhavou novinkou. Na trhu je několik rodin MSP430, jsou označovány jako: 3xx, 1xx, 4xx/LCD, 2xx, 5xx, 6xx/LCD. Jsou uvedeny v pořadí v jakém byly postupně uváděny na trh. Výhodou však je, že všechny rodiny mikrokontrolérů MSP430 jsou založeny na stejném jádře a díky stejné RISC instrukční sadě je případná migrace mezi rodinami MSP430 jednoduchou záležitostí.

Hlavní přednosti mikrokontrolérů MSP430 jsou:

- velmi nízká spotřeba energie (220 µA pti MIPS, 0,5 µA ve standby a 100 nA ve spánku)[9]
- plné probuzení ze standby režimu do 1 µs
- stejné jádro pro všechny rodiny
- 16 bit RISC architektura
- optimalizace pro jazyk C
- mnoho integrovaných periferií
- nízká cena (produkty *ValueLine* od $0,25)
- modely s FRAM[10] pamětí

[9] Hodnoty pro řadu ValueLine. TI demonstruje jak úsporné jsou jejich mikrokontroléry i na videu, kde napájí MSP430 pouze z enerie získané z různých druhů ovoce, avšak velmi nízkou spotřebu energie mají i konkurenční mikrokontroléry

[10] Ferroelectric Random Access Memory - nový typ paměti rychlý jako SRAM se schopností uchovánt data i po odpojení napájení

3.2 Vývojové kity a Programátory

3.2.1 Programování 8051

Pro tyto mikrokontroléry existuje mnoho profesionálních vývojových kitů, jejich ceny se pohybují od $190 výše. Většinou se jedná o desky větších rozměrů pro velké mikrokontroléry s 40-ti piny, obsahující množství různých periferií od tlačítek, přes různé displaye až po krokové motory.

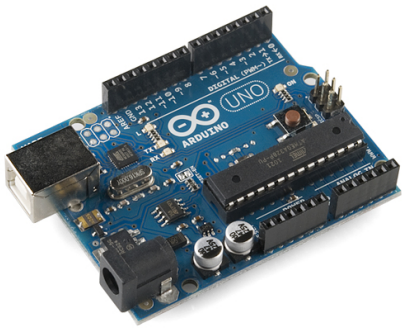
Jednoduché programátory umožňující pouze nahrávání kódu do mikrokontroléru si často kutilové vyrabí sami, cena součástek takového programátoru by se měla vejít do 400 Kč.

3.2.2 Programování AVR

Pro programování mikrokontrolérů AVR existuje nepřeberné množství všelijakých programátorů, profesionálních i doma vyrobených. Zajímavý je například slavný *PonyProg*, který podporuje i mikrokontroléry PIC. Více informací o něm najdete na Hw serveru, na kterém jsou uvedeny i schémata, ze kterých je vidět, jak je zapojení jednoduché. V podstatě jen sráží napěťové úrovně sériového portu počítače (kolem 12 V) zenerovo diodami na hodnoty kolem 5 V - používané danými mikrokontroléry. Cena je uváděna 1000 Kč.

Zjednodušené zapojení tohoto programátoru pro úzké spektrum mikrokontrolérů se dá sestrojit s náklady do 200 Kč. Nevýhodou však je, že tento programátor umožňuje pouze nahrávání programu do mikrokontroléru, bez debugingu (jednosměrná komunikace) a především fakt, že komunikuje po RS-232 sběrnici, kterou dnes najdeme v málokterém počítači. Existují i verze PonyProgu s USB portem, ovšem jsou již dražší.

Asi nejznámějším vývojovým kitem pro AVR je *Arduino* - projekt, který se v poslední době těší velké oblibě. Důkazem je stále se rozrůstající již tak velká komunita kolem Arduina. Je to opensource vývojový kit, díky čemuž vzniká mnoho klonů a vylepšení tohoto kitu. Tím, že je opensource celý návrh (zdarma dostupné schémata, tištěné spoje, CAD výkresy) si jej může každý postavit sám. K Arduinu je k dispozici množství různých doplňků, které se k němu jen jednoduše připojí, například jsou to různé displaye, bluetooth, či Ethernet rozhraní. Arduino se programuje pomocí programovacího jazyka Wiring, což je zjednodušená verze jazyka C, tak aby Arduino mohl programovat i člověk bez předchozích zkušeností s programováním MCU.



Obrázek 10: Arduino

Cena Arduina se pohybuje okolo 600 - 700 Kč, sice s poměrně slušným mikrokontrolérem, ale součástí balení kupodivu není USB kabel, což je při této ceně přinejmenším zarážející.

3.2.3 Programování PIC

Jak již bylo naznačeno v sekci o programátorech AVR, i pro mikrokontroléry PIC existuje mnoho rozmanitých programátorů a vývojových kitů. Například výše zmíněný *PonyProg* podporuje i mikrokontroléry PIC. Schémat na výrobu různých programátorů podomácku je také spousta.

Z vývojových kitů stojí rozhodně za zmínku PICkit. Jedná se o oficiální programátor s debugerem přímo od společnosti MICROCHIP. K počítači se připojuje USB kabelem. Pro připojení k mikrokontroléru slouží malý konektor, do kterého se dá napojit jedna z dodávaných vývojových desek, či rovnou zasunout dráty vedoucí např. do nepájivého pole, či hotového zařízení.

Obrázek 11: PICkit 3

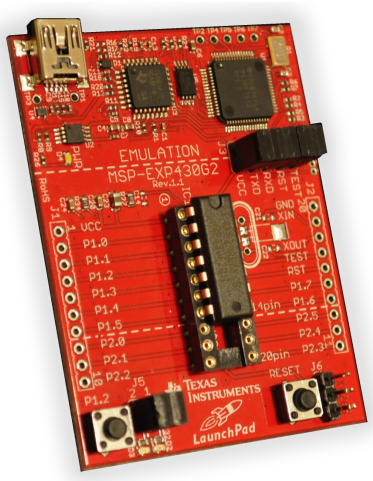
PICkit je takzvaný *in-cruit debugger*, který umožňuje debugovat mikrokontrolér přímo ve skutečném zapojení. Dokáže pozastavit periferie, když se při vykonávání kódu dojde na *breakpoint*. Také umí detekovat přepětí, či zkrat.

Cena samotného PICkitu 3 se pohybuje okolo 900 Kč a verze, která obsahuje navíc i vývojovou desku s mikrokontrolérem PIC18F45K20 stojí 2 400 Kč

Zajímavostí je, že vznikají i verze Arduina pro mikrokontroléry PIC, zdrojové kódy sice nejsou kompatibilní, ale mají kompatibilní vývody, tudíž je možné využívat široké množství doplňků určených pro Arduino i prostřednictvím mikrokontrolérů PIC.

3.2.4 Programování MSP430

Pochopitelně i pro MSP430 existuje škála různě vybavených vývojových kitů. Nevím však o tom, že by někdo podomácku vyráběl programátor pro tyto mikrokontroléry. Proč by to také dělal, když poměrně nedávno (v roce 2010) vydaný LaunchPad stojí $4,30 (přibližně 90 Kč). Za tuto skvělou cenu zákazník obdrží:



Obrázek 12: LaunchPad

- LaunchPad - Vývojový kit (MSP-EXP430G2) (programátor, debuger, USB - sériový převodník)
- USB propojovací kabel
- 2x MSP430 mikrokontroléry MSP430G2211 a MSP430G2231
- 32 kHz hodinový krystal (CL: 12.5pF +/-20ppm)
- 2 PCB konektory - volitelná montáž na LaunchPad

3.3 Vývojové prostředí

Pro každou rodinu mikrokontrolérů existuje řada programů pro jejich snadné programování, některé jsou zdarma, jiné za peníze. Někdy jsou verze zdarma s nějakým omezením, například bývá omezeba maximální velikostí kódu, či bývá zdarma pouze na určitou zkušební dobu.

Dobrá zpráva ale je, že existují pluginy pro opensource IDE *Eclipse* umožňující vývoj všech zmíněných rodin MCU v tomto skvělém vývojovém prostředí. Eclipse je zdarma a je napsáno v Javě, tudíž je bez potíží multiplatformní.

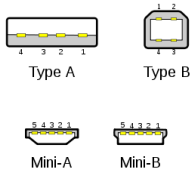
Díky tomu vývojové prostředí při výběru MCU platformy nehraje podstatnou roli.

3.4 Komunikace s PC

3.4.1 USB rozhraní

USB rozhraní je dnes běžnou součástí spotřební elektroniky připojitelné k počítači. Dá se říci, že postupem času vytlačilo v minulosti používané standardy (Sériový port RS-232, či port paralelní), které bychom dnes na osobním počítači hledali jen z těží.

Koupit dnes notebook s klasickým sériovým rozhraním je téměř nemožné a proto jsou nuceni i výrobci a uživatelé speciálních aplikací postupně přecházet na USB.



USB rozhraní používá standardizované konektory a 4 vodiče. Dva vodiče zprostředkovávají napájení (5 V, do 500 mA) a další dva slouží pro vysokorychlostní přenos dat (až 480 Mbit/s u USB 2.0)[11]. Vzdálenost přenosu je do 5-ti metrů.

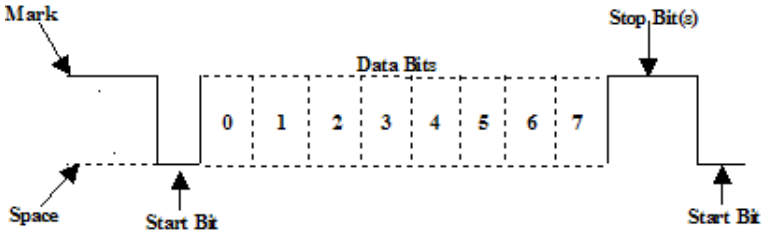
Některé mikrokontroléry mají podporu USB rozhraní, takové mikrokontroléry je vhodné použít tehdy, je-li potřeba komunikovat s počítačem vysokou přenosovou rychlostí, nebo také v případě komunikace s periferiemi vybavenými USB rozhraním. Jejich nevýhodou je vysoká cena v porovnání s modely bez USB rozhraní.

3.4.2 Sériové rozhraní

Dříve se běžně používal sériový port i na osobních počítačích. Dnes je to výjimka. Mikrokontroléry ale používají sériové rozhraní zcela běžně pro komunikaci s jinými zařízeními. Tyto zařízení mohou být například mikroprocesory, displaye, EEPROM paměti, A/D převodníky, atp. Většinou se k těmto účelům využívá sériové sběrnice SPI, či I^2C. Často se také využívá rozhraní UART, či USART.

Protokol kompatibilní s UARTem používá i sériové rozhraní osobních počítačů (RS-232). Mikrokontroléry ale používají jiné napěťové úrovně (obvykle 5 V, nebo 3,3 V TTL), než

[11] Nejnovější standard USB 3.0, který zatím není moc rozšířený má maximální rychlost až 5 Gbit/s a komunikuje po čtyřech datových vodičích (celkem 6 vodičů i s napájením)



Obrázek 13: Sériová komunikace [1]

používá standard RS-232 (kde napětí od +5 do +15 V je logická 1 a od -5 do -15 je logická 0, PC používá 12 V úrovně). Za účelem propojit tyto napěťově nekompatibilní rozhraní se používá speciálních obvodů, tzv. převodníků. Např. *MAX232* firmy MAXIM, nebo *TC232* firmy MICROCHIP.

Komunikace přes UART probíhá asynchronně, to znamená, že zde nenajdeme vodič se synchronizačním signálem, jaký má, například rozhraní SPI. Komunikace probíhá po dvou vodičích (jeden pro příjem dat a druhý pro odesílání) s předem určenými parametry, jako je rychlost a počet bitů ve slově. Odesílání jednoho slova (obvykle 8 bitů) zahajuje tzv. *start bit*, což znamená uvést vodič pro odesílání do logické nuly po dobu jednoho bitu. Díky tomu, že odesílající i přijímající strana zná rychlost přenosu, zná i dobu po kterou je přenášen jeden bit. Přijímací strana čeká na start bit, po jehož přijetí začne číst hodnoty ve stejných intervalech v jakých je vysílací strana posílá a nakonec přijme i *stop bit*[12], což je logická 1, ten slouží pro oddělení jednotlivých slov (po něm zas přijde start bit v logické 0). Někdy se také posílá i paritní bit, který slouží jako primitivní detekce chyb přenosu. Celá tato situace je znázorněna na obr. 13.

Hardwarovou podporu UARTu má mnoho mikrokontrolérů, rozhodně jich je více, než těch s HW podporou USB, navíc i na mikrokontroléru bez podpory UARTu se dá poměrně jednoduše naprogramovat softwarová implementace sériového protokolu.

3.4.3 USB ↔ RS-232 převodníky

Zatím stále nejrozšířenějším způsobem připojení mikrokontroléru k USB portu počítače je připojení přes převodníky z USB na sériové rozhraní. Buď je zařízení s mikrokontrolérem vybaveno sériovým portem s 9-ti pinovým konektorem a připojuje se k USB přes převodník podobný tomu z obrázku 14, nebo je převodník ve formě integrovaného obvodu použit přímo v zapojení s mikrokontrolérem a celé zařízení se připojuje již přímo na USB sběrnici.

Obrázek 14: USB ↔ RS-232 převodník

Asi nejvíce oblíbené jsou mezi vývojáři převodníky od společnosti FTDI. Jedním z nich je například FT232H - převodník, který umí komunikovat nejen s UARTem, ale i s JTAG, I2C, SPI, či bit-bang. Na UART rozhraní komunikuje až do rychlosti 12 Mbaud.

Takové převodníky jsou obsaženy i v samotných vývojových kitech.

[12] Stop bitů může být i víc, než jeden

3.4.4 Softwarová implementace USB

Krom již zmíněných řešení, jak připojit mikrokontrolér k USB (hardwarové USB, USB-sériový převodník) je tu ještě jedna možnost a to softwarová implementace USB protokolu. Jedním z projektů, který se zabývá SW implementací USB na mikrokontrolérech AVR je opensource projekt V-AVR (dříve AVR-USB) [5]. Jedná se o čistě softwarovou implementaci „low-speed" USB 1.1 pro mikroprocesory AVR s alespoň 2 kB flash a 12 Mhz.

Výhody jsou zřejmé:

- USB podporují jen některé mikrokontroléry[13]
- možnost nastavit vlastní identifikátory Vendor-ID a Product-ID
- není potřeba dalších obvodů (USB-Sériové převodníky, řadiče)

Nevýhody plynou ze samé podstaty softwarové implementace:

- vyžaduje větší výpočetní výkon
- zabírá 2 vstupy přerušení
- zabírá 1,5 kB paměti (u projektu V-USB)

Z dnešního pohledu již není tato možnost tak atraktivní, jak tomu bylo před pár lety, kdy bylo obtížné sehnat mikrokontrolér s hardwarovou podporou USB, ale i tak je to zajímavá možnost a je dobré o ní vědět.

3.5 Hodiny reálného času

3.5.1 Hodinový krystal

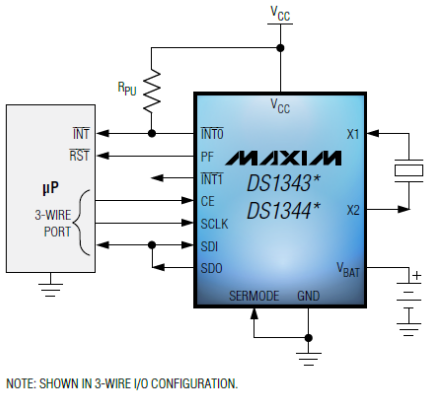
Krystal je pasivní elektrotechnická součástka používaná v elektronických obvodech jako rezonátor s velmi přesnou a stabilní rezonanční frekvencí. Pro potřeby počítání skutečného času (sekundy, hodiny...) se používají krystaly s frekvencí 32 kHz, neboli 32 768 Hz. Není to 32 000 Hz, jelikož jsou to binární kilohertze, kde 1 kHz má 1024 Hz. Je to tak záměrně, aby se dalo pomocí násobků dvou získat přesný časový údaj odpovídající realitě. Díky tomu vhodným vynásobením získáme přesně jednu sekundu:

$$\frac{1}{32768}s \cdot 4 \cdot 256 \cdot 32 = 1s$$

[13] MCU s hardwarovou podporou USB bývají většinou v provedení SMD, což může komplikovat vývoj na nepájivém poli

3.5.2 Obvody reálného času (RTC)

V případech, kdy je potřeba mít přesný zdroj reálného času a není vhodné implementovat hodiny reálného času přímo v mikrokontroléru se používají obvody reálného času. Obvod reálného času slouží k udržování aktuálního a přesného času i v případě ztráty napájení (mají k dispozici záložní zdroj energie). Mají velmi malou spotřebu energie (zpravidla stovky nanoampér), dokáží komunikovat s okolím (nejčastěji pomocí SPI, I^2C, či 3-wire) a poskytnout informace jako aktuální čas, datum, den v týdnu a mnohem víc. Na obr. 15 je příklad zapojení obvodu RTC s mikrokontrolérem.



Obrázek 15: Příklad zapojení RTC obvodu

3.6 Výběr vhodné platformy

3.6.1 Požadavky

Požadavky při výběru mikrokontrolér byly:

- dostatečně velká paměť pro uchování velkého množství hodnot (16 kB Flash viz. 4.3.3)
- možnost měnit obsah vnitřní paměti programově
- vhodný pro programování v jazyce C
- časovač pro běh RTC
- A/D převodník pro měření teploty
- integrovaný teplotní senzor výhodou
- UART pro komunikaci s počítačem
- Příznivá cena

3.6.2 Srovnání mikrokontrolérů

V následující tabulce je vidět srovnání mikrokontrolérů s co nejpodobnějším vybavením. Všechny vybrané typy mají 16 Kb Flash paměť pro program, do které mohou programově zapisovat. Všechny mají hardwarový UART nebo USART, 16-ti bitové časovače a 10-ti bitový A/D převodník. Zastoupeny jsou mikrokontroléry PIC a to 16-ti bitový *dsPIC33FJ16GP102* a 8-mi bitový *PIC18F24J10*, MSP430 zde má 16-ti bitového zástupce *MSP430G2553*, za mikrokontroléry AVR je tady 8-mi bitová *ATmega168A* a podobných parametrů dosahuje i *AT89C5115* s jádrem na bázi 8051, avšak ten jediný není v pouzdru DIP.

MCU	Arch.	SRAM (B)	16 bit časovač	čidlo teploty	ADC kanálů	freq.	komp.	Cena $
MSP430G2553	16	512	2	Ano	8	16	8	1,47
dsPIC33FJ16GP102	16	1024	3	Ne	6	16	3	2,48
PIC18F24J10	8	1024	2	Ne	10	10	2	1.47
ATmega168A	8	1024	3	Ano	8	20	6	2,3
AT89C5115	8	512	2	Ne	8	20	0	5,84

Ceny platí 9.4.2012 na avnet.com pro množství jeden kus a jsou v amerických dolarech

Jak je vidět z tabulky i když jsem se snažil vybrat mikrokontroléry tak, aby byly co nejpodobnější, stejně jsou mezi jejich parametry drobné rozdíly a nemalé rozdíly jsou i v ceně. Z toho vyplývá, že při každém návrhu zařízení je velmi důležité správně zvolit vhodnou platformu. Každý mikrokontrolér se hodí na něco jiného a na to je dobré pamatovat již od samého začátku vývoje.

3.6.3 Srovnání vývojových kitů

S ohledem na cenu a schopnosti vývojových kitů jednoznačně vede Texas Instruments se svým LaunchPadem za $4,30. Dá se použít přímo pro programování mikrokontrolérů MSP430 ValueLine v patici, stejným způsobem jako se používají běžné programátory. Existuje k němu mnoho doplňků podobně jako pro Arduino[14]. Nabízí také možnost připojit ho pomocí vodičů do nepájivého pole a programovat tak libovolný mikrokontrolér MSP430, který má *Boot-Strap Loader*[15] - stejným způsobem, jakým se používá PICkit. Dokonce nabízí i volitelné rozšíření přidáním konektoru pro programování vývojových destiček eZ430, což jsou miniaturní vývojové destičky s MSP430 procesory, které se jinak programují miniaturním programátorem velikosti USB klíče.

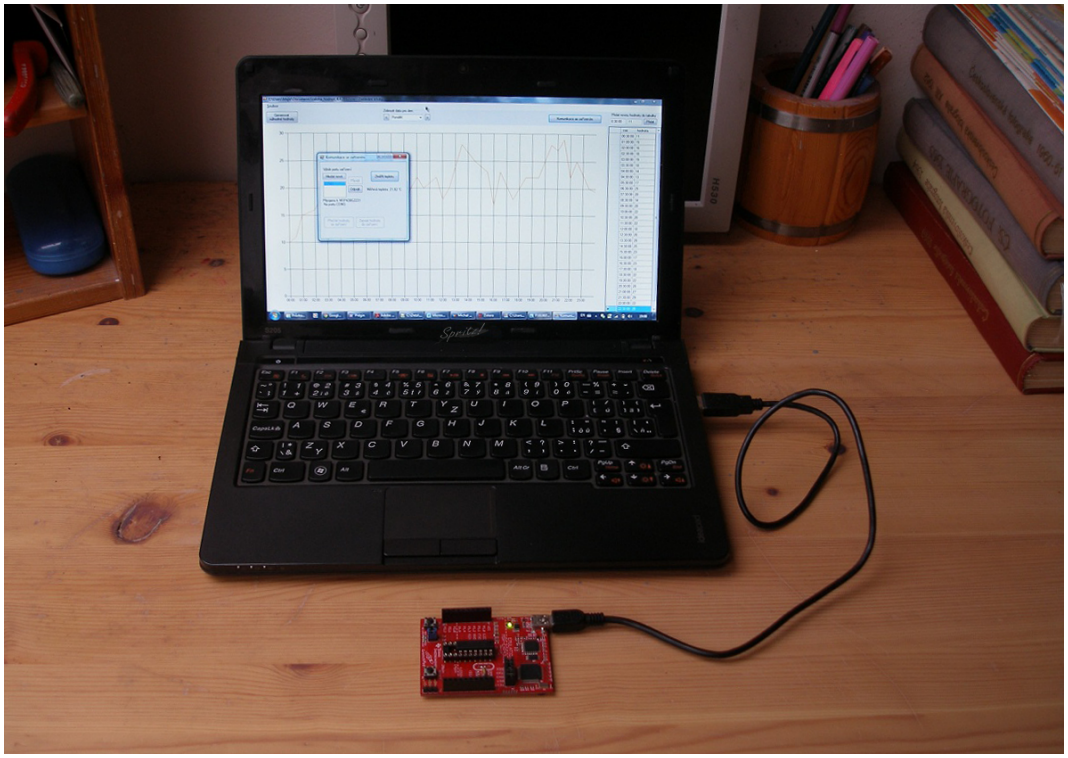
Obrázek 16: eZ430-T2012

3.6.4 Shrnutí

Po zhodnocení všech aspektů jsem došel k závěru, že nejlépe vyhovuje mým požadavkům vývojový kit *LaunchPad* od Texas Instruments a k němu mikrokontrolér *MSP430G2553*, který splňuje všechny

[14] např. kapacitní Booster Pack umožňující dotykové ovládání za $10

[15] program uložený v části Flash paměti zařízení a umožňuje naprogramovat do paměti kód přijatý přes UART, či USB



Obrázek 17: Výsledné propojení MSP430 s PC aplikací

požadavky, ba je dokonce mírně převyšuje. Velmi podstatným faktorem je také cena celého řešení, která nepřesáhne 150 Kč.

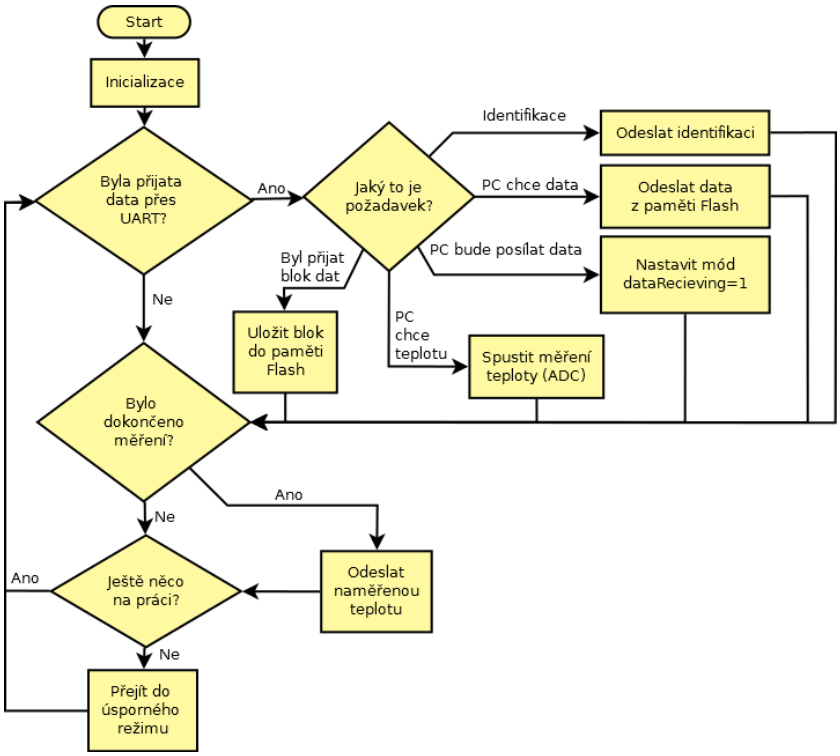
Na obrázku 17 je vidět jak vypadá výsledné zapojení mé práce. Počítač s obslužnou aplikací propojený s launchpadem. To je celé co je potřeba. Nemusel jsem vyvíjet plošný spoj s USB převodníkem a dalšími periferiemi. Při ceně launchpadu $4,3 se může tento použít nejen pro programování mikrokontroléru, ale i pro výslednou aplikaci. Vývoj vlastního zařízení by se v tomto případě nevyplatil ani ekonomicky. Pro řízení například topného tělesa stačí připojit k launchpadu pouze spínací prvek. Například tranzistorem spínané relé, triak, nebo jiný výkonový spínací prvek.

4 Programové vybavení mikrokontroléru

4.1 Vývojové diagramy

Vývojové diagramy slouží k zjednodušenému znázornění struktury počítačových programů. Zde publikované UML diagramy znázorňují programové vybavení mikrokontroléru. Celkem jsou čtyři. První znázorňuje hlavní programovou smyčku a ty zbylé tři znázorňují obsluhu přerušení (časovače, I/O portu a A/D převodníku).

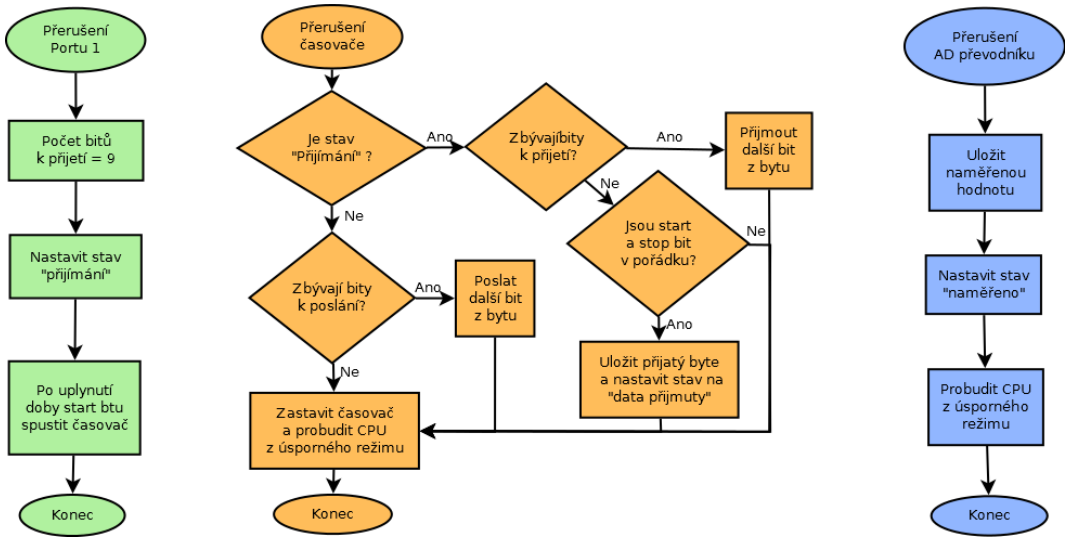
4.1.1 Hlavní programová smyčka



Obrázek 18: Hlavní programová smyčka - vývojový diagram

Mikrokontrolér vždy vykoná svoji práci a poté se přepne do režimu snížené spotřeby. Z tohoto režimu ho probudí vždy obsluha přerušení. Například přerušení portu, které nastane, když počítač začne mikrokontroléru posílat data, způsobí probuzení mikrokontroléru z režimu snížené spotřeby a díky tomu se opět začne provádět kód hlavní smyčky od začátku. Pokud by se po vykonání práce stalo, že mezitím vzniklo další přerušení, mikrokontrolér se neuspí, ale znovu opakuje vykonávání hlavní smyčky, pro zpracování nové informace. Toto chování zajišťuje podmínka „Ještě něco na práci?".

4.1.2 Přerušení



Obrázek 19: Vývojový diagram přerušení

4.2 Komunikace s PC

První věc, kterou potřebujeme je funkční komunikace mezi počítačem a mikrokontrolérem pomocí USB portu. Díky použití vývojového kitu *LaunchPad (MSP-EXP430G2)* odpadají starosti s řešením vlastního zapojení s USB ↔ UART převodníkem, jelikož ten je již součástí LaunchPadu. Vzhledem k ceně celého kitu pod 90 Kč by se ani nevyplatilo pokoušet se o vlastní výrobu. Dovedu si dokonce představit použití tohoto kitu jako část potenciálního výsledného produktu.

4.2.1 Hardwarový UART

Původně jsem zamýšlel použití mikrokontroléru s hardwarovým UARTem. Konkrétně mám objednaný mikrokontrolér *MSP430G2553*, bohužel po komplikacích s první objednávkou, čeká nová objednávka na doplnění zásob, které má proběhnout až 11.4.2012.

Při použití hardwarového UARTu by se kód posílání a přijímání dat přes UART omezil na pouhých několik řádků. Ale vzhledem k tomu, že jsem neměl k dispozici mikrokontrolér s HW UARTem, uchýlil jsem se k alternativnímu řešení a to implementaci Softwarového UARTu na mikrokontroléru *MSP430G2231*, který je standardně dodáván k LaunchPadu.

4.2.2 Softwarový UART

Funkce softwarového UARTu je naznačena v UML diagramech na obr. 18 a 19.

Při odesílání dat přes UART se nejdřív uloží byte dat pro odeslání do proměnné *TXByte* a spustí se funkce *Transmit()*. Ta do proměnné *TXByte* přidá Start a Stop Bit, nastaví počet bitů k odeslání

na 10 (Start Bit, 8 datových a Stop Bit), nastaví časovač na dobu jednoho bitu, nastaví výstup na 1 (nečinný stav UARTu) a povolí přerušení časovače.

Výpis 7: Spuštění posílání bytu přes UART

```
void Transmit() {
  while(isReceiving);  // Počkáme, až se dokončí právě probíhající příjem dat
    TXByte |= 0x100;  // Přidáme Stop bit (logická 1 na konec)
    TXByte = TXByte << 1; // Přidáme Start Bit (logická 0 na začátek)
    BitCnt = 0xA; //  Nastavíme počet bitů k odeslání na 10
    CCTL0 = OUT;  // Nastavíme výstup na 1 (aby mohl být poté rozeznán příchod
        Start Bitu na straně PC)
    TACTL = TASSEL_2 + MC_2;  // SMCLK, časovač bude neustále inkrementovat
    CCR0 = TAR; // Inicializujeme porovnávací registr
    CCR0 += Bit_time; // Přidáme čas čekání do prvního bitu
    CCTL0 =  CCIS0 + OUTMOD0 + CCIE;  // Nastavit vstup + výstup časovače +
        povolit přerušení
    while ( CCTL0 & CCIE ); // Počkáme na dokončení odesílání bytu
}
```

Jakmile uběhne nastavená doba jednoho bitu, dosáhne TAR hodnoty porovnávacího registru CCR0 a vyvolá se přerušení časovače. V obsluze přerušení časovače se v tu chvíli přičte čas pro další bit do registru CCR0 a změní se stav výstupu na hodnotu právě posílaného bitu. V případě, že již byly všechny bity poslány, vypne se časovač a zakáže se jeho přerušení. Při posílání dalšího bytu se celá situace opakuje.

Výpis 8: Odesílání bytu přes UART

```
#pragma vector=TIMERA0_VECTOR __interrupt
void Timer_A (void) // Obsluha přerušení časovače
{
if(!isReceiving) // Režim posílání dat
{
  CCR0 += Bit_time; // Přidat čas pro bit na výstupu
  if ( BitCnt == 0) // Již byly odeslány všechny bity
  {
    TACTL = TASSEL_2; // vypneme časovač SMCLK (úspora energie)
    CCTL0 &= ~ CCIE;  // Zakážeme přerušení časovače
  }
  else // Stále je co posílat
  {
    if (TXByte & 0x01) // Posílaný bit je 1
      CCTL0 &= ~ OUTMOD2; // Nastav 1
    else       //  Posílaný bit je 0
      CCTL0 |=  OUTMOD2;  // Nastav 0
    TXByte = TXByte >> 1; // Příště poslat další bit
    BitCnt --; // Zbývá o bit méně k poslání
  }
}
else ...// kód pro přijímání dat
```

Přijímání probíhá obdobným způsobem. Jak je vidět z předchozí ukázky kódu, podle příznaku *isRecieving* se pozná jestli se bude přijímat, či posílat. Rozdíl je akorát ve spouštění přijímání, to provádí obsluha přerušení portu 1. Tím, že se na portu objeví Start Bit vznikne přerušení a jeho obsluha zařídí vše potřebné, podobně jako funkce *Transmit()*, jen s malým rozdílem a to, že první čas časovače je poloviční, to proto, aby se čtení z portu trefilo přesně doprostřed každého bitu a tím bylo zajištěno jeho spolehlivé přečtení.

Jak je vidět, softwarový UART není složité udělat. Díky vhodnému využití časovače neplýtvá zbytečně procesorovým časem a energií. Má to však svá úskalí. Jedním z nich je, že mikrokontrolér *MSP430G2231* má jen jeden časovač, který je nyní použit softwarovým UARTem. Nelze ho tedy použít společně s hodinovým krystalem na realizaci hodin reálného času. Reálný čas bychom potřebovali při zpracovávání časové křivky.

Ze začátku jsem se obával o stabilitu DCO (Digitally-Controlled Oscilator), aby byla dostatečná pro udržení přesného časování potřebného pro softwarový UART. V případě, že by nebyla, bylo by nutné pořídit krystal o vhodné frekvenci (například 14,7456 Mhz), aby bylo možné děliteli 2 dosáhnout přesné frekvence přenosu UART (obdobně jako při volbě frekvence krystalu pro skutečný čas). Mé obavy se naštěstí ukázaly zbytečné a potvrdilo se, že udávaná úchylka 2 µs při 1 Mhz skutečně platí. Během testování jsem totiž nenarazil na chybu přenosu pri rychlosti 9,6 kbaud. [6]

4.3 Úložiště hodnot

4.3.1 Možnosti ukládání

Jakmile máme funkční komunikaci přes UART, je potřeba přijímané hodnoty někam ukládat. Možností je více, mohou to být externí paměti, komunikující s mikrokontrolérem po sériové sběrnici, či interní paměti mikrokontrolérů. Například se dá jednoduše komunikovat s SD, či MMC paměťovou kartou pomocí SPI protokolu. Tyto paměťové karty jsou dnes velice levné a nabízejí velké množství úložného prostoru.

Mikrokontroléry MSP430 mají výhodu, že používají Von Neumanovu architekturu. Díky tomu je program uložen ve stejné paměti, do které můžeme ukládat i svá uživatelská data. Většinou se jedná o paměti typu Fash. Texas Instruments však nabízí i řadu těchto mikrokontrolérů s pamětí FRAM, která si uchovává uložené informace i po odpojení napájení, stejně jako zmíněná Flash technologie, avšak jsou extrémně rychlé. Rychlostí dosahují stejných, jako paměti SRAM, které ovšem po odpojení napájení ztratí uložená data. Tudíž je to rozhodně zajímavá novinka.

4.3.2 Flash paměť

Při předpokládaném využití mikrokontroléru jako termostatu, nepotřebujeme extra rychlé paměťové úložiště a bohatě nám postačí paměť typu Flash. Zápis do paměti Flash je z pohledu mikrokontroléru pomalý (desítky milisekund). Zajímavé je, že do paměti je možné zapsat pouze nuly. Stejně, jako kdybychom stříleli do papíru, opakovaným střílením na stejné místo papír neobnovíme, musíme vyměnit celý papír a nastřílet díry jinde, pokud to potřebujeme. Podobně funguje paměť Flash, ta se dá po

blocích (obvykle 512 b) mazat - tzn. obnovit všude jedničky. Poté můžeme zapsat nuly na potřebná místa a jsme hotovi. Avšak při požadavku změnit ba pouze jeden bit z nuly na jedničku je potřeba zazálohovat celý blok dat, ten následně vymazat a až poté můžeme uložit aktualizovaný blok dat zpět do paměti. Do paměti Flash se přistupuje pomocí data pointerů (ukazatelů). Pro ukázku uvádím krátký kód mazání segmentu Flash paměti.

Výpis 9: Vymazání segmentu Flash paměti

```
1 void erase_segment(int address)
2 {
3   int *Flash_ptr;                // Ukazatel na Flash segment
4   Flash_ptr = (int *)address;    // Nastavení ukazatele na požadovanou adresu
5   FCTL1 = FWKEY + ERASE;         // Nastavit mód mazání
6   FCTL3 = FWKEY;                 // Odemknout zámek
7   *Flash_ptr = 0;                // Zápisem vymažeme Flash segment
8   FCTL3 = FWKEY + LOCK;          // Zamknout zámek
9 }
```

Při zápisu dat do Flash Paměti, je velice důležité zakázat veškerá přerušení, protože kdyby během zápisu, či mazání bylo vyvoláno přerušení, narušilo by to integritu dat. Bohužel mikrokontrolér *MSP430G2231* nemá k dispozici dostatečné množství paměti pro uložení všech hodnot z křivky, tudíž tato funkce není v praxi implementována.

4.3.3 Výpočet velikosti paměti

Pro zvolení správného mikrokontroléru je potřeba vypočítat velikost paměti potřebnou pro uložení hodnot. Den má:

$$24 \cdot 60 \cdot 60s = 86400s$$

Tato hodnota se vejde do **15-ti bitů**. K tomu 7 dní v týdnu, to jsou další **3 bity** a k tomu hodnota teploty v rozsahu 200 °C na dvě desetinná místa, to je dalších **15 bitů**. Celkem potřebujeme **33 bitů** na jednu hodnotu. Do jednoho bloku Flash paměti, který má 512 b, se tedy vejde 15 hodnot.

$$\frac{512b}{15b + 15b + 3b} = 15$$

Aby bylo možné uložit hodnoty pro každých pět minut, každý den, budeme potřebovat:

$$\frac{24 \cdot \frac{60}{5} \cdot 7}{15} = 134,4 = 135\,bloků = 68\,kb = 8,5\,kB$$

Potřeba je 135 bloků po 512 b, to znamená 8,5 kB volné paměti pro data, k tomu je ještě potřeba místo pro kód (cca 2 kB), tudíž je potřeba mikrokontrolér s minimálně 10,5 kB Flash. V řadě *ValueLine* jsou mikrokontroléry s 16 kB pamětí. Zvolil jsem proto mikrokontrolér *MSP430G2553*, vybavený právě 16 kB pamětí a dalšími periferiemi potřebnými k realizaci termostatu. Bohužel se mi ho nepodařilo získat včas, viz. 4.2.1.

4.4 Měření teploty

4.4.1 Interní teplotní senzor

Zvolený mikrokontrolér MSP430 obsahuje zabudovaný teplotní senzor. Podle měřené teploty se na jeho výstupu dá naměřit různé napětí. Výstup teplotního senzoru tedy připojíme na vstup A/D převodníku a ten nám z napětí udělá digitální číslo vhodné ke zpracování mikrokontrolérem.

4.4.2 A/D Převodník

Jak již bylo zmíněno k A/D převodníku se připojí teplotní senzor a měření poté bude probíhat spuštěním ADC, který po dokončení měření vyvolá svůj přerušení. V obsluze přerušení ADC se poté může použít naměřená hodnota. Mělo by to být patrné z obr. 19 a obr. 18. Nutno podotknout, že měřená hodnota je hodnota napětí teplotního senzoru, tato hodnota se musí ještě vhodně přepočítat na teplotu, pokud ji potřebujeme ve formátu čitelném pro člověka.

4.5 Hodiny reálného času

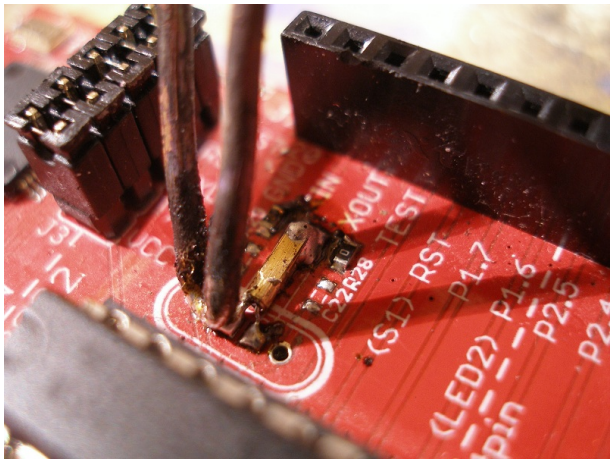
Jelikož jsem spoléhal na obvod s hardwarovým UARTem a dvěma časovači viz. 4.2.1, počítal jsem s implementací hodin reálného času v programu mikrokontroléru. Bohužel to na obvodu s jedním časovačem použitým pro softwarový UART realizovat nešlo.

Návrh je však prostý, po instalaci hodinového krystalu, kterou jsem absolvoval (obr. 20)[16] by se tento použil jako zdroj signálu ke generování přerušení časovače každou sekundu (po vydělení frekvence). Každou sekundu by se inkrementoval počet sekund, každou minutu zas počet minut atp. s podporou dne v týdnu (0-6).

Zálohování času při výpadku napájení by se dalo řešit přidáním druhého zdroje napájení (záložní baterie). Oba zdroje napájení by se daly zcela jednoduše připojit každý přes svojí diodu (aby nedošlo k průchodu proudu mezi nimi) na napájecí pin mikrokontroléru. Současně by se přivedl hlavní zdroj napájení na jeden z nevyužitých vstupních pinů mikrokontroléru pro detekci ztráty napájení. Mikrokontrolér by při ztrátě napětí na sledovaném pinu přešel do úsporného režimu, kdy jsou v činnosti jen hodiny reálného času. V tomto režimu je spotřeba řady *ValueLine* pouhých 0,7 µA [3]. Avšak spotřeba mikrokontrolérů MSP430 je natolik nízká i při běžném provozu, že za lepší řešení považuji napájet celé zařízení pouze z baterií - záleží na konkrétní aplikaci.

Synchronizace času byla zamýšlena pomocí obslužného programu na PC. Při nahrávání hodnot do zařízení by měl uživatel také možnost nastavit správný čas v zařízení.

[16] za povšimnutí stojí, že u krystalu nejsou připojeny kondenzátoy, to je tím, že MSP430 umí některé typické kapacity připojit softwarově



Obrázek 20: Pájení krystalu na LaunchPad

4.6 Regulace teploty podle křivky

Vývoj tohoto zařízení byl směřován k vytvoření zařízení, které by dokázalo přijmout a uložit křivku hodnot (například teplot) v čase do své paměti. Jako jedno z možných využití takového zařízení se nabízí regulace teploty podle počítačem zadané křivky.

Návrh takového zařízení by zahrnoval všechny doposud zmiňované funkce a mohl by vypadat následovně:

- Pomocí počítačového programu se vytvoří křivka teploty
- Pres UART se z počítače uloží křivka do flash paměti a také se zkalibruje čas v zařízení
- V opakovaném, vhodně zvoleném intervalu (např. 1 s) se:
  - načte z flash paměti předchozí a následující hodnota teploty s časem (dle aktuálního času)
  - z předchozí a následující hodnoty se vypočítá požadovaná teplota v aktuálním čase
  - pomocí teplotního čidla a A/D převodníku se změří aktuální teplota
  - z rozdílu aktuální a požadované teploty se zjistí jak upravit výkon vytápění
  - upraví se výkon vytápění na požadovanou hodnotu

Při správně nastavené regulaci získáme termostat, který přesně kopíruje teplotní křivku.

Využití by se jistě našlo více. Stejně tak i podporované funkce by se daly dle libosti přidávat. Zařízení by například mohlo teplotu naopak zaznamenávat. Zajímavé na tom celém je, že si vystačíme s jediným mikrokontrolérem, jeho napájením a hodinovým krystalem (a pochopitelně ještě zařízením na ovládání vytápění v tomto případě).

5 Závěr

Cílem práce bylo vytvořit počítačovou aplikaci umožňující jednoduchou tvorbu křivky hodnot v čase s možnostmi tuto křivku zálohovat do souboru v počítači a také ji přenášet mezi počítačem a mikropočítačovým systémem. Za cíl jsem si kladl vytvořit také funkční termostat pracující s vytvořenou křivkou pomocí mikrokontroléru MSP4305531.

Požadavky práce byly splněny. Počítačový program úspěšně komunikuje s mikrokontrolérem. Při odeslání požadavku po sériovém portu mikrokontroléru, mikrokontrolér odpoví na požadavek patřičným způsobem. Umí se identifikovat, či změřit a vrátit změřenou teplotu, kterou následně počítačový program zobrazí. Avšak mnou zvolený cíl byl splněn jen částečně. Počítačová aplikace umožňuje jednoduchou tvorbu a úpravy křivky, kterou je následně možno zálohovat do počítače. Avšak již neumožňuje ukládat křivku do mikrokontroléru. A tudíž ani podle křivky regulovat teplotu. Je to způsobeno komplikacemi při objednávce mikrokontrolérů MSP4305531 ze zahraničí, který se u nás neprodává.

V České republice není mnoho obchodů, kde by se daly mikrokontroléry MSP430 koupit. Nejvíce jich mají na www.farnell.com, ovšem konkrétně tento typ nebyl ani v jejich nabídce. Z toho důvodu jsem mikrokontrolér objednal z Ameriky přímo od Texas Instrumets, nastaly však komplikace se špatně vyplněným políčkem „Company Name", žádnou firmu nemám, tak jsem to tam i uvedl. Musel jsem po několika dnech provést objednávku znova a na místo názvu firmy uvést jméno univerzity, kde studuji (ČZU). Bohužel mezitím jim došly skladové zásoby a nové doplnění skladu mělo proběhnout až 11. 4. 2012 a také se tak stalo.

Z tohoto důvodu jsem se rozhodl implementovat alespoň zjednodušenou funkcionalitu mikropočítačového systému s použitím mikrokontroléru *MSP430G2231*, který je standardně dodáván k Launchpadu ale má malou flash paměť, jeden časovač a nemá hardwarový UART, tudíž funkce jako ukládání hodnot do Flash paměti, hodiny, či regulace nešly implementovat. Místo toho jsem na něm úspěšně zprovoznil softwarovou emulaci UARTu, přes který mikropočítač komunikuje s počítačem. Umí změřit teplotu, kterou poté počítač zobrazuje. Návrh programového vybavení mikropočítače jsem provedl v kapitole 4, kde popisuji veškeré kroky potřebné k sestrojení termostatu. Aplikaci pro PC jsem dokonce naprogramoval a odladil, základní program do mikrokontroléru také.

Ačkoli mají mikrokontroléry MSP430 mnoho výhod zmiňovaných v teoretické části o mikrokontrolérech, v průběhu této práce jsem zjistil, že jeho poměrně velká nevýhoda je dostupnost. Nemožnost sehnat vybraný typ mikrokontroléru v České republice mi zkomplikovala průběh mé práce. Avšak mimo to pro ně mluví jen samé klady. Například vynikající ceny výkonných a dobře vybavených mikrokontrolérů s ultra nízkou spotřebou energie.

Jsem rád, že jsem pro vývoj obslužné aplikace zvolil právě .NET. Po seznámení se s ním mohu konstatovat, že je to vskutku robusní řešení pro tvorbu uživatelských aplikací. Ve srovnání s konkurencí implementuje řadu užitečných a kvalitních funkcí. Za jeho největší nevýhodu jsem považoval, že je to komerční produkt tvořený jednou firmou. Ovšem po detailnějším pohledu na současný stav opensource projektu *mono* a taktéž opensource IDE *monodevelop* jsem zjistil, že již dospěly do stavu, kdy jsou velmi dobře použitelné, tudíž se z nevýhody stává výhoda (možnost volby mezi komerční a opensource implementací).

Reference

[1] Introduction To RS232 Serial Communication. Srpen 2011.
URL <http://www.wcscnet.com/Tutorials/SerialComm/Page1.htm>

[2] 8bitové mikrokontroléry PIC. Duben 2012.
URL <http://mikrokontrolery-pic.cz/katalog/8bitove-mikrokontrolery-pic/>

[3] MSP430x2xx Family User's Guide (Rev. I). Leden 2012.
URL <http://www.ti.com/litv/pdf/slau144i>

[4] Top Operating System Share Trend. Leden 2012.
URL <http://marketshare.hitslink.com/os-market-share.aspx?qprid=9>

[5] V-USB Virtual USB port for AVR microcontrollers. Červenec 2012.
URL <http://www.obdev.at/products/vusb/index.html>

[6] Conn, N.: NJC's LaunchScope (A LaunchPad Oscilloscope). Prosinec 2010.
URL <http://www.msp430launchpad.com/2010/12/njcs-launchscope-launchpad-oscilloscope.html>

[7] Kolář, P.: *Operační systémy.* Liberec, Leden 2005.
URL <http://www.nti.tul.cz/~kolar/os/>

[8] Kopelent, J.: ATMEL AVR neboli mikroprocesory rodiny 90Sxxx a dalších. Říjen 2003.
URL <http://www.hw.cz/soucastky/embedded-systemy/mcu/atmel-avr-neboli-mikroprocesory-rodiny-90sxxx-a-dalsich.html>

[9] Skalický, P.: *Mikroprocesory z řady 8051.* Praha: BEN technická literatura, druhé vydání, 2003, ISBN 80-86056-39-2.

[10] Troelsen, A.: *Pro C# 2010 and the .NET 4 Platform.* Páté vydání, Květen 2010, ISBN 978-1-4302-2549-2.

[11] Šeda, J.: Java vs. C# - který jazyk zvolit. Březen 2012.
URL <http://www.zive.cz/clanky/java-vs-c---ktery-jazyk-zvolit/sc-3-a-104694/default.aspx>

Seznam použitých zkratek

ALU	Aritmeticko-logická jednotka provádí aritmetické a logické početní operace v procesoru
Android	Linuxový operační systém od společnosti Google předurčený především pro mobilní zařízení
API	Aplikační rozhraní
DSP	Digitální Signální Procesor - návrh stvořený pro zpracování digitálních signálů ve velkých objemech
EEPROM	Electrically Erasable Programmable Read-Only Memory - Paměť podobná paměti EPROM, avšak její mazání probíhá elektricky, není potřeba UV záření
EPROM	Erasable Programmable Read-Only Memory - paměti typu ROM-RAM, jejíž obsah je mazatelný UV zářením
Flash	Nevolatilní (semipermanentní) elektricky programovatelná (zapisovatelná) paměť s libovolným přístupem. Principem je to EEPROM vnitřně organizována po blocích, kde se každý blok dá programovat samostatně bez ovlivnění ostatních bloků.
FPGA	Field Programmable Gate Array - hradlové pole
FRAM	Ferroelectric Random Access Memory - nový typ paměti rychlý jako SRAM, avšak s vlastností uchování dat i po odpojení napájení, TI je používá v některých svých MCU místo paměti FLASH
framework	Softwarová struktura, která slouží k převzetí typických problémů dané oblasti, čímž usnadňuje vývoj tak, aby se vývojáři mohli více soustředit na své zadání a nemuseli řešit rutiny.
GUI	Graphical User Interface - grafické uživatelské rozhraní
I2C	multi-masterová dvoudrátová přístrojová sběrnice používána k připojování nízkorychlostních periferií
IDE	Vývojové prostředí pro programování aplikací, obvykle program sdružující pokročilý textový editor, debugger, nástroje pro vizuální tvorbu frontendu a další pomůcky
Linux	Jádro oeračního systému, po kterém se běžně nazývá celý operační systém na bázi Unixu - GNU/Linux - Je vydáván pod licencí GPL, opensource a zadarmo.
MCU	Mikrokontrolér, mikropočítač - součástka, která lze naprogramovat, aby dělala určenou úlohu v elektronickém zapojení
MIPS	Million Instructions Per Second - milion instrukcí za sekundu - měrná jednotka rychlosti mikrokontrolérů
OTP	One Time Programmable - verze mikrokontroléru který je možný naprogramovat pouze jednou, určeno pro velké série, bývá levnější.
periferie	obvykle zařízení rozšiřující možnosti použití počítače, např klávesnice, či tiskárna

RISC	Reduced Instruction Set Controllers - Redukovaná instrukční sada -podpora jen jednoduchých instrukcí proveditelných během jednoho cyklu
RTC	Real Time Clock - bývají takto označovány např. obvody reálného času
SIMD	Single Instruction, Multiple Data - tzv. Vektorový počítač
SMD	Surface Mount Technology - elektronické součástky určené pro povrchovou montáž - nemají nožičky
SPI	Serial Peripheral Interface - sériové periferní rozhraní - Používá se pro komunikaci mezi řídícími mikroprocesory a ostatními integrovanými obvody (EEPROM, A/D převodníky, displeje...)
toolkit	sada nástrojů navržená k řešení určitého problému
UART	Universal Asynchonous Reciever Transmitter - zařízení pro asynchronní komunikaci po sériovém portu například počítače
USART	Universal Synchronous/Asynchonous Reciever Transmitter - zařízení umožňující jak synchronní tak i asynchronní komunikaci po sériovém portu
USB	Universal Serial Bus - v současné době nejpoužívanější rozhraní pro komunikaci počítače s periferiemi

Seznam obrázků

Obsah přiloženého CD

Přiložené CD obsahuje následující adresáře:

- `Text` - obsahuje tuto práci ve formátu PDF
- `Datasheets` - obsahuje datasheety a manuály používané při programování
- `PC_Sources` - obsahuje zdrojové kódy počítačové aplikace v C# jako *Visual Studio* projekt
- `MSP430_Sources` - obsahuje zdrojové kódy pro mikrokontrolér v jazyce C jako *Code Composer Studio* (Eclipse) projekt
- `Installation_Files` - soubory potřebné k instalaci programu - tj. návod na instalaci (soubor README), zkompilovaný program a instalační soubor .NET frameworku