VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

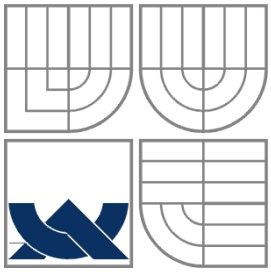
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

POMOCNÝ NÁSTROJ PRO PROGRAMOVÁNÍ MIKROKONTROLÉRŮ AVR V JAZYCE C

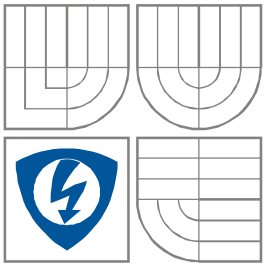
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. RADEK ŠEVČÍK
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

POMOCNÝ NÁSTROJ PRO PROGRAMOVÁNÍ MIKROKONTROLÉRŮ AVR V JAZYCE C

HELPING TOOL FOR AVR MICROCONTROLLERS PROGRAMMING IN C LANGUAGE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Radek Ševčík
AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Frýza, Ph.D.
SUPERVISOR

BRNO, 2008

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ
Fakulta elektrotechniky
a komunikačních technologií
Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student: Ševčík Radek, Bc. ***ID:*** 88495
Ročník: 2 ***Akademický rok:*** 2007/08

NÁZEV TÉMATU:

Pomocný nástroj pro programování mikrokontrolérů AVR v jazyce C

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte způsob programování mikrokontrolérů AVR firmy Atmel pomocí programovacího jazyka C. Sestavte přehled nejpoužívanějších překladačů, přičemž se zaměřte na volný nástroj GCC-AVR.

Vytvořte kostru softwarového nástroje (projekt wizard), který umožní uživatelsky příjemnou tvorbu zdrojového kódu v jazyce C pomocí GCC.

Vytvořte anglickou verzi vašeho programu, umožňující generování zdrojového kódu pro většinu běžných periférií mikrokontrolérů AVR.

DOPORUČENÁ LITERATURA:

[1] Atmel Corporation. Oficiální stránka firmy Atmel. [online]. 2007 - [cit. 25. září 2007]. Dostupné na WWW: http://www.atmel.com/

[2] NonGNU. AVR Libc Home Page. [online]. 2007 - [cit. 25. září 2007]. Dostupné na WWW: http://www.nongnu.org/avr-libc/

[3] Unis. Processor Expert OnLine. [online]. 2007 - [cit. 25. září 2007]. Dostupné na WWW: http://www.processorexpert.com/

Termín zadání: 5.10.2007 ***Termín odevzdání:*** 30.5.2008

Vedoucí projektu: Ing. Tomáš Frýza, Ph.D.

prof. Dr. Ing. Zbyněk Raida
předseda oborové rady

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií Ústav radioelektroniky

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Bc. Radek Ševčík
Bytem: Luká 93, Luká, 783 24
Narozen/a (datum a místo): 19. března 1984 v Olomouci

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 53, Brno, 602 00
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika
(dále jen „nabyvatel“)

Čl. 1

Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

   ☐ disertační práce
   ☒ diplomová práce
   ☐ bakalářská práce
   ☐ jiná práce, jejíž druh je specifikován jako .....
   (dále jen VŠKP nebo dílo)

   Název VŠKP: Pomocný nástroj pro programování mikrokontrolérů AVR v jazyce C
   Vedoucí/ školitel VŠKP: Ing. Tomáš Frýza, Ph.D.
   Ústav: Ústav radioelektroniky
   Datum obhajoby VŠKP: _____

   VŠKP odevzdal autor nabyvateli*:

   ☒ v tištěné formě – počet exemplářů: 2
   ☒ v elektronické formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

* hodící se zaškrtněte

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   ☒ ihned po uzavření této smlouvy
   ☐ 1 rok po uzavření této smlouvy
   ☐ 3 roky po uzavření této smlouvy
   ☐ 5 let po uzavření této smlouvy
   ☐ 10 let po uzavření této smlouvy
   (z důvodu utajení v něm obsažených informací)

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 30. května 2008

…….. ……

Nabyvatel Autor

Abstrakt
Tato práce popisuje program AVR Wizard, který byl vyvinut k vývojovému prostředí AVR Studio. Software AVR Wizard slouží k usnadnění programování tím, že po zadání používaných periférií vygeneruje kostru programu v jazyce C, která obsahuje přednastavení registrů a funkcí. Vytvořená kostra se zkopíruje do vývojového prostředí AVR Studio a zde je doprogramována do finální podoby. Výhodou programu je snadné ovládání a rychlé nastavení periférií bez nutnosti zdlouhavého vyhledávání v dokumentaci k programovanému obvodu.

Klíčová slova
Jazyk C, AVR, mikrokontrolér, project wizard, programování, AVR Wizard.

Abstract
This article describes AVR Wizard programme that was developed for AVR Studio. This software simpliefies microcontrollers (MCUs) programming in a way of general peripheries settings and registers settings for varied MCUs. AVR Wizard is able to generate C language-based MCU main programme structure with chosen settings. User-friendly handling and no longer detailed studies of MCU datasheets are also main advantage.

Keywords
C language, AVR, microcontroller, project wizard, programming, AVR Wizard.

Bibliografická citace
ŠEVČÍK, R. *Pomocný nástroj pro programování mikrokontrolérů AVR v jazyce C*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 49s. Vedoucí diplomové práce Ing. Tomáš Frýza, Ph.D.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma Pomocný nástroj pro programování mikrokontrolérů AVR v jazyce C jsem vypracoval samostatně pod vedením vedoucího diplomové 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 diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové 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 § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 30. května 2008 .....
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Tomáši Frýzovi, Ph.D., za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne 30. května 2008 .....
podpis autora

Obsah

1 Úvod

Začátkem devadesátých let minulého století, v norském vývojovém centru Nordic VLSI v Trondheimu, skupina návrhářů hardwaru spolu s programátory navrhla novou strukturu mikrokontrolérů. Tato struktura byla navržena tak, aby vyhovovala překladačům vyšších programovacích jazyků, zejména jazyka C. Před několika lety tuto koncepci od norských návrhářů koupila firma Atmel a založila na ní rodinu mikrokontrolérů AVR [1].

Pro programování mikrokontrolérů AVR existují komerční programy, nebo lze využít volně šiřitelný software AVR Studio, který poskytuje na svých internetových stránkách firma Atmel [7]. Výhodou komerčních programů jsou rozličné doplňkové služby, jako jsou například různé pomocné programy, či speciální knihovny pro často používané externí obvody, díky kterým se usnadňuje programování mikrokontrolérů.

Tato práce si klade za cíl vytvořit uživatelsky přívětivý software k programu AVR Studio (tzv. projekt wizard), který po zadání využívaných periférií vygeneruje kostru programu v jazyce C pro procesory AVR.

Celá práce je logicky rozčleněna do osmi kapitol. Kapitoly 2 a 3 přibližují základy architektury a popisují jednotlivé periférie mikrokontrolérů AVR. V kapitole 4 jsou stručně nastíněna pravidla programování v jazyce C a jazyce symbolických adres. Kapitola 5 slouží pro seznámení se s vývojovými prostředími pro vývoj programů MCU AVR. Pravidla pro vytvoření a čtení značkovacích (*.xml) a inicializačních (*.ini) souborů jsou popsána v kapitole 6. Kapitola 7 slouží ke seznámení se s vytvořenou aplikací AVR Wizard. Shrnutí celé práce je uvedeno v kapitole 8.

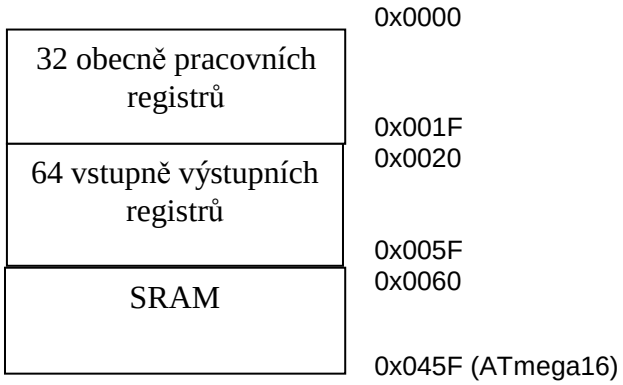
2 Základy AVR

K vytvoření programu pro mikrokontrolér nejenže programátor musí ovládat programovací jazyk, užívaný pro programování softwaru, ale i architekturu obvodu, se kterým pracuje. Jednotlivé typy obvodů AVR se liší nejen ve tvaru a velikosti pouzdra, v němž jsou vyráběny, ale i velikostí pamětí, počtem a druhem periférií, které obsahují.

2.1 Paměť

Jako většina mikrokontrolérů, tak i AVR má oddělenou paměť programu a dat (tzv. Harvardská architektura). Paměť programu je typu flash a její velikost se liší podle verze obvodu od jednotek po desítky kilobajtů. Organizace této paměti je vždy po 16 bitech z důvodu možnosti načítání většiny instrukcí v jednom hodinovém taktu.

Paměť dat je na rozdíl od paměti programu osmibitová a je rozdělena následovně (viz. obrázek 1): Prvních 32Bytů je vyhrazeno pro 32 obecných pracovních registrů (R0, R1, …, R31). Za těmito 32Byty se nachází dalších 64Bytů vstupně výstupních registrů, které slouží k ovládání jednotlivých periférií, jež mikrokontrolér obsahuje. Za vstupně výstupními registry následuje interní paměť typu SRAM, která slouží k ukládání libovolných dočasných dat. Za ní by se mohla po připojení externího obvodu nalézat externí SRAM (tedy pokud mikrokontrolér podporuje připojení vnější paměti). Velikost paměti SRAM může dosahovat až 64kB (omezeno 16b adresní sběrnicí).



Obr. 1: *Rozdělení paměti dat mikrokontrolérů AVR*

2.2 Periférie

Mikrokontroléry AVR obsahují i mnoho periférií. Jsou to například A/D převodníky, sloužící k měření vnějšího spojitého napětí, čítače/časovače, pomocí kterých procesor může odpočítávat přesné časové intervaly, EEPROM paměť, jež dokáže uchovávat informace i bez přítomnosti napájecího napětí, obvody pro sériovou komunikaci a jiná zařízení, jako například komparátory, watchdog, vnější přerušení, atd.

Každá z periférií má přiřazené některé vstupně výstupní registry, pomocí nichž se daný obvod dá zapnout nebo ovládat. Dále téměř každé zařízení může podat procesoru žádost o přerušení, pokud nastal nějaký definovaný stav (např. A/D převodník dokončil převod). Při přerušení dojde k zapamatování pozice právě probíhajícího programu a začne se vykonávat program uložený na adrese v paměti programu, která je definována podle toho, od jakého zařízení došlo k přerušení.

Jednotlivé typy mikrokontrolérů AVR se liší v počtu a druhu periférií, proto adresa, na které program při přerušení pokračuje, je pro tu samou periferii u jiného typu obvodu rozdílná.

3 Periférie mikrokontrolérů AVR

Moderní mikrokontroléry obsahují velkou řadu periférií s různými možnostmi nastavení. Jejich počet ovlivňuje cenu mikrokontroléru; proto výrobce vyrábí obvody jak s větším počtem periférií, tak i obvody, které některá zařízení neobsahují a tudíž se mohou prodávat za nižší cenu. Podrobný popis všech možností nastavení jednotlivých doplňkových obvodů, které jednotlivé řady mikrokontrolérů obsahují, uvádí firma Atmel na svých stránkách [7].

Při vývoji aplikace „Project Wizardu“ je třeba, aby výsledný software byl co nejvíce univerzální pro všechny dnes existující mikroprocesory AVR. Avšak oproti starším verzím obvodů se objevují některé nové možnosti nastavení, proto v následujících kapitolách bude uveden pokud možno co nejuniverzálnější popis s veškerou variabilitou, která může u popisovaných periférií nastat.

Funkce každé periférie je ovládána pomocí I/O registrů, řídících její funkci, proto bude uveden i popis jednotlivých bitů ve I/O registrech. Dále v následujícím popisu bude popsáno, ve kterém I/O registru bit leží. Z důvodu velkého počtu obvodů typu AVR bude popis platit pouze pro ATMega16. U dalších mikrokontrolérů se následující popis může mírně lišit a přesné umístění, ve kterém registru se bit nalézá, lze nalézt v dokumentaci k danému obvodu [7].

3.1 Porty

I když porty nejsou klasickou periférií, přesto budou pro svou specifičnost zahrnuty do popisu podpůrných obvodů mikrokontroléru.

Každý port se skládá z osmi bitů. Jednotlivé bity mohou být buď vstupní nebo výstupní. Toto nastavení se provádí v registru **DDRx** (x je písmeno od A do Z udávající název portu), kde zapsáním „0“ nastavíme pin portu jako vstupní a pro „1“ jako výstupní.

Dalším registrem používaným u portů je **PORTx** (x je písmeno od A do Z udávající název portu). Jedná se o registr, pomocí něhož můžeme nastavit výstupní hodnotu na portu. Pokud je pin nastaven jako výstupní, pak se nastavením „1“ nebo „0“ do tohoto registru nastaví úroveň „H“ nebo „L“ na příslušném pinu. Pokud je však port nastaven jako vstupní, tak pro bity nastaveny na „0“ v **PORTx** jsou piny nastaveny na tzv. vysokou impedanci (Hi-Z) a vstupní odpor se blíží ∞. Kdežto při „1“ je ke vstupu připojen tzv, pull-up rezistor, který při vstupu na prázdno vytváří úroveň „H“.

Pro čtení aktuální logické hodnoty se používá registr **PINx** (x je písmeno od A do Z), kde aktuální hodnota na pinu je zobrazena buď „0“ nebo „1“ v tomto registru.

Většina pinů portu má ještě alternativní funkci (např. sériové programování, výstup PWM, atd…), proto by se při používání portů mělo zkontrolovat, zda alternativní funkce pinu nezpůsobí problém v plánované činnosti portu. Podrobný popis portů a možné alternativní funkce pinů lze nalézt v dokumentaci k danému obvodu [7].

3.2 Vnější přerušení

Pokud má mikrokontrolér vykonat vždy při určité vnější události (např. stisku tlačítka, dokončení čítání vnějšího čítače, atd…) nějakou operaci, existují dvě možnosti. První je, že by se vždy po určitých časových intervalech softwarově kontrolovaly jednotlivé porty a

zjišťovalo se, zda nedošlo k nějaké změně na pinech. Druhou (elegantnější) možností je využití externího přerušení.

Externí přerušení funguje tak, že při změně logické úrovně na příslušném pinu dojde k přerušení a začne se hned vykonávat příslušný program, který nám danou událost obslouží.

3.2.1 Popis vnějšího přerušení

Vnější přerušení nastane, pokud na pinu s názvem **INTn** (n je číslo udávající název přerušení) nastane definovaný stav. Aby toto přerušení mohlo vzniknout, musí se nejprve povolit pomocí bitu **INTn** (ležícím v registru GICR u ATmega16), kterým při nastavení na „1“ zapneme přerušení.

Dále se musí definovat, při jaké události na vstupu přerušení vůbec nastane. Toto nastavení se provede pomocí bitů **ISCn0** a **ISCn1** (ležící v registru MCUCR u ATmega16). Jednotlivé módy jsou popsány v tabulce 1.

Tabulka 1: *Význam bitů ISCn1 a ISCn0*

ISCn1	ISCn0	Popis
0	0	Nízká úroveň na INTn vyvolává požadavek na přerušení
0	1	Jakákoliv změna na pinu INTn vyvolá požadavek na přerušení
1	0	Sestupná hrana na INTn vyvolává požadavek na přerušení
1	1	Vzestupná hrana na INTn vyvolává požadavek na přerušení

V jazyce C (překladač GCC), jestliže je pomocí funkce `sei()` zapnuto globální přerušení, se začne při vzniku externího přerušení vykonávat funkce s názvem `ISR(INTn_vect)` (Interrupt Service Rutine), kde **n** značí název přerušení.

3.2.2 Přerušení při změně na pinu

Dále se u některých tipů mikrokontrolérů vyskytuje „přerušení při změně na pinu“ tzv. PCINT (Pin Change Interrupt).

Na rozdíl od klasického vnějšího přerušení, které je jen pro jeden výrobcem definovaný pin, tak PCINT nastane, pokud dojde k libovolné změně u některého ze skupiny pinů.

Aby PCINT vůbec mohlo nastat, musí se nejprve povolit nastavením bitu **PCIEn** (n je číslo udávající název přerušení) na „1“. Dále pomocí bitů s názvem **PCINTn** (n je číslo udávající číslo pinu, který se bude povolovat × zakazovat) se povolí piny, od kterých může přerušení nastat. Nastavením **PCINTn** na „1“ je pin povolen. Nastavením **PCINTn** na „0“ je příslušný pin zakázán.

V jazyce C při vzniku přerušení od PCINT se začne vykonávat funkce `ISR(PCINTn_vect)` kde **n** je číslo značící název přerušení.

3.3 Čítač/časovač

Pro čítání různých periodických signálů slouží u mikrokontroléru čítač/časovač (Č/Č). Čítače/časovače bývají nejčastěji 8-bitové nebo 16-bitové. U Č/Č je velká variabilita. U některých typů mikrokontrolérů existuje Č/Č jen pro čítání, kdežto jiné typy umí i například generovat PWM (Pulse Wide Modulation). Názvy registrů, ve kterých jednotlivé bity leží, nebudou uváděny, protože se pro jednotlivé mikrokontroléry liší.

Nejprve je nutno bity **CSn0**, **CSn1**, **CSn2** nastavit mód Č/Č. Popis jednotlivých módů je uveden v tabulce 2, kde CK je hodinový kmitočet procesoru.

Tabulka 2: *Nastavení módu Č/Č*

CSn2	CSn1	CSn0	Popis
0	0	0	Zastavení čítání, časovač/čítač je zastaven
0	0	1	Časovač/čítač čítá s kmitočtem CK
0	1	0	Časovač/čítač čítá s kmitočtem CK/8
0	1	1	Časovač/čítač čítá s kmitočtem CK/64
1	0	0	Časovač/čítač čítá s kmitočtem CK/256
1	0	1	Časovač/čítač čítá s kmitočtem CK/1024
1	1	0	Časovač/čítač čítá z externího vstupu, sestupná hrana
1	1	1	Časovač/čítač čítá z externího vstupu, vzestupná hrana

Dále se pomocí bitů **WGMn0..3** zvolí mód plnění časovače/čítače. Příklad módů generace PWM je pro 16-bitový Č/Č je uveden v tabulce 3 a pro 8-bitový Č/Č v tabulce 4.

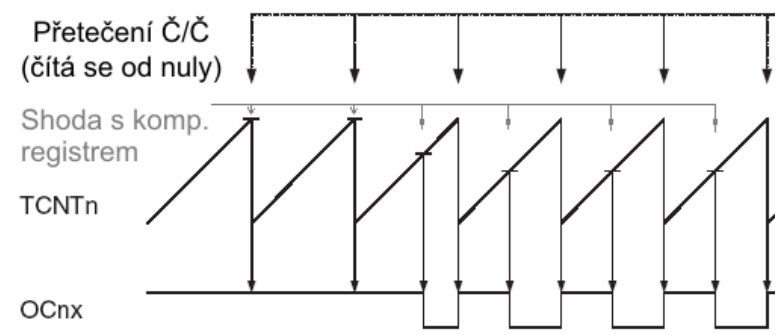
Tabulka 3: *Módy PWM (16-bitový Č/Č)*

WGM*n*3	WGM*n2*	WGM*n1*	WGM*n*0	Mód PWM
0	0	0	0	Normální čítání
0	0	0	1	Fázově korigovaná PWM, 8-bit
0	0	1	0	Fázově korigovaná PWM, 9-bit
0	0	1	1	Fázově korigovaná PWM, 10-bit
0	1	0	0	CTC (Při shodě s kom. reg T/T vynulován)
0	1	0	1	Rychlá PWM, 8-bit
0	1	1	0	Rychlá PWM, 9-bit
0	1	1	1	Rychlá PWM, 10-bit
1	0	0	0	Fázově a frekvenčně korigovaná PWM, ICRn
1	0	0	1	Fázově a frekvenčně korigovaná PWM, OCRnA
1	0	1	0	Fázově korigovaná PWM, ICRn
1	0	1	1	Fázově korigovaná PWM, OCRnA
1	1	0	0	CTC při shodě s ICRn
1	1	0	1	Rezervováno
1	1	1	0	Rychlá PWM, ICRn
1	1	1	1	Rychlá PWM, OCRnA

Tabulka 4: *Módy PWM (8-bitový Č/Č)*

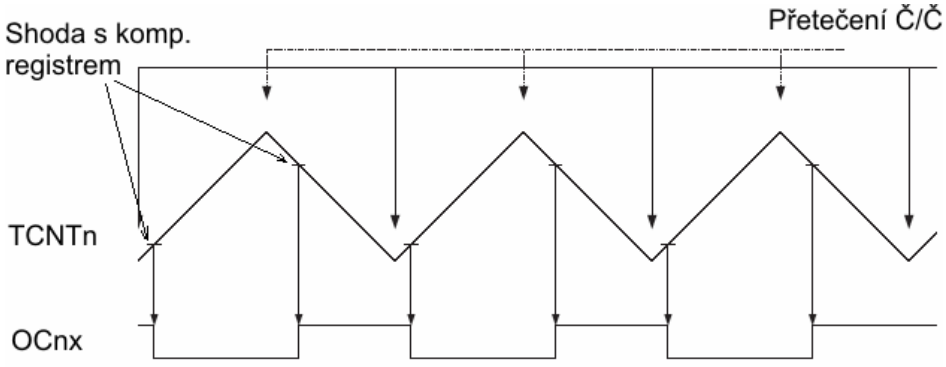
WGM*n1*	WGM*n*0	Mód PWM
0	0	Normální činnost
0	1	Fázově korigovaná PWM, 8-bitů
1	0	CTC, při shodě s OCRn
1	1	Rychlá PWM, 8-bitů

Rychlá PWM je specifická tím, že čítá ode dna do maxima (přičemž maximum lze měnit volbou režimu) a poté se vrátí zpět ke dnu. Při shodě s příslušným komparačním registrem dojde ke změně úrovně výstupu **OCnx** (kde **n** je číslo určující Č/Č a **x** je písmeno odlišující jednotlivé komparační registry od sebe). Princip činnosti časovače/čítače v tomto režimu je nakreslen na obrázku 2.



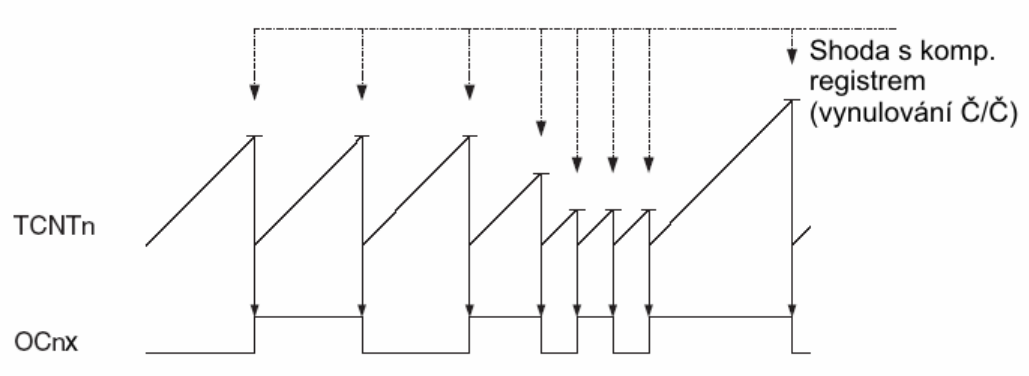
Obr. 2: *Rychlý PWM režim*

Dále časovač/čítač může pracovat ve fázově korigovaném PWM režimu. Tento režim poskytuje oproti rychlému PWM režimu 2x větší rozlišení. Je to dáno tím, že čítá nejenom nahoru, ale poté také i zpět ke dnu. Princip činnosti generování PWM je nakreslen na obrázku 3.



Obr. 3: *Fázově korigovaná PWM*

Posledním popsaným režimem bude CTC (Clear Timer on Compare). V tomto nastavení, když dojde ke shodě registru Č/Č **TCNTn** a komparačního registru, nastane vynulování obsahu **TCNTn** a čítač začne čítat od nuly. Princip činnosti tohoto módu je zobrazen na obrázku 4.



Obr. 4: *CTC režim (Č/Č při shodě s komp. registrem vynulován)*

Dalšími bity sloužícími pro nastavení časovače/čítače jsou **COM*nx*0..1**. Těmito bity se nastavuje výstup **OCnx**. Tento výstup se většinou používá pro generování PWM. V tabulce 5 jsou uvedeny možnosti tohoto výstupu.

Tabulka 5: *Možnosti výstupu OCnx*

COM*n1*	***COMn*0**	**Mód výstupu OCnx**
0	0	Výstup OCnx odpojen
0	1	Překlopení výstupu OCnx
1	0	Vynuluje výstup OCnx
1	1	Nastaví výstup OCnx

Nakonec lze ještě nastavit zachytávání aktuální hodnoty z **TCNT*n*** do **ICR*n***. Toto zachytávání může být spuštěno dvěma způsoby. První je při nastavení hodnoty na vstupní pin ICP a druhý způsob je překlopení analogového komparátoru. U tohoto zachytávání se pomocí bitu **ICES*n*** nastaví vzorkování při vzestupné (**ICES*n*** = 0), či sestupné (**ICES*n*** =1) hraně a pomocí bitu **ICNC*n*** omezení šumu. Pokud je **ICNC*n*** nastaven na „1“ pak se za nezašuměný signál bere takový, jehož následující čtyři vzorky jsou stejné.

3.3.1 Popis registrů a bitů čítače/časovače

Protože nastavení čítače/časovače je poněkud složitější než nastavení předchozích popisovaných obvodů mikrokontroléru, bude v této kapitole uvedeno krátké shrnutí významu některých důležitých bitů a I/O registrů, které slouží k nastavení čítače/časovače. Z důvodu přehlednosti nejsou uvedeny všechny existující bity a registry, ale jen ty, které jsou nezbytně nutné pro nastavení Č/Č.

Registry:

TCNT*n* – obsahuje hodnotu čítání čítače / časovače n
OCR*nx* – komparační registr
ICR*n* – záchytný registr

Bity:

OCIE*n* – povolí přerušení při shodě registrů **TCNT*n* a OCR*n***
TOIE*n* – povolí přerušení při přetečení **TCNT*n***

TICIE*n* – povolí přerušení při nastavení příznaku **ICF*n***

CS*n*0..2 – nastavení předděličky Č/Č
COM*nx*0..1 – nastavení výstupu OCnx

ICNC*n* – omezení šumu
ICES*n* – vzorkování při vzestupné (**ICES*n*** = 0), či sestupné (**ICES*n*** =1) hraně

WGM*n*0..3 – Nastavení módu PWM

Kde:

n – číslo udávající název čítače/časovače (např.: 1 - čítač/časovač 1)
x – písmeno A, B, C, … udávající název komparačního registru (pokud Č/Č obsahuje jen 1 komparační registr, koncovka **x** se nepoužívá)

3.4 Watchdog

Pokud vyvíjený program obsahuje nějaké skryté chyby, může se stát, že se v nějaké neošetřené situaci program zacyklí a potom by celá aplikace, v níž je mikrokontrolér použit, byla nefunkční. Řešením by bylo napsat dokonalý software bez chyb, ale v praxi se to ne vždy podaří. Proto druhým řešením je použití watchdogu. Dále lze watchdogu využít třeba k probuzení z některého z úsporných režimů.

Watchdog je čítač, který po svém přetečení způsobí reset mikrokontroléru. Aby k resetu nedošlo, je nutno občas watchdog vynulovat. Pokud dojde k zacyklení programu, watchdog není včas vynulován a dojde k resetu programu. Povolení watchdogu se provádí pomocí bitu **WDE**, který se nastaví na „1“. Dále nastavením bitů **WDP0..2** se nastaví dělící poměr přiváděného kmitočtu do čítače watchdogu.

Jako zdroj hodinového kmitočtu složí speciální oscilátor (např.: pro ATmega16 1MHz, ATTiny13 128kHz), který je nezávislý na hodinovém taktu mikroprocesoru. Jednotlivé dělící poměry jsou uvedeny v tabulce 6 (pro ATMega16) a v tabulce 7 (pro ATTiny13).

Kromě klasického resetu MCU může u některých obvodů (např. ATTiny13) při přetečení čítače watchdogu nastat přerušení. Další možností je, že nastane přerušení a po jeho vykonání teprve reset mikrokontroléru. Opět zde existuje variabilita nastavení a detailní popis watchdogu lze nalézt v dokumentaci právě programovaného obvodu [7].

Tabulka 6: *Dělící poměry předděličky watchdogu (ATMega16)*

WDP2	WDP1	WDP0	Popis
0	0	0	16K cyklů
0	0	1	32K cyklů
0	1	0	64K cyklů
0	1	1	128K cyklů
1	0	0	256K cyklů
1	0	1	512K cyklů
1	1	0	1024K cyklů
1	1	1	2048K cyklů

Tabulka 7: *Dělící poměry předděličky watchdogu (ATTiny13)*

WDP3	WDP2	WDP1	WDP0	Popis
0	0	0	0	2K cyklů
0	0	0	1	4K cyklů
0	0	1	0	8K cyklů
0	0	1	1	16K cyklů
0	1	0	0	32K cyklů
0	1	0	1	64K cyklů
0	1	1	0	128K cyklů
0	1	1	1	256K cyklů
1	0	0	0	512K cyklů
1	0	0	1	1024K cyklů

3.5 Předdělička hodin MCU

Některé MCU (například ATTiny13, atd…) obsahují programovatelnou předděličku systémových hodin mikrokontroléru. Bity pro nastavení předděličky se nalézají v registru **CLKPR**. Před samotnou změnou kmitočtu by se měly vypnout všechna přerušení, aby nedošlo k chybnému vygenerování přerušení. Samotná změna má dva kroky. Nejprve je nutno zapsat log. „1“ do bitu **CLKPCE** a následně změnit hodnoty bitů **CLKPS0..3** podle požadovaného dělícího poměru - viz. tabulka 8.

Tabulka 8: *Nastavení předděličky hodin MCU (ATTiny13)*

CLKPS3	CLKPS2	CLKPS1	CLKPS0	Dělící poměr
0	0	0	0	1/1
0	0	0	1	1/2
0	0	1	0	1/4
0	0	1	1	1/8
0	1	0	0	1/16
0	1	0	1	1/32
0	1	1	0	1/64
0	1	1	1	1/128
1	0	0	0	1/256

3.6 Sériová linka

Pro přenos dat a komunikaci mezi různými systémy a zařízeními (např.: MCU s PC) se používá sériová linka USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter). Komunikace může být synchronní: jedno zařízení generuje hodiny a ostatní zařízení jsou synchronizovány tímto signálem. Další možností je asynchronní komunikace, kdy má každé zařízení svůj přesný zdroj hodin nastavený na stejnou komunikační rychlost jako ostatní obvody. Asynchronní přenos se používá častěji a jeho výhodou je potřeba použití pouze dvou vodičů (vysílací a přijímací), ale nevýhodou je, že i při malém rozladění kmitočtu hodin pro generování přenosové rychlosti může docházet k chybám při přenosu dat.

Pro příjem i vysílání se používá společný registr **UDR**. Při zapsání dat do tohoto registru dojde k vyslání bajtu sériovou linkou a při příjmu se z tohoto registru čtou data. U

sériové linky se vysílání povoluje pomocí bitu **TXEN** (registr **UCSRB**) a příjem pomocí **RXEN** (registr **UCSRB**). K nastavení přenosové rychlosti slouží registr **UBRR** a bit **U2X**, který se nalézá v registru **UCSRA**. Pokud je bit **U2X** roven „0“, platí pro výpočet přenosové rychlosti vzorec (1)

$$BAUD = \frac{f_{CPU}}{16 \cdot (UBRR + 1)}, \qquad (1)$$

kde BAUD je přenosová rychlost v baudech, UBRR je hodnota **UBRR** registru a f_{OSC} je kmitočet hodin MCU. Pokud je bit U2X roven „1“, je přenosová rychlost vypočtená podle (1) dvojnásobná.

Dále lze zapsáním hodnoty 1, 0 (2) do bitů **UPM1,0** (registr **UCSRA**) zapnout sudou paritu a zapsáním hodnoty 1, 1 (3) lichou paritu. Počet stopbitů ukončujících vysílaný symbol se nastavuje pomocí bitu **USBS** (registr **UCSRA**). Pokud se **USBS** rovná „0“, je používán při komunikaci jeden stopbit, pokud **USBS** je „1“, je symbol ukončen dvěma stopbity.

Pro nastavení délky vysílaného symbolu slouží bity **UCSZ2..0** (registry **UCSRA** a **UCSRB**). Možnosti nastavení těchto bitů jsou uvedeny v tabulce 9.

Tabulka 9: *Nastavení počtu vysílaných bitů*

UCSZ2	UCSZ1	UCSZ0	Počet dat. bitů
0	0	0	5 bitů
0	0	1	6 bitů
0	1	0	7 bitů
0	1	1	8 bitů
1	1	1	9 bitů

Při použití sériové komunikace mohou nastat tři druhy přerušení. Prvním typem je přerušení při příjmu dat, které lze zapnout zapsáním „1“ do bitu **RXCIE** (registr **UCSRB**). Pokud přijdou data a jsou připravena v registru **UDR** ke čtení, nastane toto přerušení. Další přerušení nastane při odesílání bajtu z registru **UDR** na sériovou linku, pokud je zapnuto pomocí bitu **TXCIE** (registr **UCSRB**). Posledním typem je přerušení při prázdném datovém registru **UDR**, pokud je povoleno bitem **UDRIE** (registr **UCSRB**). V tomto režimu se registr **UDR** využívá jako jednobajtový buffer. Data na sériovou linku jsou vysouvána postupně ze speciálního výstupního registru. Při zapsání do **UDR** dojde k překopírování dat do výstupního registru, ze kterého jsou data postupně vysílána. Jakmile je takto registr **UDR** uvolněn, nastane přerušení při prázdném datovém registru. Při přerušení jsou do **UDR** zapsána nová data a čekají zde, než je vyprázdněn výstupní registr. Ten jakmile je volný, načte data z **UDR**, **UDR** se vyprázdní, nastane přerušení a celý proces se opakuje dokola. Výhodou tohoto způsobu vysílání je, že nedochází k prodlevě mezi jednotlivými vysílanými bajty.

3.7 TWI linka

TWI (Two-wire Serial Interface) je dvojvodičová sériová linka sloužící k připojení až 128 různých zařízení k mikrokontroléru. Jeden vodič slouží jako datový a druhý jako hodinový. Jedno zařízení vždy slouží jako hlavní (master), většinou to bývá MCU. To se stará o generování hodin a povoluje vysílání i ostatním (slave) obvodům. Jednotlivé obvody připojené k TWI lince mají vždy svou adresu, podle které se navzájem odlišují. Příkladem

použití je například připojení sériové EEPROM nebo teplotního čidla k MCU prostřednictvím linky TWI.

Pro nastavení přenosové rychlosti slouží **TWBR** registr a bity předděličky **TWPS1,0** ležící v registru TWSR. Přenosová rychlost se dá vypočítat pomocí následujícího vztahu,

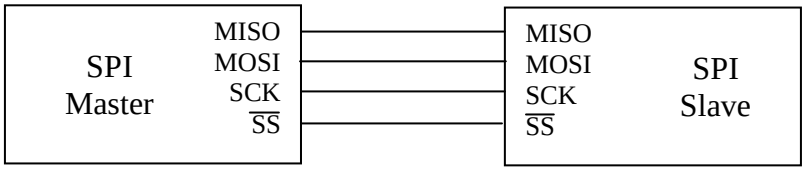
$$f_{SCL} = \frac{f_{CPU}}{16 + 2 \cdot TWBR \cdot 4^{TWPS}}, \qquad (2)$$

kde TWBR je hodnota **TWBR** registru, TWPS je hodnota bitů **TWPS1,0** a f_{CPU} je kmitočet hodin MCU. TWI se zapíná pomocí bitu **TWEN** v registru **TWCR** a pomocí bitu **TWIE** (registr **TWCR**) lze zapnout přerušení. Detailní popis protokolu a pravidla komunikace jsou uvedeny v dokumentaci připojovaného obvodu nebo v popisu příslušného mikrokontroléru [7].

3.8 SPI linka

Hlavní použití SPI (Serial Peripheral Interface) linky je pro sériové programování, kdy po propojení signálů MOSI, MISO, SCK a RESET lze do MCU nahrát pomocí programátoru program. SPI se může využít i ke komunikaci mikroprocesoru s externími přídavnými obvody (např.: EEPROM, displeje, MMC paměti…).

Komunikace je založena na zařízení typu master (většinou MCU), které generuje hodinový signál a povoluje pomocí signálu SS (Slave Select) zařízením typu slave komunikaci. Komunikace probíhá duplexně pomocí vodičů MISO (Master Input Slave Output) a MOSI (Master Output Slave Input). Propojení dvou zařízení pomocí SPI linky je na obrázku 5.



Obr. 5: *Propojení pomocí SPI linky*

SPI linka se zapíná bitem **SPE**. Všechny nastavovací bity se nalézají v registru **SPCR**. Popřípadě lze zvolit přerušení od SPI bitem **SPIE**. Bitem **DORD** se vybere směr vysouvání dat a pomocí **MSTR,** zda má být zařízení typu master (**MSTR** = 1) nebo slave. Bity **CPOL** a **CPHA** se nastaví, při jaké hraně hodin má nastat čtení a při jaké přepis dat na lince. Posledním nastavením je výběr předděličky bity **SPR1,0** a **SP2X** (registr **SPSR**). Jednotlivé kombinace jsou rozepsány v tabulce 10.

Tabulka 10: *Nastavení předděličky SPI linky*

SP2X	SPR1	SPR0	Dělící poměr
0	0	0	$f_{OSC}/4$
0	0	1	$f_{OSC}/16$
0	1	0	$f_{OSC}/64$
0	1	1	$f_{OSC}/128$
1	0	0	$f_{OSC}/2$
1	0	1	$f_{OSC}/8$
1	1	0	$f_{OSC}/32$
1	1	1	$f_{OSC}/64$

3.9 Kompararátor

Komparátor slouží k porovnávání dvou napětí. Výstup komparátoru může nabývat pouze dvou hodnot: „1“ pokud je na neinvertujícím vstupu vyšší napětí než na invertujícím a „0“ pokud je na neinvertujícím vstupu nižší napětí než na invertujícím.

Komparátor se zapne nastavením „0“ do bitu **ACD** (registr **ACSR**) a výsledek komparace se objevuje na výstupním bitu **ACO** (registr **ACSR**). U neinvertujícího vstupu lze bitem **ACBG** (registr **ACSR**) volit ze dvou možností. Pokud je bit **ACBG** = 0, pak je vstup připojen na vývod z názvem AIN0. Když se **ACBG** = 1, pak je na neinvertující vstup připojena vnitřní reference.

Invertující vstup má větší možnosti výběru. Pokud se používá AD převodník, může být invertující vstup připojen pouze k vývodu AIN1, ale pokud je převodník vypnut, lze pomocí bitů **MUXn** (viz. popis AD převodníku) vybrat některý ze vstupů převodníku. Jednotlivé možnosti jsou rozepsány v tabulce 11.

Tabulka 11: *Výběr invertujícího vstupu komparátoru*

ACME	ADEN	MUX2..0	Inv. Vstup komparátoru
0	X	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

Nakonec lze nastavit bitem **ACIE** (registr **ACSR**) přerušení a bity **ACIS1,0** (registr **ACSR**) vybrat při jaké události přerušení nastane. Jednotlivé možnosti jsou rozepsány v tabulce 12.

Tabulka 12: *Výběr invertujícího vstupu komparátoru*

ACIS1	ACIS0	Přerušení
0	0	Přerušení při překlápění výstupu
0	1	Rezervováno
1	0	Přerušení při sestupné hraně výstupu
1	1	Přerušení při náběžné hraně výstupu

3.10 AD převodník

I když většina zařízení, ke kterým se mikrokontrolér připojuje je digitální, je občas potřeba zpracovávat i analogové signály a napětí. Z tohoto důvodu je většina MCU AVR vybavena 10ti-bitovým AD převodníkem s postupnou aproximací. Nevýhodou tohoto typu převodníku je poměrně dlouhá doba převodu (maximálně 15kS/s), ale pro velkou část aplikací (měření NF napětí) a vzhledem k jeho ceně (cena MCU se pohybuje v desítkách Kč) je tento převodník dostatečný.

Výběr vstupu měřeného napětí se provádí pomocí bitů **MUXn** (registr **ADMUX**). Počet bitů **MUXn** záleží na konkrétním obvodu a je dán počtem vstupů a možností kombinovat vstupy různě diferenčně a s různým zesílením. Jako příklad je uvedena tabulka 13, kde jsou rozepsány možnosti výběru vstupního napětí pro převod.

Tabulka 13: *Možnosti vstupního signálu pro převod pomocí AD převodníku (ATMega16)*

MUX4	MUX3	MUX2	MUX1	MUX0	Neinv. vstup	Inv. vstup	Zisk
0	0	0	0	0	ADC0	-	-
0	0	0	0	1	ADC1	-	-
0	0	0	1	0	ADC2	-	-
0	0	0	1	1	ADC3	-	-
0	0	1	0	0	ADC4	-	-
0	0	1	0	1	ADC5	-	-
0	0	1	1	0	ADC6	-	-
0	0	1	1	1	ADC7	-	-
0	1	0	0	0	ADC0	ADC0	10x
0	1	0	0	1	ADC1	ADC0	10x
0	1	0	1	0	ADC0	ADC0	200x
0	1	0	1	1	ADC1	ADC0	200x
0	1	1	0	0	ADC2	ADC2	10x
0	1	1	0	1	ADC3	ADC2	10x
0	1	1	1	0	ADC2	ADC2	200x
0	1	1	1	1	ADC3	ADC2	200x
1	0	0	0	0	ADC0	ADC1	1x
1	0	0	0	1	ADC1	ADC1	1x
1	0	0	1	0	ADC2	ADC1	1x
1	0	0	1	1	ADC3	ADC1	1x
1	0	1	0	0	ADC4	ADC1	1x
1	0	1	0	1	ADC5	ADC1	1x
1	0	1	1	0	ADC6	ADC1	1x
1	0	1	1	1	ADC7	ADC1	1x

Tabulka 13: *Možnosti vstupního signálu pro převod pomocí AD převodníku (ATMega16) - pokračování*

MUX4	MUX3	MUX2	MUX1	MUX0	Neinv. vstup	Inv. vstup	Zisk
1	1	0	0	0	ADC0	ADC2	1x
1	1	0	0	1	ADC1	ADC2	1x
1	1	0	1	0	ADC2	ADC2	1x
1	1	0	1	1	ADC3	ADC2	1x
1	1	1	0	0	ADC4	ADC2	1x
1	1	1	0	1	ADC5	ADC2	1x
1	1	1	1	0	Ref. 1,23V (V_{BG})	-	-
1	1	1	1	1	0V (GND)	-	-

Tabulka 14: *Výběr referenčního napětí (ATMega16)*

REFS1	REFS0	Referenční napětí
0	0	Napětí na pinu AREF, vnitřní ref. vypnuta
0	1	Napětí na pinu AVcc s ext. C na AREF pinu
1	0	Rezervováno
1	1	Vnitřní 2,56V nap. ref. s ext. C na AREF pinu

Tabulka 15: *Předdělička hodin AD převodníku*

ADPS2	ADPS1	ADPS0	Dělící poměr
0	0	0	2x
0	0	1	2x
0	1	0	4x
0	1	1	8x
1	0	0	16x
1	0	1	32x
1	1	0	64x
1	1	1	128x

Dále je třeba vybrat referenční napětí pomocí bitů **REFS1,0** (registr **ADMUX**). Opět záleží na konkrétním obvodu, co se jakou kombinací přesně nastaví. Jako příklad je uvedena tabulka 14, která platí pro nastavení MCU ATMega16. Pro nastavení doby konverze napětí slouží bity **ADPS2..0** (registr **ADCSRA**), které nastaví předděličku. Jednotlivé kombinace jsou uvedeny v tabulce 15. Pokud se nastaví bit **ADATE** (registr **ADCSRA**) na „1“, lze pomocí bitů **ADTS2..0**, které leží v registru **SFIOR**, vybrat jeden mód spouštění AD převodníku - viz. tabulka 16. Tyto módy se používají k periodickému zapínání převodu AD převodníku při určitých událostech. Například pokud se vybere mód 1, převodník se zapíná signálem z komparátoru, když je vybrán mód 2, převodník se zapne při externím přerušení INT0, atd… Pokud bit **ADATE** není aktivován (**ADATE** = „0“), potom nastavení bitů **ADTS2..0** nemá na činnost převodníku žádný vliv a převodník se po dokončení konverze automaticky vypne.

Tabulka 16: *Módy činnosti AD převodníku*

ADTS2	ADTS1	ADTS0	Mód AD převodníku
0	0	0	Volně běžící mód
0	0	1	Komparátor
0	1	0	Externí přerušení INT0
0	1	1	Shoda komp. registru s časovačem / čítačem 0
1	0	0	Přetečení časovače / čítače 0
1	0	1	Shoda komp. registru B s časovačem / čítačem 1
1	1	0	Přetečení časovače / čítače 1
1	1	1	Časovače / čítače 1 capture event

Výsledek převodu se ukládá do výstupních 8-bitových registrů **ADCL** a **ADCH**. Rovněž lze k výsledku přistupovat 16-bitově čtením z registru **ADC**. Při čtení z **ADCL** a **ADCH** je nutno dbát na to, aby prvním přečteným registrem byl **ADCL** a až pak **ADCH**, protože při čtení z **ADCH** dochází k načtení nového změřeného čísla ze záchytného registru, a proto by výsledek uložený v **ADCL** nenavazoval na číslo v **ADCH**.

Mezi poslední nastavení patří zarovnání 10-bitového čísla v 16-bitovém výsledku. Toto nastavení se provádí bitem **ADLAR** (registr **ADMUX**). Pokud je **ADLAR** roven „0“, je výsledek zarovnán vpravo a pokud je třeba zarovnat výsledek vlevo, nastaví se **ADLAR** na „1“. Zarovnání vlevo se používá, pokud se chce k výsledku přistupovat rychle a pouze 8-bitově a tudíž nevadí, že se poslední dva nejméně významné bity nebudou brát v potaz. Pro názornost je uveden obrázek 6, kde ve spodní části (ADLAR = 1) lze vidět, že stačí číst pouze 8-bitový výsledek z **ADCH**.

ADLAR = 0

bit	7	6	5	4	3	2	1	0
ADCH	x	x	x	X	x	x	ADC9	ADC8
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

ADLAR = 1

bit	7	6	5	4	3	2	1	0
ADCH	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCL	ADC1	ADC0	x	X	x	x	x	x

Obr. 6: *Zarovnání výsledku převodu*

Nakonec se může bitem **ADIE** (registr **ADCSRA**) povolit přerušení, které se vykoná vždy při dokončení konverze. Aby mohl převodník fungovat, musí se nejprve zapnout bitem **ADEN** = 1 (registr **ADCSRA**) a samotná konverze se spustí zapsáním „1“ do bitu **ADSC** (registr **ADCSRA**). V případě, že se převodník nepoužívá, doporučuje se pro snížení spotřeby MCU převodník vypnout nastavením **ADEN** na „0“.

4 Programování mikrokontrolérů AVR

Architektura procesorů AVR byla navržena s ohledem na možnost programování v jazyce C. Na rozdíl od programování v jazyce symbolických adres, tzv. asembleru, je jazyk C jednodušší, přehlednější a z toho vyplývá rychlejší a snadnější naprogramování programu pro dané užití. Poněvadž se doba vývoje významně podílí na výsledné ceně zařízení, snaží se většina výrobců zkrátit návrh softwaru na minimum. Proto je jazyk C výhodnější než asembler.

Asembler má však i své výhody. Program vytvořený v asembleru je většinou rychlejší, zabírá méně místa v paměti a problém, který bychom pomocí jazyka C řešili složitým algoritmem, můžeme v asembleru vyřešit třeba jednou instrukcí (např. záměna niblů).

4.1 Jazyk symbolických adres

Prvním používaným programovacím jazykem byl jazyk symbolických adres. Jednalo se pouze o pojmenování jednotlivých instrukcí procesoru. Díky svým přednostem se stále používá k programování některých aplikací. Jeho nevýhodou je nutnost dobré znalosti architektury programovaného procesoru a výborné ovládání instrukčního souboru obvodu, pro nějž se vytváří software. Program vytvořený pomocí asembleru je ukázán ve zdrojovém kódu 1. Tento ukázkový program počítá matematickou operaci $\sum_{i=0}^{\infty}(a+b)-(c-i)$.

```
.include <m16def.inc>   ;Definice vstupně výstupních registrů

.def a = r21            ;Přejmenování jednotlivých registrů
.def b = r22            ;např. r22 se bude jmenovat b
.def c = r23
.def i = r24

.def vysledek = r25
.def pom = r26

ldi a,10                ;načtení konstant do proměnných
ldi b,20                ;např. b=20
ldi c,35
ldi i,0

start:                  ;návěstí start
      mov vysledek,a    ;načtení hodnoty proměnné a do proměnné
                        ;vysledek
      add vysledek,b    ;sečtení proměnné vysledek a proměnné b
      clc               ;vymazání příznakového bitu c
      mov pom,c         ;do pom uložení hodnoty v c
      sbc pom,i         ;odečtení pom – i (výsledek uložen do pom)
      clc               ;vymazání příznakového bitu c
      sbc vysledek,pom  ;odečtení vysledek – pom

      inc i             ;přičtení jedničky k proměnné i

jmp start               ;program pokračuje na návěstí s názvem start
```

Zdrojový kód 1: *Ukázka programu v asembleru*

4.2 Syntaxe jazyka C

Jazyk C pro procesory AVR je obdobný jako ANSI C užívané pro PC. Na následujících řádcích je uveden stručný popis jazyka C typu GCC užívané kompilerem WinAVR. Tento popis nemusí u jiných kompilerů platit. Jedná se pouze o základní popis jazyka C (GCC), který si neklade za cíl do hloubky popsat syntaxi a pravidla programování, ale pouze rychle seznámit se základy jazyka. V případě zájmu hlubšího seznámení s jazykem C lze doporučit specializovanou literaturu [13].

4.2.1 Proměnné a možnosti proměnných

Jazyk C pro 8-bitové MCU umí pracovat se všemi základními typy proměnných, které jsou v jazyce C definovány, jen se liší počtem bitů. Například proměnná typu integer není 32−bitová jak je tomu u PC, ale pouze 16−bitová. Dále proměnné typu double a float mají stejnou délku 32−bitů. Typ bool v GCC není vůbec definován a je nahrazen 8−bitovým typem char (byte), kde false je rovno 0 a true je libovolná nenulová hodnota (například 1). Jednotlivé datové typy jsou popsány v tabulce 17.

Tabulka 17: *Typy proměnných v jazyce C*

Počet bitů	8bitů	16bitů	32bitů	
Typ proměnné	char/byte	short int / int	long int	float / double
Unsigned	0 až 255	0 až 65535	0 až 4294967295	-
Singned	-128 až +127	-32768 až +32767	-2147483648 až +2147483647	-1,175e-38 až +3,402e38

Tabulka 18: *Možnosti zápisu konstant*

Název	Příklad	Dekadická hodnota
Desítková	15, 20, 0	15, 20, 0
Oktalová	010, 0, 065	8, 0, 53
Hexadecimální	0X1, 0x0f, 0xAB	1, 15, 171
Binární	0b10, 0B11, 0b00111	2, 3, 7

Pro zápis čísel a konstant existuje několik možností. V GCC jdou konstanty zapisovat desítkově, oktalově, hexadecimálně a binárně. Jednotlivé příklady a způsoby zápisu celočíselných konstant jsou uvedeny v tabulce 18. Pokud je třeba zapsat reálnou konstantu, používá se desetinná tečka (např.: 15.45). Pro exponent se používá symbol e/E. Číslo 1000 se pak může v exponenciálním tvaru zapsat jako 1e3.

Dále se proměnné rozlišují na lokální a globální. Rozdíl mezi lokální a globální proměnnou je, že lokální proměnná platí pouze v dané funkci a jinde již ne a globální proměnná platí v celém kódu.

4.2.2 Vkládání souborů a definice maker a konstant

Pokud se do vytvářeného projektu chce vložit kód z externího souboru, musí se pomocí direktivy `#include` definovat, které knihovny chceme využívat. V těchto knihovnách mohou být již předdefinovány například pojmenování jednotlivých registrů, nebo vytvořeny některé funkce, takže programátor již může začít pracovat na svém programu a nemusí pracně kopírovat a vytvářet funkce a definovat konstanty uložené v externích souborech. Například pokud se do projektu chce vložit soubor `io.h` ležící v systémovém adresáři `avr`, bude výsledný zápis vypadat následovně: `#include <avr/io.h>`

Pokud jsou v projektu třeba definovat konstanty, které se často používají (například pi=3,14), může programátor všude psát buď číslo 3.14 a nebo si vytvořit konstantu pomocí direktivy `#define`. Zápis pro vytvoření konstanty PI by pak vypadal následovně: `#define PI 3.14`. Kdekoli je potom třeba zapsat číslo 3,14 napíše se pouze konstanta PI. Výhodou tohoto zápisu je rychlá změna v případě změny konstanty. Například pokud PI se má rovnat 3,1415 pak stačí jen přepsat definici konstanty.

Pokud je potřeba místo nějaké konstanty vložit složitější zápis, lze pomocí direktivy `#define` vytvořit tzv. makro. Příkladem takového makra může být následující zápis: `#define TRIKRAT(x) 3*x`. Za x se potom v kódu doplní nějaká konstanta nebo proměnná a všude kde se napíše `TRIKRAT(něco)`, kompiler automaticky dosadí `3*něco`. Toto byl jednoduchý příklad, samozřejmě jdou vytvářet složitější makra s více proměnnými.

4.2.3 Operátory

Pomocí operátorů lze zapisovat matematické a binární operace. Přehled jednotlivých operátorů a jejich význam je uveden v tabulce 19. Přesný popis operátorů a jejich použití lze nalézt v specializované literatuře [13], [14].

Tabulka 19: *Matematické a logické výrazy jazyka C*

Výraz	Název
==	Rovnost
!=	Nerovnost
&&	Logický součin (AND)
||	Logický součet (OR)
!	Negace
<	Menší
<=	Menší nebo rovno
>	Větší
>=	Větší nebo rovno
++	Přičtení 1
--	Odečtení 1
<<	Bitový posuv vlevo

Výraz	Název
>>	Bitový posuv vpravo
+	Součet
-	Rozdíl
*	Násobení
/	Dělení
%	Dělení modulo
^	Exkluz. bit. součet (XOR)
|	Bitový součet (OR)
&	Bitový součin (AND)
~	Bitová negace
?:	Ternární podm. operátor
,	Operátor čárky

4.2.4 Řídící příkazy

Řídící příkazy slouží k větvení programu. Pokud za řídícím příkazem následuje pouze jeden příkaz, ukončuje se symbolem středníku `;`. Jestliže se má vykonat po řídícím příkazu více příkazů, uzavřou se jednotlivé příkazy do závorek `{}`. V syntaktickém zápisu se jednotlivé řídící příkazy zapisují tučně. Zde je uveden jen heslovitý popis jednotlivých řídících příkazů. Podrobný popis lze nalézt v literatuře [13], [14].

Mezi řídící příkazy patří:

if(podmínka)příkaz;	- pokud je podmínka pravda pak se vykoná příkaz napsaný za podmínkou
if(podmínka) příkaz1; **else** příkaz2;	- pokud platí podmínka vykoná se příkaz1 - pokud neplatí výše napsaná podmínka vykoná se příkaz2
for(a;b;c)příkaz1;	- a je počáteční inicializace, například, že proměnná i se má rovnat 0 - b je podmínka, která když platí tak se vykonává příkaz1 - c je příkaz, který se vykoná po vykonání příkazu1
while(podmínka)příkaz1;	- pokud podmínka platí vykonává se příkaz1
do {příkaz1;} **while**(podmínka)	- vykoná se příkaz1 - pokud podmínka platí vykonává se příkaz1

switch(proměnná)
{
case a: příkaz1;**break**;
case b: příkaz2;**break**;
..
default: příkn;
}
	
	- Jestli se proměnná rovná a pak se vykoná příkaz1
	- Jestli se proměnná rovná b pak se vykoná příkaz2
	
	- Jestli neplatila žádná z vyšších podmínek pak se vykoná příkn

break;	- Vyskočení ze smyčky viz. **switch**
return proměnná;	- ukončí právě vykonávanou funkci a nastaví návratovou hodnotu funkce na hodnotu proměnná
goto návěsti;	- skočí na návěstí
continue;	- skáče na konec nejvnitřnější neuzavřené smyčky

4.2.5 Funkce

Pro vykonávání opakujících se částí programu slouží tzv. funkce. V jazyce C se funkce vytváří následujícím způsobem. Funkce se skládá z názvu, návratové hodnoty a vstupních proměnných. Příklad zápisu hlavičky funkce může vypadat takto **int** `podil(`**int** `a,` **int** `b)` kde `podil` je název funkce `a` a `b` jsou vstupní proměnné typu integer a funkce vrací proměnnou typu integer.

Pokud do funkce nevstupují či nevystupují žádné proměnné, nahradíme proměnnou klíčovým slovem **void**. Tělo funkce je ohraničeno závorkami `{}` a návratová hodnota funkce se vrátí pomocí klíčového slova **return**. Pokud je funkce třeba používat dříve než je samotná funkce definovaná, lze překopírováním hlavičky funkce, tzv. prototypu, na začátek vytvářeného kódu definovat compileru s jakými proměnnými daná funkce pracuje a jakou má návratovou hodnotu.

Příkladem jednoduché funkce pro dělení může být zdrojový kód 2. `Deleni` je název funkce. Výsledkem funkce je proměnná typu integer (tj. 16-bitové celé číslo) a do funkce zadáváme dva parametry *a* a *b* typu integer.

```
int deleni(int a, int b)      //Hlavička funkce, kde a a b jsou vstupní
                              //proměnné
{                             //Začátek funkce
 int c;                       //Deklarace lokální proměnné c
 c=a/b;                       //Dělení a/b výsledek je uložen do c
 return c;                    //Výsledkem funkce deleni je hodnota v c
}                             //Konec funkce
```

Zdrojový kód 2: *Ukázka jednoduché funkce*

Při startu MCU se v jazyce C vždy začne vykonávat tzv. hlavní funkce. Hlavička hlavní funkce má následující tvar **`int`** `main(` **`void`** `)` a do ní se napíše program, který se má vykonávat po startu MCU.

Dále se u mikrokontrolérů automaticky volají funkce přerušení, které nastanou pokud některá periférie vyvolá přerušení. Zápis funkce přerušení je ve zdrojovém kódu 3. Hlavička funkce přerušení má následující tvar **`ISR`**(*`vektor`*). Kde *vektor* – je celé číslo určující, která periférie vyvolala požadavek přerušení a lze zjistit v dokumentaci k MCU. Například pokud se z dokumentace k vybranému mikrokontroléru zjistí, že třeba vektor přerušení při přetečení čítače 0 je roven 5, pak funkce **`ISR`**`(5)` se vykoná vždy při přerušení, které nastane při přetečení čítače 0.

```
ISR( nazev_preruseni ) // obsluha přerušeni
{
// Program obsluhy přerušení
}
```

Zdrojový kód 3: *Ukázka funkce přerušení*

Jako ukázka kompletního programu může sloužit zdrojový kód 4. Tento program provede dělení čísel a výsledek zobrazí na portu B. Potom se MCU zacyklí do nekonečné smyčky a již nic nevykonává. V případě externího přerušení od INT0 dojde k zvýšení globální proměnné `glob` o +1.

```
#include <avr\io.h>               //hlavičkový soubor pro použitý
                                  //mikrokontrolér
#include <avr\interrupt.h>        //hlavičkový soubor pro obsluhu
                                  //přerušení

char deleni(char a, char b);      //Hlavička funkce deleni (tzv. prototyp)
                                  //musí být uvedena na začátku
int glob=0;                       //Deklarace globální proměnné glob

ISR( INT0_vect )                  //Externího přerušeni INT0
{
glob++;                           //Program vykonávající se při přerušení
                                  //hodnota proměnné glob se zvýší o +1
}
```

```
int main( void )              //Hlavní funkce - do ní umístíme program
{
// kód hlavní funkce
//       ..
//       ..

DDRB = 0xff ;                 //Nastavení portu B na výstupní
PORTB = deleni(12,2);         // Na portu B se zobrazí výsledek funkce
                              //deleni

sei();                        //povolení globálního přerušení
while(1);                     //zacyklení do nekonečné smyčky
}                             //konec hlavní funkce

char deleni(char a, char b)   //Hlavička funkce, kde a a b jsou vstupní
                              //proměnné
{                             //Začátek funkce
 char c;                      //Deklarace lokální proměnné c
 c=a/b;                       //Dělení a/b výsledek je uložen do c
 return c;                    //Výsledkem funkce deleni je hodnota v c
}
```

Zdrojový kód 4: *Ukázka programu v jazyce C*

4.2.6 Specifika jazyka C pro MCU AVR

Výše popsaná pravidla jazyka C jsou společná pro většinu platforem a vytvořený kód by měl být částečně přenositelný i na jiné počítače a MCU. V jazyce C pro programování AVR (zde norma GCC) existuje několik dalších specifik daných architekturou mikrokontrolérů AVR. Na následujících řádcích budou některá specifika popsána.

Funkce, které se mají vykonávat během přerušení, se zapisují ve tvaru **`ISR`**(*`vektor`*). Podrobný popis tvorby funkce přerušení je popsán v kapitole 4.2.5, proto zde již nebude detailně rozepisován.

Pro práci s pamětí programu existuje knihovna `pgmspace.h`. Zde jsou definovány funkce pro čtení dat z paměti programu. Čtení lze provádět například pomocí funkce `pgm_read_byte_near(address_short)`. Data, která se mají uložit do paměti programu se deklarují pomocí datového typu `PROGMEM`. Jako příklad slouží zdrojový kód 5. Řetězec `string` je uložen v paměti programu a pomocí funkce `pgm_read_byte_near` je do proměnné `znak` načten druhý znak ‘e’.

```
char znak;                          //deklarace proměnné znak
PROGMEM char string[]="Hello";      //deklarace pole v paměti programu
znak=pgm_read_byte_near(string+1); //načtení 2.znaku do proměnné znak=‘e‘
```

Zdrojový kód 5: *Ukázka práce s pamětí programu*

Pro práci s EEPROM složí knihovna `eeprom.h`. Práce s ní je obdobná jako práce s pamětí programu. Knihovna obsahuje popisy jednotlivých funkcí pro použití EEPROM. Za zmínku stojí především funkce `eeprom_read_byte (const uint8_t *addr)`, která slouží pro čtení bajtu z EEPROM a `eeprom_write_byte (uint8_t *addr,uint8_t value)`, sloužící pro zápis dat do EEPROM.

Chce-li programátor do svého programu vkládat kód jazyka asembler, lze to provést pomocí zápisu: `__asm__ volatile (`*`kód`*`);` kde *kód* je program napsaný pomocí asembleru a po kompilaci bude vložen do programu.

5 Překladače

K vytváření programů pro mikrokontroléry AVR existuje několik vývojových prostředí. Liší se jak svými vlastnostmi, tak i cenou, za kterou se dají pořídit. Ke každému prostředí jsou uvedeny jen ty nejzákladnější vlastnosti, které o daném programu byly zjištěny. V případě zájmu se téměř každý z programů dá stáhnout v omezené verzi z internetu.

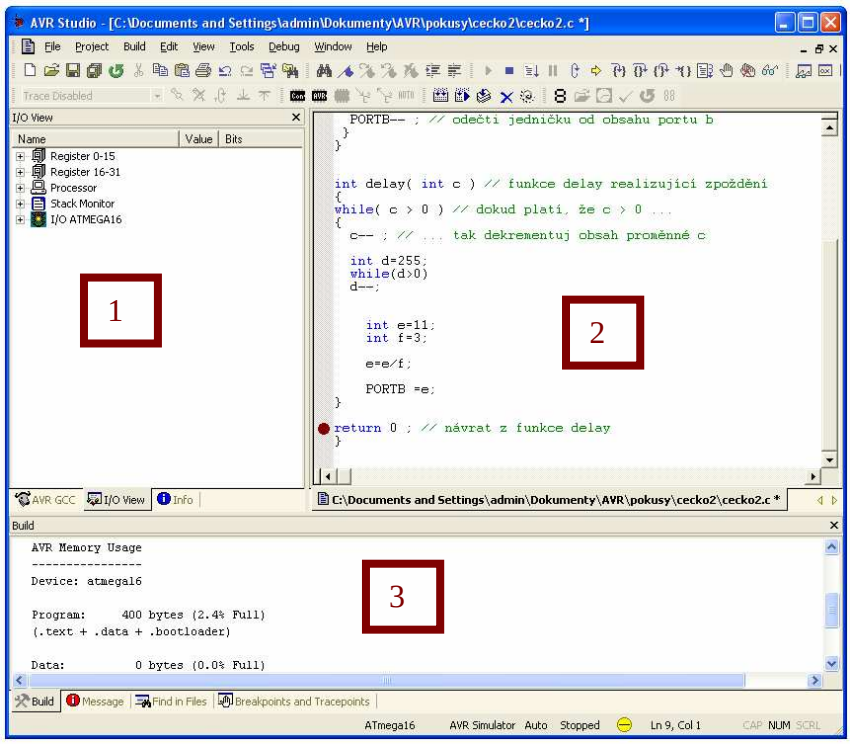
5.1 AVR Studio

AVR Studio je integrované vývojové prostředí pro vývoj programů pro procesory ATMEL AVR (v jazyce asembler) s možností integrace překladačů jazyka C. Prostředí obsahuje rovněž simulátor procesorů AVR a přímo podporuje základní druhy ladících nástrojů ATMEL [7].

Hlavní výhodou AVR Studia je, že je poskytováno bezplatně, a tudíž odpadají náklady na software. Dále AVR Studio obsahuje Debugger, pomocí kterého si můžeme funkci vytvořeného programu odsimulovat. Rovněž obsahuje software, umožňující připojit některé programátory, pomocí nichž se může vytvořený program hned nahrát do mikrokontroléru.

Pro správnou funkci musíme nejprve nainstalovat GNU překladač jazyka C pro AVR procesory. Překladač jazyka C se nazývá WinAVR a dá se nalézt na internetové adrese [10].

Ukázka okna programu AVR Studia je na obrázku 7. V části 1) jsou zobrazeny jednotlivé periférie a registry a jejich aktuální hodnoty. Část 2) je textový editor, do kterého se píše vlastní program. V části 3) se zobrazují informace (například o průběhu kompilace).



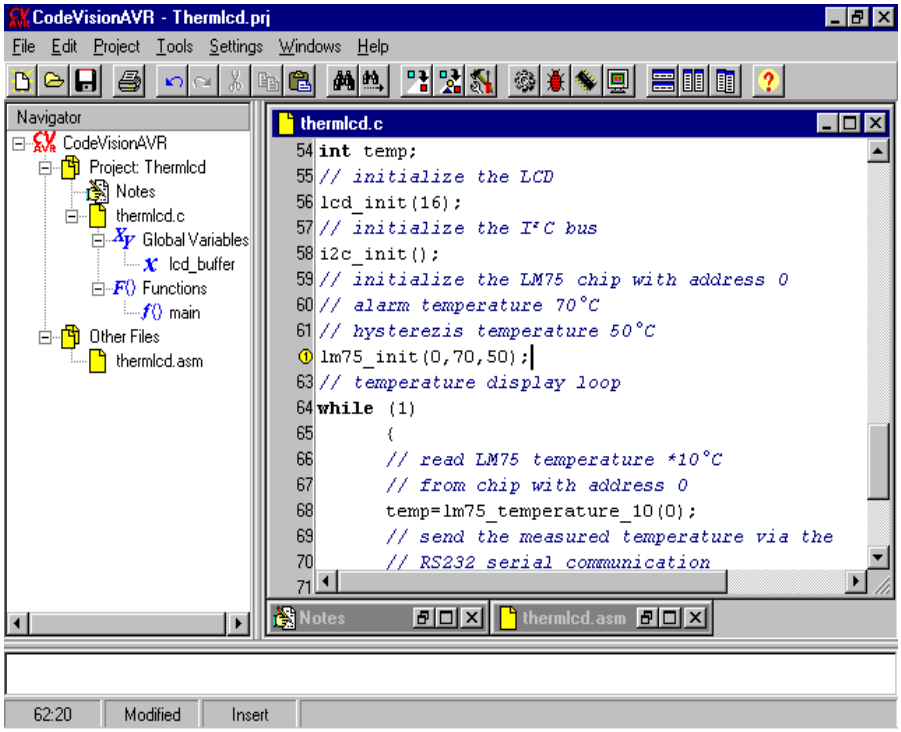
Obr. 7: *Vzhled programu AVR Studio*

5.2 CodeVisionAVR

Kompilátor CodeVisionAVR pochází od Pavla Haiduca z HP Infotech S.R.L [5]. Jedná se o plnohodnotný vývojový systém pro mikroprocesory AtmelAVR. Jeho hlavní předností je „projekt wizard“, který umí vygenerovat kostru programu v jazyce C podle zadaných používaných periférií. Bohužel tento program je komerční (omezen na 2kB výsledného kódu) a nepodporuje GCC používaného u AVR Studia. Proto vznikl tento projekt, který má vyústit v diplomovou práci, mající za úkol vytvořit obdobný „projekt wizard“ pro vývojové prostředí AVR Studio, nabízené firmou Atmel. Ukázka programu je uvedena na obrázku 8.

Dále CodeVisionAVR obsahuje různé knihovny pro práci s vnějšími zařízeními, jako jsou LCD displeje, hodiny reálného času (RTC), teplotní snímače, sériový přenos pomocí UART, SPI, atd.

Jedná se o komerční program, ale existuje i bezplatná verze, která je omezena možnou velikostí zdrojového kódu (2kB). Program se prodává ve dvou verzích. „Standart“ stojí kolem 5000Kč a „Light“ lze zakoupit za 3000Kč.



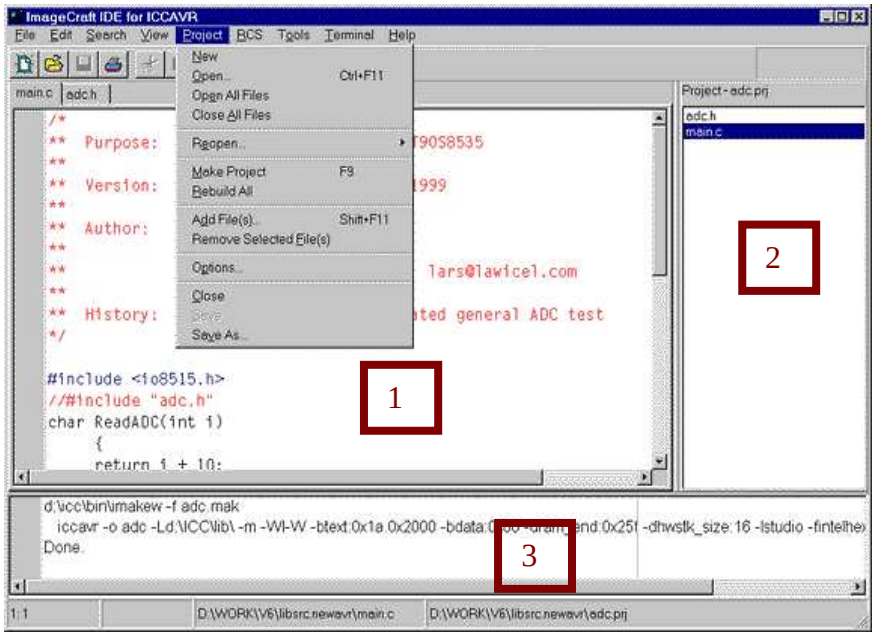
Obr. 8: *Vzhled programu CodeVisionAVR*

5.3 ImageCraft

ImageCraft je nástroj pro vývoj aplikací v jazyce C [6]. Jedná se o integrované vývojové prostředí (IDE), obsahující mimo jiné optimalizační nástroj výsledného kódu, který dokáže zmenšit zdrojový kód až o 20%. Tento překladač slouží k programování procesorů Atmel AVR, Motorola, Texas Instrument a Cypress MicroSystems. Vývojové prostředí obsahuje AVRCalc, což je program sloužící k výpočtu rychlosti sériové linky, dále slouží k nastavování čítačů/časovačů na požadované časy. Program Application Builder pomáhá s inicializací všech periferií mikroprocesoru, bez zbytečného zdlouhavého hledání v katalogových listech - například pokud chceme používat sériovou linku, zadáme tyto

informace do Application Builderu a ten nám vytvoří v okně Edit zdrojový kód v C se zápisem do všech potřebných registrů.

Na obrázku 9 je program rozdělen do tří oblastí 1) “editor”, 2) “project file list” a 3) “status window”, podle následujícího obrázku. V okně editoru se zapisuje vlastní program. V okně “project file list” jsou zobrazeny všechny soubory použité při vývoji programu. Stavové okno zobrazuje zprávy o překladu zdrojového kódu. Tato dvě poslední okna lze skrýt, aby bylo editorové okno co největší.

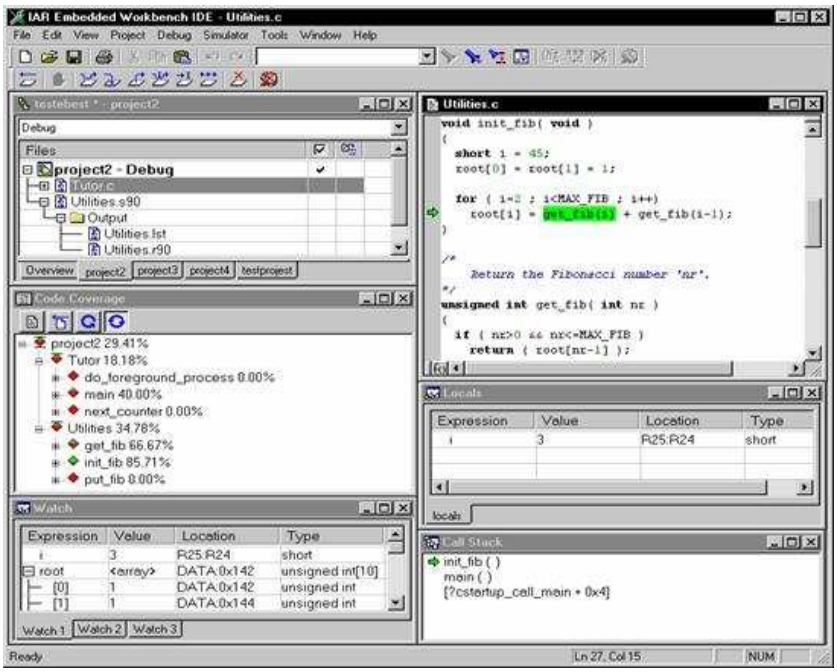


Obr. 9: Vzhled programu ImageCraf

5.4 IAR Embedded Workbench

IAR Embedded Workbench je komerční software, který kromě mikrokontrolérů AVR umí vytvářet programy pro většinu dalších mikrokontrolérů [4]. Jsou to například x51, PIC, Motorola, atd…

Vývojové prostředí (obrázek 10) obsahuje Compiler C, pracuje s asemblerem a obsahuje Debugger.

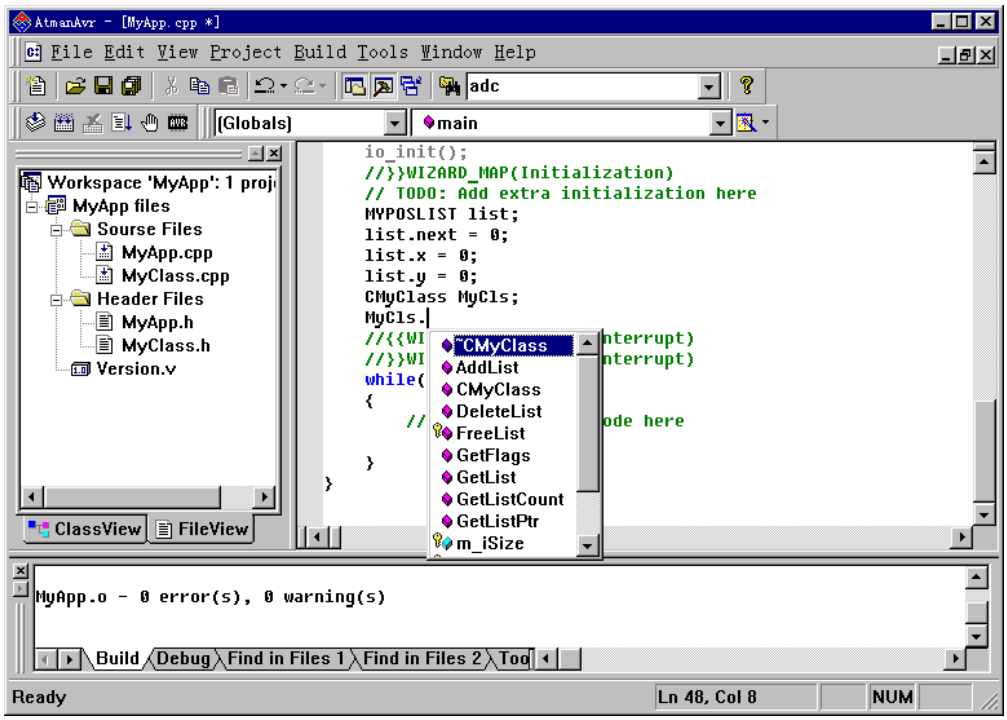


Obr. 10: *Vzhled programu IAR Embedded Workbench*

5.5 AtmanAvr C

AtmanAvr je výkonný komerční IDE C/C++ kompilátor pro rodinu mikrokontrolérů Atmel AVR. AtmanAvr obsahuje různé knihovny pro snadné vytváření programů. Dále se dá pomocí tohoto IDE snadno ladit a konfigurovat vytvořený software. Ukázka tohoto programu je na obrázku 11. Cena softwaru se pohybuje kolem 100$ [11].

Vývojové prostředí AtmanAvr C zahrnuje ProjectWizard (automatická generace kódu), CodeWizard (programovací asistent napomáhající k snadnému programování), WorkSpace (pracovní plocha projektu informující o I/O, registrech, průběhu kompilace, atd…), Textový editor (zvýrazňuje syntaxi, poznámky…), Binární editor (slouží k editaci souborů v hexa nebo ascii formátu), Debugger (simulace periférií), atd.



Obr. 11: *Vzhled programu IAR AtmanAvr C*

6 XML a INI soubory

Důležité informace o jednotlivých mikrokontrolérech jsou v AVR Studiu uloženy v XML souborech, které se nalézají v adresáři `\Partdescriptionfiles\`, jenž se nachází v kořenovém adresáři instalace AVR Studia. Informace, která se nenalézají v XML souborech, byly zapsány do INI (Inicializačních) souborů a slouží pro doplňkové informace o mikrokontroléru k primárním datům z XML souborů.

6.1 Úvod do XML

XML je zkratka eXtensible Markup Language, což v překladu znamená rozšiřitelný značkovací jazyk. U značkovacího jazyku je každá informace ohraničena značkou (anglicky tagem), která nám specifikuje, co daná informace znamená.

Výhodou XML jazyka je, že soubory jsou uloženy v textovém tvaru, který je společný pro různé aplikace nebo operační systémy, z čehož vyplývá kompatibilita při zpracování v různých systémech. Avšak při užití textových souborů nastává problém s různými národnostními znaky; proto prvním parametrem v XML souborech je parametr `encoding` udávající v jakém textovém kódování je daný dokument uložen. Pokud se v souboru tento parametr nevyskytuje, je použito kódování UTF-8, což je normální anglická abeceda.

Každá informace je zahájena tagem umístěným v ostrých závorkách `<znacka>` a ukončena tagem s lomítkem `</znacka>`. U tagů se nepoužívá diakritika. Data v XML jazyku mohou vypadat následovně: `<cena>0,50</cena>`, kde 0,50 jsou data rovnající se ceně. [2]

6.2 Skladba XML

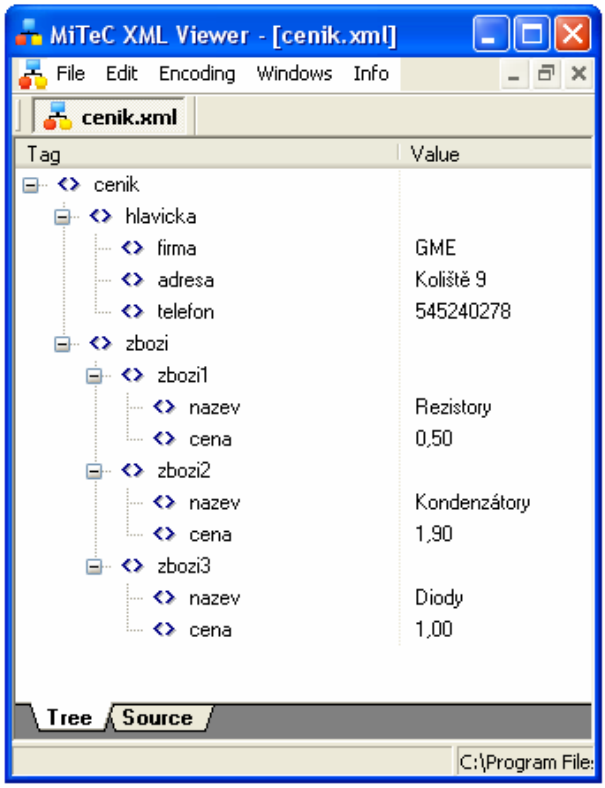
Pro příklad bude uveden zdrojový kód 6, na němž bude vysvětlena skladba XML jazyka.

```
<?xml version="1.0" encoding="Windows-1250"?>
 <cenik>
  <hlavicka>   <firma>GME</firma>
   <adresa>Koliště 9</adresa>
   <telefon>545240278</telefon>
  </hlavicka>
  <zbozi>
   <zbozi1>
    <nazev>Rezistory</nazev>
    <cena>0,50</cena>
   </zbozi1>
   <zbozi2>
    <nazev>Kondenzátory</nazev>
    <cena>1,90</cena>
   </zbozi2>
   <zbozi3>
    <nazev>Diody</nazev>
    <cena>1,00</cena>
   </zbozi3>
  </zbozi>
 </cenik>
```

Zdrojový kód 6: *Ukázka programu v XML*

V prvním řádku: `<?xml version="1.0" encoding="Windows-1250"?>` je uvedena verze XML a kódování. Pokud by byl tento řádek vynechán, je kódování nastaveno na UTF-8. Následuje tag `<cenik>`, který udává začátek ceníku. K němu se na posledním řádku nachází ukončovací tag `</cenik>`, jenž značí konec informací o ceníku. Obdobně jsou v ceníku další tagy, které ohraničují další informace. Například `<hlavicka>` udávající začátek informací o firmě, která daný soubor používá, atd…

Asi nejsnadnější pro představu je si jednotlivé tagy představit jako stromovou strukturu, která se větví podle jednotlivých značek. Potom by vzorový zápis, na kterém byla vysvětlována struktura XML souborů, vypadal jak je zobrazeno na obrázku 12 [12].



Obr. 12: *Ukázka XML souboru zobrazeného pomocí stromové struktury*

6.3 Popis INI souborů

Dalším datovým souborem, ze kterého se načítají informace, které nejsou uloženy v *.xml souborech AVR Studia, jsou *.ini (Inicializační) soubory. Tento typ souboru byl zvolen hlavně z důvodu snadné editace, díky které není problém v libovolném textovém editoru doplnit informační data o novém typu MCU, případně pozměnit nebo opravit chybná / neaktuální data.

Každý INI soubor je rozčleněn na několik sekcí, které oddělují od sebe jednotlivé údaje. Název sekce se píše do hranatých závorek []. Pod názvem sekce pokračují názvy údajů

a hodnoty údajů (hodnota je oddělena od údaje pomocí znaku =). V jedné sekci se daný název údaje může vyskytovat pouze jednou, ale v jiné sekci může být použit znovu.

Jako příklad slouží zdrojový kód 7. Jedná se o INI soubor, ve kterém jsou tři sekce [Typ_ATMega128], [Typ_ATMega16] a [Typ_ATMega161]. Každá sekce obsahuje údaje o časovači 0 až n, typu předděličky watchdogu a kmitočet hodin oscilátoru watchdogu v hertzích. Z uvedeného příkladu lze vidět, že v případě uvedení na trh nového MCU není problém v libovolném editoru dopsat do INI souboru novou sekci nazvanou podle nového mikrokontroléru a z dokumentace k danému obvodu doplnit údaje a parametry.

```
[Typ_ATMega128]
timer0=2
timer1=1
timer2=1
timer3=1
watchdog=1
wd_osc=1130000
[Typ_ATMega16]
timer0=1
timer1=1
timer2=2
watchdog=1
wd_osc=998000
[Typ_ATMega161]
timer0=1
timer1=1
timer2=2
watchdog=1
wd_osc=998000
```

Zdrojový kód 7: *Ukázka INI souboru*

7 Popis aplikace „AVR Wizard“

Hlavním cílem této práce bylo vyvinout aplikaci (tzv. „project wizard“) pro usnadnění práce programátora programujícího mikrokontroléry AVR od firmy Atmel [7], který pracuje v jazyce C s překladačem GCC. Software umí po výběru obvodu a příslušných periférií vygenerovat kód v jazyce C pro programovací prostředí AVR Studio.

Ve finální verzi programu lze nastavovat porty, watchdog, časovače/čítače, externí přerušení, přerušení od skupiny pinů, předděličku hodin, AD převodník, komparátor, SPI a TWI linku a sériovou linku USART. Aplikace je především odladěná pro obvod ATmega16, ale dá se použít i pro jiné mikrokontroléry řady ATmega a ATtiny.

Načítání informací probíhá ze souborů *.xml obsahujících informace o obvodu pro program AVR Studio. Princip zjišťování jednotlivých periférií je takový, že se zjišťuje existence bitů v souboru *.xml obsahujícím data o vybraném mikroprocesoru, které nám dané zařízení povolují a pokud takový bit existuje, musí být i u mikrokontroléru použita hledaná periférie. Proto by měl být tento způsob univerzální pro většinu typů obvodů, které AVR Studio podporuje. Dále informace které nejsou v *.xml souborech obsaženy, jsou načítány z interních *.ini souborů, které obsahují doplňkové nabídky (například možnosti vstupů pro měření pomocí AD převodníku).

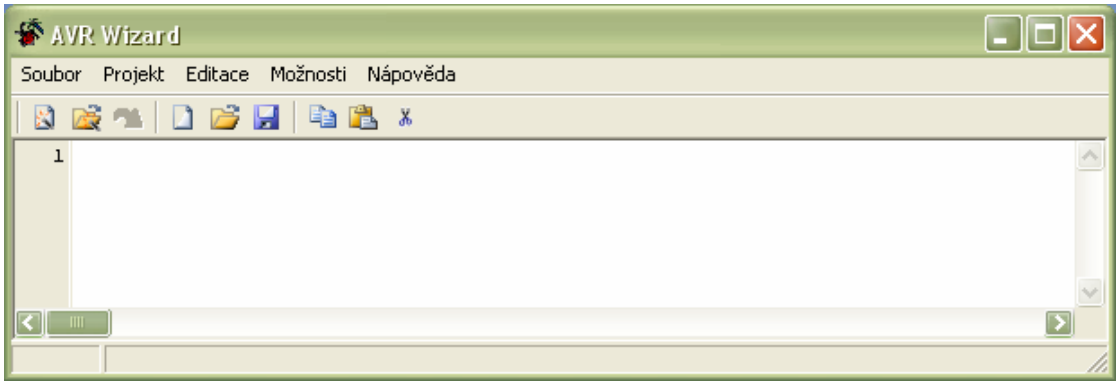
7.1 Textové okno

Při spuštění aplikace Project Wizard (vytvořené v aplikaci C++ Builder) se nejprve otevře textové okno. Do tohoto okna lze vypisovat program v jazyce C (GCC). Program má pro větší přehlednost výsledného kódu ošetřeno zvýrazňování syntaxe. V případě potřeby lze okno využít jako editor pro programování softwaru, ale protože se jedná jen o pomocný nástroj k aplikaci AVR Studio, neobsahuje kompiler, ani debuger na odladění programu.

Po spuštění programu lze pomocí nabídky **Projekt → Nový projekt** nebo klikem myši na znak čistého papíru s kouzelnou hůlkou  otevřít nabídku s výběrem programovaného obvodu a začít v novém projektu. Popřípadě pokud uživatel chce pokračovat v uloženém projektu, tak v nabídce **Projekt → Otevři projekt** či stiskem symbolu složky s kouzelnou hůlkou  , otevřít dříve uložený projekt. Pokud je v textovém okně vytvořený kód, může ho uživatel do AVR Studia přenést buď zkopírováním textu do schránky, které lze provést stiskem v hlavním menu na položku **Editace → Kopíruj vše**, nebo uložením textu do souboru pomocí **Soubor → Ulož** či kliknutím na symbol diskety  . Soubor se uloží jako obyčejný textový dokument, který lze otevřít pomocí většiny editačních nástrojů.

V horní nabídce je možno v položce **Editace** editovat vytvořený kód. V menu **Možnosti** lze provést výběr jazyka, otevřít okno s nastavením programu, nastavit zvýrazňování syntaxe a výběr písma. V menu **Nápověda → O programu** je obsažen seznam podporovaných obvodů.

Ukázka hlavního okna je zobrazena na obrázku 13.



Obr. 13: *Hlavní okno programu Project Wizard*

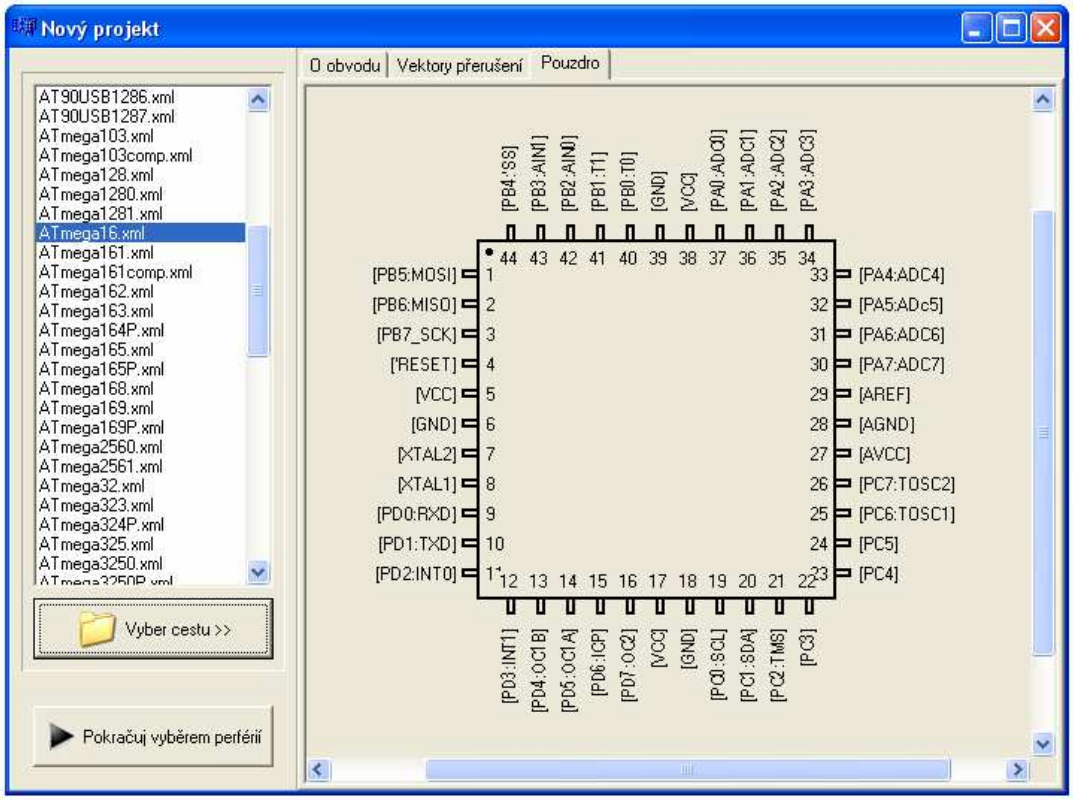
7.2 Okno pro výběr obvodu

Po výběru nového projektu v hlavním okně (viz. kapitola 7.1) se otevře nabídka s podporovanými obvody, jež je zobrazena na obrázku 14. Pokud je standardně nainstalováno AVR Studio, zobrazí se nabídka s obvody bez problémů. Pokud ne, musí se vybrat stiskem na tlačítko s názvem **Vyber cestu** a ručně nastavit složku, kde se vyskytují *.xml informační soubory. (Běžná cesta je v kořenový adresář instalace AVR Studia a v něm složka `\Partdescriptionfiles\`).

Po správném nastavení cesty se může z nabídky vybrat požadovaný obvod. V pravé části programu se zobrazují informace o aktuálním mikrokontroléru. V záložce **O obvodu** jsou popsány velikosti jednotlivých pamětí a dále se zde může nastavit kmitočet oscilátoru, plánovaný v budoucí aplikaci. Při správně nastaveném kmitočtu program bude vypisovat u některých položek i doby trvání dějů, které jsou závislé na zvoleném krystalu.

V záložce **Vektory přerušení** jsou popsány jednotlivé vektory přerušení daného obvodu. V poslední záložce **Pouzdro** je vykresleno pouzdro zvoleného mikrokontroléru. Některé typy mikroprocesorů se vyrábí i ve více pouzdrech, než je možno v nabídce vybrat, ale program umí vykreslit jen ta pouzdra, která jsou definována v *.xml souboru k danému obvodu. Při správném zvolení pouzdra program vypisuje čísla pinů kde daný port leží a zobrazuje vybranou alternativní funkci.

Stiskem na tlačítko **Pokračuj výběrem periférií** v spodním levém rohu aplikace se pokračuje ve výběru používaných periférií mikrokontroléru.



Obr. 14: *Okno pro výběr obvodu*

7.3 Nastavení periférií

Po stisku tlačítka **Pokračuj výběrem periférií**, které bylo popsáno v kapitole 7.2, se otevře okno s názvem Nastavení periférií podobné jako je zobrazeno například na obrázku 15. Toto okno obsahuje záložky s obvody, které program umí nastavit. V horní části aplikace je název vybraného obvodu, který slouží pro kontrolu, zda došlo k výběru správného mikrokontroléru.

Ve spodní části programu jsou čtyři tlačítka. První **Zpět** slouží pro opětovný návrat do okna pro výběr obvodu. Dalším tlačítkem zleva je **Ulož projekt**, po jehož stisku je možno uložit rozpracovanou práci do souboru. Dále následuje tlačítko **Vytvoř kód**, které slouží ke vygenerování kódu podle vybraného nastavení periférií do hlavního okna. Poslední pravé tlačítko **Konec** uzavře okno pro nastavení periférií.

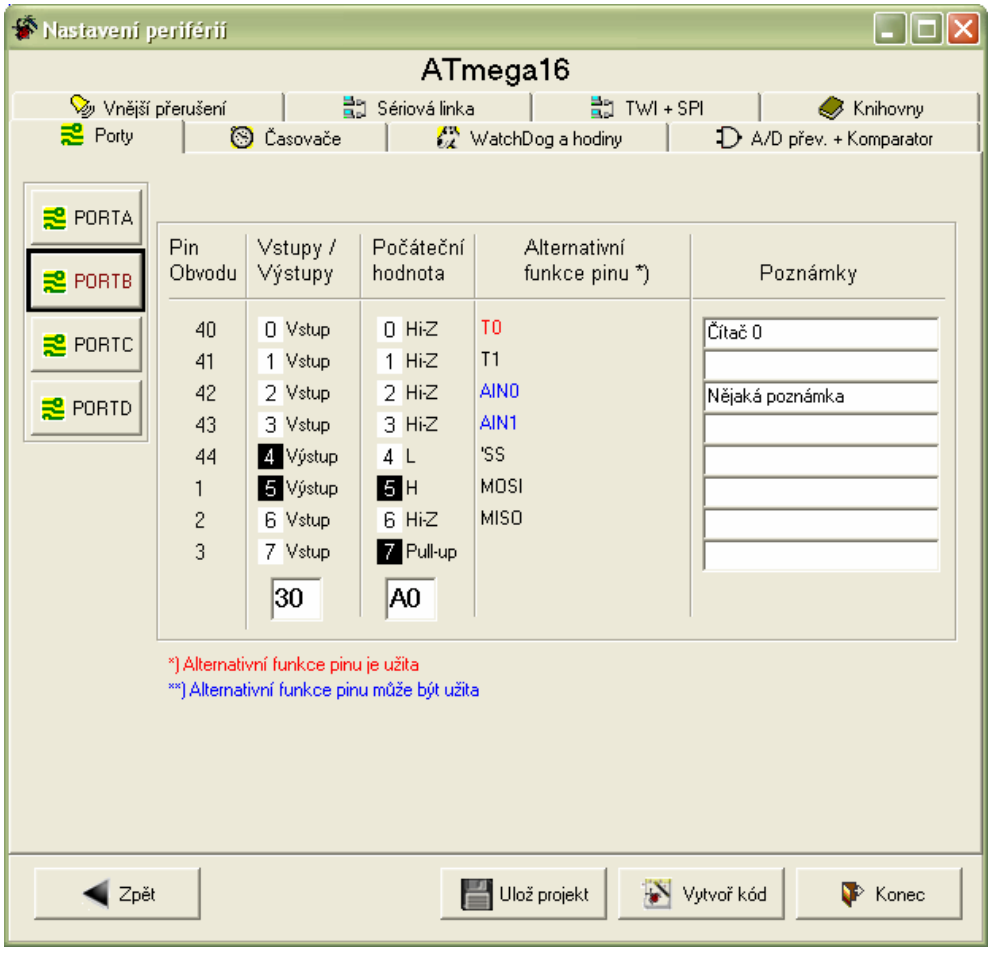
7.3.1 Porty

První položkou mezi záložkami jsou **Porty** - viz. Obrázek 15. V levé části programu jsou zobrazeny porty, které mikrokontrolér obsahuje. Kliknutím na jednotlivá tlačítka přepínáme mezi zvolenými bránami.

Ve zbylé části aplikace je tabulkovou formou vytvořeno nastavení jednotlivých pinů portu. Ve sloupci **Pin obvodu** je napsáno, ke kterému pinu pouzdra je daný bit připojen.

Ve sloupcích **Vstupy / Výstupy** a **Počáteční hodnota** se nastavuje hodnota registrů DDRn a PORTn. Tyto hodnoty nám určují zda daný pin má být vstupní nebo výstupní, jestli na výstupu má být úroveň „H“ či „L“ a zda-li ke vstupu má být připojen pull-up rezistor nebo má vstup vykazovat vysokou impedanci.

Sloupec **Alternativní funkce pinu** slouží k informaci uživatele o dalších možnostech daného vývodu. Programátor si včas může rozmyslet, zda-li není vhodnější použít jiný port, pokud plánuje využívat jinou funkci pinu. Pokud v některé záložce povolí nějakou alternativní funkci (například čítání čítače), ve sloupci **Alternativní funkce pinu** se mu daná položka zbarví a tím upozorní na své možné využití. Poslední položka **Poznámky** slouží k zaznamenání poznámky k funkci daného pinu. Programátor si pak i po čase snadněji vzpomene, k jaké funkci daný vývod zamýšlel.



Obr. 15: *Nastavení portů mikrokontroléru*

7.3.2 Časovače

Při stisku na položku **Časovače** se otevře nabídka pro nastavení čítačů/časovačů (obrázek 16). V levé části programu (opět jako u nastavení portů) jsou zobrazena tlačítka s názvy Č/Č, které vybraný mikrokontrolér obsahuje.

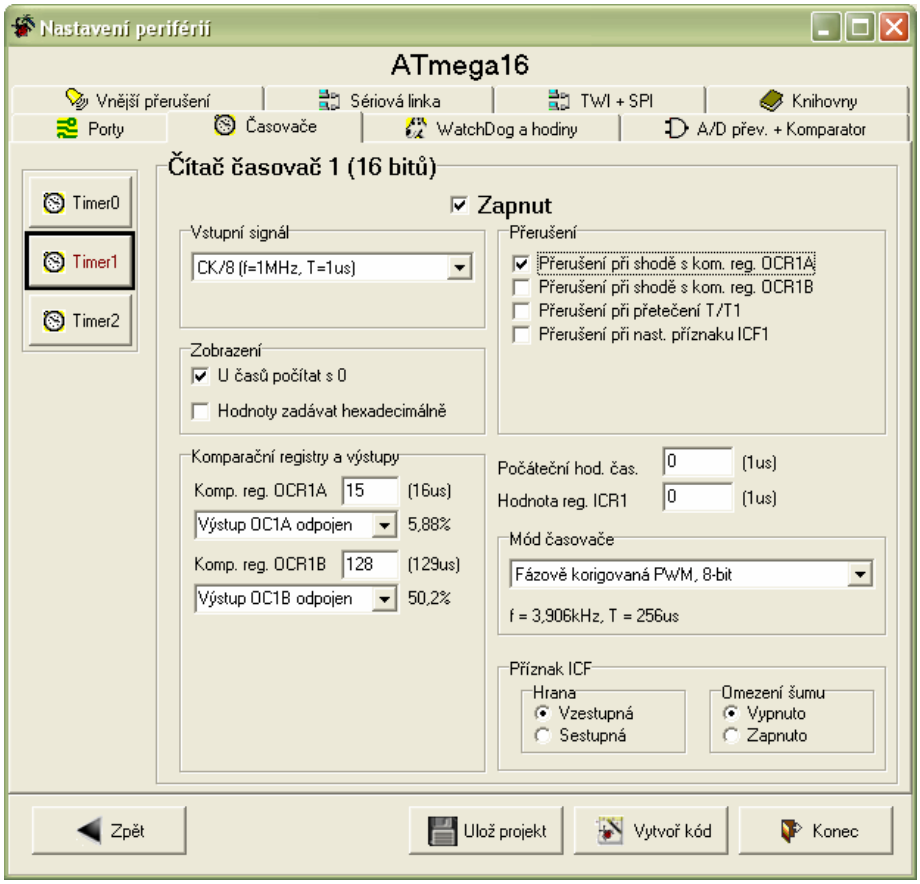
V hlavní části aplikace jsou různé tabulky, pomocí nichž se provádí nastavení čítače/časovače. Pokud se bude Č/Č používat, musí se zapnout CheckBoxem **Zapnut**. Dále pomocí výběru **Vstupní signál** (levý horní roh) se provede výběr zdroje čítaného kmitočtu.

Ve skupině **Přerušení** (Pravý horní roh) se provede výběr přerušení, která se budou u zvoleného čítače/časovače používat.

V nabídce **Komparační registry a výstupy** se provede naplnění komparačních registrů zvolenou hexadecimální hodnotou a nastaví se módy výstupů **OCnx**. Obdobně se nastaví položka **Počáteční hodnota čítače**, která přednastaví registr **TCNTn**.

U **módu časovače** se vybere způsob generace PWM. Pokud se plánuje použití zachytávání hodnoty Č/Č do záchytného registru (buď od analogového komparátoru nebo při změně na pinu **ICP**) u nabídky **Vstup** (spodní pravý roh okna), nastaví se reakce na vzestupnou či sestupnou hranu a zda-li má být použito omezení šumu.

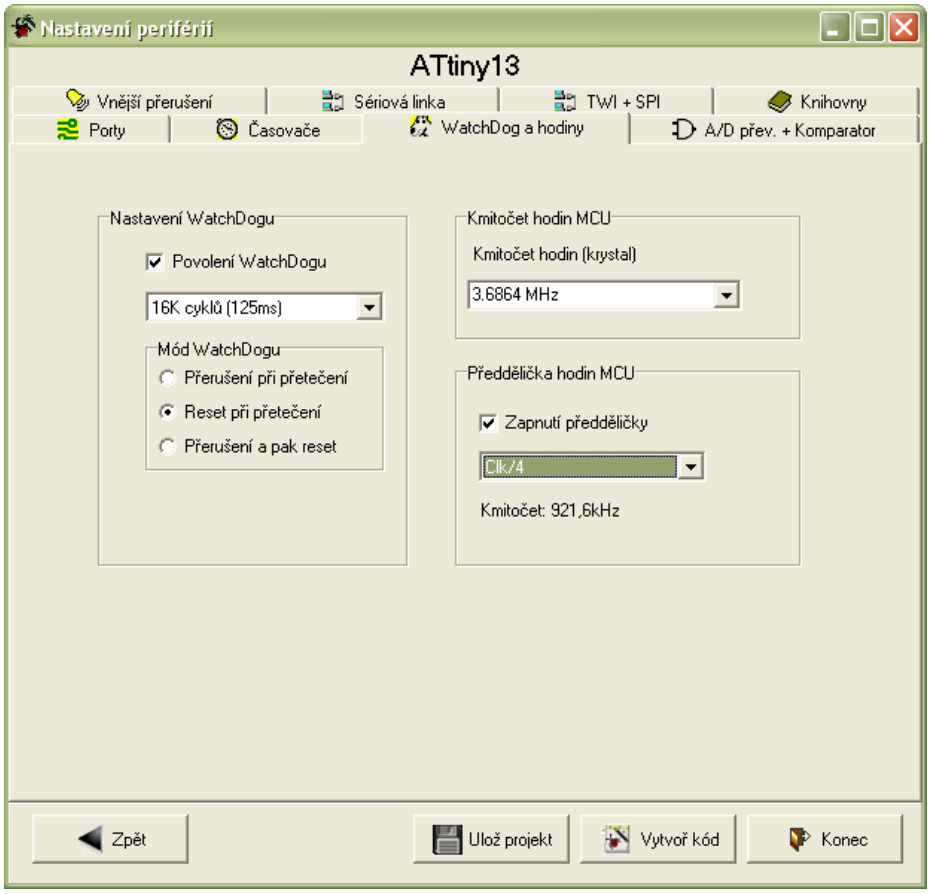
U většiny nabídek se zobrazují časy pro snadnější návrh použití čítače. Zde popsané nabídky se vždy nemusí vyskytovat, záleží na tom, jestli vybraný mikroprocesor vůbec danou možnost nastavení podporuje.



Obr. 16: *Nastavení Čítače/Časovače*

7.3.3 Watchdog a hodiny

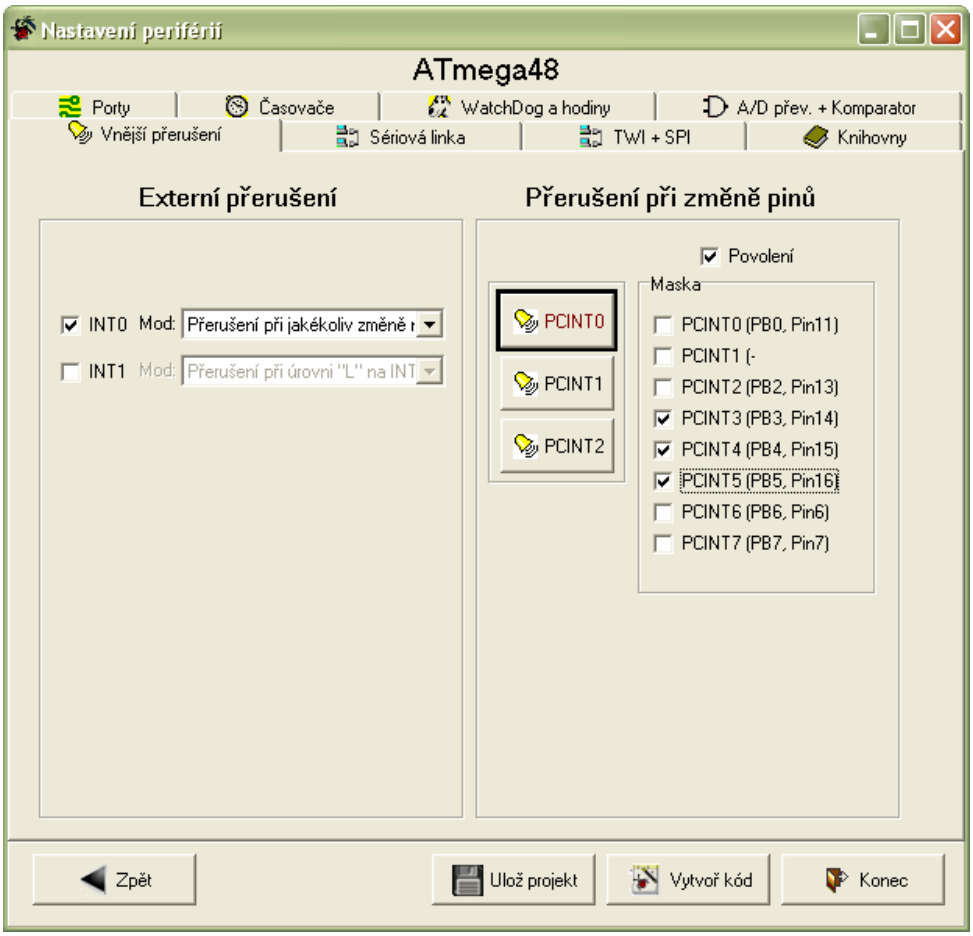
Záložka **Watchdog a hodiny** slouží k nastavení watchdogu, výběru kmitočtu hodin MCU a u mikrokontrolérů, které podporují předděličku systémových hodin, lze nastavit předděličku. Ukázka okna s možným nastavením je na obrázku 17. Kmitočet hodin slouží pouze k informativním účelům, jako jsou výpočty doby trvání časově závislých dějů (např.: jak často dojde k přetečení časovače). U položky watchdog lze nastavit po jak dlouhém časovém úseku nečinnosti nastane přetečení čítače watchdogu a když dojde k přetečení, zda má nastat přerušení nebo reset. Pomocí předděličky lze softwarově měnit kmitočet hodin MCU a tím nastavovat rychlost zpracovávání instrukcí a spotřebu mikrokontroléru (čím vyšší kmitočet, tím větší spotřeba).



Obr. 17: *Nastavení Watchdogu a hodin*

7.3.4 Vnější přerušení

Záložka **Vnější přerušení** je rozdělena na dvě poloviny (viz. obrázek 18). V levé se nastavují možnosti klasického externího přerušení (kapitola 3.2.1), které nastane pokud se na příslušném vývodu změní podle nastavení úroveň napětí. Napravo se vybírají vývody pro přerušení od skupiny pinů (popsané v kapitole 3.2.2). Skupina pinů je vždy maximálně osm vývodů, kdy lze vybrat jeden až všech osm vývodů, na kterých pokud nastane změna napětí, dojde k příslušnému přerušení.

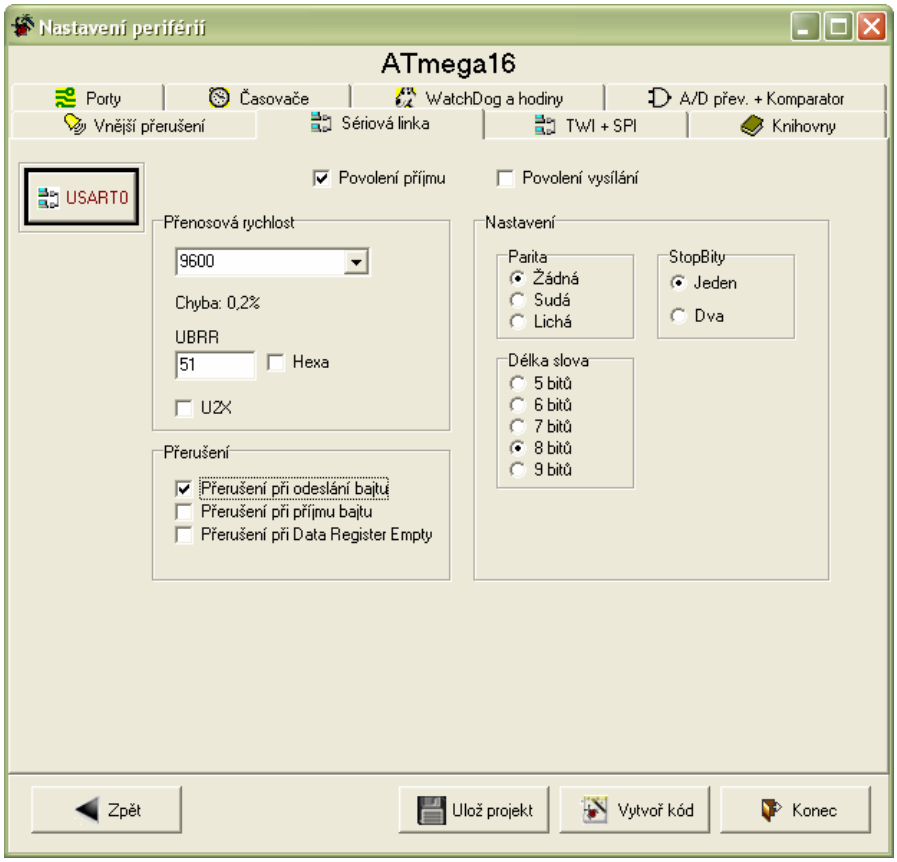


Obr. 18: *Možnosti u vnějšího přerušení*

7.3.5 Sériová linka

Záložka nastavení sériové linky je na obrázku 19. Zde lze nastavit možnosti asynchronního sériového přenosu pomocí linky USART.

Nejprve se vybere, zda má být povolen příjem a vysílání. Dále si uživatel zvolí přenosovou rychlost a z ní se vypočítá hodnota registru **UBRR**. U parametrů přenosu lze měnit počet bitů ve vysílaném slově, počet stopbitů a paritu. U sériové linky existují tři typy přerušení. Samozřejmostí je výběr jednoho až všech tří přerušení.



Obr. 19: *Nastavení sériové linky*

7.3.6 A/D převodník a Komparátor

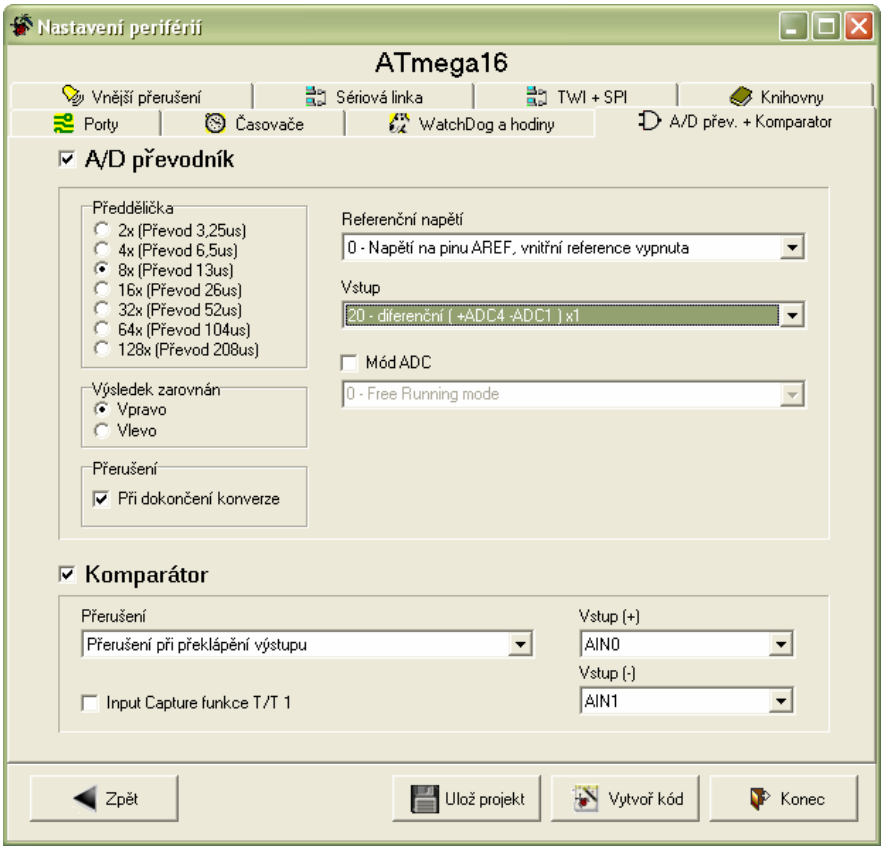
Pro nastavení A/D převodníku a komparátoru slouží společná záložka. V horní polovině se nastavuje A/D převodník a dolní polovině jsou možnosti pro správnou činnost komparátoru.

Po výběru A/D převodníku, který se zapíná checkboxem **A/D převodník,** se zpřístupní nabídky nastavení. V horní položce **Předdělička** se vybere rychlost převodu a dále lze nastavit, zda má být výsledek převodu v registrech ADCL a ADCH zarovnán směrem vlevo či vpravo. Pro správný převod je nutno vybrat na který vstup je přivedeno referenční

napětí a na kterém pinu má probíhat samotné měření napětí. Popřípadě pokud je třeba, aby převodník fungoval v některém speciálním režimu (například při ukončení převodu se převodník nevypíná, ale pokračuje v dalším převodu), lze toto nastavení provést zaškrtnutím položky **Mód ADC** a z nabídky vybrat vhodný režim činnosti.

Nakonec lze převodníku vybrat přerušení, které nastane při dokončení konverze. Okno pro nastavení A/D převodníku a komparátoru je zobrazeno na obrázku 20.

U komparátoru lze nastavit při jaké události způsobené komparátorem má nastat přerušení a na které piny jsou komparační vstupy připojeny. Položka Input capture funkce T/T 1 složí k zastavování čítače v případě komparace.



Obr. 20: *Nastavení Komparátoru a AD převodníku*

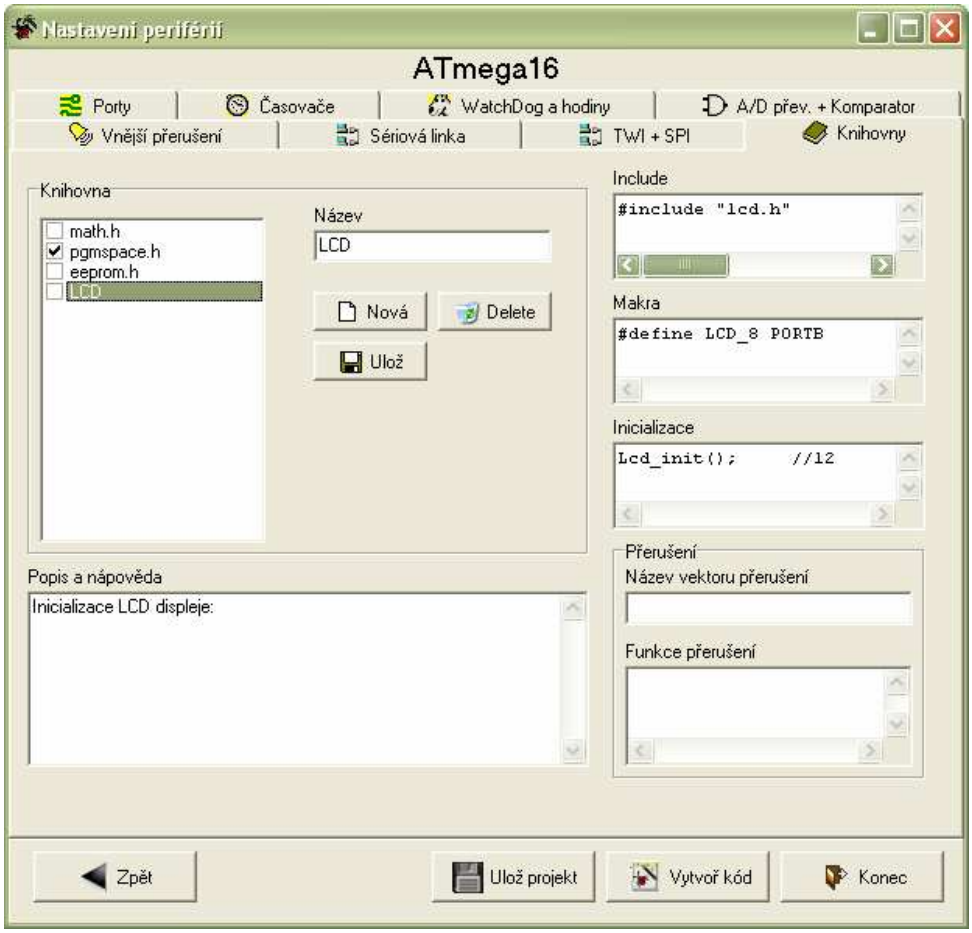
7.3.7 Knihovny

Pro tvorbu jednoduchých maker slouží položka knihovny, která programátorovi usnadní práci tím, že za něj předvyplní program o funkce, makra a definice knihoven, v nichž se nalézají funkce, které bude potřebovat v programu použít pro programování externího obvodu, jako je například LCD display a nebo různá teplotní čidla atd… Příklad záložky knihovny s předvyplněnými položkami je zobrazen na obrázku 21.

Položka knihovny obsahuje tři tlačítka. Tlačítko **Nová** pro vytvoření nové knihovny, **Smaž** pro smazání knihovny a **Ulož** pro uložení změn v knihovně. Jednotlivé knihovny se ukládají do adresáře `\LIB\` v aplikaci AVR Wizard. Soubory mají koncovku *.lib.

Pro nastavení knihoven se musí vyplnit aspoň jedna z následujících položek. První (nejhořejší) položkou je **Include**, zde se vyplní jaké knihovny se pomocí direktivy `#include` vloží do programu. Následuje okno **Makra**. Zde se definují direktivou `#define` konstanty potřebné pro správný chod jednotlivých knihoven vložených pomocí **Include**. Dále je položka **Inicializace**. Zde se vepíší funkce, které se mají vykonat v inicializaci před naběhnutím programu do hlavní smyčky. Nakonec, u **Přerušení**, se napíše název přerušení a dále kód programu v jazyce C. Dále se musí dané přerušení povolit a zapnout. Ve vygenerovaném kódu potom bude funkce k danému přerušení obsahovat předvyplněný kód. Například pokud název se zvolí `INT0_vect` a povolí se externí přerušení 0, pak ve funkci `ISR(INT0_vect)` bude napsaný předchystaný kód.

Položka **Nápověda** slouží pouze k informování uživateli o činnosti knihovny a nikde se ve vygenerovaném programu nezobrazuje.



Obr. 21: *Výběr knihoven*

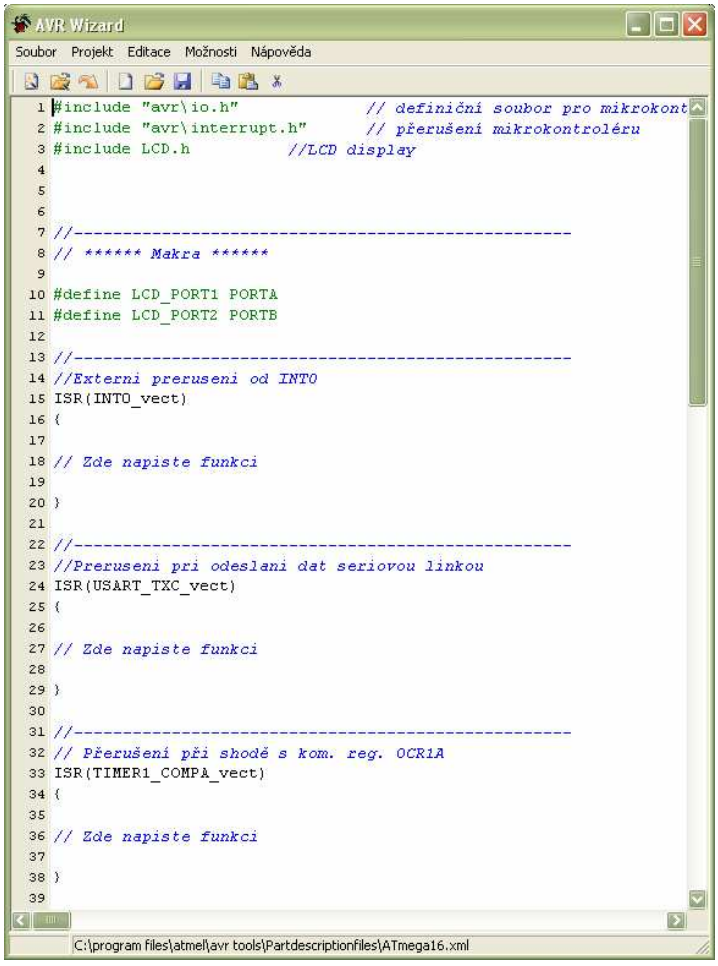
7.4 Vygenerovaný kód

Po stisku tlačítka „Generuj kód“ se do hlavního okna vygeneruje program, který přednastavuje jednotlivé periférní obvody. Nevýhodou je, že pokud je v hlavním okně již nějaký vytvořený kód, dojde během generace k jeho přepsání. Program AVR Wizard je totiž pouze pomocný nástroj a předpokládá se, že vygenerovaný kód je po vytvoření okamžitě zkopírován do AVR Studia a zde je s ním pracováno, a proto přepsání uživatelovi většinou nevadí.

Na obrázku 22 je hlavní okno obsahující automaticky vytvořený kód. Vygenerovaný vzorový kód je příliš rozsáhlý; nelze zobrazit v okně na obrázku 22, proto je kompletně rozepsán ve zdrojovém kódu 8.

Pro generaci bylo použito následující nastavení. Byl zvolen obvod ATMega16. PORTA byl nastaven jako výstupní. PORTB slouží jako vstupní s připojenými pull-up rezistory. Byl zapnut čítač 1 v režimu CTC (při shodě s komparačním registrem dojde k vynulování). Signál se čítá z hodin MCU podělí se předděličkou osmi a při shodě s komparačním registrem OCR1A dojde k vygenerování přerušení.

Dále je zapnuto externí přerušení INT0, které nastane při jakékoliv změně na pinu INT0. Sériová linka se používá pouze pro příjem a při odeslání bajtu dojde k přerušení. Přenosové parametry sériové linky jsou: 19200 baut/s, 8 datových bitů, jeden stop bit a parita není použita. Nakonec je zapnuta knihovna LCD, která provede přednastavení LCD displeje. Předvyplnění knihovny LCD je znázorněno na obrázku 21.



Obr. 22: *Hlavní okno s automaticky vytvořeným kódem*

```
#include "avr\io.h"           // definiční soubor pro mikrokontrolér
#include "avr\interrupt.h"    // přerušení mikrokontroléru
#include LCD.h                //LCD display
```

Definiční soubor registrů a přerušení

Makro vytvořené knihovnou LCD.lib (položka **Include**)

```
//---------------------------------------------------
// ****** Makra ******

#define LCD_PORT1 PORTA
#define LCD_PORT2 PORTB
```

Makro vytvořené knihovnou LCD.lib (položka **Makra**)

```
//---------------------------------------------------
//Externi preruseni od INT0
ISR(INT0_vect)
{

// Zde napiste funkci

}
```

Funkce přerušení, která se vykonává při jakékoliv změně na pinu vnějšího přerušení INT0

```
//---------------------------------------------------
//Preruseni pri odeslani dat seriovou linkou
ISR(USART_TXC_vect)
{

// Zde napiste funkci

}
```

Funkce přerušení, která se vykonává při odeslání bajtu sériovou linkou

```
//---------------------------------------------------
// Přerušení při shodě s kom. reg. OCR1A
ISR(TIMER1_COMPA_vect)
{

// Zde napiste funkci

}
```

Funkce přerušení, která se vykonává při shodě registru čítače s komparačním registrem OCR1A

```
//---------------------------------------------------
int main( void )
{
```

Hlavička hlavní funkce main

```
// ******Nastaveni portu******
// DDR: 0 - vstup, 1 - výstup
// PORT: 0 - "L"/Hi-Z , 1 - "H"/Pull-up
DDRA = 0b11111111;
PORTA = 0b00000000;
DDRB = 0b00000000;
PORTB = 0b11111111;
DDRC = 0b00000000;
PORTC = 0b00000000;
DDRD = 0b00000000;
PORTD = 0b00000000;
```

Inicializace portů
PORTA je výstupní
PORTB je vstupní s pull-up rezistory
zbylé porty vstupní s vysokou impedancí

```
// ********Nastaveni externiho preruseni********
// INT0 : On, Přerušení při jakékoliv změně na INT0.
// INT1 : Off, Přerušení při úrovni "L" na INT1.
// INT2 : Off, Přerušení při úrovni "L" na INT2.
GICR = 0b01000000;
MCUCR = 0b00000001;
```

Zapnutí externího přerušení 0

```
// ********Nastaveni seriove linky********
// USART: Příjem On, Vysílání Off
// 8 bitů, Parita Žádná, StopBity Jeden
// Přenosová rychlost 19,2kBaut
// Přerušení při odeslání bajtu: On
// Přerušení při příjmu bajtu: Off
// Přerušení při Data Register Empty: Off

UCSRB = 0b01010000;
UBRRH = 0b10000110;
UBRRL = 25;           //Bautrate 19,2kBaut
```

Zapnutí sériové linky

```
// ********Nastaveni čítačů a časovačů********

// *******Čítač časovač1*******
// Vstupní signál: CK/8 (f=1MHz, T=1us)
// Mód časovače: CTC (Při zhodě s kom. reg T/T vynulován)
// Přerušení při shodě s kom. reg. OCR1A
// Výstup OC1A odpojen
// Výstup OC1B odpojen

TCCR1B = 0b00001010;
TCCR1A = 0b00000000;
TIMSK = 0b00010000;
TCNT1 = 0;
ICR1 = 0;
OCR1A = 127;
OCR1B = 0;

// ******Inicializace******
lcd_init();             //inicializace LCD

sei();          // Zapnutí všech přerušení

while(1)
 {
                // Zde bude hlavni program
 }

return (1);
}
```

Zapnutí čítače 1 a přednastavení komparačního registru OCR1A na 127

Makro vytvořené knihovnou LCD.lib (položka **Inicializace**)

Nekonečná smyčka programu

Ukončení hlavní funkce main

Zdrojový kód 8: *Ukázka kompletního programu vytvořeného pomocí aplikace AVR Wizard*

8 Závěr

V této práci bylo provedeno základní seznámení s mikrokontroléry AVR od firmy Atmel. Nejprve byla stručně popsána architektura těchto procesorů a následně byly vysvětleny základy programování jazyka asembler a jazyka C (norma GCC). Hlubší popis nebyl prováděn z důvodu rozsahu jazyka, který by překročil rozsah této práce.

Dále bylo provedeno seznámení s vývojovým prostředím AVR Studio, které firma Atmel volně distribuuje. Pro toto prostředí byly v této práci vytvářeny všechny programy. Následovně bylo provedeno seznámení s dalšími překladači, které se pro programování mikrokontrolérů AVR používají.

Nakonec jako výsledek této práce byl představen program AVR Wizard, který slouží k usnadnění programování tím, že za programátora vytvoří část programu. Software se vyvíjel pomocí aplikace C++ Builderu 6. Načítání informací probíhá s *.xml souborů, které využívá program AVR Studio ke své činnosti. Informace, které nejsou uloženy v *.xml souborech byly dodány do interních souborů typu *.ini.

Během vývoje aplikace bylo experimentálně zjištěno, že některé *.xml soubory nejsou zcela precizně vytvořeny, jednotlivé soubory se mezi sebou liší ve vnitřní struktuře a proto nemohou být v nabídkách AVR Wizardu všechny položky zobrazeny správně. Například u některých obvodů se nezobrazují alternativní funkce pinů. Při výběru pouzdra neexistují některé typy, které se na trhu běžně prodávají a vyskytují. Řešením by bylo tyto soubory projít a případné chyby opravit, ale vzhledem k velkému počtu typů MCU AVR a vzhledem k tomu, že program AVR Wizard funguje i s těmito chybnými soubory, i když uživatel nemůže využívat všech možností programu, se během této práce tomuto problému nevěnovala pozornost a především bylo záměrem vytvoření a odladění aplikace AVR Wizard.

Další problém, který při vývoji aplikace byl zjištěn je, že některé obvody se od jiných liší částečně v architektuře. Například všechny mikrokontroléry AVR, které jsou vybaveny AD převodníky mají AD převodník 10bitový s postupnou aproximací, ale pouze obvod ATMega406 je vybaven 12bitovým Sigma-Delta převodníkem, který má jiné nastavení než převodník s postupnou aproximací. Protože se jedná jen o jeden obvod takto vybavený, nebyly do aplikace AVR Wizard zahrnuta nastavení AD převodníku pro tento obvod. Obdobné je to u některých časovačů v řadě ATTiny atd… Popis všech aktuálně podporovaných obvodů je uveden v programu AVR Wizard v nápovědě.

I přes některé výše popsané nedostatky vyvinutá aplikace AVR Wizard pracuje s velkou většinou mikrokontrolérů AVR bez problémů a především byla splněna podmínka, aby program uměl generovat kód hlavně pro obvod ATMega16.

Ve finální verzi programu lze nastavovat porty, čítače/časovače, watchdog, předděličku hodin MCU, AD převodník, komparátor, vnější přerušení, sériovou, SPI a TWI linku a uživatel může vytvářet makra pomocí položky knihovny.

9 Seznam literatury

[1] Váňa, V. *Mikrokontroléry ATMEL AVR – popis procesoru a instrukční soubor* Nakladatelství: BEN, 2003

[2] *Slabikář XML – úvod do problematiky.* [online] Učebnice jazyka XML. [cit. 20.12.2006]. Dostupný z WWW: <http://interval.cz/clanky/slabikar-xml-uvod-do-problematiky/ >.

[3] *AVR Freaks.* [online] Stránky AVR komunity. [cit. 29.4.2008]. Dostupný z WWW: < http://www.avrfreaks.net/ >.

[4] *Embedded Development Tools from IAR Systems.* [online] Oficiální stránka k programu IAR. [cit. 12.12.2006]. Dostupný z WWW: < http://www.iar.com >.

[5] *HP InfoTech.* [online] Domovská stránka CodeVisionAVR. [cit. 20.12.2006]. Dostupný z WWW: < http://www.hpinfotech.ro/ >.

[6] *ImageCraft Embeddeed systems C Developent Tools.* [online] Oficiální stránka Image craft. [cit. 20.12.2006]. Dostupný z WWW: < http://www.imagecraft.com >.

[7] *Atmel Corporation.* [online] Oficiální stránka firmy Atmel. [cit. 25.5.2008]. Dostupný z WWW: < http://www.atmel.com/ >.

[8] *NonGNU.* [online] AVR Libc Home Page. Dostupný z WWW: < http://www.nongnu.org/avr-libc/ >

[9] *Unis.* [online] Processor Expert OnLine. [cit. 20.12.2006] Dostupný z WWW: < http://www.processorexpert.com/ >

[10] *WinAVR.* [online] Stránky softwaru WinAVR. [cit. 20.12.2006] Dostupný z WWW: < http://winavr.sourceforge.net/ >

[11] *Atman Electronic.* [online] Stránky programu AtmanAvr C. [cit. 20.12.2006] Dostupný z WWW: <http://www.atmanecl.com >

[12] *MiTeC Homepage.* [online] Stránky programu XML Viewer. [cit. 20.12.2007] Dostupný z WWW: <http://www.mitec.cz/ >

[13] Herout, P. *Učebnice jazyka C,* Nakladatelství: KOPP, 2001

[14] Váňa, V. *ATMEL AVR – Programování v jazyce C – Popis a práce ve vývojovém prostředí CodeVisionAVR C,* Nakladatelství: BEN, 2003

[15] *SynEdit.* [online] Stránky komponenty pro C++ Builder - SynEdit. [cit. 20.4.2008] Dostupný z WWW: <http://synedit.sourceforge.net>

10 Seznam zkratek

AD – Analogově digitální (převodník)
ADC – Analog to Digital Converter
Č/Č – Časovač/Čítač
EEPROM – Electrically Erasable PROM
IDE – Integrated Development Environment
ISR – Interrupt Service Rutine
LCD – Liquid Crystal Display
PCINT – Pin Change Interrupt
PWM – Pulse Wide Modulation
RTC – Real Time Clock
SPI – Serial Peripheral Interface
TWI – Two-wire Serial Interface
UART – Universal Asynchronous Receiver Transmitter
USART – Universal Synchronous and Asynchronous serial Receiver and Transmitter
XML – eXtensible Markup Language