Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta – Katedra fyziky

Řízení mikrokontroléry ATMEL

Bakalářská práce

Vedoucí práce: ing. Michal Šerý

Autor: Vojtěch Bušek

Tato bakalářská práce popisuje vlastnosti mikrokontrolérů Atmel AVR a srovnává je se starší vývojovou řadou jednočipových počítačů téhož výrobce. Dále popisuje způsoby programování těchto mikrokontroléru a to jazykem symbolických adres a především programovacím jazykem Bascom. Bakalářská práce také poukazuje a vysvětluje různá úskalí, které programování v tomto jazyce často provází. Zvláštní pozornost je věnována čítačům, časovačů a pulzně šířkové modulaci která s jejich použitím souvisí. Součástí práce je i ukázková aplikace.

This work describes microcontrollers Atmel AVR and compares those with an older developmental line of one-chip computers by the same maker. It also describes ways of programming these microcontrollers by the Assembly language and foremost by the Bascom language. The work presented here also emphasizes various problems often encountered when using this programming language. Particular attention is given to counters, timers and the pulse-width modulation, which is connected to their use. Part of this project is also application illustrating the programme and the microcontroller.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

6.5.2009

Podpis

Touto formou bych rád poděkoval vedoucímu této bakalářské práce p. Ing. Michalovi Šerému za cenné rady a připomínky při zpracování mé práce.

Obsah

1 Jednočipové počítače - vlastnosti a využití

V roce 1977 byl společností Intel uveden na trh první osmibitový jednočipový počítač s označením Intel 8048 který se díky své univerzálnosti velmi rozšířil do nejrůznějších systémů a spotřební elektroniky. Tyto obvody v sobě integrovaly různé periferie a byly používány v televizorech, elektronických hračkách a podobných zařízeních, při jejichž výrobě hrála podstatnou roli cena a případně i provozní spolehlivost. Jeden z následovníků tohoto jednočipového počítače byl hojně používán v klávesnicích pro osobní počítače, kde zprostředkovával přenos dat.

Později, konkrétně v roce 1980, byl opět společností Intel uveden další z obvodů, který se svou strukturou řadil také mezi jednočipové počítače a tím byl Intel 8051. Jeho určení bylo především pro řízení samostatných systému menších rozsahů a v této oblasti zůstal dodnes populární a využívaný. Mikroprocesory odvozené od architektury Intel 8051 jsou s různými vylepšeními a inovacemi vyráběné dodnes.

Postupem času se stále zvětšují požadavky na rychlost výpočetních jednotek i uvnitř specializovaných systémů a dochází k vytlačování zmíněných jednočipových počítačů novějšími a výkonnějšími typy. Mezi ty se řadí i relativně nový typ mikropočítače Atmel AVR. Tento se poprvé objevil na trhu v roce 1997 a od té doby si získává stále větší oblibu a je nasazován do nerůznějších řídích a komunikačních systémů.

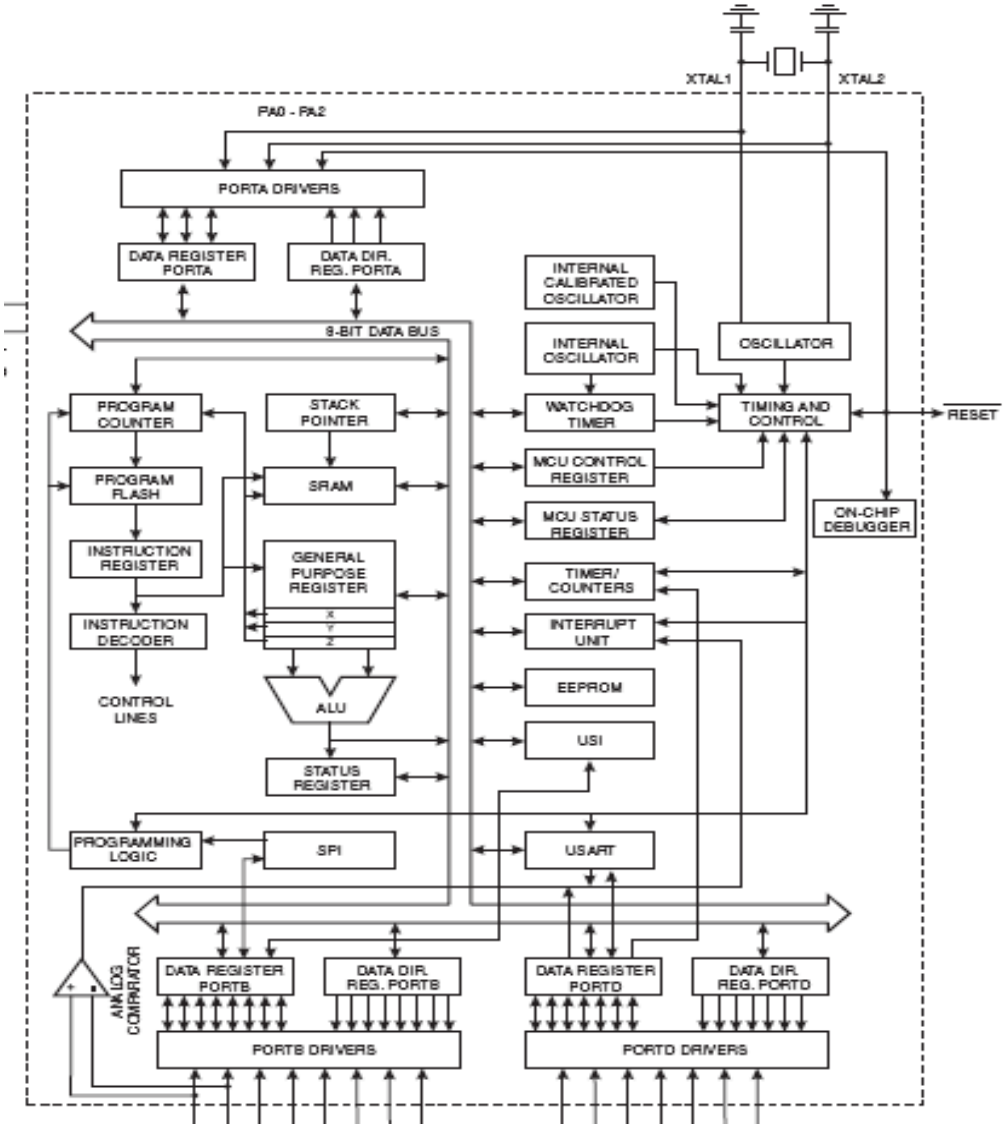
2 Struktura jednočipového počítače ATMEL

Mikrokontroléry ATMEL jsou 8bitové jednočipové počítače Harvardské architektury. Jako takové mají oddělenou paměť pro program a pro zpracovávaná data. Tyto paměti mohou být vzájemně zcela odlišné a v praxi se tak často setkáváme s použitím pamětí typu ROM (Read-Only Memory) případně EPROM (Electrically Programmable Read-Only Memory) pro uložení programu a paměti typu RWM (Read-Write Memory) pro data. V současné době (rok 2009) se setkáváme především s dvěma typy mikropočítačů, které se vzájemně liší svojí vnitřní strukturou a to s mikropočítači Atmel AVR a AT89xx. V prvním případě se jedná o mikroprocesory s redukovanou instrukční sadou – RISC. V případě druhém se jedná o typ procesoru CISC s kompletní instrukční sadou, jelikož jeho instrukční sada obsahuje i složitější instrukce které trvají déle než jeden strojový cyklus. Tyto mikroprocesory spadají do rodiny procesorů Intel 8051 vyvinuté touto společností již v roce 1980.

2.1 Struktura procesorů Atmel

Na počátku devadesátých let 20. století byl norskými návrháři a programátory uskutečněn návrh mikrokontroléru tak, aby jeho vnitřní struktura vyhovovala překladačům vyšších programovacích jazyků a především široce rozšířenému programovacímu jazyku C. Výsledkem jejich práce bylo optimalizované jádro mikroprocesorů s harvardskou architekturou s hlavními charakteristikami

procesorů s redukovanou instrukční sadou (RISC – Reduced Instruction Set Controllers) viz. obr. 2.1. Na rozdíl od svých předchůdců založených na Intel 8051 je zde zvolena šestnáctibitová šířka instrukčního slova, což umožňuje zrychlit načítání instrukcí. Instrukce tak vystačí s jedním slovem



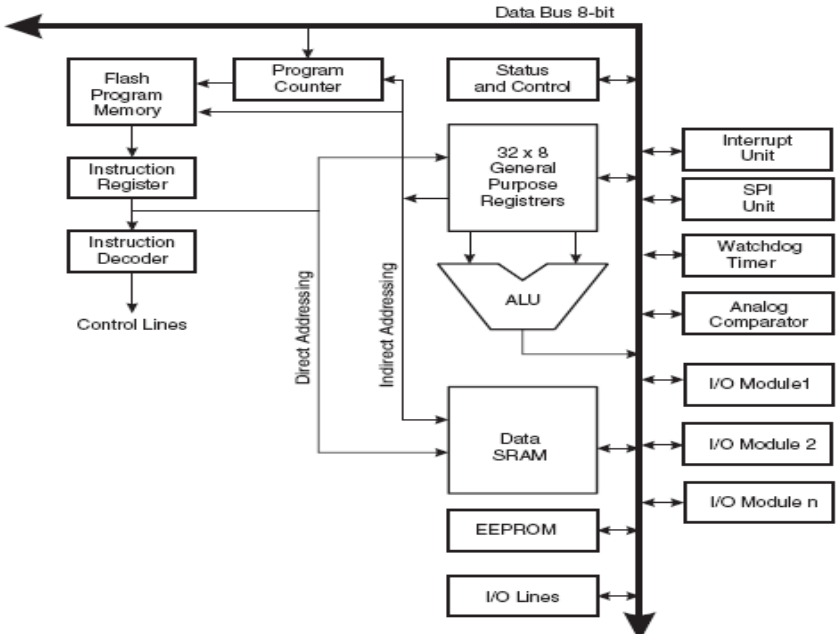
Obr. 2.1, vnitřní struktura - ATtiny 2323/V

a mikroprocesor je díky tomu schopen načíst a dekódovat tuto instrukci během jednoho hodinového cyklu. Během následujícího cyklu je tato instrukce vykonána. Použitá šestnáctibitová šířka instrukčního slova dále umožňuje i jednoduché překrývání těchto dvou fází. Operace typu registr – registr je tak vykonána během jednoho hodinového cyklu. Přeneseně poté můžeme hovořit o jednotce IPS – instrukce za sekundu. Tato má větší vypovídající schopnost o výkonu mikroprocesoru, než prosté srovnání jejich taktovacích frekvencí. Procesory Atmel AVR tak dosahují propustnosti téměř 1 MIPS / 1 MHz což je 12x více než procesory rodiny 8051 při stejné taktovací frekvenci. Veškeré výkonné a podpůrné obvody jsou uvnitř mikroprocesoru propojeny 8bitovou datovou sběrnicí, která zajišťuje přenos dat mezi těmito logickými celky. Mezi nimi mají zásadní význam především řadič, aritmeticko-logická jednotka, registr instrukcí, dekodér instrukcí, pracovní registr a registr stavů. Tento

soubor jednotek můžeme též nazývat výkonným jádrem. Další rozhraní a periferie stejně jako počet vstupně výstupních portů a jim příslušných registrů se s každým typem jednočipového počítače značně liší. Vstupně-výstupní porty bývají v počtech 1 až 8, dále se setkáváme s čítači/časovači, A/D a D/A převodníky, sériovými linkami jako např. RS232, USB a dalšími jinými, externími vstupy přerušení, výstupy pulzně šířkové modulace. Výjimkou není ani rozhraní ethernet a možnost připojení externích pamětí.[1, 2]

2.1.1 Výkonné jádro jednočipového počítače

Výkonné jádro jednočipových počítačů ATMEL AVR se skládá z následujících celků: řadič procesoru, aritmeticko-logická jednotka, registr stavů, univerzální pracovní registr, a registr a dekodér instrukcí. Schéma výkonného jádra procesoru je znázorněno na obrázku 2.2.



Obr. 2.2, výkonné jádro jednočipových počítačů ATMEL AVR

Aritmeticko-logická jednotka - je část procesoru, která je odpovědná za provádění početních, logických a funkčních bitových operací, které vždy používají soustavu, pro kterou je navržen celý obvod – v tomto případě vždy dvojková. Jedná se tedy např. o součet, rozdíl, součin, podíl, logický součet, logický součin, negace, bitový posuv atd. Aritmeticko-logická jednotka nejprve načítá data ze vstupního registru, poté je řadičem řečeno jakou operaci má použít a po jejím vykonání je výsledek uložen do výstupního registru. Jednotka také často generuje kódy k určení několika možných stavů které mohou ovlivnit výsledek operace. Těmito kódy jsou například přenos čísla z/do vyššího řádu, dělní nulou, přetečení a podobně. [3]

Řadič - je sekvenčním obvodem realizovaná řídící jednotka, která řídí činnost veškerých částí mikropočítače. Toto je realizováno pomocí tak zvaných řídících signálů, které tento obvod zasílá jednotlivým dílčím komponentám. Tyto pak sdělují svůj stav zpět řadiči pomocí signálů stavových.

Existují dva základní typy řadičů. První z nich tzv. pevně zapojený (v angl. literatuře označován „Hard-wired“) používá pro řízení mikropočítače soustavu digitálních obvodů a jako takový nemůže být již po vyrobení pozměňován. Druhý typ zvaný programovaný (v angl. literatuře označován „microprogrammed“) sám dekóduje a vykonává instrukce ve smyslu vykonávání mikroprogramu. Všechny mikropočítače Atmel používají první typ – tedy řadiče realizované obvodem.[4]

Registr stavů – obsahuje informace o výsledcích nedávno provedených instrukcí. Tyto informace mohou být použity pro změnu běhu programu – například pro vykonání podmíněných operací. Registr stavů je obnovován pokaždé, když aritmeticko-logická jednotka provede jí příslušnou operaci. Ve výkonných jednotkách mikroprocesorů Atmel není registr stavů automaticky ukládán ve chvíli, kdy dojde k vykonání obsluhy přerušení a rovněž po jeho obsluze není tento registr automaticky obnoven. Toto musí být ošetřeno softwarově. Na obrázku 2.3 můžeme vidět uspořádání tohoto registru. Význam jednotlivých bitů je následující:

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr. 2.3, uspořádání registru stavů mikropočítače Atmel AVR

Bit č.7 – I: Globální přerušení: povoleno – tento bit musí být nastaven, aby byl globálně povolen vstup přerušení. Pokud není tento bit nastaven pak není povolen žádný vstup přerušení, nezávisle na nastavení dílčích vstupů. Tento bit je také mikroprocesorem vymazán pokaždé, když přerušení nastane. Tuto skutečnost je nutné brát v úvahu při programování a do části programu zodpovědné za obsluhu přerušení zavést i příkaz pro nové globální povolení vstupu přerušení. Tento bit lze nastavovat instrukcemi *RETI*, *SEI* a *CLI*.

Bit č.6 – T: Uložení kopírovaného bitu – je využíván instrukcemi *BLD* a *BST (BitLoad a BitStore)* jako zdrojový případně cílový při operaci mezi tímto bitem a pracovním registrem.

*Bit č.*5 - *H:* Indikátor pomocného přenosu – indikuje pomocný přenos u některých aritmetických operací. Tento je užitečný při aritmetických operacích s BCD kódem.

*Bit č.*4 – *S:* Znaménkový bit – je vždy výhradním součtem (XOR) následujících dvou bitů. S = N xor V.

*Bit č.*3 – *V:* Indikátor přetečení dvojkového doplňku – je používán v aritmetice záporných čísel v binární soustavě kódovaných dvojkových doplňkem.

*Bit č.*2 – *N:* Indikátor negace/záporu – indikuje negativní nebo záporný výsledek logické nebo aritmetické operace.

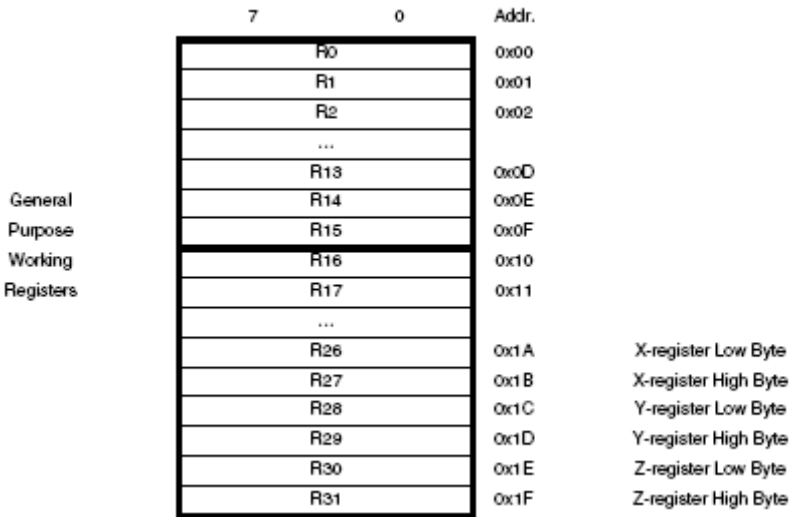
Bit č.1 – Z: Indikátor nuly – informuje o nulovém výsledku aritmetické nebo logické operace.

*Bit č.*0 – *C:* Indikátor přenosu – informuje o přenosu do vyššího řádu při výsledku aritmetické nebo logické operace.

Univerzální pracovní registr – je malá, ale velmi rychlá paměť nacházející se uvnitř mikroprocesoru. Slouží k dočasnému uchování operandů, se kterými se provádějí aritmetické

nebo logické operace. Obecně pracovní registr procesoru představuje vrchol v hierarchii pamětí a poskytuje nejrychlejší přístup ke zpracovávaným datům. Registry lze dělit podle účelu použití a práce s nimi na *uživatelsky přístupné, datové, adresové, podmínkové, univerzální, registry plovoucí čárky, registry konstant atd.* Univerzální pracovní registr je jakousi kombinací datového a adresového registru a je optimalizován pro výkonné jádro mikropočítače Atmel AVR. Aby bylo dosaženo určitého výkonu a flexibility, je univerzálním registrem podporováno následující uspořádání:

- jeden 8-bitový výstup pro operandy a jeden 8-bitový vstup pro výsledek
- dva 8-bitové výstupy pro operandy a jeden 8-bitový vstup pro výsledek
- dva 8-bitové výstupy pro operandy a jeden 16-bitový vstup pro výsledek
- dva 16-bitové výstupy pro operandy a dva 16-bitové vstupy pro výsledek



Obr. 2.4, uspořádání univerzálního registru 32x8, Atmel AVR

Většina procesorových instrukcí má přímý přístup do všech registrů a většina z nich je navíc vykonána v jednom hodinovém cyklu. Registry R26 až R31 jsou šestnáctibitové, provedené jako 2x8bitů a uplatňují se při nepřímém adresování datového prostoru, případně jsou jim přiřazeny zvláštní funkce jako automatická inkrementace, dekrementace či posouvání bitů.

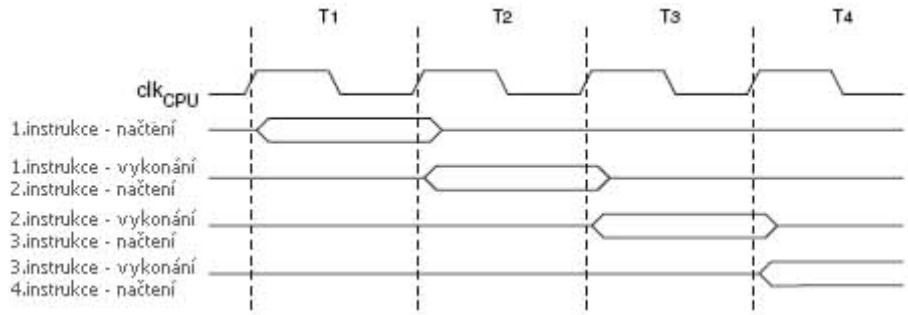
Čítač instrukcí – nebo také instrukční ukazatel je speciální registr v procesoru který je zodpovědný za adresování instrukcí strojového kódu v operační paměti. Zde uložený program je procesorem postupně zpracováván po instrukcích, které jsou uloženy za sebou na postupně vyšších adresách paměti. Strojový kód je vykonáván sekvenčně. Po načtení strojové instrukce z určité adresy paměti (na kterou ukazuje čítač instrukcí) se hodnota registru zvýší o velikost načítané instrukce. Hodnota adresy poté ukazuje na následující instrukci. Jiné chovaní tohoto registru nastává v okamžiku, kdy je načtena instrukce ke skoku na určitou adresu. Poté je hodnota adresy přímo vepsána do registru.

Dekodér a registr instrukcí – instrukce je načtena do tzv. registru instrukcí, kde je po dobu jejího vykonání také uložena. Dekodér tuto instrukci rozdělí na jednotlivé kousky kódu a ty posílá pomocí řadiče jednotlivým komponentám, které poté provedou požadovanou operaci. Instrukce je tedy slovo,

které obsahuje výkonný (binární) kód. U výkonného jádra mikropočítačů Atmel AVR je dekodér instrukcí, stejně jako u většiny procesorů s redukovanou instrukční sadou, velmi jednoduché zařízení. To z důvodu, že instrukce těchto procesorů mají pevnou délku slova - v tomto případě 16bitů a pevné pozice jednotlivých částí. Dekódovat instrukci tak můžeme pouze oddělením jednotlivých částí strojového slova a předáním těchto částí dalším obvodům procesoru.[21]

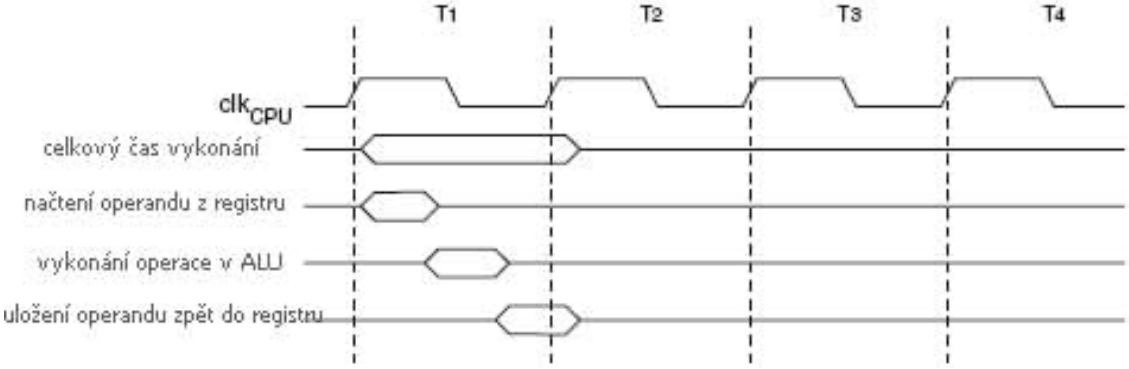
2.1.2 Časování vykonání instrukce

Instrukce prováděné jádrem procesoru jsou řízené externím hodinovým signálem. Tento signál je u mikroprocesorů obvykle přiváděn z externího nebo interního oscilátoru. Nejčastěji je tento oscilátor realizován krystalem o dané frekvenci, nebo RC členem. Časování vykonání instrukce je různé u procesorů rodiny RISC a CISC a tedy i u mikropočítačů Atmel AVR a klonů 8051. Na obrázku 2.5 je znázorněno načítání a vykonání instrukcí jádrem procesoru Atmel AVR ve vztahu k hodinovému



Obr. 2.5, paralelní načítání a vykonání instrukce, Atmel AVR

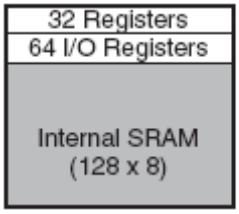
signálu. Názorně je zde vidět způsob proudového zpracování instrukcí (pipelining), kterým je dosaženo výpočetního výkonu téměř 1 MIPS na 1 MHz. Jeden strojový cyklus (načtení a vykonání instrukce) proběhne za jeden hodinový cyklus. U mikropočítačů Atmel AT89xx z rodiny procesorů 8051 je potřeba dvanácti hodinových cyklů pro načtení a vykonání instrukce. Koncept vnitřního časování a vykonání instrukce v aritmeticko-logické jednotce je znázorněn na obrázku 2.6.



Obr. 2.6, operace v ALU během jednoho hodinového cyklu

2.2 Struktura pamětí mikropočítače

Jednočipové mikropočítače Atmel obsahují dva hlavní paměťové prostory, paměť pro data a paměť pro uložení programu. Všechny tři paměťové prostory v mikrokontroléru jsou lineární.



Obr. 2.7, rozdělení paměti SRAM

S ohledem na vlastní poslání jsou zde použity celkem tři typy pamětí a to programovatelné paměti typu flash pro uložení vykonávaného programu. Tato může být minimálně 10 000x mazána a zapisována a její velikost se s každým typem mikrokontroléru mění. Z pravidla se setkáváme s kapacitou od 1kB do 256kB. Organizace této paměti je podřízena AVR architektuře a šestnáctibitovým instrukčním slovům, tedy například 1 K x 16 bit (v tomto konkrétním případě je výsledná kapacita paměti 2 kB). V této paměti může být na libovolném místě uložena i tabulka programových konstant. Dalším typem použité paměti je paměť SRAM která zastává funkci datové paměti pro ukládání zpracovávaných dat. Její část je dále členěna na vstupně/výstupní a pracovní registry. Rozložení paměti SRAM je znázorněno na obrázku 2.7. Přístup do této paměti je vykonán během dvou strojových cyklů. Posledním použitým typem je paměť typu EEPROM. Její životnost je asi 100 000 čtecích/zapisovacích cyklů a bývá přítomna ve velikostech 128 až 256 B. Vzhledem k omezenému počtu možných zápisů není vhodné ji používat pro ukládání zpracovávaných dat. Naopak často se využívá pro zápis programových konstant. Paměť EEPROM je čtena a zapisována po bytech. Při čtení z této paměti je procesorová jednotka zastavena po dobu 4 hodinových cyklů, než vykoná další instrukci. V případě zápisu do EEPROM je doba zastavení 2 hodinové cykly.

2.3 Rozhraní mikropočítačů

Mikropočítače obecně bývají vybavovány nejrůznějšími rozhraními pro komunikaci se svým okolím. Ať už jde o komunikaci v pravém slova smyslu nebo o příjem a zasílání dat okolním obvodům. Vstupně-výstupní porty bývají 8bitové kde jednotlivé piny jednočipového počítače reprezentují jednotlivé bity konkrétního portu. Nastavení portu jako vstupního/výstupního zajišťuje tzv. směrový datový registr. Krom této základní aplikace jsou k jednotlivým portům připojeny další obvody rozšiřující funkce daného obvodu. Jedná se například o A/D převodníky, vstupy externího přerušení, komunikační porty sběrnic, pulzně šířkovou modulaci atd. Pro nastavení těchto funkcí slouží speciální registr.

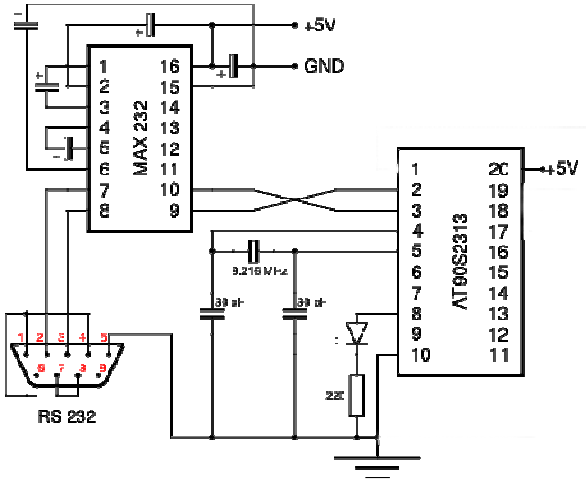
Zde je vhodné si uvědomit jak moc náročné může být programování aplikací pro mikrořadiče při použití nízkoúrovňového programovacího jazyka – jazyka symbolických adres (někdy též zvaný assembler). Programování aplikací v tomto jazyku vyžaduje naprostou detailní znalost konkrétního typu mikroprocesoru a vzhledem ke své obsáhlosti je značně časově náročné. Zde se nabízí možnost použití vyšších programovacích jazyků pro vývoj programů. Rád bych právě na tomto místě při inicializaci různých rozhraní mikropočíítačů demonstroval jednoduchost a velmi snadnou použitelnost programovacího jazyku Bascom. Ten je svou syntaxí velmi podobný programovacímu jazyku Basic.

2.3.1 Rozhraní RS-232

Jedná se o poměrně běžně integrované rozhraní sériové komunikace, které je normováno již od 60.let 20.století.

Z elektrického hlediska se u rozhraní RS-232C používají tzv. nesymetrické vazební obvody podle doporučení CCITT V.28. Doporučení RS-232C tedy definuje funkci obvodů rozhraní a doporučení V.28 způsob jejich elektrické realizace. Přiřazení minimálních úrovní signálu jeho logickým hodnotám je následující: napětí -15 V až -3 V odpovídá úrovni logické 1, napětí +3 V až +15 V odpovídá logické úrovni 0. Zakázaná oblast logických stavů odpovídá napětí v rozmezí -3 V až +3 V. Napájecí napětí obvodů je maximálně ±25 V, v současnosti se již používají častěji napětí běžněji dostupná v mikropočítačích. [5]

Úrovním logického signálu dle normy odpovídají napětí ± 15 V resp. ±12 V případně i menší. Při potřebě komunikace mezi mikrokontrolérem a jiným zařízením, používajícím odlišné napěťové úrovně, tedy musí dojít ke změně úrovní signálu. Mikrokontoléry totiž používají pro rozlišení dvou stavů i v případě sběrnice RS-232 napěťové úrovně 0 V a 5 V. K tomuto účelu se používá především v současnosti velmi rozšířený obvod MAX232. Jeho výhodou je nutná přítomnost napájecího napětí pouze 5 V. Obvod v sobě integruje dva přijímače a dva vysílače sběrnice RS232 s konverzí na napěťové úrovně logických obvodů TTL. Náznak zapojení s mikropočítačem Atmel AVR je naznačen na obrázku 2.8. Vývody mikrokontroléru pro sériovou komunikaci jsou obvykle značeny *RXD a TXD.*

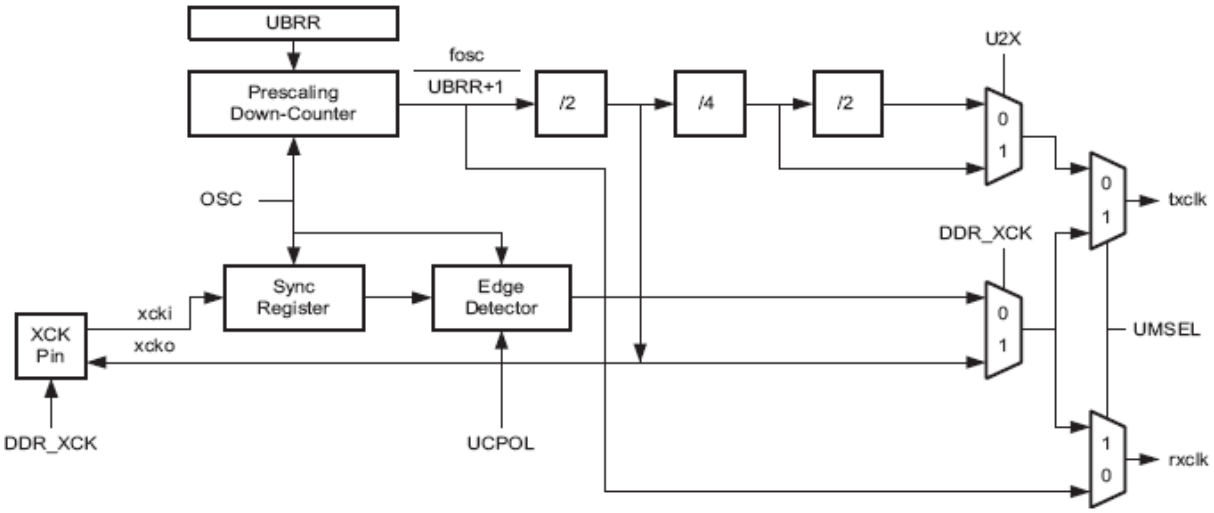


Obr. 2.8, schéma zapojení obvodu MAX232 a Atmel AT90S2313

V programovacím jazyce Bascom je implementována většina příkazů pro práci se sériovým rozhraním podobně jako v programovacím jazyce Basic. Ke komunikaci slouží především příkaz ***Print*** a ***Input***. V tom případě se předpokládá použití výstupů mikropočítače přímo určených pro sériovou komunikaci. Následující příklad programu odešle řetězec na sběrnici RS232 prostřednictvím výstupů definovaných výrobcem a specifikovaných v datovém souboru daného typu mikropočítače:

```
$regfile = "m48def.dat"        /specifikuje použitý typ mikrořadiče

$crystal = 4000000             /specifikuje frekvenci použitého krystalu

$baud = 19200                  /nastaví požadovanou rychlost komunikace

Config Com1 = Dummy , Synchrone = 0 , Parity = None , Stopbits = 1 ,
Databits = 8 , Clockpol = 0   /nastavuje parametry rozhraní

Print "text"                   /odešle řetězec „text“ do sériového rozhraní

End
```

V reálné aplikaci musíme velikou pozornost věnovat potřebné komunikační rychlosti a výběru vhodné frekvence oscilátoru. Vnitřní obvody pro časování komunikačního kanálu jsou pomocí funkčních registrů nastavitelné pro různé módy provozu sériového rozhraní a to synchronního a asynchronního o různých přenosových rychlostech. Z toho důvodu je také vlastní časování nutné a to zcela nezávislé na ostatních částech mikrokontroléru. Na obrázku 2.9 je znázorněno blokové schéma generátoru hodinových impulsů pro sériovou sběrnici.



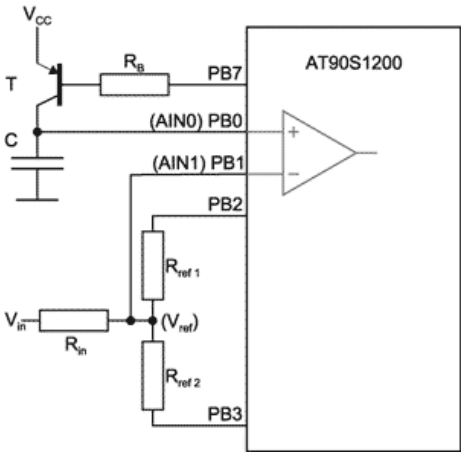
Obr. 2.9, blokové schéma generátoru hodin mikrokontroléru Atmel ATtiny2323

Přenosová rychlost je generována z frekvence externího oscilátoru pomocí předděličky (Prescaling Down-Counter) a registru přenosové rychlosti (UBRR). Pro dosažení optimálních přenosových vlastností je nutné použít oscilátor o frekvenci, která je celým násobkem požadované přenosové rychlosti sériového kanálu. Předdělička spolu s registrem UBRR je schopna podle zadaných údajů generovat časovací signál pro libovolné rychlosti s použitím prakticky libovolného, dostatečně rychlého oscilátoru. V tomto případě ovšem dochází k určitým chybám při přenosu zpráv mezi komunikujícími stranami. Tato chybovost by neměla přesáhnout 5%, jinak již příjemce není schopen zprávu přečíst.

Tabulky pro přesná nastavení registru UBRR ve vztahu k přenosové rychlosti, k použitému oscilátoru a typu přenosu jsou obvykle k dispozici v dokumentaci konkrétního typu mikropočítače.

2.3.2 Analogový komparátor

Některé typy mikropočítačů v sobě integrují analogový komparátor, který velmi prakticky rozšiřuje možnosti využití jednočipového počítače. V aplikacích mikropočítačů se obvykle používá pro měření některých fyzikálních veličin jako například odporu, kapacity, napětí apod. Přeneseně lze poté měřit i veličiny jiné – například teplotu, převedeme-li jí pomocí odpovídajícího čidla (termistoru) na odpor. Vstupní napětí komparátorů nesmí být záporné ani větší než napájecí napětí. Výstup komparátoru má hodnotu odpovídající rozdílu mezi neinvertujícím a invertujícím vstupem – tedy je-li napětí na neinvertujícím vstupu větší než na vstupu invertujícím, je výstup komparátoru ve stavu logické 1.



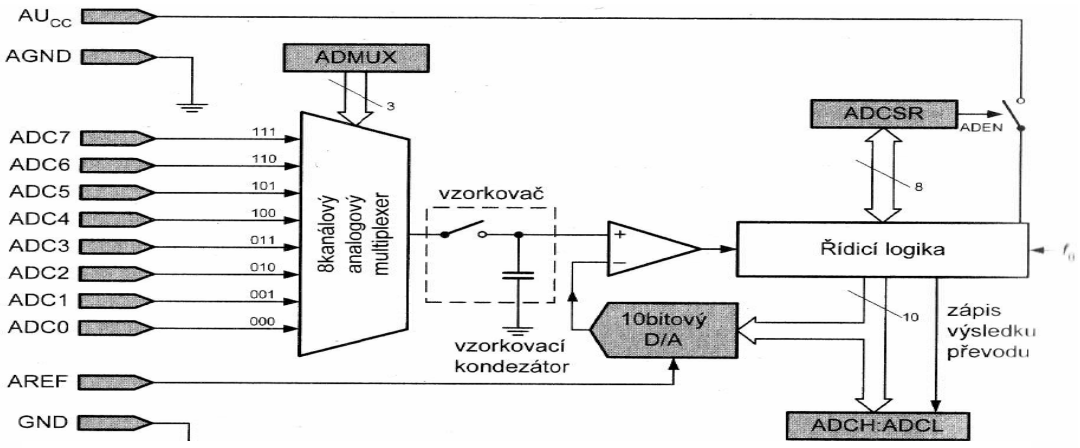
Obr. 2.10, schématické zapojení integrovaného komparátoru

Příkaz programovacího jazyka Bascom pro práci s analogovým komparátorem dovoluje vypnout a zapnout jeho funkci a umožňuje nastavit, která událost komparátoru vyvolá přerušení. Syntaxe příkazu je následující:

```
CONFIG ACI = ON|OFF, COMPARE = ON|OFF, TRIGGER=TOGGLE|RISING|FALLING
```

2.3.3 Analogově-digitální převodník

Některé jednočipové počítače v sobě integrují analogově-digitální převodníky. Ty jsou svou konstrukcí aproximační. Jejich vstupy jsou spojené se vstupně-výstupním portem mikropočítače a aby sloužily jako A/D převodník, musí být nastaven příslušný registr. A/D převodníky bývají velmi často 8mi kanálové se vstupem do multiplexeru (což vychází z konstrukce aproximačních převodníků) a mají různé bitové rozlišení. Na obrázku 2.11 můžeme vidět blokové schéma 8mi kanálového A/D



Obr. 2.11. blokové schéma 10bitového A/D převodníku – Atmel Atmega128

převodníku integrovaného do mikropočítače Atmel ATmega128.

Převodníky mají pracovní frekvenci od 50 do 200kHz a z toho důvodu je přiváděný hodinový signál dělen předděličkou tak, aby bylo dosaženo požadované frekvence. Předdělička dělí frekvenci externího oscilátoru hodnotami 2, 4, 8, 16, 32, 64 a 128. Vstupní napětí může nabývat hodnot od 0 V do napájecího napětí – z pravidla 5 V. Referenční napětí je voleno interní 2,56 V nebo externí referenční – připojené ke vstupu AREF. Integrované A/D převodníky mají určité zkreslení a integrální nelinearitu. Pro přiblížení – u mikropočítače Atmel ATmega128 je zkreslení ± 2 nejméně významné bity a integrální nelinearita je 0,5 nejméně významného bitu. Převodníky po dokončení převodu mohou generovat přerušení. Programovací jazyk Bascom obsahuje několik příkazů pro pohodlnou práci s A/D převodníky. Po specifikaci konkrétního typu mikropočítače je nutné pomocí příkazu Config Adc definovat nastavení převodníku. V následujícím příkladu je nastaven A/D převodník výše zmíněného jednočipového počítače, následně je z jeho vstupu odečtena převedená hodnota, která je v dalším kroku vytištěna.

```
Config Adc = Single , Prescaler = Auto

Dim W As Word , kanal As Byte

kanal = 0

Do

W = Getadc(kanal)

Print "Kanal " ; kanal ; " hodnota " ; W

Incr kanal

If kanal > 7 Then kanal = 0

Loop

End
```

2.3.4 Externí vstupy přerušení

Mikropočítače Atmel z pravidla obsahují i několik externích vstupů, díky kterým mohou okolní periferie žádat o obsluhu výkonnou jednotkou. V případě přítomnosti signálu na některém ze vstupů dojde k přerušení běhu aktuálně zpracovávaného programu a k výkonu části programu definované pro obsluhu přerušení. V mikropočítačích Atmel je obvykle přítomno více vstupů přerušení (ať už interní, nebo externí). Při práci s jednočipovým počítačem musíme zohlednit skutečnost, že může být obslouženo vždy jen jedno přerušení. Když nastane přerušení, dojde v registru stavů k nastavení bitu č.7 (Globální přerušení: povoleno) na hodnotu 0, a tím k zablokování možných jiných přerušení. V obslužném programu může poté opět dojít k nastavení výše zmíněného bitu a tím k povolení tzv. „vnořených přerušení“. Při návratu z obslužného programu zpět k vykonávání programu původního je

instrukcí *RETI* opět nastaven bit č.7 v registru stavů. V programovacím jazyku Bascom je obsluha přerušení velmi jednoduchá. V následujícím příkladu je povoleno přerušení, konkrétně *Int0 a příkazem On Interrupt* je specifikováno jaká část programu bude vykonána.

```
Enable Interrupts

Enable Int0

On Int0 obsluha_preruseni Nosave

Do

Zde je vlastní program, který je neustále vykonáván

Loop

End

obsluha_preruseni:

program obsluhy přerušení
```

2.3.5 Čítače a časovače

Při práci s jednočipovými počítači je často nutné jeho práci rozvrhnout do určitých časových úseků a případně časové úseky měřit. Jednočipové počítače obsahují obvody, které tyto funkce realizují samostatně. Tím nedochází ke zbytečnému vytížení procesoru a ten je tak schopen vykonávat zcela nezávisle vlastní program. Po uplynutí měřeného nebo generovaného časového úseku je generováno přerušení a definovanou částí programu je toto přerušení obslouženo. Čítače a časovače jsou z pravidla realizovány jedním obvodem s možností nastavení módu. Obvod tak tedy může vystupovat jako časovač nebo jako čítač.

Čítač je obvod, který počítá impulzy vnějšího signálu (měří jeho kmitočet).

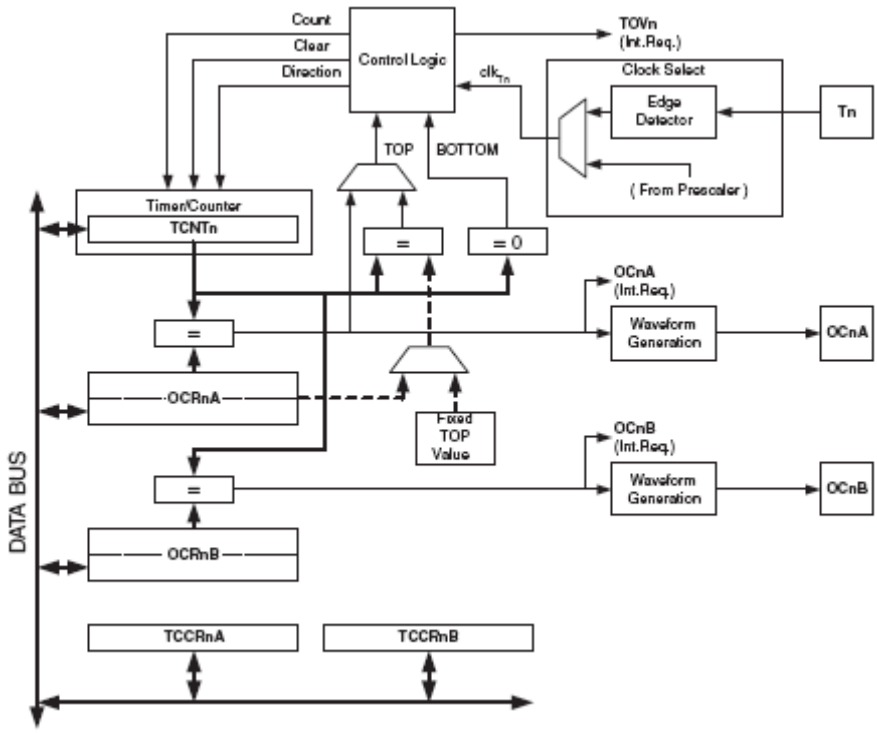
Časovač čítá pevný kmitočet, který je obvykle odvozen od hodinového signálu mikrořadiče. Napočítáním určitého počtu impulzů se zajistí odměření časového úseku. Obvykle jsou dvě funkce spojeny a pak se jedná o čítač/časovač. [6].

Čítače a časovače jsou závislé na hodinovém signálu mikropočítače. Také jsou omezeny svou bitovou šířkou a tedy i maximální velikostí měřeného intervalu nebo maximálním počtem naměřených impulsů. Standardně jsou implementovány 8mi a 16ti bitové čítače/časovače. Práce s časovači velmi závisí i na typu mikropočítače, ve kterém je čítač nebo časovač integrován. To souvisí s rychlostí vykonávání instrukcí. Při použití čítače dochází k jeho inkrementaci vždy při detekované změně logické úrovně během dvou strojových cyklů.

Jednočipové počítače rodiny Intel 8051 inkrementují časovač v každém strojovém cyklu. Doba, za kterou je inkrementován stav časovače, je $t = \left(\frac{f_{(osc)}}{12} \right)^{-1}$. Pro odměření intervalu 1 ms při frekvenci oscilátoru 12 MHz musí hodnota časovače dosáhnout 1000. V případě čítače impulsů dochází k čítání během dvou strojových cyklů a tedy mezní frekvence přiváděného signálu je dána vztahem $f_{(\max)} = \frac{f_{(osc)}}{(12 \cdot 2)}$. Rychlejší změny přiváděného signálu nebude čítač schopen zaznamenat. V případě připojení 12 MHz oscilátoru nebude možné sledovat signál s frekvencí větší než 0,5 MHz.

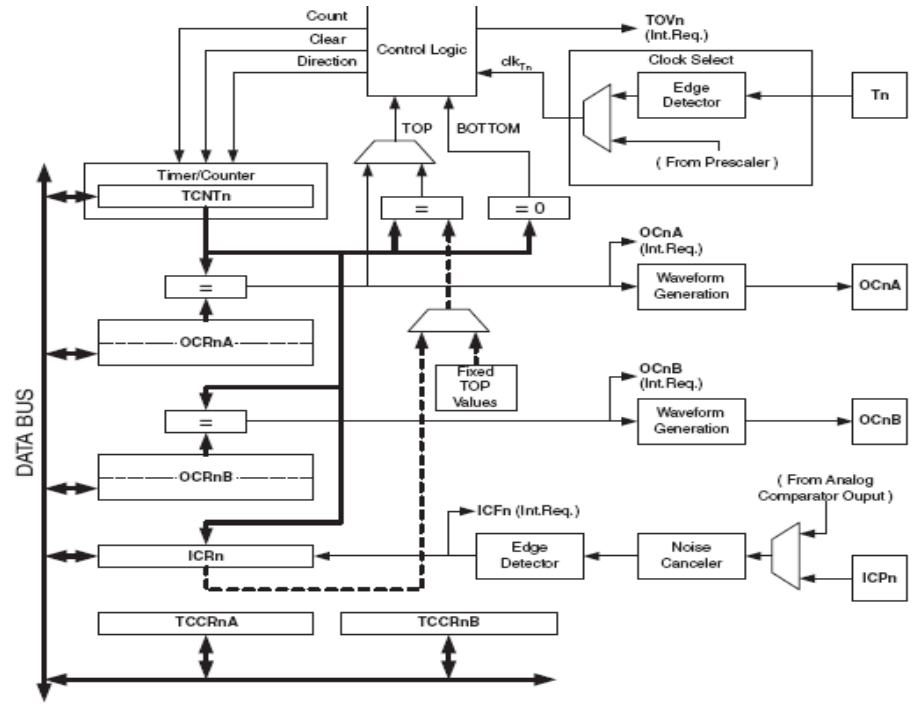
Mírně odlišná je situace u procesorů typu AVR. Zde se, z důvodu architektury RISC, mohou časovače inkrementovat s každou periodou hodinového impulsu s možností zapojení předděličky. Ta je sama tvořena 10ti bitovým čítačem a je tak schopna hodinový signál dělit hodnotami 8, 64, 256 a 1024. Doba za kterou je inkrementován stav časovače je v případě AVR procesorů dána vztahem $\frac{1}{t} = \frac{f_{(osc)}}{preddeleni}$. U čítačů je možné dále volit metodu čítání impulsů a to při sestupné nebo vzestupné hraně signálu.

Čítače mikropočítačů Atmel AVR obsahují funkci dosažení stavu. Při této funkci je stav čítače srovnáván s registrem *OCRnX* a při shodě je nastavením příslušného bitu (*OCFnX*) možné vyvolat přerušení. [7] Blokové schéma 8mi bitového čítače/časovače je uvedeno na obrázku číslo 2.12., schéma 16ti bitového čítače/časovače pak na obrázku číslo 2.13. Poslední zmíněný v sobě dále integruje registr *ICR*, který slouží pro zachytávání stavu čítače/časovače externím povelem přivedeným na pin *ICPx.* Zachytávací jednotka je vybavena digitálním filtrováním vstupního signálu, pro minimalizování možnosti zachycení šumu a jeho vyhodnocení jako platného vstupního signálu. Může také generovat přerušení.



Obr. 2.12, blokové schéma 8mi bitového čítače/časovače Atmel ATTiny2313

Nastavení čítačů a časovačů v programovacím jazyce Bascom je maximálně zjednodušeno, ale i tak vyžaduje znalost jejich konstrukce, především pokud je potřeba, aby čítač nebo časovač



Obr. 2.13, blokové schéma 16ti bitového čítače/časovače Atmel ATTiny2313

generoval přerušení. V první řadě je nutné nastavit obvod pro práci v požadovaném režimu. To lze provést za pomocí příkazu *Config Timer*.

```
Config Timer0 = Counter , Edge = Rising/Falling, Prescale = 1/8/64/256/1024

Config Timer0 = Timer , Prescale = 1/8/64/256/1024
```

Rozšířené možnosti 16ti bitového čítače/časovaače lze obdobně využívat opět příkazem *Config Timer.*

```
Config Timer1 = Counter , Edge = Falling , Capture Edge = Falling , Noise
Cancel = 1 , Prescale = 1
```

Při nutnosti čítání konkrétní hodnoty resp. odměření konkrétního časového úseku je možné do registru *TCNTx* přímo zapsat konkrétní hodnotu. Uvažujeme-li frekvenci hodinového signálu například 8 MHz a časovači bude nastaveno předdělení hodnotou 256, zvýší se hodnota v registru za 1 s na 31250. V případě 16ti bitového čítače/časovače však k přetečení a k vyvolání přerušení dojde až po napočítání 65536 pulsů. Pokud bude čítač přednastaven na hodnotu kterou vypočteme jako rozdíl nejvyššího možného stavu čítače a stavu, kterého čítač dosáhne za požadovanou jednotku času tj. $65356 - 31250 = 34286$, bude docházet, v tomto případě, každou sekundu k přetečení časovače, a v případě povolení příslušného přerušení i k jeho obsluze. Zápis do registru je proveden přiřazením hodnoty do speciální proměnné. [8] Obdobně lze kdykoliv obsah čítače načíst a přiřadit do proměnné.

```
TimerX/CounterX/TcntX = hodnota

hodnota = TimerX/CounterX/TcntX
```

Stejnou syntaxi používá programovací jazyk Bascom i pro nastavení srovnávacího registru *OCRnX*. Stejně tak pro jeho čtení.

```
Compare1a = hodnota

Compare0b = hodnota

hodnota = Compare1b
```

Při dosažení porovnávané hodnoty čítačem/časovačem mohou být vykonány určité akce. Těmi jsou:

- nastavení výstupního pinu *OCnX* do stavu logické 1
- nastavení výstupního pinu *OCnX* do stavu logické 0
- překlopení stavu výstupního pinu *OCnX*
- odpojení čítače/časovače od výstupního pinu

Nastavení čítače/časovače a těchto jeho rozšířených funkcí je možné provést zápisem s uvedením výše zmíněných parametrů.

```
Config Timer1 = Counter , Edge = Falling , Compare A = Set , Compare B =
Toggle , Clear Timer = 1
```

Jak bylo uvedeno, přerušení mohou být generována každým čítačem/časovačem při přetečení nebo dosáhnou-li stavu příslušného porovnávacího registru. Přerušení také může být generováno při zachycení hodnoty na základě vnějšího impulsu na pin *ICPx*. Výčet všech přerušení generovaných čítači/časovači je uveden v tabulce 2.1. Níže uvedené příklady zápisu jsou ekvivalentní.

událost	příznak přerušení	přerušení	alt.název Bascom
přetečení Timer0	TOV0	OVF0	Timer0
přetečení Timer1	TOV1	OVF1	Timer1
shoda s registrem OCR0A	OCF0A	OC0A	Compare0A
shoda s registrem OCR0B	OCF0B	OC0B	Compare0B
shoda s registrem OCR1A	OCF1A	OC1A	Compare1A
shoda s registrem OCR1B	OCF1B	OC1B	Compare1B
zachycení stavu	ICF1	ICP1	Capture1

Tab. 2.1, seznam přerušení čítačů/časovačů

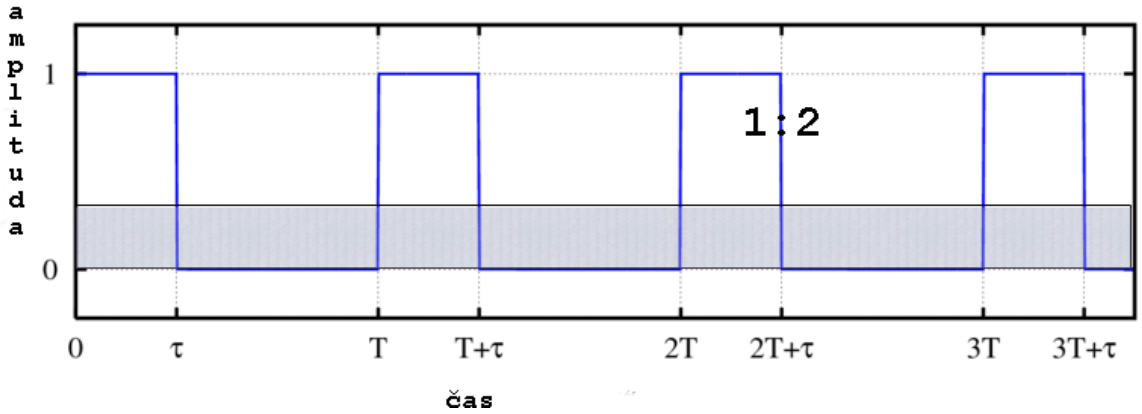
```
Enable Interrupts

Enable Timer0

On Timer0 obsluha_preruseni Nosave

Do … Loop

End
```

```
Enable Interrupts

Enable OVF0

On OVF0 obsluha_preruseni Nosave
```

```
Do … Loop

End
```

2.3.6 Pulzně šířková modulace

Porty čítačů a časovačů také mohou sloužit jako výstupy pulzně šířkové modulace. Pulsně šířková modulace, neboli PWM (Pulse Width Modulation) je diskrétní modulace pro přenos analogového signálu pomocí dvouhodnotového signálu. Jako dvouhodnotová veličina může být



Obr. 2.14, průběh výstupního signálu při PWM modulaci střídy 1:2. Šedě je znázorněna střední hodnota signálu.

použito například napětí, proud, nebo světelný tok. Signál je přenášen pomocí střídy. Vzhledem ke svým vlastnostem je pulsně šířková modulace často využívána ve výkonové elektronice pro řízení velikosti napětí nebo proudu. [9]

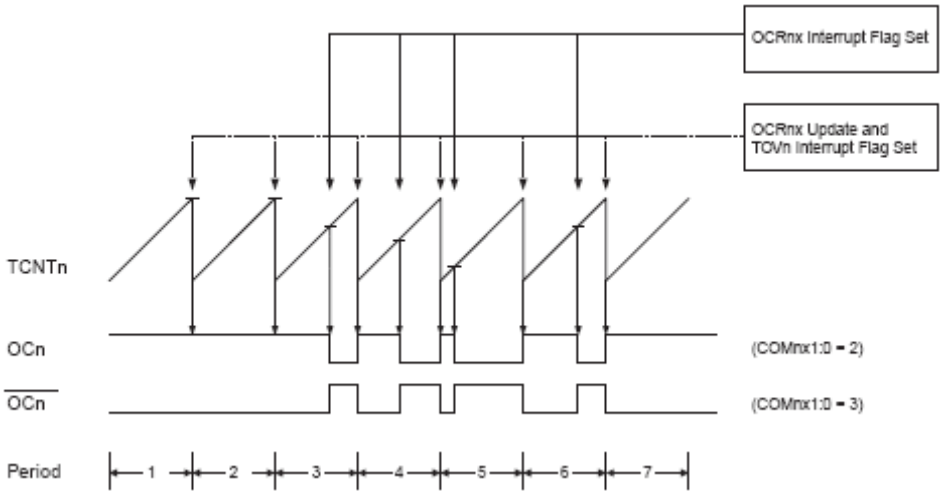
Pulzně šířková modulace využívá principů vnímání střední hodnoty časově proměnného signálu některými akčními členy obvodů jako jsou diody LED, žárovky, stejnosměrné motory, reproduktory apod. Využívá střídy, neboli poměru časů, ve kterých je signál v jednotlivých úrovních. Střída je dána poměrem doby trvání impulzu ku délce periody $D = \frac{\tau}{T}$. Při pulzně šířkové modulaci signálu tedy dochází pouze ke změně šířky impulzu a není nijak ovlivněna jeho frekvence ani amplituda jak je znázorněno na obrázku 2.14.

U jednočipových počítačů s integrovanou funkcí pulzně šířkové modulace se nachází výstupy připojené přes synchronizátor ke komparátoru stavu čítače/časovače a srovnávacího registru, čímž je možnost použití pulzně šířkové modulace značně zjednodušena a závisí pouze na nastavení a práci čítačů/časovačů. Samotný mikropočítač není vytížen řízením této modulace a může tedy obsluhovat jiné periferie. Nicméně, s použitím jednočipových počítačů lze realizovat obvody pulzně šířkové modulace i v případě, že sami nedisponují výše zmíněným specializovaným rozhraním, avšak za cenu poměrně vysokého vytížení mikropočítače.

Čítače generující signál PWM modulace vychází z konstrukce čítačů/časovačů integrovaných uvnitř mikroprocesorů. Jsou časovány hodinovým signálem vstupujícím do mikropočítače s možností

jeho předdělení a řízeny společnými registry. Chování čítače/časovače je definováno kombinací bitů generování tvarového signálu (*WGM00)* a bitů porovnávání výstupů *(COM0x1)* registru *TCCRnX*. Nastavení bitů porovnávacích výstupů na rozdíl od bitů generování tvarového signálu nijak neovlivňuje čítanou sekvenci. Bity *COM0x1* ovlivňují generování PWM modulace ve smyslu invertování nebo neinvertování této modulace. Jednočipové počítače Atmel AVR mohou generovat celkem tři druhy PWM modulace. Prvním a nejjednodušším z nich je rychlá PWM modulace.

Rychlá pulzně šířková modulace poskytuje vysokofrekvenční modulační signál a od jiných způsobů modulace se odlišuje tím, že hodnota čítače se vždy pouze zvyšuje. Čítač počítá od hodnoty 0 do své maximální možné hodnoty. Poté je vynulován a cyklus se opakuje. Během čítání je jeho stav porovnáván s obsahem v registru *OCRnX*. Při shodě je poté provedena akce se změnou stavu pinu *OCnX.* Při standardní neinvertované funkci pulzně šířkové modulace je logická úroveň zmíněného pinu nastavena na hodnotu 0, dojde-li ke shodě čítače a *OCRnX* registru. Čítač dále napočítá do své maximální hodnoty a při vynulování je nastaven pin *OCnX* na hodnotu logické 1. Generování invertovaného signálu je obdobné. Při nulovém stavu čítače je *OCnX* pin nastaven do logické 0 a při rovnosti stavů čítače a komparačního registru do logické 1. Časovací diagram pulzně šířkové modulace v rychlém módu je znázorněn na obrázku 2.15.



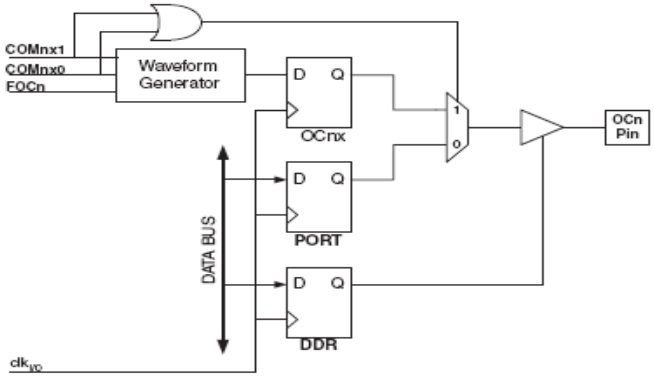
Obr. 2.15, časovací diagram pulsně šířkové modulace v rychlém PWM módu.

Při každém dosažení maximální hodnoty čítače je nastaven příznak přetečení daného čítače *TOVx* a je vyvoláno přerušení. Toho může být využito při potřebě nového nastavení hodnoty komparačního registru.

Aktuální logická hodnota výstupních pinů pulzně šířkové logické modulace je na nich viditelná pouze v případě nastavení pinů jako výstupních. Směrový registr dat totiž není nijak ovlivněn nastavením čítačů/časovačů a je proto nutné tuto podmínku splnit. Logika řízení vstupně-výstupních pinů v souvislosti s výstupy PWM modulace je zobrazena na obrázku 2.16. Frekvence modulovacího signálu může být vyjádřena jako poměr vstupního hodinového signálu a dělícího poměru předděličky násobeného 256ti, respektive maximální hodnotou čítače/časovače zvětšenou o 1.

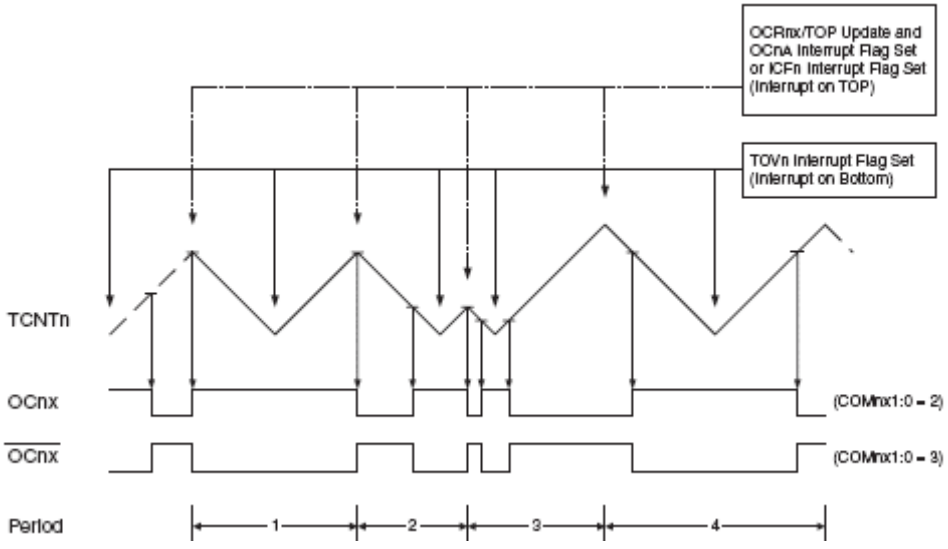
$f_{OCnXPWM} = \frac{f_{(osc)}}{N \cdot 256}$. Při použití rychlé PWM modulace mohou nastat dva zvláštní případy a to je-li hodnota *OCRnX* nastavena na hodnotu 0, nebo na hodnotu vrcholu čítače. V prvním případě bude docházet ke krátkému překlopení stavů výstupu v délce trvání jednoho hodinového cyklu, v případě druhém pak bude výstup držen stabilně ve stavu logické úrovně 0 nebo 1, což bude záviset na použití invertovaného nebo neinvertovaného režimu generování PWM.

Fázově korigovaný režim pulzně šířkové modulace oproti výše uvedenému poskytuje pulzně šířkovou modulaci s vysokým rozlišením. Čítač/časovač čítá ode dna do své maximální hodnoty. Poté dojde k přenastavení a čítač odčítá ze své maximální hodnoty až k hodnotě nula. Toto se celé opakuje. V neinvertujícím režimu je vývod *OCnX* nastaven do úrovně logické 0 při shodné hodnotě čítače a registru *OCRnX* při čítání nahoru. K nastavení do úrovně logické 1 dochází opět při schodě čítače a registru, ale při počítání dolu. V invertujícím režimu je tato činnost opačná.



Obr.2.16, řízení výstupních pinů PWM regulace

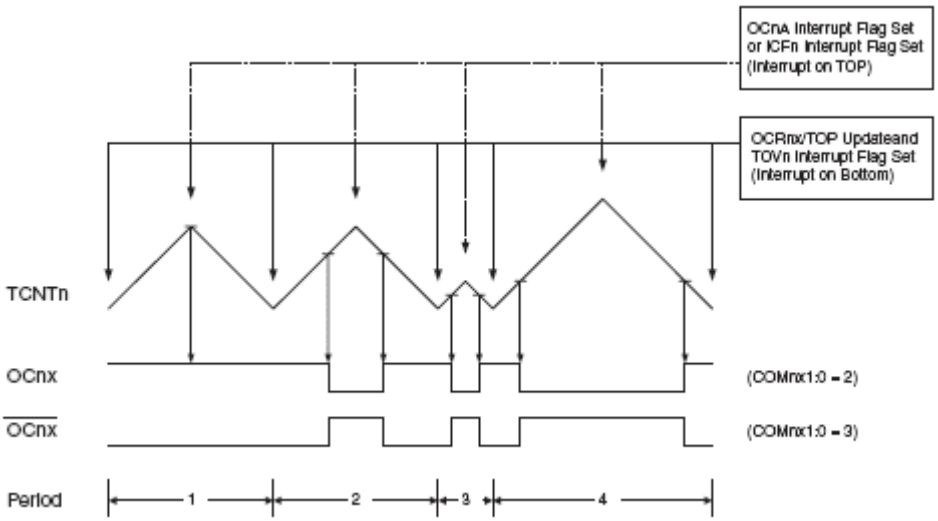
Z této funkce vyplívá i celkově nižší generovaný kmitočet PWM modulace než při rychlém režimu, ovšem v některých aplikacích se mu dává přednost. Jeho bitové rozlišení může být stanoveno pevně na 8, 9 nebo 10 bitů, případně k určení vrcholu čítačů může být použit registr *ICR1* nebo *OCR1A* (samozřejmě v případě, že pro výstup PWM modulace je použit čítač/časovač1). Časový diagram pro fázově korigovaný režim pulzně šířkové modulace je zobrazen na obrázku 2.17.



Obr. 2.17, časový diagram fázově korigovaného PWM režimu, Atmel AVR

Přerušení příslušného čítače/časovače je vyvoláno vždy, když je dosaženo dna čítače. Pro definici vrcholu čítače může být, jak již bylo uvedeno, použit registr *ICR1* nebo *OCR1A*. Při dosažení vrcholu čítače je také nastaven příznak přerušení příslušný těmto registrům, což může být využito k jejich aktualizaci. Při změně této hodnoty dochází k zajímavému jevu, který je dobře patrný na obrázku 2.17, kdy při změně hodnoty vrcholu čítače dochází i ke změně délky periody. To je způsobeno tím, že perioda pulzně šířkové modulace začíná a končí při dosažení maximální hodnoty čítače. To znamená, že délka sestupné fáze je určena předchozí hodnotou vrcholu čítače a délka vzestupné fáze je určena hodnotou novou. Tento efekt je často na škodu a proto se doporučuje místo tohoto režimu používat spíše režim fázově a kmitočtově korigované pulzně šířkové modulace. Frekvence výstupního signálu šířkové modulace je dána vztahem $f_{OCnXPWM} = \frac{f_{(osc)}}{2 \cdot N \cdot vrchol}$, kde hodnota N představuje nastavení předděličky a může nabývat hodnot 1, 8, 64, 256 a 1024. Mezní hodnoty registru *OCRnX* představují zvláštní případy generování signálu pulzně šířkové modulace. Tyto případy nastávají v okamžiku, kdy hodnota registru je rovna nule, nebo vrcholu čítače/časovače. V tom případě je výstup *OCnX* stabilně ve stavu logické 1 nebo 0 (neinvertující režim) nebo opračně při invertujícím režimu. Dalším zvláštním případem je nastavení tzv. *„toggle režimu“* při kterém dochází k periodické změně stavu výstupního pinu. V takovém případě je generován signál střídy 1:1.

Fázově a kmitočtově korigovaný režim generování pulzně šířkové modulace je velmi podobný předchozímu způsobu. Je zde použita dvoufázová realizace a tedy čítač/časovač čítá od hodnoty 0 do hodnoty vrcholu a poté zpět do hodnoty 0. V neinvertujícím režimu je výstup pulzně šířkové modulace uveden do stavu logické 0 při shodě čítače a hodnoty registru *OCR1X* při čítání nahoru, a do stavu logické 1 je uveden opět při shodě při počítání směrem k hodnotě 0. V invertujícím režimu je tomu naopak. Hodnota vrcholu je dána obsahem registru *ICR1* nebo *OCR1A*. Jak lze vidět na obrázku 2.18 dochází k aktualizaci *OCR1X* registru při dosažení dna čítače/časovače. Výstupní signál je tedy symetrický a náběžné a sestupné fáze jsou stejně dlouhé.



Obr. 2.18, časový diagram fázově a kmitočtově korigovaného PWM režimu,

Použití registru *ICR1* je vhodné pro definování pevné hodnoty vrcholu čítače, přičemž registr *OCR1A* zůstává volný k dispozici pro generování výstupu pulzně šířkové modulace na příslušném výstupu.

Modulační signál lze opět volit invertující a neinvertující. Pro výstupní kmitočet signálu platí vztah $f_{OCnXPWM} = \frac{f_{(osc)}}{2 \cdot N \cdot vrchol}$, kde hodnota N představuje nastavení předděličky a může nabývat hodnot 1, 8, 64, 256 a 1024. Mezní hodnoty registru *OCR1X* představují zvláštní případy generování signálu pulzně šířkové modulace. Tyto případy nastávají v okamžiku, kdy hodnota registru je rovna nule, nebo vrcholu čítače/časovače. V tom případě je výstup *OC1X* stabilně ve stavu logické 1 nebo 0 (neinvertující režim) nebo opačně při invertujícím režimu. Dalším zvláštním případem je nastavení tzv. *„toggle režimu“,* při kterém dochází k periodické změně stavu výstupního pinu. V takovém případě je generován signál střídy 1:1.[10]

Jednočipové počítače řady Atmel AT89xx přímo nedisponují obvody, které by umožňovaly generování PWM signálu, a proto je v případě těchto mikropočítačů nutné tento nedostatek obejít a pulzně šířkový generátor vytvořit například s použitím integrovaných čítačů/časovačů generujícími přerušení a v obsluze tohoto přerušení programově řešit přepínání určitého výstupu mezi logickými stavy 0 a 1. Časový interval je poté ovlivněn i celkovou dobou, za kterou je vykonána obsluha přerušení a vytvoření přesného generátoru pulzně šířkové modulace s pomocí těchto mikropočítačů klade nároky na vykonávaný program.

Programování výstupů čítačů/časovačů jako výstupů rychlé PWM modulace při použití jazyka Bascom je již poněkud náročnější. Programovací jazyk Bascom není přímo vybaven příkazem, který by explicitně nastavil konkrétní režim generování PWM a proto je nutné tento nedostatek obejít pomocí přímého zápisu hodnoty do řídícího registru příslušného časovače. K tomu je dobré znát

strukturu těchto registrů a význam jednotlivých bitů. Struktura těchto registrů může být různá u různých typů jednočipových počítačů Atmel AVR.

Na tomto místě je potřeba zmínit, že informace o generování pulzně šířkové modulace příkazem *Config TimerX = pwm* jazyka Bascom jsou v dostupné literatuře značně nepřehledné a leckdy zavádějící. Použitím výše zmíněného příkazu dojde ke generování fázově korigované pulzně šířkové modulace. Pomocí tohoto příkazu je rovněž nastaveno rozlišení, hodnota předdělení a zda generovaný signál bude invertovaný nebo neinvertovaný. Syntaxe příkazu je následující:

```
Config Timer1 = Pwm , Pwm = 8 , Compare A Pwm = Clear Up , Compare B Pwm =
Clear Down , Prescale = 1
```

Hodnota *Pwm* může nabývat 8, 9 a 10, čímž je dána i maximální hodnota vrcholu čítače (v bitech). Parametry *Compare A Pwm* a *Compare B Pwm* určují invertující nebo neinvertující režim pwm generátoru. Hodnota *Clear Up* určí neinvertující režim, hodnota *Clear Down* pak určí režim invertující. Hodnota *Prescale* určuje nastavení předděličky čítače/časovače a může nabývat hodnot 1, 8, 64, 256 nebo 1024. [11]

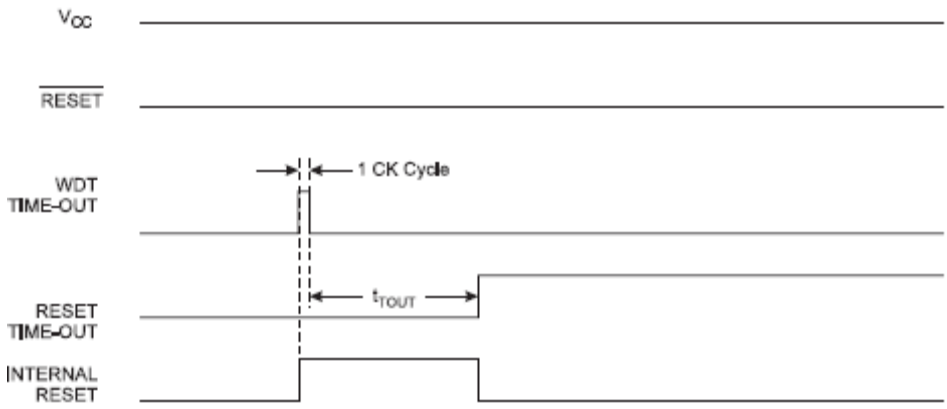
Dále je nutné zadat hodnoty *OCRnX* registrů. Jejich nastavení má přímý vliv na dobu setrvání výstupu *OCnX* v jednom z logických stavů. Lze použít příkazy, které odpovídají číslu příslušného časovače (0, 1, 2) a jemu příslušnému výstupu (a, b). Syntaxe příkazu je následující:

```
Pwm0a = 100

Pwm1b = 200
```

Absence příkazu, který by umožňoval zvolit konkrétní režim generování pulzně šířkové modulace je nevýhodou programovacího jazyka Bascom.

2.4 Obvody sledování běhu mikrokontroléru

Jednočipové počítače Atmel mohou být vybaveny integrovaným nebo externím obvodem, který sleduje běh vykonávaného programu a je v případě nutnosti schopný mikropočítač resetovat, a vyvést ho tak z nežádoucího stavu. Ten může vzniknout především vlivem nedokonalého návrhu vykonávaného programu nebo jinou situací, na kterou není mikropočítač připraven. Tento hlídací obvod (v angl. jazyce Watchdog) je v případě mikropočítačů Atmel AVR realizován samostatným čítačem připojeným k vnitřnímu oscilátoru, který kmitá na frekvenci 128 kHz. Tento kmitočet může být



Obr. 2.17, časový průběh generování resetu mikropočítače Atmel AVR

různý u různých typů jednočipových počítačů. Princip činnosti je následující: čítač hlídacího obvodu počítá impulsy z vnitřního oscilátoru. Během vykonávání programu je potřeba v daných intervalech provést vynulování tohoto čítače, jinak je při jeho přetečení vyvolán reset jednočipového počítače. Tím je zajištěno, že při nežádoucí práci procesoru (například při uvíznutí v

programové smyčce), při které z pravidla nedochází k nulování čítače, dojde v poměrně krátkém čase k vygenerování impulzu, který trvá jeden strojový cyklus. Tento impuls odstartuje časový interval t_{TOUT}, při kterém dojde k samotnému resetu viz obrázek 2.17.

Interval za který čítač hlídacího obvodu přeteče je možné měnit zapojením předděličky k čítači pulzů a v případě frekvence interního oscilátoru 128 kHz je tento časový interval 16 ms až 8 s.

Hlídací obvod je možné deaktivovat i při samotném vykonávání programu procesorem, ale je nutné provést speciální sekvenci instrukcí, která minimalizuje možnost náhodného deaktivování tohoto obvodu. [12]

Hlídací obvod v mikropočítači Atmel AVR má i další funkce, které souvisí s jeho úspornými režimy provozu. Jejich popisu je věnována část 2.5 bakalářské práce.

Jednočipové počítače Atmel řady AT89xx v sobě výše popsaný obvod neintegrují a je proto nutné ho nahradit formou externího přídavného obvodu. Externí obvody ve většině případů obsahují časovací člen, který může být realizován digitálně – například čítačem, nebo analogově – například nabíjením kondenzátoru na určitou hodnotu napětí. Příkladem je obvod MAX 690A, který, kromě zmíněné funkce, disponuje navíc funkcemi sledování napájecího napětí a zajištění správného náběhu systému po připojení napájecího napětí. To je nutné pro zabránění náhodných stavů na výstupech mikropočítače po zapnutí, což je vyvoláno rušením a neustálením přivedeného napájecího napětí. [13]

2.5 Režimy nečinnosti

Režimy nečinnosti, někdy též zvané režimy spánku dovolují výrazně snížit energetickou spotřebu mikropočítače v případě, že není požadován jeho maximální výkon. Existuje několik možných režimů energetické úspory z nichž některé jsou k dispozici pouze u specifických druhů jednočipových

počítačů. Pro přechod do některého z těchto režimů je nutné nastavení příslušného bitu obecného řídícího registru. Samotný přechod je poté realizován instrukcí *SLEEP*.

Pro vyvedení mikropočítače z režimu nečinnosti je nutné vyvolat přerušení příslušné určité periferii. Toto je u každého z druhů režimu nečinnosti různé. Je-li příslušné přerušení vyvoláno, dojde po přechodu do normálního režimu činnosti k jeho obsloužení, a dále mikropočítač pokračuje instrukcí následující za instrukcí *SLEEP*, která ho dříve uvedla do režimu nečinnosti. Dalším z možných způsobů „procitnutí“ je přivedení signálu na *RESET* vstup jednočipového počítače. Při této situaci je vynulován programový čítač a program je vykonáván od začátku jako při startu.

Prvním z režimů je režim nečinnosti (idle mode). Vyznačuje se tím, že při přechodu procesoru do tohoto režimu je jádro zastaveno a nevykonává instrukce. Periferie mikropočítače jako čítače/časovače, sériový kanál, A/D převodník, analogový komparátor, obvod sledování běhu a systém přerušení zůstávají aktivní a dovolují vyvedení z tohoto stavu prostřednictvím jim příslušných přerušení.

Režim sníženého napájecího napětí (power-down mode) je charakteristický tím, že v tomto režimu je odstaven vnější oscilátor, je zastaveno jádro procesoru a nevykonává žádné instrukce. Dále jsou zastaveny veškeré periferie a aktivní zůstává pouze obvod sledování stavu mikrokontroléru a systém přerušení. K vyvedení z tohoto stavu může dojít resetem – tj. přivedením signálu na vstup *RST* mikropočítače, případně resetem vyvolaným hlídacím obvodem, nebo pomocí úrovňově citlivých vstupů *INT0* a *INT1*. Po aktivaci tohoto režimu je možné výrazně snížit napájecí napětí, čímž výrazně klesá odběr.

Vstupy vnějšího přerušení *INT0* a *INT1* musí být pro vyvedení ze stavu nečinnosti konfigurovány jako úrovňově citlivé. To z důvodu, že při běžné práci jsou tyto vstupy citlivé na hranu signálu. Tato citlivost je při normálním běhu mikropočítače zajišťována vzorkováním zmíněných vstupů synchronně s kmitočtem externího oscilátoru a ten je právě v režimu sníženého napájecího napětí a v režimu snížené spotřeby odpojen.

Režim snížené spotřeby (power-save mode) je k dispozici pouze u některých typů mikrokontroléru a to u těch, které integrují obvod čítače/časovače 2. Tento režim je identický s předchozím s výjimkou, že při přechodu do tohoto stavu zůstane aktivní čítač/časovač 2, pokud pracuje asynchronně. Vyvedení z tohoto stavu je možné stejnými způsoby jako ze stavu sníženého napájecího napětí. Dále je možné pro tento účel použít přerušení čítače/časovače 2 při přetečení jeho obsahu, nebo lze použít příslušnou událost při dosažení stavu tohoto časovače s pomocí registru *OR2X (Output Compare)*. [14]

Jednočipové počítače řady Atmel AT89xx oproti tomu disponují pouze dvěmi způsoby snížení vlastní spotřeby. První z nich je identický s režimem nečinnosti mikropočítačů Atmel AVR. Druhý ze způsobů je identický s režimem snížení napájecího napětí mikropočítačů Atmel AVR s tím rozdílem, že v případě mikropočítačů Atmel AT89xx již není možné jej vyvést do normálního pracovního režimu s pomocí přerušení a je nutné vykonat reset obvodu, čímž dojde k vynulování programového čítače a k rozběhu programu od adresy 000h a ke ztrátě dat uložených v registrech.[15]

Pro uvedení do jednoho z výše uvedených stavů lze použít příkazy *IDLE, POWERDOWN* a *POWERSAVE* programovacího jazyka Bascom.

3 Programování jednočipových počítačů

Samotný mikropočítač by nebyl použitelný bez přítomnosti vykonávaného programu. Podle příkazů programu jsou vykonávány instrukce, na jejichž základě jsou měněny stavy jednočipového počítače. Ten prostřednictvím svých vstupně-výstupních portů řídí celou výslednou aplikaci, obsluhuje své interní i externí periferie, a případně pomocí vhodných prostředků komunikuje s okolím. Při vývoji programu pro jednočipový počítač lze použít několik základních způsobů vytváření programu. Zde přichází v úvahu především nízkoúrovňový jazyk symbolických adres nebo programovací jazyky vyšší. Z těch pak především jazyk C a Bascom.

3.1 Jazyk symbolických adres – assembler

Jazyk symbolických adres nebo také jazyk symbolických instrukcí je nízkoúrovňový programovací jazyk, který je tvořen symbolickou reprezentací jednotlivých strojových instrukcí a konstant potřebných pro vytvoření strojového kódu programu pro daný procesor. Symbolickou reprezentaci tvoří zpravidla výrobce procesoru a je založena na mnemotechnických zkratkách, které vyjadřují co daná instrukce dělá, označují symbolicky registr, slovní zkratku podmínky a podobně. Jazyk symbolických adres je proto závislý na konkrétním procesoru a zapsaný program je obtížně přenositelný na jinou platformu (na rozdíl od vysokoúrovňových programovacích jazyků).

Pro překlad jazyka symbolických adres do strojového kódu se používá program, který nazýváme překladač (z angl. assembler). Oba názvy jsou často nesprávně zaměňovány.[16]

Jazyk symbolických adres je tedy jazykem nejnižší úrovně. Je závislý na strojovém kódu, přičemž pro každý procesor, nebo skupinu procesorů je tento jazyk odlišný. I zde se ovšem dají nalézt společné rysy. Výsledný program téměř vždy obsahuje následující části:

- Překladové direktivy – ovlivňují způsob překladu.
- Instrukce – symbolicky zapsané instrukce, které jsou při překladu nahrazeny příslušným strojovým kódem.
- Definice obsahu paměti – inicializace obsahu paměti, nebo vyhrazení místa uvnitř paměti pro případnou proměnnou určité délky.
- Návěstí – návěstí umožňuje pojmenovat určité místo v paměti nebo části programu a využívá se pro snadnější orientaci ve výsledném programu.
- Makra – slouží pro nahrazení často používaných částí sekvencí instrukcí, čímž jsou vytvořeny tzv. pseudoinstrukce. Tím je zajištěno poměrně dobré zpřehlednění výsledného kódu.

Instrukce je z pravidla následována operandem. Ten určuje hodnotu, která je touto instrukcí zpracovávána. Operandem může být konkrétní hodnota, registr, přímá adresa, nepřímá adresa

a případně i konkrétní bit paměťové buňky. Příklad zápisu instrukce následované operandem je uveden níže.

```
MOV  R1, R5
```

Uvedený příklad přikazuje výkonné jednotce procesoru, aby z registru R5 přenesla v něm uloženou hodnotu do registru R1. Instrukce *MOV* je instrukcí přesunu hodnoty. Význam uvedeného příkladu je takový, že obsah registru R5 bude zkopírován do registru R1, příčemž nedochází k žádné manipulaci ani změně v registru R1.

Přímým adresováním rozumíme situaci, kdy je adresa součástí instrukce. U procesorů AVR lze adresovat registr registrového pole, vstupně-výstupní registr, datovou a programovou paměť. Ve všech případech je adresa uváděna při zápisu instrukce.

Nepřímým adresováním rozumíme situaci, kdy adresa není součástí instrukce. Tato adresa vstupuje do instrukce z některého ukazatelového registru (X,Y nebo Z). Tato možnost usnadňuje zápis sekvencí, které opakovaně přistupují k jedné buňce paměti nebo k bloku za sebou uložených buněk.

Pro vytvoření výkonného programu v jazyce symbolických adres je nutné znát poměrně přesně celou vnitřní architekturu použitého mikropočítače a seznámit se s jeho instrukčním souborem, tj. výčtem veškerých možných instrukcí, které mikropočítač ke své práci využívá. Ten se samozřejmě, spolu s vnitřní architekturou, mění s každým typem mikropočítače.

Jednočipové počítače Atmel AVR mají, přes širokou škálu vyráběných variant, podobné vnitřní uspořádání výkonných obvodů, a proto je podobný i jejich instrukční soubor. Instrukce se dále rozdělují podle druhu operace, kterou vykonávají.

3.1.1 Operandy instrukcí jednočipových počítačů Atmel AVR

- **Přímé adresování jednoho registru** – tento způsob adresování se používá pro výběr jednoho z registrů registrového pole. Operační kód je doplněn pětibitovým číslem registru (2^5=32, tedy jsou adresovány registry R0 až R31). Příkladem je instrukce nulující obsah registru: `CLR  R1`
- **Přímé adresování dvou registrů** – způsob tohoto adresování je využíván pro výběr dvou registrů z pole. Operační kód je doplněn pětibitovým číslem registru (2^5=32, tedy jsou adresovány registry R0 až R31). Příkladem je instrukce přesunu dat mezi registry Ra a Rb. $Ra, Rb \in (0,31)$: `MOV  Ra, Rb`
- **Přímé adresování vstupně-výstupního registru** – toto adresování se používá pro výběr jednoho vstupně/výstupního registru. Operační kód je doplněn šestibitovým číslem vstupně-výstupního registru P (2^6=64) a pětibitovým číslem registru Ra z registrového pole. Příkladem je instrukce zápisu dat z registru Ra do výstupního registru P: `OUT  P, Ra`
- **Přímé adresování dat –** je způsob adresování používaný pro přístup k datové paměti. Operační kód je opět doplněn pětibitovým číslem registru z registrového pole a šestnáctibitovou adresou buňky datové paměti. Tato instrukce zabírá dvě slova (32 bitů). Jejím příkladem je zápis dat obsažených v registru Ra na buňku datové paměti s adresou

z intervalu 0 až poslední platná adresa datové paměti (například u mikropočítače Atmel ATtiny2313 je to hodnota $00DF): STS $00AB, Ra

- **Nepřímé adresování dat –** je používáno pro přístup k datové paměti a operační kód je doplněn pětibitovým číslem registru registrového pole. Adresa je určena obsahem registru X, Y nebo Z. Následující příklad instrukce zapisuje data obsažená v registru Ra na buňku datové paměti s adresou určenou registrem Z: ST Z,Ra
- **Nepřímé adresování dat s posunutím –** je používáno pro přístup k datové paměti. Operační kód je doplněn 5ti bitovým číslem registru z registrového pole a šestibitovým posunutím. Posunutí se přičítá k obsahu registru Y nebo Z. Tímto součinem je získána adresa datové paměti. Příklad je instrukce zapíše data z registru Ra na buňku paměti s dresou Z+a: STD Z+a,Ra
- **Nepřímé adresování dat s pre-dekrementací** – používá se pro přístup k datové paměti. Operační kód je doplněn pětibitovým číslem registru Ra z registrového pole. Adresa je určena obsahem registr X, Y nebo Z, která je však před provedením instrukce snížena o 1. Tímto je vlastně adresována předchozí buňka datové paměti. Tento způsob adresování se využívá pro realizaci cyklů, pro blokové zpracování dat. Příkladem je instrukce zápisu dat obsažených v registru Ra na buňku paměti určenou obsahem registru Z zmenšeným o 1: ST –Z,Ra
- **Nepřímé adresování dat s post-inkrementací** – tento způsob je opět používán pro přístup k datové paměti. Operační kód se doplňuje 5ti bitovým číslem registru Ra z registrového pole. Adresa je určena obsahem registru X, Y nebo Z a po provedení instrukce je obsah tohoto registru zvýší o 1. Využití je opět při realizaci cyklů pro blokové zpracování dat. Příkladem je instrukce zápisu dat do buňky datové paměti s adresou Z: ST Z+,Ra
- **Adresování konstant uložených v paměti programu –** toto adresování je používáno pro přístup k programové paměti. Adresa je určena obsahem registru Z. Je-li nejnižší bit registru Z vynulován, načítá se dolní byte. Je-li nejnižší bit registru Z nastaven, nahrává se horní byte. Programová paměť se adresuje slovem (dvěmi byty). Existuje jediná instrukce, která používá adresování konstant uložených v paměti programu – nahrává obsah paměťové buňky Z v paměti programu do registru R0. Tou je instrukce LPM
- **Nepřímé adresování paměti programu –** je používáno pro přístup k programové paměti pro realizaci skoků nebo volání podprogramů. Adresa je určena obsahem registru Z. Nepřímé adresování paměti programu využívají pouze dvě instrukce používané pro volání konkrétní části programu. Těmi jsou: IJMP a ICALL
- **Relativní adresování paměti programu –** tento způsob adresování se používá pro přístup k programové paměti pro realizaci skoků nebo volání podprogramů. Operační kód je doplněn dvanáctibitovou relativní adresou *k* (tyto skoky jsou v rozsahu -2048 až +2047). Adresa je určena součtem PC *(program counter)* + k. Instrukce používající nepřímé adresování jsou: RJMP k a RCALL k. Hodnota *k* představuje návěstí vzdálené v rozsahu -2048 až +2047

- **Zkrácené relativní adresování paměti programu (podmíněné skoky) –** tento způsob adresování se používá pro přístup k programové paměti pro realizaci podmíněných skoků. Operační kód je doplněn sedmibitovou relativní adresou *k* v rozsahu -64 až +63. Adresa je opět určena součtem stavu programového čítače a relativní adresy. Příkladem je instrukce skoku podmíněného nastavením bitu *C* ve stavovém registru (*SREG*): BRCS k. Hodnota *k* zde představuje návěstí vzdálené v rozsahu -64 až +64.

3.1.2 Typy skoků

Instrukce skoků umožňují změnit řízení programu přechodem na novou adresu a následným pokračováním ve vykonávání programu od této adresy. Jednočipové počítače Atmel AVR mohou vykonávat různě dlouhé skoky.

Podmíněné skoky pracují se sedmibitovou relativní adresou. Podmíněné skoky se provedou pouze v případě, že je splněna určitá podmínka (nastaven určitý bit – příznak/indikátor). Adresa je relativní vůči obsahu programového čítače. Skutečná adresa se získá jako součet hodnoty kterou obsahuje programový čítač, a hodnoty této relativní adresy. Relativní sedmibitová adresa umožňuje provádět skoky pouze v rozsahu 64 adresovaných pozic zpět nebo 63 adresovaných pozic vpřed od adresy, na kterou ukazuje programový čítač.

Dalším z prvků vlastních procesorům skupiny RISC jsou přeskoky. Jedná se o zvláštní typ podmíněného skoku. Je-li splněna podmínka, přeskočí se následující instrukce. Pokud není, instrukce se vykoná. Skok se realizuje přičtením hodnoty 2 nebo 3 k obsahu programového čítače. Hodnota 2 nebo 3 je dána možností délky následující instrukce, která může být dlouhá 1 nebo 2 slova. V případě, že se přeskok neprovádí, je programový čítač zvýšen o hodnotu 1.

Relativní skoky jsou vykonávány instrukcemi *RJMP* a *RCALL*, které pracují s dvanáctibitovou relativní adresou. Ta je relativní vůči obsahu programového čítače a skutečná adresa se získá součtem hodnoty programového čítače a této hodnoty. Relativní 12ti bitová adresa umožňuje provádět skoky v rozsahu 2048 adresovaných pozic zpět nebo 2047 adresovaných pozic vpřed od adresy, na kterou ukazuje programový čítač.

Poslední možností je nepřímý skok, který je realizován instrukcemi *IJMP* a *ICALL*. V tomto případě je adresa uložena v registru Z. Nepřímý proto, že adresa není obsažena v instrukci, ale je načtena právě z registru.

3.1.3 Přesuny dat

Instrukce přesunů dat představují největší skupinu instrukcí prováděných jednočipovými počítači. Tyto instrukce neovlivňují příznaky stavového registru až na případ, že instrukce sama zapisuje do stavového registru nebo pracuje s jeho bity. Přehled všech instrukcí procesorů Atmel AVR, které řídí přesuny dat, je uveden v tabulce 3.1.

Kód	Operandy	Popis	Operace	Slova / hodinové cykly
MOV	Ra, Rb	přesun dat mezi registry	Ra←Rb	1/1
LDI	Ra, data8	nahrání přímé hodnoty, a z rozsahu 16-31	Ra←data8	1/1
LD	Ra, X	nahrání dat nepřímo adresovaných přes X	Ra←[X]	1/2
LD	Ra, X+	dtto, post-inkremet	Ra←[X], X←X+1	1/2
LD	Ra, -X	dtto, pre-dekrement	X←X-1, Ra←[X]	1/2
LD	Ra, Y	nahrání dat nepřímo adresovaných přes Y	Ra←[Y]	1/2
LD	Ra, Y+	analogicky s instrukcí LD Ra,X+		
LD	Ra, -Y	analogicky s instrukcí LD Ra, -X		
LDD	Ra, Y+off6	nahrání dat adresovaných nepřímo přes Y a posunutí	Ra←[Y+off6]	1/2
LD	Ra, Z	nahrání dat adresovaných nepřímo přes Z	Ra←[Z]	1/2
LD	Ra, Z+	analogicky s instrukcí LD Ra,X+		
LD	Ra, -Z	analogicky s instrukcí LD Ra, -X		
LDD	Ra, Z+off6	nahrání dat adresovaných nepřímo přes Z a posunutí	Ra←[Z+off6]	1/2
LDS	Ra, addr16	nahrání přímo adresovaných dat	Ra←[addr16]	2/2
ST	X, Rb	uložení dat adresovaných nepřímo přes X	[X]←Rb	1/2
ST	X+, Rb	uložení dat adresovaných nepřímo přes X, post-inkrement	[X]←Rb, X←X+1	1/2
ST	-X, Rb	uložení dat adresovaných nepřímo přes X, pre-dekrement	X←X-1,[X]←Rb	1/2
ST	Y, Rb	uložení dat adresovaných nepřímo přes Y	[Y]←Rb	1/2
ST	Y+, Rb	analogicky s instrukcí ST X+,Rb		
ST	-Y, Rb	analogicky s instrukcí ST -X, Rb		
STD	Y+off6, Rb	uložení dat adresovaných nepřímo přes Y a posunutí	[Y+off6]←Rb	1/2
ST	Z, Rb	uložení dat adresovaných nepřímo přes Z	[Z]←Rb	1/2
ST	Z+, Rb	analogicky s instrukcí ST X+,Rb		
ST	-Z, Rb	analogicky s instrukcíí ST -X, Rb		
STD	Z+off6, Rb	analogicky s STD Y+off6, Rb		
STS	addr16, Rb	uložení přímo adresovaných dat	[addr16]←Rb	2/2
LPM		nahrání konstanty z paměti programu	R0←[Flash:Z]	1/3
IN	Ra, P	čtení z portu	Ra←P	1/1
OUT	P, Rb	zápis na port	P←Rb	1/1
PUSH	Rb	uložení do zásobníku	STACK←Rb, SP←SP-1	1/2
POP	Ra	vyzvednutí ze zásobníku	SP←SP+1, Ra←STACK	1/2

Tab. 3.1, přehled instrukcí přesunů dat mikropočítačů Atmel AVR

3.1.4 Bitové operace

Bitové operace pracují s jednotlivými bity registrů a to stavového, registrového pole nebo vstupně-výstupního registru. Dále mezi bitové operace zahrnujeme instrukce posuvů a rotací. Bitové operace patří mezi velmi jednoduché. Jejich funkcí je nastavení nebo nulování příslušného bitu, který je určen přímo instrukcí. Velká část bitových instrukcí proto nemá žádné operandy. Výjimku z tohoto pravidla tvoří instrukce posuvů, rotací a přenesení bitů mezi registry. Seznam bitových instrukcí je uveden v tabulce 3.2.

Kód	Operandy	Popis	Operace	Indikátor	Slova / hodinové cykly
SBI	P,b	nastavuje bit b v registru P	P.b←1		1/2
CBI	P,b	nuluje bit b v registru P	P.b←0		1/2
LSL	Ra	logický posuv registru Ra vlevo		Z,C,N,V	1/1
LSR	Ra	logický posuv registru Ra vpravo		Z,C,N,V	1/1
ROL	Ra	rotace registru Ra vlevo přes příznak C(spojením do 9bitového registru)		Z,C,N,V	1/1
ROR	Ra	rotace registru Ra vpravo přes příznak C(spojením do 9bitového registru)		Z,C,N,V	1/1
ASR	Ra	aritmetický posuv registru Ra vpravo		Z,C,N,V	1/1
SWAP	Ra	výmena dolní a horní čtveřiice registru Ra	low(Ra)↔high(Ra)		1/1
BSET	s	nastaví příznak s stavového registru	SREG.s←1	SREG.s	1/1
BCLR	s	nuluje příznak s stavového registru	SREG.s←0	SREG.s	1/1
BST	Rb,b	přenáší bit b registru Rb do příznaku T	T←Rb.b	T	1/1
BLD	Ra,b	přenáší příznak T do bitu b registru Ra	Rb.b←T		1/1
SEC		nastaví příznak C	C←1	C	1/1
CLC		nuluje příznak C	C←0	C	1/1
analogicky nulování a nastavování dalších příznaků instrukcemi SEN, CLN, SEZ, CLZ, SEI, CLI, SES, CLS, SEV, CLV, SET, CLT, SEH, CLH					

Tab. 3.2, přehled bitových instrukcí mikropočítačů Atmel AVR

3.1.5 Skoky a přeskoky

Instrukce skoků a přeskoků se používají pro změnu řízení běhu programu, neboli pro přechod do jiné části vykonávaného programu. Existují 2 způsoby změn tohoto běhu. První ze způsobů není podmíněný a změna běhu je provedena vždy. Druhý ze způsobů řídí běh programu při splnění určité podmínky (například nastavení indikátoru C), jinak je vykonána následující instrukce. Podmíněné skoky dále obsahují takzvané přeskoky, které při splnění určité podmínky přeskočí následující instrukci. Bližší popis těchto způsobů byl uveden v části 3.1.2 bakalářské práce. Přehled podmíněných skoků je uveden v tabulce 3.3.

Test	Log. výraz	Instrukce	Opačný test	Log. výraz	Instrukce
Ra=Rb	Z=1	BREQ	Ra≠Rb	Z=0	BRNE
Ra<Rb	C=1	BRLO/BRCS	Ra≥Rb	C=0	BRSH/BRCC
Ra≤Rb	C nebo Z=1	BRSH	Ra>Rb	C nebo Z=0	BRLO
Ra>Rb	C nebo Z=0	BRLO	Ra≤Rb	C nebo Z=1	BRSH
Ra≥Rb	C=0	BRSH/BRCC	Ra<Rb	C=1	BRLO/BRCC
Ra<Rb	(N xor V)=1	BRLT	Ra≥Rb	(N xor V)=0	BRGE
Ra≤Rb	Z or (N xor V)=1	BRGE	Ra>Rb	Z and (N xor V)=0	BRLT
Ra>Rb	Z and (N xor V)=1	BRLT	Ra≤Rb	Z or (N xor V)=0	BRGE
Ra≥Rb	(N xor V)=0	BRGE	Ra<Rb	(N xor V)=1	BRLT
C	C=1	BRCS	C'	C=0	BRCC
N	N=1	BRMI	N'	N=0	BRPL
V	V=1	BRVS	V'	V=0	BRVC
Z	Z=1	BREQ	Z'	Z=0	BRNE

Tab 3.3, přehled podmíněných skoků

Všechny výše uvedené instrukce včetně instrukce provádí relativní skoky v programu. Již zmíněné přeskoky pak vykonává celkem 5 instrukcí, z nichž instrukce *CPSE Ra,Rb* porovnává obsahy zmíněných registrů a při shodě dojde k přeskoku. Instrukce *SBRC Rb,b* a *SBRS Rb,b* testují určený bit registru Rb a při jeho nastavení přeskočí následující instrukci a analogicky instrukce *SBIC P,b* a *SBIS P,b* testují určený bit b v registru P.

3.1.6 Instrukce podpory podprogramů

Podprogramy jsou určité sekvence kódu, které jsou vykonávány opakovaně při běhu programu a jako takové není vhodné je i opakovaně zapisovat, čímž by se zvětšovala celková velikost výsledného kódu. Zápis podprogramu se nijak neliší od ostatních částí programu, jen je třeba jej zavolat. K tomu slouží instrukce *RCALL, ICALL, RET* a *RETI*. Instrukce *RCALL* a *ICALL* se chovají podobně jako instrukce skoků. Rozdíl je v uložení tzv. návratové adresy, která je potřebná k návratu programového čítače zpět na místo, odkud došlo k zavolání podprogramu. Instrukce *RET* a *RETI* slouží pro návrat z podprogramu po jeho vykonání zpět na místo kde k zavolání podprogramu došlo.

Při použití těchto návratových instrukcí dochází k častým chybám při použití jazyka symbolických adres. Pokud totiž do zásobníku, ve kterém je uložena návratová adresa, na začátku podprogramu nahrajeme hodnotu další a nedojde k její vyjmutí před použitím návratové instrukce, nebude použita správná návratová adresa a nedojde k návratu na správné místo.

Instrukce *RETI* slouží obdobně pro návrat z obsluhy přerušení, což je vlastně také podprogram, který vyvolá sám procesor pokud k přerušení dojde. Vzhledem k tomu, že při obsluze přerušení jsou automaticky zakázána všechna další, musí tato instrukce po návratu na původní místo programu opět povolit přerušení. To je jediný rozdíl oproti instrukci *RET*.

3.1.7 Logické operace

Logickými operacemi, které jednočipový počítač provádí, jsou logický součin, součet, výlučný součet a negace a nastavení nebo nulování bitů. Seznam instrukcí provádějících logické operace je uveden v tabulce 3.4.

Kód	Operandy	Popis	Operace	Indikátor	Slova / hodinové cykly
AND	Ra, Rb	logický součin registrů Ra a Rb	Ra←Ra AND Rb	Z,N,V	1/1
ANDI	Ra, data8	logický součin registru Ra a konstanty	Ra←Ra AND data8	Z,N,V	1/1
OR	Ra, Rb	logický součet registrů Ra a Rb	Ra←Ra OR Rb	Z,N,V	1/1
ORI	Ra, data8	logický součet rergistru Ra a konstanty	Ra←Ra OR data8	Z,N,V	1/1
EOR	Ra, Rb	výlučný logický součet registrů Ra a Rb	Ra←Ra XOR Rb	Z,N,V	1/1
COM	Ra	negace registru Ra	Ra←$FF-Ra	Z,C,N,V	1/1
SBR	Ra, data8	nastaví bity registru Ra podle konstanty	Ra←Ra OR data8	Z,N,V	1/1

CBR	Ra, data8	nuluje bity registru Ra podle konstanty	Ra←Ra AND (data8)'	Z,N,V	1/1
TST	Ra	testuje nulovou nebo zápornou hodnotu v registru Ra	Ra←Ra AND Ra	Z,N,V	1/1
CLR	Ra	nuluje registr Ra	Ra←Ra XOR Ra	Z,N,V	1/1
SEB	Ra	nastavení všech bitů v registru Ra	Ra←$FF		1/1

Tab. 3.4, přehled instrukcí logických operací

3.1.8 Aritmetické operace

Aritmetické instrukce umožňují provádění celočíselnému součtu, rozdílu a dalších operací. Jejich hlavním znakem je, že při jejich vykonávání se mění všechny příznaky stavového registru.

Příznak *Z* stavového registru je nastaven, je-li výsledek součtu roven nule. Příznak *C* je nastaven při přenosu, neboli při přetečení registru. Příznak H je nastaven dochází li k přenosu mezi dolní a horní čtveřicí bitů *(poloviční přenos – half carry)*. Příznak N je nastaven, je li výsledek operace záporný, což je vyhodnoceno nastavením nejvyššího bitu registru. Tento příznak se uplatňuje pouze při práci se zápornými čísly. Příznak V je nastaven při tzv. přetečení dvojkového doplňku, což se projevuje změnou znaménka výsledku. Pokud se nepracuje se zápornými čísly, nemá příznak V žádný význam. Přehled instrukcí aritmetických operací je uveden v tabulce 3.5.

Kód	Operandy	Popis	Operace	Indikátor	Slova / hodinové cykly
ADD	Ra, Rb	sečte registry Ra, Rb	Ra←Ra+Rb	Z,C,N,V,H	1/1
ADC	Ra, Rb	sečte registry Ra, Rb a příznak C	Ra←Ra+Rb+C	Z,C,N,V,H	1/1
ADIW	Ra, data6	přičte konstantu ke slovu Ra+1:Ra	Ra+1:Ra←Ra+1:Ra+data6	Z,C,N,V,S	1/2
SUB	Ra, Rb	odečte registry Ra a Rb	Ra←Ra-Rb	Z,C,N,V,H	1/1
SUBI	Ra, data8	odečte od registru Ra konstantu	Ra←Ra-data8	Z,C,N,V,H	1/1
SBIW	Ra, data6	odečte konstantu ke slovu Ra+1:Ra	Ra+1:Ra←Ra+1:Ra-data6	Z,C,N,V,S	1/2
SBC	Ra, Rb	odečte registry Ra a Rb a příznak C	Ra←Ra-Rb-C	Z,C,N,V,H	1/1
SBCI	Ra, data8	odečte od registru Ra konstantu a C	Ra←Ra-data8-C	Z,C,N,V,H	1/1
INC	Ra	inkrementuje registr Ra	Ra←Ra+1	Z,N,V	1/1
DEC	Ra	dekrementuje registr Ra	Ra←Ra-1	Z,,N,V	1/1
NEG	Ra	druhý doplněk registru Ra	Ra←$00-Ra	Z,C,N,V	1/1

Tab. 3.5, přehled instrukcí aritmetických operací

3.1.9 Porovnávací a další instrukce

Porovnávací instrukce slouží k porovnávání obsahů registrů. Velmi často se používají v souvislosti s řízením běhu programu v součinnosti s instrukcemi skoků. Jsou jakousi změněnou funkcí již uvedených instrukcí. Například při porovnání operandů *Ra a Rb* instrukcí *CP* dochází k jejich vzájemnému odečtení stejně jako při provádění instrukce *SUB*, ale výsledek není uložen a jejich porovnání nezpůsobí změnu těchto registrů. Výsledek operace je možné zjistit testováním stavu registrů. Pokud jsou čísla stejná, bude jejich rozdíl roven nule a bude nastaven indikátor nulového výsledku aritmetické operace (indikátor nuly). Jinými slovy bude nastaven bit číslo 1 v registru stavů apod.

Další instrukce, které nelze zařadit mezi žádnou z výše uvedených kategorií jsou instrukce *NOP*, která nevykonává žádnou akci, ale pouze zpožďuje vykonávání programu o jeden strojový cyklus. Pokud je procesor časován hodinovým signálem o frekvenci 10 MHz, bude její vykonání trvat 100 ns. Tato instrukce se často využívá pro malé zpoždění vykonávání programu, pokud na jednočipový počítač navazují další obvody, které nejsou schopny reagovat na změnu stavu v čase kratším, než je vykonání instrukce (například obvody typu CMOS, jejichž doba reakce je až 500 ns). Instrukce *NOP* může být využita případně i k časování. Instrukce *SLEEP* uvádí jednočipový počítač do tzv. režimu spánku. Více o něm bylo uvedeno v části 2.5 bakalářské práce. Poslední instrukcí je *WDR*, která nuluje tzv. hlídací obvod (Watchdog), který sleduje korektní běh mikropočítače. Více o tomto obvodu je zmíněno v části 2.4 bakalářské práce. Seznam porovnávacích a dalších instrukcí je uveden v tabulce 3.6.[17]

Kód	Operandy	Popis	Operace	Indikátor	Slova / hodinové cykly
CP	Ra, Rb	porovnání registrů Ra a Rb	Ra-Rb	Z,C,N,V,H	1/1
CPC	Ra,Rb	porovnání registrů Ra a Rb včetně C	Ra-Rb-C	Z,C,N,V,H	1/1
CPI	Ra, data8	porovná registr Ra s konstantou	Ra-data8	Z,C,N,V,H	1/1
NOP		prázdná operace			1/1
SLEEP		volí režim spánku			1/1
WDR		nuluje hlídací obvod			1/1

Tab. 3.6, přehled porovnávacích a dalších instrukcí

Ukázkový příklad jednoduchého programu vytvořeného pomocí jazyka symbolických adres a použitého pro konfiguraci portu B jednočipového počítače jako výstupního s následujícím přepínáním portu do stavu logické 1 a 0 je uveden v příloze číslo 3 bakalářské práce. Pro srovnání jsou v příloze uvedeny i programy se stejným zadáním využívající vyšších programovacích jazyků.

3.2 Jazyk C

Programovací jazyk C je všeobecně použitelný programovací jazyk známý svou efektivitou, ekonomií a přenositelností. Jeho vznik provázela myšlenka programovacího jazyka s vyšší přehledností a abstraktností než je jazyk symbolických adres. Tato charakteristika jej předurčuje pro prakticky všechny oblasti programování. Obzvláště užitečným je jazyk C v systémovém programování, protože umožňuje psaní rychlých, kompaktních programů, které jsou snadno adaptovatelné pro jiné systémy. Dobře napsaný C program je často stejně rychlý jako program napsaný v jazyce symbolických adres. Navíc je čitelnější a snadněji aktualizovatelný.

Programování mikropočítačů v tomto programovacím jazyku je poměrně výhodné, protože jazyk C je poměrně blízký strojovému jazyku a nárůst výsledného strojového kódu vůči jazyku symbolických adres je poměrně malý. Přesto však má stupeň abstraktnosti potřebný pro vyšší programovací jazyk. Tento jazyk je však poměrně náročný na znalosti programátora z oblasti hardwarového provedení

konkrétní aplikace a z oblasti softwarového provedení operačního systému, ve kterém program pro jednočipový počítače vzniká.

Jazyk C je pružným jazykem, který ponechává řadu programátorských rozhodnutí na programátorovi. Vnucuje v podstatě jen některá omezení - například konverzi typů a byl navržen jako poměrně malý jazyk, kombinující efektivitu a výkonnost. Jazyk C v sobě obecně neobsahuje funkce pro provádění vstupu a výstupu, alokaci paměti, práci s obrazovkou a řízení procesů. Pro tyto funkce sahá programátor do systémových knihoven. Díky tomu je možné se rozhodnout, zda bude využita funkce ze standardních knihoven, nebo bude vytvořena varianta speciálně pro určité konkrétní potřeby. Tento návrh pomáhá oddělit vlastnosti jazyka od vlastností spojených s konkrétním procesorem, architekturou atd. V programovacím jazyku C je tak možné snadněji psát programy přenositelné mezi různými systémy.[18]

V případě vytváření programů pro jednočipové počítače s pomocí jazyka C narážíme na určité omezení, které má přímou souvislosti s jednoduchostí programování. A sice, že při vývoji programů pro mikropočítače nebývá k dispozici jinak standardní knihovna funkcí, která by programování mohla usnadnit. Jedná se o funkce definující práci s komunikačními porty, displeji, čítači/časovači a jinými periferiemi jednočipových počítačů. Jejich nepřítomnost je nutné řešit samostatným vytvořením těchto funkcí.

K převedení kódu jazyka C do výkonného programu je potřeba použít překladač, ostatně jako u ostatních programovacích jazyků. V případě jazyka C je na výběr veliké množství překladačů do strojového kódu. Některé jsou šířeny jako svobodný software a jsou k dispozici pro operační systémy Windows a Linux.

3.3 Bascom

Jazyk Bascom je poměrně nový programovací jazyk vytvořený v roce 1995 a je určen k vytváření výkonných programů pro jednočipové počítače Atmel. Vzhledem k rozdílné architektuře mikropočítačů Atmel AVR a Atmel AT89xx existují i dvě verze tohoto programovacího jazyku a to Bascom AVR a Bascom 8051. Rozdíl je především v dostupných příkazech pro práci s mikropočítači a samozřejmě v překladači programu, který vytvořený program převádí do jazyka srozumitelného jednočipovému počítači a musí proto respektovat jeho vnitřní strukturu a dostupné instrukce. Překladače kódu jsou k dispozici pro operační systémy Windows a ve své základní verzi jsou volně dostupné.

Programovací jazyk Bascom svým způsobem zápisu připomíná programovací jazyk Basic, který se používá jako výukový, poměrně jednoduchý, nástroj a umožňuje napsání programu pro běžný počítač.

Jazyk Bascom užívá několik základních prvků, pomocí kterých se sestavuje program jako celek. Mezi tyto prvky patří příkazy, datové prvky, operátory, procedury a funkce. Jazyk Bascom také není

citlivý na velikost písmen, a tedy dva výrazy lišící se pouze velikostí písmen jsou považovány za stejné.

3.2.1 Datové prvky

Datový typ je základní charakteristikou datových prvků. Určuje jejich vlastnosti ve smyslu určení typu dat. Datový typ prvku určuje způsob reprezentace v paměti (způsob uložení) a způsob interpretace této hodnoty. Různé datové typy vypovídají zároveň o maximální možné velikosti datového prvku. Základní datové typy shrnuje tabulka 3.7.

Typ	Velikost [bit]	Rozsah
bit	1	0,1
boolean	1	0,1 (nelze použít true a false)
byte	8	0 až 255
integer	16	-32 768 až 32 767
word	16	0 až 65 535
long	32	-2 147 483 648 až 2 147 483 647
single	32	$1,5 . 10^{-45}$ až $3,4 . 10^{38}$
string	do 254 x 8	každý znak jeden byte

Tab. 3.7, elementární datové typy

Mezi samotné datové prvky se řadí *proměnné*, které pojmenovávají úsek paměti, ve kterém se mohou uchovávat hodnoty jejichž vlastnosti jsou určené typem proměnné. Proměnná je vždy jednoznačně určena svým názvem a je deklarována příkazem. Název proměnné začíná vždy písmenem a pro jejich pojmenování není možné použít rezervovaná slova (názvy příkazů, jejich části atd.). Při ukládání čísel do proměnné lze v programovacím jazyku Bascom určit, zda je číslo v hexadecimálním nebo v binárním tvaru. Například:

`P1 = &B1010` nebo `P1 = &HA` je totéž co `P1 = 10`

Deklarování typu proměnné je v programovacím jazyku Bascom povinné a provádí se pomocí příkazu *Dim*. Při deklaraci proměnné, která obsahuje řetězec dat je nutné vždy určit i délku tohoto řetězce, a tuto délku již není možné měnit při běhu programu. Jedná se tedy o řetězec pevné délky. Syntaxe příkazu je následující:

```
Dim promenna1 As Integer, promenna2 As Integer, promenna3 As String * 16
```

Dále v programovacím jazyku Bascom není možná tzv. implicitní deklarace proměnných tj., že všechny proměnné, které nejsou explicitně definovány, jsou určitého typu. Určitou výhodou je poté možnost alespoň explicitní deklarace typu proměnných začínajících na určitý znak. Toto lze provést pomocí příkazů *Defint, Defbit, Defbyte, Defword, Deflong* a *Defsgn*. Příklad uvedený níže znamená, že veškeré proměnné, jejichž jméno začíná písmenem x, jsou typu integer.

```
Defint x
```

Dalším z datových prvků jsou *konstanty*. Ty na rozdíl od proměnných nealokují žádný prostor v paměti a jsou pouhým symbolickým vyjádřením určité hodnoty. Mohou být uvedeny přímo jako číslo nebo řetězec znaků nebo pomocí identifikátoru. Pro identifikátory konstant platí stejná pravidla jako pro proměnné. Typ konstanty je určen jejich zápisem a není tedy určován pomocí deklarace jako tomu je u proměnných. Konstanty mohou být definovány pomocí příkazu *Const.*

```
Const S = "řetězec "
Const A = 9
Const B = &B1010
```

3.2.2 Operátory a výrazy

Operátory určují druh operace mezi operandy. Tyto operátory mohou být unární nebo binární podle počtu zúčastněných operandů, přičemž unárním operátorům přísluší pouze jediný operand. Příkladem může být operátor logické negace A = *NOT* B. Binární operátor obklopují dva operandy. V programovacím jazyku Bascom je většina operátorů binárních. Podle typu výsledku operace určené operátorem rozlišujeme operátory aritmetické, logické a relační.

Aritmetické operátory se uplatňují při aritmetických operací s operandy číselných typů. Programovací jazyk Bascom rozeznává aritmetické operátory sčítání (+), odčítání (-), násobení (*), dělení (/) jako základní aritmetické operátory, mezi které můžeme řadit i tzv. unární mínus (A=-B). Dalšími operátory jsou celočíselné dělení (\), zbytek po celočíselném dělení - modulo (MOD) a mocnění (^). Operace mocnění není dostupná v některých starších verzích programovacího jazyku Bascom 8051.

Logické operátory vystupují v logických výrazech, které jsou často součástí parametrů příkazů *if* a *while*, které budou popsány později, kde slouží jako podmínka. Logické operátory jsou ve své podstatě operátory číselné, ale pracující s jednotlivými bity hodnot. Jazyk Bascom rozeznává logické operátory negace (Not), logický součin (AND), logický součet (OR) a výlučný součet (XOR).

Operátory relační slouží k určení pravdivosti nebo nepravdivosti výrazu a používají se k větvení programu například s pomocí příkazu *if* a *while*. Logickými hodnotami v tomto případě nejsou *true* a *false*, jak je tomu zvykem u podobných programovacích jazyků, ale 0 a 1. V současné době v dokumentaci programovacího jazyka Bascom není možné nalézt odpovídající datový typ *Boolean*, ale v novějších verzích tohoto jazyka je možné ho použít a kód je překladačem bez problémů přijat. Relační operátory jazyka Bascom jsou rovnost (=), nerovnost (<>), menší než (<), větší než (>), menší nebo rovno (<=) a větší nebo rovno (>=). Výsledky operací relačních operátorů vrací logickou hodnotu 1, pokud první operand splňuje podmínku danou operátorem vůči operandu druhému. V opačném případě vrací logickou hodnotu 0.

Výrazem se rozumí vhodná kombinace operandů a operátorů, které dávají výsledek číselného nebo řetězcového typu. Výrazy také mohou vystupovat jako parametry jiných příkazů.

3.2.3 Příkazy, funkce, procedury

Příkaz jazyka Bascom je samostatnou syntaktickou jednotkou a může být strukturovaný. Takový příkaz má tělo, které je ohraničené úvodním a koncovým řádkem. Právě příkazy dělají z tohoto programovacího jazyka velmi efektivní nástroj pro vytváření výkonného programu, jelikož kromě běžných příkazů pro deklarace proměnných, přiřazovacích příkazů, příkazů cyklů a příkazů pro řízení průběhu aplikace obsahuje i příkazy specializované. Ty mnohdy realizují i značně složité operace, jejichž programování by v nižším programovacím jazyce bylo náročné. Příkazy jsou rozeznávány tzv. klíčovým slovem.

Zvláštní pozornost by měla být věnována pracím s cykly, které se užívají k provádění operací a příkazů vícekrát, v závislosti na ukončovací podmínce nebo předem dané hodnotě počtu opakování. Výkonný program aplikace – přesněji řečeno jeho tělo – se z pravidla uzavírá do cyklu, který je určený příkazem *Do … Loop*, přičemž není definována podmínka pro ukončení tohoto cyklu. Vzniká tak nekonečný cyklus, ve kterém je neustále vykonáván program jednočipového počítače. Toto je podstatný rozdíl mezi programem pro jednočipový počítač a běžným program (například pro osobní počítač), kde je toto zacyklení spíše nežádoucí.

Procedury a funkce vytvořené v programovacím jazyku Bascom můžeme souhrnně nazývat podprogramy, neboť jsou částí většího celku. Z hlediska běhu programu jsou procedury a funkce jakýmsi odskokem do jiné části kódu, po jehož vykonání se běh programu vrátí zpět a pokračuje ve vykonávání dalších příkazů. Rozdíl mezi procedurou a funkcí je ve způsobu volání jednoho z těchto podprogramů a ve způsobu přenosu hodnot parametrů do, nebo z podprogramu.

Procedurou se rozumí sekvence výkonných příkazů nazvaná jedním jménem a lze ji tedy určitým způsobem chápat jako nový (vlastní) příkaz programovacího jazyka Bascom. Způsob jejího použití pak vyplývá z výše uvedeného - uplatňuje se na místech, kde se mohou uplatnit i příkazy Bascomu. Její použití je výhodné především z hlediska výsledné velikosti programu, který bude vykonáván jednočipovým počítačem. Existuje-li určitá sekvence příkazů, kterou je nutné v rámci celého programu vykonávat opakovaně, je úspornější tuto sekvenci definovat právě jako proceduru a při běhu programu na tuto sekvenci pouze odkázat. Doba vykonání programu se tímto sice prodlouží o dobu vykonání instrukce přeskoku na tuto sekvenci a o dobu vykonání instrukce návratu zpět, ale není nutné tyto sekvence příkazů zapisovat opakovaně. Příklad použití procedury se vstupním parametrem je uveden níže:

```
Declare Sub cekej(A As Long)
Call cekej(10)
…
Zde se nachází vlastní program
…
Call cekej(20)
End
Sub cekej (sec As Long)
```

```
Wait sec
End Sub
```

Funkce představují, podobně jako procedury, sekvenci příkazů, ale na rozdíl od procedur jsou funkce svázány s výstupní hodnotou. Proto vystupují v místech programu, kde lze obecně použít výraz, konstantu anebo proměnnou. Typ návratové hodnoty funkce je dán explicitní volbou při jejím zápisu. Pro přenesení návratové hodnoty zpět do těla programu se používá přiřazovací příkaz zapsaný nejčastěji na konci samotné funkce. Použití funkcí se nabízí například při potřebě opakovaného provedení složené aritmetické nebo logické operace. V programovacím jazyku Bascom totiž není možné v jednom zápisu provést několik operací. Například zápis *W* = *C* + *D* + *F* nebo *R* = S Xor *T And U* způsobí chybu v překladu a tuto složenou operaci bude nutné rozepsat na jednotlivé dílčí kroky. S pomocí deklarované funkce je zajištěno předání parametrů které do funkce vstupují a s pomocí již zmíněného přiřazení na konci funkce je zajištěno předání výsledné hodnoty zpět do těla programu.

```
Declare Function soucet_tri_operandu(Byval C As Integer, D As Integer, F As Integer) As Integer
Dim A As Integer : Dim B As Integer : Dim V As Integer : Dim P As Integer
A = 5
B = 100
V = 10
T = soucet_tri_operandu(A, B, V)
End
Function soucet_tri_operandu(Byval C As Integer, D As Integer, F As Integer) As Integer
Local mezikrok1 As Integer
Local mezikrok2 As Integer
mezikrok1 = C + D
mezikrok2 = mezikrok1 + F
soucet_tri_operandu = mezikrok2
End Function
```

S pomocí výše použitých parametrů je možné měnit chování podprogramů, čímž se tyto podprogramy mohou stát řešením celé skupiny podobných výpočetních úloh (jako v uvedeném případě). Z hlediska jednočipového počítače jsou tyto parametry vždy uložené v operační paměti. Rozdíl je ve způsobu předání tohoto parametru podprogramu a rozlišují se dva druhy tohoto předání, respektive dva druhy parametrů.

Parametry volané odkazem se do kódu podprogramu přenášejí pomocí odkazu na adresu v paměti jednočipového počítače. V podprogramu se pak s tímto odkazem pracuje stejně jako se samotnou hodnotou a každá změna, kterou v parametru podprogram provede, je promítnuta i do těla kódu, který podprogram vyvolal. Stejným způsobem se předávají parametry z podprogramu zpět do těla výkonného programu, a proto je tento způsob volen jako standardní. Jeho explicitní volbu je

možné provést pomocí klíčového slova *Byref* umístěným před parametr. Tento způsob přenosu se také volí při přenášení dlouhých datových řetězců nebo polí jejichž kopírováním do dalšího úseku paměti by mohlo dojít k vyčerpání její kapacity.

Parametry volané hodnotou se do kódu podprogramu přenášejí přes jakýsi zásobník, který je vytvořený úsekem paměti mikropočítače. S touto pamětí pak podprogram pracuje jako s každou jinou proměnnou, ale po vykonání podprogramu zanikají hodnoty uložené v této paměti a paměť mikropočítače se vrátí do stavu, ve kterém byla před zavoláním podprogramu. Výhoda tohoto přenosu parametru je v tom, že původní hodnota skutečné proměnné, která vstoupila do podprogramu jako parametr, není za žádných okolností podprogramem změněna. Pracuje se tedy pouze s kopií původní proměnné. Tento způsob přenosu parametrů prodlužuje dobu vykonání podprogramu (je potřeba provést větší množství instrukcí) a, jak již bylo zmíněno, dochází rovněž k zaplnění většího množství paměti po dobu vykonávání podprogramu. Tento způsob přenosu parametru je v programovacím jazyce Bascom zvolen pomocí klíčového slova *Byval* a je použit i výše v ukázkovém kódu deklarace funkce.[19]

4 Návrh aplikace s jednočipovým počítačem

Při návrhu aplikace s jednočipovým počítačem musíme brát ohled na několik základních bodů, kterým následně podřídíme postup při návrhu a zpracování výsledné aplikace. Hlavním bodem je samozřejmě definování významu aplikace a požadavky na ní. S tím souvisí výběr konkrétního vhodného typu mikropočítače, návrh schématu elektrického obvodu, který bude aplikaci realizovat a samotný návrh programu, který bude mikropočítačem vykonáván. Zároveň je nutné věnovat pozornost i požadavkům na rychlost aplikace, což nemalou mírou ovlivňuje veškeré části návrhu aplikace - od výběru konkrétních součástek až po výběr programovacího jazyku. Při praktickém využití aplikace bývá důležitá také cena, s čímž mohou souviset další požadavky například na minimum podpůrných obvodů, a snaha o co největší integraci použitých součástek. Mohou být kladeny nároky na jednoduchost výroby, na fyzické rozměry aplikace, na rychlost a způsob komunikace s okolím.

4.1 Generátor náhodných stavů

Jako názorný příklad aplikace s jednočipovým počítačem bude uveden návrh obvodu a programu, který je ve své podstatě generátorem náhodných stavů na portech mikropočítače.

V souladu s výše uvedeným stál na počátku aplikace požadavek na vytvoření jednoduchého a levného obvodu, kterým bude realizováno osvětlení modelů budov tak, aby co nejvíce připomínalo realitu.

Pro aplikaci byl vybrán jednočipový počítač Atmel AT89S51, který je klonem procesoru Intel 8051. Tento jednočipový počítač obsahuje 32 programovatelných vstupů/výstupů, nedisponuje žádným speciálním rozhraním, velikost jeho paměti je 4 kB a maximální frekvence připojovaného oscilátoru je 33 MHz. Vzhledem k povaze aplikace není téměř vůbec důležitá její rychlost. Z toho důvodu je také volen levnější typ mikropočítače s kompletní instrukční sadou.

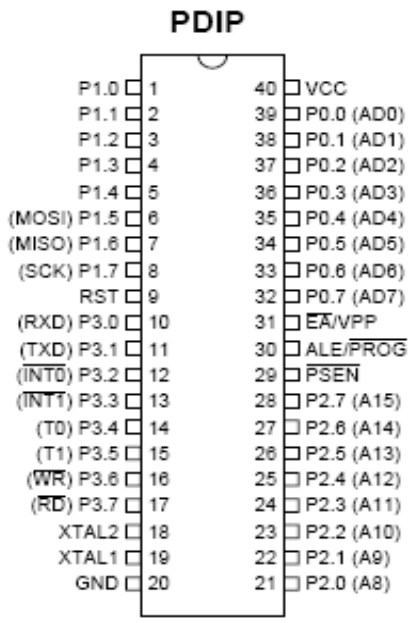
Mikropočítač v této aplikaci nepotřebuje přijímat žádná data, proto všechny jeho porty budou sloužit jako výstupní. Vykonávaný program bude náhodně generovat stavy jednotlivých výstupů. Výstup v úrovni logické 1 se projeví rozsvícením příslušné diody LED. Právě v tomto místě již vyplývá nutnost použití dalších pomocných obvodů a to z důvodu maximální proudové zatížitelnosti jednočipového počítače. Ten je schopen na většině svých portů přímo budit diody LED a to maximálním proudem 10 mA/pin ovšem s omezením proudu 15 mA-26 mA/port. Připojením buzených LED diod by v případě všech výstupů v úrovni logické 1 došlo k přetížení a k možnému poškození jednočipového počítače. Je tedy nutné použít pro buzení LED pomocný obvod, který svými parametry aplikaci vyhovuje. Tímto obvodem je třístavový budič osmibitové sběrnice 74LS245 (TTL), případně jeho pomalejší verze 74HCT245 (CMOS). Jeho celková proudová zatížitelnost je 500 mA, doba reakce se pohybuje od 10 ns u TTL verze do 500ns u verze CMOS. Výstup tohoto obvodu je přes předřadný rezistor připojen k diodám LED pomocí konektoru.

Dále v aplikaci nejsou nutné žádné obvody sledující aktivitu jednočipového počítače. Není nutné používat časovače ani čítače. Resetovací obvod je realizován RC článkem a aplikace je napájena ze stabilizovaného zdroje napětí. Jako zdroj hodinového signálu je použit krystal o frekvenci 10 MHz.

Pro napsání programu vykonávaného mikropočítačem je pro jednoduchost volen vyšší programovací jazyk Bascom 8051. Při jeho použití nemusíme do detailu znát vnitřní strukturu použitého mikropočítače. Zápis programu v tomto jazyce je také pro programátora velmi úsporný na rozdíl od jazyka symbolických adres a samotný vývoj programové části aplikace je značně jednoduchý.

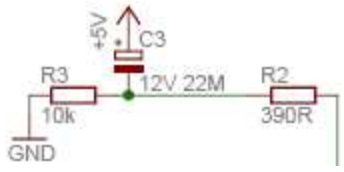
4.1.1 Parametry integrovaných obvodů a jejich význam v aplikaci

Jak již bylo zmíněno, v navržené aplikaci se uplatňují pouze dva typy integrovaných obvodů. Prvním z nich je jednočipový počítač Atmel AT89S51. Ten náleží ke skupině procesorů s kompletní instrukční sadou, obsahuje čtyři osmibitové vstupně-výstupní porty, dva šestnáctibitové čítače/časovače, disponuje sériovým komunikačním rozhraním. Velikost programové paměti je 4 kB, napájecí napětí se pohybuje v rozsahu 4,0 V až 5,5 V a frekvence externího oscilátoru se může pohybovat od 0 do 33 MHz. Pro tuto aplikaci byl vybrán procesor v pouzdru PDIP se 40ti vývody viz. obrázek. 4.1. Pro návrh elektrického obvodu je nutné prostudovat dokumentaci výrobce tohoto mikropočítače a vyhovět jeho požadavkům na zapojení. V našem případě sledujeme především pin *RST*, který slouží pro resetování mikropočítače. V dokumentaci je řečeno, že pro resetování obvodu je nutné po dobu dvou strojových cyklů přivést na pin *RST* úroveň log. 1. Dále sledujeme porty *EA/VPP* a *PSEN*, které v naší aplikaci nebudou využity. Pin *EA/VPP* slouží pro přístup k externí paměti. V případě nevyužití této funkce je nutné vývod připojit k napájecímu napětí. Pin *PSEN* vysílá externí paměti čtecí impuls. Vzhledem k tomu, že tato aplikace externí paměť nevyužívá a výrobce nespecifikuje žádný požadavek pro tento případ, můžeme nechat vývod nezapojený. Tímto jsou splněny veškeré požadavky na vlastní provoz jednočipového počítače. Dalším požadavkem výrobce mikroprocesoru je připojení blokovacích kondenzátorů k oscilátoru. Jejich hodnota je dána výrobcem a to 30 pF ± 10 pF pro krystalový oscilátor a 40 pF ± 10 pF pro RC oscilátory.



Obr.4.1, rozložení pinů mikropočítače Atmel AT89S51

Obvod pro přivedení logické úrovně 1 na vstup *RST* při startu aplikace bude realizován RC členem s ochranou *RST* vstupu při poruše a probití kondenzátoru. Jeho schéma můžeme vidět na obr. 4.2. Doba přivedení úrovně logické 1 na vstup *RST* je dána hodnotami součástek R3 a C3. V tomto obvodu využíváme tzv. přechodové jevy. To jsou fyzikální děje probíhající v časovém intervalu mezi dvěma ustálenými stavy. Jak uvádí dostupná literatura RC obvod je obdobně tvořen zdrojem napětí a sériovým zapojením ideálního odporu (rezistoru) a ideálního kondenzátoru. Po připojení zdroje napětí bude obvodem procházet nejvyšší možný proud, který bude omezen pouze rezistorem $I = \frac{U}{R}$. Nenabitý kondenzátor se tak bude zpočátku chovat jako zkrat. Čím více se však bude kondenzátor nabíjet (bude v něm vzrůstat nahromaděný náboj) a poroste jeho napětí (u_c), tím menší proud procházející obvodem bude. Za dobu *T* tak bude obvodem procházet



Obr.4.2, resetovací obvod

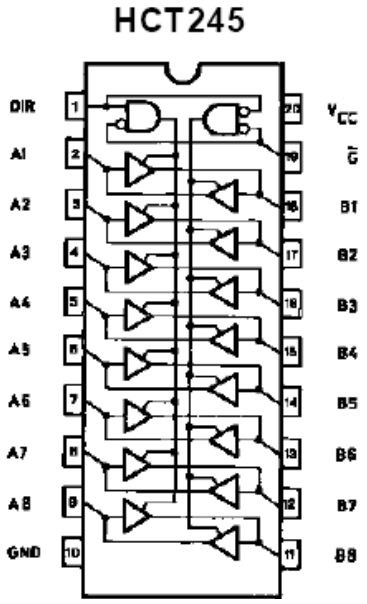
proud $i = \frac{U - u_c}{R}$ “. [20] Rezistor R2 pak slouží jako již zmíněná ochrana mikropočítače tím, že v případě poškození kondenzátoru omezí proud do *RST* vstupu.

Použitý budič sběrnice pracuje s napájecím napětím v rozsahu -0,5 V až 7 V. Můžeme proto použít společné napájení pro tento obvod a pro mikropočítač. Maximální výstupní proud každým jednotlivým výstupem je ± 35 mA a vyhovuje tedy pro přímé buzení LED. Uspořádání a popis vývodů je znázorněn na obrázku 4.3. Obvod obsahuje 8 vstupů a výstupů vodičů sběrnice *A1-A8* a *B1-B8*, napájecí vývody *Vcc* a *GND*, vývod pro určení směru toku dat *DIR* a dále vstup *G* pro převedení výstupů do stavu vysoké impedance (odpojení) . Jedná se tedy o integrovaný obvod s třístavovou logikou. Pravdivostní tabulka obvodu 74245 je uvedena v tabulce 4.1.

VSTUP		**FUNKCE**	
G	***DIR***	***A sběrnice***	***B sběrnice***
L	L	výstup	vstup
L	H	vstup	výstup
H	X	odpojeno	odpojeno

L - úroveň log.0
H - úroveň log.1
X - úroveň log. 0 nebo 1

Tab. 4.1, pravdivostní tabulka obvodu 74245



Obr. 4.3, uspořádání vývodů obvodu 74245

Vstup *G* integrovaného obvodu je použit pro odpojení výstupů po dobu resetování jednočipového počítače, kdy jsou jeho výstupy ve stavu logické 1, což je pro tuto aplikaci nežádoucí. Tato funkce aplikace je realizována připojením vstupu *G* k resetovacímu obvodu. Celkové elektrické schéma aplikace je uvedeno v příloze č.1., návrh desky plošných spojů pak v příloze č.2 bakalářské práce.

4.1.2 Výkonný program aplikace

Program vykonávaný jednočipovým počítačem řídí běh celé aplikace. Pro jeho napsání byl zvolen programovací jazyk Bascom 8051. Řídící program realizuje samotný generátor náhodných stavů. Program generuje náhodný stav vždy jen jednoho bitu a respektuje tak požadavek na plynulé přecházení mezi různými stavy jednotlivých portů mikropočítače. Celý zdrojový kód programu je uveden v příloze číslo 4. Pro generování náhodných hodnot je použit příkaz *RND(x)* jazyka Bascom. Této funkci je v aplikacích jednočipových počítačů potřeba věnovat pozornost. Jedná se o matematickou funkci, která je generována deterministicky a proto při každém opětovném zapnutí aplikace bude generována stejná „náhodná“ posloupnost. V případě nutnosti použití skutečných náhodných čísel by bylo nutné tyto generovat s využitím různých fyzikálních veličin – například měřením aktuální hodnoty napětí v obvodu, šumu, teploty atd.

5 Ukázková aplikace s jednočipovým počítačem – pulzně šířková regulace elektromotoru

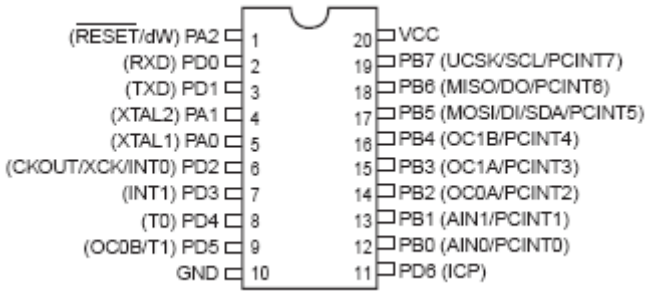
Jako ukázková aplikace je uveden modul řízení otáček stejnosměrného elektromotoru s využitím jednočipového počítače. Tento v aplikaci zastává místo aktivního obvodu a generuje pulzně šířkový signál jehož prostřednictvím dochází k regulaci otáček elektromotoru. Dále jednočipový počítač komunikuje prostřednictvím sériové sběrnice RS232 s připojeným osobním počítačem, který předává aplikaci požadavky na rychlost otáčení elektromotoru.

Pro tuto aplikaci byl vybrán běžně dostupný jednočipový počítač Atmel ATtiny2313 z řady mikropočítačů AVR s redukovanou instrukční sadou. Ten v sobě integruje sériové rozhraní, dva čítače/časovače a tedy i 4 nezávislé kanály, které mohou generovat signál pulzně šířkové modulace. Dále obsahuje analogový komparátor, obvod sledování běhu a integruje v sobě i rezonanční obvod s předděličkou, který je možné použít místo externího zdroje hodinového signálu.

Mikropočítač v této aplikaci komunikuje s osobním počítačem připojeným přes rozhraní RS232. Jejich prostřednictvím je aplikaci předávána informace o požadované rychlosti otáčení stejnosměrného motoru. Dle přijatého požadavku je nastaven jeden z kanálů pulzně šířkové modulace jehož výstup je připojen na bázi spínacího tranzistoru. Ten spíná pracovní proud do regulovaného motoru. Vzhledem k vlastnostem použitého jednočipového počítače a k povaze aplikace bude použit externí zdroj hodinového signálu. Pro napsání programu vykonávaného mikropočítačem je pro jednoduchost volen vyšší programovací jazyk Bascom AVR. Při jeho použití není nutné do detailu znát vnitřní strukturu použitého mikropočítače. Zápis programu v tomto jazyce je také pro programátora velmi úsporný na rozdíl od jazyka symbolických adres a samotný vývoj programové části aplikace je značně jednoduchý.

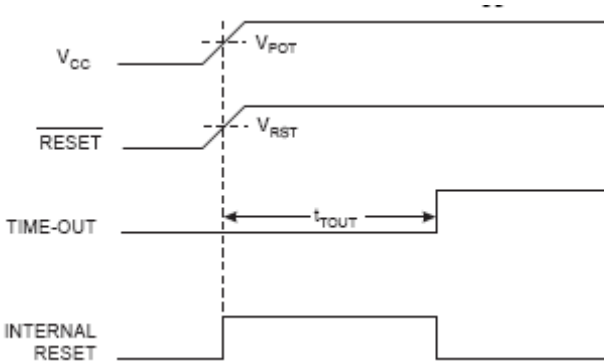
5.1 Parametry použitých obvodů a komponent a jejich význam v aplikaci

V aplikaci se uplatňuje jednočipový počítač Atmel ATtiny2313 z řady procesorů AVR. Ten je vybaven 2 kB vnitřní paměti pro program s garantovaným počtem 10 000 zápisů a dále pamětí EEPROM, která v této aplikaci nenachází využití. Mikropočítač v sobě integruje jeden 8mi bitový a jeden 16ti bitový čítač/časovač se 4mi kanály pulzně šířkové modulace, analogový komparátor, obvod sledování stavu, univerzální sériové rozhraní



Obr. 5.1, rozložení pinů Atmel ATtiny2313

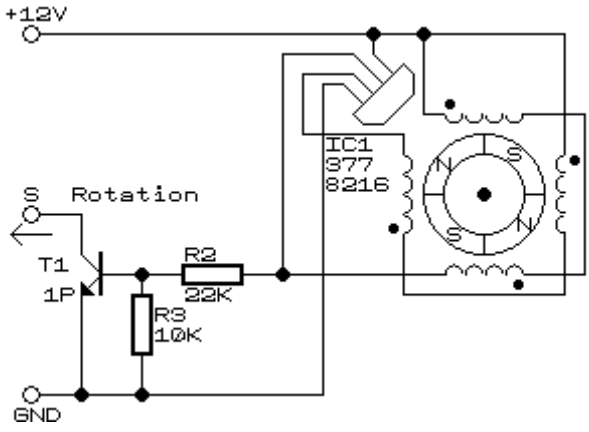
a rozhraní sériové sběrnice. V případě nevyužití těchto periferií disponuje mikropočítač třemi vstupně-výstupními porty a to 3 bitovým portem A, 8mi bitovým portem B a 6ti bitovým portem D. V aplikaci je použita jeho verze obvodu v pouzdru PDIP s 20ti vývody. Schéma vývodů je zobrazeno na obrázku 5.1. Napájecí napětí jednočipového počítače je stanoveno výrobcem a doporučuje se napětí v rozmezí 2,7 V – 5,5 V při pracovní frekvenci v rozsahu 0 – 10 MHz. Pro pracovní frekvence v rozsahu 10 – 20 MHz by přiváděné napětí nemělo klesnout pod hodnotu 4,5 V. Jak je patrné z obrázku 5.1 – vstup reset mikropočítače je invertován (viz resetovací obvod aplikace pro generování náhodných stavů obrázek 4.2), a tedy při potřebě externího resetovacího impulzu by bylo nutné tento vstup přivést do stavu logické 0. V této aplikaci však nemusí být přítomen externí obvod pro počáteční reset jako tomu bylo v předchozí aplikaci, neboť mikropočítač je vybaven obvodem *zapnutí po resetu (Power On Reset)*. Tento obvod sleduje přiváděné napájecí napětí a je aktivní pokaždé, když hodnota napájecího napětí je menší než 1,2 V. Při dosažení tohoto napětí je vstup *RESET* interně uveden do stavu logické 1 a zároveň je spuštěn speciální čítač, který stanoví dobu tohoto impulzu. Časový průběh interního resetu jednočipového počítače je patrný z obrázku 5.2.



Obr. 5.2, časový průběh interního resetu mikropočítače Atmel ATtinv 2313

Dále je v aplikaci použit externí oscilátor realizovaný krystalem.

Dalším, v aplikaci použitým, obvodem je MAX232, který je prostředníkem komunikaci mezi mikropočítačem a osobním počítačem. Jeho použití je dáno komunikací aplikace a osobního počítače pomocí sériové sběrnice RS232, přičemž je nutné vysílané a přijímané signály převádět na odpovídající napěťové úrovně, které každá z komunikujících stran používá. V případě osobního počítače jsou používány napěťové úrovně ± 15 V, v případě Atmel ATtiny2313 pak 0 a 5 V. Napájení obvodu je požadováno v rozsahu 4,5 až 5,5 V. Dále pro správnou funkci obvodu je nutné na některé jeho vývody připojit elektrolytické kondenzátory o kapacitě 1 μF. Tento obvod pracuje na principu tzv. nábojové pumpy. Schéma propojení obvodu MAX232 s jednočipovým počítačem bylo uvedeno na obrázku 2.8.



Obr. 5.3, elektrické schéma motoru s feritovým rotorem

Pro další návrh celkového obvodu je nutné znát parametry řízeného spotřebiče a těmto poté přizpůsobit i další použité součástky. V případě ukázkové aplikace je jako spotřebič

použit ventilátor běžně používaný v osobních počítačích. Motory těchto ventilátorů jsou bezkomutátorové, stejnosměrné, s feritovým rotorem a statorem ze spínaných cívek. Cívky pootáčejí rotorem podle signálu přicházejícího z Hallovy sondy umístěné uvnitř statoru.

Některé ventilátory také disponují výstupem pro měření otáček. Ten je obvykle realizován pomocí jednoho z vývodů Hallovy sondy nebo je připojen ke kolektoru jednoho ze spínaných tranzistorů. Elektrické schéma ventilátoru je uvedeno na obrázku 5.3. Jeho pracovní napětí je dáno výrobcem na 12 V stejnosměrných a jmenovitý proud 180 mA.

Jelikož napájecí napětí i proud přiváděný do použitého motoru je zcela mimo rozsah hodnot kterými disponuje jednočipový počítač, je nutné tento proud spínat s pomocí tranzistoru. Jeho parametry jsou voleny v závislosti na spínaném napětí a proudu a na frekvenci generovaného signálu pulzně šířkové modulace použité ke spínání tohoto tranzistoru. Těmto parametrům vyhovuje celá škála tranzistorů různých typů a výrobců.

Generovaný signál pulzně šířkové modulace je zvolen fázově korigovaný v 8mi bitovém režimu. Tím je dána maximální hodnota souvisejícího čítače/časovače. Výstupní frekvenci generovaného signálu určíme dle příslušného vzorce $f_{OCnXPWM} = \frac{f_{(osc)}}{2 \cdot N \cdot vrchol}$. V tomto konkrétním případě, kdy není použito předdělení a maximální hodnota čítače/časovače je 255, dostáváme dosazením do vzorce frekvenci pulzní modulace 15,686 kHz a doba periody je tedy 63,75 µs. Použitý tranzistor musí být schopný reagovat na změnu stavu (logická úroveň 0 nebo 1) v čase kratším než je nejkratší možný vygenerovaný impuls. V případě fázově korigované pulzně šířkové modulace bez předdělení vstupního hodinového signálu je minimální doba trvání tohoto impulzu 2 periody řídícího vstupního signálu - tomto případě (frekvence oscilátoru 8 MHz) 250 ns.

Pro účely ukázkové aplikace je vybrán tranzistor typu NPN, konkrétně typ KSY21 což je spínací tranzistor pro průmyslové použití. Výrobcem tranzistoru KSY21 udávaná hodnota zavření nebo otevření tranzistoru je 40 ns, maximální napětí mezi emitorem a kolektorem U_{CE} je 15 V a maximální spínaný proud I_C je 500 mA. Tento tranzistor vyhovuje použití v aplikaci ve všech sledovaných parametrech. Dalším parametrem tranzistoru, který je nutný brát v úvahu je jeho maximální proud bází. Tento údaj se nepodařilo z tabulky výrobce tohoto tranzistoru zjistit, ale lze předpokládat, že tento proud je až 10x nižší než proud mezi emitorem a kolektorem. Pro omezení tohoto proudu na odpovídající hodnotu bude mezi výstup generátoru pulzně šířkové modulace a bázi tranzistoru zařazen rezistor o hodnotě 330 Ω.

Paralelně k motoru je také připojen keramický kondenzátor o kapacitě 100 nF, který slouží jako jednoduchá ochrana před rušením vznikajícím v motoru.

Celkové elektrické schéma ukázkové aplikace je uvedeno v příloze číslo 5 bakalářské práce.

5.2 Návrh výkonného programu ukázkové aplikace

Pro vytvoření řídícího programu byl zvolen vyšší programovací jazyk Bascom AVR zejména z důvodu jednoduššího řízení běhu pulzně šířkové modulace a komunikace s osobním počítačem.

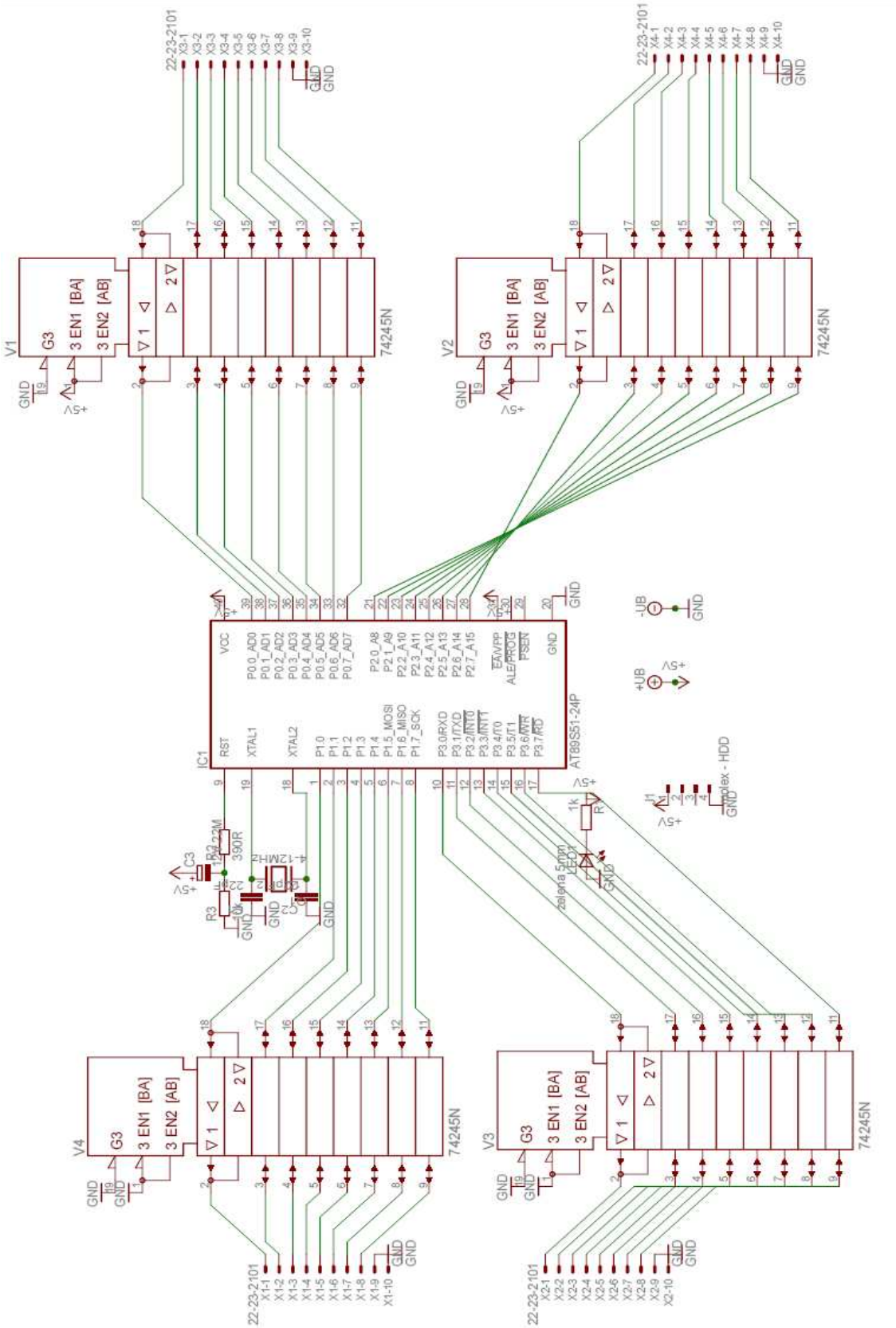
Vytvoření programu je ulehčeno především použitím příkazů jazyka Bascom pro práci se sériovým rozhraním a pro práci s čítači/časovači. Příkazy *Print* a *Input* slouží k vypsání řetězce do sériového rozhraní, příkaz *Input* pak i pro načtení hodnoty do příslušné proměnné.

Příkazem *Print* jednočipový počítač informuje o svém stavu a zároveň informuje obsluhu o prováděných činnostech. Příkaz vždy přidává na konec vysílaného řetězce i znaky pro návrat kurzoru a odřádkování, což při použití jazyka symbolických adres musí učinit programátor. Příkazem *Input* je načítána hodnota do komparačních registrů časovače 1 a to nejdříve do registru *OCR1A* a poté do *OCR1B*. Pro větší přehlednost je možné v jazyce Bascom tyto registry nazvat jako *pwm1a* a *pwm1b.* Pro přenos této hodnoty ze sériového portu jsou použity proměnné *motor0* a *motor1*. Ty jsou dimenzovány pro délku 1 Bytu a to vzhledem k použitému osmibitovému módu pulzně šířkové modulace. Jiným, taktéž možným způsobem zadání hodnot do komparačních registrů by bylo opakovaným použitím funkce *Waitkey()* s následným přenosem čteného znaku do proměnné. Tento cyklus by byl přerušen čtením znaku enter.

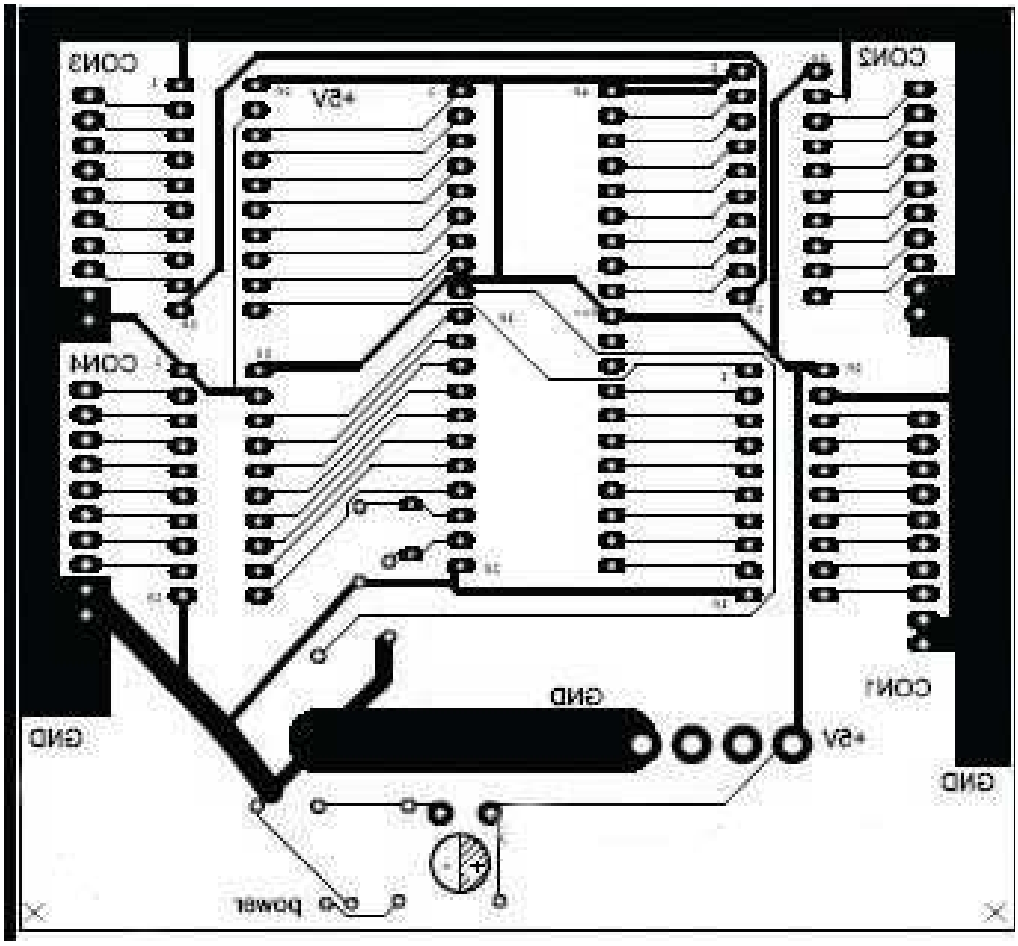
Vzhledem k omezenému počtu periferií použitého jednočipového počítače již není možné tuto aplikaci rozšířit o další funkcionality jako například měření otáček připojeného elektromotoru jak bylo původně zamýšleno pro účely ukázkové aplikace. Pro tuto funkcionalitu by bylo potřeba mít volně k dispozici další dva čítače/časovače, z nichž jeden by pracoval v režimu čítače a druhý v režimu časovače. Jednoduchými početními operacemi by poté bylo možné vyhodnocovat počet čítačem přijatých impulzů za čas který naměří časovač.

V případě jednočipového počítače Atmel ATtiny2313 jsou však k dispozici pouze dva čítače/časovače a výše zmíněnou funkcionalitu by bylo možné realizovat pouze s pomocí externího signálu přivedeného na vstup *ICP*. Přivedením tohoto signálu by mohlo dojít k odečtení aktuální hodnoty časovače. Použité typy stejnosměrných motorů sice disponují vývodem pro měření vlastních otáček, avšak elektronické obvody těchto motorů generují 2 impulzy za jednu otáčku. Toto by dávalo možnost měřit dobu, za kterou zmíněný motor vykoná půl otáčky. Pro určitou informační hodnotu by bylo nutné tento údaj pomocí aritmetických funkcí přepočítat na počet otáček za minutu nebo sekundu. Takto vypočtený údaj by však byl pouze informativní a zároveň lze předpokládat, že by byl i velmi nepřesný.

Příloha číslo 1, elektrické schéma generátoru náhodných stavů



Příloha číslo 2, návrh desky plošných spojů generátoru náhodných stavů



Příloha číslo 3, ukázka výkonného programu v jazyce symbolických adres a v jazyce Bascom

Jazyk symbolických adres:

```
1.      .INCLUDE "m16def.inc"
2.      .LIST
3.
4.      .CSEG           ;kódový segment
5.      .DEF REG=R16    ;pracovní registr
6.      .EQU DDR=DDRC   ;řízení směru
7.      .EQU PORT=PORTC ;port
8.
9.      LDI REG,$FF
10.     OUT DDR,REG
11.     LDI REG,LOW(RAMEND)
12.     OUT SPL,REG
13.     LDI REG,HIGH(RAMEND)
14.     OUT SPH,REG
15.     LDI REG,$FF
16.     LDI R20,1
17.
18. SMYCKA:         OUT     PORT,REG
19.         RCALL   CEKEJ
20.         ADD REG,R20
21.         RJMP    SMYCKA
22.
23. ;čekací rutina (čeká asi 0,5 s)
24. CEKEJ:
25.         LDI     R17,36
26. CYK3:   LDI R18,150
27. CYK2:   LDI R19,250
28. CYK1:   DEC R19
29.     BRNE    CYK1
30.         DEC R18
31.     BRNE    CYK2
32.         DEC     R17
33.     BRNE    CYK3
34.         RET
```

Jazyk Bascom:

```
$crystal = 12000000
P0 = 0
Do
P0 = 1
Waitms 500
P0 = 0
Waitms 500
Loop
```

Příloha číslo 4, zdrojový kód generátoru náhodných stavů

```
$crystal = 12000000
Dim Sec As Integer , A As Byte , Sec1 As Integer
Dim X As Byte , Y As Byte
X = 0
Declare Sub Pockej(sec As Long)        'cekaci procedura
P0 = 0                'vsechny porty do 0
P1 = 0
P2 = 0
P3 = 0
Call Pockej(12)

For X = 0 To 7   'postupne na porty O-3 nastavi 0,1,3,7,15,31,63,127,255

For A = 0 To 3
Select Case A
Case 0 : P0.x = 1
Case 1 : P1.x = 1
Case 2 : P2.x = 1
Case 3 : P3.x = 1
End Select

Call Pockej(11)
Next
Next             'nahodne nastavi vzdy jeden pin portu do 0 nebo 1

Do

Generuj:         'A - port, X - pin, Y - O,1
A = Rnd(5)
X = Rnd(9)
Y = Rnd(3)

Gen_delay:       'generuje zpozdeni
Sec1 = Rnd(19)
If Sec1 <= 7 Then
Goto Gen_delay
End If
```

```
Call Pockej(sec1)

If Y = 0 Then

Select Case A
Case 0 : Reset P0.x
Case 1 : Reset P1.x
Case 2 : Reset P2.x
Case 3 : Reset P3.x
End Select

Else

Select Case A
Case 0 : Set P0.x
Case 1 : Set P1.x
Case 2 : Set P2.x
Case 3 : Set P3.x
End Select
End If

Loop
```

Použitá literatura

[1] KOPELENT, Jiří. ATMEL AVR neboli mikroprocesory rodiny 90Sxxx a dalších. *HW.cz : Vše o elektronice a programování* [online]. 2003 [cit. 2009-04-03]. Dostupný z WWW: <http://hw.cz/Teorie-a-praxe/Programovani/ART993-ATMEL-AVR-neboli-mikroprocesory-rodiny-90Sxxx-a-dalsich.html>.

[2] VOBORSKÝ, Pavel. *AT89C2051 | Pavel Voborský* [online]. [2007] , 3. dubna 2009 18:20:48 [cit. 2009-04-04]. Dostupný z WWW: <http://pavob.net/tag/2051/>.

[3] *Arithmetic logic unit - Wikipedia, the free encyclopedia:* [online]. [2007] , 22 March 2009, at 22:40 [cit. 2009-04-04]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Arithmetic_logic_unit>.

[4] *Control unit - Wikipedia, the free encyclopedia:* [online]. [2008] , 11 March 2009, at 10:51 [cit. 2009-04-04]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Control_Unit>.

[5] VLACH, Jaroslav. *Počítačová rozhraní - přenos dat a řídící systémy*. Praha : BEN - technická literatura, 1995. Rozhraní RS-232C (V.24), s. 67.

[6] MATOUŠEK, David. *Práce s mikrokontroléry Atmel AT89C2051*. 1. vyd. Kubicová Iveta. Praha : BEN - technická literatura, 2002. ISBN 80-7300-048-2. Popis čítačů a časovačů, s. 136.

[7] DŘÍNEK, Milan. *Architektura AVR* [online]. 2000 [cit. 2009-04-13]. Dostupný z WWW: <http://avr.hw.cz/architektura/arch_avr.html>.

[8] VÁŇA, Vladimír. *Mikrokontroléry Atmel AVR programování v jazyce BASCOM*. 1. vyd. Praha : BEN - technická literatura, 2004. ISBN 80-7300-115-2. Příklady programování v jazyce BASCOM, s. 94.

[9] *Pulzně šířková modulace : Wikipedie, otevřená encyklopedie* [online]. [2007] , 23. 2. 2009 v 04:38 [cit. 2009-04-18]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Pulzn%C4%9B_%C5%A1%C3%AD%C5%99kov%C3%A1_modulace>.

[10] MATOUŠEK, David. *Práce s mikrokontroléry Atmel ATmega16 : 4. díl*. 1. vyd. Praha : BEN - technická literatura, 2006. ISBN 80-7300-174-8. s. 191-219.

[11] *BASCOM-AVR : CONFIG TIMER1* [online]. c2008 , 31. října 2008 22:51:55 [cit. 2009-04-20]. Dostupný z WWW: <http://avrhelp.mcselec.com/config_timer1.htm>.

[12] MATOUŠEK, David. *Práce s mikrokontroléry Atmel AVR AT90S*. 2. vyd. Praha : BEN - technická literatura, 2006. ISBN 80-7300-209-4. WDT reset, s. 203-204.

[13] MATOUŠEK, David. *Práce s mikrokontroléry Atmel AT89C2051*. 1. vyd. Praha : BEN - technická literatura, 2002. ISBN 80-7300-048-2. Hlídací obvody watchdog, power-on reset a power-fail, s. 221-224.

[14] MATOUŠEK, David. *Práce s mikrokontroléry Atmel AVR AT90S*. 2. vyd. Praha : BEN - technická literatura, 2006. ISBN 80-7300-209-4. s. 354-356.

[15] MATOUŠEK, David. *Práce s mikrokontroléry Atmel AT89C2051*. 1. vyd. Praha : BEN - technická literatura, 2002. ISBN 80-7300-048-2. s. 220.

[16] *Jazyk symbolických adres : Wikipedie, otevřená encyklopedie* [online]. [2007] , 20. března 2009 15:20:55 [cit. 2009-04-15]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Assembler>.

[17] MATOUŠEK, David. *Práce s mikrokontroléry Atmel AVR AT90S*. 2. vyd. Praha : BEN - technická literatura, 2006. ISBN 80-7300-209-4. Instrukční soubor, s. 73-97.

[18] *Programování v jazyce C - 1. Úvod : HW.cz* [online]. 2003 [cit. 2009-04-20]. Dostupný z WWW: <http://hw.cz/Teorie-a-praxe/Programovani/ART857-Programovani-v-jazyce-C---1.-Uvod.html>.

[19] VÁŇA, Vladimír. *Mikrokontroléry ATMEL AVR programování v jazyce BASCOM*. 1. vyd. Praha : BEN - technická literatura, 2004. ISBN 80-7300-115-2. Základy jazyka Bascom, s. 15-32.

[20] *Přechodové jevy - Wikipedie, otevřená encyklopedie : Přechodový jev v RC obvodu* [online]. [2008] , 8. 4. 2008 v 23:42 [cit. 2009-04-13]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/P%C5%99echodov%C3%A9_jevy>.

[21] ATMEL® ATtiny2323 DATA SHEET, http://www.atmel.com