VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MODULÁRNÍ SIMULÁTOR MIKROKONTROLÉRU

MODULAR SIMULATOR OF MICROCONTROLLER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PAVEL VOSYKA
AUTHOR

VEDOUCÍ PRÁCE — Dr. Ing. PETR PERINGER,
SUPERVISOR

BRNO 2019

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

22047

Student: **Vosyka Pavel**
Program: Informační technologie
Název: **Modulární simulátor mikrokontroléru**
Modular Simulator of Microcontroller
Kategorie: Modelování a simulace
Zadání:

1. Prostudujte problematiku modelování a simulace mikrokontrolérů (MCU) a elektrických obvodů. Seznamte se s existujícími volně dostupnými simulátory MCU.
2. Navrhněte modulární simulátor MCU umožňující snadnou změnu typu simulovaného MCU. Navrhněte jednoduchý simulátor okolí MCU pro ověřování funkčnosti. Zaměřte se na vhodnou formu vizualizace stavu modelu pro výukové účely.
3. V C++ implementujte simulátor jednoho vhodně zvoleného MCU a ověřte jeho funkčnost na několika demonstračních aplikacích.
4. Zhodnoťte dosažené výsledky a navrhněte možná vylepšení simulátoru.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění prvních dvou bodů zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Peringer Petr, Dr. Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

V práci byly prozkoumány existující volně dostupné simulátory mikrokontrolerů. Byl navrhnut simulátor mikrokontroléru s grafickým uživatelským rozhraním. Simulátor je schopen vyměňovat simulovaný mikrokontrolér za jiný. Okolí simulace mikrokontroléru je ve formě několika předdefinovaných elektrických obvodů, které lze libovolně připojit k mikrokontroléru. Simulátor byl naimplementován a odzkoušen. Také byl do simulátoru naimplementován mikrokontrolér ATmega328p.

Abstract

In this thesis, free access existing microcontroller simulators were found. A microcontroller simulator with a graphical user interface was developed. The simulator is able to change between different microcontrollers. Surroundings of microcontroller's simulation are made of predefined electrical circuits, which can be arbitrarily connected with the microcontroller. The simulator was implemented and tested. Also, ATmega328p microcontroller was implemented in the simulator.

Klíčová slova

mikrokontrolér, MCU, simulace, AVR, ATmega328p

Keywords

microcontroller, MCU, simulation, AVR, ATmega328p

Citace

VOSYKA, Pavel. *Modulární simulátor mikrokontroléru.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dr. Ing. Petr Peringer,

Modulární simulátor mikrokontroléru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Dr. Ing. Petra Peringera. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .
Pavel Vosyka
15. května 2019

Poděkování

Děkuji panu Dr. Ing. Petru Peringerovi za jeho obětavou pomoc a jeho odborné rady, které mi v průběhu řešení této práce poskytoval.

Obsah

Kapitola 1

Úvod

Simulátory mikrokontrolérů se dají využít různými způsoby. Mohou být využité pro výukové účely, vývoj softwaru, testování a další. Každý simulátor má okolí mikrokontroléru zpracované jinak. Některé pouze sledují výstupy a umožňují manuální zadávání vstupů, některé simulují i připojené elektrické obvody. Modulární simulátor mikrokontrolérů je simulátor mikrokontrolérů, který umožňuje simulovat různé typy mikrokontrolérů.

Zdá se, že volně dostupné simulátory nevznikají příliš často a neočekává se, že se tato skutečnost brzy změní. V práci bude vytvořen simulátor mikrokontrolérů (MCU) s grafickým uživatelským rozhraním.

Ve druhé kapitole práce budou představena různá volně dostupná existující řešení. V třetí kapitole bude simulátor navržen včetně simulace elektrických obvodů, grafického uživatelského rozhraní a mikrokontroléru ATmega328p. Ve čtvrté kapitole bude popsáno řešení implementace navržených částí. Budou probrána řešení zajímavých částí a popsány drobné změny od návrhu a nedostatky, které vznikly při implementaci. Po implementaci bude také ověřena funkčnost simulátoru.

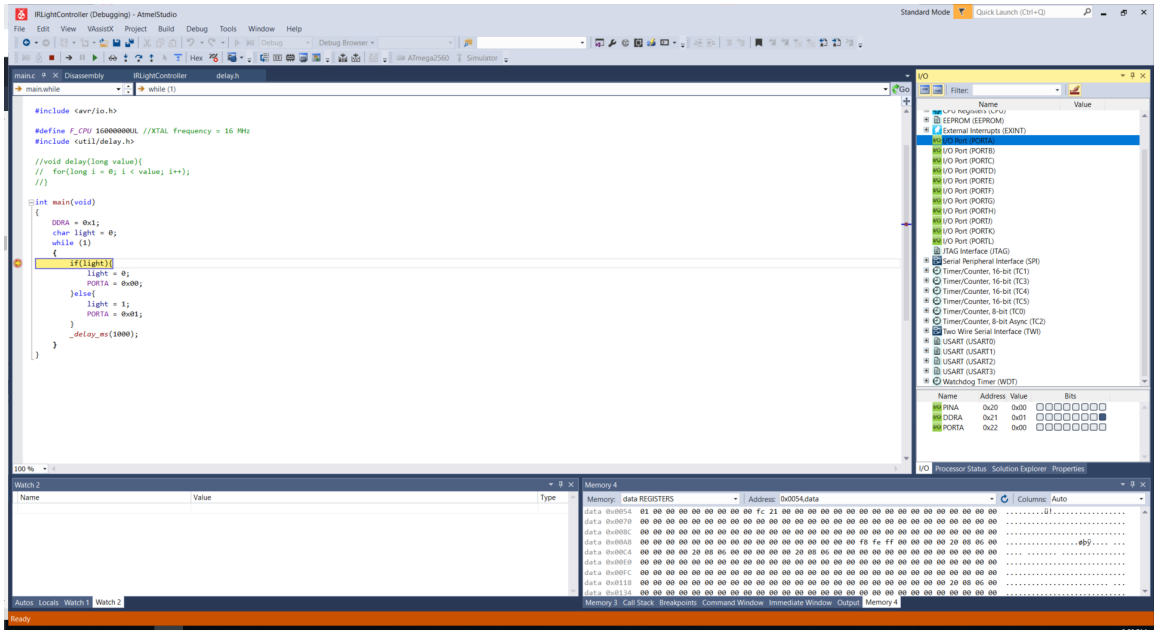
Kapitola 2

Volně dostupné simulátory mikrokontrolérů

Byl učiněn průzkum volně dostupných simulátorů mikrokontrolérů. V kapitole jsou simulátory představeny a porovnány mezi sebou.

2.1 Atmel Studio

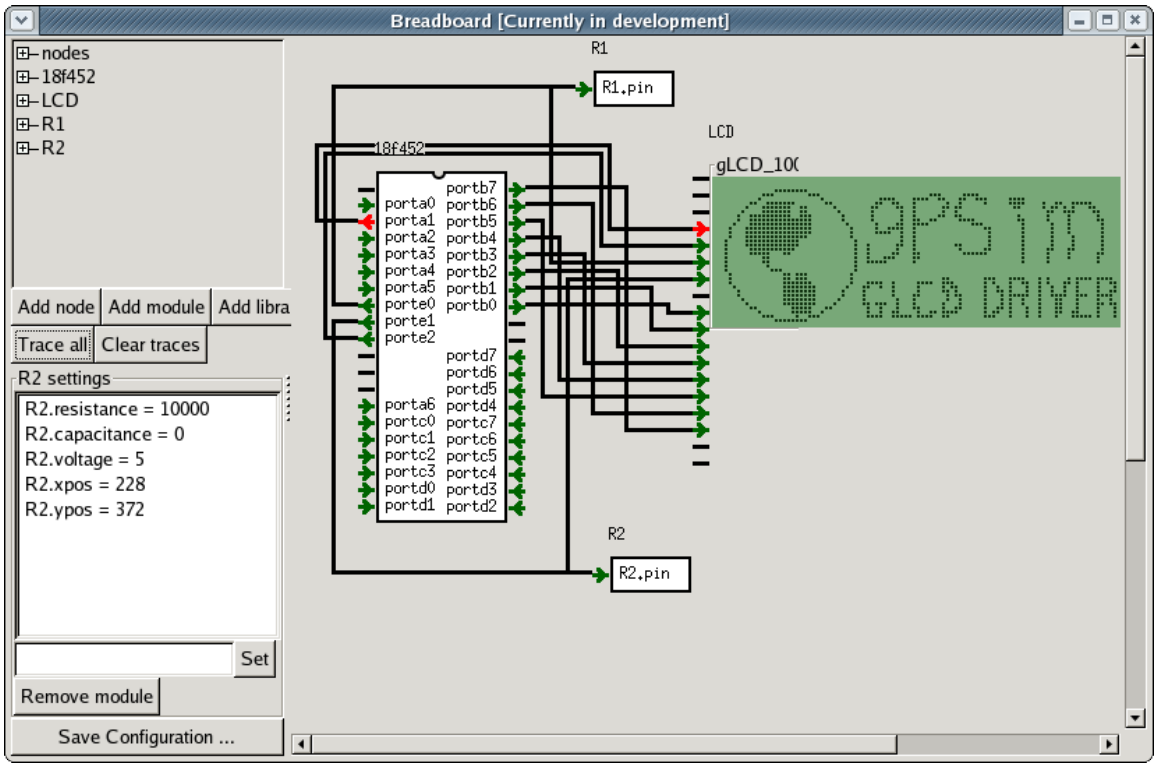
Atmel Studio je oficiální vývojové prostředí pro mikroprocesory s *AVR* architekturou. Je postavené na základě Visual Studia od Microsoftu. Kromě jiného, obsahuje také vestavěný simulátor. Snímek obrazovky na 2.1.



Obrázek 2.1: Atmel Studio

2.2 gpsim

Gpsim je simulátor PIC mikrokontrolérů. Umožňuje spouštění v příkazové řádce, ale také má své grafické rozhraní. Na grafické rozhraní používá framework gtk. Rozhraní je členěno do více oken. K mikrokontroléru je možné připojovat zařízení ve formě modulů. Například tlačítka, USART nebo display, ten je vidět na obrázku 2.2. Dalšími okny jsou například prohlížeč zdrojových kódů na obrázku 2.3 nebo prohlížeč registrů na obrázku 2.4.



Obrázek 2.2: qpsim[1]

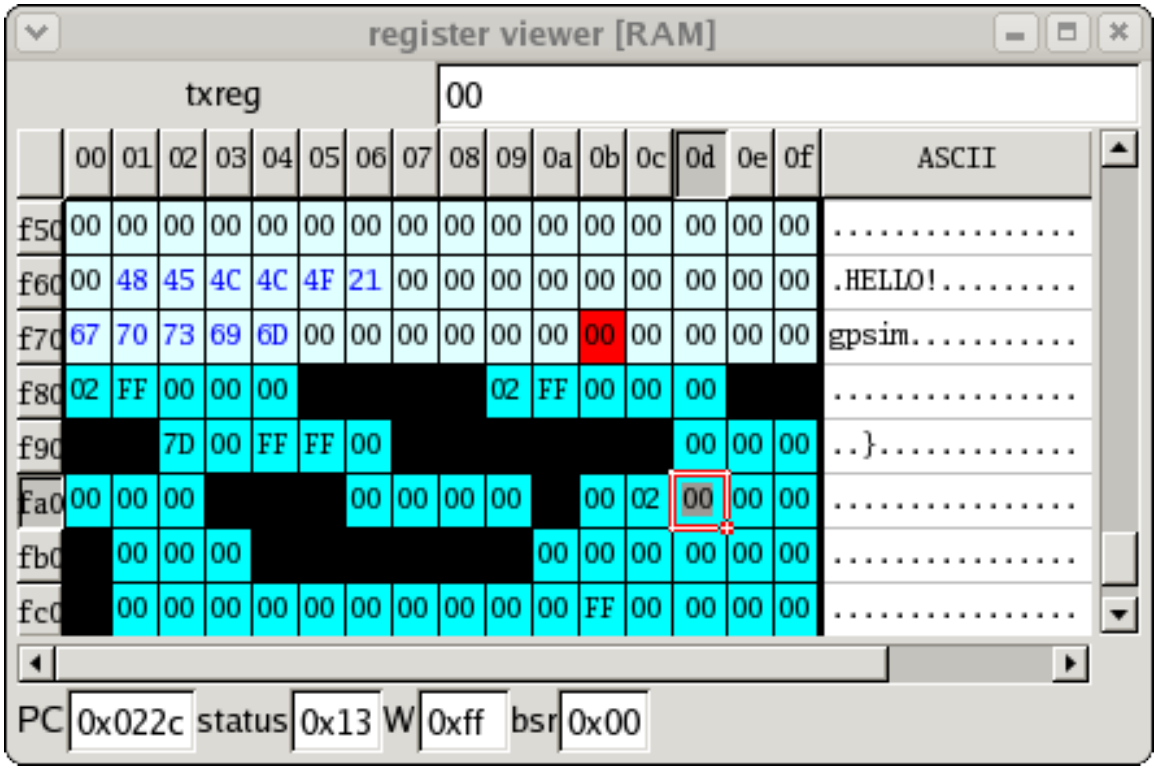
Source Browser: [Stopped]

tmr0_16bit.asm | p18f452.inc | coff.inc | delay.inc | delay.asm | tmr0_16bit.lst

```
123                 TMR0_8BitModeTest:
124
125                  .command  "echo *** Testing 8-bit mode"
126
127                   ; Stop the timer and clear its value
128 003E:6AD5               CLRF    T0CON
129 0040:6AD6               CLRF    TMR0L
130 0042:6AD7               CLRF    TMR0H
131                   .assert "(tmr0l == 0) && (tmr0h == 0)"
132
133                   ; The timer is off, so any value written to it should be read back
134                   ; unchanged.
135 0044:0E2A               MOVLW   42
136 0046:6ED6               MOVWF   TMR0L
137                   .assert "tmr0l == 42"
138
139 0048:6ED7               MOVWF   TMR0H
140                   .assert "tmr0h == 42"
141 004A:6F86               MOVWF   tmr0Hi
142                   ;-----------------------------------------------------------
143                   ; 8-bit mode tests
144                   ;
145                   ; First, TMR0 is tested in 8-bit mode and with interrupts disabled.
146                   ; All 8 combinations of PSA are tested.
147
148 004C:6B87               CLRF    psa     ; Shadow's the PSA bits of T0CON
149
150                 L1_tmr0_8BitModeTest:   ; Beginning of the loop
151
152                     ; Start off with T0CON and TMR0L in a known state.
153
```

PC 0x0042 status 0x0c W 0x2a bsr 0x00

Obrázek 2.3: prohlížeč zdojového kódu v gpsim[1]



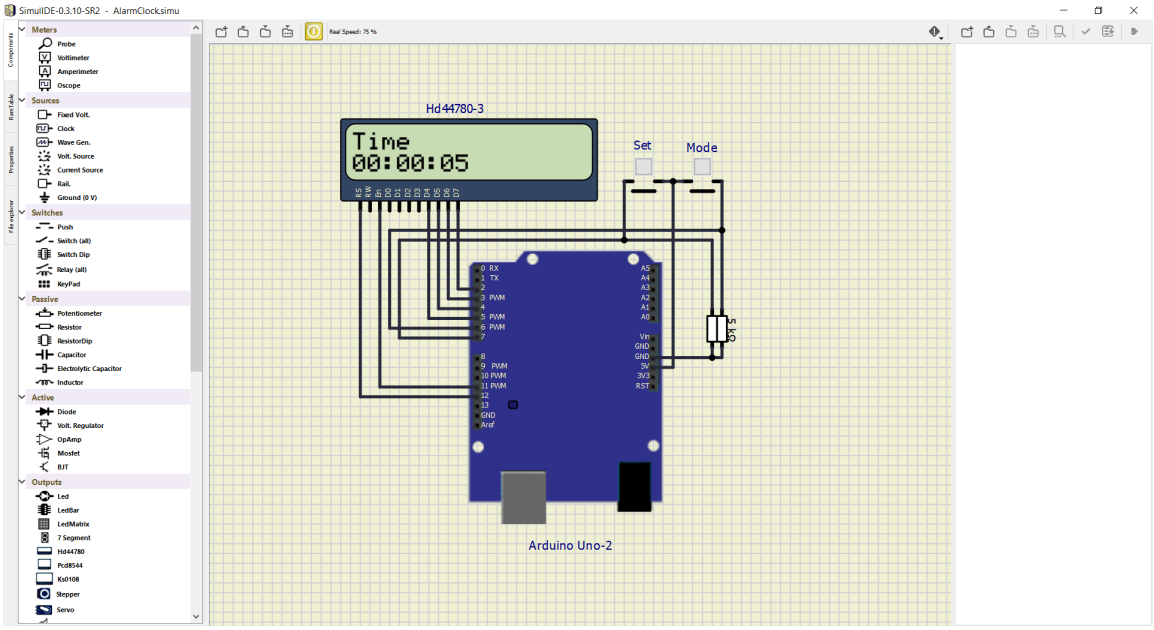
Obrázek 2.4: prohlížeč registrů v qpsim[1]

qpsim umožňuje načítání modulů. Moduly jsou absolutně nezávislé na simulátoru a jsou distribuované zvlášť jako knihovny. Cílem qpsim je poskytovat pro uživatele infrastrukturu pro jednoduchou konstrukci vlastního simulačního prostředí. [5]

2.3 SimulIDE

SimulIDE je vývojové prostředí podporující AVR i PIC mikrokontroléry. Editor je jednoduchý, soustředí se hlavně na simulaci. K mikroprocesoru je možné připojit elektrické obvody postavené ze součástek, které jsou dostupné v editoru.

[1] Zdroj: http://gpsim.sourceforge.net/

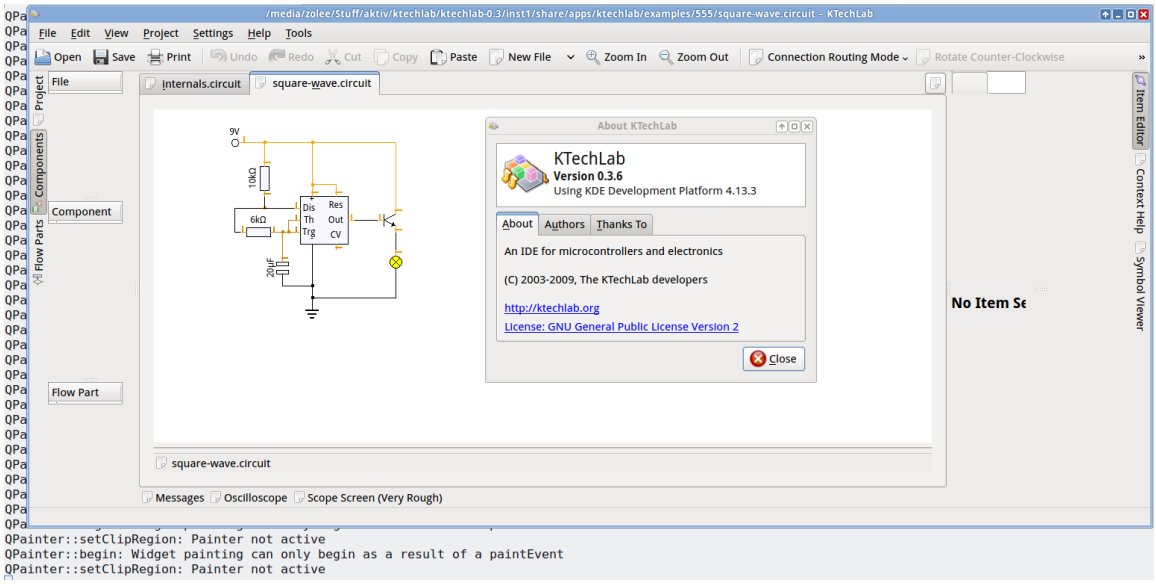


Obrázek 2.5: simulIDE

Simulátor není určen pro analýzu obvodů. Zaměřuje se na rychlost simulace a jednoduchost použití, proto přesnost simulace obvodů není příliš přesná. SimulIDE je mířený na studenty a amatéry na učení se a experimentování s jednoduchými obvody. [4]

2.4 KTechlab

Vývojové prostředí KTechLab obsahuje editor elektrických obvodů a mikrokontrolérů. Momentálně podporuje pouze mikrokontroléry PIC. Umožňuje vývoj programu pro MCU, kromě klasického textového editoru disponuje také grafovým editorem. Program je možné vytvořit v tomto editoru jako rozhodovací graf. Ukázka programu je na obrázku 2.6.

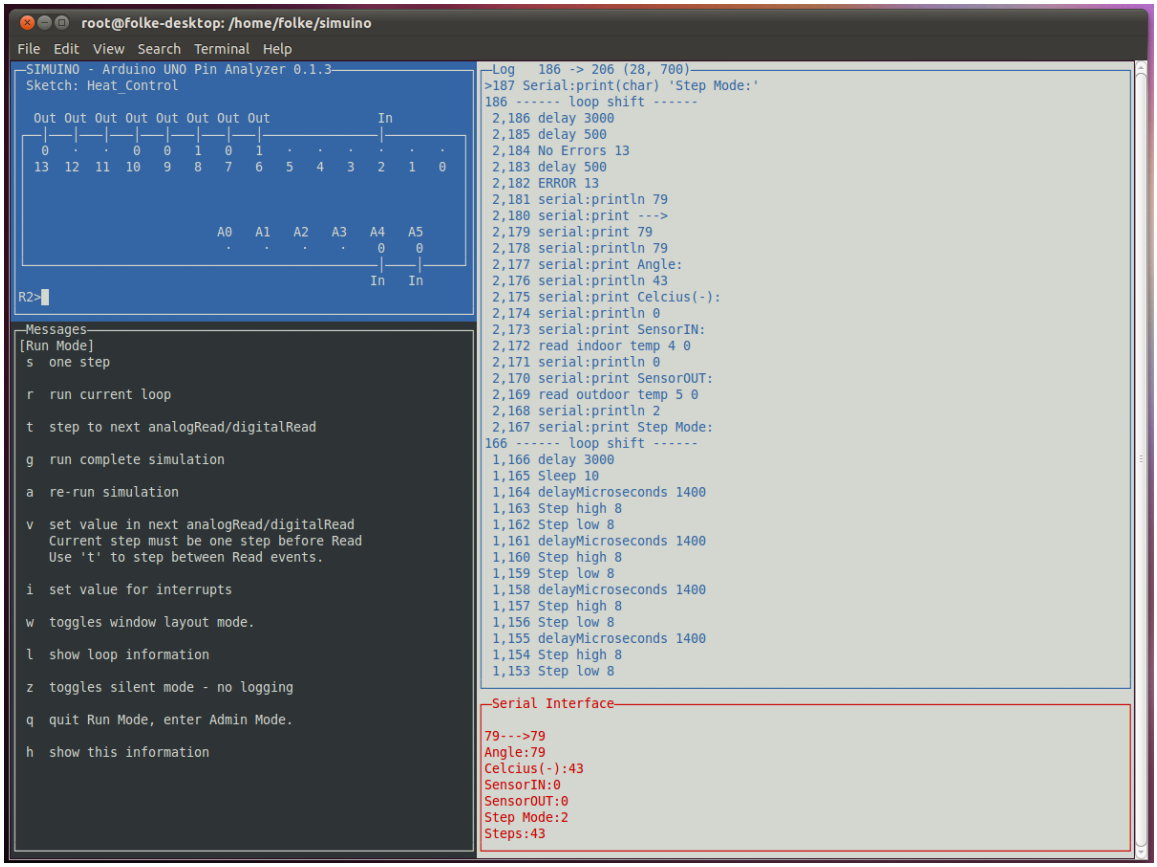


Obrázek 2.6: KTechLab[2]

2.5 Simuino

Simuino je simulátor platformy Arduino. Neposkytuje komplexní okolí MCU, piny se dají nastavovat pouze manuálně. Simulátor existuje ve dvou verzích. Verze v terminálu (obrázek 2.7) a verze webová (obrázek 2.8).

[2]Zdroj: https://github.com/ktechlab/ktechlab/wiki/Screenshots



Obrázek 2.7: Simuino[3]

Obrázek 2.8: Simuino[3]

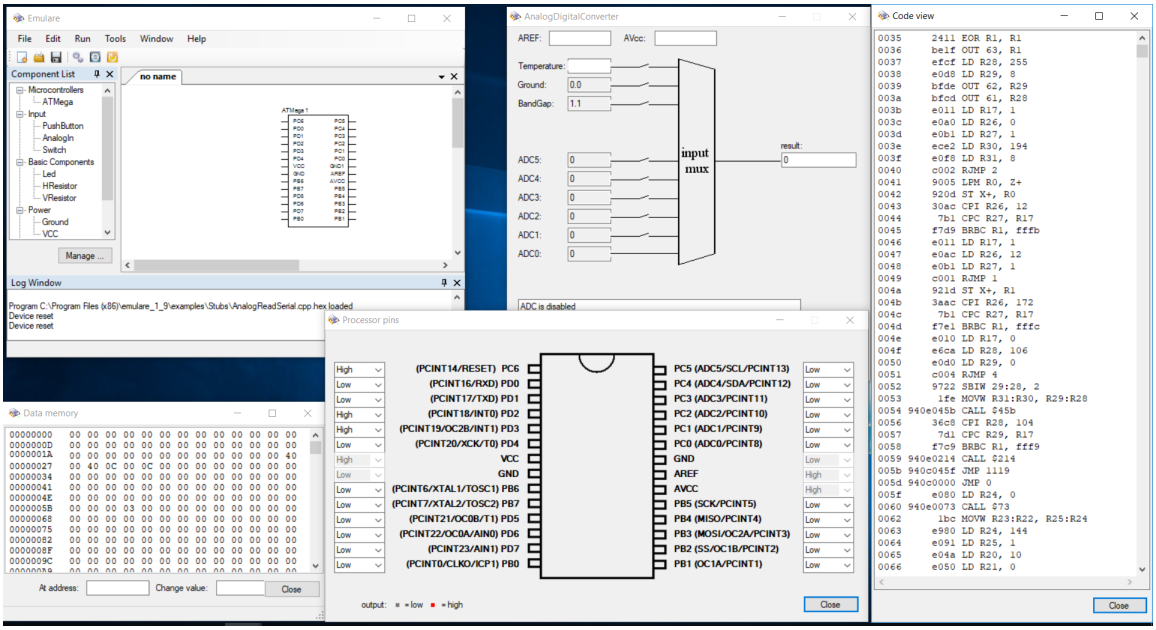
The purpose is to give anybody new to the Arduino concept, a possiblity to learn the basics of writing sketches. You can verify the functionality without having the Arduino board available. Simuino runs the sketch and shows the status of the digital, analog pins and serial output.

Cílem simuino projektu je dát komukoli bez přístupu k platformě Arduino možnost se naučit základy psaní programů.S Simuino vykonává program a ukazuje stav digitálních pinů, analogových pinů a sériovou komunikaci. [8]

2.6 emulare

Emulare simuluje mikrokontroléry řady ATmega. Momentálně podporuje pouze ATmega328p. Obsahuje také jednoduchý editor elektrických obvodů s několika komponentami. Grafické rozhraní je členěno do více oken. Na obrázku 2.9 je zobrazeno několik těchto oken. V levém horním rohu je editor obvodů, na pravé straně disassembler programu, v horní části uprostřed analogově digitální převodník MCU, v dolní části uprostřed vstup a výstup MCU a v levém dolním rohu zobrazení paměti MCU.

[3]Zdroj: http://web.simuino.com/



Obrázek 2.9: Emulare

2.7 simavr

simavr je simulátor mikrokontrolérů architektury AVR. simavr neobsahuje žádné okolí mikrokontroléru. Simulátor lze nainstalovat také jako knihovnu.

simavr umí zapisovat stav většiny svých pinů, proměnných a přerušení do souboru za běhu simulace. Soubor může být vykreslen do grafu pomocí programu qtkwave. Tento systém umožňuje následnou analýzu výstupu. Přeložený soubor s firmwarem MCU může obsahovat informace o tom, jaké signály se mají zaznamenávat. [2]

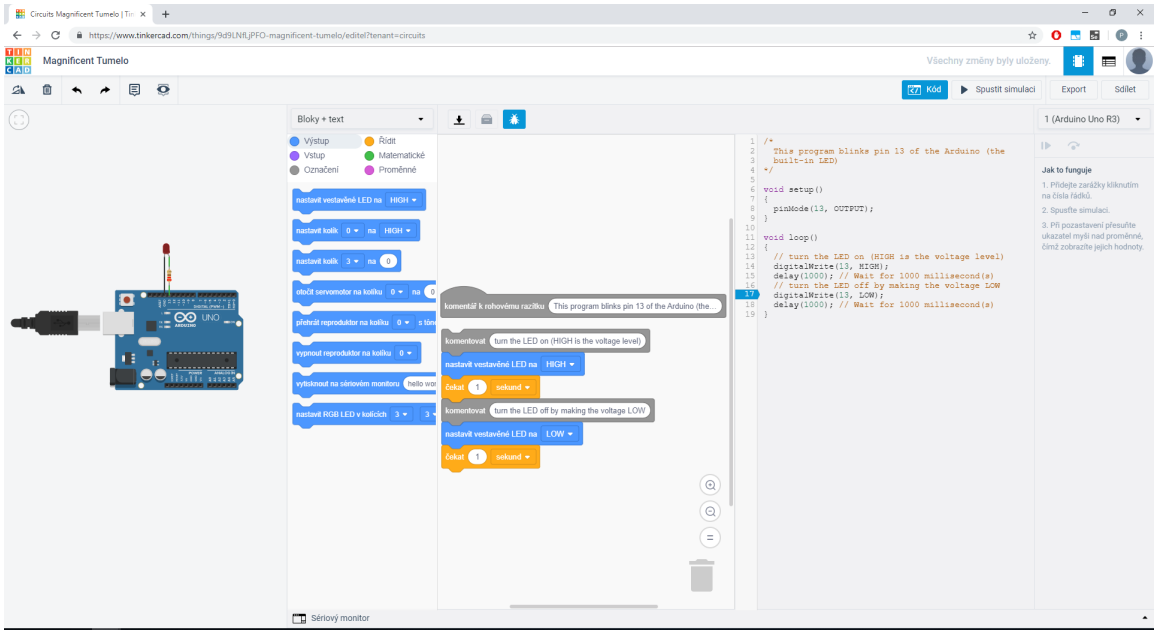
2.8 SimulAVR

SimulAVR je simulátor mikrokontrolérů architektury AVR. SimulAVR lze použít jako knihovnu nebo samostatně. Neimplementuje žádné okolí MCU, umí však zaznemenávat výstup ve formátu VCD, který lze zobrazit jako graf například programem gtkwave.

Simulátor SimulAVR se zabývá simulací MCU řad ATmega a ATtiny. Simulátor může být použit samotný nebo se na něj jde připojit nástrojem avr-gdb. Existují pro něj také grafická rozhraní napsaná v pythonu a TCL. [1]

2.9 Tinkercad Circuits

Tinkercad je webová služba obsahující několik nástrojů od společnosti Autodesk Inc pro výukové účely. Jedním z nástrojů je Tinkercad circuits, simulátor Arduina s jednoduchým editorem elektrických obvodů. Tinkercad circuits obsahuje editor kódu Arduina, kód lze editovat v normální textové podobě, ale také ve formě skládání grafických bloků. Editování kódu pomocí grafických bloků je vidět na obrázku 2.10.



Obrázek 2.10: Tinkercad

Kapitola 3

Návrh simulátoru

Návrh spočívá v návrhu uživatelského rozhraní, simulace elektrických obvodů, která bude sloužit jako okolí MCU, a samotného simulátoru MCU. Uživatelské rozhraní bude zobrazovat stav okolí (el. obvodů), ale hlavně zobrazovat a editovat vnitřní stav simulovaného mikrokontroléru. Tedy obsah pamětí a vykonávaný program.

Program bude obsahovat simulaci MCU, která bude řešena jako diskrétní simulace, a simulaci el. obvodů, což je spojitá simulace. Simulátor tedy bude řešit simulaci jako kombinovanou.

3.1 Návrh jádra modulárního simulátoru

Simulátor má být modulární. V ideálním případě by mělo být možné naimplementovat jakýkoli mikrokontrolér s jakoukoli architekturou. Dosáhnout tohoto by bylo praxi příliš náročné a to nejen v samotné simulaci, ale také v rámci uživatelského rozhraní. Proto se návrh omezuje na Harvardskou architekturu. Harvardská architektura byla zvolena proto, že ji používají populární MCU jako jsou PIC nebo AVR. A také z toho důvodu, že je zřejmě častěji používaná[10].

Jádro simulátoru musí být postaveno tak, aby se dal snadno přidat nový MCU. Bylo proto navrženo, aby se jevilo jako framework pro programátora MCU. Jádro simulátoru poskytuje základní komponenty mikrokontroléru, jako jsou paměti, periferie a centrální procesorovou jednotku (CPU) neboli jádro mikroprocesoru. Mezi další komponenty patří také el. obvody. Řeší komunikaci mezi těmito komponentami, ale také propojení s uživatelským rozhraním.

Pro splnění modularity bylo navrženo rozhraní, podle kterého budou jednotlivé implementace mikrokontrolérů komunikovat. Bylo navrženo několik tříd, které budou tvořit toto rozhraní. Na obrázku 3.1 třída *CPU* je určena jako rodičovská třída pro všechny typy MCU. Představuje logické a výpočetní jádro MCU. Třída *Memory* představuje paměť s určitou délkou a šířkou. MCU může mít těchto pamětí libovolné množství. Například pro registry, operační paměť atd. Každé MCU má nějaký počet pinů, ty reprezentuje třída *Pin*. Na každém pinu je vždy jedna nebo více periferií, které poskytují výstup nebo zpracovávají vstup. Například GPIO (general–purpose input/output) nebo ADC (analogově digitální převodník). Periferie reprezentuje třída *Periphery*. Třída *Pin* je odpovědná za propojení periferií MCU s okolím (el. obvody). Pokud má pin více periferií, zajišťuje přepínání mezi nimi. Jelikož přepínání mezi periferiemi je ovládáno různě na různých MCU, třída *Pin* je také určená ke zdědění. Stejně tak třída *Periphery* se očekává být zděděná.

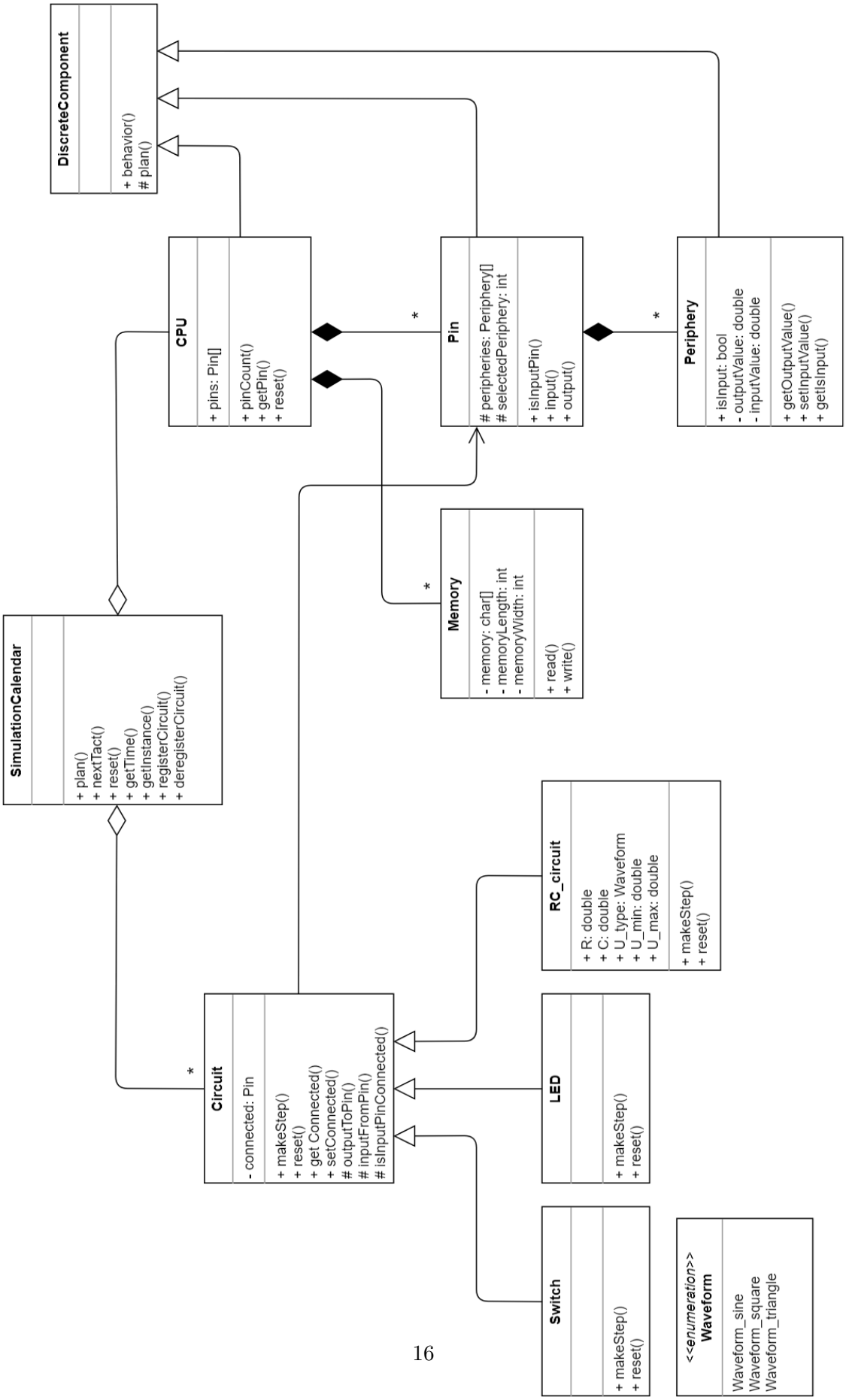
Třída *SimulationCalendar* řeší řízení kombinované simulace. Kalendář ovládá simulační čas. Spojité komponenty (třídy typu *Circuit*) řídí jako spojitou simulaci. Diskrétním komponentám (třídy typu *DiscreteComponent*) poskytuje možnost se naplánovat na určitý čas. Simulace MCU se řeší na vyšší abstrakci, kdy se počítá, že každá diskrétní komponenta bude provádět akce pouze ve chvíli, kdy přijde signál z generátoru hodin. Typicky při nástupné hraně signálu. Proto kalendář bude poskytovat plánování na takty hodin místo času.

Jelikož Kalendář řídí celou simulaci a poskytuje komponentám možnost plánování, tak je potřeba, aby k němu všechny komponenty měli stále přístup. Dostupnost byla zajištěna pomocí návrhového vzoru singleton[9].

Třída *DiscreteComponent* je rodičovská třída všech diskrétních komponent simulace, které se potřebují plánovat. *DiscreteComponent* má metodu *plan* pro usnadnění naplánování události v kalendáři diskrétním komponentám. *DiscreteComponent* má také metodu *behavior*, která slouží k obdržení události.

Pro elektrické obvody je k dispozici společná rodičovská třída *Circuit*. Ta zajišťuje propojení s pinem MCU. Také definuje metodu *step*, pomocí které funguje spojitá simulace. Kalendář ji volá vždy, když se provede krok v čase, a předá obvodu délku kroku v argumentu, aby obvod mohl spočítat svůj stav v novém čase. Hodinový takt MCU se uvažuje konstantní a délka kroku spojité simulace dělitelem taktu MCU. Tímto se dosáhne toho, že kalendář nemusí provádět dokračování, které se běžně u kombinované simulace provádí, a šetří se tím výpočetní výkon.

Jádro simulátoru je důležitou částí programu. Poskytuje softwarový framework konkrétním implementacím mikrokontrolérů. Poskytuje rozhraní pro zobrazování a manipulaci stavu simulace uživatelskému rozhraní. Zajišťuje propojení s el. obvody a synchronizaci všech komponent se simulačním časem.

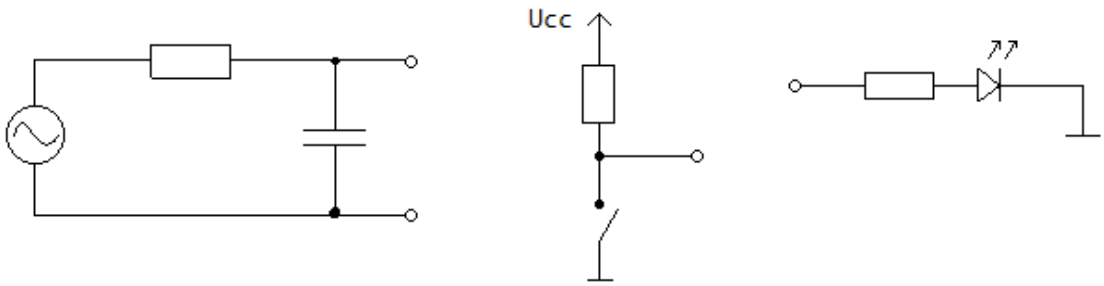


Obrázek 3.1: diagram tříd jádra simulátoru

3.2 Návrh simulace elektrických obvodů tvořících okolí mikrokontroléru

Elektrické obvody tvoří okolí mikrokontroléru. Byly navrženy tři obvody tak, aby pokryly různé aspekty simulace. Jedná se o způsob použití těchto obvodů v simulaci. Jeden obvod, který by se používal v kombinaci s pinem pracujícím ve výstupním režimu, aby MCU měl možnost produkovat nějaký výstup. Druhý se vstupním pinem, který by sám generoval data pro zpracování mikrokontrolérem. A obvod, který by umožňoval uživateli kontrolovat vstupní data přímo.

Tři navržené obvody jsou na obrázku 3.2. Jedná se o RC článek s funkčním generátorem na vstupu. Pin je možné připojit k výstupu článku. Článek bude mít nastavitelné hodnoty rezistoru, kondenzátoru a funkčního generátoru. Druhý obvod je spínač, pro manuální zadávání vstupu uživatelem. Třetím obvodem je LED. Simulace obvodů je zjednodušená. Při simulaci obvodů se uvažuje s ideálními součástkami.



Obrázek 3.2: navržené elektrické obvody tvořící okolí mikrokontroléru

LED díky zjednodušení na ideální prvky se rozsvěcuje a zhasíná okamžitě. Chová se tedy diskrétně. Klasická červená dioda začíná svítit od napětí 1,8 V. Pokud bude na pinu toto napětí nebo vyšší, dioda bude svítit. S proudem, který by řídil normálně intenzitu LED, se nepočítá, dioda má pouze dva stavy, svítí nebo nesvítí.

Spínač se jako LED chová také diskrétně. Je sepnutý nebo není. Spínač je v konfiguraci s *pull-up* rezistorem. V případě, že je v sepnutém stavu, je výstup připojen spínačem na nulový potenciál. Pokud je rozepnutý, je přes odpor připojen k napájecímu napětí. Spínač je ideální a tak nemá žádné zpoždění ani zákmity.

RC článek je spojitý prvek. Na straně vstupu má připojený funkční generátor. Ten je možné nastavit na požadovaný průběh signálu (funkci). Byly zvoleny tři často používané průběhy. Sinusový, obdélníkový a trojúhelníkový. Na generátoru bude také možné nastavit amplitudu signálu a frekvenci.

Při návrhu se vycházelo z rovnice 3.1. Rovnice vyjadřuje závislost napětí a proudu na kondenzátoru v derivační formě. Případný pin MCU připojený na výstup článku bude omezen na vstupní konfiguraci. Pin ve vstupním režimu má vysokou impedanci a jeho proud bude velmi nízký. Z praktických důvodů bude zanedbán. Po zanedbání pinu se kondenzátor nalézá v sériovém zapojení s rezistorem. Rovnice 3.2 vychází z Ohmova zákona a popisuje proud na rezistoru. Stejný proud na rezistoru prochází i kondenzátorem. Rovnice 3.2 byla dosazena do rovnice 3.1 a po úpravě vznikla 3.3. Rovnice 3.4 je druhý Kirchhoffův zákon s vyjádřeným napětím na rezistoru a byla dosazena do 3.3. Po úpravě je výsledkem rovnice 3.5. Rovnice 3.5 je již ve vhodném tvaru a lze ji použít k vytvoření simulace.

$$i(t) = C * \frac{dV_c(t)}{dt} \tag{3.1}$$

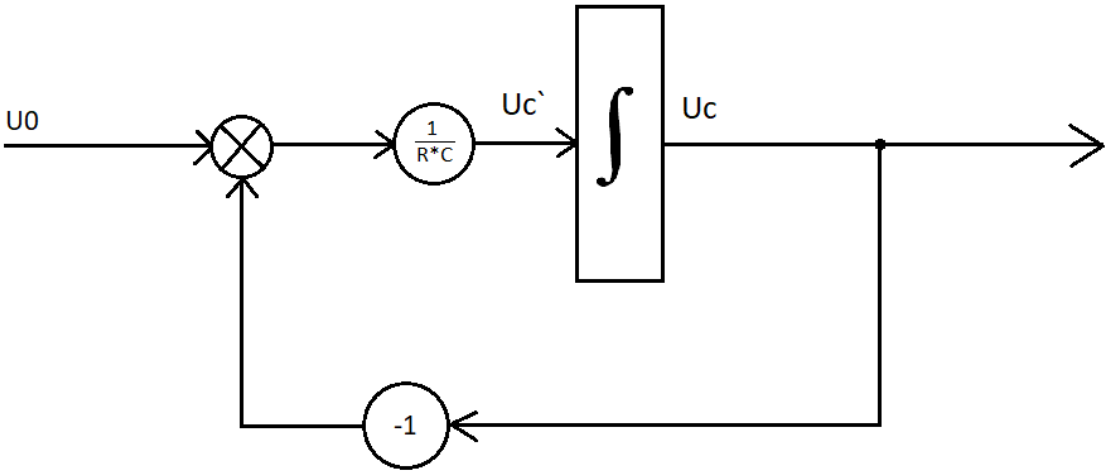
$$i(t) = \frac{U_R}{R} \tag{3.2}$$

$$U_R = R * C * \frac{dV_c(t)}{dt} \tag{3.3}$$

$$U_R = U_0 - U_C \tag{3.4}$$

$$U'_C = \frac{U_0 - U_C}{R * C} \tag{3.5}$$

Ze sestavené rovnice 3.5 byl vytvořen blokový model na obrázku 3.3. Model obsahuje jeden vstup, vstupní napětí. Jeden výstup, napětí na kondenzátoru, které je rovné výstupnímu napětí na celém článku. Uvnitř modelu je také jeden integrátor a několik aritmetických bloků.



Obrázek 3.3: graficky znázorněná simulace RC článku

Implementace modelu bude pomocí metody Runge-Kutta čtvrtého řádu. Metoda poskytuje dobrou přesnost s relativně nízkými výpočetními nároky. Je tak vhodná pro použití v tomto simulátoru.

3.3 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní spočívá ve stanovení požadavků na obsah rozhraní a vytvoření prototypu dle těchto požadavků. Nakonec také výběr použitého frameworku.

3.3.1 Požadavky

Požadavky se stanovily tak, aby byly v souladu se zadáním práce a aby byl simulátor dostatečně intuitivní, a byl tak použitelný pro nezaučené uživatele, kteří jsou zvyklí používat jiný simulátor.

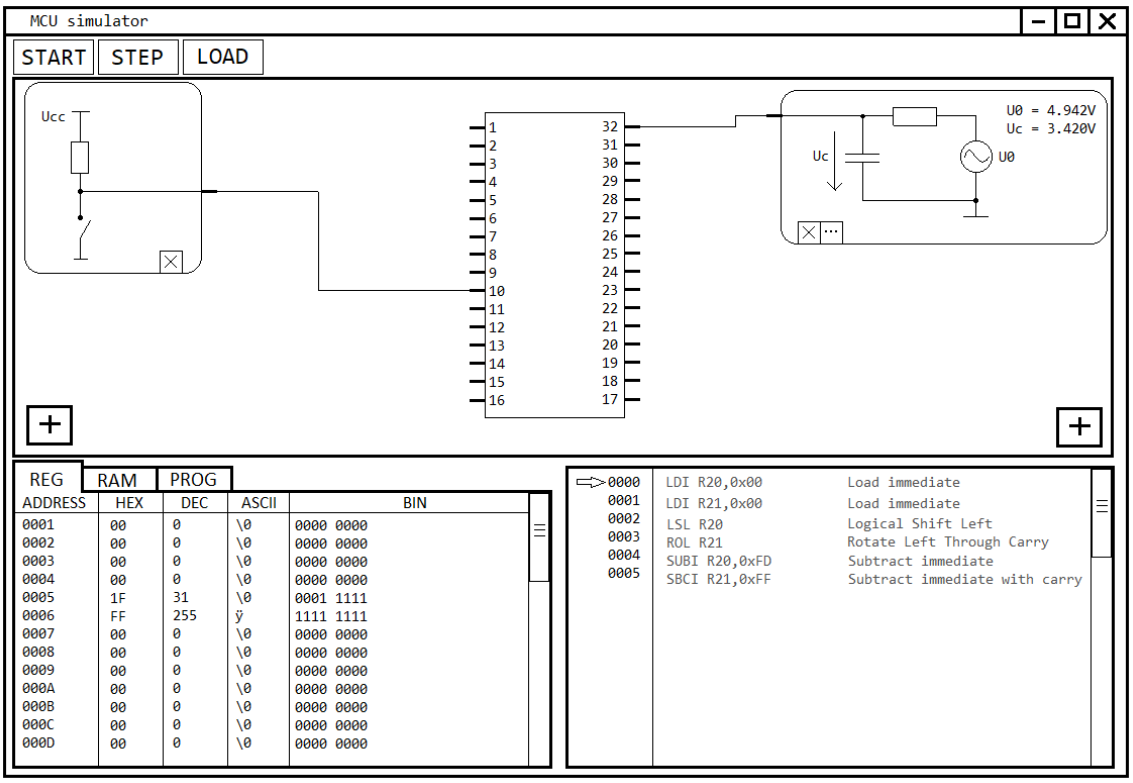
Požadavky na uživatelské rozhraní jsou následující. Rozhraní musí umožňovat připojování všech dostupných druhů el. obvodů k pinům MCU a zobrazení vnitřního stavu MCU ve formě pamětí, právě vykonávaného programu a aktuální pozice v něm.

Aplikace je pro výukové účely a je cílena na studenty oboru informatiky. Předpokládá se tedy, že uživatelé budou mít alespoň minimální znalost elektrotechniky a jsou seznámeni s principem mikrokontrolérů a počítačů obecně.

3.3.2 Prototyp

Na základě požadavků bylo navrhnuto uživatelské rozhraní ve formě prototypu. Prototyp byl vytvořen jako grafická vizualizace návrhu.

Výsledný prototyp je na obrázku 3.4. Aplikace bude zobrazena v jednom okně. V horní sekci okna je menu s ovládacími tlačítky na spuštění simulace, krokování programu a naprogramování MCU programem ze souboru. Pod menu je grafické zobrazení simulovaného mikrokontroléru a jeho okolí (el. obvody). Dolní část okna je rozdělena na zobrazení obsahu pamětí nalevo a zobrazení programu napravo.



Obrázek 3.4: Prototyp uživatelského rozhraní

S ohledem na znalosti cílové skupiny uživatelů, mikrokontrolér i s obvody jsou zobrazeny schématicky. Mikrokontrolér má své piny očíslované, pro rozlišení.

Obvody se řadí pod sebe po stranách grafické plochy v pořadí, v jakém jsou přidány uživatelem. Každý obvod má přípojné místo pro připojení k MCU. Pro propojení obvodu s MCU musí uživatel kliknout na přípojné místo na daném obvodu a poté vybrat požadovaný pin na MCU také kliknutím. Obvody je možné přidávat tlačítky na spodní části zobrazovací plochy. Tlačítka mají ikonu plus, spolu s umístěním by se měla zvýšit jejich intuitivnost. Odstraňovaní obvodů je řešeno tlačítky s ikonou křížku umístěnými vždy u příslušného obvodu. RC článek je jediný obvod, který má možnost editace jeho parametrů, má tedy kromě tlačítka na odstranění také tlačítko na editaci, které otevře nové okno s jednoduchým formulářem.

MCU může mít libovolný počet pamětí, a tak jsou paměti řazeny do záložek. Pro každou paměť se bude zobrazovat obsah jako seznam dvojic adresa a hodnota. Hodnota může být v různých programech různě interpretována, a proto se bude zobrazovat v několika často používaných interpretacích, v hexadecimální soustavě, dekadické, ASCII dekódovaná a v binární podobě.

Program je zobrazen pomocí disasembleru. Aktuální pozice v programu je zobrazena pomocí šipky ukazující na instrukci, která je na adrese PC registru(program counter register).

Prototyp zobrazuje hlavní okno aplikace. Prototyp nepokrývá formuláře a upozornění, které se zobrazují v dialogových oknech.

3.3.3 Qt framework

Uživatelské rozhraní je grafické a využívá standardní ovládací prvky. Proto je vhodné použít některý dostupný framework. Byl zvolen framework QT.

QT je multiplatformní framework vytvořený pro jazyk C++. Hlavním faktorem pro volbu frameworku byla rozsáhlá podpora a komunita.

Zapojení QT frameworku do aplikace znamená použití QT systému signálů a slotů. Třídy aplikační logiky budou používat signály k zobrazování dat. Aby třída mohla používat signály, je potřeba ji zdědit od *QObject* a označit makrem *Q_OBJECT*. Třídy implementující signály jsou *Circuit*, *Memory* a *CPU*. Třída *CPU* již dědí od jiné třídy, tento problém řeší vícenásobná dědičnost, která je podporovaná v C++.

3.4 Návrh mikrokontroléru ATmega382p

Do simulátoru byl přidán jeden mikrokontrolér. Zvolen byl mikrokontrolér *ATmega382p* od společnosti Microchip Technology. Mikrokontrolér byl vytvořen pomocí knihovny *simavr*. V kapitole je vybraný mikrokontrolér představen a je řešen způsob propojení knihovny *simavr* se simulátorem.

3.4.1 Mikrokontrolér ATmega382p

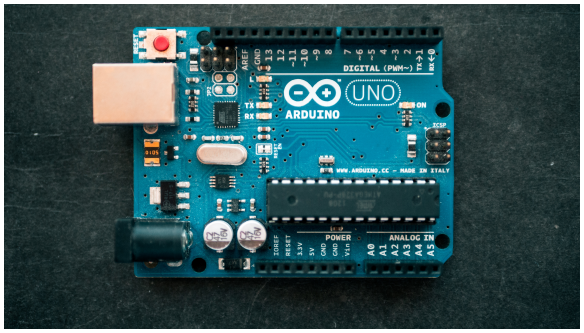
ATmega382p je populární mikrokontrolér od společnosti Microchip Technology.

ATmega382p je osmibitový mikrokontrolér s *AVR* architekturou. Může bezpečně pracovat na frekvenci 20 MHz. Jelikož je založený na RISC (reduced instruction set computer) architektuře, může při této frekvenci dosáhnout až 20 MIPS (milion instructions per second). Další parametry jsou vypsány v tabulce 3.1. [6]

Typ programové paměti	Flash
Velikost programové paměti (KB)	32
SRAM velikost (B)	2048
EEPROM/HEF (B)	1024
Periferie digitální komunikace	1-UART, 2-SPI, 1-I2C
Další periferie	1 Input Capture, 1 CCP, 6 PWM
Časovače	2 x 8-bit, 1 x 16-bit
Počet komparátorů	1
Teplotní rozsah (°C)	-40 to 85
rozsah napájecího napětí (V)	1.8 - 5.5
Počet pinů	32

Tabulka 3.1: parametry ATmega382p [6]

ATmega382p byl zvolen převážně z důvodu jeho vysoké popularity. *ATmega382p* je nejznámější právě díky platformě *Arduino*. Mikrokontrolér je osazen na deskách řady *Arduino Uno*. *Arduino Uno* je na obrázku 3.5.



Obrázek 3.5: Arduino Uno[1]

3.4.2 Propojení s knihovnou simavr

Pro implementaci MCU byla zvolena knihovna *Simavr*. *Simavr* je neoficiální opensource simulátor, umožňující simulaci mnoha mikroprocesorů s architekturou *AVR* včetně *ATmega382p*. Knihovna byla propojena se simulátorem.

Propojení simulátoru s knihovnou bude realizované tak, že bude použito stejné rozhraní, které bylo navrhnuto v kapitole *Návrh jádra modulárního simulátoru*. Bude vytvořena třída *ATmega382p*, která bude dědit od *CPU*. Třída bude inicializovat MCU z knihovny, bude zařizovat krokování a další logiku pro funkci. Třída *avrDisassembler* dědí od *Disassembler*. Podle rozhraní poskytuje překlad strojového kódu instrukční sady *AVR* na assembler. Na překlad bude použit vývojový nástroj pro *AVR* mikroprocesory *avr-objdump*. Třída *AVRPeriphery* bude zajišťovat komunikaci pinů MCU s knihovnou. Třída *AVRPin*, dědící od *Pin* by podle návrhu řešila přepínání periferií, ale *simavr* toto už řeší interně. *AVRPin* tak bude mít pouze jednu periferii.

[1]Zdroj: https://unsplash.com/photos/fZB51omnY_Y

MCU pro některé funkce vyžaduje stanovené napájecí napětí. Například pro převodníky a komparátory. Bylo zvoleno napájecí napětí 5,0 V. Stejné napětí jako na deskách *Arduino Uno*.

Simavr knihovna je pod licencí GNU General Public License v3.0. Podle požadavků licence knihovny, vytvářený simulátor bude muset být také pod stejnou licencí.

Navržené třídy odpovídají návrhovému vzoru adaptér, převádí rozhraní *simavr* na rozhraní simulátoru. *avrDisassembler* využívá nástroj *avr-objdump* a poskytuje rozhraní k získaným datům.

Kapitola 4

Implementace simulátoru

Tato kapitola se zabývá implementací simulátoru a řešením problémů, které při ní vznikly. Simulátor byl implementován podle návrhu s menšími změnami v uživatelském rozhraní. Simulátor je implementovaný v jazyce C++. Jako první je popsáno řešení implementace některých tříd jádra simulátoru. Poté je popsáno řešení implementace mikrokontroléru ATmega328p. Dále pak implementace elektrických obvodů podle navržených modelů.

4.1 Implementace jádra simulátoru

Oproti návrhu byla přidána pomocná třída *EventPriorityQueue* ke kalendáři událostí. Vazby "1 ku N"byly řešené dynamickými poli *std::vector* ze standardní knihovny C++.

4.1.1 Kalendář událostí

Události se plánují do kalendáře vždy na určitý takt hodin. A to většinou hned na příští takt, popřípadě několik málo do budoucnosti. Nepředpokládá se, že toto číslo bude příliš vysoké. Proto lze vytvořit optimalizovanou implementaci oproti verzi podporující libovolný čas. Události jsou vkládány do front, každá fronta obsahuje události naplánované na jeden takt. Kalendář má pole těchto front pro několik taktů dopředu:

```
std::vector<EventPriorityQueue *> events;
```

EventPriorityQueue je pomocná třída, která vznikla za účelem spravovat priority událostí ve frontě. Implementuje všechny potřebné funkce. Vložení s prioritou a odebrání nejprioritnější události. *EventPriorityQueue* ja implementována pomocí statického pole dynamických polí:

```
vector<DiscreteComponent *> eventQueue[PRIORITY_LEVELS];
```

Index do *eventQueue* určuje prioritu a konstanta *PRIORITY_LEVELS* počet stupňů priorit. Hodnota *PRIORITY_LEVELS* byla zvolena tři: nízká, vysoká a střední priorita, jelikož je předpokládáno, že většina MCU bude vyžadovat každý takt provést s nejvyšší prioritou, tedy jako první, zpracování vstupů. V druhé fázi vykonat zpracování instrukce a nakonec zpracování výstupů. Tři úrovně by tedy měly stačit, případně lze konstantu bez problému navýšit. Snížením by se však riskovalo rozbití implementací existujících MCU.

4.1.2 Rodičovská třída obvodů *Circuit*

Třída *Circuit* poskytuje bázovou třídu elektrickým obvodům. Poskytuje také *protected* metody pro usnadnění práce s připojeným pinem MCU. Jedná se o metody *void outputToPin(voltage value)*, *voltage inputFromPin()* a *bool isInputPinConnected()*. Tyto metody mají v návrhu nedefinované chování ve stavu, kdy není obvod spojen s žádným pinem, obvody mají tedy možnost použít *getConnected* metodu a otestovat návratovou hodnotu na *nullptr*. Implementace metod byla provedena tak, aby obvody se nemusely starat o to, zda jsou připojené nebo ne.

Žádná metoda nesmí selhat, pokud není připojený pin. A byly naprogramovány tak, aby vracely použitelné hodnoty. Metoda *outputToPin* pouze předává třídě *Pin* vstupní hodnotu, v případě, že pin není připojen, nedělá nic.

Metoda *inputFromPin* vrací výstupní napětí pinu, pokud pin není připojen, vrací nulové napětí. Nulové napětí má sice význam připojení výstupu obvodu na zem, což není korektní, ale u mnoha obvodů to nevadí a ostatní, u kterých by to způsobovalo nežádoucí chování, mohou stále využít testování, zda je pin připojen.

Metoda *isInputPinConnected* vrací zda je připojený pin nastaven jako vstupní nebo výstupní. V případě že pin není připojen, udává, že je připojený vstupní pin. Pin ve vstupním stavu má vysokou impedanci a jelikož obvody zanedbávají malý proud vstupující dovnitř, je to stejné, jako by pin připojen byl.

Zde je implementace zmíněných metod:

```
void Circuit::outputToPin(voltage value)
{
    if(connected != nullptr)
        connected->input(value);
}

voltage Circuit::inputFromPin()
{
    if(connected != nullptr)
        return connected->output();
    return 0;
}

bool Circuit::isInputPinConnected()
{
    if(connected != nullptr)
        return connected->isInputPin();
    return true;
}
```

4.2 Implementace mikrokontroléru ATmega328p

4.2.1 Třída ATmega328p

Třída slouží jako adaptér ve stejnojmenném návrhovém vzoru. Adaptuje knihovnu *simavr* na rozhraní použité v simulátoru.

Inicializace MCU z knihovny se provádí v metodě *makeNewAvrFromProgramFile*. Vytváří se zde struktura *avr_t*, která je hlavní strukturou *simavr*. Obsahuje všechny informace a aktuální stav MCU. Metoda také při vytváření nahrává program do MCU. Kód metody byl převzat ze zdrojových souborů *simavr* a následně upraven.

Metoda *Load* provádí načítání nového programu. Používá přitom metodu *makeNewAvrFromProgramFile*.

Metoda *reset* uvádí MCU do výchozího stavu. *simavr* neposkytuje žádný jednoduchý způsob jak provést reset MCU. Musí proto vytvořit vždy nové MCU a naprogramovat jej. Stejně jako metoda *Load*, používá k tomu metodu *makeNewAvrFromProgramFile*.

Metoda *behavior* vykonává jeden takt procesoru. K vykonání jedné instrukce se používá knihovní funkce *avr_run*. Také aktualizuje programový čítač podle čítače ze struktury *avr_t*.

getMemoryList předává paměť z proměnné *sram*. Paměť obsahuje obecné registry, i/o registry a SRAM. Tyto paměti jsou v *AVR* architektuře ve stejném adresním prostoru, a je tak běžné je udávat jako jednu paměť. *simavr* je také realizuje jako jednu paměť. *simavr* si paměť sám alokuje, což je v rozporu s rozhraním simulátoru. Problém byl vyřešen výměnou ukazatele na paměť ve struktuře *avr_t* za ukazatel na paměť třídy *Memory* používané simulátorem. Knihovna tak pracuje přímo s pamětí třídy *Memory*. To způsobuje ale další problém. Nelze totiž sledovat, která paměť byla změněna. Proto se po vykonání každé instrukce musí označit celá paměť za pozměněnou, aby se mohla překreslit v uživatelském rozhraní. K tomu byla třídě *Memory* přidána pomocná funkce *allChanged*.

makePins metoda slouží k vytvoření pinů (třída *AVRPin*). Každému pinu je přiděleno jméno a číslo pinu v daném portu. Metoda *setPinIrqs* pak předává pinům *irq* (interrupt request). Knihovna *simavr* používá *irq* ke komunikaci různých komponent. Piny používají *irq* ke komunikaci hodnot na pinech.

Signatura výsledné třídy je následující:

```
class ATmega382p : public CPU
{
public:
    ATmega382p(QString programFilename);
    void getMemoryList(std::vector<Memory *> *memories) override;
    void behavior() override;
    void reset() override;
    void load(QString programFilename) override;

private:
    avr_t *avr;
    Memory *sram;
    QString programFilename;

    avr_t *makeNewAvrFromProgramFile(QString programFilename);
    void makePins();
    void setPinIrqs();
};
```

4.2.2 Periferie

Vstupy a výstupy MCU propojují s knihovnou *simavr* dvě třídy, *AVRPin* a *AVRPeriphery*. Třída *AVRPin* dědí od třídy *Pin*. Třída *AVRPeriphery* dědí od třídy *Periphery*.

AVRPeriphery komunikuje s knihovnou *simavr* pomocí systému *irq* (interrupt request). V metodě *behavior*, pokud operuje ve vstupním režimu, vyvolá *irq* se vstupní hodnotou. Pro digitální vstup knihovna přijímá logickou 1 nebo 0. Podle technické specifikace (*datasheet*)[7] výrobce čipu je při napájecím napětí 5 V považováno jako logická 0 napětí do 0,3 V. A jako logická 1 napětí vyšší jak 0,6 V. Od 0,3 V do 0,6 V je šedé, nedefinované, pásmo. Naimplementován byl ideální případ, kdy ke změně z logické 0 na 1 dojde v polovině šedé zóny, tedy při 0,45 V.

Pro výstup z MCU byla zaregistrována *callback* funkce na *irq* patřičného pinu. *simavr* poskytuje také možnost předat vlastní parametr *callback* funkci. Předává se instance *AVRPeriphery*, aby funkce mohla předat nové hodnoty výstupu. Pro správnou funkci byla ještě potřeba detekce režimu pinu, zda je ve vstupním nebo výstupním režimu. Ta byla zajištěna také *callback* funkcí příslušného *irq*.

Použité *irq* odpovídá logickému portu MCU. Ostatní periferie používají své vlastní *irq*, to znemožňuje jednoduché použití periferií. Při implementaci se periferie, kromě periferie *GPIO* (general–purpose input/output), nepodařilo zprovoznit.

4.2.3 Disassembler

Překlad ze strojového kódu na assembler řeší třída *avrDisassembler*. Překlad se bude provádět pomocí nástroje *avr-objdump*.

V metodě *load* třídy *avrDisassembler* se spouští nástroj *avr-objdump*. Výstup z nástroje je zpracován a uložen v objektu. Ke zpracovaným datům pak objekt umožňuje přístup přes své rozhraní. *avr-dump* se spouští jako příkaz s několika argumenty, aby byl výstup snadněji zpracovatelný. Jedná se o argument *"–wide"* a *–prefix-addresses*. Argument *"–wide"* sděluje nástroji, že může překročit hranici 80 znaků na řádek. Nehrozí, že by instrukce zabrala více jak jeden řádek, a lze tak výstup zpracovávat po jednotlivých řádcích. Argument *"–prefix-addresses"* vypisuje adresu instrukce na začátku každého řádku. To opět ulehčuje zpracovávání, adresa by se musela jinak dopočítávat. Data se plní do privátních členů objektu definovaných takto:

```
std::vector<QString> disassembledProgram;
std::vector<address> instructionAddresses;
```

Výsledný assembler je uložen po řádcích do *disassembledProgram*. Každý řádek je jedna instrukce a každá instrukce má adresu. Adresy instrukcí jsou uloženy v *instructionAddress* také po řádcích.

4.3 Implementace elektrických obvodů

Implementované elektrické obvody jsou LED, spínač a RC článek. Všechny obvody dědí od třídy *Circuit*. Všechny jsou, stejně jako *Circuit*, *Q_OBJECT*, aby mohly používat signály z frameworku QT.

4.3.1 Implementace LED

LED uchovává svůj vnitřní stav v proměnné *lightOn*. Ke stavu se přistupuje pomocí metody *isLightOn*. Metoda *makeStep* nejprve ověří, zda je připojena na pin ve výstupním režimu, pokud ne, dioda nesvítí. Metoda poté čte napětí na připojeném pinu MCU. Pokud napětí na pinu přesáhne prahové napětí diody, dioda se rozsvítí. Signatura třídy je následující:

```
class LED : public Circuit
{
    Q_OBJECT

public:
    LED();
    void makeStep(deltaTime stepSize) override;
    void reset() override;
    bool isLightOn() const;

private:
    bool lightOn;
};
```

4.3.2 Implementace spínače

Spínač má metodu *switchState*, která přepíná mezi sepnutým a rozepnutým stavem. Metodu volá uživatelské rozhraní, když se uživatel snaží přepnout spínač. Stav spínače určuje proměnná *turnedOn*. Metoda *makeStep* posílá napětí na připojený pin MCU podle svého stavu. Také kontroluje, zda nedošlo ke zkratu. Pokud ke zkratu dojde, je nastavena proměnná *shortOccured*. Uživatelské rozhraní zjišťuje, jestli se vyskytl zkrat pomocí metody *isShortOccured*. Signatura třídy je následující:

```
class Switch : public Circuit
{
    Q_OBJECT

public:
    Switch();
    void makeStep(deltaTime stepSize) override;
    void reset() override;
    void switchState();
    bool isTurnedOn() const;
    bool isShortOccured() const;

private:
    bool turnedOn;
    bool shortOccured;
};
```

4.3.3 Implementace RC článku

RC článek je řešen ve třídě *RCCircuit*. Model RC článku obsahuje integrátor a několik aritmetických operací. Třída má nastavitelné parametry rezistoru, kondenzátoru, amplitudy funkčního generátoru a jeho zvolené funkce na funkčním generátoru.

Integrátor byl implementován pomocí vzorce metody Runge–Kutta čtvrtého řádu‘[3]. Aritmetická část modelu vypočítává derivaci výstupního napětí U'_C. Výstupní napětí se počítá ve funkci *dynamic*. Počítá se ze vstupního napětí *sourceVoltage*, odporu rezistoru *rezistance*, kapacity kondenzátoru *capacitance* a výstupního napětí *outputVoltage*. Toto je princip implementace výpočtu:

```
double outputDerivative = (sourceVoltage - outputVoltage) / (rezistance * capacitan
```

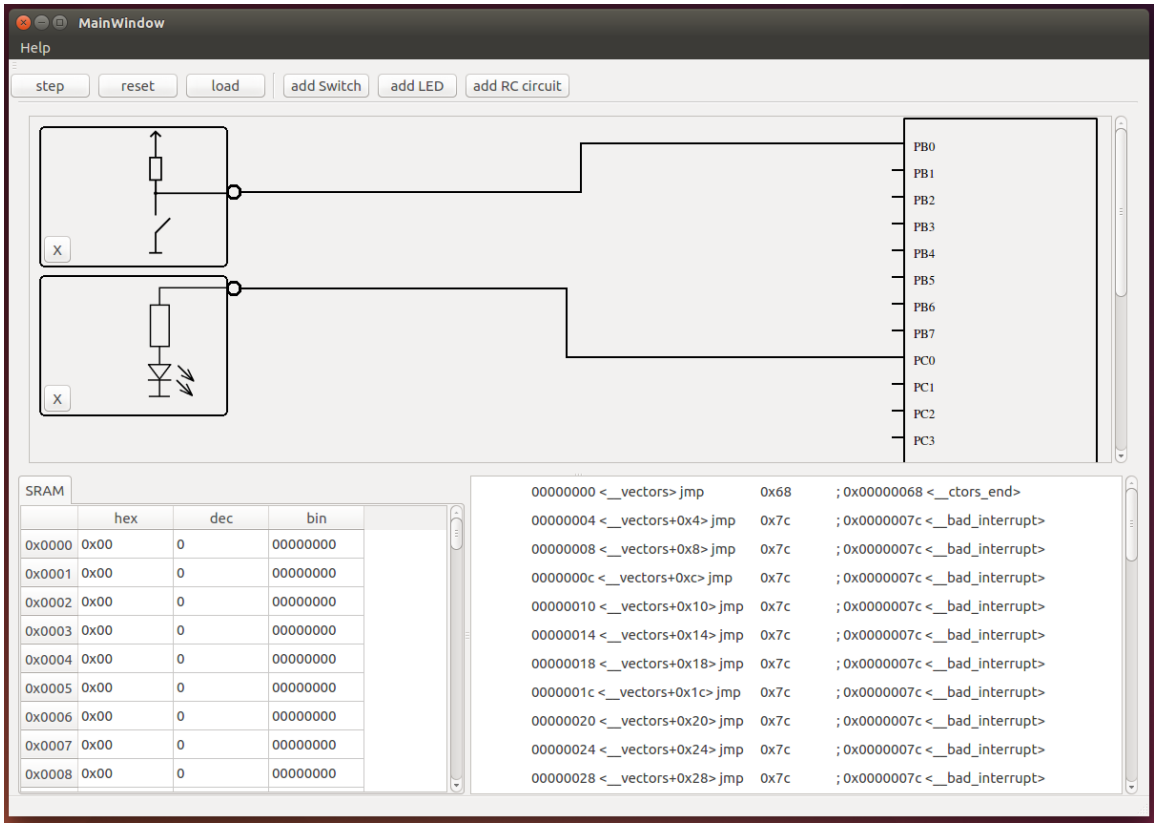
Vstupní napětí článku se počítá v metodě *calculateInput*. Vstupní napětí je v každém kroku přepočítáno. Počítá se z amplitudy *sourceAmplitude*, frekvence *frequency*, zvolené funkce funkčního generátoru *sourceFunction* a z aktuálního simulačního času *time*. Výsledná implementace vypadá takto:

```
voltage RCCircuit::calculateInput(Time time)
{
    switch (sourceFunction) {
        case Waveform_sine:
            return sourceAmplitude * sin(360 * time * frequency * (M_PI / 180));
        case Waveform_square:
        {
            double period = 1 / frequency;
            return fmod(time, period) < period / 2 ? sourceAmplitude : -sourceAmplitude;
        }
        case Waveform_triangle:
        {
            double period = 1 / frequency;
            double periodPossition = fmod(time + period / 4, period) / period;
            if(periodPossition < 0.5){
                return 2 * sourceAmplitude * 2 * periodPossition - sourceAmplitude;
            }else{
                return 2 * sourceAmplitude - 2 * sourceAmplitude * 2 * (periodPossition -
            }
        }
    }
    return 0;
}
```

4.4 Implementace uživatelského rozhraní

Uživatelské rozhraní bylo implementováno pomocí QT frameworku. Uživatelské rozhraní využívá rozhraní jádra simulátoru z podkapitoly *Návrh jádra modulárního simulátoru*.

Výsledné uživatelské rozhraní je na obrázku 4.1. Oproti návrhu byla udělána jedna změna. Elektrické obvody se neskládají na obě strany MCU, ale pouze doleva. Změna byla provedena, protože obvod položený na jednu stranu by nebylo možné jednoduše připojit na pin na druhé straně.



Obrázek 4.1: grafické rozhraní výsledného simulátoru

Uživatelské rozhraní bylo vyvíjeno podle návrhového vzoru MVC (model–view–controller). Třídy z jádra simulátoru slouží jako modely. Ke každému obvodu byl vytvořen view, který obvod zobrazuje a řeší interakci s uživatelem. Tyto třídy jsou potomky třídy *QWidget*, která je bázovou třídou všech vizuálních komponent frameworku QT.

4.5 Ověření funkčnosti

Funkčnost výsledného simulátoru byla ověřena na jednoduchých příkladech. Příklady byly vloženy do složky *examples* v kořenovém adresáři projektu. První příklad je jednoduché blikání LED na otestování výstupu. Druhý příklad rozsvěcí LED v reakci na sepnutí spínače. Tím se testuje vstup MCU.

Kapitola 5

Závěr

Cílem této práce bylo navrhnout a implementovat simulátor mikrokontrolérů. Simulátor měl být modulární, disponovat grafickým uživatelským rozhraním, obsahovat několik elektrických obvodů a mít jeden implementovaný mikrokontrolér. Simulátor byl vytvořen a cíl práce byl splněn.

V rámci práce byla prostudována problematika modelování a simulace mikrokontrolérů a elektrických obvodů. Byly nalezeny a prozkoumány volně dostupné simulátory mikrokontrolérů. Byl proveden návrh simulátoru MCU. Simulátor byl navrhován tak, aby byl schopen jednoduché výměny typu MCU. Součástí návrhu bylo také jednoduché okolí MCU ve formě třech elektrických obvodů. Také byla navržena vhodná vizualizace stavu modelu. Simulátor byl navrhován pro výukové účely. Simulátor byl naimplementován v jazyce C++ společně s mikrokontrolérem ATmega328p. Funkčnost byla ověřena na několika demonstračních aplikacích. Výsledky byly zhodnoceny.

Simulátor by bylo možné vylepšit. Okolí mikrokontroléru by se mohlo dát vytvářet v editoru elektrických obvodů z jednotlivých elektronických součástek a modulů. Také by se mohl zobrazovat vnitřní stav periferií mikrokontroléru.

Literatura

[1] Bill Rivet, T. K.: Overview — Simulavr homepage. 2012, [Online; navštíveno 5. 2. 2019].
URL https://www.nongnu.org/simulavr/

[2] buserror: GitHub — buserror/simavr: simavr is a lean, mean and hackable AVR simulator for linux & OSX. 2016, [Online; navštíveno 5. 2. 2019].
URL https://github.com/buserror/simavr

[3] Gottlieb, S.; Ketcheson, D. I.; Shu, C.-W.: *Strong stability preserving Runge-Kutta and multistep time discretizations*. World Scientific, 2011.

[4] gpsim: SimulIDE / Wiki / Home. [Online; navštíveno 5. 2. 2019].
URL https://sourceforge.net/p/simulide/wiki/Home/

[5] gpsim: gpsim. 2017, [Online; navštíveno 5. 2. 2019].
URL http://gpsim.sourceforge.net/

[6] Microchip: ATmega328P - 8-bit AVR Microcontrollers. 2019, [Online; navštíveno 1. 9. 2019].
URL https://www.microchip.com/wwwproducts/en/ATmega328P

[7] Microchip: ATmega48A, ATmega48PA, ATmega88A, ATmega88PA, ATmega168A, ATmega1688PA, ATmega328, ATmega328P datasheet. 2019, [Online; navštíveno 1. 9. 2019].
URL http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf

[8] SIMUINO: SIMUINO. 2017, [Online; navštíveno 5. 2. 2019].
URL http://web.simuino.com/

[9] Wikipedia contributors: Singleton pattern — Wikipedia, The Free Encyclopedia. 2019, [Online; navštíveno 12. 5. 2019].
URL https://en.wikipedia.org/w/index.php?title=Singleton_pattern&oldid=886672542

[10] Wikipedie: Jednočipový počítač — Wikipedie: Otevřená encyklopedie. 2018, [Online; navštíveno 12. 05. 2019].
URL https://cs.wikipedia.org/w/index.php?title=Jedno%C4%8Dipov%C3%BD_po%C4%8D%C3%ADta%C4%8D&oldid=16434177