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

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

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.

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. července 2019
Datum odevzdání: 31. července 2019
Datum schválení: 4. července 2019

Abstrakt

Cílem práce bylo prozkoumat existující volně dostupné simulátory mikrokontrolerů a vytvořit nový, který dokáže podporovat více typů mikrokontrolérů. Simulátor má své uživatelské rozhraní, které bylo řešeno pomocí grafického frameworku QT. Do simulátoru byla přidána podpora pro ATmega328p mikrokontrolér pomocí knihovny simavr. Pro okolí mikrokontroléru byly modelovány tři elektrické obvody, spínač, LED a RC článek. Simulátor byl implementován v C++ a úspěšně testován na několika příkladech, výsledkem práce je funkční simulátor podporující simulaci mikrokontroléru ATmega328p.

Abstract

This thesis deals with design and implementation of new simulator with possibility of supporting multiple types of microcontrollers and has its own graphical user interface. The user interface was implemented using the QT graphical framework. Support for the ATmega328p microcontroller was added to the simulator using the simavr library. Three electrical circuits were modeled as microcontroller surroundings, these are switch, LED and RC circuit. The simulator was implemented in C++ and its functionality was verified. The result of this work is a functional simulator supporting simulation of microcontroller ATmega328p.

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
29. července 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

Vývoj každého softwaru vyžaduje alespoň minimální testování na cílové platformě před použitím. V případě vývoje software pro mikrokontroléry, je potřeba přístup k samotnému mikrokontroléru a často i k hardwaru, s kterým má komunikovat. Toto může výrazně prodražit a zpomalit vývoj, zvláště když se jedná o drahý nebo špatně dostupný hardware. Testování s hardwarem může být v některých případech i nebezpečné. Tento problém řeší simulátory. V simulátoru lze testovat část nebo i celý vyvíjený systém bez nutnosti hardwaru. Simulátory mají také výhodu, že lze jednoduše sledovat co se přesně v hardwaru děje každý časový okamžik, což zjednodušuje hledání chyb. V softwaru se dá takto jednoduše odhalit hodně chyb aniž by se riskoval reálný hardware nebo zdraví testerů.

Důležitými vlastnostmi každého simulátoru mikrokontroléru je jaký mikrokontrolér/mikrokontroléry simuluje, jak moc simulace odpovídá realitě a jak řeší okolí mikrokontroléru. Modulární simulátor mikrokontrolérů je simulátor mikrokontrolérů, který umožňuje simulovat různé typy mikrokontrolérů. Každý simulátor má okolí zpracované jinak, některé ani žádné nemají a pouze zobrazují výstupy z mikrokontroléru a umožňují manuální zadávání vstupů. Některé umožňují připojit k mikrokontroléru moduly, které simulují reálné elektrické obvody. A některé umožňují elektrické obvody pro mikrokontrolér přímo postavit v editoru. Další důležitou vlastností je uživatelské rozhraní. Simulátory lze rozdělit na základě toho zda mají grafické uživatelské rozhraní nebo ne.

Tento text popisuje postup návrhu a implementace simulátoru mikrokontrolérů (MCU). Simulátor je modulární a disponuje vlastním grafickým uživatelským rozhraním. Okolí simulátoru je řešeno pomocí modulů simulujících jednoduché elektrické obvody.

Ve druhé kapitole jsou představeny některé již existující simulátory. Všechny představené simulátory jsou volně dostupné. Třetí kapitola obsahuje popis návrhu simulátoru. Nejprve je v kapitole popsáno jádro simulátoru, které je základní částí simulátoru. Návrh simulace elektrických obvodů. Návrhu grafického uživatelského rozhraní. A na závěr návrh implementace jednoho konkrétního mikroprocesoru. Jako realizovaný mikrokontrolér byl zvolen ATmega328p s architekturou AVR. Ve čtvrté kapitole je popsáno řešení implementace navržených částí. Jsou probrána řešení zajímavých částí a popsány drobné změny od návrhu, které vznikly při implementaci. A na závěr je také ověřena funkčnost hotového simulátoru na několika příkladech.

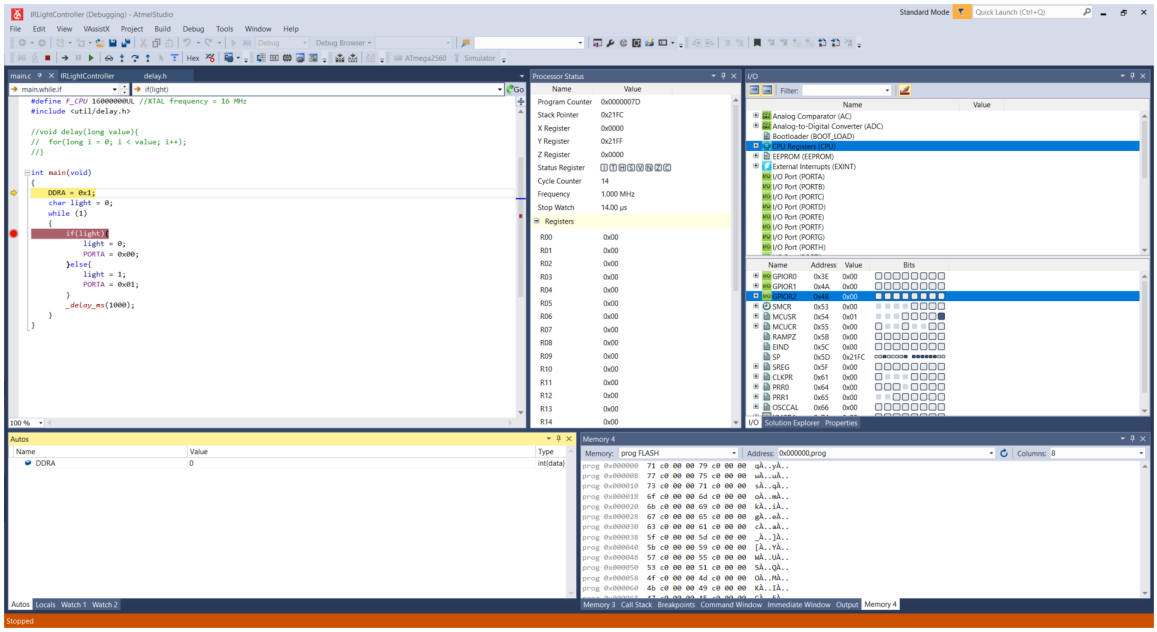
Kapitola 2

Volně dostupné simulátory mikrokontrolérů

Byl učiněn průzkum volně dostupných simulátorů mikrokontrolérů. Protože je na trhu velmi mnoho různých architektur MCU, je tato práce zaměřena na dvě velmi běžné osmibitové architektury, je to architektura AVR a PIC. V kapitole jsou simulátory představeny a jsou popsány některé jejich zajímavé vlastnosti. Mezi představenými simulátory jsou simulátory s grafickým uživatelským rozhraním i bez něj. Pokud grafické rozhraní existuje, je dodán ilustrační obrázek.

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 aplikace je na obrázku 2.1.



Obrázek 2.1: Atmel Studio

Simulátor zobrazuje obsah pamětí včetně registrů mikroprocesoru. Periferie jsou detailně zobrazovány v seznamu s jejich registry. Obsah registrů periferií se zobrazuje jak hexadecimálně tak v grafickém provedení, kde každý bit je reprezentován čtvercem, pokud má bit logickou 1, je čtverec vyplněný. U každého registru lze také zobrazit jednotlivé jména bitů i s popisem jejich funkce.

Po spuštění simulátoru, je k dispozici krokování programu. Program může být zapsaný v jazyce C, C++ nebo i v asembleru. Při krokování Atmel Studio ukazuje hodnoty proměnných.

Okolí mikrokontroléru je řešeno pomocí "stimuli"souborů. Tyto soubory obsahují jednoduché příkazy pro vytváření vstupů do mikrokontroléru. Také obsahují příkazy pro logování výstupu.

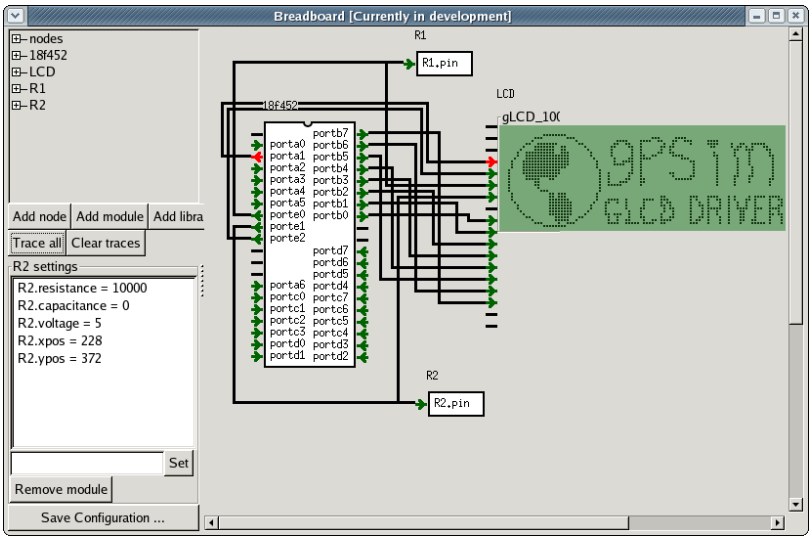
Atmel Studio je velmi propracované vývojové prostředí s vestavěným simulátorem. Simulátor poskytuje dobré a velmi přehledné uživatelské rozhraní. Simulátor je velmi dobrý pro ladění programů, ale má omezené možnosti co se týče okolí MCU.

2.2 gpsim

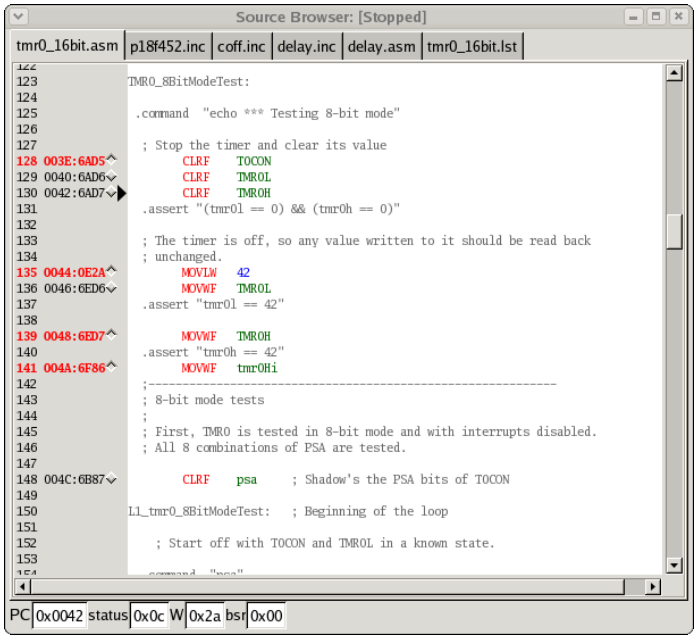
Gpsim je simulátor PIC mikrokontrolérů. Umožňuje spouštění v příkazové řádce, ale také má grafické rozhraní. Simulátor je navržený tak, aby byl co nejpřesnější a nejrychlejší co se týče jádra MCU a jeho interních periferií.

Okolí MCU je tvořeno připojitelnými komponentami. V simulátoru je k dispozici několik komponent. Patří k nim tlačítka, USART modul, LED, logická hradla atd.. Také je k dispozici grafický display, ten je vidět použitý na obrázku 2.2. Komponenty jsou simulátorem načítány zvlášť jako moduly, lze tak jednoduše použít moduly vytvořené komunitou nebo vytvořit vlastní.

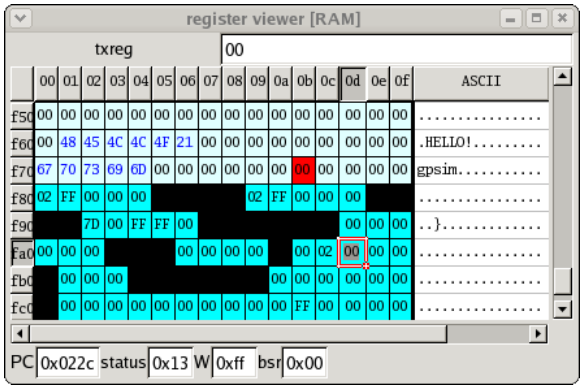
Na grafické rozhraní používá framework gtk. Rozhraní je členěno do více oken. Dalšími okny jsou například prohlížeč zdrojových kódů na obrázku 2.3 nebo prohlížeč registrů na obrázku 2.4.



Obrázek 2.2: qpsim[1]



Obrázek 2.3: prohlížeč zdojového kódu v gpsim[1]



Obrázek 2.4: prohlížeč registrů v qpsim[1]

Simulátor je kvalitní a výkoný nástroj pro simulaci PIC mikrokontrolérů. V základu neobsahuje velké množství komponent připojitelných k MCU, ty lze však najít na webu nebo vytvořit si vlastní.

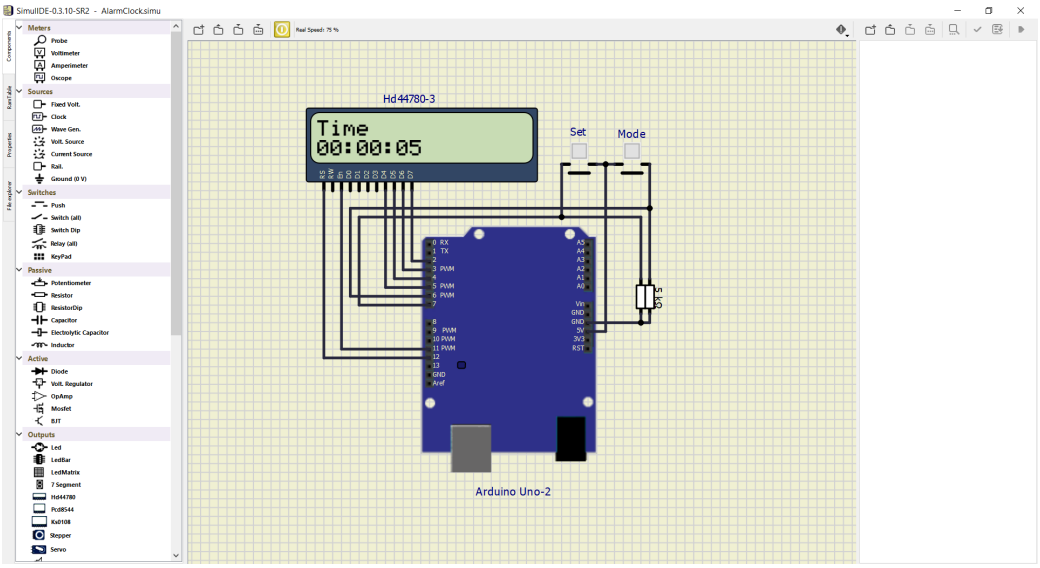
2.3 SimulIDE

SimulIDE je multiplatformní vývojové prostředí pro AVR i PIC mikrokontroléry. Součástí vývojového prostředí je i simulátor těchto dvou skupin MCU. Simulátor poskytuje intuitivní grafické prostředí pro začátečníky.

Okolí MCU se vytváří pomocí integrovaného editoru. Editor umožňuje stavět elektrické obvody ze základních elektrotechnických součástek. Kromě základních součástek jsou přítomny i komplexnější moduly, jako je alfanumerický display na obrázku 2.5, servo motor

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

nebo krokový motor. Uživatelské rozhraní je postaveno na frameworku QT. Simulace MCU se řeší pomocí knihoven simavr pro AVR MCU a gpsim pro PIC MCU.



Obrázek 2.5: simulIDE

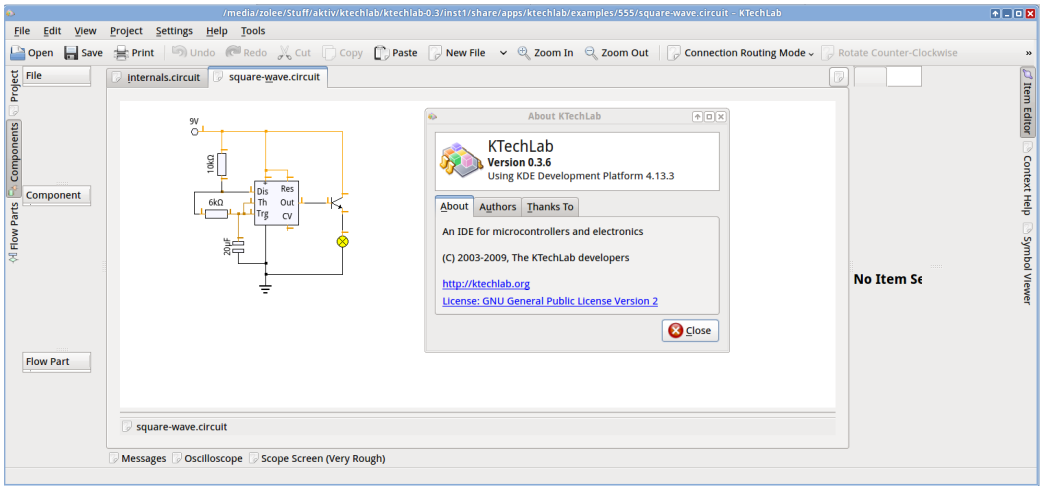
Simulátor není určen pro analýzu obvodů. Zaměřuje se na rychlost simulace a jednoduchost použití, proto přesnost simulace obvodů není příliš vysoká. SimulIDE je mířený na studenty a amatéry pro výuku a experimentování s jednoduchými obvody. [5]

SimulIDE je jednoduché vývojové prostředí a simulátor. Jeho velkou výhodou je, že umožňuje vytváření prakticky libovolných obvodů. Další výhodou je podpora jak AVR tak PIC mikrokontrolerů. Avšak editor kódu je pouze základní a postrádá spoustu nástrojů a funkcí. Simulace okolí MCU není také příliš přesná. Simulátor se tak hodí spíše pro začátečníky což je jeho zaměření.

2.4 KTechlab

KTechLab je vývojové prostředí s integrovaným simulátorem mikrokontrolérů PIC. Simulátor je vytvořený pro platformu linux. Ukázka programu je na obrázku 2.6. KTechLab obsahuje editor elektrických obvodů. V editoru lze vytvářet obvody ze základních součástek, tak i z komplexnějších.

Vývoj programu pro MCU lze provádět kromě klasického textového editoru pomocí grafového editoru. Program se tvoří v tomto editoru jako graf toku řízení. Tato funkce dovoluje novým uživatelům platformy PIC rychlý vývoj programů.



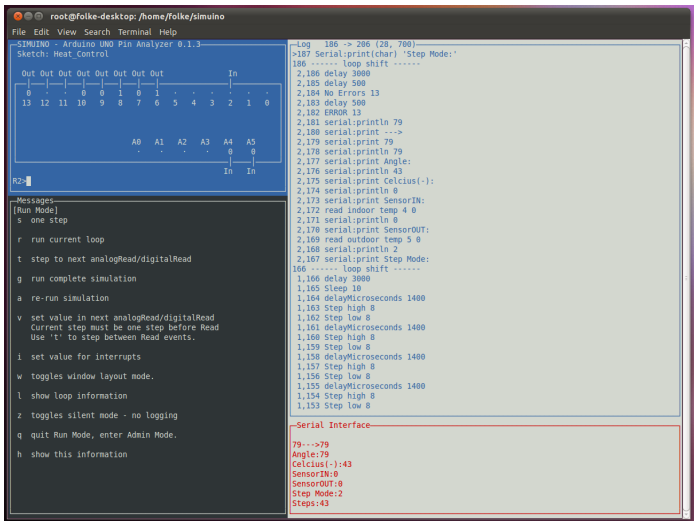
Obrázek 2.6: KTechLab[2]

Simulátor využívá knihovny qpsim pro simulaci MCU. Uživatelské rozhraní je realizováno pomocí frameworku QT. Stejně jako SimulIDE KTechLab obsahuje editor elektrických obvodů. Oproti SimulIDE však podporuje pouze PIC MCU. Editor kódu je však mnohem pokročilejší.

2.5 Simuino

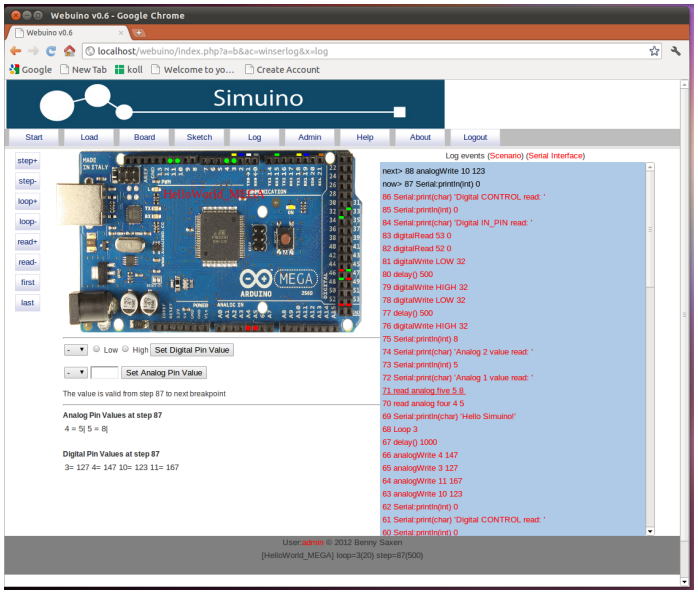
Simuino je simulátor platformy Arduino. Simulátor existuje ve dvou verzích. Verze v terminálu (obrázek 2.7) a verze webová (obrázek 2.8). Verze v terminálu je na platformu Linux.

Simulátor neposkytuje komplexní okolí MCU. Piny MCU se dají nastavovat manuálně. Druhá možnost je posílat předdefinované hodnoty ze souboru. Uživatelské rozhraní verze v terminálu je tvořeno pomocí knihovny ncurses.



Obrázek 2.7: Simuino[3]

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



Obrázek 2.8: Simuino[3]

Cílem simuino projektu je dát komukoli bez přístupu k platformě Arduino možnost se naučit základy psaní programů. Simuino vykonává program a ukazuje stav digitálních pinů, analogových pinů a sériovou komunikaci. [11]

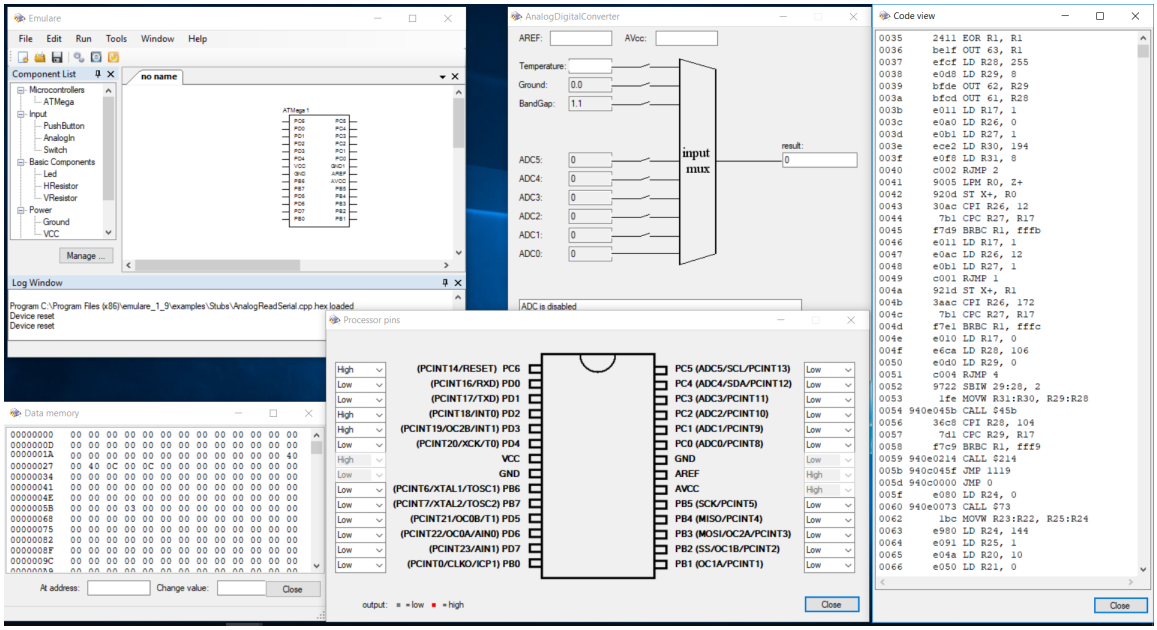
Simuino je velmi jednoduchý simulátor. Neposkytuje komplexní okolí MCU a uživatelské rozhraní není příliš uživatelsky příjemné, terminálová verze simulátoru je však přívětivější.

2.6 emulare

Emulare je simulátor mikrokontroléru ATmega328p. Tento simulátor je multiplatformní. Obsahuje také jednoduchý editor elektrických obvodů s několika komponentami. Několik základních elektronických součástek a LCD display. Nové komponenty lze přidat jako moduly.

Simulátor je napsaný v jazyce C#. Grafické rozhraní používá Windows Forms knihovnu. 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. Výhodou simulátoru je, že je multiplatformní a má editor elektrických obvodů. Jeho nevýhodou je ale, že editor má v základu málo komponent.

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



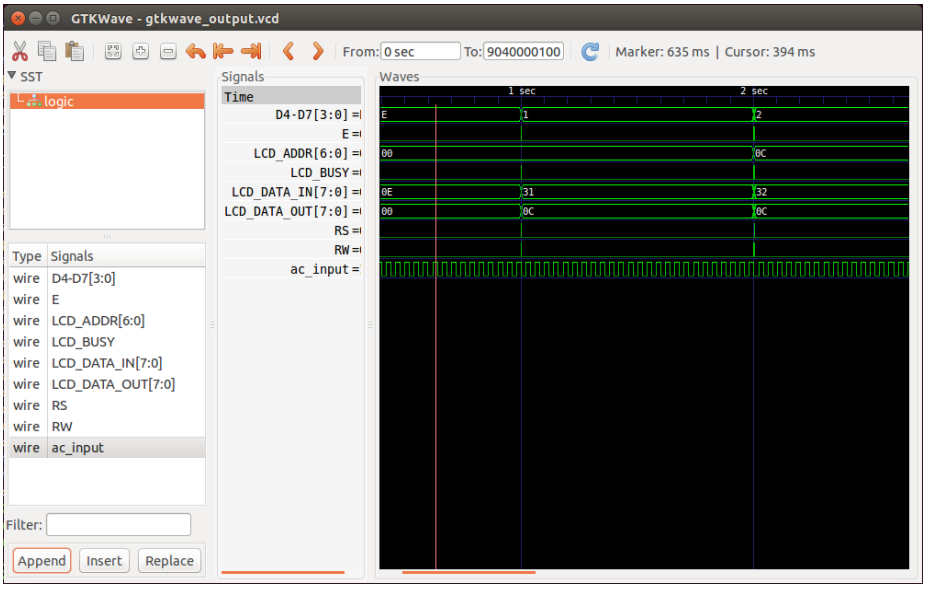
Obrázek 2.9: Emulare

2.7 simavr

simavr je simulátor mikrokontrolérů architektury AVR. simavr nemodeluje žádné okolí mikrokontroléru. Simulátor nemodeluje grafické rozhraní, je spíše zamýšlený jako knihovna. Je napsaný v jazyce C, ale hlavičkové soubory jsou upraveny, tak aby šel jednoduše použít i v jazyce C++. Simulátor je multiplatformní.

simavr umí zapisovat stav většiny svých pinů, proměnných a přerušení do souboru za běhu simulace. Záznam je ve formátu VCD, takový soubor může potom být vykreslen do grafu například pomocí programu GTKWave. Na obrázku 2.10 je výstup z příkladu výpisu na display zobrazen pomocí GTKWave. Tento systém umožňuje následnou analýzu výstupu. [2]

Simulátor umí načítat binární soubory s firmwarem mikrokontroleru ale také soubory ve formátu ELF. Formát ELF je na rozdíl od jednoduchého binárního souboru komplexní formát, proto pro zpracování souboru je využita knihovna libelf. V ELF souborech mohou být uloženy i jiné data než jenom firmware a simavr toho využívá. Pomocí maker lze při kompilaci firmwaru MCU specifikovat některé parametry simulace, které budou uloženy do ELF souboru. Takto lze specifikovat typ MCU, který bude použit, jeho napájecí napětí, referenční napětí pro komparátor MCU, sledované hodnoty, které se mají zapisovat do souboru a další. Výhodou je, že tyto nastavení jsou v jiné sekci než samotný firmware, nezaplňují tak paměť MCU.



Obrázek 2.10: záznam průběhu simulace zobrazený programem GTKWave

Simulátor podporuje ladění programu pomocí protokolu GDB Remote Serial Protocol. Pomocí tohoto protokolu lze připojit GDB a program MCU ladit. simavr podporuje několik základních příkazů protokolu. Mezi ně patří příkazy pro čtení a zápis paměti včetně registrů, krokování a resetování simulace. Také podporuje příkazy pro automatické pozastavení programu (breakpoint) a sledování přístupu do paměti (watchpoint).

Poskytuje pokročilý způsob ladění a záznam výstupu. Je napsaný v jazyce C a ne jen díky tomu je poměrně rychlý. simavr neobsahuje žádné okolí MCU a musí se vytvořit vlastní, ale jako knihovna je dobře použitelný.

2.8 SimulAVR

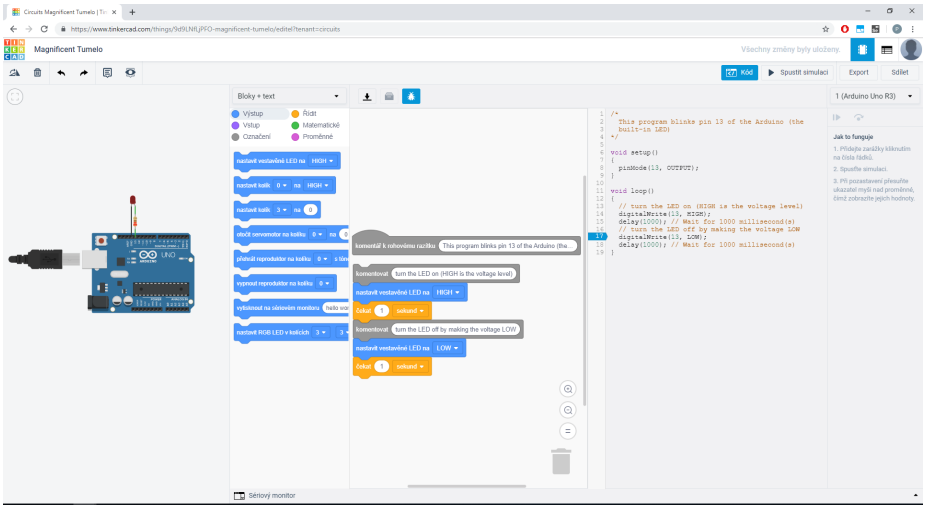
SimulAVR je simulátor mikrokontrolérů architektury AVR. SimulAVR lze použít jako knihovnu nebo samostatně. Simulátor je multiplatformní. Neimplementuje žádné okolí MCU, umí však zaznamenávat výstup ve formátu VCD, který lze zobrazit jako graf například programem gtkwave.

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 2.11.



Obrázek 2.11: Tinkercad

Tinkercad Circuits je jednoduchý a intuitivní simulátor. Editor obvodů je grafický. Zobrazuje součástky v jejich reálné podobě namísto schémat. Proto je tento nástroj vhodný pro začátečníky.

2.10 PICsim

PICSim je multiplatformní simulátor PIC mikrokontrolerů. Nemá grafické uživatelské rozhraní, rozhraní je pouze textové. Simulátor nemodeluje ani žádné okolí MCU, nicméně jeho rozhraní poskytuje jednoduchou implementaci vlastního okolí[7].

Simulátor emuluje sériovou linku MCU v operačním systému, MCU může tak komunikovat s okolím. Kromě sériové linky však PICsim nemodeluje žádné okolí MCU. Simulátor je celý napsaný v jazyce C a lze jej nainstalovat jako knihovnu. Hlavičkové soubory simulátoru jsou upraveny tak, že lze jednoduše použít i v jazyce C++.

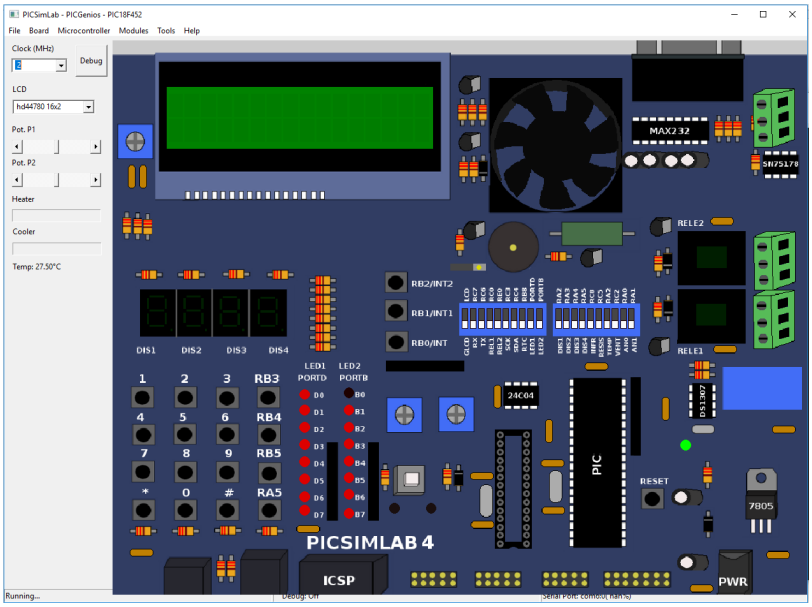
PICSim nepodporuje veškeré integrované periferie, které mikrokontroléry obsahují, a některé periferie pouze částečně. Jedná se hlavně o složitější periferie jako USB řadič nebo méně důležité jako senzor doteku. Některé typy MCU mají pouze podporu instrukční sady a postrádají většinu svých periferií.

Simulátor sice neobsahuje žádné okolí MCU ani grafické rozhraní, má ale velmi jednoduché API, je proto vhodný pro použití jako knihovna, pokud dané aplikaci nevadí, že knihovna postrádá některé periferie.

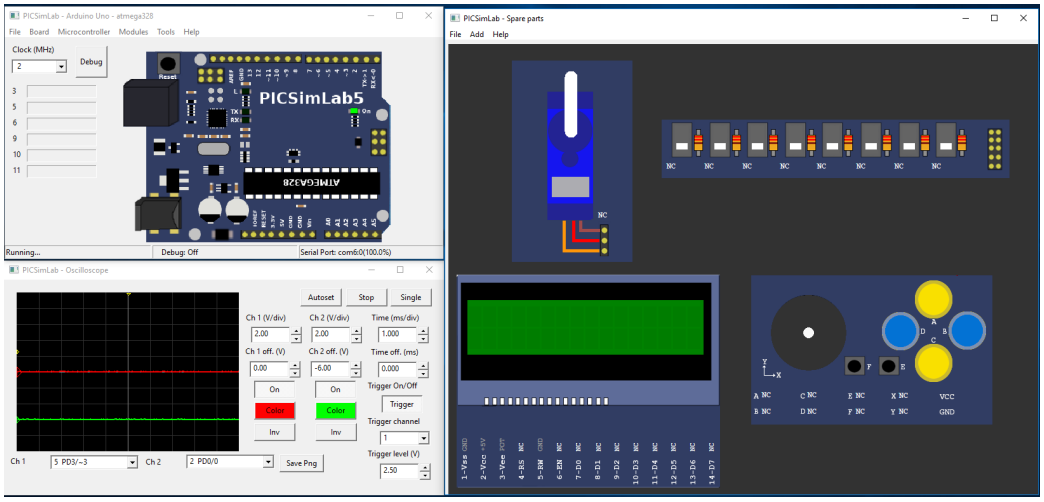
2.11 PICSimLab

PICSimLab je multiplatformní simulátor. Umožňuje simulaci několika PIC mikrokontrolérů ale také jednoho AVR mikrokontroleru. V simulátoru je na výběr ze čtyř vývojových desek, na které lze umístit PIC mikrokontrolér.

Na obrázku 2.12 je vidět simulátor s deskou PICGenios. Dále je k dispozici deska Arduino Uno s ATmega328 MCU. Ta je vidět na obrázku 2.13. Na obrázku je také vidět osciloskop a okno s několika moduly, které lze k desce připojit. K dispozici jsou displaye, gamepad, LED, potenciometry, spínače a další.



Obrázek 2.12: PICSimLab s PICGenios deskou



Obrázek 2.13: PICSimLab s Arduino Uno deskou, osciloskopem a modulama

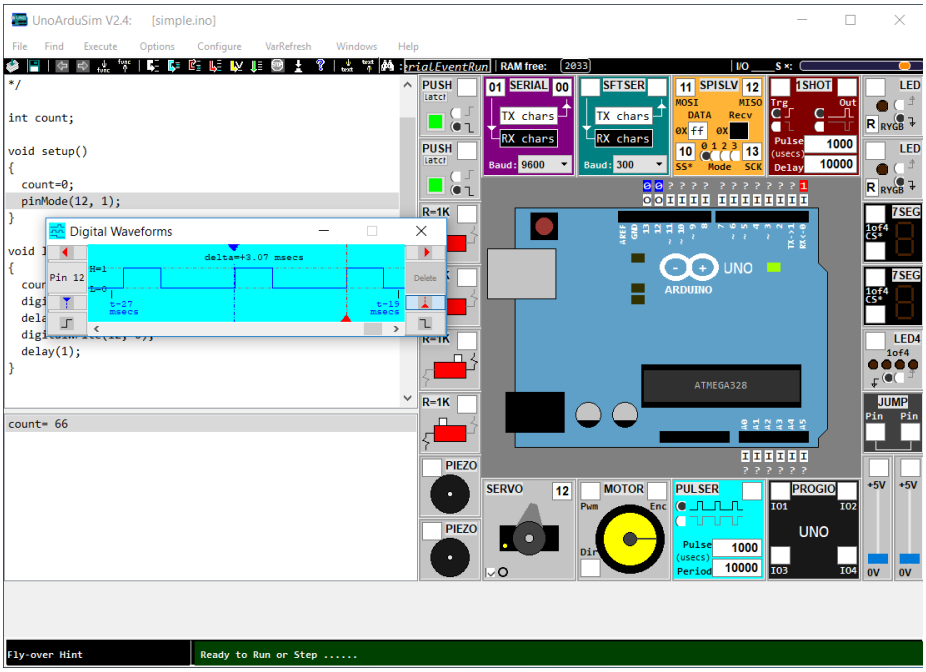
Simulátor neobsahuje žádný editor kódu. Místo toho se program načítá ze souboru. MCU lze ale také programovat přímo z vývojových prostředí MPLABX a Arduino IDE[8]. Pro simulaci PIC MCU využívá knihovny Picsim. Pro simulaci AVR MCU využívá simavr knihovny. Uživatelské rozhraní používá wxWidgets knihovnu.

PICSimLab je multiplatformní simulátor s grafickým rozhraním. Simulátor je celkem intuitivní a podporuje jak AVR tak PIC mikrokontroléry. Okolí mikrokontroleru je řešeno pomocí poměrně dobrého počtu modulů.

2.12 UnoArduSim

UnoArduSim je simulátor desky Arduino Uno. Uživatelské rozhraní simulátoru je postaveno pomocí frameworku QT, simulátor je ale pouze na platformu Windows. Na obrázku 2.14 je snímek obrazovky se simulátorem.

Kromě velmi jednoduchého editoru kódu, obsahuje uživatelské rozhraní také graficky vyobrazenou desku s mikrokontrolérem. Arduino desku lze propojovat s několika dostupnými moduly. Propojování je řešeno pomocí napsání čísla pinu do volného textového pole u modulu. U modulů je však velké omezení, moduly jsou již rozmístěny na obrazovce kolem desky a nelze jich použít více. Stav pinů lze sledovat ne jenom aktuální hodnotou, ale také pomocí grafu.



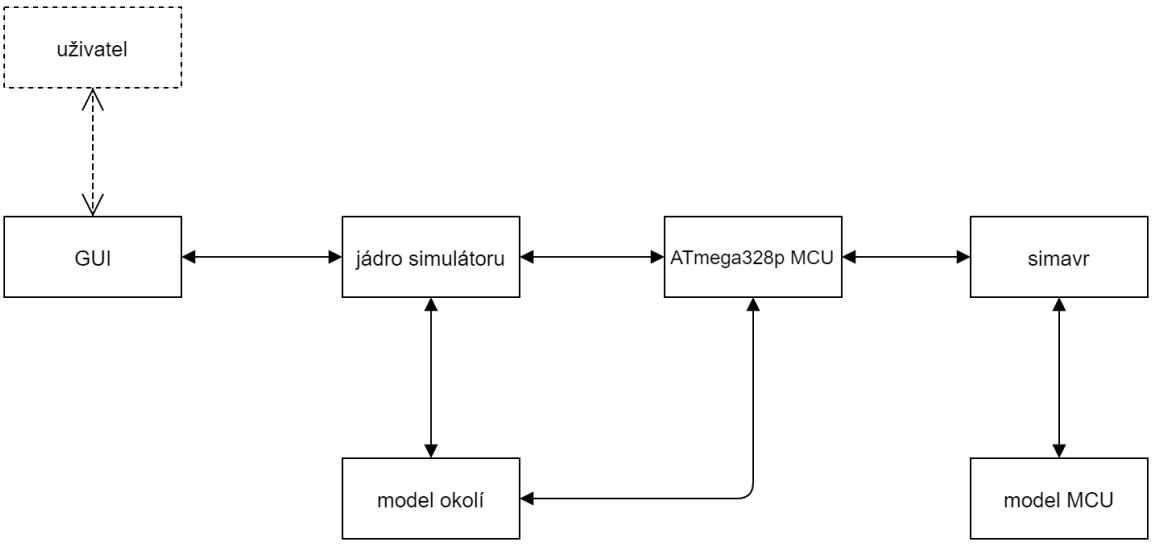
Obrázek 2.14: UnoArduSim

Simulátor poskytuje možnost základního editování kódu v jazyce C++ s knihovnami platformy Arduino, editor kódu je pouze základní a postrádá spoustu funkcí. Následně lze simulaci spustit nebo krokovat podle kódu. V režimu krokování lze sledovat obsah proměnných.

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. Na obrázku 3.1 je blokové schéma vysvětlující strukturu simulátoru. Program zahrnuje algoritmus řízení diskrétní simulace (pro MCU) a jednoduchý algoritmus řízeni spojité simulace (pro elektrické obvody v okolí MCU). Jde tedy o zjednodušenou formu algoritmu řízení kombinované simulace bez stavových podmínek a stavových událostí.



Obrázek 3.1: blokové schéma popisující strukturu výsledného simulátoru

3.1 Návrh jádra modulárního simulátoru

Simulátor bude navržen modulárně, aby umožňoval snadnou změnu implementace konkrétního MCU. V ideálním případě by mělo být možné naimplementovat jakýkoli mikrokontrolér s jakoukoli architekturou. Navrhnout takto univerzální simulátor by bylo náročnější a to nejen na realizaci jádra simulátoru, ale také uživatelského rozhraní. Proto se návrh omezuje na Harvardskou architekturu. Harvardská architektura byla zvolena proto, že ji používají populární MCU jako jsou PIC nebo AVR. Jádro simulátoru poskytuje základní kompo-

nenty mikrokontroléru, jako jsou paměti, periferie a centrální procesorová jednotka (CPU), která je jádrem mikroprocesoru. Mezi další komponenty patří také el. obvody. Také řeší komunikaci mezi těmito komponentami, ale také propojení s uživatelským rozhraním.

3.1.1 Rozhraní pro mikrokontroléry

Jádro simulátoru musí být postaveno tak, aby se dal snadno přidat nový MCU. Bylo proto navrženo rozhraní pomocí kterého lze mikrokontrolér jednoduše přidat. Jádro simulátoru definuje základní komponenty mikrokontroléru, jako jsou paměti, periferie a centrální procesorová jednotka (CPU) neboli jádro mikroprocesoru.

Pro splnění modularity bylo navrženo rozhraní, podle kterého budou jednotlivé implementace mikrokontrolérů komunikovat. Bylo navrženo několik tříd, které budou tvořit toto rozhraní. Na obrázku 3.2 třída *CPU* je určena jako rodičovská třída pro všechny typy MCU. Představuje logické a výpočetní jádro MCU. *CPU* obsahuje dvě virtuální metody, *load* a *getMemoryList*. Obě musejí být předefinované při dědění třídy. *load* slouží k načtení programu do MCU a *getMemoryList* vrací seznam pamětí MCU pro zobrazení v uživatelském rozhraní.

Třída *Memory* představuje paměť. MCU ji může použít jak potřebuje. Při jejím vytváření lze určit délku paměti a šířku sběrnice. MCU může mít těchto pamětí libovolné množství. Například pro registry, operační paměť atd.. Avšak MCU musí mít přesně jednu programovou paměť.

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*.

Každý mikrokontrolér potřebuje také disassembler. Pro něj je určena rodičovská třída *Disassembler*. Modul s mikrokontrolérem dodá třídu dědící od *Disassembler* pro překlad instrukční sady MCU.

Simulátor definuje rozhraní pro moduly mikroprocesorů. Rozhraní je navrženo, tak aby nebylo pro moduly příliš omezující. Nicméně jsou zde nějaká omezení, například omezení na Harvardskou architekturu.

3.1.2 Rozhraní pro elektrické obvody

Pro elektrické obvody je k dispozici společná rodičovská třída *Circuit*. Atributy třídy jsou vidět na obrázku 3.2. Třída poskytuje rozhraní pro řízení simulace. Také zajišťuje propojení s pinem MCU.

Circuit definuje metodu *makeStep*, pomocí které funguje spojitá simulace. Řízení simulace 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.

Třída si také drží odkaz na pin v atributu *connected*. Pokud není obvod připojený k žádnému pinu, je *connected* null. Pin vazbu na obvod nemá. Vždy když obvod potřebuje výstup z pinu musí si ho od pinu získat. Tento způsob není příliš efektivní, protože obvod hodnotu zjišťuje pokaždé i když se nezměnila. Vazba byla přesto navržena jednosměrná. Tímto způsobem nemůže dojít k datové nekonzistenci.

Aby se obvody nemusely starat o připojený pin a nebyly závislé na jeho rozhraní, byly navrženy tři metody, které práci usnadňují. Je to *outputToPin*, která předává výstup z obvodu pinu. *inputFromPin*, pomocí které obvod získá vstup z pinu. A *isInputPinConnected* podle které pozná, zda je připojený pin nastavený jako vstupní nebo výstupní. Třída *Circuit* poskytuje obvodům rozhraní. Obvody tak nejsou závislé na žádné jiné komponentě. A také poskytuje jednoduché rozhraní pro řízení simulace.

3.1.3 Řízení simulace

Simulátor obsahuje simulaci mikrokontroléru, což je diskrétní simulace. A simulaci elektrických obvodů, které jsou spojité komponenty. Celá simulace se proto musí řídit jako spojitá simulace. Z toho důvodu vznikla třída *SimulationManager*. Vznikla také třída *DiscreteComponent* pro vytvoření rozhraní diskrétním třídám. Tyto třídy jsou vidět v diagramu tříd na obrázku 3.2.

Třída *SimulationManager* řeší řízení kombinované simulace. Třída 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 bude třída poskytovat plánování na takty hodin místo času.

Jelikož *SimulationManager* 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[3]. Pro získání instance třídy se vytvořila třídní metoda *getInstance*.

Třída *DiscreteComponent* je rodičovská třída všech diskrétních komponent simulace, které se potřebují plánovat. *DiscreteComponent* má metodu *plan* pro usnadnění naplánování události diskrétním komponentám. A pro obdržení události stačí, když diskrétní komponenta předefinuje virtuální metodu *behavior*.

Hodinový takt MCU se uvažuje konstantní a délka kroku spojité simulace dělitelem nebo násobkem taktu MCU. Tímto se dosáhne toho, že se nemusí provádět dokračování, které se běžně u kombinované simulace provádí, a šetří se tím výpočetní výkon. Některé periferie MCU by mohli být citlivé na časové jevy a vyžadovat stavové události i mimo takt. Dobrým příkladem je třeba analogový komparátor. Toto by však až na výjimky nemělo vadit, jelikož výstup z těchto periferií je nakonec synchronizovaný s taktem procesoru.

Řízení simulace je důležitá část celého simulátoru. Řídí spojitou a diskrétní část simulace. Při návrhu bylo rozhodnuto, že se budou události plánovat na takty místo libovolného času. Toto rozhodnutí znamená jednodušší rozhraní za cenu snížení volnosti.

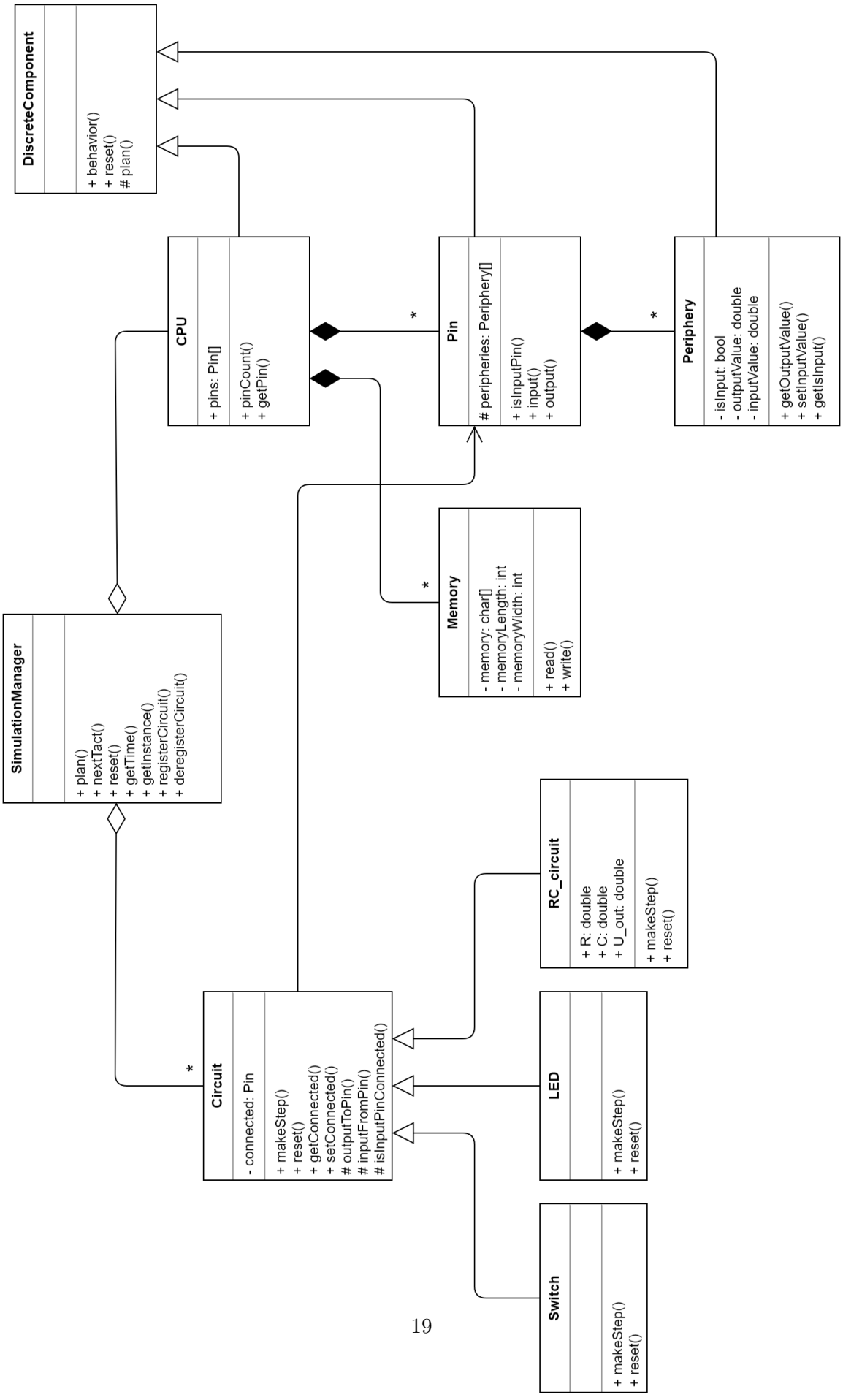
3.1.4 Systém resetování simulace

Simulátor obsahuje uživatelské rozhraní, které ovládá člověk, a je interaktivní. Uživatel bude pravděpodobně chtít spustit program MCU několikrát. Proto byl zaveden systém resetování simulace. Systém umožňuje uživateli opětovně spustit simulaci bez nutnosti znovu spouštět celý program. V případě opětovného spuštění simulace systém musí informovat všechny komponenty jádra, aby se uvedly do původního stavu.

Funkce je realizována metodami *reset*. Tyto metody implementují všechny třídy, které reset vyžadují. Metodu *reset* obsahuje třída *SimulationManager*, touto metodou dojde k resetu celé simulace. *SimulationManager* nastaví simulační čas na 0 a zavolá metodu *reset* u všech ostatních komponent. Metodu *reset* definuje třída *DiscreteComponent* jako virtu-

ální. A každá diskrétní komponenta, která potřebuje být resetována, ji může předefinovat. Metodu *reset* také definuje třída *Circuit*, také virtuální a po vzoru *DiscreteComponent* ji mohou obvody předefinovat podle potřeby.

Systém resetování simulace je velmi důležitý pro použitelnost simulátoru. Systém uvede všechny komponenty do počátečního stavu včetně simulačního času. Výhodou je, že lze jednoduše určit které části se neresetují. Například spínače, které uživatel přepnul zůstanou přepnuté i po resetu.

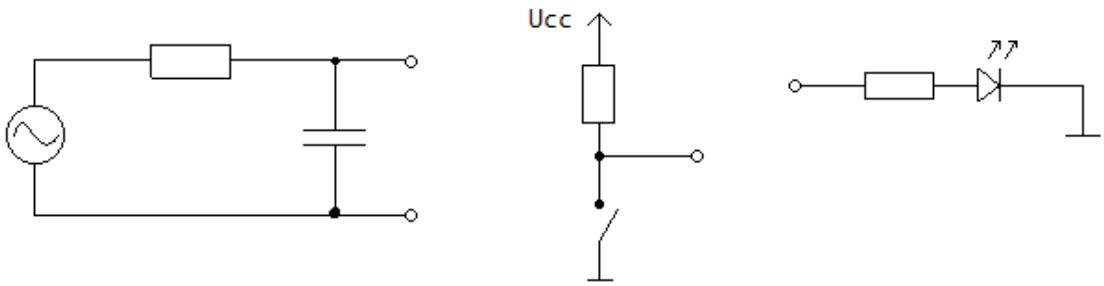


Obrázek 3.2: diagram tříd jádra simulátoru

3.2 Návrh modelů jednoduchých 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. Obvody, které by se používaly v kombinaci s pinem pracujícím ve výstupním režimu, aby MCU měl možnost produkovat nějaký výstup. A obvod s pinem ve vstupním režimu, aby měl mikrokontrolér možnost zpracovávat nějaká vstupní data.

Tři navržené obvody jsou na obrázku 3.3. Prvním obvodem zleva je RC článek. Pin mikrokontroléru bude možné připojit ke vstupu RC článku. Článek bude mít nastavitelné hodnoty rezistoru i kondenzátoru. Druhý obvod je spínač, pro manuální zadávání vstupu uživatelem. Třetím obvodem je LED, sloužící jako jednoduchý výstup. Simulace obvodů je zjednodušená a při simulaci obvodů se uvažuje s ideálními součástkami.



Obrázek 3.3: 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.

Při návrhu modelu RC článku se vycházelo z rovnice 3.1. Rovnice vyjadřuje závislost napětí a proudu na kondenzátoru v derivační formě. Rovnice 3.2 vychází z Ohmova zákona a popisuje proud na rezistoru. Výstup článku není nikam připojený a kondenzátor je tedy v sériovém zapojení s rezistorem. Jelikož jsou obě součástky v sériovém zapojení, stejný proud procházející rezistorem prochází i kondenzátorem. Proto mohla být rovnice 3.2 dosazena do rovnice 3.1 a po úpravě vznikla rovnice 3.3. Následně byla použita rovnice 3.4, která vznikla vyjádřením napětí na rezistoru z druhého Kirchhoffova zákona, 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.

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

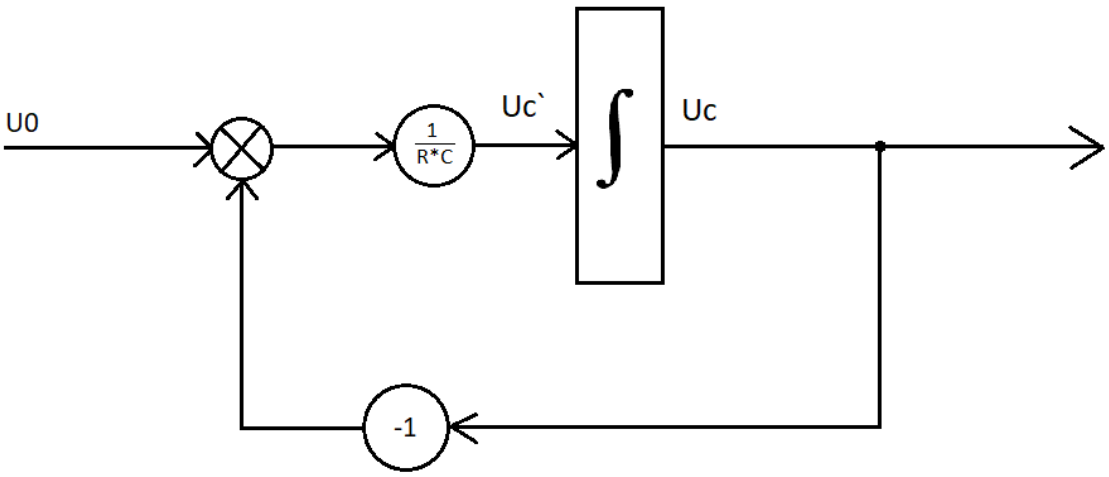
$$i(t) = \frac{U_R}{R} \tag{3.2}$$

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

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

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

Ze sestavené rovnice 3.5 byl vytvořen blokový model na obrázku 3.4. Model obsahuje jeden vstup, vstupní napětí článku. A jeden výstup, napětí na kondenzátoru, které je rovné výstupnímu napětí na celém článku. Model se skládá z několika jednoduchých aritmetických bloků a jednoho integrátoru.



Obrázek 3.4: blokové schéma popisující model RC článku

Implementace integrátoru v modelu RC článku bude pomocí metody Runge-Kutta čtvrtého řádu. Při použití numerických metod se projevují chyby aritmetických operací. Chyba zaokrouhlování se projevuje hlavně při příliš malém kroku numerické metody[6]. Chyba ořezání se zase projevuje při příliš dlouhém kroku[6]. Proto se musí zvolit vhodná délka kroku, kde celková chyba je nízká, v takovém případě bude mít metoda dostatečnou přesnost s relativně nízkými výpočetními nároky.

3.3 Návrh uživatelského rozhraní

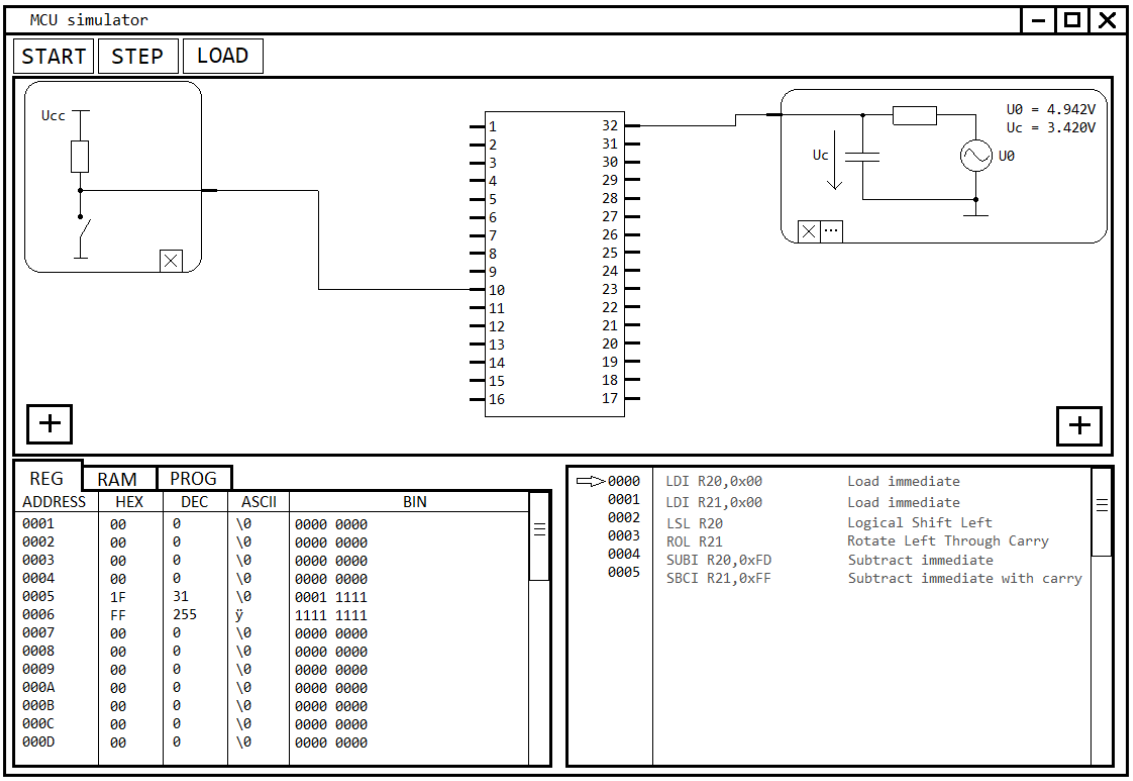
Aplikace je určena pro výukové účely. Měla by být dostatečně intuitivní a přehledná. Návrh rozhraní spočívá ve vytvoření návrhu ovládacích prvků aplikace a návrhu realizace daného rozhraní. Realizace zahrnuje výběr použitého frameworku, důležitých tříd a propojení uživatelského rozhraní s jádrem simulátoru.

3.3.1 Ovládací prvky uživatelského rozhraní

Pro simulátor byly navrženy ovládací prvky a jejich rozestavení. Rozhraní se skládá převážně ze standardních prvků grafického rozhraní, což ulehčuje implementaci jelikož většina

grafických frameworků tyto prvky podporuje. Návrh však obsahuje i nestandardní prvky, jako schémata obvodů a grafické spoje, jejich realizace je řešena v podkapitolách 3.3.2 a 4.4. Celá aplikace bude zobrazena v jednom okně, kromě pár dialogových oken. Výsledný návrh je na obrázku 3.5.

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 3.5: grafický návrh uživatelského rozhraní

S ohledem na znalosti cílové skupiny uživatelů, mikrokontrolér i s obvody jsou zobrazeny schématicky a 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. Návrh řeší pouze hlavní okno aplikace a nepokrývá formuláře a upozornění, které se budou zobrazovat v dialogových oknech, budou totiž standardní.

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 MCU je zobrazen v jazyku symbolických adres. Aktuální pozice v programu je zobrazena pomocí šipky ukazující na instrukci, která je na adrese v PC registru (program counter register).

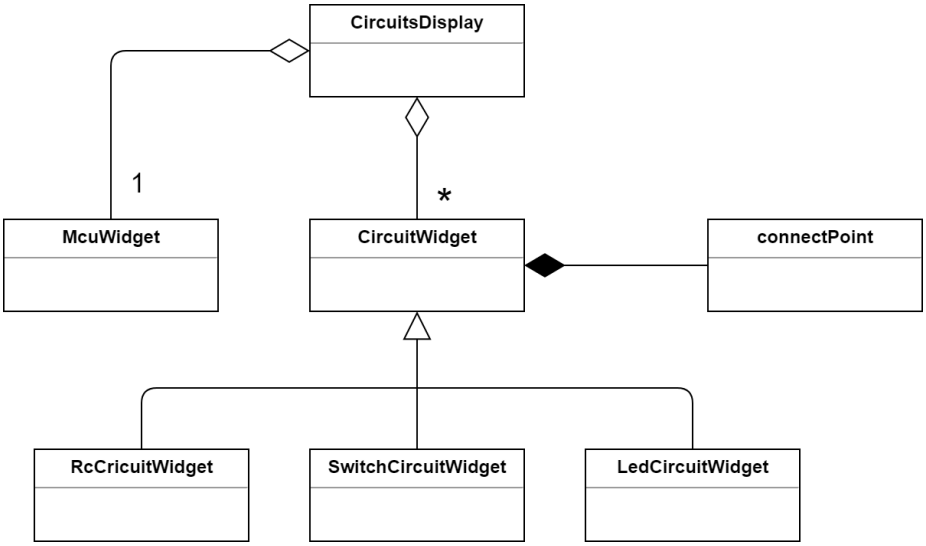
3.3.2 Návrh uživatelského rozhraní v QT frameworku

Tento podkapitola se zabývá realizací uživatelského rozhraní, které bylo navrhnuto v podkapitole 3.3.1. Pro realizaci byl zvolen framework QT. Byla řešena komunikace s jádrem simulátoru, bylo navrženo několik hlavních tříd rozhraní a navržena realizace některých prvků grafického rozhraní pomocí existujících tříd z QT frameworku.

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*. *CPU* posílá signál, když dojde ke změně programového čítače. *Memory* posílá signál, když dojde k zápisu paměti. A *Circuit* posílá signál, když dojde ke změně stavových hodnot. Třída *CPU* již dědí od *DiscreteComponent*, tento problém je vyřešen vícenásobnou dědičností jazyku C++.

Uživatelské rozhraní bylo navrženo pomocí často používaného návrhového vzoru MVC (model–view–controller)[12]. Třídy z jádra simulátoru slouží jako modely. Ke každému obvodu byl vytvořen view a controller, který obvod zobrazuje a řeší interakci s uživatelem. Tyto třídy jsou potomky třídy *QWidget*, která je bázovou třídou všech vizuálních komponent frameworku QT.

Obrázek 3.6 popisuje třídy sloužící ke grafickému zobrazení obvodů a MCU v horní části okna aplikace. Třída *CircuitsDisplay* dědí od *QWidget* a vytváří prázdnou grafickou plochu v okně pro grafiku obvodů a MCU a také se stará o vykreslení spojů mezi nimi. Obsahuje místo pro třídu *MCUWidget*, která zobrazuje MCU. Také obsahuje pole tříd *CircuitWidget*. *CircuitWidget* je rodičovská třída všech tříd zobrazujících obvody. Vykreslování schémat obvodů třídou *CircuitWidget* je řešeno transparentními obrázky. Třída *ConnectPoint* vytváří přípojné místo na obvodu pro MCU.



Obrázek 3.6: diagram tříd zobrazení obvodů

Disassembler je řešen pomocí tabulky, která je součástí frameworku. Jeden sloupec je pro ukazatel programového čítače, druhý pro adresu instrukce a třetí obsahuje instrukci s popisem. Stejně jsou řešeny paměti. Rozdíl je, že každá tabulka paměti je ve vlastní záložce. Záložky jsou tvořeny třídou *QTabWidget* frameworku.

Uživatelské rozhraní bylo navrženo pomocí frameworku QT. QT nepomáhá pouze se zobrazením grafiky, ale i s komunikací uvnitř aplikace. Také řeší platformě závislé věci jako otevírání souborů, aplikace tak může zůstat multiplatformní.

3.4 Návrh simulace mikrokontroléru ATmega328p

Do simulátoru byl přidán jeden mikrokontrolér. Zvolen byl mikrokontrolér *ATmega328p* od společnosti Microchip Technology. Simulace mikrokontroléru je řešena pomocí knihovny *simavr*. V kapitole je vybraný mikrokontrolér představen a je řešen způsob propojení knihovny *simavr* se simulátorem.

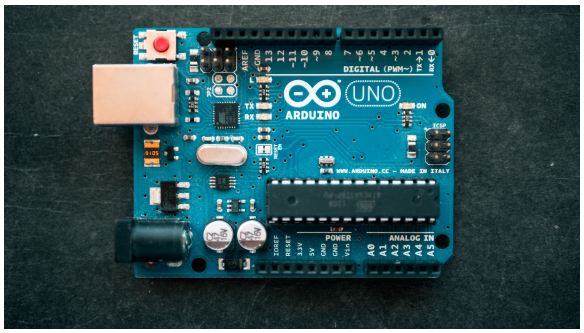
3.4.1 Mikrokontrolér ATmega328p

ATmega328p je populární mikrokontrolér od společnosti Microchip Technology. Je to 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. [9]

ATmega328p byl zvolen převážně z důvodu jeho vysoké popularity. *ATmega328p* je nejznámější právě díky platformě *Arduino*. Mikrokontrolér je osazen na deskách řady *Arduino Uno*. *Arduino Uno* je na obrázku 3.7.

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 ADC, 1 komparátor, 6 PWM, 1 Input Capture
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	až 32

Tabulka 3.1: parametry ATmega328p [9]



Obrázek 3.7: Arduino Uno[1]

3.4.2 Návrh propojení s knihovnou simavr

Pro implementaci MCU byla zvolena knihovna *simavr*. Knihovna byla představena v podkapitole 2.7. Knihovna podporuje celou řadu mikrokontrolérů architektury AVR, mezi které také patří *ATmega328p*. *simavr* knihovna je pod licencí GNU General Public License v3.0. Podle požadavků licence knihovny, vytvářený simulátor bude muset být také pod stejnou licencí. Propojení simulátoru s knihovnou bude realizované tak, že bude použito stejné rozhraní, které bylo navrhnuto v kapitole 3.1. Knihovna má odlišné rozhraní než navržený simulátor, návrh tedy využívá návrhového vzoru adaptér[3]. Byly navrženy čtyři třídy. Třída *ATmega328p*, *avrDisassembler*, *AVRPin* a *AVRPeriphery*.

Návrh třídy ATmega328p

Třída *ATmega328p* dědí od třídy *CPU*. Třída bude inicializovat MCU z knihovny, bude zařizovat vykonávání instrukcí a další logiku pro funkci. Třída předefinovává všechny virtuální metody své rodičovské třídy *CPU* a její rodičovské třídy *DiscreteComponent*. Jsou to metody *behavior*, *reset*, *load* a *getMemoryList*. Jejich funkce je popsána v kapitole 3.1.

getMemoryList předává paměť mikrokontroléru. Paměť obsahuje obecné registry, v/v registry a SRAM. Tyto paměti jsou v *AVR* architektuře dostupné ve stejném adresním

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

prostoru, a není tak neobvyklé je udávat jako jednu paměť. *simavr* je také realizuje jako jedno pole bajtů.

Metoda *Load* kromě načtení programu do MCU slouží také k vytvoření pinů (třída *AVRPin*). Každému pinu je přiděleno jméno a jedna periferie (třída *AVRPeriphery*). Periferiím se předává číslo *irq*. *irq* (interrupt request) je vnitřní komunikační systém knihovny *simavr*. Používá se ke komunikaci mezi moduly knihovny. Periferie *irq* využijí k předávání hodnot na pinech.

Návrh třídy avrDisassembler

Třída *avrDisassembler* dědí od *Disassembler*. Poskytuje překlad strojového kódu instrukční sady *AVR* do jazyka symbolických adres. Na překlad bude použit vývojový nástroj pro *AVR* mikrokontroléry *avr-objdump*. Tento nástroj je součástí oficiálních nástrojů (toolchain). Nástroj ukazuje informace o binárních souborech a také je dokáže přeložit do jazyka symbolických adres.

Třída bude spouštět *avr-objdump* jako příkaz příkazové řádky pomocí třídy *qProcess* z frameworku QT. Pomocí argumentů se specifikuje, že je požadován překlad programu. Po získání výstupu z programu se data zpracují. Výstup se rozdělí na jednotlivé řetězce po instrukcích a každé instrukci je přiřazena její adresa v programové paměti.

Návrh třídy AVRPeriphery

Třída *AVRPeriphery* dědí od *Periphery*. Třída bude zajišťovat komunikaci pinů MCU s knihovnou. Ve vstupním režimu bude předávat hodnotu na pinu do knihovny. A ve výstupním hodnotu dodanou knihovnou vystaví na pin.

simavr pro některé periferie vyžaduje stanovené napájecí napětí. Například pro převodníky a komparátory. Zvoleno bylo napájecí napětí 5,0 V. Stejné napětí jako na deskách Arduino Uno.

Návrh třídy AVRPin

Třída *AVRPin* dědí od třídy *Pin*. Třída by podle rozhraní simulátoru měla řešit přepínání periferií, ale *simavr* toto už řeší interně. *AVRPin* tak bude mít pouze jednu periferii. Kromě poskytování funkcí své rodičovské třídy, třída nic podstatného obsahovat nebude.

Jedinou věc co dělá je inicializace v konstruktoru. Konstruktor přijímá jméno a jednu periferii. Inicializace spočívá v předání jména rodičovské třídě. A přidání periferie do pole periferií své rodičovské třídy.

AVRPin neplní žádnou zajímavou funkci, má pouze jednoduchý konstruktor. Avšak je velmi důležitá protože slouží jako prostředník mezi svojí jednou periferií, el. obvody a třídou *ATmega328p*.

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++ a pro vývoj bylo použito vývojové prostředí Qt Creator. 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

Implementace jádra vychází z kapitoly 3.1. Jako první se popisuje řešení zprávy událostí u řízení simulace. Oproti návrhu byla přidána pomocná třída *EventPriorityQueue*, která slouží jako fronta událostí. Jako druhé je ukázáno řešení rozhraní pro elektrické obvody. Zde se popisuje jak byly řešeny metody usnadňující komunikaci obvodům.

4.1.1 Implementace řízení simulace

Události se plánují v *SimulationManager* 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. *SimulationManager* 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 funkci vložení události s prioritou a odebrání nejprioritnější události. Události v *EventPriorityQueue* jsou uchovávány 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 kdykoliv bez problému navýšit. Snížením by se však riskovalo rozbití implementací existujících MCU.

4.1.2 Rodičovská třída obvodů

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.

4.2 Implementace simulace mikrokontroléru ATmega328p

Mikrokontrolér je realizován pomocí knihovny simavr. Implementace mikrokontroléru se odvíjí od návrhu z podkapitoly 3.4.2. Bylo navrženo několik tříd, které slouží jako adaptér ve stejnojmenném návrhovém vzoru pro knihovnu *simavr*.

4.2.1 Implementace třídy ATmega328p

Třída *ATmega328p* dědí od třídy *CPU* a představuje jádro celého mikrokontroléru. Tato třída provádí inicializaci MCU z knihovny, načtení programu pro MCU ze souboru, uvádí MCU do původního stavu, řídí vykonávání instrukcí a další.

Inicializace MCU z knihovny se provádí v metodě *Load*. Vytváří se zde struktura *avr_t*, která je hlavní strukturou *simavr*. Obsahuje všechny informace o aktuálním stavu MCU. Metoda také při vytváření nahrává program do MCU. Kód pro čtení programu ze souboru byl převzat ze zdrojových souborů *simavr* a následně upraven.

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*.

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

Metoda *getMemoryList* předává paměť MCU, která je ve formě instance třídy *Memory*. *simavr* si však paměť alokuje vlastní. 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 alokovanou třídou *Memory*. To způsobuje ale další problém. Jelikož knihovna přistupuje k paměti přímo a ne přes metodu *write* třídy *Memory*, nelze sledovat, která část paměti 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 metoda *allChanged*, která se volá po vykonání každé instrukce.

4.2.2 Implementace periferií mikroprocesoru

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). *AVRPin* nijak neinteraguje s knihovnou, je pouze prostředníkem pro periferii a zbytek simulátoru.

Systém *irq* je interní systém pro komunikaci komponent knihovny *simavr*. V metodě *behavior* třídy *AVRPeriphery*, pokud operuje ve vstupním režimu, vyvolává *irq* se vstupní hodnotou. Pro digitální vstup knihovna přijímá logickou 1 nebo 0. Podle technické specifikace (*datasheet*)[10] 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 nedefinované pásmo. Naimplementován byl ideální případ, kdy ke změně z logické 0 na 1 dojde v polovině nedefinované 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*.

Při implementaci bylo zjištěno, že komunikace s periferiemi je v *simavr* obtížnější. *irq* pinu dovoluje pouze logické hodnoty na vstupu. Periferie jako AD převodník, vyžadují vstup hodnotou napětí na pinu. A musí se tedy použít *irq* patřící konkrétní komponentě. Při implementaci byl problém vyřešen vytvořením nové třídy *AVRPeripheryADC*. Třída dědí od *AVRPeriphery*. Původní chování bylo zachováno, pouze přidán vstup do AD převodníku.

4.2.3 Implementace disassembleru mikrokontroleru

Překlad ze strojového kódu na jazyk symbolických adres řeší třída *avrDisassembler* dědící od *Disassembler*. Překlad byl realizován pomocí nástroje *avr-objdump*. Spouští se jako příkaz příkazové řádky pomocí třídy *qProcess* z frameworku QT. Pomocí argumentů se specifikuje, že je požadován překlad programu.

V metodě *load* třídy *avrDisassembler* se spouští nástroj *avr-objdump*. Výstup z nástroje je zpracován a uložen v instanci třídy. 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 od jiných řádků. Dalším argumentem je *"-d"*, který udává, že se má provést překlad programu a posledním argumentem je cesta k souboru. Výsledná data se ukládají do privátních členů objektu definovaných takto:

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

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

také po řádcích. Tyto pole jsou vždy při novém volání metody *load* vyprázdněny pro příklad načítání nového programu.

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. Implementace obvodů se odvíjí od návrhu z kapitoly 3.2.

4.3.1 Implementace LED

Dioda se má rozsvítit při překonání prahového napětí 1,8 V. LED byla navržena jako ideální prvek a její implementace není složitá. Je implementována v třídě *LED* jenž je potomkem třídy *Circuit*.

LED uchovává svůj vnitřní stav v proměnné jako logickou hodnotu pravda, když svítí a nepravda, když nesvítí. 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. Vždy když se stav diody změní, informuje se uživatelské rozhraní pomocí vyvolání příslušného signálu. Rozhraní si pak může zjistit stav diody metodou *isLightOn*.

4.3.2 Implementace spínače

Spínač má dva stavy, rozepnutý a sepnutý. Napětí na pinu spínač nastavuje podle jeho stavu. Stejně jako u diody je implementace spínače relativně jednoduchá. Spínač je implementován v třídě *Switch* dědící od třídy *Circuit*.

Spínač má metodu *switchState*, která tvoří rozhraní pro ovládání spínače. Metoda přepíná mezi sepnutým a rozepnutým stavem. Metodu volá uživatelské rozhraní, když se uživatel snaží přepnout spínač. Stav spínače se ukládá v logické proměnné. Metoda *makeStep* posílá napětí na připojený pin MCU podle stavu této proměnné.

V metodě *makeStep* se také kontroluje, zda nedošlo ke zkratu. Pokud je spínač sepnutý je pin přímo připojený na zem a pokud se v tomto stavu pin nastaví jako výstupní, dojde ke zkratu ve chvíli kdy bude na pinu nenulové napětí. Pokud ke zkratu dojde, je nastavena proměnná. Pomocí metody *isShortOccured* může uživatelské rozhraní zjistit, jestli ke zkratu došlo.

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.

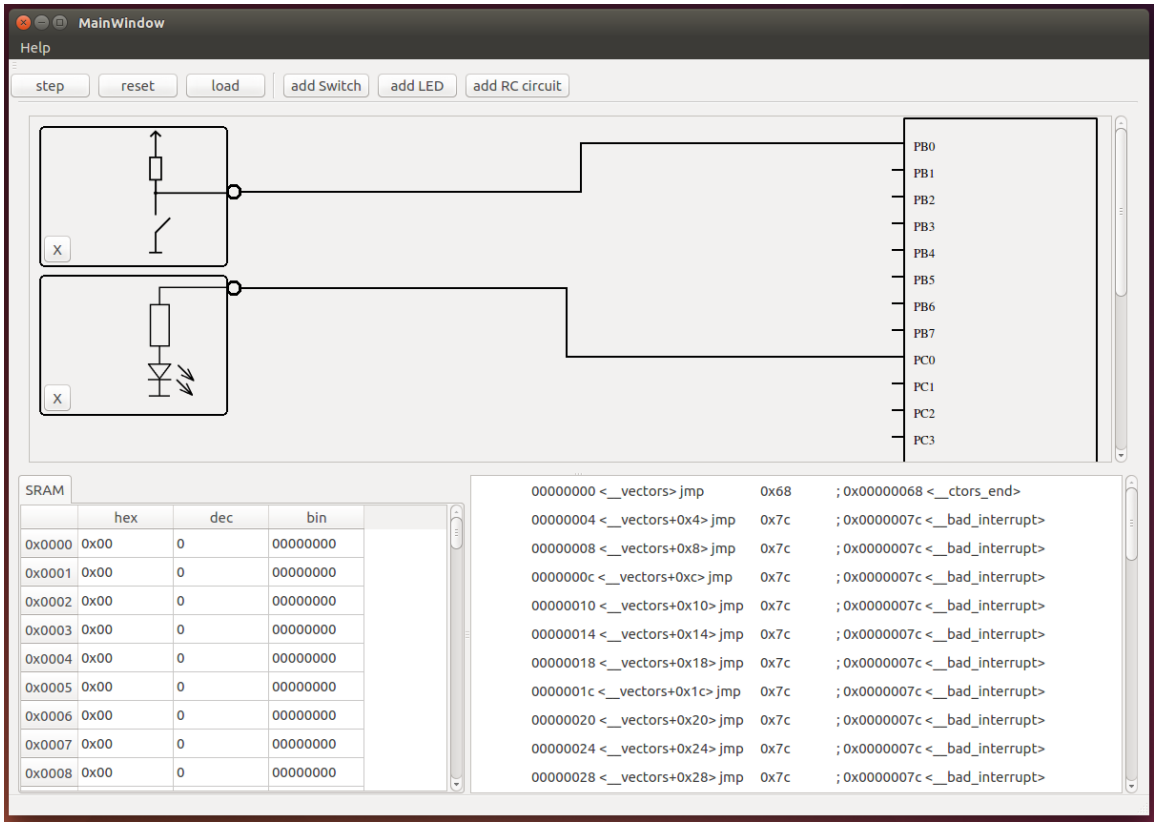
Integrátor byl implementován pomocí vzorce metody Runge–Kutta čtvrtého řádu[4]. Aritmetická část modelu vypočítává derivaci výstupního napětí U'_C. Derivace se počítá ve funkci *dynamic*. Počítá se ze vstupního napětí *sourceVoltage*, odporu rezistoru *rezistance*, kapacity kondenzátoru *capacitance* a výstupního napětí *outputVoltage*. Toto je princip implementace výpočtu:

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

4.4 Implementace uživatelského rozhraní

Uživatelské rozhraní bylo implementováno pomocí QT frameworku. Uživatelské rozhraní bylo vytvářeno po vzoru prototypu na obrázku 3.5. Implementace se týká především tříd z obrázku 3.6.

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

Pro některé grafické elementy se nepodařilo nalézt vhodnou třídu ve frameworku QT. Takové elementy se vykreslují manuálně pomocí třídy *QPainter* frameworku. Třída *CircuitWidget* takto vykresluje svoje ohraničení. *McuWidget* pomocí *QPainter* kreslí celý MCU i s piny. Také třída *ConnectPoint* používá *QPainter*, kreslí malé kolečko, které zobrazuje přípojný bod.

QPainter používá také třída *CircuitDisplay* pro kreslení spojů mezi obvody a MCU. Spoje jsou kresleny jednoduchým algoritmem. Spoje jsou tvořeny ze tří čar. Jedna čára vede horizontálně z přípojného bodu obvodu, druhá horizontálně z pinu na MCU. V bodě, kde se čáry potkávají v **y** souřadnici, jsou propojeny vertikální čarou. Místo vertikálního propojení čar je v polovině délky spoje, tedy mezi přípojným bodem a pinem. Aby se spoje nepřekrývaly, je k tomuto místu přičteno odsazení. Každý spoj má odsazení jiné.

Vytváření spojů je také řízeno třídou *CircuitDisplay*. *CircuitDisplay* je připojena na signál kliknutí na *ConnectPoint* každého obvodu. Dále je také připojena na signál *McuWidget*,

který signál posílá po kliknutí na pin. *McuWidget* posílá signál s číslem pinu v argumentu signálu.

K odstraňování obvodů slouží tlačítko na každém obvodu, které umisťuje na sebe třída *CircuitWidget*. Při stisku tlačítka se posílá signál *CircuitWidget* třídě, která obvod odstraní.

Při implementaci rozhraní se podařilo dosáhnout podobného vzhledu jako prototyp, který byl předlohou. Oproti prototypu bylo provedeno pár změn. Největší změna je asi přemístění MCU na pravou stranu okna.

4.5 Ověření funkčnosti

Funkčnost výsledného simulátoru byla ověřena na pěti příkladech a příklady byly vloženy do složky *examples* v kořenovém adresáři projektu. Testování simulátoru probíhalo na platformě linux. Simulátor byl otestován na systémech Ubuntu 16 32bit a Debian 10 64bit.

První příklad je jednoduché blikání LED. Jmenuje se *blink* a byl vytvořen pro otestování výstupu. Program přepíná logické hodnoty na portu B v nekonečném cyklu a k jednomu z pinů portu je připojena světelná dioda. Program byl odzkoušen na reálném hardware, obvod byl postaven ze světelné diody a rezistoru v sériovém zapojení s odporem 200Ω. Blikání bylo pozorovatelné až po zavedení dostatečné časové prodlevy mezi přepnutím výstupu. Stejný program byl otestován v simulátoru s LED modulem a podle předpokladu bylo také pozorováno blikání diody.

Ostatní příklady byly testovány podobně jako v prvním příkladu. Druhý příklad je *switch*. Příklad rozsvěcí LED v reakci na sepnutí spínače. Tím se testuje vstup MCU. Další příklady testují periferie integrované v MCU. Příklad *adc* testuje AD převodník. Příklad *Interrupt* testuje externí přerušení. A příklad *pwm* testuje generování PWM.

Korektnost příkladů byla ověřena na reálném ATmega328p. Příklady následně byly vyzkoušeny v simulátoru a výsledky porovnány. Simulátor se choval podle očekávání na obou systémech. Ačkoli bylo testování prováděno pouze na platformě linux, simulátor nepoužívá žádnou knihovnu, která by nebyla multiplatformní, výsledný simulátor by tak mělo být možné zprovoznit i na jiných platformách.

Kapitola 5

Závěr

Výsledkem této práce je nový simulátor MCU typu ATmega328p, který je snadno modifikovatelný díky modulárnímu návrhu. Simulátor disponuje grafickým uživatelským rozhraním. Okolí mikrokontroléru je tvořeno třemi modely elektrických obvodů. V rámci práce byla prostudována problematika modelování a simulace mikrokontrolérů a elektrických obvodů. Byly nalezeny a prozkoumány volně dostupné simulátory mikrokontrolérů. Simulátor byl implementován na základě vytvořeného návrhu. Protože byl simulátor vytvořen pro výukové účely, byl kladen důraz na grafické rozhraní.

Simulátor byl implementován v jazyce C++. Mikrokontrolér ATmega328p byl do simulátoru přidán za použití knihovny simavr. Grafické uživatelské rozhraní simulátoru bylo vytvořeno za použití frameworku QT. Nakonec byla funkčnost simulátoru úspěšně ověřena na několika demonstračních aplikacích.

V budoucnu by bylo možné simulátor doplnit o více elektrických obvodů, rozšířit o systém zaznamenávání výstupů mikrokontroléru do souboru nebo možnost zobrazovat výstupní a vstupní hodnoty do grafu.

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] Erich Gamma, R. J., Richard Helm; Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

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

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

[6] Kluknavský, F.: Vliv přesnosti aritmetických operací na přesnost numerických metod. FIT VUT v Brně, 2012.

[7] Lopes, L. C. G.: PICsim - PIC Simulator. [Online; navštíveno 25. 7. 2019].
URL https://github.com/lcgamboa/picsim

[8] Lopes, L. C. G.: PICsimLab - PIC Simulator Laboratory. [Online; navštíveno 14. 07. 2019].
URL https://github.com/lcgamboa/picsimlab

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

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

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

[12] Trygve Reenskaug, J. C.: The DCI Architecture: A New Vision of Object-Oriented Programming. [Online; navštíveno 10. 07. 2019].
URL https://www.artima.com/articles/dci_vision.html