Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Logická/elektronická stavebnice

Plzeň, 2022 Jakub Klimeš

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub KLIMEŠ**
Osobní číslo:	**A19B0250P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Výpočetní technika**
Téma práce:	**Logická/elektronická stavebnice**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte nabídku logických/elektronických stavebnic a jejich prvky. Seznamte se s problematikou modelování a simulace logických prvků.
2. Navrhněte logickou stavebnici založenou na simulaci součástek/logických členů/hradel.
3. Implementujte knihovny pro různé prvky a simulační jádro s detekcí chyb v zapojení.
4. Ověřte funkci na netriviální cvičné úloze a zdokumentujte funkce. Zhodnoťte positiva a negativa tohoto řešení.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Tomáš Mainzer, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **4. října 2021**
Termín odevzdání bakalářské práce: **5. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 14. října 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 3. května 2022

Jakub Klimeš

Abstract

There is plenty of theory behind logical systems, but it is perhaps as important to actually try out such circuits for oneself. Therefore numerous logical kits and simulators are available. This work focuses on merging purely hardware kits with purely software simulators so that the solution acts like a hardware kit, but with additional functionality. This work lists and describes chosen existing products, theoretically analyses the new kit, describes the final implementation including an example, and reviews the product's characteristics as well as compares them with assumptions.

Keywords

Logic systems, logic functions, logic gates, logic kit, microprocesor, Arduino, I2C, simulation, ESP32, STM32

Abstrakt

S logickými systémy je spojeno množství teorie, ale stejně důležité je si logické obvody prakticky vyzkoušet. K tomuto účelu slouží množství stavebnic a simulátorů. Tato práce se zabývá spojením čistě hardwarových stavebnic s čistě programovými simulátory tak, že bude výsledek fungovat jako stavebnice s funkčností navíc. V práci je uveden výběr a popis již existujících produktů, rozebrán teoretický návrh nové stavebnice, její praktická implementace včetně demonstračního příkladu a výsledky jsou zhodnoceny a porovnány s předpoklady.

Klíčová slova

Logické systémy, logické funkce, logické členy, logická stavebnice, mikroprocesor, Arduino, I2C, simulace, ESP32, STM32

Obsah

1 Úvod

Digitální obvody dnes zastávají velkou část veškeré používané elektroniky, především výpočetní technika je založena na dvojkové soustavě s hodnotami 1/0, jejíž vývoj byl také úzce spjat s vývojem spínacích součástek – s postupným vývojem polovodičů v druhé polovině minulého století došlo k poklesu jejich ceny zároveň se zlepšujícími se možnostmi miniaturizace. Výsledkem je, že v podstatě každý nosí denně v kapse několik čipů v podobě mobilních telefonů, platebních karet, flash disků a podobně. Základem těchto zařízení jsou logické členy (hradla) zpracovávající dílčí logické funkce – NOT (inverze), OR (logický součet), AND (logický součin) a XOR (exkluzivní součet). Za pomoci těchto funkcí lze skládat větší celky, jako např. dekodéry, posuvné registry, sčítačky, a z těchto celků lze vytvořit procesor, komplexní stroj na zpracování dat, který je dnes v nejrůznějších provedeních srdcem nejrůznějších zařízení. Samotná hradla jsou dnes vytvořena převážně z komplementárních CMOS tranzistorů (CMOS logika), které postupně vytlačily (ale ne úplně) starší TTL logiku využívající bipolární tranzistory. Historicky byly využívány i relé nebo elektronky.

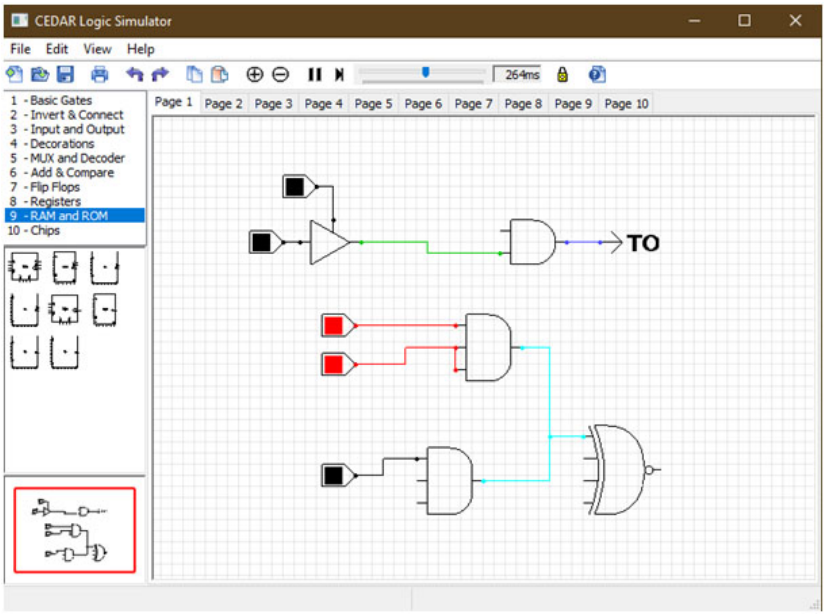
K výuce tohoto odvětví samozřejmě patří množství teorie, ale stejně důležité je si i na vlastní kůži vyzkoušet a experimentovat s funkčními zapojeními logických bloků tak, že je možné si dát teorii s praxí do přímé souvislosti. Právě pro tento účel vznikaly a vznikají nejrůznější logické, ale obecně i elektrotechnické, stavebnice které uživateli umožňují si sestavit, odladit a otestovat obvod podle potřeby. Historicky byly tyto stavebnice hardwarové, tj. uživatel dostal k dispozici skutečné obvody, které realizovaly danou logickou funkci. S rozšířením osobních počítačů se začaly prosazovat i „digitální“ stavebnice, možná lépe řečeno simulátory, ve formě programů. Některé příklady stavebnic, simulátorů a porovnání jejich vlastností jsou dále rozebrány v kapitole 2.

Cílem této práce je teoreticky připravit/navrhnout stavebnici, která bude na pomezí mezi hardwarovou a softwarovou implementací tak, že si z obou světů vezme pokud možno výhody. Konkrétně by měla interně běžet programově jako simulátor, ale pro uživatele by se měla tvářit jako HW stavebnice s některými funkcemi, které by byly pro skutečně HW stavebnici složité na implementaci. Mezi tyto funkce patří detekce nesprávného propojení prvků, změna dynamických parametrů hradel (doba zpoždění) a změna součástkové základny. Jinými slovy by se mělo jednat o simulátor, jehož interface s uživatelem nebude tvořen běžnými periferiemi osobního počítače, ale nějakým zvláštním HW z vnějšku shodným s klasickou stavebnicí.

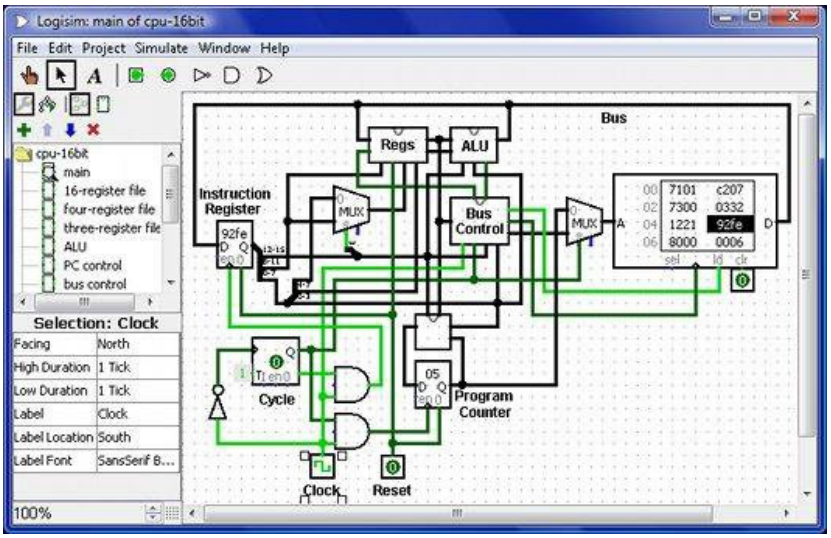
2 Logické stavebnice/simulátory

2.1 Cedar Logic

Cedar Logic [1] je jednoduchý softwarový simulátor logiky, vyvinutý na univerzitě Cedarville. Jeho autory jsou Benjamin Sprague, Matt Lewellyn, David Knierim, Joshua Lansford a Nathan Harro. Obvod se může skládat ze základních logických členů, ale také z pokročilejších obvodů jako RAM paměti nebo registry. Tyto prvky uživatel umístí do pracovní plochy a podle potřeby propojí mezi sebou stylem drag and drop, případně pomocí pojmenovaných virtuálních propojek.
Program během simulace různými barevnými kombinacemi znázorňuje stav vodičů, např. základní logické úrovně vysoká/nízká jsou znázorněny barvami červená/černá, vysoká impedance zeleně, spojení dvou výstupů tyrkysově. Kromě barev je pro sledování průběhů možné použít funkci osciloskopu, který pracuje s virtuálními propojkami a zobrazuje elektrické průběhy obdobně jako skutečný osciloskop. Je také možno měnit rychlost simulace v rozsahu jednotek až několika stovek ms, popřípadě simulaci úplně pozastavit. Simulátor bere v potaz určité zpoždění jednotlivých logických bloků, ale jeho hodnotu nelze měnit.
Výsledný obvod umí program uložit pro další použití. Ukázka programu je na obrázku 1.



Obrázek 1: Hlavní okno programu Cedar Logic.



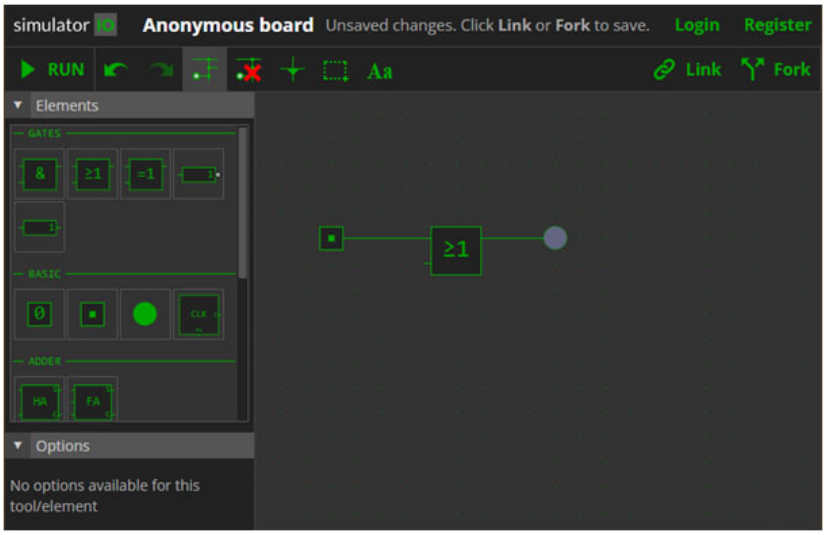
Obrázek 2: Hlavní okno programu Logisim.

2.2 Logisim

Logisim [2] je podobně jako Cedar Logic softwarový nástroj pro simulaci logických zapojení, autorem je Carl Burch z univerzity Hendrix. Rozhraní je na pohled velice podobné simulátoru Cedar logic a stejně tak jsou podobné i základní funkce. Logisim je možná trochu komplexnější, kromě samotné simulace nabízí i možnost analýzy obvodů pravdivostními tabulkami, následnou minimalizaci a automatické sestavení obvodu podle vygenerované funkce. Podporuje i sdružování větších celků pro použití jako jeden ucelený obvod (např. klopný obvod ze dvou explicitních NAND lze sloučit do jednoho bloku „klopný obvod“). Stejně jako u Cedar logic, simulace zahrnuje i zpoždění bloků, lze nastavit i určitou nahodilost jeho délky, ale nelze ho volně upravovat.
Program je napsaný v programovacím jazyce Java, může tedy běžet na různých platformách. Ukázka programu je na obrázku 2.

2.3 simulator.io

Velmi jednoduchý online simulátor simulator.io [3], který poskytuje v podstatě jen základní logické členy, klopné obvody, dekodér a de/multiplexer. Simulace je taktéž jednoduchá, bez rozšířených možností. Výhodou může být přístup přes webový prohlížeč odkudkoli, kdykoli a pro nenáročné ukázky bohatě dostačuje.
Podobných webových simulátorů lze najít větší množství i s více (či méně) funkcemi, např. `https://logic.ly/`, `https://circuitverse.org/simulator`, či `https://sciencedemos.org.uk/logic_gates.php`. Ukázka programu simulator.io je na obrázku 3.



Obrázek 3: Pracovní plocha programu simulator.io, zdroj

2.4 Sparkfun LogicBlocks kit

Sparkfun LogicBlock kit [4] je „tradiční“ HW stavebnice skládající se z jednotlivých modulů, kde každý tvoří jedno hradlo, s LED indikující logický stav nízká/vysoká. Vzájemného propojení je dosaženo fyzickým spojením jednotlivých článků do sebe pomocí připájených konektorů. Jinak stavebnice neposkytuje žádné pokročilé funkce, ale pro seznámení s problematikou může být užitečná.
Podoba stavebnice je zachycena na obrázku 4.

2.5 Voltík

Voltík [5] je česká elektronická stavebnice určená pro děti. Existují tři varianty této stavebnice lišících se obsáhlostí a použitými součástkami. Zatímco Voltík I. neobsahuje o moc více než přepínače, tlačítka a LED a je určen nejmenším dětem bez znalostí elektrotechniky, Voltík II. dává k dispozici základní diskrétní součástky — rezistory, kondenzátory, cívku, tranzistory. Pomyslně nejpokročilejší (a také nejdražší) stavebnicí je Voltík III., který se věnuje především logickým obvodům a k bádání nabízí základní hradla NOT, OR, AND a statickou paměť RAM.
Součástky jsou zakryté a až na výjimky běžně nepřístupné pod pracovní plochou, jednotlivé bloky součástek jsou na ploše viditelně označeny různými barvami. K propojování slouží otvory s vodivými hranami, do kterých se vloží vodič tak, aby se dotýkal hrany a gumovým „špuntem“ se připevní. Výrobce přikládá ke stavebnicím návody s desítkami připravených obvodů, které si může vyzkoušet uživatel, který se chce nechat inspirovat, ale samozřejmě je možno provádět vlastní pokusy podle libosti.

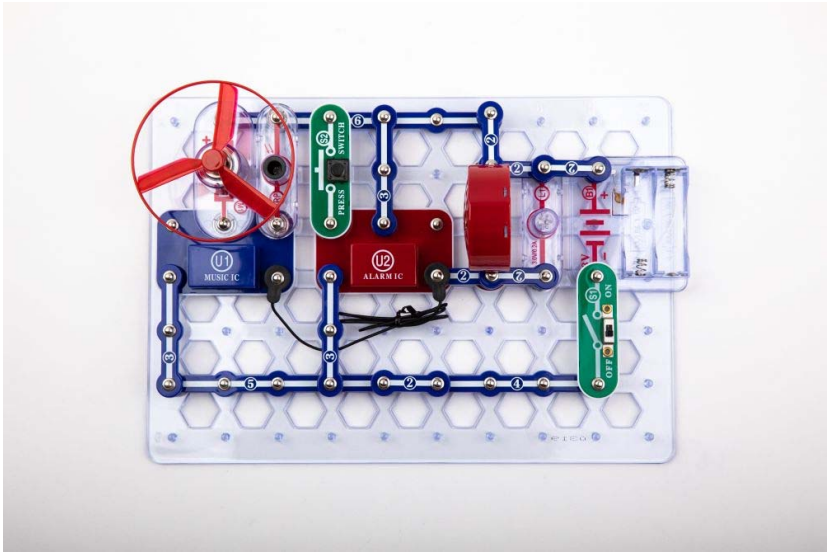
Napájení stavebnice je zajištěno čtyřmi tužkovými bateriemi AA. Podoba stavebnice Voltík II. je zachycena na obrázku 5.

Obrázek 4: Stavebnice SparkFun LogickBlocks kit, zdroj [4].

Obrázek 5: Stavebnice Voltík, zdroj [6].

2.6 Boffin

Podobně jako Voltík je i Boffin [7] stavebnice zaměřená na děti a jejich první setkání s elektrotechnikou, ale nejen jim. Stavebnice se skládá ze stavební

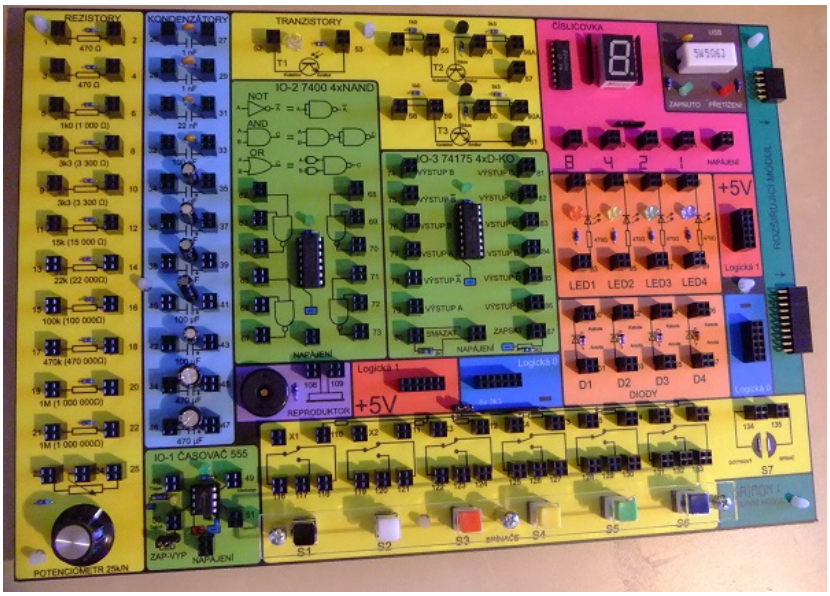


Obrázek 6: Stavebnice Boffin, zdroj [7].

plochy a jednotlivých součástek, které se na plochu uchycují jako „cvočky“. Vytváření obvodů tak není limitováno pevným umístěním součástek jako u Voltíku, mohou se nacházet kdekoliv. Do určité míry mohou být obvody také „3D“, jelikož jako podklad pro součástku není potřeba využívat jen stavební plochu, ale i jiné součástky.
Mezi dodávanými prvky jsou rezistory, kondenzátory, polovodiče, ale i integrované obvody, relé, reproduktory nebo elektromotorky. Samozřejmostí je návod na sestavení předem připravených obvodů, ale nic nebrání vlastní tvořivosti. Pro napájení stavebnice slouží tužkové baterie AA umístěné na stavební desce v samostatných modulech.
Typy stavebnic jsou od sebe rozlišeny jednak typy na Boffin I, Boffin II a Boffin III, přičemž „základní“ stavebnicí je Boffin I, který se dále rozlišuje počtem projektů, které lze z dodaných součástek sestavit (100, 300, 500, 750). Boffin II a Boffin III zachovávají původní ideu a způsob provedení, které dále rozšiřují/specializují, např. na zelenou energii, zvuky, atd.
Zvláštním druhem této stavebnice je Boffin Magnetic, který, jak název napovídá, pro propojení součástek používá magnety namísto mechanických úchytů. Navíc se spolu s touto stavebnicí distribuuje i fix s elektricky vodivou náplní, se kterou lze některé součástky teoreticky i nakreslit.
Podoba základní stavebnice je zachycena na obrázku 6.

2.7 Saimon

Stavebnice Saimon [8] je v mnoha ohledech podobná Voltíku. Jejím autorem je Ing. Lukáš Krejčík, který ji vyvíjel jako výbavu pro elektrotechnické kroužky, inspirován starší stavebnicí Logitronik (2.8). V současnosti je sta-

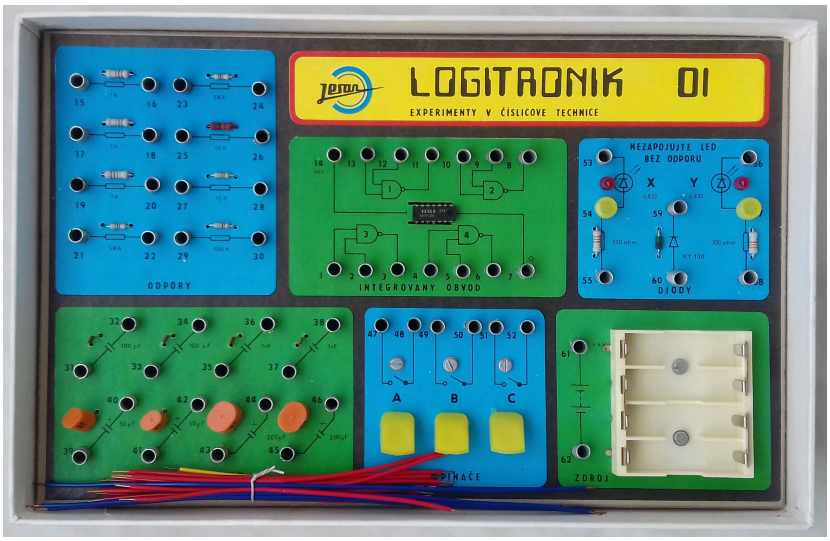


Obrázek 7: Stavebnice Saimon, zdroj [8].

vebnice tvořena čtyřmi bloky Saimon 1 až Saimon 4. Hlavním modulem je Saimon 1, další bloky jsou rozšiřující.
Saimon 1 obsahuje základní elektrotechnické prvky, jako jsou rezistory, kondenzátory, polovodiče, spínače a mimo jiné i čtyři NAND hradla. Jako hlavní modul obsahuje i napájení, zajištěné jedním USB mini B konektorem. Pro připojení dalších rozšiřujících modulů jsou určeny konektory na straně stavebnice. Saimon 2 rozšiřuje součástkovou základnu o další logické obvody AND a OR. Saimon 3 dává uživateli k dispozici paměti, posuvný registr, čítače či displej a Saimon 4 ke stavebnici připojuje Arduino Nano, které je možno dle libosti naprogramovat a použít k řízení dalších obvodů.
Stavebnice je tvořena jednostrannou deskou plošných spojů, ze strany cínu opatřenou průhlednou ochrannou deskou tak, aby byl tištěný spoj dobře vidět. Strana součástek je obdobně jako u jiných stavebnic barevně rozdělena podle typu součástek. Konkrétní součástky lze propojit stejně jako u nepájivých polí.
Podoba stavebnice je zachycena na obrázku 7.

2.8 Logitronik

Logitronik [9] je dnes už poměrně historická stavebnice z 80. let minulého století. Na pohled se velmi podobá Voltíku i Saimonu, ale nabízí poněkud menší rozsah součástek. v případě Logitroniku 1 se jednalo pouze o rezistory, kondenzátory, diody, spínače a čtyři NAND hradla. Logitronik 2, jež měl být nástupcem či rozšířením první verze, rozšířil výběr součástek o další 4 hradla NAND, tranzistory a reproduktor.



Obrázek 8: Stavebnice Logitronik, zdroj [9].

Základní pracovní deska je vyrobena z kartonu, propojení součástek je zajištěno provléknutím vodiče skrz pružinu — do stejných pružin, akorát na opačné straně desky, jsou připojené samotné součástky. Spolu se stavebnicemi byl dodáván i návod, který ale kromě popisů různých zapojení osvětloval i teoretické základy elektrotechniky či číslicové techniky.
Podoba stavebnice je zachycena na obrázku 8.

2.9 littleBits

Zajímavou open-source elektronickou stavebnici littleBits [10] tvoří malé bloky podobné Legu, které se spojují pomocí magnetů. Tyto magnety navíc zabraňují nesprávnému propojení. Nepodařilo se dohledat, jaké všechny prvky lze v této stavebnici použít – prodávané balíčky jsou na oficiálním eshopu propagované stroze jako „pro studenty“, „pro třídy“, apod., bez explicitního vyjmenování obsahu, nicméně ve stejném obchodě lze k zakoupení najít jednotlivé klasické logické členy NOT, AND, OR, stejně jako spínače, motorky, LED, a další.
Podoba stavebnice je zachycena na obrázku 9.

2.10 Shrnutí

Popsané stavebnice a simulátory mají některé věci společné a jiné zásadně odlišné, čímž dávají uživateli možnost si vybrat ideální přípravek pro své potřeby.
Z HW stavebnic byly zmíněny SparkFun LogicBlocks Kit (2.4), Voltík (2.5), Boffin (2.6), Saimon (2.7), Logitronic (2.8) a littleBits (2.9). Všechny tyto stavebnice ze své HW podstaty vydávají své součástky napospas uživateli,

Obrázek 9: Stavebnice littleBits, zdroj [10].

který je může nevhodným použitím poškodit, konkrétně logické členy např. vzájemným spojením výstupů. Stavebnice něčemu takovému může zabránit mechanicky tak, že prostě některé piny nelze spojit s jinými (tak jako v případě littleBits), ale tato metoda problém nedokáže plně odstranit (pokud si uživatel usmyslí, že dva kontakty spojí, tak je může násilím spojit vždy) a nelze ji aplikovat ve všech případech – stavebnice, kde se propojení zajišťuje prostými vodiči (Voltík, Saimon, ...), nemají jak zakázat některé kombinace propojek.

Velkým rozdílem mezi stavebnicemi je, jakým způsobem dávají součástky na výběr. První skupina má součástky pevně zabudované v nějaké základní desce, ze které je (běžný uživatel) nedokáže vyjmout. Mezi tyto stavebnice patří Voltík, Saimon a Logitronic. V podstatě spoléhají na to, že uživatel nalezne vše, co potřebuje, již zabudované. Pokud tomu tak není, nelze přidávat součástky jednotlivě, je nutné přidat celou další stavebnici, případně ke stávající přidat rozšiřující bloky. Tento přístup je výhodný, když uživatel nepočítá s tvorbou rozsáhlých obvodů a spíše tvoří ty, co jsou od výrobce dodány jako vzorové. Pak součástky dojít nemohou a navíc lze stejné zapojení replikovat na jakékoliv stejné stavebnici – je zaručeno, že bude totožná. Nevýhodou je pak již zmíněná nemožnost součástky (snadno) vyměnit. Pokud dojde k nějakému poškození, buď je možno používat jinou součástku stejného druhu (pokud taková existuje), v opačném případě si lze nechat stavebnici opravit (konkrétně Voltík nabízí doživotní záruku), nebo rovnou koupit novou, v obou případech to znamená nějakou časovou prodlevu.

Druhá skupina stavebnic je modulární. Součástky nejsou pevně zabudované, podle libosti je lze buď přidávat, nebo odebírat, či dokupovat další. Do této skupiny patří Sparkfun LogicBlocks Kit, Boffin a littleBits. Mají v podstatě

opačné vlastnosti oproti předchozí skupině. Se stavebnicemi je dodáváno nějaké množství součástek, ale umožňují větší flexibilitu – pokud uživatel potřebuje větší rozsah, může si dokoupit další, ať už jednotlivé nebo celé rozšiřující balení. Stejný postup následuje i v případě poškození. Časová prodleva zůstává, ale není potřeba celý obvod rozebírat, prostě se dotčená součástka vymění. Další výhodou je možnost přímo použít výsledný obvod pro „skutečnou“ aplikaci – sestavený obvod lze prostě přesunout na nějaké další zařízení, které jím má být řízeno a demonstrovat smysl např. logické funkce. Výsledek pravděpodobně nebude esteticky optimální a ne každou stavebnici lze integrovat, ale modulární stavebnice jsou pro tento účel vhodnější. Určitý problém představuje přenositelnost výsledného zapojení. Jiná stavebnice nemusí obsahovat kompletní sadu součástek potřebnou pro replikaci daného obvodu.

Ze simulátorů byly uvedeny Cedar Logic (2.1), Logisim (2.2) a simulator.io (2.3). Hrubě je lze rozdělit na desktopové aplikace (Cedar, Logisim), a webové (simulator.io a další u něj uvedené). Webové aplikace jsou výhodnější v tom, že je není potřeba instalovat a dostupné jsou odkudkoli přes internet a internetový prohlížeč, zároveň mohou podporovat cloudové ukládání schémat nebo spolupráci více uživatelů. Desktopové aplikace bývají rozsáhlejší a poskytují větší výběr funkcí a implementovaných logických členů (i když to nutně není pravidlem). Fundamentální funkčnost všech programů je ale až na drobné odlišnosti ve značení a stylu stejná.

Simulátory by se daly přirovnat k modulárním stavebnicím a mají oproti nim několik výhod. Uživateli mohou dát na výběr (oproti stavebnicím) poměrně pestrou paletu obvodů, jelikož nejsou limitovány cenou součástek a jejich dostupností. Obvody mohou být prakticky neomezeně rozsáhlé, větší celky lze pro lepší přehlednost slučovat (tak jak to nabízí Logisim), v omezené míře lze simulovat zpoždění a z nich plynoucí hazardy (přestože zpoždění obvodů není ani v jednom ze zmíněných programů uživatelsky nastavitelné, je implementované). Přenos nakreslených obvodů je bezproblémový mezi různými stroji se stejným programem. Případné chyby v zapojení, jako již dříve zmíněné spojení dvou výstupů, simulátory vhodně grafickým způsobem zobrazují, stejně jako ostatní stavy, ve kterých se vodiče mohou nacházet (vysoká/nízká úroveň, vysoká impedance, nedefinovaný stav, . . .) a tak dále uživateli usnadňují práci při ladění a obecně zjišťování, v jakém stavu se obvod nachází. Dále nemůže dojít k poruše součástky, která by se v HW špatně odhalovala, např. přerušený vodič.

Obecně simulátory poskytují vyšší míru komfortu při porovnání se stavebnicemi. Všechny zmíněné výhody mohou být paradoxně ale i nevýhody. Hardwarové obvody se dělají právě proto, že jsou hardwarové a oproti softwaru mnohonásobně rychlejší, a k jejich vývoji neodmyslitelně patří nuance, které softwarové simulátory odstiňují – ať už jsou to několikrát zmiňované potenciální poruchy/poškození součástek, ale i samotné fyzické propojování vodičů či ladění obvodu pomocí osciloskopu. Dá se argumentovat, že právě fyzickým

kontaktem s problematikou se člověk nejvíce naučí. Oproti tomu simulátory nejsou nic jiného než jiné počítačové programy, které vytvářejí jakýsi sandbox – uživatel si může dělat, co chce, jak chce, to nejhorší, co se může stát, je, že se obvod rozsvítí nejrůznějšími barvami (v krajním případě aplikace skončí pádem), což je pravděpodobně i důvod, proč stavebnice nebyly zcela vytlačeny programy, ale stále hájí svoje místo na trhu.
Tato problematika samozřejmě není černobílá. Jak už bylo napsáno na začátku této části, každá alternativa se hodí pro trochu jiné použití a je na uživateli, aby zvážil, jaká varianta je pro něj nejvýhodnější. Stavebnice navrhovaná v tomto dokumentu by měla spojit oba světy dohromady a nabídnout stavebnici, která se jako hardwarová chová, interně to ale bude spíše simulátor. Uživatel tak stále bude muset fyzicky propojit jednotlivé součástky, ale nebude hrozit jejich zničení. Přitom lze nabídnout velké množství různých součástek, jak už bylo u simulátorů zmíněno, protože nejsou ovlivněny cenou ani dostupností, jen složitostí implementace. Přináší ale i nové problémy, které se nevyskytují ani u stavebnic ani u simulátorů, tím je především rychlost, s jakou lze zjišťovat stav propojení a s ní související celkový rozsah ve formě množství součástek, které lze obsluhovat, aniž by výsledek byl příliš pomalý na to, aby se mohl tvářit jako skutečná stavebnice.

3 Možnosti implementace

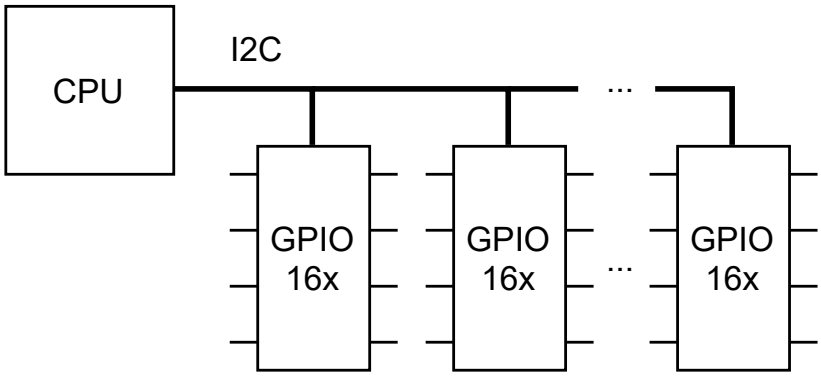
3.1 Procesor

Zásadní otázkou je výběr srdce stavebnice, tj. hlavního procesoru. V ideálním případě by tento procesor měl obsahovat velké množství vstupně výstupních digitálních pinů, které by byly přímo použity jako vývody simulovaných součástek – řádově až několik stovek. Navíc se simulovanými součástkami je vhodné, aby bylo možno se stavebnicí nějak komunikovat navenek, což znamená další piny navíc pro displeje, tlačítka, či LED.
Pro další rozšíření interaktivního potenciálu stavebnice součástkami jako serva, motory, potenciometry bude potřeba i několik analogových pinů stejně jako pinů, které umí pulzně šířkovou modulaci (PWM). Pro potenciální další rozšiřování stavebnice je vhodné, aby procesor poskytoval některé možnosti další konektivity, ať už formou fyzického propojení či bezdrátově a obecně nějakou rezervu ve všech dostupných zdrojích.
Zřejmě nebudou vhodné nejmenší mikroprocesory právě z důvodu omezených zdrojů, na druhou stranu ani nejdražší procesory nedokáží poskytnout několik stovek vstupně výstupních pinů – tento problém bude muset být řešen nezávisle na vybraném procesoru.
Na základě doporučení byl vybrán procesor ESP32, který nabízí široké spektrum dostupných periferií integrované na jednom čipu, jako Wi-Fi, Bluetooth, I2C, SPI a je tedy poměrně flexibilní.

3.2 Rozšíření I/O pinů

Jak už bylo zmíněno, od vlastního procesoru nelze čekat, že nabídne dostatek vstupně výstupních pinů na realizaci stavebnice. Konkrétně ESP32 nabízí 34 víceúčelových vstupně výstupních pinů (GPIO), což by vystačilo jen na 11 třívývodových logických obvodů, navíc by pak tyto piny nebylo možno využít k dalším funkcím (displeje, svítící diody, ...).
Souvisejícím problémem je rychlost, s jakou lze získávat informace o stavu propojení. Čistě HW stavebnice pracují přirozeně paralelně, každý zapojený obvod je nezávislý na ostatních. Programové řešení bude sekvenční, i přitom si musí ale zachovat rozumnou míru responzivnosti – změna v zapojení by se měla na výstupu projevit co nejdříve. Ideální obnovovací frekvenci lze těžko stanovit exaktně, ale rozumným odhadem může být 10 čtení za sekundu jako hranice, kdy už člověk nepozná zpoždění, nebo bude tak malé, že nebude na obtíž. Teoreticky lze frekvenci dále zvyšovat, ale prakticky by to zbytečně zvyšovalo nároky na dobu, jakou trvá samotný přenos a zpracování dat a procesor se kromě zjišťování stavu musí věnovat i samotnému výpočtu logických funkcí.
Řešením je k ESP připojit obvody, které budou poskytovat další GPIO poskytovat (viz dále). Pro jejich připojení byla zvolena sběrnice I^2C pro svou jednoduchost a rozšířenost. Jedná se o synchronní sériovou sběrnici původně vyvinutou Phillips Semiconductor (dnes NXP Semiconductor), posléze převzatou velkým množstvím dalších výrobců integrovaných obvodů. Přenos je zajištěn pouze dvojicí vodičů SDA a SCL operujících jako otevřené kolektory. Zařízení se adresují sedmi bity s možným rozšířením na deset bitů. Plný popis sběrnice je dostupný v uživatelské příručce výrobce [11].

3.2.1 IO extendery



Obrázek 10: Schématické znázornění metody rozšíření GPIO pinů pomocí I/O extenderů.

První možností je použití IO extenderů připojených po sběrnici k jednomu

procesoru, např. MCP23017 [12] od výrobce Microchip. Tento obvod nabízí 16 rozšiřujících pinů rozdělených do dvou portů, přístupných přes sběrnici I2C. Obvod podporuje sekvenční čtení interních registrů bez nutnosti posílat po každém čtení novou adresu. Díky třem adresním pinům lze těchto obvodů na sběrnici umístit až 8, a tedy získat 128 pinů. To je množství, které by vystačilo na 42 třívývodových součástek. Pro porovnání: nejmenší stavebnice Boffin 100 obsahuje 30 dílů, Boffin 300 60 dílů, Voltík III. obsahuje zhruba 140 vývodů. Ani jedno porovnání není příliš přesné, jelikož zmíněné stavebnice neobsahují jen součástky se třemi vývody. V absolutních číslech ale 128 pinů není až tak mnoho. Tímto způsobem nicméně nelze získat víc – resp. lze, ale byla by k tomu potřeba další samostatná sběrnice. Na druhou stranu jedním ze smyslů popisované stavebnice je právě možnost dynamicky podle potřeby měnit součástkovou základnu obdobně jako v programových simulátorech. Počet pinů je stále do značné míry omezující, ale v limitním případě mohou být beze zbytku všechny využity narozdíl od výše zmiňovaných stavebnic, kde pokud některá součástka není pro zapojení potřeba, zůstává nezapojena a její piny nelze jiným způsobem využít.
Tento způsob rozšíření GPIO schematicky znázorňuje obrázek 10.
Testování spojení pinů lze provést nastavením všech pinů jako vstup a připojením interního pull-up rezistoru. Pull-up rezistor je vhodné připojit, protože při použití jako vstup je impedance pinu vysoká a v případě, že je pin nezapojený, by se na něm mohla indukovat téměř jakákoliv úroveň napětí, zkreslující čtení. Následně je jeden pin nastaven jako výstupní s nízkou úrovní napětí, což má za následek pokles napětí na nízkou úroveň i u všech ostatních pinů, které jsou s ním vodivě spojené. Tyto piny už pak zbývá jen najít.
Aby bylo dosaženo výše uvedeného postupu, je potřeba po sběrnici I2C přenést následující povely (uvažuje se sekvenční přístup k registrům):
Adresace interních registrů:

$$1\ \textit{start bit} + 8\ \textit{bitů adresa zařízení a příznak zápisu} + 8\ \textit{bitů adresa registru} + 2\ \textit{potvrzovací bity} = 19 \text{ bitů}$$

Zápis dat po adresaci:

$$\begin{aligned} &8\ \textit{datových bitů} + 1\ \textit{potvrzovací bit po každém zápisu} \\ &\quad + 1\ \textit{stop bit na konci přenosu} \\ &= n \cdot\ 9 + 1 \text{ bitů, kde } n \text{ je počet zapisovaných registrů} \end{aligned}$$

Čtení dat po adresaci:

$$\begin{aligned} &1\ \textit{restart bit} + 8\ \textit{bitů adresa zařízení a příznak čtení} \\ &\quad + 1\ \textit{potvrzovací bit po každém čtení} \\ &+ 1\ \textit{stop bit} = 10 + n \cdot\ 9 + 1 \text{ bitů, kde } n \text{ je počet čtených registrů} \end{aligned}$$

Nastavení obou portů jako vstup:

$$\mathit{adresace} + 2 \cdot \ \mathit{zápis} = 19 + 19 = 38 \text{ bitů}$$

Připojení pull-up rezistorů na obou portech – shodné s předchozím krokem: 38 bitů
Nastavení jednoho pinu jako výstup s nízkou úrovní (v tomto kroku je zároveň opětovně nastaven jako vstupní předchozí pin, pokud byl na stejném portu):

$$2(\mathit{adresace} + 1 \cdot \ \mathit{zápis}) = 2(19 + 10) = 38 \text{ bitů}$$

Čtení vstupních hodnot z jednoho obvodu:

$$\mathit{adresace} + 2 \cdot \ \mathit{čtení} = 2(19 + 29) = 48 \text{ bitů}$$

První nastavení portů jako vstupních a zapojení pull-up rezistorů je nutné provést pro všechny obvody:

$$\mathit{počet\ obvodů} \cdot (\mathit{nastavení\ vstupu} + \mathit{připojení\ pull\text{-}up\ rezistorů}) = 8 \cdot 76 = 608 \text{ bitů}$$

Nastavení výstupu a čtení stavu ostatních pinů:

$$\mathit{počet\ pinů} \cdot (\mathit{nastavení\ výstupního\ pinu} + \mathit{počet\ obvodů} \cdot (\mathit{adresace} + 2 \cdot \mathit{čtení})) = 128(58 + 8(19 + 29)) = 56576 \text{ bitů}$$

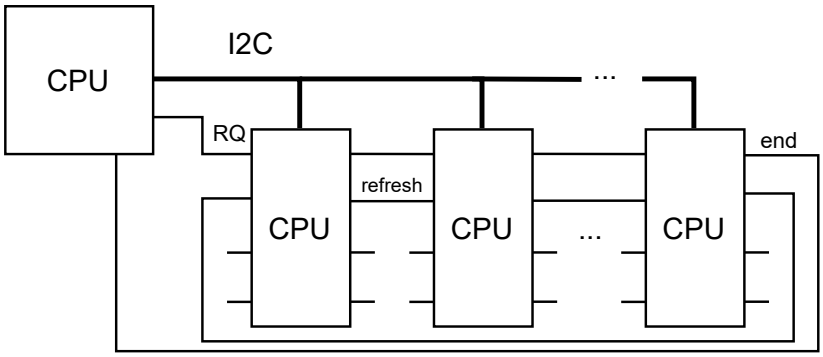
Pokud je pin nastavovaný jako výstupní v jiném portu než předchozí, je potřeba předchozí explicitně nastavit jako vstup, těchto pinů je o 1 méně než množství portů (každý obvod má dva porty):

$$\mathit{počet\ pinů} \cdot (\mathit{adresace} + 1 \cdot \mathit{zápis}) = 15(19 + 10) = 435 \text{ bitů}$$

Celkem:

$$\mathit{inicializace} + \mathit{nastavení\ výstupu\ a\ čtení} + \mathit{dodatečné\ nastavení\ vstupu} = 608 + 65576 + 435 = 57619 \text{ bitů}$$

Při přenosové rychlosti 400 kB/s je doba přenosu zhruba 0,14 s. Pokud je cílem stav propojení číst zhruba desetkrát za sekundu, tento přístup by byl na hraně, a to pouze za předpokladu, že se centrální procesor nemusí věnovat ničemu jinému, než jen periodickému obnovování propojek. Reálně to tak bohužel nebude, společně se čtením vývodů je potřeba počítat vlastní logické funkce propojených prvků a řešit i případné další vstupně výstupní periferie jako tlačítka a displeje. Potenciálně by bylo možné využít jinou (rychlejší) sběrnici, např. SPI, která ale vyžaduje větší množství vodičů. Samotná sběrnice I2C podporuje i větší přenosové rychlosti (až do řádů Mb/s), nicméně nejvyšší použitelná je právě 400 kb/s kvůli procesoru ESP32.



Obrázek 11: Schématické znázornění metody rozšíření GPIO pinů pomocí mikroprocesorů.

3.2.2 Mikroprocesory

Druhou možností, jak rozšířit počet GPIO pinů, je místo dedikovaných IO extenderů použít mikroprocesory, které se jako IO extendery budou chovat. Bylo by teoreticky možné každému mikroprocesoru přidělit vlastní adresu a pracovat s nimi v podstatě stejným způsobem jako s IO extendery (popsaným v kapitole 3.2.1). Procesory umožňují volně konfigurovat své adresy, čímž by jich bylo možné na sběrnici umístit oproti např. zmiňovanému MCP23017 větší množství. Nicméně nahrazením extenderů za plnohodnotné procesory získává stavebnice nezanedbatelné množství výpočetního výkonu navíc, který lze využít k rychlejšímu přenosu, respektive úspoře přenášených dat.
Pro tuto úlohu se jako nejvhodnější jeví pověřovací systém, možné zapojení znázorňuje obrázek 11. Hlavní procesor v této konfiguraci dává pouze jedním bitem znamení, že si přeje zjistit stav propojení. Mikroprocesory mají stejně jako IO extendery v základním stavu porty nastavené jako vstupní s připojeným pull-up rezistorem. Vysílání je založeno na předávání pověření tak, aby v jednu chvíli nemohl vysílat více než jeden mikroprocesor.
Po detekci aktivního signálu RQ první mikroprocesor v řadě nastaví jeden ze svých pinů jako výstup s nízkou úrovní a na sběrnici posílá stav svých zbylých vstupních pinů. Po odeslání dalšímu mikroprocesoru předává signál refresh jako znamení, že má obvod pouze odeslat stav vstupních pinů a nenastavovat aktivně některý svůj pin jako výstupní. Jakmile signál refresh proběhne celou řadou, dostane se zpět na první mikroprocesor, ten posune aktivní pin a celý cyklus běží znovu. Po vyčerpání všech pinů první mikroprocesor nastaví na aktivní úroveň pin RQ k následujícímu mikroprocesoru, ten díky tomu ví, že má sám aktivně nastavovat výstupní piny. Po tom, co signál RQ doputuje až k poslednímu mikroprocesoru a ten odešle svá data, signál refresh dále neposílá a naopak generuje signál end pro hlavní procesor, čímž dává najevo, že čtení pinů bylo dokončeno. Nevýhodou je smyčka v tomto zapojení a tedy potřeba mít dva odlišné programy pro poslední mikroprocesor a všechny

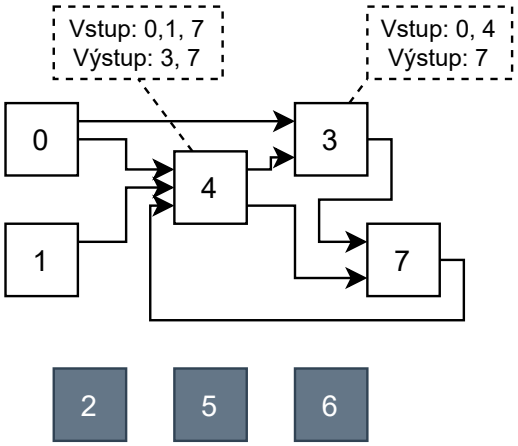
ostatní mikroprocesory.
Je patrné, že hlavnímu procesoru toto řešení usnadňuje práci, jelikož nemusí sám řešit nastavování portů, výstupních pinů a aktivně se portů dotazovat na stav. Tím také odpadne velká část komunikace, po sběrnici I2C budou putovat výhradně informace o stavu vstupů, tj. v nejjednodušším případě matice 128 krát 128 bitů (při zachování stejného množství pinů jako v případě s IO extendery). Už to je poměrně výrazná úspora, díky které by bylo možné vzorkovat desetkrát za sekundu.
Dá se navíc předpokládat, že přenášená matice bude řídká – určitě nedojde k propojení každého pinu se všemi ostatními. Některé mikroprocesory pak můžou odesílat nulová data (informaci, že daný pin není spojen se žádným jiným, který má daný mikroprocesor k dispozici), a tak v podstatě zbytečně prodlužovat dobu potřebnou k přenesení. Místo toho by prostě nemusel odesílat nic a předat pověření dále. Pro realizaci tohoto principu je nutné k přenášeným datům přidat nějakou identifikaci konkrétního vysílače, aby hlavní procesor mohl správně přijatá data zpracovat, např. tak, že v první odesílané zprávě bude jeho index a až v dalších zprávách samotná informace o propojení. Tím samozřejmě dojde ke zvětšení objemu dat odesílaných každým mikroprocesorem, který by měl být ale vykompenzován právě přeskočením vysílačů, které nemají žádná vhodná data k odeslání.
Dalším možným vylepšením je posílání pouze informací o změnách – tedy prvním krokem by bylo odeslání kompletní matice, následované periodicky už jen dílčími informacemi. Tímto způsobem by bylo možné ušetřit další množství přenášených dat, ale za cenu toho, že by musel každý mikroprocesor mít uložený stav „své“ části matice a pokaždé ho porovnávat s novým. Vzhledem k tomu, že by už předchozí uvedené vylepšení mělo stačit pro docílení požadovaných vlastností, může tato varianta zůstat nevyužita.
Obecnou výhodou použití mikroprocesorů oproti IO extenderům je možnost podle potřeby rozšiřovat množství pinů – počet mikroprocesorů na sběrnici není omezen množstvím adresních vodičů, a zároveň potenciální úspora velkého množství přenášených dat, a tedy i urychlení.
Nevýhodou je potřeba dražšího HW, rozšířenějšího programového vybavení (kromě hlavního procesoru je potřeba naprogramovat ještě rozšiřující mikroprocesory) a potenciální režijní zabrání dalších pinů mikroprocesorů kromě samotného rozšíření IO (v navrhované variantě signály RQ, refresh, end).

3.3 Logické bloky

Logické bloky muset být v paměti reprezentovány datovou strukturou, která bude uchovávat všechny informace potřebné k jejich funkci, např. typ obvodu (OR, AND, ...), počet vstupů, počet výstupů, pole se vstupními a výstupními hodnotami, odkazy na bloky předcházející a následující. Pro realizaci časového zpoždění by mohla přibýt hodnota tohoto zpoždění spolu s informací, kdy byl výstup naposledy změněn.

3.3.1 Vyhodnocování funkcí

Procesor pak bude v nejjednodušší variantě cyklicky procházet pole se všemi logickými bloky a na základě vstupních hodnot vypočítávat nové výstupní. Výstupní hodnota se nezmění, pokud doba od poslední změny nepřekročila hodnotu zpoždění. Je to jednoduchá metoda, která vůbec nezohledňuje, že ne všechny bloky musí být v jednu chvíli používané a že by bylo tedy možno častěji obnovovat skutečně využité bloky a naopak vůbec neřešit ty nevyužívané.
Nástavbou nad tímto způsobem výpočtu je tvorba grafu nad logickými bloky. Každý blok tvoří jeden vrchol, hrany je možno reprezentovat pomocí odkazů, které si každý blok uchovává. V ideálním případě by se jednalo o strom, ale v tomto případě se bude jednat o obecný graf. Reálné zapojení může obsahovat cykly a dá se očekávat, že bude taktéž obsahovat více než jeden kořen – výchozími vrcholy budou bloky s pevně nastavenými hodnotami 0/1, případně další vstupní prvky.

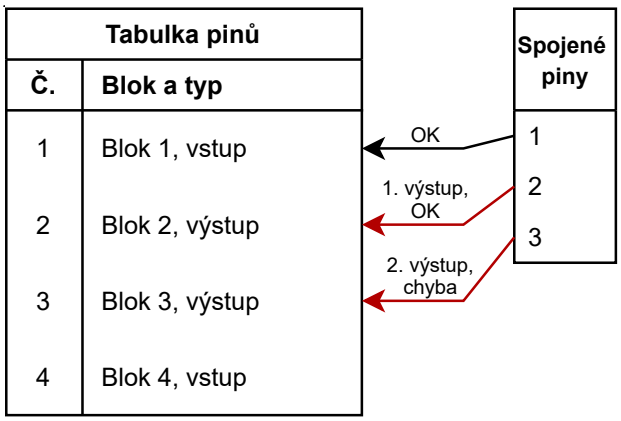


Obrázek 12: Graf z logických členů.

Z počátečních vrcholů lze spustit algoritmus prohledávání do šířky (BFS) – do fronty vrcholů jsou zařazeny všechny vrcholy, jejichž vstup využívá výstup jednoho z kořenů, následně algoritmus BFS probíhá standardně, přičemž pořadí, ve kterém jsou vrcholy zpracovány, je zároveň i pořadí, v jakém se budou následně ve smyčce přepočítávat výstupní hodnoty každého logického bloku. Navíc toto řešení eliminuje bloky, které nejsou k ničemu připojené a není potřeba se jimi zabývat. Nevýhodou je, že je graf (resp. seznam bloků) potřeba vytvořit po každé aktualizaci propojení pinů a z toho plynoucí zpoždění. Toto zpoždění by nicméně nemělo být příliš zásadní, jelikož bloků nebude tak velké množství. Schématické znázornění je na obrázku 12 - naznačené bloky 2, 5 a 6 jsou nevyužité, a tedy nejsou součástí grafu. Oproti předchozí variantě s prostým polem se interně logické bloky nezměnily, obsahují stále stejné informace, odlišný je jen přístup k nim.

3.3.2 Vyhodnocování chyb zapojení

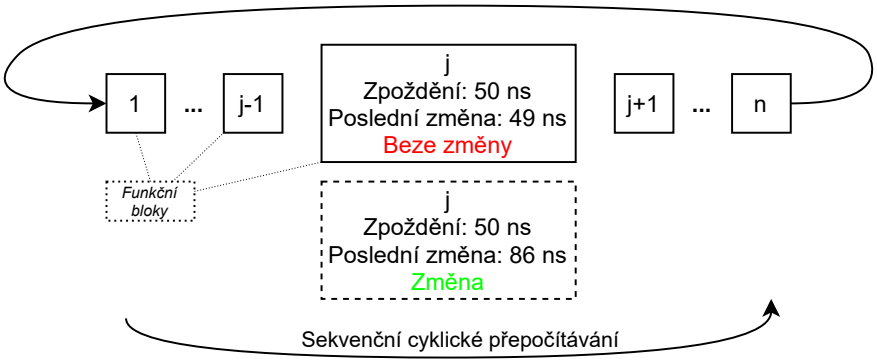
U „skutečných“ logických bloků nelze přímo vodivě spojovat dva výstupní piny – při stejných výstupních úrovních by se teoreticky nemuselo nic stát, ale pokud budou stavy rozdílné, mezi bloky poteče téměř zkratový proud, který logické členy zlikviduje. HW řešením je oddělení pomocí třístavových obvodů. Obdobným, i když ne vyloženě kritickým problémem, jsou u HW členů nezapojené vstupy. Protože jsou vstupy typicky vysokoimpedanční, jejich nezapojení z nich vytvoří anténu, na kterou se může cokoli indukovat. Následkem je, že je daná logická operace nedefinovaná (vstup může nabývat libovolné úrovně). U simulovaných logických prvků tyto výsledky implicitně nehrozí, zároveň ale nedává smysl tato propojení akceptovat a lze rovnou při jejich detekování ohlásit chybu. Problém s nezapojenými vstupy je možno detekovat při průchodu polem logických bloků a výpočtu nových výstupních hodnot: pokud se narazí na blok, který má nějaký vstup nezapojený, je to chyba v zapojení.
Problém se spojenými výstupy je možno přímočaře řešit dalším průchodem logickými bloky a kontrolou, zda výstup testovaného bloku nevede na výstup jiného. Řešení je to funkční, ale poměrně zdlouhavé. Rychlejší metodou je zavedení tabulky pinů, která v každé řádce uchovává informaci o tom, kterému logickému bloku pin náleží a zda je vstupní, či výstupní. Pak je možné detekovat chybu hned při příjmu informací o propojení, které budou tvořeny množinou pinů (či jejich indexů), navzájem vodivě spojených. Tyto indexy lze použít jako klíč do tabulky a zjistit, zda je mezi spojenými piny nanejvýše jeden výstup, tak jak to zobrazuje obrázek 13. Pokud ne, zapojení je chybné, v opačném případě se pokračuje s vyhodnocováním dalšího propojení.



Obrázek 13: Použití tabulky pinů.

3.3.3 Simulování zpoždění

Výsledná simulace pak může běžet ve „zjednodušeném“ režimu bez uvažování zpoždění jako jednoduchá logická struktura, nebo naopak s uvažováním zpoždění jednotlivých členů, kde se může potenciálně projevit větší množství přechodových jevů a s nimi spojených hazardů. V prvním případě se nové výstupní hodnoty umístí na výstup bloku okamžitě. V druhém případě bude navíc kontrolován údaj o čase poslední změny výstupu. Tím, že bude vykonávání sekvenční, ale nebude požadovaná doba zpoždění nikdy dodržena přesně, bude se jednat spíše o minimální dobu zpoždění – výstup nebude nikdy změněn dříve než její hodnota, ale později s nějakou náhodnou odchylkou, závislou na době průchodu skrz všechny bloky a na dalších činnostech, kterým se CPU bude věnovat. Na vymyšlené situaci to ilustruje obrázek 14.



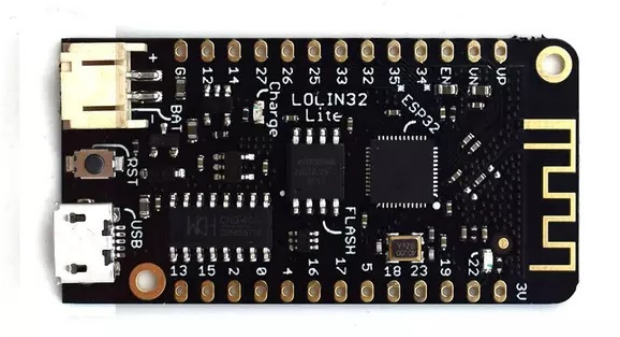
Obrázek 14: Příklad doby zpoždění funkčního bloku.

4 Použitý hardware

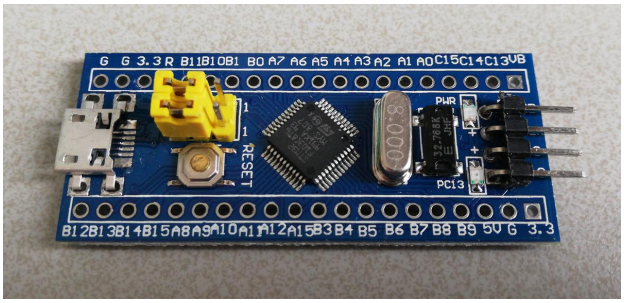
ESP32 WeMos LOLIN32 Lite Vývojová deska s procesorem ESP32, která bude sloužit jako základ stavebnice. Procesor lze k počítači připojit a programovat přes zabudovaný Micro USB konektor a USB/Serial převodník. USB běžně slouží i pro napájení, ale lze využít i LiPo baterii připojenou přes vyhrazený konektor, která se v případě napájení přes USB nabíjí. Podoba desky je zachycena na obrázku 15.

STM32F103C8T6 BluePill Vývojová deska s procesorem STM32, určená pro rozšíření IO pinů po sběrnici I^2C. Micro USB konektor slouží výhradně pro napájení, programování probíhá přes dedikovaný programátor. Deska je zachycena na obrázku 16.

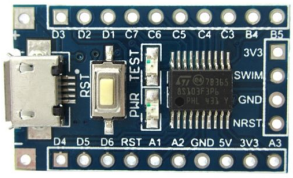
STM8S103F2 Vývojová deska s procesorem STM8, určená pro rozšíření IO pinů po sběrnici I^2C. Stejně jako u předchozí desky k napájení slouží Micro USB konektor, k programování je potřeba dedikovaný programátor.



Obrázek 15: WeMos LOLIN32 Lite, zdroj [13]



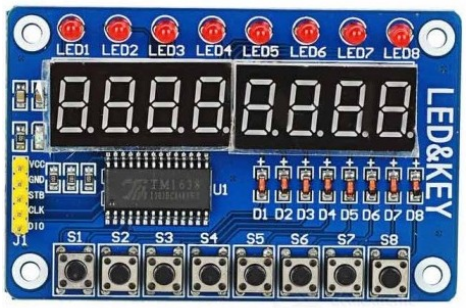
Obrázek 16: STM32F103C8T6 BluePill, zdroj [14]



Obrázek 17: Vývojová deska s STM8, zdroj [15]

ST-LINK V2 Programátor pro procesory STM32 a STM8, připojitelný k počítači přes USB.

LED & KEY modul Deska s osmi sedmisegmentovými LED displeji, osmi samostatnými LED a osmi tlačítky, řízená integrovaným obvodem TM1638. Tato deska bude sloužit pro interakci stavebnice s uživatelem. Komunikace s řídícím procesorem probíhá pomocí tří vodičů (strobe, clock, data input/output), napájení zajišťuje dvojice VCC a GND. Pro ovládání desky bude využita knihovna `TM1638plus`, jejíž autorem je Gavin Lyons. Kompletní dokumentace je k dispozici v [16].



Obrázek 18: Deska LED & KEY s obvodem TM1638, zdroj [16]

Piezoelektrický reproduktor a zesilovač PAM8403 Pro akustický výstup ze stavebnice. Desku s integrovaným zesilovačem PAM8403 lze použít, pokud by intenzita zvuku po připojení přímo na GPIO pin ESP nebyla dostačující. Zesilovač podporuje dva kanály (levý a pravý), při použití pouze jednoho reproduktoru bude využit pouze jeden z nich. Datasheet PAM8403 je dostupný z [17]. Vstupní signál bude generován pomocí PWM driveru integrovaného v ESP. K jeho ovládání slouží knihovna `ledc` poskytovaná k Arduino frameworku, dokumentaci lze nalézt v [18].

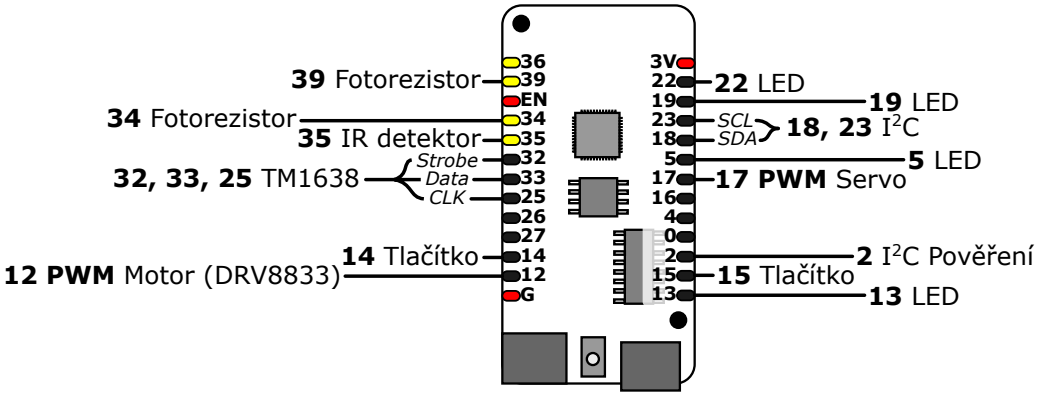
Stejnosměrný motor a driver DRV8833 Pokud by bylo potřeba stavebnicí řídit klasické stejnosměrné motory, kvůli velkým odběrům je nelze napájet přímo z IO pinů ESP. Pro tento účel slouží motor driver DRV8833, jehož datasheet lze nalézt v [19]. Obsahuje dva H-můstky pro řízení dvou klasických motorů, případně jednoho krokového motoru. Jeho vstupem bude PWM signál opět generovaný interními PWM drivery ESP přes knihovnu `ledc`.

Servo Klasické servo řízené PWM signálem o frekvenci 50 Hz. Pro ovládání bude do třetice využito PWM schopností ESP.

Ostatní Další diskrétní součástky: LED, fotorezistory, tlačítka, ...

4.1 Rozvržení připojení součástek na ESP

ESP32 umožňuje díky multiplexerům a demultiplexerům připojit cokoliv téměř na jakýkoliv GPIO pin. Nicméně pro některé GPIO existují omezení (zejména během bootovací fáze), kvůli kterým může být jejich použití pro určité aplikace nevhodné, a některé piny jsou pouze GPI – tedy výhradně vstupní. Přehledným způsobem to zobrazuje článek [20].
Pozn.: Různé vývojové desky nemusí mít vyvedené všechny piny samotného čipu, konkrétně WeMos LOLIN32 Lite nemá vyvedené např. piny 1, 3 sloužící pro sériovou linku, ani piny 6 až 11, sloužící pro připojení paměti.
Výsledný návrh rozvržení součástek je na obrázku 19. Piny vybarvené žlutě jsou pouze vstupní, červeně vybarvené piny neslouží jako GPIO.



Obrázek 19: Návrh rozvržení součástek

5 Popis softwarové implementace

5.1 Logika stavebnice - ESP32

Program pro vlastní stavebnici se skládá ze zdrojových souborů `main.cpp`, `main.h`, `logic.cpp`, `logic.h` a `function_map.h`. Program je napsán pod Arduino frameworkem pro ESP32, jehož oficiální dokumentaci lze nalézt ve zdroji [18].

Definice součástek Je potřeba zajistit předávání definic použitých součástek do programu. Nejpřímějším způsobem je zápis přímo do dedikovaného zdrojového souboru. Tento způsob není vhodný, protože obecně není příliš praktický: spolu se samotnými součástkami je potřeba zapisovat další data a formátování navíc, aby daný soubor zvládnul překladač přeložit a nahrát,

a také proto, že by se změnou i jedné jediné součástky byl zbytečně opětovně nahráván celý program.
ESP32, stejně jako jiné mikrokontroléry s rozsáhlejší vnitřní pamětí, podporuje zavedení lokálního filesystému (SPIFFS – SPI flash file system) – část integrované flash paměti se vyhradí pro tento file systém a dále se s ním pracuje téměř identicky jako s filesystémy na desktopových počítačích, jen je jeho velikost výrazně omezená (navíc případně závislá i na velikosti samotného programu a využívání over-the-air aktualizací). Pro účely stavebnice, kde budou potřebné definice jednoduché plaintextové soubory, však bohatě dostačuje. Do tohoto filesystému jsou soubory nahrávány stejně jako program, jen bohužel ne jednotlivě, ale vždy všechny potřebné naráz.

Nahrávání souborů Pokud je ESP a především jeho USB konektor volně dostupný, nahrávání dat probíhá standardně tímto konektorem. Nicméně vzhledem k povaze stavebnice je předpokládáno její zabudování do nějaké krabičky tak, že USB konektor nebude dosažitelný. V takovém případě lze s výhodou využít wifi zabudované přímo v ESP32 k provádění tzv. over-the-air (OTA) aktualizací. V zásadě se nejedná o nic jiného než o nahrávání programu přes bezdrátovou síť místo fyzického propojení. Možností jak OTA provádět je více – ESP může přes internet samo přistupovat na web, ze kterého si stáhne nové verze souborů, může vytvářet vlastní přístupový bod, na který se připojí druhé zařízení, které ESP zašle nové soubory, nebo se ESP může připojit na již existující síť, přes kterou k němu přistoupí druhé zařízení a nahraje data.
Síťování není hlavním předmětem této práce, takže byl převzat fungující příklad z [21]. Odkazovaný příklad funguje na posledním zmíněném principu v předchozím odstavci, tj. ESP se připojí na existující lokální síť a vytvoří HTTP server, přes který je možné zasílat aktualizace. Nevýhodou je, že přihlašovací údaje do sítě musí být uvedeny explicitně v kódu a před každým přesunem do jiné sítě je potřeba je aktualizovat, jinak se ESP na síť nepřipojí a OTA není možné využít. Druhou nevýhodou je, že pro připojení na HHTP server ESP je potřeba znát jeho IP adresu, která v sítích s dynamickým přidělováním může být pokaždé jiná. Obecným důsledkem používání OTA je další zmenšení úložného prostoru na integrované paměti – pokud by přes OTA došlo k nahrání chybného programu, už by nemuselo být možné se k ESP opětovně připojit k nahrání opravy, což je v případě fyzické nedostupnosti čipu problematické. V paměti jsou tedy vyhrazeny dva oddíly pro program. OTA aktualizace se provede vždy v oddílu, který není využíván pro běh programu a pokud se zjistí, že je nově nahraný program problematický, automaticky se přejde na předchozí funkční.

I²C a procesory rozšiřující IO Návrh implementace tak, jak byl popsán v kapitole 3.2.2, počítal s připojením ESP na sběrnici jako slave a rozšiřujícími procesory jako master, nicméně na upozornění vedoucího práce, že slave mód není na ESP implementován korektně, musely být role obráceny. Důsledkem je, že ke správné funkci stačí pouze pověřovací signál, na obrázku 11 označen na výstupu z hlavního procesoru jako RQ.
Problémem posledních let je nedostatek či nedostupnost čipů. Důsledek pro tuto práci je takový, že nebylo možné sehnat uspokojující množství procesorů, které by sloužily pro IO extend. Vedoucí práce pro vyzkoušení a prokázání alespoň proof-of-concept poskytnul dva procesory: STM32 a STM8 (viz kapitola 4). Kromě dalších aspektů jsou tyto dva čipy odlišné množstvím dostupných portů, ale i softwarovou podporou – zatímco pro STM32 je Arduino framework, především jeho část starající se o I²C, plně implementován, pro STM8 je podpora výrazně horší a I²C je implementováno pouze pro master mód, tedy pro účely stavebnice nevhodně. Bohužel celým smyslem programování pod tímto frameworkem bylo, že budou programy mezi různými platformami přenositelné, čímž se tato myšlenka stala nereálnou (alespoň pro platformy STM32 a STM8).
Při snaze implementovat komunikaci na STM32 bylo naraženo na anomálii: framework nějakým způsobem znemožňoval (nebo se autorovi nezdařilo) zapisovat přímo do registrů mikroprocesoru pro nízkoúrovňové řízení periferií. Implementaci to zkomplikovalo ve dvou případech:

1. Všechny procesory pro rozšíření IO mají stejnou I²C adresu, kvůli tomu si periferie, obsluhující sběrnici, „myslí“, že je každá zpráva jen pro ni, což není zcela pravda. Arduino framework pro obsluhu definuje jen dvě funkce vyvolané přerušeními (při přijetí dat a vyžádání dat masterem), ve kterých není možné na základě pověření „vysokoúrovňově“ periferii odstavit/resetovat. A při opuštění funkce, aniž by došlo k předání dat pro odeslání, periferie udržuje na sběrnici nízké CLK, čímž zabraňuje jakýmkoliv dalším přenosům. Periferie umožňuje toto chování vypnout nastavením registru I²C Control Register 1 (viz reference manuál [22], str. 772, případně obrázek 20), konkrétně bitu 7 NOSTRETCH, případně lze bitem 0 PE celou periferii odstavit a zapnout jen v případě, že má daný obvod pověření. Protože ale nebylo možné k registrům přistoupit a Arduino samo o sobě tyto funkce nepodporuje, nebylo možné je realizovat.

2. Po neúspěchu s využitím periferie byl proveden pokus řešit I²C protokol čistě programově (tzv. „bitbanging“). Obecně se používat nedá, jelikož se jedná o v podstatě nejneefektivnější způsob využití výpočetního výkonu procesoru, ale v tomto konkrétním případě mají procesory jen jednu konkrétní úlohu, kterou je právě I²C. Tímto přístupem byl úspěšně naprogramován jednoduchý program pouze vysílající potvrzení ACK přijetí své adresy, ale vznikl nový problém: maximální frek-

26.6.1 I²C Control register 1 (I2C_CR1)

Address offset: 0x00
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res.	SMBUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

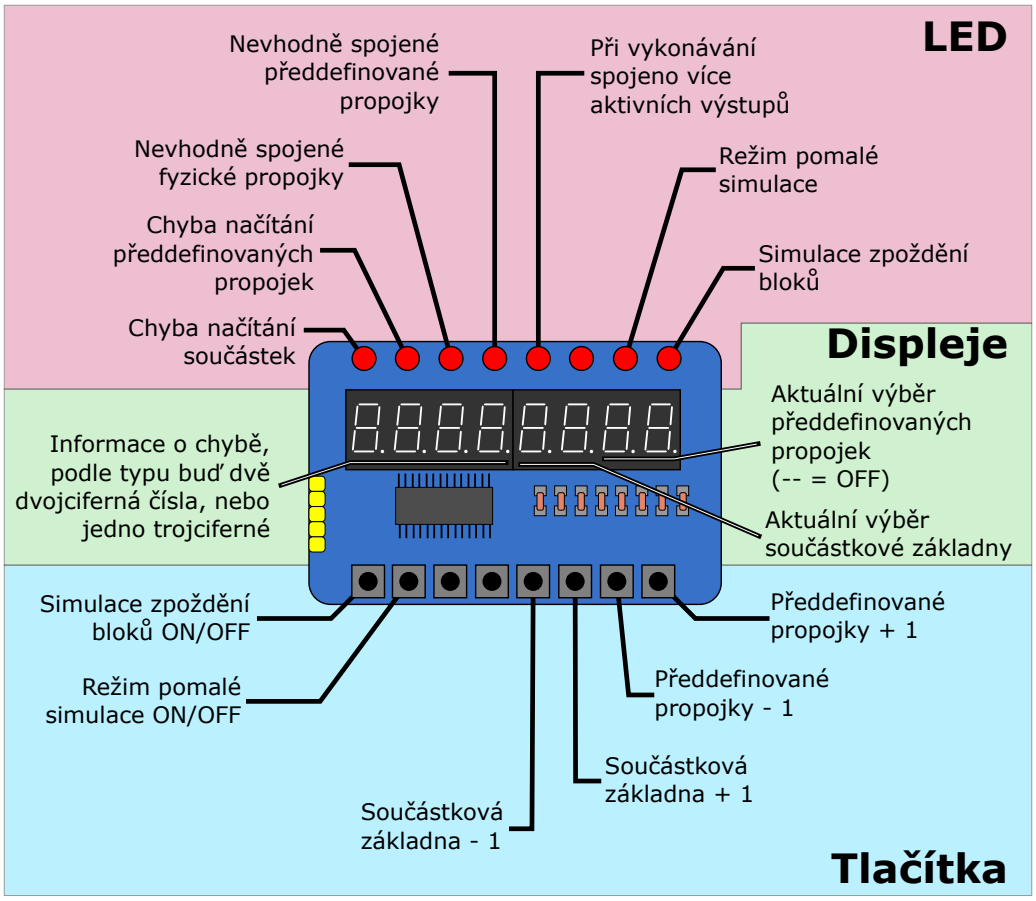
Obrázek 20: I2C Control Register 1, zdroj [22]

vence, při které byl procesor ještě schopný programově přistupovat ke sběrnici, se pohybovala okolo 150 kHz – jak ESP32 tak STM32 přitom (samozřejmě přes dedikované periferie) podporují Fast-mode, tj. až 400 kHz. Zrychlení programu by bylo jistě možné nízkoúrovňovými přístupy k ovládání pinů, ale opět kvůli nemožnosti přímého zápisu do řídících registrů nerealizovatelné.

Řešením nakonec bylo využití periferie procesoru, ale s omezením, že všechny procesory vysílají zároveň. Toto je možné díky fyzické vrstvě sběrnice I²C, jejíž dva vodiče pracují v režimu s otevřeným kolektorem – v klidovém stavu je vodič udržován přes pullup rezistor na úrovni kladného napájecího napětí, uzly účastnící se komunikace nedělají nic, pokud chtějí vysílat vysokou úroveň, a pokud chtějí vysílat nízkou úroveň, vodič spojí se zemí. Pokud budou tímto způsobem všechny vysílače s výjimkou jednoho vysílat slovo skládající se výhradně z „jedniček“, ten jeden konkrétní vysílač může stále výslednou hodnotu libovolně měnit. Smysl ale ztrácí acknowledge bity, které každý obvod vyšle, pokud detekuje, že byl adresován. Pro účely stavebnice se bez nich dá obejít.
Výhodou je, že bude mít každý procesor přehled o tom, kolik čtení na sběrnici proběhlo. Při zavedení inicializační fáze, kdy se navíc každému procesoru sdělí jeho pořadí, pak už není potřeba předávat pověření, jelikož všichni ví, kdy mohou a nemohou vysílat.

Interakce s uživatelem Jak už bylo naznačeno v kapitole 4, pro interakci s uživatelem slouží modul LED & KEY s integrovaným obvodem TM1638. Destička poskytuje LED diody, které jsou použity pro signalizaci chybových stavů a zapnutí/vypnutí časových módů stavebnice.
Levá polovina sedmisegmentových displejů zobrazuje případné další údaje k chybě (v případě chyb při načítání ze souborů jsou to dvě dvouciferná čísla – číslo řádky a číslo prvku, u kterého došlo k chybě, u ostatních chyb je zobrazeno jedno až trojciferné číslo, udávající pin, u kterého došlo k chybě). Pravá polovina displejů zobrazuje dvě dvouciferná identifikační čísla použitých souborů pro definice součástek a předdefinované propojky. Pokud nejsou propojky použity, jsou zobrazeny dvě pomlčky (k vypnutí dojde při snížení čísla souboru při aktuálním čísle 00).
K samotnému ovládání stavebnice slouží osm tlačítek.

Graficky jsou výše uvedené informace zobrazeny na obrázku 21.



Obrázek 21: Ovládání stavebnice přes desku LED & KEY

5.1.1 Vysokoúrovňový popis

Celý program je v podstatě postaven na třech datových strukturách – `block` reprezentující funkční bloky stavebnice, `pin` reprezentující vstupy/výstupy bloků a `wire` reprezentující propojky mezi piny (potažmo i bloky). Jejich popisy jsou v 5.1.2.
Funkční bloky jsou před startem vykonávání navzájem zaměnitelné. Jejich funkce je uživatelsky definována zpracováním souboru s definicí součástek (viz 5.1.6) – na základě tohoto souboru je každému bloku přiřazen odkaz na jeho vykonávanou funkci. Tato funkce může být jednak čistě logická, ale i vstupně výstupní (tlačítka, LED, ...) – viz 5.1.3. Druhým smyslem vstupního souboru je s konkrétním blokem asociovat fyzické piny stavebnice. Piny slouží jako spojovací prvky mezi funkčními bloky a propojkami mezi nimi.
Jakmile jsou nastaveny funkční bloky, získají se propojky. Tyto propojky jsou snímány periodicky přímo z fyzických výstupů stavebnice (zajišťova-

ných přídavnými mikroprocesory, viz 5.1.6), případně je možno propojky definovat „staticky“ pomocí souboru, podobně jako funkční bloky (viz 5.1.6). Datová struktura `wire` si udržuje pouze identifikátory pinů. Získaná propojka (nezávisle na zdroji) se jako množina propojených pinů předává funkci, která specifikované piny virtuálně propojí (viz 5.1.6). Pokud by to bylo potřeba, dojde ke spojení více již existujících propojek (viz 5.1.6). Program neobsahuje žádný způsob jak propojky odebrat – místo toho se vždy před aktualizací provádí návrat do výchozího stavu.
Po nastavení stavebnice se cyklicky volají funkce vykonávané funkčními bloky, resp. obalující funkce zajišťující navíc transformaci hodnot (viz 5.1.3) a korektní zápis zpět do struktur stavebnice (viz 5.1.3).

5.1.2 `logic.h`

Tento soubor se věnuje hlavně simulované stavebnici jako takové, lze ho rozdělit na tři části: definice konstant, definice datových typů a hlavičky logických funkcí.
První jsou definice nejrůznějších konstant použitých na různých místech programu. Především jsou to čísla pinů ESP, na které jsou připojené fyzické periferie stavebnice (I^2C, tlačítka, LED, ...) a další případné informace potřebné k jejich konfiguraci – např. čísla PWM kanálů, jejich frekvence a debounce čas tlačítek. Mezi konstantami jsou definované i hodnoty, které představují logické úrovně – vysoká, nízká, vysoká impedance a nedefinovaná hodnota.
Následují definice nejdůležitějších struktur programu, které popisují stavebnici jako takovou (`block`, `pin`, `wire`), detailněji rozepsané níže. Společně s těmito strukturami jsou zde pro větší přehlednost celého programu definovány nové datové typy: čísla zmíněných prvků stavebnice (fakticky jen přejmenované `unsigned char`), stav vodiče/pinu (pro který byl taktéž využit `unsigned char`) a odkaz na funkci, kteoru bloky vykonávají (přejmenování z výchozího tvaru jazyka C na kratší výraz `fp`).
Poslední částí `logic.h` jsou hlavičky funkcí, které poskytuje soubor `logic.c`. Téměř všechny představují právě funkce vykonávané jednotlivými funkčními bloky stavebnice, s výjimkou `write_outputs()` a `execute_block()`. Jejich bližší popisy jsou v kapitole 5.1.3.
Důležité struktury:

- `block`: Struktura, která reprezentuje v paměti jeden funkční blok stavebnice.

  Atributy:

  - `fp function_pointer`: Odkaz na funkci, kterou blok vykonává. Argumenty jsou pole vstupních stavů, množství využitých vstupů, a pole, do kterého umístí výsledky funkce (tj. výstupní stavy bloku). Podrobněji jsou popsány v kapitole 5.1.3.

  - `pin_number input[INPUT_NR]`: Pole pinů, které danému bloku patří jako vstupní, jejich maximální množství je dáno makrem `INPUT_NR`.
  - `unsigned char inputs_used`: Počet vstupů, které blok skutečně využívá, z pole `input[]`. Čistě logické funkce jsou napsané obecně pro teoreticky neomezené množství vstupů, takže je potřeba, aby měly možnost se včas zastavit a nedošlo k ovlivnění výsledku nedefinovanými hodnotami.
  - `pin_number output[OUTPUT_NR]`: Pole pinů, které danému bloku patří jako výstupní, jejich maximální množství je dáno makrem `OUTPUT_NR`.
  - `state new_pin_state[OUTPUT_NR]`: Pole se stavy, které slouží pro kontrolu, zda je možné výstupní stavy skutečně zapsat na výstupní piny v případě, že se uvažuje zpoždění bloků.
  - `unsigned long next_update`: Čas, při kterém je možné výstupní stavy zapsat na výstupní piny.

- `pin`: Struktura, která v paměti reprezentuje jeden simulovaný pin stavebnice.

  Atributy:

  - `unsigned char io_nr`: index pinu v poli `input` nebo `output` struktury `block`, které náleží – rozlišení mezi vstupem a výstupem je zajištěno pomocí hodnoty `IO_OUTPUT`, pokud je hodnota `io_nr` menší než `IO_OUTPUT`, pak je pin vstupní, v opačném případě je výstupní,
  - `block_number block_nr`: číslo bloku, kterému daný pin náleží,
  - `state pin_state`: logický stav pinu (nízká, vysoká, vysoká impedance, nedefinováno),
  - `wire_number *wire_nr`: číslo drátu, na který je daný pin připojen,
  - `unsigned char tristate`: příznak, zda je pin vybaven třístavovým výstupem (dokáže na výstup nastavit stav vysoké impedance), nebo ne.

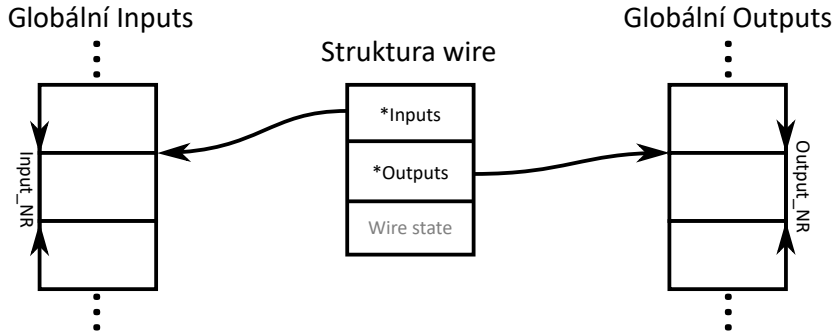
  Atribut `*wire_nr` je pointerem pro usnadnění globálního nulování, resp. obnovování stavu těchto atributů ve všech pinech naráz. Definováno je jedno globální pole čísel drátů, kde každá položka odpovídá jednomu pinu. Během inicializace se toto pole projde a ukazatele na konkrétní čísla drátů se umístí do daných pinů pro přehlednost.

- `wire`: Struktura, která v paměti reprezentuje jedno propojení několika pinů stavebnice.

  Atributy:

  - `pin_number *inputs`: pole čísel pinů, které z drátu „čtou“ stav,
  - `pin_number *outputs`: pole čísel pinů, které „zapisují“ stav na drát – platí pro ně omezení uvedené v kapitole 3.3.2,
  - `state wire_state`: logický stav vodiče – shodné se stavy pinů výše.

  Obdobně jako u atributu `*wire_nr` struktury `pin`, i zde jsou atributy `*inputs` a `*outputs` pro snažší úpravy vytvořené ve dvou velkých polích (zvlášť pro vstupy a výstupy), a při inicializaci ve funkci `setup()` (viz kapitola 5.1.6) jsou do struktur `wire` umisťovány pouze odkazy na konkrétní indexy tohoto pole. Výsledný stav je pro názornost zobrazen na obrázku 22.



Obrázek 22: Umístění referencí na `*inputs` a `*outputs` do struktury `wire`

5.1.3 `logic.c`

Soubor, který obsahuje výkonný kód funkcí, jejichž hlavičky byly definovány v `logic.h` (kapitola 5.1.2). Všechny tyto funkce s výjimkou dvou (`write_outputs()` a `execute_block()`), které budou popsány později, reprezentují jeden funkční blok stavebnice. Tyto funkční bloky z většiny tvoří „skutečně“ logické bloky AND, OR a jiné, ale stavebnice podporuje i další speciální součástky, se kterými může zapojení fyzicky interagovat navenek, jako jsou například tlačítka, motory, reproduktory.
Všechny tyto funkce mají jednotné rozhraní, aby mohly být libovolně umisťovány do datové struktury `block`, resp. jeho atributu `function_pointer`. Jako argumenty přijímají pole vstupních stavů, množství využitých vstupů a pole (prázdných) výstupních stavů. Pole vstupních stavů má pro všechny bloky fixní velikost, která je stanovená maximálním množstvím vstupů –

makrem `INPUT_NR`. Je nepravděpodobné, že by všechny použité bloky využily celé toto pole – mohlo by se procházet celé, ale hrozilo by, že bude výsledek některých logických funkcí ovlivněn hodnotami, které vlastně vůbec „na vstupu“ nemají. Proto je druhým argumentem množství využitých vstupů, který říká, kolik prvků pole vstupních stavů od nultého indexu má funkce brát v potaz. Výsledek funkce je uložen do pole výstupních stavů, které je opět fixně velké, tentokrát podle makra `OUTPUT_NR`. Množství obsazených prvků funkce nijak ven nepropaguje. Návratová hodnota je `char`, nabývající hodnoty `NG`, pokud si funkce nepřeje aktualizovat blok svým výsledkem – typické využití je u D klopného obvodu, který svůj výstup nemění, pokud není aktivní řídící signál CLK, či u tlačítek, které svou hodnotu nemění, pokud neuplynul debounce time. Jinak je vrácena hodnota `OK`.

Logické funkce Postup při zpracovávání obyčejných logických funkcí je zpravidla stejný jen s drobnými odlišnostmi – postupně je procházeno pole se vstupy a průběžně počítán výsledek, který je po dokončení průchodu vrácen v poli výstupů. V konkrétním případě logické funkce AND: výsledek je inicializován na vysokou logickou úroveň, pro každý prvek vstupního pole se kontroluje, jestli není roven nízké logické úrovni. Pokud ano, výsledek funkce je logická nízká úroveň a ta je hned vrácena. Pokud se dojde až na konec obsazené části pole a všechny vstupy byly přitom vysoké, výsledek je také vysoký (resp. zůstává vysoký už od inicializace). Funkce berou ohledy na vstupy v nedefinované úrovni nebo ve vysoké impedanci, v takovém případě opět záleží na konkrétní funkci, jak se zachová poté, co takový vstup detekuje. Pro zmíněný AND by hrály nedefinované vstupy roli pouze tehdy, že by žádný jiný vstup nebyl nízký. V takovém případě by i výstupní hodnota byla nedefinovaná. Pokud by nějaký jiný vstup nízký byl, výstup je vždy také nízký.

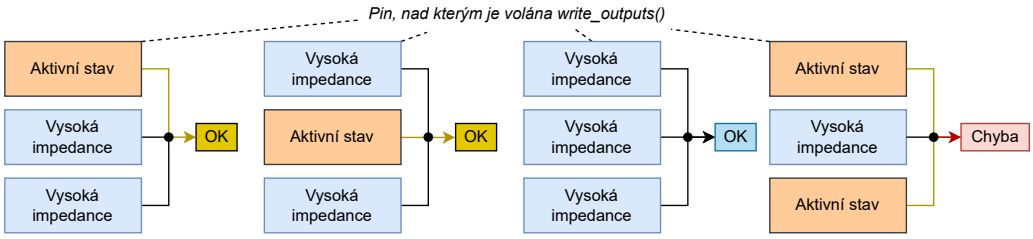
Funkce pro periferie Funkce ovládající fyzické periferie nevykonávají samy o sobě žádnou logickou funkci, pouze mapují stavy programu na stavy periferií nebo obráceně. Každá samostatná fyzická součástka má vlastní funkci (např. dvě tlačítka `but0()` a `but1()`), jelikož by jinak bylo potřeba nějakým způsobem v logickém bloku uchovávat navíc informaci o tom, o jakou konkrétní periferii se jedná (či spíše na jakém fyzickém pinu ESP je připojena). Protože jsou funkce poměrně jednoduché a periferií není příliš mnoho, byla upřednosněna varianta se samostatnými funkcemi pro každou z nich.
Vykonávaná činnost se liší v podstatě jen v tom, je-li součástka vstupní nebo výstupní. Jako vstupní lze klasifikovat mechanická tlačítka, IR detektor a fotorezistor. Jejich výstupem je vždy buď vysoká, nebo nízká úroveň v závislosti na tom, jestli došlo ke stisku, přiblížení nebo změně světla, vstupy nemají žádné. Výstupní součástky jsou stejnosměrný motor, servo, LED,

a reproduktor (piezo). Kromě LED všechny vyžadují PWM výstup, řízeny jsou na základě vstupní logické hodnoty, kterou beze změny předávají na výstup.

Funkce `execute_block()` Aby zbytečně nebobtnala hlavní smyčka programu, kde se periodicky volají funkce bloků stavebnice, byla vytvořena „vysokoúrovňová“ funkce, která z pohledu volajícího zajistí vykonání funkce bloku včetně dalších náležitostí. Funkce přebírá pouze jeden argument, kterým je číslo bloku, který má být vykonán.
Předtím, než je možné samotnou funkci bloku zavolat, je nutné provést transformaci vstupních hodnot – datová struktura `block` neuchovává přímo vstupní hodnoty, místo toho má pole s čísly pinů, které bloku náleží. Každý z těchto pinů je připojen na drátě (struktuře `wire`), ze které je teprve možné čerpat konkrétní logický stav (pin připojený „nikam“ čerpá hodnotu z drátu 0, jehož hodnota je nedefinovaná). V cyklu se tedy vstupní piny bloku prochází, a z drátů, na kterých jsou připojené, se získává `state`, tj. logický stav. Stav se získává jenom z tolika vstupních pinů, které blok skutečně využívá, tato informace je uložena v atributě `inputs_used` struktury `block`.
Takto přetransformované vstupy se společně s jejich množstvím a připraveným polem pro výstupy předají vlastní funkci bloku. Pokud funkce vrátí hodnotu `NG`, výstupy se zahodí a `execute_block` rovnou končí.
V opačném případě bylo vráceno `OK`. Pokud je aktivována simulace časového zpoždění bloků, jsou nové výstupní hodnoty porovnávány se stavy v `new_pin_states`. Pokud nejsou stejné, dojde k posunutí času, kdy se výstupy skutečně dostanou na výstup (`next_update`) o hodnotu `block_delay` a původní stavy v `new_pin_states` se přepíší novými.
Pokud se následně zjistí (porovnáním simulačního času `sim_time` s hodnotou `next_update`), že ještě nenastal čas pro aktualizaci výstupů, `execute_block` končí. Tato kontrola je prováděna nezávisle na tom, jestli je časové zpoždění povoleno nebo zakázáno, nicméně čas příští aktualizace `next_update` je inkrementován pouze v případě, že je zpoždění zapnuto – pokud zapnuto není, `sim_time` je vždy větší než `next_update`, podmínka nikdy není splněna a funkce v tomto místě neskončí.
Pokud funkce kvůli podmínkám výše neskončila předčasně, její poslední akcí je zapsání výstupních stavů na výstupní piny a potažmo vodiče prostřednictvím funkce `write_outputs()`.

Funkce `write_outputs()` Pokud by byl na jednom vodiči přípustný pouze jeden výstupní pin, udávající jeho stav, pak by bylo možné tento stav zapisovat přímo bez dalších komplikací. Obecně ale na vodiči může být i víc výstupů za předpokladu, že umožňují nastavení stavu vysoké impedance, ve kterém zbytek drátu neovlivňují, a že v jiném stavu než ve vysoké impedanci je nanejvýše jeden pin z množiny výstupů. O tento aspekt se stará právě

funkce `write_outputs()`. Přijímá dva argumenty: stav k zapsání a číslo pinu, ze kterého k zápisu dochází.
Nejříve se přes číslo pinu získá i číslo drátu, na který se má hodnota zapsat a rovnou se zapíše do atributu `pin_state` pinu. Následně prochází všechny výstupní piny, které jsou na drátě připojené a zjišťuje, zda jsou ve stavu vysoké impedance. Pokud nějaký ve stavu vysoké impedance není, ještě se kontroluje, zda stav pinu, ze kterého zapisuji, není ve vysoké impedanci. V takovém případě by na drátě byl stále jen jeden aktivní pin, což je přípustné. Pokud jsou na drátě skutečně dva aktivní piny, je to chyba, která je zobrazena na displeji voláním funkce `show_error()` s nastavením chybové proměnné na hodnotu `RUN_ERR` (viz kapitola 5.1.5). Nový stav drátu je nedefinovaný a funkce `write_outputs()` končí.
Pokud při kontrole nedošlo k detekci více jak jednoho aktivního pinu, nový stav drátu je buď dán stavem toho jednoho aktivního pinu, nebo je stav změněn na vysokou impedanci, pokud není aktivní ani jeden pin. Grafické znázornění je na obrázku 23.



Obrázek 23: Grafické znázornění výsledků funkce `write_outputs()`.

5.1.4 `function_map.h`

Krátký soubor, sloužící výhradně pro definici pomocné struktury pro mapování názvů funkčních bloků (tak jak jsou zapisovány do konfiguračních souborů uživatelem) na jména funkcí, které dané bloky vykonávají, aby bylo možné je snadno zapisovat do struktur `block`.
Definovaná struktura má název `name_to_function` a obsahuje atributy:

- `const char *name`: textový název bloku, uváděný v konfiguračních souborech,
- `fp function_poiner`: odkaz na funkci, kterou blok vykonává,
- `unsigned char input_amount`: počet vstupů, které blok skutečně využije,
- `char starting`: příznak, jestli je blok považován za vstupní a bude sloužit jako jeden z počátečních vrcholů pro průchod grafem (viz kapitola 5.1.6).

Následuje definice pole těchto struktur (`mapping[]`), kde jsou vyjmenované a vypsané hodnoty pro všechny podporované bloky. Aby bylo možno korektně ukončit prohledávání pole, poslední prvek je tvořen pouze prvky s hodnotou nula.

5.1.5 `main.h`

Hlavičkový soubor `main.h` obsahuje pouze definice konstant, označujících chyby, ke kterým může dojít v různých stavech programu, a hlavičku funkce `show_error()`, která je využívána pro „vyhození“ chyby (tj. zobrazení na displeji a nastavení příslušné proměnné), pokud nějaká nastane.
Možné chyby a informace o místě chyby:

- `NO_ERR`: žádná chyba,
- `DEV_ERR`: došlo k chybě při načítání součástek ze souboru, zobrazeno číslo řádky a číslo objektu,
- `PRE_CONN_ERR`: došlo k chybě při načítání předdefinovaných propojek ze souboru, zobrazeno číslo řádky a číslo propojky,
- `PHY_CONN_ERR`: fyzické propojky jsou nevhodně spojené (spojeno víc výstupních pinů, které nepodporují tři stavy), zobrazeno číslo jednoho pinu z propojky,
- `INV_CONN_ERR`: v souboru s předdefinovanými propojkami je nevhodné spojení (spojeno víc výstupních pinů, které nepodporují tři stavy), zobrazeno číslo jednoho pinu z propojky,
- `RUN_ERR`: při zápisu nové hodnoty funkčního bloku na drát o více výstupech se třemi stavy je víc jak jeden pin v aktivním stavu (zobrazeno číslo jednoho pinu z propojky), případně pokud je výstup nějakého funkčního bloku nedefinovaný (zobrazeno číslo pinu, na který se má nedefinovaná hodnota zapsat).

5.1.6 `main.cpp`

Vstupní bod programu obsahující funkce `setup()` a `loop()`. Dále obsahuje funkce, které se týkají ovládání a nastavování stavebnice jako celku, komunikaci s mikroprocesory sloužícími jako IO extendery a komunikaci s uživatelem přes desku s LED a tlačítky. Jsou zde deklarovány hlavní proměnné představující stavebnici.
Na několika místech jsou využity bitmapy, pro usnadnění práce s nimi obsahuje `main.cpp` funkce `bitmap_is_set()` a `bitmap_set()`. Tyto funkce ale pouze přímo obalují bitové operace, nekontrolují, zda předané argumenty (především číslo prvku) dávají smysl, to je povinnost volajícího.

Funkce `setup()` Vstupní bod programu. V této funkci dochází k inicializaci:

- webserveru pro OTA aktualizace, tj. zapnutí wifi a nastavení HTTP serveru
- LED modulu s tlačítky (`displayBegin()`) včetně prvního rozsvícení pomocí funkce `display_refresh()`,
- I^2C sběrnice jako objektu Wire frameworku Arduino (`Wire.begin()`),
- konkrétních GPIO pinů ESP, které slouží pro připojení fyzických periferií stavebnice (LED, motory, fotorezistory, ...) funkcí `pin_setup()`,
- mikroprocesorů zajišťujících rozšíření IO pinů funkcí `aux_init()`.

Dále jsou zde rozdělena globální pole představující vstupy a výstupy všech drátů do konkrétních struktur `wire`, stejně jako čísla drátů, na kterých jsou připojeny piny (viz 5.1.2), nultý `wire` představuje nezapojený pin, jeho stav se nastaví na nedefinovanou úroveň, pokud by z něj nějaký logický blok čerpal vstup, získá nedefinovaný stav. Jako poslední se zavolá funkce `get_devices()` s pevně daným argumentem "0", čímž se načte nultá sada součástek, pokud existuje, a `setup()` končí.

Funkce `display_refresh()` Jediná funkce, která pracuje s LED umístěnými na modulu s LED a tlačítky pro interakci s uživatelem. Nepřebírá žádné argumenty a nic nevrací.
Při každém zavolání obnoví vypisovaná čísla aktuálně používané konfigurace součástek a propojek – pokud nejsou předdefinované propojky využity, na příslušné dvojici sedmisegmentových displejích rozsvítí pouze dvě pomlčky. Pokud stavebnice není v chybovém stavu, funkce znuluje (zhasne) všechny LED, které jsou jinak využity pro zobrazení chybového stavu. V opačném případě se dále vypisují informace o chybě. Rozsvítí se jedna ze samostatných LED indikující typ chyby (viz kapitola 5.1.5). V závislosti na typu chyby se na čtveřici sedmisegmentových displejů nahlíží buď jako na dvě dvojice sedmisegmentových displejů pro vypsání dvou až dvouciferných čísel, nebo jako na trojici displejů pro až tříciferné číslo (jeden panel zůstane nevyužitý). Pro ovládání displeje je využita knihovna TM1638plus, jejíž autorem je Gavin Lyons [16].

Funkce `button_check()` Kontroluje stisk tlačítek na modulu s LED a tlačítky. Nepřebírá žádné argumenty. Vzorkování probíhá jen tehdy, pokud uplynul debounce time. Pokud ano, je načten nový stav tlačítek a pomocí předchozího načteného stavu se zjistí, na kterém tlačítku došlo ke změně. Pokud byla změna z nestisknutého do stisknutého stavu, provede se příslušná akce, tj. změna vybraných souborů se součástkami a propojkami. Tímto

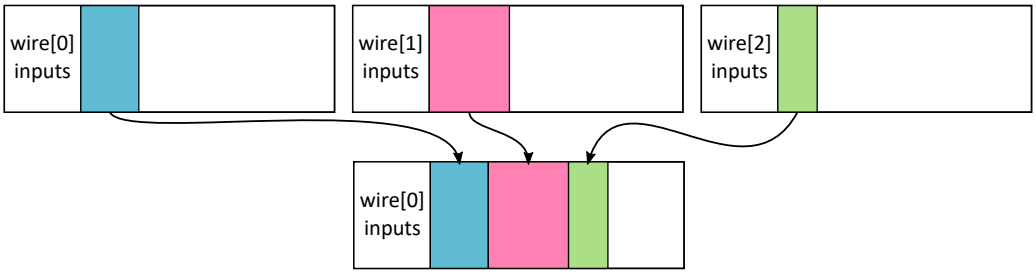
způsobem může na jedno stisknutí dojít vždy jen k jedné změně parametrů (pokud se nepřipouští zákmity tlačítka), pro násobnou změnu je potřeba tlačítko vícekrát stisknout. Funkce vrací `unsigned char` o hodnotě 0, pokud nedošlo k žádné akci, a nenulovou hodnotu, pokud bylo nějaké tlačítko stisknuto.

Funkce `get_devices()` Slouží k načtení součástkové základny ze souboru. Přebírá jediný argument typu `char`, který udává číslo požadovaného souboru – soubory se součástkami mají názvy XX.PC, kde XX je právě dvouciferné číslo specifikované argumentem.
Součástky se v souboru očekávají zadané ve formátu AAAA-X-Y-...;, kde AAAA je název součástky a X s Y jsou čísla pinů, na kterých se součástka nachází. Množství pinů závisí na konkrétní součástce, implicitně se nejprve očekávají vstupní piny, po načtení takového počtu, který odpovídá dané součástce, se další piny považují za výstupní. Každá součástka (včetně té poslední v souboru) musí být ukončena středníkem. Vzorová součástka může vypadat např. takto: AND2-4-5-6;. Došlo by k načtení bloku vykonávajícího logický součin se dvěma vstupy na pinech 4 a 5 a výstupem na pinu 6. Uvedené piny jdou v posloupnosti za sebou, ale obecně to není potřeba, vstupní piny mohou být podle libosti např. 13 a 2, výstupní 5.
Po otestování, že požadovaný soubor existuje a lze ho otevřít, začíná hlavní smyčka funkce, ve které se načítají ze souboru znaky tak dlouho, dokud se nenarazí na první pomlčku. Tato pomlčka odděluje název bloku od čísel pinů. Načtený název je porovnán se záznamy v mapovací tabulce (viz kapitola 5.1.4), a pokud je nalezen shodný záznam, jsou překopírovány ukazatele na vykonávanou funkci bloku a počet využitých vstupů. Pokud je navíc načtený blok „počáteční“ (především tlačítka, ale i generátory konstantních logických úrovní), je zařazen mezi výchozí bloky pro výpočet průchodu výsledným grafem. Pokud žádný shodný záznam nalezen nebyl, funkce končí chybou.
Po načtení jména se pokračuje načítáním čísel pinů. Pin je načtený ve chvíli, kdy se narazí buď na další pomlčku, případně na středník (kterým končí i celá součástka). Do struktury `pin` se zanesou informace o bloku, kterému náleží – jeho číslo, a index načteného pinu v poli vstupů/výstupů bloku (v případě výstupu se navíc přičte hodnota `IO_OUTPUT`). Opačně se také do bloku do patřičného pole vstupů nebo výstupů zapíše číslo načteného pinu. Navíc pokud je načítaná součástka buffer, jeho výstupním pinům se nastaví příznak `TRISTATE`.
Pokud se po středníku zjistí, že soubor už neobsahuje další znaky, načítání součástek je dokončeno a funkce končí. Návratovou hodnotou je `char` o hodnotě 0, pokud vše proběhlo v pořádku, nebo 1, pokud došlo při načítání k nějaké chybě.

Funkce `get_connections()` Slouží k načtení předdefinové sady propojek ze souboru. Její funkčnost se velmi podobá předchozí `get_devices()`: přejímá jeden argument typu `char`, který udává číslo požadovaného souboru – soubory s propojkami mají názvy XX.PP, kde XX je právě dvouciferné číslo specifikované argumentem.
Propojky se očekávají v následovném formátu: X-Y-Z-...;, kde X, Y a Z jsou čísla navzájem spojených pinů – např. 12-5-23;. Každý set propojených pinů (i ten poslední v souboru) musí být ukončen středníkem, čísla pinů můžou být uvedena v libovolném pořadí.
Pokud soubor existuje a byl v pořádku otevřen, začnou se z něj načítat znaky tak dlouho, dokud se nenarazí na pomlčku nebo středník. Tím bylo načteno jedno číslo pinu, které se umístí do pole uchovávajícího všechna čísla vzájemně spojených pinů. Jakmile je načten středník, toto pole se předá funkci `add_wire()`, která z dodaných pinů vytvoří drát (respektive využije informací o pinech k umístění správných hodnot do některé, zatím nevyužité, struktury `wire`, viz dále). Jakmile za středníkem neexistuje další znak, soubor byl bez chyby přečten celý.
Posledním krokem této funkce je „zazálohování“ stavu polí se vstupy a výstupy struktur `wire` a také pole udávajícího čísla vodičů, na které jsou piny připojené. Tato záloha bude následně vyvolána při každé obnově fyzických propojek jako návrat do výchozího stavu, raději než z přijatých dat určovat kromě toho, jaké piny jsou spolu spojené, i to, jaké piny spolu *nejsou* spojené, a podle toho přímo struktury modifikovat.

Funkce `add_wire()` Z dodaných čísel pinů upraví volnou strukturu `wire` tak, aby reflektovala propojení mezi danými piny. Přebírá pole čísel pinů, které jsou spolu spojené.
Předané pole je postupně procházeno. Pin je nejprve kontrolován, zda už není na nějakém drátě připojen. Pokud by byl, tato informace se uloží a bude vyřešena později. Číslo pinu se zanese do pole `inputs`, případně `outputs`, nového drátu, a samotnému pinu se nastaví číslo drátu, na který je připojen. Pokud by byl pin výstupní, je kontrolováno, zda je ve spoji jediný, nebo pokud není, jestli mají všechny piny třístavové výstupy (v tomto případě stačí kontrolovat vždy nově přidávaný s předchozím přidaným). Pokud by jedna nebo druhá podmínka nebyla splněna, funkce končí s chybovou návratovou hodnotou.
Posledním krokem po odbavení všech pinů je kontrola, jestli už nějaké nebyly připojeny na jiných drátech. Pokud ano, musí být tyto dráty sloučeny do jednoho zavoláním funkce `merge_wires()`, přičemž po jejím provedení může být „uvolněna“ (znulována) poslední struktura `wire` (tzn. ta, do které zapisovala probíhající funkce `add_wire()`), neboť `merge_wires()` data tohoto drátu přesunula (sloučila) do jiného.
Pokud vše proběhlo v pořádku, funkce vrací 0, v opačném případě nenulovou hodnotu.

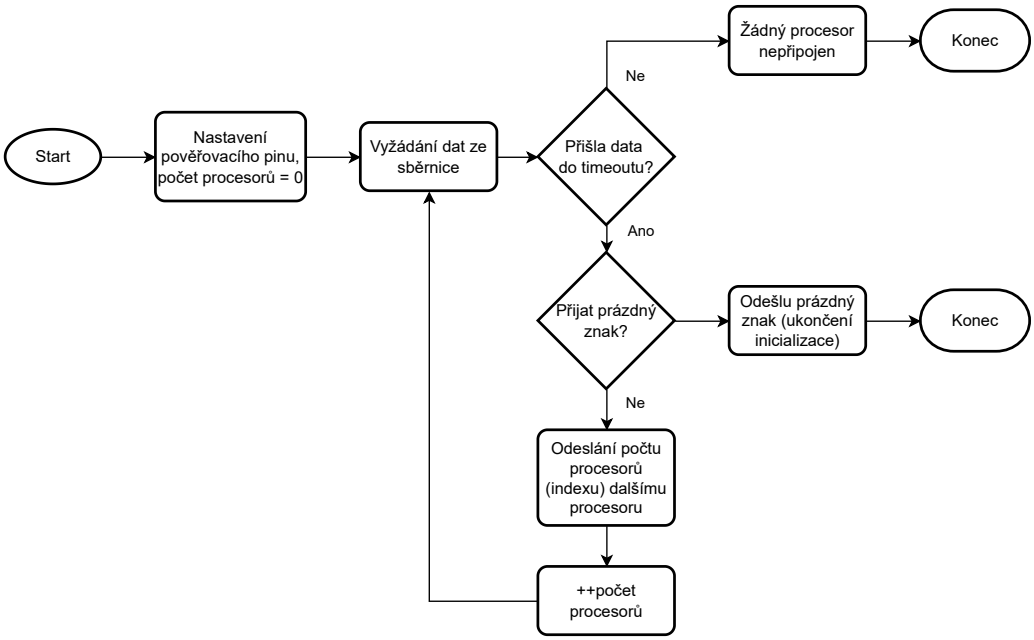
Funkce `merge_wires()` Zajišťuje sloučení několika drátů do jednoho. Přebírá pole čísel drátů, které mají být spojeny, a počet prvků v tomto poli.
Funkce ve vší obecnosti předpokládá, že se v předaných drátech mohou piny vyskytovat násobně – ve výsledném drátu ale násobně být nesmí a z tohoto důvodu je využita bitmapa s čísly pinů, která bude opakovanému vkládání zabraňovat.
Je potřeba stanovit drát, do kterého se všechny sloučí. Volba může být prakticky náhodná, jako cíl slučování byl zvolen první drát v poli všech drátů ke sloučení.
Následně probíhá dvě smyčky přes všechny předané dráty s výjimkou nultého („cílového“), ve které se postupně prochází všechny piny spojovaných drátů z polí `inputs` a `outputs`, pomocí bitmapy se kontroluje, zda už nejsou součástí cílového drátu, a pokud ne, jsou na něj zapsány na první neobsazený index. První smyčka spojuje vstupní piny, kde kromě násobného přidávání není v podstatě potřeba řešit nic víc, druhá smyčka prochází výstupní piny, kde je navíc potřeba kontrolovat, aby výsledný drát respektoval pravidla pro spojování výstupů, tj. buď pouze jeden, nebo víc, ale se třemi stavy.
Obrázek 24 zobrazuje stav tří drátů, respektive jejich polí s čísly vstupních pinů `input`, před a po provedení sloučení.
Pokud by množství výsledných vstupů/výstupů přesáhlo velikosti polí, piny navíc budou jednoduše odříznuty. V současné implementaci to není nijak kontrolováno a dále ošetřeno.



Obrázek 24: Před a po sloučení vstupů drátů

Funkce `aux_init()` Slouží k inicializaci procesorů zajišťujících rozšíření IO pinů a ke zjištění, kolik jich na sběrnici vlastně je. Nepřebírá žádné argumenty a nic nevrací.
Celá funkce běží ve smyčce. Před jejím zahájením je na vysokou úroveň nastaven pověřovací pin. Nejprve se po sběrnici zkusí vyžádat nějaká data z podřízeného procesoru. Pokud by na sběrnici nebyl připojený žádný, do timeoutu žádná data nepřijdou a funkce skončí s nulovým počtem detekovaných obvodů. V opačném případě (pokud nějaká data dorazila) mohou nastat dvě možnosti: buď je přijat „prázdný“ znak, který znamená, že už byly všechny procesory inicializovány – ESP na sběrnici pošle ukončovací

znak, čímž se inicializace ukončí a funkce může skončit, nebo je přijato cokoliv jiného, čímž se nějaký procesor hlásí o přidělení indexu. Tento index je na sběrnici zapsán, číslo připojených obvodů je inkrementováno o 1 a cyklus pokračuje.
Graficky je algoritmus znázorněn na obrázku 25.
Výsledkem je, že všechny připojené procesory mají jednoznačně přidělený index (pořadí), v jakém budou na sběrnici vysílat (což znamená, že už při běžném provozu není potřeba předávat pověření) a ESP ví, kolik procesorů se na sběrnici celkem nachází.



Obrázek 25: Algoritmus pro inicializaci připojených procesorů

Funkce `get_physical_connections()` Funkce sloužící k získání stavu fyzických propojek, které jsou zajišťovány podřízenými procesory přes sběrnici I^2C. Nepřijímá žádné argumenty, vrací nulové číslo, pokud vše proběhlo správně, v opačném případě nenulové číslo.
Data o propojkách si lze představit jako čtvercovou matici o hraně s takovým počtem prvků, jaké je množství pinů. Význam mají jednotlivé řádky. V řádku n je aktivní pin s indexem n, na stejném sloupcovém indexu se musí objevit jednička. Diagonála matice bude tedy vždy obsazena jedničkami. Pokud je aktivní pin spojen s dalšími piny, budou i na sloupcových indexech těchto pinů jedničky. Příkladem budiž matice 1, popisující propojky pěti pinů.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 1 & 0 & 0 \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{bmatrix} \tag{1}$$

Ideální by bylo, kdyby bylo možno načíst celou matici v jeden okamžik, aby nemohlo dojít během načítání ke změně. V takovém případě by stačilo brát v úvahu vždy jen část matice nad diagonálou – buď se zjistí, že je pin někam připojený ve chvíli, kdy je sám aktivní, nebo už nemůže být připojený nikam. Realita je bohužel složitější, matici naráz přečíst nejde, ale algoritmus k ní tak bude přistupovat – pokud bude vzorkování probíhat dostatečně rychle, důsledky budou minimální. Prvky, které teoreticky není potřeba zjišťovat, jsou v matici 1 zapsány šedou barvou.

Jedno spojení pinů se v matici objeví násobně – pokud je spojený pin 1 s piny 3 a 4, stejně tak je spojen pin 3 s piny 1 a 4. Opět lze teoreticky předpokládat, že se matice během načítání nezmění, a po načtení první řádky je možné všechny ostatní, obsahující stejnou informaci, rovnou přeskočit. Na příkladové matici 1 by to byly řádky s indexy 1, 3 a 4 (zvýrazněny tučným písmem).

Ještě před začátkem načítání a faktickým přidáváním propojek je z paměti načtena záloha propojek ze souboru (viz funkce `get_connections()`), aby byly nové fyzické propojky vždy přidávány z nějakého výchozího stavu. Pokud funkce `get_connections()` neproběhla (není zvolený žádný konfigurační soubor), dojde k prostému znulování.

Funkce běží v cyklu přes všechny řádky. Pro daný řádek (resp. pin) pomocí bitmapy zjistí, jestli už ho v nějakém předchozím cyklu nenačetla. Pokud ano, místo čtení se vysílá povel k jeho přeskočení. Načítání řádky probíhá po bytech – začíná se od bytu, kde je aktivní pin. Přijatá binární data se převedou na čísla pinů, která se zazamenají do bitmapy, aby se mohly případně přeskočit. Není potřeba řešit předávání pověření na sběrnici – podřízené obvody znají svoje pořadí a počet již vykonaných čtení a jsou schopné se řídit samy nezávisle na sobě.

Jakmile je řádka načtena celá, čísla spojených pinů se předávají funkci `add_wire()`, která se postará o jejich začlenění do případných už existujících propojek.

Funkce `get_block_order()` Slouží k získání optimálnějšího pořadí vykonávání bloků než obyčejný průchod jejich polem od prvního do posledního indexu. K tomuto účelu bude sloužit pole čísel bloku `compute_order[]`, ve kterém budou čísla bloků v pořadí, v jakém se mají vykonávat. Funkce nepřijímá žádné argumenty a nic nevrací.

Použitý algoritmus se v podstatě shoduje s procházením grafu do šířky – odlišností je jen to, že výchozí není jeden, ale obecně víc vrcholů. Tyto

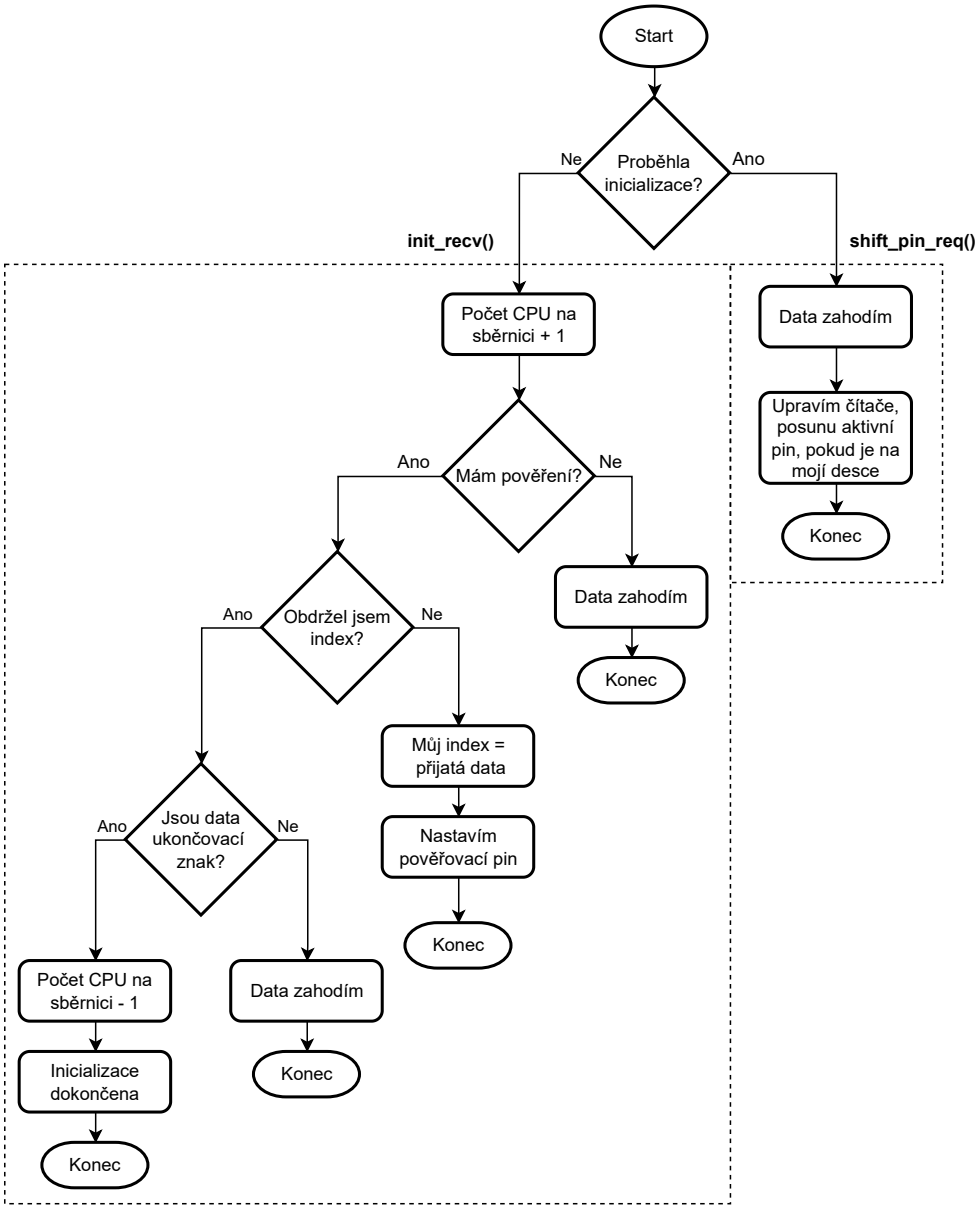
vrcholy jsou stanoveny při načítání bloků, kde některé z nich mají status „počátečních“ a jsou rovnou zařazeny do pole `compute_order[]`.
Funkce načítá bloky podle jejich čísel v `compute_order[]` a do stejného pole za poslední využité indexy umisťuje bloky, které jsou na jejich výstupu. Jakmile se dojde na index v poli, který neukazuje na žádný blok, funkce zpracovala všechny struktury, které jsou využité pro běh stavebnice, a pole `compute_order[]` udává, v jakém pořadí se mají vykonávat. Aby nedocházelo k opětovnému vkládání bloků do seznamu, je využita bitmapa. Pro potenciální urychlení je využita ještě bitmapa pro dráty – pokud by se slučovaly výstupy z více bloků, budou všechny na jednom drátě a nebude potřeba ho vícekrát procházet.

Funkce `loop()` Hlavní smyčka programu. V každém cyklu zkontroluje zavoláním funkce `button_check()`, zda nebylo stisknuto tlačítko, případně nechá funkcemi `get_devices()` a `get_connections()` načíst nové konfigurační soubory.
V závislosti na chybách (či jejich absenci) po načítání souborů či předchozích výpočtech funkčních bloků nechá načíst přes `get_physical_connections()` fyzické propojky, zavoláním `get_block_order()` získá pořadí funkčních bloků pro vykonávání a následně je všechny nechá vykonat.

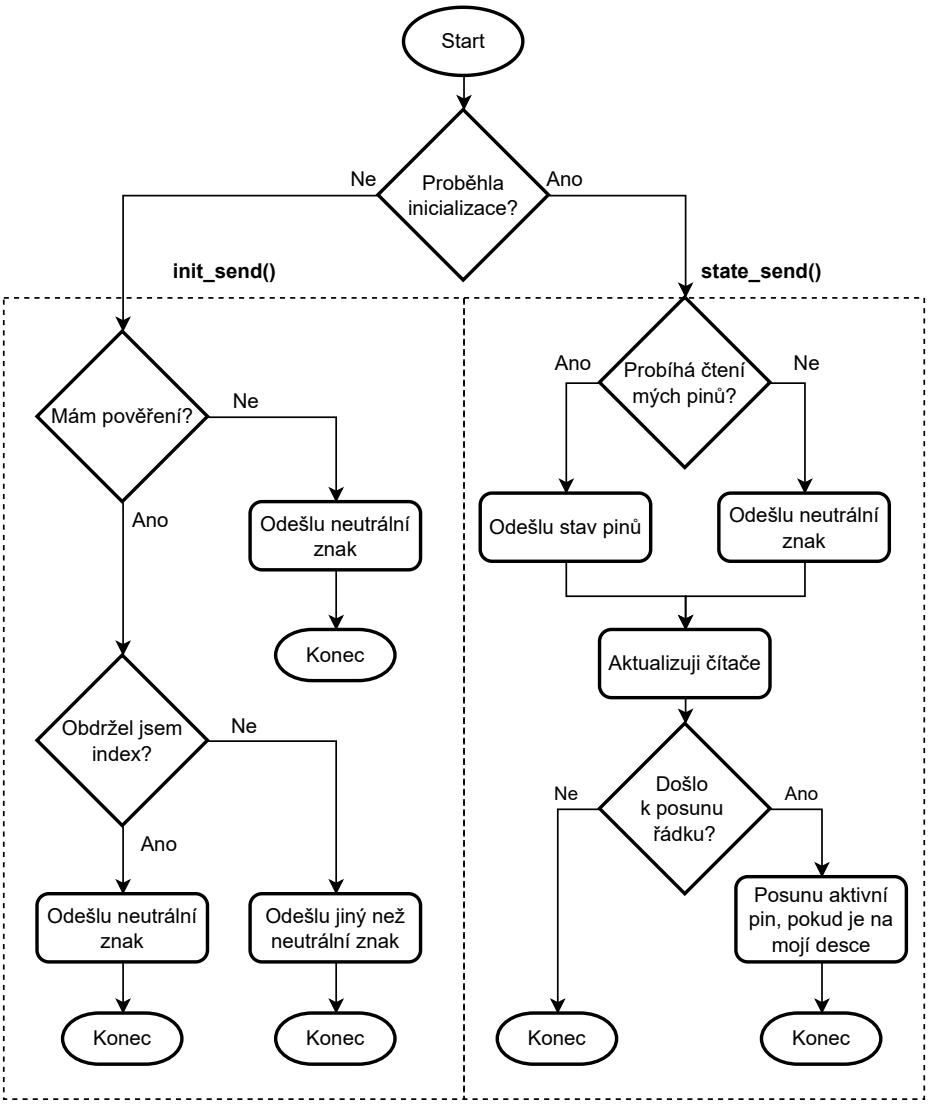
5.2 IO extending mikroprocesory

Tyto obvody pouze obsluhují požadavky na zjišťování stavu propojek od ESP po sběrnici I^2C. Samotný provoz předchází inicializační fáze, ve které je každému procesoru v řadě postupně přiděleno pořadí, v jakém bude vysílat. Tím, že nejsou periferie pro ovládání I^2C vypínané (viz část v kapitole 5), procesory ví o každém požadavku na čtení (případně zápis). Tím je následně umožněno, aby se na sběrnici samy střídaly i bez explicitního předávání pověření – pověření putuje pouze při inicializaci.
Ve funkci `setup()` se pouze nastaví použité piny, inicializuje I^2C periferie a jako handlery událostí I^2C (tj. příjmu a odesílání dat) se nastaví inicializační funkce `init_send()` a `init_recv()`.
Následně probíhá inicializační fáze, která je ukončena příjmem ukončovacího znaku z ESP. Po přijetí tohoto znaku se handlery událostí přepisují na funkce `state_send()` pro odeslání aktuálního stavu propojek a `shift_pin_req()` pro příjem dat z ESP – reálně se ale zápis do podřízených obvodů používá jen jako signalizace, že se přeskakuje nějaký aktivní pin, čemuž odpovídá i název. Procesory po inicializaci kromě svého pořadí znají i celkový počet obvodů připojených na sběrnici.
Předpokladem je, že jsou všechny rozšiřující procesory stejné (nebo alespoň rozšiřují o stejný počet pinů) – tato hodnota je definována jako konstanta, aby obvody věděly, v jaké části načítání se nachází. Procesory si během komunikace udržují číslo, kolik bylo provedeno čtení na jedné řádce (viz

část o funkci `get_physical_connections()` v kapitole 5.1.6). Jakmile byla přečtena celá jedna řádka, dojde k posunu na další, a aktivaci dalšího pinu (s deaktivováním toho předchozího). Data jsou kvůli I^{2}C dělena a posílána po bytech, řádky se začínají číst vždy až od bytu, kde je aktivní pin (tak jak bylo popsáno v `get_physical_connections()`). Algoritmy pro příjem a odesílání dat jsou přehledněji a podrobněji zachyceny na obrázcích 26 a 27.



Obrázek 26: Algoritmus pro příjem dat z ESP po I^{2}C



Obrázek 27: Algoritmus pro odesílání dat do ESP po I^2C

6 Jednoduchá demonstrační úloha

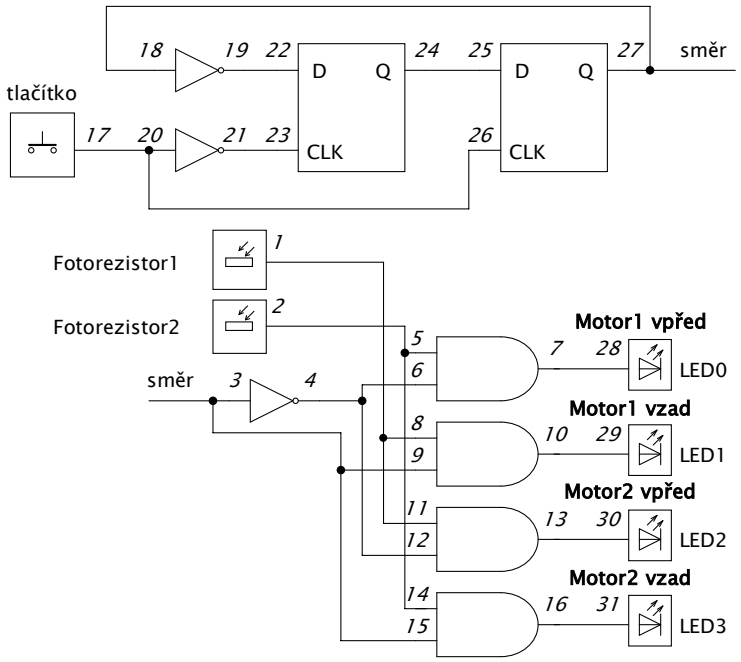
Pro předvedení funkčnosti stavebnice na reálné úloze byl vybrán „robot“ jezdící buď za světlem, nebo pryč od světla. Tento robot by měl dva fotorezistory pro snímání světelné intenzity a dva motory pro ovládání pohybu. Směr bude volen tlačítkem (každý stisk odpovídá změně směru).
Pravdivostní tabulky a tvorba a minimalizace logických rovnic zde není podstatná, ani z hlediska obtížnosti zajímavá, a bude přeskočena. Výsledné zapojení je na obrázku 28, včetně naznačených čísel pinů, na kterých budou součástky připojeny. Pro změnu směru byl ze dvou hladinově řízených D klopných obvodů vytvořen T klopný obvod řízený náběžnou hranou z tla-

čítka. Předpokládá se, že na výstupu fotorezistorů je vysoká úroveň, pokud jsou nasvíceny. Pro drobné zjednodušení jsou na výstupech místo skutečných motorů LED diody, které signalizují, který signál je zrovna aktivní.
K dispozici je 16 fyzických pinů poskytovaných STM32. Jelikož demonstrační úloha vyžaduje větší množství, přebytečné piny (čísla 17 a výše) budou spojeny pomocí předdefinovaného konfiguračního souboru.
Výsledný obsah souboru s definicemi součástek (např. 00.PC) bude následovný (bez zalomení řádky):

```
FOTO_R1-1;FOTO_R2-2;NOT-3-4;AND2-5-6-7;AND2-8-9-10;AND2-11-12
-13;AND2-14-15-16;BUT1-17;NOT-18-19;NOT-20-21;D-22-23-24;D-25-26-27;
```

Obsah souboru s předdefinovanými propojkami (např. 00.PP) bude následovný:

```
17-20-26;21-23;19-22;24-25;27-18-3;7-28;10-29;13-30;16-31;
```



Obrázek 28: Schéma zapojení demonstračního robota

7 Zhodnocení vlastností

Jak již bylo zmíněno, smyslem vytvořené stavebnice je zkombinovat klasický hardwarový přístup s modernějším softwarovým a pokusit se z obou typů vybrat ty lepší vlastnosti.
Uživatel, který bude ke stavebnici přistupovat jako ke skutečně hardwarové, jistě ocení její odolnost proti zkratování výstupů, což je možná nejzásadnější hardwarový problém. Stavebnice umí detekovat „statickou“ chybu, kdy jsou spojeny výstupy bez vysokoimpedančního stavu, stejně tak jako „dynamickou“, kdy sice výstupy vysokoimpedanční stav umí, ale došlo k aktivování více naráz. Tuto vlastnost typické stavebnice nemají a byla by obtížně řešitelná. Simulátory jsou v tomto ohledu ale stále uživatelsky přívětivější, jelikož mohou (např. odlišnou barvou vodiče) snáz zobrazit, kde k problému došlo. Stavebnice popsaná touto prací to sice umí taky, ale v mnohem omezenější variantě – oznámí chybu a číslo pinu, u kterého je problém. Do detailu musí ale diagnostiku provést až sám uživatel.
Oproti hardwarovým stavebnicím je vylepšením možnost použití „napřímo“ součástek, které nejsou čistě logické – reproduktory, serva, tlačítka atp. Jejich spojení s reálnými logickými bloky není nemožné, ale zejména pro PWM signály by byly potřeba další obvody, které by musely logickou úroveň nějakým způsobem převádět. Simulátory z principu žádné fyzické výstupy neposkytují s výjimkou světelné indikace na obrazovce.
Množství připojených fyzických periferií je nicméně zásadně omezeno množstvím volných pinů na mikroprocesoru, kde běží simulace (ESP32), případně použitelnými PWM kanály. Použitá deska WeMos LOLIN32 Lite poskytuje 23 pinů, z nichž 4 mohou být využity jen jako vstupní, některé mají nevhodné chování během bootovací sekvence a několik jich je zabráno pro zajištění funkce stavebnice (displej, I^2C). Dostupné piny tak poměrně rychle mizí. Programově není problém jejich funkci změnit a všechny dostupné maximálně využít, ale prakticky to není ideální.
Zásadním problémem pro věrohodnou imitaci hardwaru je dostatečná responzivita, především při obnovování fyzických propojek zajišťovaných mikroprocesory připojenými přes I^2C. K dispozici je bohužel jen jeden funkční rozšiřující mikroprocesor, ale bez potíží jich lze simulovat víc (víckrát jsou požadována data od stejného obvodu). Vzhledem k tomu, že celé obstarávání dat ze sběrnice zajišťuje jediná funkce (`get_physical_connections()`, viz kapitola 5.1.6), je možné snadno měřit časovou prodlevu od vstupu po výstup. Měření bylo provedeno základní časovou funkcí `millis()` frameworku Arduino pro namátkově vybrané množství obvodů, přičemž nebyly spojené žádné piny. Naměřené hodnoty činily 55 ms při pěti obvodech (80 pinů), 105 ms při sedmi obvodech (112 pinů) a 210 ms při 10 obvodech (160 pinů). Patrná je kvadratická složitost algoritmu (pole s propojkami je čtvercové) – při 80 pinech je průměrný čas načítání zhruba 0,69 ms na jeden pin, zatímco při 160 pinech už je to zhruba 1,3 ms na pin. Pokud by bylo třeba striktně

dodržet původní předpoklad vzorkování alespoň desetkrát za sekundu, maximem by bylo přibližně 100 pinů. Nicméně, čím hustější propojky budou, tím se bude čas potřebný k načtení snižovat díky implementaci přeskakování duplicitních řádků matice – k přeskočení je sice také potřeba přenést nějaká data, ale řádově menší. Částečného urychlení by bylo možno ještě dosáhnout distribucí načítání mezi výpočty funkčních bloků, ale rozdíl v době zpracovávání je tak velký, že by to prakticky zřejmě nepřineslo velké ovoce (viz dále).

Stejně tak musí centrální procesor funkční bloky zpracovávat sériově a pokud by jich bylo nadměrné množství, mohlo by se projevit zpomalení. Klasicky hardwarové stavebnice ani simulátory tento problém neřeší – hardware pracuje přirozeně paralelně, simulátory se zpravidla hardware imitovat nesnaží, místo toho naopak nabízí možnost „zpomalení“ času tak, že jsou graficky patrné průchody signálů jednotlivými funkčními bloky. Pro ilustraci byl opět funkcí `millis()` změřen čas výpočtu všech použitých funkčních bloků z demonstračního zapojení 6, tj. celkem 16 bloků různých funkcí. Změřená doba trvání se pohybovala mezi 1 a 2 ms. Přestože zpracovávané funkce nejsou zcela totožné a jejich doba výpočtu se může mírně lišit, jedná se o tak krátké časy, že lze prohlásit, že čas strávený výpočtem funkčních bloků je oproti času potřebnému pro načtení stavu fyzických propojek zanedbatelný (samozřejmě za předpokladu, že je fyzických pinů netriviální množství).

Nevýhodou současné implementace simulované stavebnice je nepřehledné zadávání součástkové základny a předdefinovaných propojek. K tomu slouží prosté textové soubory, do kterých musí uživatel zapsat potřebné bloky včetně čísel pinů, na kterých mají být připojené. Pro malé demonstrační úlohy to není problém, ale pro větší zapojení (desítky funkčních bloků) už to může být problematické – snadno dojde k chybě a než dělat případné korekce, je téměř snažší celý soubor zapsat nově. V tomto ohledu mají lepší vlastnosti v podstatě jak hardwarové stavebnice, tak simulátory, kde čísla pinů vlastně neexistují – piny se prostě propojují a je vizuálně patrné, kam je který potřeba vést. Pokud ale odhlédneme od nepřehlednosti, už s jednou definicí součástek má vytvořený simulátor srovnatelné vlastnosti se stavebnicemi. Oproti stavebnicím umožňuje ale navíc, díky možností předdefinovat propojky, zapojení připravit tak, že nezajímavé propoje nejsou vidět, obvod je přehlednější a snažší na zapojení (např. pro výuku).

Při testování sloužil jako IO extender mikroprocesor STM32, je ale možné použít naprosto libovolné obvody podporující I^2C s rychlostí 400 kHz. Ideální mikroprocesor pro tuto aplikaci má již zmíněné I^2C, relativně velké množství pinů (32, 64) a co nejméně dalších periferií pro co nejnižší pořizovací náklady. Prakticky je tento požadavek vznesen jen pro maximální usnadnění fyzické manipulace – počet obvodů nezmění chování stavebnice jako celku. Doba načítání je ovlivněna pouze absolutním množstvím poskytovaných pinů, z hlediska ESP je na sběrnici jen jeden obvod.

8 Závěr

Cílem této práce bylo vytvořit jakýsi přechodový článek mezi fyzickými, čistě elektronickými, logickými stavebnicemi a simulátory ve formě programů. Obě tyto varianty mají do značné míry odlišné vlastnosti a jejich vhodným spojením by mohla vzniknout zajímavá alternativa.
Konkrétní představou bylo vytvořit simulátor, který poběží místo klasického PC na mikroprocesoru a bude mít přístup ke konkrétním (fyzickým) periferiím s dostatečnou obnovovací frekvencí tak, že by neznalý uživatel mohl dojít k přesvědčení, že interaguje se skutečnou stavebnicí. Pro běh simulace byl vybrán pro svoje široké spektrum funkcí procesor ESP32, integrovaný na desce WeMos LOLIN32 Lite.
Klíčovým problémem bylo zajistit dostatek pinů, se kterými by uživatel mohl interagovat a které by sloužily pro fyzické propojování simulovaných součástek – jediný mikroprocesor jejich dostatkem nedisponuje. Řešením bylo použití I^2C sběrnice, na které jsou připojené další, podřízené, mikroprocesory, jejichž jediným smyslem je tyto piny poskytovat. Zásadní nepříjemností je globální nedostatek procesorů – nebylo možné zajistit větší množství pro plnohodnotnou demonstraci stavebnice. V této části došlo k potížím souvisejícím s použitým frameworkem, tj. Arduinem, které neumožňuje využít plné funkčnosti některých periferií (v tomto konkrétním případě vypínání/odpojování I^2C kontroléru).
Samotná implementace simulačního jádra nepřinesla zásadnější problémy. Pro definici součástkové základny jsou využity soubory uložené přímo v oddílu paměti ESP, dedikovaného pro file systém, stejným způsobem lze zapsat i předdefinované propojky. Při běhu se funkční bloky přepočítávají a čas od času se obnovuje stav fyzických propojek. Stavebnice umí detekovat chyby při spojování výstupů, které by u fyzických stavebnic mohly být destruktivní.
Pro stavebnici jsou důležité fyzické výstupy, díky kterým může uživatel skutečně pozorovat, že se něco děje. Tyto výstupy mohou být v podstatě libovolné součástky nebo funkční celky, se kterými dokáže ESP komunikovat. Počet pinů ESP je ale omezený, stejně jako počet PWM kanálů potřebných pro některé funkce, výstupy tak bohužel nelze rozšiřovat do nekonečna.
Stavebnici by bylo možno dále z hlediska uživatelského rozhraní rozšířit a vylepšit vytvořením programu pro PC, který by napohled připomínal typické simulátory a bylo v něm možno graficky vytvořit soubory se součástkovými základnami a předdefinovanými propojkami. Současný přímý textový způsob tvorby je sice funkční, ale poměrně nepřehledný, náchylný k chybám a nepříjemný na úpravy.
Dalším vylepšením by mohla být úprava programu na objektově orientovaný. Současná verze je procedurální (i přes využití jazyka C++) a změna na objekty by mohla přinést zpřehlednění programu a usnadnění práce s ním – např. každý typ funkčního bloku by mohl být specifikován svou třídou.

Výsledný produkt jistě není nic, co by otřáslo poměry na trhu s logickými stavebnicemi a simulátory, ale v souladu s předpoklady se jedná o hybrid, který by existující nabídku mohl vhodně doplňovat.

Literatura

[1] B. Sprague, M. Lewellyn, D. Knierim, J. Lansford, and N. Harro, "Cedar Logic," accessed on 2022-1-1. [Online]. Available: https://sourceforge.net/projects/cedarlogic/

[2] C. Burch, "Logisim," accessed on 2022-1-1. [Online]. Available: https://sourceforge.net/projects/circuit/

[3] B. Born, "simulator.io," accessed on 2022-1-1. [Online]. Available: https://simulator.io/

[4] Sparkfun, "Sparkfun LogicBlocks Kit," accessed on 2022-1-1. [Online]. Available: https://www.sparkfun.com/products/11006

[5] Voltík.cz, "Stavebnice Voltík," accessed on 2022-1-1. [Online]. Available: https://www.voltik.cz/stavebnice-voltik/

[6] G. Electronic, "Elektronická stavebnice Voltík II," accessed on 2022-1-8. [Online]. Available: https://www.gme.cz/elektronicka-stavebnice-voltik-ii

[7] S. Boffin, "Boffin," accessed on 2022-1-2. [Online]. Available: https://www.stavebnice-boffin.cz/boffin/

[8] Ing. Lukáš Krejčík, "Stavebnice Saimon," accessed on 2022-1-2. [Online]. Available: https://www.stavebnicesaimon.cz/

[9] Elektroráj, "Logitronik 01," accessed on 2022-1-1. [Online]. Available: http://www.elektroraj.cz/2017/10/05/logitronik-01/

[10] Sphero, "littleBits," accessed on 2022-1-1. [Online]. Available: https://sphero.com/pages/littlebits

[11] NXP, "I^2C-bus specification and user manual," 2021, accessed on 2021-12-15. [Online]. Available: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[12] Microchip, "MCP23017/MCP23S17 Data Sheet," 2016, accessed on 2021-12-12. [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/20001952c.pdf

[13] "Wemos LOLIN32 Lite," smalldevices.com, accessed on 2022-4-17. [Online]. Available: https://smalldevices.com.au/products/wemos-lolin32-lite

[14] "STM32 BluePill development board," newInnovations, accessed on 2022-4-17. [Online]. Available: https://www.newinnovations.nl/blue-pill-stm32-stm32f103-stm32f103c8t6/

[15] "STM8 Development Board," Deltakit, accessed on 2022-4-17. [Online]. Available: https://www.deltakit.net/product/micro-usb-stm8s103f3-stm8-motherboard-development-board/

[16] G. Lyons, "Tm1638plus," GitHub, 06 2019, accessed on 2022-4-14. [Online]. Available: https://github.com/gavinlyonsrepo/TM1638plus

[17] D. Incorporated, "PAM8403 Data Sheet," 2020, accessed on 2022-4-10. [Online]. Available: https://www.diodes.com/assets/Datasheets/products_inactive_data/PAM8403.pdf

[18] "Arduino-ESP32 2.0.2 documentation," docs.espressif.com, accessed on 2022-2-8. [Online]. Available: https://docs.espressif.com/projects/arduino-esp32/en/latest/index.html

[19] T. Instruments, "DRV8833 Motor Driver," 2011, accessed on 2022-4-10. [Online]. Available: https://www.ti.com/lit/ds/symlink/drv8833.pdf

[20] "ESP32 Pinout Reference: Which GPIO pins should you use?" Random Nerd Tutorials, accessed on 2022-4-18. [Online]. Available: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

[21] "ESP32 OTA (Over-the-Air) Updates - AsyncElegantOTA Arduino | Random Nerd Tutorials," Random Nerd Tutorials, 03 2021, accessed on 2022-4-15. [Online]. Available: https://randomnerdtutorials.com/esp32-ota-over-the-air-arduino/

[22] STMicroelectronics, "RM0008 Reference Manual," 2021, accessed on 2022-4-2. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

A Popis adresářové struktury příloh

- `/Aplikace_a_knihovny`
  - `/lolin32_lite`: Program pro ESP32 včetně metadat vývojového prostředí platform.io. Vlastní zdrojové kódy se nachází v `/src`.
  - `/STM32_I2C`: Program pro STM32 (rozšíření GPIO přes I2C), včetně metadat vývojového prostředí platform.io. Vlastní zdrojové kódy se nachází v `/src`.
  - `/AsyncElegantOTA, /AsyncTCP, /ESPAsyncWebServer`: Knihovny pro HTTP webserver.
  - `/TM1638Plus`: Knihovna pro ovládací panel s TM1638.
- `/Vstupni_data`
  - `/00.PC`: Definice součástek demonstračního zapojení.
  - `/00.PP`: Předdefinované propojky demonstračního zapojení.
- `/Text_prace`
  - `/BP.pdf`: Zkompilovaná práce.
  - `/main.tex, /references.bib`: Zdrojové soubory pro LaTeX.
  - `/img`: Adresář obsahující grafiku a zadání práce bez podpisů.