Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vytvoření aplikace na platformě Raspberry Pi k ovládání stimulátoru pro informatické experimenty

Plzeň, 2018 Milan Hajžman

Sem přijde zadání

Prohlášení

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

V Plzni dne 28. června 2018

Milan Hajžman

Poděkování

Rád bych poděkoval Ing. Pavlu Mautnerovi, Ph.D. za podporu při psaní této bakalářské práce a za jeho cenné rady, Ing. Evě Hajžmanové za jazykovou a typografickou pomoc, rodině a přátelům za podporu při psaní práce.

Abstract

Main Goal of this work is creating application on Raspberry Pi platform. This application will allow setup stimulator of neuroinformatic experiments and presenting of visual and acoustic stimuli. Work describes pitfalls of using QT framework to design graphical user interface and using SDL library to present stimuli. Measurements of delays of implemented solution are part of work.

Abstrakt

Cílem této práce je vytvoření aplikace na platformě Raspberry Pi. Tato aplikace umožní nastavení stimulátoru pro neuroinformatické experimenty a prezentaci vizuálních a akustických stimulů. Práce popisuje úskalí spojená s vytvářením uživatelského rozhraním QT frameworkem a prezentací stimulů SDL knihovnou. Součástí práce je měření odezvy implementovaného řešení.

Obsah

1 Úvod

Katedra informatiky a výpočetní techniky Západočeské univerzity v Plzni vyvíjí již řadu let modulární přístroj určený k neuroinformatickým experimentům. Přístroj funguje na bázi externího zařízení. Během vývoje se však programování pro dotykový displej ukázalo jako komplikované. Tato práce zkoumá možnosti rozšíření stimulátoru o mikropočítač Raspberry Pi, který nahradí dotykový displej a jeho rozhraní jednoduchou aplikací zobrazovanou na běžném počítačovém monitoru. Součástí tohoto řešení je i vizuální a akustická stimulace.

Teoretická část práce čtenáře seznámí s technikou stimulování evokovaných potenciálů. Následovat bude porovnání některých již hotových řešení. Dále se práce zabývá návrhem a implementací aplikace, popisem použitých knihoven a řešení klíčových prvků aplikace. Na konci práce provedu měření opoždění zobrazování obrazových stimulů na počítačovém monitoru.

2 Teoretická část

2.1 EEG vlny a jejich dělení

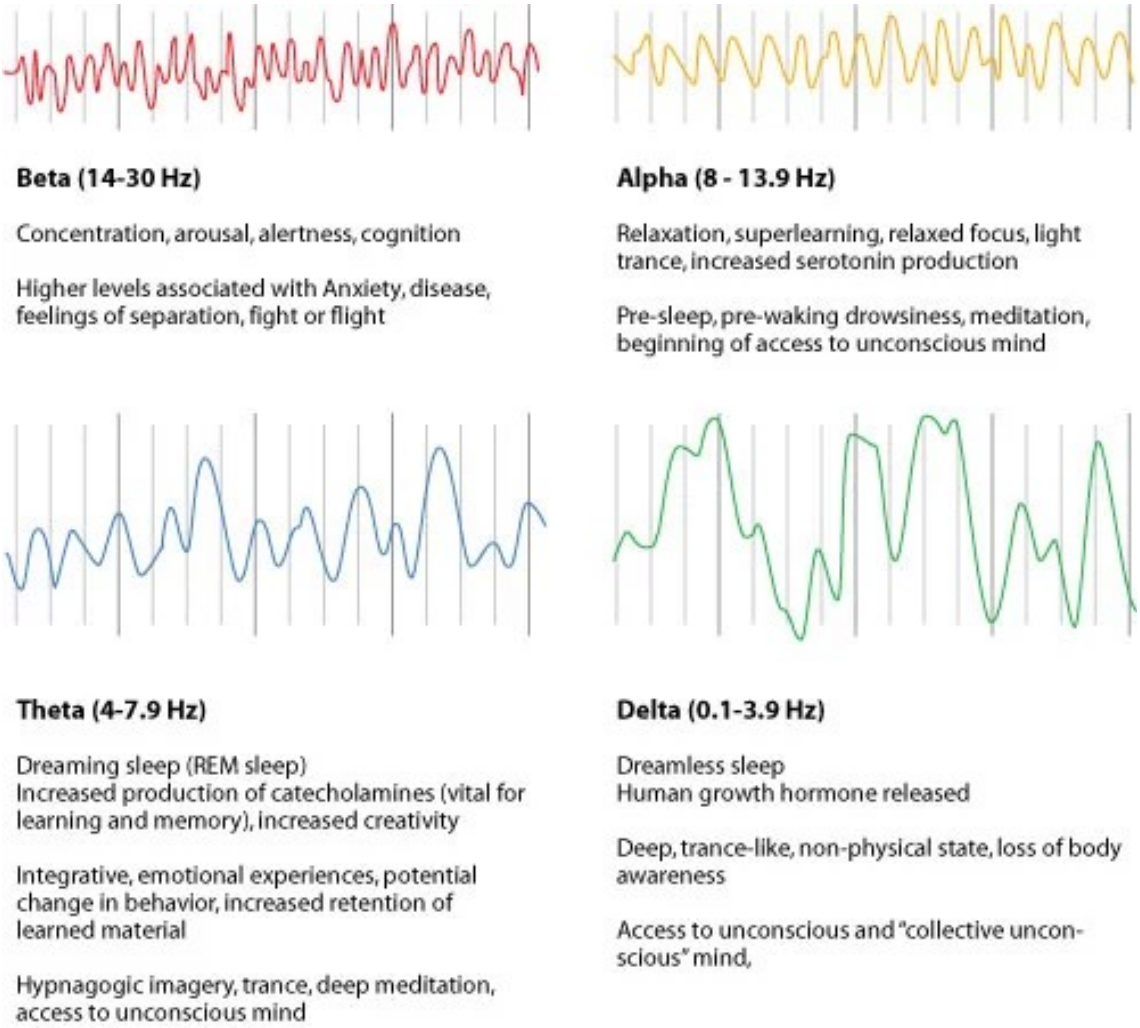
Elektroencefalografie, zkráceně EEG je diagnostická metoda záznamu elektrické aktivity mozku[1]. Probíhá měřením potenciálů na povrchu hlavy pomocí soustavy elektrod. Tyto potenciály mají amplitudu v řádech desítek μV a je zapotřebí je zesílit a odstranit šum. Výsledný záznam vzniká synchronizovanou činností velkého množství neuronů. Tyto signály se obtížně izolují.

Nejvíce známou složkou EEG vln je **aktivita pozadí**. Ta má několik typů. Ty se liší frekvencí, amplitudou vln a výskytem na povrchu hlavy. Podle přítomnosti či intenzity můžeme posoudit stav vědomí měřeného. Tyto vlny se střídají v průběhu dne[2]. Na obrázku 1 jsou vidět průběhy základních typů aktivity pozadí.

- **Beta aktivita** (14-40 Hz). Je spojena s běžným bdělým stavem, s logickým myšlením, řešením problémů, koncentrací, mentální aktivitou. Během aktivní konverzace, sportu a pracovního nasazení se nacházíme ve stavu beta. Vyšší úrovně této aktivity zvyšují hladinu stresu úzkosti a neklidu. Mnoho lidí tráví v beta stavu většinu bdělého života, výsledkem toho je, že žijí ve velkém stresu. Beta stav je ale důležitý pro účinné fungování v každodenním životě.

- **Alfa aktivita** (8-13 Hz). Oproti aktivitě beta souvisí s uvolněnějším stavem vědomí. Tato aktivita se projevuje během relaxace nebo tvořivé činnosti. Aktivita podporuje představivost, učení a práci s pamětí. Meditace vede ke stupňování vln alfa. U dětí jsou vlny alfa silnější než u dospělých. Aktivita je silnější při zavřených očích. Alfa vlny jsou považovány za nejzdravější stav mozku v bdělém stavu.
- **Théta aktivita** (4-7 Hz). Tyto vlny se aktivují během hluboké relaxace a meditace, lehkého spánku nebo denního snění. Objevuje se během REM fáze spánku. Může způsobovat živé představy, hluboká vnuknutí, trans a přístup do podvědomí. Meditace a jóga jsou uklidňující, protože navozují stav, v němž mysl začne generovat vlny théta.
- **Delta aktivita** (0.1-3 Hz). Nejpomalejší vlny a mají nejvyšší amplitudu (20-200 μV). Mozek je generuje během hlubokého, bezesného spánku. Informace přijímané v tomto stavu nejsou dostupné na vědomé úrovni. Tyto vlny jsou nezbytné pro regeneraci a samoléčivé procesy těla. Jsou dominantní u novorozenců do dvou let.
- **Gama vlny** (36-44 Hz). Nejrychlejší frekvence na níž dokáže mozek fungovat. Umožňuje náhlá jasnozření a vysoce efektivní zpracování informací. Aktivita v nás vzbuzuje pocit, že dokážeme všechno.
- **Lambda vlny** - pozitivní nebo negativní ostrá vlna ve tvaru písmene lambda. Souvisí se sledováním ostře osvětlených předmětů nebo při otevření očí.

Během měření se můžeme setkat s artefakty nepocházejícími z mozkové aktivity. Zdroje těchto artefaktů mohou být například mrkání očí, srdeční aktivita a externí zdroje jako např. elektrická síť (50 Hz) nebo elektromagnetické pole od okolních přístrojů. Tyto artefakty mohou kontaminovat EEG měření.



Obrázek 1: Průběhy základních typů mozkové aktivity[2]

2.2 Evokované potenciály

Další složkou mozkových vln jsou změny způsobené podněty z vnějšího prostředí. Nazýváme je Evokované potenciály, zkráceně EP. Za normálních okolností jsou tyto reakce okamžité. Při poruše dochází k opoždění. Tyto evokované potenciály dále dělíme podle typu podnětu, který je vyvolal. Do evokovaných potenciálů řadíme i potenciály šířící se nervovou soustavou mimo mozek[3].

- **Zrakově evokované potenciály** (zkr. **VEP** z angl. Visually evoked potentials). Jsou EP vyvolané krátkým vizuálním stimulem. Zrakově můžeme stimulovat pomocí záblesků (flash VEP, FVEP), strukturované šachovnice, kde dochází ke změnám na základě určitého vzoru (pattern reversal) a nebo pohybově (motion VEP). Tyto jsou vhodné pro tvorbu BCI (Brain-

computer interface), více v kapitole 2.4. Při snímání používáme 3 elektrody v oblasti zrakového centra.

- **Sluchové evokované potenciály** (zkr. **AEP** z angl. Acoustic EP). Malé elektrické napěťové potenciály směřující z ušního hlemýždě do mozku. Evokujeme pomocí zvuku generovaného z vnějšího prostředí. Můžeme měřit poškození sluchu. Tato měření dosahují vyšší objektivity než jiné diagnostické metody. Elektrody jsou umístěné poblíž ucha.
- **Motorické evokované potenciály** (**MEP**). Tyto potenciály vznikají stimulací pyramidových buněk (buňky v mozkové kůře a jiných částech mozku) nebo magnetickou stimulací eferentních motorických drah. Testujeme funkční integritu motorických drah. Stimulovat můžeme i kořeny nad krční nebo bederní míchou pro rychlejší reakci než z mozkové kůry.
- **Somatosenzorické evokované potenciály** (**SEP**). Pomocí kontrakcí svalů vyšetřujeme periferní části nervové soustavy. Využíváme k detekci mozkového a míšního poškození. Stimulace provádíme silným elektrickým impulzem, laserem či krátkým proudem vzduchu na mechanoreceptory.
- **Kognitivně evokované potenciály** (zkr. **ERP** z angl. Event Related potential). Jejich měření se věnuje následující kapitola.

2.3 Měření ERP

Jedním z typů Evokovaných potenciálů jsou Kognitivně evokované potenciály, anglicky Event-related potential (ERP). Jedná se o elektrickou odezvu mozku na událost na základě vnější stimulace různého původu (obrazová, zvuková nebo senzorická). Pro záznam ERP nemusíme používat takové množství elektrod jako pro celkový záznam EEG. Hlavní je získat čistá data[4]. To může být dost těžké, protože tyto potenciály jsou velmi slabé a jsou zašuměné aktivitou pozadí. Menší množství elektrod cílené na oblasti mozku, které stimulujeme, je pro nás výhodnější.

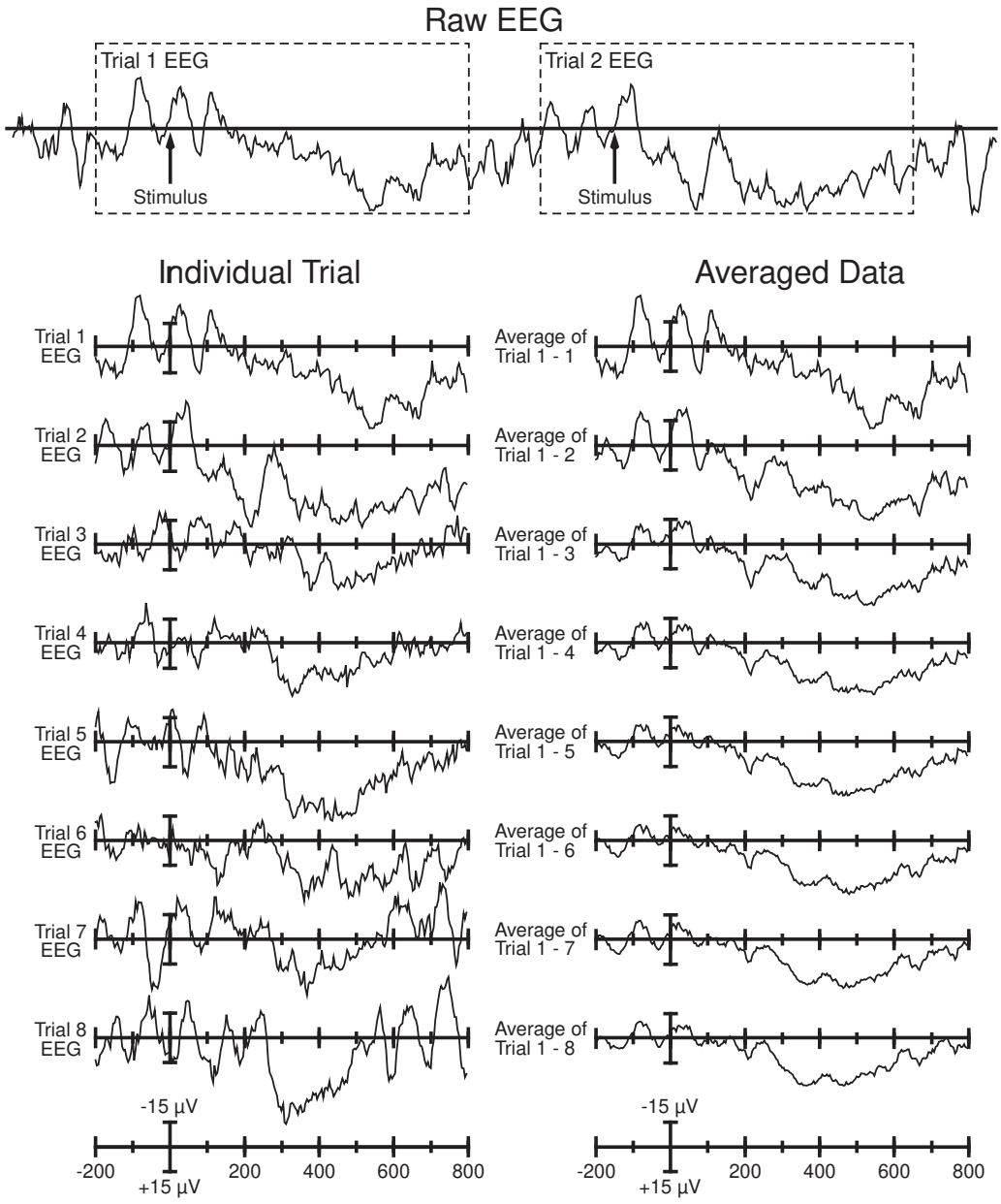
Jednou z možností, jak ERP od pozadí izolovat, je průměrování. Na obrázku 2 je vidět, jak se průměrná vlna rýsuje s každým průběhem. Průměrování funguje za předpokladu, že každá ERP reakce v průměrované sadě probíhá stejně, takže je nutné subjekt stimulovat stejným podmětem opakovaně. Je také zapotřebí do záznamu přidat značky stimulů (markery), podle kterých se pokládají vlny na sebe. Pokud nejsou tyto značky synchronizovány

přesně se stimulem, pak to způsobí rozmazání celkové ERP vlny. Proto je důležité, aby opoždění stimulu bylo co nejmenší nebo alespoň konstantní.

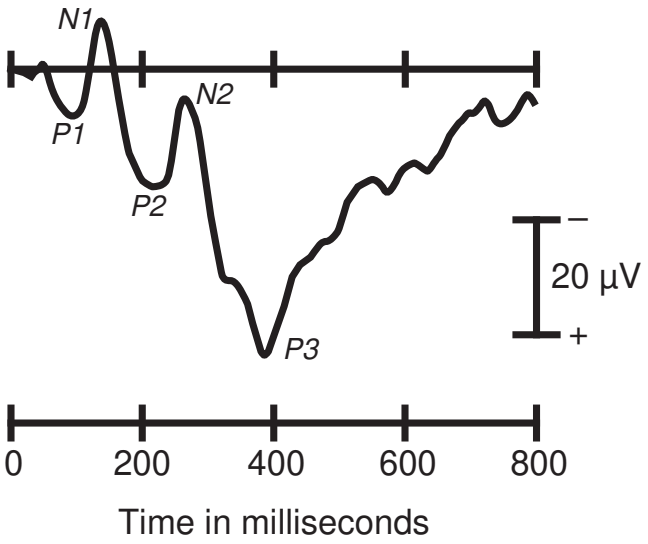
Stimulovat můžeme zrakové ERP (měnící se obrazy jednoduchých tvarů, písmena, blikající obraz). V tomto případě umísťujeme elektrody na týl hlavy. Sluchové ERP evokujeme krátkým tónem. Elektrody umísťujeme za levé a pravé ucho a na vrchol hlavy. Podle charakteru ERP se mění i jeho složení.

Z ERP měření pak zkoumáme tzv. ERP komponenty. Tyto komponenty chápeme jako specifické vychýlení signálu určitým směrem. Charakter ERP mění sadu komponent, které můžeme očekávat. Sluchový ERP generuje úplně jiné komponenty než zrakový. Ukázka komponent konkrétního průběhu zrakového ERP je na obrázku 3. Zvláštní konvencí ve světě ERP výzkumu je vykreslovat pozitivní signál směrem dolů na rozdíl od zbytku vědeckého světa. Některé zajímavé komponenty:

- **C1** první komponenta po stimulu (vrchol v 50-80 ms). Může být záporná i kladná, má velmi malou amplitudu a hodně závisí na vlastnostech stimulu. U vizuálního stimulu je závislá na poloze stimulu. Pokud se vyskytne v horním zorném poli, získá záporný potenciál, v dolním pak kladný.
- **P3** vrcholí obvykle 300 ms po stimulu. Má pozitivní potenciál. Objevuje se pokud subjekt neví jaký bude následující stimul a bývá silnější v případě překvapení. Bývá snadno rozpoznatelný a v počátcích výzkumů ERP byl předmětem velkého množství studií. Můžeme ho vyvolat tím, že budeme mít dva stimuly, třeba písmena na obrazovce. Méně časté písmenko bude provázeno silnější P3 komponentou. Naopak očekávaný stimul vyvolá slabší P3.
- **N400** je jazykově založená komponenta. Vrchol je přibližně 400 ms po stimulu a má záporný potenciál. Objevuje se například pokud vypisujeme větu na monitor slovo po slovu. Na konci věty pak zaznamenáme N400. Podobně jako P3 bude N400 intenzivnější pokud se na konci věty objeví nečekané či nesmyslné slovo.



Obrázek 2: Proces průměrování na sadě měření[4]



Obrázek 3: ERP komponenty[4]

2.4 Měření BCI VEP

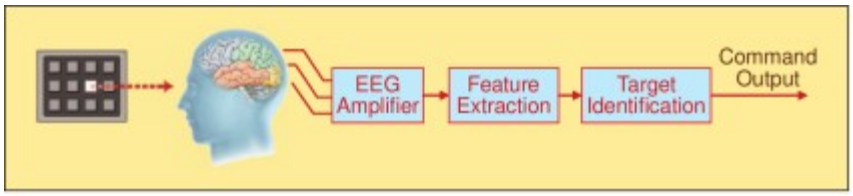
Brain-computer interface je rozhraní, které převádí úmysly na kontrolní signál a vytváří tak přímé spojení mozku s externím zařízením. Umožňuje tak nový způsob komunikace osobám, jejichž periferní nervová soustava je poškozena. Jednou z možností jak BCI realizovat, je pomocí systému využívající VEP (vizuálně evokované potenciály) pro stimulaci mozku[5].

Stimulaci provádíme v centrálním zorném poli pro evokování silnějších potenciálů. Stimulujeme pomocí více stimulačních cílů (targets). Ty mají unikátní stimulační sekvenci, která vyvolá odlišný VEP. Analyzováním charakteristik čtených potenciálů můžeme identifikovat, na který cíl se subjekt soustředí. Na obrázku 4 je vidět schéma takového BCI systému. Existuje více přístupů jak od sebe cíle odlišit.

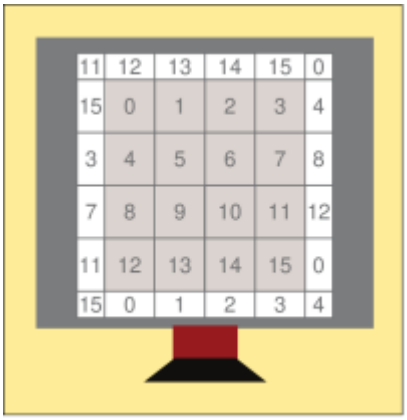
- **t-VEP** (z angl. time moduled VEP, časově modulované VEP). Pomocí FVEP (blikající potenciály) vyblikává krátkou sekvenci unikátní pro daný cíl. Soustředění na konkrétní cíl nám zesílí odezvu souvisejícího VEP. Pro identifikaci cíle v zaznamenaném EEG je zapotřebí přidat synchronní značky začátků cílů. Pro přesnou identifikaci cílů v t-VEP je zapotřebí průměrování několika měření, podobně jako při měření ERP. Jednotlivé cíle jsou prezentovány postupně za sebou. Překrytí jednotlivých cílů se předchází pomocí nižší frekvence blikání. Díky tomu nedo-

sahuje t-VEP takové přenosové rychlosti jako následující systémy.

- **f-VEP** (z angl. frequency moduled VEP, frekvenčně modulované VEP). Každý cíl periodicky bliká o určité frekvenci. Cíle jsou prezentovány spolu. Subjekt se soustředí na konkrétní cíl, čímž vyvolá VEP o stejné frekvenci. Identifikaci cíle pak provádíme spektrální analýzou (například Rychlou Fourierovou transformací) signálu z EEG. V záznamu jsou pak prudké vrcholy na stimulované frekvenci a jejích harmonických násobcích. Výhodou tohoto systému je jednoduchá konfigurace (systém nepotřebuje synchronizační značky v záznamu), není zapotřebí téměř žádný trénink, odpadá nám trénovací fáze a dosahuje vyšší přenosové rychlosti informací (30-60 bitů/min).
- **c-VEP** (z angl. pseudorandom code moduled VEP, pseudonáhodně kódované VEP). Monitor je rozdělený na šachovnici, kde každý cíl představuje okénko. Tato šachovnice je organizována tak, aby se okraje opakovali a cíle měli vždy stejné sousedy. Na obrázku 5 je vidět, jak je tato šachovnice na monitoru organizována. Každý cíl pak bliká stejnou pseudonáhodnou sekvencí, ale s jiným posunutím. Pro identifikaci cílů se využívá porovnávání šablon, získaných z trénovací fáze experimentu. Při trénování je subjekt instruován k soustředění na určitý cíl. Šablona se získá průměrováním několika průběhů sekvence. Tato technika se stejně jako t-VEP musí synchronizovat s EEG záznamem pomocí značek. Přenosová rychlost získaných informací z c-VEP může dosahovat rychlostí i přes 100 bitů/min.



Obrázek 4: Schéma BCI systému založeném na VEP[5]



Obrázek 5: Rozmístění cílů u c-vep BCI systému[5]

2.5 Stimulátory

Pro měření ERP a VEP budeme potřebovat kvalitní zdroj stimulů. Dostupné řešení se dělí na dva typy.

- **Softwarové stimulátory** běží jako samostatný program na PC, což sebou nese problémy se zpožděním zobrazení stimulu a zpracováním odezvy. Tyto opoždění se mohou na různých počítačích lišit a synchronizace je obtížná.
- **Hardwarové stimulátory** jsou většinou součást záznamového zařízení. Uzavřenost systému neumožňuje rozšířit sadu experimentů. Často chybí výstup pro externí synchronizaci, takže jsme závislí na použití vestavěného záznamového zařízení.

Softwarové stimulátory se většinou zaměřují na psychologické testy obecně. Srovnám některé softwarové stimulátory a vypíchnu některé jejich vlastnosti.

- **Presentation** je placený softwarový stimulátor od firmy Neurobehavioral Systems. Umožňuje prezentovat 2D i 3D vizuální stimuly i zvukové stimuly separátně nebo současně. Dokáže synchronizovat obraz s vertikální synchronizací a přesně prezentovat stimuly. Pracuje s celou škálou externích zařízení. Poskytuje velké množství úprav stimulů pomocí různých efektů. Pro tvorbu experimentů je možné použít vlastní jazyk, grafická rozhraní nebo python rozšíření[6]. Jeho hlavní nevýhodou je vysoká cena.
- **OpenViBE** je opensource platforma na testování BCI (brain-computer interface). Hlavním cílem OpenViBE byla tvorba VR

(virtuální reality) ovládané pomocí BCI. Nyní se snaží OpenViBE pokrýt širší záběr psychologických experimentů a neomezuje se jen na VR. Umožňuje připojení velkého množství záznamových zařízení. Už základní balíček obsahuje několik scénářů pro záznam a stimulaci ERP[7]. Pro skriptování používá jazyk LUA.

- **OpenSesame** je opensource program pro tvorbu experimentů pro psychologii, neurovědu a experimentální ekonomiku. Experimenty se v něm tvoří v jazyce python. Pro tvorbu markerů na paralelním portu používá plugin parallel_port_trigger[8].
- **PEBL** neboli The Psychology Experiment Building Language je poměrně nový opensource software pro tvorbu psychologických experimentů. Jeho uživatelské rozhraní i prezentační část jsou vytvořeny v knihovně SDL. Jeho největší předností je PEBL Test Battery, což je rychle se rozšiřující balík experimentů psaný komunitou. Tyto testy jsou snadno upravitelné díky tomu, že PEBL používá vlastní jednoduchý stejnojmenný jazyk, vytvořený přímo pro psaní experimentů. Od verze 0.12 je možné používat paralelní port a je tedy možné do testů přidat markery stimulů pro záznamové EEG zařízení[9].

2.5.1 Stimulátor ZČU

Stimulátor na Západočeské univerzitě se snaží řešit nedostatky současných řešení. Je koncipován jako externí hardwarové zařízení pro vizuální a akustickou stimulaci. Oproti současným hardwarovým řešením má výhodu, že jeho funkcionalitu lze snadno rozšířit úpravou firmwaru. Lze ho připojit k libovolnému záznamovému zařízení, které je vybaveno vstupem pro externí synchronizaci.

Srdcem stimulátoru je levný, ale dostatečně výkonný mikrokontrolér architektury ARM cortex-m3. Díky rychlé obsluze přerušení a časovači je toto zařízení přesnější než softwarová řešení. Součástí stimulátoru jsou vývody na připojení LED panelů, externí tlačítka, USB port pro připojení PC a bluetooth modul pro připojení mobilního telefonu.

Stimulátor umožňuje ERP stimulaci, kde externí LED panely prezentují jednotlivé stimuly. Stimulům můžeme nastavit různou distribuci, délku a různou intenzitu jasu. Obsahuje také nastavitelné t-VEP, f-VEP a c-VEP stimulace. U f-VEP lze nastavit až 4 nezávislé cíle o frekvenci 0 až 20 Hz s krokem 0,5 Hz. c-VEP stimulace umožňuje až 8 cílů a délka nastavitelného vzoru je 32 bitů. Také umožňuje realizovat reakční experiment a měřit rychlost reakcí pomocí

připojitelných tlačítek. Tento experiment uživateli zaznamenává průměrný čas a čítače chybných a promeškaných stisků.

Stimulátor umožňuje externí nastavení pomocí připojeného zařízení. Pomocí USB nebo UART sběrnice je možné přenést parametry experimentů. Toto spojení je více popsané v kapitole 3.5. Tímto způsobem je možné připojit mobilní aplikaci pro operační systém Android vyvinutou Petrem Štechmüllerem. Aplikace komunikuje se stimulátorem pomocí bluetooth modulu.

Na vrchním panelu přístroje je dotykový displej, umožňující nastavení parametrů experimentů a jejich spuštění a vypnutí. Původně měl tento displej sloužit i k vizuální stimulaci, ale komplikované programování samostatného mikrokontroléru v displeji se ukázalo jako překážka. Stejně tak ovládání externího monitoru pomocí VGA modulu vykazovalo značné problémy (vykreslování na monitor bylo pomalé).

Řešení těchto nedostatků je předmětem této práce. Spočívá v nahrazení dotykového displeje a obrazového výstupu pomocí levného mikropočítače Raspberry Pi. Více v kapitole 3.2.

3 Návrh

3.1 Mikropočítač Raspberry Pi

Raspberry Pi je počítač o velikosti kreditní karty. Obsahuje USB vstupy pro periferie, výstup na HDMI monitor, LAN konektor a od verze 3 i integrovanou Wi-Fi a bluetooth. Toto je celkem standardní vybavení běžného PC. Raspberry ale přináší navíc patici GPIO. Použití těchto pinů bylo doménou embedded zařízení a spotřební elektroniky. Právě přidáním této patice, je to, co dělá z Raspberry revoluční produkt. Umožnilo totiž programátorům naplno využít senzory a periferie, které do té doby byly přístupné pouze pro mikrokontroléry. Raspberry Pi obsahuje grafický čip, umožňující zobrazování grafiky a přehrávání videí. Počítač umožňuje běh standardního operačního systému pro ARM architekturu, např. Linuxové distribuce.

Jádrem Raspberry Pi je SoC (system on chip) od společnosti Broadcom, který je novějšími modely postupně vylepšován. Současná varianta v modelu Raspberry 3 B+ obsahuje 4-jádrový mikroprocesor ARM Cortex-A53 bežící na 1.4GHz a 1GB RAM.

Od vydání prvního Raspberry Pi se mnoho výrobců snažilo Raspberry Pi napodobit. Málokterý jednodeskový počítač se ale těší takové podpoře komunity jako právě Raspberry Pi.

3.2 Změny v uspořádání stimulátoru

Na obrázku 6 je blokové schéma stimulátoru. Jak můžeme vidět podle původního návrhu má LCD modul obstarávat, jak funkci nastavení parametrů, tak poskytovat vizuální stimulaci. V současném stavu je však implementováno pouze nastavení parametrů. Pro změny v uživatelském rozhraní je zapotřebí naprogramovat firmware pro MCU (mikrokontrolér) v LCD modulu pomocí nástrojů výrobce a patřičně upravit i firmware hlavního MCU. To komplikovalo vývoj stimulátoru.

Řešení můžeme najít v odstranění LCD modulu a nahrazení jeho funkcí pomocí Raspberry Pi. Uživatelské rozhraní pak nahradí běžná aplikace naprogramovaná pro operační systém běžící na Raspberry Pi. Uživatel bude s tímto rozhraním operovat pomocí standardních počítačových periferií (klávesnice, myši a monitoru). Parametry z aplikace pak můžeme předat stimulátoru pomocí sběrnice UART. Aplikace může i zobrazovat vizuální stimuly na počítačový monitor. Pro tuto stimulaci však nestačí předávat parametry UARTem. Čekání na UART přenos by nebylo dostatečně rychlé. Proto budeme stimuly předávat pomocí sběrnice GPIO, která umožňuje téměř okamžitou reakci.

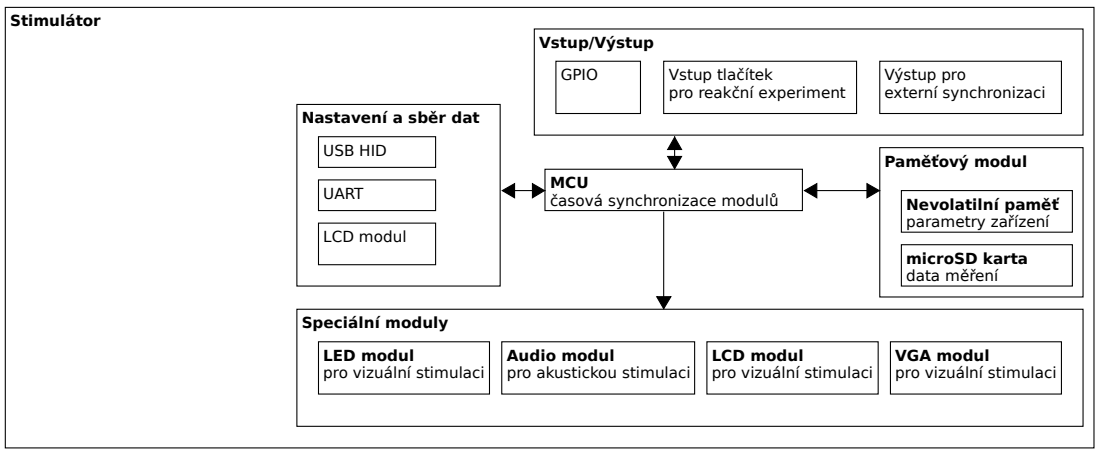
Blokové schéma těchto změn je na obrázku 7. Byl odstraněn LCD modul, audio modul a VGA modul. Raspberry se připojuje ke stimulátoru pomocí sběrnic UART a GPIO. Audio modul z obr. 6 by mohl zůstat, ale pro jednodušší správu potřebných souborů pro obsluhu akustických stimulů, bude lepší nechat jejich obsluhu na Raspberry Pi. Díky tomu budou soubory, jak pro obraz, tak pro zvuk pohromadě na Raspberry Pi. Odstranění VGA modulu je logické, protože slouží stejnému účelu jako Raspberry Pi – připojení monitoru.

Připojení k GPIO bude realizováno následovně. Aby bylo možné přenést až 8 stimulů, bude jeden GPIO pin představovat počátek a konec stimulu. Zbylé tři piny představují adresu stimulu. Adresní piny se nastaví před změnou hrany pinu, představujícím počátek stimulu. Nastavení potřebných obrazů a zvuků těchto stimulů bude realizováno v aplikaci. V případě použití těchto rozšířených výstupů nebude stimulátor „vědět“ co zobrazuje.

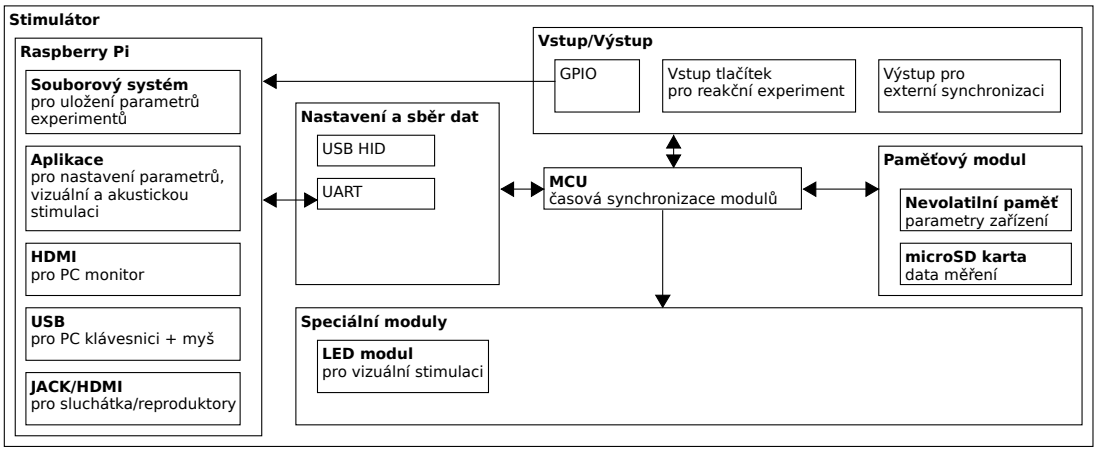
Raspberry Pi pak bude sdílet stejné pouzdro jako stimulátor a stává se tak de facto součástí přístroje. Ze stimulátoru se tak stává hybridní zařízení, částečně hardwarový a částečně softwarový stimulátor. Kritické časování poskytuje přesnější mikrokontrolér a grafiku a PC periferie obsluhuje výkonné Raspberry Pi.

Programování takové aplikace bude jistě o mnoho snazší, protože bude usnadněno o množství dostupných knihoven pro programování běžných počítačových aplikací. Množina funkcí, které je možné takovou aplikací obsluhovat, je silně závislá na možnostech protokolu, kterým lze stimulátor ovládat.

Ke stimulátoru je možné připojit mobilní aplikaci (přes bluetooth modul, zapojený do UART sběrnice) a nastavovat v ní parametry experimentů. Abychom ji nepřipravili o možnosti vizuální a akustické stimulace pomocí Raspberry Pi, musíme jí umožnit předání parametrů a potřebných souborů. Tato mobilní aplikace je na uvedené změny připravena a umožňuje komunikovat pomocí rozšíření komunikace popsané v kapitolách 3.5.1 a 3.5.2.



Obrázek 6: Blokové schéma stimulátoru



Obrázek 7: Blokové schéma stimulátoru po přidání Raspberry Pi

3.3 Případy užití

Nyní se pokusím představit scénáře, které by měla aplikace splňovat. Scénáře užití obsahují i popisy přenosů přes datové sběrnice. Popisované scénáře odpovídají schématům užití na obrázcích 8, 9 a 10.

3.3.1 Aplikace jako konfigurátor stimulátoru

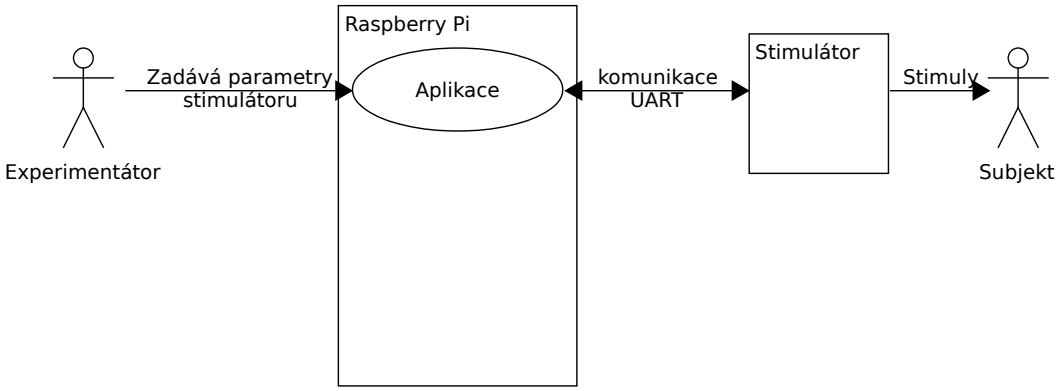
Aplikace bude v roli konfigurační utility. Experimentátor vytvoří v grafickém rozhraní aplikace nastavení experimentu. Po stisknutí tlačítka **START** se přes sběrnici **UART** přenese nastavení do stimulátoru a posledním příkazem se zapnou výstupy stimulátoru (standardně LED matice). Poté může experiment ukončit stiskem tlačítka **STOP**, který odešle příkaz k vypnutí výstupů stimulátoru.

3.3.2 Aplikace jako rozšíření výstupů stimulátoru

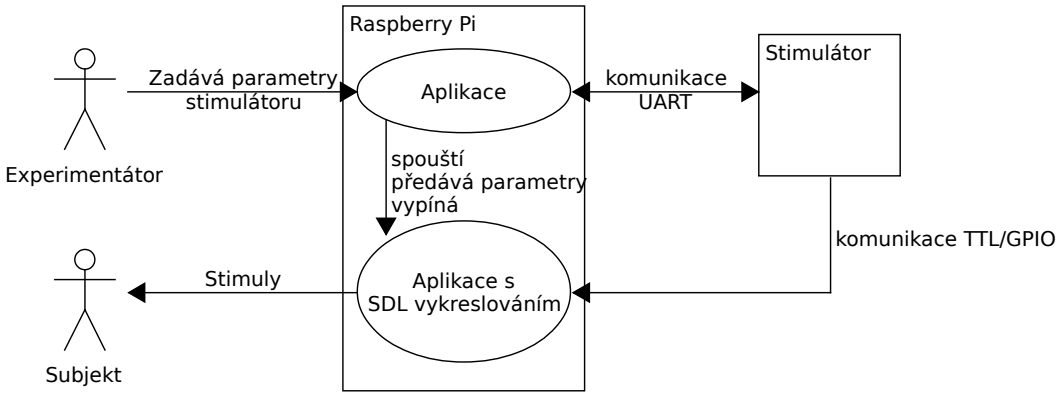
V tomto případě bude Raspberry Pi číst informace o probíhajícím stimulu ze sběrnice GPIO a aplikace je bude vizuálně či zvukově prezentovat. Subjekt je stimulován vizuálními stimuly na monitoru připojenému k Raspberry Pi. V tomto případě se počítá se zpožděním aplikace a je možné ho kompenzovat pomocí vyslání stimulu ze stimulátoru v předstihu, za předpokladu, že budeme znát průměrné opoždění aplikace a tato prodleva bude víceméně neměnná. Při stisku tlačítka **START** v aplikaci se kromě nahrání konfigurace spustí zobrazování výstupů na monitor.

3.3.3 Aplikace jako rozšíření mobilní aplikace

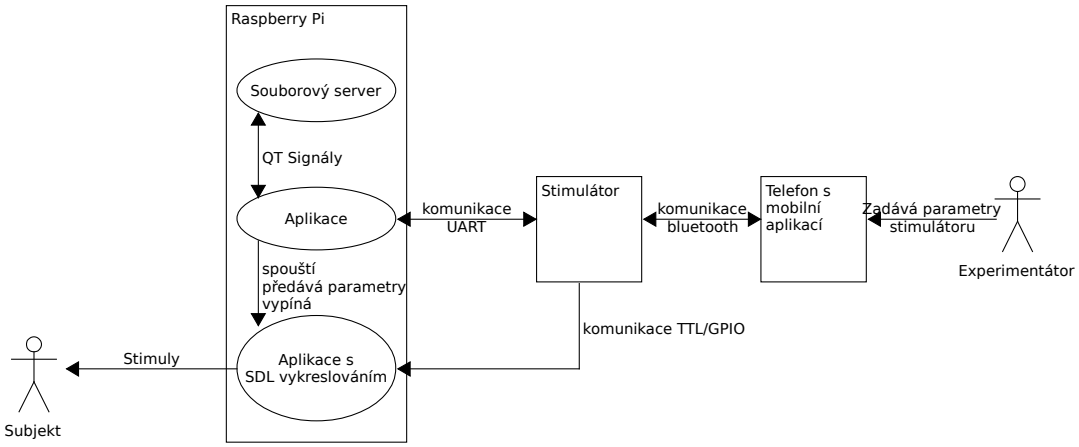
Aby měla mobilní aplikace přístup k rozšířením z minulého scénáře, je zapotřebí nějakým způsobem předat potřebné informace. V tomto případě experimentátor nastaví mobilní aplikací parametry stimulátoru. Stimulátor a mobilní zařízení jsou propojeny pomocí **bluetooth**. Stimulátor předává zprávy přes **UART** do aplikace běžící na Raspberry Pi. Přenosová rychlost těchto sběrnic je velmi pomalá, proto je zapotřebí potřebné obrázky a zvuky nahrát v předstihu před spuštěním experimentu. Pro tuto potřebu bude součástí aplikace i souborový server. V momentě spuštění experimentu musí aplikace vyslat signál nejen stimulátoru, ale i aplikaci na Raspberry Pi, aby mohlo proběhnout spuštění zobrazování na monitor.



Obrázek 8: Schéma užití - aplikace jako konfigurátor stimulátoru



Obrázek 9: Schéma užití - aplikace jako rozšíření výstupů stimulátoru



Obrázek 10: Schéma užití - aplikace jako rozšíření mobilní aplikace

3.4 Souborový server

Aby mohlo mobilní zařízení využít obrazových a zvukových stimulů, musí mít možnost nahrát potřebná obrazová a zvuková data na diskový prostor Raspberry Pi. K tomuto účelu bude sloužit souborový server, který bude součástí aplikace. Tento server bude umožňovat následující operace:

- **HELLO** oznámení o přihlášení nového zařízení. Pro server to znamená zahození nedokončených zpráv a vynulování čítačů.
- **BYE** ohlášení je spíše pro úplnost. Funkci neplní zatím žádnou.
- **MD** vytvoření adresáře.
- **LS** vypsání adresáře.
- **GET** stažení souboru.
- **PUT** upload souboru.
- **DEL** smazání souboru.
- **START** zapne grafický výstup aplikace. Tato operace se volá před spuštěním samotného experimentu ve stimulátoru a mobilní zařízení tím informuje Raspberry Pi, že se chystá spustit experiment.
- **STOP** vypne grafický výstup. Ukončení experimentu.
- **GET_PREVIEW** vrací náhled obrázku.

Každá operace vrací patřičnou odpověď, buď OK nebo číslo návratové chyby. Server sám od sebe nic nedělá, veškerá aktivita musí začínat

u klienta. Následující odstavce popisují současný stav komunikace se stimulátorem a jak tento protokol rozšířit, aby bylo možné pracovat se souborovým serverem.

3.5 Komunikační protokol

Stimulátor umožňuje nastavení pomocí externích zařízení. Ty lze připojit pomocí rozhraní, které má stimulátor k dispozici:

- **USB,** v tomto případě se komunikuje ve formě **USB HID**. Toto rozhraní se používá při připojení PC.
- **UART** se základní rychlostí 9600 baudů za sekundu. Tímto rozhraním je připojeno Raspberry Pi, které je ve stejném pouzdře jako stimulátor.
- **Bluetooth** tímto rozhraním připojujeme mobilní zařízení. Pro stimulátor se jedná o stejné zařízení jako předchozí. O samotnou bluetooth komunikaci se stará bluetooth modul s UART rozhraním.

Stimulátor má vlastní komunikační protokol, kterým můžeme nastavovat parametry stimulátoru. Komunikace sestává z paketů, které mají konstantní délku, 64 bajtů. Jejich struktura je následující

```
[ 1B hlavička ][ 1B typ zprávy ][ 62B data ]
```

Hlavička obsahuje 6 bitovou délku zprávy a určuje počet platných datových bajtů. Typ zprávy určuje operaci, kterou chceme provádět. Některé operace vyžadují dodatečná data. Délka těchto dat nabývá hodnot 0 až 63. Pokud je délka nastavena na 63, pak to znamená, že data pokračují v dalším paketu. Celý protokol je možné prohlédnout v příloze A Protokol stimulátoru.

Současný stav protokolu umožňuje plně nastavit pouze experiment ERP, a to ještě v omezené míře. Například nastavit jas můžeme jen dohromady pro LED 5 a 6 a pro 7 a 8, protože se nastavují stejným příkazem. Doplněním protokolu by jsme mohli dosáhnout plné konfigurace všech experimentů, ale to by znamenalo rozsáhlé zásahy do firmwaru stimulátoru. Takovou rozsáhlou úpravu musí provést vývojář stimulátoru.

3.5.1 Rozšíření komunikačního protokolu

Komunikační protokol stimulátoru umožňuje snadné rozšíření a to přes rezervní nepoužívané typy zpráv. Pro naše rozšíření o přenos dat mezi Raspberry Pi (připojeném přes UART) a mobilním zařízením

(připojeném přes bluetooth) jsem se rozhodl zvolit co nejjednodušší strategii, aby doprogramování této funkcionality do samotného stimulátoru bylo pro vývojáře stimulátoru co nejjednodušší. Proto rozšíříme protokol o jedinou operaci a její implementace bude vypadat následovně. Pokud přijde paket této zprávy na rozhraní UART, tento paket bude zkopírován a odeslán na rozhraní bluetooth. Pokud přijde paket této zprávy na rozhraní bluetooth, bude tento paket zkopírován a odeslán na rozhraní UART. Protokol, který budeme používat na komunikaci mezi Raspberry Pi a mobilním zařízením, bude umožňovat vlastní rozdělení zpráv do menších paketů a proto nebudeme používat mechanismus protokolu stimulátoru pro rozdělení zpráv. Díky tomu budou mít všechny pakety délku 62 bajtů a tudíž hlavička bude mít při této komunikaci vždy hodnotu 62. Díky tomuto přístupu nepotřebuje stimulátor vědět obsah zpráv a nemusí tedy našemu protokolu rozumět. Bude pouze slepě předávat data mezi mobilním zařízením a Raspberry Pi.

3.5.2 Datový protokol (ke komunikaci se soubo-rovým serverem)

Tento protokol bude přenášen hostujícím protokolem stimulátoru a bude tedy také paketového typu. Pakety datového protokolu budou mít konstantní délku 62 a budou mít vlastní hlavičku, nezávislou na té vnější.

```
[ 2B protokol stimulátoru ][ 62B datový protokol]
```

První dva bajty jsou bajty z protokolu popsaného v předchozí části. Jejich hodnota bude konstantní délky zprávy 62 a operace datového protokolu. Samotný protokol bude vypadat následovně

```
[ 1B příznaky+OP kód ][ 1B iterátor][ 60B data]
```

kromě kódu operací (OP kód) se v první bajtu vyskytuje typ zprávy (dotaz, odpověď, upload nebo download) a také bit, který značí, zda zpráva pokračuje nebo zda se jedná o poslední zprávu.

Iterátor je libovolné číslo, zprávy s tímto číslem patří ke stejné operaci. Pokud například položím nějaký dotaz, odpověď bude mít stejný iterátor. Pro další dotaz je nutné použít nový iterátor. Je tedy na klientovi, aby iterátor postupně zvětšoval, aby mohl přiřadit odpovědi k zaslaným dotazům. Delší přenosy jsou zabezpečené MD5 hashem.

Celý protokol, včetně příkladů, je možné prohlédnout v příloze B Datový protokol.

4 Implementace

4.1 Použité technologie

4.1.1 Raspbian

Raspbian je distribucí Linuxu pro Raspberry Pi odvozené z Debianu. Linuxové distribuce se liší zaměřením softwaru, který je dodáván jako jejich součást. Máme například distribuci pro emulování starých her nebo distribuci obsahující multimediální centrum. V případě Raspbianu má toto zaměření širší záběr. V základu poskytuje Raspbian velmi základní software a umožňuje uživateli rozšířit tento dodaný software pomocí instalovatelných balíčků. Základní repozitáře Raspbianu obsahují přibližně 35000 takových balíčků. Instalace je distribuována ve dvou obrazech:

- **Raspbian with desktop** obsahuje kompletní desktopové prostředí pro nahrazení osobního počítače (internetový prohlížeč, prohlížeč obrázků, přehrávač videí), IDE pro výuku programování a mnoho dalšího softwaru.
- **Raspbian lite** velmi základní verze obsahuje pouze příkazový řádek a pár základních balíků. Tato varianta je spíše určená pro použití Raspberry Pi jako serveru s nějakým specifickým využitím, např. jako webový/souborový server nebo sběr dat ze senzorů či časosběr z kamery.

Jelikož moje aplikace vyžaduje grafické prostředí, rozhodl jsem se vycházet z **with desktop** varianty. Obrazy se do Raspberry instalují pouhým zkopírováním obrazu na SD kartu, například nástrojem **dd**.

4.1.2 QT

Multiplatformní knihovna pro objektové programování grafického uživatelského rozhraní (GUI). QT byla nedílnou součástí mobilního operačního systému Symbian a v dobách největší slávy tohoto systému ji vlastnila firma Nokia. Na QT je založeno linuxové grafické prostředí KDE. Knihovna značně rozšiřuje jazyk C++ a přidává vlastní datové typy a mechanizmus *signál/slot*, který umožňuje dynamicky propojovat různé části aplikace.

4.1.3 SDL

Multiplatformní knihovna pro přístup k grafickému hardwaru, zvuku a vstupnímu hardwaru jakožto klávesnice, myši a joysticky. Tuto knihovnu používá nepřeberné množství her (hlavně 2D), emulátorů

a přehrávačů. Pro rychlejší zobrazování může knihovna běžet v OpenGL režimu. V případě Raspberry Pi mě ale přišel hardwarový režim paradoxně o mnoho pomalejší, a proto jsem se rozhodl nechat si uživatele vybrat mezi hardwarovým renderem a softwarovým BlitSurface.

4.1.4 WiringPi

Velmi oblíbená C knihovna k manipulaci s GPIO na všech různých Raspberry Pi deskách. Navenek práce s ní připomíná arduinovskou základní knihovnu, ale uvnitř používá naprosto odlišné techniky. Pro přístup k pinům používá systémový nástroj **gpio**, pomocí kterého si exportuje do filesystému virtuální soubory prezentující stav pinů. Pro zaznamenání přerušení využívá knihovna *poll* těchto souborů. Pro urychlení čtení a zápisu může v novějších verzích také používat zařízení */dev/gpiomem* nebo */dev/mem*[10]. Jelikož je sama závislá na nástroji **gpio**, který vyžaduje rootovská práva, tak i tato knihovna vyžaduje tato vyšší oprávnění k vyexportování virtuálních souborů.

4.1.5 CMake

Psaní makefile (skriptů pro kompilaci projektů v jazyce C/C++) může být velmi náročné, obzvlášť pokud chceme vytvořit makefile pro více platforem. CMake je nástroj ke generování makefile. Umí například vyhledat knihovny v systému nebo můžeme pomocí podmínek přidat konkrétní zdroje v závislosti na prostředí. Díky tomu lze ošetřit, aby generovaný makefile byl co nejlepší pro konkrétní platformu.

4.2 Tvorba aplikace v QT frameworku

Základním stavebním kamenem uživatelského rozhraní QT aplikací jsou widgety. Okno, popisek i tlačítko jsou widgety. Všechny widgety jsou popsány třídami odvozenými od bázové třídy QWidget. Rodičem QWidget je QObject, který je bázovou třídou celého QT, obdobně jako máme v Javě třídu Object.

QObject je srdcem objektového modelu QT a poskytuje mechanismus signálu a slotu. Každý potomek QObject může mít definované speciální funkce tzv. „signály“, které mají prázdný kód a samy o sobě nic nedělají. Jejich voláním dávají objekty najevo, že došlo k určité události, např. stisk tlačítka. Protikladem k signálům jsou tzv. „sloty“, ty jsou také funkce, ale tyto už obsahují implementaci. Příkladem může být změna textu na widgetu popisku QLabel slotem *setText*. Aby mohly signály a sloty interagovat mezi sebou, musíme je nějak propojit. To provádíme funkcí *connect*. Její použití vypadá následovně

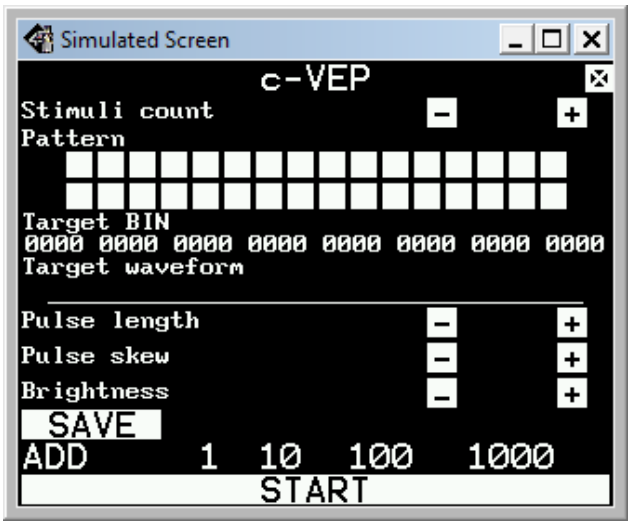
```
connect(odesílatel,SIGNAL(signál_odesílatele),příjemce,SLOT(slot_příjemce));
```

Výhodné na tomto principu je, že můžeme vytvářet mnohonásobná propojení (více slotů na jeden signál a naopak), již existující propojení rušit funkcí *disconnect* nebo blokovat signály funkcí *blockSignals*.

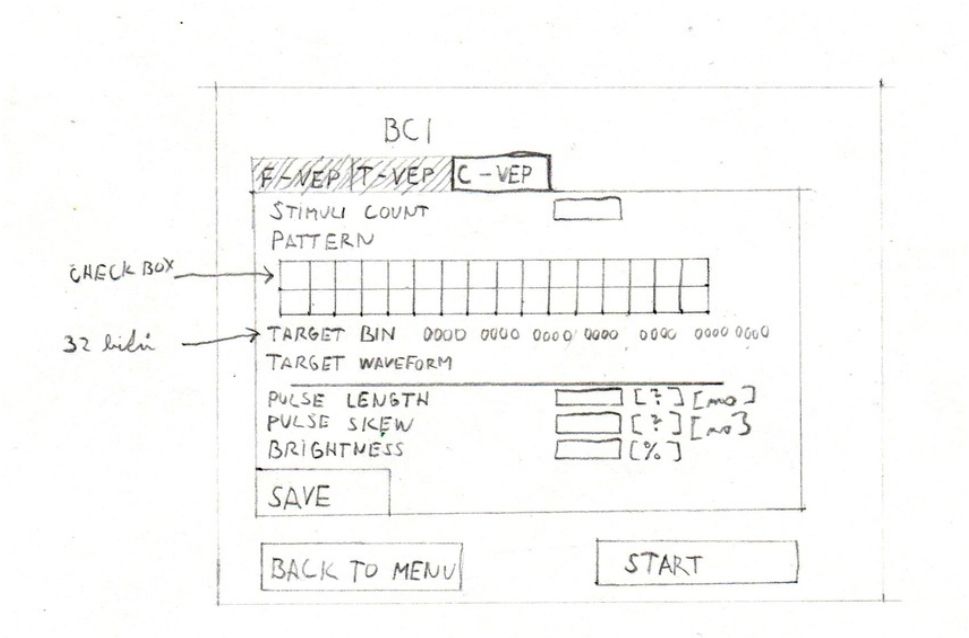
Samotné uživatelské rozhraní pak tvoříme skořápkovým principem. Widgetům, představujícím nějaký celek (okno, přepínací záložka, scrollovací oblast), nastavujeme rozložení funkcí *setLayout*. Do těchto rozložení pak můžeme přidávat další widgety. Těm můžeme dát znovu rozložení a tak můžeme pokračovat do nekonečna. Když vytvoříme příliš mnoho takových vrstev, stane se aplikace příliš komplikovanou. Zjednodušit jí můžeme například pomocí vlastních složených widgetů.

4.2.1 Podoba aplikace

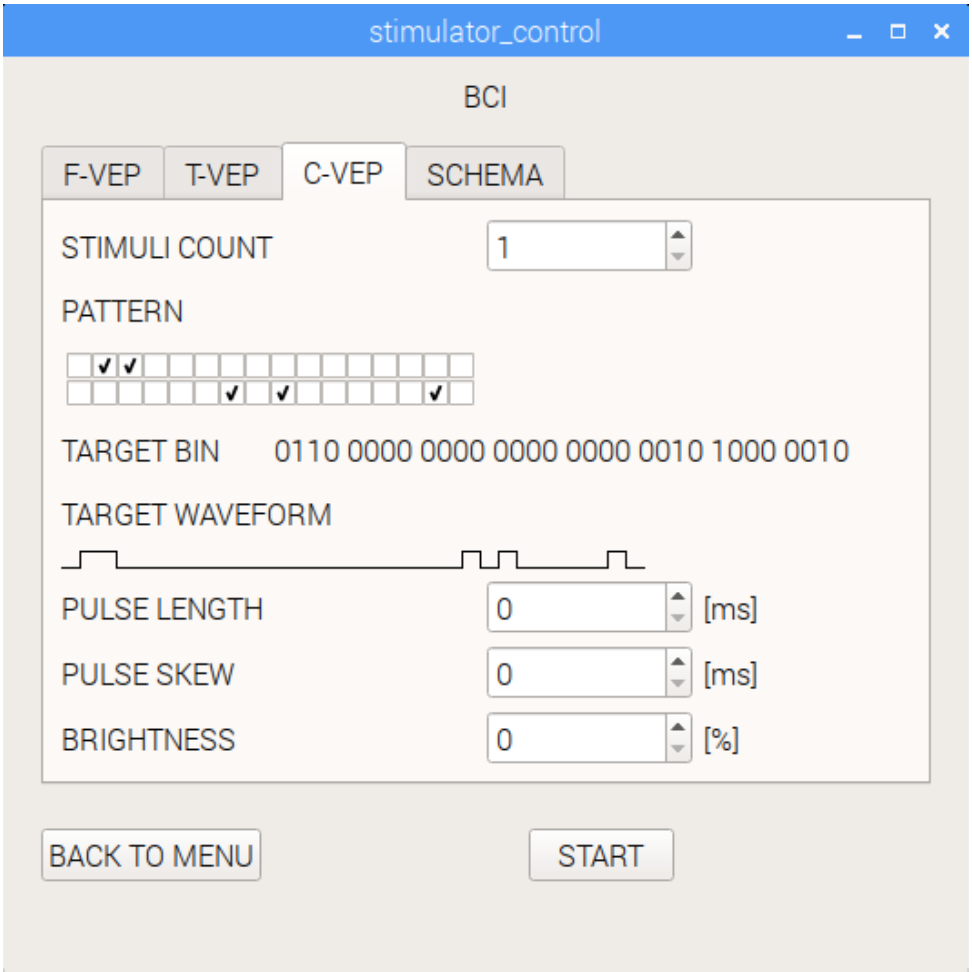
Při tvorbě rozhraní jsem se snažil držet původního rozhraní displeje stimulátoru jako předlohy. Část uživatelského rozhraní vznikla během práce na předmětu ZSWI. Ze snímků displeje stimulátoru (obr. 11) byl vytvořen návrh na papír (obr. 12) a podle něj vypracováno rozhraní aplikace (obr. 13), dokud se dostatečně nepodobalo návrhu. Na obrázcích je rozhraní experimentu c-VEP, hlavní část rozhraní se věnuje tvorbě pseudonáhodné sekvence.



Obrázek 11: Snímek displeje stimulátoru, experiment c-vep



Obrázek 12: Návrh rozhraní experimentu c-VEP



Obrázek 13: Implementace rozhraní experimentu c-VEP v aplikaci

4.2.2 Tvorba vlastních widgetů

QT obsahují velké množství widgetů. Ne vždy nám však stačí jejich funkcionalita a musíme si tedy vytvářet widgety vlastní. Speciálním případem vlastních widgetů jsou widgety složené[11]. Ty spojují již existující widgety do větších celků a usnadňují programátorovi práci s větším množstvím widgetů. Na obrázku 13 jsou vidět dva widgety vlastní výroby.

- **QCheckGrid** je složený widget obsahující více QCheckBoxů (třída zaškrtávacího pole v QT) uspořádaných do mřížky. Na obrázku je umístěn pod popiskem *PATTERN*. Tento widget organizuje QCheckBoxy do rozložení QGridLayout (mřížkové rozložení). Umožňuje například změnit počty řádků a sloupců bez ztráty informace, jak byla políčka zaškrtaná. QCheckGrid umožňuje přistupovat k hodnotám políček, ale ne k políčkům samotným. Ty jsou soukromými proměnnými QCheckGrid. V případě změny políčka uživatelem, widget pohltí signál políčka a vyvolá signál vlastní.
- **QWaveForm** je na obrázku pod popiskem *TARGET WAVEFORM*. Jedná se o widget vlastní grafikou. Vlastní grafiky dosáhneme tak, že přepíšeme funkci *paintEvent,* ve které kreslení naimplementujeme. Překreslení můžeme vynutit slotem *update*. Aby s takovým widgetem mohla pracovat rozložení, musíme přepsat i funkci *sizeHint,* ve které vracíme velikost widgetu. Podle této velikosti pak rozložení widget vhodně umístí.

4.3 Použití QSerial ke komunikaci se stimulátorem

Ke komunikaci se stimulátorem jsem se rozhodl použít knihovnu QSerial, která je součástí QT od verze 5. Tento modul QT plně zprostředkovává mechanizmus signálu a slotu programátorovi. Například přijatá data se obsluhují signálem. Umožňuje také získat informace o připojených sériových zařízeních a platných konfiguracích pro použitý systém.

Pro snazší práci s protokolem jsem vytvořil třídu Stimulator, která pohlcuje příchozí signály z QSerial a přidává je do bufferu. Když tento buffer dosáhne velikosti paketu, pak se zkontroluje jeho obsah a v případě, že se nejedná o vícepaketovou zprávu, vyvolá se signál s kompletní zprávou. Vícepaketové zprávy vyvolají signál po příchodu poslední části. K těmto signálům se pak můžou přihlašovat různé

části aplikace, které potřebují se stimulátorem komunikovat. Takto je obvykle připojen souborový server a aktivní experiment.

4.4 Oddělení SDL od zbytku programu

Aplikace vytvořené v QT frameworku jsou založené na událostmi řízené architektuře *(Event-driven Architecture).* To znamená, že po zavolání QApplication::exec() program vstupuje do nekonečné smyčky, ve které čeká na události (stisk klávesnice, pohyb myši, příchozí paket ze sítě) a tyto události jsou obvykle obsluhovány v hlavním vlákně programu.

Oproti tomu SDL knihovna nemá nic podobného a nechává konstrukci smyčky na programátorovi[12]. V případě hry programátor vytvoří smyčku, kde v každém cyklu přečte vstupy. Na základě těchto vstupů a kvanta uplynulého času upraví herní data, vykreslí jeden snímek na obrazovku a pokud zbyl čas vyhrazený na jeden snímek, pak čeká. Toto nám ale knihovna nediktuje a můžeme zkonstruovat smyčku jinak. V našem případě by bylo vykreslování 60 snímků za vteřinu mrháním. Jelikož zobrazujeme pouze statické obrázky, stačí nám vykreslovat pouze v případě změny vstupu.

Zde však v kombinaci těchto dvou knihoven nastává problém. QT po nás vyžaduje, po zpracování události, návrat do smyčky programu. Žádné čekání na vstupy není možné, protože by zastavovalo hlavní vlákno programu. Řešení spuštěním SDL v jiném vlákně je tedy nasnadě. Bohužel to není možné. I tato knihovna vyžaduje, aby byla spuštěna v hlavním vlákně. Je tedy nutné, aby byla část programu s SDL knihovnou oddělena od zbytku programu.

Vytvořením malého programu na obsluhu vykreslování událostí z GPIO, který budeme spouštět jako jiný proces, má více výhod. Knihovna wiringPi vyžaduje práva root, aby mohla nastavit přerušení z GPIO pinů. Můžeme tedy při spuštění tohoto malého programu použít systémovou utilitu *sudo* a tím získat potřebná oprávnění. Výhodou v případě Raspbianu je, že tento program po nás nechce heslo uživatele, jak tomu bývá v jiných distribucích Linuxu zvykem a nemusíme tedy zadávání hesla řešit. Při použití *sudo* ztrácíme možnost program ukončit jako jeho rodič. Proto k vypnutí programu používám kombinaci utilit *sudo pkill*.

Pro předání potřebných dat požívám XML soubor. Příklad tohoto souboru je ilustrován v útržku 1.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

<outputs>
    <output filename="0.bmp" type="IMAGE" />
    <output filename="1.jpg" type="IMAGE" />
    <output filename="2.jpg" type="IMAGE" />
    <output filename="3.png" type="IMAGE" />
    <output filename="4.jpg" type="IMAGE" />
    <output filename="5.jpg" type="IMAGE" />
    <output filename="6.mp3" type="AUDIO" />
    <output filename="7.mp3" type="AUDIO" />
</outputs>
```

Útržek 1: XML formát dat pro definování výstupů

4.5 Vykreslování pomocí SDL

Vykreslování je tedy prováděno samostatným programem. Program můžeme nastavit pomocí přepínačů a zvolit mezi hardwarovým nebo softwarovým vykreslováním, změnit rozlišení anebo zda má program běžet v okně či plné obrazovce. Program při svém spuštění načte obrázky a zvuky do SDL struktur podle dodaného XML souboru. Během načítání obrázků jim je změněna velikost na cílovou velikost obrazovky/okna tak, aby byl zachován poměr obrazu.

Funkčnost různých obrazových a zvukových formátů je závislá na podpoře ze strany SDL. Interně SDL podporuje BMP a WAV. Další formáty vyžadují externí knihovny jako libz, libpng nebo libvorbis. Tyto knihovny si načítá SDL_image a SDL_mixer podle zadaných příznaků při iniciaci knihovny. Pokud však na Raspbianu spustíme SDL_image s příznaky *IMG_INIT_JPG* | *IMG_INIT_PNG* a SDL_mixer s příznaky *MIX_INIT_OGG* | *MIX_INIT_MP3* | *MIX_INIT_FLAC* zjistíme, že vše funguje jak má a tudíž můžeme konstatovat, že všechny potřebné knihovny jsou již součástí distribuce Raspbianu. Mohli bychom příznaky rozšířit i o další formáty, potom by ale program spotřebovával více paměti a nám stačí pokrýt pouze ty běžné.

Dále program běží ve smyčce, kde čte změnu vstupů z klávesnice a změny na GPIO pinech. Pro jednodušší testování funkčnosti jsem ponechal čtení z klávesnice, kdy můžeme jednotlivé výstupy vyvolat numerickými klávesami 0 až 7.

Existují dva doporučené postupy, jak zobrazovat obrázky na obrazovku. Starší postup spočívá ve vytvoření SDL_Surface z plochy okna a připravení obrazu do jiného SDL_Surface. K zobrazení připraveného obrazu použijeme metodu *SDL_BlitSurface,* která vyjme část z SDL_Surface a vloží jí do jiného SDL_Surface. Tato varianta mi fungovala pouze bez příznaku SDL_WINDOW_OPENGL. V případě zapnuté akcelerace OpenGL jsem se dočkal černé obrazovky.

Druhý postup doporučovaný hlavně v kombinaci s OpenGL spočívá ve vytvoření SDL_Renderer a jeho funkci *SDL_RenderCopy* s parametrem SDL_Texture. Obrázky jsou načítány do těchto textur. U tohoto postupu jsem se dočkal velmi pomalého vykreslování velkých textur. Problém vězí v použitém Raspberry Pi 3, které má mizerný výkon při použití s OpenGL. Proto jsem ponechal obě varianty a nechal výběr na uživateli. Věřím, že časem může být akcelerovaná grafika na Raspberry Pi 3 opravena, nebo může dojít k použití jiného miniaturního počítače.

4.6 Čtení přerušení z GPIO pomocí wiringPi

Čtení stimulů je realizováno knihovnou wiringPi. Ukázka, jak se v aplikaci nastavuje přerušení touto knihovnou, je v útržku 2. Z ukázky je zřejmé, že se syntaxem autor knihovny inspiroval u arduinovských základních knihoven.

Použití knihovny je zakomentováno pomocí *#ifdef* __arm__. Díky tomu je možné kód přeložit i na PC, kde wiringPi není dostupná. *PIN_0* je pin signalizující začátek a konec stimulu, ostatní určují adresu 0-7. Po přečtení pinů se sestaví SDL_Event a je do něj přidané číslo adresy. Funkce SDL_PushEvent přidá tuto událost do fronty událostí, abychom ji mohli zpracovat ve smyčce čekající na vstup. SDL_PushEvent je podle dokumentace thread-safe[13] a je ji možné volat z ostatních vláken, což je důležitá vlastnost při použití s wiringPi. Funkce wiringPiISR totiž vytváří nové vlákno[10], ve kterém polluje příslušný soubor v */sys* a v případě příchozího znaku spouští svůj třetí parametr, v tomto případě lambda funkci, ve které se SDL_PushEvent vyskytuje.

```
#include <SDL2/SDL_events.h>
#ifdef __arm__
#include <wiringPi.h>
#endif

bool Events::start() {
    if (running) return false;
    if (gpio_event_type ==((Uint32)-1)) gpio_event_type =
SDL_RegisterEvents(1);
    this->setup_interrupt();
    return true;
}

void Events::setup_interrupt() {
#ifdef __arm__
    wiringPiSetup();
    pinMode (PIN_0, INPUT);
    pinMode (PIN_1, INPUT);
    pinMode (PIN_2, INPUT);
    pinMode (PIN_3, INPUT);
    wiringPiISR(PIN_0,INT_EDGE_BOTH,[] {
        int pin_0_state = digitalRead(PIN_0);
        int pin_1_state = digitalRead(PIN_1);
        int pin_2_state = digitalRead(PIN_2);
        int pin_3_state = digitalRead(PIN_3);
        SDL_Event event;
        SDL_zero(event);
        event.type = gpio_event_type;
        if (pin_0_state == HIGH) {
            event.user.code =
            EVENT_CODES::GPIO_EVENT_0 +
            pin_1_state * 1 +
            pin_2_state * 2 +
            pin_3_state * 4 ;
        } else {
            event.user.code = EVENT_CODES::GPIO_EVENT_END;
        }
        // SDL_PushEvent je bezpecna, muze byt volana z jinych vlaken
        SDL_PushEvent(&event);
    });
#endif
}
```

Útržek 2: Registrace přerušení knihovnou wiringPi, útržek z třídy Events

5 Měření

Při vývoji stimulátoru ZČU byl kladen důraz na co nejmenší zpoždění stimulů. V případě našeho rozšíření vykreslování grafiky na počítačový monitor se však určitému opoždění nevyhneme. Naštěstí je stimulátor schopný dávat impulzy s předstihem, pokud ho k tomu nastavíme. V této kapitole se pokusím identifikovat zdroje opoždění měřením na osciloskopu a najít vhodnou hodnotu pro nastavení předstihu stimulátoru.

5.1 Technika měření

Měření jsem prováděl osciloskopem RIGOL DS1054Z. Tento osciloskop umožňuje snímat až 1Gsa/s(gigasampl za vteřinu). Měřený monitor byl EIZO FORIS FS2434, ten má podle výrobce velmi malé vstupní opoždění 0,05ms. Toto tvrzení potvrdil nezávislý test[14]. Odezva panelu (tedy přebarvení pixelu z černé na bílou a zpět) je v případě tohoto monitoru 4,9ms. Typ testovaného Raspberry byl Raspberry Pi 3 Model B. Jiné modely Raspberry se mohou rychlostí vykreslování značně lišit.

5.2 Měření odezvy knihovny wiringPi

Když jsem zjistil, že knihovna wiringPi používá ke čtení přerušení z GPIO souborový systém, byl jsem trochu skeptický, zda takové řešení nebude působit prodlevy při čtení. Proto jsem se rozhodl podrobit wiringPi měření odezvy. Při měření jsem použil Arduino, na kterém běžel program **Blink**, a Raspberry Pi s běžícím programem, používajícím knihovnu wiringPi.

Program Blink periodicky bliká jedním pinem, připojeným k interní LED a je jedním z ukázkových příkladů dodávaných k Arduinovskému IDE. Doba zapnutí a vypnutí pinu je 1 vteřina a neměla by nijak ovlivnit měření. Tento blikající pin jsem použil jako vstup a připojil ho k Raspberry Pi.

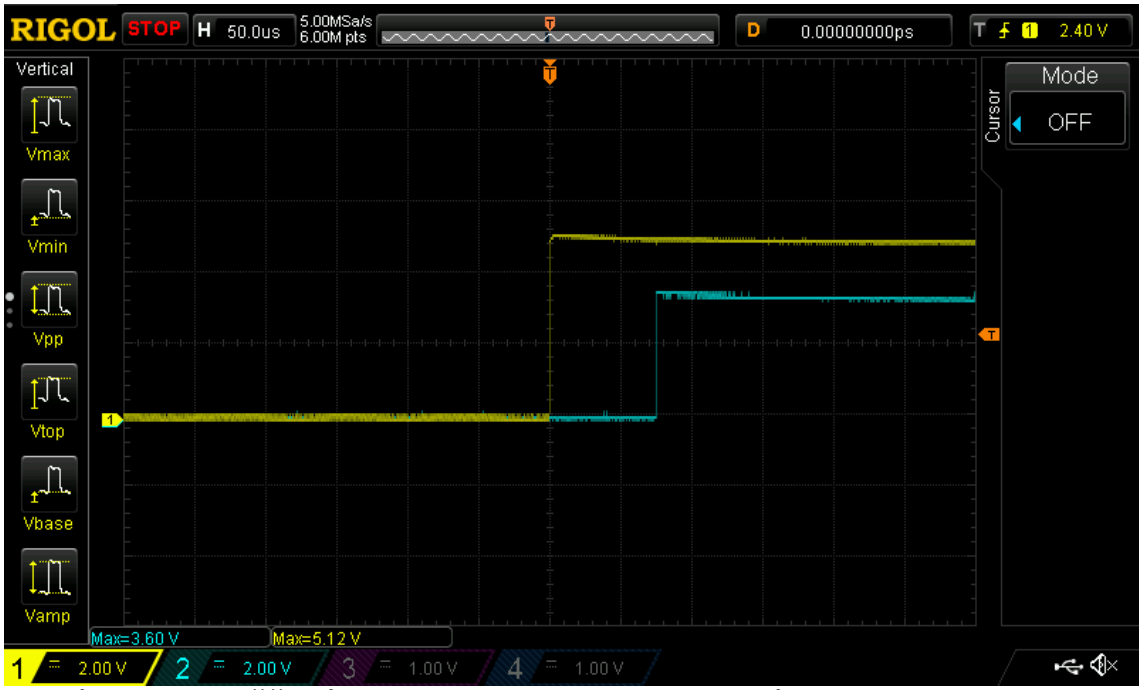
Na Raspberry Pi jsem si připravil jednoduchý prográmek. Zdrojový kód je vidět v útržku 3. Při spuštění se nastaví *PIN_0* jako vstup a *PIN_1* jako výstup. Přerušení je znovu obslouženo lambda funkcí. Při změně stavu pinu 0 se přečte její hodnota a tato hodnota se zapíše do pinu 1. Ukončení programu brání čtení ze standardního vstupu.

K pinům 0 a 1 jsem připojil kanály 1 a 2 osciloskopu. Na kanálu 1 jsem nastavil trigger při vzestupné i sestupné hraně, takže při měření tyto hrany představují čas 0. Podle pracovních voltáží na obrázku 14 je patrné, že zdrojem signálu na kanálu 1 (žlutý) je Arduino s 5V a na kanálu 2 (modrý) Raspberry, které operuje na 3,3V. Opoždění hrany na kanálu 2 jsem změřil pomocí kurzorových čar na obrázku 15. Podle těchto čar je opoždění signálu přibližně 75µs. Stejná situace nastává při sestupné hraně (obrázek 16), tady je čas 75,6µs, tedy hodnota podobná. Při opakovaném měření byla hodnota opoždění vždy v rozmezí 75-76µs.

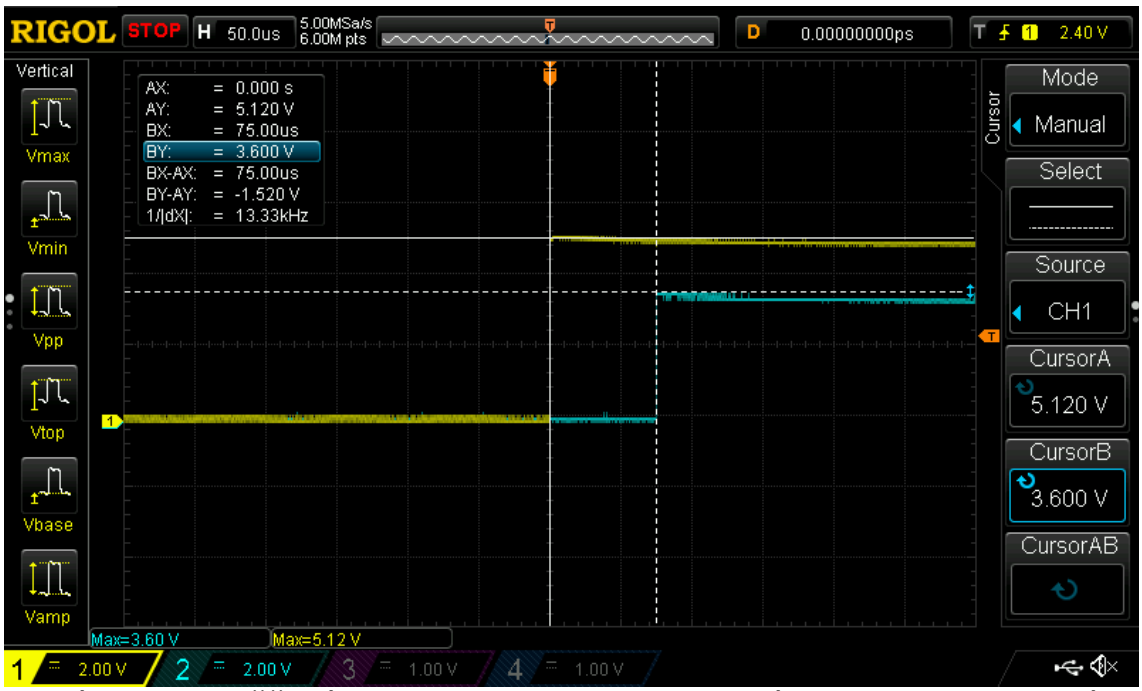
Bohužel toto měření nám neumožní určit, kolik času zabere čtení pinu a kolik zápis. Rychlost celkové reakce a tedy i rychlost a konstantní délka zápisu nám přináší zajímavé možnosti. Například, kdybychom použili Raspberry Pi jako stimulátor, mohli bychom použít GPIO pro zápis markerů pro EEG záznamové zařízení s jistotou, že markery budou mít správné umístění.

```
#define PIN_0 0
#define PIN_1 1

#include <iostream>
#include <wiringPi.h>

int main(int argc, char *argv[]) {
      wiringPiSetup();
      pinMode (PIN_0, INPUT);
      pinMode (PIN_1, OUTPUT);
      wiringPiISR(PIN_0,INT_EDGE_BOTH,[] {
            int pin_state = digitalRead(PIN_0);
            digitalWrite(PIN_1,pin_state);
            std::cout << "state changed to " << pin_state <<
std::endl;
      });
      std::cout << "Press ENTER to end" << std::endl;
      std::cin.get();
}
```

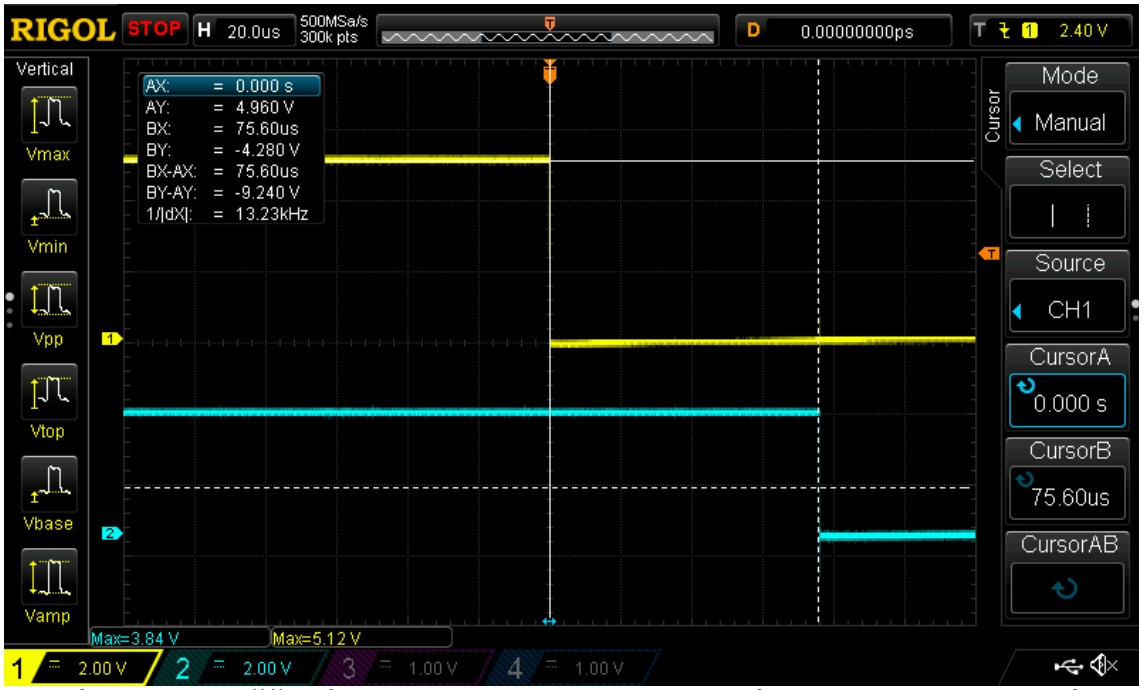
Útržek 3: Program na otestování odezvy knihovny wiringPi



Obrázek 14: Měření odezvy wiringPi, vzestupná hrana, bez kurzorových čar



Obrázek 15: Měření odezvy wiringPi, vzestupná hrana, s kurzorovými čárami



Obrázek 16: Měření odezvy wiringPi, sestupná hrana, s kurzorovými čárami

5.3 Měření zpoždění vykreslování na monitor

Pro měření odezvy vykreslování jsem si připravil na Arduinu jednoduchou náhradu za stimulátor. Tento program nastavuje na adresní piny náhodný stimul 0-7 a pak přepne pin pro zahájení stimulu. Po náhodně dlouhém časovém úseku pak sníží napětí na všech pinech dolů. Periodicky tak simuluje činnost stimulátoru. Časy stimulů a pauz mezi nimi jsou náhodné.

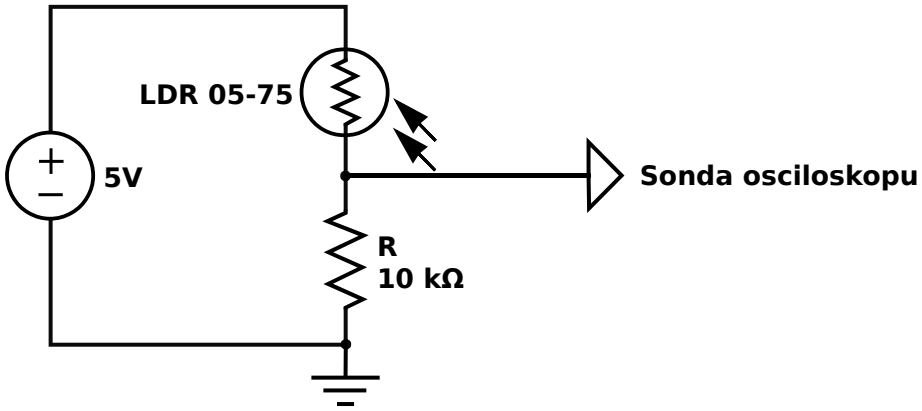
Tento pseudostimulátor jsem připojil k Raspberry Pi a spustil aplikaci s připravenými grafickými výstupy. Pro tuto příležitost jsem upravil program na vykreslování grafiky tak, aby do každého obrazu přidal bílý čtvereček v pravém dolním rohu obrazovky.

Pro měření času zobrazení jsem se rozhodl použít fotorezistor LDR 05-75 přilepený k monitoru v místě, kde se vykresluje bílý čtverec. Tento fotorezistor snižuje svůj odpor se stoupajícím osvětlením, ve tmě má odpor 5 MΩ. V případě vystavení bílému světlu z LCD monitoru se mu snížil odpor až na 10 kΩ. Zapojil jsem ho do série s 10 kΩ odporem, podle zapojení na obrázku 17. Tyto odpory tak spolu tvoří napěťový dělič a umožňují snadné měření osciloskopem.

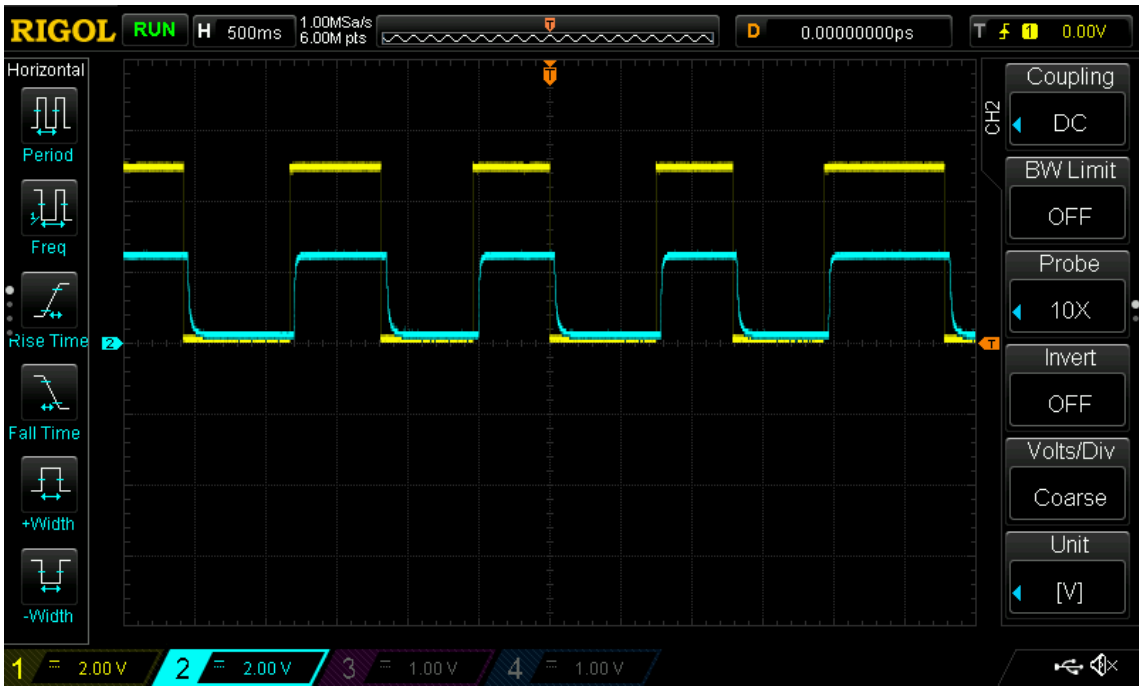
Průběh napětí je vidět na obrázku 18. Ke kanálu 1 osciloskopu je připojen pin zahajující/ukončující stimul, druhým kanálem je připojen fotorezistor. Na průběhu je možné vidět jasné opoždění za vstupním signálem. Signál z fotorezistoru nemá tak jasné strmé hrany, což bude pravděpodobně způsobeno dobou odezvy než pixely změní barvu z černé na bílou a naopak. Pro předejití nejasnostem, jsem během měření za směrodatnou považoval událost, kdy začne signál strmě stoupat, klesat. Využil jsem možnost osciloskopu měřit opoždění mezi dvěma kanály a naměřil jsem 100 hodnot pro vzestup a sestup. Grafy rozložení těchto opoždění jsou vidět na obrázcích 19 a 20. Monitor zobrazuje 60 snímků za sekundu, tím pádem připadá 16,67 ms na jeden snímek. V ideální světě by tedy vzdálenost dvou mezních měření nepřesáhla tuto hodnotu. Naše měření tuto hodnotu přesáhlo, ale není to nijak tragické. Všechny měřené hodnoty se vejdou do intervalu dvou snímků. Může nás těšit, že po dobré kalibraci stimulátoru se může vykreslování opožďovat maximálně o jeden snímek monitoru. Průměrné opoždění během měření bylo 34,26 ms při vzestupné hraně a 34,51 ms při sestupné.

Mezní hodnoty měření jsou od sebe vzdálené 27 ms. Markery vytvářené do EEG záznamu pomocí stimulátoru nemohou úplně vystihnout dění na LCD monitoru. I v případě rozsahu jednoho snímku by tato proměnlivá opoždění působila posunutí ERP průběhu. Výsledný průměr by pak působením těchto dílčích opoždění ztratil malé výchylky a byl celkové hladší. Proto by bylo vhodné nějakým způsobem zahrnout do řešení synchronizační marker vytvářený přímo monitorem. Například malinký čtvereček v rohu a permanentní senzor by mohli snímat aktivitu displeje. Aplikace by byla přizpůsobena na vytváření těchto malých čtverců při každém obrazu. Tímto senzorem bychom získali dostatečně přesné markery. Tato oblast by byla odstíněna černým materiálem, aby subjekt nerušila.

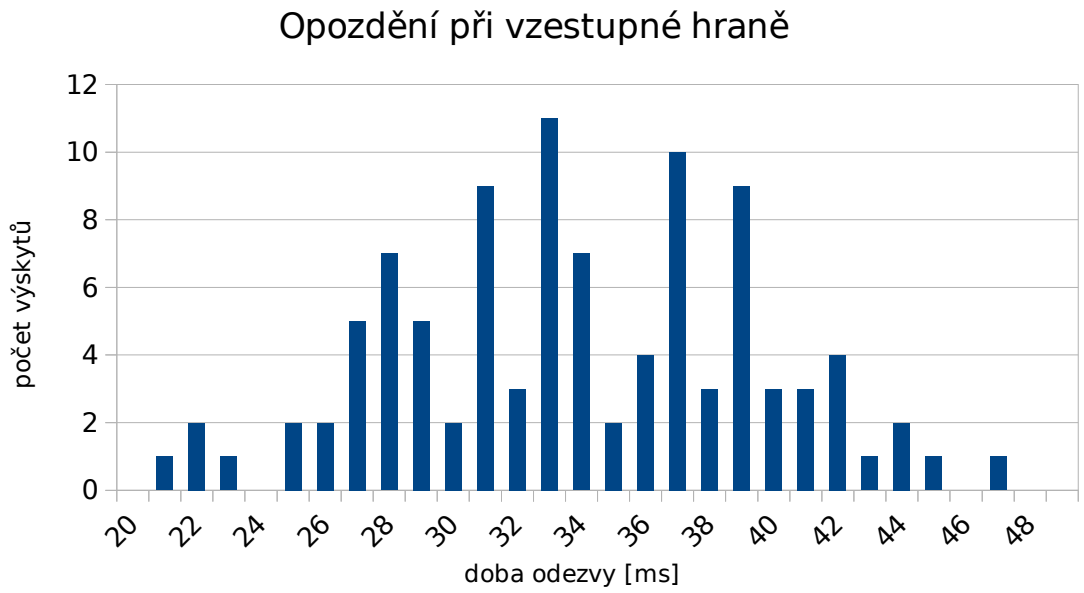
V případě použití hardwarové akcelerace bylo opoždění naprosto fatální a někdy se ani nestihl obraz vykreslit během časového okna stimulu. Proto není třeba měřit dobu. Implementaci hardwarového vykreslování v aplikaci ponechávám pro případy, že by se situace v budoucnu změnila při použití jiného miniaturního počítače či opravy Raspberry Pi ze strany výrobce.



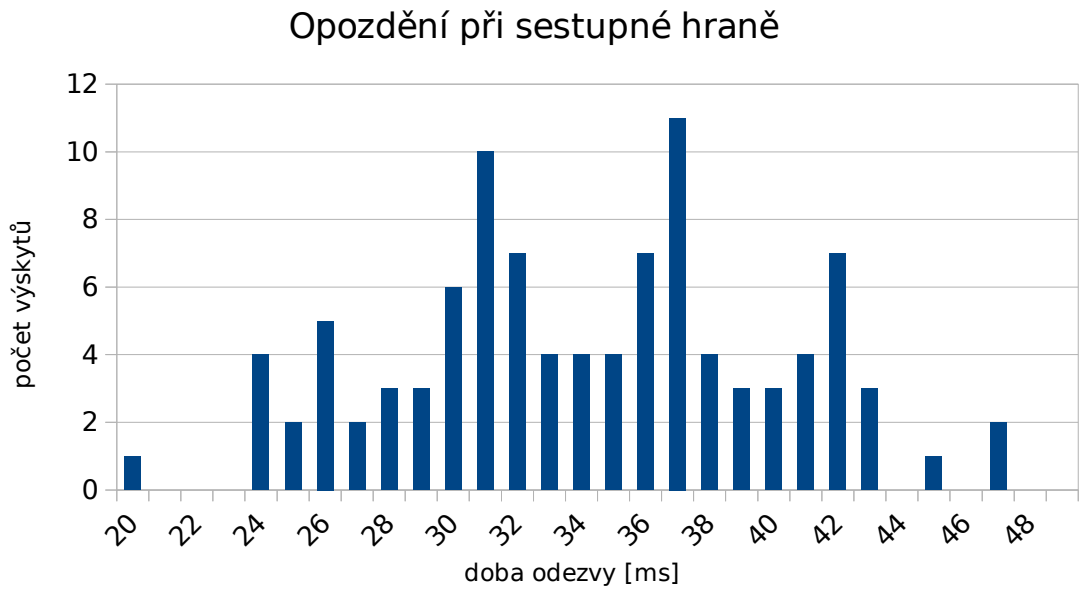
Obrázek 17: Připojení fotorezistoru k osciloskopu



Obrázek 18: Průběh během měření odezvy vykreslování



Obrázek 19: Graf opoždění vykreslování, vzestupná hrana



Obrázek 20: Graf opoždění vykreslování, sestupná hrana

6 Závěr

Na základě případů užití jsem vytvořil dvě aplikace, které spolu splňují požadované scénáře. Oddělením aplikace pro vykreslování stimulů jsem se vyhnul zbytečně komplikované aplikaci.

Co se týče ovládání stimulátoru, byla naimplementována část, kterou umožňuje současný stav stimulátoru a jeho komunikačního protokolu. V případě doimplementování dalších částí protokolu je jednoduché dodat aplikaci funkcionalitu, protože uživatelské rozhraní dalších experimentů je již součástí aplikace.

Při měření odezvy knihovny wiringPi jsem naměřil velmi pohotovou reakci knihovny. Hlavním akterém opoždění Raspberry Pi je tedy samotné zobrazování grafiky. Všechna měření byla v rozsahu 27 ms, tedy téměř dvou snímků obrazovky. Kdybychom však chtěli použít metodu průměrování podle vstupního markeru ze stimulátoru, určitě bychom přišli o část dat díky rozmazání průběhu. Proto by bylo vhodné do řešení přidat markery přímo z monitoru.

Zdroje

[1] Elektroencefalografie - WikiSkripta. [Online]. [cit. 2018-06-21] Dostupné z: https://www.wikiskripta.eu/w/Elektroencefalografie

[2] Frekvence, které jsou důležité pro vaše zdraví a rozšíření vědomí - AC24.cz. [Online]. [cit. 2018-06-26] Dostupné z: https://ac24.cz/-/zpravy-ze-sveta/3384-mozkove-vlny-a-rozsirene-vedomi

[3] Evokované potenciály – WikiSkripta. [Online]. [cit. 2018-06-26] Dostupné z: https://www.wikiskripta.eu/w/Evokovan%C3%A9_potenci%C3%A1ly

[4] Steven J. Luck, An Introduction to the Event-Related Potential Technique. A Bradford Book, 2014. ISBN 9780262525855.

[5] Guangyu Bin, Xiaorong Gao, Yijun Wang, Bo Hong a Shangkai Gao, "VEP-based brain-computer interfaces: time, frequency, and code modulations" vyšlo v IEEE Computational Intelligence Magazine, sv. 4, č. 4, s 22-26. IEEE, Listopad 2009.

[6] Presentation Features. [Online]. [cit. 2018-06-21] Dostupné z: https://www.neurobs.com/menu_presentation/menu_features/features_list

[7] Recording ERPs and other stimulus-driven data | OpenViBE. [Online]. [cit. 2018-06-21] Dostupné z: http://openvibe.inria.fr/recording-erps/

[8] Parallel port (EEG triggers) // OpenSesame documentation. [Online]. [cit. 2018-06-21] Dostupné z: http://osdoc.cogsci.nl/3.2/manual/devices/parallel/

[9] Using the Parallel Port - PEBL WIKI. [Online]. [cit. 2018-06-21] Dostupné z: http://pebl.sourceforge.net/wiki/index.php/Using_the_Parallel_Port

[10] Zdrojový kód WiringPi/wiringPi.c. [Online]. [cit. 2018-06-21] Dostupné z: https://github.com/WiringPi/WiringPi/blob/master/wiringPi/wiringPi.c

[11] Martin Chroboczek, Grafická uživatelská rozhraní v Qt a C++. Praha: Computer Press, 2013. ISBN 9788025141243.

[12] Game Loops. Lazy Foo' Productions.. [Online]. [cit. 2018-06-21] Dostupné z: http://lazyfoo.net/articles/article04/index.php

[13] SDL_PushEvent - SDL Wiki'. [Online]. [cit. 2018-06-21] Dostupné z: https://wiki.libsdl.org/SDL_PushEvent

[14] Eizo FS2434 review - FlatpanelsHD. [Online]. [cit. 2018-06-21] Dostupné z: https://www.flatpanelshd.com/review.php?subaction=showfull&id=1413168564

Přílohy

A Protokol stimulátoru

Programovatelný EEG stimulátor pro neuroinformatiku
Definice paketu pro USB ovládání

Jednotlivé pakety mají pevnou délku 64B s následující strukturou:
1B ID + LEN
1B MSG TYPE
62B DATA

1. byte - Hlavička
ID 2bity
Rezerva pro další implementaci, v této verzi ponechávat 00
LEN 6bitů
0-62 délka dat v bytech, 1 paket
63 fragment paketu, poslední část fragmentu má délku <=62B

2. byte - Typ zprávy
0x00 - 0x0F Základní ovládání - nemá žádné parametry (data)
0x00 - Obnovení displeje (REFRESH)
překreslí aktuální hodnoty proměnných na display
0x01 - Povolení LED - povolí blikání LED
0x02 - Zakázání LED - zakáže blikání LED
0x03 - Stav LED - vrátí stav LED
0x04 - Povolení náhodnosti - povolí náhodnost mezi intervaly (doba svícení a nesvícení je náhodně protažena nebo zkrácena)
0x05 Zakázání náhodnosti
0x06 - Stav náhodnosti - vrátí stav náhodnosti
0x07 - 0x0E - Rezerva
0x0F - Čtení USB bufferu

0x10 - 0x5F Nastavení stimulátoru
0x10 - Doba svícení LED0 - data 2B, hodnota je v 0.1ms
0x11 - Pauza LED0 - data 2B, hodnota je v 0.1ms
0x12 - Doba svícení LED1 - data 2B, hodnota je v 0.1ms
0x13 - Pauza LED1 - data 2B, hodnota je v 0.1ms
0x14 - Doba svícení LED2 - data 2B, hodnota je v 0.1ms
0x16 - Pauza LED2 - data 2B, hodnota je v 0.1ms
0x16 - Doba svícení LED3 - data 2B, hodnota je v 0.1ms
0x17 - Pauza LED3 - data 2B, hodnota je v 0.1ms
0x18 - Zastoupení LED0 - data 1B, hodnota je v %
min 0 max 100, součet přes LEDx = 100
0x19 - Zastoupení LED1 - data 1B, hodnota je v %
min 0 max 100, součet přes LEDx = 100
0x1A - Zastoupení LED2 - data 1B, hodnota je v %
min 0 max 100, součet přes LEDx = 100
0x1B - Zastoupení LED3 - data 1B, hodnota je v %
min 0 max 100, součet přes LEDx = 100
0x1C - Jas LED0 - data 1B, hodnota je v %, min 0 (nesvítí) max 100
0x1D - Jas LED1 - data 1B, hodnota je v %, min 0 (nesvítí) max 100
0x1E - Jas LED2 - data 1B, hodnota je v %, min 0 (nesvítí) max 100
0x1F - Jas LED3 - data 1B, hodnota je v %, min 0 (nesvítí) max 100
0x20 - Doba sync pulsu - data 2B, hodnota je v ms
max zobrazení na displeji 9999

0x21 - Hrana synchronizačního pulsu - 0 náběžná, 1 sestupná
0x22 - Nastavení frekvence LED0 - data 1B
frekvence periody svítí-nesvítí v Hz
0x23 - Nastavení středu periody LED0 - data 1B
určuje podíl svití-nesvítí (výchozí 50%)
0x24 - Nastavení frekvence LED1 - data 1B
frekvence periody svítí-nesvítí v Hz
0x25 - Nastavení středu periody LED1 - data 1B
určuje podíl svití-nesvítí (výchozí 50%)
0x26 - Nastavení frekvence LED2 - data 1B
frekvence periody svítí-nesvítí v Hz
0x27 - Nastavení středu periody LED2 - data 1B
určuje podíl svití-nesvítí (výchozí 50%)
0x28 - Nastavení frekvence LED3 - data 1B
frekvence periody svítí-nesvítí v Hz
0x29 - Nastavení středu periody LED3 - data 1B
určuje podíl svití-nesvítí (výchozí 50%)
0x2A - Přehrání zvuku z SD karty - data 8 bytů
název zvukového souboru bez přípony
0x2B - Nastaví všechny položky aktuálního schématu
data **XX B** v definovaném formátu
0x2C - Doba svícení LED4 - data 2B, hodnota je v 0.1ms
0x2D - Pauza LED4- data 2B, hodnota je v 0.1ms
0x2E - Doba svícení LED5 - data 2B, hodnota je v 0.1ms
0x2F - Pauza LED5 - data 2B, hodnota je v 0.1ms
0x30 - Doba svícení LED6 - data 2B, hodnota je v 0.1ms
0x31 - Pauza LED6 - data 2B, hodnota je v 0.1ms
0x32 - Doba svícení LED7 - data 2B, hodnota je v 0.1ms
0x33 - Pauza LED7 - data 2B, hodnota je v 0.1ms
0x34 - Zastoupení LED4 - data 1B, hodnota je v %
min 0 max 100, součet přes LEDx = 100
0x35 - Zastoupení LED5 - data 1B, hodnota je v %
min 0 max 100, součet přes LEDx = 100
0x36 - Zastoupení LED6 - data 1B, hodnota je v %
min 0 max 100, součet přes LEDx = 100
0x37 - Zastoupení LED7 - data 1B, hodnota je v %
min 0 max 100, součet přes LEDx = 100
0x38 - Jas LED56 - data 1B, hodnota je v %
min 0 (nesvítí) max 100
0x39 - Jas LED78 - data 1B, hodnota je v %
min 0 (nesvítí) max 100
0x3A - Nastavení frekvence LED4 - data 1B, frekvence periody
svítí-nesvítí v Hz
0x3B - Nastavení středu periody LED4 - data 1B, určuje podíl
svití-nesvítí (výchozí 50%)
0x3C - Nastavení frekvence LED5 - data 1B, frekvence periody
svítí-nesvítí v Hz
0x3D - Nastavení středu periody LED5 - data 1B, určuje podíl
svití-nesvítí (výchozí 50%)
0x3E - Nastavení frekvence LED6 - data 1B, frekvence periody
svítí-nesvítí v Hz
0x3F - Nastavení středu periody LED6 - data 1B, určuje podíl
svití-nesvítí (výchozí 50%)

0x40 - Nastavení frekvence LED7 - data 1B, frekvence periody svítí-nesvítí v Hz
0x41 - Nastavení středu periody LED7 - data 1B, určuje podíl svití-nesvítí (výchozí 50%)
0x42 - 0x5F - Rezerva

0x60 - 0xBF Kontrola stimulátoru
0x60 - 0x89 - Vrací nastavenou hodnotu viz Nastavení stimulátoru
0x8A - Vrací seznam zvuků (*.WAV) na SD kartě
0x8B - Vrátí všechny položky aktuálního schématu - data 33B v definovaném formátu
0x8C - 0xA1 - Vrací nastavenou hodnotu viz Nastavení stimulátoru
0xA2 - 0xBF - Rezerva

0xC0 - 0xCF
Práce s pamětí - parametrem je adresa (pointer na stránku v paměti) nebo data v předepsaném formátu
0xC0 - Uložení současného nastavení do paměti - uloží data na zadanou stránku
0xC1 - Nahrání nastavení z paměti - nahraje data ze zadané stránky do zařízení
0xC2 - Uložení dat do paměti - zapíše do paměti celé schéma. 1B dat udává stránku, zbytek dat je samotný obsah v předem definovaném formátu
0xC3 - Čtení dat z paměti - přečte z paměti celé schéma dané stránkou a odešle přes USB.
0xC4 - 0xCE - Rezerva
0xCF - Čtení celé paměti
přečte obsah celé paměti a odešle přes USB

0xD0 - 0xEF - Rezerva

0xF0 - 0xFF DEBUG
Zprávy pro přímý přístup na systémové sběrnice UART, SPI, IIC. Určeno pouze pro účely ladění stimulátoru a bude odstraněno ve verzi RELEASE.

3. a další byty

Data, typ je definován typem zprávy uvedeným výše. Platná data se rozlišují polem LEN (délka paketu), nevyužité byty jsou automaticky doplněny nulami. Rezervované kódy mohou být použity v dalších verzích jednotky stimulátoru.

B Datový protokol

Datový protokol slouží k přenosu souborů (obrázků, zvuků a konfigurací) mezi Raspberry a mobilním zařízením.

Tento dokument popisuje struktury datového protokolu. Každý paket má na začátku dva bajty.

`[FULL_LENGTH_MESSAGE][COMMUNICATION_OP_CODE]`

Dále jsou popsány pakety o délkách 62.

Typy paketů

- *TYPE_REQUEST* odesílán z klienta, zahájí operaci.
- *TYPE_RESPONSE* odpověď od serveru na REQUEST.
- *TYPE_DOWNLOAD* data odesílaná ze serveru na klienta.
- *TYPE_UPLOAD* data odesílaná z klienta na server.

UPLOAD, DOWNLOAD jsou doprovodná data v případě jistých operací, které vyžadují větší přenos. Jednotlivé zprávy se od sebe rozeznají zvýšením iterátoru ITER. Iterátor si určuje klient a podle nich pozná odpověď na příslušnou zprávu. Díky tomu je možné obsluhovat současně více zpráv s různými iterátory.

Rozdělení paketů

Pokud je zapotřebí data rozdělit do více paketů, děje se to následujícím způsobem. Dlouhá zpráva je rozdělena do více paketů a ty mají příznaky PART_CONTINUE a poslední paket má příznak PART_LAST.

LONG REQUEST

```
[OP+TYPE_REQUEST+PART_CONTINUE][ITER]{ARGS1}
[OP+TYPE_REQUEST+PART_CONTINUE][ITER]{ARGS2}
[OP+TYPE_REQUEST+PART_LAST][ITER]{ARGS3}
```

LONG RESPONSE

```
[OP+TYPE_RESPONSE+PART_CONTINUE][ITER]{ARGS1}
[OP+TYPE_RESPONSE+PART_CONTINUE][ITER]{ARGS2}
[OP+TYPE_RESPONSE+PART_LAST][ITER]{ARGS3}
```

LONG DOWNLOAD

```
[TYPE_DOWNLOAD+PART_CONTINUE][ITER]{DATA1}
[TYPE_DOWNLOAD+PART_CONTINUE][ITER]{DATA2}
```

[TYPE_DOWNLOAD+PART_LAST][ITER]{DATA3}

LONG UPLOAD

[TYPE_UPLOAD+PART_CONTINUE][ITER]{DATA1}
[TYPE_UPLOAD+PART_CONTINUE][ITER]{DATA2}
[TYPE_UPLOAD+PART_LAST][ITER]{DATA3}

Popis operací

Řetězce jsou ukončené nulovým bajtem. V případě použití zprávy s DOWNLOAD nebo UPLOAD přenosem je tento přenos zabezpečen MD5 hashem, který je součástí REQUEST nebo RESPONSE.

HELLO (přihlášení k serveru)

[OP_HELLO+TYPE_REQUEST][ITER][HELLO_VER]{NAME}[0]

HELLO_RESPONSE (odpověď na přihlášení k serveru)

[OP_HELLO+TYPE_RESPONSE][ITER][RESPONSE_OK]{NAME}[0]

BYE (odpojení ze serveru)

[OP_BYE+TYPE_REQUEST][ITER]

BYE_RESPONSE (odpověď na odpojení ze serveru)

[OP_BYE+TYPE_RESPONSE][ITER][RESPONSE_OK]

MD (požadavek na vytvoření adresáře)

[OP_MD+TYPE_REQUEST][ITER]{DIR_PATH}[0]

MD_RESPONSE (odpověď na vytvoření adresáře)

[OP_MD+TYPE_RESPONSE][ITER][RESPONSE_OK]

or

[OP_MD+TYPE_RESPONSE][ITER][RESPONSE_MD_DIR_EXIST]

or

[OP_MD+TYPE_RESPONSE][ITER][RESPONSE_MD_FAIL]

LS (výpis obsahu adresáře)

[OP_LS+TYPE_REQUEST][ITER][FLAGS]{DIR_PATH}[0]{MASK}[0]

LS_RESPONSE (odpověď na výpis adresáře)

[OP_LS+TYPE_RESPONSE][ITER][RESPONSE_OK][4:DATA_SIZE]
[16:DATA_MD5]

```
[TYPE_DOWNLOAD][ITER][2:COUNT][4:FILE1_SIZE][16:FILE1_MD5]
{FILE1_NAME}[0][4:FILE2_SIZE][16:FILE2_MD5]{FILE2_NAME}
[0]...
```

or

```
[OP_LS+TYPE_RESPONSE][ITER][RESPONSE_LS_DIR_NOT_FOUND]
```

GET (požadavek na stažení souboru)

```
[OP_GET+TYPE_REQUEST][ITER]{FILE_PATH}[0]
```

GET_RESPONSE (odpověď se staženým souborem)

```
[OP_GET+TYPE_RESPONSE][ITER][RESPONSE_OK][4:DATA_SIZE]
[16:DATA_MD5]
[TYPE_DOWNLOAD][ITER]{FILE_BYTES}
```

or

```
[OP_GET+TYPE_RESPONSE][ITER][RESPONSE_GET_FILE_NOT_FOUND]
```

GET_PREVIEW (požadavek na náhled souboru)

```
[OP_GET_PREVIEW+TYPE_REQUEST][ITER]{FILE_PATH}[0]
```

GET_PREVIEW_RESPONSE (odpověď s náhledem souboru)

```
[OP_GET_PREVIEW+TYPE_RESPONSE][ITER][RESPONSE_OK]
[4:DATA_SIZE][16:DATA_MD5]
[TYPE_DOWNLOAD][ITER]{PREVIEW_FILE_BYTES}
```

PUT (nahrání souboru na server)

```
[OP_PUT+TYPE_REQUEST][ITER][4:DATA_SIZE][16:DATA_MD5]
{FILE_PATH}[0]
[TYPE_UPLOAD][ITER]{FILE_BYTES}
```

PUT_RESPONSE (odpověď na nahrání souboru)

```
[OP_PUT+TYPE_RESPONSE][ITER][RESPONSE_OK]
```

or

```
[OP_PUT+TYPE_RESPONSE][ITER][RESPONSE_PUT_MD5_FAIL]
```

DEL (požadavek na smazání souboru)

```
[OP_DEL+TYPE_REQUEST][ITER]{FILE_PATH}[0]
```

DEL_RESPONSE (odpověď na smazání souboru)

```
[OP_DEL+TYPE_RESPONSE][ITER][RESPONSE_OK]
```

or

```
[OP_DEL+TYPE_RESPONSE][ITER][RESPONSE_DEL_FAIL]
```

START (požadavek na spuštění SDL zobrazování)

`[OP_START+TYPE_REQUEST][ITER]{CONFIG_PATH}[0]`

START_RESPONSE (odpověď na spuštění SDL zobrazování)

`[OP_START+TYPE_RESPONSE][ITER][RESPONSE_OK]`

STOP (vypnutí SDL zobrazování po skončení experimentu)

`[OP_STOP+TYPE_REQUEST][ITER]`

STOP_RESPONSE (odpověď na vypnutí SDL zobrazování)

`[OP_STOP+TYPE_RESPONSE][ITER][RESPONSE_OK]`

Konstanty

```
FULL_LENGTH_MESSAGE         0x3E
COMMUNICATION_OP_CODE       0xEF

TYPE_REQUEST                0x00
TYPE_RESPONSE               0x40
TYPE_DOWNLOAD               0x80
TYPE_UPLOAD                 0xC0

PART_CONTINUE               0x00
PART_LAST                   0x20

OP_HELLO                    0x01
OP_BYE                      0x02
OP_MD                       0x03
OP_LS                       0x04
OP_GET                      0x05
OP_PUT                      0x06
OP_DEL                      0x07
OP_START                    0x08
OP_STOP                     0x09
OP_GET_PREVIEW              0x10

RESPONSE_OK                 0x00
RESPONSE_MD_DIR_EXIST       0x01
RESPONSE_MD_FAIL            0x02
RESPONSE_LS_DIR_NOT_FOUND   0x03
RESPONSE_DEL_FAIL           0x04
RESPONSE_PUT_MD5_FAIL       0x05
RESPONSE_GET_FILE_NOT_FOUND 0x06
```

C Uživatelská dokumentace

Instalace

V tomto návodu předpokládám, že uživatel vychází z čistého Raspbianu ve variantě **with desktop**, Raspberry Pi je připojeno k internetu. Uživatel má otevřený příkazový řádek dle vlastní volby.

Nainstalujeme potřebné závislosti.

```
$ sudo apt-get install \
qtbase5-dev libqt5serialport5-dev cmake
```

Zdrojové kódy aplikace stáhneme z internetu programem git.

```
$ git clone \
https://github.com/qwerty2586/stimulator_control.git
```

Nyní máme stažené zdrojové kódy v nově vzniklém adresáři *stimulator_control*. Přepněme se tedy do toho adresáře.

```
$ cd stimulator_control
```

V tomto adresáři provedeme kompilaci následujícími příkazy.

```
$ cmake .
$ make -j4
```

Vznikne nám spustitelný soubor *stimulator_control*. Tento obsahuje grafické rozhraní pro ovládání stimulátoru. Pro obsluhu stimulů budeme potřebovat program SDL_output. Jeho instalace je podobná. Nejprve se vrátíme do nadřazeného adresáře.

```
$ cd ..
```

Nainstalujeme potřebné závislosti.

```
$ sudo apt-get install \
libsdl2-dev libsdl2-image-dev libsdl2-mixer-dev cmake
```

Stáhneme zdrojové kódy.

```
$ git clone https://github.com/qwerty2586/SDL_output.git
```

Přepneme se do vzniklého adresáře.

```
$ cd SDL_output
```

Zkompilujeme a výslednou binárku přidáme do adresáře ke stimulator_control. Aby mohl stimulator_control použít SDL_output, musí být ve stejném adresáři.

```
$ cmake .
```

```
$ make -j4
$ cp SDL_output ../stimulator_control
```

Nyní již můžeme spustit stimulator_control.

```
$ cd ../stimulator_control
$ ./stimulator_control
```

Připojení Raspberry Pi ke stimulátoru

Pro komunikaci se stimulátorem je zapotřebí připojit 2 piny z UART sběrnice. Pro obsluhu přerušení a výstup na monitor je třeba připojit další 4 piny. Z pinů stimulátoru je možné Raspberry Pi i napájet. Dovoluji si upozornit, že v případě použití klasických kabelů určených pro nepájivé pole, může dojít při napájení Raspberry Pi k podpětí. To je indikováno jako duhový čtverec (Raspberry Pi 2 a starší) nebo jako blesk na monitoru. Pro napájení skrz piny doporučuji použití jiných kabelů nebo aspoň zdvojit ty klasické.

Následující zapojení odpovídá patici Raspberry Pi 3. Starší modely mají patici kratší, ale zapojení je pro nás identické. Připojení sériového portu je označeno jako UART. Tento sériový port je potom přístupný ze systému jako */dev/ttyAMA0*. Připojení pinů stimulů je označeno jako GPIO. GPIO pin 0 je pin značící začátek a konec stimulu, ostatní představují adresu stimulu.

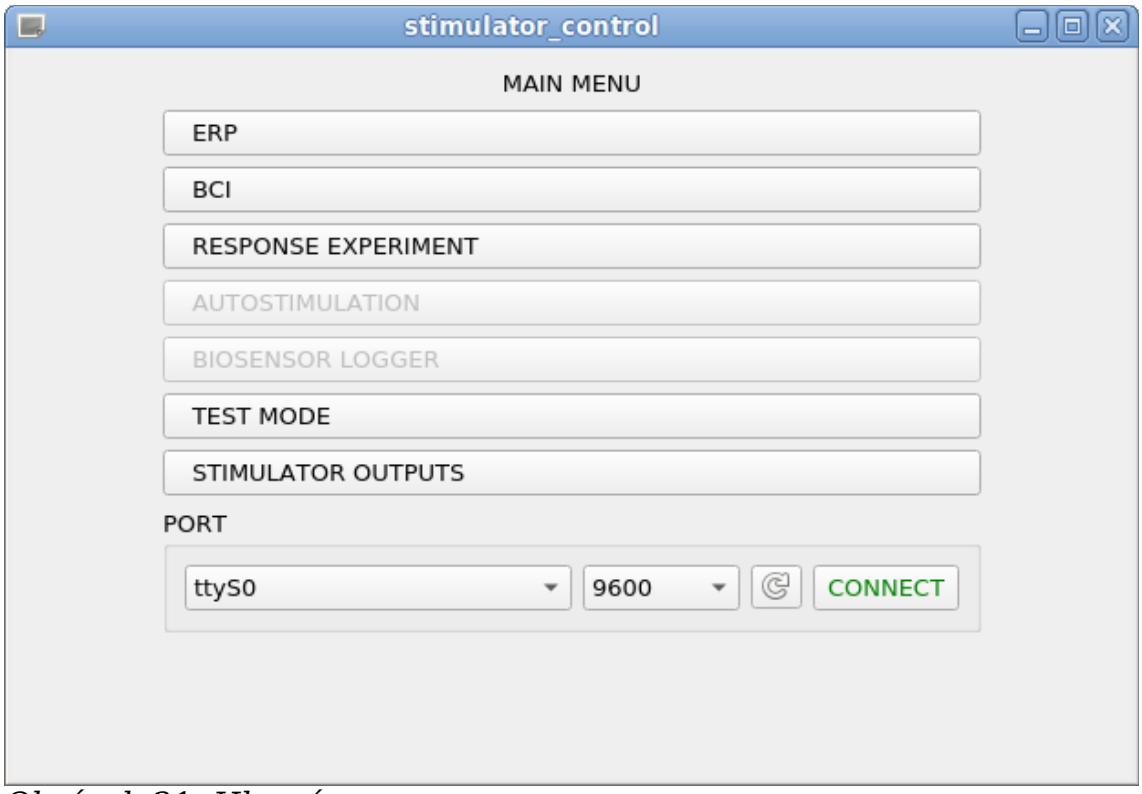
```
           Pin 1 Pin2
        +3V3 [ ] [ ] +5V
             [ ] [ ] +5V
             [ ] [ ] GND
             [ ] [ ] UART TXD0
         GND [ ] [ ] UART RXD0
GPIO PIN0    [ ] [ ] GPIO PIN1
GPIO PIN2    [ ] [ ] GND
GPIO PIN3    [ ] [ ]
        +3V3 [ ] [ ]
             [ ] [ ] GND
             [ ] [ ]
             [ ] [ ]
         GND [ ] [ ]
             [ ] [ ]
             [ ] [ ] GND
             [ ] [ ]
             [ ] [ ] GND
             [ ] [ ]
             [ ] [ ]
         GND [ ] [ ]
          Pin 39 Pin 40
```

Uživatelské rozhraní

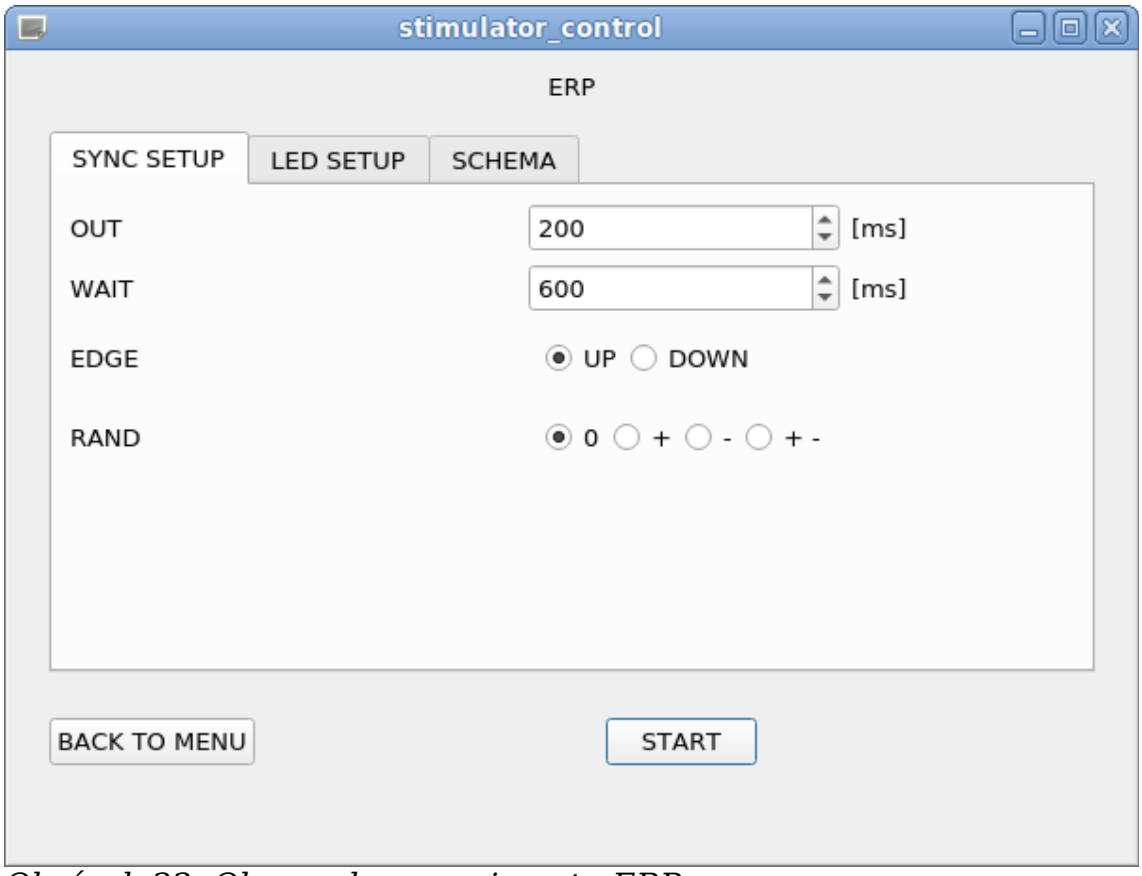
Po spuštění programu jsme v hlavním menu. Na obrázku 21 můžeme vidět tlačítka menu, tlačítka ERP, BCI a RESPONSE EXPERIMENT, která nás dovedou k jednotlivým experimentům. Tlačítko TEST MODE slouží ke čtení stavu stimulátoru jako je vnitřní napětí a teplota. Z výše uvedených tlačítek má dokončenou implementaci pouze ERP. Ostatní mají dokončené pouze uživatelské rozhraní, interakci se stimulátorem ale bohužel komunikační protokol neumožňuje. Všechny experimenty mají možnost nastavení uložit a načíst do XML souborů. Poslední tlačítko menu STIMULATOR OUTPUTS nás zavede do nastavení vizuálních a akustických stimulů. Z hlavního menu můžeme spravovat připojení přes sériová rozhraní. Můžeme vybrat zařízení a nastavit baudovou rychlost.

Rozhraní experimentu ERP je vidět na obrázku 22. Zde můžeme nastavit parametry experimentu. Tlačítkem BACK TO MENU se vrátíme do hlavního menu. Tlačítkem START/STOP spustíme nebo ukončíme experiment.

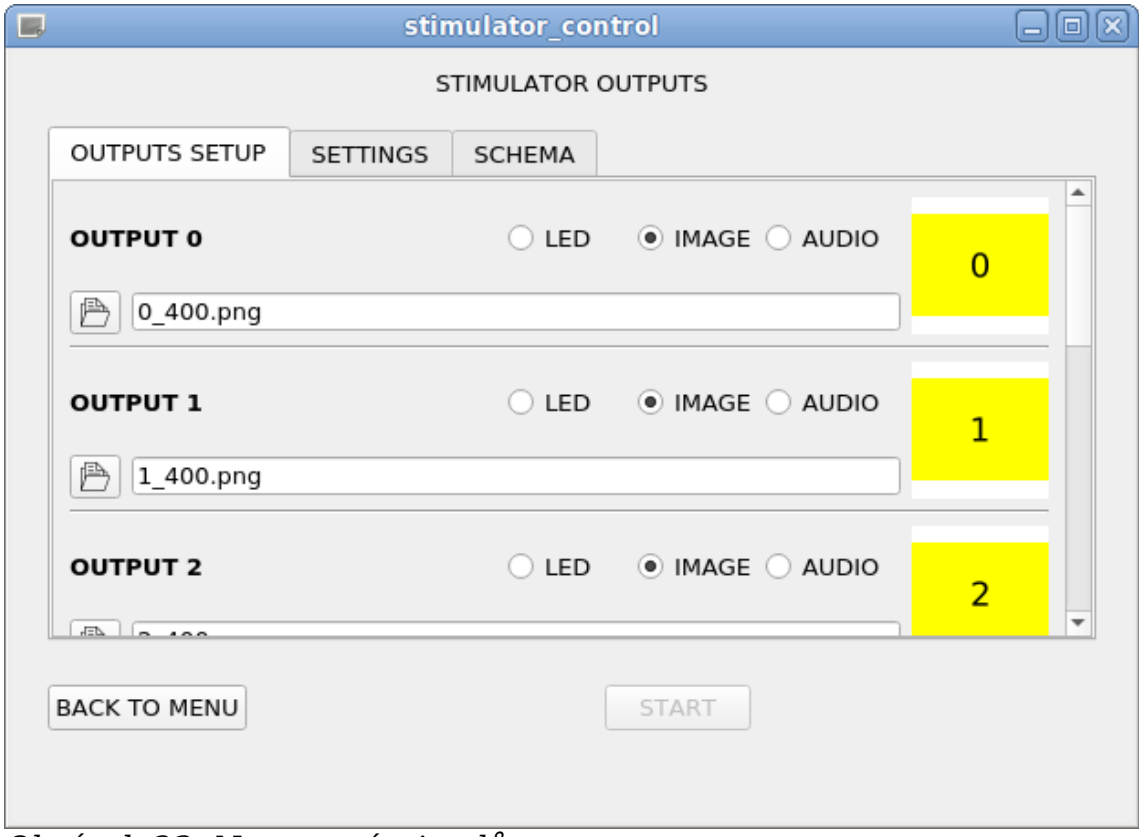
Na obrázku 23 je nastavení stimulů. Zde můžeme nastavit obrázky či zvuky. V případě kliknutí na malý náhled se spustí aplikace, která je v systému s daným formátem asociovaná a poskytne tak lepší náhled. V obrázku 24 je vidět rozhraní pro nastavení, které je obecné a souvisí se způsobem zobrazení SDL okna. Můžeme nastavit rozlišení, SW/HW renderování. Zaškrtávací pole SDL OUTPUT určuje, zda se SDL okno vůbec objeví, v případě zaškrtnutí se okno objevuje při spuštění každého experimentu. Na obrázku 25 pak vidíme rozhraní pro uložení nastavení pro daný experiment. Toto rozhraní je stejné v každém experimentu.



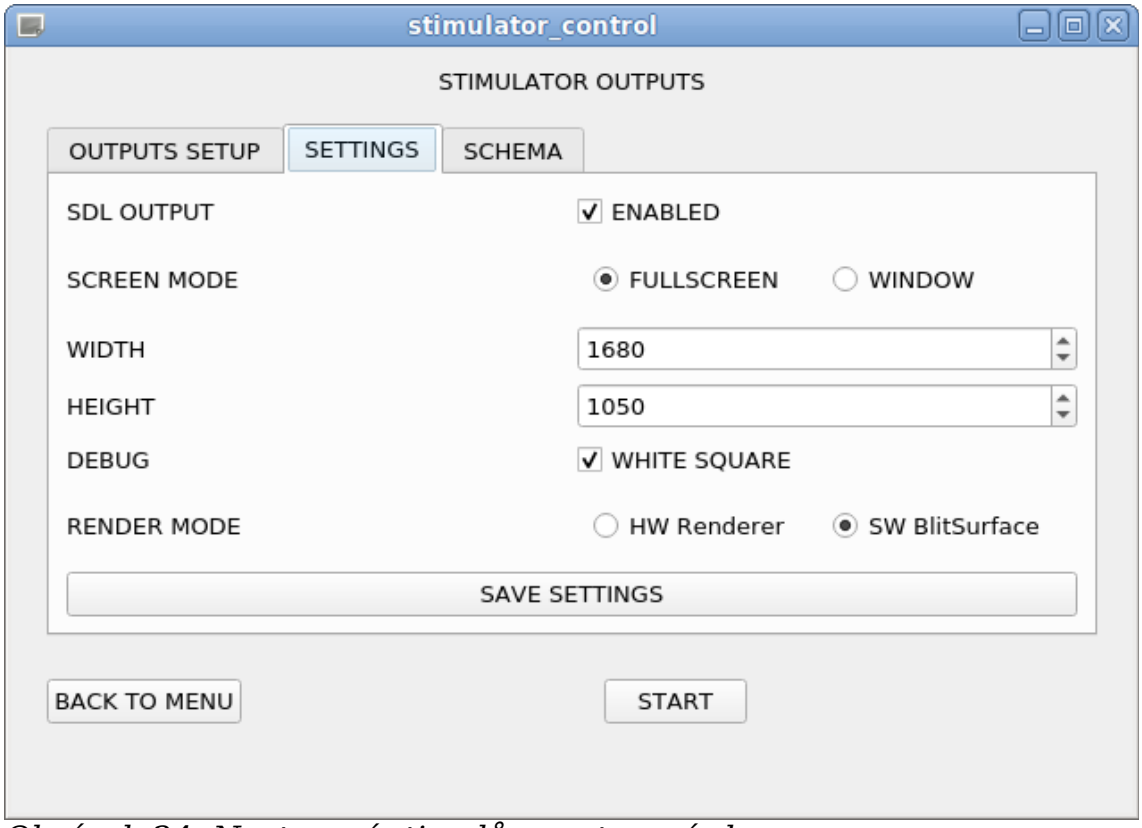
Obrázek 21: Hlavní menu



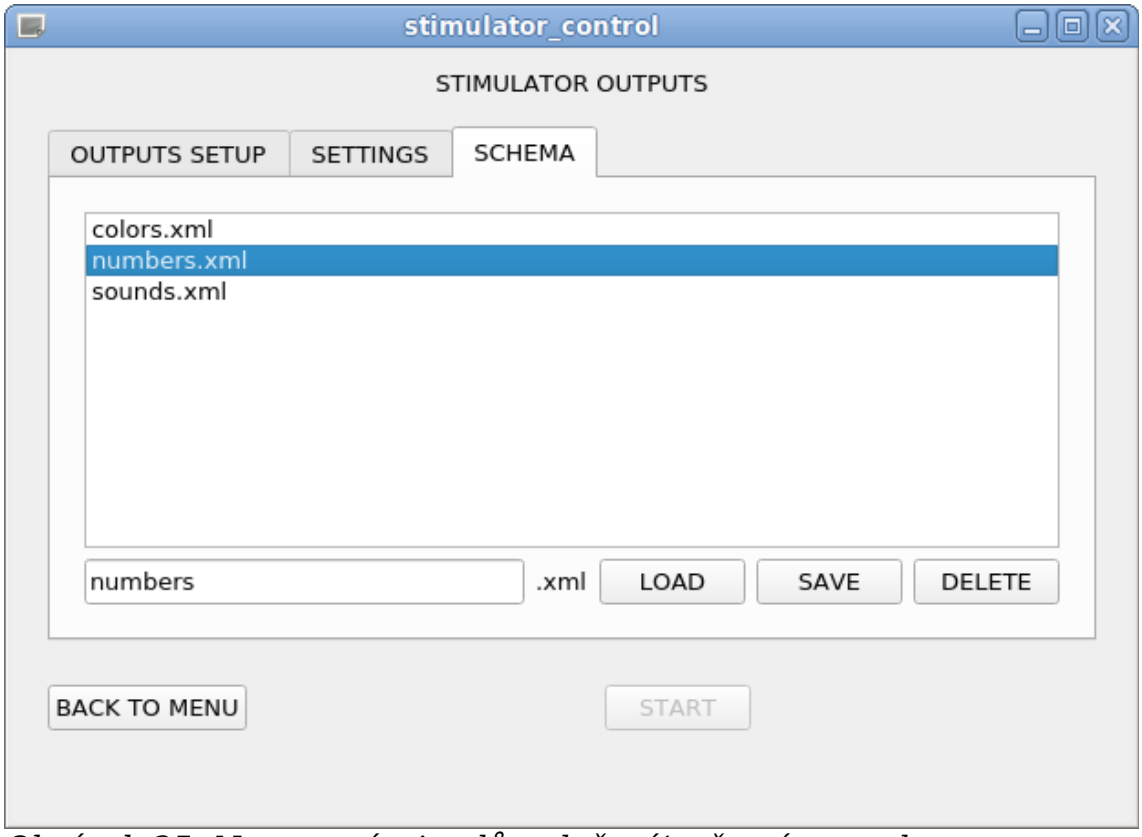
Obrázek 22: Obrazovka experimentu ERP



Obrázek 23: Nastavení stimulů



Obrázek 24: Nastavení stimulů - nastavení obrazu



Obrázek 25: Nastavení stimulů - uložení/načtení ze souboru

D Obsah CD

- stimulator_control

  zdrojové kódy uživatelského rozhraní pro ovládání stimulátoru

- SDL_output

  zdrojové kódy rozšiřujícího programu pro prezentaci vizuálních a akustických stimulů

- doc
  - bakalářská práce Milan Hajžman.pdf
- doc_src

  soubory, které jsou součástí bakalářské práce, obrázky, grafy, měření

- readme.txt

  (obsah této přílohy)