UNIVERZITA P.J. ŠAFÁRIKA V KOŠICIACH
Prírodovedecká fakulta

Automatizácia laboratórnych experimentov v prostredí LabVIEW

doc. RNDr. Erik Čižmár, PhD.

Košice 2024

Automatizácia laboratórnych experimentov v prostredí LabVIEW
Vysokoškolský učebný text k predmetu Grafické programovanie

AUTOR:
doc. RNDr. Erik Čižmár, PhD.
Prírodovedecká fakulta, Univerzita P.J. Šafárika v Košiciach

RECENZENTI:
doc. RNDr. Jan Prokleška, PhD.
Matematicko-fyzikální fakulta, Univerzita Karlova, Praha RNDr.

Jozef Kačmarčík, PhD.
Ústav experimentálnej fyziky, Slovenská akadémia vied, Košice

UMIESTNENIE: ww.unibook.upjs.sk
DOSTUPNÉ OD: 24.10.2024

ISBN 978-80-574-0354-8 (e-publikácia)

Ohana means family.
Family means nobody gets left behind, or forgotten.

— Lilo & Stitch

ABSTRACT

Textbook *Automation of Laboratory Experiments in LabVIEW environment* is devoted to the basics of graphical programming and shows the way how to communicate with measuring equipment and *Arduino* development boards in *LabVIEW*. The basic concepts of creating a virtual instrument and its interactive user environment are introduced. Three physical tasks are presented, the practical implementation of which takes place with the use appropriate instrumentation and *LabVIEW*. Practical tasks are dedicated to measuring the transmission characteristics of a low-pass RC filter, measuring the natural resonant frequency of a quartz tuning fork using the heterodyne pulse method, and mapping the magnetic field of a permanent magnet using *Arduino*.
Keywords: LabVIEW, graphical programming, virtual instrument, diagram, Arduino, quartz tuning fork, filter, magnetic field.

ABSTRAKT

Učebný text *Automatizácia laboratórnych experimentov v prostredí LabVIEW* je venovaný základom grafického programovania a spôsobu komunikácie s laboratórnymi prístrojmi a vývojárskymi doskami *Arduino* v *LabVIEW*. Predstavené sú základné koncepty tvorby virtuálneho prístroja a jeho interaktívneho užívateľského prostredia. Uvedené sú zadania troch fyzikálnych úloh, ktorých praktická realizácia prebieha s využitím vhodného prístrojového vybavenia a *LabVIEW*. Praktické úlohy sú venované meraniu prenosovej charakteristiky RC filtra typu dolná priepusť, meraniu vlastnej rezonančnej frekvencie kremennej ladičky pomocou heterodynnej pulznej metódy a mapovaniu magnetického poľa permanentného magnetu s využitím *Arduina*.
Kľúčové slová: LabVIEW, grafické programovanie, virtuálny prístroj, diagram, Arduino, kremenná ladička, filter, magnetické pole.

OBSAH

ZOZNAM SKRATIEK

API	Application programming interface
ASCII	American Standard Code for Information Interchange
DAQ	Data acquisition
dstp	DataSocket Transport Protocol
GPIB	General Purpose Interface Bus
GPIO	General-Purpose Input/Output
GUI	Graphical User Interface
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
IoT	Internet of Things
IVI	Interchangeable Virtual Instrument
LVM	LabVIEW Measurement File
LXI	LAN eXtensions for Instrumentation
MAX	Measurement & Automation Explorer
NI	National Instruments
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect Express
PWM	Pulse Width Modulation
PXI	PCI eXtensions for Instrumentation
SCPI	Standard Commands for Programmable Instruments
SQUID	Superconducting Quantum Interference Device
SVE	Shared Variable Engine
TDMS	Technical Data Management Streaming
UART	Universal Asynchronous Receiver/Transmitter
URL	Uniform Resource Locator
USB	Universal Serial Bus
VISA	Virtual Instrument Software Architecture
VPP	Peak-to-peak voltage

VRMS Root-mean square voltage

VI Virtual Instrument

1 ÚVOD

LabVIEW™ (**Lab**oratory **V**irtual **I**nstrument **E**ngineering **W**orkbench) predstavuje integrované vývojové prostredie, *angl.* Integrated Development Environment (IDE), vytvorené spoločnosťou National Instruments (NI), ktorá sa v roku 2023 stala súčasťou technologického koncernu Emerson. Spoločnosť NI vznikla v roku 1976 ako jedna z podobných technologických spoločností založených nadšencami v garáži Jamesa Trucharda (neskoršieho prezidenta spoločnosti) v Austine v Texase, kde pracoval s Jeffom Kodoskym a Billom Nowlinom na vývoji pripojenia meracích prístrojov k počítačom. V roku 1983 začal Jeff Kodosky, označovaný ako otec *LabVIEW*, pracovať na koncepte samotného softvéru a v roku 1986 bol *LabVIEW* uvedený na trh na platforme *Apple Macintosh*. Dnes sa *LabVIEW* bežne používa na zber údajov, riadenie prístrojov a priemyselnú automatizáciu v rôznych operačných systémoch vrátane *Microsoft Windows*, rôznych verzií *Unixu*, *Linuxu* a *MacOS*. Jeff Kodosky označil *LabVIEW* takto: „LabVIEW je inžinierske prostredie na vytváranie testovacích, meracích a riadiacich systémov pomocou grafického programovacieho jazyka založeného na virtuálnych prístrojoch, ktorý inšpiruje používateľov, aby sa snažili a dosiahli viac, ako očakávali.“ [1].

Samotný programovací jazyk má oficiálny názov „*G*“, pretože je grafický, aj keď mnohí označujú tento jazyk ako *LabVIEW*. Jazyk *G* je jedinečný v spôsobe, akým sa vytvára a ukladá kód. Neexistuje žiadny textový kód, ale je to diagramové zobrazenie toho, ako údaje prúdia programom. *LabVIEW* sa preto stal veľmi obľúbeným nástrojom vedcov a inžinierov, ktorí si takto môžu vizualizovať tok dát a nevenovať sa tvorbe samotného konvenčného programového kódu. *LabVIEW* ponúka aj programovacie nástroje určené pre skúsených programátorov, napr. objektovo-orientované programovanie. Aj neskúsený programátor však pomocou spôsobu ako je navrhnuté grafické programovanie a konceptu *virtuálneho prístroja*, *angl.* Virtual Instrument (VI), môže jednoducho riadiť zber experimentálnych údajov z rozsiahlej sady meracích a testovacích prístrojov, a tak jednoducho ovládať komplexné experimentálne zostavy v laboratóriu.

Existuje niekoľko rôznych verzií *LabVIEW* aj spôsob ich licencovania, v súčasnosti sa nové vydania ponúkajú vo forme ročného predplatného, pre rôzne veľký počet užívateľov pre komerčné využitie, vzdelávanie a výskum. Od roku 2017 spoločnosť NI oznámila vytvorenie novej generácie *LabVIEW* s názvom *LabVIEW NXG*, jej vývoj však bol zastavený po rôznych vývojárskych problémoch a posledné vydanie 5.1 bolo zverejnená v roku 2021. V súčasnosti je dostupný aj *LabVIEW Community Edition* ako bezplatná verzia, ktorá sa môže používať len na osobné, nekomerčné, nepriemyselné a neakademické účely. Poskytuje všetky možnosti profesionálnych verzií, dostupná je však len v 32-bitovej architektúre. *LabVIEW Community Edition* je určený pre osobné projekty ako sú: domáce hobby projekty, tvorba bezplatných projektov alebo doplnkov s otvoreným zdrojovým kódom pre komunitu

na opätovné použitie, použitie na štúdium na nadchádzajúcu certifikáciu, použitie doma na udržiavanie zručností alebo skúšanie nových nápadov mimo práce. Aj táto verzia *LabVIEW* obsahuje *G Web Development Software* [2] na vytváranie webových aplikácií a *LabVIEW Hobbyist Toolkit* [3][1], na podporu práce s populárnymi vývojárskymi mikročipovými doskami *Raspberry Pi* [4], *BeagleBoard* [5] a *Arduino* [6].

V tomto učebnom texte k predmetu *Grafické programovanie* sa budeme venovať základom tvorby programov v prostredí *LabVIEW*, spôsobu komunikácie s laboratórnymi prístrojmi a vývojárskymi doskami *Arduino*. Predstavíme aspoň základné koncepty práce v *LabVIEW*, podrobné informácie o knižniciach funkcií je možné naštudovať v dokumentácii *LabVIEW*, ktorá je dostupná aj online [7], prípadne v cudzojazyčnej literatúre [8–14]. V závere predstavíme zadania a spôsob riešenia troch praktických úloh automatizácie jednoduchého fyzikálneho merania a ovládania periférií *Arduino* v prostredí *LabVIEW*. Vzhľadom na to, že prostredie *LabVIEW* je dostupné len v siedmich jazykoch (angličtina, čínština, francúzština, japončina, kórejčina, nemčina a španielčina), pri popise programovacích funkcií a IDE budeme využívať anglickú terminológiu, pričom všeobecné pojmy budú uvedené aj v slovenčine.

1 *Hobbyist Toolkit* vznikol pôvodne ako komunitný projekt *LabVIEW MakerHub LINX toolkit*

2 PROSTREDIE LABVIEW

2.1 ZÁKLADNÉ POJMY V LABVIEW

2.1.1 Virtuálny prístroj

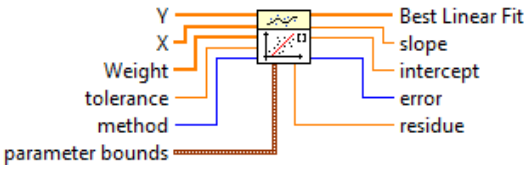
Pojem virtuálneho prístroja v skutočnosti nie je vlastný len prostrediu *LabVIEW*, ktoré tento pojem používa na pomenovanie programu aj so súborovou príponou *.vi*. Je to všeobecný názov sady nástrojov, ktoré presúvajú možnosti jednotlivých fyzických meracích prístrojov a ovládacích prvkov do počítača tak, že samotné softvérové riešenie zabezpečuje získanie údajov zo senzorov a ovládanie aktuátorov (prepínače, motory, ventily a pod.) s kompletným grafickým užívateľským prostredím, *angl.* Graphical User Interface (GUI). Dokonca aj bežný textový vstup a výstup tradičných programovacích jazykov môže reprezentovať VI. Až predstavenie osobných počítačov s operačnými systémami podporujúcimi GUI a predstavenie *LabVIEW* prinieslo predstavu VI, kde môžeme zreprodukovať ovládací panel samotných fyzických meracích prístrojov, skombinovať ich, spojiť do zložitého celku, ktorý môže ovládať komplexné systémy akými môžu byť distribúcia elektrickej energie alebo zemného plynu v energetickej infraštruktúre. NI definuje VI ako štandardizované počítače vybavené užívateľsky prívetivým aplikačným softvérom, cenovo výhodným hardvérom a softvérom ovládačov, ktoré spoločne vykonávajú funkcie tradičných prístrojov. Spojenie softvérového a hardvérového vývoja (rôzne systémy na zber dát) zo strany NI umožnilo spoločnosti pretaviť *LabVIEW* do uznávaného štandardu a dominovať trhu s podobnými riešeniami. V súčasnosti priamo NI alebo výrobcovia meracej techniky poskytujú ovládače potrebné na jej integráciu v *LabVIEW*.

2.1.2 Grafická reprezentácia funkcií a beh programu

V konvenčných programovacích jazykoch je volanie a vykonanie funkcie alebo podprogramu zvyčajne zapísané ako riadok programového kódu, kde zadávame sadu vstupných a výstupných parametrov spolu s názvom volanej funkcie alebo podprogramu napríklad vo forme

```
vystupy = funkcia (vstupy)
```

pričom v prostredí *LabVIEW* toto volanie funkcie bude reprezentované len ikonou s označeným pripojením vstupných a výstupných parametrov ako na obr. 2.1, ikona predstavuje uzol vo vykonávanom programe. *LabVIEW* používa pri spúšťaní VI model *toku údajov* (*angl. dataflow*). Vstupné a výstupné parametre pripájame k uzlu pomocou *wires* (*angl.*), čo môžeme v slovenskej terminológii označiť ako drôty, vodiče, vlákna alebo spoje, v tomto



Obr. 2.1: Grafická reprezentácia funkcie s pripojením vstupných a výstupných parametrov v prostredí *LabVIEW*.

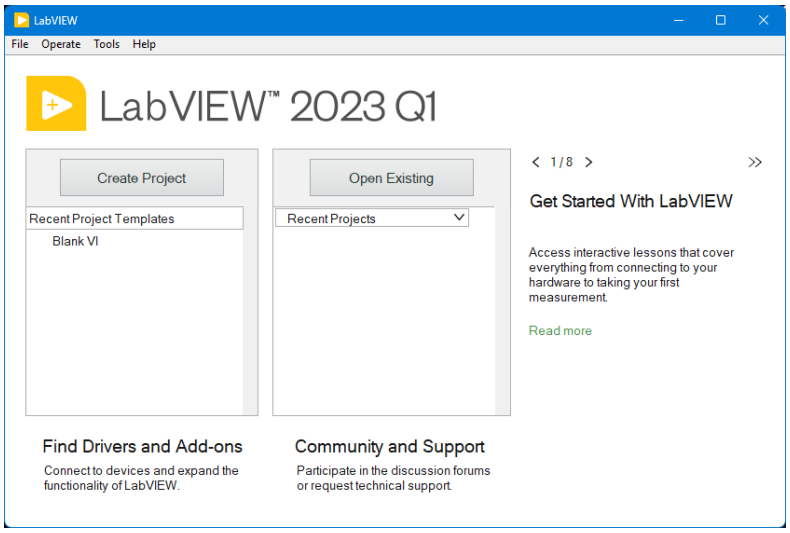
texte budeme využívať názov *vodiče*. Vodiče v prostredí *LabVIEW* predstavujú spôsob ako sa definuje tok údajov, teda samotná následnosť vykonávania jednotlivých krokov VI. Nie je preto dôležité ako sú v programe uložené jednotlivé ikony reprezentujúce funkcie, ale ako sú spojené pomocou vodičov. Uzol sa spustí, keď sú k dispozícii aktuálne hodnoty na všetkých jeho vstupných termináloch. Keď uzol ukončí vykonávanie, dodá údaje do svojich výstupných terminálov a odovzdá ich ďalšiemu uzlu v ceste toku údajov. Zvýraznené vodiče ako na obr. 2.1 predstavujú pripojenie povinných parametrov, a ich nepripojenie bude znamenať chybu vo vytvorenom VI a *LabVIEW* neumožní spustenie pripraveného VI. Tento spôsob vykonávania programu umožňuje aj schopnosť pracovať s paralelne bežiacimi kódmi. Môžu súčasne bežať dve rôzne štruktúry cyklov a rýchlosť vykonávania jedného cyklu neovplyvní rýchlosť vykonávania druhého cyklu bez ohľadu na jeho obsah. Väčšina textových programovacích jazykov ako *C++*, *Java*, *Python*, a pod., sa riadi modelom riadeného toku vykonávania programu. V tomto modeli určuje postupnosť vykonávania programu sekvenčné poradie programových prvkov, takže dokážu naraz vykonávať iba jeden cyklus v jednom okamihu.

2.2 NÁSTROJE VÝVOJÁRSKEHO PROSTREDIA LABVIEW

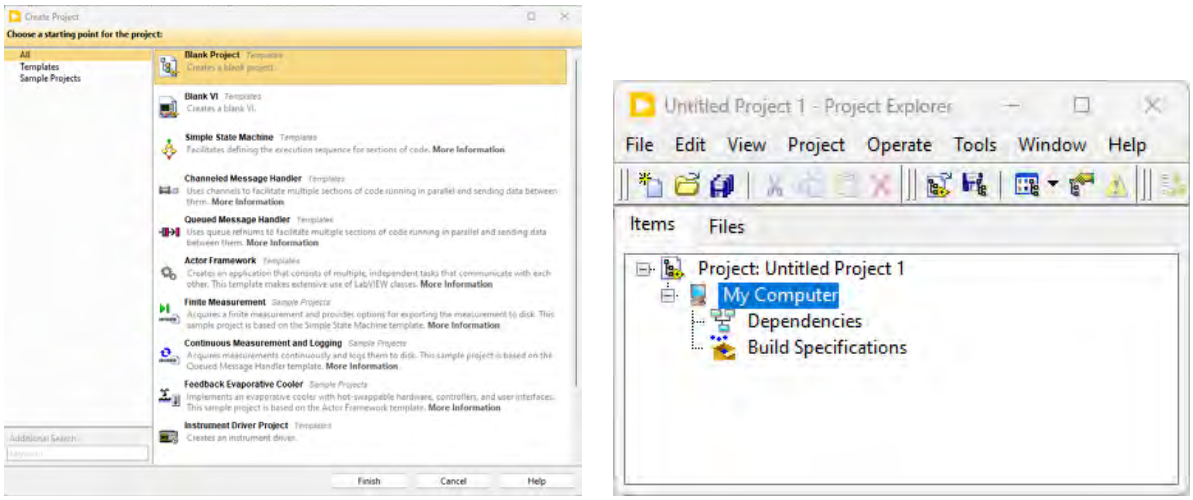
2.2.1 Predný panel a Blokový diagram

Pri vytváraní programov v *LabVIEW* máme možnosť po spustení vývojárskeho prostredia na úvodnom okne obr. 2.2 vytvoriť nový projekt alebo len jeden nový VI (prípadne otvoriť existujúce). V prípade, že pracujeme na zložitejšom projekte, ktorý zahŕňa viaceré VI, komunikáciu počítača s hardvérom, vytváranie spustiteľného súboru alebo konfiguračného súboru, je vhodné vytvoriť nový projekt, prázdny alebo podľa jednej z ponúkaných šablón (obr. 2.3a). S obsahom projektu potom pracujeme cez *Project Explorer* (obr. 2.3b). Cez *Project Explorer* môžeme vytvoriť nový VI v ponuke *File ▸ New VI*, prípadne sa môžeme priamo z úvodného okna prostredia obr. 2.2 vydať cestou tvorby jedného VI (*Blank VI*). Pre jednoduchosť vytvoríme len *Blank VI* a ukážeme si základné prvky vývojárskeho prostredia. Po vytvorení *Blank VI* sa nám otvoria dve okná, *Front Panel* (*Predný panel*) a *Block Diagram* (*Blokový diagram*) ako na obr. 2.4.

Predný panel je používateľským rozhraním VI, ktoré zostavíme pomocou ovládacích prvkov a indikátorov, ktoré sú interaktívnymi vstupnými, resp. výstupnými uzlami VI. Ovládacie



Obr. 2.2: Úvodná obrazovka prostredia *LabVIEW* po spustení.

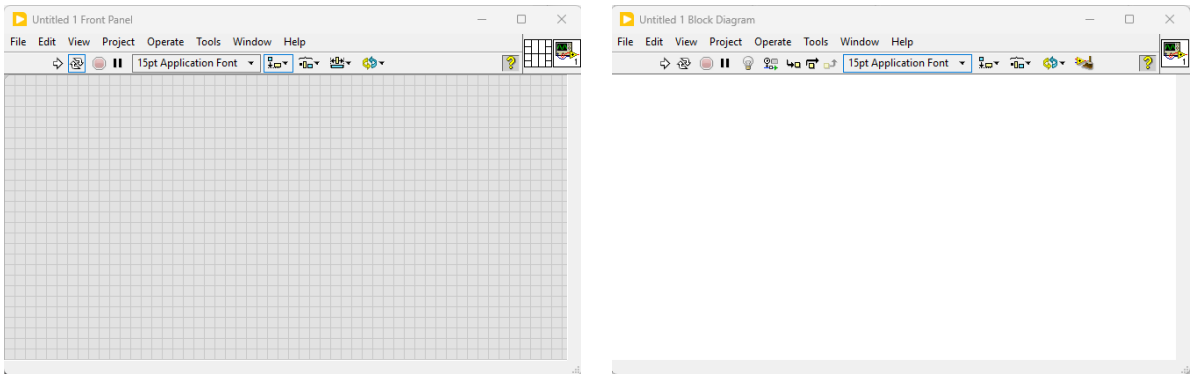


(a) **(b)**

Obr. 2.3: Sprievodca na vytvorenie nového projektu a *Project Explorer* po vytvorení prázdneho projektu.

prvky sú gombíky, tlačidlá, ovládače a iné typy vstupov. Indikátory sú grafy, virtuálne LED diódy a iné zobrazovacie formy. Ovládacie prvky simulujú vstupné zariadenia prístroja a dodávajú údaje do *Blokového diagramu*. Indikátory simulujú výstupné zariadenia prístroja a zobrazujú údaje, ktoré *Blokový diagram* získava alebo generuje. *Blokový diagram* obsahuje grafický zdrojový kód. Objekty z *Predného panelu* sa na *Blokovom diagrame* zobrazujú ako uzly, ktoré je možné pripojiť k vstupným alebo výstupným terminálom funkcií a štruktúr zo zabudovaných knižníc *LabVIEW*. Vodiče spájajú jednotlivé uzly na *Blokovom diagrame* vrátane riadiacich a indikačných uzlov, funkcií a štruktúr.

Okrem spoločnej ponuky nástrojov *Menu* (*File*, *Edit*, *View*, *Project*, *Operate*, *Tools*, *Window* a *Help*) obsahuje *Predný panel* a *Blokový diagram* aj niekoľko panelov nástrojov, ktoré sú rovnaké pre oba panely, ale aj špecifické pre každý z nich:



(a) (b)

Obr. 2.4: *Front Panel* (*Predný panel*) a *Block Diagram* (*Blokový diagram*).

- *Program execution* (*Beh programu*) v oboch paneloch obsahuje tlačidlá pre jednorázové spustenie (*Run*), spustenie kontinuálneho behu (*Run Continuously*), prerušenie behu (*Abort execution*), dočasné pozastavenie behu (*Pause/Continue*);
- *Debugging tools* (*Nástroje ladenia*) v *Blokovom diagrame* obsahujú tlačidlá pre zvýraznenie vykonávania príkazov (*Highlight Execution*) – počas behu VI sa graficky zvýrazňuje vykonávaná časť diagramu, zobrazenie hodnôt tečúcich na vodičoch (*Retain Wire Values*) – umiestnenie sondy na vybranom vodiči, vykonávanie kódu krok za krokom (*Step Into* , *Step Over* a *Step Out*);
- *Font properties* (*Vlastnosti písma*) v oboch paneloch obsahuje nastavenie typu, veľkosti, štýlu a farby písma objektov;
- *Object organization* (*Rozloženie objektov*) v oboch paneloch obsahuje tlačidlá pre rôzne spôsoby automatického usporiadania objektov na ploche panelov: zarovnanie (*Align Objects*), reorganizovanie (*Distribute Objects*), usporiadanie vrstiev a vytváranie skupín objektov (*Reorder*), zmena rozmerov objektov na *Prednom paneli* (*Resize Objects*);
- *Clean up Diagram* (*Vyčistenie diagramu*) v *Blokovom diagrame* automaticky reorganizuje diagram a vymaže prerušené vodiče, vhodné pre malé blokové diagramy, zložitejšie diagramy môžu byť pri takej automatickej zmene usporiadania ťažko čitateľné;
- *Show/Hide Context Help Window* (*Zobraz/Skry okno kontextového pomocníka*) v oboch paneloch umožní zobraziť plávajúce okno s pomocou k objektu, nad ktorým práve máme umiestnený ukazovateľ myši.

Dôležitým objektom vo vývojárskom prostredí *LabVIEW* je *VI icon* (ikona VI) a *Connector pane* (panel konektorov) na obr. 2.4. Oba sú zobrazené v hornom pravom rohu okna

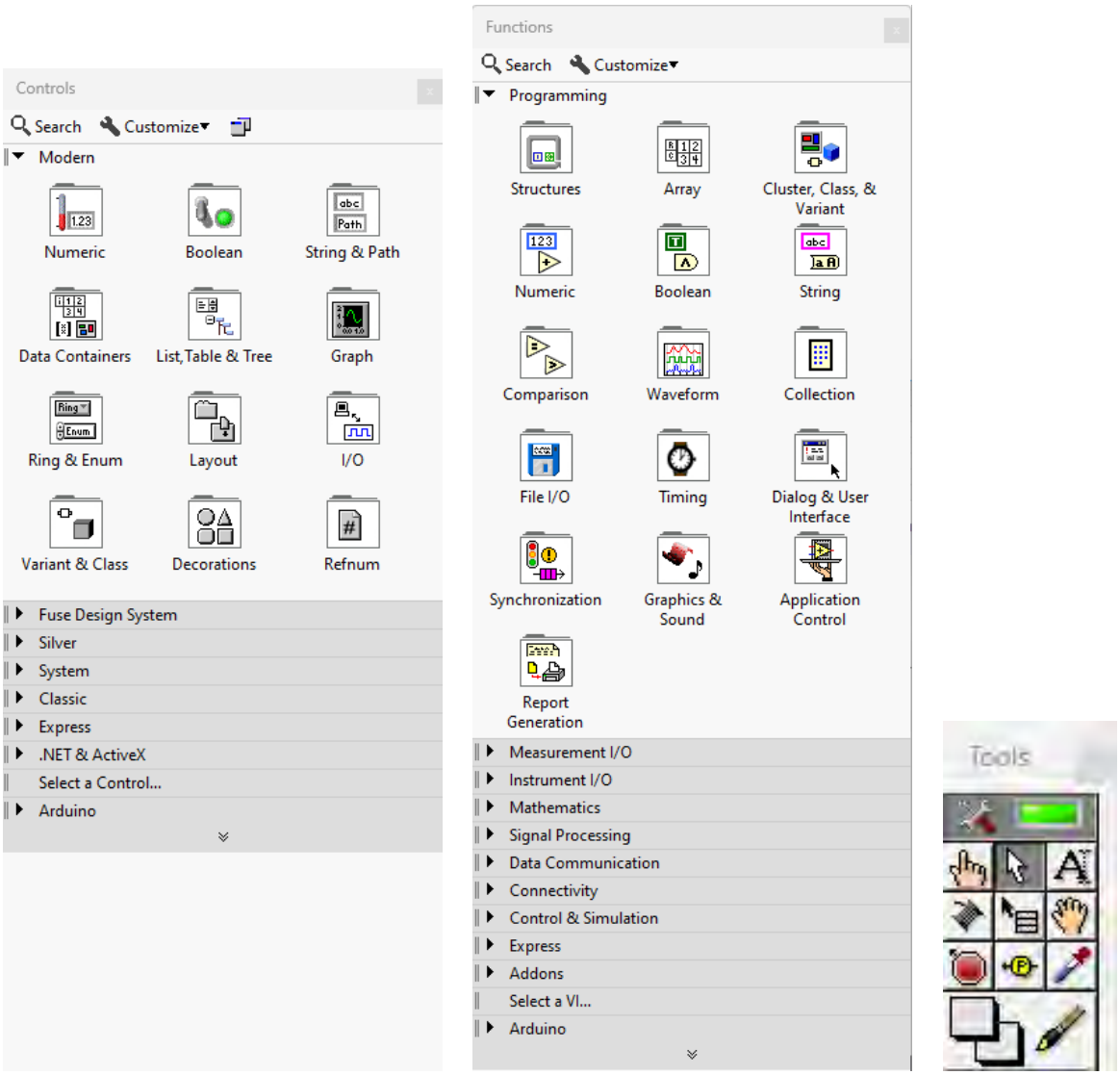
Blokového diagramu, v okne *Predného panelu* je zobrazená len ikona VI. Ako uvidíme neskôr, každý VI môže byť vložený do iného VI ako podprogram, resp. funkcia, preto ho v *Blokovom diagrame* iného VI musíme vedieť identifikovať. Na tento účel máme zabudovaný editor ikon, aby sme pre každý vytvorený VI mohli vytvoriť aj jeho reprezentačnú ikonu. Panel konektorov zabezpečuje priradenie vstupných a výstupných parametrov jednotlivým premenným vo vnútri podprogramu, resp. funkcie, ktoré sme vytvorili ako nové VI.

2.2.2 Palety nástrojov

Na samotnú pracovnú plochu *Predného panelu* a *Blokového diagramu*, ktoré sú farebne odlíšené (šedá alebo biela), môžeme vkladať objekty použitím zodpovedajúcej palety nástrojov, ktorá sa zobrazí po kliknutí pravým tlačidlom myši na pracovnú plochu, prípadne sa dá zobraziť natrvalo ako plávajúce okno z ponuky *View*. Pre *Predný panel* je to *Controls Palette* (*Paleta ovládačov*) a pre *Blokový diagram* je to *Functions Palette* (*Paleta funkcií*) zobrazené na obr. 2.5a a obr. 2.5b. V súčasnosti *Controls Palette* obsahuje viacero grafických štýlov ovládacích prvkov a indikátorov zobrazených na *Prednom paneli*, ktoré sa postupne objavovali v novších vydaniach *LabVIEW*. V najnovších vydaniach sú to *Modern*, *Fuse Design System*, *Silver*, *System* a *Classic* štýl, ich funkčnosť je však identická.

Pre oba panely je k dispozícii naviac *Tools Palette* (*Paleta nástrojov*) na obr. 2.5c, v ktorej je možné prepínať nástroje na manipuláciu objektov na pracovnej ploche *Predného panelu* a *Blokového diagramu*. Zmenou stavovej virtuálnej svetelnej diódy  je možné *Tools Palette* prepnúť do automatického režimu, kedy sa nám zobrazuje potrebný nástroj podľa toho, nad akým objektom práve máme ukazovateľ myši. Ukazovateľ myši sa tomu adekvátne graficky mení. Táto paleta obsahuje:

- *Operation tool* (*Nástroj na ovládanie*)  – interakcia z objektami ako je zmena hodnoty, zmena polohy posuvného ovládača, stlačenie tlačidla a pod.;
- *Positioning tool* (*Polohovací nástroj*)  – posúvanie alebo zmena veľkosti objektov na pracovnej ploche;
- *Labeling tool* (*Nástroj na značkovanie*)  – pridanie textu v pracovnom priestore;
- *Wiring tool* (*Nástroj na vytváranie vodičov*)  – prepojenie terminálov pomocou vodiča;
- *Object shortcut menu tool* (*Nástroj na otvorenie menu nastavení objektov*)  – po kliknutí na objekt sa objaví ponuka s nastavením jeho vlastností. Tento nástroj sa otvára aj po kliknutí pravým tlačidlo myši z kontextovej ponuky;
- *Scrolling tool* (*Nástroj na posúvanie v pracovnom priestore*)  ;
- *Breakpoint tool* (*Nástroj na vloženie bodu prerušenia*)  – len pre *Blokový diagram*, označenie miesta programu, kde sa má pri behu zastaviť;



(a) (b) (c)

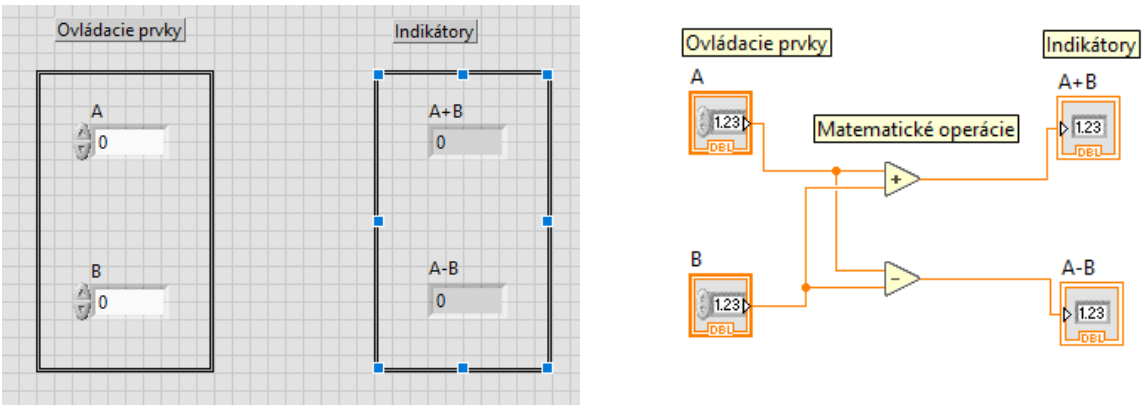
Obr. 2.5: *Controls Palette* (*Paleta ovládačov*), *Functions Palette* (*Paleta funkcií*) a (*Paleta nástrojov*) nastavená na automatickú zmenu nástrojov.

- *Probe tool* (*Nástroj na vloženie sondy*) – len pre *Blokový diagram*, označenie miesta na vodiči, kde sa má pri behu monitorovať aktuálna hodnota presúvaná vodičom;
- *Get color tool* (*Nástroj na určenie farby*) – určenie aktuálnej farby v mieste pracovnej plochy;
- *Coloring tool* (*Nástroj na zafarbenie*) – vyfarbenie vybranej plochy.

2.2.3 Vytvorenie VI

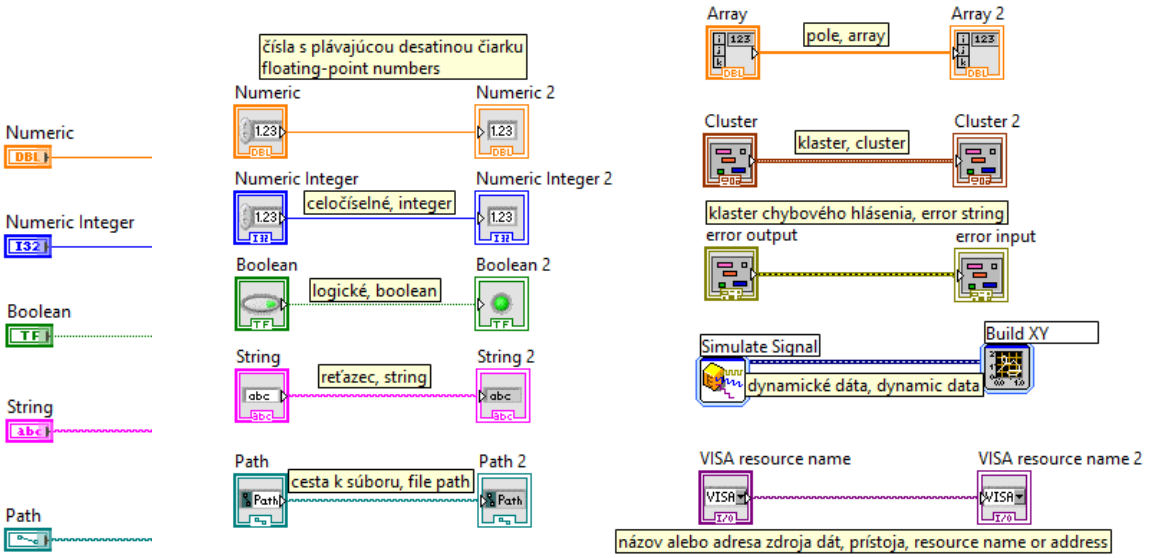
Po vytvorení prázdneho VI nasleduje samotná tvorba programu pomocou *Controls Palette* a *Functions Palette*. Jednotlivé prvky *Predného panelu* a *Blokového diagramu* jednoducho vložíme kliknutím na vybraný ovládač, indikátor alebo ikonu funkcie v zodpovedajúcej palete nástrojov, presunutím kurzora myši na miesto vloženia a kliknutím. Keď vytvoríme objekt na *Prednom paneli*, na *Blokovom diagrame* sa vytvorí uzol. Tieto uzly umožňujú prístup k objektom na *Prednom paneli* z kódu *Blokového diagramu*. Vstupy a výstupy uzlov objektov z *Predného panelu* a funkcií *Blokového diagramu* sú zvýraznené terminálmi na pripojenie vodiča. *Wiring tool* sa následne používa na prepojenie terminálov. Mierime koncom vodiča, ktorý visí zo zobrazeného ukazovateľa cievky s vodičom na *Wiring tool*  . Na tomto mieste bude vodič pripojený k terminálu. Pod nástrojom sa pri pohybe nad objektami zobrazuje názov terminálov, ku ktorým sa pripájame. Keď *Wiring tool* presúvame nad terminál, bude blikať. Pomôže nám to identifikovať miesto, kam sa vodič pripojí, vykonáme to kliknutím myši. Ak výsledné automatické trasovanie vodiča po zapojení nevyzerá dobre, klikneme pravým tlačidlom myši na príslušný vodič a vyberieme možnosť *Clean Up Wire*, čím upravíme jeho trasu. Príklad takto vytvoreného jednoduchého VI, ktorý sčíta a odpočíta dva vstupy A a B a následne výsledok zapíše do dvoch indikátorov A+B a A-B je zobrazený na obr. 2.6.

Každý uzol obsahuje užitočné informácie o objekte na *Prednom paneli*, ktorému zodpovedá. Napríklad farba a symboly poskytujú informáciu o type údajov. Čísla s plávajúcou desatinnou čiarkou sú reprezentované oranžovými svorkami a písmenami *DBL*. Logické uzly sú zelené so znakmi *TF*. Vo všeobecnosti platí, že sa môžu vodičmi prepájať uzly a terminály rovnakej farby, zelené k zeleným atď. Nie je to pevne stanovené pravidlo, *LabVIEW* umožňuje používateľovi pripojiť napríklad modrý terminál (celočíselná hodnota) k oranžovému terminálu, pretože niektoré uzly a funkcie sú polymorfné, teda správajú sa podľa typu zadaných vstupných údajov. Ovládacie prvky majú na uzle terminál so šípkou zobrazený na pravej strane a majú hrubý okraj. Indikátory majú na uzle terminál so šípkou



(a) (b)

Obr. 2.6: *Predný panel* a *Blokový diagram* programu, ktorý sčíta a odpočíta dva vstupy A a B, výsledok zapíše do dvoch indikátorov A+B a A-B.



Obr. 2.7: Bežné typy údajov a prepájajúcich vodičov identifikovaných farbou.

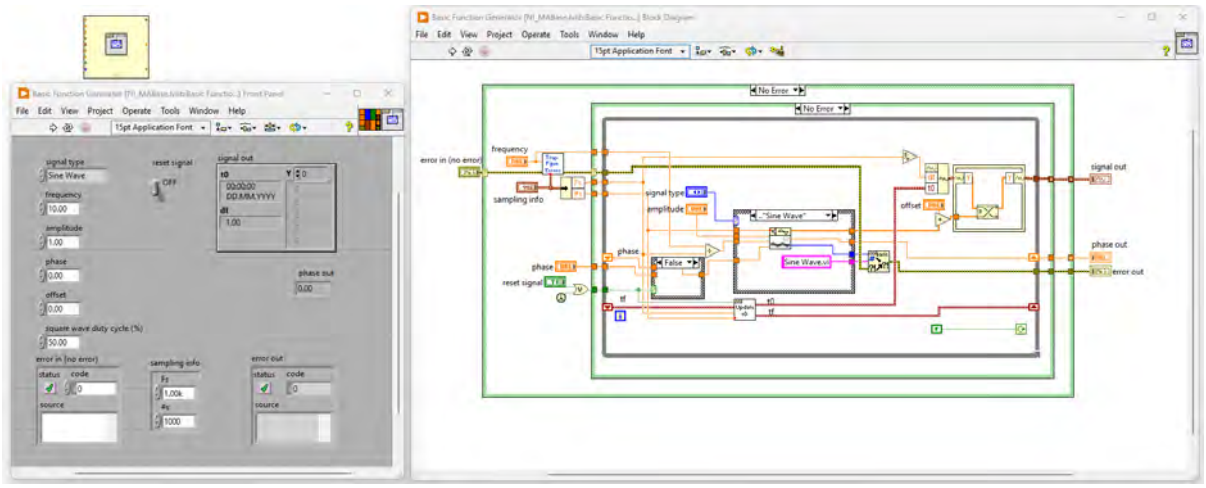
zobrazený na ľavej strane a tenký okraj. Pre zapojenie vodičov v *LabVIEW* platia logické pravidlá: Každý vodič musí mať jeden (ale len jeden) zdroj (alebo ovládací prvok) a každý vodič môže mať viacero cieľov (indikátorov alebo funkcií). Podľa farby zapojeného vodiča je tiež možné identifikovať typ údajov presúvaných vodičom medzi terminálmi. Konštanty, indikátory alebo ovládacie prvky možno vytvoriť a automaticky pripojiť k vstupu alebo výstupu funkcie aj kliknutím pravým tlačidlom myši na vstup/výstup a výberom položky z kontextovej ponuky vytvoriť konštantu, indikátor alebo ovládací prvok. Konštanty, indikátory alebo ovládacie prvky v *Blokovom diagrame* môžu byť zobrazené dvoma spôsobmi: ako ikony na obr. 2.7b alebo jednoduché uzly na obr. 2.7a, spolu so zodpovedajúcou farbou prepájajúcich vodičov. Najbežnejšie sú numerické typy dát (oranžová, modrá), reťazce (ružová), logické (zelená), adresárová alebo súborová cesta v operačnom systéme (modrozelená), adresa alebo názov prístroja alebo hardvérového zdroja (purpurová) a dátový klaster (hnedá).

Do *Blokového diagramu* je možné vložiť tri typy funkcií:

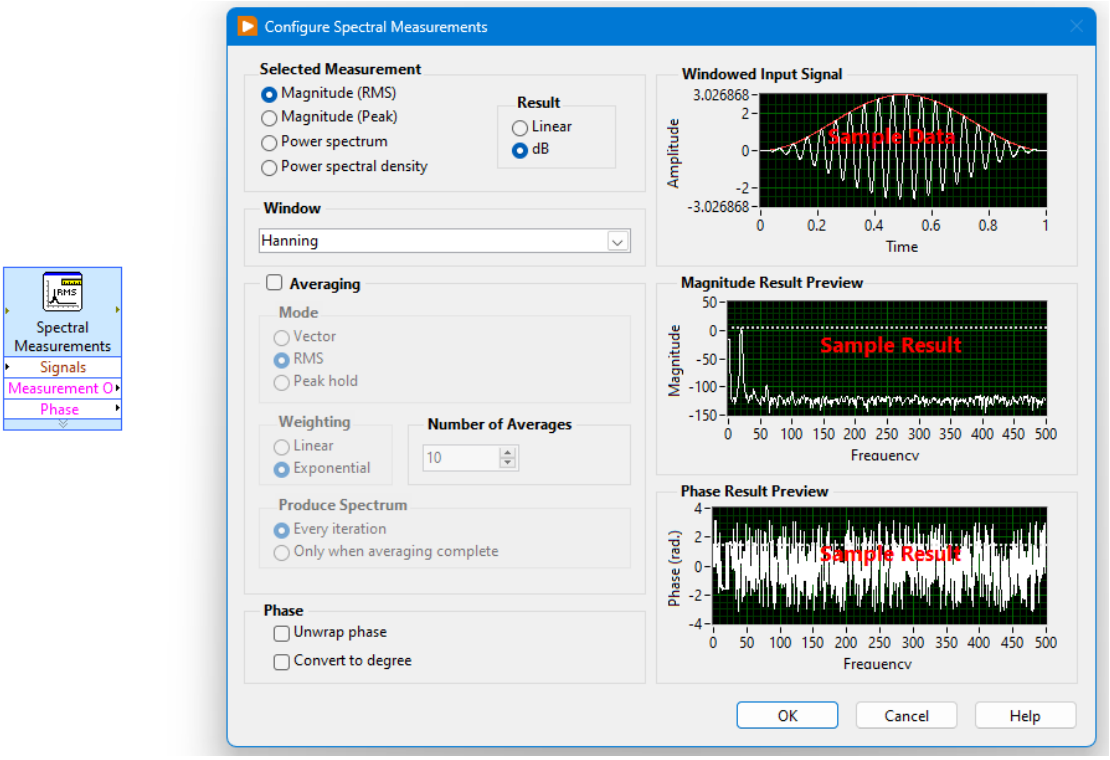
- *Fuction* (*Funkcia*) – predstavujú základné operácie, napr. matematické operácie, nie je možné zobraziť ich *Predný panel* ani *Blokový diagram*. Je možné ich zobraziť ako ikonu.

- *Standard VI* (*Štandardné VI*) – predstavujú zložité manipulácie s údajmi, komunikáciu s prístrojmi a pod., môžu to byť aj užívateľom vytvorené VI použité ako podprogramy, je možné zobraziť ich *Predný panel* aj *Blokový diagram* (obr. 2.8).

- *Express VI* (*Expresné VI*) – predstavujú zložité manipulácie s údajmi, komunikáciu s prístrojmi a pod., zavedené vo vydaní 7.0. Ide o interaktívne VI, ktoré majú konfiguračné dialógové okno, ktoré umožňuje používateľovi prispôsobiť funkčnosť *Express VI*.



Obr. 2.8: Zobrazenie *Predného panelu* a *Blokového diagramu* vloženého *Štandardného VI*.



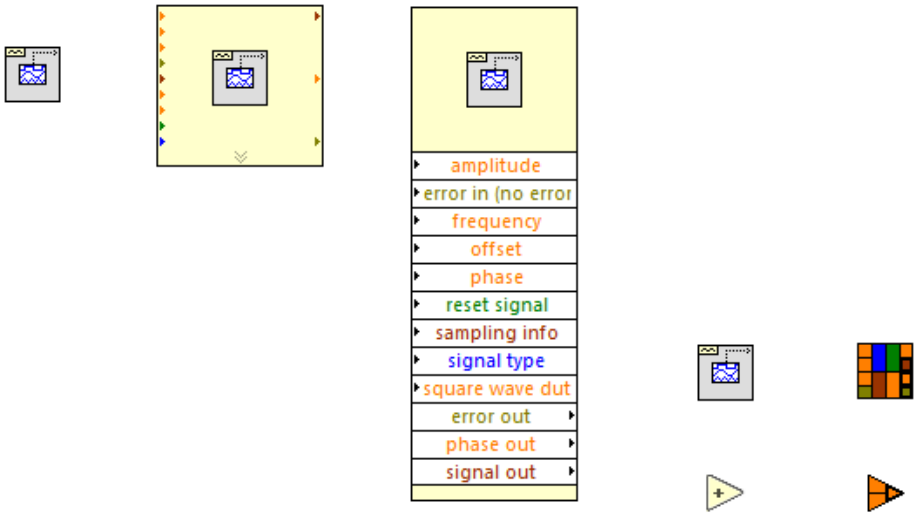
Obr. 2.9: Zobrazenie konfiguračného dialógového okna vloženého *Expresného VI*.

LabVIEW potom na základe týchto nastavení vygeneruje vložené VI. Nie je možné zobraziť ich *Predný panel* ani *Blokový diagram*, len konfiguračné dialógové okno (obr. 2.9).

Štandardné VI a *Expresné VI* je možné v *Blokovom diagrame* zobraziť ako ikonu, uzol (*angl. expandable node*) alebo uzol v rozšírenom zobrazení (*angl. expanded node*), keď sú vstupy a výstupy označené zodpovedajúcim textom (obr. 2.10a). V zobrazení uzla majú *Štandardné VI* svetložltú farbu rámu, *Expresné VI* majú svetlomodrú farbu rámu. Každý

typ vloženej funkcie alebo VI je možné zobraziť aj ako tzv. konektor, keď sú namiesto ikony pre ľahkú identifikáciu zobrazené farebne odlíšené vstupné a výstupné terminály funkcie alebo VI, (*Connector pane*) ako na obr. 2.10b. Na výber je niekoľko geometrických vzorov konektorov (*angl. Patterns*). Ak chceme zobraziť konektor funkcie, klikneme pravým tlačidlom myši na funkciu a vyberieme položku *Visible Items* ▸ *Terminals*.

Po vytvorení VI s vlastnou ikonou a definovaným konektorom ho teda môžeme použiť v inom VI ako tzv. *subVI*, čo zodpovedá tvorbe podprogramu v textových programovacích jazykoch. Používanie *subVI* pomáha rýchlo spravovať zmeny a ladiť *Blokový diagram*. Konektor definuje vstupy a výstupy, ktoré môžete pripojiť k VI, aby sme ho mohli používať ako *subVI*. Každý obdĺžnik na konektore *subVI* predstavuje terminál, ktorému priradíme ovládacie prvky alebo indikátory na *Prednom paneli*. Počet terminálov, ktoré *LabVIEW* zobrazí na konektore, závisí od počtu ovládacích prvkov a indikátorov na *Prednom paneli*.[1] Ak chceme priradiť terminál k ovládaciemu prvku alebo indikátoru na *Prednom paneli*, klikneme na terminál na konektore pomocou *Wiring tool* a následne klikneme na ovládací prvok alebo indikátor, ktorý chceme priradiť terminálu. Terminál zmení farbu podľa pripojeného dátového typu. Takto upravené *SubVI* je možné vložiť do iného VI, napr. cez ponuku *Select a VI...* z *Functions Palette*.



(a) (b)

Obr. 2.10: a) Zobrazenie vloženého VI, v tomto prípade *Štandardné VI* ako ikony, uzla (*angl. expandable node*) alebo uzla v rozšírenom zobrazení (*angl. expanded node*). b) Zobrazenie vloženej funkcie alebo VI vo forme ikony alebo konektora.

1 V starších vydaniach *LabVIEW* zobrazíme konektor kliknutím pravým tlačidlom myši na ikonu VI v pravom hornom rohu okna *Predného panela* a z kontextovej ponuky vyberieme položku *Show connector*.

2.2.4 Lokálne premenné

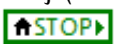
Na rozdiel od bežných textových programovacích jazykov je práca s premennými a ich hodnotou v *LabVIEW* odlišná. V textových programovacích jazykoch deklarujeme premennú *a*, priradíme jej konkrétnu hodnotu, ktorú môžeme zapísať do inej premennej *b*

```
a = 5

b = a
```

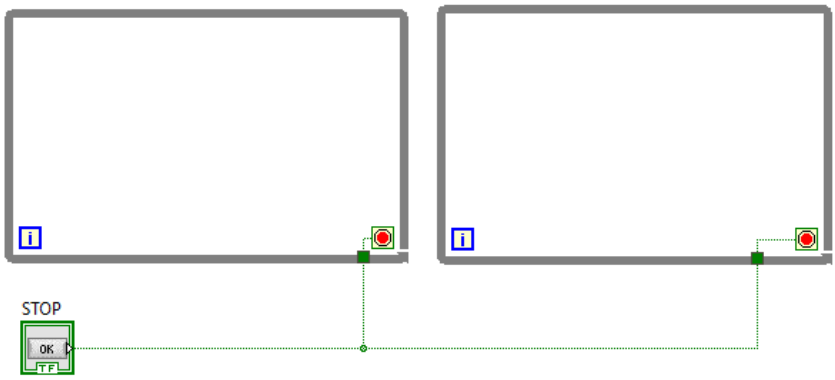
a následne ich používať v ďalšom behu programu. Vytvorenie ovládača alebo indikátora na *Prednom paneli* vytvára jeho uzol v *Blokovom diagrame* a stav, resp. hodnotu, reprezentovanú uzlom môžeme zapísať do vstupného terminálu funkcie alebo z výstupného terminálu funkcie môžeme zapísať stav, resp. hodnotu, do uzla pomocou prepojenia vodičom. Čo však v prípade, ak danú hodnotu potrebujeme zapísať alebo načítať v inej časti zložitého diagramu, kde to nie je možné vykonať pomocou pripojenia vodičom, alebo by to viedlo k nekorektnému behu VI? Príkladom môžu by dve paralelne bežiace podmienené opakovacie cykly[2], ktoré chceme vypnúť stlačením tlačidla, z ktorého získavame logickú hodnotu pravda alebo nepravda (*true* alebo *false*). V takom prípade uzol tlačidla nemôže byť umiestnený mimo oboch cyklov a z neho pripojené vodiče na rozhodovací terminál podmienených opakovacích cyklov, lebo jeho hodnota sa počas opakovanie cyklov neaktualizuje, ako je zobrazené na obr. 2.11a. Riešenie takejto úlohy vyžaduje použitie tzv. *lokálnych premenných* (*angl. Local Variables*), ktoré umožňujú prenášať informáciu o stave z uzla ovládača alebo do uzla indikátora bez použitia vodiča. Informácia v lokálnej premennej sa zachováva v rámci jedného bežiaceho VI. Riešenie problému paralelne bežiacich podmienených opakovacích cyklov je pomocou lokálnych premenných jednoduché, zobrazené na obr. 2.11b. V prvej slučke zaznamenávame priamo stav tlačidla *STOP* a v druhej stav tlačidla reprezentuje lokálna premenná *STOP*. Je však potrebné vhodne zabrániť prípadu, že hodnota lokálnej premennej by bola prepísaná z iného miesta diagramu pred tým, ako sme potrebovali zistiť jej predchádzajúcu hodnotu.

Lokálnu premennú je možné vytvoriť dvoma spôsobmi:

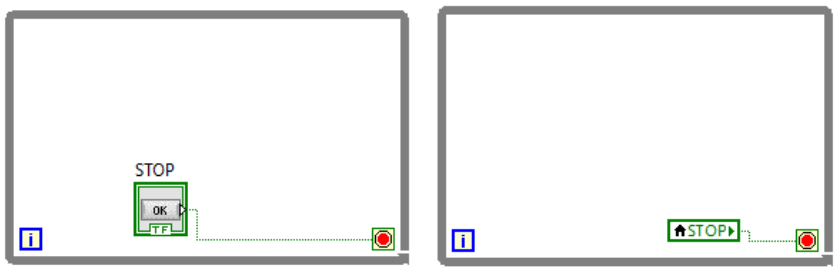
- Z *Tools Palette* vyberieme ponuku *Structures*, kde sa nachádza položka *Local Variable*  a vložíme prázdnu lokálnu premennú do diagramu. Kliknutím myši na ikonu vloženej lokálnej premennej vyberieme zo zoznamu, ktorému ovládaču alebo indikátoru sa má priradiť lokálna premenná. Prednastavená vlastnosť novej lokálnej premennej je možnosť zapisovať hodnoty do nej (*Write*)  . Ak hodnotu chceme čítať hodnoty z lokálnej premennej (*Read*)  , zmeníme túto vlastnosť pomocou kontextovej ponuky menu kliknutím pravým tlačidlom myši. Jednotlivé kroky sú zobrazené na obr. 2.12a.

- Z kontextovej ponuky uzla ovládača alebo indikátora po kliknutí pravým tlačidlom myši vytvoríme priamo lokálnu premennú tohto uzla (obr. 2.12b). Znova je potrebné určiť vlastnosť lokálnej premennej na *Read* alebo *Write*.

2 podrobnosti o cykloch sa dozvieme v kap. 2.3.1

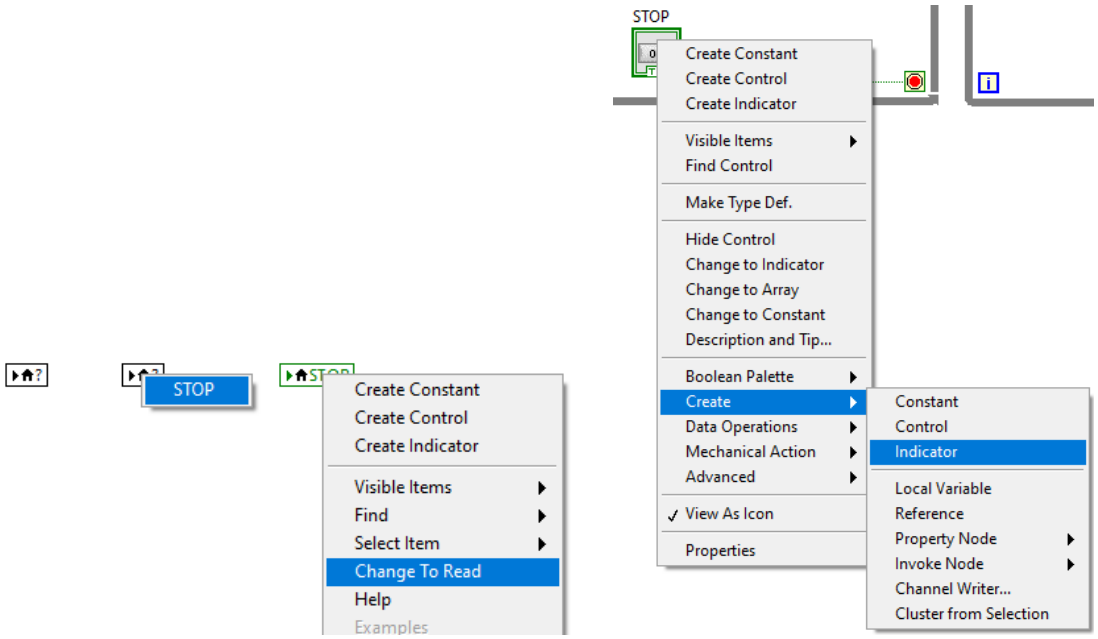


(a)



(b)

Obr. 2.11: Nesprávny (a) a správny (b) dizajn diagramu pre tlačidlom ovládané zastavenie podmienených opakovacích cyklov.



(a) (b)

Obr. 2.12: Vytvorenie lokálnej premennej z ponuky *Tools Palette* (a) a z kontextovej ponuky ovládača alebo indikátora (b).

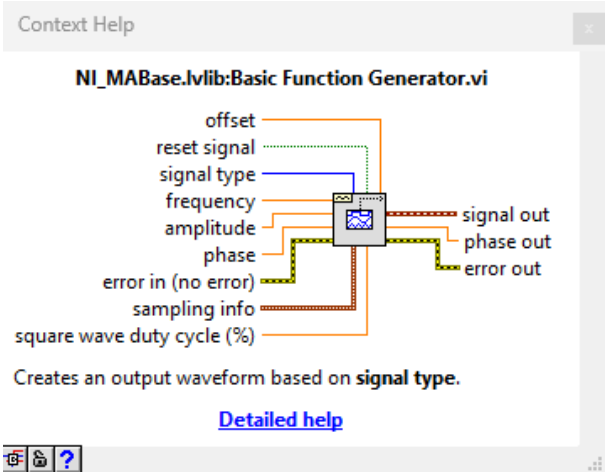
2.2.5 Pomocník a kompatibilita vydaní LabVIEW

Každý sa raz stretne pri vytváraní nového programu s otázkou ako funguje táto funkcia (alebo aká je syntax, v prípade uzlov v *LabVIEW* ide skôr o otázku aké vstupné a výstupné terminály má daný uzol, a ktoré z nich sú povinné na zapojenie), alebo aký nástroj použiť na vyriešenie určitého problému. Vtedy sa treba obrátiť na dokumentáciu softvérového balíka alebo diskusiu na internetových fórach. Vývojári *LabVIEW* ponúkajú aj rýchly a prehľadný nástroj na rýchlu pomoc pre každý objekt na *Blokovom diagrame* či *Prednom paneli*, ktorým je *Context Help*, okamžitý kontextový pomocník na obr. 2.13. Tento nástroj zobrazíme z ponuky *Help* ▸ *Show Context Help* alebo pomocou klávesovej skratky *Ctrl+H*. V novom okne sa bude zobrazovať kontextový pomocník objektu, nad ktorým práve držíme ukazovateľ myši.

Kliknutím na tlačidlo *Simple/Detailed Context Help* (v novších vydaniach *Show/Hide Optional Terminals and Full Path*), ktoré sa nachádza v ľavom dolnom rohu okna kontextového pomocníka, môžeme prepínať medzi jednoduchým a podrobným kontextovým pomocníkom. Kliknutím na tlačidlo *Lock Context Help* uzamkneme aktuálny obsah okna kontextového pomocníka. Keď je obsah uzamknutý, presunutím ukazovateľa myši na iný objekt sa obsah okna nezmení. Tretie tlačidlo *Help* umožní otvoriť kompletného pomocníka *LabVIEW*.

Pre začiatočníkov je k dispozícii aj zbierka vzorových príkladov VI, ktoré je možné otvoriť z ponuky *Help* ▸ *Find Examples* a využiť ich ako šablónu pre tvorbu vlastného VI.

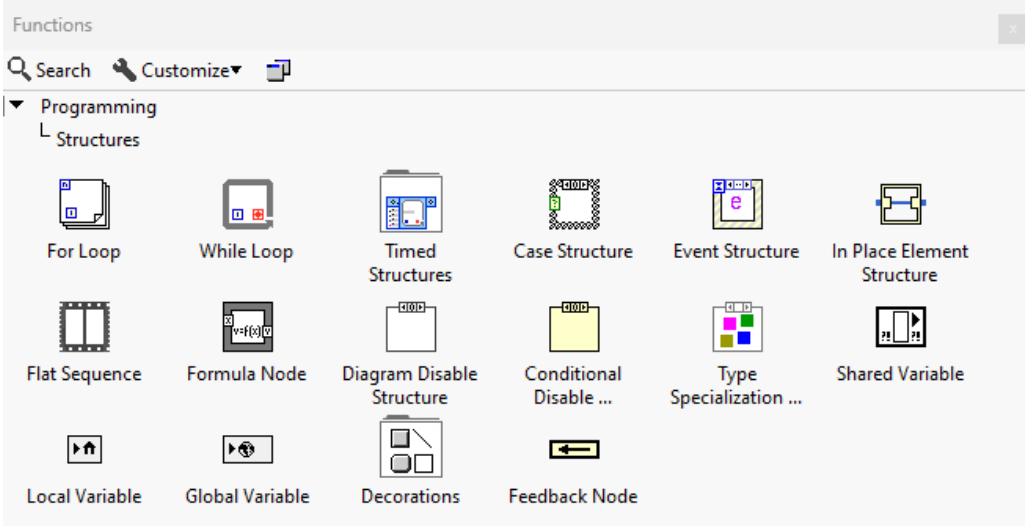
Jedným z dôležitých pojmov z pohľadu vývoja softvéru je kompatibilita vytvorených VI s inými vydaniami *LabVIEW*. VI vytvorené v rôznych starších vydaniach *LabVIEW* je možné otvoriť v najnovšom vydaní, možno s potrebnými úpravami kódu pri väčšom rozdieli medzi vydaniami. Opačne však kompatibilita nebýva zachovaná a VI vytvorené v najnovšom vydaní *LabVIEW* nie je možné používať a otvoriť v starších vydaniach. Existuje však možnosť v novšom vydaní uložiť VI vo formáte zodpovedajúcom staršiemu vydaniu z ponuky *File* ▸ *Save for Previous Version...*.



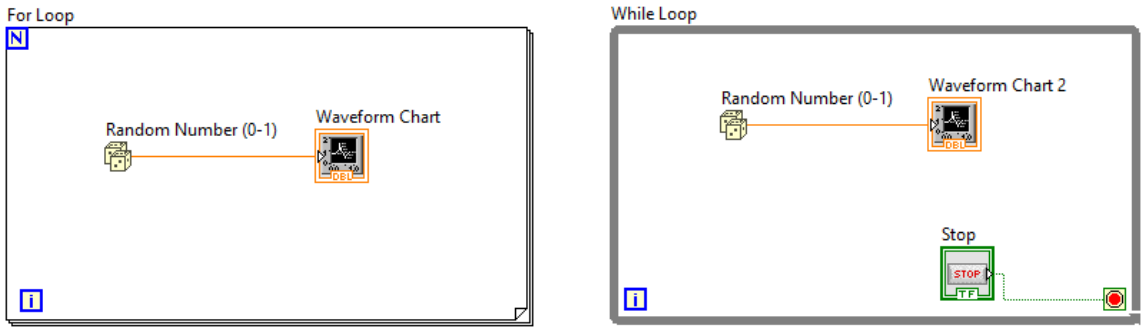
Obr. 2.13: Okno *Context Help*.

2.3 NÁSTROJE NA TVORBU PROGRAMOVÉHO KÓDU

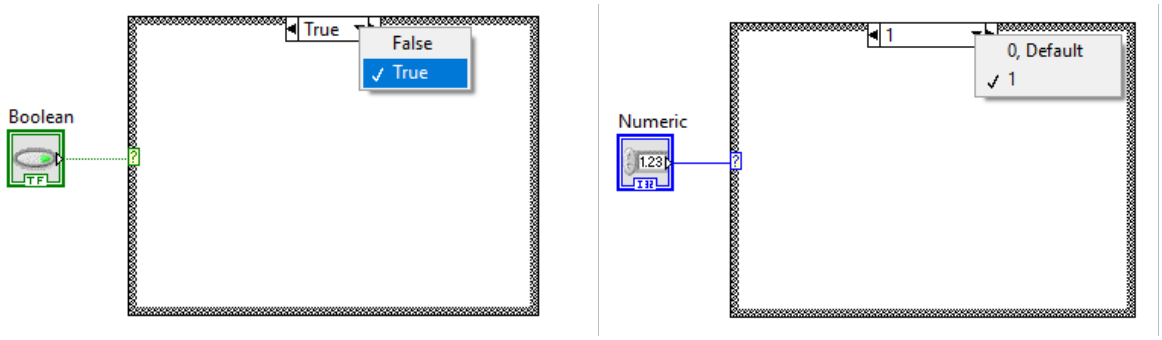
Štruktúry v *LabVIEW* predstavujú uzly diagramu, ktoré vykonávajú inštrukcie, ktoré sú v nich umiestnené. Údaje do štruktúry môžu prúdiť vodičmi alebo využitím lokálnych premenných. Štruktúry nájdeme v ponuke *Programming*▸*Structures* (obr. 2.14) vo *Functions Palette*. Štruktúry predstavujú opakovacie cykly, podmienené opakovacie cykly, podmienené vykonávanie kódu ako ich poznáme aj z konvenčných textových programovacích jazykov. V *LabVIEW* máme naviac k dispozícii sekvenčné a časované štruktúry, štruktúru udalostí a uzly na vloženie programového kódu. Štruktúry v *Blokovom diagrame* predstavujú rám do ktorého vkladáme kód, ktorý sa má vykonávať, preto pri vložení štruktúry do diagramu stlačeným tlačidlom myši zakreslíme jej rám dostatočne veľký. Rozmery rámu štruktúry môžeme dodatočne meniť, ťahaním za okraj rámu pri stlačení tlačidla myši. Problematike toku dát v, do a zo štruktúr sa budeme podrobnejšie venovať v závere tejto kapitoly.



Obr. 2.14: Ponuka štruktúr vo *Functions Palette*.



Obr. 2.15: Cyklus *For Loop* (a) a *While Loop* (b) s vloženým jednoduchým kódom.



Obr. 2.16: *Case structure* s logickým (a) a číselným (b) vstupom.

2.3.1 Opakovacie cykly

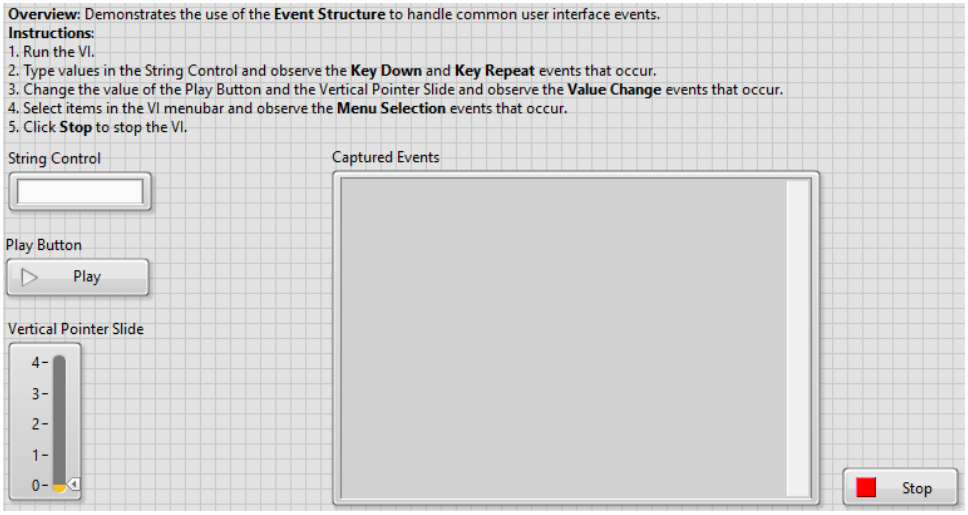
Opakovacie cykly implementované v *LabVIEW* sú:

- *For Loop* - opakovací cyklus s definovaným počtom opakovaní (obr. 2.15a) vykoná vložený kód stanovený počet krát. Hodnota v počítacom termináli (vstupný terminál), reprezentovaným písmenom *N*, udáva koľkokrát sa má vložený diagram zopakovať. Terminál iterácie (výstupný terminál), reprezentovaný písmenom *i*, obsahuje počet dokončených iterácií. Počet iterácií sa vždy začína na nule a počas prvej iterácie terminál iterácie vracia hodnotu 0.
- *While Loop* - podmienený cyklus s vykonávaním kódu až do splnenia logickej podmienky (obr. 2.15b) vykonáva vložený kód, kým nie je splnená logická podmienka pripojená do podmienkového terminálu. Predvolené správanie a vzhľad podmienkového terminálu je *Stop If True* (*Zastaviť, ak je pravda*) , cyklus vykonáva vložený kód, kým podmienkový terminál nedostane hodnotu *pravda* (*TRUE*). Toto správanie je možné zmeniť v kontextovej ponuke terminálu na *Continue If True* (*Pokračovať, ak je pravda*) , kedy cyklus vykonáva vložený kód, kým podmienkový terminál nedostane hodnotu *nepravda* (*FALSE*). Terminál iterácie (výstupný terminál), reprezentovaný písmenom *i*, obsahuje počet dokončených iterácií. Počet iterácií sa vždy začína na nule a počas prvej iterácie terminál iterácie vracia hodnotu 0.

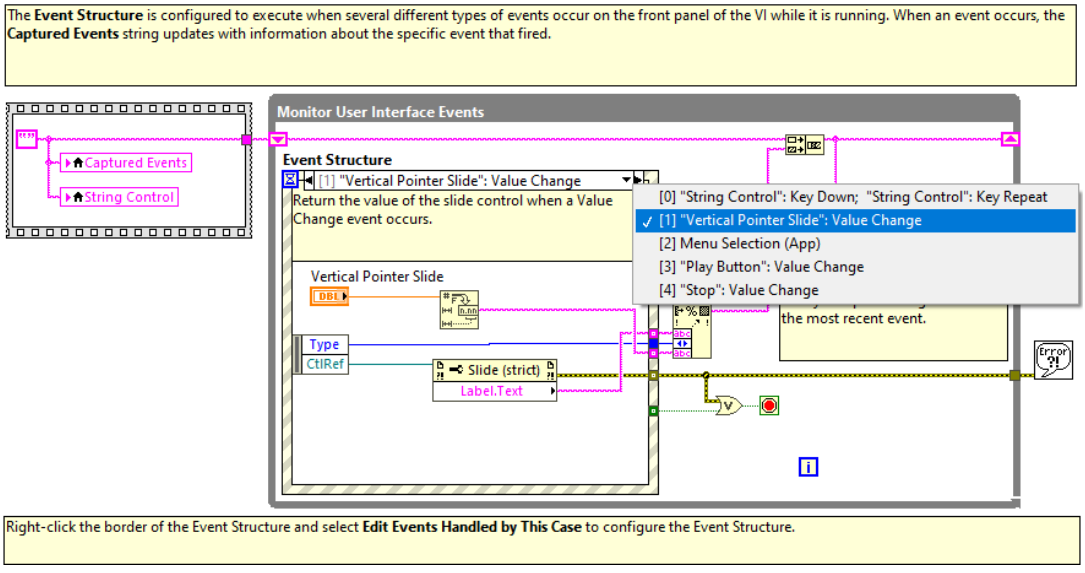
2.3.2 Podmienené vykonávanie kódu a sekvenčné štruktúry

Podmienené vykonávanie kódu je možné pomocou štruktúry *Case structure*, ktorá zodpovedá príkazom *If...then...else* a *Case* v konvenčných textových programovacích jazykoch. Podľa typu pripojeného údaju do vstupného terminálu *Case structure* sa mení aj jej správanie. Ak do vstupného terminálu pripojíme logickú hodnotu, *Case structure* vykonáva príkazy v ráme *TRUE* alebo v ráme *FALSE*, podobne ako príkaz *If...then...else* (obr. 2.16a). Ak do vstupného terminálu pripojíme iný typ údajov, napr. celé číslo alebo reťazec, je možné zadefinovať rám pre každú možnú hodnotu pripojenú do vstupného terminálu *Case structure*

(obr. 2.16b). Jednu z možností však musíme označiť ako prednastavenú (*Default*), ktorá sa vykoná vždy (môže to však byť prázdny rám bez kódu). Všetky nastavenia môžeme vykonať pomocou kontextovej ponuky štruktúry. Vykonávanie podmienených štruktúr môže byť spojené s využitím logických ovládačov, preto je potrebné aj správne nastavenie mechanickej akcie logických ovládačov, aby bola zabezpečená správna funkčnosť podmienených štruktúr a cyklov. Možnosti nastavenia mechanických akcií logických ovládačov sú krátko zhrnuté v kap. A.2.



(a)

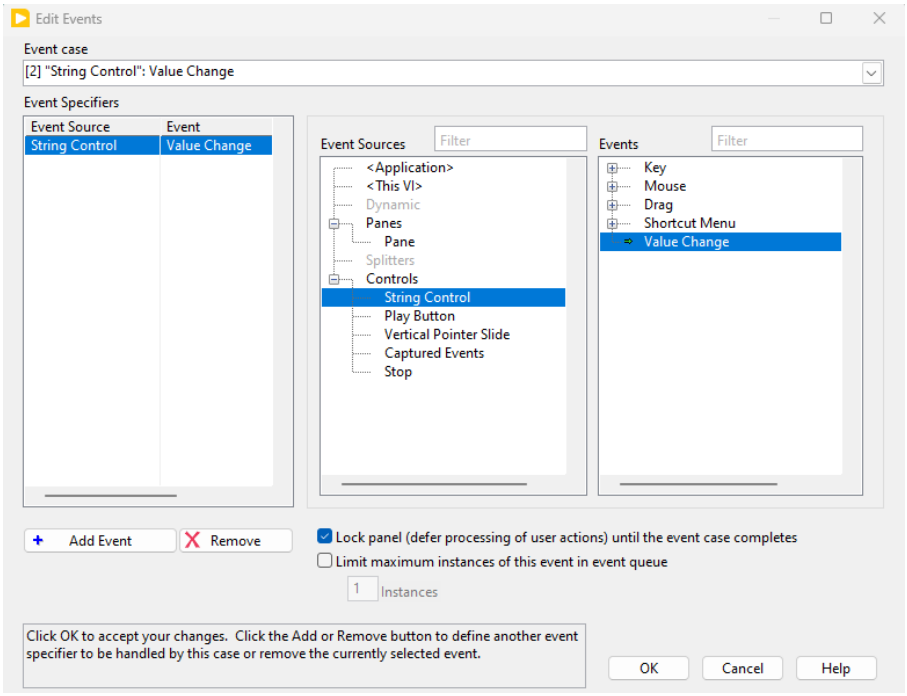


(b)

Obr. 2.17: *Handling Common User Interface Events.vi* opisujúce reakciu VI na interakciu užívateľa s objektami na *Prednom paneli* pomocou *Event Structure*.

Podmienené vykonávanie kódu je možné riadiť aj na základe rôznych udalostí počas behu VI s využitím *Event Structure*. Udalosťou dôležitou pre beh kódu môže byť napr. zmena hodnoty ovládača, výsledok výpočtovej operácie, chybové hlásenie funkcie, ale aj

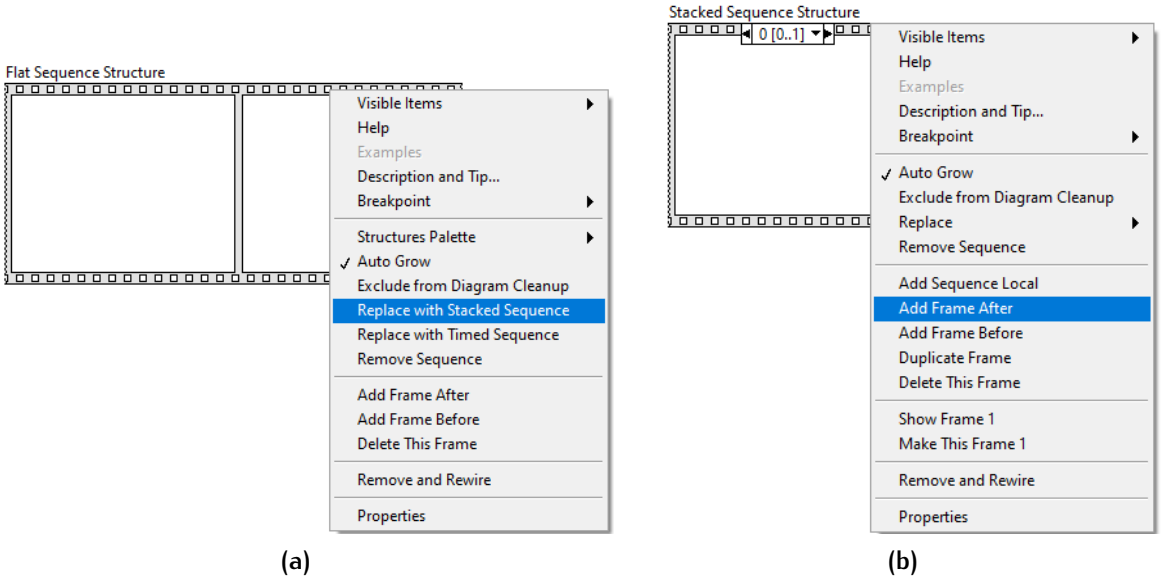
interakcia užívateľa s objektami na *Prednom paneli* (dokonca aj posunutie ukazovateľa myši nad objekt). Ako príklad uvádzame na obr. 2.17 vzorové VI *Handling Common User Interface Events.vi* z ponuky *Find Examples...* opisujúce reakciu VI na interakciu užívateľa s objektami na *Prednom paneli*. Po vložení *Event Structure* do diagramu si môžeme vybrať typy udalostí pomocou dialógového okna *Edit Events* na obr. 2.18. Tento typ štruktúry bol uvedený vo vydaní *LabVIEW* 6.1 v roku 2001, avšak správnym použitím nástrojov *LabVIEW* a rôznych typov štruktúr je možné tiež dizajnovať VI, ktoré bude reagovať na udalosti (tento spôsob predstavíme neskôr).



Obr. 2.18: Dialógového okno *Edit Events* pre správu udalostí *Event Structure*.

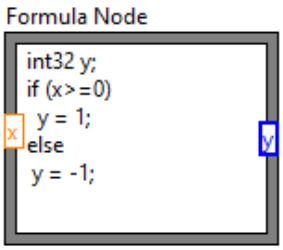
V textových programovacích jazykoch sa príkazy programu vykonávajú v poradí, v akom sú uvedené. V *LabVIEW* sa operácie uzla vykonajú, keď sú k dispozícii údaje na všetkých jeho vstupných termináloch. Niekedy je ťažké určiť presné poradie vykonávania, často sa však určité udalosti musia uskutočniť pred inými udalosťami. Ak potrebujeme kontrolovať poradie vykonávania kódu v diagrame tak aby nezáviselo na toku údajov, je možné použiť sekvenčnú štruktúru *Sequence Structure*. Sekvenčnú štruktúru vkladáme ako *Flat Sequence* s jedným rámom, počet za sebou nasledujúcich rámov môžeme nastaviť, vzhľadom pripomína okienka celuloidového filmu. Vo *Flat Sequence* je možné zabezpečiť tok údajov pomocou vodičov, ktoré prechádzajú cez hranice jednotlivých rámov. Ak sekvenčná štruktúra bude obsahovať väčšie množstvo rámov s rozsiahlym diagramom kódu, je možné zmeniť *Flat Sequence* (obr. 2.19a) na *Stacked Sequence* (obr. 2.19b), riadenie toku údajov však už nemusí byť triviálne.

Špeciálnym príkladom sekvenčnej štruktúry je *Formula Node* (obr. 2.20), ktorá umožňuje vloženie textového kódu na vykonanie matematických operácií, ktoré by ste mohli vytvoriť pomocou uzlov *Blokového diagramu*, ale vyžadovali by si v ňom veľa priestoru. Syntax



Obr. 2.19: *Sequence Structure* vo forme *Flat Sequence* (a) a *Stacked Sequence* (b).

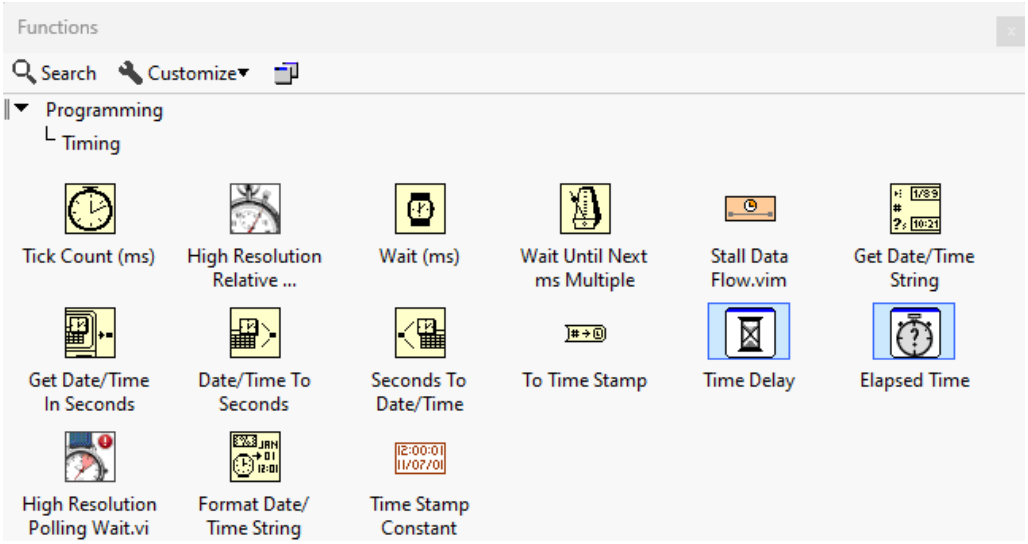
textového kódu je podobná jazyku *C*. Do vstupných terminálov pripájame hodnoty do nezávislých premenných a z výstupných terminálov získavame hodnoty závislých premenných textového kódu. Desatinný znak čísel spracúvaných vo *Formula Node* môže byť len bodka a riadky sú ukončené bodkočiarkou. Ak je k dispozícii na používanom počítači programovací balík *Mathworks MATLAB*, je možné vložiť do diagramu aj štruktúru *MATLAB script*, ktorá je dostupná z ponuky *Mathematics* ▸ *Scripts & Formulas* ▸ *Script Nodes* vo *Functions Palette*.



Obr. 2.20: *Formula Node* so vzorovým kódom.

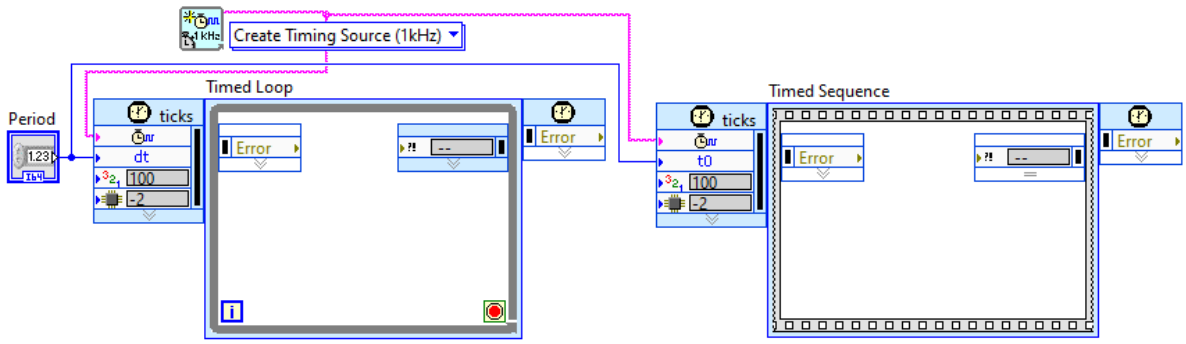
2.3.3 Časová kontrola behu VI

Časovanie je dôležité na kontrolu času vykonávania programu, ktorý dáva čas procesoru na dokončenie iných úloh. Zároveň môžeme definovať presné časové odstupy medzi jednotlivými operáciami. To môžeme jednoducho dosiahnuť vložením príkazu čakania *Wait (ms)* z ponuky *Programming* ▸ *Timing* vo *Functions Palette* (obr. 2.21) do rámov sekvenčnej štruktúry definujúci počet milisekúnd čakania. Ubehnutý čas môžeme získať ako rozdielovú hodnotu získanú z výstupu dvoch príkazov *Tick Count (ms)* umiestnených v dvoch miestach diagramu.



Obr. 2.21: Ponuka funkcií na prácu s časom vo *Functions Palette*.

Pre veľmi presné ovládanie časovania sú k dispozícii štruktúry *Timed Loop* a *Timed Sequence* v ponuke *Programming* ▸ *Structures* ▸ *Timed Structures* vo *Functions Palette* (obr. 2.22). Jedným zo vstupných terminálov je *Timing Source*, teda zdroj synchronizačného signálu s definovaným časovým rozlíšením, čo môže vyžadovať pripojenie špeciálneho hardvéru ak samotný počítač nedokáže poskytnúť takýto zdroj.



Obr. 2.22: *Timed Loop* a *Timed Sequence* s vytvoreným *Timing Source* a ovládačom časového intervalu *Period*.

2.3.4 Spätná väzba údajov v štruktúrach

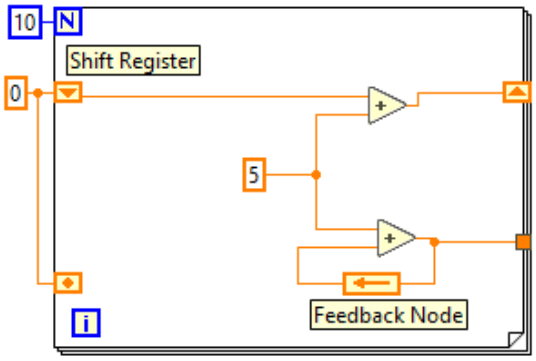
Často je potrebné najmä v iteratívnych cykloch ako sú *For Loop* alebo *While Loop* prenos údajov získaných v predchádzajúcej iterácií ako vstup do nasledujúcej iterácie. Túto úlohu je možné riešiť použitím lokálnych premenných, avšak *LabVIEW* ponúka efektívne riešenie použitím operácie *Shift Register* alebo *Feedback Node*:

- *Shift Register* pridáme z kontextovej ponuky pri kliknutí pravým tlačidlom myši na ľavú alebo pravú stranu rámu štruktúry.

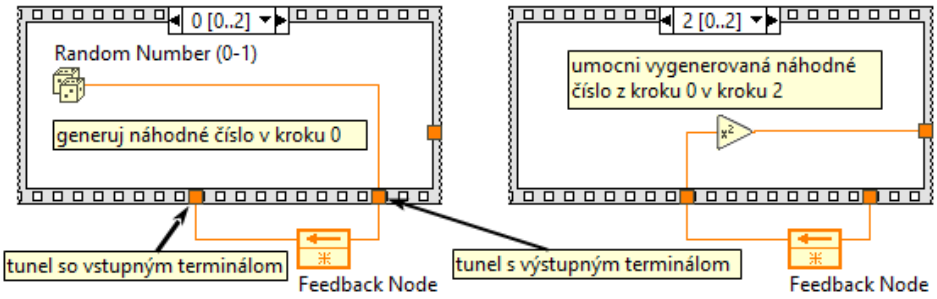
- *Feedback Node* vložíme na vhodné miesto rámu štruktúry z ponuky *Programming ▸Structures* vo *Functions Palette* a následným nastavením *Move Initializer One Loop Out* z kontextovej ponuky *Feedback Node*.

Jednoduchý cyklus, ktorý opakovane pripočítava hodnotu 5 k výsledku poslednej iterácie pomocou *Shift Register* aj *Feedback Node* je zobrazený na obr. 2.23. Dôležitá je v tomto prípade inicializácia počiatočnej hodnoty vstupujúcej do cyklu v prvej iterácii (v tomto príklade je to konštanta s hodnotou 0).

Feedback Node je však možné použiť aj pre prácu v iných typoch štruktúr, ako je napr. *Stacked Sequence*, ak je potrebné načítať údaj vygenerovaný v jednom ráme a použiť v jednom alebo viacerých nasledujúcich rámoch *Stacked Sequence*. Keďže údaj vygenerovaný v ráme môžeme viesť von z rámu štruktúry cez tzv. tunel (*angl. tunnel*) s výstupným terminálom, nemôžeme tento tunel v nasledujúcich rámoch použiť ako vstupný terminál. Je potrebné vytvoriť nový tunel so vstupným terminálom. Vytvorenie tunela je jednoduché, akonáhle generujeme vodič z výstupu ovládača alebo funkcie vo vnútri rámu a uložíme jeho koniec na rám štruktúry, automaticky sa vytvorí tunel. Na výstupný terminál tunela pripojíme vodičom vstupný terminál *Feedback Node* a z jej výstupného terminálu vedieme vodič na rám štruktúry, kde sa automaticky vytvorí tunel so vstupným terminálom ako je zobrazené na obr. 2.24. Tunel je možné umiestniť na akomkoľvek mieste rámu štruktúry.



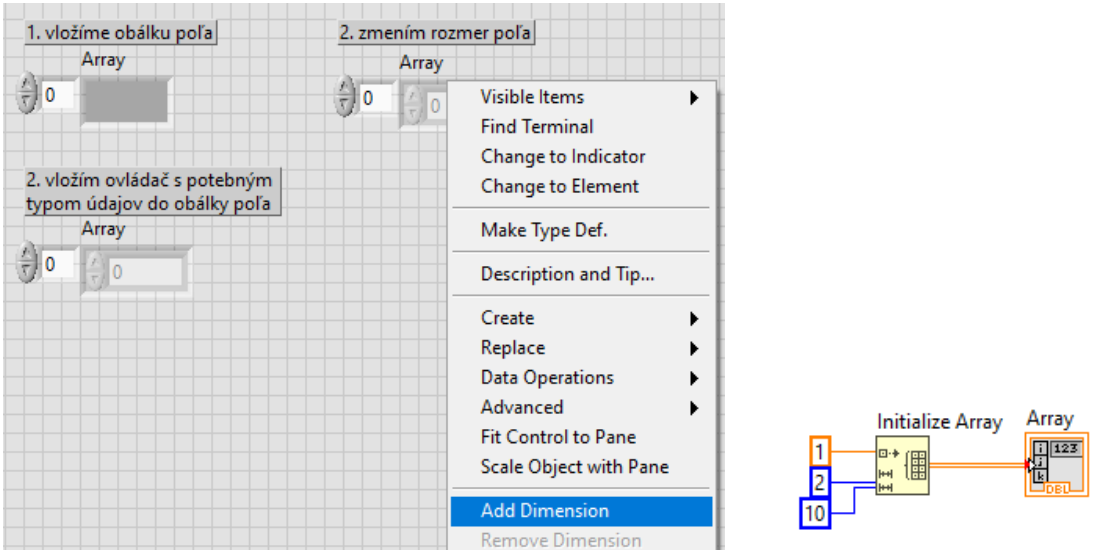
Obr. 2.23: Jednoduchý cyklus, ktorý opakovane pripočítava hodnotu 5 k výsledku poslednej iterácie pomocou *Shift Register* aj *Feedback Node*.



Obr. 2.24: Práca s *Feedback Node* v *Stacked Sequence*.

2.3.5 Polia a vlastnosti opakovacích cyklov

Polia predstavujú sadu údajov rovnakého typu a môžu byť jednorozmerné, dvojrozmerné alebo viacrozmerné. Polia sa používajú najmä vtedy, keď pri práci s opakujúcimi sa operáciami chceme postupne zaznamenať výsledky výpočtov do jedného objektu, premennej. Ich previazanie s cyklami prináša v *LabVIEW* špeciálne nastavenia cyklov určujúce, ako sa budú výstupy opakujúceho sa kódu v cykle zapisovať napr. do indikátorov. Ak potrebujeme vytvoriť ovládač alebo indikátor poľa na *Prednom paneli*, musíme najprv vložiť z ponuky *Data Containers* v *Controls Palette* tzv. obálku poľa *Array*. Do nej následne vložíme ovládač alebo indikátor typu údajov, ktoré budeme používať, napr. *Numeric Control* z ponuky *Numeric* v *Controls Palette*. Takto je vytvorený ovládač jednorozmerného poľa s jedným indexovacím políčkom vľavo od indikátora samotného. Na zväčšenie rozmeru použijeme z kontextovej ponuky poľa položku *Add dimension* (obr. 2.25a). V *Blokovom diagrame* je možné definovať presný rozmer poľa a zapísané počiatočné hodnoty indikátora poľa pomocou funkcie *Initialize Array* z ponuky *Programming* ▸*Array* vo *Functions Palette* (obr. 2.25b).

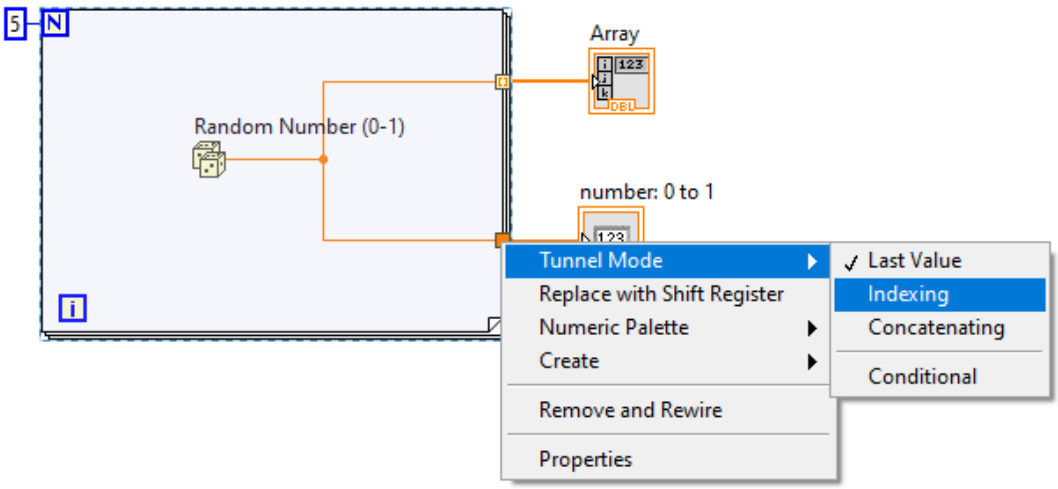


(a) (b)

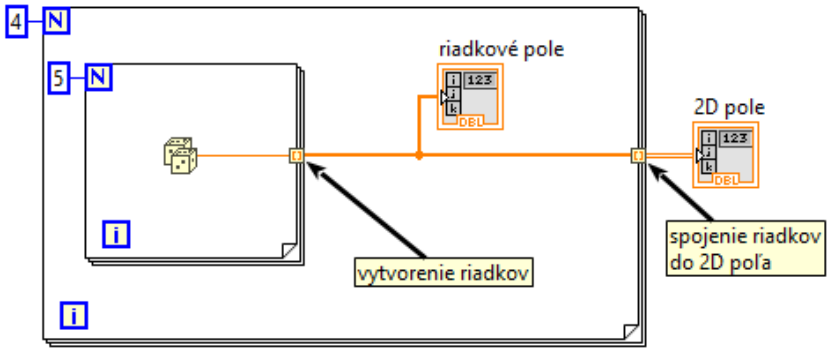
Obr. 2.25: Postup vytvorenia ovládača viacrozmerného poľa na *Prednom paneli* (a) a inicializácia indikátora poľa s presným rozmerom s počiatočnou hodnotou v *Blokovom diagrame* pomocou funkcie *Initialize Array* (b).

Ak pripojíme pole k vstupnému terminálu tunela cyklu *For Loop* alebo *While Loop*, môžeme čítať a spracovávať postupne všetky prvky v tomto poli zapnutím automatického indexovania[3]. Ak automaticky indexujeme výstupný terminál tunela cyklu, do pripojeného poľa zapíšeme nový prvok z každej iterácie cyklu. Vodič od výstupného terminálu tunela

3 Ak povolíme automatické indexovanie na poli zapojenom do vstupnej svorky slučky *For Loop*, počítadlo cyklu sa automaticky nastaví na veľkosť poľa (ak povolíme automatické indexovanie pre viac ako jeden tunel alebo ak zapojíme terminál počítania, počtom opakovaní sa stane menšia z možností).



(a)



(b)

Obr. 2.26: Možnosti tunelovania vodičov z cyklu *For Loop* pre postupný zápis prvkov poľa alebo len jedinej hodnoty (a) a pre vytvorenie dvojrozmerného poľa (b).

k indikátoru poľa sa pri zmene na zápis poľa na hranici cyklu zmení na hrubšie a výstupný tunel sa zobrazuje s hranatými zátvorkami (obr. 2.26a). Na vytvorenie dvojrozmerného poľa môžeme použiť dva cykly *For Loop*, jeden vložený v druhom. Vonkajší cyklus vytvorí riadkové prvky poľa a vnútorný cyklus vytvorí stĺpcové prvky poľa ako na obr. 2.26b.

Typ tunela môžeme zmeniť kliknutím pravým tlačidlom myši na tunel a výberom položky *Disable Indexing* alebo *Enable Indexing* z kontextovej ponuky pre staršie vydania *LabVIEW*, alebo kliknutím na jednu z možností *Last Value* alebo *Indexing* z položky *Tunnel Mode* pre novšie vydania *LabVIEW*. Automatické indexovanie zakážeme napríklad vtedy, ak potrebujeme iba poslednú hodnotu odovzdanú do tunela v predchádzajúcom príklade bez vytvorenia poľa. Pre cyklus *For Loop* je automatické indexovanie predvolene zapnuté pre každé pripojené pole, pre *While Loop* je automatické indexovania predvolene vypnuté.

V ponuke *Programming* ▸*Array* vo *Functions Palette* nájdeme množstvo funkcií na manipuláciu polí (ich popis nájdeme v užívateľskej príručke *LabVIEW*), najbežnejšie sú:

- *Array Size* – vráti počet prvkov v každom rozmere poľa. Ak je pole n-rozmerné, výstupom je pole s n prvkami.

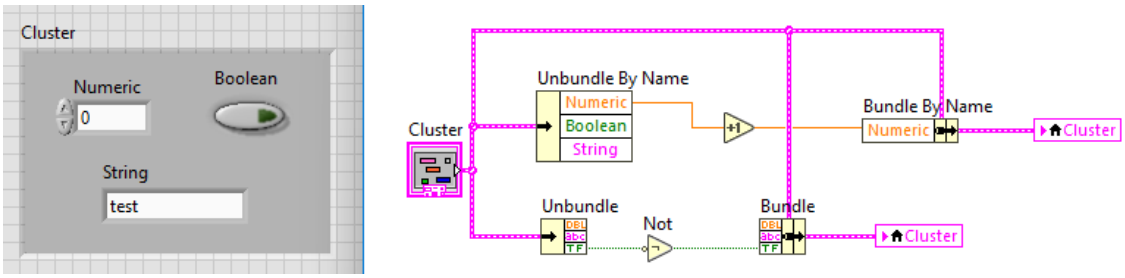
- *Build Array* – spojí viacero polí alebo pridá prvky do *n*-rozmerného poľa. Uzlu funkcie môžeme zväčšiť alebo zmenšiť počet vstupných terminálov, aby sa zvýšil počet prvkov výstupného poľa. Ak chceme spojiť vstupy do dlhšieho poľa rovnakého rozmeru, klikneme pravým tlačidlom myši na uzol funkcie a z kontextovej ponuky vyberieme položku *Concatenate Inputs*. Ak spájame dve jednorozmerné polia do dvojrozmerného poľa vypneme *Concatenate Inputs*.
- *Array Subset* – vráti časť poľa začínajúcu na určenom indexe a obsahujúcu daný počet prvkov.
- *Index Array* – vráti prvok poľa na určenom indexe. Funkciu môžete použiť aj na extrakciu riadku alebo stĺpca dvojrozmerného poľa a vytvoriť tak podmnožinu pôvodného poľa. Na tento účel pripojíme dvojrozmerné pole na vstupný terminál funkcie a k dispozícii budú dve indexové terminály, označujúce riadok a stĺpec.

2.3.6 Klastre

Výhoda polí je prenos veľkého množstva údajov v jednom objekte, avšak musia to byť údaje rovnakého typu. Pre zjednodušenie programového kódu, v prípade *LabVIEW* pre zjednodušenie *Blokového diagramu*, je možné využiť iný typ objektu typu klaster (*angl. cluster*), ktorý spája údaje rôzneho typu do jedného objektu. Všetky objekty v klastri musia byť len ovládače alebo indikátory. Klaster v *LabVIEW* je objekt analogický napr. *struct* v jazyku *C*. Výhoda pre tvorbu *Blokového diagramu* je aj zjednodušenie prenosu údajov jedným vodičom prípadne jednou lokálnou premennou (vodič prepájajúci klastre si môžeme predstaviť ako kábel s viacerými vodičmi ). Špeciálnym klastrom používaným v *LabVIEW* je chybové hlásenie funkcií, ktoré obsahuje tri objekty: logický *status*, celočíselný *code* a textový *source* identifikujúce typ problému v behu funkcie.

Ak potrebujeme vytvoriť klaster na *Prednom paneli*, musíme najprv vložiť z ponuky *Data Containers* v *Controls Palette* tzv. obálku klastra *Cluster*. Do nej následne vložíme ovládače alebo indikátory typu údajov, ktoré budeme používať. Vytvorený klaster sa stane indikátorom ak prvý vložený objekt je indikátor, alebo ovládačom ak prvý vložený objekt je ovládač. Ak je potrebné v diagrame VI pracovať s jednotlivým objektami v klastri, môžeme použiť funkciu *Unbundle* alebo *Unbundle by Name* z ponuky *Programming* ▸ *Cluster, Class & Variant* vo *Functions Palette* alebo z kontextovej ponuky výstupnej svorky klastra. Na výstupných termináloch *Unbundle* alebo *Unbundle by Name* sa objavia hodnoty objektov v klastri identifikované podľa typu údajov alebo podľa menovky objektu. Po vykonaní operácií s objektami v klastri ich znova zviažeme do klastra pomocou funkcií *Bundle* alebo *Bundle by Name*. Pre prácu s nimi je nutné pripojiť do vstupného terminálu *input cluster* vodič z výstupného terminálu klastra pre identifikáciu jeho obsahu. Použitie oboch variantov uvedených funkcií je zobrazené na obr. 2.27.

Pri práci s meracími zariadeniami je využitie klastrov vítané, keďže je takto možné spojiť do jedného klastra sadu ovládačov, ktoré predstavujú sadu parametrov meracieho zariadenia, napr. nastavenie vstupného filtra na multimetri alebo fázovo-citlivom zosilňovači.



Obr. 2.27: Klaster obsahujúci tri indikátory (*Numeric*, *Boolean*, *String*) a spôsob práce s hodnotami jednotlivých indikátorov pomocou funkcií *Bundle*, *Bundle by Name*, *Unbundle* a *Unbundle by Name*.

2.4 ZOBRAZOVANIE A ZÁZNAM ÚDAJOV

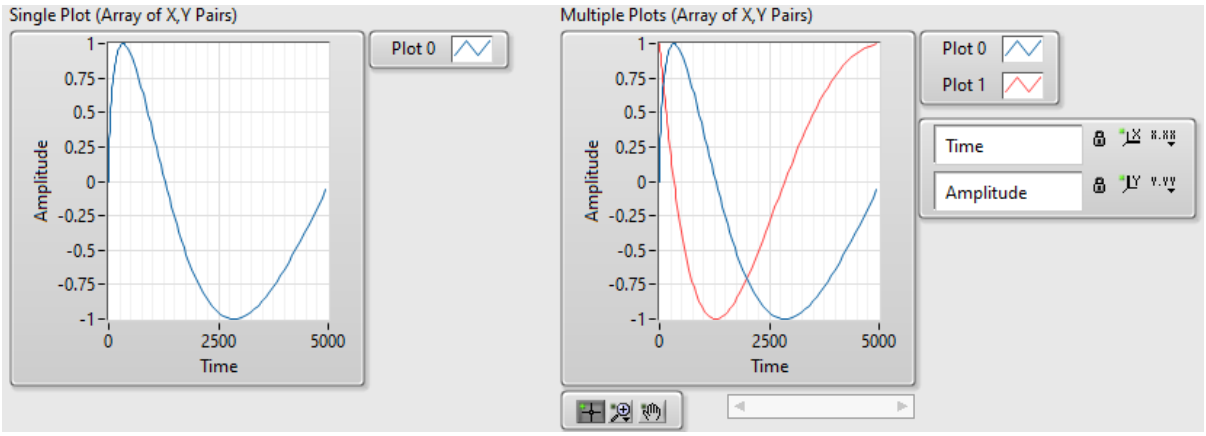
Keďže od pripraveného projektu alebo VI očakávame zvyčajne ovládanie experimentálnej zostavy a zber nameraných údajov, je dôležité mať možnosť zobraziť a uložiť vo vhodnej forme nazbierané údaje. Zobrazenie údajov v tabuľke poľa nemusí mať priamočiaru výpovednú hodnotu a poznanie napr. časového priebehu zbieraných údajov vo forme grafu je veľkou výhodou, ktorú poskytuje *Predný panel* VI vo forme indikátorov typu graf. Následné spracovanie zbieraných údajov je možné po ich uložení na pevný disk počítača vo vybranom formáte súboru (textovom, tabuľkovom alebo binárnom).

2.4.1 Grafy

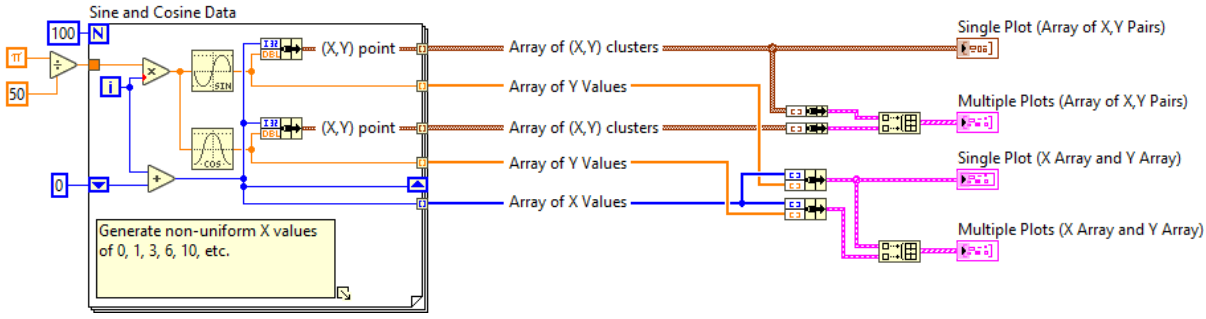
Controls Palette obsahuje v ponuke *Graph* viacero indikátorov typu graf pre grafické znázornenie získaných údajov. Medzi najbežnejšie používané patria:

- *Waveform Chart* - zobrazuje časový priebeh jedného alebo viacerých údajov. Vstupom *Waveform Chart* je jedno pole hodnôt y a interpretuje údaje ako body na grafe s indexom x zvyšujúcim sa o jednotku počnúc $x = 0$. Vstupom môže byť aj klaster počiatočnej hodnoty x, kroku v x hodnotách a pole údajov y.
- *XY Graph* - zobrazuje priebeh jednej alebo viacerých sád údajov x a y. Vstupom *XY Graph* je klaster dvoch polí x a y, alebo pole klastrov obsahujúcich dvojicu hodnôt (x,y) ako je zobrazené na obr. 2.28.

Minimálne a maximálne hodnoty osi x alebo y na grafe môžeme zmeniť dvojitým kliknutím na hodnotu pomocou nástroja na označovanie a zadaním novej hodnoty. Podobne môžeme zmeniť aj označenie osí. Môžeme tiež kliknúť pravým tlačidlom myši na legendu grafu a zmeniť štýl, tvar a farbu stopy, ktorá je zobrazená na grafe pomocou *Plot Legend*. Aj počas behu VI máme k dispozícii viacero nástrojov na manipuláciu indikátora grafu: *Scale Legend* (škálovanie a formát hodnôt na osiach grafu), *Graph Palette* (približovanie, vzďaľovanie a posun plochy grafu), *Cursor Legend* (zobrazí značku na súradnici definovaného bodu)

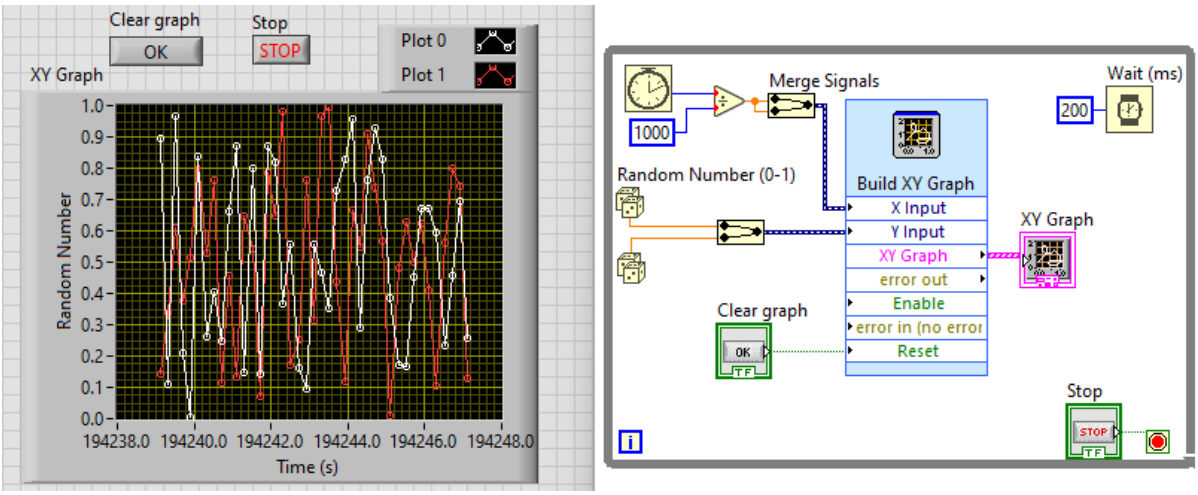


(a)



(b)

Obr. 2.28: Využitie indikátora *XY Graph* v *XY Graph Data Types.vi* z ponuky príkladov, pre graf vpravo sú zobrazené *Plot Legend*, *Scale Legend*, *Graph Palette* a *X Scrollbar*.



Obr. 2.29: Príklad využitia *Express XY Graph*.

a *X Scrollbar* (pomocou rolovacieho pruhu môžeme zobraziť údaje, ktoré graf momentálne nezobrazuje) ako na obr. 2.28.

Pre zjednodušenie práce s grafmi je k dispozícii aj vloženie indikátora *Express XY Graph*, ktorý využíva funkciu *Build XY Graph* na prípravu vhodného vstupu do indikátora *XY*

Graph. Vstupom funkcie *Build XY Graph* sú dve hodnoty *x* a *y* alebo dve polia *x* a *y* (polia sú automaticky konvertované pomocou funkcie *Convert to Dynamic Data* na tzv. dynamické údaje) pri zobrazení jednej závislosti na grafe. V prípade zobrazenia aspoň dvoch závislostí je potrebné spojiť *x*-ové a *y*-ové hodnoty do jedného vodiča dynamických údajov pomocou funkcie *Merge Signals* z ponuky *Express* ▸*Sig Manip* vo *Functions Palette*. Vo vzorovom VI na obr. 2.29 generujeme v čase dve náhodné čísla a zobrazujeme ich časový vývoj simultánne na indikátore *XY Graph*. Jedným zo vstupných terminálov funkcie *Build XY Graph* je *Reset*, kde pomocou logického ovládača je možné počas behu vymazať zobrazené grafy. Konfiguračné dialógové okno tejto expresnej funkcie má len jeden parameter nastavenia, *Clear data on each call*, ktorým umožníme premazať graf pri každom z opakovaných volaní tejto funkcie, napr. ak sú vstupom polia, nie jednotlivé hodnoty (*x*,*y*).

2.4.2 Súbory

Používanie softvérových prostriedkov operačného systému, ako je práca so súbormi je zabezpečené podobne ako v iných programovacích jazykoch. Najprv je potrebné otvoriť alebo vytvoriť daný prostriedok, potom ho použiť a nakoniec uzavrieť. Je dôležité poznamenať, že softvérový prostriedok sa po otvorení nemôže používať žiadnym iným programom. V *LabVIEW* sa so súbormi dá pracovať rôznymi spôsobmi. Existujú funkcie, ktoré umožňujú vykonať všetky tri kroky práce so súborom, ale aj individuálne kroky otvorenia, zápisu alebo čítania a uzatvorenia súboru, nájdeme ich v ponuke *Programming* ▸*File I/O* vo *Functions Palette*. Pre začiatočníkov sú vhodné:

- Štandardné funkcie *Read Delimited Spreadsheet* a *Write Delimited Spreadsheet* pre prácu s textovým súborom vo formáte American Standard Code for Information Interchange (ASCII) [15]. Pre zápis do súboru, pri štandardných nastaveniach a vstupných číselných hodnotách vo forme poľa, vytvorený súbor obsahuje stĺpce hodnôt oddelené znakom tabulátora. Formát zapísaných hodnôt je možné vhodne zmeniť z prednastaveného špecifikátora formátu *%.3f* vo vstupnom termináli *format*, pre ktorý sa zapisujú číselné hodnoty do textu s tromi číselnými znakmi za desatinným znakom, na vhodnejší formát podľa syntaxe v kap. A.1. Ak zmeníme oddeľovací znak z tabulátora na čiarku vo vstupnom termináli *delimiter* a súbor budeme ukladať s príponou *.csv*, získame tzv. *comma delimited file*, ktorý je možné otvoriť priamo v softvéri *Microsoft Excel* (je však nutné zabezpečiť aby desatinným znakom bola bodka). Na zapísanie popisnej hlavičky súboru, napr. názvy stĺpcov, je však potrebné vopred pripraviť textový reťazec, ktorý zapíšeme so súboru pred zápisom samotných hodnôt pomocou funkcie *Write Text File*.

- Expresné funkcie *Read From Measurement File* a *Write To Measurement File* pracujú podobne ako *Read Delimited Spreadsheet* a *Write Delimited Spreadsheet*, avšak nastavenia výstupných súborov a ich formátu vykonávame pomocou konfiguračného dialógového okna. Preddefinovaný je špeciálny textový formát súboru LabVIEW

Measurement File (LVM), pričom je možnosť nastavenia zápisu priamo do formátu *Microsoft Excel* (*.xlsx*).

Príklad použitia funkcií *Write Delimited Spreadsheet* a *Write To Measurement File* je zobrazený na obr. 2.30a, pričom konfiguračné dialógové okno pre zápis LVM súboru poskytuje množstvo nastavení formátu súboru (obr. 2.30a). Pre výber súboru na zápis alebo čítanie počas behu VI je možné použiť funkciu *File Dialog* z ponuky *Programming*▸*File I/O*▸*Advanced File Functions* alebo *Express*▸*Input* vo *Functions Palette*.

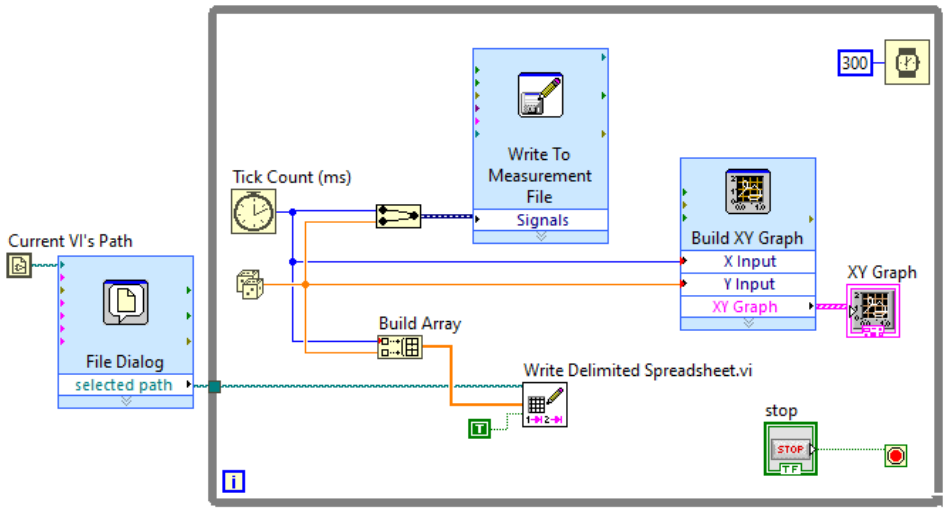
Práca s textovými súbormi však nemusí byť dostatočne rýchla v prípade využitia hardvéru na rýchly zber údajov, v tomto prípade sa na záznam údajov odporúča použitie binárne kódovaných súborov pomocou funkcií *Write to Binary File* a *Read from Binary File* z ponuky *Programming*▸*File I/O* vo *Functions Palette*. Binárne súbory vyžadujú menej bitov na reprezentáciu čísel ako ASCII znaky a umožňujú náhodný prístup k súboru. Tieto súbory sa používajú v systémoch vysokorýchlostného zberu údajov na ukladanie informácií (napr. z analógového vstupného kanála) do súboru. Neexistuje priamy spôsob, ako prečítať binárny súbor, preto je dôležité vedieť, ako bol binárny súbor zapísaný, aké sú jeho dátové typy a kódovanie [16]. *LabVIEW* ponúka aj vlastný formát štruktúrovaných binárnych súborov Technical Data Management Streaming (TDMS) [17] a potrebné funkcie na prácu s nimi.

2.5 POKROČILÉ NÁSTROJE NA TVORBU PROGRAMOVÉHO KÓDU

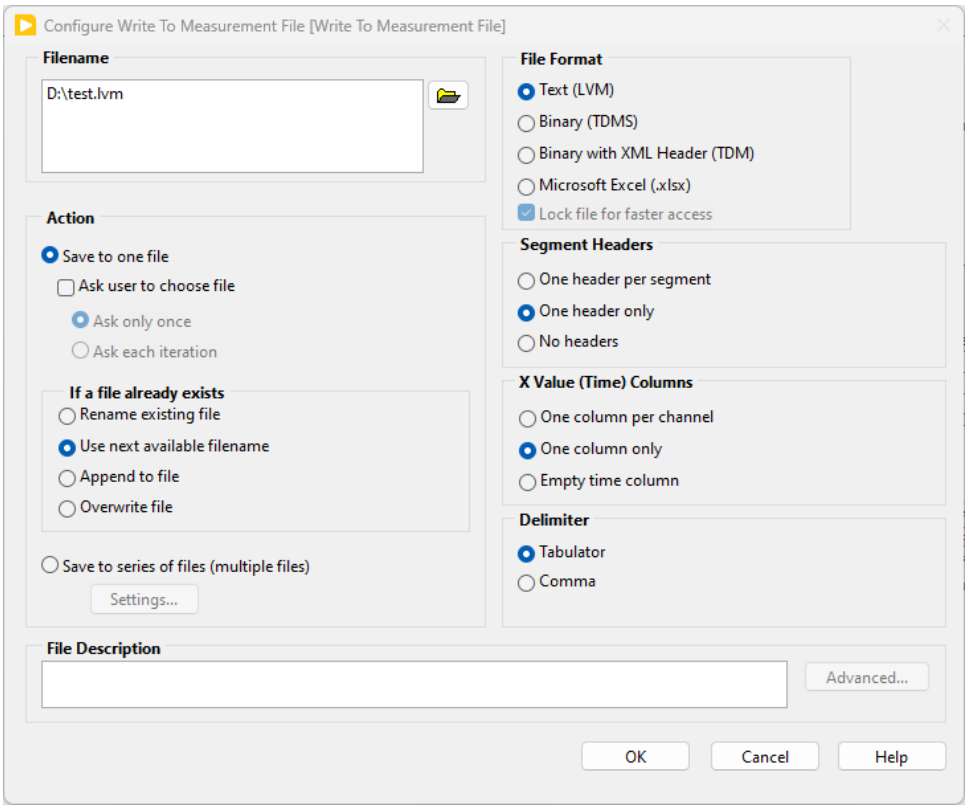
2.5.1 Property Node

Pri vytváraní komplexných VI, pri ktorých je potrebná interakcia s užívateľom, je často potrebné ovládať vlastnosti objektov na *Prednom paneli* na základe udalostí bežiaceho programového kódu. Príkladom môže byť napr. zákaz ovládania tlačidla na *Prednom paneli* v určitej časti vykonávaného programového kódu z dôvodu ochrany prístrojového zariadenia alebo samotného experimentu. Takéto ovládanie zabezpečuje použitie *Property Node*, uzla odkazov na vlastnosti objektov. Pomocou *Property Node* môžeme nastavovať alebo čítať také vlastnosti objektov, akými sú farba, formátovanie a presnosť údajov, viditeľnosť, popisný text, veľkosť a umiestnenie na *Prednom paneli*, atď. Pri tvorbe VI je možné všetky vlastnosti objektov na *Prednom paneli* prednastaviť z kontextovej ponuky objektu *Properties* (*Vlastnosti*) v konfiguračnom dialógovom okne na obr. 2.31. Všetky tieto nastavenia je však možné zmeniť pomocou *Property Node* v *Blokovom diagrame*. Pre vytvorenie uzla *Property Node* použijeme kontextovú ponuku daného objektu a nastavenie *Create*▸*Property node*, kde vyberieme vhodnú vlastnosť objektu. Po vložení potrebného uzla *Property Node* do Blokového diagramu zmeníme jeho termináli na vstupné alebo výstupné podľa potreby programového kódu.

Na obr. 2.32 bol vytvorený tzv. implicitne viazaný uzol *Property Node* z kontextovej ponuky tlačidla *OK Button* pre vlastnosť *Disabled*, ktorá umožňuje meniť funkčnosť tlačidla.

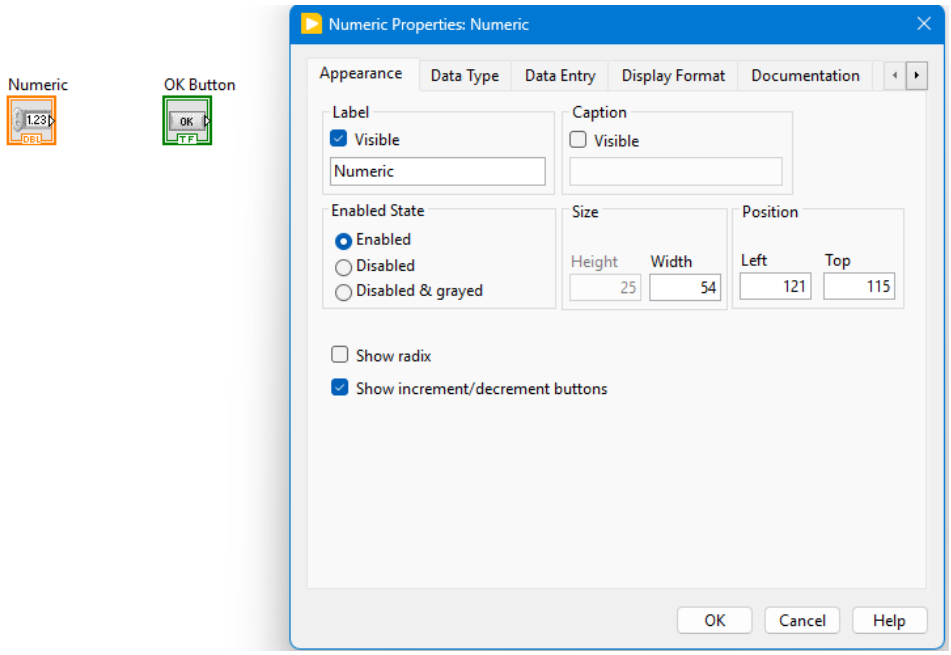


(a)



(b)

Obr. 2.30: (a) Využitie *Write Delimited Spreadsheet* a *Write To Measurement File* na zápis generovanej dvojice údajov (čas, náhodné číslo) do textového súboru alebo formátovaného LVM súboru. Pomocou funkcie *File Dialog* z ponuky *Programming*▸*File I/O*▸*Advanced File Functions* alebo *Express*▸*Input* vo *Functions Palette* vyvoláme systémové dialógové okno na určenie názvu a umiestnenia súboru pre funkciu *Write Delimited Spreadsheet*. (b) Konfiguračné dialógové okno funkcie *Write To Measurement File*.

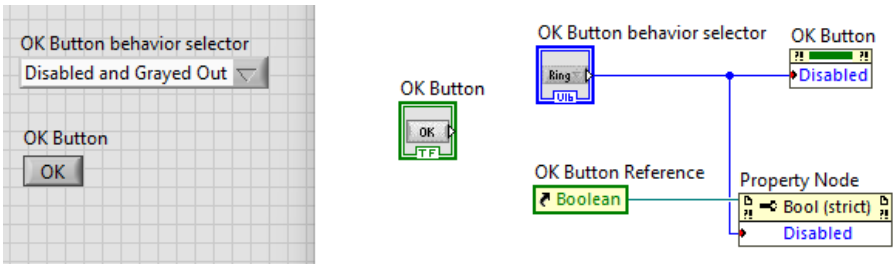


Obr. 2.31: Príklad nastavenia vlastností numerického ovládača pomocou konfiguračného dialógového okna.

Pre zmenu funkčnosti tlačidla bol vytvorený aj ovládač typu *Menu Ring*, ktorý umožňuje zmeniť stav tlačidla na *Enabled* (funkčné), *Disabled* (nefunkčné) a *Disabled and Grayed Out* (nefunkčné, s vizuálnou zmenou). Zmenu funkčnosti v závislosti od vykonávaného programového kódu by sme vykonali zápisom konštanty do uvedeného uzla *Property Node* spojeného s použitím *Case structure* na rozhodnutie akú funkčnosť si má tlačidlo zachovať v nasledujúcom behu VI.

Ak je potrebné upozorniť na niektorý z prvkov *Predného panelu* počas behu VI, je možné využiť vlastnosť *Blinking* (blikanie, periodická zmena farby) alebo *Key Focus* (zvýraznenie objektu). Pre zobrazenie kontextuálneho pomocníka ovládača na *Prednom paneli* pri pohybe ukazovateľa myši nad ovládačom je možné zapísať a zobraziť zodpovedajúci text pomocou vlastnosti *TipStrip*. Na preskúmanie ďalších možností odporúčame náhľad do dokumentácie *LabVIEW*.

Do *Blokového diagramu* je tiež možné vložiť prázdny uzol *Property Node* z ponuky *Programming*▸*Application control* vo *Functions Palette* a objekt, s ktorým budeme pracovať, priradiť pomocou kontrolného odkazu (*Reference*) ako na obr. 2.32, ide o tzv. explicitne viazaný uzol *Property Node*. Kontrolný odkaz objektu vytvoríme pomocou kontextovej ponuky objektu *Create*▸*Reference*. Takýmto spôsobom je možné ovládať nie len ako budeme meniť vlastnosti objektu, ale aj ktorý objekt bude meniť svoje vlastnosti zmenou kontrolného odkazu na vstupnej svorke *reference* uzla *Property Node*.



Obr. 2.32: Pre tlačidlo *OK Button* je vytvorená možnosť zmeny vlastnosti *Disabled*, ktorá umožňuje meniť funkčnosť tlačidla. Implicitný uzol *Property Node* je vytvorený priamo z kontextovej ponuky objektu alebo je využitý kontrolný odkaz na objekt tlačidla (*Reference*) pre vytvorenie explicitne viazaného uzla *Property Node*.

2.5.2 Invoke Node

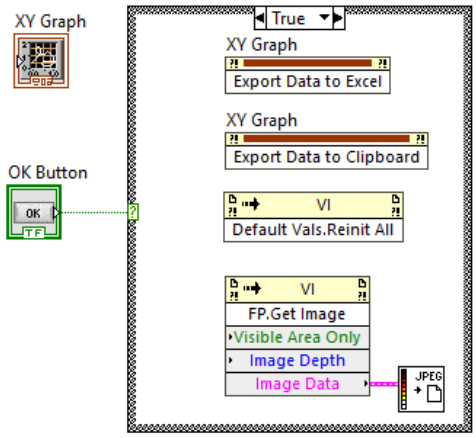
Pri potrebe vykonávania komplexných manipulácií objektov *Predného panela*, ovládania samotného VI z programového kódu alebo vykonávania tzv. metód, operácií, ktoré je možné vykonávať na rôznych aplikáciách, *LabVIEW* poskytuje možnosť vloženie uzla *Invoke Node*. Táto funkcia vyvoláva metódu alebo akciu na odkaz v uzle.

Pre vytvorenie uzla *Invoke Node* pre konkrétny objekt použijeme kontextovú ponuku daného objektu a nastavenie *Create*▸*Invoked node*, kde vyberieme vhodnú metódu alebo akciu s objektom. Prázdny uzol *Invoke Node* je tiež možné vložiť do diagramu z ponuky *Programming*▸*Application control* vo *Functions Palette* a priradiť mu odkaz na objekt alebo triedu aplikácie (*Class*), s ktorým budeme pracovať, a vybrať metódu (*Method*) zo zoznamu všetkých dostupných metód pre aktuálny odkaz.

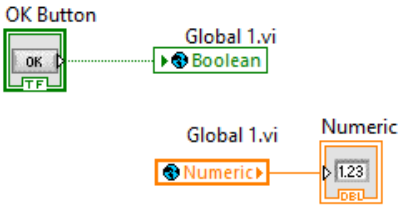
Ako jednoduchý príklad práce s *Invoke Node* môžeme uviesť export údajov z indikátora grafu do schránky operačného systému alebo konkrétneho softvéru (metóda *Export Data to Clipboard* alebo *Export Data to Excel* pre objekt grafu), inicializáciu všetkých ovládačov vo VI na prednastavenú hodnotu (metóda *Default Vals.Reinit All*), alebo vytvorenie snímky *Predného panelu* (metóda *FP.GetImage*) ako sú zobrazené na obr. 2.33.

2.5.3 Globálne premenné

Pri potrebe zdieľať hodnoty zapísané v ovládačoch alebo indikátoroch na *Prednom paneli* viacerých bežiacich VI je k dispozícii špeciálny druh *SubVI*, ktorý sa nazýva globálna premenná (*Global Variable*). Takéto *SubVI* obsahuje len vložené objekty zodpovedajúce ovládačom alebo indikátorom rovnakého typu, ktorých zapísané hodnoty chceme zdieľať vo viacerých simultánne bežiacich VI. Do každého VI vložíme vytvorené *SubVI* globálnej premennej a zapisujeme do jeho vstupných terminálov alebo čítame z jeho výstupných terminálov potrebné hodnoty ako na obr. 2.34. Globálnu premennú vytvoríme vložením objektu *Global Variable* z ponuky *Structures* vo *Functions Palette* na *Blokový diagram* vybraného VI. Následne otvoríme *Predný panel* tohto *SubVI*, vložíme potrebné typy a počet



Obr. 2.33: Použitie *Invoke Node* na export údajov z indikátora grafu do schránky operačného systému a do listu *Microsoft Excel*, inicializáciu všetkých ovládačov na prednastavenú hodnotu a vytvorenie snímky *Predného panelu* s 8-bitovým farebným rozlíšením a následným uložením obrázku vo formáte JPEG.



Obr. 2.34: Použitie *SubVI* globálnej premennej s názvom *Global 1.vi* na zápis logickej hodnoty a načítanie číselnej hodnoty.

ovládačov a uložíme *SubVI* na pevný disk pod vhodným názvom. Následne je možné vytvorené *SubVI* vkladať do iných VI z ponuky *Select a VI...* vo *Functions Palette*.

2.5.4 DataSocket

Použitie globálnych premenných umožňuje zdieľanie údajov medzi VI bežiacimi na jednom počítači. Vývoj meracej techniky a ovládanie zložitých meracích a testovacích sústav si však vyžaduje zdieľanie údajov medzi počítačmi cez internetovú sieť aj na obrovské geografické vzdialenosti. Od vydania *LabVIEW* 6i je k dispozícii nástroj *DataSocket*, multiplatformný spôsob prenosu mnohých druhov údajov. Spojenie *DataSocket* pozostáva z klienta a servera, ktoré sa môžu, ale nemusia nachádzať na tom istom fyzickom počítači. Zdroj alebo cieľ údajov určíme prostredníctvom známeho protokolu Uniform Resource Locator (URL), rovnako ako sa používa vo webovom prehliadači. Natívny protokol pre spojenia *DataSocket* je DataSocket Transport Protocol (dstp). *DataSocket* server, ktorý je súčasťou inštalácie *LabVIEW* je potrebné spustiť manuálne na jednom z počítačov.

Samotné URL potrebné na pripojenie do vstupného terminálu funkcií na interakciu s *DataSocket* serverom je vo forme

```
dstp://localhost/premenna
```

pre server bežiaci na počítači, z ktorého chceme so serverom komunikovať,

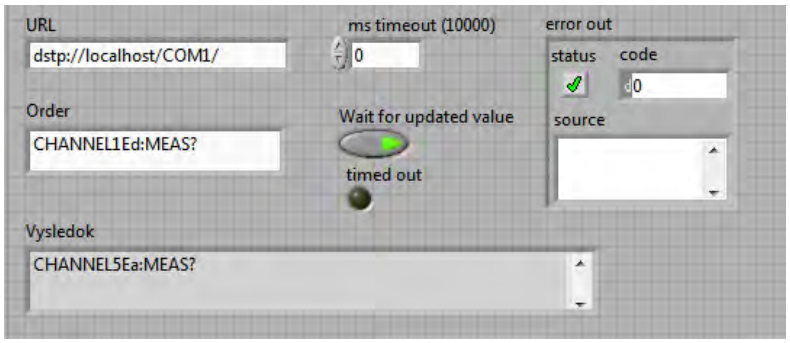
```
dstp://adresa.com/premenna
```

pre server identifikovaný verejnou internetovou doménou, alebo

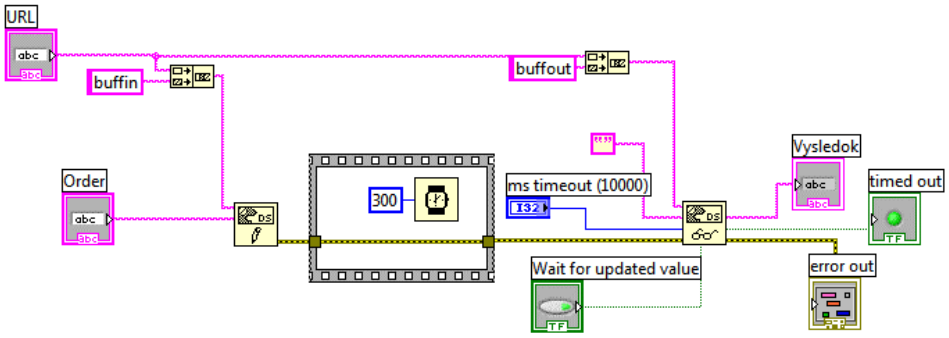
```
dstp://158.197.33.191/premenna
```

pre server identifikovaný číselnou IP adresou. Časť URL adresy *premenna* je označenie údajov, ktoré sa budú čítať z alebo zapisovať do *DataSocket* servera. Na komunikáciu s *DataSocket* serverom sú k dispozícii funkcie v ponuke *Data Commnunication*▸*DataSocket* vo *Functions Palette*. Príklad využitia *DataSocket* servera na prenos nameraných údajov z aplikácie výrobcu meracieho prístroja do VI, ktoré spracuje zbierané údaje je zobrazený na obr. 2.35.

Hodnoty zapísané v ovládačoch alebo indikátoroch je však možné synchronizovať cez *DataSocket* server bez použitia funkcií v *Blokovom diagrame*. Ovládač alebo indikátor je možné zviazať s *DataSocket* serverom cez konfiguračné dialógové okno objektu na paneli *Data Binding* (dátova väzba), kedy budú aktuálne údaje s označením *premenna* synchronizované s ovládačom alebo indikátorom (obr. 2.36).

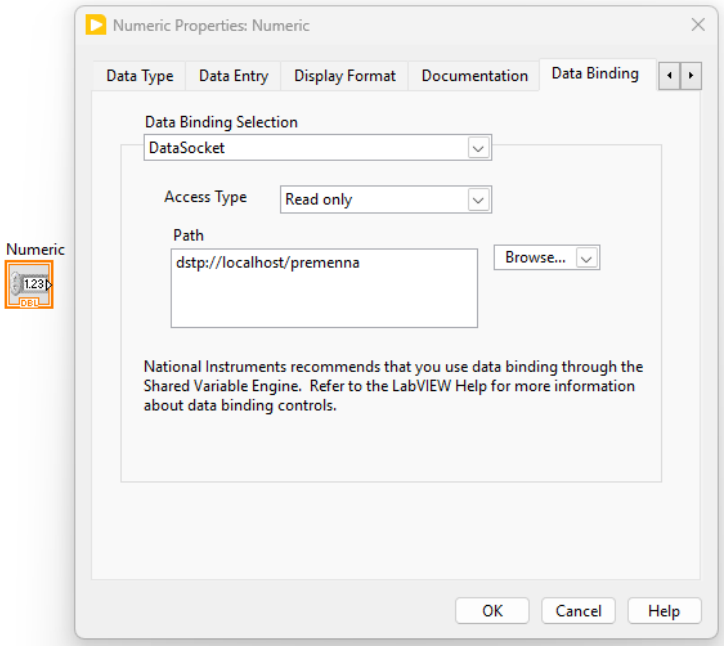


(a)



(b)

Obr. 2.35: Využitie *DataSocket* servera na prenos nameraných údajov z aplikácie výrobcu meracieho prístroja do VI, ktoré spracuje zbierané údaje. Údaje zapísané aplikáciou výrobcu sa načítajú z položky *buffout* pomocou funkcie *DataSocket Read* a pokyny meraciemu prístroju sa zapisujú do položky *buffin* pomocou funkcie *DataSocket Write*.

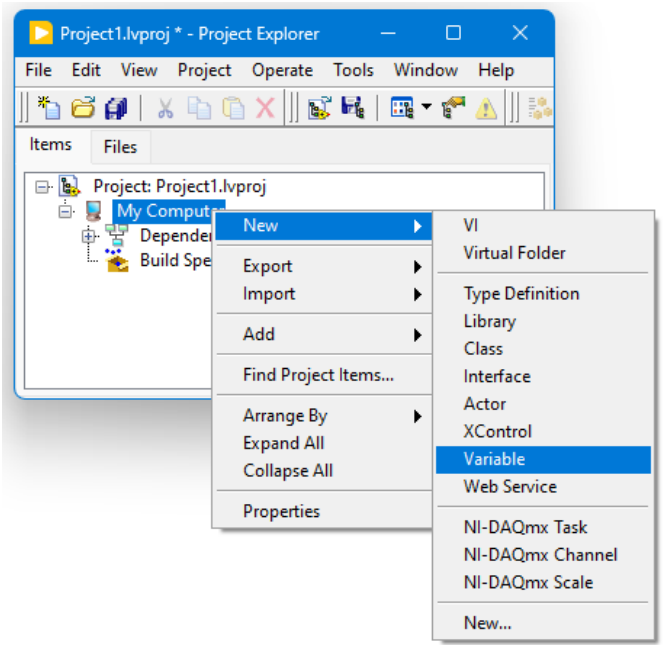


Obr. 2.36: Dialógové okno pre nastavenie synchronizácie údajov z ovládača do *DataSocket* servera cez panel *Data Binding*.

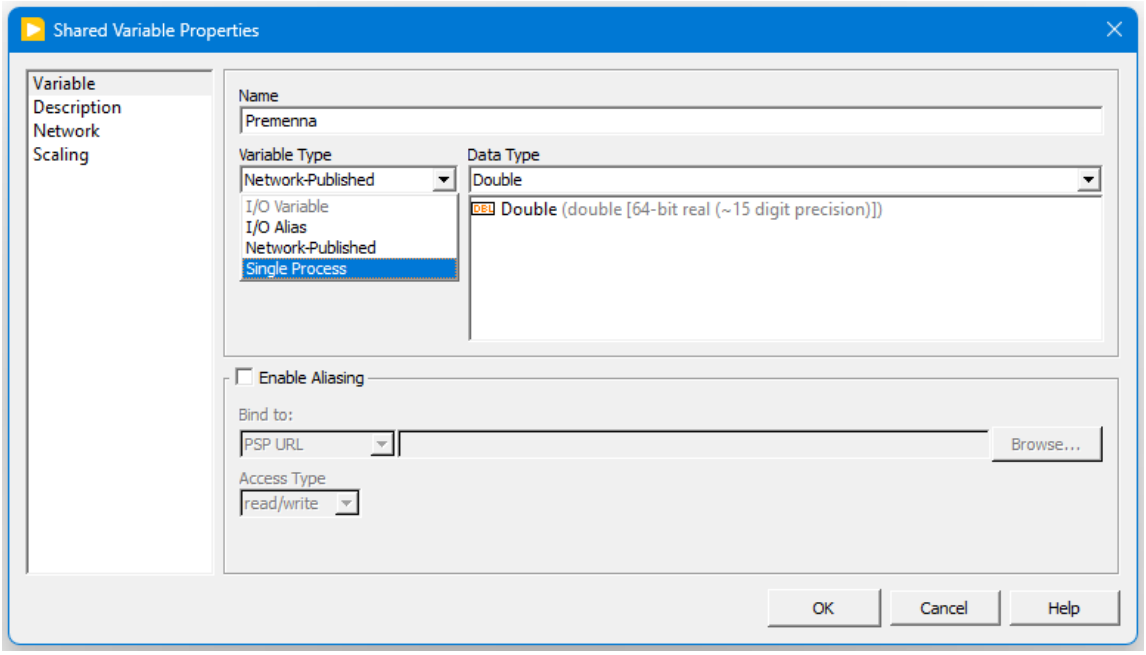
2.5.5 Zdieľané premenné

Pomocou zdieľaných premenných (*Shared Variables*) je možné zdieľať údaje medzi cyklami v jednom VI alebo aj medzi VI bežiacimi na rôznych počítačoch zapojených v internetovej sieti. K dispozícii sú dva typy zdieľaných premenných: jednoprocesové (*Single Process*) a publikované na sieti (*Network-Published*). Na umožnenie sieťovo publikovanej zdieľanej premennej posielať hodnoty prostredníctvom siete je k dispozícii softvérový rámec Shared Variable Engine (SVE). V operačnom systéme *Microsoft Windows* nakonfiguruje *LabVIEW* SVE ako službu a spustí ju pri štarte systému. SVE musí byť spustený aspoň na jednom z komunikujúcich počítačov. Každý počítač, ktorý ma prístup k SVE môže čítať z alebo zapisovať do zdieľaných premenných podľa potreby. Jednoduché vytvorenie zdieľanej premennej je možné v prehliadači *LabVIEW* projektu z kontextovej ponuky *New ▸ Variable* ako na obr. 2.37a a v konfiguračnom dialógovom okne na obr. 2.37b je možné upraviť nastavenie novej zdieľanej premennej.

Podobne ako v prípade využitia zdieľania údajov pomocou *DataSocket* servera, zdieľané premenné je možné vytvárať a konfigurovať priamo pomocou funkcií z ponuky *Data Commnunication ▸ Shared Variable* vo *Functions Palette*, následne z nich čítať a zapisovať do nich pomocou uzla zdieľanej premennej na *Blokovom diagrame* ako na obr. 2.38 alebo prostredníctvom dátovej väzby *Data Binding* cez konfiguračné dialógové okno objektu.



(a)

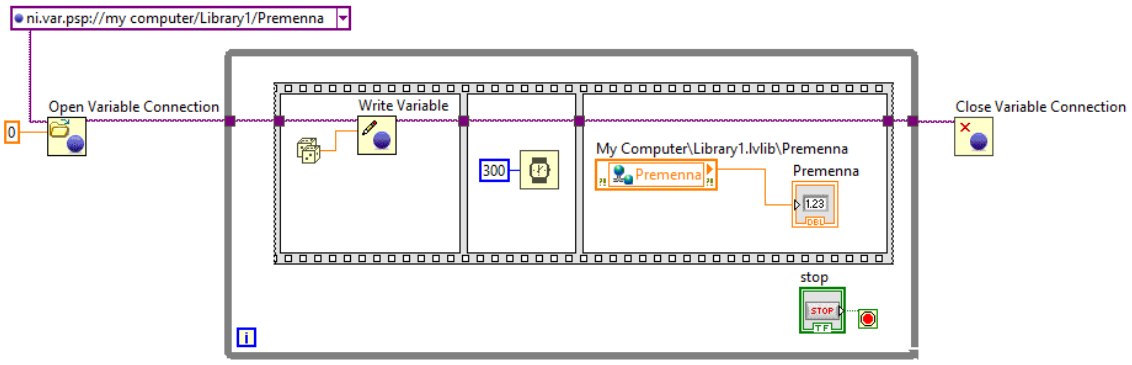


(b)

Obr. 2.37: Vytvorenie zdieľanej premennej z prehliadača *LabVIEW* projektu (a) a konfiguračné dialógové okno vytvorenej zdieľanej premennej (b).

2.6 ARCHITEKTÚRA PROGRAMU

Dizajn samotného programového kódu VI je možné navrhnúť v závislosti od toho, aké funkcie bude plniť, od jednoduchých úloh až po zložité riadiace štruktúry s rozhodovaním vo vykonávaní kódu podľa interakcie s užívateľom alebo podľa výsledku predchádzajúcich



Obr. 2.38: Použitie funkcií na prácu so zdieľanými premennými. Najprv je vytvorené spojenie s premennou *Premenna* (pre sieťový prístup je adresa napr. vo forme *ni.var.psp://localhost/Library1/Premenna*), postupne sa do nej zapisujú náhodne generované číselné hodnoty a pomocou uzla zdieľanej premennej sa zobrazujú do indikátora. Pri ukončení VI sa ukončí aj spojenie na premennú.

operácií. Najjednoduchšiu architektúru VI je možné použiť pri rýchlych laboratórnych meraniach. Program môže pozostávať z jediného VI, ktorý po spustení vykoná meranie, vykoná výpočty a zobrazí výsledky alebo ich zaznamená na disk a ukončí sa, ako je jednoduchý príklad na obr. 2.35, VI spúšťame tlačidlom *Run* . Táto architektúra sa bežne používa aj pre tvorbu *SubVI* ako funkčných komponentov v rámci väčších projektov.

Všeobecná architektúra VI je vhodná pri navrhovaní programu, ktorý má vo všeobecnosti tri hlavné fázy vykonávania:

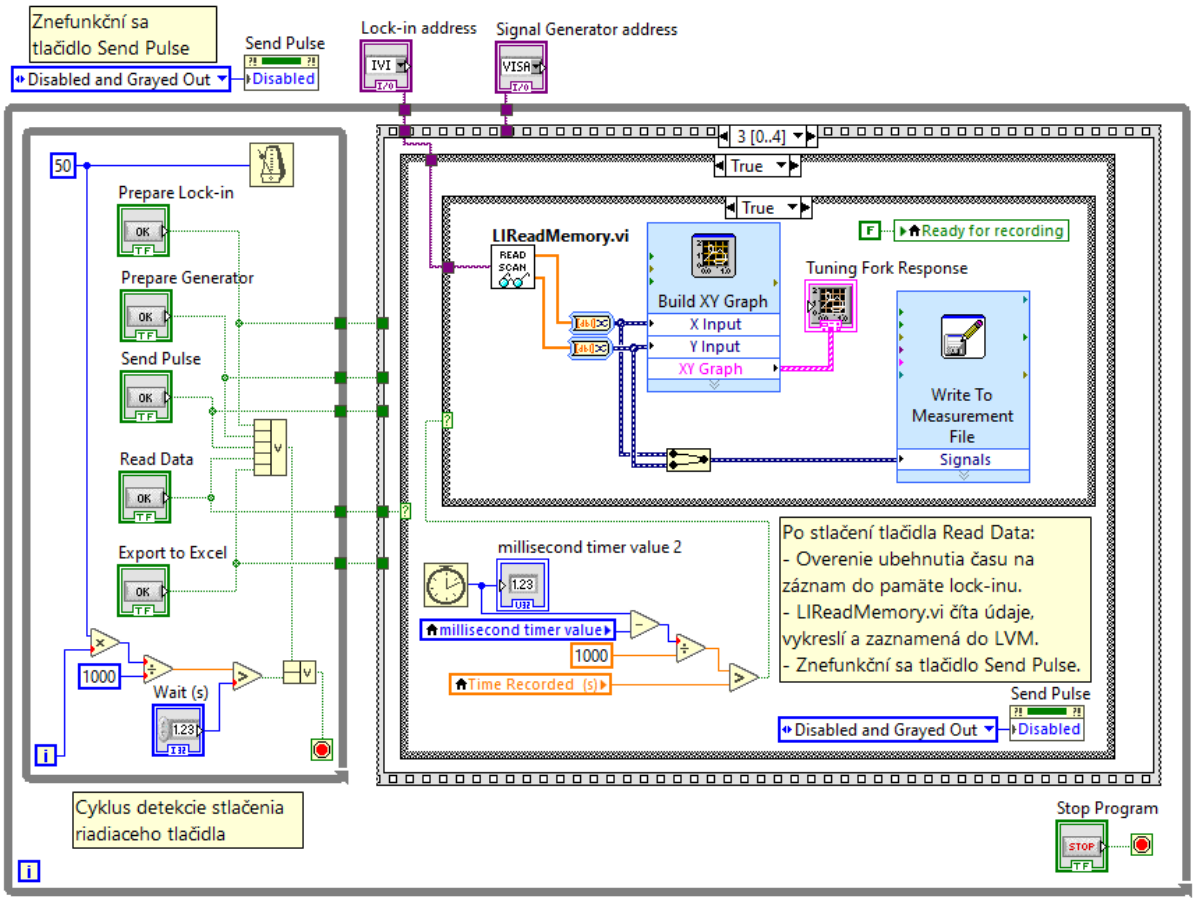
- Spustenie - inicializáciu hardvéru, načítanie konfiguračných informácií zo súborov alebo na výzva používateľovi na umiestnenie súborov na záznam údajov.
- Hlavná časť programu - zvyčajne pozostáva aspoň z jedného cyklu, kým sa používateľ nerozhodne program ukončiť, alebo kým sa program neukončí z iných dôvodov. V prípade jednoduchých aplikácií môže byť obsah cyklu pomerne jednoduchý. Pri použití zložitého používateľského rozhrania alebo pri použití viacerých podmienených štruktúr však môže byť komplikovanejší.
- Vypnutie - uzatvorenie súborov, zápis konfiguračných informácií na disk alebo obnovenie predvoleného stavu zariadení.

Príkladom tejto architektúry je *Blokový diagram* zobrazený na obr. 2.38, kde je pred cyklom umiestnená inicializácia zdieľanej premennej a po ukončení cyklu ovládacím tlačidlom je umiestnené uzatvorenie pripojenia k zdieľaným premenným. Tento typ VI spúšťame tlačidlom *Run Continuously* .

VI s architektúrou stavového stroja (*angl. state machine architecture*) je vhodný pri potrebe vykonávať všetky funkcie hocikedy, je relatívne jednoduchý pri potrebe zmeny a odladenia, nie je potrebné presne kontrolovať časovú následnosť. Týmto spôsobom tvorby sú diagramy kompaktnejšie v dôsledku použitia jedinej *Event Structure* na spracovanie

všetkých udalostí ako na obr. 2.17. Počas behu sa len prehľadáva zoznam možných udalostí alebo stavov a vykoná sa zodpovedajúca akcia. Nevýhodou môže byť aj možnosť preskočenia udalosti, ak sa práve spracúva iná udalosť, alebo ak sa vyskytnú dve udalosti v rovnakom čase. Zložitejšie verzie architektúry stavového stroja by obsahovali dodatočný kód, ktorý vytvára frontu udalostí (stavov).

Tento typ VI je možné vytvoriť aj bez použitia *Event Structure*, vyžaduje si však kombináciu podmieneného cyklu *While Loop*, logických ovládačov, rozhodovacích štruktúr *Case structure*, prípadne aj *Stacked Sequence* za zachovanie určitej postupnosti vykonávania príkazov. Počas behu sa program rozhoduje na základe získaných hodnôt alebo operácie užívateľa s logickými ovládačmi medzi možnými stavmi: spustenie, nečinnosť, udalosť 1, udalosť 2, atď., vypnutie a každý stav má priradený svoj rám s príslušným diagramom. Príklad na obr. 2.39 využíva interakciu užívateľa s logickými ovládačmi, tlačidlami, na spustenie vybraného rámu v *Stacked sequence* podľa toho, ktorá časť meracieho programu sa má vykonať.



Obr. 2.39: Zobrazený *VI s architektúrou stavového stroja* využíva interakciu užívateľa s logickými ovládačmi, tlačidlami, na spustenie vybraného rámu v *Stacked sequence* podľa toho, ktorá časť meracieho programu sa má vykonať. *While Loop* vnorená do hlavného cyklu obsahuje algoritmus načítania zmeny stavu logických ovládačov, ak dôjde k zmene jedného z nich, vykoná sa vybraný rám v *Stacked sequence*.

2.7 NÁSTROJE NA PRÁCU S HARDVÉROVÝMI PROSTRIEDKAMI

2.7.1 Komunikačné rozhrania

LabVIEW je nástroj na ovládanie a získavanie údajov z rôznych zariadení a meracích prístrojov, preto poskytuje podporu na identifikáciu a ovládanie rôznych typov zariadení pripojených k počítaču. Často sa využívajú Data acquisition (DAQ) systémy (systémy zberu údajov), sú to zariadenia, ktoré získavajú informácie z okolitého prostredia do počítača alebo posielajú rôzne druhy signálov do okolitého prostredia [18], môžu vykonávať rôzne úlohy, napríklad digitálny vstup/výstup, analógový vstup/výstup, generovanie Pulse Width Modulation (PWM) signálov (signály s impulzovou šírkovou moduláciou) [19], komunikácia s inými zariadeniami atď. Zariadenia DAQ často kombinujú viacero uvedených komponentov, ktoré sú prístupné cez zbernicu mikrokontroléra, ktorý môže spúšťať malé programy. Je množstvo výrobcov DAQ zariadení, aj samotná spoločnosť NI. Funkčnosť DAQ systémov je však možné v jednoduchých prípadoch nahradiť ekonomickou variantov vo forme vývojárskych mikročipových dosiek, ako je napr. *Arduino*. Vysokocitlivé merania, najmä vo vedeckých laboratóriách, sa však často vykonávajú meracími zariadeniami ako sú fázovo-citlivé zosilňovače (*angl.* nazývané aj *lock-in amplifier*) [20], funkčné generátory, nízkošumové generátory napätia alebo prúdu, kapacitné alebo odporové mostíky na presné meranie elektrických fyzikálnych veličín, elektronické stabilizátory teploty a pod.

Každé takéto zariadenie môže byť pripojené k počítaču viacerými typmi dátových zberníc, každá využívajúca iný štandardizovaný protokol komunikácie podľa pripojenie k portom počítača (paralelný, sériový, Universal Serial Bus (USB), atď.), ide o tzv. medzisystémove komunikačné rozhranie ako na obr. 2.40. DAQ zariadenia môžu predstavovať jednodoskové meracie moduly s funkciou meracieho prístroja vo forme kariet pripojených k slotom na základnej doske PC alebo v modulárnej skrinke so štandardizovaným rozhraním, ale bez ovládacích a zobrazovacích prvkov. V tomto prípade ide skôr o tzv. systémové komunikačné rozhranie.

Najbežnejšie medzisystémové komunikačné rozhrania predstavujú:

- Zbernica General Purpose Interface Bus (GPIB) (označená aj podľa medzinárodného štandardu IEEE 488.I) [21] je jedno z najbežnejších rozhraní v samostatných prístrojoch. GPIB je digitálne 8-bitové paralelné komunikačné rozhranie s rýchlosťou prenosu údajov až 8 Mb/s. Zbernica poskytuje jeden systémový radič až pre 14 prístrojov a kabeláž je obmedzená na menej ako 20 m. Obe tieto obmedzenia je možné prekonať použitím expandérov a extendérov GPIB. Káble a konektory GPIB sú univerzálne a priemyselne definované na použitie v akomkoľvek prostredí (obr. 2.41a). Keďže GPIB nie je štandardnou zbernicou pre osobné počítače, používajú sa zásuvné karty, napr. *PCI-GPIB*, alebo externý prevodník, napr. *GPIB-USB* (obr. 2.41b), aby bola pridaná do počítača funkcia ovládania prístrojov pomocou zbernice GPIB.

- USB bol navrhnutý predovšetkým na pripojenie periférnych zariadení počítača, ako sú klávesnice, myši, skenery a diskové jednotky. V súčasnosti takmer všetky meracie prístroje podporujú pripojenie cez USB. USB je technológia *plug-and-play*, pri ktorej počítač automaticky zistí pridanie nového zariadenia, požiada zariadenie o jeho identifikáciu a vhodne nakonfiguruje ovládače zariadenia. Pri používaní USB na riadenie prístrojov však existujú určité nevýhody: káble USB nie sú priemyselne klasifikované, čo potenciálne umožňuje stratu údajov pri zvýšenom elektrickom rušení; nemajú žiadny zaisťovací mechanizmus a dajú sa pomerne ľahko vytiahnuť z počítača alebo prístroja; maximálna dĺžka kábla je 30 m aj pri zapojení tzv. opakovača; pripojenie len jedného zariadenia na jeden USB vstup počítača.
- Sériové rozhranie typu RS232, RS485, RS422, alebo RS423 posiela a prijíma bajty informácií po jednom bite. Hoci je to pomalšie ako paralelná komunikácia, ktorá prenáša celý bajt naraz, je to jednoduchšie a môžete to používať na väčšie vzdialenosti. Na prenos údajov cez sériový port sa využívajú ASCII znaky. Komunikáciu prebieha pomocou troch prenosových liniek: zem, vysielanie a príjem. Keďže sériový port je asynchrónny, môže vysielať údaje na jednej linke a zároveň prijímať údaje na druhej. Ďalšie linky sú k dispozícii na tzv. *handshaking*, ale nie sú potrebné. Dôležité charakteristiky sériového portu sú prenosová rýchlosť, dátové bity, stop bity a parita. Aby dva porty mohli komunikovať, musia sa tieto parametre zhodovať.
- LAN eXtensions for Instrumentation (LXI) štandard, ktorý definuje komunikačné protokoly pre prístroje a systémy zberu údajov využívajúce internet.

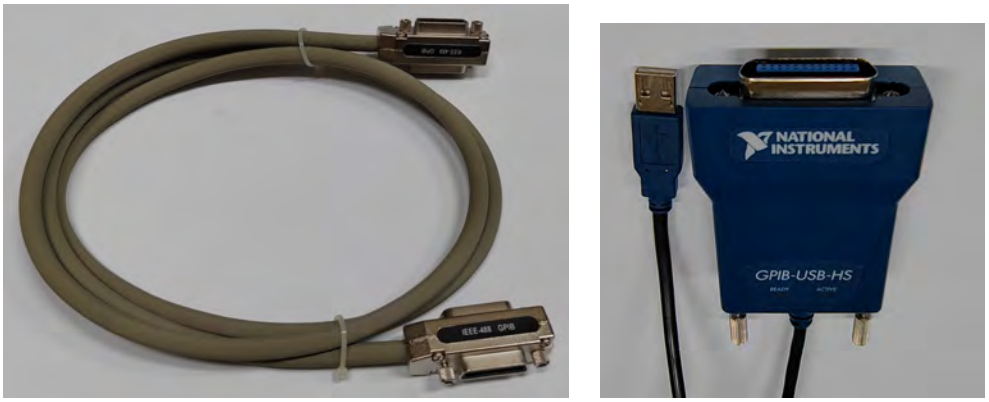
Najbežnejšie systémové komunikačné rozhrania predstavujú:

- Peripheral Component Interconnect (PCI) a Peripheral Component Interconnect Express (PCIe) zbernica sa nepoužíva priamo na ovládanie prístroja, ale ako periférna zbernica na pripojenie GPIB alebo sériových zariadení na ovládanie prístroja. Vďaka svojej veľkej šírke prenosového pásma sa PCI a PCIe používa aj ako nosná zbernica pre modulárne prístroje.
- PCI eXtensions for Instrumentation (PXI) je odolná platforma pre meracie a automatizačné systémy. PXI kombinuje vlastnosti zbernice PCI s robustným, modulárnym, mechanickým rámom, ktorý pridáva špecializované synchronizačné zbernice a softvérové funkcie. Takéto systémy slúžia na aplikácie vo výrobných testoch, vo vojenskom, leteckom a automobilovom priemysle, alebo pri monitorovaní strojov.

Vzhľadom na rôznorodosť technických riešení *LabVIEW* ponúka softvérové nástroje na zvládnutie riadenia experimentu a zberu údajov aj bez podrobnej znalosti každého komunikačného protokolu užívateľom alebo vývojárom vytvárajúcim VI knižnicu. Pre zvládnutie komunikácie s prístrojmi spočiatku boli potrebné špecifické ovládače komunikačných rozhraní, čo je však v súčasnosti vyriešené použitím rozhrania pre programovanie aplikácií (Application programming interface (API)) označeného Virtual Instrument Software Architecture (VISA), ktorý zjednocuje komunikáciu so zariadeniami bez ohľadu na fyzickú verziu

(a) (b)

Obr. 2.40: Príklady rozhrania: a) GPIB (IEEE 488) a RS232, b) LXI a USB.



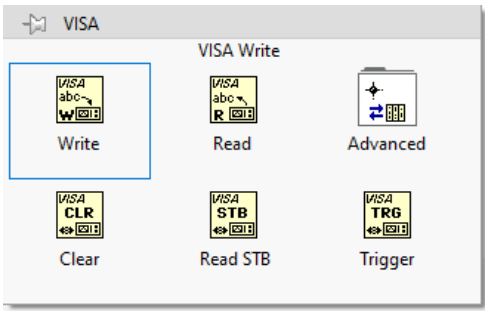
(a) (b)

Obr. 2.41: Kábel s konektormi GPIB (a) a externý prevodník *GPIB-USB* od výrobcu NI (b).

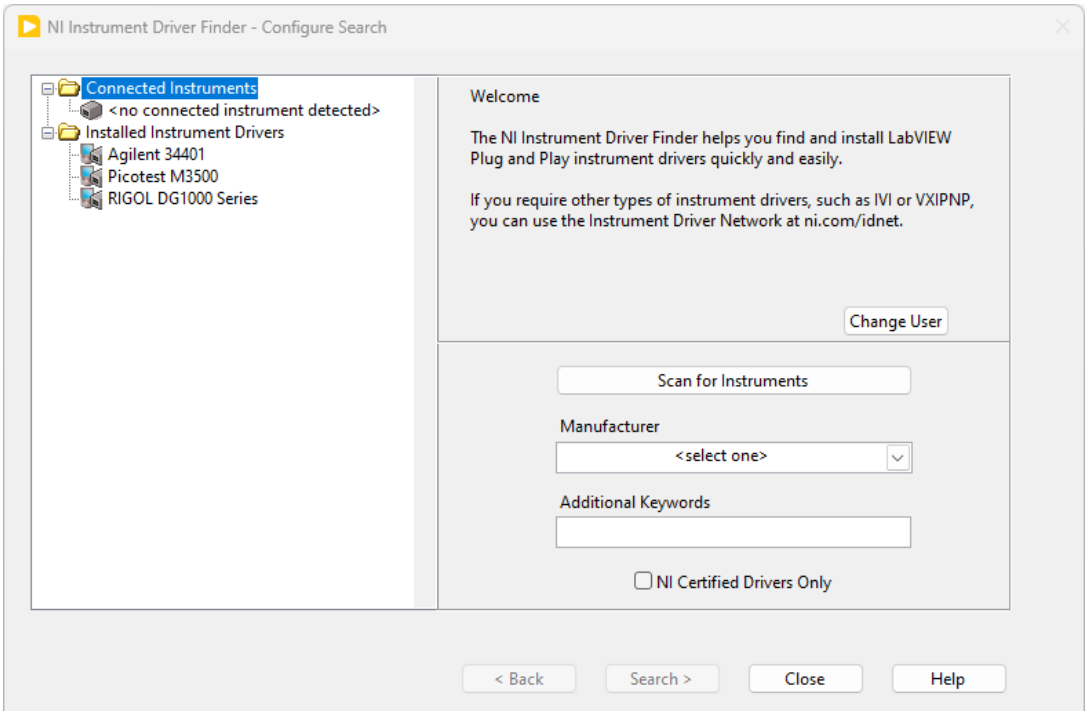
komunikačného rozhrania. VISA môže ovládať prístroje podporujúce GPIB, RS232, USB, LXI, alebo PXI protokol, pre ktorý VISA použije zodpovedajúci ovládač, a teda sa nemusíme učiť špecifický typ komunikácie pre každé zariadenie. S využitím VISA stačí zadať adresu (*VISA resource*), ktorá je špecifická pre každý komunikačný protokol a samotný príkaz. Funkcie na prácu s VISA v *LabVIEW* sú v ponuke *Instrument I/O ▸ VISA* vo *Functions Palette* (obr. 2.42). VISA je možné použiť na priamu komunikáciu s prístrojmi zaslaním príkazu, zvyčajne vo forme ASCII znakov, alebo si stiahnuť tzv. ovládače pre zariadenie, ktoré boli vytvorené pomocou knižníc VISA výrobcom zariadenia, komunitou vývojárov a pod. Spomenuté ovládače majú charakter *SubVI*, ktoré je možné využiť vo vlastnom VI, je ich možné nájsť aj cez ponuku *Help ▸ Find Instrument Drivers...* v *LabVIEW* (obr. 2.43). Výhodou je možnosť nazretia do *Blokového diagramu* takéhoto ovládača a jeho úpravy pre vlastné potreby ako je zobrazený na obr. 2.44.

LabVIEW ponúka aj inú triedu ovládačov, Interchangeable Virtual Instrument (IVI) [22] ako nadstavbu VISA. IVI je štandard ovládača prístroja, ktorý poskytuje spoločné API pre niekoľko tried prístrojov ako sú osciloskopy, generátory alebo digitálne multimetre. Sú to teda generické funkcie pre triedu prístrojov (volajú špecifické ovládače pre konkrétne prístroje) a prístroje sú týmto spôsobom jednoducho zameniteľné. Výhodou IVI ovládačov je napr. simulácia prístroja pri vývoji aplikácie bez použitia samotného prístroja.

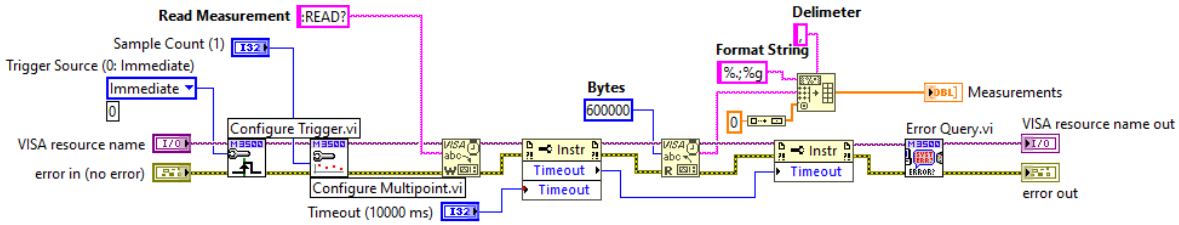
Pre DAQ systémy takisto existuje špecifická trieda ovládačov, ktorá je súčasťou knižnice *NI DAQmx* (najmä pre DAQ systémy vyrábané spoločnosťou NI).



Obr. 2.42: Ponuka VISA funkcií *Instument I/O ▸ VISA* vo *Functions Palette*.



Obr. 2.43: Nástroj na vyhľadávanie prístrojových ovládačov v online katalógu NI, ktoré je následne možné vložiť z *Functions Palette* do vlastného VI.

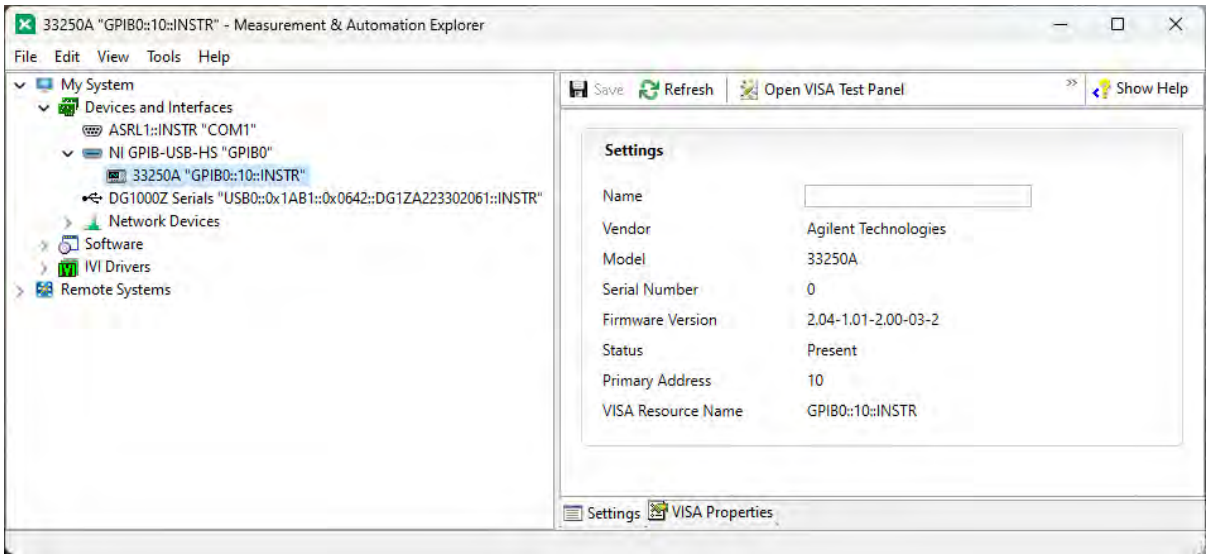


Obr. 2.44: Náhľad do *Blokového diagramu SubVI* na čítanie nameraných údajov z multimetra *Picotest* M3500 s využitím príkazov *VISA Write*, *VISA Read* a ďalších *SubVI* z knižnice ovládača potrebných na konfiguráciu multimetra.

2.7.2 Measurement & Automation Explorer

Pre zabezpečenie komunikácie so zariadeniami pripojenými k počítaču je potrebný nástroj na overenie pripojenia a získanie adresy alebo identifikátora hardvérového zdroja (angl. *resource*), ktorý je následne pripojený napr. vo vstupnom termináli s názvom *VISA Resource* vo funkciách *VISA Write* alebo *VISA Read*. Takýmto nástrojom je NI Measurement & Automation Explorer (MAX), s ktorým je možné:

- konfigurácia hardvéru a softvéru;
- export/import konfigurácie systému;
- vytváranie a úprava kanálov, úloh, rozhraní, škály a virtuálnych prístrojov (napr. pre zariadenia *NI DAQmx*);
- vytváranie simulovaných zariadení (napr. pre zariadenia *NI DAQmx*);
- vykonávanie diagnostiky systému a spúšťanie testovacích panelov;
- zobrazenie zariadení/prístrojov pripojených k systému a softvéru nainštalovaného v systéme (obr. 2.45).



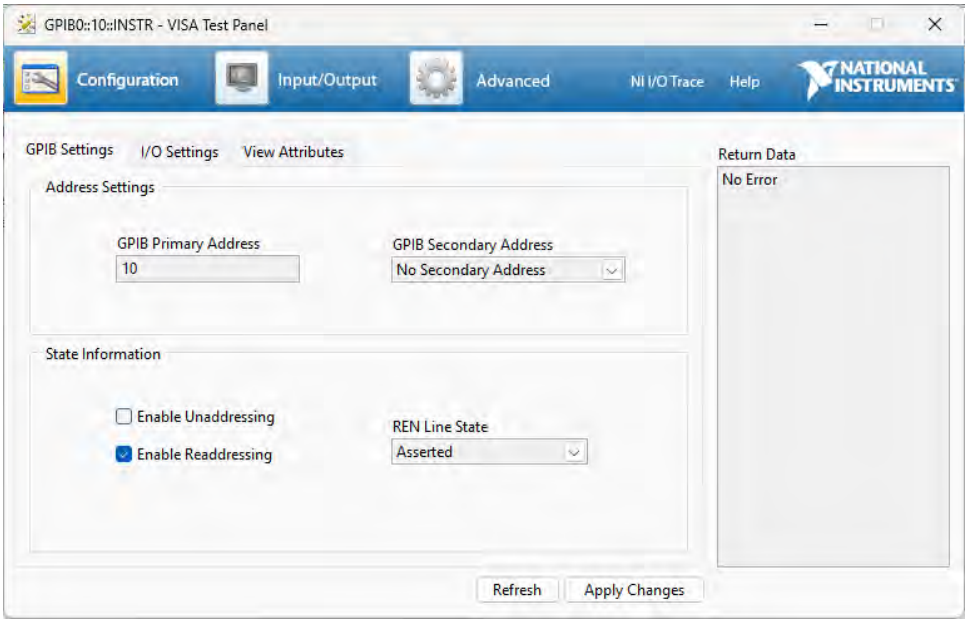
Obr. 2.45: *NI MAX* poskytuje jednoduchý spôsob overenia pripojených zariadení.

Pre každé pripojené zariadenie je možné zmeniť nastavenia a overiť komunikáciu pomocou ponuky *VISA Test Panel*, ako je zobrazené na obr. 2.46. Adresy alebo identifikátory hardvérových zdrojov majú štandardizovaný formát, ktorý však špecificky poukazuje na typ komunikačného protokolu:

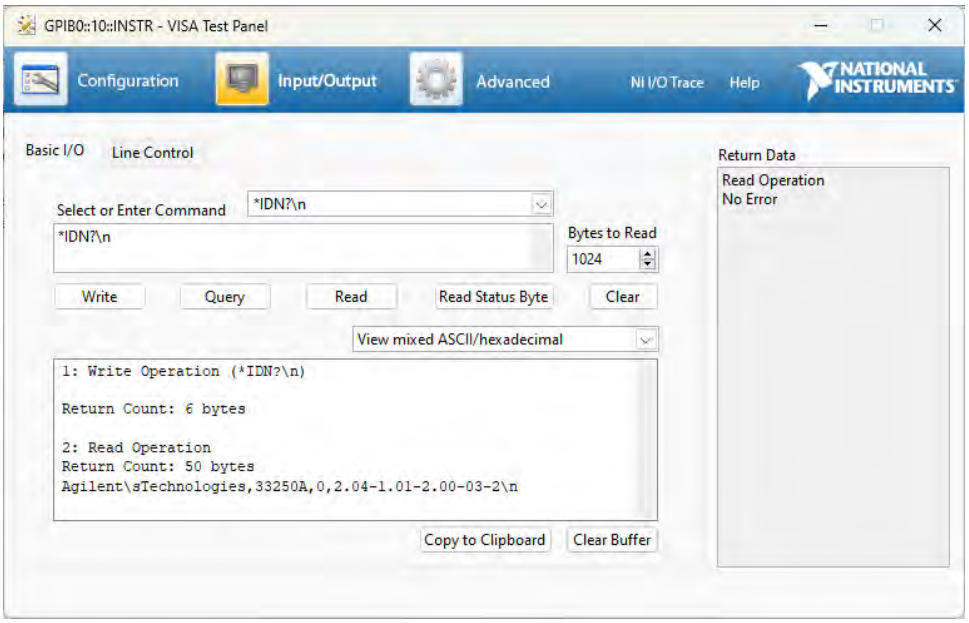
```
GPIB[board]::primary address[::GPIB secondary address][::INSTR]
ASRL[board][::INSTR]
TCPIP[board]::host address[::LAN device name][::INSTR]
```

```
USB[board]::manufacturer ID::model code::serial number[::USB interface number
    ][::INSTR]
USB[board]::manufacturer ID::model code::serial number[::USB interface number
    ]::RAW
```

teda napr. `GPIB0::10::INSTR` a je možné ich automaticky načítať do ovládača typu *VISA resource name* VISA na *Prednom paneli* pri vytváraní VI po úspešnej detekcii zariadenia v *NI MAX*.



(a)



(b)

Obr. 2.46: Testovanie komunikácie so zariadeniami v *NI MAX* pomocou ponuky *VISA Test Panel*.

2.7.3 Syntax komunikačných príkazov

Medzinárodný štandard IEEE-488.2 [23] poskytuje syntax základných komunikačných príkazov nezávislých od prístroja, ktorými je možné získať alebo nastaviť základné informácie komunikačnej zbernice zariadenia. Všeobecný príkaz (*angl. command*) alebo dopyt (*angl. query*) začína znakom hviezdičky (*). Dopyt, po ktorom vždy očakávame odpoveď zariadenia je ukončený znakom otáznika (?). Zariadenia často nepodporujú úplnú kompatibilitu s týmto štandardom a nepodporujú všetky standardné príkazy, zachovávajú si však aspoň podporu dopytu `*IDN?`, ktorý vyhľadá a vráti reťazec obsahujúci názov výrobcu, číslo modelu, sériové číslo a verziu firmvéru prístroja. Tento dopyt je možné otestovať pomocou ponuky *VISA Test Panel* v *NI MAX* (obr. 2.46). Hoci štandard IEEE-488.2 poskytoval syntax nezávislú od prístroja, až v roku 1990 bol predstavený štandard Standard Commands for Programmable Instruments (SCPI) [24] pre príkazy špecifické pre prístroj ako nadstavba IEEE-488.2. Sú to teda príkazy na ovládanie rovnakej triedy prístrojov, napr. multimetrov, ktoré sa dovtedy u rôznych výrobcov a dokonca aj modelov líšili.

SCPI štandard špecifikuje spoločnú syntax, štruktúru príkazov a formáty údajov, ktoré sa majú používať so všetkými prístrojmi. Príkazy SCPI sú založené na hierarchickej štruktúre, známej aj ako stromový systém. V tomto systéme sú súvisiace príkazy zoskupené do skupín pod spoločným uzlom alebo koreňom, čím vytvárajú podsystémy. Boli zavedené všeobecné príkazy (napr. `CONFigure` a `MEASure`), ktoré sa môžu používať s akýmkoľvek prístrojom. SCPI definuje aj niekoľko tried prístrojov, ktoré by mali implementovať rovnakú základnú triedu funkcií pre danú triedu prístrojov. Triedy prístrojov špecifikujú, ktoré subsystémy sa majú implementovať, ako aj všetky funkcie špecifické pre daný prístroj. Samotné príkazy SCPI aj odpovede zariadenia sú textové reťazce ASCII[4], ktoré sú sériou jedného alebo viacerých kľúčových slov, z ktorých mnohé majú parametre. Klúčové slová je možné použiť celé `CONFigure` alebo skrátené `CONF` (iba časť uvedenú veľkými písmenami). Protokol však nerozlišuje veľké a malé písmená, nie je teda *case sensitive* (*angl.*). Platia nasledujúce syntaktické pravidlá:

- dvojbodka (`:`) oddeľuje kľúčové slovo vyššej úrovne od kľúčového slova nižšej úrovne;
- jednoduché príkazy, ktoré začínajú dvojbodkou (`:`), sa interpretujú vzhľadom na koreň stromu príkazov, v opačnom prípade sa implicitne vzťahujú na posledný uzol predchádzajúceho príkazu (pokiaľ sa už nezačínajú hviezdičkou);
- v jednom reťazci možno prístroju zadať viacero príkazov, sú zložené z jednoduchých príkazov oddelených znakom bodkočiarky (`;`), ale pri zaslaní viacerých príkazov v jednom reťazci oddelených bodkočiarkou sa každý príkaz po bodkočiarke začína symbolom dvojbodky `:`;
- niektoré príkazy prijímajú alebo vyžadujú jeden alebo viacero ďalších argumentov, ktoré sa uvádzajú za príkazom a od príkazu sa oddeľujú medzerou;

4 V prípade hromadných údajov sa však môžu použiť pri odpovedi aj binárne formáty.

- dopyty, po ktorých očakávame odpoveď zariadenia, sú ukončené znakom otáznika (?).

Špecifické príkazy a možnosti pri zadávaní parametrov pre kľúčové slová je nutné si naštudovať v užívateľskom manuáli používaného zariadenia, zvyčajne sú však uvedené vo forme:

```
FREQuency {<frequency>|MINimum|MAXimum}
```

pričom

- v zložených zátvorkách sú uvedené možnosti parametrov pre daný reťazec príkazov, zátvorky samotné sa neposielajú spolu s príkazovým reťazcom;
- zvislá čiara | oddeľuje viacero možností parametrov pre daný príkazový reťazec;
- trojuholníkové zátvorky < > označujú nutné zadanie hodnoty parametra, napr. `FREQ 5000`;
- hranaté zátvorky označujú, že parameter je nepovinný a možno ho vynechať a nastaví sa predvolená hodnota.

2.7.4 Stavba komunikačných príkazov a dekódovanie odpovede zariadenia

Vzhľadom na textový charakter príkazov a odpovedí zariadení na nich, je často potrebné konverzia číselných hodnôt a ich správne formátovanie do reťazca pri zaslaní príkazu s číselnými parametrami prístroju a následná konverzia textovej odpovede na číselné hodnoty. *LabVIEW* preto ponúka široké spektrum funkcií v ponuke *Programming* ▸ *String* vo *Functions Palette* (obr. 2.47), potrebných na spájanie reťazcov, vyhľadávanie určitých častí reťazca, konverziu číselných hodnôt na reťazec a konverziu reťazca na číselnú hodnotu (v ponuke *Programming* ▸ *String* ▸ *Number/String Conversion*).

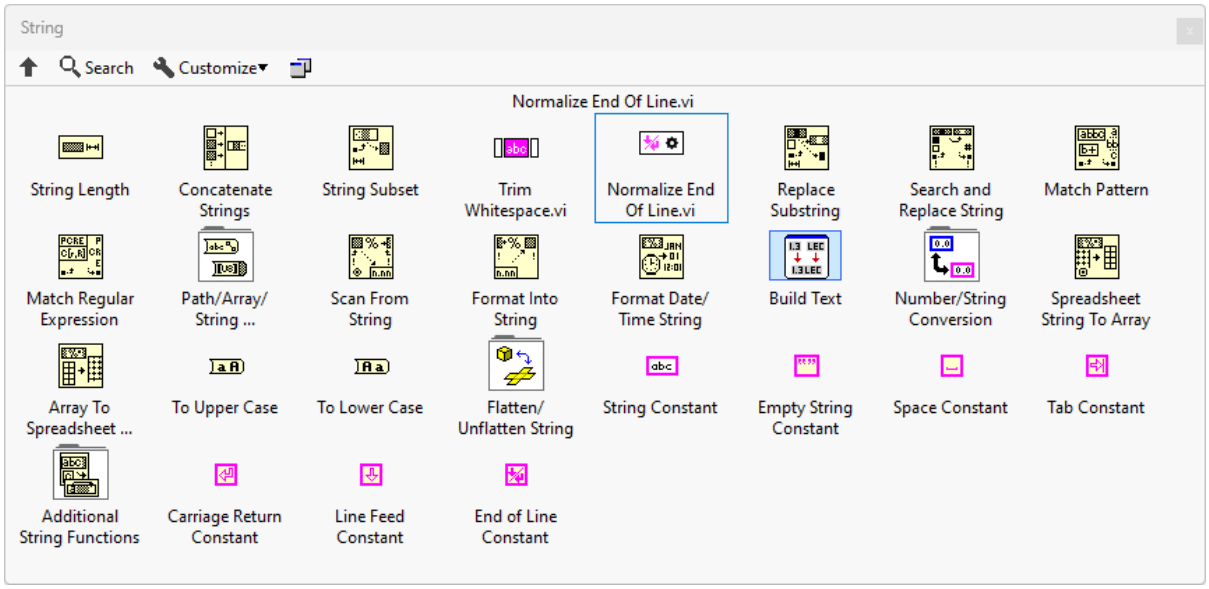
Najčastejšie je na konštrukciu príkazov využívaná funkcia *Format Into String* a na dekódovanie odpovede funkcia *Scan From String* zobrazené na obr. 2.48. Vstupom pre funkciu *Format Into String* je počiatočná textová časť príkazu, ku ktorej sa pripojí jedna alebo viacero číselných hodnôt konvertovaných na reťazec na základe špecifikátora formátu (kap. A.1). V novších vydaniach *LabVIEW* je k dispozícii na podobné účely aj expresná funkcia *Build Text*. Výstupom funkcie *Scan From String* je prvá identifikovaná číselná hodnota vo vstupnom reťazci a zvyšná časť reťazca, ktorá môže byť vstupom pre nasledujúcu funkciu *Scan From String*, až kým sa nezískajú všetky číselné hodnoty. Ak však odpoveď zariadenia obsahuje len jednu číselnú hodnotu bez ďalších znakov, často postačuje využitie jednoduchých funkcií pre konverziu reťazca na číselnú hodnotu.

Príklad komplexnej stavby zasielaného príkazu pre funkčný generátor *Rigol* série DG1000 je zobrazený na obr. 2.49, kde je presne určené aký typ signálu s definovanými parametrami sa má generovať na fyzickom výstupe zariadenia. Postupne sa odošlú príkazy

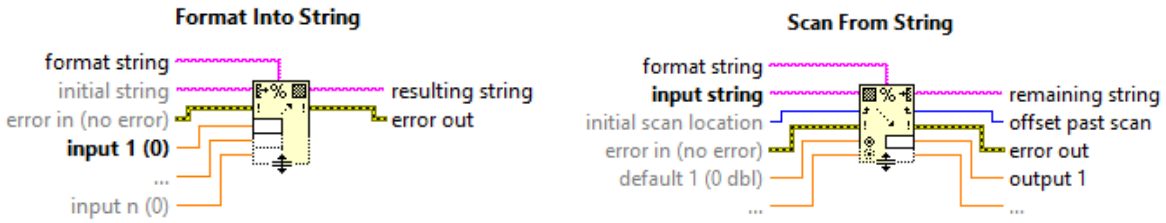
```
FUNC:CH1 SIN
VOLT:CH1 5.000000
```

```
FREQ:CH1 1000.000000
VOLT:OFFS:CH1 0.000000
```

a overia sa chybové hlásenia. Pri zasielaní príkazov pomocou VISA je zabezpečené aj zaslanie znaku označujúceho koniec riadku (*angl. end-of-line*), ktorý je potrebný pre korektné zaslanie príkazov štandardu SCPI aj IEEE-488.2. Pri práci s funkciami pre komunikáciu cez konkrétny komunikačný protokol (napr. *GPIB Write*) je potrebné ukončiť reťazec príkazu napr. pomocou funkcie *Normalize End Of Line* alebo pripojením znaku textovej konštanty *End of Line Constant* z ponuky *Programming*▸*String* (obr. 2.47).



Obr. 2.47: Ponuka funkcií na prácu s reťazcami.

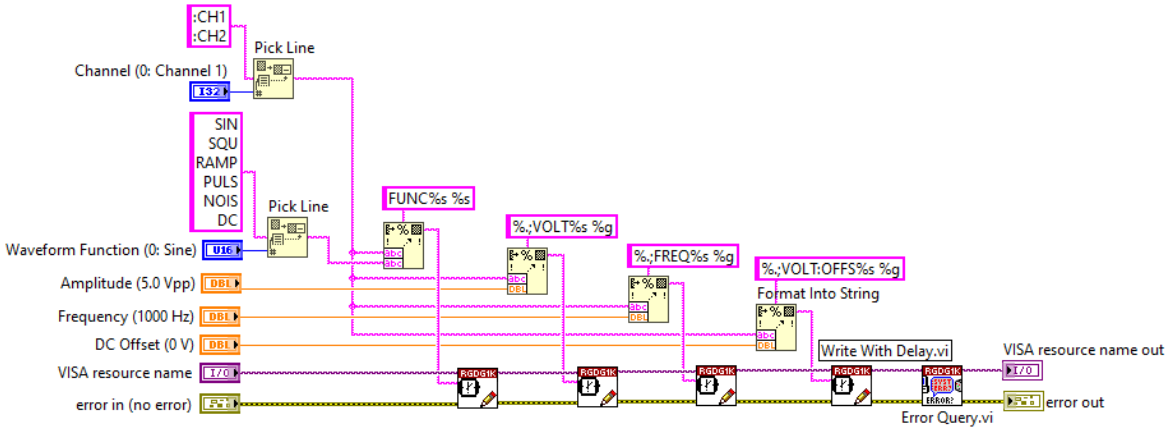


(a) (b)

Obr. 2.48: Funkcia *Format Into String* využívaná na konštrukciu príkazov a funkcia *Scan From String* na dekódovanie odpovede.

2.7.5 Podpora komunikácie s vývojárskymi mikročipovými doskami Arduino

Vývojárske mikročipové dosky *Arduino* už dlhodobo nachádzajú obľubu medzi domácimi hobby nadšencami aj profesionálmi pri stavbe testovacích projektov v oblasti mikroelektroniky, robotiky, Internet of Things (IoT) (Internet vecí) a podobne [25–29]. Je možné nájsť



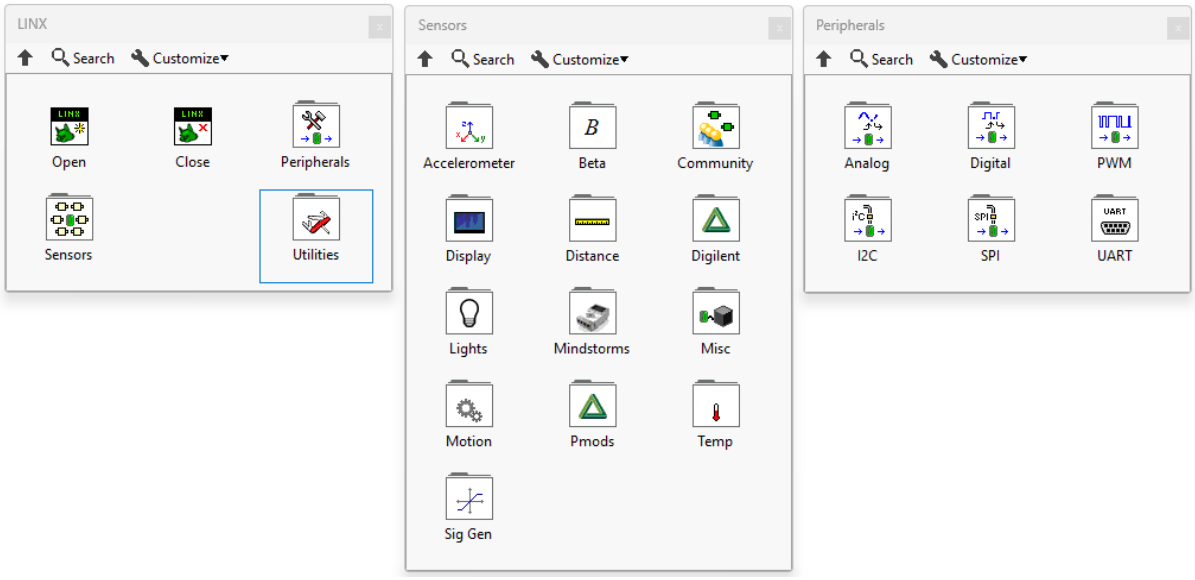
Obr. 2.49: Príklad komplexnej stavby zasielaného príkazu pre funkčný generátor *Rigol* série DG1000.

množstvo komplexných projektov riadených pomocou dosky *Arduino* ako sú plotery [30, 31], laserové gravírovačky [32] alebo dokonca 3D tlačiarne [33–36] a IoT projekty [37–39]. Pri riešení praktickej úlohy venovanej práci s doskou *Arduino* v *LabVIEW* budeme využívať model *Arduino Mega2560 Rev3* na obr. B.1a, ktorého podrobné technické špecifikácie sú uvedené v kap. B.1.

Aj keď vývojové prostredie na tvorbu riadiaceho kódu *Arduino IDE* [40] je postavené na programovacom jazyku *C/C++*, objavuje sa aj podpora pre programovací jazyk *Python* [41–43] obľúbený v *OpenSource* komunite. Jednoduchosť vytvárania riadiaceho kódu a vizuálneho zobrazenia ovládačov a grafickej prezentácie údajov získaných z periférií *Arduina* pre začiatočníkov a neskúsených programátorov viedla k vytvoreniu rozširujúcich balíkov na komunikáciu s doskami *Arduino* aj pre *LabVIEW*. Podpora *Arduina* v posledných vydaniach *LabVIEW* prešla rôznymi vývojovými krokmi. Hoci existoval rozširujúci balík *LabVIEW* poskytovaný spoločnosťou NI, medzi užívateľmi bol obľúbený rozširujúci balík pod názvom *LINX* dostupný na portáli *LabVIEW MakerHub* (www.labviewmakerhub.com) pre vydania *LabVIEW* 2014 a vyššie. Neskôr bol oficiálne podporovaný spoločnosťou NI pod názvom *NI LabVIEW LINX Toolkit*.

Tieto a iné rozširujúce balíky *LabVIEW* vytvárané komunitou vývojárov je možné nájsť na portáli *VIPM* (www.vipm.io), z ktorého je možné stiahnuť priamo samotné rozširujúce balíky *LabVIEW* alebo nástroj na ich vyhľadávanie, inštaláciu a správu pod názvom *VIPM* (alebo aj *JKI VI Package Manager*). Tento nástroj je možné v najnovších vydaniach získať priamo počas inštalácie *LabVIEW* a dopĺňa oficiálny nástroj na inštaláciu a správu súčastí *LabVIEW* pod názvom *NI Package Manager*.

V súčasnosti *LabVIEW* ponúka cez *NI Package Manager* najnovšie vydanie balíka na podporu komunikácie s doskami *Arduino* pod názvom *LabVIEW Hobbyist Toolkit*, ktorý je automaticky súčasťou inštalácie *LabVIEW Community Edition* a je odporúčaným balíkom od vydania *LabVIEW* 2021. Oba rozširujúce balíky sú podporované len pre 32-bitové vydania.



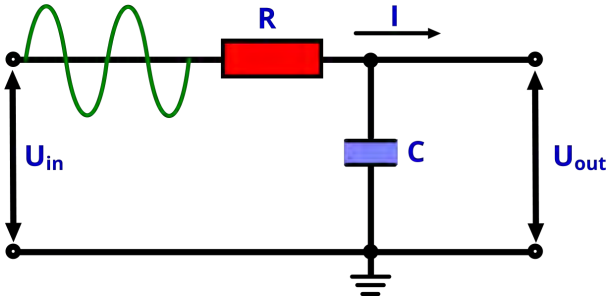
Obr. 2.50: Ponuka funkcií *MakerHub ▸LINX*, *MakerHub ▸LINX ▸Sensors* a *MakerHub ▸LINX ▸Peripherals* vo *Functions Palette* po inštalácii balíka *NI LabVIEW LINX Toolkit*.

Po inštalácii *NI LabVIEW LINX Toolkit* sa objaví vo *Functions Palette* nová ponuka *MakerHub ▸LINX* (obr. 2.50), alebo ponuka *Hobbyist* pre *LabVIEW Hobbyist Toolkit*. Ponuka obsahuje množstvo základných funkcií na priame ovládanie výstupných a vstupných portov dosky *Arduino*, ale aj špecializované funkcie na čítanie údajov zo štandardizovaných senzorov alebo ovládanie aktuátorov a obrazoviek, ktoré sú na trhu dostupné ako periférie pripojiteľné k doskám *Arduino*. K využívaniu uvedených rozširujúcich balíkov je potrebná jednorázová konfigurácia, overenie komunikačného protokolu a nahranie riadiaceho kódu (firmvér) do vybraného modelu *Arduino*. Podporované modely *Arduino* sú: *Leonardo*, *Mega2560*, *Nano*, *Pro Micro* a *UNO*. Konfiguračného sprievodcu je možné spustiť z ponuky *Tools ▸Hobbyist ▸Firmware Wizard...* alebo *Tools ▸MakerHub ▸LINX ▸LINX Firmware Wizard...* a jeho podrobný popis je zhrnutý v kap. B.2.

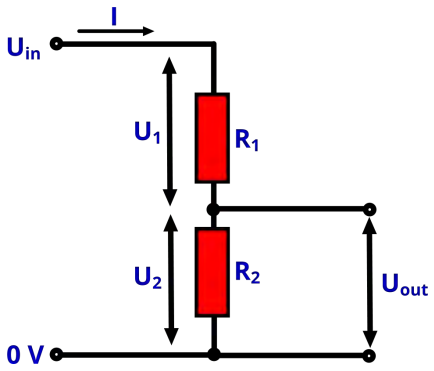
3 PRAKTICKÉ ÚLOHY

3.1 PRENOSOVÁ CHARAKTERISTIKA RC FILTRA TYPU DOLNÁ PRIEPUSŤ

V prvej praktickej úlohe je potrebné pripraviť program jednoduchého typu na zaznamenanie prenosovej charakteristiky RC obvodu na obr. 3.1, ktorý predstavuje pasívny filter striedavých signálov typu dolná priepusť. Samotná realizácia jednoduchého experimentu spočíva v postupnej zmene frekvencie harmonického napäťového signálu pri jeho konštantnej amplitúde pomocou funkčného generátora na vstupe obvodu a zaznamenávanie amplitúdy napäťového signálu na výstupe obvodu pomocou multimetra.



Obr. 3.1: Jednoduchý RC obvod predstavujúci pasívny filter striedavých signálov typu dolná priepusť.



Obr. 3.2: Jednoduchý rezistorový delič napätia.

Filtre premenlivých signálov sa vo všeobecnosti nazývajú podľa frekvenčného rozsahu signálov, ktoré cez ne môžu prejsť, pričom zvyšok blokujú alebo tlmia. Najčastejšie používané konštrukcie filtrov sú:

- Filter typu dolná priepusť - prepúšťa iba nízkofrekvenčné signály od 0 Hz do svojej medznej frekvencie f_c, pričom blokuje signály vyšších frekvencií

- Filter typu horná priepusť - prepúšťa iba vysokofrekvenčné signály od svojej medznej frekvencie f_c, pričom blokuje signály nižších frekvencií
- Pásmový filter - prepúšťa signály spadajúce do určitého frekvenčného pásma nastaveného medzi dvoma bodmi, pričom blokuje nižšie aj vyššie frekvencie na oboch stranách tohto frekvenčného pásma.

V nízkofrekvenčných aplikáciách (do 100 kHz) sa pasívne filtre zvyčajne konštruujú pomocou jednoduchých RC sietí (rezistor-kondenzátor), zatiaľ čo filtre s vyššou medznou frekvenciou (nad 100 kHz) sa zvyčajne vyrábajú z RLC komponentov (rezistor-induktor-kondenzátor). Pasívne filtre sa skladajú z pasívnych súčiastok, ako sú rezistory, kondenzátory a induktory, a nemajú žiadne zosilňovacie prvky (tranzistory, operačné zosilňovače atď.), ich výstupná amplitúda je vždy nižšia než vstupná.

Pasívny RC filter typu dolná priepusť na obr. 3.1 (označovaný naviac aj ako filter prvého rádu) používaná v tejto praktickej úlohe odfiltruje nežiadúce signály s frekvenciou vyššou ako medzná frekvencia f_c a je nenáročná na technickú realizáciu. V závislosti od toho, akým spôsobom pripojíme rezistor a kondenzátor vzhľadom na výstupný signál, určíme typ filtra, pri výmene rezistora a kondenzátora vzniká pasívny RC filter typu horná priepusť. Keďže funkciou každého filtra je umožniť signálom daného pásma frekvencií prechádzať bez zmeny a zároveň potlačiť všetky ostatné, amplitúdová charakteristika ideálneho filtra predstavuje skokovú funkciu pri f_c. Prechodová charakteristika jednoduchého filtra prvého rádu ma však zložitejší charakter. Súvisí to s reaktanciou kondenzátora (kapacitanciou), ktorá sa mení nepriamo úmerne s frekvenciou, zatiaľ čo hodnota rezistora zostáva pri zmene frekvencie konštantná. Pri nízkych frekvenciách bude kapacitancia X_C kondenzátora veľmi veľká v porovnaní s odporom rezistora R. Rozdiel potenciálov U_C na kondenzátore bude oveľa väčší ako úbytok napätia U_R na rezistore, pri vysokých frekvenciách je to naopak v dôsledku zmeny X_C. Kapacitancia sa mení s frekvenciou ako

$$X_C = \frac{1}{2\pi f C} \tag{3.1}$$

a impedancia obvodu na obr. 3.1 so sériovo zapojeným rezistorom a kondenzátorom je

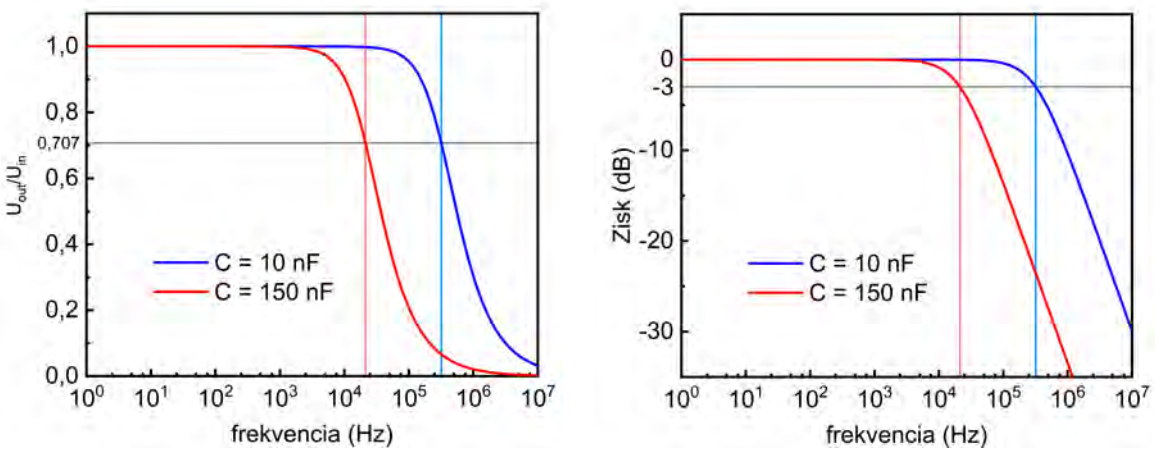
$$Z = \sqrt{R^2 + X_C^2}. \tag{3.2}$$

Uvedený obvod je analógiou rezistorového deliča napätia na obr. 3.2, ale závislého od frekvencie. Prúd pretekajúci takýmto obvodom je

$$I = U_{in}\,(R_1 + R_2) \tag{3.3}$$

a výstupné napätie U_{out} na zobrazených svorkách je

$$U_{out} = U_{in}\frac{R_2}{R_1 + R_2}. \tag{3.4}$$



Obr. 3.3: Prenosová charakteristika RC filtra typu dolná priepusť pre $R = 50\ \Omega$ a dve hodnoty $C = 10$ nF (modrá krivka) a $C = 150$ nF (červená krivka), vertikálne čiary (svetlomodrej a svetločervenej farby) zodpovedajú medznej frekvencii f_c, horizontálna šedá čiara označuje utlmenie signálu pre f_c.

Jednoduchou zámenou X_C za R_2 a impedancie z rov. 3.2 za celkový odpor rezistorovej siete získame vzťah pre prenosovú charakteristiku pasívneho RC filtra typu dolná priepusť na obr. 3.1

$$U_{out} = U_{in} \frac{X_C}{\sqrt{R^2 + X_C^2}}, \tag{3.5}$$

kde X_C je frekvenčne závislá podľa rov. 3.2. Samotná medzná frekvencia obvodu je daná vzťahom

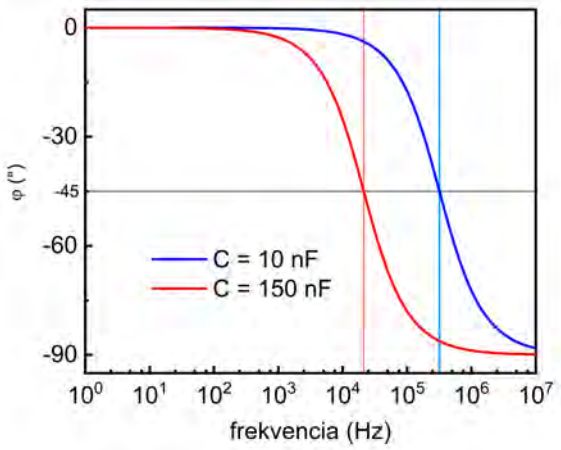
$$f_c = \frac{1}{2\pi RC}. \tag{3.6}$$

Prenosová charakteristika filtra, nazývaná aj Bodeho graf, zobrazená na obr. 3.3a, je pre nízke frekvencie takmer plochá a celý vstupný signál prechádza priamo na výstup, čo má za následok zosilnenie takmer 1, až kým nedosiahne f_c keďže kapacitancia je pri nízkych frekvenciách vysoká a blokuje akýkoľvek tok prúdu cez kondenzátor. V technickej praxi sa prenosová funkcia určuje ako závislosť zisku (alebo tlmenia) v decibeloch (dB)

$$A = 20 \log \frac{U_{out}}{U_{in}} \tag{3.7}$$

na frekvencii, ako je zobrazené na obr. 3.3b.

Na krivke prenosovej charakteristiky po bode f_c klesá odozva obvodu na nulu so sklonom -20 dB/dekádu a bude rovnaký pre akúkoľvek kombináciu R a C. Signály s vyššou frekvenciou sa výrazne zoslabia, keďže pri veľmi vysokých frekvenciách sa kapacitancia stáva takou nízkou, že na výstupných svorkách nastáva skrat, ktorého výsledkom je nulové



Obr. 3.4: Fázový posun RC filtra typu dolná priepusť pre $R = 50\ \Omega$ a dve hodnoty $C = 10$ nF (modrá krivka) a $C = 150$ nF (červená krivka), vertikálne čiary (svetlomodrej a svetločervenej farby) zodpovedajú medznej frekvencii f_c, horizontálna šedá čiara označuje fázový posun $\varphi = -45°$ pre f_c.

výstupné napätie. Všetky signály s frekvenciou pod f_c sú tlmené len málo alebo vôbec, teda sú v pásme priepustnosti filtra. Medzná frekvencia f_c je definovaná rovnosťou kapacitancie a odporu $R = X_C$, kedy sa výstupný signál zoslabí na 70,7 % (alebo $\frac{1}{\sqrt{2}}$) hodnoty vstupného signálu alebo utlmí o 3 dB.

V obvode vďaka prítomnosti kondenzátora dochádza k fázovému posunu výstupného napätia voči vstupnému o uhol

$$\varphi = -\arctan(2\pi f R C). \tag{3.8}$$

Je to spôsobené časom potrebným na nabitie platní kondenzátora pri zmene vstupného napätia. Výstupné napätie teda zaostáva za vstupným napätím, čím vyššia je frekvencia vstupného signálu aplikovaná na filter, tým viac kondenzátor zaostáva a obvod je viac mimo fázy. Frekvenčná závislosť fázového posunu je zobrazená na (obr. 3.4), pri hodnote f_c je fázový posun $\varphi = -45°$.

Technická realizácia:

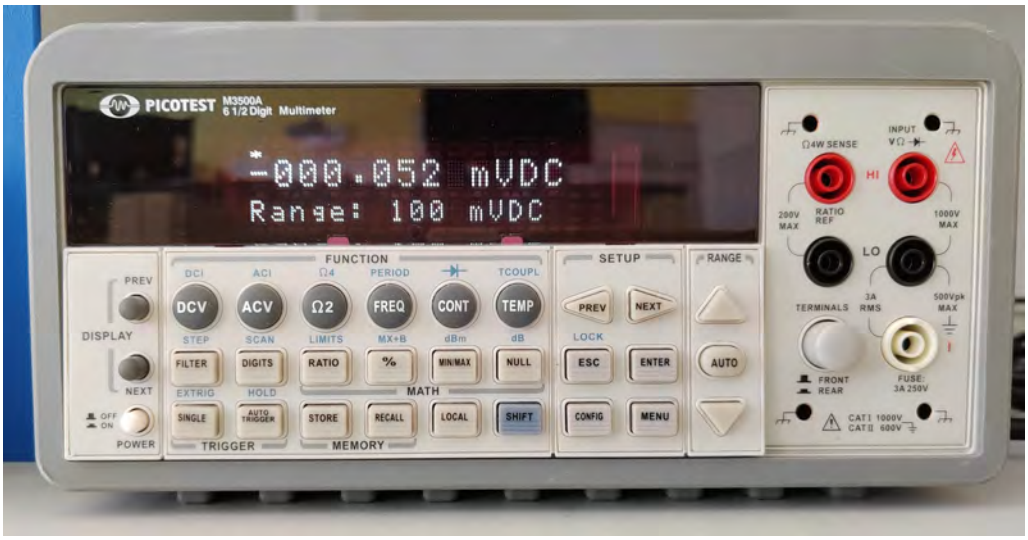
Na získanie prenosovej charakteristiky RC filtra typu dolná priepusť použijeme funkčný generátor *Rigol* DG1022Z [44] (obr. 3.5) pripojený na vstupné svorky filtra a multimeter *Picotest* M3500A [45] (obr. 3.6) pripojený na výstupné svorky filtra. Pri zostavení filtra využijeme dve hodnoty kapacity kondenzátora 10 nF a 150 nF. Samotný odpor vo filtri reprezentuje výstupná impedancia funkčného generátora 50 Ω.

Pri zostavení riadiaceho programu postačuje využiť jednoduchú architektúra VI, je potrebné dodržať nasledujúci algoritmus:

1. Pripravíme frekvenčný rozsah merania s definovaným krokom 1 kHz, prípadne 100 Hz od hodnoty $f_0 = 1$ kHz do dostatočne vysokej f_{max}[1], podľa cieľovej hodnoty medznej

1 Maximálny merací rozsah striedavých signálov multimetra *Picotest* M3500A je do 300 kHz.

Obr. 3.5: Ovládací panel funkčného generátora *Rigol* DG1022Z.



Obr. 3.6: Ovládací panel multimetra *Picotest* M3500A.

frekvencie f_c. Na tento účel využijeme funkciu *Labview Ramp Pattern* z ponuky *Signal Processing* ▸ *Signal Generation*. Vstupné parametre sa načítajú s číselných ovládačov.

2. Nastavíme budiaci harmonický signál (sínusový) a jeho amplitúdu $U_{VRMS} = 1$ V pre funkčný generátor[2].

3. Nastavíme typu merania (striedavé napätie) a merací rozsahu (automatický) pre multimeter.

2 Root-mean square voltage (VRMS) je efektívne napätie, teda amplitúda napätia predelená faktorom $\sqrt{2}$. Funkčný generátor umožňuje zadávať napätie ako Peak-to-peak voltage (VPP), teda dvojnásobok amplitúdy napätia.

4. Zapneme výstup generátora harmonického signálu, teda pripojíme výstup generátora na vstupné svorky filtra.
5. Postupne meníme frekvenciu v cykle, pri každej frekvencii načítame napätie na výstupe filtra, vykreslíme na graf a zaznamenáme do LVM súboru.
6. Vypneme výstup generátora harmonického signálu.

Pre komunikáciu s prístrojmi je možné využiť protokol SCPI. Takže pre komunikačný príkaz typu

```
MEASure:VOLTage:AC? {<range>|MIN|MAX|DEF},{<resolution>|MIN|MAX|DEF}
```

môžeme použiť plný zápis `MEASure:VOLTage:AC?` alebo skrátený zápis `MEAS:VOLT:AC?` s významom `meraj:napatie:striedave?` a otáznik naznačuje, že očakávame zaslanie odpovede (dopyt). Pri zaslaní viacerých príkazov v jednom reťazci je potrebné ich oddeliť bodkočiarkou a každý príkaz po bodkočiarke začína symbolom dvojbodky.

Funkčný generátor *Rigol* DG1022Z k riadiacemu počítaču pripojíme pomocou USB rozhrania s identifikátorom tvaru `USB0::0x1AB1::0x0642::DG1ZA223302061::INSTR`. Špecifikom pri zasielaní príkazov je výber výstupného kanálu. *Rigol* DG1022Z má dva výstupné kanály, pri využití kanála číslo 2 (výstup označený CH2), je potrebné špecifikovať v zaslanom príkaze reťazcom `SOURce2:` alebo `:SOUR2:` a nasleduje nastavenie parametrov pre tento kanál. Ak nie je toto kľúčové slovo využité, automaticky sú nastavené parametre pre kanál číslo 1 (výstup označený CH1). Potrebné príkazy pri nastavení funkčného generátora sú:

- typ signálu, sínusový signál — `APPL:SIN`
- frekvencie v Hertzoch — `FREQ 1000`
- automatický rozsahu amplitúdy — `VOLT:RANG:AUTO ON`
- typ amplitúdy — `VOLT:UNIT VPP` alebo `VOLT:UNIT VRMS`
- amplitúda harmonického signálu — `VOLT 1.0`
- zapnutie a vypnutie výstupu generátora — `OUTP ON` alebo `OUTP OFF`

Multimeter *Picotest* M3500A k riadiacemu počítaču pripojíme pomocou GPIB-USB prevodníka s identifikátorom tvaru `GPIB0::22::INSTR`. Potrebné príkazy pri nastavení multimetra sú:

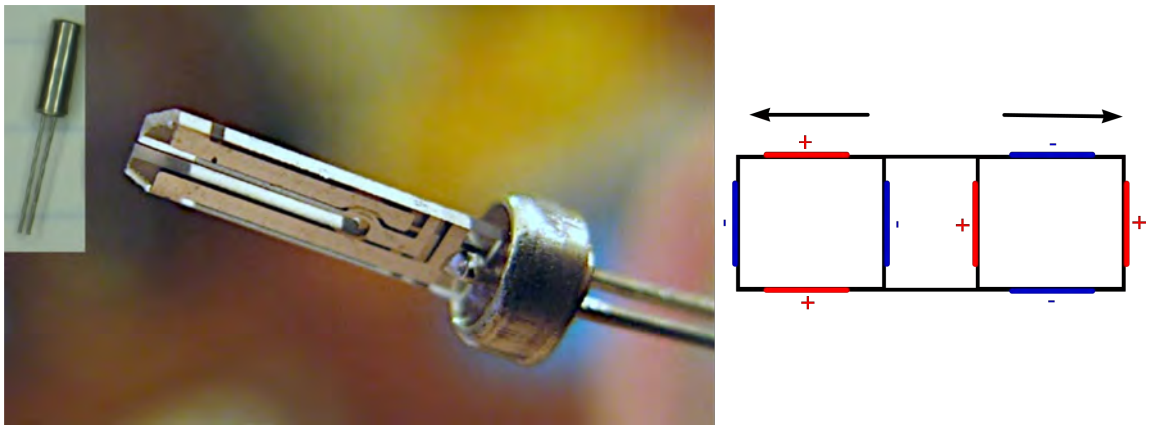
- druh merania — `CONF:VOLT:AC`
- automatický rozsahu merania — `SENS:VOLT:AC:RANGE:AUTO ON`
- meranie striedavého napätia — `MEAS:VOLT:AC?`

Nameraná hodnota amplitúdy napätia zaslaná počítaču je reťazec obsahujúci len číselnú hodnotu vo vedeckom formáte $\pm 1.000000e \pm 01$.

Časti vzorového VI pre získanie prenosovej charakteristiky RC filtra typu dolná priepusť sú zhrnuté v kap. C.1 na obr. C.1 a obr. C.2.

3.2 URČENIE VLASTNEJ REZONANČNEJ FREKVENCIE KMITOV KREMENNEJ LADIČKY

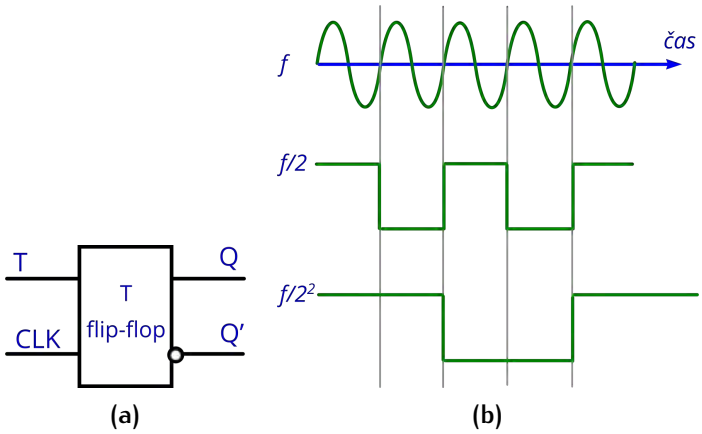
Ladička je zariadenie, ktoré vydáva akustický tón s presnou frekvenciou. Zvyčajne je to dvojramenná tyčka tvaru U z kovu. Po údere na ladičku ramená kmitajú s presnou frekvenciou, ktorá je určená rozmermi ramien a typom materiálu. Kremenná ladička (*angl. quartz tuning fork*) [46–49] je elektromechanická súčiastka vyhotovená z kryštálu kremeňa na ktorej ramenách sú pripojené elektrické kontakty (obr. 3.7). Kremeň je piezoelektrický materiál, dôsledkom čoho sa po pripojení zdroja elektrického napätia na kryštál menia jeho rozmery. Ak aplikujeme na elektródy ladičky na krátky moment premenlivé elektrické napätie s frekvenciou blízkou tzv. vlastnej rezonančnej frekvencii kmitov ramien ladičky, tak ladička bude vykonávať tlmené kmity s vlastnou frekvenciou po relatívne dlhú dobu. Vhodný dizajn elektród zabezpečí, že ramená ladičky budú kmitať v protifáze (obr. 3.7). Tieto mechanické zmeny rozmerov však zároveň vytvárajú elektrický prúd s tou istou frekvenciou s akou kmitajú ramená ladičky. Využitie kmitov kremennej ladičky s presne definovanou frekvenciou spôsobil revolúciu v konštrukcii náramkových hodiniek, prvý model bol komerčne uvedený na trh v roku 1969 spoločnosťou *Seiko* pod názvom *Quartz Astron 35SQ*. Vhodne nadizajnovaná kremenná ladička má frekvenciu vlastných kmitov v ultrazvukovej oblasti 32768 Hz a jej kmitanie na vlastnej rezonančnej frekvencii sa udržiava dodatočným elektronickým obvodom (prvé kremenné ladičky využívané spoločnosťou *Seiko* v náramkových hodinkách pracovali na frekvencii 8192 Hz, pri ktorej však vydávali počuteľný zvuk).



Obr. 3.7: Typická konštrukcia kremennej ladičky s frekvenciu vlastných kmitov 32768 Hz v cylindrickom kovovom púzdre. Vhodný dizajn elektród (vpravo, pohľad zhora) zabezpečí, že ramená ladičky budú kmitať v protifáze.

Elektronický obvod vyvolá kmitanie ladičky a následne pomocou logických preklápacích obvodov typu T, tzv. *T flip-flop*[3] na obr. 3.8a, postupne delí frekvenciu vstupných pulzov vo forme prúdu generovaného kremennou ladičkou na polovicu. Pri sekvenčnom zaradení pätnástich *T flip-flop* prvkov je frekvenciu 32768 Hz (čo je 2^{15}) postupne delená na konečnú

3 Preklápací logický obvod typu T je modifikáciou všeobecnejšieho preklápacieho logického obvodu typu JK.



Obr. 3.8: Preklápací logický obvod typu T, tzv. *T flip-flop* (a) pracujúci ako delič frekvencie (b). Do vstupu T sa privádza logický signál 1 a do vstupu CLK časovo-premenlivý signál z kremennej ladičky alebo z predradeného *T flip-flop* obvodu.

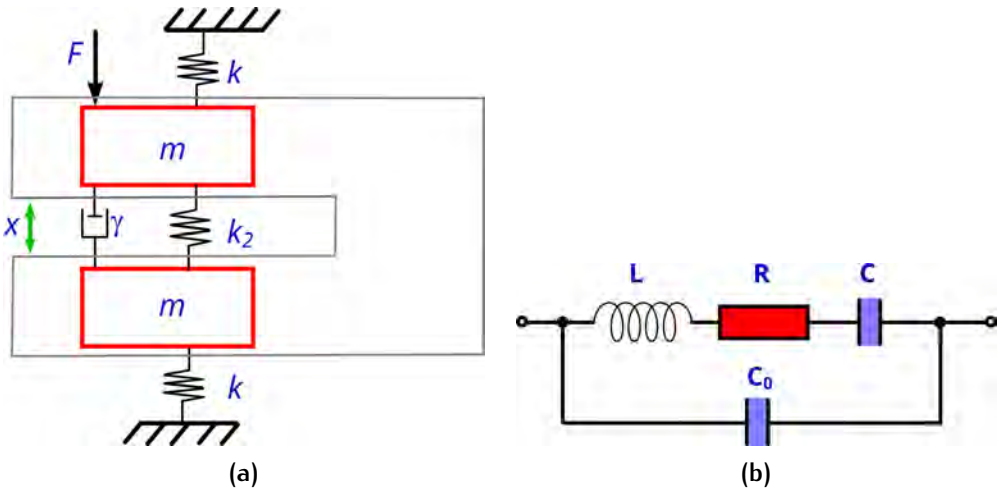
frekvenciu 1 Hz ako na obr. 3.8b. Vo výsledku získame digitálny signál s periódou 1 s pre určenie času. Rezonančná frekvencia nemusí byť vždy presná, čo môže viesť k oneskoreniu hodiniek, naviac je teplotne závislá s parabolickou teplotnou závislosťou v oblasti izbovej teploty [50]

$$f = f\left[1 - T_c\left(T - T_0\right)^2\right], \tag{3.9}$$

kde T_0 je zvyčajne teplota $25°C$ zodpovedajúca maximálnej vlastnej rezonančnej frekvencii f_0. T_c reprezentuje charakteristický parameter ladičky udávaný s typickou hodnotou 0,034 ppm/$°C$. Bežná odchýlka od f_0 na trhu dostupných ladičiek pre náramkové hodinky je ± 20 ppm pre oblasť $25 \pm 5°C$ (avšak u ladičiek používaných v náramkových hodinkách je to zvyčajne len ± 6 ppm), čo zodpovedá zaostávaniu alebo predbiehaniu času o 1,7 s za 24 hodín. Kremmenná ladička je hermeticky uzatvorená v kovovom púzdre a v náramkových hodinkách sa využíva kontakt s telom a je teda udržiavaná na stabilnej teplote blízkej teplote ľudského tela. Avšak pre veľmi presné meranie v technických aplikáciach môže byť konštrukcia oveľa zložitejšia s dodatočnou stabilizáciou teploty vo vnútri špeciálneho kontajnera alebo kompenzáciou zmeny vlastnej rezonančnej frekvencie s teplotou [51]. Zmena vlastnej rezonančnej frekvencie v závislosti od teploty však umožňuje využiť ladičku aj ako teplotný senzor pri veľmi nízkych kryogénnych teplotách, s výhodou nezávislosti od aplikovaného magnetického poľa [48].

Pre popis mechanických kmitov ladičky je možné využiť zjednodušený model, jednorozmerný symetrický model dvoch tlmených oscilátorov s hmotnosťou m na obr. 3.9a. Závažia s hmotnosťou m predstavujú ramená ladičky, väzba k tuhosť jednotlivých ramien a tuhosť k_2 s lineárnym tlmením γ sa v modeli nachádza kvôli vzájomnému pôsobeniu oboch ramien. Systém po uvedení do kmitania budiacou silou F je možné popísať pohybovou rovnicou

$$m\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + \omega_0 x = F_0 \sin\left(\omega t + \delta\right), \tag{3.10}$$



Obr. 3.9: Mechanický model kremennej ladičky (a) a jeho elektrický ekvivalent vo forme Butterworthovho-van Dykovho RLC obvodu (b).

kde $\omega_0 = \frac{k+2k_2}{m}$, F_0 je amplitúda budiacej sily, ω je budiaca kruhová frekvencia, δ fázový posun a x je vzájomná výchylka ramien. Riešenie takej rovnice má tvar:

$$x(t) = x_a \sin(\omega t) + x_d \cos(\omega t), \tag{3.11}$$

kde x_a a x_d sú absorpčné a disperzné komponenty oscilácií. Pre vlastnú rezonančnú frekvenciu, keď sú oscilácie maximálne platí

$$f_0 = \frac{\sqrt{km + 2k_2 m - \gamma^2}}{2\pi m}. \tag{3.12}$$

Elektrický ekvivalent mechanickej ladičky je Butterworthov-van Dykov RLC obvod popísaný rovnicou

$$\frac{d^2 I}{dt^2} + \frac{R}{L}\frac{dI}{dt} + \frac{I}{LC} = \frac{1}{L}\frac{dU}{dt}, \tag{3.13}$$

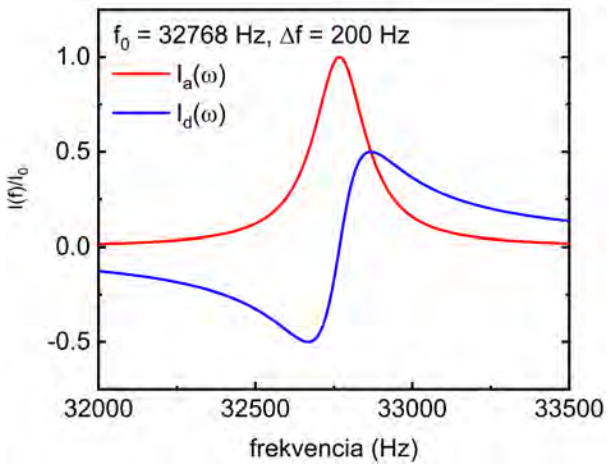
kde $U(t)$ je budiaci sínusový napäťový signál. Porovnaním s rov. 3.10 je vidieť, že $\omega_0 = \frac{1}{LC}$, $\gamma = \frac{R}{L}$ a riešenie má tvar

$$I(t) = I_a \sin(\omega t) + I_d \cos(\omega t) \tag{3.14}$$

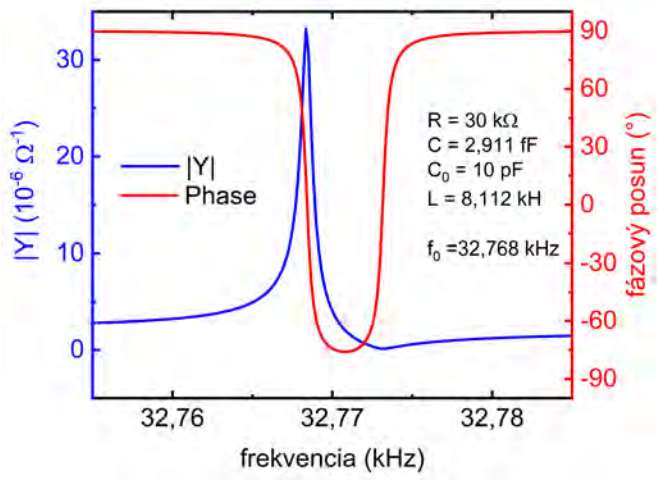
s I_a a I_d ako absorpčným a disperzným zložkám oscilácií prúdu

$$I_a(f) = \frac{I_0 (f\Delta f)^2}{(f\Delta f)^2 + \left(f^2 - f_0^2\right)^2}, \tag{3.15}$$

$$I_d(f) = \frac{I_0 f\Delta f \left(f^2 - f_0^2\right)}{(f\Delta f)^2 + \left(f^2 - f_0^2\right)^2}, \tag{3.16}$$



Obr. 3.10: Priebeh absorpčnej (rov. 3.15) a disperznej (rov. 3.16) zložky oscilácií prúdu kremennej ladičky.



Obr. 3.11: Priebeh veľkosti admitancie a fázového posunu kremennej ladičky určených z rov. 3.17 s vlastnou frekvenciou kmitov 32768 Hz.

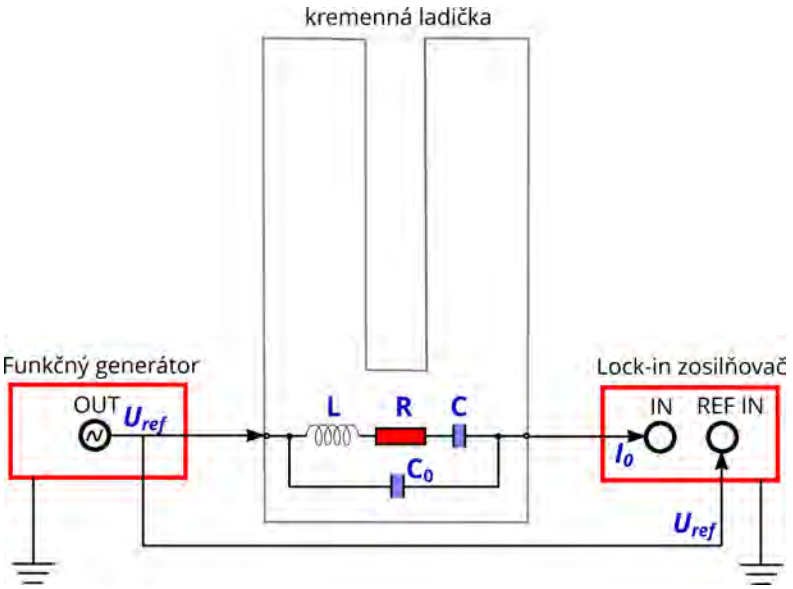
kde I_0 zodpovedá maximálnemu prúdu pri dosiahnutí vlastnej rezonančnej frekvencie a Δf reprezentuje šírku absorpčného maxima v polovici výšky absorpčného signálu a ich priebeh je zobrazený na obr. 3.10.

Komplexná admitancia uvedeného rezonančného obvodu je

$$Y(\omega) = \frac{1}{R + \frac{1}{i\omega C} + i\omega L} + i\omega C_0, \tag{3.17}$$

kde C_0 reprezentuje kapacitu medzi ramenami ladičky, ktorej charakteristické parametre je možné zviazať s mechanickými vlastnosťami ladičky po zavedení piezoelektricko-mechanickej väzbovej konštanty

$$\alpha = \frac{Q}{x}, \tag{3.18}$$



Obr. 3.12: Schéma merania vlastnej frekvencia kmitov kremennej ladičky heterodynnou pulznou metódou s využitím funkčného generátora a fázovo-citlivého zosilňovača.

ktorá udáva množstvo statického náboja Q na elektródach piezoelektrického materiálu na mechanickú výchylku x. Modifikácie upevnenia ladičky alebo napríklad zmena prostredia nemenia tento parameter. Priebeh veľkosti admitancie a fázového posunu v obvode je zobrazený na obr. 3.11. Následne je možné zviazať mechanické a elektrické parametre ladičky:

$$R = \frac{\gamma}{\alpha^2}, \quad L = \frac{m}{\alpha^2}, \quad C = \frac{\alpha^2}{k} \tag{3.19}$$

a zadefinovať parametre rezonančnej krivky

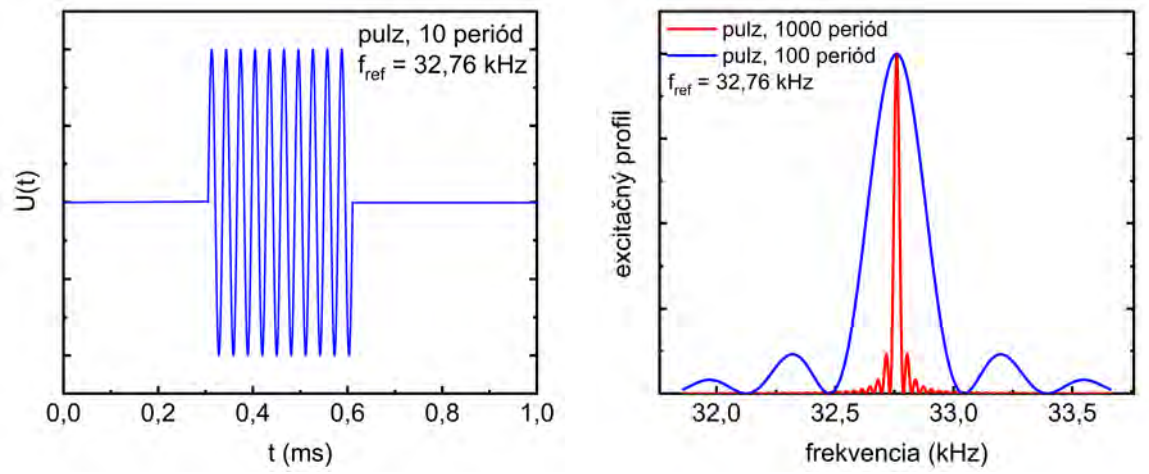
$$A_0 = \frac{1}{\gamma}, \quad \Delta f_B \propto \frac{\gamma}{m}, \quad Q = \frac{f_0}{\Delta f_B}, \tag{3.20}$$

kde A_0 je amplitúda rezonančnej krivky pri rezonančnej frekvencii, Δf_B je šírka rezonančnej krivky a Q je kvalita rezonátora.

Vlastná frekvencia kmitov kremennej ladičky je následne priradená tej hodnote frekvencie napájacieho napätia, pre ktorú obvodom preteká maximálny prúd (čo zodpovedá maximu admitancie). Pre určenie vlastnej frekvencie kmitov je možné zaznamenať odozvu ladičky zobrazenú na obr. 3.11 na postupnú zmenu frekvenciu budiaceho signálu, čo by si však vyžadovalo zmenu s jemným frekvenčným krokom a dostatočným časom na ustálenie oscilácii ladičky na určenej frekvencii[4].

Alternatívou je meranie metódou analogickou pulznej jadrovej magnetickej rezonancii po zapojení na obr. 3.12. Pulzná metóda je založená na súčasnom budení ladičky veľkou šírkou frekvenčného pásma obsiahnutého v krátkom napäťovom pulze obdĺžnikového

4 Pri vysokej kvalite Q je práve v okolí vlastnej frekvencie kmitov ladičky odozva ladičky na zmenu frekvencie budiaceho signálu pomalá.

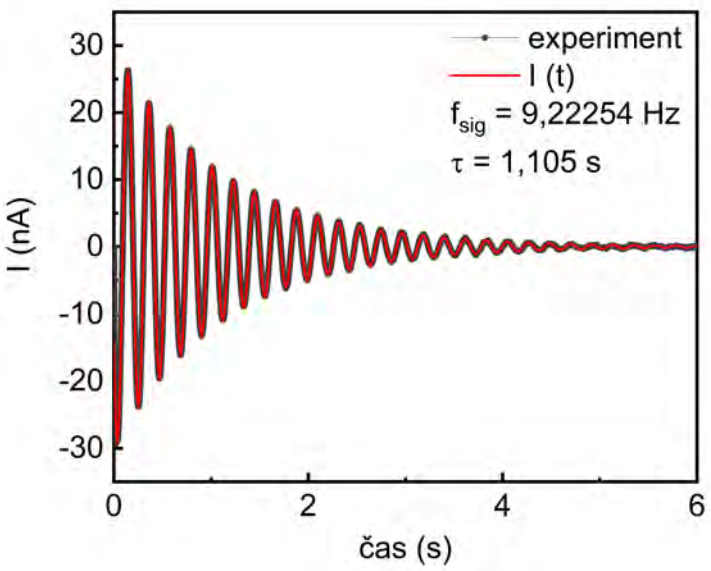


Obr. 3.13: Tvar obdĺžnikového pulzu a jeho excitačný profil pre dve rôzne dĺžky pulzu (100 a 1000 periód) pri $f_{ref} = 32{,}76$ kHz. Krátky pulz má oveľa širšie frekvenčné pásmo s vysokou intenzitou frekvencií, ktoré sú aj relatívne ďaleko od f_{ref}.

tvaru modulovanom frekvenciou blízkou vlastnej frekvencii kmitov ladičky. Dĺžka pulzu definuje šírku pásma budenia, teda aké harmonické frekvencie sú obsiahnuté v pulze, ako je znázornené na obrázku obr. 3.13a. Fourierovou transformáciou tvaru pulzu sa získa excitačný profil pulzu (obr. 3.13b). Pri vhodnom výbere základnej frekvencie pulzu je teda v spektre obsiahnutá aj vlastná frekvencia kmitov ladičky. Kremenná ladička teda po ukončení pulzu vykonáva tlmené kmity s vlastnou frekvenciou kmitov a táto odozva sa zaznamenáva metódou nazvanou heterodynná detekcia [52], založenou na zmiešavaní užitočného signálu s harmonickým signálom inej frekvencie. Pri tomto procese sa zmiešajú dva signály s frekvenciami f_1 a f_2, čím sa vytvoria dva nové signály, jeden so súčtom dvoch frekvencií $f_1 + f_2$ a druhý s rozdielom dvoch frekvencií $f_1 - f_2$. Zvyčajne je potrebný len jeden z výsledných signálov a druhý signál je odfiltrovaný z výstupu zmiešavača. Keďže pre presné zaznamenanie oscilácií prúdu tečúceho cez kremennú ladičku v oblasti desiatok kHz, by bolo potrebné využiť meracie zariadenie s veľmi rýchlou odozvou, práve vysokofrekvenčná zložka $f_1 + f_2$ je v tomto prípade odfiltrovaná a spracúva sa nízkofrekvenčná zložka $f_1 - f_2$, kde f_1 predstavuje frekvenciu kmitov ladičky f_0, ktorú chceme určiť, a f_2 predstavuje vhodne zvolenú frekvenciu budiaceho signálu f_{ref}. Zariadenie, ktoré priamo vykonáva zmiešavanie signálov, filtruje vysokofrekvenčnú zložku $f_0 + f_{ref}$ a spracuje $f_0 - f_{ref}$ je fázovo-citlivý zosilňovač (alebo lock-in zosilňovač)[5].

Kmity ladičky po pulze budiaceho signálu s f_{ref} generujú teda v obvode piezoelektrický prúd s frekvenciou f_0 postupne tlmený s časovou konštantou τ v závislosti od parametrov ladičky (γ, m, k), ktoré sú ovplyvnené napr. aj viskozitou plynu v okolí ramien ladičky. Pokles

5 Bežné lock-in zosilňovače spracúvajú napäťové signály, preto je potrebné previesť prúdovú odozvu ladičky na napäťový signál pomocou tzv. transimpedančného zosilňovača [53]. Pri praktickej realizácii tejto úlohy bude využitý lock-in zosilňovač so zabudovaným prevodom prúdového signálu na napäťový signál.



Obr. 3.14: Časový priebeh prúdovej odozvy ladičky po budiacom pulze popísaný rov. 3.21.

amplitúdy tlmených kmitov piezoelektrického prúdu po skončení budiaceho pulzu by bolo možné po prevode na napäťový signál zaznamenávať pomocou lock-in zosilňovača s $f_{ref} = f_0$. Rezonančnú frekvenciu však chceme presne určiť. Preto je vhodnejšie f_{ref} nastaviť napr. o 10 Hz nižšiu než je f_0, ktorá je daná technickými špecifikáciami ladičky. Lock-in zosilňovač teda bude merať premenlivý signál s relatívne malou frekvenciou $f_{sig} = f_0 - f_{ref}$, ktorý je tlmený s časovou konštantou τ z počiatočnej hodnoty I_0 (oneskorenie ϕ označuje posun krivky v okamihu začiatku záznamu voči ideálnemu prípadu, keď $I(t) = I(0)$)

$$I(t) = I(0)e^{-\frac{t}{\tau}} \sin\left(2\pi f_{sig} t + \phi\right) \tag{3.21}$$

namiesto jednoduchého poklesu amplitúdy signálu z ladičky

$$I(t) = I(0)e^{-\frac{t}{\tau}}, \tag{3.22}$$

ak by sa použila presne $f_{ref} = f_0$. Preložením rov. 3.21 na zmeraný premenlivý signál ladičky v závislosti od času metódou nelineárnej regresie môžeme presne určiť f_{sig} a rezonančnú frekvenciu $f_0 = f_{sig} + f_{ref}$ ako je zobrazené na obr. 3.14.

Technická realizácia:

Na získanie odozvy kremennej ladičky zobrazenej na obr. 3.14 pomocou zapojenia na obr. 3.12 použijeme funkčný generátor *Rigol* DG1022Z [44] (obr. 3.5) a lock-in zosilňovač *NF Corp.* LI5460 (obr. 3.15). Vstupné budenie ladičky pripojíme na výstup generátora `CH1` a výstupný signál z ladičky pripojíme na prúdový vstup lock-in zosilňovača `SIGNAL INPUT I`. Pri tomto meraní je zároveň nutné priviesť na vstup `EXT TRIG` na zadnom paneli lock-in zosilňovača signál z multifunčného výstupu na zadnom paneli funkčného generátora označeného `CH1/Sync/Ext Mod/Trig/FSK`, pomocou ktorého sa po oznámení o ukončení

Obr. 3.15: Ovládací panel lock-in zosilňovač *NF Corp.* LI5460.

budiaceho pulzu spustí rýchly záznam odozvy ladičky do pamäte lock-in zosilňovača pomocou signálu spúšte (*angl. trigger*). Ako referenčná frekvencia f_{ref} pre zmiešavač lock-in zosilňovača využijeme jeho interný oscilátor s amplitúdou napätia 1 V, keďže samotný budiaci napäťový signál z funkčného generátora je len na úrovni desiatok mV.

Pri zostavení riadiaceho programu využijeme modifikovanú architektúra stavového stroja s využitím reakcie VI na operácie s logickými ovládačmi zobrazenú na obr. 2.39. Pri zostavení meracieho programu je potrebné uvážiť samotný postup merania podľa nasledujúceho algoritmu:

1. Nastavíme parametre pulzu (`BURST` režim - frekvencia, napätie, počet cyklov, povoliť softvérovú spúšť, povoliť otvorenie výstupu, nastaviť výstupnú spúšť), použijeme pripravený SubVI súbor *RigolPreparePulse.vi*. Akcia sa vykoná po stlačení ovládacieho tlačidla.

2. Pripravíme záznam signálu do pamäte LI5460 (nastavenie referenčnej frekvencie, počet bodov a vzorkovací čas určí dobu záznamu, výber zaznamenaného signálu – napr. amplitúda signálu, priprava na záznam – tzv. *arming*), použijeme pripravený SubVI súbor *LIPrepare.vi*. Akcia sa vykoná po stlačení ovládacieho tlačidla.

3. Zaslanie budiaceho pulzu (spustiť spúšť – vykonanie pulzu s nastavenými parametrami zaslaním príkazu `*TRG` generátoru). Akcia sa vykoná po stlačení ovládacieho tlačidla.

4. Čakáme na dokončenie záznamu (doba je určené v kroku 1), automatický proces.

5. Načítame zaznamenanú časovú závislosť piezoelektrického prúdu (zastaviť *arming*, zistiť počet zaznamenaných bodov a vzorkovací čas, zistiť nastavený merací rozsah, prepočítať časovú os a konvertovať zaznamenané hodnoty na prúd) - použijeme pripravený SubVI súbor *LIReadMemory.vi*. Akcia sa vykoná po stlačení ovládacieho tlačidla.

6. Vykreslíme zaznamenané údaje a uložíme do súboru LVM automaticky po ukončení načítania údajov. Možný export údajov na grafe po stlačení ovládacieho tlačidla.

7. Cyklus sa vracia k bod 1 a reaguje na akciu ovládacích tlačidiel.

Parametre pre meranie:

- Rezonančná frekvencia kremennej ladičky s uzavretým púzdrom pri izbovej teplote je $f_0 = 32764{,}97$ Hz (závisí od viacerých vonkajších faktorov).
- Amplitúda budiaceho signálu $U_{VRMS} = 30 - 100$ mV (pre prípadne statické budenie pri testovaní obvodu neprekročiť 50 mV pri pripojení na prúdový vstup lock-in zosilňovača LI5460).
- Dĺžka pulzu 100 a 1000 períód (cyklov).
- Amplitúda referenčného signálu $U_{ref} = 1$ V (vnútorný generátor LI5460)
- Referenčná frekvencia $f_{ref} = 32760$ Hz (v tomto frekvenčnom rozsahu LI5460 umožňuje krok 10 Hz).

Pre komunikáciu s funkčným generátorom *Rigol* DG1022Z je možné využiť protokol SCPI ako je uvedené v kap. 3.1. Funkčný generátor *Rigol* DG1022Z k riadiacemu počítaču pripojíme pomocou USB rozhrania. Pri tvorbe meracieho VI využijeme knižnice ovládača funčného generátora z ponuky *LabVIEW* a pomocné SubVI *RigolPreparePulse.vi* a *RigolSendPulse.vi* opísané v kap. C.2. Pre zaslanie signálu spúšte z generátora signálu pre oznámenie začiatku záznamu údajov v lock-in zosilňovači po ukončení pulzu je potrebné nastaviť správne kedy bude zaslaný signál spúšte v tzv. `BURST` režime (režim pulzu) na tzv. negatívny sklon (`NEGATIVE`, `TRAILING`), keďže samotný pulz sa nezaznamenáva.

Lock-in zosilňovač *NF Corp.* LI5460 k riadiacemu počítaču pripojíme pomocou GPIB-USB prevodníka s identifikátor typu `GPIB0::3::INSTR`. Na komunikáciu je potrebný IVI ovládač poskytnutý výrobcom zariadenia, ktorý však je dostupný len pre 32-bitové vydanie *LabVIEW*. Pri tvorbe meracieho VI využijeme pomocné SubVI *LIPrepare.vi* a *LIReadMemory.vi* opísané v kap. C.2. Špecifické vlastnosti a nastavenia lock-in zosilňovača:

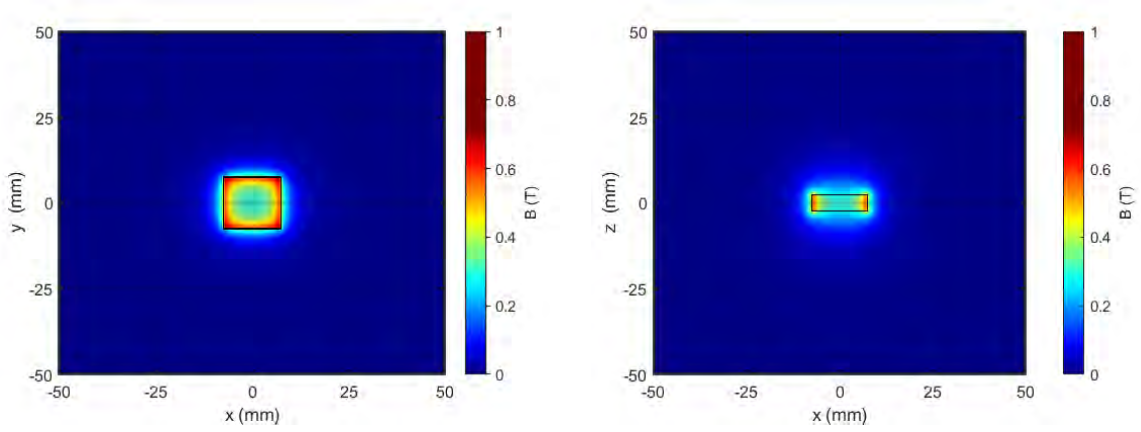
- Pred samotným meraním je potrebné softvérovo povoliť použitie vstupu externej spúšte `EXT TRIG`.
- Pre dostatočne rýchly záznam sa časový priebeh $I(t)$ zaznamenáva do vnútornej pamäte po zaregistrovaní signálu spúšte na vstupe `EXT TRIG`.
- Záznam do vnútornej pamäte je vykonaný vo forme 16-bitového čísla A, skutočná hodnota sa spočíta $I = A \times 2^{15} \times 1{,}2 \times I_{max}$ (I_{max} je použitý rozsah merania).
- Pôvodná verzia SubVI dodaná výrobcom vykonávajúca operáciu *Data Memory Read* má zabudovanú maximálnu veľkosť zásobníka hodnôt (angl. *buffer*) fixne na 2048 hodnôt. Je potrebné upraviť SubVI, ak sa zaznamenáva do pamäte viac ako 2048 hodnôt.

Časti vzorového VI a ich popis sú zhrnuté na obr. C.3 až obr. C.8 v kap. C.2.

3.3 MAPOVANIE MAGNETICKÉHO POĽA PERMANENTNÉHO MAGNETU

Mapovanie magnetických polí je dôležitou úlohou pri konštrukcii elektromagnetických zariadení, ktoré pozostávajú z permanentných magnetov alebo elektromagnetov, napr. v motoroch s permanentnými magnetmi a mikromotoroch, motoroch napájaných striedavým prúdom, elektromagnetických spínačoch a podobne. Aj v laboratórnej práci je často potrebné navrhnúť alebo overiť špecifické konfigurácie s magnetickým poľom pre rôzne experimenty. Na obr. 3.16 a obr. 3.17 je zobrazené rozloženie veľkosti magnetickej indukcie permanentného magnetu vo tvare kvádra a kocky, ktorý je polarizovaný v smere osi z s remanentnou magnetickou indukciou B_r, vypočítané pre $B_r = 1$ T pomocou balíčka *Magnetic field of solenoids and magnets* v prostredí *MATLAB* [54].

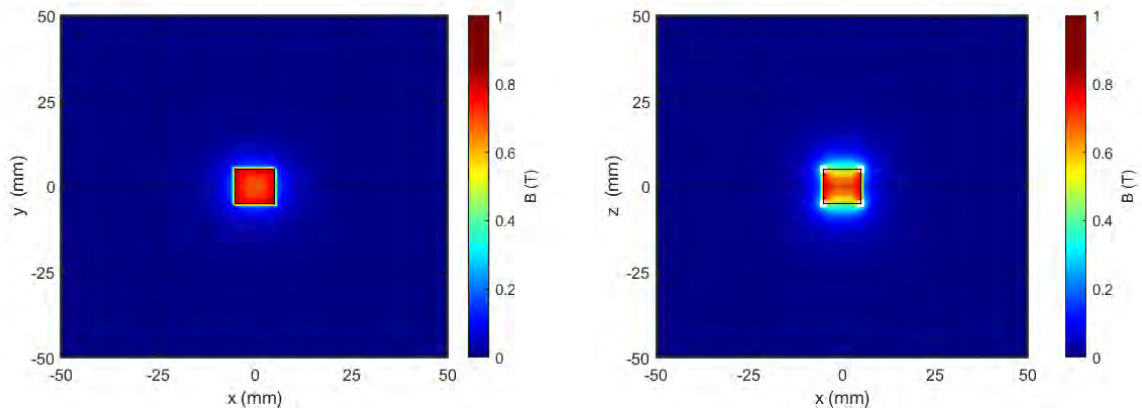
Existujú komerčné zariadenia, ktoré dokážu mapovať priestorové rozloženie magnetického poľa [55–57], avšak zvyčajne s vysokou cenou, a na vedecké účely pri mapovaní magnetického poľa je možné dosiahnuť citlivosť až niekoľko pT (pomocou Superconducting Quantum Interference Device (SQUID) senzora) [58] a atomárne rozlíšenie (pomocou spinovo-polarizovaného skenovacieho tunelového mikroskopu) [59]. V súčasnosti je však možné získať jednoduché a ekomonicky dostupné riešenia s presným polohovacím mechanizmom, ktoré tvoria napr. základ 3D tlačiarní, a pomocou trojosého mechanizmu založeného na presných krokových motoroch [60] mapovať rozloženie magnetického poľa pomocou vhodného senzora v kombinácii s *Arduinom* a *LabVIEW* [61–64]. Hriadeľ krokového motora, rotor, sa otáča pomocou sledu vstupných impulzov do série elektromagnetov o presne definovaný uhol a pomocou skrutkového hriadeľa premieňa otáčavý pohyb na lineárny[6].



(a) (b)

Obr. 3.16: Magnetická indukcia permanentného magnetu v tvare kvádra s rozmermi $15 \times 15 \times 5$ mm^3 polarizovaného v smere osi z v dvoch význačných rovinách pre $B_r = 1$ T.

6 Jednoduchšie krokové motory používané v 3D tlači majú rotor s permanentným magnetom, ktorý môže ovplyvniť samotné testované prostredie.



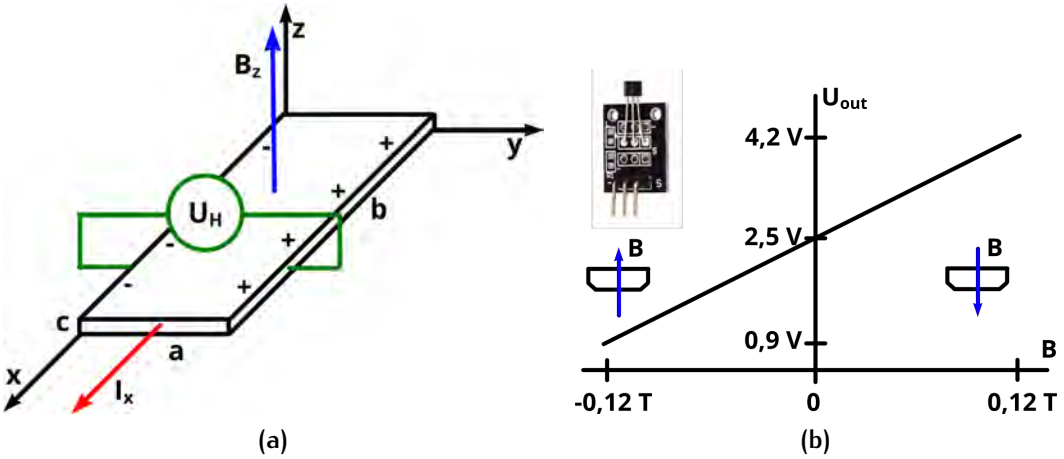
Obr. 3.17: Magnetická indukcie permanentného magnetu v tvare kocky s rozmermi $10 \times 10 \times 10$ mm^3 polarizovaného v smere osi z v dvoch význačných rovinách pre $B_r = 1$ T.

Pri realizácii praktickej úlohy zároveň využijeme jeden z najuniverzálnejších senzorov magnetického poľa, ktorý umožňuje merať magnetickú indukciu v širokom rozsahu na základe Hallovho javu [65]. Ich výhoda spočíva v nízkej cene a možnosti vyrábať aj miniatúrne snímače s mikrometrovým priestorovým rozlíšením. Hallov jav je založený na pôsobení Lorentzovej sily na pohybujúce sa nosiče náboja vo vodiči, ktorým preteká elektrický prúd ako je zobrazené na obr. 3.18a. Vektor magnetickej indukcie B_z je orientovaný kolmo na smer jednosmerného elektrického prúdu I_x pretekajúceho cez tenký vodivý element tvaru doštičky alebo vrstvy s rozmermi $a \times b \times c$. Následkom pôsobenia Lorentzovej sily na nosiče náboja sa hromadí záporný náboj na jednej strane a kladný náboj na druhej strane vodivého elementu, medzi ktorými vzniká potenciálový rozdiel (Hallovo napätie) U_H

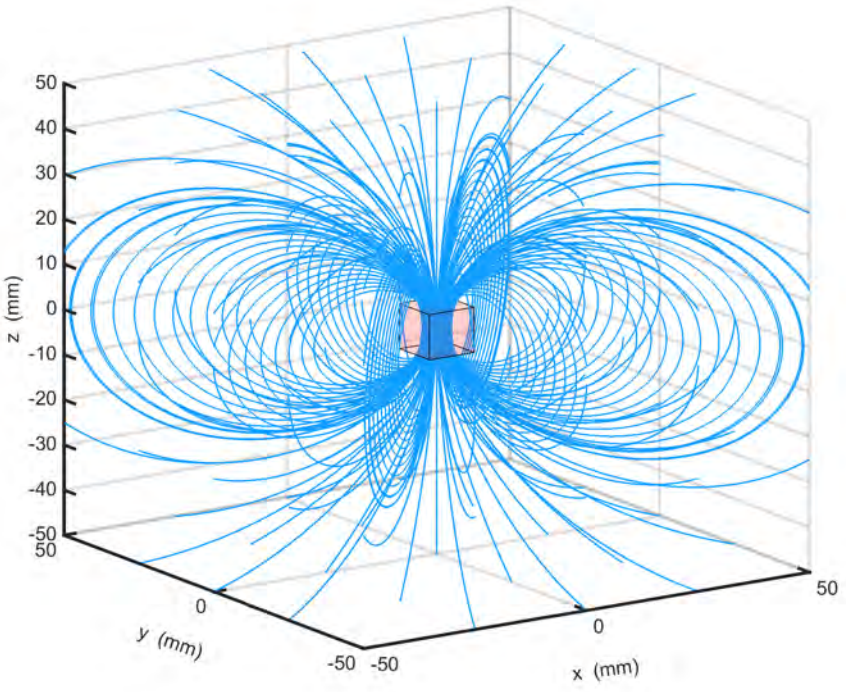
$$U_H = \frac{I_x B_z}{nec}, \tag{3.23}$$

kde n je hustota nosičov nábojov vo vodiči a e je náboj elektrónu. Hallovo napätie je teda lineárne závislé od veľkosti magnetickej indukcie, čo je vhodná detekčná charakteristika pre takýto senzor. Vyššie hodnoty Hallovho napätia zároveň získame pre menšiu hrúbku c vodiča a menšiu hustotu nosičov náboja n, preto je vhodnejší na konštrukciu senzora polovodič.

Vzhľadom na komplexnosť tvaru vektorového poľa magnetickej indukcie v okolí permanentného magnetu (magnetické indukčné čiary permanentného magnetu tvaru kocky sú zobrazené na obr. 3.19) je pri použití jednoduchého Hallovho senzoru dôležitý smer vektora magnetickej indukcie (kolmý na plochu vodivého elementu ako na obr. 3.18a), cieľom praktickej úlohy bude teda mapovanie magnetického poľa len v smere osi z (smer polarizácie magnetu) smerom od stredu magnetu, kde si vektor magnetickej indukcie zachováva smer kolmý na povrchu magnetu.



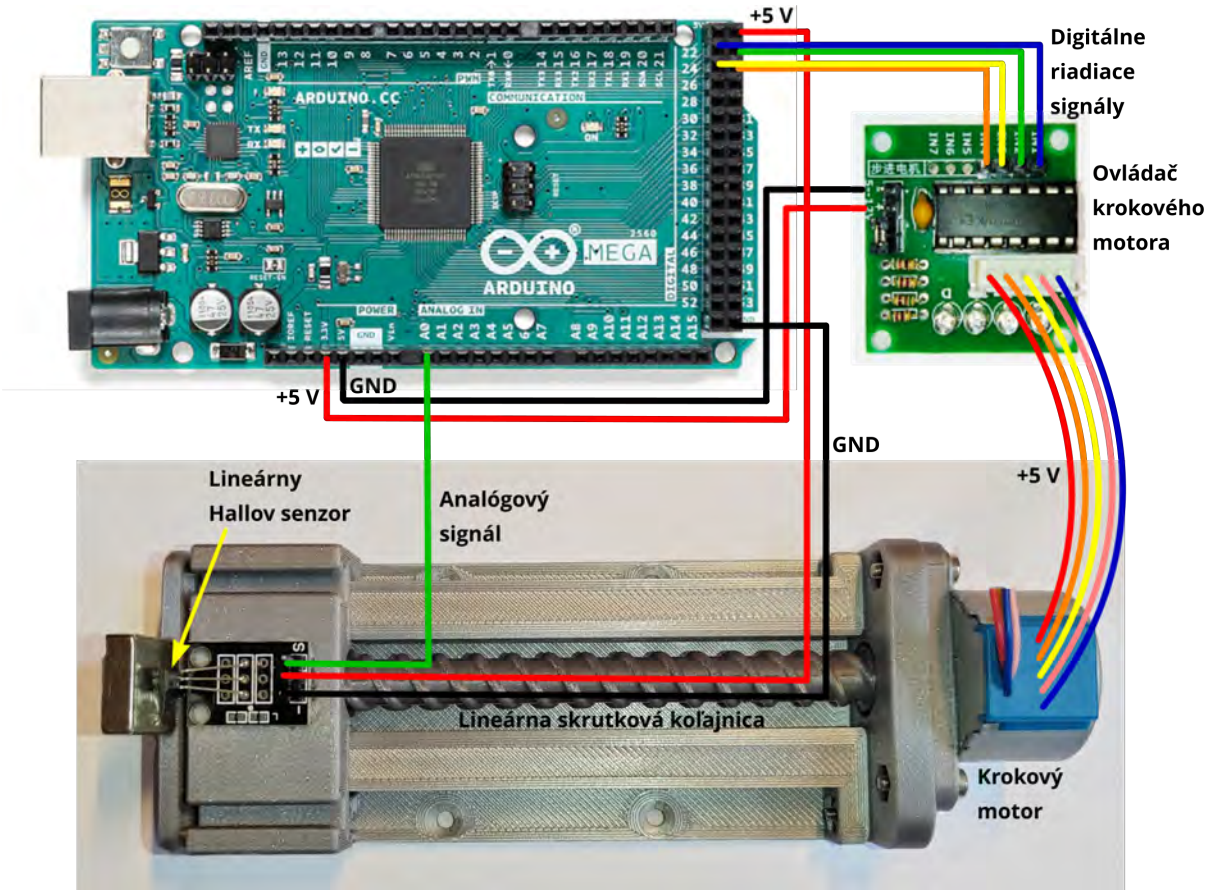
Obr. 3.18: (a) Zobrazenie fyzikálneho princípu Hallovho javu. (b) Lineárna prevodová charakteristika Hallovho senzora typu 49E. Zároveň je označený smer kladného a záporného smeru magnetickej indukcie voči púzdru senzora.



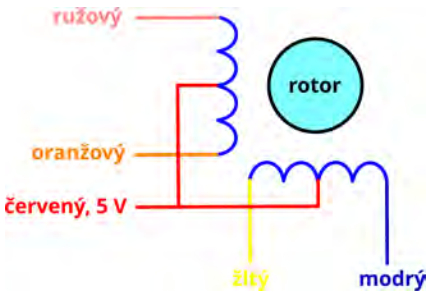
Obr. 3.19: Zobrazenie magnetických indukčných čiar permanentného magnetu v tvare kocky s rozmermi $10 \times 10 \times 10$ mm^3 polarizovaného v smere osi z.

Technická realizácia:

Zostava na riešenie praktickej úlohy mapovania magnetickej indukcie neodýmového permanentného magnetu na obr. 3.20 bude pozostávať z krokového motora, lineárneho skrutkového mechanizmu pre jednoosý pohyb a lineárneho Hallovho senzora, ktoré budú ovládané pomocou dosky *Arduino Mega2560 Rev3* z prostredia *LabVIEW*. Samotný jednoosý pohybový mechanizmus je možné vyrobiť 3D tlačou na základe voľne šíriteľného dizajnu *High Accuracy Dovetail Stepper Slide* [66], ktorý je aj kompatibilný s populárnym 4-fázovým krokovým motorom 28BYJ-48 s 5 V napájaním a ovládačom s tranzistorovým poľom ULN2003 dostupným od viacerých výrobcov [67]. Pri menšej záťaži je možné tento krokový motor napájať priamo z dosky *Arduino*. Zapojenie riadiacich elektromagnetov krokového motora je zobrazené na obr. 3.21, pričom na vykonanie jedného kroku otáčania proti smeru hodinových ručičiek vyšleme sériu pulzov podľa tab. 3.1, postupnosť pulzov je vizuálne zobrazovaná aj svetelnými diódami na ovládači motora. Jedna celá otáčka rotora (360°) pritom predstavuje 512 krokov. Prevodový pomer použitého lineárneho skrutkového mechanizmu je lineárny posun 12,8 mm na jednu celú otáčku, čo predstavuje 0,025 mm na jeden krok motora, alebo 76,8 mm na 6 celých otáčok rotora. Naše meranie vykonáme v rozsahu 0 – 80 mm, takže maximálny počet krokov bude nastavený na hodnotu 3200.



Obr. 3.20: Zostava na mapovanie magnetického poľa neodýmového permanentného magnetu v smere osi polarizácie z s označením zapojenia elektrických signálov.



Obr. 3.21: Pripojenie riadiacich elektromagnetov krokového motora 28BYJ-48 s farebným označením vodičov pripojených k ovládaču s tranzistorovým poľom ULN2003.

VODIČ \ PULZ	1	2	3	4	5	6	7	8
oranžový (4)	•	•						•
žltý (3)		•	•	•				
ružový (2)				•	•	•		
modrý (1)						•	•	•

Tabuľka 3.1: Pre vykonanie jedného kroku v otáčaní rotora krokového motora 28BYJ-48 je potrebné vyslať sériu ôsmich pulzov • (pulz 1 až 8) na jednotlivé riadiace elektromagnety.

Na pohyblivú platformu na lineárnom skrutkovom mechanizme upevníme lineárny Hallov senzor 49E na module typu KY-035, taktiež dostupný od viacerých výrobcov. Pre správnu orientáciu púzdra senzora voči smeru vektora magnetickej indukcie je potrebné ohnúť piny senzora o 90°. Modul má tri piny, napájanie 5 V, zem a analógový signálny výstup, na ktoré je priamo pripojený senzor 49E s integrovaným obvodom a detekčným elementom v púzdre typu TO-92S. Výstupné napätie senzora je 2,5 V pri nulovej indukcii magnetického poľa, jeho lineárna prevodová charakteristika je zobrazená na obr. 3.18b s typickou citlivosťou 19 mV/mT pri napájaní 5 V a maximálnou detegovanou magnetickou indukciou 120 mT. Pri meraní magnetickej indukcie v kladnom smere teda môžeme podľa obr. 3.18b odpočítať referenčnú hodnotu 2,5 V a priradiť rozsah 0 - 1,7 V magnetickej indukcii v rozsahu 0 - 120 mT. Analógový signálny výstup pripojíme na jeden z analógových vstupov dosky *Arduino* (obr. 3.20), ktorý pomocou 10-bitového analógovo-digitálneho prevodníka prenáša hodnoty napätia do počítača v digitálnej forme.

Pri zostavení riadiaceho programu využijeme modifikovanú architektúra stavového stroja s využitím reakcie VI na operácie s logickými ovládačmi. Na komunikáciu s doskou *Arduino* využijeme funkcie z balíka *NI LabVIEW LINX Toolkit* alebo *LabVIEW Hobyyist Toolkit*, podľa dostupnosti v použitom vydaní *LabVIEW*. Merací program by mal obsahovať nasledovné ovládacie prvky ako vzorový program zobrazený v kap. C.3:

- Numerický kontrolný ovládač *Go to position (mm)* s cieľovou polohou Hallovho senzora na lineárnom skrutkovom mechanizme. V konfiguračnom dialógovom okne ovládača *Go to position (mm)* je potrebné nastaviť minimálnu hodnotu 0, maximálnu hodnotu 80 a možný krok 0,025 ako je zobrazené na obr. C.12.
- Ovládač typu numerického poľa na zadanie adresy 4 digitálnych výstupov a numerický ovládač na zadanie adresy jedného analógového výstupu *Arduina*.
- Logické tlačidlá, ktoré spustia chod krokového motora k polohe 0 (tlačidlo *Go Home*), alebo k polohe definovanej v ovládači *Go to position (mm)* (tlačidlo *Go To Position*).
- Smer pohybu môžeme manuálne určiť vhodným ovládačom *Direction* na výber dvoch hodnôt, napr. typu *Menu Ring*.
- Logický prepínač *Move*, ktorý označuje, či sa má vykonávať pohyb motora.
- *XY Graph* na zobrazenie meranej závislosti, spolu s logickými tlačidlami *Reset Graph* a *Export to Excel*.
- Logické tlačidlo *STOP* na zastavenie programu.

Samotný postup merania bude prebiehať podľa nasledujúceho algoritmu:

1. Inicializácia komunikácie s doskou *Arduino* pomocou funkcie *Open.vi*. Priradený sériový port je zvyčajne COM3 alebo COM4, ak nie sú k počítaču pripojené iné USB zariadenia. Následne v cykle *While Loop* sledujeme akciu ovládacích tlačidiel.
2. V každom kroku cyklu sa zobrazuje aktuálna poloha v mm, počet krokov motora, počet otočení rotora a uhol otočenia rotora do vhodných indikátorov.
3. V každom kroku cyklu sa načíta hodnota napätia z analógového vstupu pomocou funkcie *Analog Read.vi* a zobrazuje sa do vhodného indikátora.
4. Po stlačení tlačidla *Go To Position* určíme smer pohybu a zmeníme hodnotu logického prepínača *Move* na TRUE.
5. Po stlačení tlačidla *Go Home* zapíšeme hodnotu 0 do ovládača *Go to position (mm)* a zmeníme hodnotu logického prepínača *Move*, na TRUE.
6. Po zopnutí spínača *Move* na hodnotu TRUE, alebo po jeho aktivácii tlačidlami *Go Home* alebo *Go To Position*, sa krokový motor môže začať otáčať. V prvom kroku sa overí nastavená poloha *Go to position (mm)* s aktuálnou polohou a hraničnými polohami. Ak aktuálna poloha je rovná jednej z nich, motor nevykoná pohyb, zmeníme stav prepínača *Move* na FALSE. Ak sme na hraničných polohách, zároveň zmeníme smer pohybu v ovládači *Direction* na opačný. Jeden krok motora sa vykoná pomocou SubVI *MotorStep.vi*, len ak požadovaná poloha nie je rovná ani aktuálnej, ani hraničným polohám, následne sa aktualizuje poloha o vykonaný krok.

7. Na *XY Graph* vykresľujeme polohu a namerané hodnoty signálu z Hallovho senzora len v prípade, že prepínač *Move* je v stave TRUE.
8. Po stlačení tlačidla *Reset Graph* sa zmaže obsah grafu.
9. Po stlačení tlačidla *Export to Excel* využijeme *Invoke Node* na zápis údajov z grafu do *Excelu*.
10. V každom kroku cyklu môžeme vyčkať krátky moment určený približne ako počet krokov motora za sekundu pomocou vhodného numerického ovládača.
11. Ak nie je stlačené tlačidlo *STOP*, cyklus sa vracia na bod 2 a reaguje na akciu ovládacích tlačidiel.
12. Ukončenie cyklu a komunikácie s doskou *Arduino* pomocou funkcie *Close.vi* po stlačení ovládacieho tlačidla *STOP*.

Časti vzorového VI a ich popis sú zhrnuté na obr. C.9 až obr. C.12 v kap. C.3.

A ŠPECIFICKÉ NASTAVENIA LABVIEW

A.1 ŠPECIFIKÁTORY FORMÁTU ČÍSELNÝCH HODNÔT

Syntax špecifikátorov formátu, ktoré zadávame na vstupnom termináli *format specifier* funkcií pri zápise číselných hodnôt do textových súborov vo formáte ASCII alebo pri použití funkcií na prevod číselných hodnôt do reťazca (napr. *Array To Spreadsheet String* alebo *Format Into String*) je nasledovná:

% [$] [−] [+] [#] [^] [0] [*Width*] [.*Precision*||_*SignificantDigits*] [{*Unit*}]
[< *Embedded information* >] *Conversion Code*,

pričom hranaté zátvorky [ ] uzatvárajú nepovinné prvky. Význam jednotlivých formátovacích znakov je nasledovný:

- % - označuje začiatok formátovacieho reťazca;
- $ - označuje poradie v akom sa majú zobraziť viaceré formátované premenné, napr. %2$*s*;
- − - zarovnať výsledný reťazec doľava;
- + - zobraziť znamienko vždy, aj keď je číslo kladné;
- # - odstrániť nadbytočné nuly na konci čísla (*angl. trailing zeros*)
- ^ - inžinierske zobrazenie, exponent je vždy v násobkoch 3;
- 0 - pridaj naľavo nuly namiesto medzier na doplnenie čísla, aby malo definovaný počet znakov;
- *Width* - počet znakov, musí byť číslo väčšie ako nula;
- .*Precision* - presnosť, musí byť číslo väčšie alebo rovné nule;
- _*SignificantDigits* - počet platných číslic, musí byť číslo väčšie alebo rovné nule;
- *Unit* - pri číslach s jednotkami udáva konverziu jednotiek;
- *Conversion Code* - spôsob zápisu, typ konverzie;
- ⟨*Embedded information*⟩ - kódy na formátovanie času.

Príklady špecifikátorov formátu sú uvedené v tab. A.1.

FORMÁT	HODNOTA	ŠPECIFIKÁTOR FORMÁTU	VÝSLEDOK
Automatický %g	12.00	%#g	12 [1]
	12000000	%#g	1.2E+7
Decimálny %d	12.67	%d	13
S plávajúcou desatinnou čiarkou %f	12.67	%5.3f	12.7
	12.67 N	%5.3{mN}f	12670.000 mN [2]
Vedecký formát %e	12.67	%.3e	1.267E+1
	12.67	%^.3e	12.670E+0
SI formát %p	12000000	%.2p	12.00M
Relatívny čas %t	91.80	%.2t	01:31.80 [3]
Absolútny čas %T	00:00:00.000 AM 1/1/2001	%⟨%.3*X*%*x*⟩*T*	12:00:00.000 AM 01/01/2001 [4]

Tabuľka A.1: Výber syntaxe špecifikátorov formátu pre prevod číselných hodnôt do reťazca.

[1] Znak *g* vyberie vedecký zápis alebo zápis s pohyblivou desatinnou čiarkou podľa vstupu.

[2] Konverzia jednotiek pre zobrazenie.

[3] Formát uplynutého času v podobe celých týždňov %*W*, dní %*D*, hodín %*H*, minút %*M*, sekúnd %*S* a zlomkov sekúnd %*u*. Je možné individualizovať, napr. pomocou špecifikátora %⟨*Hod* : %*H Min* : %*M Sek* : %*S*⟩*t* získame konverziu Hod:00 Min:01 Sek:31.

[4] Formát, vrátane interpunkcie, sa mení v závislosti od regionálnych nastavení počítača, je možné zobraziť aj skratku alebo názov dňa (%*a*, %*A*) alebo mesiaca (%*m*, %*M*).

A.2 AKTIVÁCIA LOGICKÝCH OVLÁDAČOV

Logické ovládače majú mechanické akcie, ktoré riadia, ako aktivácia myšou ovplyvní hodnotu ovládacieho prvku. Mechanická akcia umožňuje ovládaciemu prvku napodobniť určité fyzické spínače (napr. vypínač svetla alebo tlačidlo núdzového zastavenia). Existuje šesť typov mechanických akcií zhrnutých v tab. A.2.

SPRÁVANIE SA	MECHANICKÁ AKCIA	POPIS
Switch when pressed (prepni pri stlačení)		Hodnota ovládacieho prvku sa zmení pri každom kliknutí. Je to podobné ako pri vypínači svetla.
Switch when released (prepni pri uvoľnení)		Hodnota ovládacieho prvku sa zmení až po uvoľnení tlačidla myši v rámci hranice ovládacieho prvku.
Switch until released (prepni až do uvoľnenia)		Hodnota ovládacieho prvku je zmenená len dovtedy, kým je tlačidlo myši stlačené. Po uvoľnení tlačidla myši sa ovládací prvok vráti na svoju predvolenú hodnotu.
Latch when pressed (zopni pri stlačení)		Hodnota ovládacieho prvku sa aktualizuje po stlačení tlačidla myši. Keď VI prečíta hodnotu ovládacieho prvku, vráti sa na svoju predvolenú hodnotu.
Latch when released (zopni pri uvoľnení)		Hodnota ovládacieho prvku sa aktualizuje po uvoľnení tlačidla myši v rámci hranice ovládacieho prvku. Keď VI prečíta hodnotu ovládacieho prvku, vráti sa na svoju predvolenú hodnotu.
Latch until released (zopni až do uvoľnenia)		Hodnota ovládacieho prvku sa aktualizuje, kým je stlačené tlačidlo myši alebo kým VI neprečíta hodnotu, podľa toho, čo nastane skôr.

Tabuľka A.2: Mechanické akcie logických ovládačov.

B ARDUINO

B.1 TECHNICKÉ ŠPECIFIKÁCIE ARDUINO MEGA2560

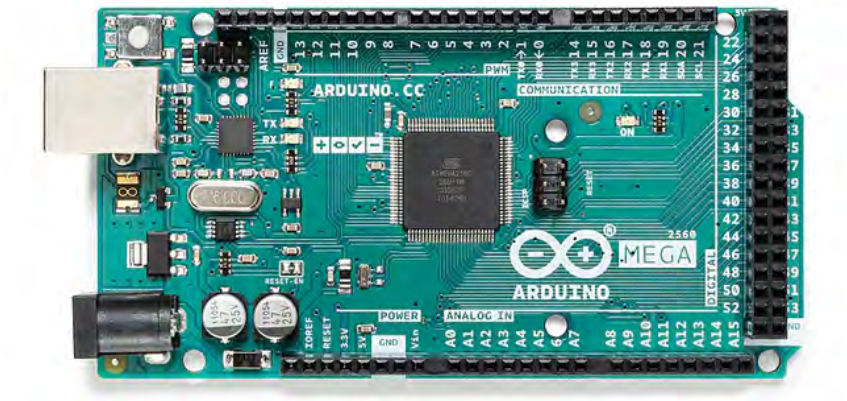
Arduino Mega2560 Rev3 [68] na obr. B.1a je mikrokontrolér založený na čipe *ATmega2560* [69]. Má 54 digitálnych vstupných/výstupných pinov (*DIGITAL*, z ktorých piny 2-13 možno použiť ako PWM výstupy), 16 analógových vstupných pinov (*ANALOG IN*, piny A0-A15), 4 hardvérové sériové porty Universal Asynchronous Receiver/Transmitter (UART), 16 MHz kryštálový oscilátor, 256 kB flash pamäte, pripojenie USB, napájací konektor, programovacie pripojenie In-Circuit Serial Programming (ICSP) a tlačidlo *reset*. Model *Mega2560 Rev3* je kompatibilný s väčšinou rozširujúcich dosiek určených pre model *UNO* [70], je však určený pre aplikácie a projekty, ktoré vyžadujú veľký počet vstupných a výstupných pinov a prípady použitia, ktoré potrebujú vyšší výpočtový výkon.

Na trhu je možné nájsť aj presné kópie originálneho *Arduino Mega2560 Rev3* s potlačou originálneho loga ako na obr. B.1b. Takisto sú dostupné aj neoriginálne varianty *Arduino UNO Rev3*, niektoré dokonca s dvoma prepínateľnými mikročipmi (ATmega238P [71] zhodný s originálnym *UNO Rev3* a ESP8266 [72]), väčšou flash pamäťou a podporou bezdrôtového internetu Wi-Fi, ako je výrobok spoločnosti *RobotDyn* [73] (obr. B.1b). Najnovšie verzie *UNO Rev4* už majú podporu bezdrôtového internetu a využívajú mikročip *Renesas RA4M1* (*Arm® Cortex®-M4*) [74], avšak *LabVIEW Hobbyist Toolkit* ich v roku 2024 ešte nepodporuje.

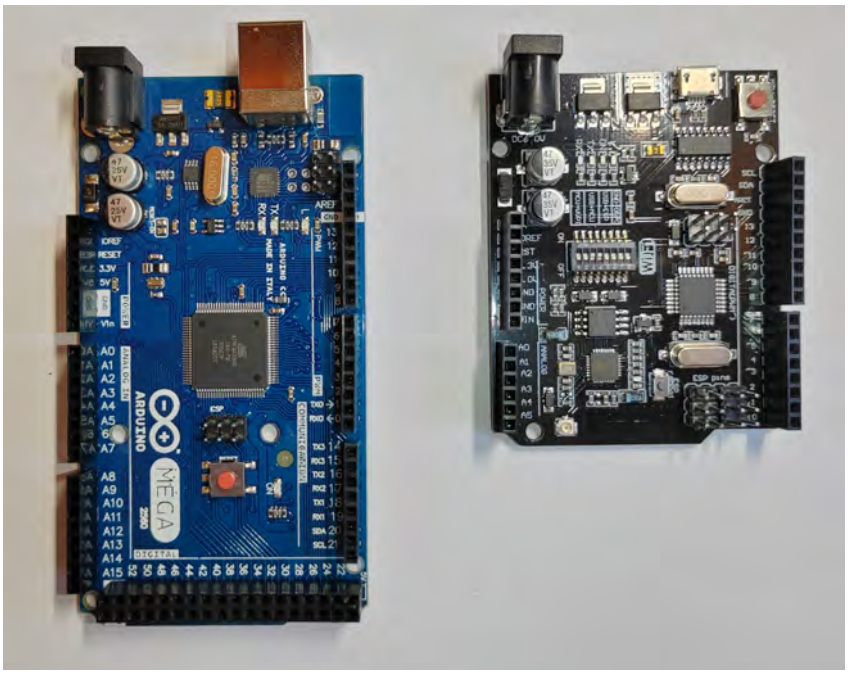
Na spustenie stačí *Arduino* pripojiť k počítaču pomocou kábla USB. Napájanie je možné priamo cez USB alebo cez pin označený *VIN* na doske, v prípade vyšších energetických nárokov pomocou AC-DC adaptéra s napätím 7-12 V. Niektoré z pinov na doske je možné použiť spolu s pinom *GND* (zem) na jednosmerné 3,3 V alebo 5 V napájanie senzorov, aktuátorov, motorov alebo rozširujúcich dosiek.

Digitálne piny predstavujú tzv. univerzálny vstup/výstup (General-Purpose Input/Output (GPIO)). GPIO spracováva prichádzajúce aj odchádzajúce digitálne signály. Ako vstupný port sa môže používať na prenos signálov *ON*/*OFF* prijatých zo spínačov alebo digitálnych údajov prijatých zo snímačov do mikroprocesora. Ako výstupný port sa môže použiť na riadenie vonkajších operácií na základe pokynov mikroprocesora a výsledkov výpočtov, napr. na riadenie LED displeja na základe výsledkov výpočtov alebo na výstupné signály pohonu motora. Analógové piny sú určené na privedenie napäťových signálov zo senzorov, ktoré sú digitalizované pomocou analógovo-číselného prevodníka s 10-bitovým rozlíšením (1024 hodnôt) v rozsahu 0-5 V.

Podrobnejšie označenie pinov a súčastí *Arduino Mega2560 Rev3* je možné nájsť v dokumentácii výrobcu [75], spoločnosti *Arduino S.r.l. - Partita IVA*.



(a)



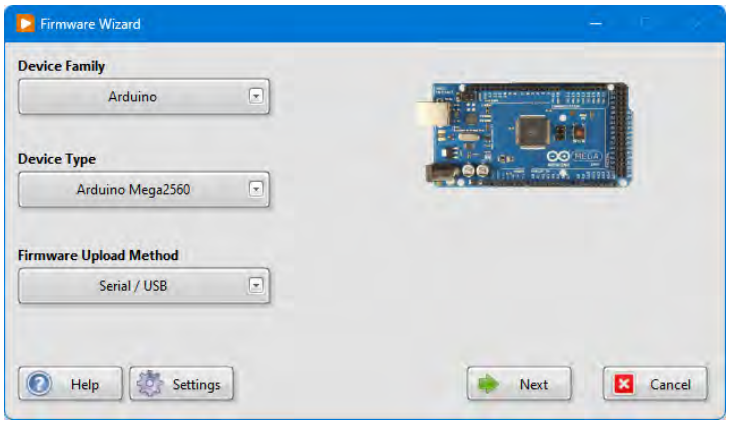
(b)

Obr. B.1: Pohľad na dosku *Arduino Mega2560 Rev3* (a), klon *Mega2560 Rev3* s červeným mikrospínačom *reset* (vľavo) a klon modelu *UNO Rev3* od spoločnosti *RobotDyn* (vpravo) (b).

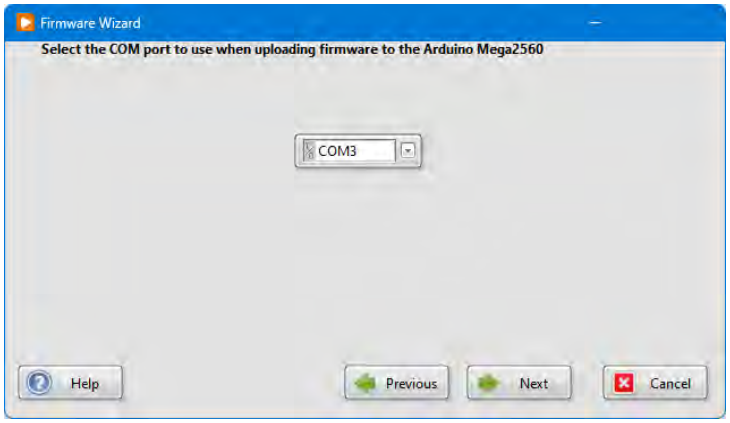
B.2 KONFIGURAČNÝ SPRIEVODCA PRE ARDUINO

Konfiguračného sprievodcu pre *LabVIEW Hobbyist Toolkit* je možné spustiť z ponuky *Tools ▸Hobbyist ▸Firmware Wizard...* (konfiguračný sprievodca *NI LabVIEW LINX Toolkit* je takmer identický). Postupné kroky konfigurácie sú nasledovné:

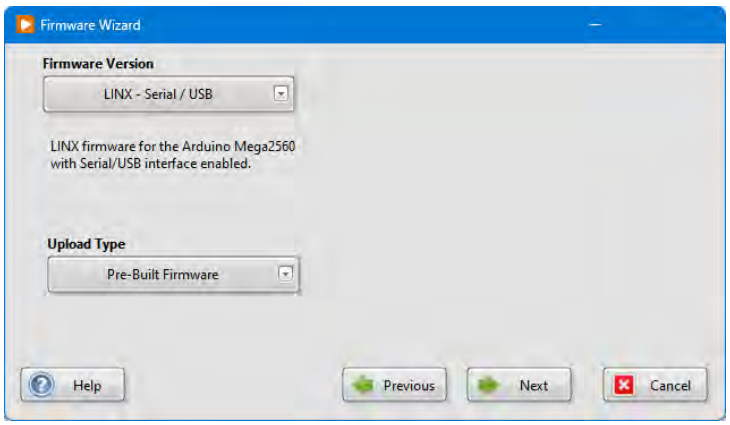
- Výber modelu dosky, ktorý bude používaný, napr. *Arduino Mega2560*, s výberom spôsobu na nahranie firmvéru *Serial/USB* zobrazený na obr. B.2a. V tejto fáze je potrebné mať vybraný model dosky už pripojený k USB portu počítača.
- Detekcia priradeného sériového portu pre pripojené zariadenie, zvyčajne to je *COM3* alebo *COM4*, no môže to byť aj port s vyšším číselným označením, obr. B.2b.
- Výber verzie firmvéru (*Firmware Version*) - *LINX - Serial/USB* a nahrávaného typu (*Upload Type*) - *Pre-Built Firmware*, obr. B.2c.
- Nahranie firmvéru, trvá niekoľko desiatok sekúnd.
- Na záverečnej obrazovke (obr. B.2d) je možnosť otvoriť príklad na vykonanie jednoduchého testu funkčnosti ovládania dosky pod názvom *Blink (Simple).vi*, ktorý umožňuje zapnúť alebo vypnúť notifikačnú svetelnú diódu na doske *Arduino* pomocou logického ovládača, obr. B.3.



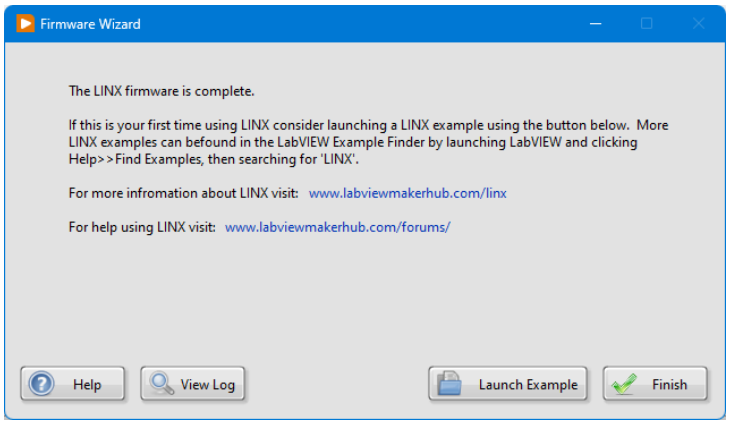
(a)



(b)

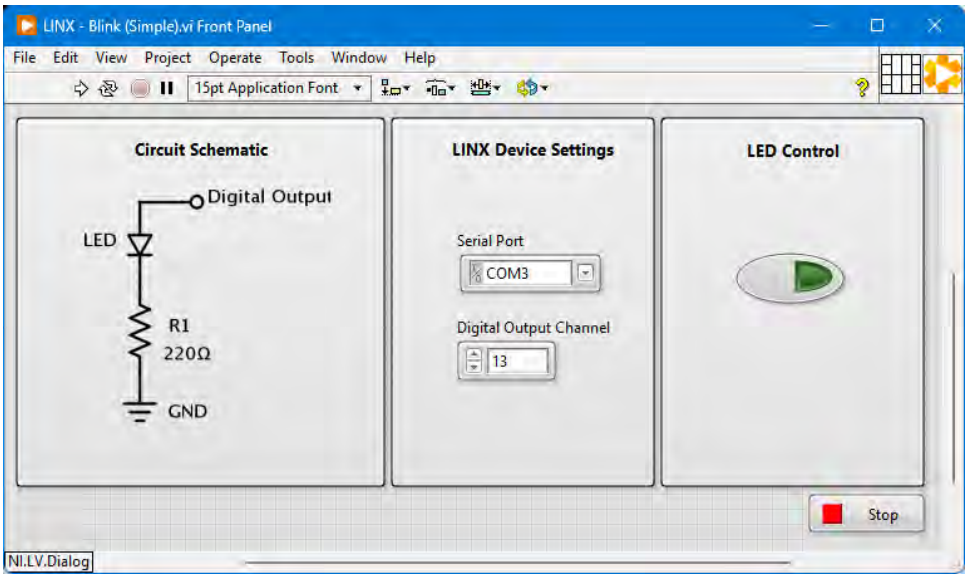


(c)

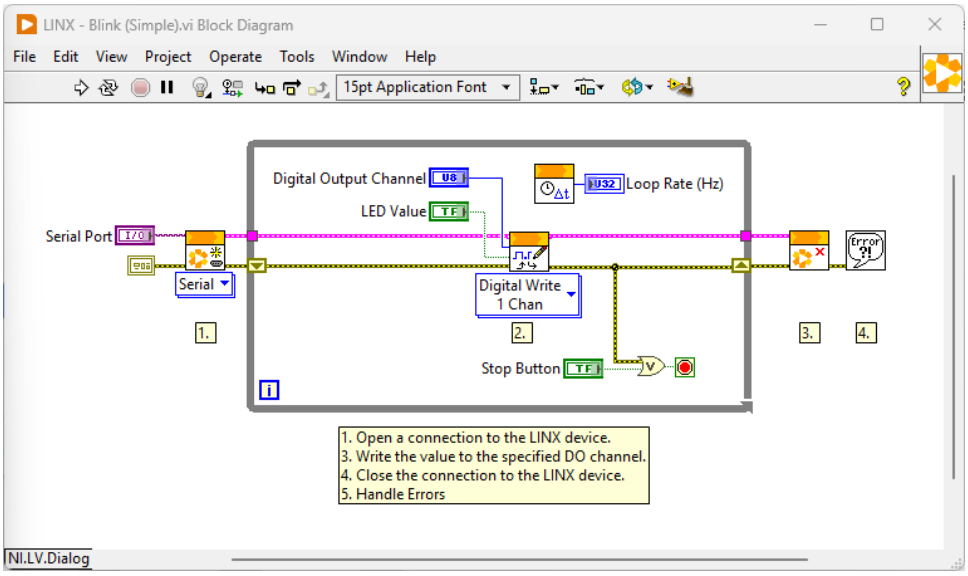


(d)

Obr. B.2: Konfiguračný sprievodca pre *LabVIEW Hobbyist Toolkit*.



(a)



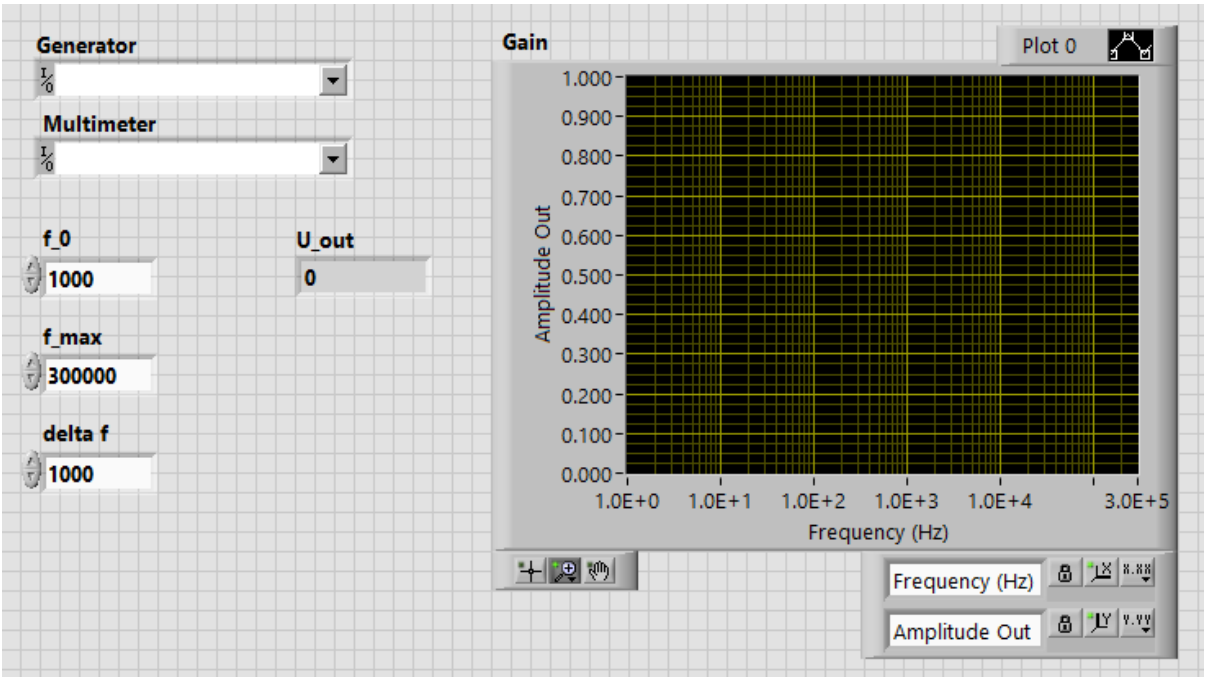
(b)

Obr. B.3: *Predný panel* (a) a *Blokový diagram* (b) *Blink (Simple).vi* pre overenie funkčnosti ovládania dosky *Arduino*.

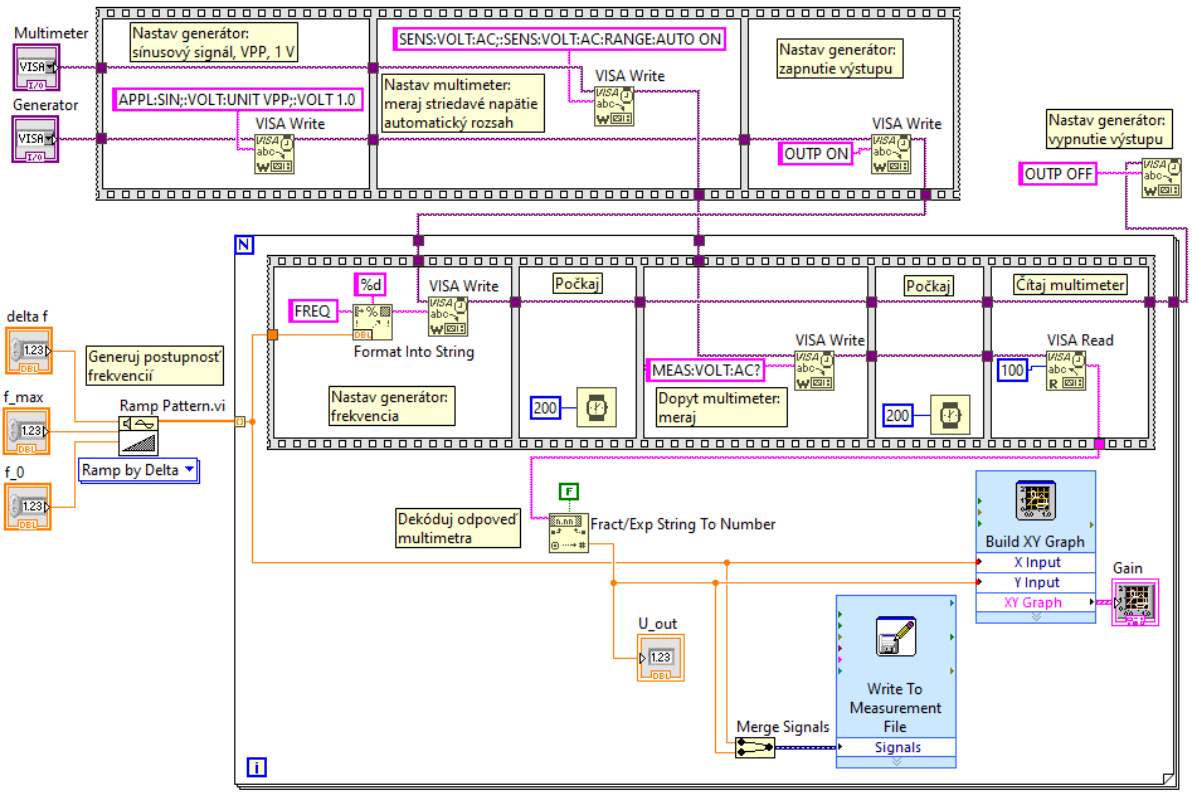
C PRÍKLADY VI PRE RIEŠENIE PRAKTICKÝCH ÚLOH

C.1 PROGRAM RC.VI

Vzorový VI *RC.vi* pre meranie prenosovej charakteristiky RC filtra typu dolná priepusť s jednoduchou architektúru: príprava merania, cyklus merania, a ukončenie merania. Pripojené sú funkčný generátor *Rigol* DG1022Z na vstupné svorky filtra a multimeter *Picotest* M3500A na výstupné svorky filtra. Pri komunikácii s prístrojmi sa využíva protokol SCPI pomocou VISA API.



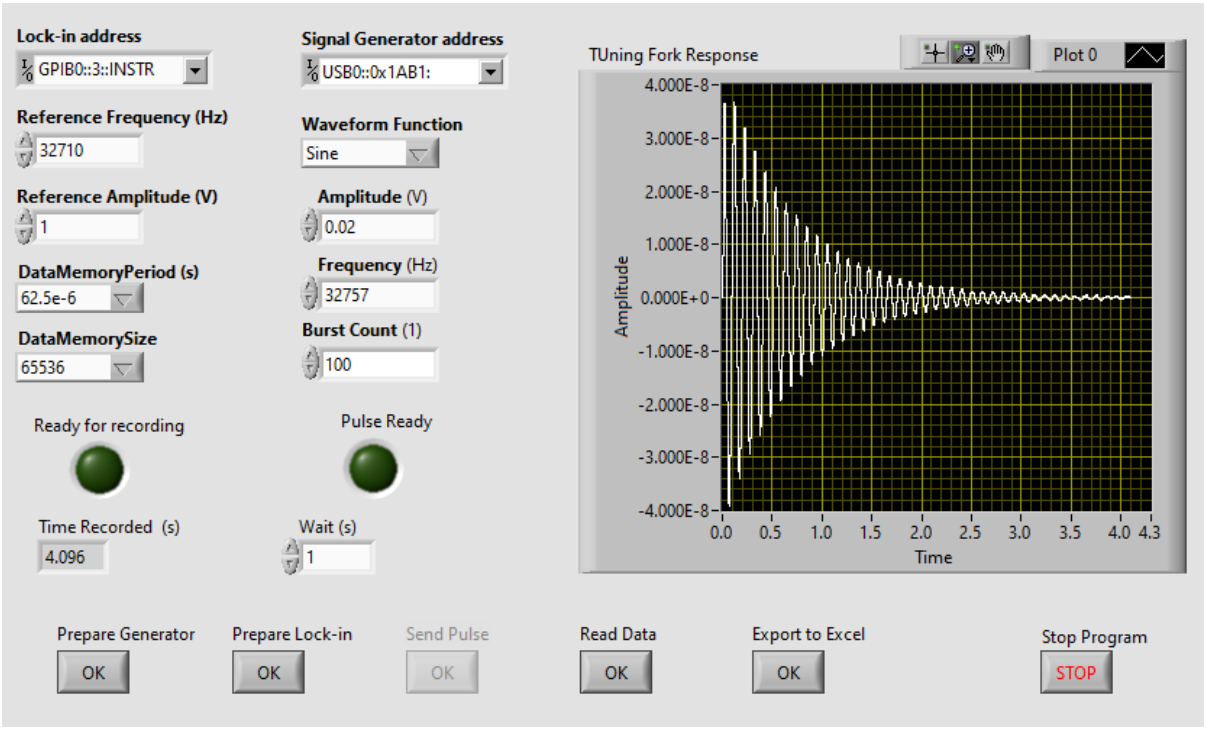
Obr. C.1: *Predný panel RC.vi* pre meranie prenosovej charakteristiky RC filtra typu dolná priepusť.



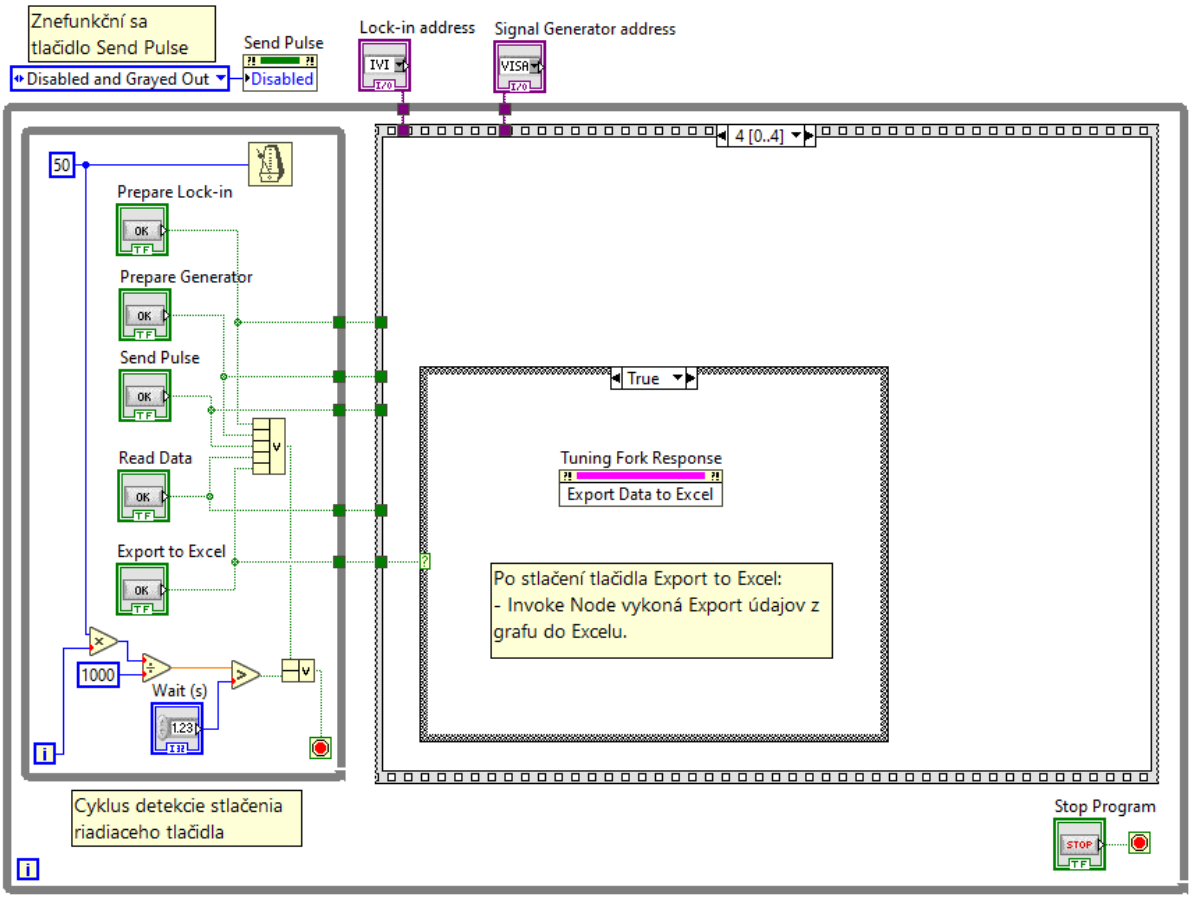
Obr. C.2: *Blokový diagram RC.vi* pre meranie prenosovej charakteristiky RC filtra typu dolná priepusť. VI má jednoduchú architektúru: príprava merania, cyklus merania, ukončenie merania. Pre jednoduchosť nie sú spracovávané chybové hlásenia funkcií *VISA Write* a *VISA Read*.

C.2 PROGRAM LADICKA.VI

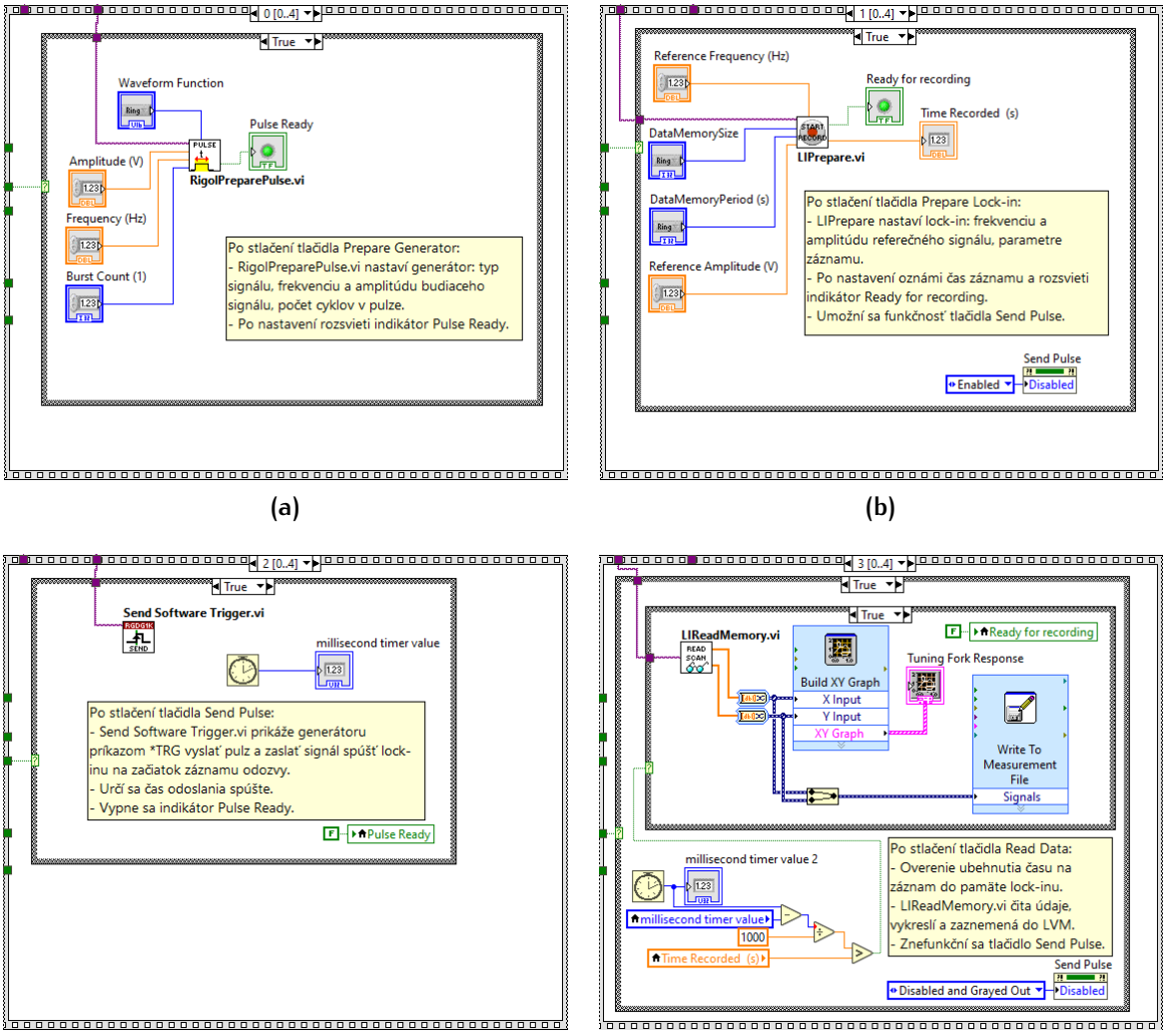
Vzorový VI *Ladicka.vi* pre meranie odozvy kremennej ladičky a určenie jej vlastnej rezonančnej frekvencie kmitov heterodynnou metódou s pulznou excitáciou kmitov ladičky. VI má modifikovanú architektúru stavového stroja s reakciou VI na operácie s logickými ovládačmi (ovládacie tlačidlá) a na základe interakcie vykoná rôzne akcie. Pripojené sú funkčný generátor *Rigol* DG1022Z a lock-in zosilňovač *NF Corp.* LI5460. K zjednodušeniu *Blokového diagramu* sa využívajú SubVI *RigolPreparePulse.vi*, *Send Software Trigger.vi*, *LIPrepare.vi* a *LIReadMemory.vi*. Komunikácia s funkčným generátorom prebieha pomocou protokolu SCPI a s lock-in zosilňovačom pomocou IVI ovládača poskytnutého výrobcom.



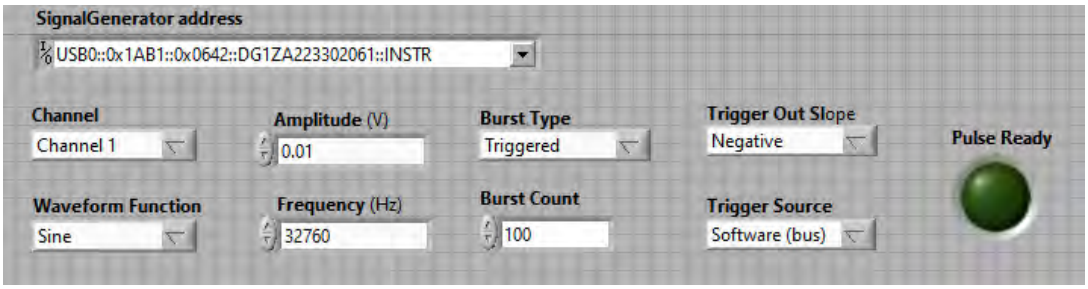
Obr. C.3: *Predný panel Ladicka.vi* pre meranie odozvy kremennej ladičky a určenie jej vlastnej rezonančnej frekvencie kmitov heterodynnou metódou s pulznou excitáciou kmitov ladičky.



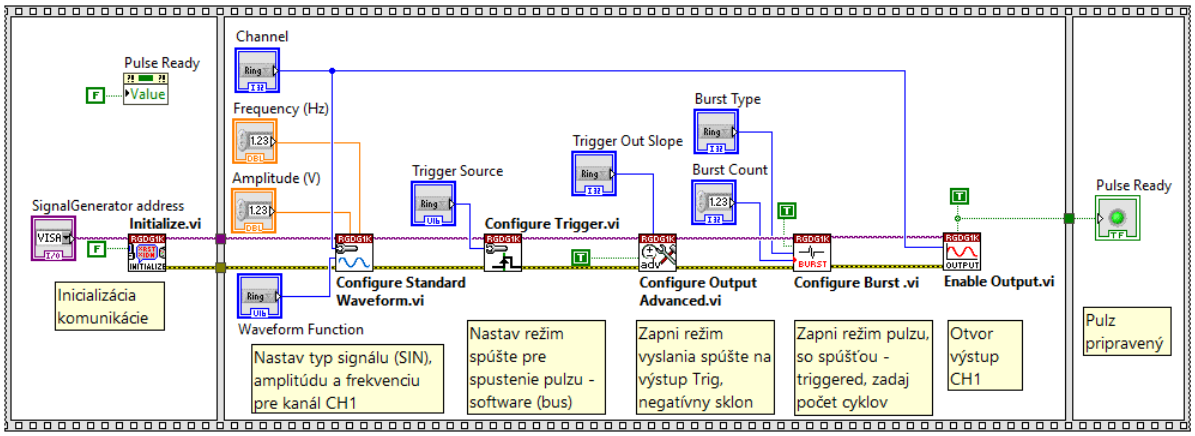
Obr. C.4: *Blokový diagram Ladicka.vi* pre meranie odozvy kremennej ladičky a určenie jej vlastnej rezonančnej frekvencie heterodynnou metódou s pulznou excitáciou kmitov ladičky. VI reaguje na stlačenie ovládacích tlačidiel a na základe interakcie vykoná rôzne akcie. V zobrazenej časti *Blokového diagramu* umožní export údajov zobrazených na grafe do Excelu.



Obr. C.5: Časti *Blokového diagramu* SubVI *RigolPreparePulse.vi* vykonávajúce nasledujúce operácie: (a) nastaví parametre pulzu pomocou SubVI *RigolPreparePulse.vi* a rozsvieti indikátor *Pulse Ready*, (b) pripraví záznam signálu do pamäte lock-in zosilňovača pomocou SubVI *LIPrepare.vi*, oznámi určený čas záznamu, rozsvieti indikátor *Ready for recording* a umožní funkčnosť tlačidla *Send Pulse*, (c) zašle signál na spustenie budiaceho pulzu pomocou SubVI *Send Software Trigger.vi*, spustí sa automaticky záznam odozvy ladičky, určí čas spustenie pulzu a vypne indikátor *Pulse Ready*, (d) ak ubehol dostatočný čas na záznam údajov, načíta ich z pamäte lock-in zosilňovača pomocou SubVI *LIReadMemory.vi*, vykreslí údaje na graf, uloží do súboru LVM) a znefunkční tlačidlo *Send Pulse*.

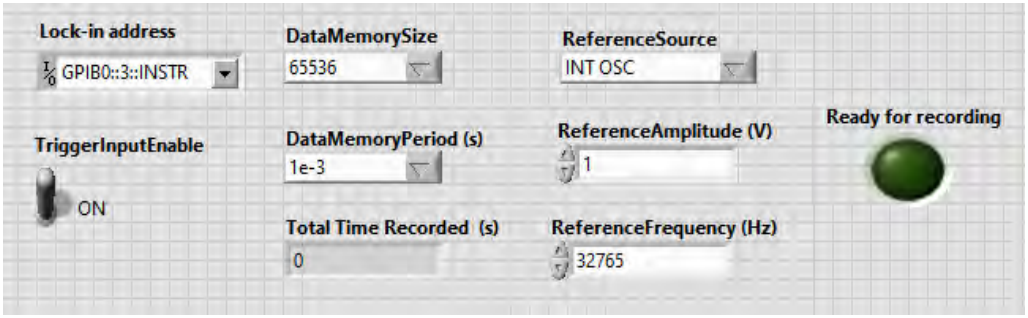


(a)

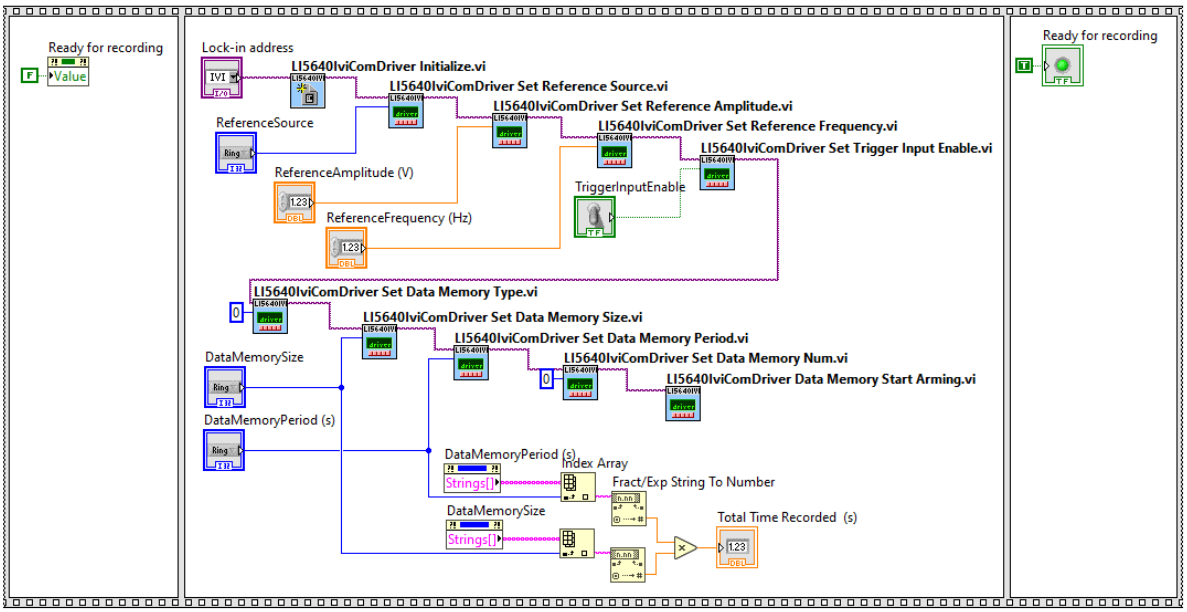


(b)

Obr. C.6: SubVI *RigolPreparePulse.vi* pripraví generátor na zaslanie budiaceho pulzu do kremennej ladičky. Postupne sú nastavené parametre pre kanál 1 (*Channel 1*): typ signálu (*Waveform Function - Sine*), amplitúda a frekvencia signálu (*Amplitude*, *Frequency*), nastavenie režimu spúšte pre spustenie pulzu (*Trigger Source - Software (bus)*, teda po zaslaní príkazu `*TRG`), zapnutie režime vyslania hardvérovej spúšte na výstup `Trig` s negatívnym sklonom (*Trigger Out Slope - Negative*, teda pri ukončení pulzu), zapnutie režimu pulzu (*Burst*) so spúšťou a zadanie počtu cyklov (*Burst Count*).

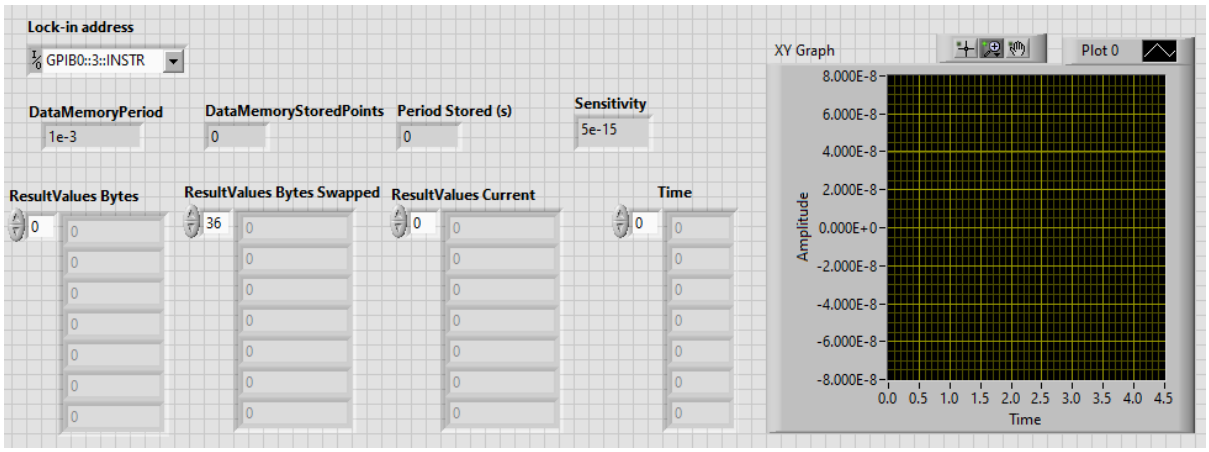


(a)

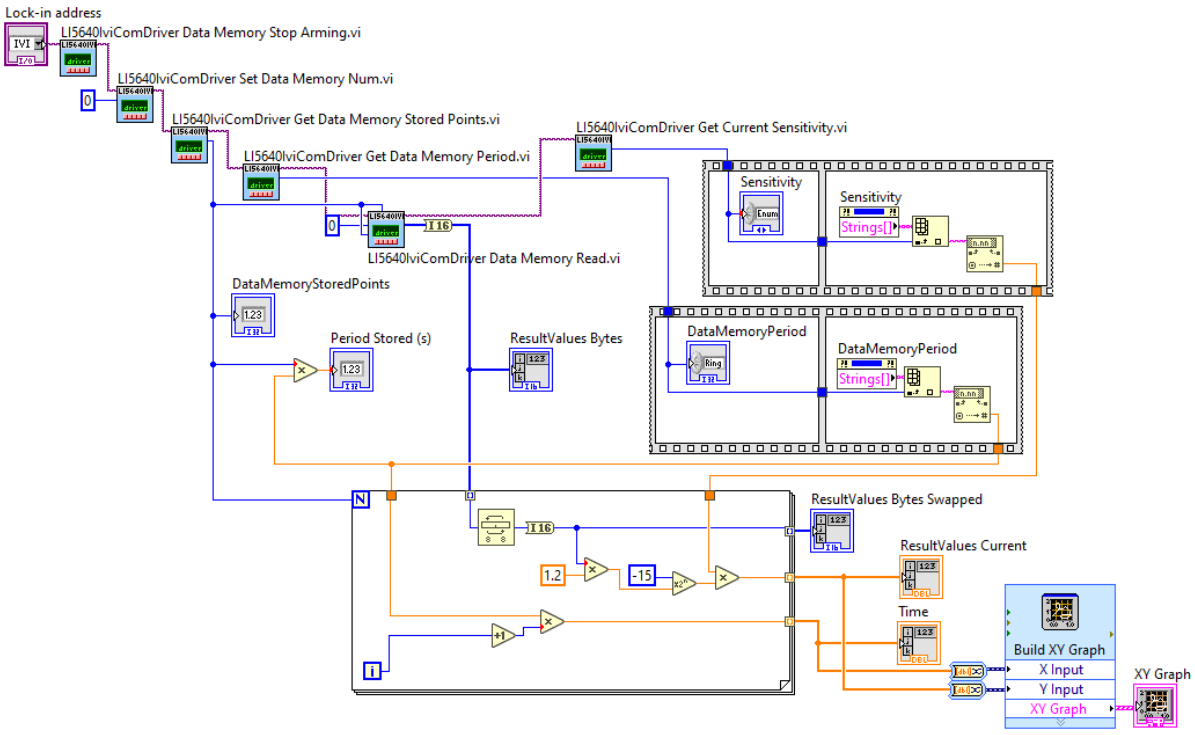


(b)

Obr. C.7: SubVI *LIPrepare.vi* pripraví lock-in zosilňovač na záznam odozvy kremennej ladičky. Po inicializácii komunikácie sú postupne nastavené parametre: zdroj referenčného signálu (*ReferenceSource* - *INT OSC*, interný oscilátor), amplitúda referenčného signálu (*ReferenceAmplitude*), frekvencia referenčného signálu (*ReferenceFrequency*), amplitúda referenčného signálu (*ReferenceAmplitude*), umožni vstup `EXT TRIG` (ovládač *TriggerInputEnable*), typ zaznamenávaných údajov (*Data Memory Type* - 0, teda reálna zložka detegovaného signálu), počet hodnôt v pamäti (*DataMemorySize*), časový rozostup bodov v pamäti (*DataMemoryPeriod*), blok pamäte (*Data Memory Num* - 0) a spustenie armingu (*Data Memory Start Arming*, zariadenie čaká na spúšť). Nakoniec sa vypočíta čas, ktorý bude potrebný na záznam (*Total Time Recorded*). Využité funkcie sú z knižnice ovládačov typu IVI poskytnutej výrobcom lock-in zosilňovača.



(a)

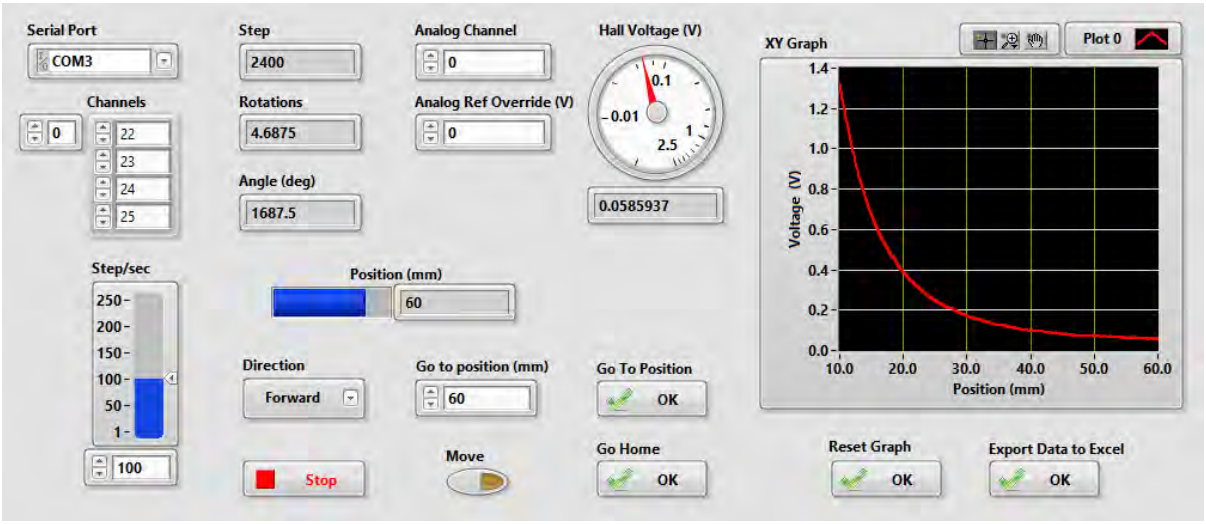


(b)

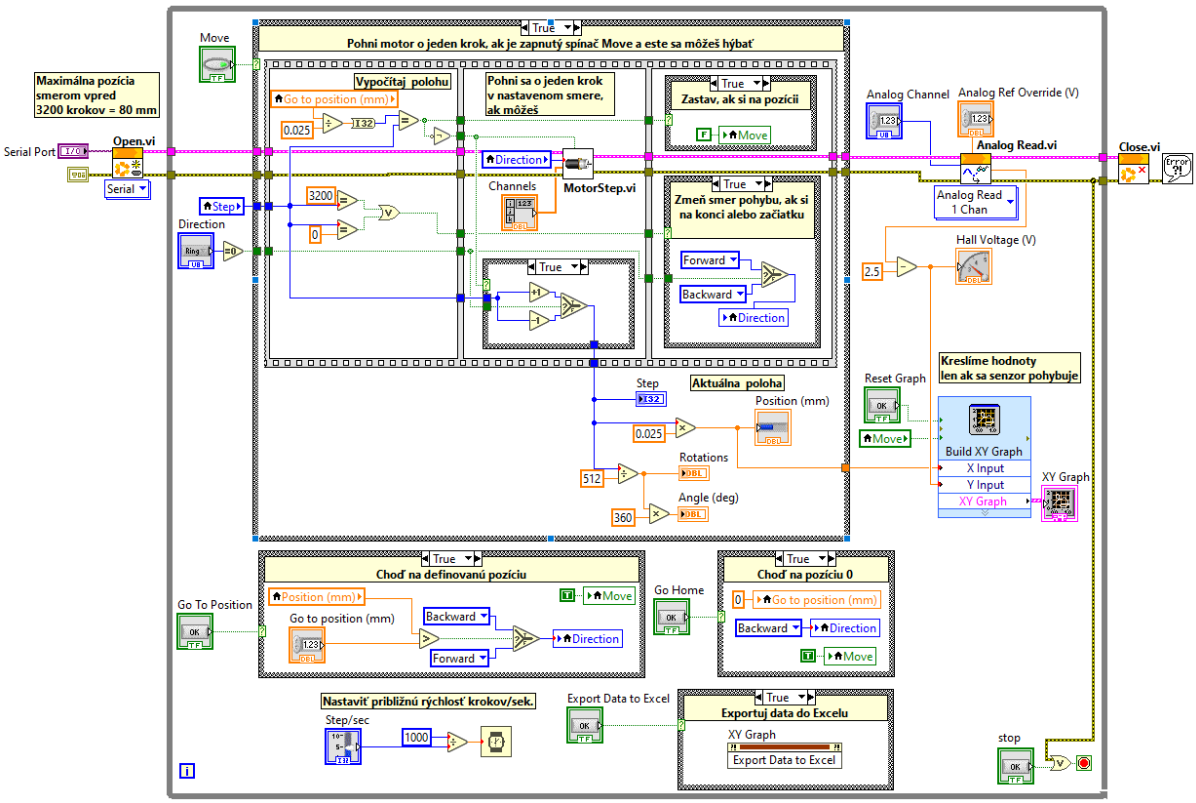
Obr. C.8: SubVI *LIReadMemory.vi* načíta zaznamenané údaje z pamäte lock-in zosilňovača. Po inicializácii komunikácie a ukončení armingu (zastavenie ďalšieho možného záznamu a prepísania existujúcich údajov) sú postupne zistené parametre záznamu: počet zaznamenaných hodnôt v pamäti (*DataMemoryStoredPoints*), časový rozostup bodov v pamäti (*DataMemoryPeriod*) a aktuálne nastavenie rozsahu merania (*Sensitivity*). Pomocou funkcie (*Data Memory Read*) sa načítajú údaje, ktoré sa so získanými parametrami využijú na zápis časového poľa (*Time*), poľa hodnôt nameranej prúdovej odozvy ladičky (*ResultValues Current*) a dĺžky záznamu (*Period Stored*).

C.3 PROGRAM PROFILPOLA.VI

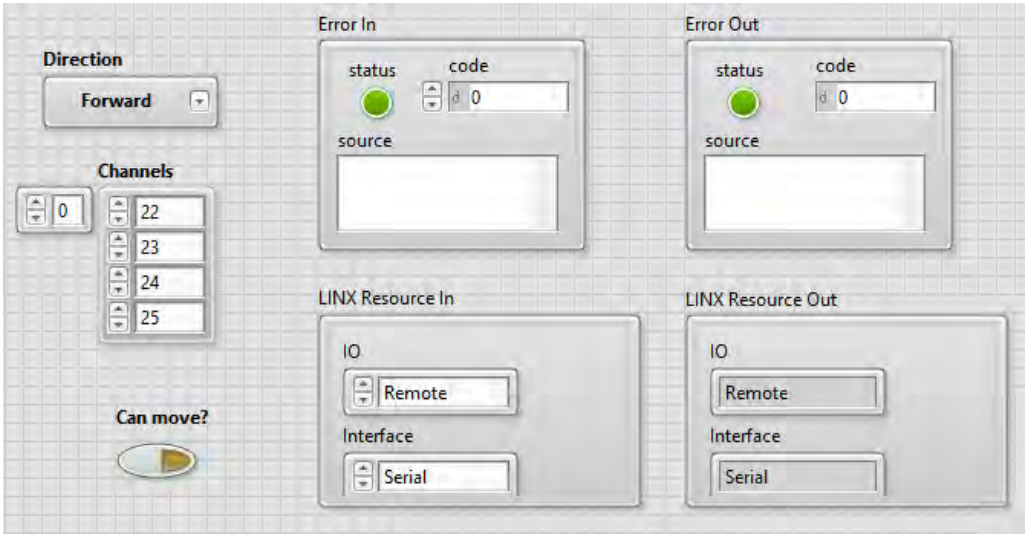
Vzorový VI *ProfilPola.vi* pre mapovanie magnetickej indukcie neodýmového permanentného magnetu pomocou zostavy na obr. 3.20, ktorá pozostáva zo 4-fázového krokového motora 28BYJ-48 s 5 V napájaním a ovládačom s tranzistorovým poľom ULN2003, lineárneho skrutkového mechanizmu pre jednoosý pohyb vyrobeného pomocou 3D tlače a lineárneho Hallovho senzora 49E na module typu KY-035, ktoré sú ovládané pomocou dosky *Arduino* z prostredia *LabVIEW*. VI využíva funkcie z balíka *NI LabVIEW LINX Toolkit* alebo *LabVIEW Hobyyist Toolkit* a SubVI *MotorStep.vi* pre vykonanie jedného kroku motora podľa nastaveného smeru otáčania.



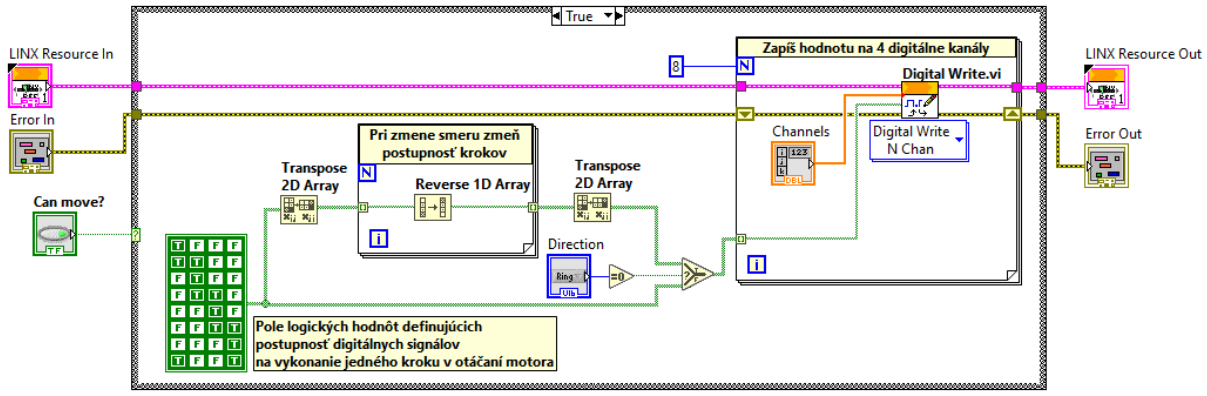
Obr. C.9: *Predný panel ProfilPola.vi* pre jednoosé mapovanie magnetickej indukcie permanentného magnetu pomocou Hallovho senzora. Použité sú ovládače a indikátory štýlu *Silver*.



Obr. C.10: *Blokový diagram ProfilPola.vi* pre jednoosé mapovanie magnetickej indukcie permanentného magnetu pomocou Hallovho senzora. VI reaguje na stlačenie ovládacích tlačidiel a vykoná rôzne akcie.

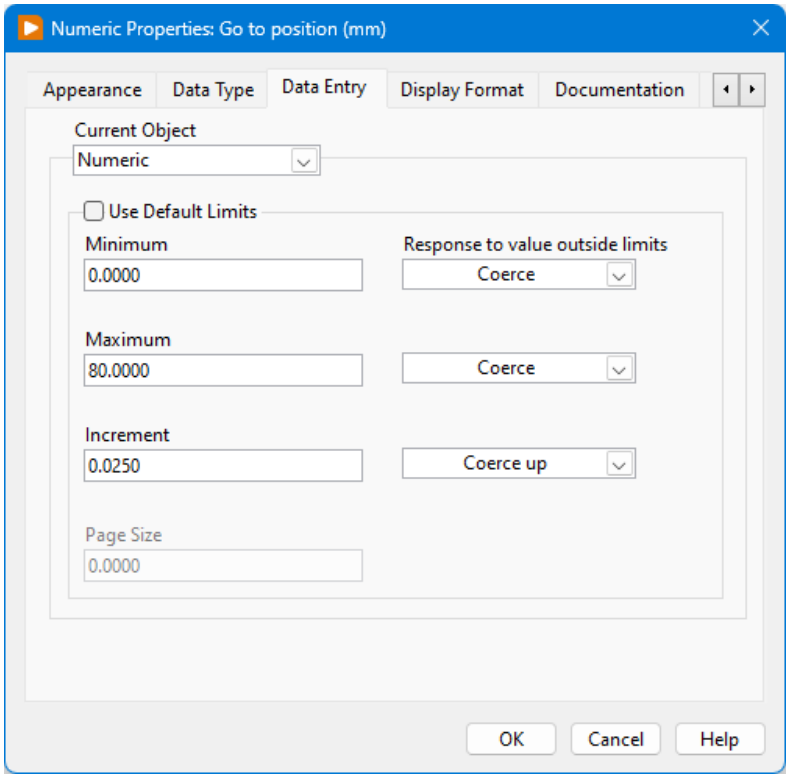


(a)



(b)

Obr. C.11: SubVI *MotorStep.vi* overí, či sa motor môže otáčať pomocou logického prepínača *Can move?*, ak áno, nastaví smer otáčania podľa ovládača *Direction* a postupne zasiela sériu pulzov na digitálne vstupy *Arduina* pomocou funkcie *Digital Write.vi*.



Obr. C.12: Konfiguračné dialógové okno ovládača *Go to position (mm)*, v ktorom na paneli *Data Entry* nastavíme možnú minimálnu a maximálnu hodnotu a možný krok. Nastavenie *Coerce* zabezpečí, že pri prekročení minimálnej alebo maximálnej hodnoty sa nastaví hraničná hodnota. Pri zadaní takej hodnoty, ktorá spadá medzi možné hodnoty definované minimálnym krokom sa nastaví najbližšia vyššia hodnota (*Coerce up*).

LITERATÚRA

1. *Jeff Kodosky, NIWeek2017, video* www.youtube.com/watch?v=paYlImL_qlw.
2. *G Web Development Software* www.ni.com/en/shop/electronic-test-instrumentation/programming-environments-for-electronic-test-and-instrumentation/what-is-g-web-development-software.html.
3. *LabVIEW Hobbyist Toolkit* www.ni.com/en/support/downloads/tools-network/download.labview-hobbyist-toolkit.html.
4. *Raspberry Pi* www.raspberrypi.org.
5. *BeagleBoard* www.beagleboard.org.
6. *Arduino* www.arduino.cc.
7. *LabVIEW Programming Reference Manual* www.ni.com/docs/en-US/bundle/labview-api-ref/page/intro.html.
8. Vlach, J., Havlíček, J. & Vlach, M. *Začínáme s LabVIEW* (Technická literatura BEN, Praha, 2008).
9. Bitter, R., Mohiuddin, T. & Nawrocki, M. *LabVIEW Advanced Programming Techniques* (CRC Press, Boca Raton, Fl., 2007).
10. Johnson, G. & Jennings, R. *LabVIEW Graphical Programming* (McGraw-Hill, New York, N.Y., 2006).
11. Schwartz, M. & Manickum, O. *Programming Arduino with LabVIEW* (Packt Publishing, Birmingham, U.K., 2015).
12. Rodríguez-Quiñonez, J. & Real-Moreno, O. *Graphical Programming Using LabVIEW, Fundamentals and advanced techniques* (The Institution of Engineering a Technology, London, U.K., 2022).
13. Gupta, S. & John, J. *Virtual Instrumentation Using LabVIEW (Principles and Practices of Graphical Programming)* (Tata McGraw Hill Education Private Limited, New Delhi, India, 2010).
14. Yang, Y. *LabVIEW Graphical Programming Cookbook* (Packt Publishing, Birmingham, U.K., 2014).
15. Mackenzie, C. *Coded Character Sets, History and Development* (Addison-Wesley Publishing Company, Reading, Ms., 1980).
16. Kalicharan, N. *C by Example* (Cambridge University Press, Cambridge, N.Y., 1994).
17. Toro, C., Wang, W. & Akhtar, H. *Implementing Industry 4.0* (Springer International Publishing, New York, N.Y., 2021).

18. Ehsani, B. *Data Acquisition using LabVIEW* (Packt Publishing, Birmingham, U.K., 2016).
19. Gladišová, I. & Mihalík, J. *Modulované signály* (Technická univerzita v Košiciach, Košice, 2016).
20. Orendáč, M. *Meranie a spracovanie malých signálov* (UPJŠ v Košiciach, Košice, 2010).
21. IEEE Standard Digital Interface for Programmable Instrumentation. *ANSI/IEEE Std 488-1978 (Revision of ANSI/IEEE 488-1975. Includes supplement IEEE Std 488A-19801),* 1 (1978).
22. *IVI Foundation* www.ivifoundation.org/.
23. IEEE Standard Codes, Formats, Protocols, and Common Commands for Use With IEEE Std 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation. *IEEE Std 488.2-1992,* 1 (1992).
24. *The SCPI Standard* www.ivifoundation.org/About-IVI/scpi.html.
25. Selecký, M. *Arduino* (Computer Press, Brno, 2016).
26. Schwartz, M. *Arduino for Secret Agents* (Packt Publishing, Birmingham, U.K., 2015).
27. Schwartz, M. *Internet of Things with the Arduino Yún* (Packt Publishing, Birmingham, U.K., 2014).
28. Anderson, R. & Cervo, D. *Pro Arduino* (Springer Science+Business, New York, N.Y., 2013).
29. Wilcher, D. *Learn Electronics with Arduino* (Springer Science+Business, New York, N.Y., 2012).
30. *How to build a plotter with Arduino* opensource.com/article/18/3/diy-plotter-arduino.
31. *DIY Arduino CNC Drawing Machine* www.instructables.com/DIY-Arduino-Drawing-Machine/.
32. *Mini laser CNC 3D Printed* yaowiz.com/2024/07/mini-laser-cnc-3d-printed/.
33. *DIY Arduino Printers: How to Make a 3D Printer with Arduino* www.elegoo.com/blogs/learn/diy-arduino-printers-how-to-make-a-3d-printer-with-arduino.
34. *Here Is How to Build Your Own DIY Mini Arduino 3D Printer* interestingengineering.com/videos/here-is-how-to-build-your-own-how-to-mini-arduino-3d-printer.
35. *The Best DIY Arduino 3D Printer Projects* all3dp.com/2/best-diy-arduino-3d-printer-projects/.
36. *Arduino Mega Pololu Shield* reprap.org/wiki/Arduino_Mega_Pololu_Shield.

37. *The Open Weather Station* openweatherstation.com/.
38. *Arduino Wireless Weather Station* www.instructables.com/Arduino-Wireless-Weather-Station/.
39. *Weather Station with Arduino Tutorial* core-electronics.com.au/guides/weather_station_with_arduino_tutorial/.
40. *Arduino IDE* www.arduino.cc/en/software.
41. *MicroPython* docs.arduino.cc/micropython/e.
42. *Arduino Lab for MicroPython, Lightweight editor for MicroPython* labs.arduino.cc/en/labs/micropython.
43. Desai, P. *Python Programming for Arduino* (Packt Publishing, Birmingham, U.K., 2015).
44. *Rigol DG1022Z online support* rigolshop.eu/products/waveform-generators/dg1000z/dg1022z.html.
45. *Picotest M3500A online support* www.picotest.com/support/m3500a-m3510a/.
46. Novotný, A. *Mechanické buzení SPM sond s integrovaným senzorem* (Vysoké učení technické v Brně, Brno, 2019).
47. Bruech, M. *Introduction of tuning fork quartz crystals. Nakagawa Electronics Limited* nkg.com.hk/wp-content/uploads/pdf/NKG-TIT_TuningForkCrystals.pdf.
48. Človečko, M. & Skyba, P. Quartz tuning fork – A potential low temperature thermometer in high magnetic fields. *Applied Physics Letters* **115,** 193507 (2019).
49. Friedt, J.-M. & Carry, E. Introduction to the quartz tuning fork. *American Journal of Physics* **75,** 415 (2007).
50. Meaney, D. *Tuning fork crystal frequency and parabolic temperature curve, ECS Inc.* ecsxtal.com/tuning-fork-crystal-frequency-and-parabolic-temperature-curve/.
51. Valeanu, A. *Temperature Compensation of a Tuning Fork Crystal Based on MCP7941X, Microchip Technology Inc.* ww1.microchip.com/downloads/en/Appnotes/00001413B.pdf.
52. Horowitz, P. & Hill, W. *The Art of Electronics* (Cambridge University Press, Cambridge, U.K., 1989).
53. Carter, B. & Mancini, R. *Op Amps for Everyone* Fifth Edition (Newnes, Elsevier, Oxford, U.K., 2018).
54. Cebron, D. *Magnetic fields of solenoids and magnets, MATLAB Central File Exchange* www.mathworks.com/matlabcentral/fileexchange/71881-magnetic-fields-of-solenoids-and-magnets.
55. *DEXINMAG, Multidimensional magnetic field test system* www.xmdexing.cn/index/products/index/cateid/21.

56. *GMW Associates, Magnetic Field Mapper* gmw.com/product/magnetic-field-mapper/.
57. *SENIS Magnetic Field Mappers* www.senis.swiss/mappers/.
58. Murzin, D., Mapps, D., Levada, K., Belyaev, V., Omelyanchik, A., Panina, L. & Rodionova, V. Ultrasensitive Magnetic Field Sensors for Biomedical Applications. *Sensors* **20,** 1569 (2020).
59. Samuely, T. *Skenovacia tunelová mikroskopia* (ŠafárikPress, Košice, 2023).
60. Lipták, B. G. *Instrument Engineers' Handbook: Process Control and Optimization* (CRC Press, Boca Raton, Fl., 2005).
61. Omelyanchik, A., Marqués, J., Rivas, M., Rodionova, V., Canepa, F. & Peddis, D. A do-it-yourself approach for developing a magnetic field mapping setup using a 3D printer. *Measurement Science and Technology* **34,** 107001 (2023).
62. Vavoulas, A., Vaiopoulos, N., Hedström, E., Xanthis, C., Sandalidis, H. & Aletras, A. Using a modified 3D-printer for mapping the magnetic field of RF coils designed for fetal and neonatal imaging. *Journal of Magnetic Resonance* **269,** 146 (2016).
63. Han, H., Moritz, R., Oberacker, E., Waiczies, H., Niendorf, T. & Winter, L. Open Source 3D Multipurpose Measurement System with Submillimetre Fidelity and First Application in Magnetic Resonance. *Scientific Reports* **7,** 13452 (2017).
64. *MagField-MT, low-cost magnetic field mapping platform* github.com/LGST-LAB/MagField-MT.
65. Orendáč, M. *Základy fyzikálnych metód vo fyzike kondenzovaných látok* (UPJŠ v Košiciach, Košice, 2011).
66. *High Accuracy Dovetail Stepper Slide* www.thingiverse.com/thing:2809285.
67. *MikroE* www.mikroe.com/step-motor-5v.
68. *Arduino Mega2560 REV3 Product Reference Manual* docs.arduino.cc/resources/datasheets/A000067-datasheet.pdf.
69. *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V datasheet* ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf.
70. *Arduino Uno R3 Product Reference Manual* docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf.
71. *ATmega48A/PA/88A/PA/168A/PA/328/P* ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf.
72. *ESP8266 cost-effective and highly integrated Wi-Fi MCU for IoT applications* www.espressif.com/en/products/socs/esp8266.
73. *RobotDyn UNO+WiFi R3 ATmega328P+ESP8266, 32Mb flash, USB-TTL CH340G, Micro-USB* robotdyn.com/uno-wifi-r3-atmega328p-esp8266-32mb-flash-usb-ttl-ch340g-micro-usb.html.

74. *RA4M1 32-bit Microcontrollers with 48MHz Arm Cortex-M4 and LCD Controller and Cap Touch for HMI* www.renesas.com/us/en/products/microcontrollers-microprocessors/ra-cortex-m-mcus/ra4m1-32-bit-microcontrollers-48mhz-arm-cortex-m4-and-lcd-controller-and-cap-touch-hmi.

75. *Arduino Mega2560 REV3 pinout* docs.arduino.cc/resources/pinouts/A000067-full-pinout.pdf.

Tento dokument bol zostavený s použitím typografického štýlu `classicthesis`, ktorý vyvinul André Miede. Štýl bol inšpirovaný knihou Roberta Bringhursta o typografii „The Elements of Typographic Style“. Štýl `classicthesis` je dostupný pre LaTeX a LyX:

https://bitbucket.org/amiede/classicthesis/wiki/Home

Automatizácia laboratórnych experimentov v prostredí LabVIEW
Vysokoškolský učebný text k predmetu Grafické programovanie

AUTOR: doc. RNDr. Erik Čižmár, PhD.

VYDAVATEĽ: Univerzita P.J. Šafárika v Košiciach
Vydavateľstvo ŠafárikPress

ROK VYDANIA: 2024
POČET STRÁN: 112
ROZSAH: 5 AH
VYDANIE: prvé

ISBN 978-80-574-0354-8 (e-publikácia)