TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Řízení technologického procesu v prostředí LabVIEW Statechart

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 3906T001 – Mechatronika

Autor práce: **Bc. Tomáš Lauer**
Vedoucí práce: doc. Ing. Milan Kolář, CSc.

Liberec 2016

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Technological process control using LabVIEW Statechart

Diploma thesis

Study programme: N2612 – Electrical Engineering and Informatics
Study branch: 3906T001 – Mechatronics

Author: **Bc. Tomáš Lauer**
Supervisor: doc. Ing. Milan Kolář, CSc.

Liberec 2016

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2015/2016**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Tomáš Lauer**
Osobní číslo:	**M14000202**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Mechatronika**
Název tématu:	**Řízení technologického procesu v prostředí LabVIEW Statechart**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s prostředím NI LabVIEW, zejména s jeho modulem Statechart.
2. Použitím modulu LabVIEW Statechart navrhněte stavový předpis pro řízení modelového technologického procesu třídicího stroje.
3. Odzkoušejte navržené řešení na platformě NI CompactRIO.
4. Proveďte diskusi navrženého řešení při srovnání s obdobnou aplikací řešenou s řídicím systémem PLC (např. TECO, OMRON, Mitsubishi apod.).

Rozsah grafických prací:	**dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 40–50 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **NI LabVIEW Statechart Module – National Instruments. [online]. 2015 [cit. 2015-10-08]. Dostupné z http://www.ni.com/labview/statechart/**

[2] **Kretschmerová, L., Vlach, J.: Programování v LabVIEW v příkladech. Skriptum TUL, Liberec 2014, ISBN: 978-80-7372-167-2.**

Vedoucí diplomové práce:	**doc. Ing. Milan Kolář, CSc.** Ústav mechatroniky a technické informatiky
Konzultant diplomové práce:	**Ing. Jaroslav Vlach, Ph.D.** PRECIOSA a.s., Jablonec n. Nisou
Datum zadání diplomové práce:	**10. října 2015**
Termín odevzdání diplomové práce:	**16. května 2016**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -
L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2015

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 16.5.2016

Podpis: Jan Tomáš

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé diplomové práce panu doc. Ing. Milanovi Kolářovi, CSc. za věcné rady a připomínky.

Dále bych chtěl poděkovat panu Ing. Jaroslavovi Vlachovi, Ph.D. a panu Josefovi Havlíčkovi z firmy Preciosa a.s. za jejich čas, který mi věnovali při konzultacích praktické části diplomové práce.

Abstrakt

Diplomová práce se zabývá návrhem stavového předpisu pro řízení technologického procesu třídicího stroje s využitím modulu LabVIEW Statechart.

Teoretická část pojednává o funkcích a jednotlivých komponentách rozšiřujícího modulu, potřebných k návrhu stavového předpisu.

Praktická část popisuje samotný návrh algoritmu pro třídění bižuterních kamenů a jeho implementaci na platformě NI CompactRIO s využitím modulu LabVIEW Statechart. Dále je pro stejnou úlohu vytvořen algoritmus pomocí čisté instalace LabVIEW bez modulu Statechart.

Závěrem práce jsou vyhodnoceny výhody a nevýhody jednotlivých řešení. V poslední řadě je diskutována vhodnost investice do rozšiřujícího modulu LabVIEW Statechart ve firmě Preciosa a.s.

Klíčová slova

LabVIEW, LabVIEW Statechart, CompactRIO, stavový automat, třídicí stroj

Abstract

This diploma thesis deals with the state diagram design for control of the technological process of sorting machine using LabVIEW Statechart.

The theoretical part is focused on features and components of utilized LabVIEW module.

The practical part describes the creation of the algorithm for sorting jewelry stones and its implementation with NI CompactRIO system. Furthemore, algorithm without using LabVIEW Statechart is created.

In the final part are discussed advantages and disadvantages of each methods and the suitability of investment in the extension module LabVIEW Statechart for company Preciosa Inc.

Keywords

LabVIEW, LabVIEW Statechart, CompactRIO, state machine, sorting machine

Obsah

Seznam obrázků

Úvod

Tato diplomová práce se zabývá návrhem aplikace určené k řízení technologického procesu ve vývojovém prostředí LabVIEW Statechart od firmy National Instruments. Konkrétně se jedná o proces třídění bižuterních kamenů na základě jejich kvality. Hlavní ideou však není pouze návrh aplikace v daném prostředí, ale především posouzení efektivity tohoto nástroje při vývoji řídicího algoritmu konkrétní úlohy.

Na začátku práce dochází k seznámení s rozšiřujícím modulem LabVIEW Statechart. Tento modul je nadstavbou umožňující vývoj stavových automatů a jejich následující implementaci, například pomocí kontroléru firmy National Instruments CompactRIO. V práci je dále čtenář seznámen se základní architekturou CompactRIO, s technickým a programovým vybavením a možnostmi využití kontroléru.

Praktická část diplomové práce se zaměřuje na oživení modelu třídicího stroje bižuterních kamenů, řešení reálných problémů týkajících se programování FPGA a aplikací reálného času pomocí LabVIEW Statechart realizovaných prostřednictvím kontroléru CompactRIO. Závěrem práce je provedena diskuse efektivity tohoto způsobu programování oproti vývoji algoritmu s využitím LabVIEW bez modulu Statechart nebo jiných programovacích jazyků.

1. Modul LabVIEW Statechart

LabVIEW Statechart je rozšiřující modul grafického vývojového prostředí LabVIEW od společnosti National Instruments. Tato nadstavba umožňuje tvorbu stavových automatů pomocí vývojových diagramů, které však neztrácí na přehlednosti a to hlavně díky poměrně vysoké úrovni abstrakce.

Po instalaci modulu LabVIEW Statechart je třeba vytvořit soubor Statechart.lvsc, ve kterém je navržen samotný stavový diagram a jsou definovány jeho vstupy, výstupy, akce stavů, triggery přechodů atd. Jednotlivé prvky a funkce modulu jsou popsány níže.

Po vytvoření stavového diagramu a definování potřebných náležitostí je třeba diagram implementovat pomocí tzv. Caller VI (Virtual Instrument). Jedná se o VI, ve kterém je použit funkční blok *Run Statechart,* umožňující nastavení vstupů a další zpracování výstupů stavového automatu. [1]

1.1 Nastavení stavového automatu

Ihned po vytvoření souboru stavového automatu s příponou .lvsc je doporučeno zkontrolovat, a případně změnit, defaultní nastavení generování kódu stavového automatu, nacházející se ve vlastnostech stavového automatu pod kolonkou *Statechart Code Generation Page*. Přesněji řečeno se jedná o určení typu zařízení, na kterém bude stavový automat implementován, dále o využití stavového automatu a nakonec nastavení front. Na základě této konfigurace se odvíjí velikost, rychlost a funkčnost generovaného kódu. [2]

Typ zařízení

- **Desktop** – Tento typ zařízení je nastaven defaultně. Umožnuje spustit stavový automat na osobním počítači s operačním systémem Windows.
- **LabVIEW Real-Time Target** – Implementace stavového automatu na zařízení, které pracuje v reálném čase.
- **LabVIEW FPGA Target** - Stavový automat realizovaný pomocí programovatelných hradlových polí (FPGA).
- **LabVIEW Mobile/Touch Panel/Embedded** - Se stavovým automatem lze také pracovat na mobilních zařízeních.
- **LabVIEW Embedded Static** – Tímto nastavením je možné spustit stavový automat na tzv. embedded zařízení vyžadující statickou alokaci polí. [2]

Využití

- **Synchronní** – Defaultní nastavení stavového automatu.
- **Asynchronní**

V případě implementace stavového automatu na FPGA:

- **Jednocyklová smyčka** – Defaultní nastavení stavového automatu. Jedna iterace proběhne vždy během jednoho cyklu procesoru.
- **Smyčka while** – Stavový automat je realizován pomocí cyklu WHILE**.** [2]

Nastavení front

U asynchronních stavových automatů lze nastavit externí frontu triggerů a u aplikací reálného času frontu *RT FIFO*, což je fronta určená speciálně pro aplikace reálného času. Další nastavení front se týká velikosti interní fronty triggerů. Konkrétně určení počtu triggerů, které interní fronta dokáže uchovat.

V sekci nastavení generování kódu se také nachází zaškrtávací pole, které udává, zda dojde po každé iteraci k vymazání hodnot výstupů stavového automatu. Dále je zde možné povolit či zakázat ladění automatu. [2]

1.2 Synchronní a asynchronní stavový automat

Ještě před vytvořením souboru Statechart.lvsc je nutné vědět, k čemu bude stavový automat sloužit a podle toho rozhodnout, zda se bude jednat o synchronní nebo asynchronní (v případě, že stavový automat poběží na osobním počítači). Typ stavového automatu má vliv na konfiguraci funkčního bloku *Run Statechart*, která se provede při generování kódu stavového automatu. Jestliže je však nutné tento typ změnit po návrhu diagramu, musí se také změnit způsob, jakým ostatní VI odesílají data do stavového automatu. [1] [3]

Výběr typu stavového automatu záleží na tom, kdy je třeba provádět iterace. U synchronního stavového automatu se iterace provede vždy v momentě, kdy jsou všechna vstupní data přijata funkčním blokem *Run Statechart*. Zatímco asynchronní stavový automat provede iteraci pouze tehdy, když funkční blok *Run Statechart* přijme všechna vstupní data a trigger (spouštěč přechodu mezi stavy nebo statické reakce) se nachází v externí frontě. Pokud jsou vstupní data přijata a trigger se nenachází v externí frontě, funkční blok *Run Statechart* se neprovede a čeká, dokud není trigger nastaven. [1] [3]

1.3 Soubor Statechart.lvsc

Otevřením souboru Statechart.lvsc se v průzkumníku projektů rozbalí šest řádků s možným nastavením stavového automatu.

- **Edit Triggers and Groups** – Vytvoření nebo úprava triggerů.
- **Inputs.ctl** – Definice datových typů vstupů. Stavový diagram může tyto data pouze číst.
- **Outputs.ctl** – Definice datových typů výstupů. Stavový diagram může tyto data číst i přepisovat.
- **Statedata.ctl** – Definice datových typů vnitřních dat stavového diagramu.
- **CustomDataDisplay.vi** – Nastavení informací o stavovém automatu, které se zobrazí při odlaďování programu.
- **Diagram.vi** – Tvorba stavového diagramu. [4]

Dále jsou rozebrány jednotlivé komponenty a pojmy potřebné k vytvoření stavového diagramu v *Diagram.vi.*

1.4 Stavy

Stavy (Obrázek 1) jsou jedny ze základních prvků modulu Statechart. Definováním tohoto prvku se vytvoří unikátní stav, který může stavový automat nabývat.

U každého stavu lze nastavit několik akcí, které modifikují jeho výstup a vnitřní data. Mezi tyto akce patří vstupní akce, výstupní akce a statická reakce. [1] [5]

Obrázek 1: Stav

Vstupní a výstupní akce

Vstupní akce proběhne ihned po vstupu do stavu, tzn. po vykonání přechodu z jiného stavu. Naopak výstupní akce proběhne při opuštění daného stavu, ještě než proběhne přechod do následujícího stavu.

Každý stav může mít právě jednu vstupní a výstupní akci. Tyto akce lze nastavit dvojklikem na hranu stavu.

Zmiňované akce se vykonají vždy pouze jednou, a to při vstupu nebo výstupu ze stavu. Jestliže je potřeba vykonat některé akce podmíněně, je vhodné použít statickou reakci. [1] [5]

Statické reakce

Statické reakce definují chování stavu, pokud je v klidu. To znamená, že nedochází k přechodu z předešlého stavu nebo do dalšího stavu. Aby se tato akce mohla vykonat, musejí být splněny dvě podmínky:

- Stavový automat musí obdržet trigger.
- Guard neboli podmínka aktivace (musí být splněna pro pokračování dané reakce) asociovaná s reakcí musí nabývat hodnoty TRUE. Pokud statická reakce nemá nastavenou podmínku aktivace, tak je vždy nastavena na hodnotu TRUE.

Každý stav může mít několik statických reakcí a lze je vykonat v jedné iteraci. Reakce se vykonávají jedna za druhou podle toho, v jakém pořadí jsou definovány. [5]

1.5 Regiony, vnořené stavy a superstavy

Regiony (Obrázek 2) definují prostor, do kterého je možné vkládat stavy a pseudostavy. Po otevření *Diagram.vi* lze na pracovní plochu ihned vložit stav. Z toho vyplývá, že pracovní plocha je v podstatě velký region. Každý další region, který je třeba k funkčnosti určitého stavového diagramu, je možné vložit pouze do vytvořeného stavu.

Pokud stav obsahuje region, nazývá se superstavem (Obrázek 2) a stav uvnitř tohoto regionu je označován jako vnořený stav (Obrázek 2). Přechody a statické reakce superstavu mají vyšší prioritu než přechody a statické reakce vnořeného stavu. Z čehož je zřejmé, že i když je aktivní vnořený stav, tak superstav může kdykoliv provést statickou reakci či přechod (např. ukončení).

Vzniká tu také možnost existence více regionů v jednom superstavu. Tyto regiony se nazývají ortogonální a stavy uvnitř nemohou probíhat paralelně. Priorita regionů je stanovena abecedně podle názvu. [1] [5] [6]

Obrázek 2: Superstav, Region a vnořený stav

1.6 Přechody

Přechody definují podmínky, za kterých se může stavový automat pohybovat mezi stavy. Vytvořením přechodu se stanoví, jak stavový automat reaguje na triggery a podmínky aktivace. Tím se určí, v jakém pořadí se stavový diagram vykoná. [1] [7]

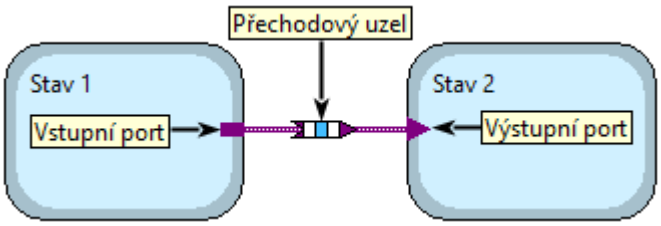
Porty

Porty jsou místa ve stavu označující připojení přechodů. Vstupní port má tvar trojúhelníku a výstupní port má tvar obdélníku. [7]

Přechodový uzel

Nejdůležitější částí přechodů jsou přechodové uzly (Obrázek 3). Jedná se o automaticky vytvořené objekty sloužící ke konfiguraci triggerů, podmínek přechodů a akcí přechodů.

Uzel se skládá ze tří obdélníkových polí. Pole vlevo, tzn. neblíže k předešlému stavu, značí trigger přechodu. Pokud je bílé, znamená to, že je trigger nastavený na hodnotu null. Tato hodnota se automaticky nastaví pro nové přechody. Jestliže je pole vyplněno modrou barvou, trigger přechodu je nastaven minimálně na jednu jinou hodnotu než je null. Prostřední pole značí nastavení podmínky aktivace. Pole je bílé, avšak po nastavení, změní barvu na modrou. Třetí pole, nejblíže k následujícímu stavu, značí nastavení akce, která se vykoná během přechodu. Jako v předchozích případech modrá výplň pole znamená, že byla akce nastavena. [1] [7]



Obrázek 3: Přechod mezi stavy

Fáze přechodů

Každý přechod mezi stavy má dvě fáze. První fáze nastává ihned po přijetí triggeru a dochází k vyhodnocení, zda má přechod nastavenou podmínku aktivace. Druhá fáze nastává pouze tehdy, když je vyhodnocena podmínka aktivace jako TRUE. Poté dochází k vykonání specifikované akce přechodu a k samotnému přechodu do následujícího stavu. Pořadí, v jakém stavový automat provádí vyhodnocení přechodů, závisí na prioritě dané uživatelem. [1] [7]

1.7 Triggery, podmínky aktivace a akce

Vykonání přechodů a statických reakcí stavů je podmíněno správným nastavením triggeru, podmínky aktivace a akce. Na základě těchto třech elementů se stavový automat rozhoduje, zda má být vykonán přechod mezi stavy nebo provedena statická reakce stavu.

Nastavení těchto elementů je možné dvojitým klikem na přechodový uzel nebo na okraj stavu. [6] [8]

Triggery

Triggery by se daly volně přeložit jako spouštěče daných akcí. Kdykoliv dojde k vytvoření přechodu nebo statické reakce, musí být specifikován trigger, podle kterého stavový automat vyhodnotí, zda se daný přechod či statická reakce vykoná. Triggerem může být například stisk tlačítka či změna určité hodnoty provedená v Caller VI.

Stavový automat může přijímat triggery z Caller VI nebo reaguje na triggery vzniklé přímo ve stavovém diagramu. V obou situacích je nutné triggery definovat, jinak jsou nastaveny na hodnotu null.

Null trigger se využívá hlavně u synchronních stavových automatů, kde jsou tyto triggery generovány každou iterací. Dochází tedy k přechodu mezi stavy s určitou periodou.

Naproti tomu asynchronní stavový automat ukládá příchozí triggery z Caller VI do externí fronty a postupně je zpracovává. Existují dvě funkce, pomocí kterých lze poslat trigger do fronty stavového automatu. Tyto funkce se nazývají *Send External Trigger* a *Send Internal Trigger*. [8]

Podmínka aktivace

Po přijetí určitého triggeru vyhodnotí stavový automat zdrojový kód podmínky aktivace. Je-li podmínka aktivace splněna (kód vrací hodnotu TRUE), stavový automat pokračuje dál s přechodem či statickou reakcí. Pokud však není podmínka aktivace splněna (kód vrací hodnotu FALSE), nemůže být daný přechod nebo statická reakce uskutečněna.

Defaultně je podmínka aktivace nastavena na hodnotu TRUE. [8]

Akce

V akci se nachází zdrojový kód určený k modifikaci výstupů, stavových dat a k posílání triggerů do interní fronty. Stavový automat vykonává akce v různých časech závislých na tom, jestli je akce přidružena k přechodu nebo statické reakci. Akce mohou být provedeny ve dvou situacích:

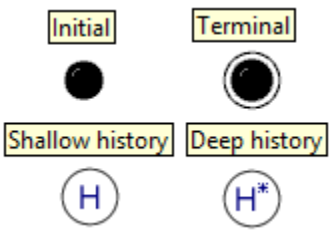
- Během přechodu – po vykonání výstupní akce aktuálního stavu, ale před vykonáním vstupní akce následujícího stavu. Akce se provede, jestliže je splněna podmínka aktivace.
- Při provádění statické reakce – poté, co hlídač vrátí hodnotu TRUE. [1] [8]

1.8 Pseudostavy a spojky

K zajištění plné funkčnosti vývojového diagramu je nutné použít v jeho návrhu některé pseudostavy a spojky.

Pseudostavy jsou komponenty, které mají specifické vlastnosti. Pseudostavy modulu Statechart a jejich vlastnosti:

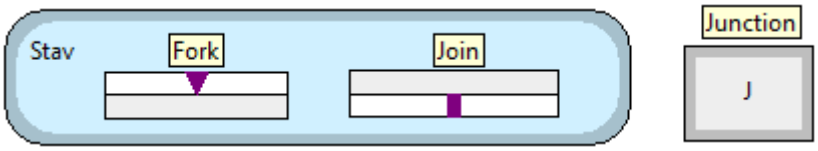
- **Initial (Inicializace)** – Tento pseudostav se musí nacházet v každém regionu, aby program věděl, který stav se má v daném regionu vykonat jako první. Inicializace muže být v regionu pouze jedna.
- **Terminal (Ukončení)** – Do pseudostavu *Terminal* se připojuje stav, ze kterého může dojít k ukončení stavového diagramu. V jednom regionu může být použito více těchto pseudostavů.
- **Shallow history a Deep history (Mělká a Hluboká minulost)** - Jestliže dojde k opuštění a návratu do daného regionu, začne probíhat substav, ke kterému je připojen *Initial*. Je-li třeba, aby stavový automat pokračoval tím stavem, ve kterém byl před opuštěním regionu, použije se *Shallow history*. Jedná-li se o region s dalšími vnořenými stavy a je žádáno pokračování hluboko vnořenými stavy, využije se pseudostav *Deep history*. [1] [6] [9]



Obrázek 4: Pseudostavy

Spojky jsou objekty, které mají za úkol slučovat, či rozdělovat přechody. Modul Statechart obsahuje následující spojky:

- **Fork (Rozvětvení)** – Rozděluje jeden přechod (Obrázek 5) do více přechodů. Přechod z jednoho stavu do více vnořených stavů.
- **Join (Sloučení)** – Slučuje více přechodů (Obrázek 5) do jednoho. Přechod ze dvou vnořených stavů do jednoho stavu. Rozvětvení a Sloučení je možné vložit pouze do stavu.
- **Junction (Spojka) -** Spojuje dohromady společné prvky (Obrázek 5) několika přechodů. Jedná se například o spojení přechodů se stejnou akcí, přestože může mít každý přechod stejný trigger. Stavový automat provede všechny přechody vycházející ze Spojky v jedné iteraci, proto je zde možné nastavit prioritu těchto přechodů. [1] [6] [9]



Obrázek 5: Spojky

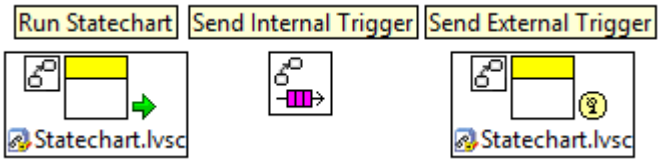
1.9 Fronty

Každý stavový automat má jednu nebo více front triggerů. LabVIEW Statechart pracuje s frontami metodou FIFO (first in – first out). To znamená, že pokud je trigger přijatý do fronty jako první, je také jako první poslán stavovému automatu. Stavové automaty kontrolují tyto fronty každou iteraci.

Asynchronní stavový automat může pracovat s externí a interní frontou. Externí fronta pracuje s triggery zaslanými pomocí funkce *Send External Trigger* (Obrázek 6). Když je stavový diagram spuštěný, umí přijímat triggery od jakéhokoli VI. Je tu také možnost triggery posílat do externí fronty z paralelního vlákna z Caller VI. Například funkce *Run statechart* bude spouštěna pomocí smyčky WHILE a funkce *Send External Trigger* bude spouštěna z paralelní smyčky WHILE.

Interní fronta je vlastně kruhový buffer uchovávající triggery poslané do stavového diagramu pomocí funkce *Send Internal Trigger*. Tato funkce může být použita pouze ve zdrojovém kódu akce daného stavu či přechodu. Proto nemůže být funkce *Send Internal Trigger* (Obrázek 6) použita v Caller VI. [1] [10]

Synchronní stavový automat pracuje pouze s interní frontou. Místo posílání triggerů do stavového automatu pomocí funkce *Send External Trigger* lze připojit triggery přímo ke vstupu funkce *Run Statechart* (Obrázek 6). Avšak typicky se synchronnímu stavovému automatu triggery neposílají, protože zahájení iterace tohoto automatu závisí na toku dat. Jestliže není ke vstupu připojen žádný trigger, je automaticky posílán null trigger. [1] [10]



Obrázek 6: Komunikační funkční bloky

Priority front v asynchronním stavovém automatu

Asynchronní stavové automaty zahájí iteraci vždy až po přijetí triggeru z externí fronty. Kontrola externí fronty nastává pouze v případě, že je interní fronta prázdná. To znamená, že pokud během jedné iterace pošle stavový automat několik triggerů do interní fronty a zároveň je poslán trigger z Caller VI do externí, nedojde ke zkontrolování externí fronty, dokud nebude interní fronta prázdná. [1] [10]

1.10 Caller VI

Po vytvoření stavového automatu a definování jeho náležitostí v souboru s příponou .lvsc je nezbytné vygenerovat jeho zdrojový kód kliknutím na tlačítko *Code Generation for this Statechart* v *Diagram.vi*.

Protože soubor s příponou .lvsc neumožňuje vytvořený stavový automat spustit a dále s ním pracovat, musí se vytvořit tzv. Caller VI, ve kterém je použit funkční blok *Run Statechart*. Tato funkce vytváří a spouští instanci stavového automatu. Spuštění stavového automatu znamená zaslání vstupních dat, čtení a zpracování výstupních dat automatu. Caller VI může být také použit k zaslání triggeru.

Struktura Caller VI je závislá na typu stavového diagramu. [11]

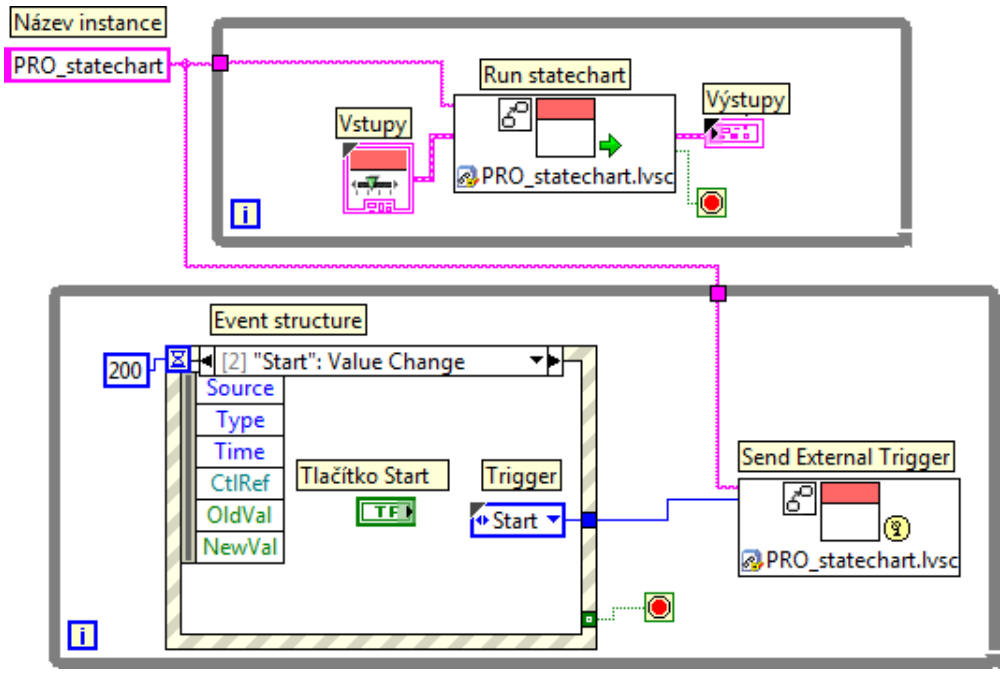
Asynchronní stavový automat

Spuštění asynchronního stavového diagramu zahrnuje dva procesy:

- spuštění stavového diagramu pomocí funkce *Run Statechart*
- odesílání triggeru do externí fronty skrze funkci *Send External Trigger* [11]

V horní části Caller VI (Obrázek 7) se nachází smyčka stavového automatu, která je tvořena cyklem WHILE. Tato smyčka má na starosti spouštění samotného stavového automatu. K funkčnímu bloku *Run Statechart* je nutné připojit název instance, vstupy a výstupy.

Paralelně se smyčkou stavového diagramu probíhá trigger smyčka. *Event structure*, která je v tomto případě součástí trigger smyčky, zpracovává události vyskytující se v daném VI. Zde *Event structure* vyhodnocuje stisknutí tlačítka Start. Když dojde ke stisknutí tohoto tlačítka, funkční blok *Send External Trigger* odešle trigger Start do stavového automatu PRO_statechart.lvsc. [11]

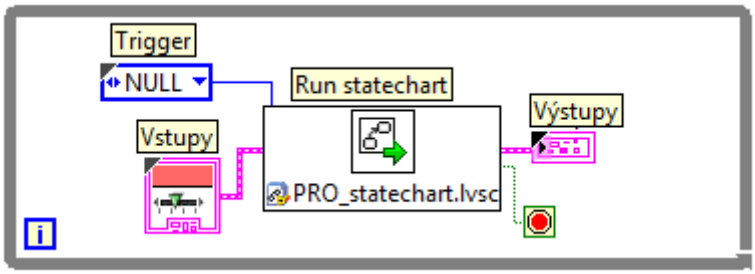


Obrázek 7: Caller VI asynchronního stavového automatu

Synchronní stavový automat

Jelikož synchronní stavové automaty nepracují s externí frontou triggerů, odpadá zde použití funkčního bloku *Send external trigger*. Z této skutečnosti plyne, že k implementaci synchronního stavového automatu (Obrázek 8) postačuje jedna smyčka s funkčním blokem *Run Statechart*, který umožňuje oproti asynchronnímu zaslat

trigger přímo. Této možnosti se však příliš nevyužívá, protože se synchronní automaty navrhují především tak, aby reagovali na tok vstupních dat. [11]



Obrázek 8: Caller VI synchronního stavového automatu

Stavový automat běžící na FPGA

Dalším typem může být stavový automat využívající cyklus WHILE nebo jednocyklovou smyčku. Tyto typy jsou dostupné pouze v případě, že stavový automat bude implementován na FPGA zařízení.

Caller VI pro stavový automat běžící na FPGA se v podstatě ničím neliší od Caller VI synchronního. Pokud je automat nastaven na využití jednocyklové smyčky, musí být i v Caller VI umístěn funkční blok *Run Statechart* v tomto typu smyčky, jestliže tomu tak není, pak se stavový automat nebude chovat jako jednocyklová aplikace (kód umístěný uvnitř jednocyklové smyčky se vykoná vždy během jednoho cyklu procesoru). [1] [11]

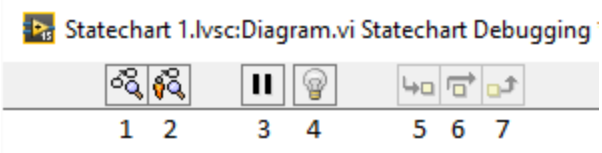
1.11 Odlaďování stavových automatů

Nezbytnou součástí programování bývá velmi často nepříjemné hledání chyb ve zdrojovém kódu nebo blokovém diagramu. I na tuto skutečnost firma National Instruments myslela a modul LabVIEW Statechart obsahuje tzv. debug mód umožňující právě zmiňované odladění stavového automatu. [12]

Debug mód je podporován na automatech realizovaných na osobním počítači a na zařízení běžícím v reálném čase. Stavový automat spuštěný na FPGA zařízení debug mód nemá.[12]

Mód ladění programu lze spustit z Caller VI kliknutím pravého tlačítka myši na funkční blok *Run Statechart* a zvolením položky *Debug Statechart*. Po kliknutí se zobrazí okno s panelem ladících funkcí (Obrázek 9) a daným stavovým diagramem. Funkce s číslem 1 slouží k otevření náhledu do stavových dat, zobrazení výstupních dat a interní fronty triggerů. Číslem 2 se zobrazí stavová data, která je možné upravovat. Stisknutím

tlačítka s číslem 3 se pozastaví běh programu, aktivuje se tzv. mód *Highlight execution* (lze spustit i bez pozastavení programu, funkcí s číslem 4) zvýrazňující aktivní prvek stavového diagramu a aktivuje tlačítka 5, 6 a 7 sloužící ke krokování (krokování funguje stejně jako při odlaďování klasického VI). [12] [13]



Obrázek 9: Funkce odlaďování

Při spuštěném módu *Highlight execution* se budou v levém horním rohu aktivních stavů zobrazovat tři různě barevné tvary:

- žlutý čtverec indikující probíhající vstupní akci daného stavu
- zelený trojúhelník indikující probíhající statickou reakci daného stavu
- červený kruh indikující probíhající výstupní akci daného stavu [12] [13]

Je-li třeba odladit posledních pár stavů poměrně rozsáhlého stavového diagramu, je vhodné umístit například na přechod před těmito stavy tzv. brakepoint (kliknutím pravého tlačítka myši na daný přechod či stav a vybráním *Set Brakepoint*). Po spuštění stavového automatu se program u brakepointu pozastaví a lze pokračovat krokováním.

Z důvodu šetření zdrojů je možné debug mód vypnout v nastavení generování kódu (viz kapitola 1.1), což je užitečné především tehdy, pokud má stavový automat běžet na zařízení reálného času. [12] [13]

1.12 Přenos dat v prostředí LabVIEW Statechart

Aby mělo smysl pracovat se stavovým automatem, je třeba mu předat vstupní data a naopak od něj získat výstupní data.

Nejužívanější metodou přenosu dat je využití vstupních a výstupních parametrů funkčního bloku *Run Statechart*. Dále se nabízí možnost využít *VI Server References,* sdílené proměnné či globální proměnné. [14]

Vstupní a výstupní parametry

Funkční blok *Run Statechart* má vstupní a výstupní parametry odpovídající vstupním a výstupním datům připojeného stavového automatu. Tyto vstupy a výstupy musí být předem definovány (viz kapitola 1.3). Načtení vstupů proběhne vždy před iterací

a výstupy jsou dostupné po dokončení dané iterace. Z tohoto důvodu není možné tuto metodu použít k přenosu dat uprostřed iterace. [14]

Použití vstupů a výstupů může vést ke zvětšení souborů *Outputs.ctl* a *Inputs.ctl*, což může zapříčinit další generování kódu. Další nevýhodou tohoto způsobu přenosu dat je, že při každé iteraci dojde k načtení všech dat, i když nejsou pro danou iteraci třeba.

I přes tyto nevýhody firma National Instruments doporučuje tuto metodu používat především z důvodu zachování správného datového toku. Další výhoda této metody je jednodušší odlaďování stavových automatů a minimalizace vzniku souběhu. [14]

VI Server References

VI Server References se hodí především tehdy, pokud je třeba aktualizovat data na front panelu přímo za stavového automatu. Vytvoření VI Server Reference jako vstupu stavového automatu přináší tu výhodu, že dojde k předání pouze dané hodnoty a nedojde tedy k načtení celého datového souboru jako tomu je v předchozím případě. Dále není třeba čekat, až stavový automat provede iteraci, aktualizace hodnoty je okamžitá. [14]

Sdílené proměnné

Sdílená proměnná může být jakékoliv datového typu používaného v prostředí LabVIEW a je přístupná i pro více VI či více počítačů. Se sdílenou proměnnou lze pracovat přímo ve stavovém automatu, z čehož vyplývá, že není třeba použít pro přenos dat vstupní a výstupní parametry funkčního bloku *Run Statechart*.

Při užití sdílené proměnné je nezbytné počítat s tím, že může dojít k souběhu, např. když se k proměnné přistupuje z více počítačů najednou. Sdílené proměnné nelze použít na FPGA zařízeních. [14]

Funkční globální proměnné

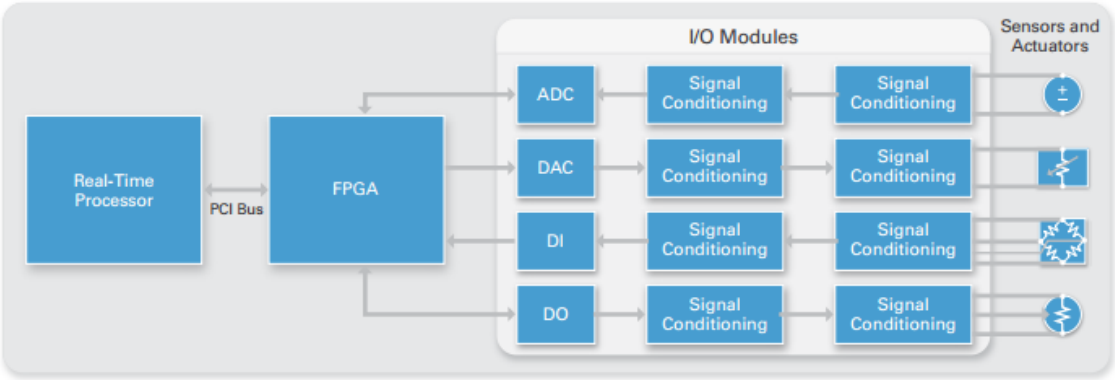
Funkční globální proměnné se podobají sdíleným proměnným, s tím rozdílem, že funkční globální proměnná je VI obsahující smyčku WHILE a posuvný registr k uchování hodnoty. Tato struktura zmenšuje pravděpodobnost souběhu tak, že dokáže specifikovat, kdo a kdy může data do proměnné zapisovat či číst. [14]

2. CompactRIO

Kontrolér CompactRIO od firmy National Instruments je rekonfigurovatelný řídicí systém sloužící k implementaci řídicích a monitorovacích aplikací využívaných v průmyslu. CompactRIO je komplexní systém s širokým využitím. Popis jeho funkcí by dokázal obsáhnout několik kapitol této práce. Dále jsou proto popsány pouze nezbytné informace, které musel autor nabýt, aby byl schopen realizovat řízení třídicího stroje. [15]

2.1 Hardware

CompactRIO se skládá ze tří částí (Obrázek 10). Konkrétně se jedná o procesor běžící na operačním systému reálného času (RT), rekonfigurovatelné programovatelné hradlové pole a vyměnitelné průmyslové vstupně/výstupní moduly. Zatímco RT procesor nabízí spolehlivé a prediktivní chování, velmi dobře provádí výpočty s plovoucí desetinnou čárkou a analýzy, FPGA vyniká v menších úlohách postavených především na vysokorychlostní logice a přesném časování. [15] [16]



Obrázek 10: Architektura kontroléru CompactRIO [16]

RT jednotka

Procesor obsažený v RT jednotce má zajistit spolehlivý a deterministický běh aplikací vytvořených v LabVIEW Real-Time, a také umožňuje záznam dat, trasování jednotlivých spuštění či komunikaci s periferiemi.

Operační systém reálného času (NI Linux Real-Time) se od běžného desktopového operačního systému liší především tím, že dokáže spouštět programy spolehlivě a se specifickým časováním. To je bezpodmínečně nutné v průmyslových aplikacích, ať už se jedná např. o přesné sesazení dílů či jde o bezpečnost práce. [15] [16]

Díky operačnímu systému reálného času lze využít výhod jako je např. realizace více smyček se stejným časováním, detekce zda bylo časování smyčky splněno nebo třeba plnit úlohy v rámci garantovaného nejhoršího možného časového rámce. [15] [16]

FPGA

Rekonfigurovatelné FPGA šasi slouží k přímé komunikaci se zásuvnými moduly. Takto zvolená architektura řídicího systému zajišťuje řízení téměř bez zpoždění. Primárně umožňuje FPGA deterministický přístup RT jednotce k jednotlivým vstupním či výstupním modulům. Velikost šasi, a tedy i tomu odpovídající možný počet zásuvných modulů, závisí na typu CompactRIO. [15] [16]

Vstupní/výstupní moduly

Do FPGA šasi je možné připojit analogové a digitální vstupní/výstupní moduly. Tyto moduly jsou odizolované, obsahují převodní obvody a zajišťují přímé připojení průmyslových senzorů či akčních členů. Firma National Instruments nabízí přes 70 typů modulů, které umožňují připojit ke CompactRIO opravdu velké množství různých druhů senzorů a akčních členů. Například termočlánkové vstupy, připojení sběrnice CAN, výstupy určené k řízení pneumatického systému či měření deformací. [15] [16]

2.2 Software

Při programování CompactRIO je nezbytné počítat s tím, že jde o heterogenní systém (RT procesor a FPGA). Je nezbytné správně vyhodnotit situaci a určit k čemu má vyvíjená aplikace sloužit a podle toho se rozhodnout zda je vhodné vytvořit RT aplikaci či aplikaci běžící na FPGA. [15] [16]

RT programování

Vývojář RT aplikace má na výběr, jaké vývojové prostředí použije. První možností je využít rozšiřující modul LabVIEW Real-Time, který do LabVIEW přidává funkční bloky umožňující např. komunikaci s FPGA, synchronizaci smyček atd. Další možností je programovací jazyk C/C++ v prostředí Eclipse či jiném vývojovém prostředí. Tato práce se zabývá výhradně programováním v LabVIEW Real-Time. [15] [16]

FPGA programování

Jazyk VHDL pracuje se strukturovaným textem a vzhledem k jeho náročné jazykové sémantice znesnadňuje využití plných možnosti FPGA. Proto byl pro programování FPGA v CompactRIO vyvinut rozšiřující modul LabVIEW FPGA, který umožňuje vyšší úroveň abstrakce a vývoj aplikací je tak efektivnější a intuitivnější. [15] [16]

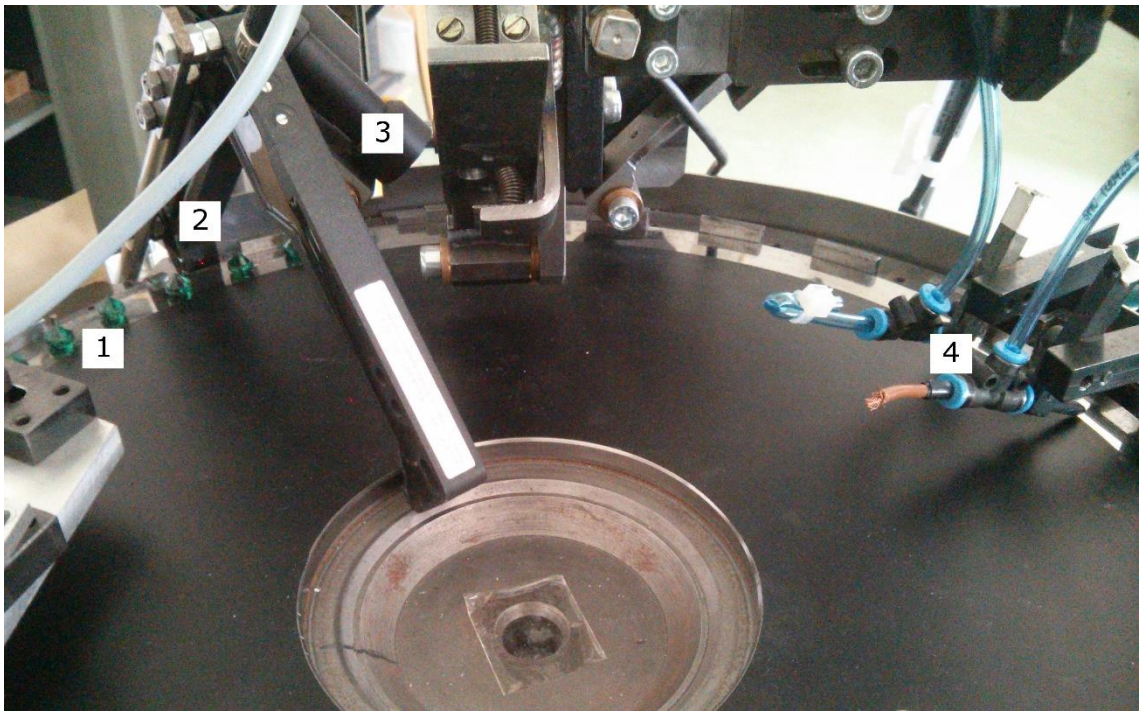
3. Třídicí stroj

Hlavním úkolem diplomové práce je vytvořit algoritmus v prostředí LabVIEW Statechart pro modelový třídicí stroj bižuterních kamenů, který se zaměřuje na jakost kamenů.

3.1 Princip třídění

Jednotlivé kameny (Obrázek 11 č.1) jsou postupně umisťovány na okraj rotujícího disku, který je poháněn servomotorem s IRC (inkrementální rotační snímač). Světelná závora (Obrázek 11 č.2) je umístěna tak, aby jednotlivé kameny při průchodu přerušily její světelný tok. Slouží tedy k zaznamenání příchozích kamenů.

Dále se nad rotačním diskem nachází soustava kamer (Obrázek 11 č.3), která snímá jednotlivé kameny ze čtyř úhlů a po expozici vyšle snímky k vyhodnocení. Po vyhodnocení systém čeká, až bude pozice daného kamene u správné kapsy, aby došlo k aktivaci vzduchové trysky (Obrázek 11 č.4) a sfuku bižuterního kamene do sběrné kapsy. Pokud dojde k špatnému vyhodnocení, zůstává kámen umístěn na rotačním disku, ze kterého je pomocí mechanické zábrany shozen do sběrné nádoby.



Obrázek 11: Náhled na třídicí stroj

3.2 Technické vybavení

Třídicí stroj se skládal z těchto částí.

Servomotor (HF-KP43)

Pohonem rotačního disku je střídavý třífázový servomotor od firmy Mitsubishi.

- Vstup 102 V 2,9 A
- Výstup 400 W
- 3000 otáček/min
- Krytí IP65

Světelná závora (BWL 9090D-L011-S49)

Světelná závora je realizována úhlovým optickým senzorem od firmy Balluff

- Červený laser
- Vlnová délka 640 nm

Pneumatický systém

Ovládání vzduchových trysek je zajištěno pneumatickým systémem od firmy SMC PNEUMATICS.

- 5x ventil SMC SY3140-5MOU-Q
- filtr AFD30-F03
- filtr regulátor EAW2000-F02-X64

Inkrementální rotační snímač

Informace o poloze rotačního disku je získávána z výstupu IRC od firmy LARM.

Řídicí systém (NI cRIO-9074)

Pomyslným mozkem třídicího stroje je řídicí systém od firmy National Instuments CompactRIO s 8 sloty pro vstupní/výstupní moduly.

- RT procesor 400 MHz
- FPGA šasi Spartan-3 2M
- DRAM 128 MB NVM 256 MB
- 2x 10/100BASE-TX Ethernet Port
- Sériový port RS232

Vstupní/výstupní moduly

Nezbytnou součástí řídicího systému musí být vstupní a výstupní moduly.

- NI 9425
- NI 9476
- NI 9474
- NI 9411

4. Zprovoznění třídicího stroje

Před samotným vývojem aplikace muselo být provedeno tzv. oživení techniky, což znamenalo připojení CompactRIO přes jeden z ethernetových portů a nahrání aktuálního softwaru do řídicího systému. Po úspěšném nastavení síťové komunikace a instalaci skrze utilitu NI MAX (Measurement and Automation Explorer) se mohlo přejít k vytvoření projektu.

Počítač, ze kterého je instalován software do CompactRIO a dále je i vyvíjen algoritmus pro aplikaci k řízení třídicího stroje, musí být vybaven tímto softwarem:

- Vývojové prostředí LabVIEW
- Modul LabVIEW Real-Time
- Modul LabVIEW FPGA
- Ovladače NI-RIO

4.1 Projekt

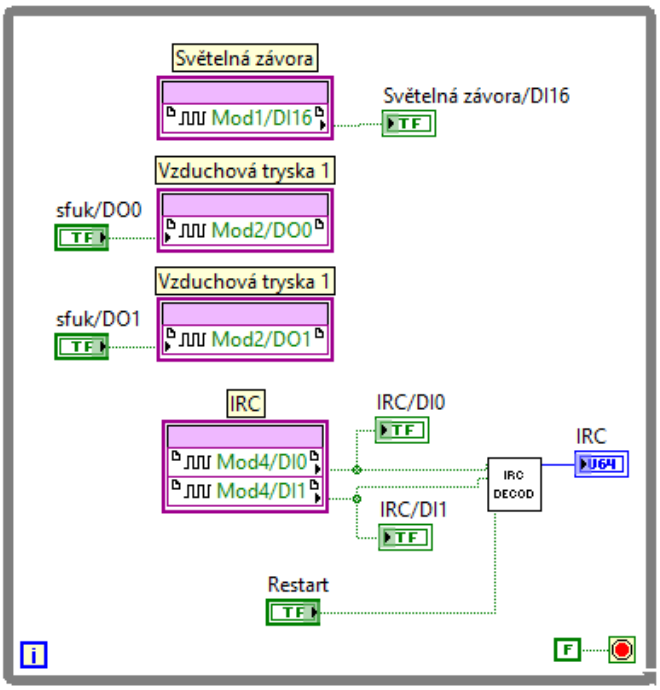
Po vytvoření prázdného projektu se v průzkumníku projektu nachází pouze místní počítač a je třeba přidat nové zařízení CompactRIO. Pokud je během přidávání nového zařízení řídicí systém připojen k hostitelskému počítači, dojde k rozpoznání typu připojeného kontroléru, a také k automatickému rozpoznání připojených modulů.

Kliknutím na název připojeného řídicího systému dojde k rozbalení nabídky obsahující položku šasi a v ní FPGA zařízení.

4.2 Testovací VI

První vytvořené VI (Obrázek 12) slouží k testování správného připojení a funkčnosti jednotlivých komponent. Jelikož jsou zde testovány vstupy a výstupy systému, je vhodné toto testovací VI implementovat na FPGA. Aby VI bylo realizováno na FPGA musí být vytvořeno kliknutím pravým tlačítkem myši na zařízení FPGA->New->VI. Je-li žádané vytvořit RT VI, je ho třeba vytvořit kliknutím pravým tlačítkem myši na kořenový soubor CompactRIO-> New->VI.

Testovací VI obsahuje vstupní a výstupní uzly s připojenými indikátory nebo řídicími prvky. IRC má dva výstupy, aby bylo možné rozpoznat, jakým směrem se otáčí.



Obrázek 12: Testovací VI

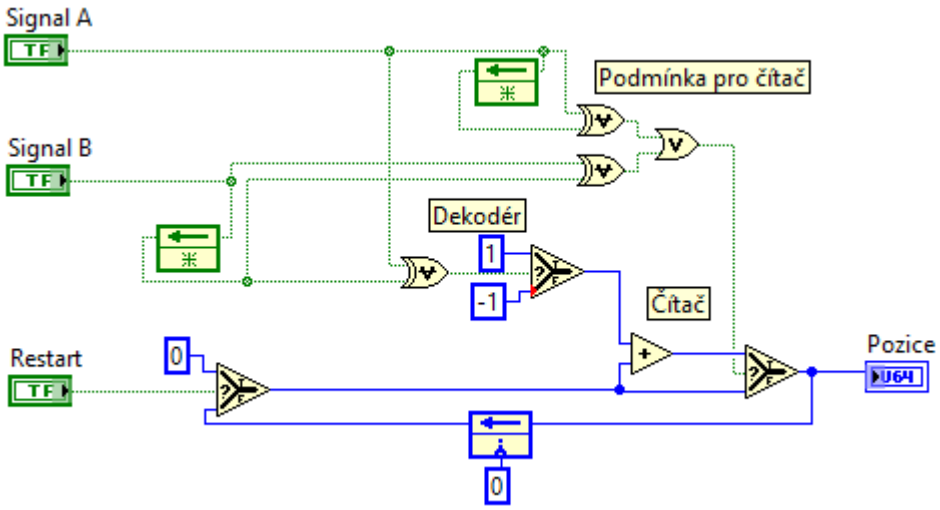
4.3 SubVI IRC DECOD

V rámci testovacího VI byl také vytvořen čítač a dekodér výstupních signálů IRC. Výstupní signály jsou navzájem fázově zpožděné o 90°, aby bylo možné rozeznat směr otáčení. Kódování pro otáčení po směru hodinových ručiček je zobrazeno v Tabulce 1.

Tabulka 1: Kódování pro otáčení po směru hodinových ručiček

Fáze	Signál A	Signál B
1	0	0
2	0	1
3	1	1
4	1	0

K dekódování a čítání bylo vytvořeno SubVI (podprogram) s názvem IRC DECOD (Obrázek 13). Dekodér tvoří exkluzivní disjunkce aktuální hodnoty signálu A a minulé hodnoty signálu B. Je-li výsledkem hodnota TRUE dochází k otáčení po směru hodinových ručiček a naopak. Rozhodnutí, zda má být čítač navýšen případně snížen, je dáno logickým součtem exkluzivních disjunkcí aktuální a minulé hodnoty signálu A a aktuální a minulé hodnoty signálu B.

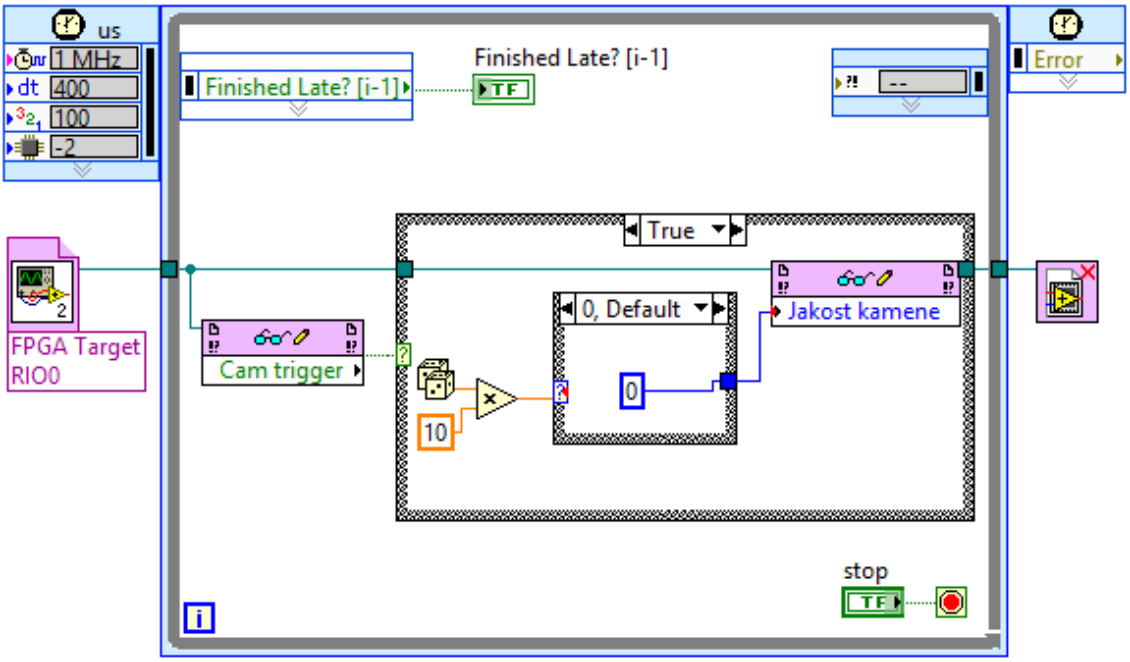


Obrázek 13: Dekodér a čítač

4.4 Vyhodnocení jakosti

Získání obrazu kamerami a jeho zpracování nebylo autorovi firmou Preciosa a.s. umožněno a není ani tématem práce. Proto bylo vytvořeno VI (Obrázek 14) simulující výstup zpracování obrazu, tedy třídu jakosti bižuterního skla.

Zpracování obrazu by probíhalo v RT aplikaci, z toho důvodu bylo i simulační VI vytvořeno na RT zařízení.



Obrázek 14: Simulace zpracovaného výstupu jakosti

Komunikace mezi RT a FPGA aplikací probíhá pomocí odkazu na FPGA VI (Caller VI). Uvnitř časovatelné smyčky se nachází struktura CASE, na jejíž selektor je přiveden výstup stavového automatu *Cam trigger*. Tento výstup je nastaven na hodnotu TRUE vždy, když se bižuterní kámen nachází pod soustavou kamer. Pokud tomu tak je, dojde ke generování náhodného čísla od 0 do 2 znázorňující třídu jakosti skla daného bižuterního kamene. Tato hodnota je poté přivedena na vstup stavového automatu.

5. Návrh řídicího algoritmu

Řídicí algoritmus je realizován na FPGA (rychlé výpočty a deterministické úlohy). Z tohoto důvodu musí být i stavový automat, tedy soubor s příponou .lvsc, vytvořen v průzkumníku projektů pod zařízením FPGA.

V první řadě je nezbytné nastavit vlastnosti stavového automatu. Ideální by bylo, aby stavový automat využíval jednocyklovou smyčku. Bohužel jednocyklová smyčka nepodporuje vstupní/výstupní uzly FPGA zařízení nacházející se v CompactRIO 9074. Proto je nastaveno využíti cyklu WHILE.

5.1 Vstupy, výstupy a stavová data

Před návrhem stavového diagramu je třeba definovat vstupy, výstupy a vnitřní data stavového automatu.

Vstupy

Vstupy stavového automatu jsou pouze tři a zajišťují tyto signály:

- signál ze světelné závory
- dekódovaný signál z inkrementálního rotačního snímače
- jakost tříděného kamene (zpracovaný výstup kamery)

Výstupy

Pro funkci třídicího stroje postačují tři výstupy, avšak pro programování a odlaďování bylo použito výstupů více. Jsou to výstupy pro tyto signály:

- signál pro aktivaci vzduchové trysky 1
- signál pro aktivaci vzduchové trysky 2
- signál aktivující kamery

Stavová data

Stavová data tvoří informace o pozicích kamenů u světelné závory či pod kamerami, o jakosti skla a o hodnotě čítače IRC, kdy má být aktivována vzduchová tryska. Dále jsou použita jako pomocné proměnné, sloužící k aktivaci některých přechodů.

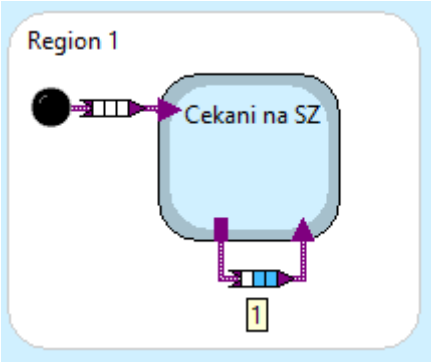
Před tvorbou stavového diagramu je vhodné upravit *CustomDataDisplay.vi* určené k odladění stavového automatu. Avšak u FPGA je mód odlaďování zakázaný, proto není potřeba *CustomDataDisplay.vi* řešit.

5.2 Stavový diagram

Stavový diagram (Příloha 1) se skládá z jednoho stavu a jednoho superstavu. V tomto prvním stavu dojde k vymazání obsahu všech použitých front. Přechod do superstavu je podmíněn přerušením světelné závory prvním kamenem. V superstavu se nachází celkem pět ortogonálních regionů, přičemž každý region slouží k řízení jiné části procesu třídění. Toto řešení pomocí ortogonálních regionů je nezbytné, protože jinak by nebylo možné třídit řadu bižuterních kamenů s malým rozestupem a muselo by se vždy čekat, až jeden kámen samostatně projde celým třídicím cyklem.

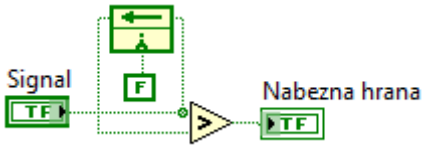
Region 1

V Regionu 1 (Obrázek 15) se nachází jeden stav, který slouží k zaznamenání průchodu bižuterního kamene světelnou závorou. Přesněji řečeno má přechod 1 nastavenou podmínku aktivace tak, že čeká na náběžnou hranu vstupního signálu světelné závory. Dále obsahuje také akci sloužící k uložení aktuálního stavu čítače IRC do fronty *FIFO SZ*. Po vykonání akce, čeká stav opět na náběžnou hranu signálu.



Obrázek 15: Region 1

Náběžná hrana signálu světelné závory je kontrolována pomocí vytvořeného subVI (Obrázek 16).



Obrázek 16: Kontrola náběžné hrany

Region 2

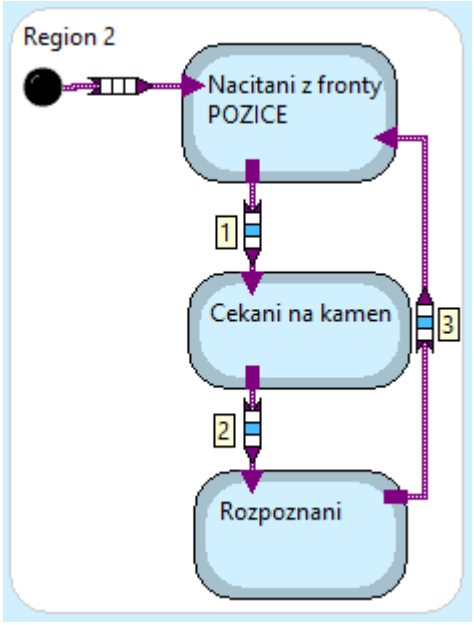
Druhý region (Obrázek 17) zajišťuje načtení hodnoty z fronty *FIFO SZ* a kontroluje, zda je kámen pod soustavou kamer. Jestliže tomu tak je, vyšle trigger pro

spuštění kamer a uloží aktuální stav čítače IRC do fronty *FIFO Pozice u kamery*. Nakonec uloží hodnotu ze vstupu, na který je přivedena hodnota vyjadřující jakost skla daného kamene, do fronty *FIFO Jakost.*

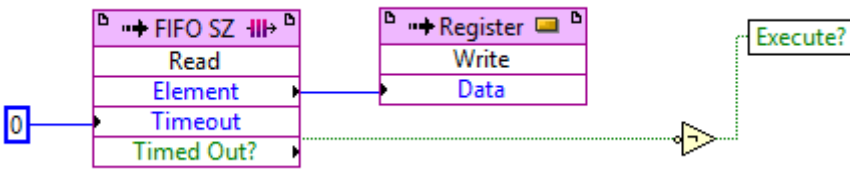
První stav *Nacteni z fronty POZICE* řeší vyčítání z fronty. Je třeba ošetřit, aby byla vyčtena pouze jedna nenulová hodnota, a tato hodnota byla uložena do stavových dat. Funkční blok pro čtení z fronty má výstup *Timed Out?*. Pokud ve frontě nejsou žádná data, nabývá tento výstup hodnoty TRUE.

Funkční blok pro čtení z fronty je umístěn v podmínce aktivace (Obrázek 18) statické reakce, kde je neustále kontrolován výstup *Timed Out?*. Jestliže tento výstup nabude hodnoty FALSE, dojde k vyčtení hodnoty z fronty. Proto je nezbytné ještě v podmínce aktivace tuto hodnotu uložit. Z podmínky aktivace není možný zápis do stavových dat, a tak musel být vytvořen registr, do kterého se hodnota uloží, a poté se v akci statické reakce z tohoto registru vyčte a vloží do stavových dat.

Přechod do druhého stavu je podmíněn ukončením statické reakce prvního stavu.



Obrázek 17: Region 2



Obrázek 18: Vyčítání z fronty

Druhý stav *Cekani na kamen* obsahuje statickou reakci. Podmínkou aktivace je rovnost aktuální hodnoty čítače IRC a načtené hodnoty z fronty *FIFO SZ* plus 2000. To znamená, že je kámen pod kamerami. Po splnění této podmínky dojde k aktivaci výstupu zajišťující zaslání žádosti pro zpracování obrazu a načtení aktuální hodnoty čítače IRC do fronty *FIFO Pozice u kamery*.

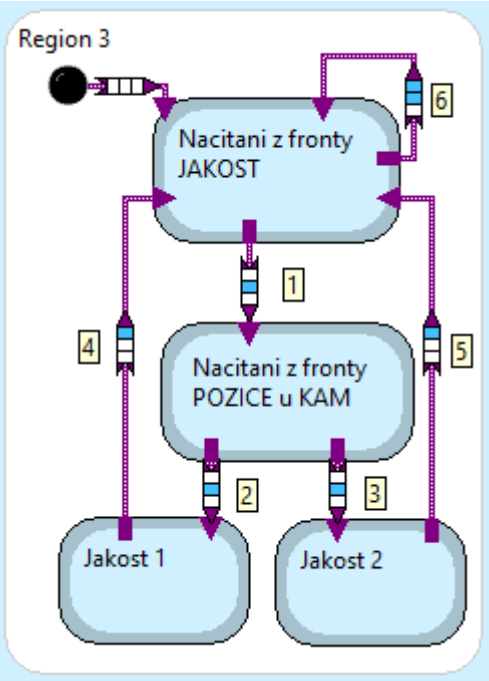
Přechod 3 je podmíněn aktivním výstupem zajišťující zaslání žádosti pro zpracování obrazu.

Statická reakce třetího stavu *Rozpoznani* má podmínku aktivace, které kontroluje, zda byla načtena vstupní hodnota vyjadřující jakost bižuterního skla daného kamene. V akci je pak tato hodnota uložena do fronty *FIFO Jakost* a je deaktivován výstup určený k aktivaci kamery.

Přechod 4 je podmíněn ukončením statické reakce stavu *Rozpoznani*.

Region 3

Vnořené stavy regionu 3 (Obrázek 19) slouží k načtení hodnot z front *FIFO Jakost* a *FIFO Pozice u kamery*, roztřídění podle jakosti a vložení hodnoty získané z fronty *FIFO Pozice u kamery* do fronty *FIFO Sfuk 1* nebo *FIFO Sfuk 2*.



Obrázek 19: Region 3

První stav *Nacitani z fronty JAKOST* má na starosti načíst hodnotu z fronty *FIFO Jakost* a uložit ji do stavových dat. Ošetření načtení prázdné hodnoty je stejné jako

u předchozího regionu. Tedy v podmínce aktivace dojde k načtení hodnoty z fronty do registru a v akci se poté uloží hodnota z registru do stavových dat.

Přechod 1 je podmíněn tím, že se stavová data s hodnotou jakosti rovnají číslu jedna nebo dva. Pokud je jakost vyhodnocena číslem 0 (špatné vyhodnocení či nedostačující jakost), není třeba přecházet do stavu *Nacitani z fronty POZICE u KAM*. Dojde k provedení přechodu 6 a nové aktivaci prvního stavu.

Druhý stav *Nacitani z fronty POZICE u KAM* vyčítá hodnotu z fronty *FIFO Pozice u kamery*. Princip výčtu z fronty je stejný jako v minulém případě.

Podmínkou aktivace přechodu 2 je, že musí být načtena hodnota čítače IRC z fronty *FIFO Pozice u kamery* ve stavových datech a hodnota jakosti se rovná jedné.

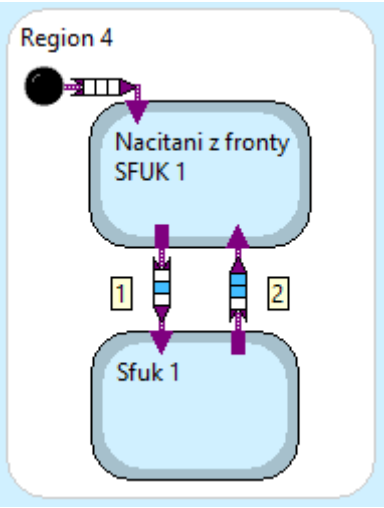
Vstupní akce stavu *Jakost 1* vkládá hodnotu čítače IRC, tedy pozici kamene pod kamerami do fronty *FIFO Sfuk1*.

Přechod 4 nemá podmínku aktivace, tudíž se vykoná ihned po provedení vstupní a výstupní akce stavu *Jakost 1*. Akce přechodu nuluje stavová data, do kterých se ukládá pozice kamene pod soustavou kamer.

Přechod 3, stav *Jakost 2* a přechod 5 fungují stejně jako přechod 2, stav *Jakost 1* a přechod 4 s tím rozdílem, že je zde ukládána pozice kamene pod soustavou kamer s jakostí 2 do fronty *FIFO Sfuk2*.

Region 4

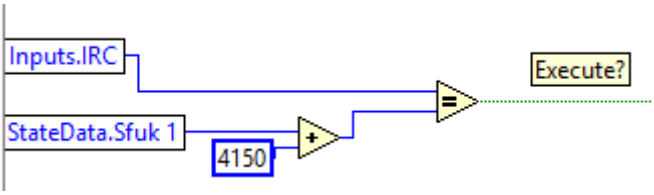
Region 4 (Obrázek 20) se skládá pouze ze dvou stavů a má na starosti výčet z fronty *FIFO Sfuk1*, kontrolu zda je kámen před správnou kapsou a aktivaci vzduchové trysky 1.



Obrázek 20: Region 4

Statická reakce prvního stavu *Nacitani z fronty SFUK 1* zajišťuje načtení hodnoty z fronty *FIFO Sfuk1*. Podmínkou aktivace přechodu 1 je načtená hodnota z fronty do stavových dat.

Druhý stav *Sfuk 1* obsahuje statickou reakci. Podmínkou aktivace je, že se daný kámen nachází před kapsou 1 (Obrázek 21). Pokud je podmínka splněna, dojde k nastavení hodnoty TRUE na výstupu aktivující vzduchovou trysku 1. Ve výstupní akci tohoto stavu je umístěn funkční blok zajišťující zpoždění 100 ms, aby se kámen stihl sfouknout do kapsy, než dojde k deaktivaci vzduchové trysky.



Obrázek 21: Podmínka aktivace vzduchové trysky 1

Podmínkou aktivace přechodu 2 je nastavená hodnota TRUE na výstupu aktivující vzduchovou trysku 1. Tento výstup se nastaví na hodnotu FALSE při vstupu do prvního stavu *Nacitani z fronty SFUK 1* a tím je uzavřen průchod vzduchu tryskou 1.

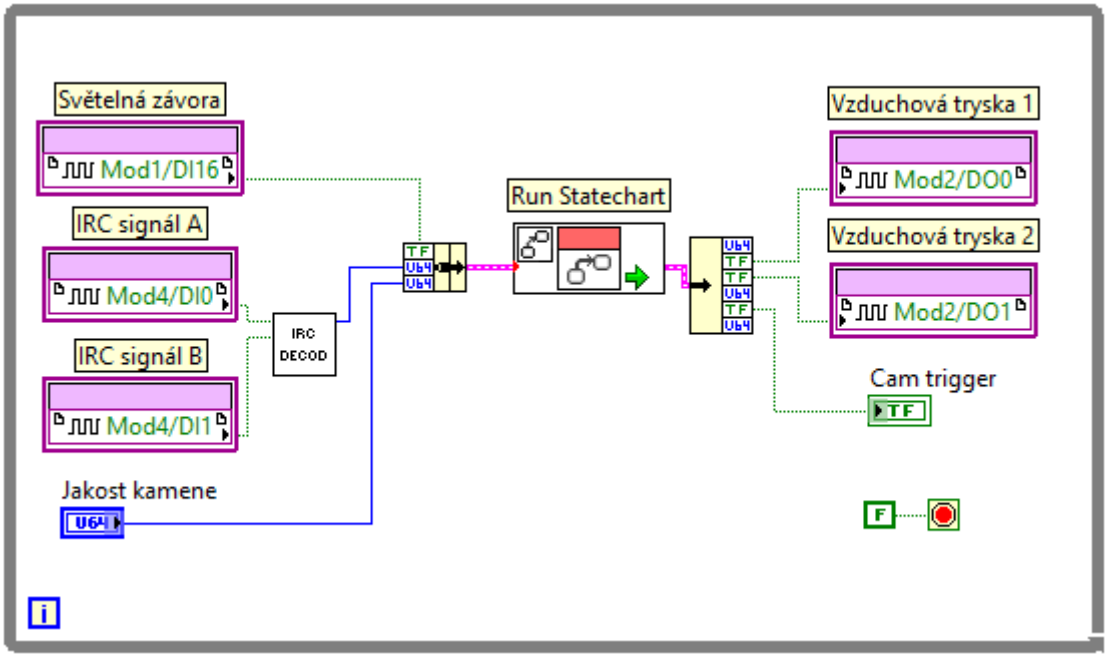
Region 5

Poslední region plní v podstatě stejnou funkci jako region 4 s tím rozdílem, že obsluhuje frontu FIFO Sfuk2 a vzduchovou trysku 2.

5.3 Caller VI

Caller VI (Obrázek 22) je vytvořeno a spouštěno na stejném cíli jako stavový automat, v tomto případě tedy na FPGA. Stavový automat je nastavený na využití smyčky WHILE, proto se i celý blokový diagram v této smyčce nachází.

Caller VI se skládá z již zmiňované smyčky WHILE, funkčního bloku *Run Statechart,* vstupních a výstupních FPGA uzlů a funkčních bloků sloužících ke spojení a rozpojení vstupů a výstupů.



Obrázek 22: Caller VI

6. Návrh řídicího algoritmu bez využití LabVIEW Statechart

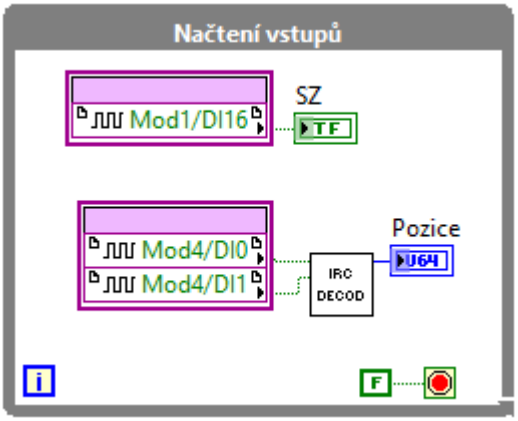
Aby bylo možné objektivně posoudit výhody a nevýhody LabVIEW Statechart pro vývoj řídicího algoritmu, bylo nezbytné vytvořit algoritmus pro stejný proces bez využití tohoto modulu.

Struktura algoritmu (Příloha 2) je velmi podobná. Celý blokový diagram se nachází ve struktuře sekvence (FLAT SEQUENCE). V prvním rámci se nachází funkční bloky zajišťující vynulování všech použitých front FIFO a v druhém rámci je 6 paralelních smyček, které obsluhují dané části procesu třídění.

Některé ze smyček zde budou popsány, aby bylo zřetelné rozdílné řešení určitých problémů.

6.1 Načtení vstupů

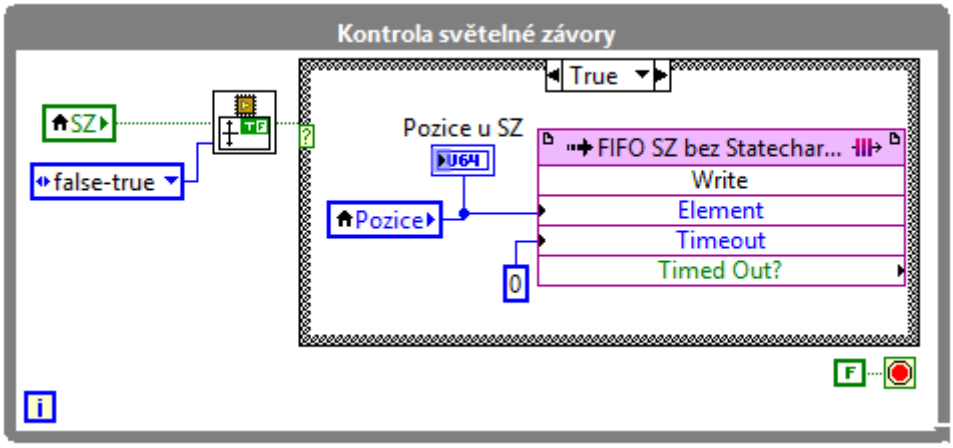
Hodnota ze vstupu světelná závora a aktuální hodnota čítače IRC jsou načítány v samostatné smyčce. V programu jsou dále tyto hodnoty získávány pomocí lokálních proměnných, což mimo jiné zajišťuje větší přehlednost blokového diagramu.



Obrázek 23: Smyčka načtení vstupů

6.2 Kontrola světelné závory

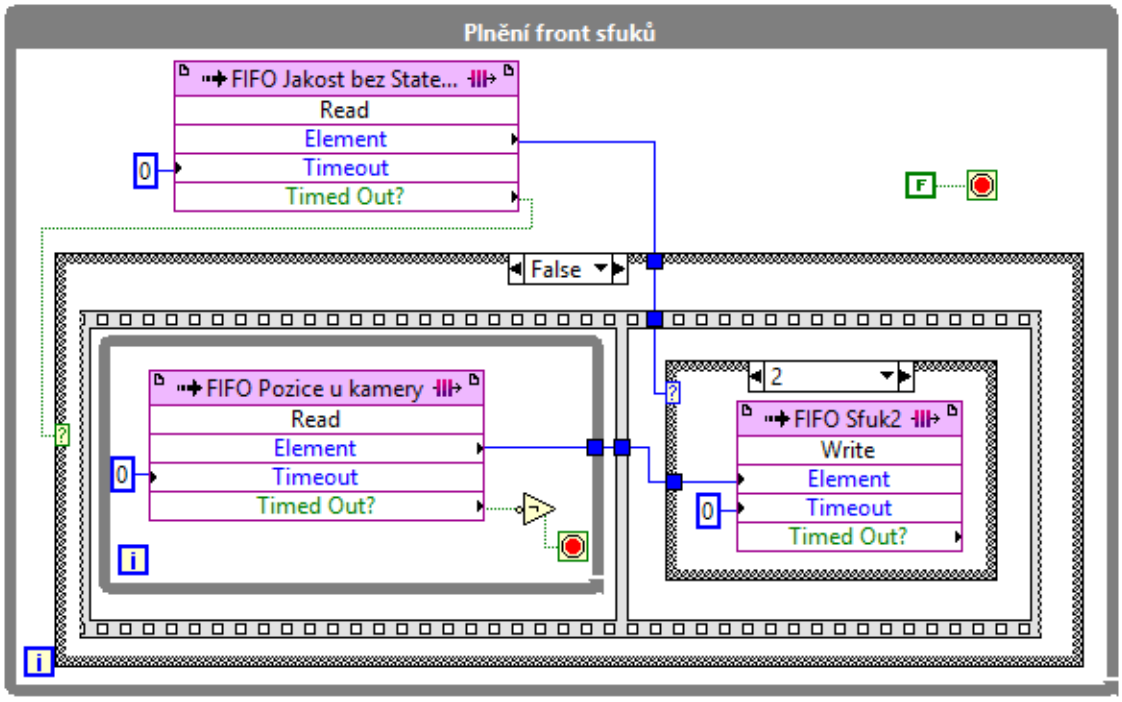
Obsluha světelné závory (Obrázek 24) je zajištěna strukturou CASE, na jejíž selektor je přiveden výstup z funkčního bloku, který zajišťuje detekci náběžné hrany. V případě, že je přivedena hodnota TRUE dojde k uložení aktuální hodnoty čítače IRC do fronty *FIFO SZ*.



Obrázek 24: Smyčka zajišťující kontrolu světelné závory

6.3 Roztřídění do front vzduchových trysek

Smyčka (Obrázek 25) sloužící k roztřídění do front vzduchových trysek obsahuje struktury CASE, WHILE a FLAT SEQUENCE.



Obrázek 25: Smyčka zajišťující roztřídění

Ošetření výčtu prvků z front je zde o něco jednodušší než při využití LabVIEW Statechart. Není zde potřeba použít registry, stačí využít vlastností struktury CASE.

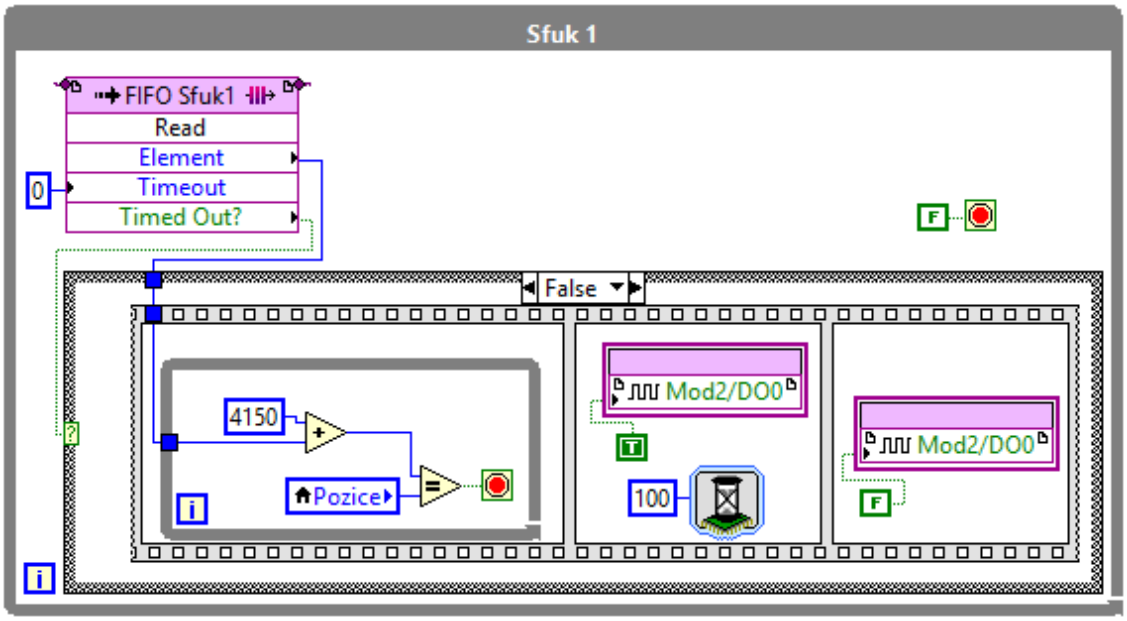
Jako první je v této smyčce kontrolován výstup *Timed Out?* fronty *FIFO Jakost*. Jestliže se ve frontě nachází hodnota znázorňující jakost, dostane se program do prvního rámce struktury sekvence a tam čeká ve smyčce WHILE, dokud není načtena hodnota

z fronty *FIFO Pozice u kamery*. Je-li hodnota načtena, dochází k ukončení smyčky WHILE a k přechodu do druhého rámce. Druhý rámec obsahuje strukturu CASE, na jejíž selektor je přivedena hodnota ukazatele jakosti a tím se rozhodne, do jaké fronty bude zapsána hodnota čítače IRC z fronty *FIFO Jakost*.

6.4 Aktivace vzduchové trysky

Aktivace vzduchových trysek (Obrázek 26) je řešena opět pomocí stejných struktur jako v předchozím případě.

Nejdříve dochází k vyčtení hodnoty z fronty *FIFO* Sfuk1. Dále se v prvním rámci struktury sekvence nachází smyčka WHILE, ve které je kontrolována pozice bižuterního kamene. V okamžiku, kdy se kámen nachází před kapsou jedna, dochází k ukončení smyčky a k přechodu do druhého rámce. Tento rámec slouží k nastavení výstupu zajišťujícího aktivaci vzduchové trysky. Musí se zde také nacházet funkční blok zajišťující zpoždění 100 ms, aby se daný kámen stihl sfouknout do kapsy. V posledním třetím rámci dochází k deaktivaci vzduchové trysky.



Obrázek 26: Smyčka zajišťující aktivaci vzduchové trysky

7. Srovnání zápisu algoritmů a vývojových prostředí

Základní struktura algoritmu pro řízení třídicího stroje bude ve všech programovacích jazycích či vývojových prostředích velmi podobná, ať už se jedná o LabVIEW nebo LabVIEW Statechart využívající řídicí systém CompactRIO nebo o úplně jiný řídicí systém např. PLC. Nezáleží, zda bude algoritmus tvořený strukturovaným textem či blokovým diagramem. Vždy se program k řízení této třídičky bude muset skládat z několika paralelních cyklů, které budou obsluhovat danou část třídicího procesu.

Tato práce se zaměřuje především na srovnání vývoje algoritmu s využitím modulu LabVIEW Statechart a bez využití tohoto modulu.

Srovnání zápisu algoritmů

Algoritmus vytvořený bez modulu LabVIEW Statechart se skládá ze základních struktur LabVIEW. Jednotlivé stavy jsou realizovány pomocí struktur FLAT SEQUENCE, WHILE a CASE. Ortogonální regiony jsou nahrazeny paralelními smyčkami WHILE. Podmínky aktivace a statické reakce jsou řešeny pomocí struktury CASE.

Přehlednost

Hlavní výhodou prostředí LabVIEW Statechart je rozhodně přehlednost finálního stavového diagramu. Tato skutečnost je patrná ihned po zhlédnutí obou algoritmů (Příloha 1) (Příloha 2).

Ve vývojovém prostředí LabVIEW Statechart je možné, díky vysoké úrovni abstrakce, integrovat do jednoho stavu vstupní akci, výstupní akci a několik statických reakcí. Ty se od sebe mohou lišit jak zdrojovým kódem, který vykonají, tak podmínkou aktivace. Tato skutečnost je velkou výhodou u rozsáhlých a sofistikovaných algoritmů.

Další výhodou LabVIEW Statechart, související s přehledností, je vizuální popis chování systému. Vývojář má možnost vidět všechny funkce a veškeré možné stavy systému či aplikace, a může tak lépe předejít uváznutí.

Využití prostředků FPGA

Při návrhu řídicí aplikace je důležité sledovat využití prostředků, které jsou poskytovány řídicím systémem. V tomto případě je sledováno využití LUT (Lookup table) a registrů FPGA.

Využití prostředků FPGA aplikací vytvořené v LabVIEW Statechart:

- registry: 38,9% (15953 z 40960)
- LUT: 51,3% (21028 z 40960)

Využití prostředků FPGA aplikací vytvořené v LabVIEW:

- registry: 31,2% (12764 z 40960)
- LUT: 34,5% (14145 z 40960)

Aplikace vytvořená prostřednictvím LabVIEW Statechart využívá větší množství zdrojů FPGA než aplikace bez Statechartu, zejména LUT. Důvodem větší náročnosti je mimo jiné univerzálnost jednotlivých komponent modulu.

Během jedné iterace je kontrolováno poměrně velké množství akcí (vstupní, výstupní a statická) a podmínek aktivací, které nejsou využity.

Tato skutečnost má vliv i na celkovou velikost zdrojového kódu, který musí být před nahráním do FPGA zkompilován do tzv. bitfilu.

Rychlost kompilace

Rychlost kompilace zdrojového kódu závisí na výkonu počítače a na velikosti a složitosti překládaného kódu. Kompilace byla prováděna na této sestavě:

- OS: Windows 10 Home x64
- CPU: Intel Core i5 2430M 2.40 GHz
- RAM: 6 GB

Čas kompilace:

- s využitím LabVIEW Statechart 32:04 min
- bez využití LabVIEW Statechart 15:18 min

Chybovost třídění

Řízení třídicího procesu pomocí aplikace vytvořené v prostředí LabVIEW bez modulu Statechart probíhalo plynule i při vyšších rychlostech otáčení rotačního disku. To však nelze tvrdit o řízení s využitím modulu LabVIEW Statechart.

Při nižších rychlostech otáčení (10 ot/min) algoritmus vytvořený v LabVIEW Statechart stíhal kameny sfukovat včas, avšak při vyšších rychlostech (25 ot/min) docházelo u některých kamenů k opožděné aktivaci vzduchové trysky.

Tento problém je s největší pravděpodobností způsoben souběžností vnořených stavů ortogonálních regionů. V každém regionu může být aktivní pouze jeden stav. Tento stav může být souběžný s aktivním stavem dalšího ortogonálního regionu.

Souběžnost stavů znamená, že jsou tyto stavy vykonány během jedné iterace stavového automatu, ne však zároveň. Pořadí vstupu do ortogonálních regionů je dáno abecedně podle názvu regionů.

LabVIEW Statechart nepodporuje paralelní chod stavů, což je zásadní problém pro řízení třídicího stroje.

8. Závěr

Závěrem této diplomové práce je provedeno vyhodnocení vhodnosti modulu LabVIEW Statechart pro vývoj řídicího algoritmu třídicího stroje. Řízení technologického procesu bylo provedeno pomocí algoritmu navrženého ve vývojovém prostředí LabVIEW a v tomtéž prostředí s využitím modulu LabVIEW Statechart. V obou případech byl algoritmus implementován na řídicím systému firmy National Instruments CompactRIO. V závěrečné fázi byly vyhodnoceny výhody a nevýhody jednotlivých algoritmů. Všechny cíle diplomové práce byly splněny.

Obsluha třídicího stroje se skládá celkem ze tří aplikací. První aplikace, běžící na osobním počítači, slouží k řízení servomotoru. Druhá aplikace je spuštěna na procesoru reálného času a zajišťuje simulaci zpracování obrazu. Třetí aplikace je určena k řízení celého procesu třídění a je realizována na FPGA. Poslední z těchto aplikací byla vytvořena v prostředí LabVIEW Statechart.

Stavový diagram se skládá z jednoho stavu (proběhne na začátku procesu) zajišťujícího rutinní operace a superstavu obsahujícího pět ortogonálních regionů. Každý region má na starosti obsluhu určité části třídicího procesu, ať už se jedná o vyhodnocování signálu ze světelné závory či aktivování vzduchových trysek. Komunikace mezi regiony je zprostředkována frontami FIFO. Při práci s frontami bylo třeba zabránit výčtu z prázdné fronty. Kontrola, zda fronta obsahuje hodnotu a uložení této hodnoty, musí proběhnout v podmínce aktivace. Protože podmínka aktivace neumožnuje zápis do stavových dat, musí být použit registr z FPGA.

Při vývoji algoritmu bez modulu LabVIEW Statechart byly použity základní struktury LabVIEW. Jednotlivé části třídicího procesu řídí pět paralelních smyček WHILE. Stavy jsou tvořeny strukturami CASE a FLAT SEQUENCE. Řešení výčtu hodnot z front se obejde bez užití registrů.

V poslední části práce je porovnána přehlednost algoritmu, náročnost na zdroje, rychlost kompilace a chybovost třídění obou algoritmů. Hlavní výhodou při využití modulu LabVIEW Statechart je přehlednost stavového diagramu, což je významným přínosem při řešení aplikací s velkým počtem stavů. Další výhodou je zápis algoritmů na vyšších úrovních abstrakce. Tím je umožněn náhled na každý možný stav systému a snižuje se tak riziko uváznutí či jiné neočekávané události. Větší náročnost na zdroje FPGA u algoritmu vytvořeném v LabVIEW Statechart vyplývá z komplexnosti jednotlivých komponent.

Největším problémem pro řízení technologického procesu je skutečnost, že modul LabVIEW Statechart nepodporuje paralelní chod stavů, ale pouze souběžný chod. Díky tomu dochází při vyšších otáčkách servomotoru ke zpožděným reakcím vzduchových trysek a některé bižuterní kameny nejsou správně roztříděny. Jediná možnost, jak dosáhnout paralelního chodu stavů, je vytvořit pro každou část obsluhy třídicího procesu samostatný stavový diagram. To se ve výsledku jeví jako velmi neefektivní, hlavně při nutnosti velkého množství paralelních smyček. Řešení, jak zrychlit algoritmus a minimalizovat tak chyby třídění, by bylo možné vytvořením stavového automatu spouštěného v jednocyklové smyčce. To však není v tomto případě reálné, protože vstupní a výstupní FPGA uzly CompactRIO 9074 nejsou jednocyklovou smyčkou podporovány.

U algoritmu vytvořeného v LabVIEW bez modulu Statechart je plně využito hardwarového paralelismu, který FPGA umožňuje a třídicí proces funguje i při vyšších otáčkách servomotoru.

Při nasazení třídicího stroje do provozu by bylo nezbytné ošetřit přetečení čítače IRC a vynechání třídění bižuterních kamenů, které by se nacházely na rotačním disku příliš blízko u sebe. Aby bylo možné přesně definovat efektivitu a využitelnost modulu LabVIEW Statechart, muselo by se do budoucna provést širší spektrum testů a vytvořit například algoritmus určený k měření a záznamu dat či implementaci komunikačního protokolu. Z výsledků porovnání algoritmů a z praktické zkušenosti návrhu těchto řídicích algoritmů nemůže autor užití modulu LabVIEW Statechart pro řízení technologického procesu třídění bižuterních kamenů doporučit.

Seznam použité literatury

[1] LAUER, Tomáš. *Studium metod řízení technologického procesu v prostředí LabVIEW Statechart*. Liberec, 2015. Semestrální projekt.

[2] National Instruments: *Statechart Code Generation Page*.[online].2013 [cit. 2016-01-04]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103F-01/lvsc/sc_codegen/

[3] National Instruments: *Synchronous and Asynchronous Statecharts*. [online]. 2013 [cit. 2016-01-05]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103F-01/lvscconcepts/sc_c_syncasync/

[4] National Instruments: *Input,Output and State Data*.[online].2010[cit. 2016-01-05]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103D-01/lvscconcepts/sc_c_iodata/

[5] National Instruments: *States*.[online]. 2013 [cit. 2016-01-09]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103F-01/TOC4.htm

[6] National Instruments: *Introduction to UML Terminology in the LabVIEW Statechart Module*.[online]. 2008 [cit. 2016-1-10]. Dostupné z: http://www.ni.com/white-paper/7413/en/#toc5

[7] National Instruments: *Transitions*.[online]. 2013 [cit. 2016-01-12]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103F-01/lvscconcepts/sc_c_trans/

[8] National Instruments: *Triggers, Guards, and Actions*.[online]. 2013 [cit. 2016-02-15]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103F-01/lvscconcepts/sc_c_tga/

[9] National Instruments: *Pseudostates and Connectors*.[online]. 2013 [cit. 2016-02-18]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103F-01/lvscconcepts/sc_c_pseudocon/

[10] National Instruments: *Trigger Queues*.[online]. 2013 [cit. 2016-02-19]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103F-01/lvscconcepts/sc_c_trigqueue/

[11] National Instruments: *Using a Caller VI to Execute a Statechart*.[online].[2013] [cit. 2016-02-19]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103F-01/lvscconcepts/sc_c_callervi/#sync

[12] National Instruments: *Debugging Statecharts*.[online]. 2009 [cit. 2016-03-02]. Dostupné z: http://zone.ni.com/reference/en-XX/help/372103C-01/TOC10.htm

[13] KRETSCHMEROVÁ, Lenka a Jaroslav VLACH. *Programování v LabVIEW v příkladech*. Liberec: Skriptum TUL, 2014. ISBN 978-80-7372-167-2.

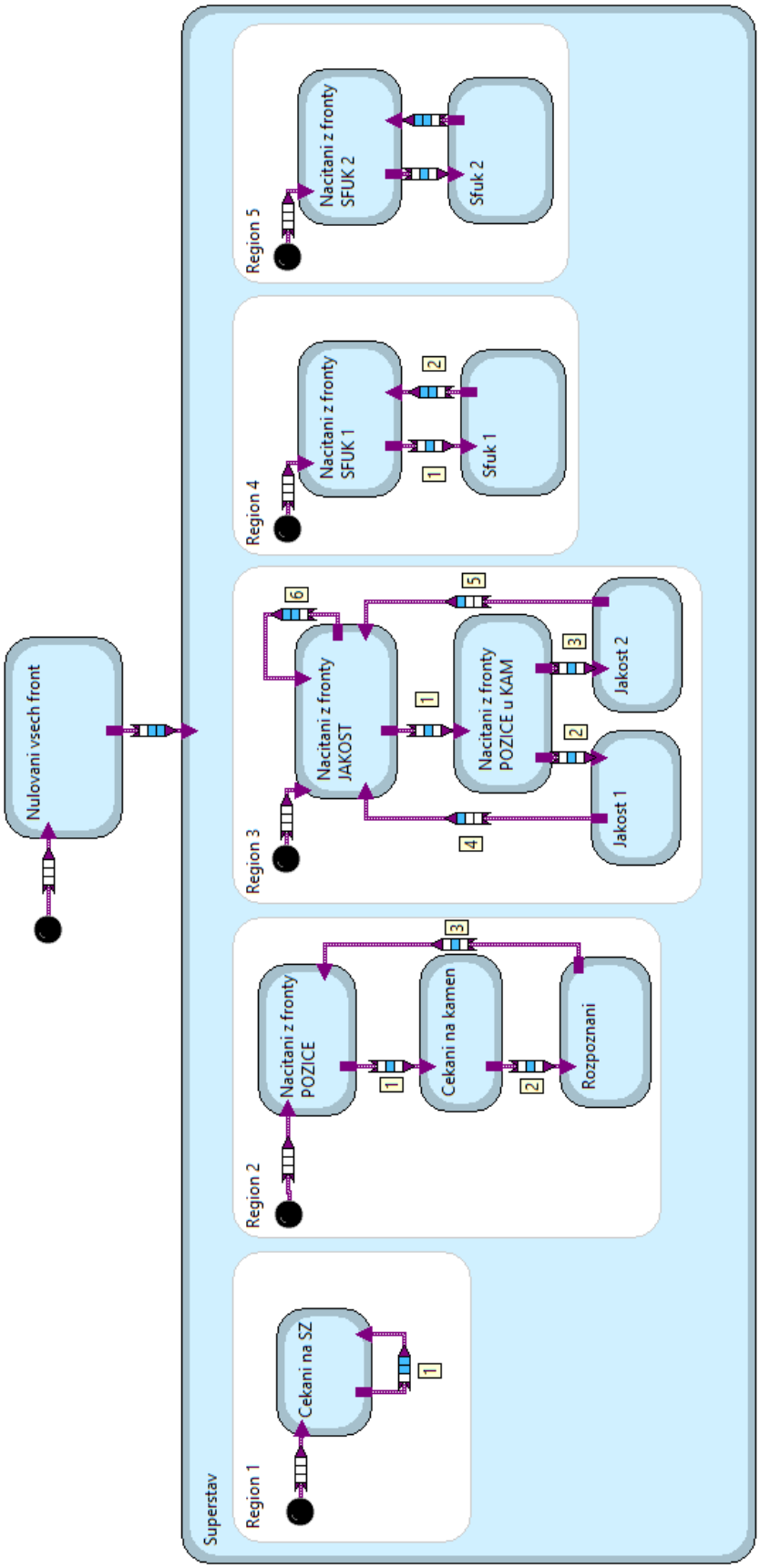
[14] National Instruments: *Transferring Data in LabVIEW Statecharts*.[online]. 2009 [cit. 2016-03-05]. Dostupné z: http://www.ni.com/tutorial/7812/en/

[15] National Instruments: *C/C++ Embedded System Design Tools*.[online]. 2016 [cit. 2015-03-06]. Dostupné z: http://www.ni.com/white-paper/14623/en/

[16] NATIONAL INSTRUMENTS. *NI LabVIEW for CompactRIO Developer's Guide: Recommended LabVIEW Architectures and Development Practices for Control and Monitoring Applications*[online]. 2014 [cit. 2016-03-06]. Dostupné z: http://www.ni.com/pdf/products/us/fullcriodevguide.pdf

Přílohy

Příloha 1: Stavový diagram



Příloha 2: Blokový diagram bez využití LabVIEW Statechart