

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  
FAKULTA ELEKTROTECHNICKÁ  
KATEDRA ELEKTRICKÝCH POHONŮ A TRAKCE

## Rozhraní otáčkového čidla

### Bakalářská práce

Autor práce: Kuanysh Stikeyev

Vedoucí práce: Ing. Jan Bauer, Ph.D.

Studijní program: Elektrotechnika, energetika a management

Obor: Aplikovaná elektrotechnika

České vysoké učení technické v Praze  
Fakulta elektrotechnická

katedra elektrických pohonů a trakce

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Kuanysh Stikeyev**

Studijní program: Elektrotechnika, energetika a management

Obor: Aplikovaná elektrotechnika

Název tématu: **Rozhraní otáčkového čidla**

Pokyny pro vypracování:

- 1) Proveďte rešerši otáčkových čidel pro elektrické pohony
- 2) Proveďte rešerši možností získání informace o rychlosti z čidla
- 3 Vytvořte program v jazyce VHDL, který bude zpracovávat informace z IRC
- 4) Zvolte a popište vhodné komunikační rozhraní pro komunikaci mezi procesorem a IRC interfacem

Seznam odborné literatury:

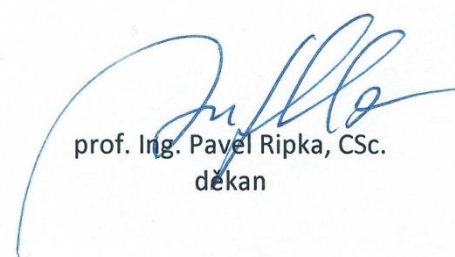
- [1] Javůrek, J., Regulace moderních elektrických pohonů, Grada Publishing, a.s., 2003
- [2] <http://www.xilinx.com/tools/microblaze.htm>
- [3] Pinker, J., Poupa, M., Číslicové systémy a jazyk VHDL, BEN, 2009
- [4] Haskell, R. E., Hanna, D. M., Digital Design Using FPGA Boards, LBE Books, 2009

Vedoucí: Ing. Jan Bauer, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

  
prof. Ing. Jiří Lettl, CSc.  
vedoucí katedry



  
prof. Ing. Pavel Ripka, CSc.  
děkan

V Praze dne 1. 10. 2014

## **Prohlášení**

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

.....

Kuanysh Stikeyev

## **Poděkování**

Děkuji vedoucímu této bakalářské práce panu Ing. Janu Bauerovi, Ph.D., za odborné vedení práce, za cenné rady a připomínky. Dále bych chtěl poděkovat své rodině za plnou podporu při studiu.

## **Anotace**

V bakalařské práci je popsáno rozdělení a principy otáčkových čidel. Dále je zde probírána problematika měření periody a frekvence, jejich vzájemné porovnání. Je zde také popsána komunikace u mikropočítačů. Podrobněji vysvětlena sběrnice SPI, schemy zapojení a princip komunikace. Práce se dále zabývá návrhem a implementací program pro vyhodnocení otáček a směru otáčení. Program je napsan v jazyce VHDL a otestovan v simulátoru.

## **Abstract**

In this bachelor thesis are described divisions and principles of rotary sensors. There is also an explanation of frequency measuring, period measuring, and their compariso. Next part of this thesis deals with microprocessor's comunication. SPI bus, circuit of connections and pricile of comunication are explained in detail. Finally, the thesis also covers a design and implementation of program for evaluating the rotary speed and direction. This program was written in VHDL language and the functionality was tested in a simulation.

# Obsah

|  |           |
|--|-----------|
| <b>1. ÚVOD</b> .....                             | <b>7</b>  |
| <b>2. SNÍMAČE OTÁČEK</b> .....                   | <b>8</b>  |
| 2.1. ROZDĚLENÍ SNÍMAČŮ OTÁČEK.....               | 8         |
| 2.2. ANALOGOVÉ SNÍMAČE OTÁČEK.....               | 9         |
| 2.3. IMPULZNÍ SNÍMAČE OTÁČEK.....                | 9         |
| 2.4. INKREMENTÁLNÍ ENKODÉRY.....                 | 10        |
| 2.5. ABSOLUTNÍ ENKODÉRY.....                     | 10        |
| 2.6. INDUČNÍ SNÍMAČE.....                        | 11        |
| 2.7. KAPACITNÍ SNÍMAČE.....                      | 12        |
| 2.8. MAGNETICKÉ SNÍMAČE.....                     | 13        |
| 2.9. OPTICKÉ SNÍMAČE.....                        | 14        |
| <b>3. METODY VYHODNOCOVANÍ</b> .....             | <b>15</b> |
| 3.1. METODY VYHODNOCOVANÍ RYCHLOSTI OTÁČENÍ..... | 15        |
| 3.1.1. Měření periody.....                       | 15        |
| 3.1.2. Měření frekvence.....                     | 16        |
| 3.2. METODA URČOVÁNÍ SMĚRU OTÁČENÍ.....          | 16        |
| <b>4. KOMUNIKAČNÍ ROZHRANÍ</b> .....             | <b>18</b> |
| 4.1. SYNCHRONNÍ A ASYNCHRONNÍ ROZHRANÍ.....      | 18        |
| 4.2. VNITŘNÍ ZAPOJENÍ.....                       | 20        |
| 4.3. SCHÉMA ZAPOJENÍ.....                        | 20        |
| 4.4. PRINCIP KOMUNIKACE.....                     | 22        |
| 4.5. REŽIMY SPI.....                             | 22        |
| <b>5. NÁVRH A IMPLEMENTACE</b> .....             | <b>24</b> |
| 5.1. FILTRACE.....                               | 24        |
| 5.2. SČÍTÁNÍ HRAN SIGNÁLU.....                   | 24        |
| 5.3. RYCHLOST OTÁČENÍ.....                       | 26        |
| 5.4. SMĚR OTÁČENÍ.....                           | 29        |
| <b>6. SIMULACE</b> .....                         | <b>30</b> |
| <b>7. ZÁVĚR</b> .....                            | <b>33</b> |
| <b>8. SEZNAM OBRÁZKŮ</b> .....                   | <b>34</b> |
| <b>9. SEZNAM POUŽITÉ LITERATURY</b> .....        | <b>35</b> |
| <b>PŘÍLOHA</b> .....                             | <b>36</b> |

# 1. Úvod

Cíl bakalářské práce je navrhnout program v jazyce VHDL pro vyhodnocení snímačů otáček. Program by měl umožnit vyhodnocení otáček z libovolných inkrementálních snímačů, které mají dva základní výstupní signály posunuté o 90° elektrických a jeden nulový impulz pro určení absolutní polohy hřídele. Další požadavek na výstup programu je informace o směru otáčení hřídele.

V kapitole č.2 je proveden podrobný rozbor otáčkových čidel, rozdělení, princip činnosti a jejich výstupní signály. V kapitole č.3 jsou probrány metody vyhodnocování rychlosti a směru otáčení, a také problematika měření periody a frekvence a jejich vzájemné porovnání. V kapitole č.4 je vysvětlena možnost komunikace s mikropočítačem. A detailně rozebrána sběrnice SPI, schémata zapojení, princip komunikace a režimy práce.

V následující kapitole č. 5 je popis návrhu programu pro vyhodnocení čidel otáček a jeho implementace.

Kapitola č. 6 se zaměřuje na testování programu pro vyhodnocení inkrementálního rotáčního snímače.

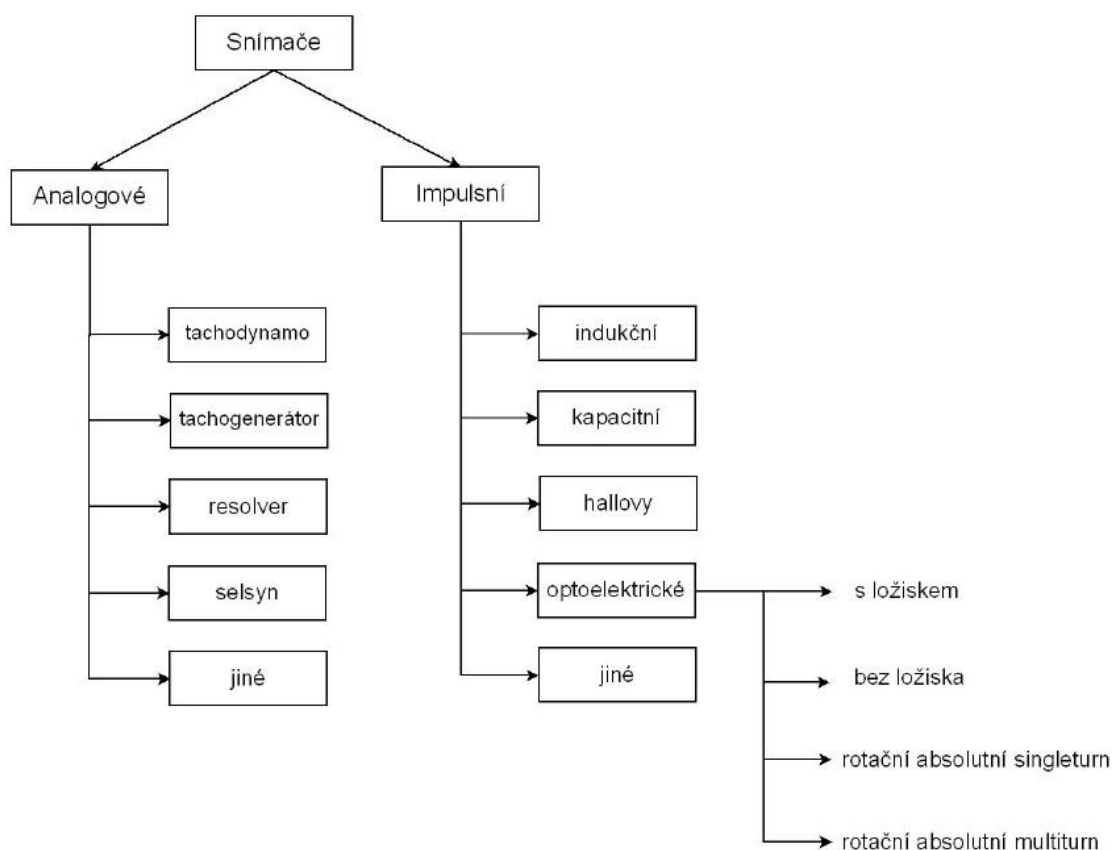
Závěrečná kapitola č. 7 shrnuje výsledky práce.

## 2. Snímače otáček

Informace o snímačích otáček jsem čerpal z [2], [5] a [8].

### 2.1. Rozdělení snímačů otáček

Snímače otáček mohou pracovat na základě různých fyzikálních principů. Podle formy výstupu je lze dělit do dvou skupin, na analogové a impulsní. Analogové snímače mají výstupní napětí úměrné rychlosti otáčení. Výstupem impulsních snímačů je obdélníkový signál, který původně byl sinusové podoby. Je však zesílen a předzpracován uvnitř snímacího zařízení. Na Obr. 2.1 můžeme vidět jedno z možných rozdělení snímačů otáček.[2]

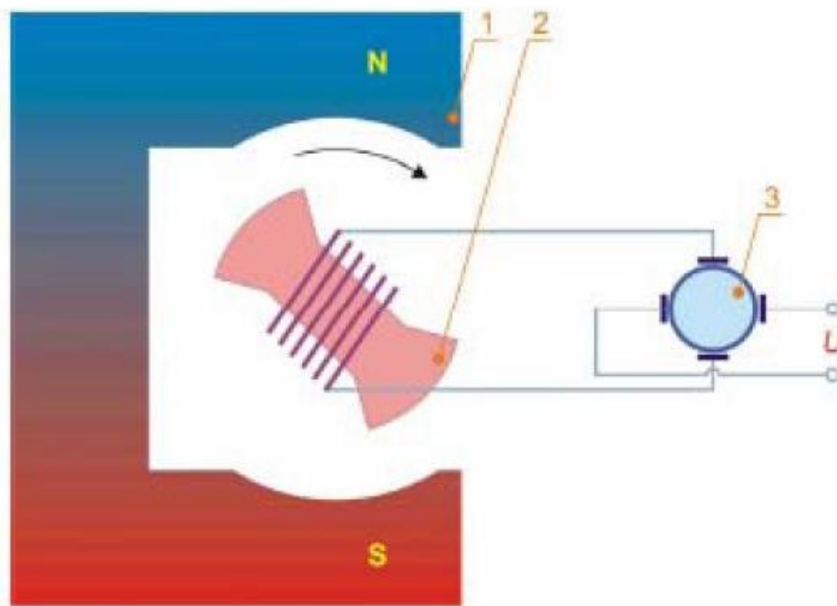


Obr. 2.1 Rozdělení snímačů otáček.



## 2.2. Analogové snímače otáček

Jak už bylo zmíněno dříve princip práce analogových snímačů je založen na závislosti výstupního napětí na otáčkách. Příkladem takových snímačů jsou tachodynamo, tachogenerátor, resolver a jiné. V rámci této práce podrobněji probrány nebudou. Protože se chceme zabývat digitálním zpracováním digitálního výstupu a u analogových snímačů to není možné, bez dalších obvodů pro zpracování signálu.



Obr. 2.2 Princip zapojení tachodynamu.  
1 - permanentní magnet, 2 – kotva, 3 – komutátor [2].

## 2.3. Impulzní snímače otáček

Další název pro daný druh otáčkových čidel je enkodéry. Enkodér je speciální zařízení, pomocí kterého se zjistí poloha rotujících hřídelů. Jejich základní funkce je měření lineárních a úhlových přemístění. V daném zařízení dochází k transformaci mechanického pohybu v elektrické signály, poskytující informace o úhlu natočení hřídele, jeho poloze a směru otáčení. Bez enkodérů jsou nepředstavitelné moderní systémy přesného pohybu. V každém zařízení, kde dochází k točení, je potřeba mít pro kontrolu přesnosti toho otáčení zpětnou vazbu mezi točivými a netočivými částmi tohoto systému. Například právě pomocí enkodéru můžeme sledovat a kompenzovat přeskočené kroky při prokluzu krokového motoru. Enkodéry lze rozdělit na základě dvou zásadně odlišných principů:

- Inkrementální
- Absolutní

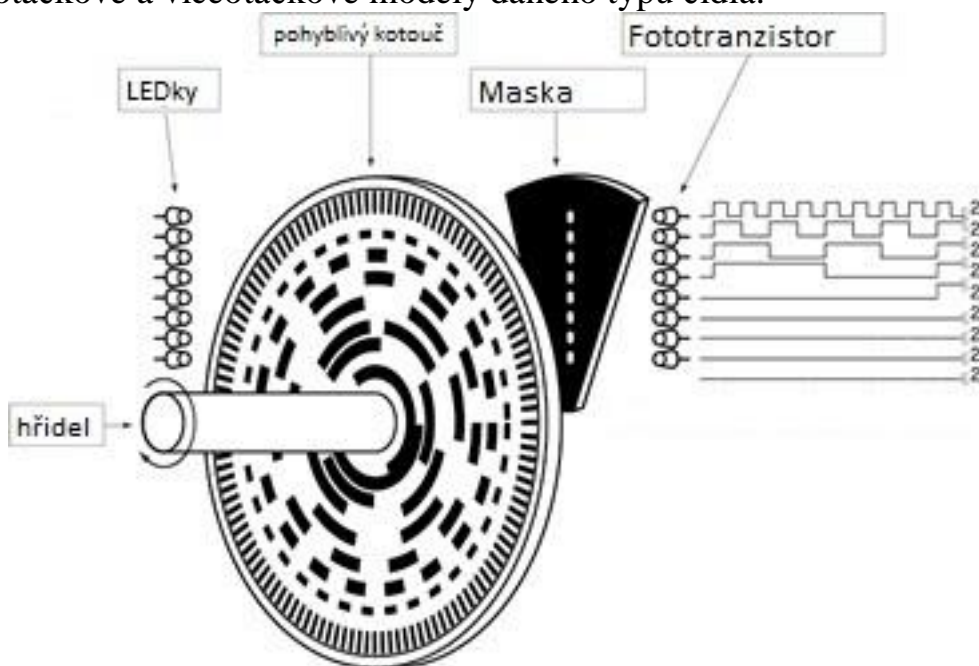
## 2.4. Inkrementální enkodéry.

Inkrementální enkodér generuje data o poloze objektu prostřednictvím elektrických impulzů. Počet impulzů na jednu otáčku je hlavní parametr, charakterizující inkrementální enkodéry. Impulzy vznikají, jen když se hřídel otáčí. Jakmile se hřídel zastaví vysílání impulzů přestává a pomocí toho typu enkodéru se polohu hřídel už nedozvíme. Což znamená, že tento typ čidla je vhodné použít na měření rychlosti otáčení, a kvůli tomu že si nepamatuje minulou polohu tak před novým spuštěním musíme znova nastavit počáteční polohu.

Inkrementální enkodéry jsou zdrojem impulzů, tyto impulzy vznikají na základě různých jevů. Na obvodu hřídele, která se otáčí úhlovou rychlostí  $\omega$  je značka (výstupek), může jí být například malý magnet, kovová destička atd., a ta je snímána detektorem, který je citlivý na její přiblížení.

## 2.5. Absolutní enkodéry.

Absolutní otáčková čidla poskytují data o počtu otáček, úhlu, a poloze hřídele pomocí unikátních kódů, odpovídajících každému kroku. Počet unikátních kódů na jednu otáčku a počet otáček jsou základními charakteristikami daného typu zařízení. Počáteční nastavení čidla není potřeba, protože absolutní poloha se určí unikátním kódem. Existují jednootáčkové a víceotáčkové modely daného typu čidla.



Obr. 2.3 Zjednodušená struktura absolutního enkodéru [4]

## 2.6. Induční snímače.

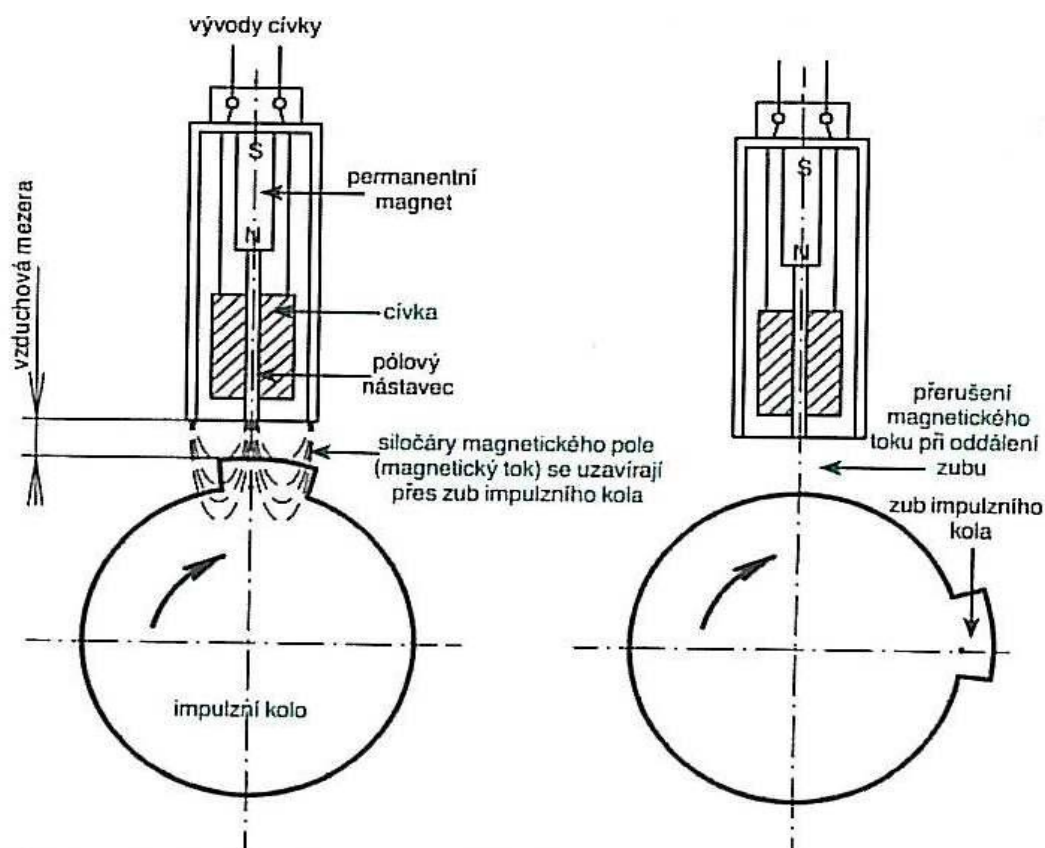
Pracují na fyzikálním principu změny magnetického toku procházejícího přes cívku, čímž se projeví změnou indukovaného napětí na cívkce dle vztahu:

$$u_i = N * \frac{\Delta\Phi}{\Delta t}$$

kde:

$u_i$  je indukované napětí  
 $N$  je počet závitů cívkky  
 $\frac{\Delta\Phi}{\Delta t}$  je změna magnetického toku v čase

Změna magnetického toku je vyvolaná pohybem zubu impulzního kola, když zub, udělaný z feromagnetického materiálu, prochází v blízkosti snímače zmenšuje magnetický odpor a tím zvětšuje magnetický tok procházející cívkou. Konstrukční provedení takového snímače je vidět na Obr. 2.4.



Při obíhání impulzního kola okolo snímače dochází střídavě k uzavírání magnetického toku přes zub kola (obrázek vlevo) a přerušování magnetického toku při oddálení zubu od snímače (obrázek vpravo). Změna magnetického toku způsobuje indukování napětí v cívkce.

Obr. 2.4 Konstrukční provedení indukčního snímače.[5]

## 2.7. Kapacitní snímače.

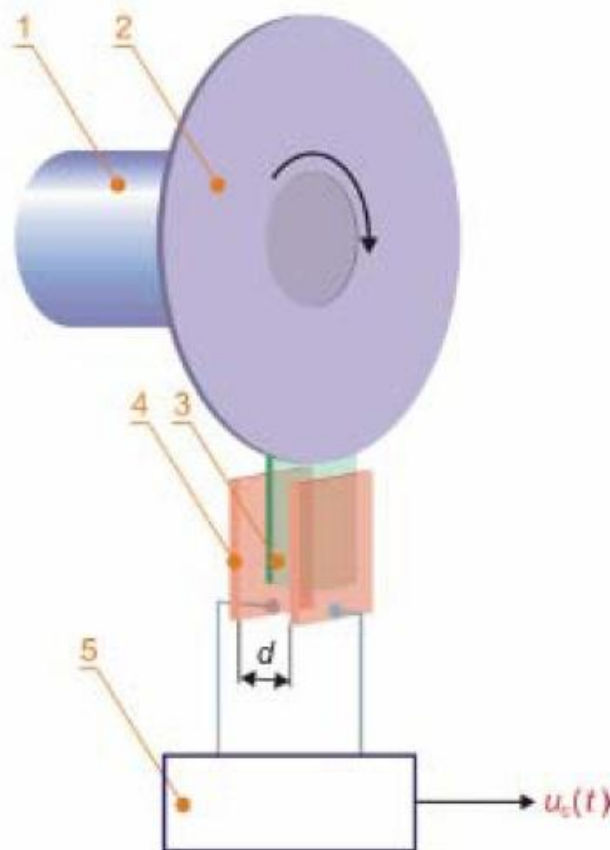
Kapacitní snímače využívají principu změny kapacity deskového kondenzátoru. Kapacita deskového kondenzátoru je dána vztahem:

$$C = \frac{\varepsilon * S}{d}$$

kde:

- $\varepsilon$  je permitivita dielektrika,
- $S$  je obsah desky kondenzátoru,
- $d$  je vzdálenost desek kondenzátoru.

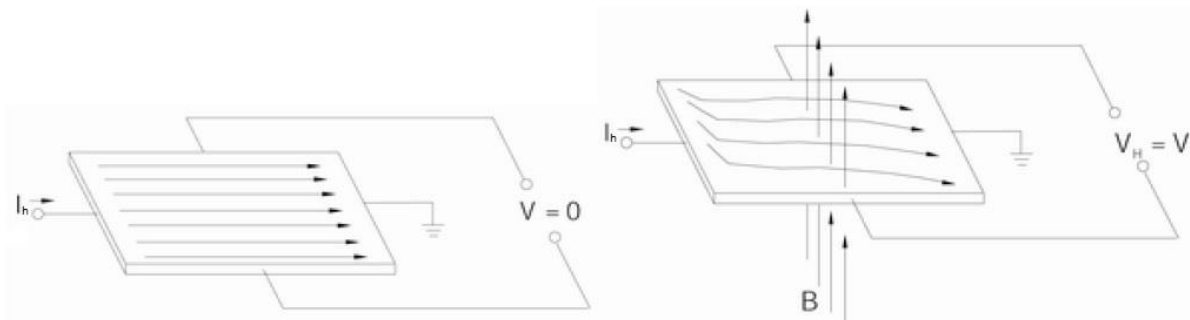
Na hřídeli, stejně jak u indukčního snímače, je umístěno ozubené kolo a deskový kondenzátor je umístěn tak, aby zub hřídele procházel mezi deskami kondenzátoru, tímto dějem se permitivita prostředí mezi deskami zvětšuje a tím roste i kapacita kondenzátoru. To je zaznamenáno měřičem kapacity jako pulz a z počtů pulzů, lze stanovit rychlost otáčení hřídele.



Obr. 2.5 Princip činnosti kapacitního snímače.  
1-hřídel,2-kotouč,3-kovová destička,4-desky kondenzátoru,5-měřič kapacity[2]

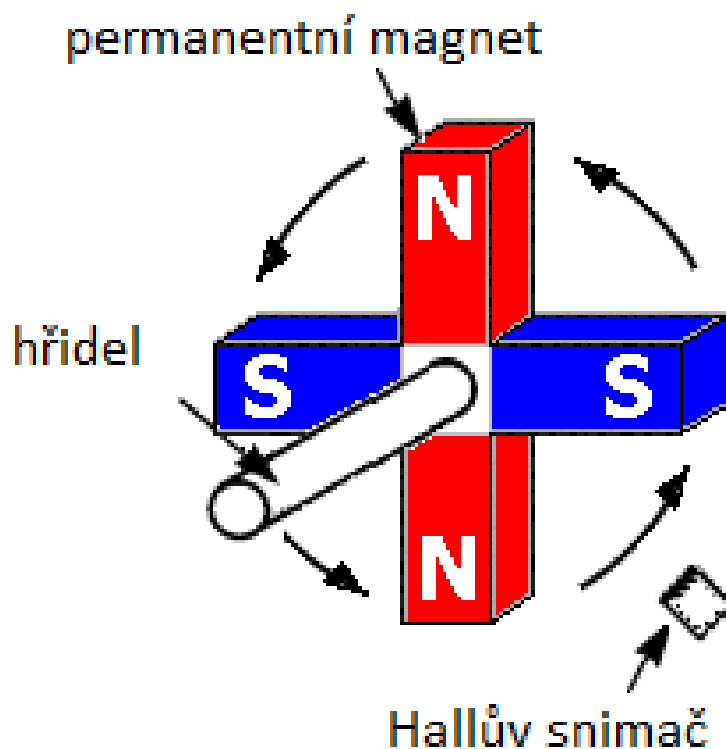
## 2.8. Magnetické snímače

Tyto snímače využívají Hallova jevu. Hallův snímač se skládá z polovodičové destičky napájené proudem  $I_h$ . Pokud je permanentní magnet v takové poloze, že jeho magnetické pole  $B$  působí kolmo na protékající proud destičkou, je generováno Hallovo napětí  $U_h$ . Pro vyhodnocování výsledku je generované napětí příliš malé a musí se ještě zesílit.



Obr. 2.6 Princip činnosti Hallova snímače.[6]

Jeden z možných konstrukčních provedení je zobrazen na Obr. 2.7. Enkodér se skládá z hřídele s permanentní magnety a Hallova snímače, který registruje posloupnost procházení magnetických pólů, s čehož určuje rychlost a směr otáčení.

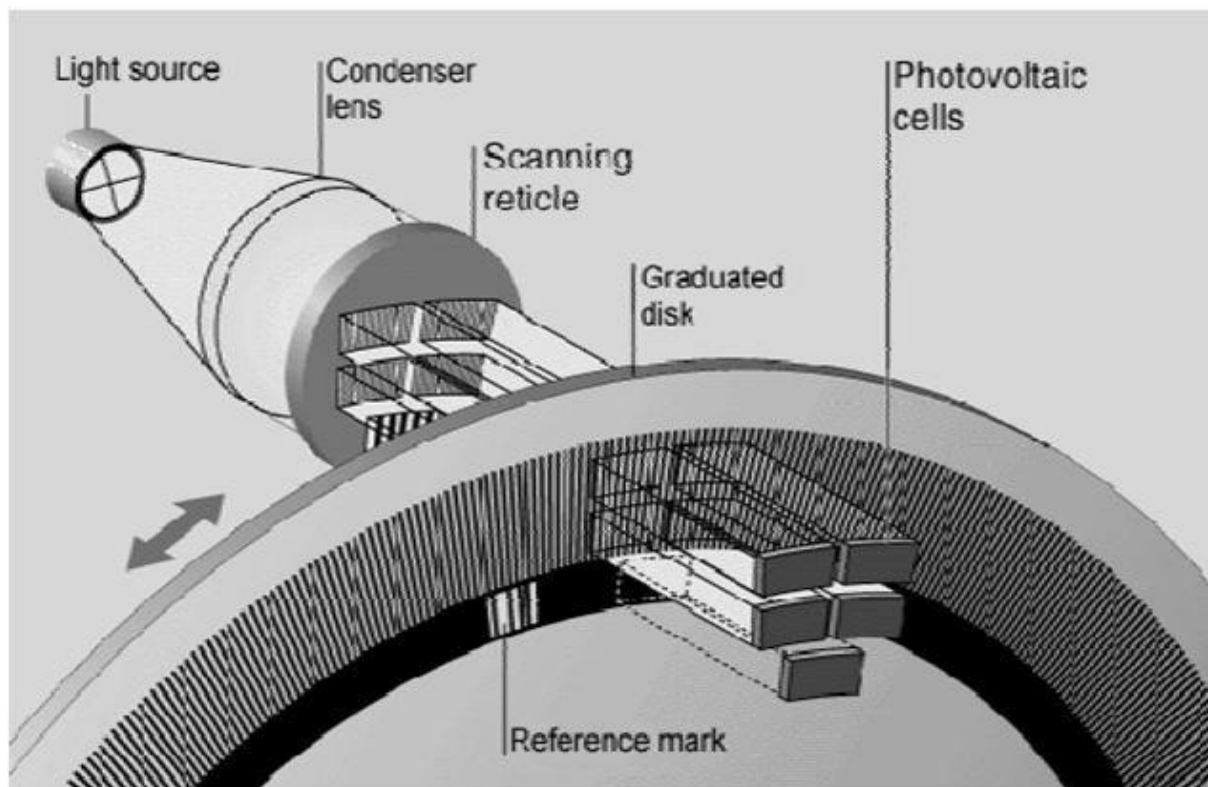


Obr. 2.7 Konstrukční provedení magnetického snímače.

## 2.9. Optické snímače

Nejvíce používaný typ rotačních inkrementálních snímačů otáček jsou optické snímače.

Princip senzoru spočívá ve clonění světelného toku mezi zdrojem světla a fotodetektorem. Základem je pulsní disk, tento disk je mechanicky spojen s hřídelem. Disk obsahuje světlá a tmavá pole. Světlo generované kvalitními diodami, pracujícími většinou v infračervené oblasti spektra, prochází přes membránu a tento pulsní disk a následně je zachyceno fotodetektorem umístěným na druhé straně. Při otáčení hřídele pak disk střídavě světlo propouští a nepropouští (zacloňuje fotodetektor). Princip činnosti je zobrazen na Obr. 2.8. Počet tmavých (nepřůhledných) a světlých (průhledných) polí odpovídá počtu pulsů na jednu otáčku, což je jeden z nejdůležitějších parametrů udávaných u každého inkrementálního snímače. Standardem jsou snímače 3-kanálové. Signály dvou kanálů jsou vzájemně posunuty o  $90^\circ$ , což umožňuje rozpoznat směr otáčení. Třetí kanál generuje puls jednou za otáčku a zpravidla se nazývá „nulovým pulsem“. Z principu činnosti si inkrementální snímače na rozdíl od snímačů absolutních „nepamatují“ polohu při vypnutí napájení.

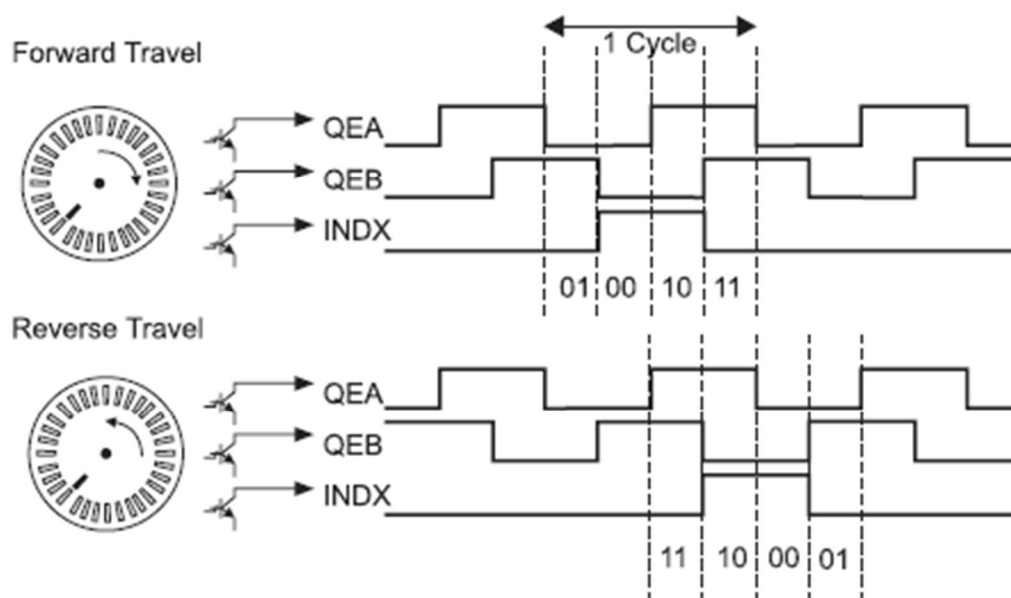


Obr. 2.8 Princip činnosti optického snímače.[8]

Existuje ještě několik principů jak se dá zjišťovat rychlost otáčení atd, ale ty nejsou moc rozšířené nebo nedosahují takový přesností.

### 3. Metody vyhodnocování

Jak už bylo zmíněno dříve výstupem z inkrementálního čidla jsou dva obdélníkové průběhy vzájemně posunuté o 90 stupňů. Obvykle je ještě jeden signál – nulový, ten se objeví jednou za otáčku.



Obr. 3.1 Vstupní signály inkrementálního čidla. [7]

Dále je potřeba signály dekodovat, aby jsme získali údaje, které jsou líp pochopitelné, jako rychlost otáčení a směr otáčení.

#### 3.1. Metody vyhodnocování rychlosti otáčení

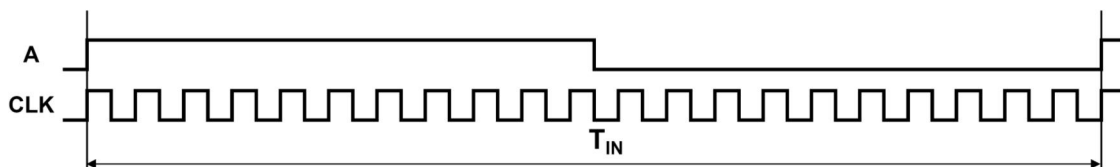
Rychlost otáčení hřídele zjistíme měřením periody nebo frekvence pulzů. Informace do této kapitoly jsem získal z [3].

##### 3.1.1. Měření periody

Perioda  $T_{IN}$  vstupního signálu je rovna obrácené hodnotě frekvence signálu  $f_{IN}$  :

—

Měření periody neznámého signálu se provede čítáním hodinových pulzů po dobu periody měřeného signálu. Měření můžeme provést pro několik period za sebou, výsledná perioda se potom určí jako průměrná hodnota naměřených period. Měření periody umožňuje přesnější měření signálů o nízké frekvenci než měření frekvence. Měření periody je znázorněno na Obr. 3.2.



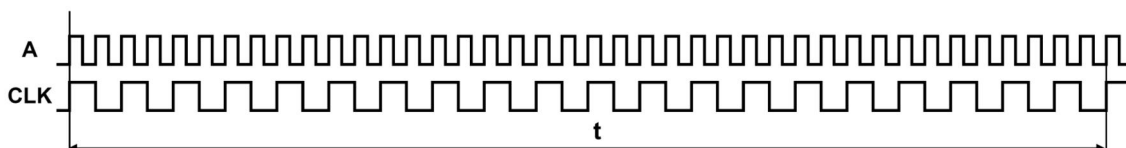
Obr. 3.2 Měření periody.

### 3.1.2. Měření frekvence

Frekvence signálu může být definovaná počtem pulsů měřeného signálu za daný časový interval:

—

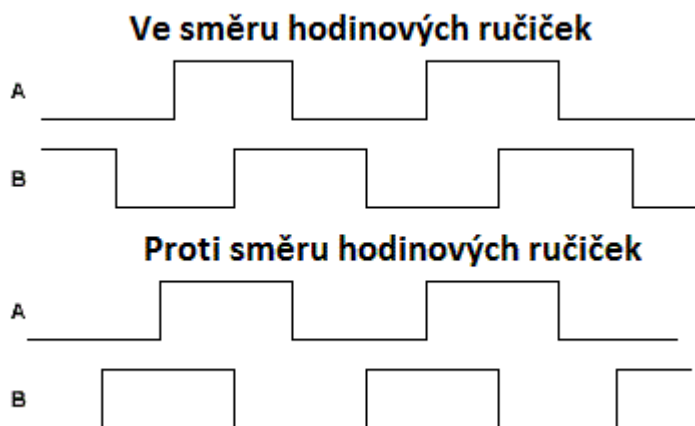
kde  $n$  je počet pulsů a  $t$  je časový interval v kterém jsme pulsy počítali. Měření frekvence je přesnější pro měření signálů o vysoké frekvenci než měření periody. Měření frekvence je znázorněno na Obr. 3.3



Obr. 3.3 Měření frekvence.

### 3.2. Metoda určování směru otáčení

Hlavní princip spočívá v tom, že zaznamenáváme změnu signálu na jedné stopě a ve stejný okamžik odečítáme hodnotu druhé stopy a na základě toho určujeme směr.

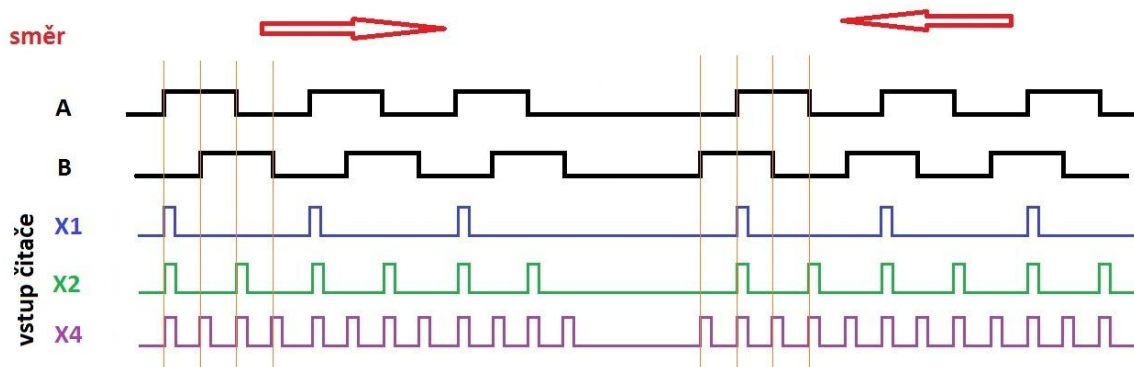


Obr. 3.4 Rozdíl vystupních signálů při různých směrech otáčení.



Data můžeme odečítat s různou citlivostí. Celkem můžeme použít tři různé varianty.

- s jednonásobnou přesností (X1) – detekujeme náběžnou nebo sestupnou hranu pouze jednoho kvadrurního signálu
- s dvojnásobnou přesností (X2) – detekujeme náběžnou a sestupnou hranu pouze jednoho kvadrurního signálu
- s čtyřnásobnou přesností (X4) – detekujeme náběžnou a sestupnou hranu obou kvadrurních signálů.



Obr. 3.5 Jednotlivé způsoby dekódování kvadrurních signálů

## 4. Komunikační rozhraní

Informace do této kapitoly jsem získal z [9], [10], [11], [12] a [13].

Další důležitou částí je také komunikace mezi řídicí jednotkou a ostatními částmi soustavy. U mikropočítačů existují dva druhy komunikace. Sériové a paralelní. Sériová komunikace je proces, při kterém se data posílají po jednotlivých bitech, za sebou. Paralelní komunikace je proces, při kterém se najednou posílá několik bitů (např. 8). Dále můžeme rozdělit komunikace podle synchronnosti. Synchronní komunikace je, když přijímač a vysílač pracují podle jednoho signálu, například hodinového signálu. Asynchronní komunikace probíhá bez synchronizačního signálu, ale je řízena pomocí speciálních symbolů (start bit, stop bit) a potřebné časování je vyhodnoceno každým zařízením pomocí vlastních hodin. Základní rozdíl mezi synchronní a asynchronní komunikací je v tom, že při synchronní komunikaci se kromě dat přenášejí po zvláštní lince ještě synchronizační signál, tak zvané „společné hodiny“.

Komunikaci po sériové lince využívá většina mikropočítačů. Umožňuje velmi jednoduchý přenos dat mezi mikropočítačem a jiným zařízením vybaveným sériovým rozhraním, např. počítačem PC. A proto se jí budu zabývat podrobněji.

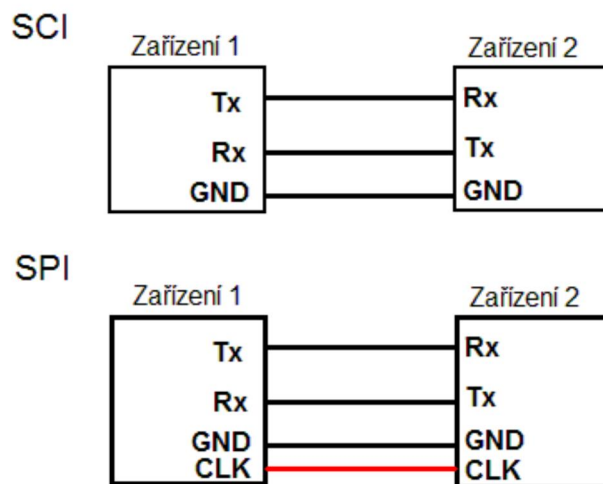
### 4.1. Synchronní a asynchronní rozhraní

Existují dva základní druhy sériového rozhraní a to synchronní (SPI) a asynchronní (SCI).

- SPI – zkratka Serial Peripheral Interface, tj. sériové periferní rozhraní – používá se pro rychlou komunikaci na kratší vzdálenosti, typicky v rámci plošného spoje. Příkladem použití je komunikace mezi procesorem a externími paměťovými moduly nebo A/D převodníkem.

- SCI – zkratka Serial Communication Interface, tj. sériové komunikační rozhraní – používá se pro přenos dat na větší vzdálenosti. Příkladem použití je připojení mikropočítače k vývojovému PC.

Podle počtu vodičů potřebujeme při asynchronní komunikaci pro obousměrný provoz minimálně 3 vodiče, a pro synchronní budeme potřebovat jeden vodič navíc pro hodinový signál.



*Obr. 4.1 Srovnání asynchronní (SCI) a synchronní (SPI) komunikace podle počtu vodičů*

V rámci této práce jsem uvažoval o synchronní sériové komunikaci (SPI) a proto ji probereme podrobněji.

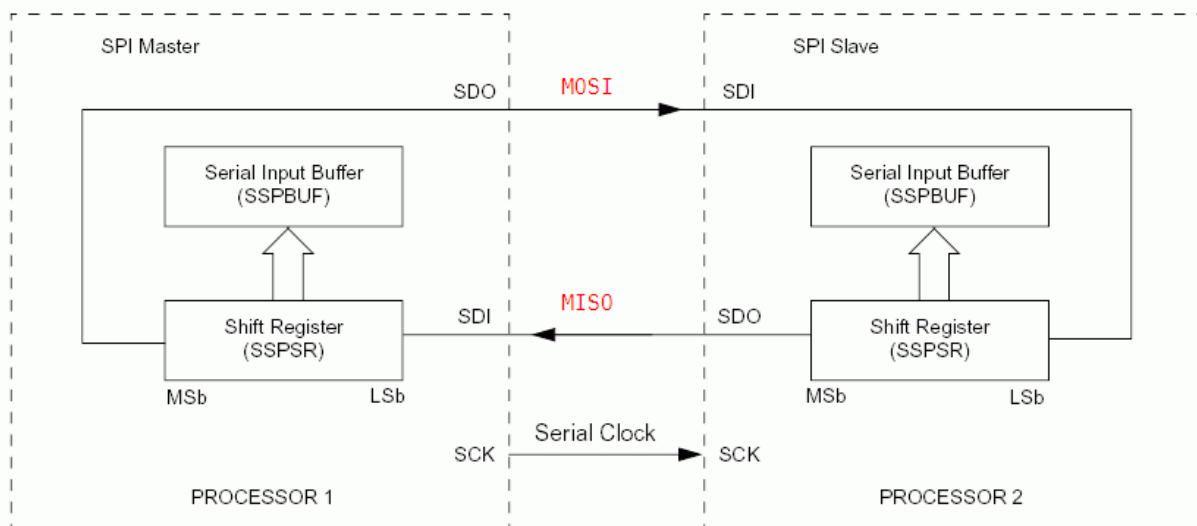
SPI sběrnice byla navržena a uvedena na trh firmou Motorola, ale v současné době jí používají všichni výrobci.

Používá se pro komunikaci mezi řídicími mikropočítači a ostatními integrovanými obvody (A/D převodníky, displeje ...). Komunikace je realizována pomocí společné sběrnice.

Zařízení na SPI sběrnici jsou řídicí (Master) a podřízené (Slave). Master řídí komunikaci podle hodinového signálu a určuje, s jakým zařízením bude komunikovat. Slave vysílá data podle hodinového signálu, pokud je aktivován.

## 4.2. Vnitřní zapojení

Obě zařízení v nejjednodušším případě mají jeden datový záchytný registr (*Serial Input Buffer*) a posuvný registr (*Shift Register*). Datové registry zajišťují, aby nedocházelo ke ztrátě dat.

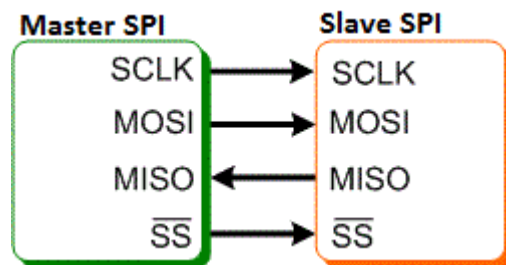


Obr. 4.2 Vnitřní zapojení Master - Slave

U SPI komunikace je vysílání a příjem dat, jak je to vidět z obrázku, nedělitelný proces, a oba dva procesy probíhají současně. Typickou pracovní frekvencí je v rozmezí 1-50 MHz. Díky jednoduchosti algoritmu přenášení dat je SPI rozšířené v nejrůznějších elektronických zařízeních.

## 4.3. Schéma zapojení

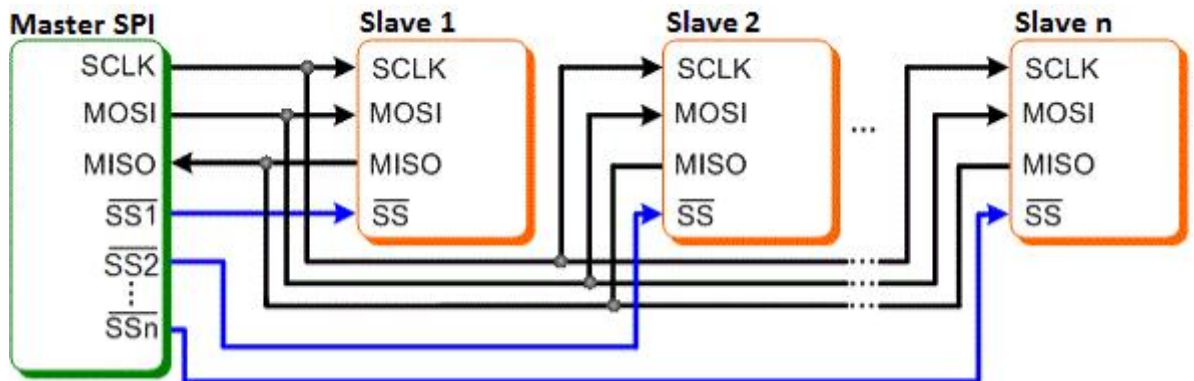
Nejjednodušší variantou SPI komunikace je jeden řídicí a jeden podřízený.



Obr. 4.3 Jednoduché zapojení Master - Slave  
SCLK – „Serial Clock“, hodinový signál  
MOSI – „Master Out, Slave In“, výstup z Masteru  
MISO – „Master In, Slave Out“, výstup ze Slave  
SS – „Slave Select“, volba podřízeného

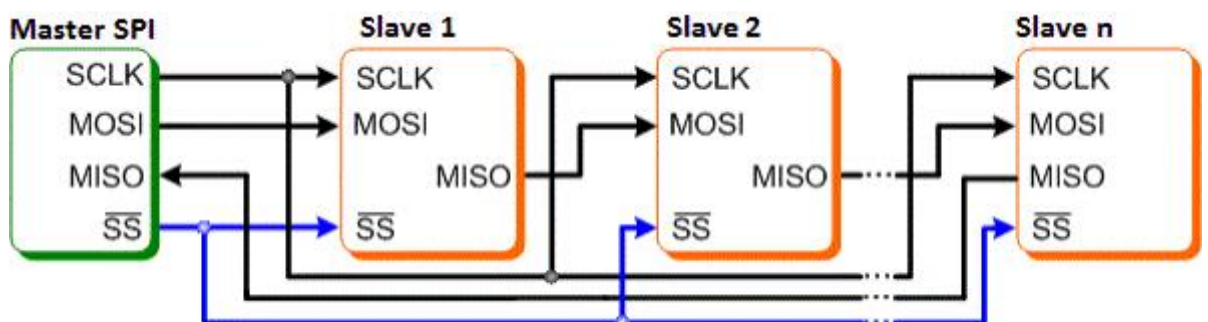
Pro případ, kde potřebujeme připojit k masteru víc než jednoho podřízeného, využívá se jedna ze dvou variant, nezávislé (paralelní) nebo kaskádní (seriové) řazení podřízených.

Častěji se využívá nezávislá varianta. Tj. všechny signály, kromě volby podřízeného, jsou paralelní a uvedením nějakého signálu SS do log 0, master zvolí podřízeného, se kterým bude komunikovat. Hlavní nevýhodou tohoto zapojení je, že s rostoucím počtem podřízených, vzrůstá počet SS vodičů. Celkový počet vodičů pro komunikaci se rovná  $3+n$ , kde  $n$  je počet podřízených.



Obr. 4.4 Nezávislé zapojení pro více podřízených.

Kaskádní zapojení eliminuje tento problém, protože zde ze všech zařízení vzniká jeden velký posuvný registr. Toho lze dosáhnout tím, že výstup z podřízeného se připojí na vstup dalšího podřízeného, a tím se spojí všechna zařízení do kola. Nevýhodou je, že ne každý mikro počítač je tak schopný pracovat. Tato schopnost se da zjistit z dokumentace, anglický název parametru 'daisy-chaining'.

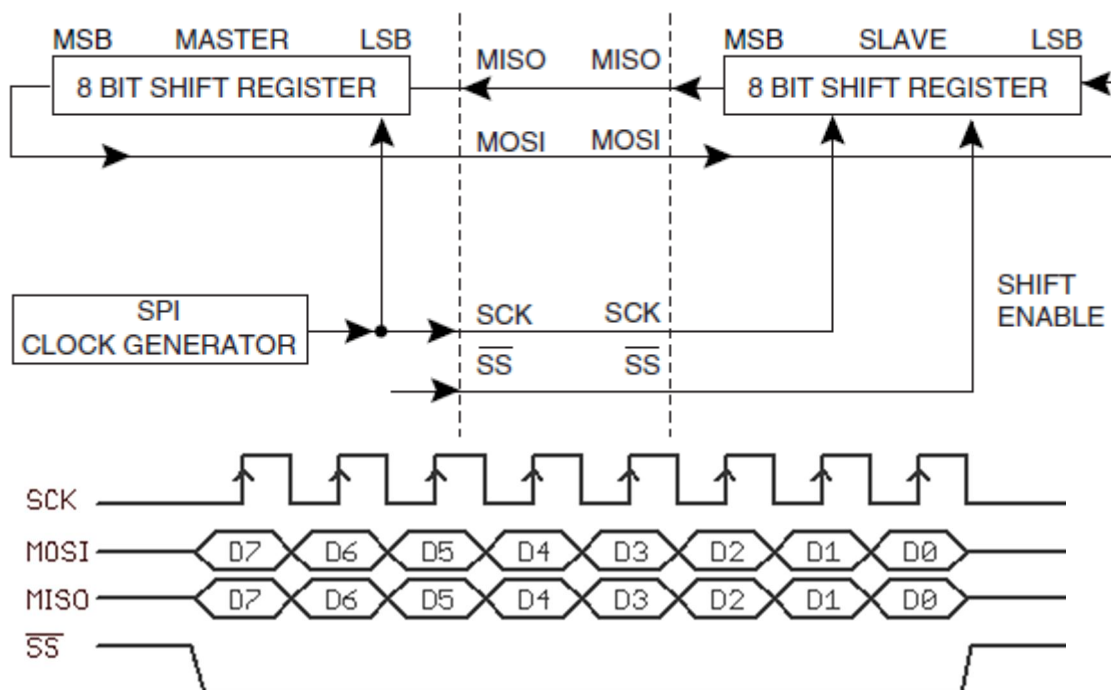


Obr. 4.5 Kaskádní zapojení pro více podřízených.

## 4.4. Princip komunikace

Komunikace probíhá v několika krocích.

1. Master nastaví log.0 na SS zvoleného podřízeného, s kterým chce komunikovat.
2. S hodinovým signálem se začnou přenášet data.
3. Délka vyslaných dat je 8 bitů nebo 16 bitů.
4. Po vyslání všech dat, Master nastaví na SS log 1 a tím je komunikace ukončena.



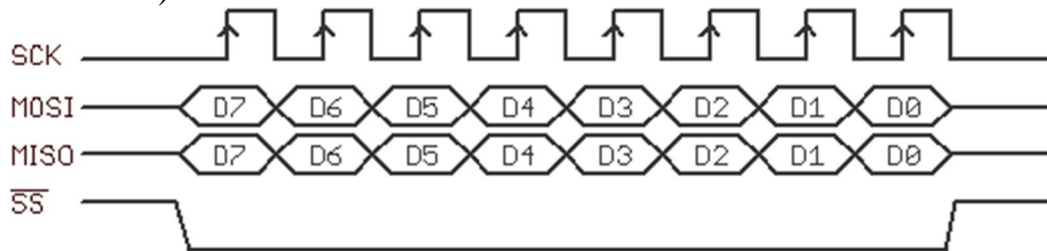
Obr. 4.6 Princip komunikace.

## 4.5. Režimy SPI.

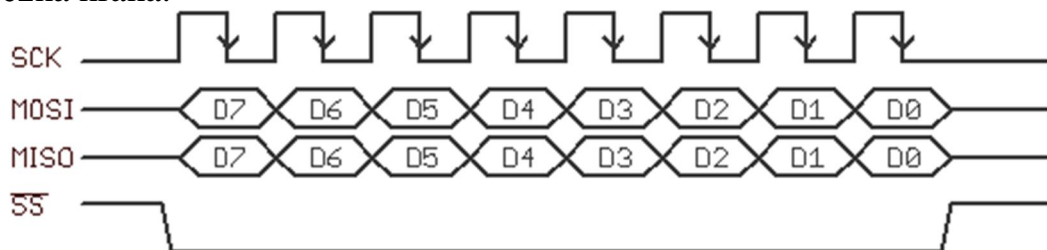
Specifikace SPI zajišťuje 4 režimy, které se liší vztahem mezi polaritou hodinového signálu (clock polarity, CPOL) a fází synchronizace hodinového signálu (clock phase, CPHA). Tyto konfigurační bity jsou definovány podle konvencí od firmy Freescale Semiconductor (původně Motorola).

Jinak řečeno, CPOL – hladina hodinového signálu před začátkem a po vyslání dat. A fáze, CPHA určuje na jakou hranu proběhne čtení a zápis.

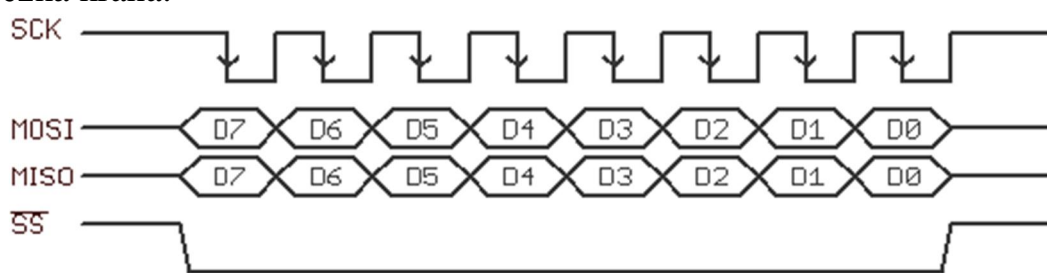
**SPI mode 0: CPOL = 0, CPHA=0.** Ke čtení bitu dochází na náběžnou hranu hodinového signálu (přechod 0 - 1), a k zápisu - na sestupnou hranu (přechod 1 - 0).



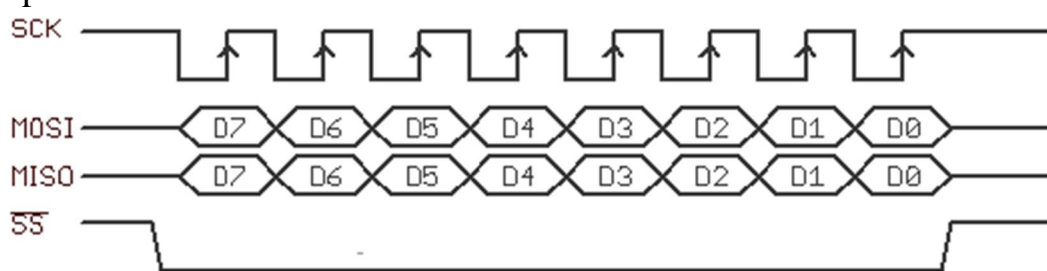
**SPI mode 1: CPOL = 0, CPHA=1.** Čtení – sestupná hranu, zápis - náběžná hranu.



**SPI mode 2: CPOL = 1, CPHA=0.** Čtení – sestupná hranu, zápis - náběžná hranu.



**SPI mode 3: CPOL = 1, CPHA=1.** Čtení – náběžná hranu, zápis - sestupná hranu.



Obr. 4.7 Řežimy SPI.

Řídicí a podřízené zařízení musí pracovat se stejném režimu, jinak jsou vzájemně neslučitelné. V současné době většina mikro počítačů má možnost volby libovolného režimu komunikace. Dále může přenos dat přes SPI probíhat buď vyšším bitem jako prvním, nebo od nejnižšího k nejvyššímu bitu. Počet bajtů přenesených během komunikace s jedním slavem není omezen a závisí jen na specifikaci podřízeného.

## 5. Návrh a implementace

### 5.1. Filtrace

Nejprve byla provedena filtrace vstupních signálů, aby nedocházelo k rušení a falešným detekcím. Filtrace byla provedena následujícím blokem kódu.

```
process(clc)
  begin
    if clc'event and clc = '1' then
      if a_mem = '0' then
        if a = '0' then
          a_f <= a;
        end if;
      else
        if a = '1' then
          a_f <= a;
        end if;
      end if;
      a_mem <= a;
    end if;
  end process;
```

Každou náběžnou hranu hodinového signálu bude sledovaná minulá (*a\_mem*) a aktuální (*a*) hodnota vstupního signálu. Když budou signály stejné, filtrovaná hodnota (*a\_f*) bude aktualizovaná. Na konci každého cyklu se aktualizuje hodnota minulého signálu.

Ukázka je jen pro jeden vstupní signál (*a*), filtrace ostatních je řešena stejným způsobem.

### 5.2. Sčítání hran signálu.

Prvním pokusem ožívání programu bylo počítání hran vstupního signálu. Pro tyto účely byla realizovaná detekce náběžných (*s1*) a sestupných (*s2*) hran a jejich zpracování. Za úkol jsem taky měl udělat tři režimy sčítání hran: jen náběžné hrany, jen sestupné hrany a obě dvě hrany. Přepínání je realizováno pomocí multiplexeru, kde jsou na vstup přivedeny signály detekující náběžné a sestupné hrany, a přepínacím signálem stisknutí tlačítka (*t1*). V závislosti na zvoleném režimu bude na výstup propuštěn pulz nějaké hrany nebo obou dvou hran. Dále, když přijde pulz (*pulz*), tak se hodnota výsledku sčítání (*sum*) se zvětší o jedničku. Výsledek sčítání se vyšle, jen když přijde *N signal*, což znamená jednou za otáčku. Při změně režimu se hodnota „*sum*“ vynuluje. Realizace je v následujícím kusu kódu.



```

process(clc)
  begin
    if clc'event and clc = '1' then
      t1 <= (not tlacitko_pam) and tlacitko; --nabezna hrana tlacitka, detekce na
stisknuti
      -- kanal A
      s1 <= (not a_pam) and a_f; --nabezna hrana signalu vstupu
      s2 <= (not a_f) and a_pam; --sestupna hrana signalu
      -- kanal B
      s3 <= (not b_pam) and b_f; --nabezna hrana signalu vstupu
      s4 <= (not b_f) and b_pam; --sestupna hrana signalu
      --multiplexer
      case stav is
        when "00" => pulz <= s1;
        when "01" => pulz <= s2;
        when "10" => pulz <= (s1 or s2);
        when others => null;
      end case;
      --nulovani sum pro pripad zmeny stavu_tlacitka nebo kdy pride n-signal
      n1 <= (not n_pam) and n; --nabezna hrana signalu N
      if sumRes = '1' or n1 = '1' then
        if n1 = '1' then
          sum_SH <= std_logic_vector(to_unsigned(sum,32));
        end if;
        sum <= 0;
      elsif pulz = '1' then --kdyz pride pulz pricte 1
        sum <= sum + 1;
      end if;
      --aktualizace pameti
      tlacitko_pam <= tlacitko;
      a_pam <= a_f;
      b_pam <= b_f;
      n_pam <= n;
    end if;
  end process;

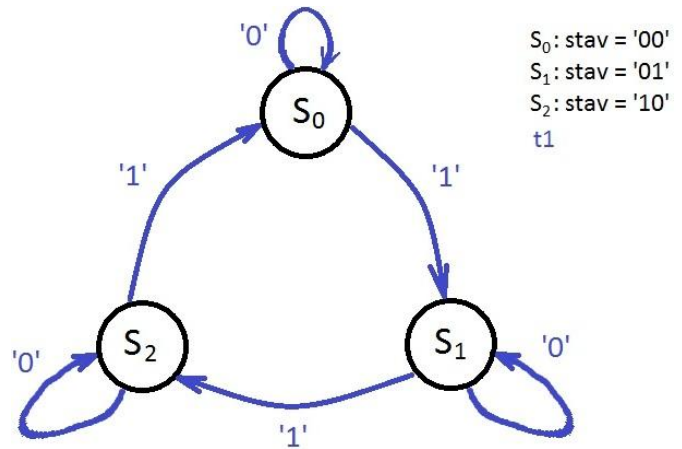
```

Logika tlačítka je realizovaná jako konečný automat typu Moore. Proměna *stav* stanoví aktuální režim sčítání. Ukázka realizace je v následující části programu.

```

process (t1) --pozoruje zmenu t1
begin
  if t1 = '1' then --kdyz byla
stisknuta
    if stav = "00" then
      stav <= "01";
      sumRes <= '1';
    elsif stav = "01" then
      stav <= "10";
      sumRes <= '1';
    elsif stav = "10" then
      stav <= "00";
      sumRes <= '1';
    end if;
  else sumRes <= '0';
  end if;
end process;

```



Obr. 5.1 Stavový diagram tlačítka.

### 5.3. Rychlost otáčení

V mém programu byla rychlost otáčení vyhodnocena jak metodou měření frekvence tak i metodou měření periody.

Na začátku je třeba nastavit pár konstant. Těmi jsou frekvence hodin (*clc\_frek*), počet pulsů na jednu otáčku čidla (*pno*), dobu po které bude proveden výpočet rychlosti (*time\_konst*), maximální hodnota timeru (*tmr\_max*) a pomocná konstanta (*ck*) - je poměru frekvence hodin a *time\_konst*.

```

constant clc_frek : integer := 1000000 ;      -- f = 1 MHz, T = 1us
constant pno : integer := 4;
constant time_konst : integer := 1000;
constant tmr_max : integer := 65536;
constant ck : integer := 1000;

```

Nejprve implementuji timer tak, že na každou náběžnou hranu hodinového signálu přičte ke své hodnotě jedničku, po dosažení maximální hodnoty se vynuluje a pokračuje pořád do kola. Paralelně k tomu mám sčítání hran signálu jedné stopy snímače.

```

process(clc)
begin
if clc'event and clc = '1' then
    if tmr1 = tmr_max then
        res_kolo <= '1';
        tmr1 <= 0;
        sum_a <= 0;
    else
        tmr1 <= tmr1 + 1;
        res_kolo <= '0';
    end if;
    if pulz = '1' then
        sum_a <= sum_a + 1;
    end if;
end if;
end process;

```

V dalším procesu probíhá samotné vyhodnocování rychlosti metodou měření frekvence v jednotkách otáčky/min. Princip je v tom, že máme dva registry do kterých ukládám hodnoty sčítání hran. Do registru *registr\_ppt1* ukládám aktuální hodnotu *sum\_a* a do registru *registr\_ppt2* to co jsem měl před tím v *registr\_ppt1*. Rozdíl těchto registrů dává počet pulsů za předem definovaný čas *time\_konst*. Dale jen stačí vynásobit převodem kolik pulsů „nateče“ během minuty a vydělit počtem pulsů na jednu otáčku čidla.

```

process (tmr1)
begin
    if res_kolo = '1' then
        kolo <= 1;
        registr_ppt2 <= 0;
        registr_ppt1 <= 0;
    end if;
    if tmr1 = time_konst * kolo then
        registr_ppt2 <= registr_ppt1;
        registr_ppt1 <= sum_a;
        kolo <= kolo + 1;
        ppt <= registr_ppt1 - registr_ppt2;
        if ppt > 0 then
            detkol <= '1'; --detekce zmeny kola
        end if;
    end if;
    if detkol = '1' then
        rozdil <= 60 * ck * ppt;
        tempRoz <= vysledek * pno;
        if(tempRoz < rozdil) then
            vysledek <= vysledek + 1;
        else
            detkol <= '0';
            ppt_SH <= std_logic_vector(to_unsigned(vysledek,32));
            vysledek <= 0;
            tempRoz <= 0;
        end if;
    end if;
end process;

```

```

        end if;
    end if;
end process;

```

Výstupní hodnota je *ppt\_SH*, ukazuje rychlost v otáčkách za minutu.

Další proces počítá jak dlouho trvá jedna perioda vstupního signálu, což znamená měření rychlosti otáčení metodou měření periody. Proces sleduje náběžnou hranu vstupního signálu a stejným způsobem zapisuje do dvou registrů minulou a aktuální hodnotu, ale na rozdíl od minulého případu, zapisuje hodnoty timeru. Následně mám čas na začátku a konci jedné periody vstupního signálu, rozdíl který mi dá periodu. Dalším zpracováním lze dosáhnout stejného měřítka jako s minulou metodou. Blok kódu popisující metodu následuje

```

process (s1)
begin
    if res_kolo = '1' then
        registr_kntr1 <= 0;
        registr_kntr2 <= 0;
    end if;
    if s1 = '1' then
        registr_kntr2 <= registr_kntr1;
        registr_kntr1 <= tmr1;
        kntr_cislo <= (registr_kntr1 - registr_kntr2) * pno;
        if kntr_cislo > 0 then
            detzmen <= '1';
        end if;
    end if;
    if detzmen = '1' then
        tempRoz2 <= vysl2 * kntr_cislo;
        if (tempRoz2 < 600000000) then -- 600000000 us = 1 min
            vysl2 <= vysl2 + 1;
        else
            detzmen <= '0';
            kntr_cislo_SH <= std_logic_vector(to_unsigned(vysl2,32));
            vysl2 <= 0;
            tempRoz2 <= 0;
        end if;
    end if;
end process;

```

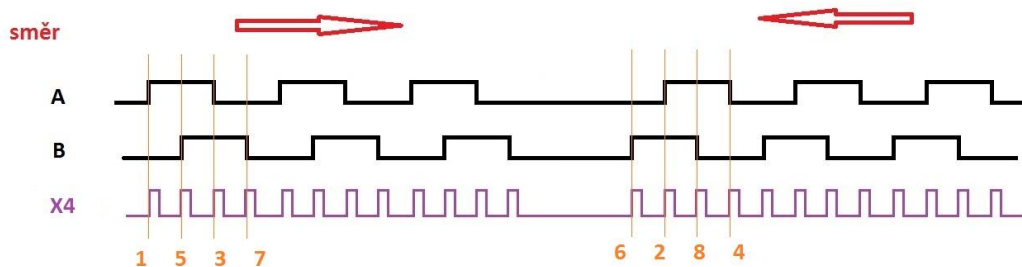
Výstupní hodnota je *kntr\_cislo\_SH*, ukazuje rychlost v otáčkách za minutu.

## 5.4. Směr otáčení

Směr otáčení se určuje ze signálu detekce hran vstupních signálů, stop A i B, které jsem nadefinoval úplně na začátku programu. Směr jsem určoval s čtyřnásobnou přesností, což je vidět z toho, že sleduji všechny hrany signálů.

```
process (s1,s2,s3,s4)
begin
  if s1 = '1' then --n_A
    if b_f = '1' then
      smer <= '1'; --2
    else
      smer <= '0'; --1
    end if;
  end if;
  if s2 = '1' then --s_A
    if b_f = '1' then
      smer <= '0'; --3
    else
      smer <= '1'; --4
    end if;
  end if;
  if s3 = '1' then --n_B
    if a_f = '1' then
      smer <= '0'; --5
    else
      smer <= '1'; --6
    end if;
  end if;
  if s4 = '1' then --s_B
    if a_f = '1' then
      smer <= '1'; --8
    else
      smer <= '0'; --7
    end if;
  end if;
end process;
```

Čísla napsaná v kódu do komentáře znamenají místa, ve kterých se určí směr. Podrobněji obrázek 3.2.



Obr. 3.2: Mista ve kterých se určí směr.

## 6. Simulace

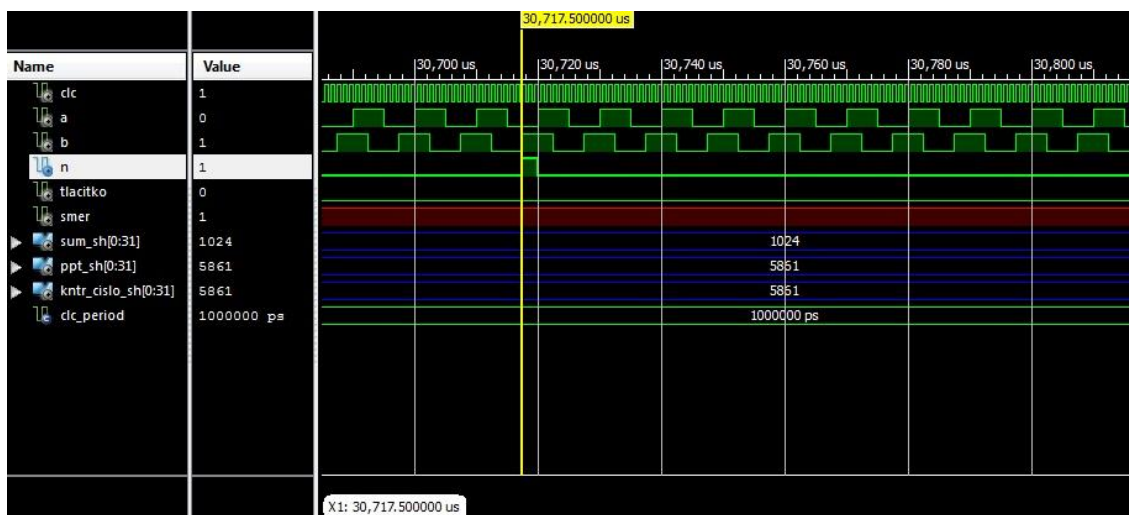
Simulace napsaného programu jsem provedl v programu ISim. To je vestavěný simulátor programovacího prostředí XilinxISE. Testbench má název *test2.vhdl*. Testbench umožňuje nastavit frekvenci oscilátoru, generování různých vstupních signálů.

Část VHDL kódu pro simulaci programu:

```
constant clc_period : time := 1 us;
clc_process :process                                -- Clock process definitions
begin
  clc <= '0';
  wait for clc_period/2;
  clc <= '1';
  wait for clc_period/2;
end process
N_process :process
begin
  n <= '0';
  wait for 10237500 ns;
  n <= '1';
  wait for 2500 ns;
end process;
StopyAB_process :process                          --generace signalu A B
begin
  a <= '1';
  wait for 2500 ns;
  b <= '0';
  wait for 2500 ns;
  a <= '0';
  wait for 2500 ns;
  b <= '1';
  wait for 2500 ns;
end process;
stim_proc: process                                -- Stimulus process
begin
  wait for 100 ns;
  wait;
end process;
```

Kód je součástí simulačního procesu. Jsou v něm popsány jednotlivé procesy generující hodinový signál, simulace výstupů enkodéru, stopy A i B, N signál.

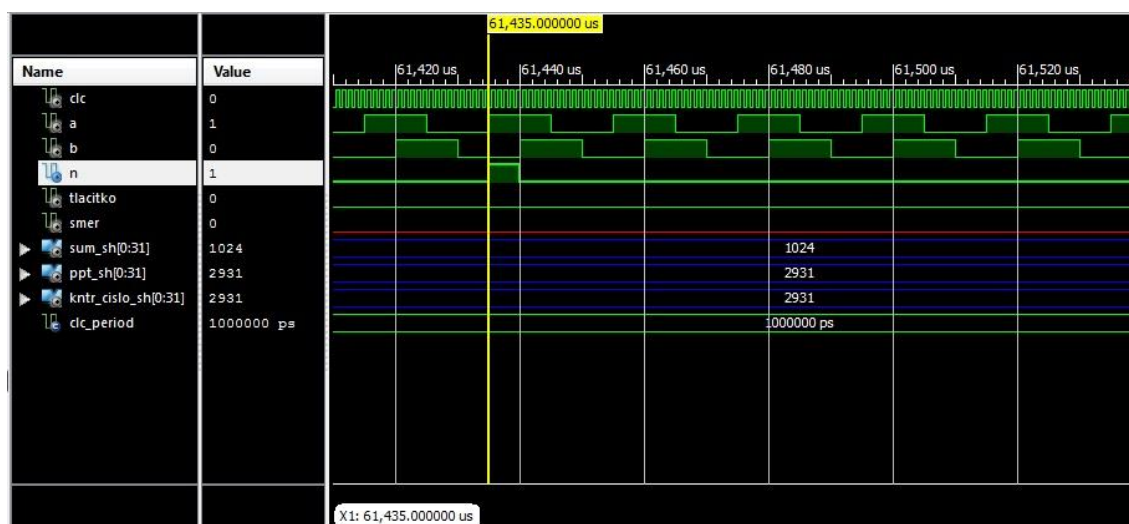
## Průběhy simulace:



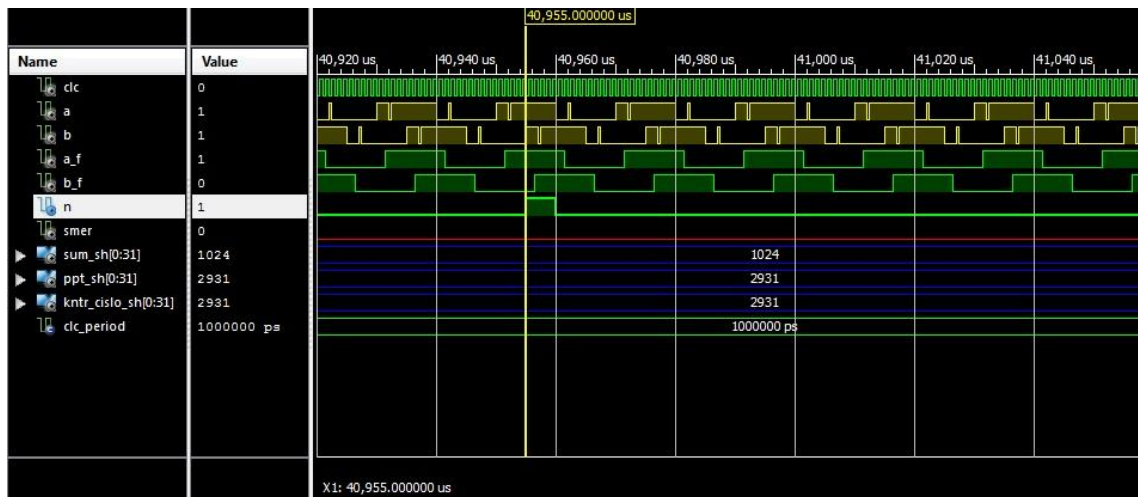
Vstupní signály na obrázku jsou označeny zelenou barvou. Červený signál je směr otáčení, aktuálně vidíme hodnotu 1, což znamená otáčení proti směru hodinových ručiček. Ze signálu stop A i B vidíme, že B signál předbíhá, což taky znamená směr proti směru hodinových ručiček. Tím pádem můžeme říct, že program funguje jak má.

Dále máme modré výstupní signály, signál `sum_SH` v poslení verzi programu sčítá počet pulzů mezi `N` signály, což vyjadřuje počet pulzů na otáčku, jeden z důležitých parametrů enkodéru. Nastavenou hodnotu mám 1024 pulzů na otáčku. Dále jsou signály se stejnou hodnotou, ale z různých zdrojů. `ppt_SH` je výsledek určování rychlosti otáček metodou měření frekvence a `kntr_cislo_SH` – metodou měření periody.

Výstupní hodnoty zcela odpovídají očekávaným a program počítá správně.



Zde je nasimulovaná jiná situace, směr otáčení je ve směru hodinových ručiček, to se dá ověřit stejným způsobem jak v minulém příkladu. `sum_SH` ukazuje stejný počet pulzů na otáčku a změnila se rychlost, jak vidíme na obrázku, ale hodnoty `ppt_SH` a `kntr_cislo_SH` jsou stejné.



V další simulaci byla ověřena funkce filtrace vstupních signálů. Žluté průběhy jsou vstupní signály, umyslně zašuměné. A signaly *a\_f* a *b\_f* jsou výstupní signály filtrace. Je vidět že proces proběhl v pořádku a šum neovlivnil výstupní hodnoty programu (pro porovnání parametry jsou stejné jak i u minulé simulace).



## 7. Závěr

Cílem této práce bylo navrhnout program pro vyhodnocování otáčkových čidel. Program byl realizován v jazyce VHDL v programovacím prostředí XilinxISE. Výsledkem vyhodnocování otáčkových čidel jsou hodnoty rychlosti otáčení v ot/min a směr otáčení. Rychlost byla odečtena dvěma metodami a jak vidět z simulace oba výsledky jsou stejné. Ale je to jen pro simulace v ideálních podmínkách, ve skutečnosti platí, že měření periody by bylo lépe použít pro nízké frekvence vstupního signálu a měření frekvence – pro větší frekvence vstupního signálu. Kvůli tomu, že výstupem programu je hodnota v otáčkách za minutu, je potřeba velký počet násobiček a děliček a to zatěžuje FPGA obvod a spotřebovává víc času na výpočet hodnot. Určení směru otáčení bylo realizováno s čtyřnásobnou přesností a jako výstupní signál nám stačil jen jeden bit, kde log.0 znamená směr otáčení ve směru hodinových ručiček a log.1 směr otáčení proti směru hodinových ručiček.

Simulace ukazují, že je navržený program funkční a po doplnění modulu umožňujícímu komunikaci po SPI sběrnici a nahrání na nějakou vývojovou desku může být použit v reálné aplikaci.

## 8. Seznam obrázků

|   |    |
|---|----|
| Obr. 2.1 Rozdělení snímačů otáček.[2].....  | 8  |
| Obr. 2.2 Princip zapojení tachodynamu.[2].....  | 9  |
| Obr. 2.3 Zjednodušená struktura absolutního enkodéru [4] .....                                | 10 |
| Obr. 2.4 Konstrukční provedení indukčního snímače.[5].....                                    | 11 |
| Obr. 2.5 Princip činnosti kapacitního snímače.[2] .....                                       | 12 |
| Obr. 2.6 Princip činnosti Hallova snímače.[6] .....   | 13 |
| Obr. 2.7 Konstrukční provedení magnetického snímače. ....                                     | 13 |
| Obr. 2.8 Princip činnosti optického snímače.[8] .....   | 14 |
| Obr. 3.1 Vstupní signaly inkrementálního čidla. [7] .....                                     | 15 |
| Obr. 3.2 Měření periody. ....   | 16 |
| Obr. 3.3 Měření frekvence.....  | 16 |
| Obr. 3.4 Rozdíl vystupních signálů při různých směrech otáčení.....                           | 16 |
| Obr. 3.5 Jednotlivé způsoby dekódování kvadrturních signálů .....                             | 17 |
| Obr. 4.1 Srovnání asynchronní (SCI) a synchronní (SPI) komunikace<br>podle počtu vodičů ..... | 19 |
| Obr. 4.2 Vnitřní zapojení Master - Slave .....  | 20 |
| Obr. 4.3 Jednoduché zapojení Master - Slave .....   | 20 |
| Obr. 4.4 Nezávislé zapojení pro více podřízených.....   | 21 |
| Obr. 4.5 Kaskádní zapojení pro více podřízených.....  | 21 |
| Obr. 4.6 Princip komunikace.....  | 22 |
| Obr. 4.7 Řežimy SPI. ....   | 23 |
| Obr. 5.1 Stavový diagram tlačítka. ....   | 26 |

## 9. Seznam použité literatury

- [1] Pinker, J., Poupa, M., Číslicové systémy a jazyk VHDL, BEN, 2009
- [2] Zbynek Jehlar, Vliv externích elektromagnetických poli na funkci snímačů otáček, diplomová práce, Brno, 2009 [online]. Dostupné z WWW: <[https://www.vutbr.cz/www\\_base/zav\\_prace\\_soubor\\_verejne.php?file\\_id=14936](https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=14936)>
- [3] Vaclav Kanta, IP funkce pro vyhodnocení otáčkového čidla, diplomová práce, 2013
- [4] [online]. Dostupné z WWW: <<http://www.tamagawa-seiki.com/english/encoder/>>
- [5] Indukční snímače, přednáška, [online]. Dostupné z WWW: <<http://www.skolahostivar.cz/DownloadPF/INDUK%C4%8CN%C3%8D%20SN%C3%8DMA%C4%8CE.ppt>>
- [6] [online]. Dostupné z WWW: <<http://automatizace.hw.cz/magneticke-senzory-s-hallovym-efektem-1-princip>>
- [7] [online]. Dostupné z WWW: <[http://www.kit-e.ru/articles/dsp/2003\\_6\\_106.php](http://www.kit-e.ru/articles/dsp/2003_6_106.php)>
- [8] [online]. Dostupné z WWW: <<http://eluc.cz/verejne/lekce/1578>>
- [9] [online]. Dostupné z WWW: <<http://robocraft.ru/blog/arduino/518.html>>
- [10] [online]. Dostupné z WWW: <<http://www.gaw.ru/html.cgi/txt/interface/spi/index.htm>>
- [11] [online]. Dostupné z WWW: <<https://eewiki.net/pages/viewpage.action?pageId=4096096&&>>
- [12] [online]. Dostupné z WWW: <<http://chipenable.ru/index.php/programming-avr/item/137-avr-spi-module.html>>
- [13] [online]. Dostupné z WWW: <<http://www.root.cz/clanky/externi-seriove-sbernice-spi-a-i2c/>>

# Příloha

## Zdrojový kód programu vyhodnocování otáček

```
-----  
-- Company: FEL CVUT  
-- Engineer: Kuanysh Stikeyev  
--  
-- Create Date: 15:39:34 03/12/2015  
-- Design Name:  
-- Module Name: pr1 - Behavioral  
-- Project Name:  
-- Target Devices:  
-- Tool versions:  
-- Description:  
--  
-- Dependencies:  
--  
-- Revision:  
-- Revision 0.01 - File Created  
-- Additional Comments:  
--  
-----  
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use IEEE.NUMERIC_STD.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;  
-- Uncomment the following library declaration if instantiating  
-- any Xilinx primitives in this code.  
--library UNISIM;  
--use UNISIM.VComponents.all;  
  
entity pr1 is  
    port (  
        a,b,tlacitko, clc, n : in std_logic;  
        smer : out std_logic;  
        sum_SH : out std_logic_vector(0 to 31);  
        ppt_SH : out std_logic_vector(0 to 31);  
        kntr_cislo_SH : out std_logic_vector(0 to 31)  
    );  
end pr1;  
  
architecture Behavioral of pr1 is  
    ---VYPLNIT !!!  
    constant clc_frekv : integer := 1000000 ;           -- Frekvence hodin (1 us => 1 MHZ)  
    constant pno : integer := 1024;                   -- pulzu na otacku, pocet carek na impulznim disku  
    constant time_konst : integer := 1000;           -- (* clc_period) doba po ktere bude probedeno  
    vyhodnocovani rychlosti  
    constant tmr_max : integer := 65536;             -- (2^16) Timer max value  
    constant ck : integer := 1000;                   -- (clc_frekv / time_konst)  
  
    signal a_pam, b_pam, n_pam, tlacitko_pam, t1,n1,s1, s2, s3, s4,d1, pulz: std_logic;  
    signal stav : std_logic_vector (1 downto 0) := "00";  
    signal sum : integer := 0;  
    signal sumRes : std_logic;  
    -- signaly pro filtrace  
    signal a_mem, b_mem : std_logic := '0';  
    signal a_f, b_f : std_logic;  
    -- signaly pro otackomer  
    signal tmr1 : integer := 0;  
    signal registr_ppt1, registr_ppt2, ppt: integer:= 0;
```

```

signal kolo: integer:= 1;
signal res_kolo : std_logic;
signal registr_kntr1, registr_kntr2, kntr_cislo: integer:= 0;
signal sum_a, sum_b : integer := 0;
signal tempRoz, tempRoz2, rozdil : integer := 0;
signal vysledek, vysl2: integer:= 0;
signal detkol, detzmen : std_logic := '0';
begin
    process(clc)
    begin
        if clc'event and clc = '1' then
--filtration
            if a_mem = '0' then                --stopa A
                if a = '0' then
                    a_f <= a;
                end if;
            else
                if a = '1' then
                    a_f <= a;
                end if;
            end if;
            if b_mem = '0' then                --stopa B
                if b = '0' then
                    b_f <= b;
                end if;
            else
                if b = '1' then
                    b_f <= b;
                end if;
            end if;
            a_mem <= a;                        --nova pamet
            b_mem <= b;
-- tlacitko na zmenu modu, po4itani(1 nabeznych, 2 sestupnych, nebo 3 obou hran)
            t1 <= (not tlacitko_pam) and tlacitko; --nabezna hrana tlacitka, detekce na
stisknuti
-- kanal A
            s1 <= (not a_pam) and a_f; --nabezna hrana signalu vstupu
            s2 <= (not a_f) and a_pam; --sestupna hrana signalu
-- kanal B
            s3 <= (not b_pam) and b_f; --nabezna hrana signalu vstupu
            s4 <= (not b_f) and b_pam; --sestupna hrana signalu
--multiplexer
            case stav is
                when "00" => pulz <= s1;
                when "01" => pulz <= s2;
                when "10" => pulz <= (s1 or s2);
                when others => null;
            end case;
--nulovani sum pro pripad zmeny stavu_tlacitka nebo kdy pride n-signal
            n1 <= (not n_pam) and n; --nabezna hrana signalu N
            if sumRes = '1' or n1 = '1' then
                if n1 = '1' then
                    sum_SH <= std_logic_vector(to_unsigned(sum,32));
                end if;
                sum <= 0;
--kdyz pride pulz pricte 1
                elsif pulz = '1' then
                    sum <= sum + 1;
                end if;
--aktualizace pameti
                tlacitko_pam <= tlacitko;
                a_pam <= a_f;
                b_pam <= b_f;
            end if;
        end if;
    end process;
end;

```

```

        n_pam <= n;

--      Timer1
if tmr1 = tmr_max then
    res_kolo <= '1';
    tmr1 <= 0;
    sum_a <= 0;
else
    tmr1 <= tmr1 + 1;
    res_kolo <= '0';
end if;

if pulz = '1' then
    sum_a <= sum_a + 1;
end if;
end if; --clock end
end process;

process (t1) --pozoruje zmenu t1
begin
    if t1 = '1' then --kdyz byla stisknuta
        if stav = "00" then --(mozhne stavy: 00,01,10)
            stav <= "01";
            sumRes <= '1'; -- --kdyz se zmeni stav reset sum
        elsif stav = "01" then
            stav <= "10";
            sumRes <= '1'; -- --kdyz se zmeni stav reset sum
        elsif stav = "10" then
            stav <= "00";
            sumRes <= '1'; -- --kdyz se zmeni stav reset sum
        end if;
    else sumRes <='0';
    end if;
end process;

--process zapisuje do registru hodnoty na zacatku a konci preddefinovaneho casu(time_konst)
--      z hodnot dostaneme PocetPulzu za jednotku casu
process (tmr1)
begin
    if res_kolo = '1' then
        kolo <= 1;
        registr_ppt2 <= 0;
        registr_ppt1 <= 0;
    end if;
    if tmr1 = time_konst * kolo then
        registr_ppt2 <= registr_ppt1;
        registr_ppt1 <= sum_a;
        kolo <= kolo + 1;
        ppt <= registr_ppt1 - registr_ppt2;
        if ppt>0 then
            detkol <= '1'; --detekce zmeny kola
        end if;
    end if;
    if detkol = '1' then
        rozdil <= 60* ck * ppt;
        tempRoz <= vysledek * pno;
        if(tempRoz < rozdil) then
            vysledek <= vysledek + 1;
        else
            detkol <= '0';
            ppt_SH <= std_logic_vector(to_unsigned(vysledek,32));
            vysledek <= 0;
            tempRoz <= 0;
        end if;
    end if;
end process;

```

```

        end if;
    end if;
end process;
-- Metoda mereni rychlosti merenim periody
process (s1)
begin
    if res_kolo = '1' then
        registr_kntr1 <= 0;
        registr_kntr2 <= 0;
    end if;
    if s1 = '1' then
        registr_kntr2 <= registr_kntr1;
        registr_kntr1 <= tnr1;
        kntr_cislo <= (registr_kntr1 - registr_kntr2) * pno;
        if kntr_cislo > 0 then
            detzmen <= '1';
        end if;
    end if;
    if detzmen = '1' then
        tempRoz2 <= vysl2 * kntr_cislo;
        if(tempRoz2 < 60000000) then           -- 600000000 us = 1 min
            vysl2 <= vysl2 + 1;
        else
            detzmen <= '0';
            kntr_cislo_SH <= std_logic_vector(to_unsigned(vysl2,32));
            vysl2 <= 0;
            tempRoz2 <= 0;
        end if;
    end if;
end process;
--Urcovani smeru otaceni
process (s1,s2,s3,s4)
begin
    if s1 = '1' then --n_A
        if b_f = '1' then
            smer <= '1'; --2
        else
            smer <= '0'; --1
        end if;
    end if;
    if s2 = '1' then --s_A
        if b_f = '1' then
            smer <= '0'; --3
        else
            smer <= '1'; --4
        end if;
    end if;
    if s3 = '1' then --n_B
        if a_f = '1' then
            smer <= '0'; --5
        else
            smer <= '1'; --6
        end if;
    end if;
    if s4 = '1' then --s_B
        if a_f = '1' then
            smer <= '1'; --8
        else
            smer <= '0'; --7
        end if;
    end if;
end process;
end Behavioral;

```