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

Rozhraní otáčkového čidla

Bakalařská práce

Autor práce: Kuanysh Stikeyev
Vedoucí práce: Ing. Jan Bauer, Ph.D.
Studijní program: Elektrotechnika, enegetika 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

Prohlašuju, že jsem svoji bakalařskou práci na tema „Rozhrani otačkového čidla“ vypracoval samostatně, na zakladě rad a pokynů vedoucího práce. Použil jsem přitom své znalostí a podkady uvedené v přiloženém seznamu.

Dále prohlašuji, že veškerý software, použitý při řešení bakalařské práce, je legální.

.....

Kuanysh Stikeyev

Poděkovaní

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 otačkových čidel. Dále je zde probrána problematika měření periody a frekvence, jejich vzájemné porovnání. Je zde take popsana komunikace u mikropočitačů. Podrobněji vysvětlena sběrnice SPI, schemy zapojeni a princip komunikace. Práce se dále zabývá návrhem a implementací program pro vyhodnocení otaček a směru otačení. 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

1. Úvod

Cíl bakalařské práce je navrhnout program v jazyce VHDL pro vyhodnocení snímačů otáček. Program musí 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 otačení hřídele.

V kapitole č.2 je proveden podrobný rozbor otačkových čidel, rozdělení, princip činosti a jejich vystupní sidnaly. V kapitole č.3 je probrány metody vyhodnocovani rychlosti a směru otačení, a také problematika měření periody a frekvence a jejich vzájemné porovnání. V kapitole č.4 je vysvětlena komunikace u mikropočitačů. A detailně rozebrana sběrnice SPI, schemy zapojeni, princip komunikace a řežimy prace.

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

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

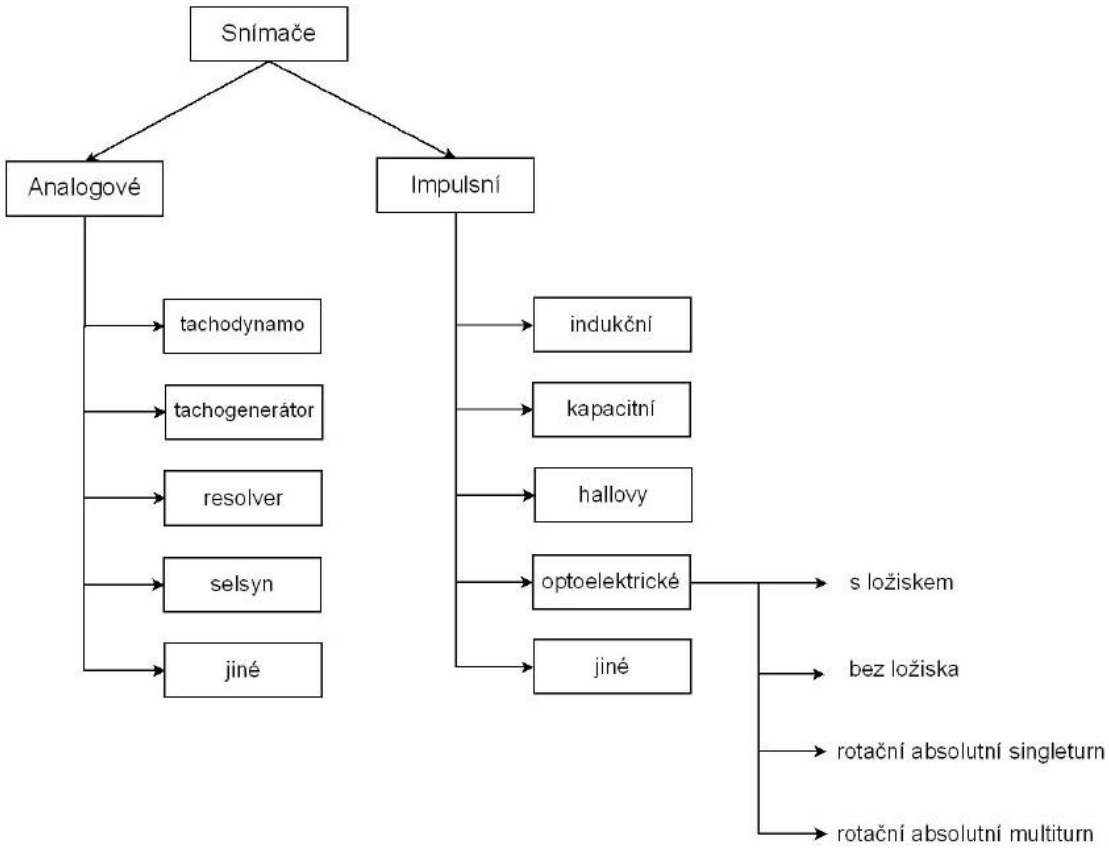
Závěrečná kapitola č. 7 shrnuje výsledky práce.

2. Snímače otáček

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

2.1. Rozdělení snimačů otáček

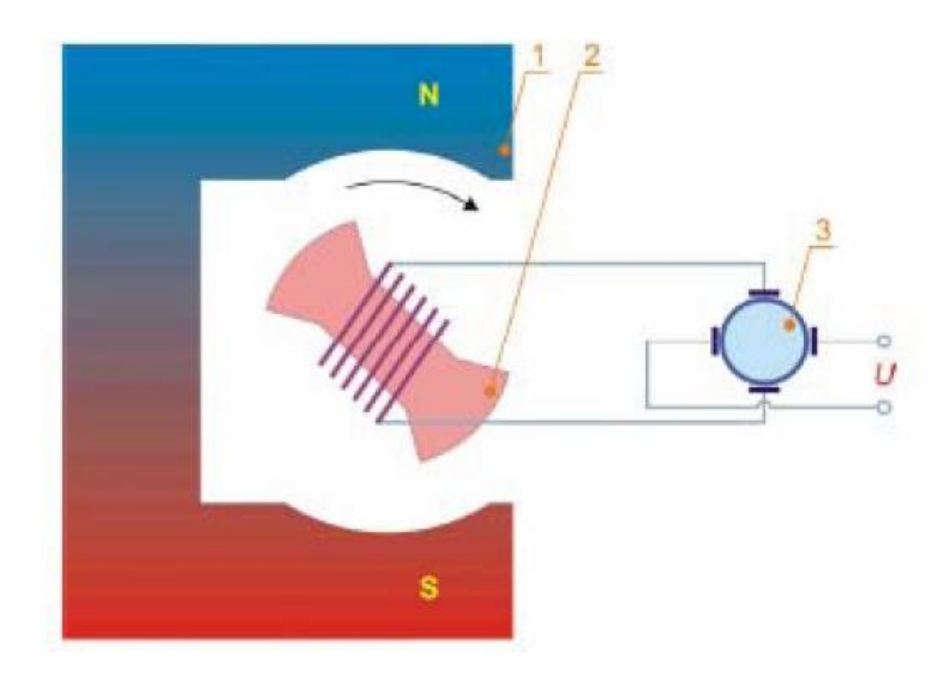
Snimače otaček můžou pracovat na zakladě různých fyzikalních principů. Rozdelime podle formy vystupu do dvou skupin, na analogové a impulzní. Analogové snimače mají vystupní napěti uměrné rychlosti otačeni. Vystpem impulzních snimačů je obdelnikový signal, který původně byl sinusové podoby do zesilení a zpracovaní uvnitř snimacího zařizeni. Na Obr. 2.1 můžeme vidět jedno z možných rozdělení snímačů otáček.[2]



Obr. 2.1 Rozdělení snímačů otáček.

2.2. Analogové snimače otaček

Jak už bylo zmíněno dřivě princip prace analogových snimačů je založen na zavislosti vystupního napěti na otačkach. Přikladem takových smimačů jsou tachodymano, tachogenerator, resolver a jiné. V ramci teto praci podrobněj probraný nebudou. Protože jde o digitlni zpracovaní digitalnícho vystupu a u analogových snimaču to neni možné, bez třetích pomucek.



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

2.3. Impulzní snimače otaček

Další nazev pro daný druh otačkových čidel je enkodery. Enkoder je specialni zařizeni, pomoci kterého se zjisti poloha rotujicích hřidelů. Jejich zakladní funkce je měření linearních a úhlových přemistěni. V daném zařizeni dohazi k tranformaci mechanického pohybu v elektrické signaly, určujicí polohu objektu, poskytuji informace o úhlu otačeni hřidla, jeho polohu a směr otačeni. Bez enkoderů nepředstavitelne moderní systemy přesného pohybu. V každém zařizení, kde dohazi k točeni, pro kontrolu přesnosti toho točeni nutně mít zpětnou vazbu mezi točivymi a netočivými časti tohoto systemu. Například pravě pomoci enkoderu můžeme sledovat a kompenzovat propouštene kroky při skluzu krokového motoru. Rozdělime enkodery na zakladě dvou zasadně rozlišných principů:

- Inkrementalní
- Absolutní

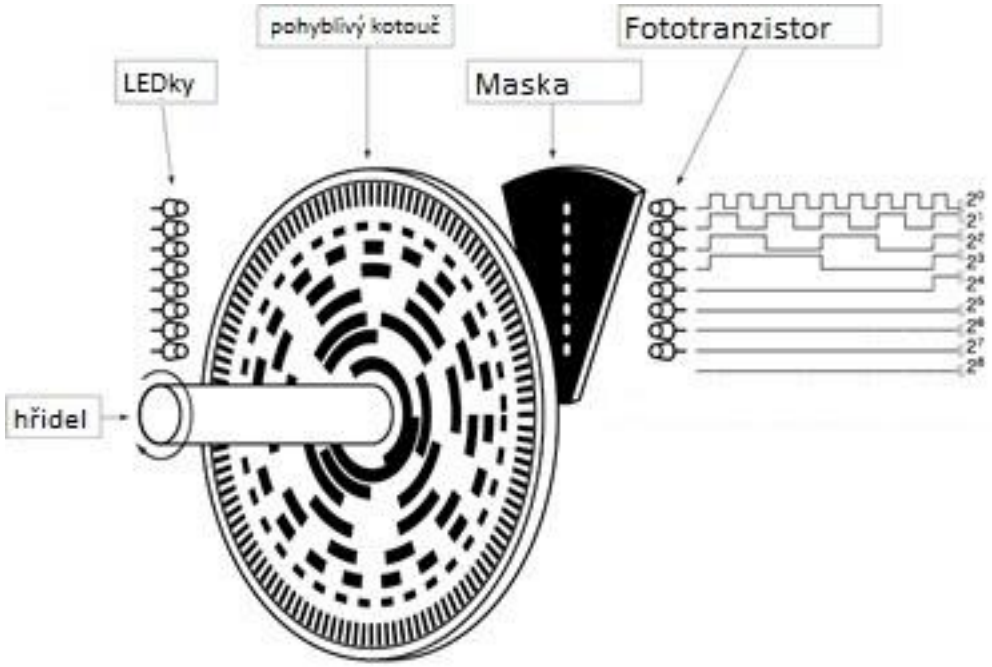
2.4. Inkrementalní enkodery.

Inkrementalní enkoder generuje data o polohe objektu prostřednictvim elektrických impulzů. Počet impulzů na jednu otačku je hlavní pracovní parametr, charakterizujicí inkrementalní enkodery. Impulzy vznikají, jen když se hřidlo otači. Jakmile hřidlo zastavime vysilani signalů přestava a pomoci toho typu enkoderu polohu hřidla už nedozvíme. Což znamená, že tento typ čidla bylo by vhodne použit na měření rychlosti otačeni a kvůli tomu že nepamatuje minulou polohu tak před novým spouštením musíme nastavit na počateční polohu.

Inkrementalní enkodery jsou zdrojem impulzů, teto impulzy vznikají na zakladě různých jevů. Na obvodu rotující hřídele, která se otáčí úhlovou rychlostí ω 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 enkodery.

Absolutní otačková čidla poskitují data o počtu otaček, uhlu, a pohoze hřidla pomoi unikatních kodů, odpovidajicích každemu kroku. Čislo unikatních kodu na jednu otačku a počet otaček jsou zakladními charakteristiky danného typu zařizení. Počateční nastavení čidla neni potřeba, protože absolutní poloha se určí unikatním kodem. Existuje jednootačkové a viceotačkové modely daného typu čidla.



Obr. 2.3 Zjednodušená struktura absolutního enkoderu [4]

2.6. Indučí snimače.

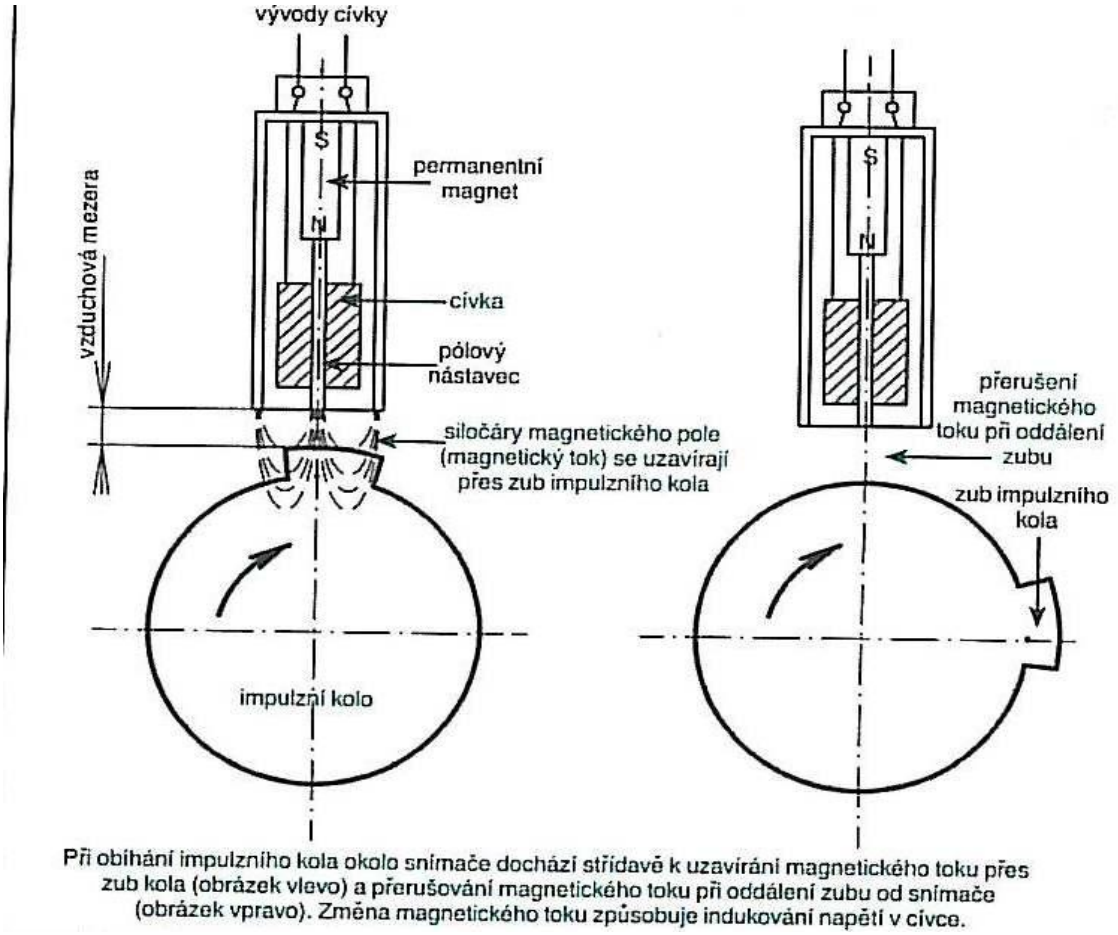
Pracuji na fyzikalním principu změny magnetického toku prohazejicího přes civku, čimž se projeví změnou indukovaného napěti na civke dle vztahu:

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

kde:

u_i	je indukované napěti
N	je počet zavitů civky
$\frac{\Delta\Phi}{\Delta t}$	je změna magnetického toku v čase

Změna magnetického toku je vyvolana pohybem zuba impulzního kola, když zub, udělany z feromagnetického materialu, prohazi v blizkozti snimače zmenšuje magnetický odpor a tim zvětšuje magnetický tok prohazející civkou. Konstrukční provedení takového snimače je vidět na Obr. 2.4.



Obr. 2.4 Konstrukční provedení indukčního snimače.[5]

2.7. Kapacitní snimače.

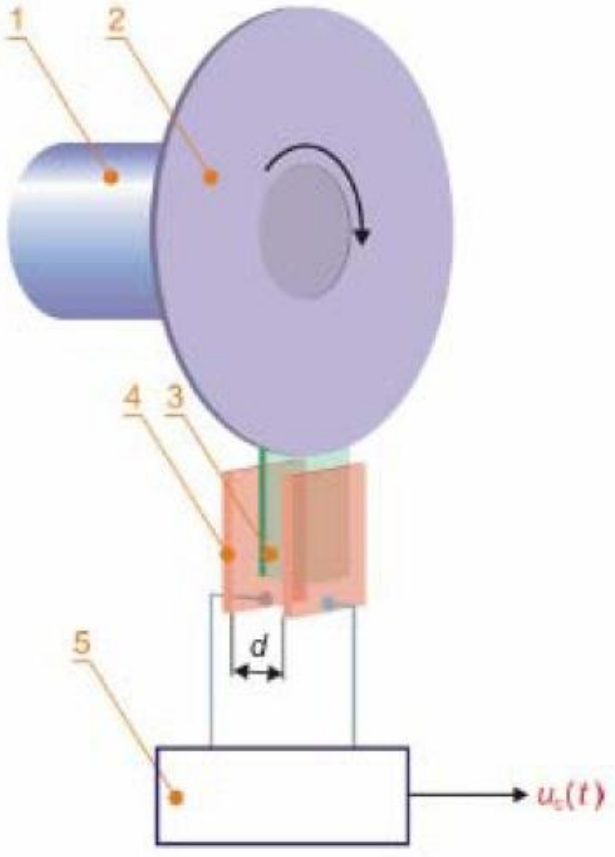
Kapacitní snimače využuvají principu zmeny kapacity deskového kondensatoru. Kapacita deskového kondensatoru je dana vztahem:

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

kde:

ε je permitivita dielektrika,
S je obsah desky kondensátoru,
d je vzdálenost desek kondensátoru.

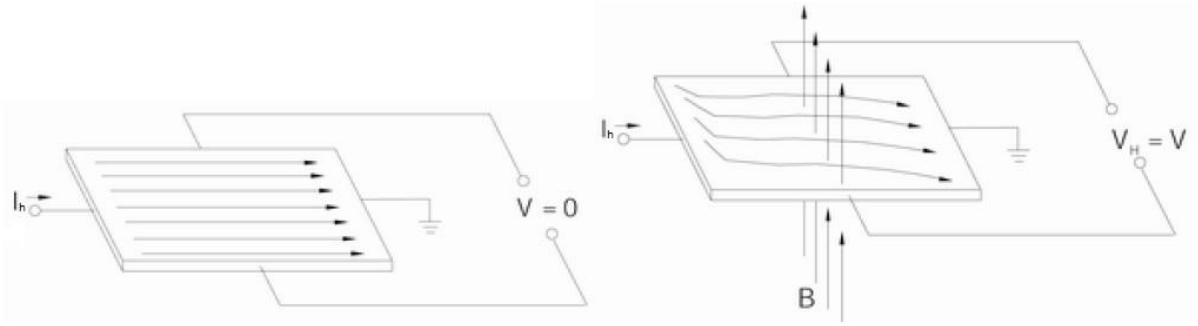
Na hřidelu, stejně jak u indukčního snimače, je umisteno impulzní kolo a deskový kondenzator umisten tak aby zub hřidelu prohazel mezi deskami kondezatoru, tim to dějem permetivita prostředi mezi deskami se zvětšuje a nasledně i kapacita kondenzatoru. Tohle to bude zaznamemno měřičem kapacity jako pulz a z počtů pulzu, lze bude stanovit rychlost otačeni hřidelu.



Obr. 2.5 Princip činosti kapacitního snimače.
1-hřídel,2-kotouč,3-kovová destička,4-desky kondensátoru,5-měřič capacity[2]

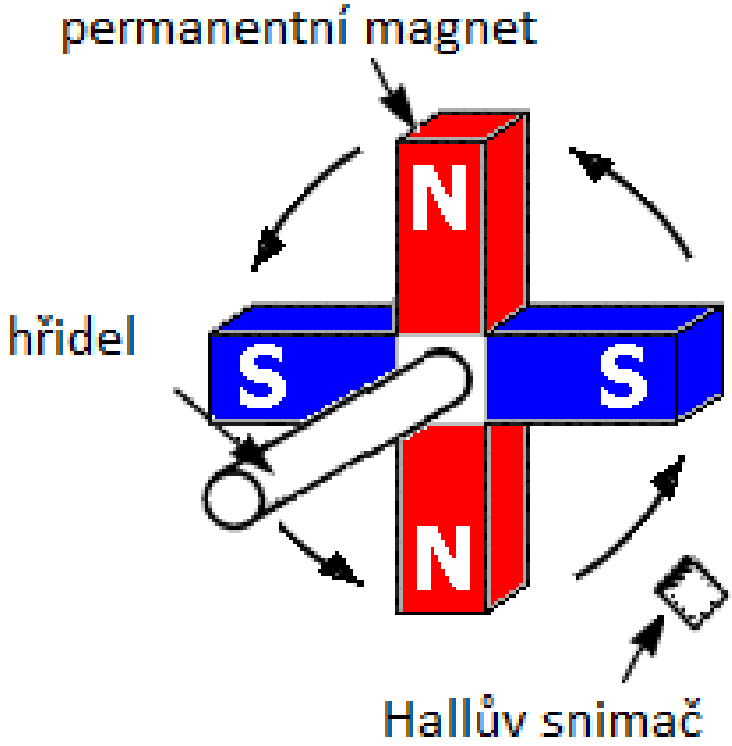
2.8. Magnetické snimače

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 vyhodnocovaní vysledku generovano napěti je preliš malo a musí se ještě zesilit.



Obr. 2.6 Princip činosti Hallova snímače.[6]

Jeden z možných konstrukčních provedení je zobrazen na Obr. 2.7. Enkoder se skládá z hřidele s permanentní magnety a Hallova snimače, který registruje posloupnost prochazeni magnetických polů, s čehož určuje rychlost a směr otačeni.

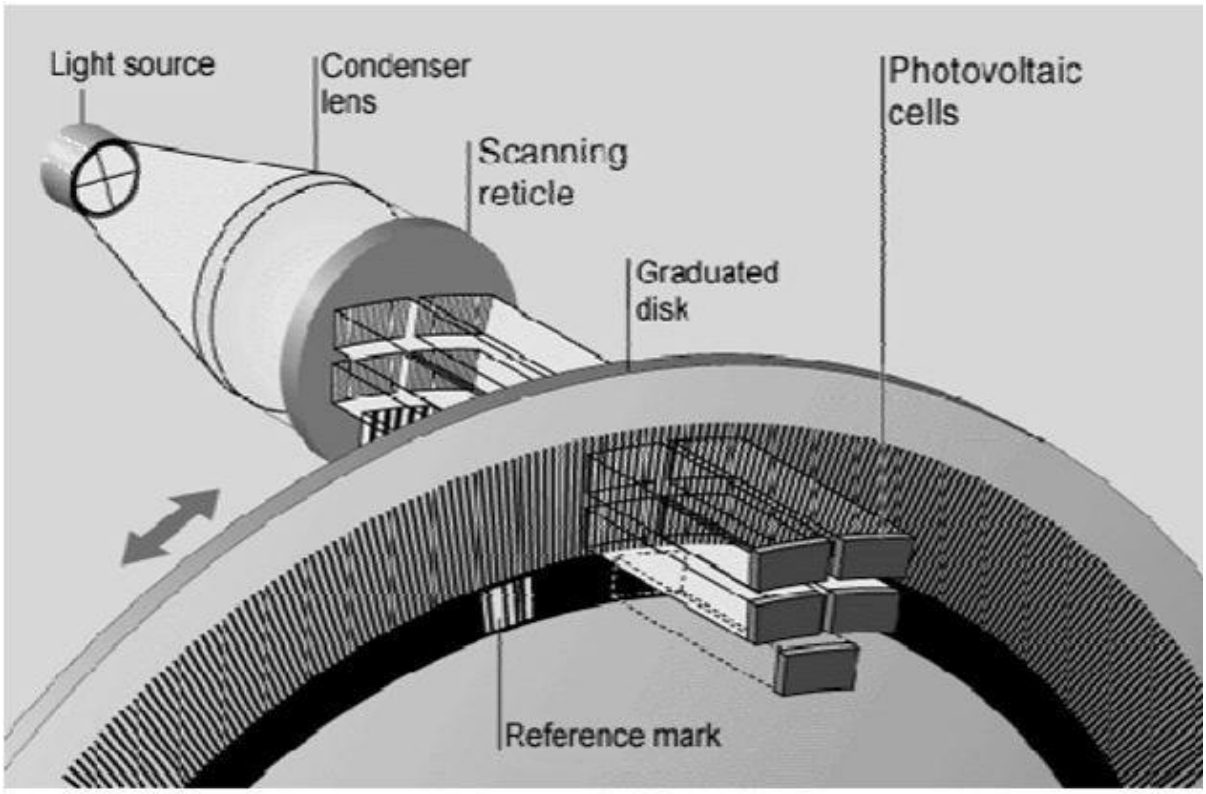


Obr. 2.7 Konstrukční provedení magnetického snimače.

2.9. Optické snimače

Jako jeden z nejvíce používaných rotačních inkrementálních snímačů otáček je optické snímače.

Princip senzoru spočívá ve clonění světelného toku mezi zdrojem světla a fotodetektorem. Zakladém je pulzní disk, tento disk mechanický spojen s hřidlem. Disk obsahuje světlá a tmavá pole. Světlo generované kvalitními diodami, pracujícími většinou v infračervené oblasti spektra, prochází přes membránu a tento pulsní disk a je zachyceno fotodetektorem umístěným z jeho druhé strany. Při otáčení hřídele pak disk střídavě světlo propouští a nepropouští (zacloňuje fotodetektor). Princip činnosti je zobrazen na Obr. 2.8. Počet tmavých (neprůhledných) 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°, 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.

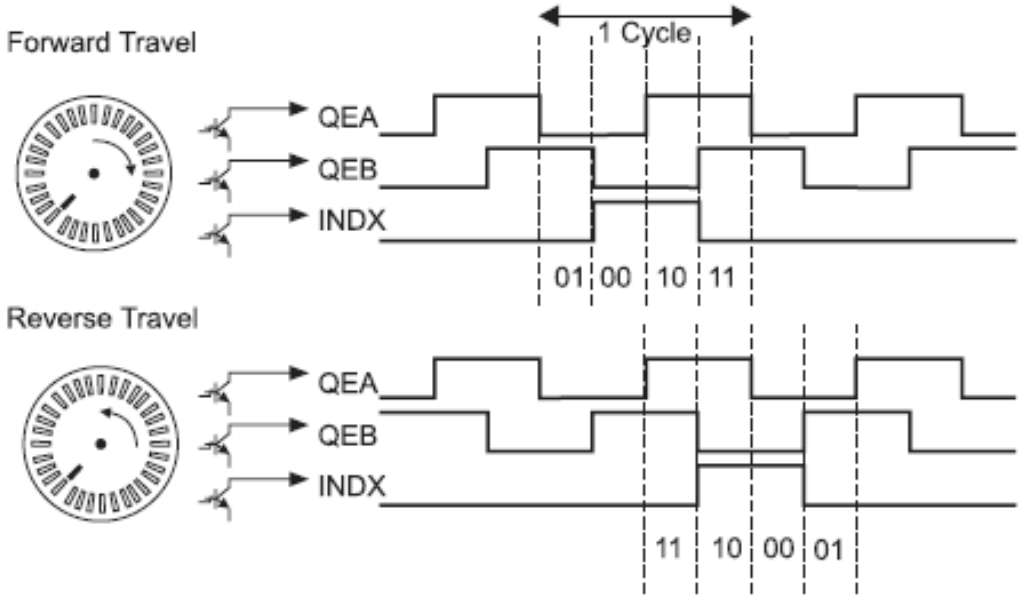


Obr. 2.8 Princip činosti optického snimače.[8]

Existuje jěště několik principu jak se dá usuzovat rychlost otačeni atd, ale kvůli tomu že nejsou moc rošiřený nebo přesný, zastavíme na těch typech co jsem popsal.

3. Metody vyhodnocovani

Jak už bylo zmíněno dřivě vystupem z inkrementalního čidla je dva obdelnikových průběha vzajemně posunutých o 90 stupně. Obvyklě je ještě jeden signal – nulový, ten se objevi jednou za otačku.



Obr. 3.1 Vystupní signaly inkrementalního čidla. [7]

Dale signale potřeba dekodovat, aby dostat udaje které jsou lip pochopitelné, jako rychlost otačeni a směr otačeni.

3.1. Metody vyhodnocovani rychlosti otačeni.

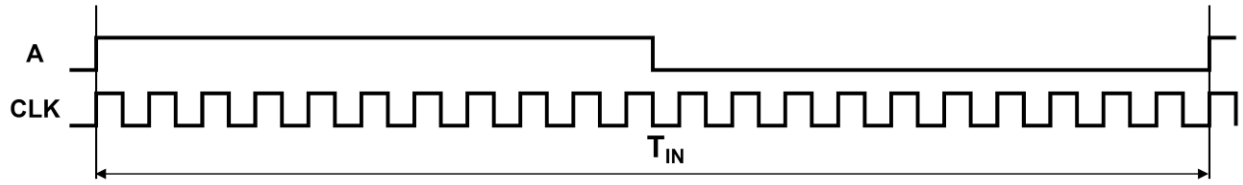
Rychlost otáčení hřídele zjistíme měřením periody nebo frekvence pulzů. Informace do této kapitoly jsem získal z [3].

3.1.1.Měření periody

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

$$T_{IN} = \frac{1}{f_{IN}}$$

Měření periody neznámého signálu se provede čítáním hodinových pulsů 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.



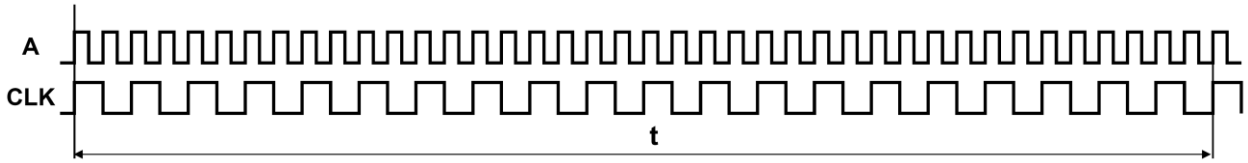
Obr. 3.2 Měření periody.

3.1.2.Měření frekvence

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

$$f = \frac{n}{t}$$

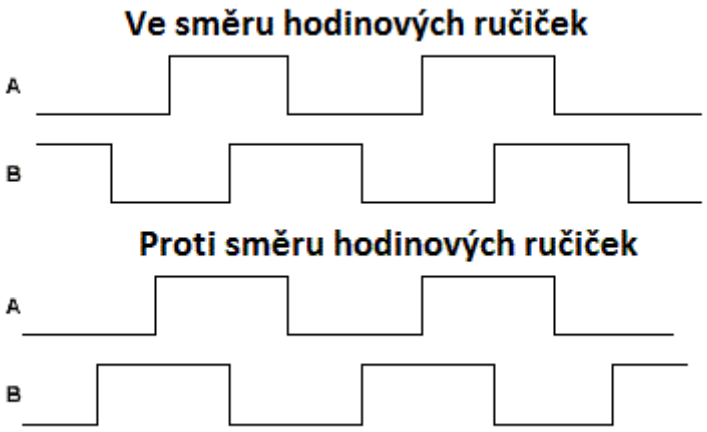
kde *n* je počet pulsů a *t* je časový interval v kterém jsme pulsy počítali. Měření frekvence je přesnější pro měření signálů o vysoké frekvenci než měření periody. Měření frekvence je znázorněno na Obr. 3.3



Obr. 3.3 Měření frekvence.

3.2. Metoda určovani směru otačeni.

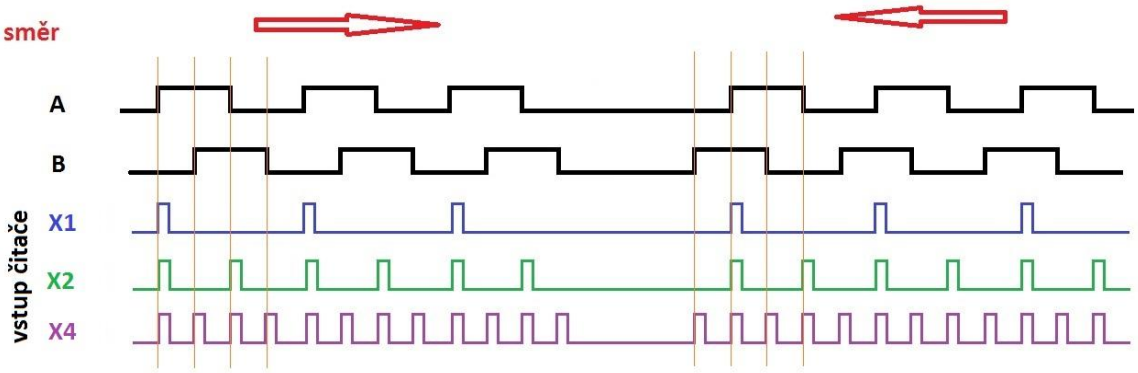
Hlavní princip spočivá v tom, že zaznamenovame zmenu signalu na jedné stopě a ve stejný okažik odečitame hodnotu druhé stopy a zakladě toho určujeme směr.



Obr. 3.4 Rozdil vystuních signalů při různých směrach otačeni.

Data mužeme odečitat s různou citlivosti. Celkem mužeme vydeli tři různé varianty.

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



Obr. 3.5 Jednotlivé způsoby dekódování kvadraturních signálů

4. Komunikačni rohrani.

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

Další důležutou časti je taky komunikace mezi řidicí jednotkou a ostatnímy častmi sostavy. Existuje dva druhy komunikace u mikropočitačů. Seriové 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). Dale mužeme rozdelit komunikace podle synchronnosti. Synchronní komunikace je, když přijímač a vysílač pracují podle jednoho signalu, napřiklad hodinového signalu. Asynchronní komunikace probihá bez synchronizačního signalu, ale řizeno pomoci spealních symbolů (start bit, stop bit) a potřebné časovaní vyhodnoceno každým zařizenim pomoci vlastních hodin. Zakladní rozdil mezi synchronní a asynchronní komunikace je v tom, že při synchronní komunikace se kromě dat přenášejí po zvlaštní lince jěště synchronozační signal, tak zvané „společné hodiny“.

Komunikace po sériové lince je vlastní mnoha mikropočítačům. Umožňuje velmi jednoduchý přenos dat mezi mikropočítačem a jiným zařízením vybaveným sériovým rozhraním, např. počítačem PC. A proto se podrobnějc podiváme na seriovou komunikace.

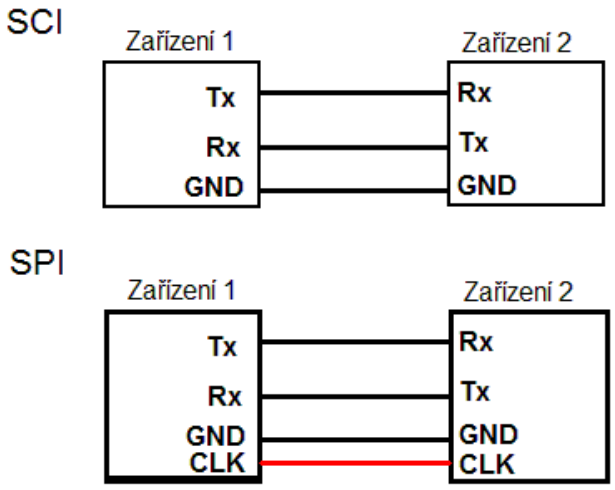
4.1. Synchronní a asynchronní rozhraní

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 pro obousměrný provoz pro asynchronní komunikace potřebujeme minimalně 3 vodiče, a pro synchronní budeme potřebovat jeden vodič navic pro hodinový signal.



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

V ramci teto prace nas zajimá synchronní seriová komunikace (SPI) a proto probereme ji podrobneji.

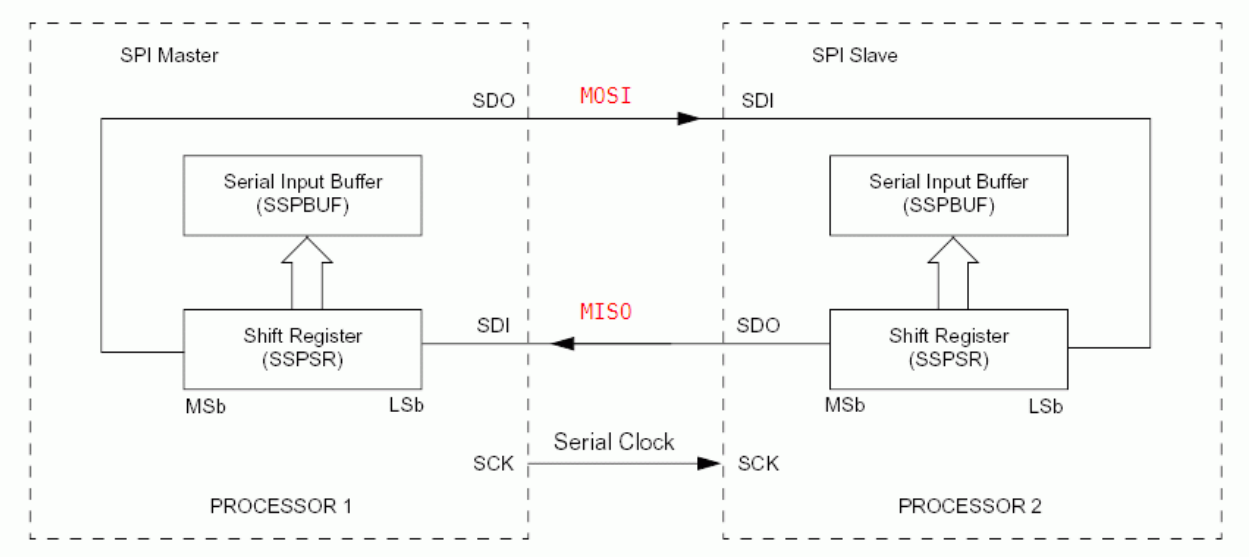
Poprvé SPI sbernice byla navržena a zpracovana kompanií Motorola, ale v současné době ji používají všechny vyrobce.

Používá se pro komunikace mezi řidicími mikropičitače a ostatními integrovanými obvody (A/D převodniky, displeje ...). Komunikace je realizována pomocí společné sběrnice.

Zařízení na SPI sběrnici jsou řidicí (Master) a podřizené (Slave). Master řidí komunikace podle hodinového signalu a určuje, s jakým zařizením bude komunikovat. Slave vysilá data podle hodinového signalu, pokud je aktivovan.

4.2. Vnitřní zapojení

Oba zařizeni v nejjednodušem připade mají jeden datový zachytný registr (*Serial Input Buffer*) a a posuvný registr (*Shift Register).* Datové registry zajištuje, aby nedochazelo ke ztratě dat.

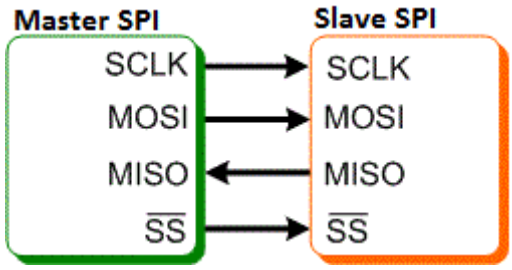


Obr. 4.2 Vnitřní zapojení Master - Slave

V SPI komunikace vysilaní a přijem dat, jak je to vidět z obrazku, nedelitelný proces, a oba dvě procesy probihaji současně. Tipickou pracovní frekvenci je v rozmezi 1-50 MHz. Diky jednoduhosti algoritmu prenašeni dat, SPI je rozšiřené v nejrůznějších electronických zařizení.

4.3. Schemy zapojeni

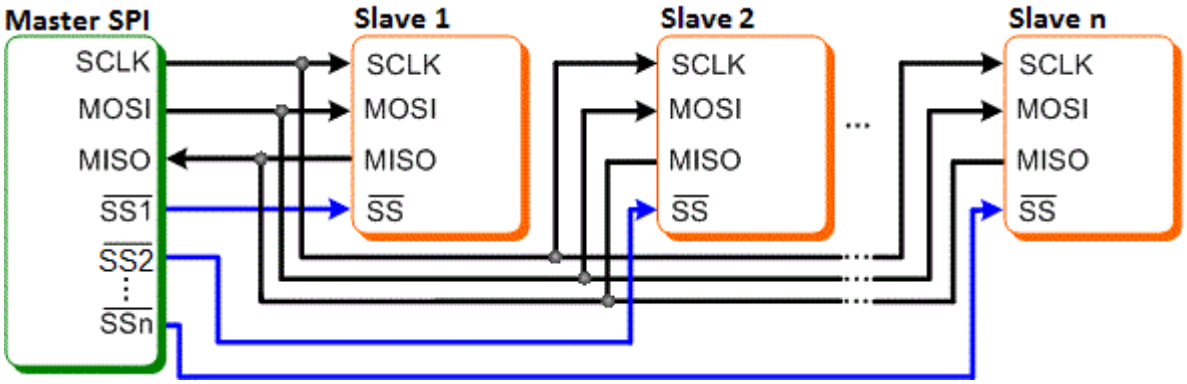
Nejjednodusší variantou je jeden řidicí a jeden podřizený.



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

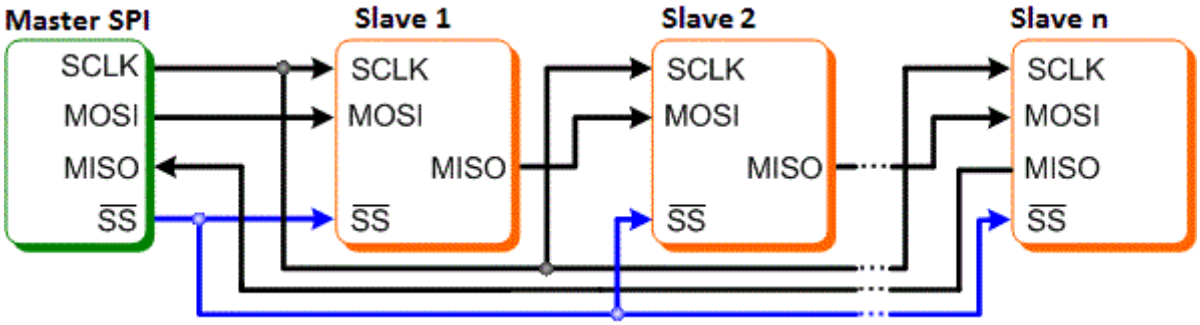
Pro připad, kde podtřebujeme připojit k masteru vic než jedenoho podřizeného, využivá se jedna ze dvou variant, nezavislé (paralelní) nebo kaskadní (seriové).

Častej se využiva nezavislou variantu. Tady všechny signalu, kromě volby podřizeného, jsou paralelní a uvedením nějakého signalu SS do log 0, master zvolí podřizeného z kterým bude komunikovat. Ale hlavní nevyhodou je, že s rostoucím počtem podřizených, je potřeba doplňkové linie. Celkový počet linie pro komunikace se rovná 3+n, kde n je počet podřizených.



Obr. 4.4 Nezavislé zapojení pro vice podřizených.

Kaskadní zapojení eliminuje tento problem, protože zde ze všech zařizeni vznika jeden velký posuvný registr. Toho lze dosahnout tím, že vystup z podřizeného se připojí na vstup dalšiho podřizeného, a takhle všechna zařizeni se spojí do kola. Nevyhodou je, že ne každý mikropočitač je shopny tak pracovat. Tento parametr se da ujistit z dokumentace, anglický nazev parametru 'daisy-chaining'.

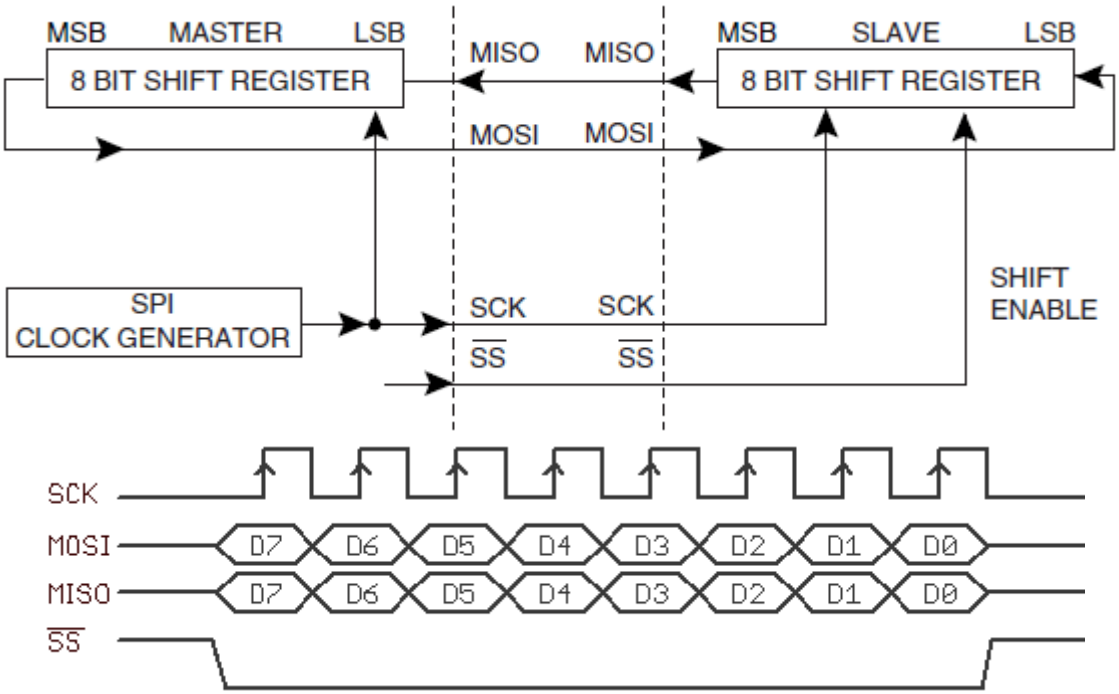


Obr. 4.5 Kaskadní zapojení pro vice podřizených.

4.4. Princip komunikace

Komunikace probihá v několika krocich.

1. Master nastavi log.0 na SS zvoleného podřizeného s kterým chce komunikovat.
2. S hodinovým siganalem se data začnou přenašet.
3. Delka vyslaných dat je 8 bit nebo 16 bit.
4. Dale jestli data jsou vyslána, Master nastavi na SS log 1 a tim komunikace ukončena.



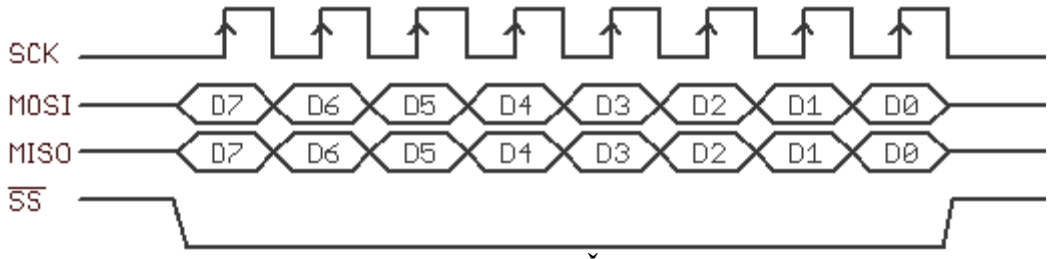
Obr. 4.6 Princip komunikace.

4.5. Řežimy SPI.

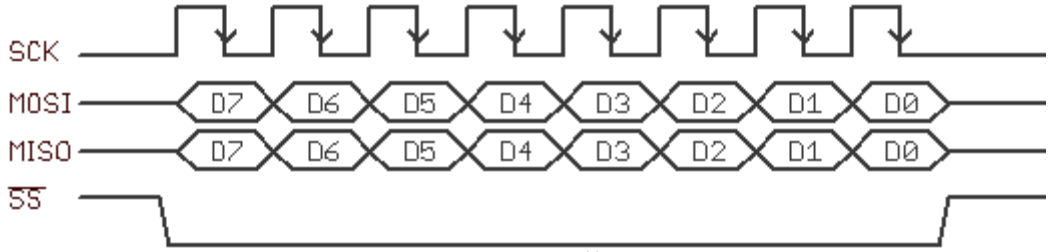
Specifikace SPI zajištuje 4 řežimy, které se liší vztahem mezi polaritou hodinového signalu (clock polarity, CPOL) a fazi synchronizaci hodinového signalů (clock phase, CPHA). Tyto konfiguračni bity jsou definovany podle konvekcii od firmy Freescale Semiconductor (původně Motorola).

Jinak řečeno, CPOL – hladina hodinového signalu před začatkem a po vysilani dat. A faze, CPHA určuje na jakou hranu proběhne čteni a zapis.

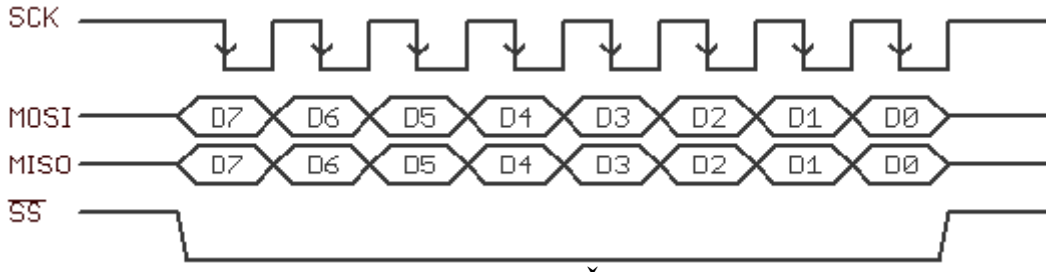
SPI mode 0: CPOL = 0, CPHA=0. Ke čteni bitu dochází na naběžnou hranu hodinového signalu (přechod 0 - 1), a k zápisu - na sestupnou hranu (přechod 1 - 0).



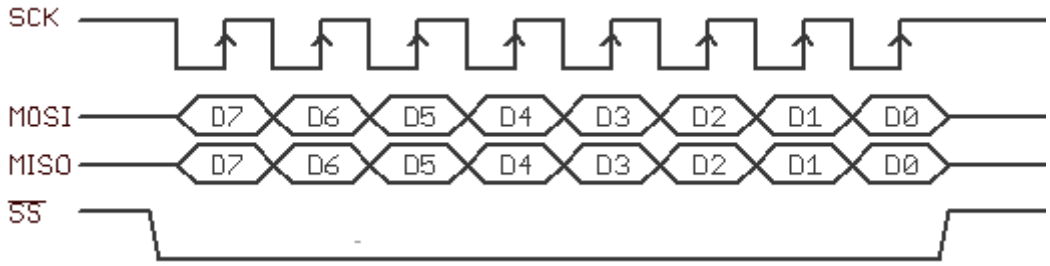
SPI mode 1: CPOL = 0, CPHA=1. Čteni – sestupná hrana, zápis - naběžná hrana.



SPI mode 2: CPOL = 1, CPHA=0. Čteni – sestupná hrana, zápis - naběžná hrana.



SPI mode 3: CPOL = 1, CPHA=1. Čteni – naběžná hrana, zápis - sestupná hrana.



Obr. 4.7 Řežimy SPI.

Řidicí a podřizené zařizeni musí pracovat se stejném řežumu, jinak jsou neslučitelné. V současné době většiná mikropočitačů májí možnost volby libovolného řežimu. Přenos dat přes SPI může probihat ne jenom vyšším bitem do předu, ale i nižším. A počet bajtů přenesených během udrženi SS není omezeno a zavisi je na specifikace podřizeného.

5. Navrh a implementace

5.1. Filtrace

Prvním krokem byla provedena filtrace vstupních signalů, aby nedohazelo k rušení a chybným hlaškam. Filtrace byla provedena nasledujicím kusem kodu.

```
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 naběžnou hranu hodinového signal bude sledovana minulá (*a_mem*) a aktualní (*a*) hodnota vstupního signalu. Když signaly budou stejné, filtrovana hodnota(*a_f*) bude aktualizovana. Na konci každého kola se aktualizuje hodnota minulého signalu.

Ukazano jen pro jeden vstupní signal (*a*), pro ostatní je udělano stejným způsobem.

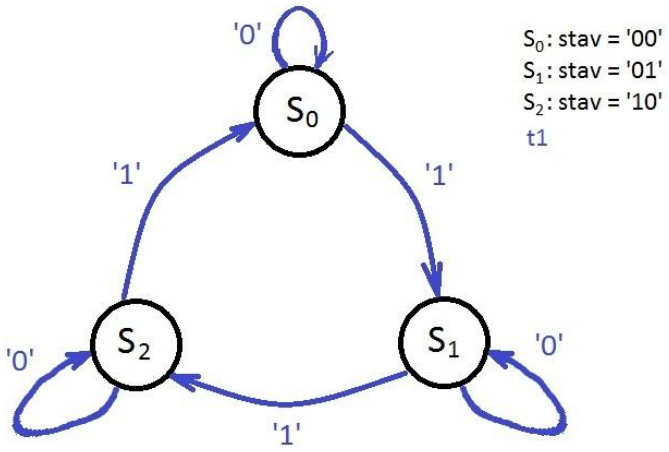
5.2. Sčitaní hran signalu.

Prvním pokusem oživovaní programu bylo počitaní hran vstupního signalu. Pro tyto učely byla realizovana detekce naběžných(*s1*) a sestupných(*s2*) hran a jejich zpracovaní. Za úkol jsem taky měl udělat tři řežimy sčitani hran: jen naběžné hrany , jen sestupné hrany a obě dvě hrany. Přepinani je realizovano pomoci multiplexeru, kde na vstup převeděny signaly detekujicí naběžné a sestupné hrany, a řidicím signalem slouží signal oznamenujicí o stisknuti tlačitka (*t1*). V zavislosti na potřebný řežim na vystup bude poslan puls nějaké hrany nebo obou dvou hran. Dale když přide puls (*pulz*), hodnota vysledku sčitaní (*sum*) se zvětší o jedničku. Vysledek sčitani se posilá ven, jen kdýž přide *N signal*, což znamena jednou za otačku. Při změně řežimu hodnota „*sum“* se vynuluje. Realizace je v nasledujicím kusu kodu.

```
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 tlačitka realizovana jako konečný automat typu Moore. Proměna *stav* stanovi aktulaní řežim sčitaní. Realizovano v nasledujicí časti 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 tlačitka.

5.3. Rychlost otačeni

V mem programu rychlost otačeni byla vyhodnocena jak metodou měření frekvence tak i metodou měření periody.

Na začatku mamé nastavit par konstantních hodnot. To je frekvence hodin (*clc_frek*), počet pulsů na jednu otačku čidla (*pno*), dobu po které bude provedeno vyhodnocovani rychlosti (*time_konst*), maximalni hodnota timeru (*tmr_max*) a pomocná konstanta (*ck)* - je vysledkém děleni frekvence hodin na *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 implementuju timer, na každou naběžnou hranu hodinového signalu pričte ke své hodnotě jedničku, po dosaženi maximalní hodnoty se vynuluje a pokračuje furt do kola. Take mám tam sčitani hran signalu jedné stopy snimače.

```
    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 probiha samotné vyhodnocovani rychlosti metodou měření frekvence v jednotkach otačky/min. Prncip je v tom, že mamé dva registru do kterých ukladam hodnoty sčitani hran. Do registru *registr_ppt1* ukladám aktualní hodnotu *sum_a* a do registru *registr_ppt2* to co jsem měl před tim v *registr_ppt1*. Rozdil těch to registru dava mě počet pulsů za předem definovaný čas *time_konst.*

Dale stači jen vynasobit převodem kolik pulsů „nateče“ během minuty a vydělit počtem pulsů na jednu otačku 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;
```

Vystupní hodnota je *ppt_SH*, ukazuje rychlost v otačkach za minutu.

Další proces počitá jak dlouho trva jedena perioda vstupního signalu, což znamená měření rychlosti otačeni metodou měření periody. Process sleduje naběžnou hranu vstupního siganlu a stejným způsobem zapisuje do dvou registrů minulou a aktuaní hodnotu, ale, na rozdil od minulého připada, zapisuje hodnoty timeru. Nasledně mám čas na začatku a konci jedné periody vstupního signalu, rozdil kterých mě dá periodu. Daším zpracovanim lze dosahnou jednoho měrítka z minulou metodou. Zde je kus kodu popisujicí ten proces.

```
    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 < 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;
```

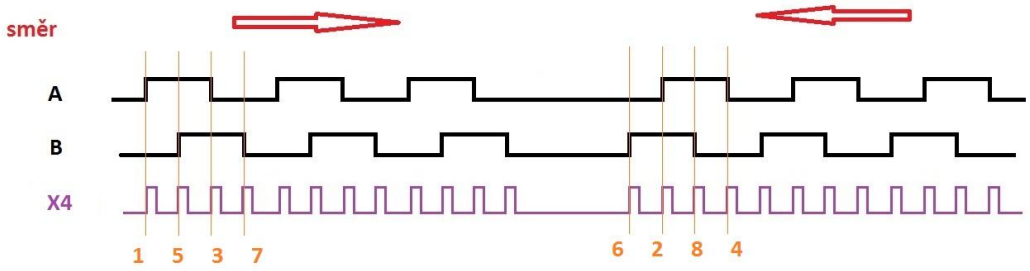
Vystupní hodnota je *kntr_cislo_SH*, ukazuje rychlost v otačkach za minutu.

5.4. Směr otačeni

Směr otačeni se určuje ze signalu detekce hran vstupních signalů, stop A i B, které jsem nadefinoval uplně na začatku programu. Směr jsem určoval s čtyřnasobnou přesnosti, což je vidět že sleduju všechny hrany sugnalů.

```
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;
```

Čisla napsané v kodu do komentaře znamenají mista ve kterých se určí směr. Podrobněji obrazek 3.2.



Obr. 3.2: Mista ve kterých se určí směr.

6. Simulace

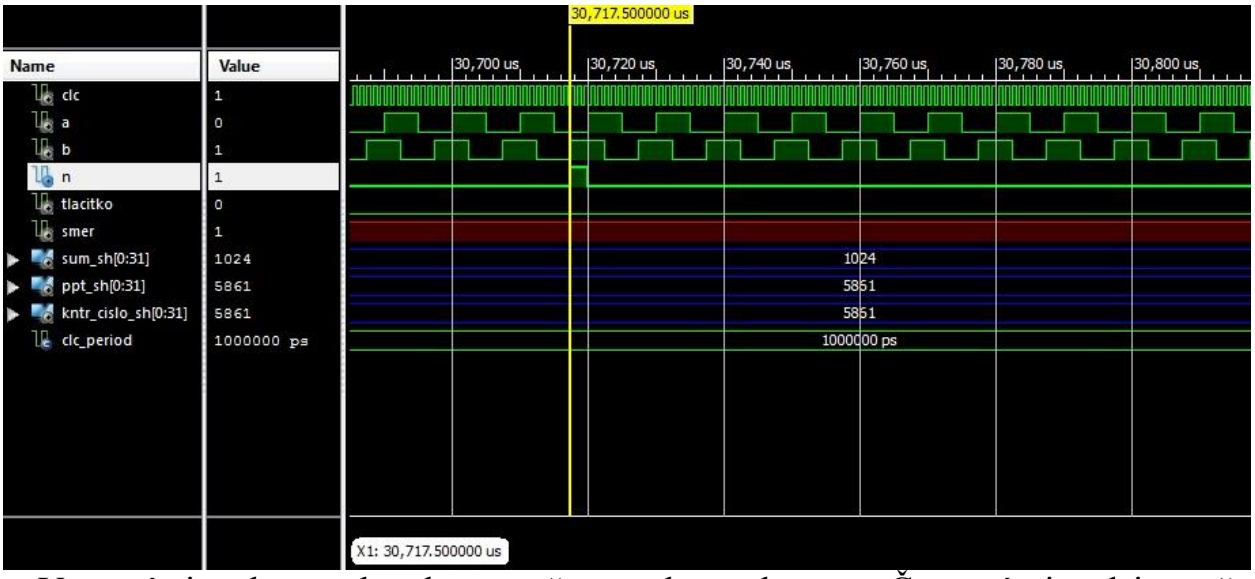
Simulace napsaného programu jsem provedl v ISimu, vestavený simulator programovacího prostředi XilinxISE. Testbench má název *test2.vhdl*. Testbench umožňuje nastavit frekvenci oscilátoru, generovani různých vstupních signalů.

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.Tady popsany jednotlivé procesy generujicí hodinový signal, simulace vystupy enkoderu, stopy A i B, N signal.

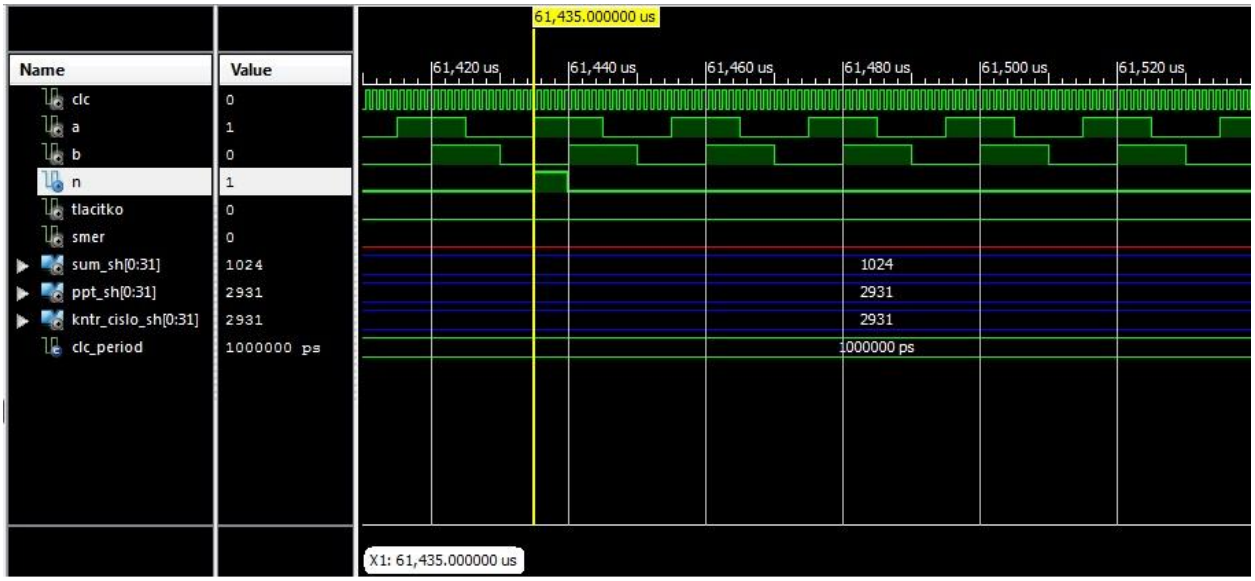
Průběhy simulace:



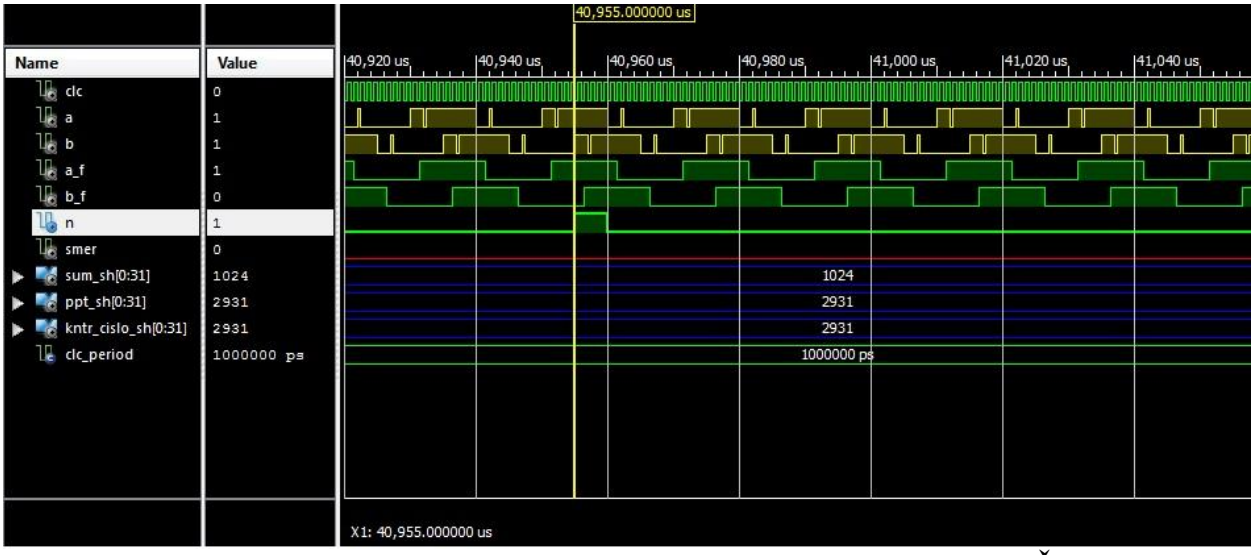
Vstupní signaly na obrazku označeny zelenou barvou. Červený signal je směr otačeni, aktualně vidime hodnotu 1, což zmanena otačení proti hodinových ručiček. Z signalu stop A i B vidime, že B signal předbihá, což taky znamena směr proti hodinových ručiček.Tim padem můžeme řict, že program funguje jak má.

Dale mame modré vystupní signaly, signal sum_SH v posleni verze programu sčitá počet pulzu mezí N signaly, což vyjadřuje počet pulzu na otačku, jeden z vyznamných parametru enkoderu. Nastavenou hodnotu mam taky 1024 pulsů na otačku. Dale jsou signaly se stejnou hodnotou, ale z různých zdrojů. *ppt_SH* je vysledek určovaní rychlosti otaček metodou měření frekvence a *kntr_cislo_SH* – metodou měření periody.

Vystupní hodnoty zcela odpovidají očekovaným a program počita spravně.



Tady nasimulovana jiná situace, směr je ve směru hodinových ručiček, to se da ověřit stejným způsobem jak v minulem přikladu. *sum_SH* ukazuje stejný počet pulsů na otačku a změnila se rychlost, jak vidime na obrazku, ale hodnoty ppt_SH a *kntr_cislo_SH* jsou stejné.



V další simulaci byla ověřena funkce filtrace vstupních signalů. Žluté průběhy jsou vstupní signaly, umyslně zašumlené. A signaly *a_f* a *b_f* jsou vystupní signaly filtrace. Je vidět že process proběhl v pořadku a šum neovlivníl na vystupní hodnoty programu (pro porovnaní parametry jsou stejné jak i u minulé simulace).

7. Zavěr

Cilem teto praci bylo navhnou program pro vyhidnocovani otačkových čidel. Program byl realizovan v jazyce VHDL v programovacím prostředi XilinxISE. Vysledkem vyhodnocovaní otačkových čidel jsou hodnoty rychlosti otačeni v ot/min a směr otačeni. Rycholost byla odečtena dvěma metodami a jak vidět z simulace jsou ve stejné. Ale to je jen pro simulace v idialních podminkach, ve skutečnosti že měření periody by bylo lepe použit pro nizké frekvence vstupního signalu a měřeni frekvence – pro větší frekvence vstupního signalu. Kvůli tomu, že vystupem je hodnota v otačkach za minutu, to vyžaduje velký počet nasobiček a děliček a to zatěžije program a potřeba vic času na vypočet hodnot. Určeni směru bylo realizovano s čtyřnasobnou přesnosti a pro vystupní signal nam stačil jen jeden bit, kde log.0 znamena směr ve směru hodinových ručiček a log.1 směr proti hodinových ručiček.

Celkem program je fukční a po připojení k modulu umožnujicí komunikace po SPI sběrnice a nahrani na nejakou vyvojovou desku muže byt požit v životě.

8. Seznam obrazků

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 snimačů otaček, diplomová práce, Brno, 2009 [online]. Dostupné z WWW: <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 snimače, přednaška, [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>

[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 kod programu vyhodnocovaní otaček

```
----------------------------------------------------------------------------------
-- 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_frek : 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_frek / 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;
				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;
    -- 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 <= 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 < 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;
```