Okruhy z SPS

Karel Horák

December 18, 2011

1 Karnaughovy mapy

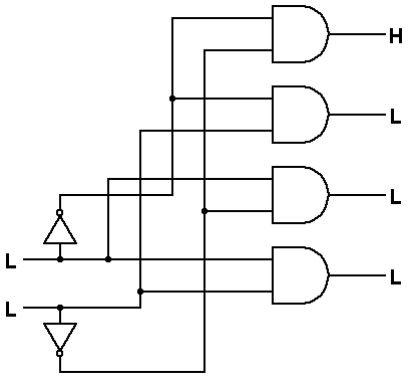
> Bezpečná znalost tvorby KM až pro 8 proměnných, vytvoření KM z logické funkce a logické funkce z KM pomocí minimalizačních postupů S-o-P (sum-of-products) a P-o-S (products-of-sums). Čekejte, že u mnohých příkladů se bude výsledek psát jako KM, jelikož se v ní lépe kontroluje správnost řešení

2 Kombinační logické obvody a Booleova algebra

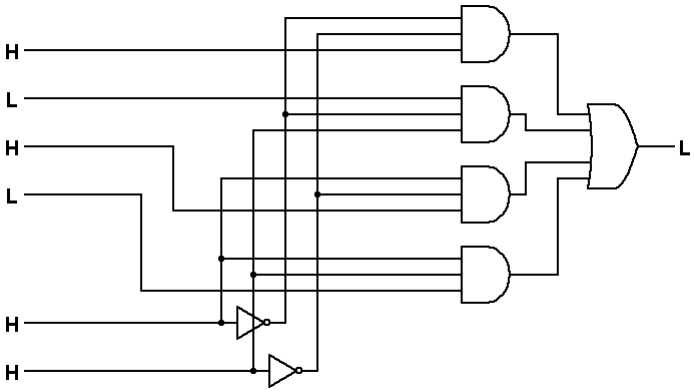
Umět navrhnout obvod provádějící zadanou funkci, a naopak schopnost rozumět logickému schématu, což znamená samozřejmě bezpečnost znalost funkce základních hradel AND, OR, XOR, NOR a NAND. Otázka se bude zkoušet nejen na příkladech "Navrhněte zapojení,", ale také na zadáních typu "Jaký bude výstup zadaného obvodu, pokud jeho vstupy mají tyto hodnoty?". Všichni by navíc měli umět bez rozmýšlení nakreslit zapojení pro

- dekodér 1 z N
- multiplexor M:1
- demultiplexor 1:M
- generátor liché nebo sudé parity s XOR obvody

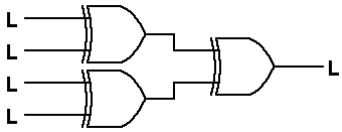
2.1 Dekodér 1 z 4



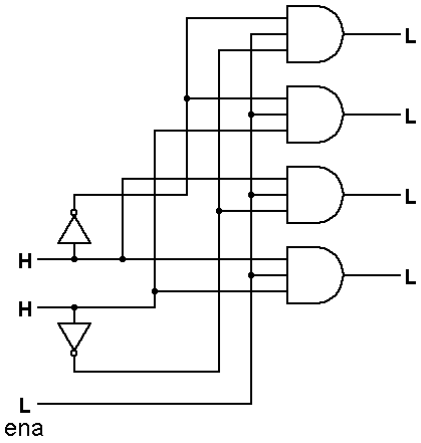
2.2 Multiplexor 4:1



2.3 Generátor sudé parity pro 4 bity



2.4 Demultiplexor 1:4



3 Statické hazardy v kombinačních obvodech a jejich odstranění

Definice, praktické použití Karnaughových map pro jejich eliminaci

Statické hazardy jsou dvou typů:

- *Static-1* hazardy - nastávají pokud se vstupy X_1, X_2 liší pouze v jedné proměnné a jejich výstupem má být konstantní jednička.
- *Static-0* hazardy - obdobně pro nulu.

Jejich lokalizace je jednoduchá. Pokud v Karnaughově mapě jsou dvě sousední pole jsou v 1, ale tyto dvě 1 nejsou pokryty stejným mintermem. A odstranění je taky jednoduché - stačí chybějící minterm přidat.

4 Shannonova dekompozice logické funkce

Její znění a praktické uplatnění pro rozklad funkce třeba příkladech typu zapojit logickou funkci jen pomocí multiplexorů.

Je dobrá k tomu, abychom mohli zapojit složitou funkci pomocí multiplexorů. Myšlenka je následující:

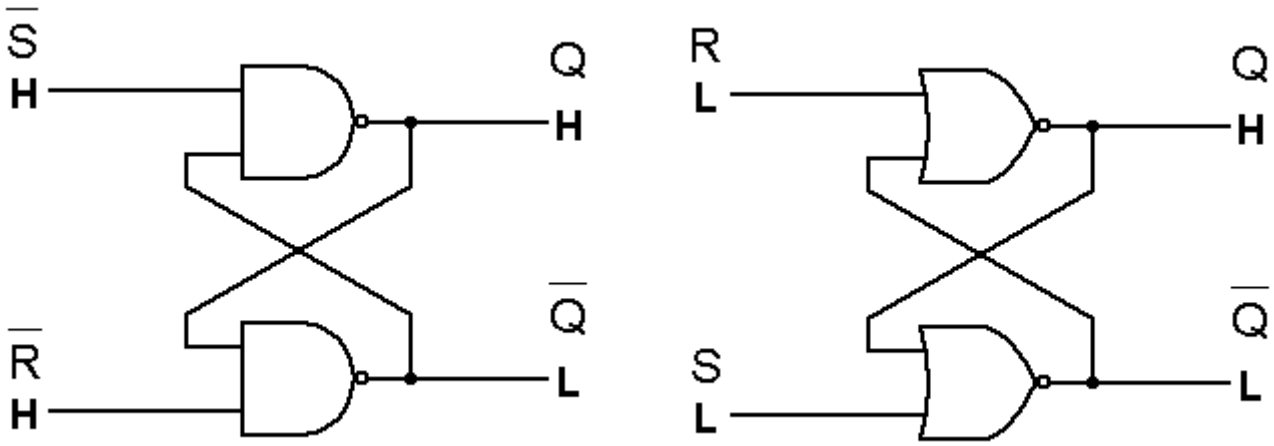
$$\tilde{f}_a = af(1, b, c, \ldots) + \bar{a}f(0, b, c, \ldots)$$

5 Základní sekvenční logické obvody

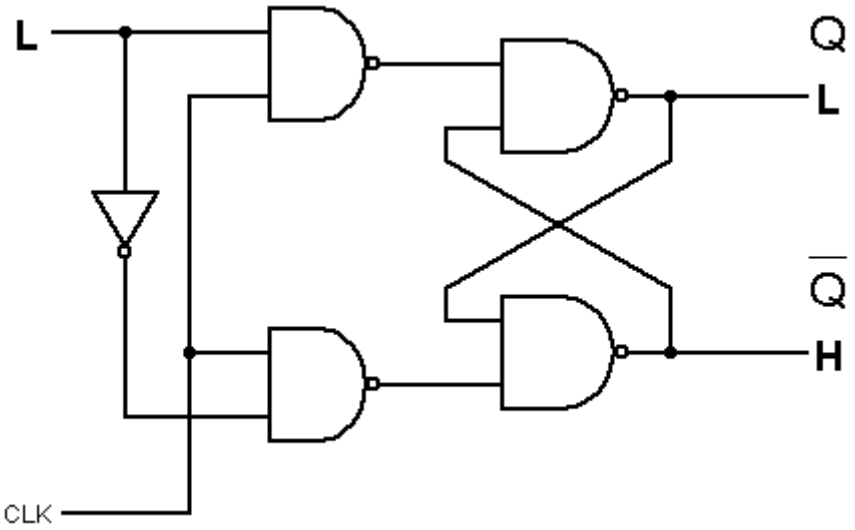
- asynchronní RS a norRS obvod zapojený jak z hradel NAND, tak i z hradel NOR!
- D-latch (7475) pomocí hradel NAND;
- synchronní D master-slave (respektive JK master-slave) pomocí hradel NAND;
- bezpečná znalost činnosti základních synchronních D klopných obvodů v Quartusu dff, dffe a dffe a JK klopných obovdů v Quartusu jkff a jkffe, a to včetně funkce asynchronních vstupů Pre a Clr.
- schopnost navrhnout asynchronní dělič zadaným číslem používající synchronní obvody JK a D.
- znát posuvný registr jednosměrný, obousměrný a s paralelním vstupem
- umět zpětnovazební posuvný registr pro zadaný polynom (bude ještě na přednášce)

Otázka se bude zkoušet nejen na příkladech "Navrhněte zapojení,", ale také na zadáních typu "Jaký bude výstup zadaného obvodu, pokud jeho vstupy mají hodnoty...".

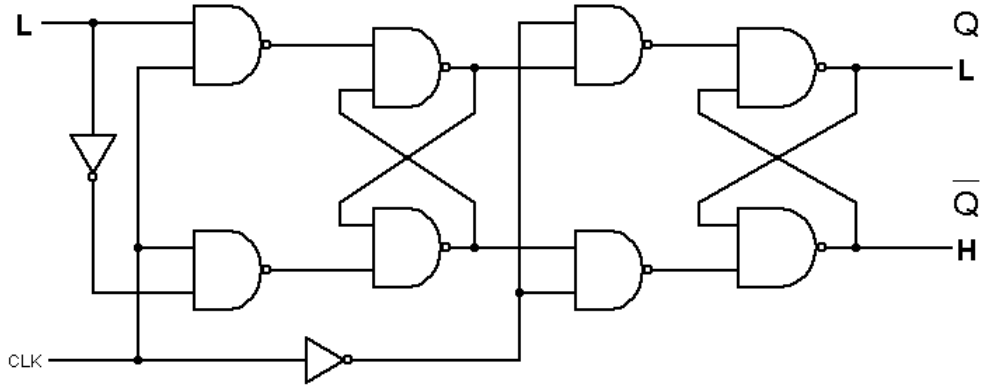
5.1 RS-latch



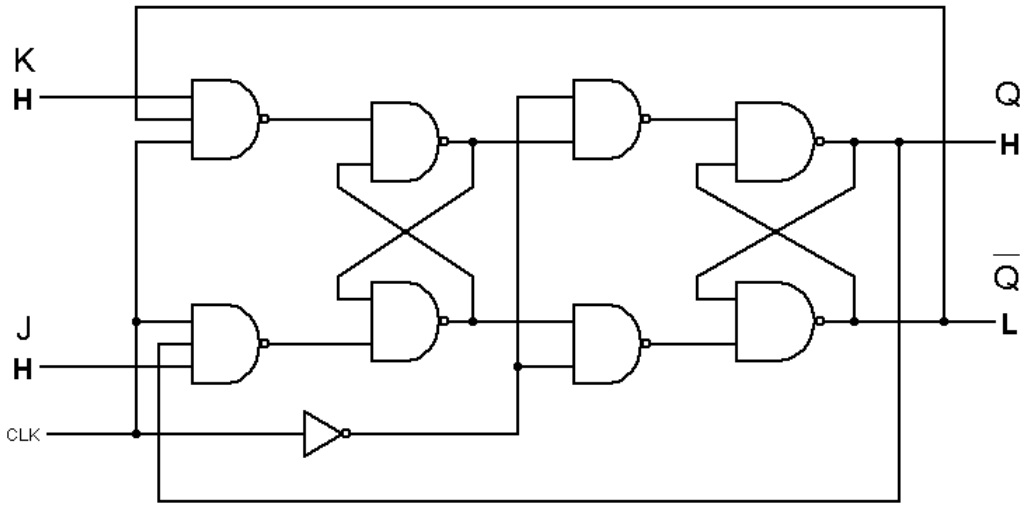
5.2 D-latch



5.3 D master-slave



5.4 JK master-slave



6 Automat Moore a Mealy

!!! Definice, návrh automatu ze slovního zadání jako přechodový diagram a tabulku, praktické použití automatu , tedy ze zadaného příkladu sestavit přechodovou tabulku automatu a umět návrh synchronního čítače z obvodů D nebo JK mající na výstupech zadanou posloupnost

6.1 Mooreův automat

Mooreův automat je šestice $(Q, \Sigma, Y, \delta, q_0, \beta)$. β je značkovací funkce $\beta : Q \to Y$.

6.2 Mealyho automat

Mealyho automat se od Mooreova liší tím, že výstupní funkce $\lambda : (Q \times \Sigma) \to Y$.

6.3 Návrh čítače majícího na výstupu zadanou posloupnost

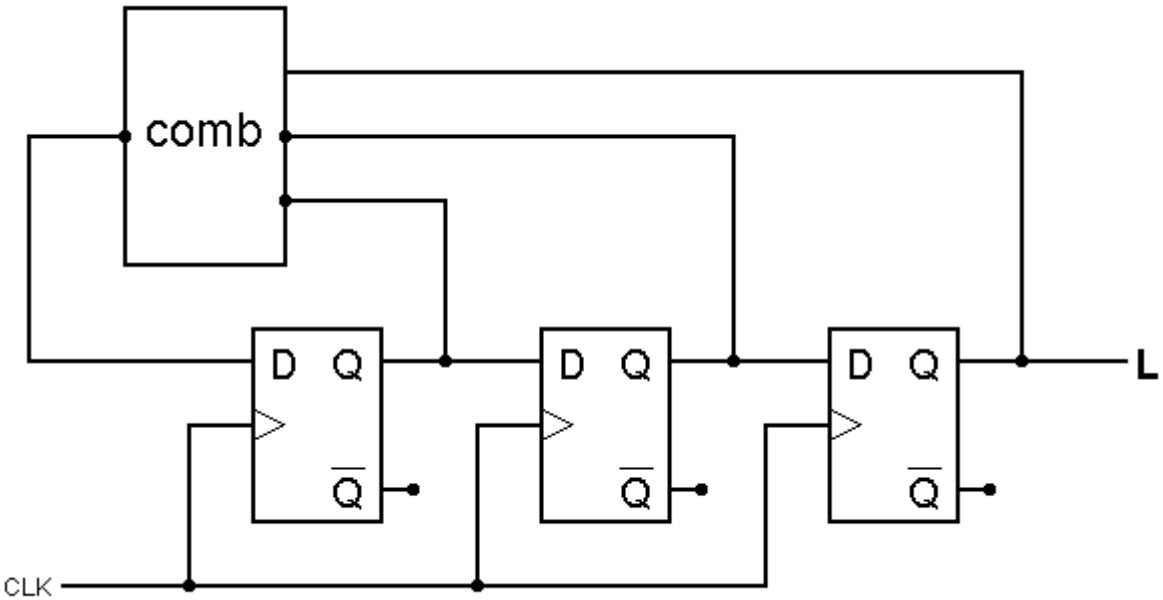
Jsou v zásadě dvě možnosti návrhu:

- Každý prvek posloupnosti reprezentujeme jako jeden člen posuvného registru (to je ale zbytečně náročné)
- Pokusíme se redukovat počet paměťových prvků na minimum

První způsob je jednoduchý, druhý je ale efektivnější. Pokud máme posloupnost periody n, pak budeme potřebovat alespoň $k = log_2 n$ paměťových prvků.

Nyní se podíváme na prvky posloupnosti přes posuvné okénko délky k. Sestavíme si cyklický orientovaný graf z prvků, které přes toto okénko vidíme. Pokud se v takto vzniklém cyklu nějaké prvky opakují, znamenalo by to, že nemůžeme deterministicky rozhodnout, jaké budou následující prvky. Proto bychom položili $k = k + 1$ a postup opakovali.

Až nalezneme vhodné k můžeme přistoupit ke kódování kombinační logiky pro nahrávání do prvního prvku posuvného registru. Každé dva sousední stavy se liší tím, že vznikly binárním posunem prvního a doplněním vhodného bitu. Vytvoříme logickou funkci, která nám tento doplněný bit bude reprezentovat. Funkci zminimalizujeme a můžeme přistoupit k realizaci obvodu.



7 Hazardy a metastabilita v synchronních obvodech

> Vysvětlení pojmu, metody jejich odstranění, zapojení synchronizátoru, **fázový závěs** - jeho princip, hlavní zásady pro správné RTL návrhy. Otázka se bude zkoušet nejen vyjmenováním, ale hlavně na příkladech jako třeba - najděte možný hazard nebo odstraňte synchronní hazard

Dynamický hazard je několik změn na výstupu sekvenčního obvodu v důsledku pouhé jedné změny.

Hazardy v synchronních obvodech nastávají tehdy, pokud dojde k porušení časování flip-flopů. Před každou hodinovou hranou, na kterou flip-flop reaguje, musí být datový vstup konstantní po dobu *setup-time* (t_{SU}). Po této hraně musí být datový vstup konstantní po dobu *hold-time* (t_H).

Pokud tyto požadavky porušíme může dojít k náhodné změně stavu, nebo ke stavu, který se nazývá **metastability**. V případě, že se flip-flop nachází v metastabilním stavu, tak je jeho neplatný výstup mezi nulou a jedničkou. V takovém případě se flip-flop vrátí do platného stavu po určité době, která zhruba odpovídá hodnotě *resolution time*, kterou lze empiricky změřit.

7.1 Zapojení synchronizátoru

Synchronizátor může být například dvojbuňkový posuvný registr.

7.2 Zásady RTL návrhu

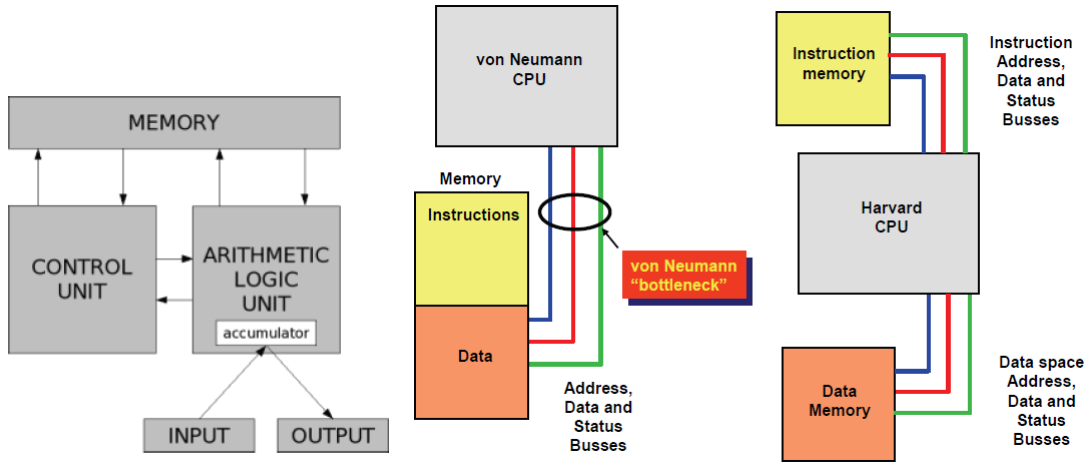
- Všechny výstupy kombinačních obodů by měly být registrovány
- Proto tedy používáme synchronní registry a ne *level-triggered* latche
- Všechny vstupy jsou synchronní
- *Fast paths* - je potřeba je eliminovat kvůli možnému porušení *hold-time*
- Snažíme se minimalizovat zpoždění hodin přes celý obvod
- Více hodinových domén jenom s nejvyšší opatrností (synchronizátory)

8 Architektura počítače a jeho činnost procesoru

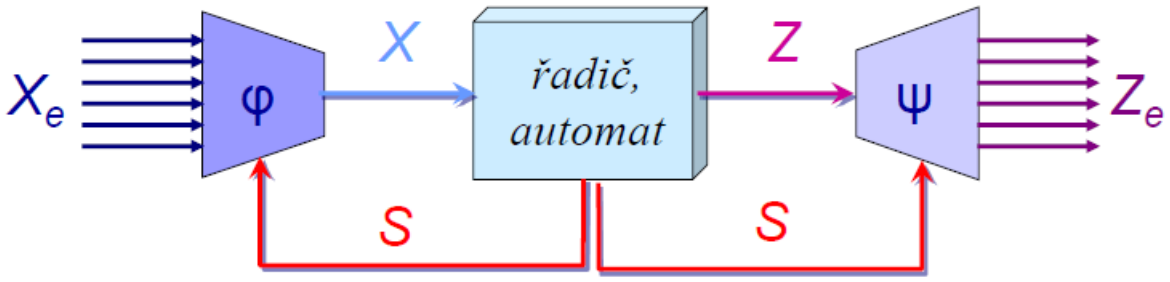
> Znát obrázek pro architekturu von Neumana a Harvard a umět nakreslit obecný model, **umět sestavit řadič**, znát schéma obecné řídicí jednotky, pojem datové cesty, RISC, CISC, fáze zpracování instrukce, pipelining, predikce

8.1 Architektura von Neumann a Harvard

Architektura Harvard má na rozdíl od von Neumannovy architektury liší tím, že má oddělenou instrukční a datovou paměť. Tento přístup sám o sobě nezaručuje vyšší výkon systému.



8.2 Schéma obecné řídící jednotky



8.3 Datová cesta

Datová cesta jsou propojení a komponenty řízené řadičem.

8.4 RISC a CISC

Architektura CISC implementuje velkou sadu instrukcí pro práci s různými datovými typy atd. Takový přístup se může zdát lepší, ale není tomu vždy tak, protože instrukce jsou daleko komplexnější. Naproti tomu procesory s instrukční sadou RISC implementují pouze základní aritmeticko-logické operace a operace přesunů. Jejich instrukce jsou jednotikové.

8.5 Pipelining

Pipelining je metoda časování procesoru. Instrukční cyklus skládá z několika cyklů strojových, například:

- čtení
- dekódování
- čtení operandů
- operace
- zápis

Pomocí *pipeliningu* můžeme v průběhu dekódování instrukce již načítat instrukci další a tak urychlit zpracování instrukcí, které by se jinak začaly zpracovávat až poté, co by předcházející instrukce byla plně dokončena.

Nicméně *pipelining* přináší také některá úskalí, pokud instrukce přímo závisí na instrukci předcházející. Nejčastějším typem závislosti je závislost na datech (Předchozí instrukce modifikovala data, která se současná snaží načíst. Tyto data však ještě nemusí být zapsaná) Dále mohlo v předchozí instrukci dojít ke skoku, a další instrukce se tak vůbec neměla zpracovat. S oběma problémy se však lze vypořádat.

8.6 Predikce

9 Základní obvody ve VHDL

Znát VHDL kódy pro základní logické a kombinační obvody, rozumět "concurrent" a "sequential" příkazům, rozdílům mezi synchronními a asynchronními vstupy obvodů - otázka se bude zkoušet na základě příkladů typu:

- doplnění prototypu VHDL kódu o některé řádky,
- nakreslení logického schématu, který odpovídá VHDL kódu, zde doporučuji zejména pečlivě prostudovat přednášku 8.
- opravení vážné chyby ve VHDL kódu.

10 Assembler

> Znát adresní módy NIOSu, umět stanovit výsledek kódu programu NIOSu, nebo naopak napsat jednoduchou operaci v asembleru, např. pro cyklus for, while, switch, if-then-else. Určitě byste měli byste znát chování základních instrukcí LD, ST, MOV, aritmetiky, logických instrukcí s registrovými a přímými operandy, skoky (jump) a větvení (branch). Pozn. Symbolické kódy instrukcí budou součástí nápovědy k zadání

10.1 Adresní módy NIOSu

- Register direct addressing: `mov rA, rB`
- Immediate addressing (operand obsahuje hodnotu): `movi rA, 0x10`
- Displacement addressing (výsledná adresa je dána součtem relativní adresy `addr` a bázového registru `rB`): `ldw rA, addr(rB)`

10.2 Smyčka v NIOSu

Následující kódy jsou ekvivalentní:

```
movi r6,10
movi r7,1
movi r8,1
stwio r7,0x810(r0)
LOOP:
sub r6,r6,r8
xor r7,r7,r8
stwio r7,0x810(r0)
bgt r6,r0,LOOP
```

```
int main(void) {
        register int i = 10;
        register int led = 1;
        do {
                i--;
                led = led ^ 1;
                *((int*)(0x810)) = led;
        } while(i > 0);
}
```

10.3 Branch × Jump

Instrukce větvení používají relativní adresu oproti adrese následující instrukce. Instrukce skoku naproti tomu používají absolutní 26-bitovou adresu vynásobenou čtyřmi (instrukce NIOSu jsou zarovnané na 4 byty). Pomocí `jmpi` lze tedy skočit na libovolné místo v paměti adresovatelné program-counterem (tj. 256MB).

11 Zobrazení celých čísel v počítači

> Umět převést kladné i záporné dekadické číslo na bitový řetězec, znát zápis v hexadecimálním kódu a jeho převody na bitové řetězce a dekadická čísla, rozumět znaménkovému rozšíření (se) u NIOS instrukcí LDw, LDhu, LDh, LDb, LDbu, a relativním skokům, umět sečíst dvě čísla v aritmetice zadané délky

11.1 Reprezentace celých čísel

Uvažuji k-bitovou reprezentaci celého čísla n:

- Pokud $n > 0$, pak číslo n reprezentujeme jako n, pokud se vejde do datového typu (v opačném případě přetečení, což je ale jasný)
- Pokud $n < 0$, pak číslo n reprezentujeme jako $2^k - n$ (příp. zase podtečení)

Této reprezentaci čísel se říká dvojkový doplněk. Její výhodou je, že se při sčítání a odčítání nemusíme zabývat znaménkem.
Jsou dva přístupy k ukládání čísel do paměti:

- Big-endian: reprezentace začíná MSB (tj. Big mocnina je vepředu)
- Little-endian: reprezentace začíná LSB

11.2 Instrukce NIOSu

Instrukce NIOSu pro načítání celých čísel z paměti jsou ve formátu `ldX`, kde `X` je datový typ, se kterým chceme pracovat, tj:

- `b` - pro signed char
- `bu` - pro unsigned char
- `h` - pro signed short
- `hu` - pro unsigned short
- `w` - pro signed int
- `wu` - pro unsigned int

Všechny tyto metody mají jednotnou syntaxi `ldX rB, byte_offset(rA)`, kde `rB` je registr, kam chceme hodnotu načíst, a adresa, ze které chceme číst, je `byte_offset + rA`.
Analogicky k těmto metodám jsou k dispozici metody `stX rB, byte_offset(rA)` sloužící pro ukládání dat do paměti.
Existují i necachované varianty těchto instrukcí `ldXio` a `stXio`.

12 Typy přerušení na procesoru

Rozdíl ve zpracování nemaskovaného a maskovaného přerušení, **vektorové zpracování výjimek**

Rozlišujeme dva základní typy přerušení:

12.1 Maskovaná × Nemaskovaná přerušení

- Maskovaná přerušení - pro umožnění jejich zpracování musí být nastavený příslušný bit
- Nemaskovaná přerušení - nelze je deaktivovat

13 Sběrnice počítačů

> Jejich typy, obvody s otevřeným kolektorem a třístavové výstupy,asynchronní a synchronní komunikace po sběrnici,DMA přenosy, funkci arbiteru sběrnice, master a slave bridge mezi sběrnicemi - může se zkoušet nejen pomocí vyjmenování, ale také **nakreslením průběhů signálů na sběrnici** daného typu či **obvodu pro připojení ke sběrnici**

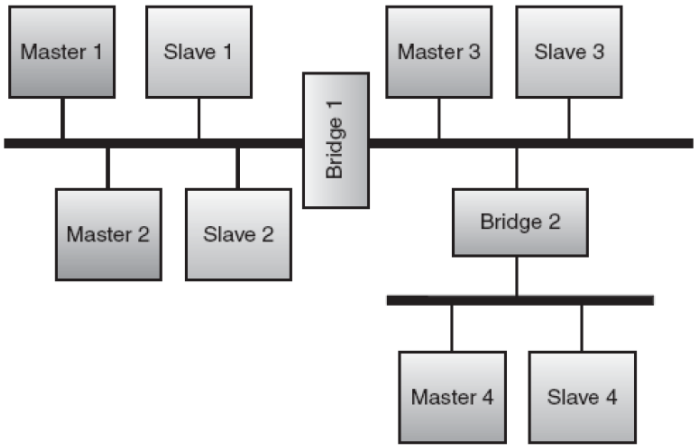
Sběrnice je sdílená datová cesta (*datapath*). Její konstrukce je poměrně jednoduchá, ale naráží na problémy se škálovatelností. Typicky na sběrnici nalezneme tři druhy linek - datové, adresní a řídící linky.

Komponenty na sběrnici jsou připojeny na sběrnici buďto v módu **master**, kdy daná komponenta může vyžádat komunikaci po sběrnici, **slave**, kdy daná komponenta komunikuje po sběrnici pouze na vyžádání. Některé komponenty jako například DMA řadič pracují v hybridním **master-slave** režimu.

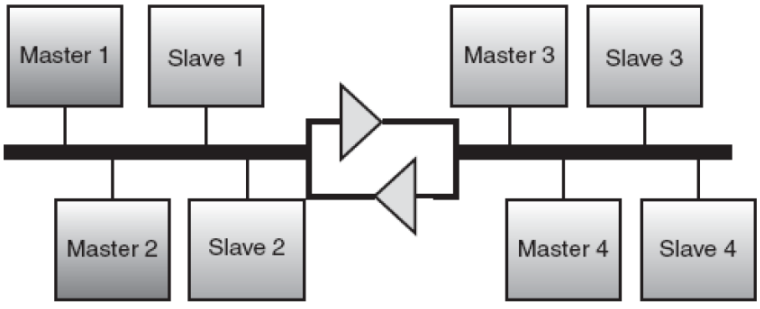
Provoz na sběrnici je řízen **arbiterem**. Jedná se o obvod, který zajišťuje pouze jednu probíhající komunikaci po sběrnici. Master komponenty posílají požadavky na komunikaci, arbiter je zpracuje a v případě uvolnění komunikačního kanálu předá řízení některé z komponent.

13.1 Typy sběrnic

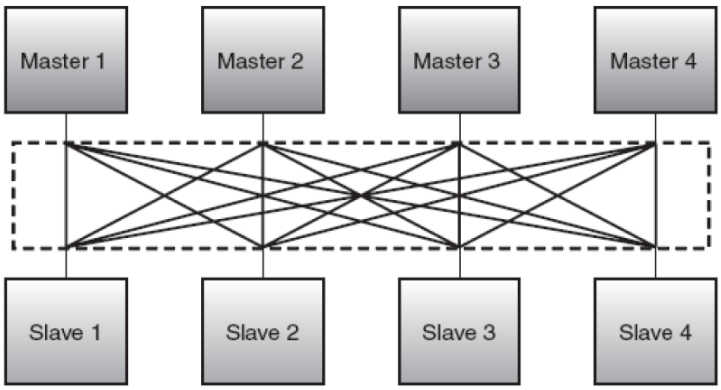
V systému může být jedna jediná sdílená sběrnice (**shared bus**). Kvůli vyšší propustnosti se však sběrnice často strukturují na dílčí sběrnice (**hierarchical shared bus**), na kterých paralelně probíhají přenosy. Tím lze dosáhnout vyšší propustnosti. Sběrnice jsou pak propojený můstky (**bridge**).



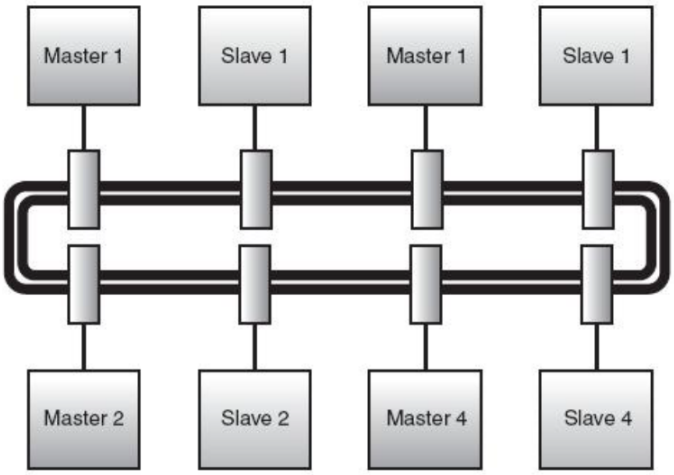
Dlouhé sběrnice se často rozdělují na jednotlivé segmenty. Vznikají tak tzv. **split bus**, jejichž myšlenkou je snížení odporu na dlouhých vodičích.



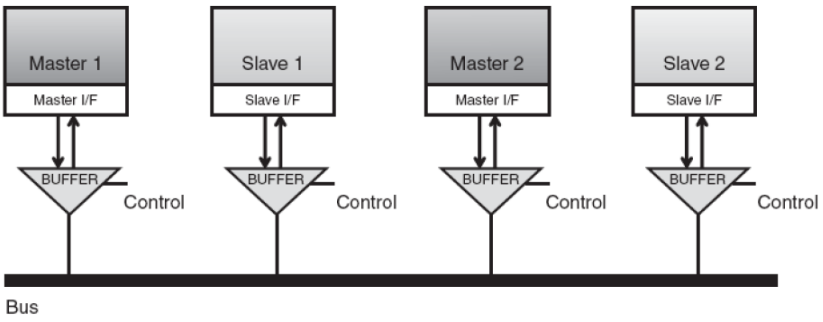
Je také možné propojit každou komponentu s každou, čímž dostaneme **point-to-point** architekturu. Takový typ sběrnice je ale velice komplikovaný.



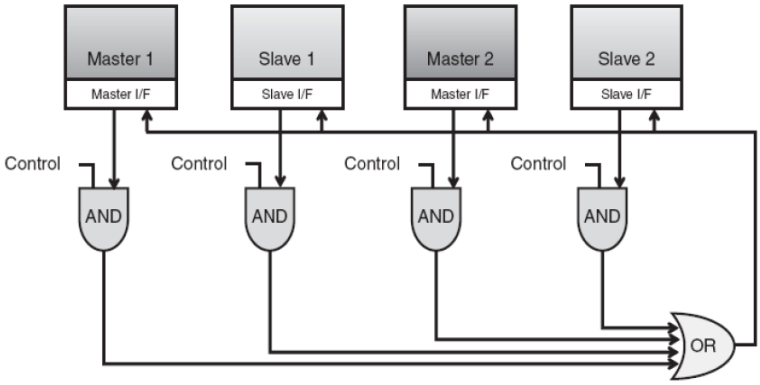
Jako u sítí je také možné použít prstencovou architekturu (**ring bus**).



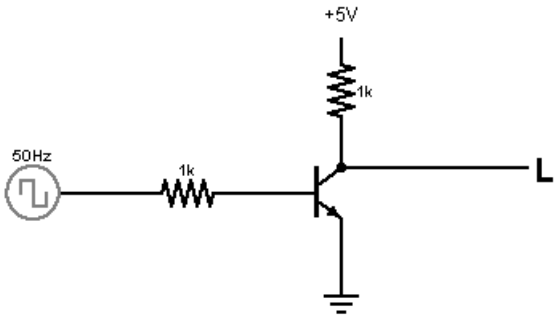
Tristate buffer sběrnice



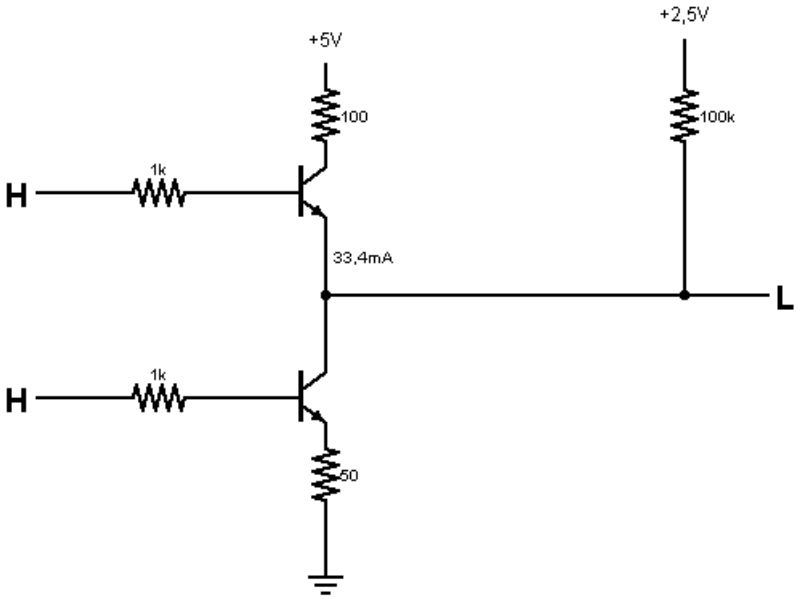
Sběrnice může být také založena na logických hradlech **and-or**. Výstupy jednotlivých komponent jsou maskovány hradlem and, hradlo or výstupy vybere a vrátí je na vstupy všech komponent.



13.2 Obvody s otevřeným kolektorem



13.3 Třístavové výstupy

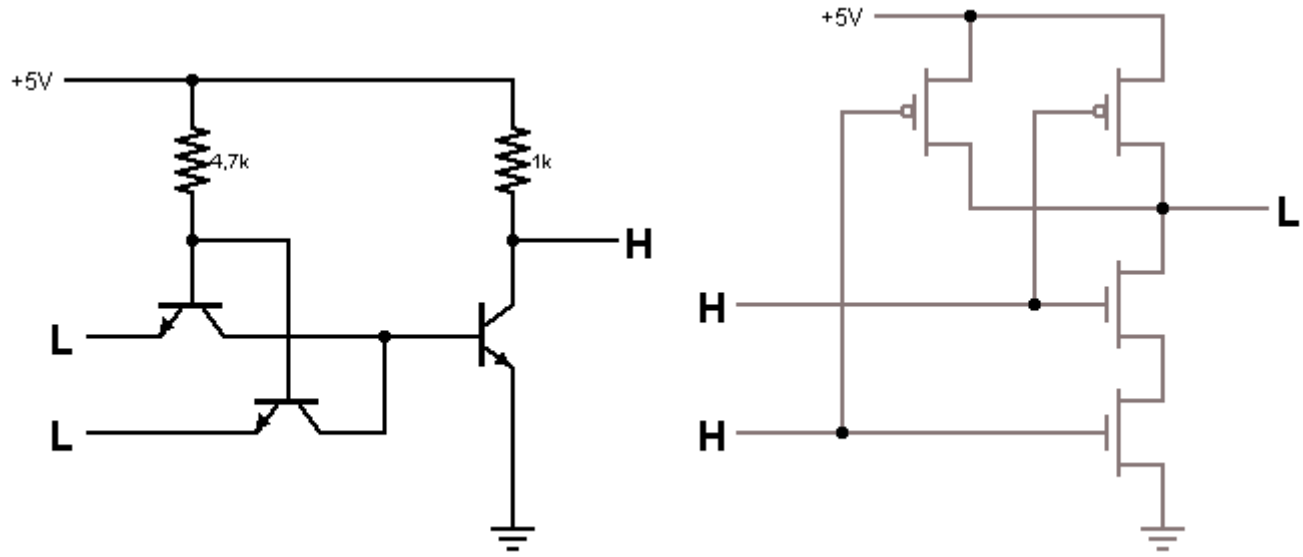


14 Interfacing

Struktura hradla NAND v provedeni TTL a CMOS [vyžaduje se nakreslení schématu], **ošetření nezapojených vstupů** (floating inputs), aspoň 2 metody pro připojení přepínače a jeho odrušení (debouncing switch), připojení LED diody k výstupu hradla CMOS, připojení indukční zátěže k hradlům CMOS, připojení vedení a jeho chování, metody pro oddělení obvodů – principielní schéma, PS/2 komunikace a RS232 úrovně signálů a struktura signálu

14.1 Hradlo NAND v provedení TTL a CMOS

Nalevo je zapojení hradla v provedení TTL, napravo v provedení CMOS.

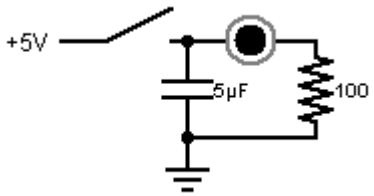


14.2 Ošetření nezapojených vstupů

Nezapojené vstupy ošetříme tak, že je buď připojíme přímo na zem (v případě, že potřebujeme přípojit logickou nulu), nebo na zem přes invertor (v případě, že potřebujeme logickou jedničku).

14.3 Metody připojení přepínače a jeho odrušení

- Nejjednodušší možností připojení switche je použití kondenzátoru. Kondenzátor se bude určitou dobu vybíjet a nabíjet, než dojde ke změně výstupu. Mezi tím se signál ustálí.

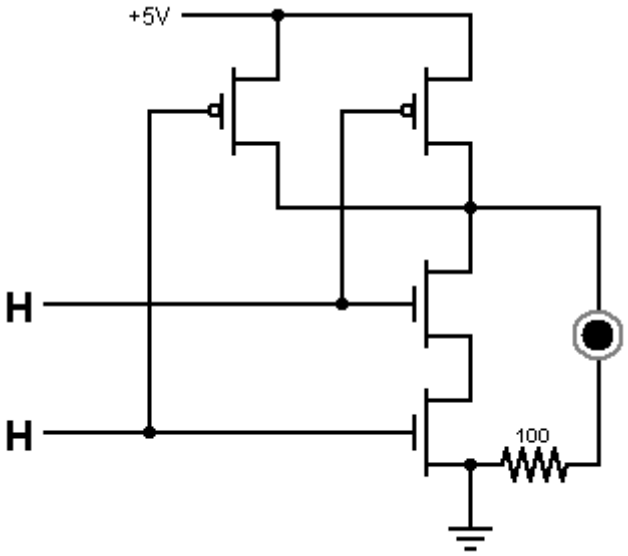


- Další možností je použití vzorkování. Použijeme posuvný registr, kdy čekáme na dva tiky s vstupem v jedničce, před kterými byla nula.

14.4 Připojení vedení

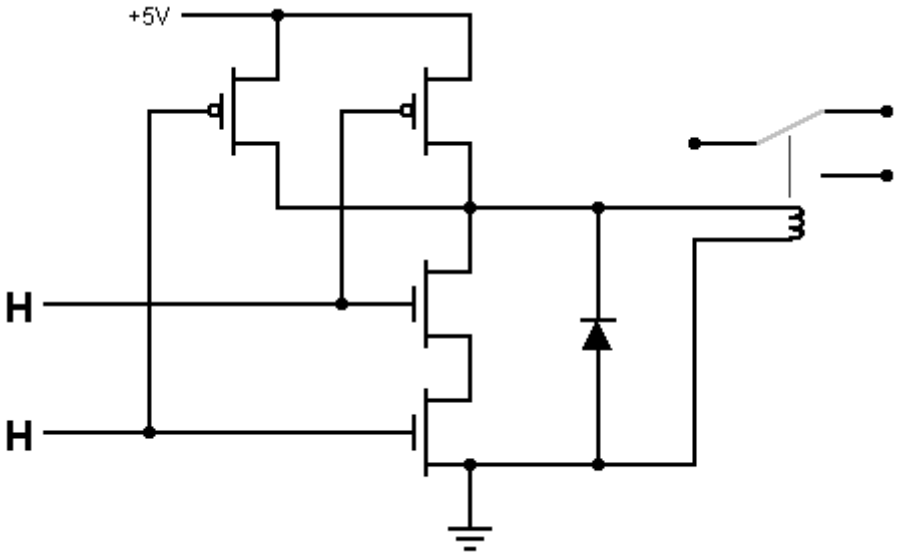
Vedení je vhodné odrušit pomocí kondenztoru, který přípojíme na oba konce linky.

14.5 Připojení LED diody k hradlu CMOS



14.6 Připojení indukční zátěže k hradlu CMOS

Musíme ošetřit zpětný proud, který by mohl poškodit hradlo. Proto použijeme diodu na vybití.



14.7 Metody pro oddělení obvodů

- Jednou z možností je použití optočlenu. Optočleny jsou součástky tvořené diodou emitující světlo a tranzistorem reagujícím na světlo.
- V případě, že jde o datovou linku mezi dvěma obvody s jinou hodinovou frekvencí, je potřeba použít synchronizátory.

14.8 RS232 komunikace

Při asynchronní komunikaci přes RS232 jsou nejdůležitější piny TX, RX a GND. TX je pin používaný na vysílání, RX na přijímání, GND je společná zem.

Komunikace přes RS232 probíhá v rámcích. Obě strany komunikace musí používat stejnou velikost rámce. Rámec začíná tzv. *start-bitem*, poté co se pošlou data rámce následuje *paritní bit*. Celý rámec je ukončen tzv. *stop-bitem*.

15 ASIC obvody