Návrh číslicových obvodů

Jan Kořenek, Tomáš Martínek

Brno University of Technology, Faculty of Information Technology
Božetěchova 2, 612 66 Brno
korenek@fit.vutbr.cz

Osnova

- ***Proč mapovat algoritmus do hardware***
- Mapování základních konstrukcí do hardware
  - Sekvence
  - Selekce
  - Iterace
- Modulární návrh
- Shrnutí

Realizace algoritmu

Algoritmus

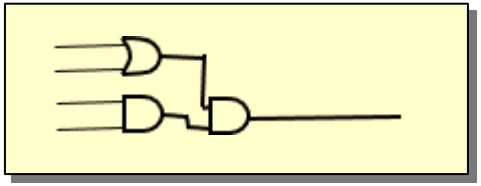
- Hledání řetězců
- Šifrování dat
- Filtrace obrazu
- …

Implementace pomocí programu


```
void main(){
    ...
    return 0;
}
```

- Řízení programem
- Sekvenční zpracování programu na jednom nebo více procesorových jader

Implementace pomocí hardwarové architektury



- Chování dáno zapojením obvodových prvků, které pracují paralelně
- Základní obvodové prvky se liší podle použité technologie – ASIC, FPGA, …

Sečtení prvků posloupnosti: Software vs Hardware

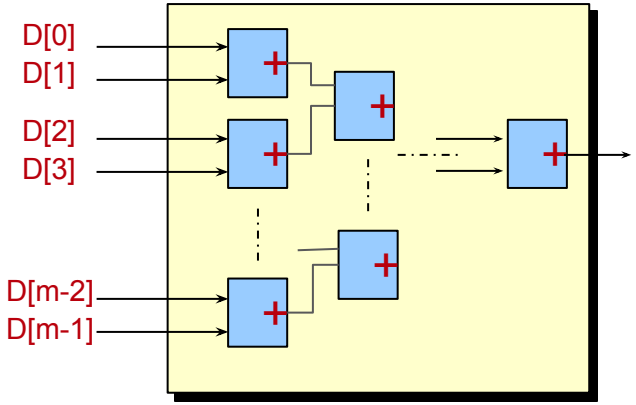
Implementace pomocí SW algoritmu

```
sum=0;
for(i=0;i<m;i++)
    sum = sum + D[i];
```

- Procesorové jádro obsahuje v ALU jednu sčítačku. V jednom kroku je proveden jeden součet
- Celý výpočet bude dokončen v **m** krocích

Kolik procent plochy procesoru je využito při výpočtu?

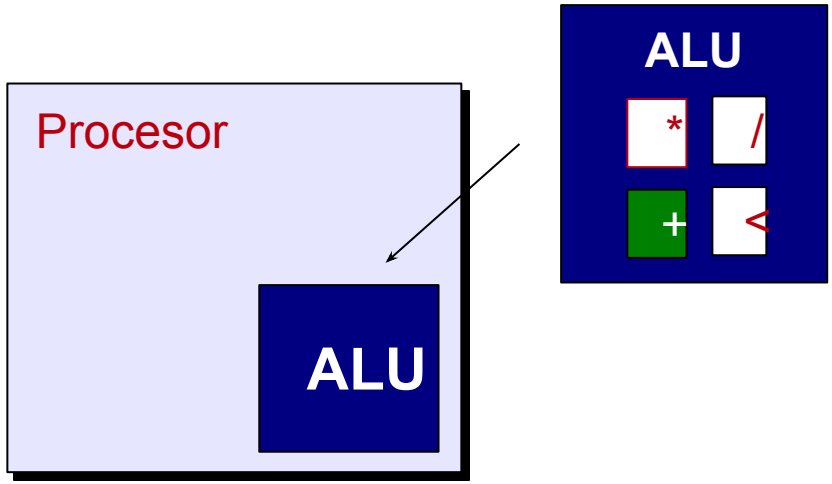
Implementace pomocí HW architektury



- Na čipu může být umístěno a vzájemně propojeno více paralelních sčítaček
- Pro **m-1** sčítaček bude celý výpočet dokončen v **$\log_2(m)$** krocích.

Implementace pomocí SW algoritmu

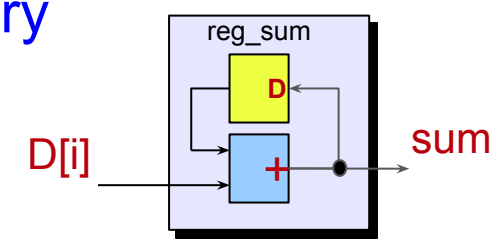
```
sum=0;
for(i=0;i<m;i++)
     sum = sum + D[i];
```



- Procesorové jádro obsahuje v ALU jen jednu sčítačku. V jednom kroku je sečten pouze jeden prvek posloupnosti.

Implementace pomocí HW architektury

- Menší plocha na čipu
- Menší spotřeba energie

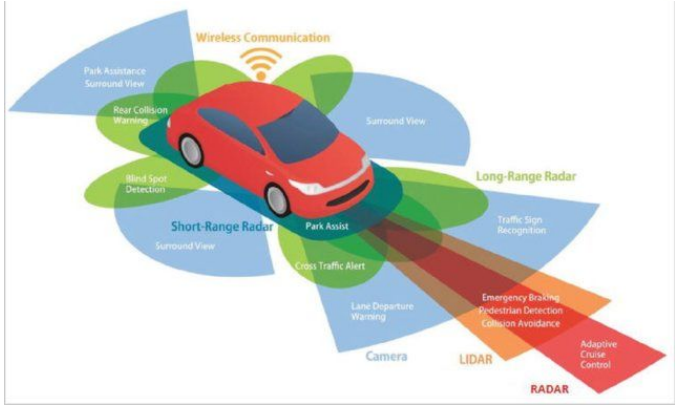


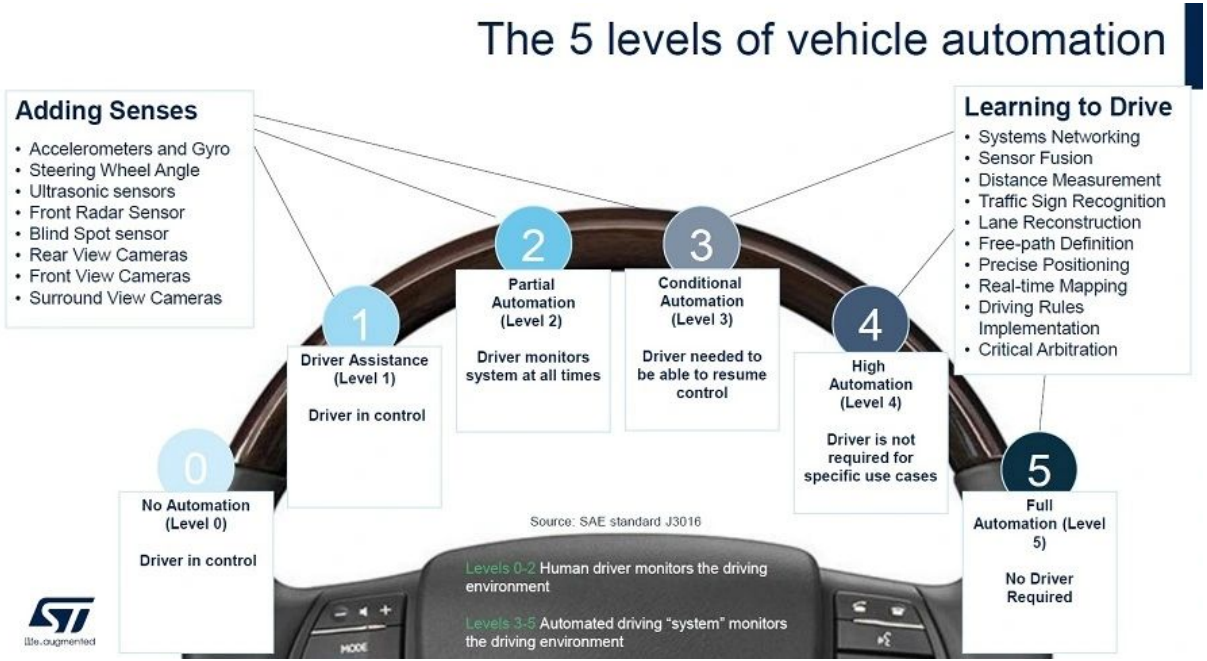
Narůstá objemu zpracovávaných dat

- Multimediální data s možností třídění, náhledů, detekcí osob a dalším zpracování
- Strojové učení a umělá inteligence potřebuje datové sady
- Autonomní systémy vyžadují zpracování dat v reálném čase
- Rozšiřuje se internet věcí a narůstá objem dat ze senzorů

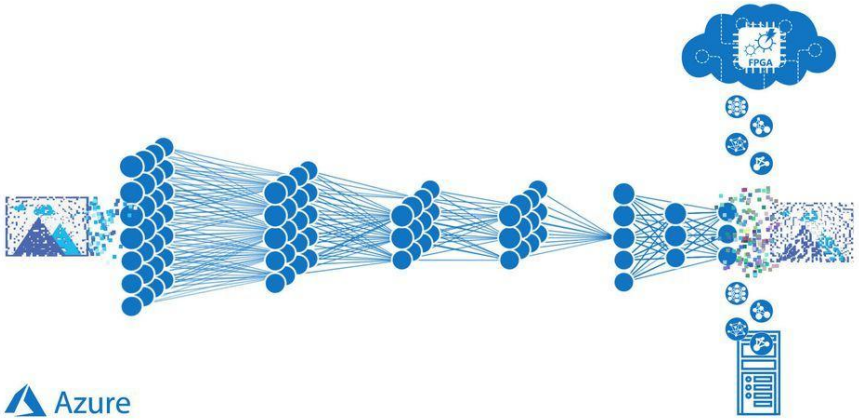
Autonomní řízení a zpracování velkého objemu dat

- Příjem a vyhodnocení dat ze senzorů v reálném čase
- Spousta senzorů → velký objem dat
  - Přední a zadní kamera
  - LiDAR (long- and short-range radar)
  - Ultrazvuk





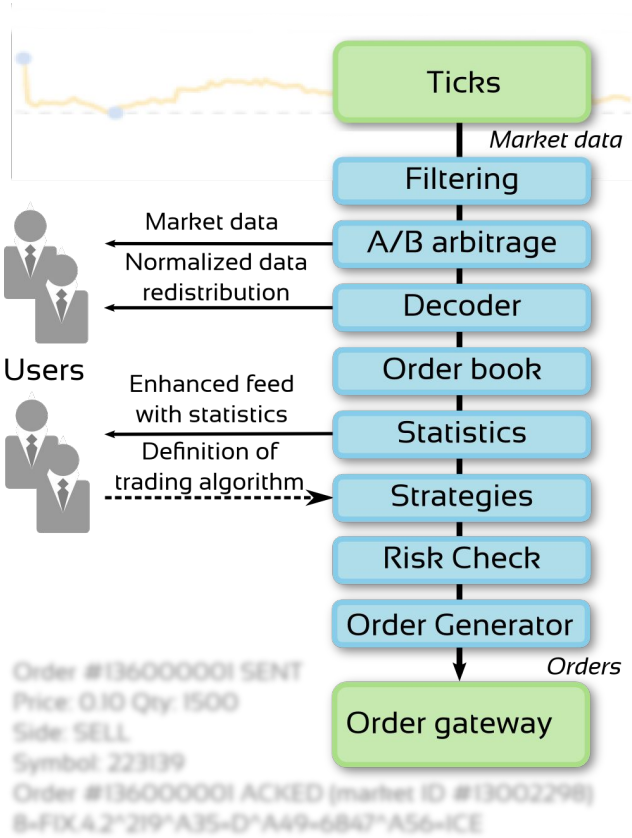
Zpracování v DC: Projekt Catapult a Brainwave AI



- V datových centrech je potřeba zpracovávat velké objemy dat
- Microsoft v roce 2015 odstartoval projekt Catapult pro akceleraci vyhledávače Bing.
- Úspěšný projekt pokračoval v roce 2018 projektem Brainwave
- Cílem projektu Brainwave je akcelerace zpracování dat v datových centrech pomocí hlubokých neuronových sítí
- Díky hardwarové akceleraci výrazná úspora spotřeby DC

Speciální aplikace: Rychlé obchodování na burze

- Rychlá reakce na aktuální stav burzy
  - Dekódování dat z burzy, ukládání stavu burzy
  - Model nejlepších nabídkových a poptávkových cen sledovaných aktiv (akcie, komodity, deriváty)
  - Závod o nejrychlejší reakci
- Brněnská firma Magmio
  - Celé zpracování dat realizováno v hardware
  - Reakce na změnu stavu burzy v řádu stovek nanosekund



Proč použít hardware?

- Většinu aplikací je možné realizovat čistě softwarově
- Hardware umožňuje dosáhnout lepších parametrů
  - Vyšší rychlost zpracování a tím i ***vyšší propustnost***
  - ***Nižší spotřeba*** a delší běh aplikace na baterie
  - Rychlejší reakce na nastalou událost, dosažení ***nižší latence***
  - Miniaturizace výsledného řešení
  - …

Softwarové řešení

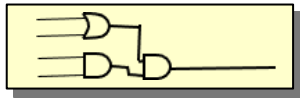
```
void main(){
    ...
    return 0;
}
```

CPU

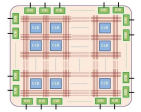
Identifikace kritických částí pro HW zpracování (propustnost, latence, spotřeba, …)

Softwarové řešení s hardwarovou podporou

```
void main(){
    …
    hw_proc(...);
    …
    return 0;
}
```



CPU + HW



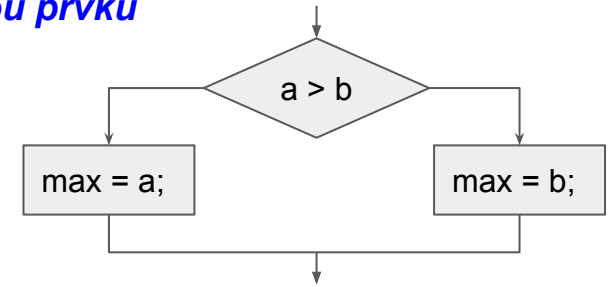
FPGA/ASIC

Co je to algoritmus?

- Intuitivně
  - Postup, který nás dovede k řešení úlohy
- Přesnější definice z kurzu základy programování
  - Přesně definovaná konečná posloupnost příkazů (kroků), jejichž prováděním pro každé přípustné vstupní hodnoty získáme po konečném počtu kroků odpovídající výstupní hodnoty
- Algoritmus je sestaven na základě
  - **Datových struktur** – proměnné, záznamy, pole, lin. seznamy, apod.
  - **Řídících struktur** – sekvence, podmínka, iterace

Příklad algoritmu pro výběr maxima dvou prvků

```
int a, b, max;
if(a>b)
    max = a;
else
    max = b;
```

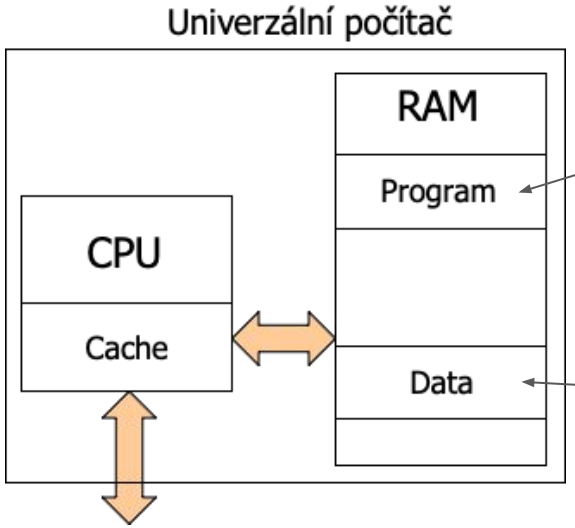


Softwarová realizace algoritmu

- Výpočet běží na ***univerzálním procesoru (CPU)***
- ***Datové struktury*** i předpis programu jsou uloženy ***v paměti RAM***
- Na základě lokality jsou data i program přesouvány mezi pamětí RAM a interní pamětí cache procesoru
- ***Vstupy/Výstupy*** – dodávány skrze V/V zařízení (disk, monitor, porty, apod.)

Algoritmus

```
int a, b, max;
if(a>b)
    max = a;
else
    max = b;
```



Program

```
if(a>b)
    max = a;
else
    max = b;
```

Data

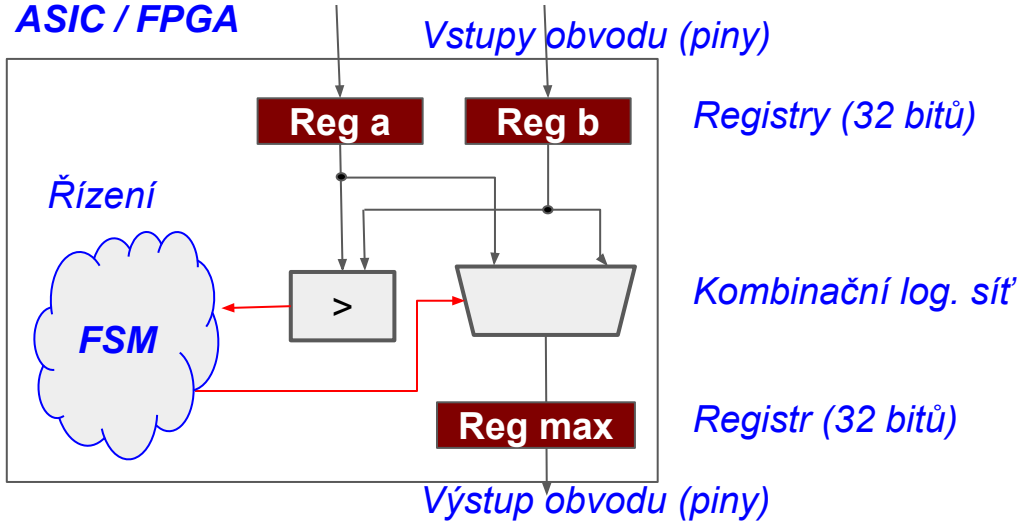
```
int a, b, max;
```

Hardwarová realizace algoritmu

- Výpočet běží na ***specializovaném čipu*** (obvykle ***ASIC*** nebo ***FPGA***)
- ***Datové struktury*** – uloženy v registrech nebo paměťových blocích
- ***Řídící struktury*** – výpočet realizovat skrze ***datovou cestu (funkční jednotky)***, která je ovládána ***řadičem (FSM)***
- ***Vstupy/výstupy*** – dostupné skrze vstupní/výstupní piny obvodu

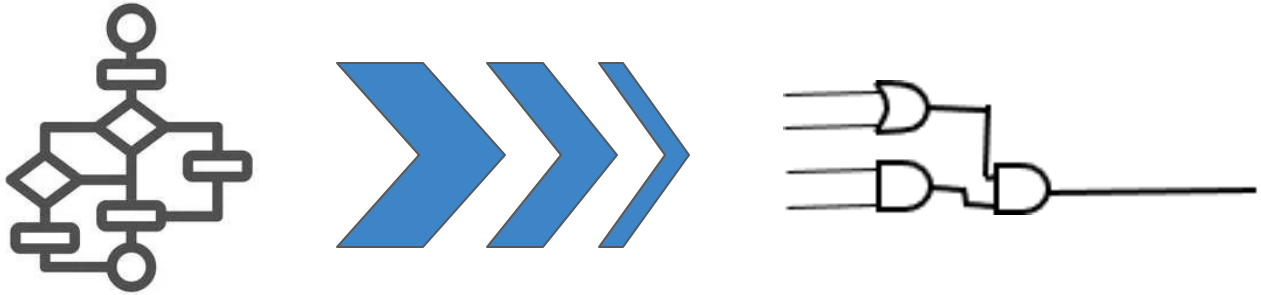
Algoritmus

```
int a, b, max;
if(a>b)
    max = a;
else
    max = b;
```



Od algoritmu k číslicovému obvodu

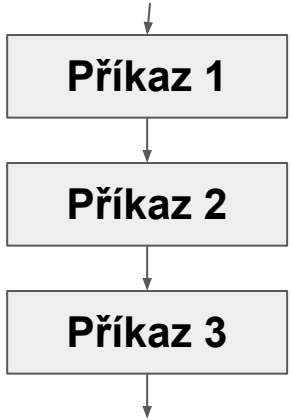
- Dijkstra dokázal, že libovolný algoritmus lze zaznamenat skrze kombinaci tří řídících struktur:
  - **Sekvence (posloupnost)**
  - **Selekce (podmínka)**
  - **Iterace (cyklus)**
- Pojďme si ukázat, jak lze tyto struktury realizovat v hardware



Osnova

- Proč mapovat algoritmus do hardware
- ***Mapování základních konstrukcí do hardware***
  - Sekvence
  - Selekce
  - Iterace
- Modulární návrh
- Shrnutí

- ***Sekvence příkazů*** je řada za sebou navazujících kroků - jednotlivé kroky jsou definovány pomocí příkazů



- Datová závislost
  - Dvojice příkazů je ***datově závislá***, pokud ***vstup jednoho příkazu závisí na výstupu druhého příkazu*** nebo naopak. V opačném případě jsou příkazy datově nezávislé
- Příklad datově závislého příkazu
  - Druhý příkaz potřebuje hodnotu C z prvního příkazu

```
C = A + B;
E = C * D;
```

- Příklad datově nezávislého příkazu
  - Příkazy nepracují s výstupem jiného příkazu

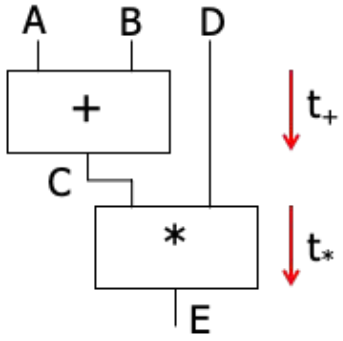
```
C = A + B;
E = A * D;
```

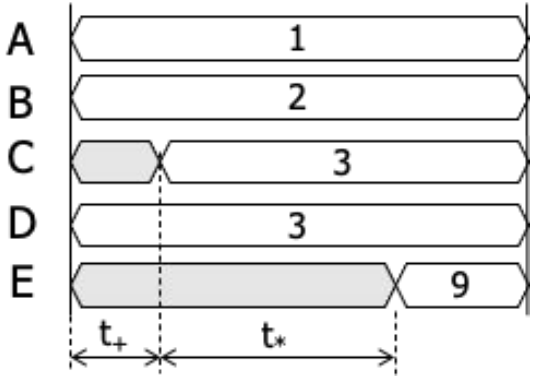
- Doba pro vykonání datové závislých příkazů je dána součtem dob jednotlivých příkazů (bloků)

Příklad

Datová závislost

```
C = A + B;
E = C * D;
```





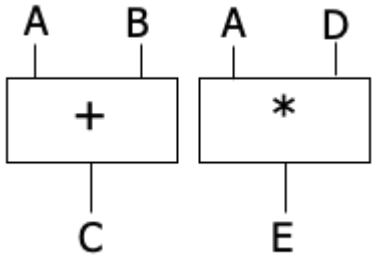
- Celková doba výpočtu $\mathbf{t = t_+ + t_*}$
- V reálném obvodu je nutné uvažovat také zpoždění vodičů
- Vstupní *signály A, B, D musí být stabilní* po celou dobu t
- Dokud se výpočet C a E neustálí, jsou na vodičích nedefinované hodnoty

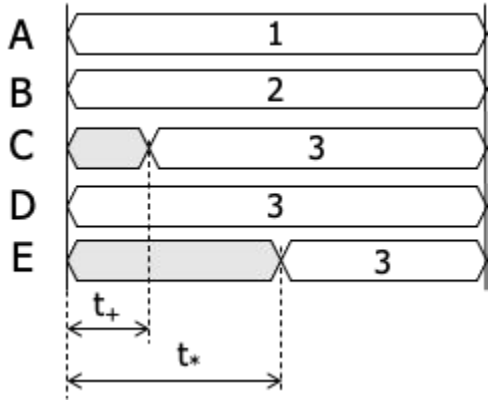
- Doba pro vykonání datové nezávislých příkazů je dána maximem dob jednotlivých příkazů (bloků)

Příklad

Bez datové závislosti

```
C = A + B;
E = A * D;
```



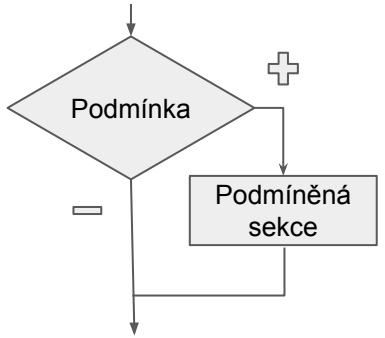


- Celková doba výpočtu $\mathbf{t = max(t_+, t_*)}$
- V reálném obvodu je nutné uvažovat také zpoždění vodičů

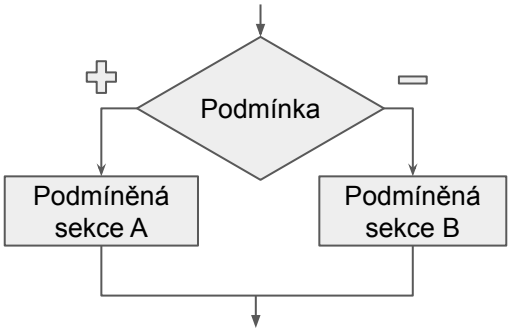
Selekce

- Skrze selekci lze ovládat průběh vykonávání jednotlivých příkazů algoritmu
- Existují různé varianty
  - Selekce s jednou podmíněnou sekcí (if)
  - Výběr mezi dvojicí podmíněných sekcí (if else)
  - Výběr mezi několika podmíněnými sekcemi (switch)
- ***Realizace na úrovni software***
  - Nejprve je vyhodnocena podmínka a až potom vykonány příkazy některé z podmíněných sekcí
- ***Realizace na úrovni hardware***
  - Všechny podmíněné sekce je možné vykonávat paralelně včetně vyhodnocení podmínky
  - Na závěr se pomocí multiplexoru vybírá požadovaný výstup z podmíněné sekce, která odpovídá vyhodnocené podmínce
  - Alternativně lze výpočetní zdroje mezi podmíněnými sekcemi sdílet

if(Podmínka) sekce;



if(Podmínka) A else B;



Příklad

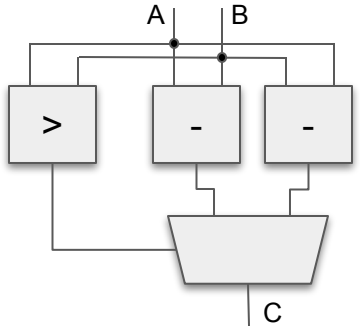
absolutní hodnota z A a B

```
if (A > B)
   abs = A-B;
else
   abs = B-A;
```

- **Rychlá varianta**
  - Výpočet obou sekcí probíhá paralelně
  - Na závěr je multiplexorem vybrán výsledek
- **Varianta malá plocha**
  - Nejprve jsou vybrány multiplexorem vstupní parametry
  - Na závěr je proveden výpočet

Rychlá varianta

Paralelní sekce



Doba výpočtu

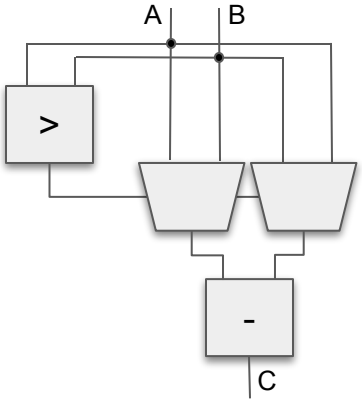
$t = \max(t_{>}; t_{-}) + t_{mx}$

Zdroje

- ➔ 2x odčítačka,
- ➔ 1x multiplexor,
- ➔ 1x komparátor

Varianta malá plocha

Sdílení zdrojů



Doba výpočtu

$t = t_{>} + t_{mx} + t_{-}$

Zdroje

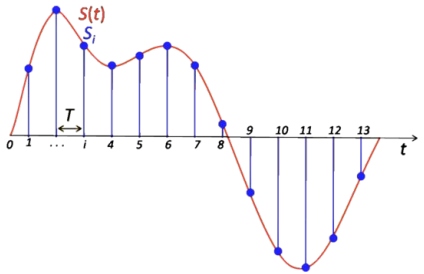
- ➔ 1x odčítačka,
- ➔ 2x multiplexor,
- ➔ 1x komparátor

- Často potřebujeme, aby byla ***sekvence příkazů*** resp. celý algoritmus vykonáván ***opakovaně***, v některých případech i v nekonečné smyčce

Příklady aplikací vyžadující prodouvé zpracování dat

- Senzory produkují proud naměřených dat

- Digitální audio - proud vzorků dat signálu zvuku



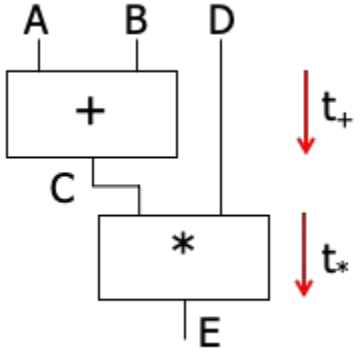
- Digitální video - proud jednotlivých obrazových bodů a snímků obrazovky
- Přenosy po síti - data se přenáší ve formě paketů a síťových toků

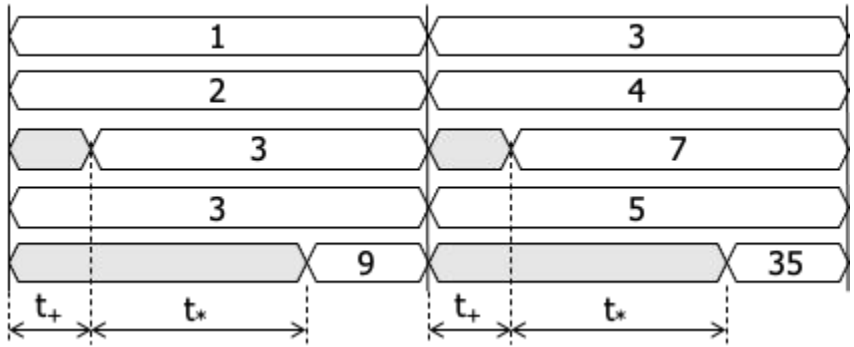
Opakované vykonávání sekvence příkazů

- U kombinačních obvodů je potřeba udržet vstupní parametry stabilní alespoň do té doby, dokud se výsledek neustálí
- Přiložením nových vstupních parametrů začíná opět proces výpočtu a výstupy jsou po určitou dobu nestabilní

Příklad

```
C = A + B;
E = C * D;
```



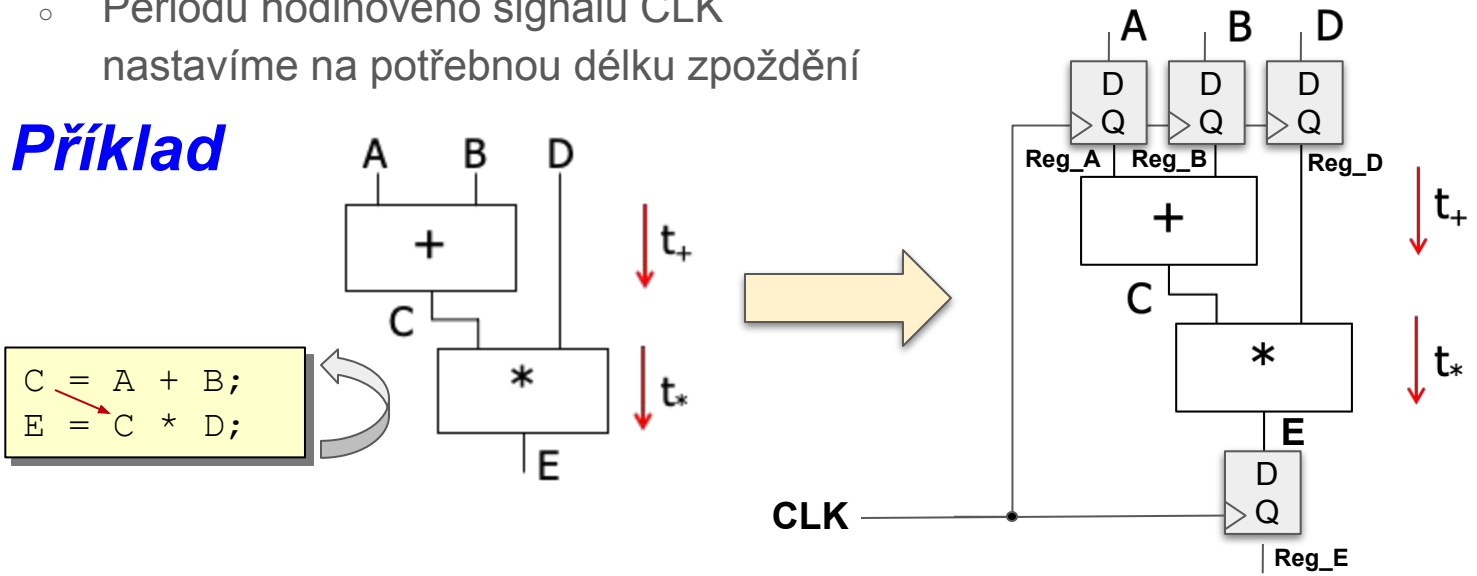


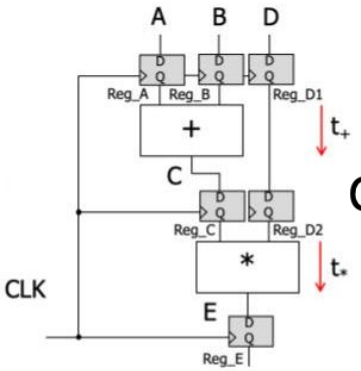
- Jak realizovat požadované zpoždění?
- Jak udržet vstupní parametry stabilní po celou dobu výpočtu?
- Jak ve správný okamžik uložit výsledek?

Opakované vykonávání sekvence příkazů

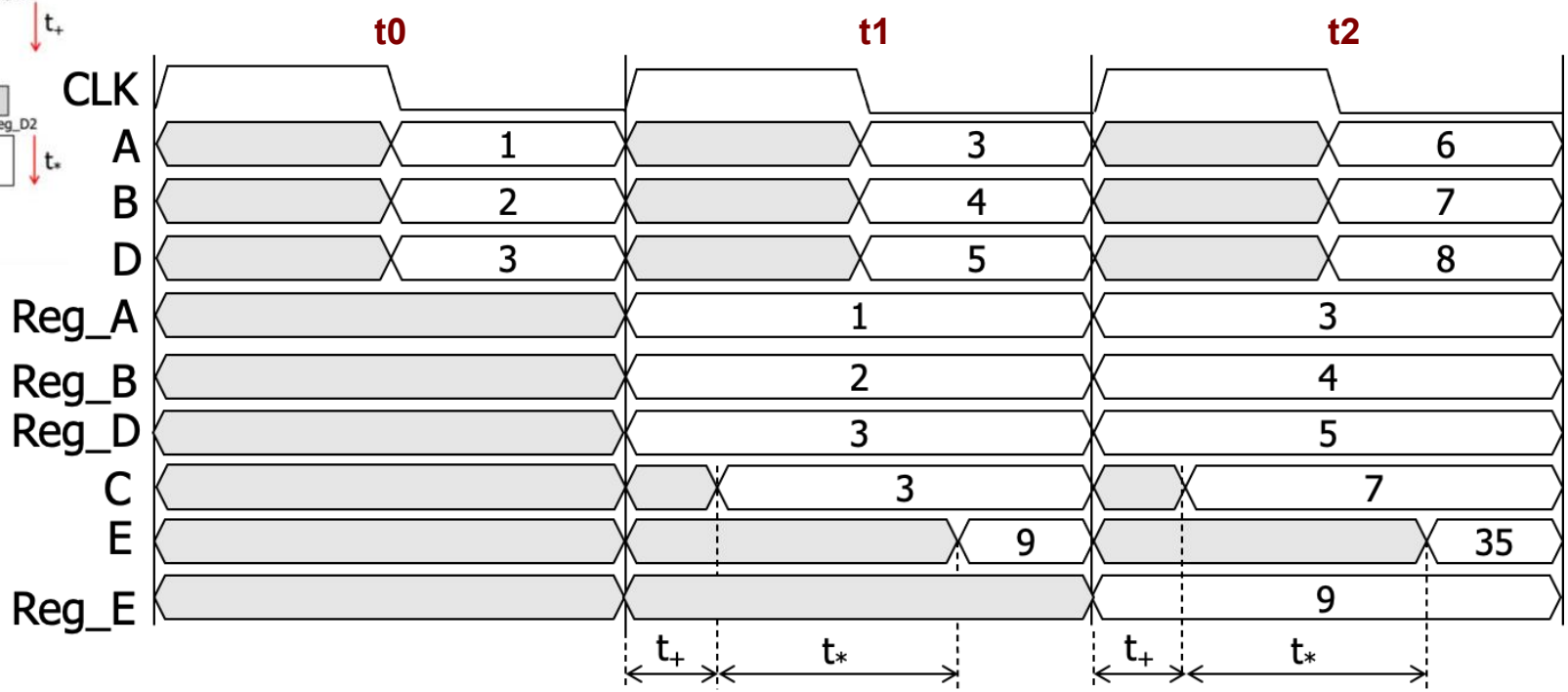
- Uvedené problémy lze efektivně řešit převedením obvodu
  na synchronní obvod řízený hodinovým signálem CLK
- Postup řešení:
  - Stabilitu vstupních a výstupních parametrů zajistíme vložením registru
  - Periodu hodinového signálu CLK nastavíme na potřebnou délku zpoždění

Příklad





Časový diagram synchronního obvodu

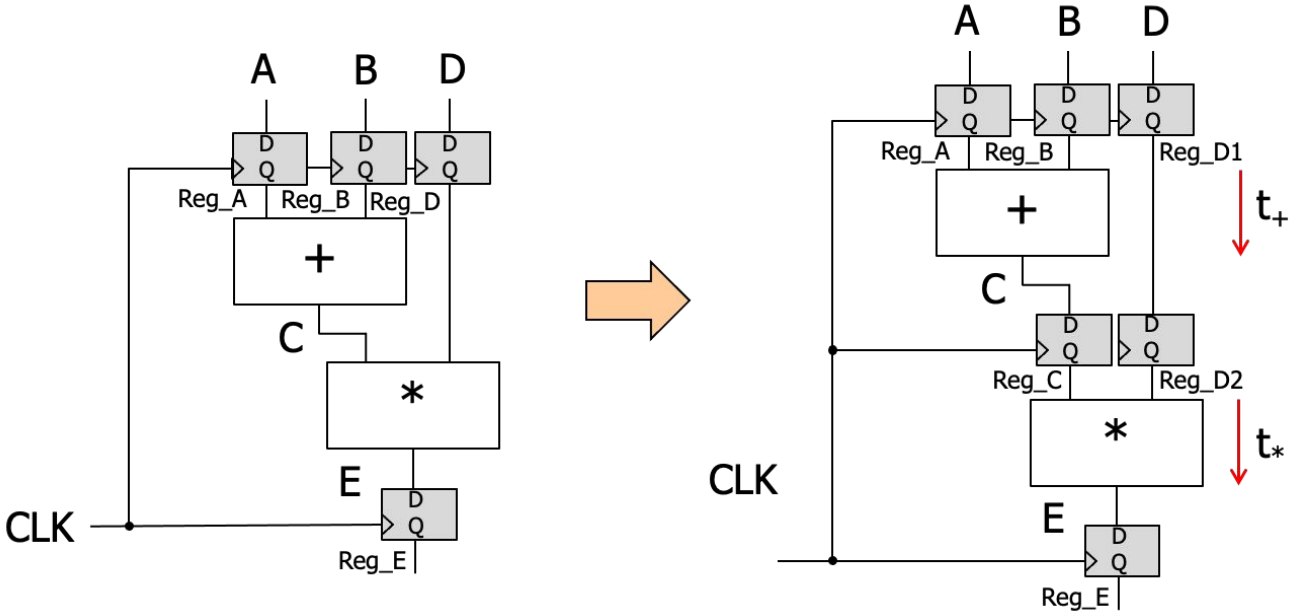


Opakované vykonávání sekvence příkazů

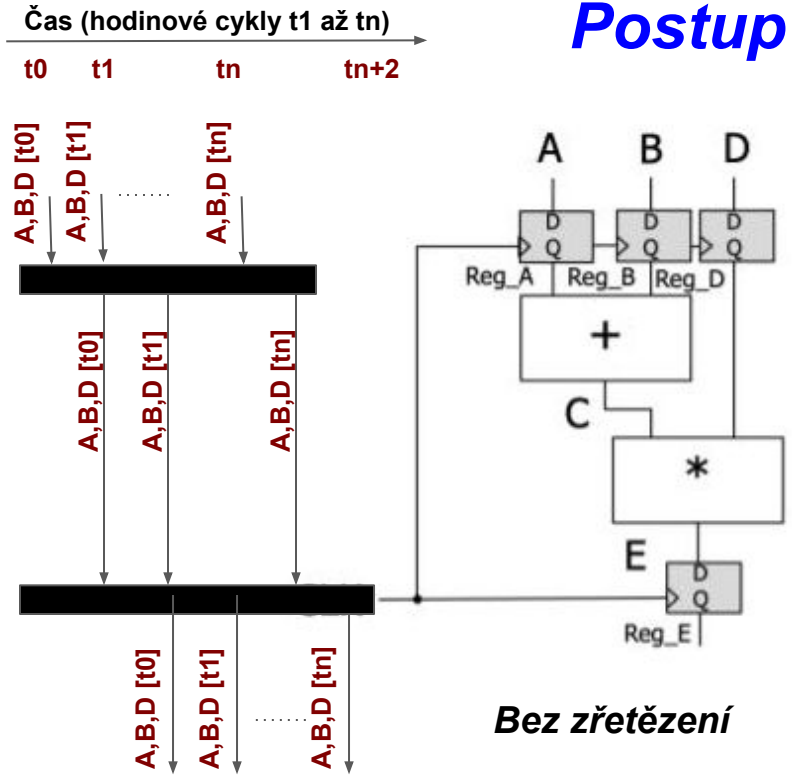
- ***Důsledky:***
  - Výstupy registrů jsou stabilní po celou dobu periody
  - *Přidáním registrů se prodlužuje latence obvodu*
    - Vstupní parametry přiložené na vstup obvodu v rámci hodinového taktu i se zúčastní výpočtu až v hodinovém taktu i+1
    - Výstupní hodnota vypočtené v hodinovém taktu i+1 je k dispozici na výstupu obvodu až v hodinovém taktu i+2
  - Doba periody musí pokrývat dobu výpočtu + setup time pro uložení výsledku do výstupního registru + zpoždění vodičů
- ***Problémy:***
  - Pokud je sekvence příkazu příliš dlouhá nebo dokonce proměnlivá (např. podmíněné sekce s různou délkou), potom se nadměrně prodlužuje i délka periody hodinového signálu
  - Řešením je *zřetězené zpracování*

Opakované vykonávání sekvence příkazů - zřetězení

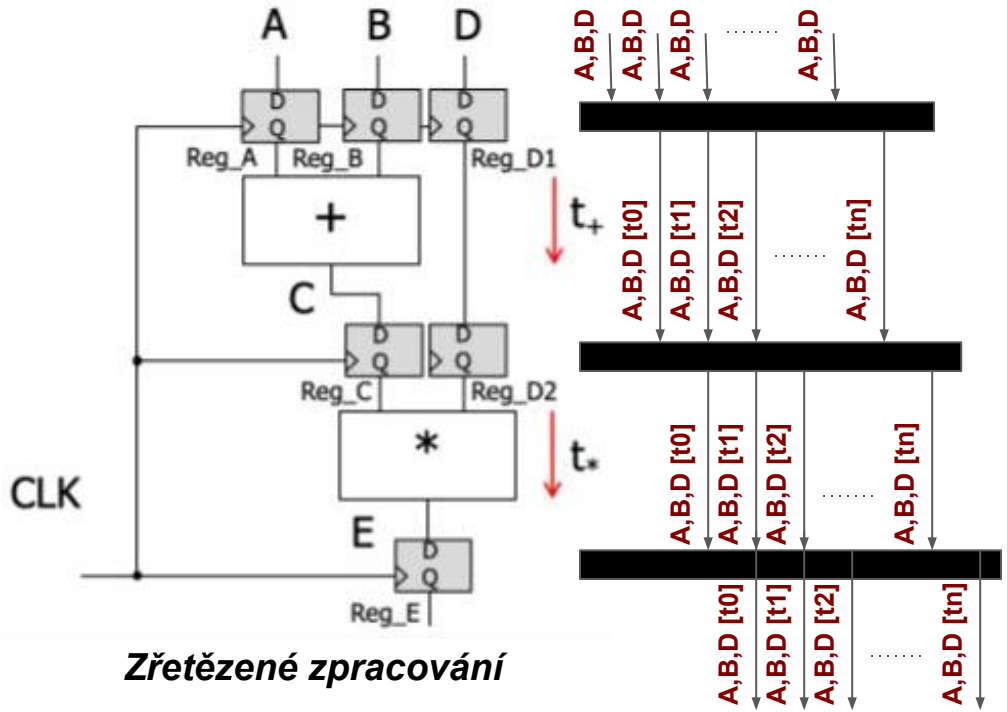
- Vložením dalších registrových stupňů dovnitř sekvence příkazu je možné výpočet rozdělit na menší části, které však mohou současně (paralelně) zpracovávat oddělené části po sobě jdoucích iterací



Postup zpracování dat

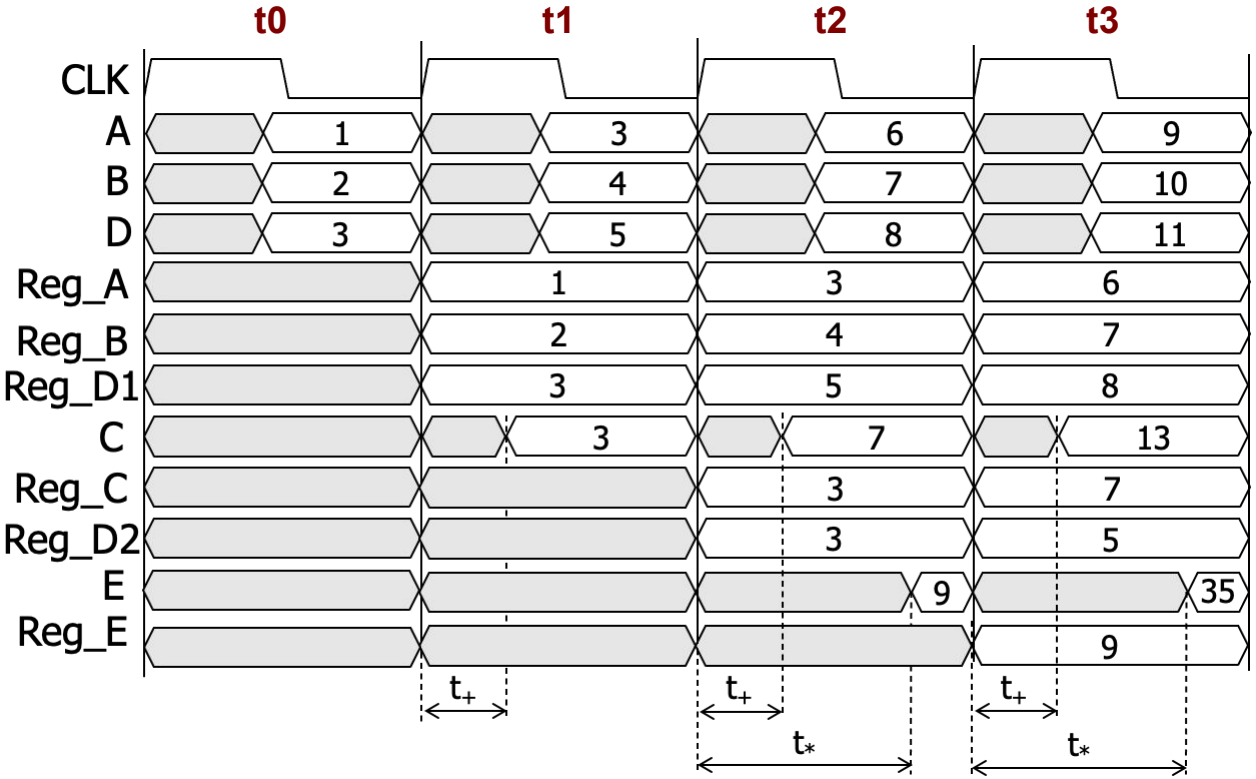


Bez zřetězení



Zřetězené zpracování

Časový diagram pro zřetězené zpracování



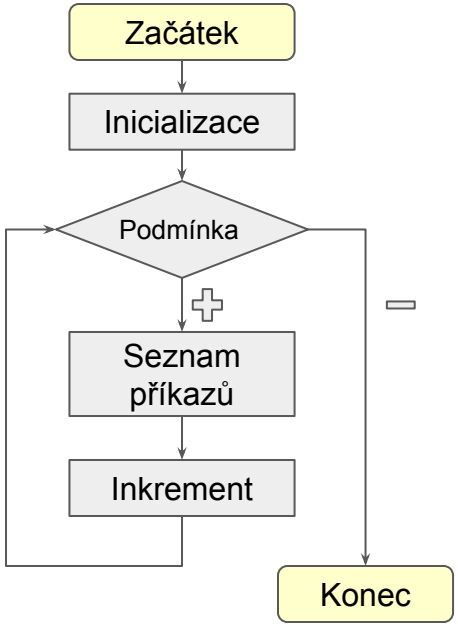
Výpočet je rozdělen do více hodinových cyklů

Zřetězené zpracování

- Poznámky:
  - Při vytváření stupně se musí registry vložit i doprostřed propojovacích vodičů (např. **Reg_D2**), jinak by nebyla v navazujícím stupni data ve správný čas a výpočet by nefungoval správně
- Důsledky:
  - Přidáním registrů dovnitř obvodu se dále prodlužuje latence obvodu, tj. výsledek výpočtu je dostupný o takt později
  - Doba periody se zkracuje na ***t = max[t_+,t_*] + setup time + zpoždění vodičů***, neboť jednotlivé příkazy (operace) se vykonávají nezávisle v oddělených stupních
  - Technika zřetězení je nejvíc efektivní pokud je délka výpočtu v jednotlivých stupních ***vyvážená*** – vede na minimální délku periody tj. maximální rychlost výpočtu
- Problémy:
  - Zřetězení nelze aplikovat na výpočty se zpětnou vazbou, neboť vložení registru by prodloužil zpětnou vazbu o jeden takt. Obvod by dostal data zpětnou vazbou pozdě.

Iterace

- Umožňuje opakování určité části algoritmu
- Rozlišujeme několik typů
  - Cyklus ***for*** s pevným počtem iterací
  - Cyklus ***while*** a ***do while*** s proměnným počtem iterací
- Všechny typy cyklů zahrnují:
  - Inicializační část
  - Podmínku ukončení cyklu
  - Inkrement
  - Seznam příkazů



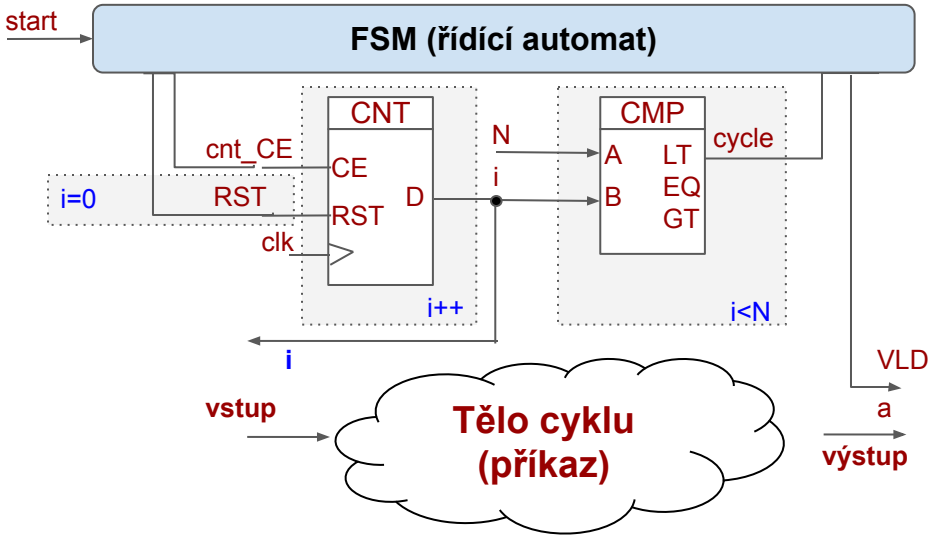
Příklad pro smyčku FOR

```
for(inicializace; podmínka; inkrement)
{ seznam příkazů }
```

- Všechny tyto 4 části je potřeba realizovat i na úrovni hardware

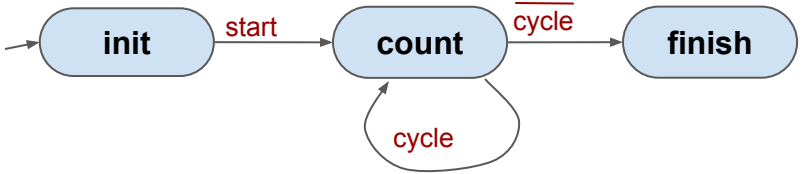
- Zpracování dat v cyklu

```
for (i=0;i<N;i++){
   příkaz;
}
```



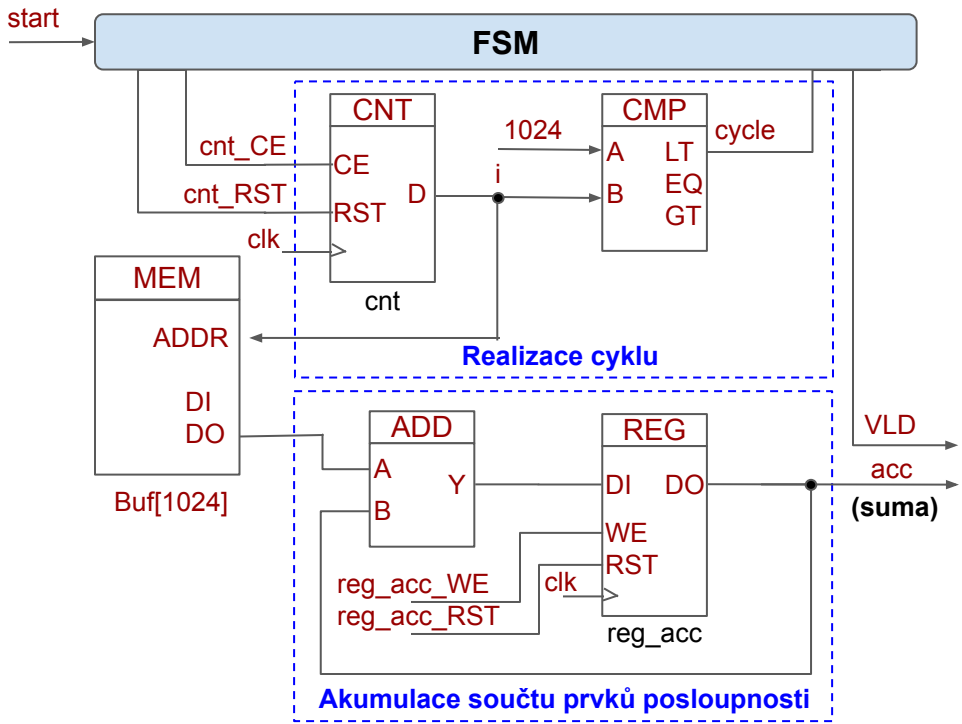
Činnost obvodu

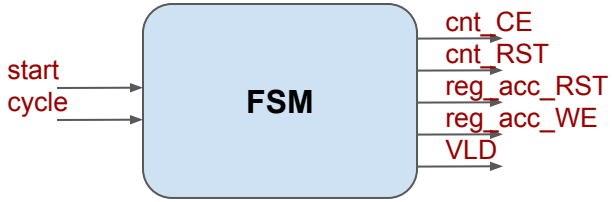
- Před začátkem cyklu je potřeba ve stavu ***init*** inicializovat proměnou cyklu *i (i=0)*. Čítač je inicializován signálem reset *(RST=1)*
- Cyklus je startován signálem start *(start=1)*, kdy obvod přechází do stavu ***count***
- Ve stavu ***count*** v každém cyklu hodin (tiku) je inkrementován čítač *(i++)* signálem *cnt_CE* a je provedeno tělo cyklu
- Komparátor kontroluje podmínku ukončení cyklu *(i<N)* a v případě nesplnění podmínky je cyklu ukončen - přechod do stavu ***finish***



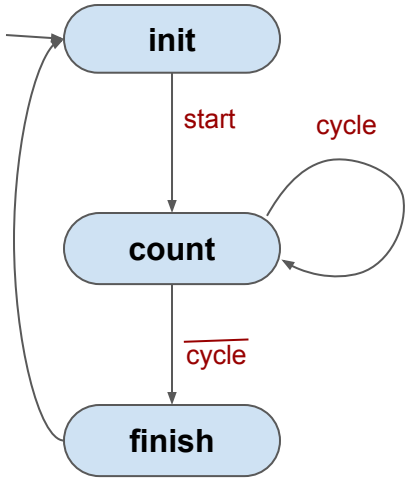
Příklad smyčky FOR: sečtení prvků posloupnosti

```
for (i=0;i<1024;i++)
   acc=acc+Buf[i];
```





Stavový automat




```
init:
     cnt_RST=1;
     reg_acc_RST=1;
```

```
count:
     cnt_CE=1;
     reg_acc_WE=1;
```

```
finish:
     VLD=1;
```

Pozn: Řízení signálů na základě aktuálního stavu. Pokud není některý výstup ve stavu nastaven, má nulovou hodnotu.

Příklad smyčky FOR: sečtení prvků posloupnosti

```
for (i=0;i<1024;i++)
   acc=acc+Buf[i];
```

- Vstupní pole ***Buf[1024]*** je uloženo v paměti RAM nebo ROM
- Před začátkem výpočtu je potřeba inicializace, která je řešená stavem ***init***:
  - Vynulovat proměnnou cyklu ***i*** signálem (cnt_RST = 1)
  - Vynulovat obsah registru ***reg_acc*** signálem (reg_acc_RST = 1)
- Ve stavu ***count*** probíhá tělo cyklu a v každém cyklu (tiku) hodinového signálu:
  - je inkrementovaná proměnná cyklu ***i*** pomocí čítače (cnt_CE = 1). Hodnota čítače současně slouží jako adresa do paměti ROM, odkud se přečte prvek pole indexovaný proměnou ***i***
  - Prvky posloupnosti čtené z paměti jsou postupně akumulovány v registru ***reg_acc***
- Celý cyklus končí v okamžiku nesplnění podmínky cyklu ***i < 1024*** signál (cycle = 0), obvod přechází do stavu ***finish*** a v registru ***reg_acc*** je výsledek

Rozbalení smyčky do paralelního zpracování

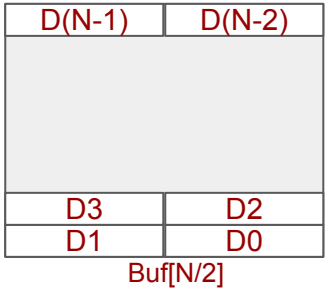
```
for (i=0;i<N;i++)
   a = a + Buf[i];
```

N řádků (iterací)

Paralelní zpracování těla smyčky

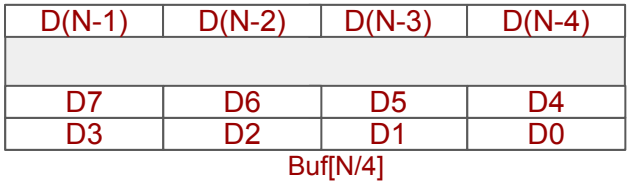
```
for (i=0;i<N/2;i++){
   a = a + Buf[i].D0;
   a = a + Buf[i].D1;
}
```

N/2 řádků (iterací)



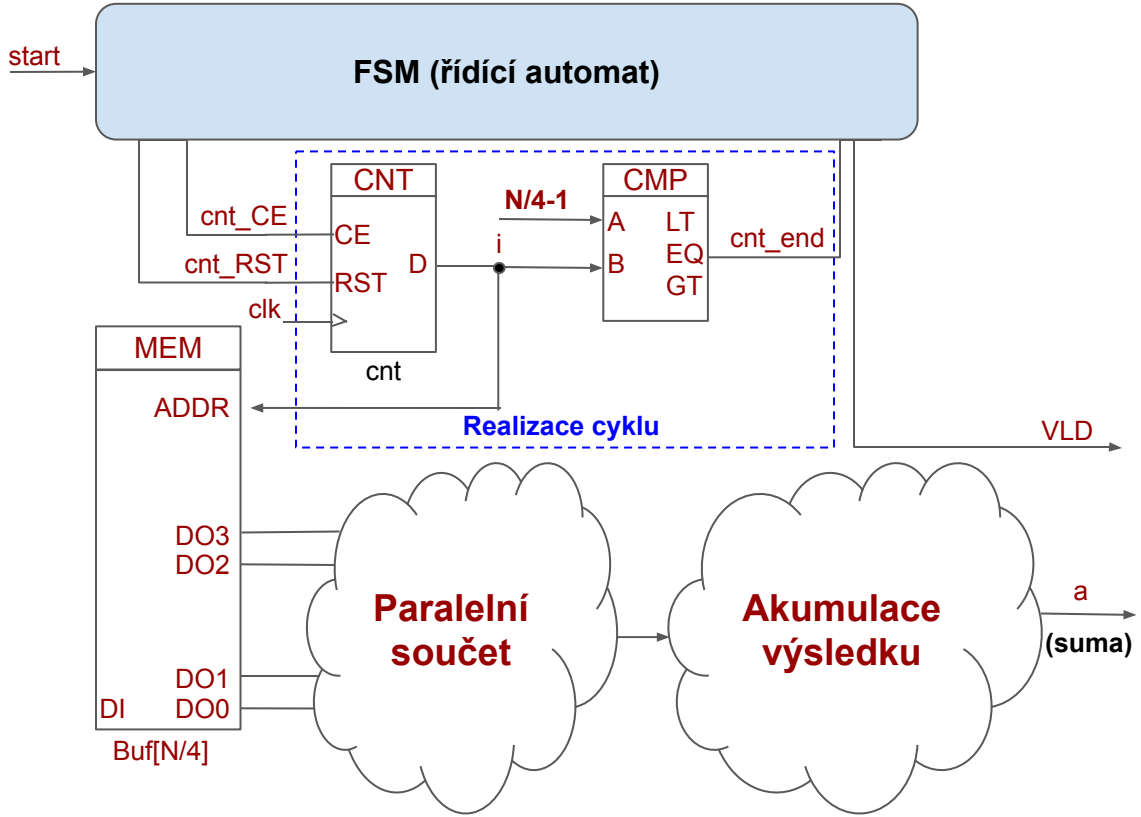

```
for (i=0;i<N/4;i++){
   a = a + Buf[i].D0;
   a = a + Buf[i].D1;
   a = a + Buf[i].D2;
   a = a + Buf[i].D3;
}
```

N/4 řádků (iterací)

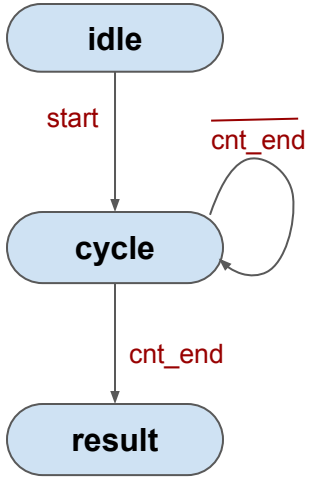


Paralelní čtení dat → paralelní zpracování dat → redukce počtu cyklů

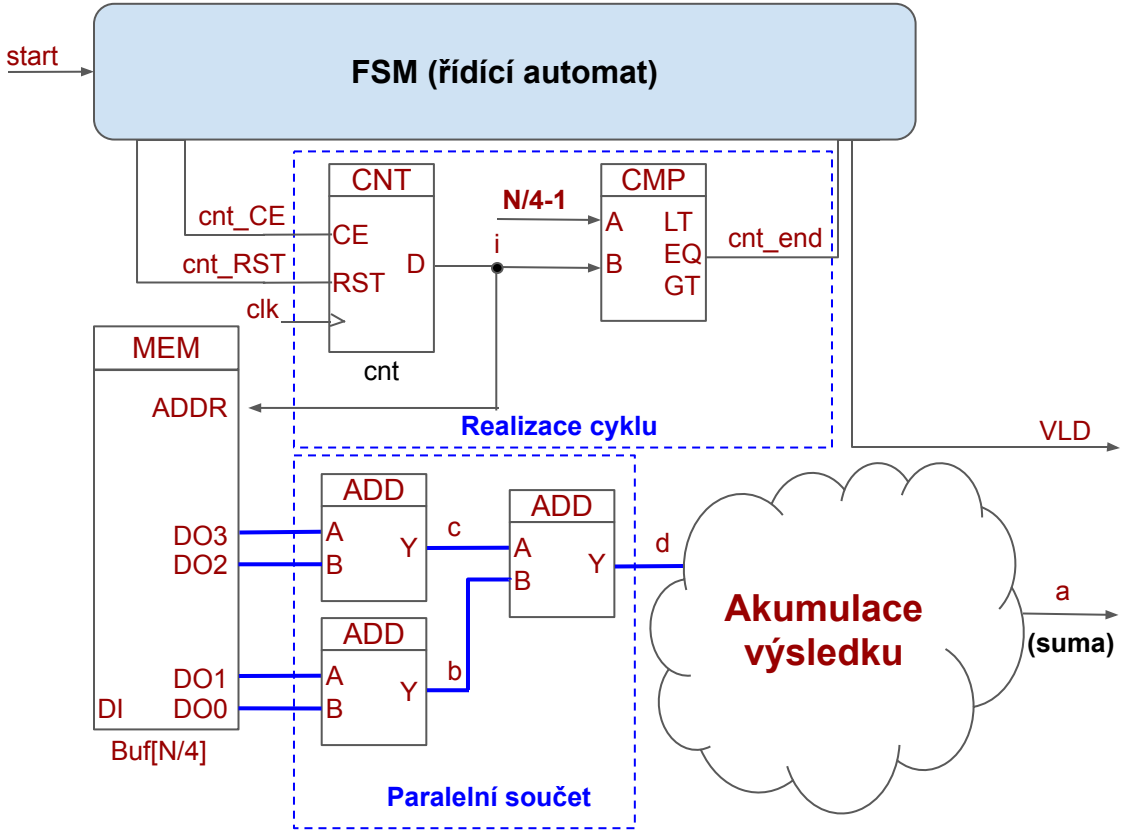
Iterace - příklad rozbalení smyčky



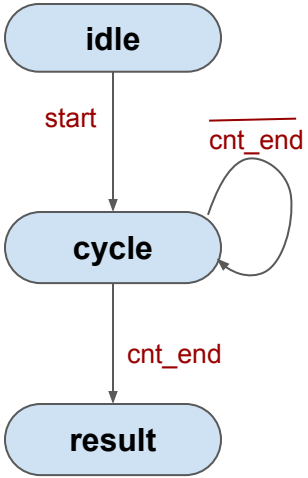

```
for (i=0;i<N/4;i++){
   a=a+Buf[i].D0;
   a=a+Buf[i].D1;
   a=a+Buf[i].D2;
   a=a+Buf[i].D3;
}
```



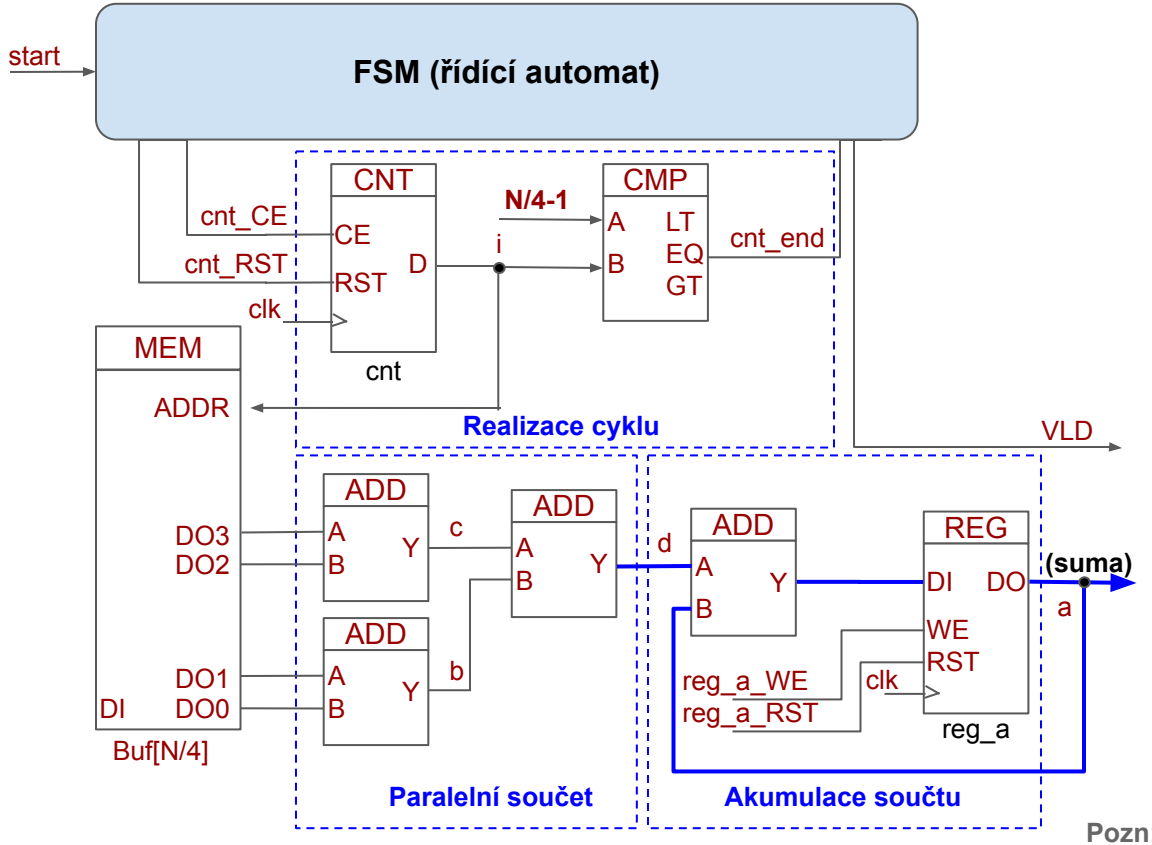
Iterace - příklad rozbalení smyčky



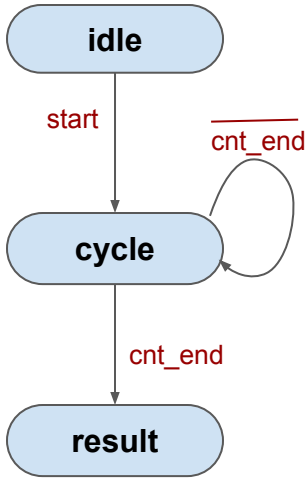

```
for (i=0;i<N/4;i++){
   b=Buf[i].D0+Buf[i].D1;
   c=Buf[i].D2+Buf[i].D3;
   d=b+c;
   a=a+d;
}
```



Iterace - příklad rozbalení smyčky




```
for (i=0;i<N/4;i++){
   b=Buf[i].D0+Buf[i].D1;
   c=Buf[i].D2+Buf[i].D3;
   d=b+c;
   a=a+d;
}
```



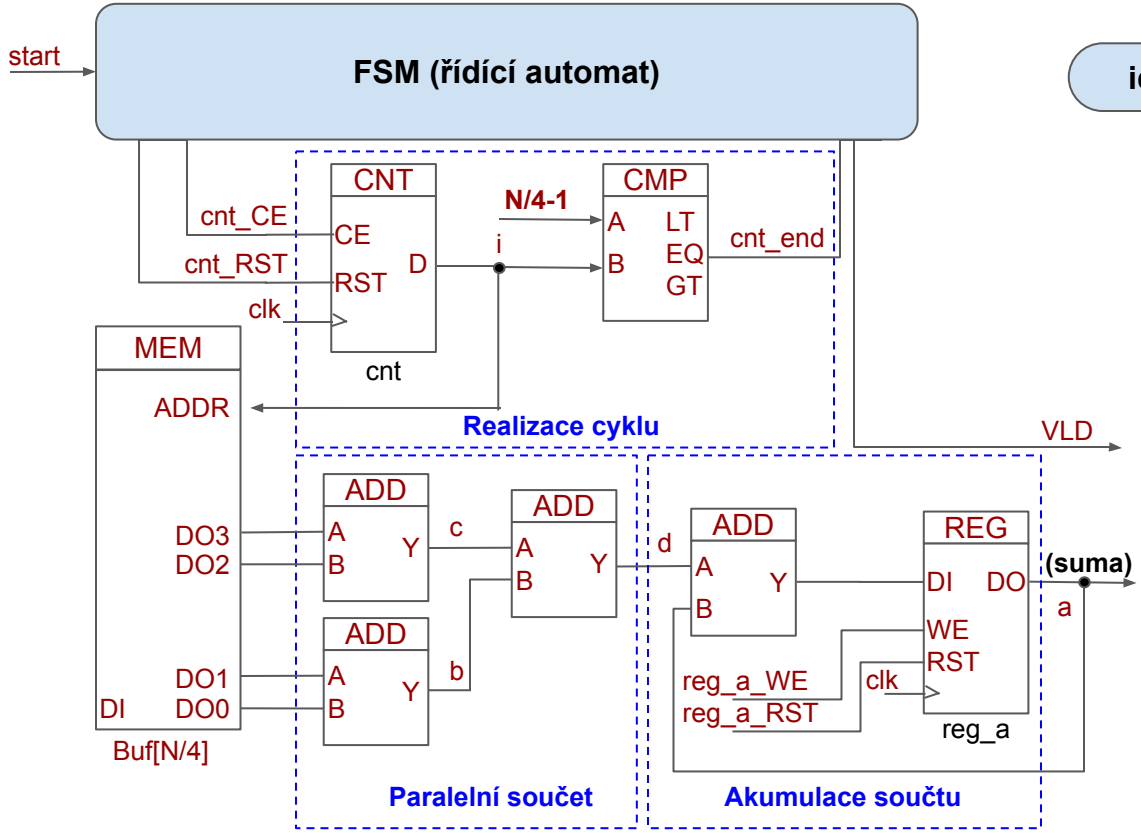

```
idle:
     cnt_RST=1;
     reg_a_RST=1;
```

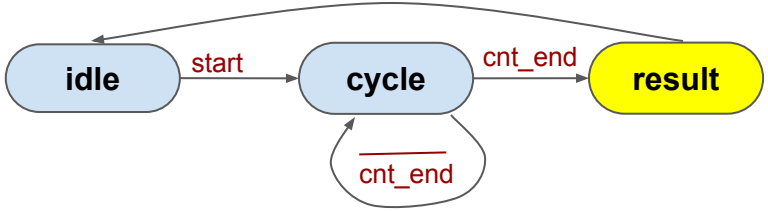
```
cycle:
     cnt_CE=1;
     reg_a_WE=1;
```

```
result:
     VLD=1;
```

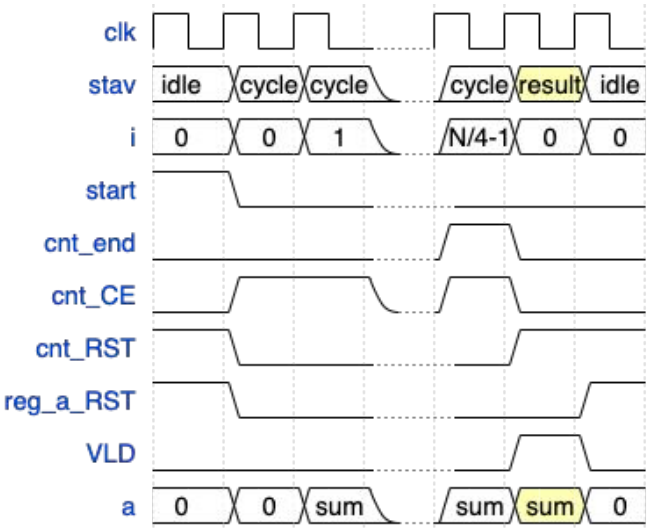
Pozn: Řízení signálů na základě aktuálního stavu. Pokud není některý výstup ve stavu nastaven, má nulovou hodnotu.

Iterace - příklad rozbalení smyčky





Časový diagram



Iterace - rozbalení smyčky

- Rozbalením smyčky se zápis algoritmu stává delším, avšak snižuje se režie spojená s vyhodnocením podmínky cyklu
- Rozlišujeme:
  - Částečné rozbalení smyčky
  - Úplné rozbalení smyčky

Příklad

Rozbalení smyčky s 10 iteracemi

```
int a[10];
int acc = 0;

for(i=0; i<10; i++)
   acc = acc + a[i];
```

Nerozbalená smyčka

```
int a[10];
int acc = 0;

for(i=0; i<10; i+=2){
   acc = acc + a[i];
   acc = acc + a[i+1];
}
```

Částečně rozbalená smyčka

```
int a[10];
int acc = 0;

acc = acc + a[0];
acc = acc + a[1];
acc = acc + a[2];
…
acc = acc + a[9];
```

Úplně rozbalená smyčka

Rozbalení smyčky do paralelního zpracování

- ***Rozbalení smyčky umožňuje vykonání obecně N iterací cyklu paralelně*** (v jednom kroku).
- Musí být splněno několik předpokladů:
  - Fixní počet iterací smyčky, počet iterací nelze měnit v průběhu výpočtu.
  - Pro N paralelních iterací rozbalené smyčky ***musí být k dispozici všechna vstupní data***.
  - ***Mezi i a i+1 iterací není datová závislost:*** Jinými slovy není potřeba v další iteraci smyčky výsledek z předcházející iterace.
- Pokud jsou vstupní data uložena v paměti, musíme při rozbalení smyčky zvýšit propustnost paměti.
  - Zvýšení datové šířky čtených dat.
  - Více paralelních pamětí nebo portů paměti.
  - Násobná frekvence rozhraní pro přístup do paměti.

Iterace - rozbalení smyčky

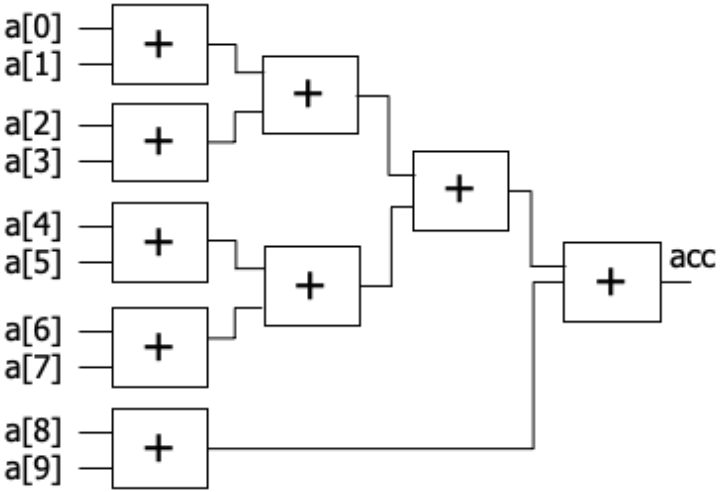
- Pokud jsou jednotlivé příkazy rozbalené smyčky datově nezávislé (popř. je lze upravit na datově nezávislé), potom mohou být zpracovány v hardware paralelně

Příklad

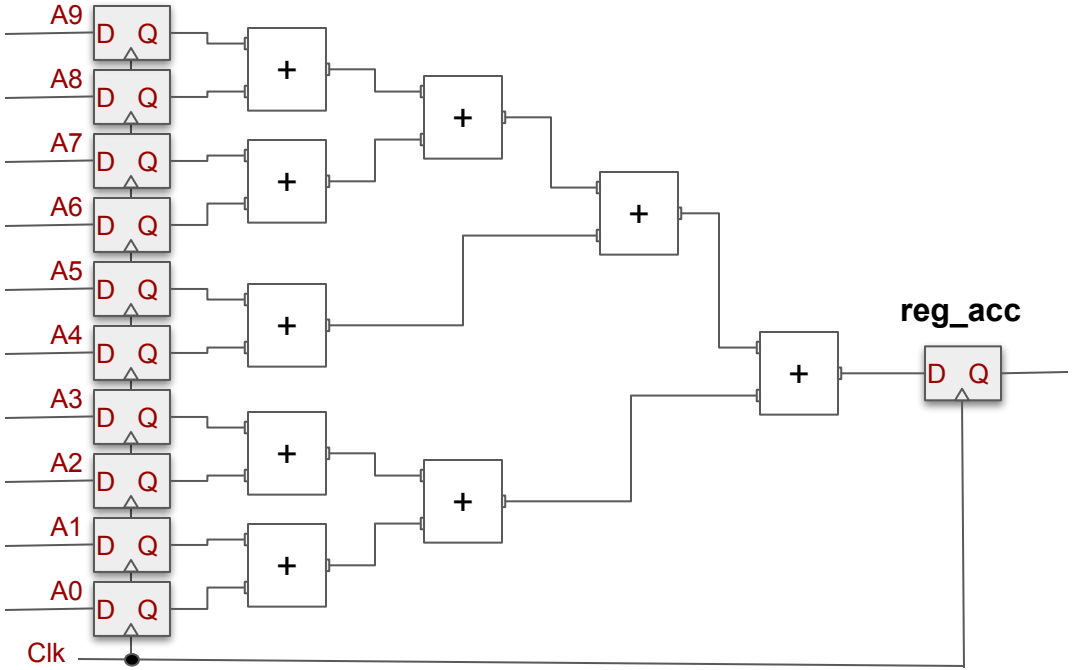
Úplné rozbalení smyčky s 10 iteracemi

```
int a[10];
int acc = 0;

acc = acc + a[0];
acc = acc + a[1];
acc = acc + a[2];
…
acc = acc + a[9];
```

```
int a[10];
int acc;

acc = a[0]+a[1]+ … +a[9];
```

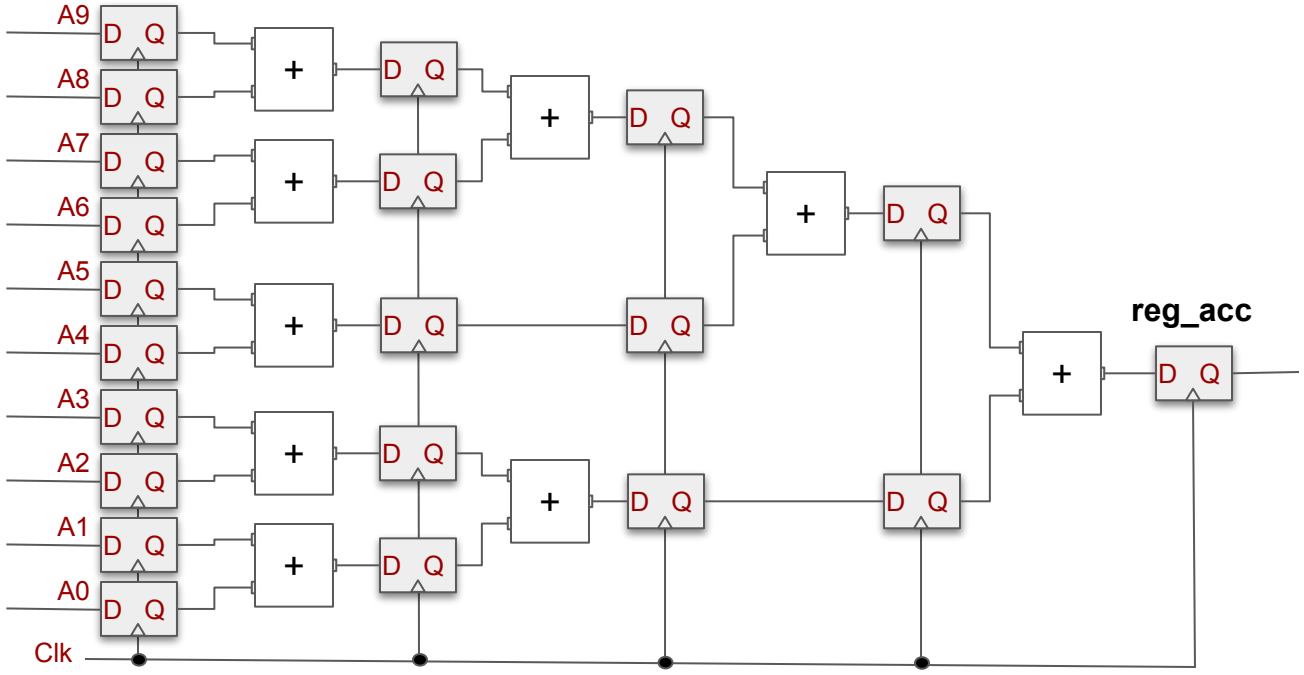


Iterace - úplné rozbalení smyčky



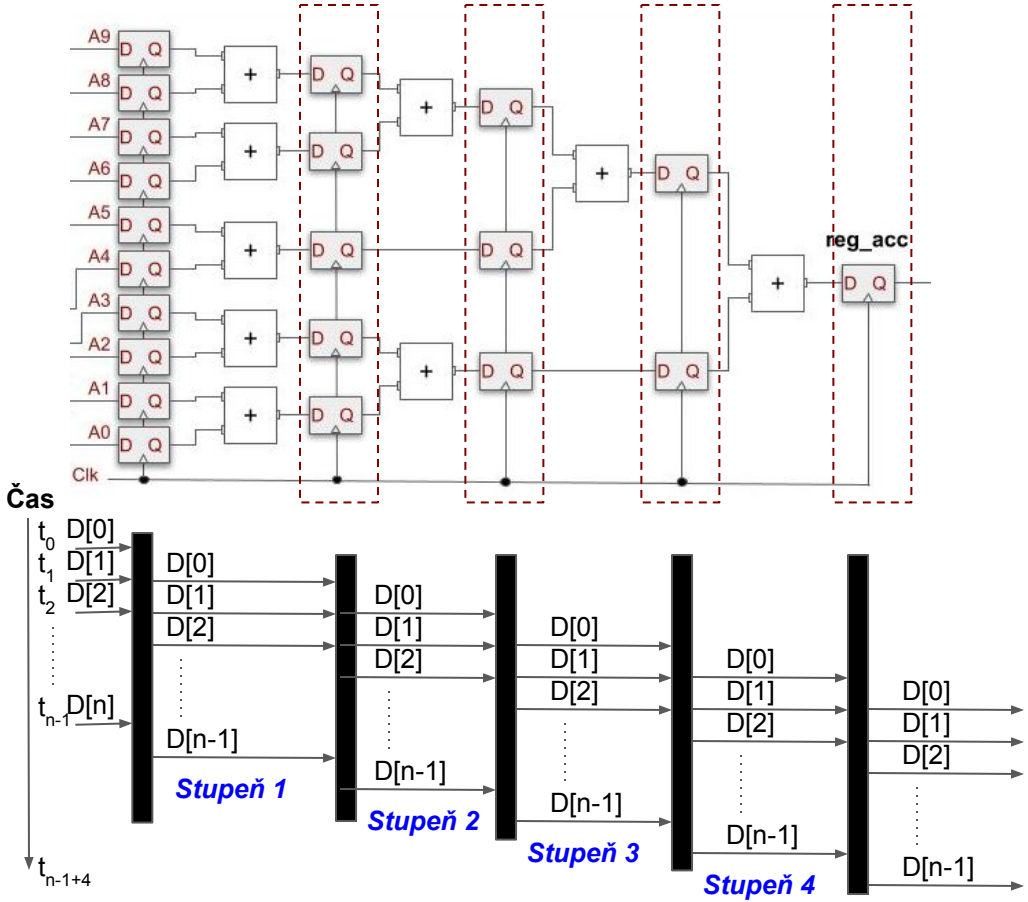
- Délka periody hodin: ***T* = *(4 * t_+)* + *setup time* + *zpoždění vodičů***
- Doba výpočtu: 1 takt

Iterace - rozbalení smyčky se zřetězením



- Délka periody hodin: $T = t_+$ + ***setup time*** + ***zpoždění vodičů***
- Doba výpočtu: 4 takty

Zřetězené zpracování dat



- ***Rozdělení obvodu do M=4 stupňů*** (jednodušších obvodů) pracujících ve zřetězené lince za sebou.
- Každý stupeň zpracuje výsledek za jeden takt hodin Clk a předá jej do navazujícího stupně.
- ***Díky jednoduššímu zpracování*** v jednotlivých stupních ***může obvod pracovat na vyšší frekvenci***, v ideálním případě M-krát rychleji.
- ***První vzorek dat je zpracován až po průchodu všemi stupni linky***, tedy za M=4 taktů hodin Clk.
- ***Zpracování n vzorků dat proběhne za n+M taktů hodin Clk.***

- ***Propustnost***: počet vzorků dat zpracovaných za jednotku času. Nevíme, za jak dlouho dostaneme první výsledek.
- ***Latence***: čas od příchodu prvního vzorku dat po získání prvního výsledku. Většinou se počítá v taktech hodinového signálu.

Příklad

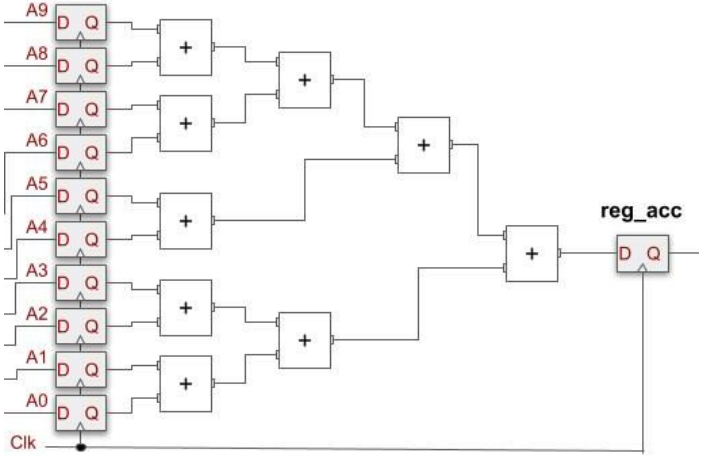
Ukázka rozdílu mezi propustností a latencí

Auto převeze 1 mil. ks DVD nosičů s kapacitou 8,3 GB z Brna do Prahy. Urazí přibližně vzdálenost 200 km za 2 hodiny. V případě plně zaplněných DVD dojde k přenosu dat s propustností 1 000 000*8,3 / (2*3600) = 1152,7 GB/s = 9222 Gb/s, ale latence komunikace je 2 hodiny, neboť data se dostanou do Prahy až za dvě hodiny.

Propustnost a Latence při zřetězeném zpracování dat

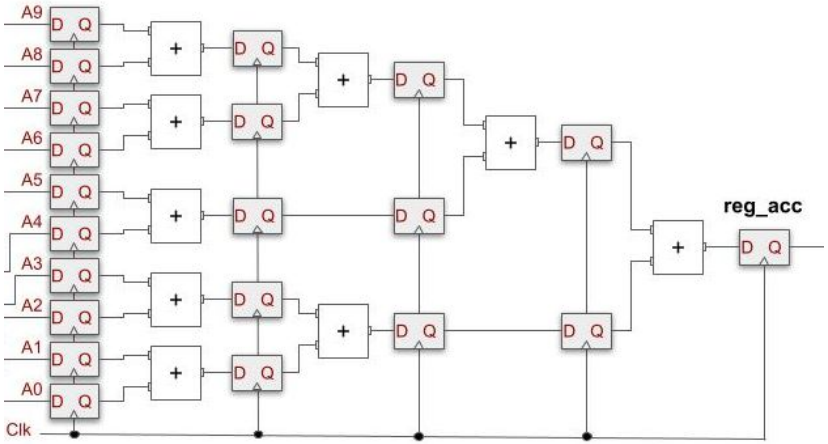
- Na jaké frekvenci může pracovat obvod, když sčítačka má zpoždění výpočtu t_+ = 2 ns, na každém vodiči je zpoždění t_{wire} =1 ns a setup time registru je t_{setup} = 1 ns.

Zpracování v jednom cyklu



T = 5*1+4*2+1 = 14 ns
f = 1000/14 = 71,4MHz
Latence = 1 takt hodin

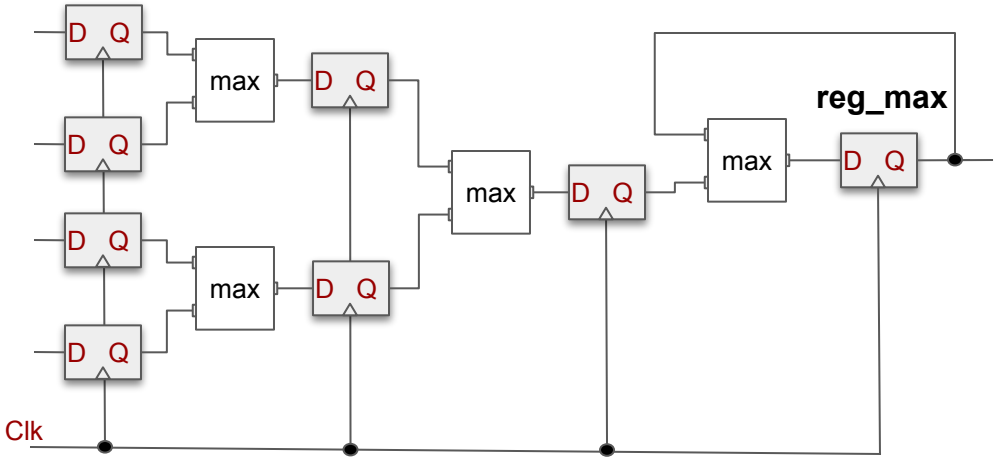
Zřetězené zpracování



T = 2*1+1*2+1 = 5 ns
f = 1000/5 = 200 MHz
Latence = 4 takty hodin

Příklad: Zpracování sady vzorků dat obvodem

- Předpokládejte, že potřebujete zpracovat 8M vzorků dat obvodem na následujícím obrázku. Uvažujte zpoždění každého vodiče t_{wire} = 1 ns, setup time t_{setup} = 1 ns a zpoždění každé sčítačky t_{max} = 7ns
  - Na jaké maximální frekvenci může obvod pracovat?
  - Jakou dobu bude trvat zpracování všech 10M vzorků dat?
  - Za jak dlouho budeme mít k dispozici první výsledek?



T = ... ns

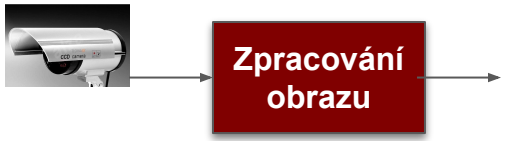
f = ... MHz

t_{8MB} = ... ns

Latence = ... ns

- ***Proudové zpracování dat (nekonečná smyčka)***
  - Vstupní tok dat je tvořen souvislým proudem vzorků, které přicházejí s určitou četnost (frekvence vzorků za sekundu).
  - Data přicházejí bez adresace, po datové sběrnici nejčastěji ve formě rámců.
  - ***Četnost vzorků na vstupu definuje požadovanou propustnost pro zpracování dat***
  - Výstupem je často modifikace vstupu nebo agregovaná, případně vyfiltrovaná data.

Zpracování obrazu



4K, 60fps, 10b ~ 55,6 Gb/s

Síťové aplikace

Síťové linky na rychlosti 40, 100 nebo 400 Gb/s

Osnova

- Proč mapovat algoritmus do hardware
- Mapování základních konstrukcí do hardware
  - Sekvence
  - Selekce
  - Iterace
- ***Modulární návrh***
- Shrnutí

- Procedury a funkce umožňují
  - stavebnicově ***skládat z jednodušších částí kódu složitější konstrukce***,
  - realizovat návrh systému shora dolů a ***vyšší čitelnost kódu***
  - ***opakování volání kódu*** pomocí volání funkcí ***nebo replikaci kódu*** pomocí inline funkcí
  - funkce má právě jednu návratovou hodnotu, procedura spouští akci nebo umožňuje získat více návratových hodnot

Příklad funkce

Absolutní hodnota rozdílu prvků a, b

```
int abs(int a, int b){
    if(a>b)
        return a-b;
    else
        return b-a;
}
```

Standardní volání funkce

```
int abs(int a, int b){
    if(a>b)
        return a-b;
    else
        return b-a;
}
```

```
int main(){
    int i=10,j=5,y1;
    int k=10,l=5,y2;
    y1=abs(i,j);
    y2=abs(k,l);
    return 0;
}
```

Inline funkce po kompilaci

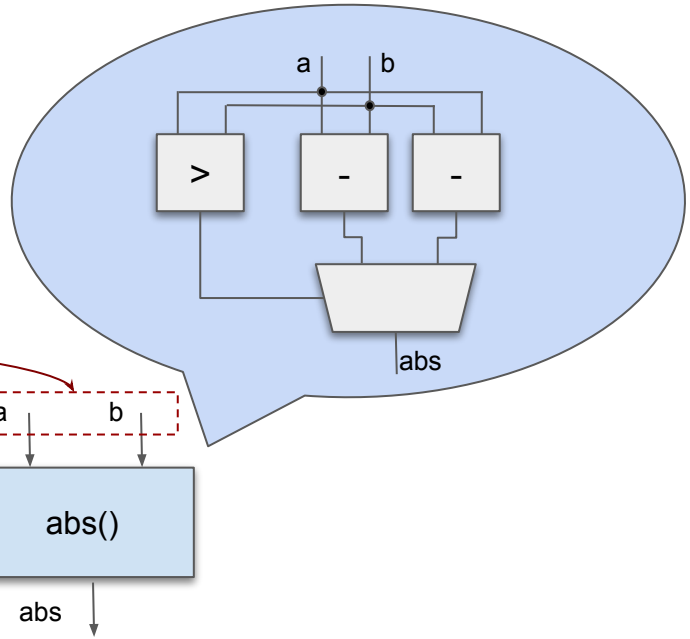
```
int main(){
    int i=10,j=5,y1;
    int k=10,l=5,y2;
    if(i>j)
        y1=i-j;
    else
        y1=j-i;
    if(k>l)
        y2=k-l;
    else
        y2=l-k;
    return 0;
}
```

Inline funkce nejsou volány, ale jsou po kompilaci vloženy do kódu (replikovány)

- Procedury a funkce se mapují na HW komponenty
  - Jednotlivé ***vstupně/výstupní parametry funkce odpovídají vstupům*** a ***případně i výstupům obvodu.***
  - ***Sekvenci příkazů definuje architekturu obvodu***
  - Návratová hodnota funkce odpovídá výstupu hardwarové komponenty

Funkce výpočtu absolutní hodnoty

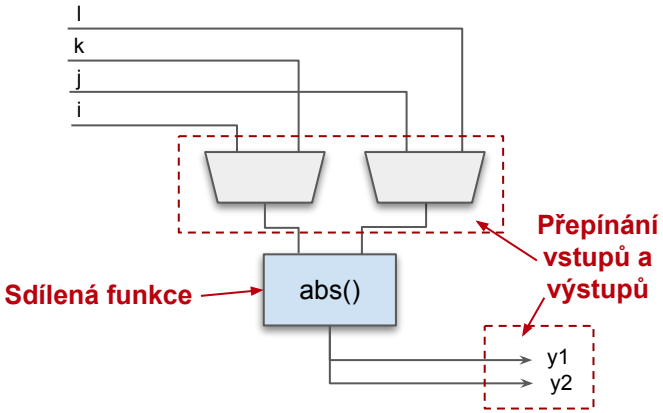
```
int abs(int a, int b){
    if(a>b)
        return a-b;
    else
        return b-a;
}
```



HW komponenta realizující funkci výpočtu absolutní hodnoty

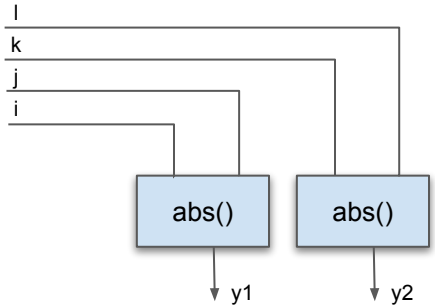
Mapování funkcí do hardware

Sdílení komponent
(Standardní volání)




```
int abs(int a, int b){
    if(a>b)
        return a-b;
    else
        return b-a;
}

int main(){
    int i=10,j=5,y1;
    int k=10,l=5,y2;
    y1=abs(i,j);
    y2=abs(k,l);
    return 0;
}
```

Replikace komponent
(Inline funkce)



- Nelze spustit funkci současně ze dvou míst → je nutné zajistit vzájemné vyloučení.
- Přepínání vstupů z míst volání funkce
- Propojení výstupu do místa volání funkce
- ***Šetří hardwarové zdroje, ale s počtem míst volání roste složitost propojení***

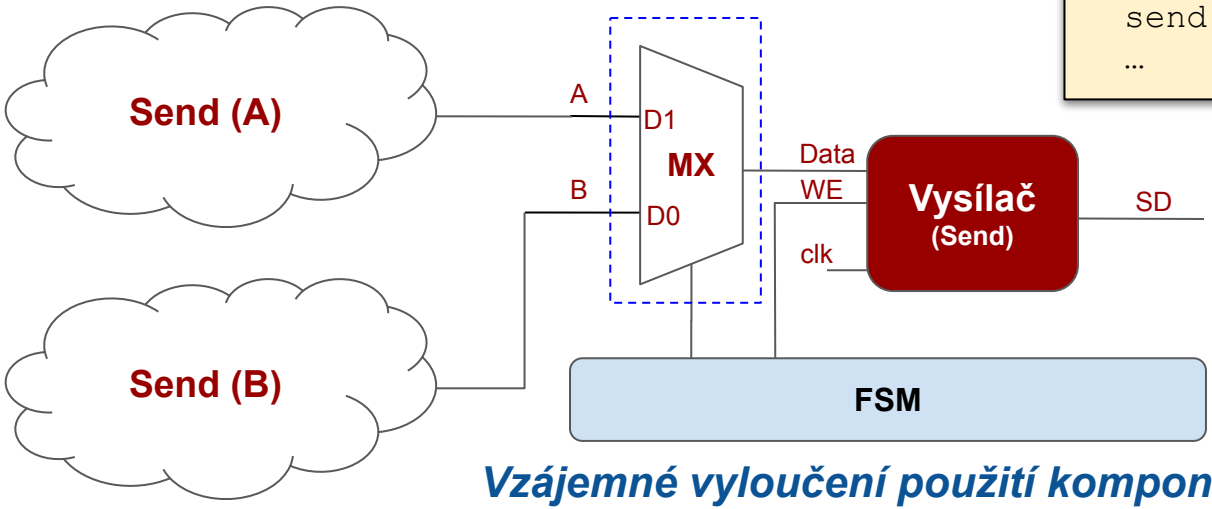
- Vložení komponenty do HW architektury a napojení na vstupní a výstupní signály
- Umožňuje paralelní zpracování
- ***S každou replikovanou komponentou narůstá spotřeba hardwarových zdrojů.***

- Jak přepínat vstupy a výstupy sdílené komponenty?
  - Multiplexory nebo třístavové sběrnice

Příklad

Odesílání dat po sériové lince

Přepínání pomocí multiplexoru




```
…
send(A);
…
send(B);
…
```

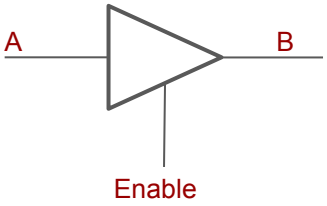
Sdílení komponenty sice šetří hardwarové zdroje, ale s počtem míst volání roste složitost propojení, počet použitých multiplexorů → obvod pak musí pracovat na nižší frekvenci

Vzájemné vyloučení použití komponenty

(časový multiplex)

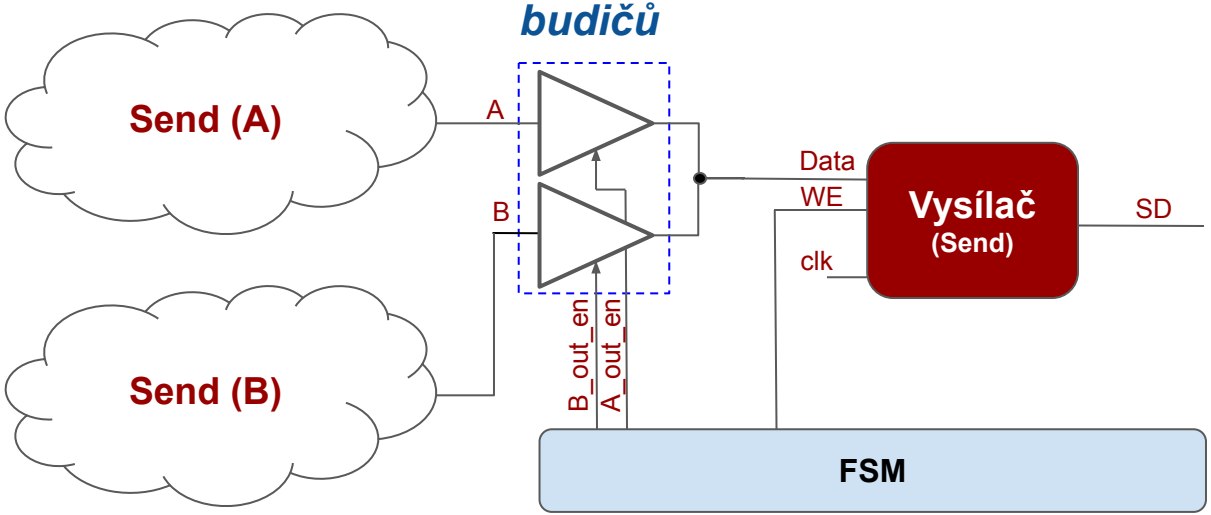
- Realizace pomocí třístavových budičů.

Třístavový budič



Enable	A	B
0	0	Z
0	1	Z
1	0	0
1	1	1

Přepínání vstupů pomocí třístavových budičů



Vzájemné vyloučení použití komponenty

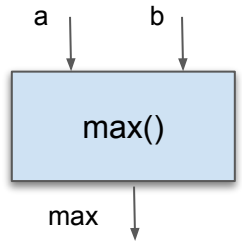
(časový multiplex)

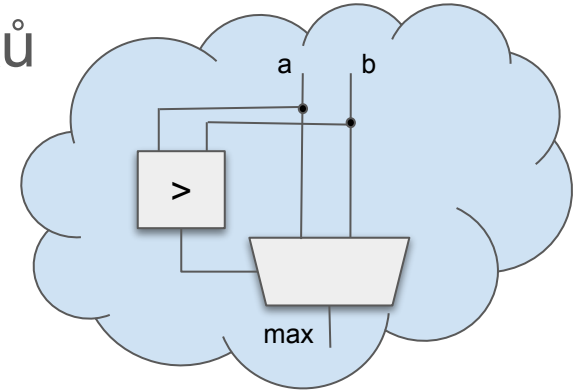
Příklad využití funkcí (replikace komponent)

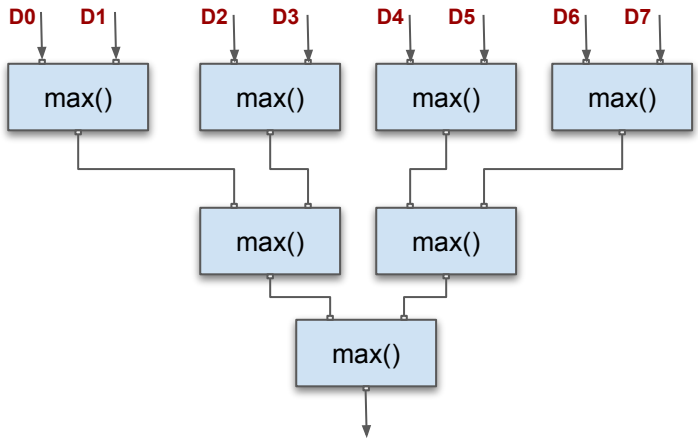
- Výběr maxima z posloupnosti 8 prvků

Funkce výpočtu maxima

```
int max(int a, int b){
    if(a>b)
        return a;
    else
        return b;
}
```







Replikace jednotek umožňuje zpracovat osm prvků posloupnosti za $T = 3 * t_{max}$

V případě ***sdílení jedne funkce*** max() by se zpracovalo všech osm prvků posloupnosti za $T = 7 * t_{max}$

Osnova

- Proč mapovat algoritmus do hardware
- Mapování základních konstrukcí do hardware
  - Sekvence
  - Selekce
  - Iterace
- Modulární návrh
- ***Shrnutí***

Shrnutí

- Libovolný algoritmus vytvořený na základě ***sekvence***, ***selekce*** a ***iterace*** lze rovněž realizovat na úrovni hardware
- Srovnání HW s implementací algoritmu v software
  - Hardwarové zpracování nabízí vyšší potenciál k paralelizaci výpočtu
  - Při návrhu obvodu lze využít různých technik akcelerace např. ***zřetězení***, ***rozbalení smyček*** nebo jejich kombinaci – vede na ***velké množství potenciálních realizací*** s různou dobou výpočtu a množstvím požadovaných výpočetních zdrojů
  - Návrhář aplikace však musí mnohem více zohlednit ***časové aspekty obvodu***, což výrazně zvyšuje složitost návrhu
- Složitější obvody je možné skládat z jednodušších
  - Podobně jako se používají pro strukturované programování procedury a funkce, je možné v hardware použít ke skládání složitých obvodu vytvořené komponenty.
  - Komponenty je možné vkládat samostatně nebo je sdílet, a tak optimalizovat výsledný obvod buď na rychlost zpracování nebo na použité hardwarové zdroje.

Příklad návrhu obvodu

- Navrhněte obvod, který vybere maximální a minimální prvek z N-prvkového pole uloženého v paměti.
  - Kolik taktů hodin bude trvat zpracování 1024 prvků pole?
  - Jaká bude maximální frekvence obvodů, pokud zpoždění všech použitých komponent je shodně 2 ns, zpoždění na každém vodiči shodně 1 ns a setup time registrů je také 1 ns?
  - Zamyslete se nad různými způsoby urychlení při zpracování v hardware.

Konec přednášky

Děkuji za pozornost