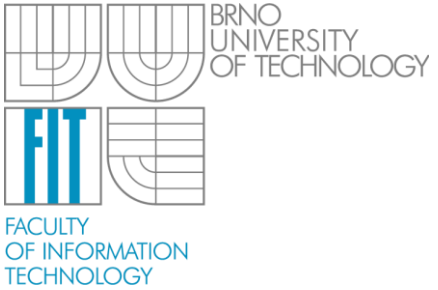
CatapultC

Tomáš Martínek

Vysoké učení technické v Brně, Fakulta informačních technologií v Brně
Božetěchova 2, 612 66 Brno
ant@fit.vutbr.cz



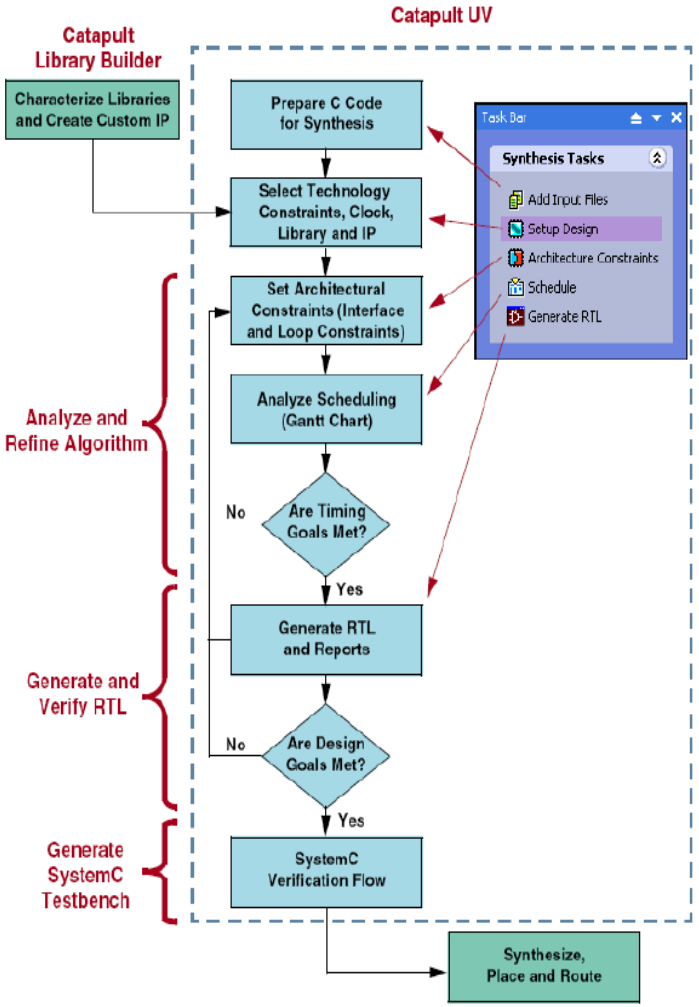
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Osnova

- Základní schéma návrhu
- Datové typy s bitovou přesností
- Syntéza smyček
- Mapování rozhranní a pamětí
- Shrnutí

Základní schéma návrhu

- Postup při vytváření obvodů
  1. Příprava zdrojových kódů v jazyce C/C++
  2. Výběr technologie
  3. Nastavení omezujících podmínek
  4. Plánování
  5. Generování RTL
  6. Verifikace RTL

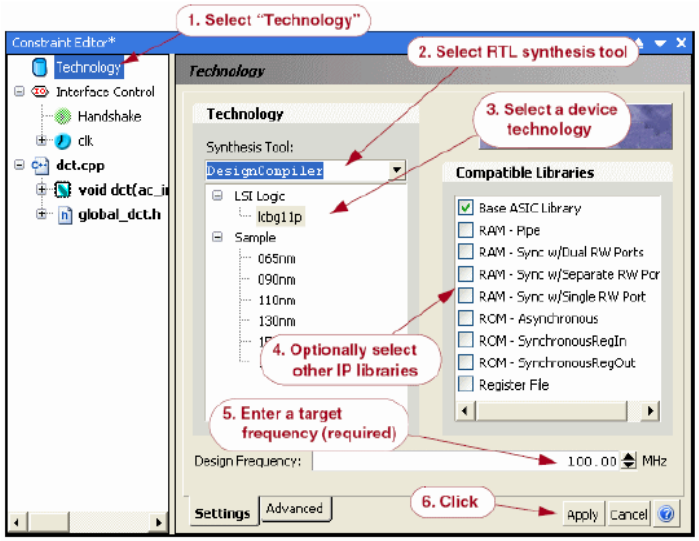


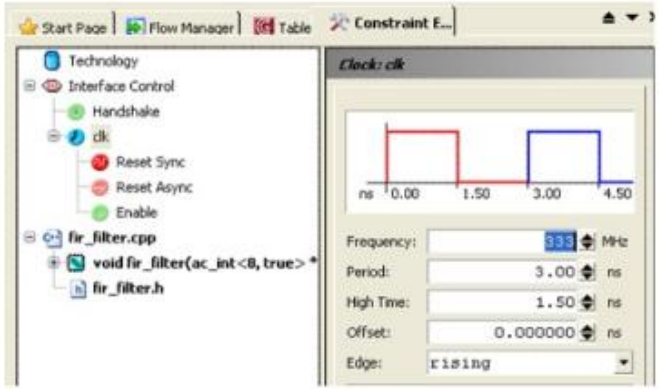
Příprava vstupních souborů

- Popis algoritmu, který by měl být realizován pomocí hardwarového obvodu se očekává v jazyce ANSI C/C++
- Zdrojový kód neobsahuje žádnou informaci o hodinovém signálu a rozdělení výpočtu do jednotlivých kroků
- Popisovaný program lze zkompilovat na běžně dostupných kompilátorech (GCC, Microsoft Visual C++) a lze dokonce spustit na běžném procesoru
- Jazyk C++ lze efektivně použít např. pro konstrukci objektů nebo šablon generických obvodů pracujících s různou datovou šířkou/datovými typy (posuvný registr, apod.).
- Některé konstrukce nejsou syntézou podporovány, např.:
  - Dynamická alokace paměti
  - Union struktury
  - Datové typy float a double (nahrazeno novými typy ac_fixed)
  - Rekurze s neznámou hloubkou zanoření

Výběr technologie

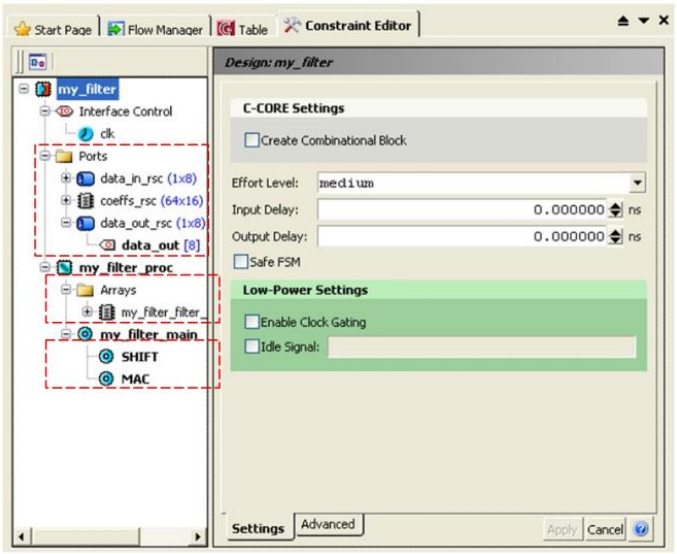
- Výběr cílové technologie
  - ASIC, FPGA
- Výběr nástroje generování RTL
  - DesignCompliler, Precision, apod.
- Výběr volitelných knihoven prvků
  - Paměťové bloky RAM, ROM
- Nastavení požadované hodinové frekvence výsledného obvodu
- Nastavení podrobnějších vlastností hodinového signálu a signálu Reset
  - CLK: fázový posun, rozložení 0/1
  - RESET: synchronní/asynchronní

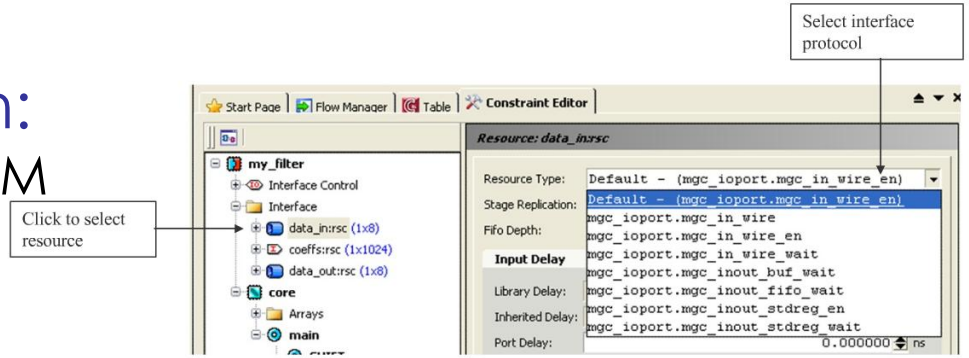




Nastavení omezujících podmínek

- Cíl syntézy:
  - Latence vs. Plocha
  - Spotřeba
- Optimalizace smyček:
  - Rozbalení
  - Zřetězení
- Mapování rozhraní:
  - Běžný vodič
  - Vodič s EN signálem
  - Handshake protokol
  - Paměťové rozhraní
- Mapování proměnných:
  - Pole do paměti RAM/ROM

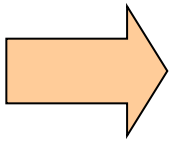


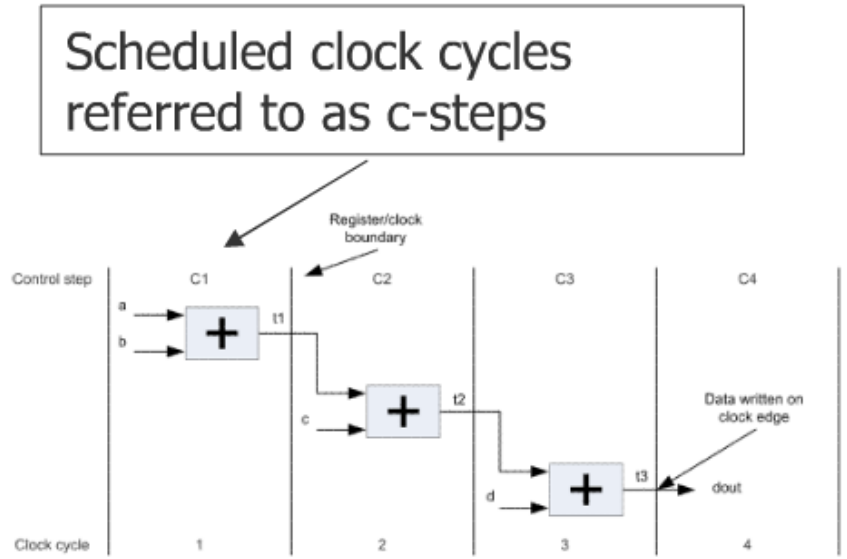


Plánování

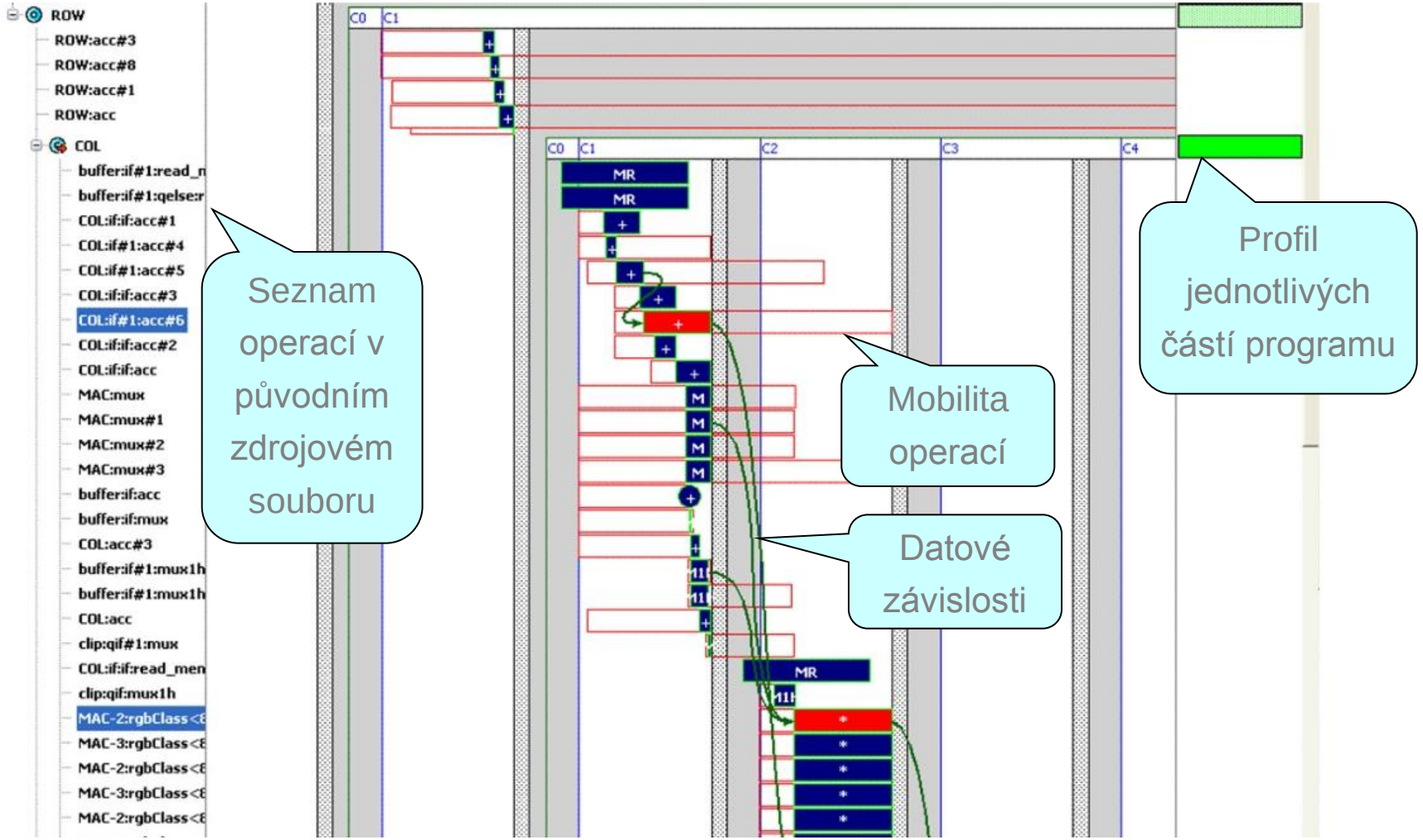
- Na základě zadaných omezujících podmínek je vytvořen plán výpočtu – tj. rozdělení do kontrolních kroků (C-steps)
- Při plánování se využívá vlastností prvků cílové technologie – různý poměr latence vs. množství zdrojů
- Lze mapovat více operací do jednoho kontrolního kroku popř. použít jejich zřetězené varianty
- Výstupem je Ganttův diagram

```
void accumulate(int a, int b, int c, int d,
                int &dout){
  int t1,t2;
  t1 = a + b;
  t2 = t1 + c;
  dout = t2 + d;
}
```



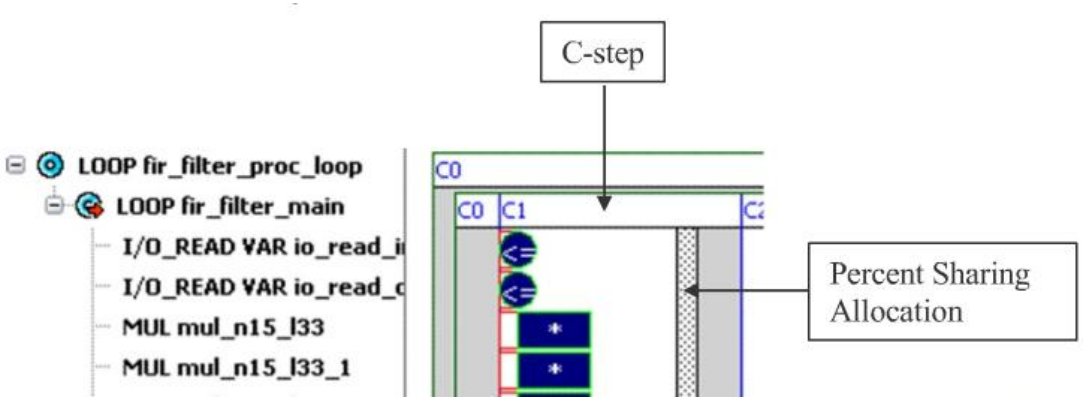


Plánování – Ganttův diagram



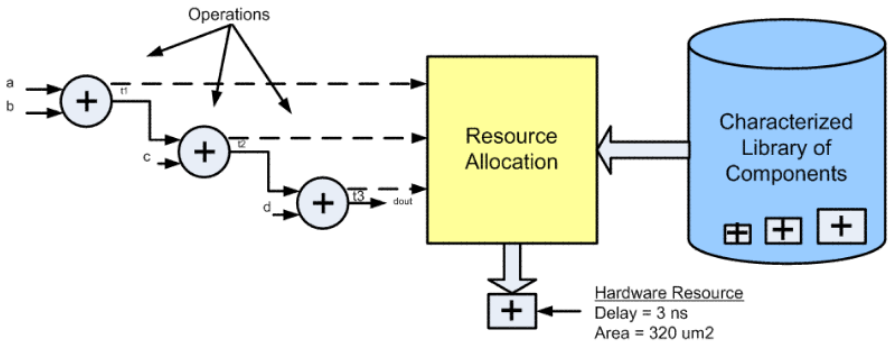
Plánování – C-step

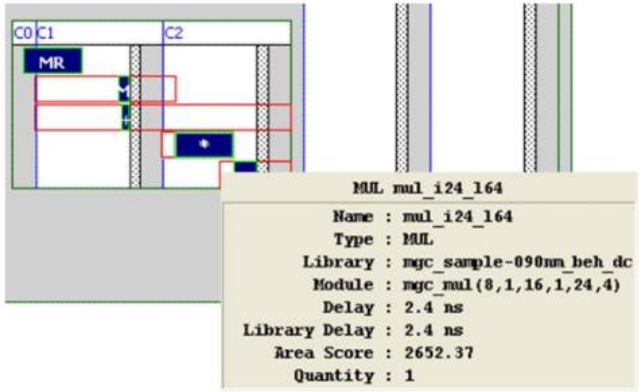
- Co je to „C-step“?
  - Odpovídá stavu konečného automatu
  - C-steps mohou vytvářejí hierarchickou strukturu (např. při zanoření smyček)
  - Na nejnižší úrovni nesmí být kratší než takt hodinového signálu
- C-step zahrnuje:
  - množinu naplánovaných operací
  - režii spojenou s implementaci stavu automatu, implicitně nastavena na 20% u ASIC a 40% u FPGA (routování)



Plánování – Výběr komponent

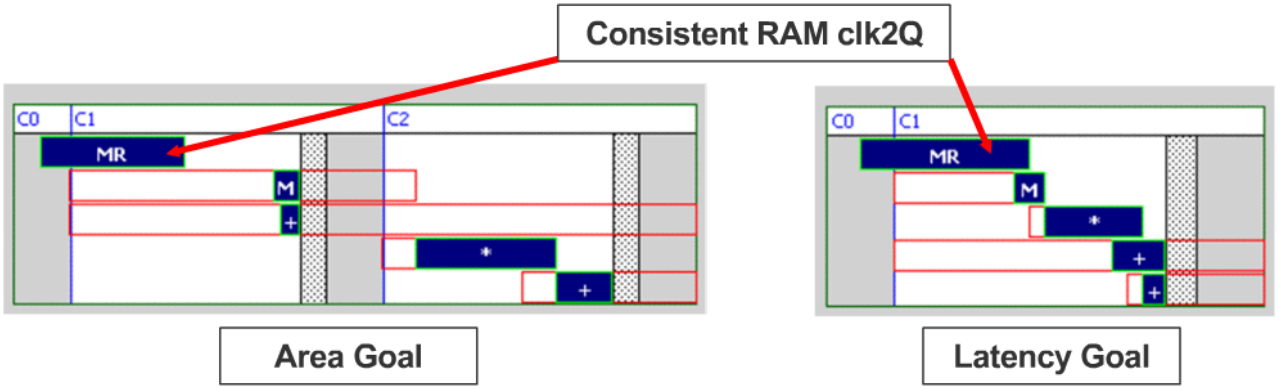
- S ohledem na cíle plánování jsou vybírány z knihovny implementace komponent s různým poměrem plocha/zpoždění
- ASIC: obvykle 4 implementace/operaci
- FPGA: obvykle 1 implementace/operaci





Plánování – Cíle plánování

- Plocha
  - Používá komponenty s nejmenší plochou za cenu vyšší latence
- Latence
  - Používá komponenty s nejmenší latencí za cenu vyšší plochy
- Plocha + omezení na maximální latenci
  - Nesmí přesáhnout zadanou latenci a snaží se dosáhnout co nejmenší plochy = Časově omezené plánování
- Latence + omezení na maximální plochu
  - Nesmí přesáhnout zadanou plochu a snaží se dosáhnout co nejmenší latence = Prostorově omezené plánování



Generování RTL

- Pokud vyhovuje vytvořený plán s ohledem na latenci a odhadované množství zdrojů je vygenerováno výsledné RTL schéma
- Zahrnuje:
  - Alokaci zdrojů
  - Generování FSM
  - Vytvoření schémat
  - Výstupní reporty zahrnující přesnou informaci o množství zdrojů a doby výpočtu, včetně seznamu kritických cest
- Podporuje řadu výstupů (VHDL, Verilog, Netlist, apod.)

Výstupní RTL schéma a další reporty

Start Page | Flow Manager | Table | Constraint Editor | Schedule | rtl.rpt

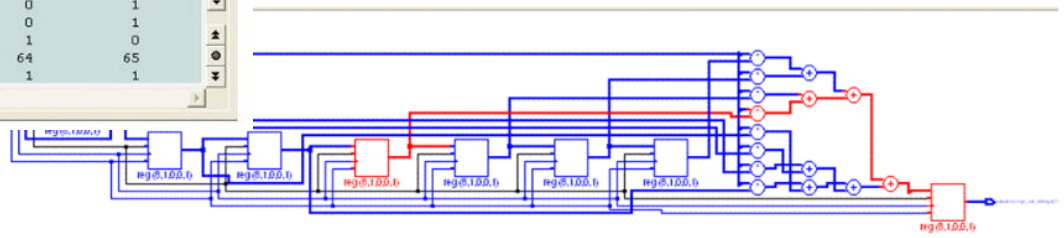
```
Solution Settings: my_filter.v2
  Current state: extract
  Project: Catapult

Design Input Files Specified

Processes/Blocks in Design

Bill Of Materials (Datapath)
  Component Name                          Area Score  Delay Post Alloc Post Assign
  --------------------------------------- ---------- ------ ---------- -----------
  [Lib: mgc_ioport]
  mgc_in_wire_en(2,8)                          0.000  0.000          1           1
  mgc_out_stdreg_en(4,8)                       0.000  0.000          1           1
  [Lib: mgc_sample-090nm_beh_dc]
  mgc_add(15,1,1,0,16,3)                     170.718  0.594          1           0
  mgc_add(15,1,1,0,16,4)                     141.969  1.079          0           1
  mgc_add(30,0,24,1,30,4)                    522.215  2.778          1           1
  mgc_add(6,0,1,1,7,4)                       102.578  0.766          1           1
  mgc_and(1,2,4)                               3.131  0.031          0           1
  mgc_and(1,3,4)                               4.715  0.054          0           1
  mgc_and(30,2,4)                             93.937  0.031          1           0
  mgc_and(30,3,4)                            141.439  0.054          0           1
  mgc_and(6,2,4)                              18.787  0.031          0           1
  mgc_and(8,2,4)                              25.050  0.031          0          63
  mgc_mul(8,1,16,1,24,4)                    2652.370  2.435          1           1
  mgc_mux(1,1,2,4)                             5.716  0.133          0           1
  mgc_mux(6,1,2,4)                            34.295  0.133          0           1
  mgc_mux(7,1,2,4)                            40.011  0.133          1           0
  mgc_mux(8,1,2,4)                            45.727  0.133         64          65
  mgc_mux(8,6,64,4)                         1835.141  0.453          1           1
```

Bill of Materials
Podrobný seznam spotřebovaných zdrojů pro jednotlivé operace



RTL schéma a výpis kritických cest

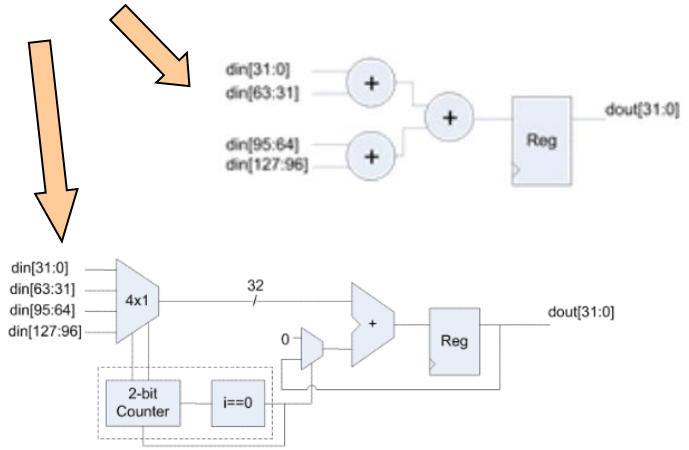
Path	Start Point	End Point	Delay	Slack
Path 1	fir_filter:core...	fir_filter:core...	2.8995	0.1035
Path 2	fir_filter:core...	fir_filter:core...	2.8995	0.1035
Path 3	fir_filter:core...	fir_filter:core...	2.8995	0.1035
Path 4	fir_filter:core...	fir_filter:core...	2.8995	0.1035

```
Path Start Point
5    fir_filter:core/reg(MAC:asn#3.dft)
     Instance
     fir_filter:core/reg(MAC:asn#3.dft)
     fir_filter:core/MAC:asn#3.dft
     fir_filter:core/MAC-4:mul
     fir_filter:core/MAC-4:mul.itm
     fir_filter:core/MAC:acc#5
```

Výstupní RTL schéma a další reporty

- Konkrétní realizace na úrovni RTL se označuje jako mikroarchitektura
- Úpravou omezujících podmínek a vytvořením nového plánu lze vytvářet různé mikroarchitektury
- Catapult shromaďuje informace o jednotlivých mikroarchitekturách a umožňuje se mezi nimi snadno přepínat

```
void accumulate4(int din[4], int &dout){
 int acc=0;
 ACCUM:for(int i=0;i<4;i++){
   acc += din[i];
 }
   dout = acc;
}
```



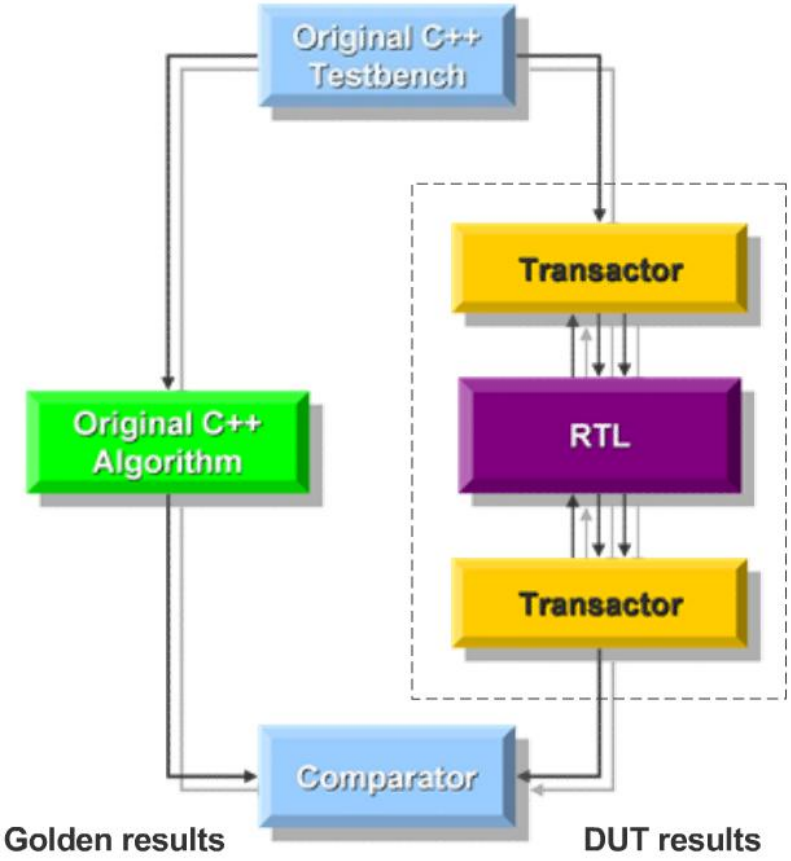
Start Page | Flow Manager | Table | Constraint Editor

Report: General

Solution	Latency Cycles	Latency Time	Throughput Cycles	Throughput Time	Total Area	Slack
my_filter.v1 (extract)	65	650.00	67	670.00	30266.60	2.51
my_filter.v2 (extract)	65	650.00	64	640.00	20823.69	1.41
my_filter.v3 (extract)	34	340.00	32	320.00	24844.10	2.60
my_filter.v4 (extract)	16	160.00	16	160.00	34850.11	1.95
my_filter.v5 (extract)	2	20.00	1	10.00	244390.65	2.91

Verifikace výsledného obvodu

- Pro verifikaci obvodu je potřeba napsat testbench v jazyce C/C++
- Testbench = program, který otestuje požadovanou funkci dostatečnou sadou vstupních hodnot
- Testbench je aplikován na původní zdrojový kód (spuštěný na běžném proceosoru) a současně na vygenerovaný RTL obvod (simulovaný v ModelSIMu)
- CatapultC automaticky zajistí vytvoření rozhraní mezi testbench souborem a prostředím ModelSIM
- Výsledky obou testů jsou porovnány a v případě neshody je chyba dohledána a zobrazena na časovém diagramu ModelSIMu



Osnova

- Základní schéma návrhu
- Datové typy s bitovou přesností
- Syntéza smyček
- Mapování rozhranní a pamětí
- Shrnutí

Datové typy s bitovou přesností

- Proč datové typy s bitovou přesností?
  - Na úrovni hardware se počítá každý bit
  - Aby bylo možné porovnat, zda je implementace na RTL úrovni shodná s původním algoritmem v C++
- Vychází se z knihovny Algorithmic C
  - Podporuje jak celočíselné typy, tak typy s pevnou řádovou čárkou
  - Efektivní implementace – vede na rychlou simulaci i na úrovni jazyka C++

Celočíselné datové typy

- Celočíselný datový typ `ac_int` vytváří bitový vektor zadané délky
- Je k dispospozici v podobě šablony (template), která umožňuje nastavit nejen datovou šířku, ale i rozlišit znaménkové/neznaménkové hodnoty
- Pro použití: `#include <ac_int.h>`
- Je potřeba si hlídat možné přetečení!

Neznaménkový	Znaménkový

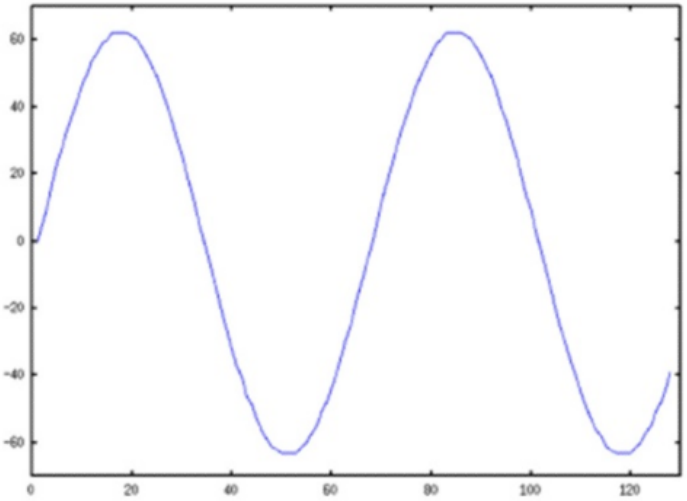
`ac_int<W, false>`
 	
Rozsah:
$0 <= x <= 2^W-1$
 	
Příklad: 10 bitů neznaménkově
`ac_int<10,false>`
	`ac_int<W, true>`
	
	Rozsah:
	$-2^{W-1} <= x <= 2^{W-1}-1$
	
	Příklad: 10 bitů znaménkově
	`ac_int<10,true>`

Celočíselné datové typy

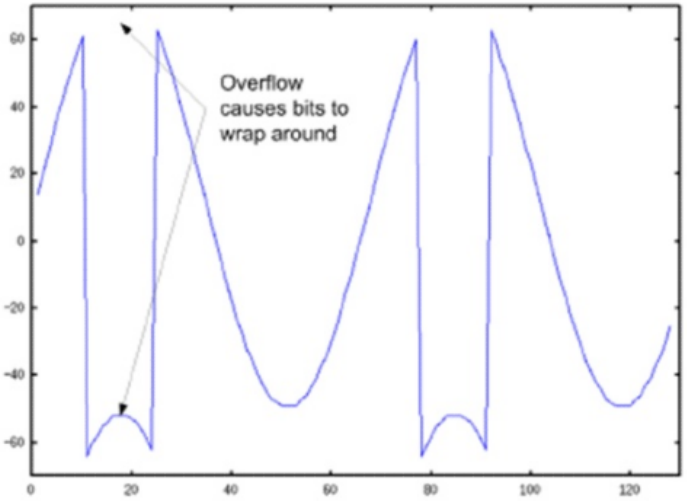
```
const double pi = 3.14;
int main(){
  fstream fptr;
  fptr.open("tmp.txt", fstream::out);
  ac_int<7,true> x[128];
  for(int i=0;i<128;i++){
    x[i] = OFFSET + 63*sin(2*pi*i/64);
    fptr << x[i] <<endl;
  }
  fptr.close();
}
```

7-bit signed: -64 < x < 63

OFFSET = 0;



OFFSET = 14;



S pevnou řádovou čárkou

- Datový typ `ac_fixed` reprezentuje bitový vektor se zadanou pozicí desetinné čárky
- Skrze šablonu lze volit datovou šířku celočíselné i desetinné části včetně znaménkovosti

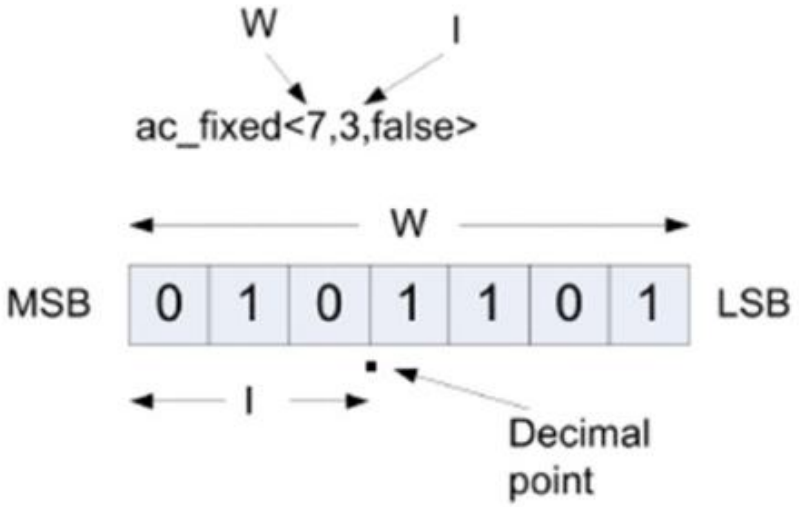
Deklarace:

```
ac_fixed<W,I,S> x;
```

`W` – celková datová šířka
`I` – datová šířka celočíselné části
`S` – znaménkovost (true, false)

Rozsah:

$0 <= x <= (1-2^{-W})2^{I}$



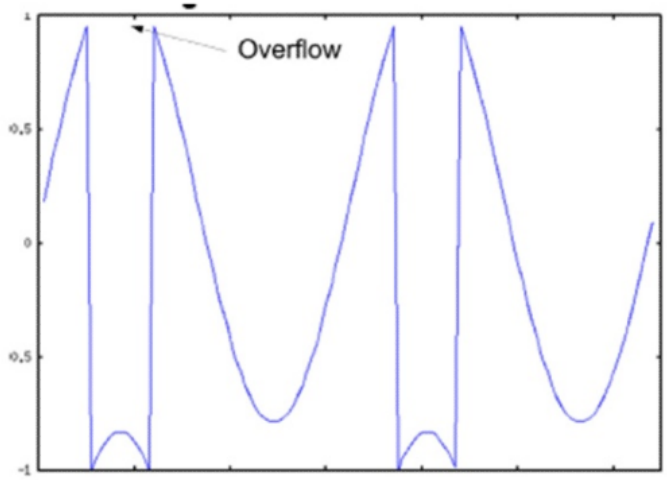
S pevnou řádovou čárkou

- Volitelné parametry datovéo typu `ac_fixed`:
  - `ac_fixed<W,I,S,Q,O>`
  - kde `Q` je režim a `O` je způsob ošetření přetečení
- Kvantizace
  - Způsob ošetření ztráty bitů napravo od LSB
    - `AC_TRN`: useknutí (implicitní chování)
    - `AC_RND`: zaokrouhlení
- Přetečení
  - Způsob ošetření při hodnoty mimo rozsah
    - `AC_WRAP`: přetečení nebo podtečení (implicitní chování)
    - `AC_SAT`: saturace, tj. posun na hraniční hodnotu

Saturace

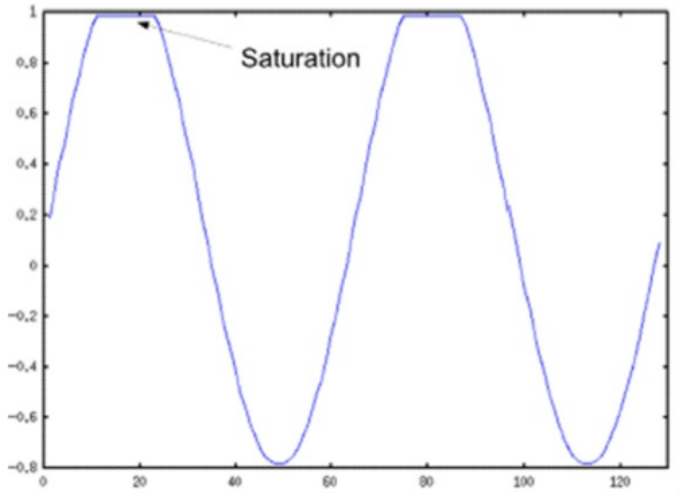
```
#include <ac_fixed.h>
const double pi = 3.14;
const double OFFSET = 0.2;
int main(){
 fstream fptr;
 fptr.open("tmp.txt", fstream::out);
 ac_fixed<7,1,true,AC_TRN,AC_WRAP> x[128];

 for(int i=0;i<128;i++){
   x[i] = OFFSET + 0.98*sin(2*pi*i/(double)64);
   fptr << x[i] <<endl;
 }
 fptr.close();
}
```




```
#include <ac_fixed.h>
const double pi = 3.14;
const double OFFSET = 0.2;
int main(){
 fstream fptr;
 fptr.open("tmp.txt", fstream::out);
 ac_fixed<7,1,true,AC_TRN,AC_SAT> x[128];

 for(int i=0;i<128;i++){
   x[i] = OFFSET + 0.98*sin(2*pi*i/(double)64);
   fptr << x[i] <<endl;
 }
 fptr.close();
}
```



Poznámky a doporučení

- Operace zaokrouhlování a saturace vyžadují výpočetní zdroje navíc – měli by se používat obezřetně v
  - Systémy odolné proti poruchám
  - Komunikačních systémech
  - Zpracování videa (přetečení/podtečení barvy pixelu)
- Knihovna Algorithmic C implementuje na datovými typy všechny běžně používané operace (součet, rozdíl, násobení, apod.)
- Výrazným způsobem šetří návrháři práci a zabraňuje častým chybám, které vznikají v případech, kdy si návrháři vytváření tuto knihovnu sami

Pomocné operace a metody

- Operátor pro výběr bitu []
  - `y[k] = x[i];`
- Metoda pro čtení více bitů
  - `slc<W>(int lsb)`
  - `x = y.slc<2>(5)` odpovídá `x:=y(6 downto 5)`
- Metoda pro zápis více bitů
  - `set_slc(int lsb, ac_int &slc)`
  - `x.set_slc(7, y);`
- Pomocné metody pro konverzi
  - `.to_int, .to_uint, .to_double, .to_string`

Osnova

- Základní schéma návrhu
- Datové typy s bitovou přesností
- Syntéza smyček
- Mapování rozhranní a pamětí
- Shrnutí

Smyčky

- Jedna z konstrukcí, která může výrazným způsobem ovlivnit výsledek syntézy
  - Techniky rozbalovaní, zřetězení nebo spojování smyček
- Typy podporovaných smyček
  - `for, while, do while`
- Každá smyčka obsahuje 4 části:
  1. Inicializaci
  2. Testovací podmínku
  3. Inkrement
  4. Seznam příkazů
- Příklad syntaxe pro smyčku `for`:

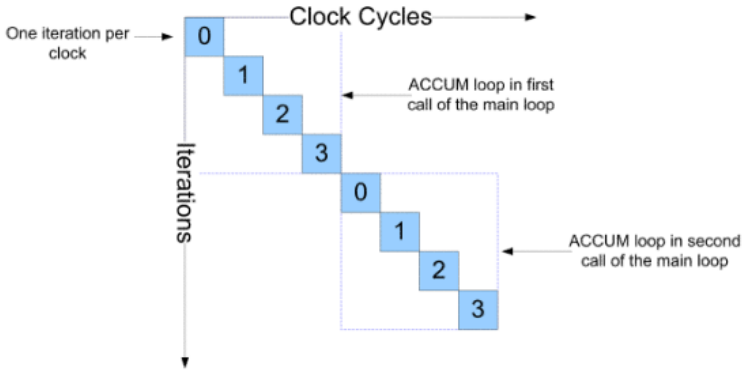
```
LABEL: for( initialization; test-
condition; increment ) {
        statement-list or loop body;
}
```

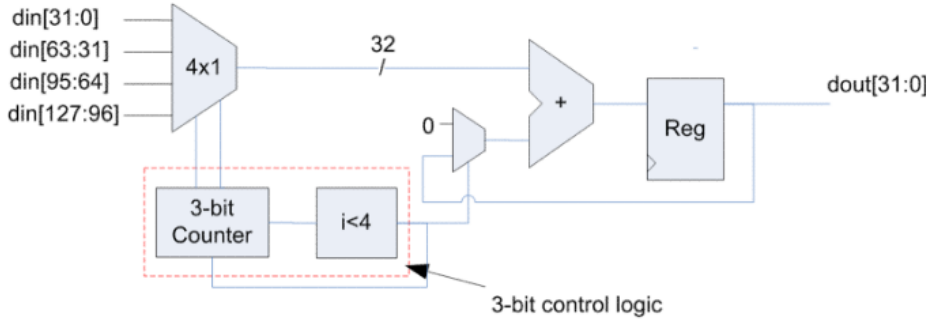
Nerozbalené smyčky

- Všechny smyčky programu jsou implicitně nerozbalené – tělo smyčky je prováděno sekvenčně
- Výpočet těla trvá vždy nejméně jeden takt hodinového signálu

```
void accumulate(int din[4], int &dout){
  int acc=0;
  ACCUM:for(int i=0;i<4;i++){
    acc += din[i];
    <Implied wait-until-clock>
  }
  dout = acc;
}
```

Design Constraints
All loops left rolled



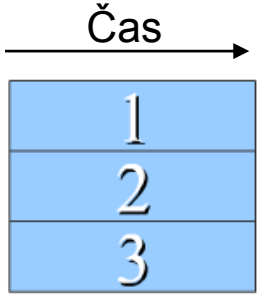


Rozbalování smyček (Loop Unrolling)

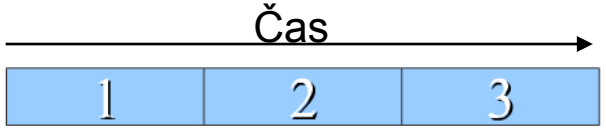
- Rozbalování smyček je jeden z nejefektivnějších způsobů, jak zvýšit stupeň paralelizmu
- Výsledný stupeň paralelizmu však mohou výrazně ovlivnit datové závislosti

Rozbalení těla smyčky do několika kopií

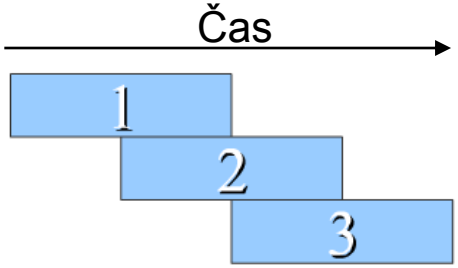
Datové závislosti nejsou ve smyčce přítomny – všechny těla smyček lze provádět paralelně



Vzhledem k datovým závislostem nelze smyčku rozbalit



Obvykle je výsledek někde mezi

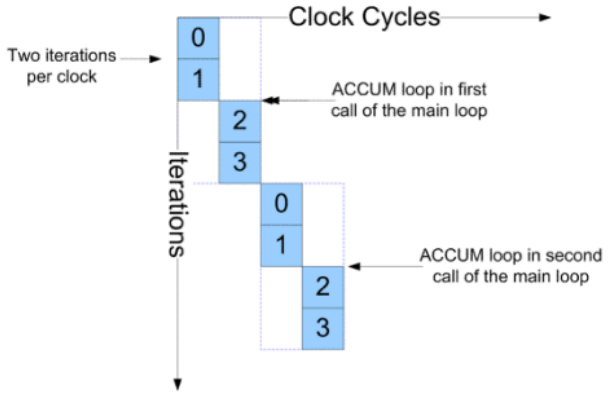


Částečné rozbalení smyčky

- Příklad: dvojnásobné rozbalení smyčky – replikuje tělo smyčky dvakrát a snaží se jej vykonat paralelně

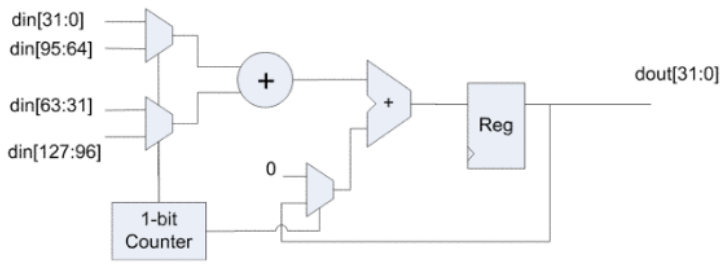
Design Constraints
Clock frequency slow enough to ignore dependencies
Unroll by two

```
void accumulate(int din[4], int &dout){
  int acc=0;
  ACCUM:for(int i=0;i<4;i++){
    acc += din[i];
  }
  dout = acc;
}
```



Manual unroll by two

```
void accumulate(int din[4], int &dout){
  int acc=0;
  ACCUM:for(int i=0;i<4;i+=2){
    acc += din[i];
    acc += din[i+1];
  }
  dout = acc;
}
```

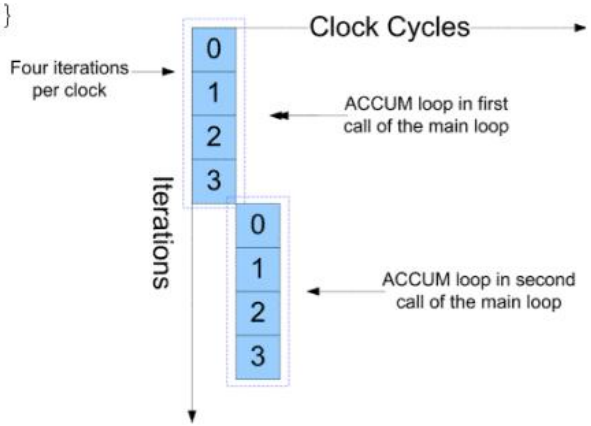


Úplné rozbalení smyčky

- Příklad: Úplné rozbalení smyčky – všechny 4 iterace jsou rozbaleny a naplánovány současně

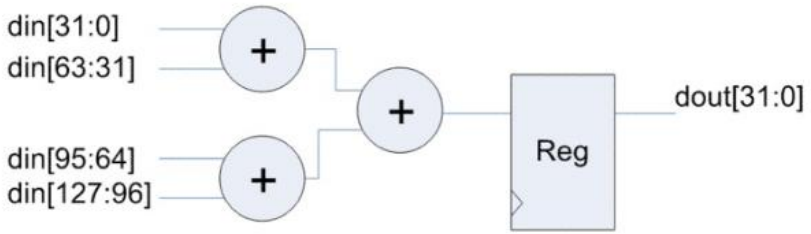
Design Constraints
Clock frequency slow enough to ignore dependencies
Unroll fully

```
void accumulate(int din[4], int &dout){
  int acc=0;
  ACCUM:for(int i=0;i<4;i++){
    acc += din[i];
  }
  dout = acc;
}
```



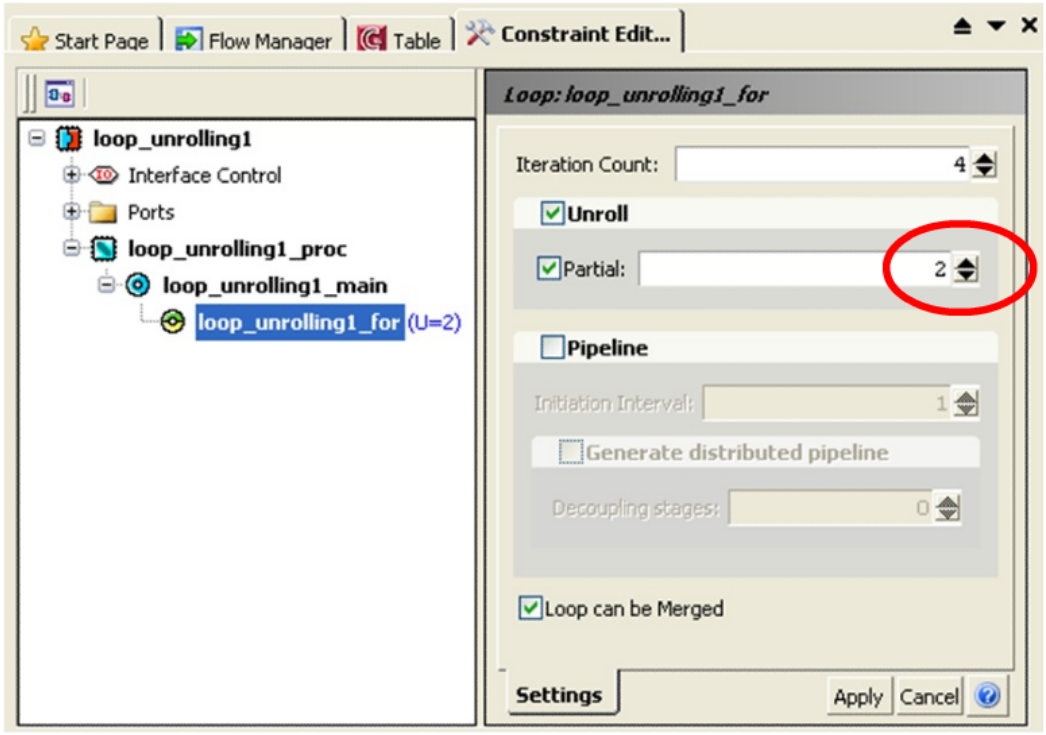
Manual unroll fully

```
void accumulate(int din[4], int &dout){
    int acc=0;
    acc += din[0];
    acc += din[1];
    acc += din[2];
    acc += din[3];
    dout = acc;
}
```



Rozbalovaní smyček - Nastavení

- U každé smyčky lze individuálně nastavit zda má:
  - Zůstat nerozbalena,
  - Zcela se rozbalit
  - Rozbalit se jen částečně



Rozbalování smyček – pravidla

- Počet iterací musí být předem znám
- Při rozbalování vnořených smyček je vhodné začít od nejvnitřnějších
- Rozbalovat s ohledem na dostupnou vstupně/výstupní propustnost
- Smyčky vhodné rozbalení:
  - Malý počet iterací
  - Malý počet operací
  - S minimem zpětných vazeb
  - S minimem nedostatkových zdrojů

Zřetězení smyček (Loop Pipelining)

- Zřetězení smyček umožňuje vykonávat následující iteraci ještě dříve, než skončí provádění předchozí iterace
- Zřetězení umožňuje překrytí po sobě jdoucích iterací smyčky a zvyšuje výkonnost obvodu
- Jako smyčku lze chápat i hlavní program (Main), který reprezentuje obvod běžící v nekonečné smyčce
- Příklad:

Main loop →

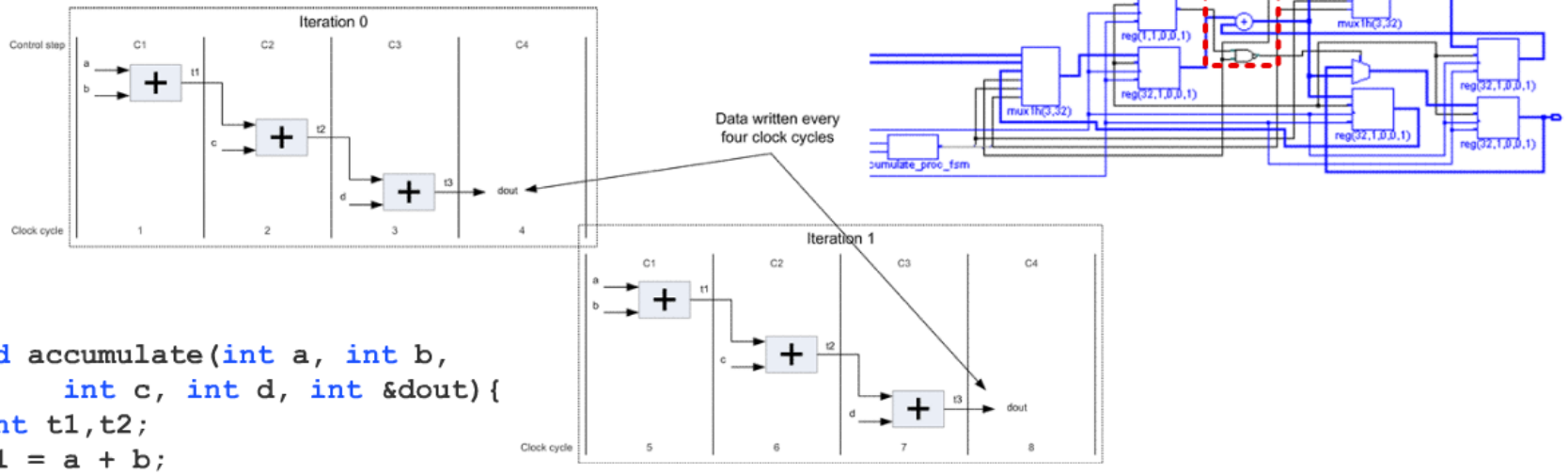
```
void accumulate(int a, int b, int c, int d, int &dout){
  int t1,t2;
  t1 = a + b;
  t2 = t1 + c;
  dout = t2 + d;
}
```

Zřetězení smyček - Definice

- Počáteční interval (Initialization Interval - II)
  - Vyjadřuje počet taktů před spuštěním následující iterace
- Latence (Latency)
  - Vyjadřuje počet taktů od začátku výpočtu po získání prvního výstupu
- Propustnost (Throughput)
  - Vyjadřuje četnost (v počtech taktů) s jakou mohou být generovány výstupy

Zřetězení smyček - Příklad

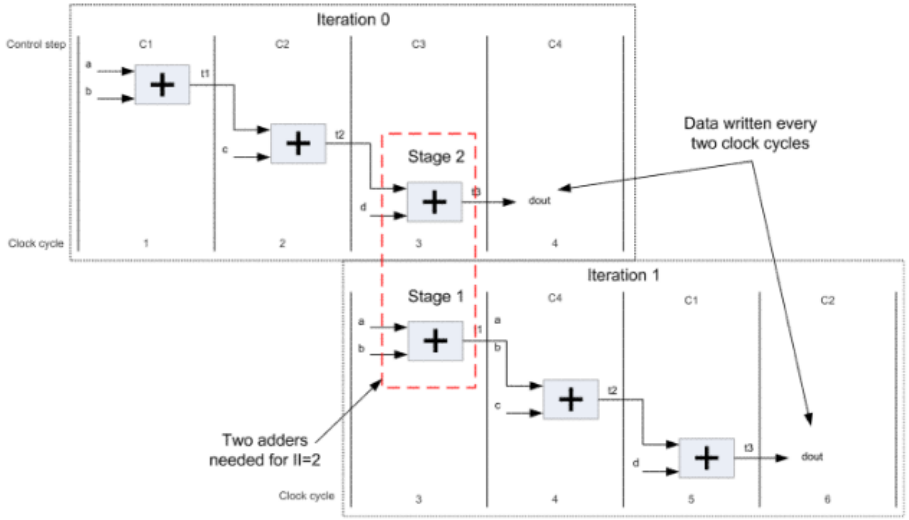
- Implicitně není smyčka zřetězena
- Tělo smyčky je ukončeno po čtyřech taktech (1 takt režie)
- Až po ukončení celé iterace se začne provádět další iterace
- Pro implementaci postačuje jedna sčítačka
- Latence = 3, Propustnost = 4

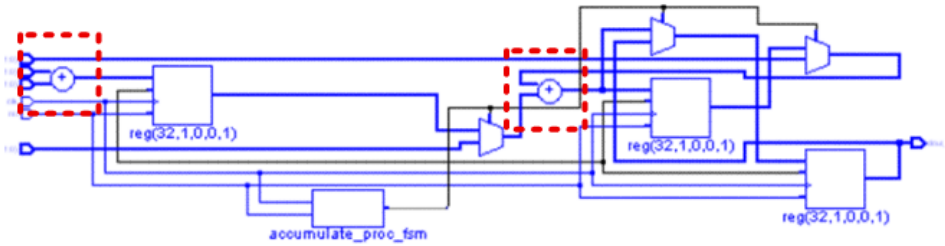



```
void accumulate(int a, int b,
        int c, int d, int &dout){
  int t1,t2;
  t1 = a + b;
  t2 = t1 + c;
  dout = t2 + d;
}
```

Zřetězení smyček - Příklad

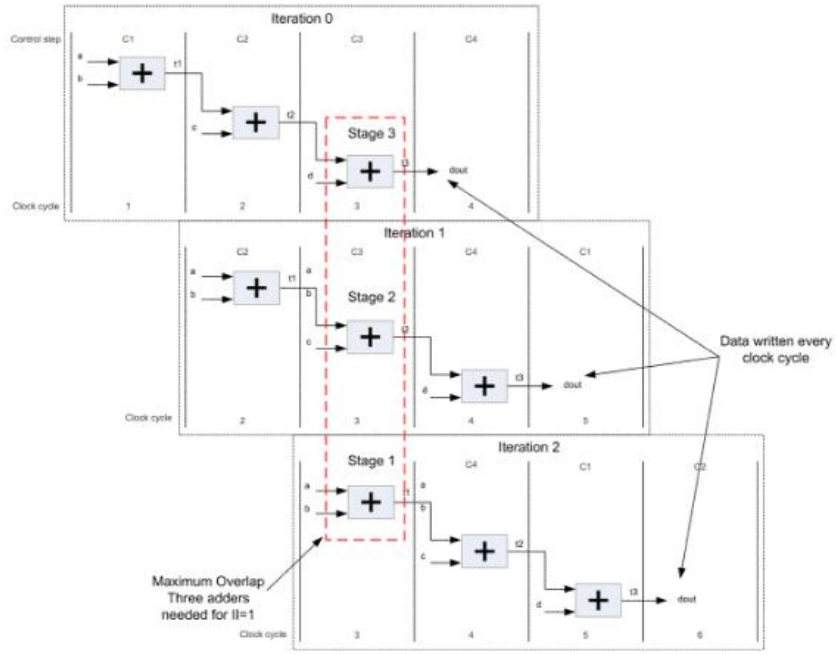
- Nastavením II = 2 jsou nové iterace spouštěny co dva takty hodinového signálu
- Obvod obsahuje dva stupně zřetězení
- V kroku C3 je potřeba provádět dva součty => obvod musí obsahovat dvě sčítačky
- Latence = 3, Propustnost = 2

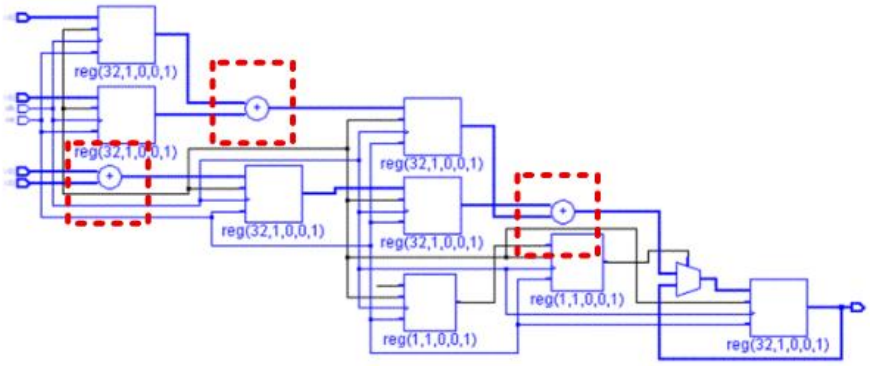




Zřetězení smyček - Příklad

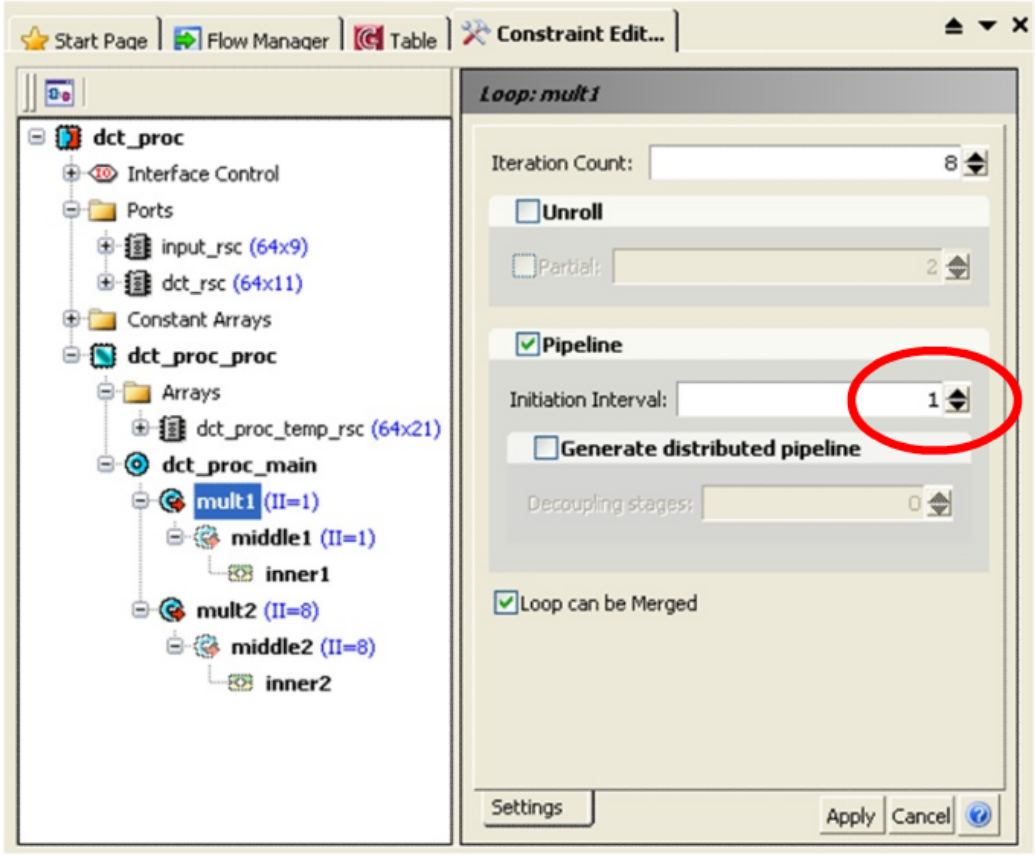
- Nastavením II = 1 jsou nové iterace spouštěny každý takt
- Obvod obsahuje tři stupně zřetězení
- V kroku C3 je potřeba provádět až tři součty => obvod musí obsahovat tři sčítačky sčítačky
- Latence = 3, Propustnost = 1





Zřetězení smyček - Nastavení

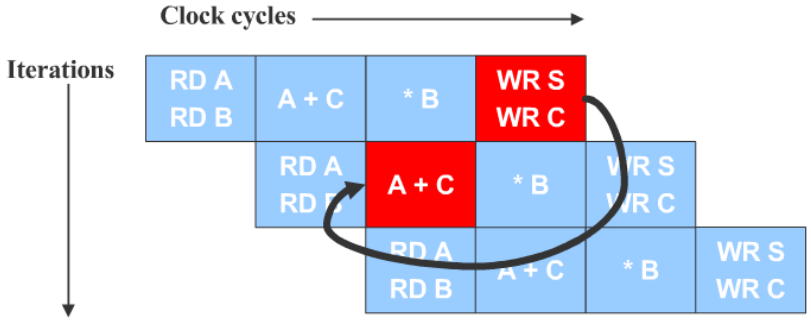
- U každé smyčky lze samostatně nastavit inicializační interval zřetězení
- Nastavení se automaticky aplikuje i na všechny vnořené smyčky



Zřetězení smyček - Chyby

- U některých obvodů nelze zřetězení provést díky zpětným vazbám
- Příklad:
  - Zřetězení s II=1 není možné
  - Výpočet nové hodnoty C vyžaduje výsledek z předchozí iterace
  - Zřetězení s II=2 již bude pracovat správně

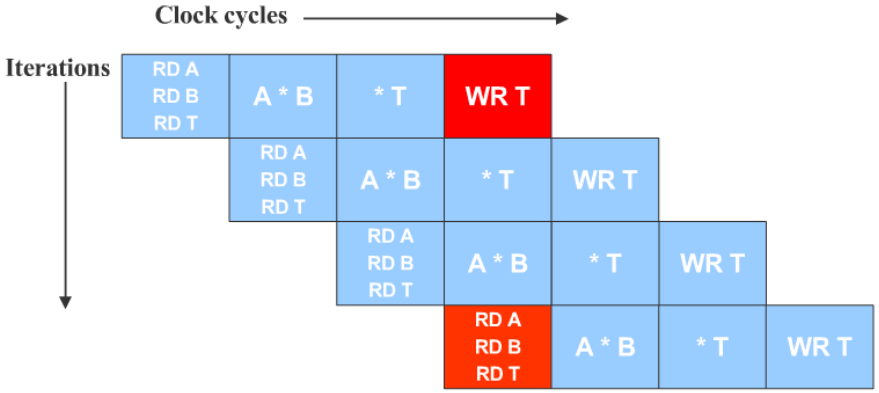
```
for (int i=0; i<N; i++) {
   temp = (A[i] + C) * B[i];
   S[i] = temp;
   C    = temp >> 16;
}
```



Zřetězení smyček - Chyby

- Zřetězení může bránit omezený přístup z/do externí paměti
- Příklad:
  - V jednom okamžiku je požadován zápis výsledku z první iterace a načtení nových operandů pro čtvrtou iteraci
  - Paměť je ale pouze jednoportová
  - Jedno z řešení může být např. použití dvouportové paměti

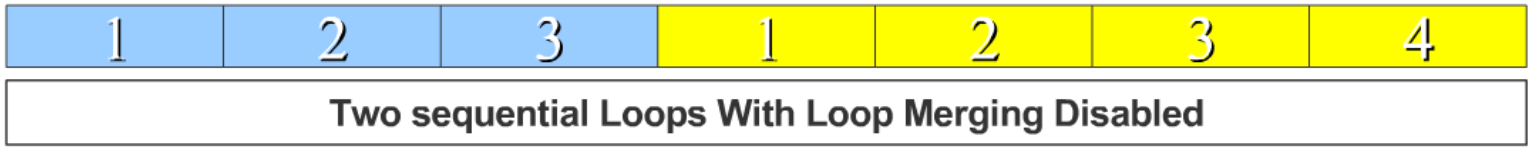
```
for (int i=0; i<N; i++) {
      T[i] = A[i] * B[i] * T[i];
}
```

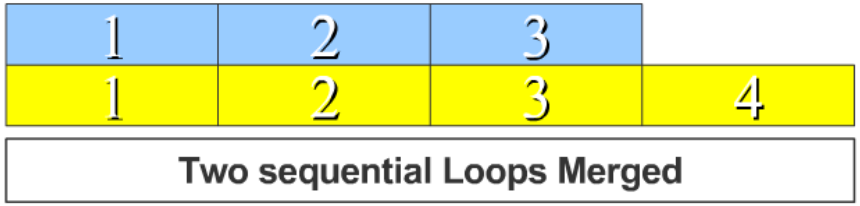


Zřetězení smyček - Doporučení

- Zřetězení smyček je vhodné pro:
  - Zvýšení propustnosti dat skrze obvod
  - Snížení latence
- Zvýšení počátečního intervalu (II) vede úsporu zdrojů (vyšší míra sdílení zdrojů)
- Zřetězení vnořených smyček aplikujte postupně od nejvnitřnější smyčky k vnějším smyčkám

Spojování smyček (Loop Merging)






```
void loop_merging1 (
   unsigned char &mod,
   char a[3],
   char b[3],
   char c[4],
   char d[4]
) {
   for (int i = 0; i<3; i++)
      b[i] = (a[i] * mod) >> 8 ;
   for (int j = 0; j<4; j++)
      d[j] = (c[j] * mod) >> 8 ;
}
```

- Spojování smyček (Loop Merging) je schopno implementovat dvě nezávislé smyčky paralelně
- Výsledná doba výpočtu je pak rovna maximu z obou smyček

Spojování smyček (Loop Merging)

```
void loop_merging1 (
   unsigned char &mod,
   char a[3],
   char b[3],
   char c[4],
   char d[4]
) {
   for (int i = 0; i<3; i++)
      b[i] = (a[i] * mod) >> 8 ;
   for (int j = 0; j<4; j++)
      d[j] = (c[j] * mod) >> 8 ;
}
```

Equivalent Code

```
void loop_merging2 (
   unsigned char &mod,
   char a[3],
   char b[3],
   char c[4],
   char d[4]
) {
   for (int j = 0; j<4; j++) {
      if (j<3) b[j] = (a[j] * mod) >> 8 ;
      d[j] = (c[j] * mod) >> 8 ;
   }
}
```

- Příklad: Při spojení smyček jsou vyžadovány dvě násobičky, v původní verzi postačila jedna
- Implicitně je u všech smyček povoleno spojení
- Důvody pro vypnutí spojení
  - Snaha sdílet zdroje
  - Omezena vstupně/výstupní propustnost může vyžadovat sekvenční přístup k externím zdrojům

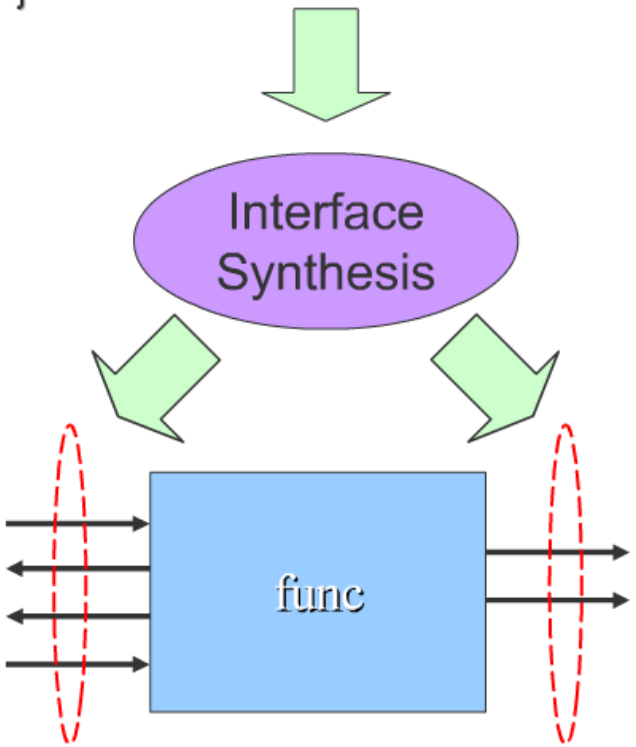
Osnova

- Základní schéma návrhu
- Datové typy s bitovou přesností
- Syntéza smyček
- Mapování rozhranní a pamětí
- Shrnutí

Syntéza rozhraní

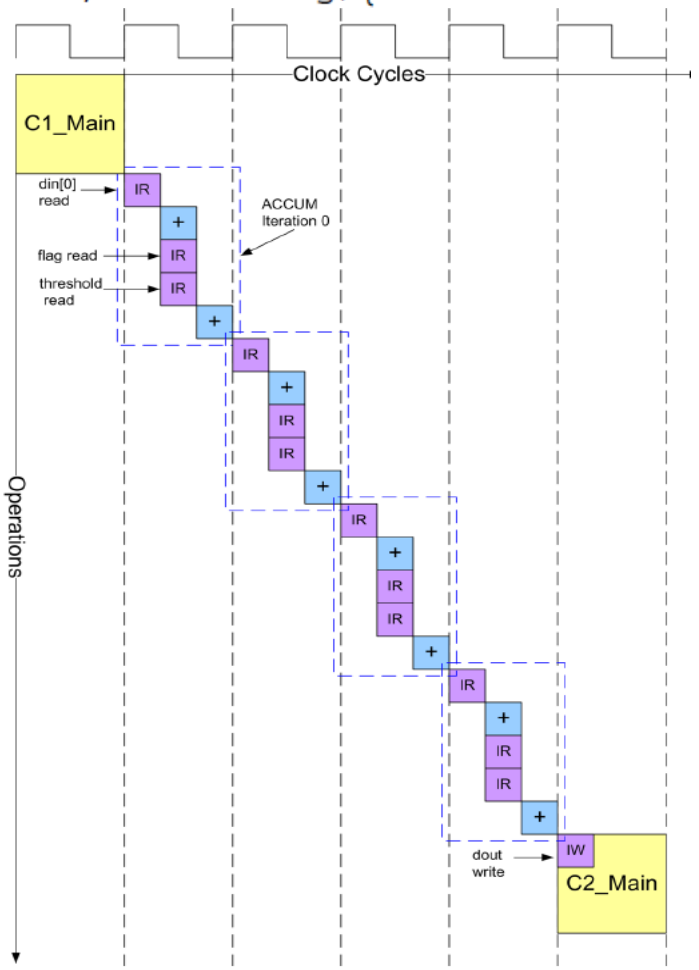
- Vstupy a výstupy jsou odvozeny od parametrů hlavní (main) funkce
- Podle způsobu použití parametru automaticky rozpozná směry IN, OUT a INOUT
- Jazyk C již ale neříká nic o tom, jak jsou hodnoty na rozhraní předávány (komunikační protokol)
- Z knihovny lze k jednotlivým parametrům přiřadit předdefinované typy nejčastěji používaných protokolů (wired, wired_en, handshake), popř. definovat svůj vlastní

```
void func ( int  A[5][16],
            char B[16],
            bool mode   ) {
   ...
}
```



Rozhraní – předání parametru referencí

```
void accumulate(int din[4], int &dout, int &threshold, bool &flag){
  int acc=0;
 ACCUM:for(int i=0;i<4;i++){
    acc += din[i];
    if(flag)
      if(acc > threshold)
        acc = threshold;
  }
  dout = acc;
}
```

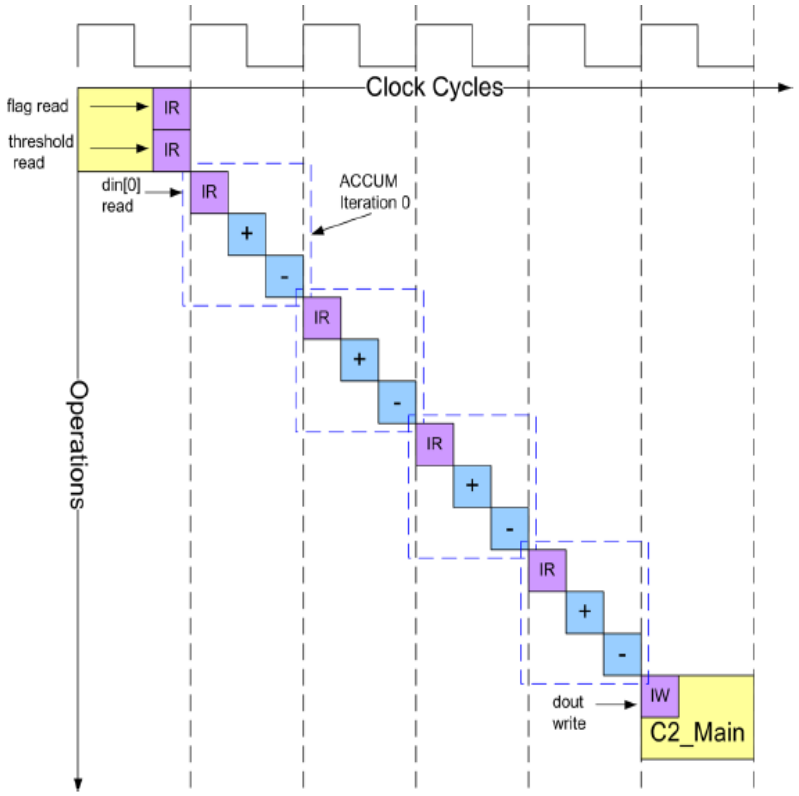


- Parametry předávané referencí jsou označeny znakem &
- Jsou čteny pokaždé, když se využívají – parametr pouze odkazuje na hodnotu, která se může v průběhu zpracování měnit

Rozhraní – předání parametru hodnotou

```
void accumulate(int din[4], int &dout, int threshold, bool flag){
  int acc=0;
 ACCUM:for(int i=0;i<4;i++){
    acc += din[i];
    if(flag)
      if(acc > threshold)
        acc = threshold;
  }
  dout = acc;
}
```

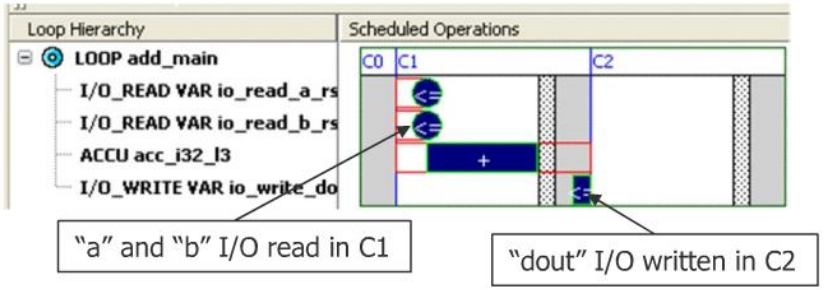
- Parametry předané hodnou jsou čteny pouze jednou a to na začátku rutiny
- Jsou uloženy do pomocných registrů odkud se využívají po celou dobu výpočtu

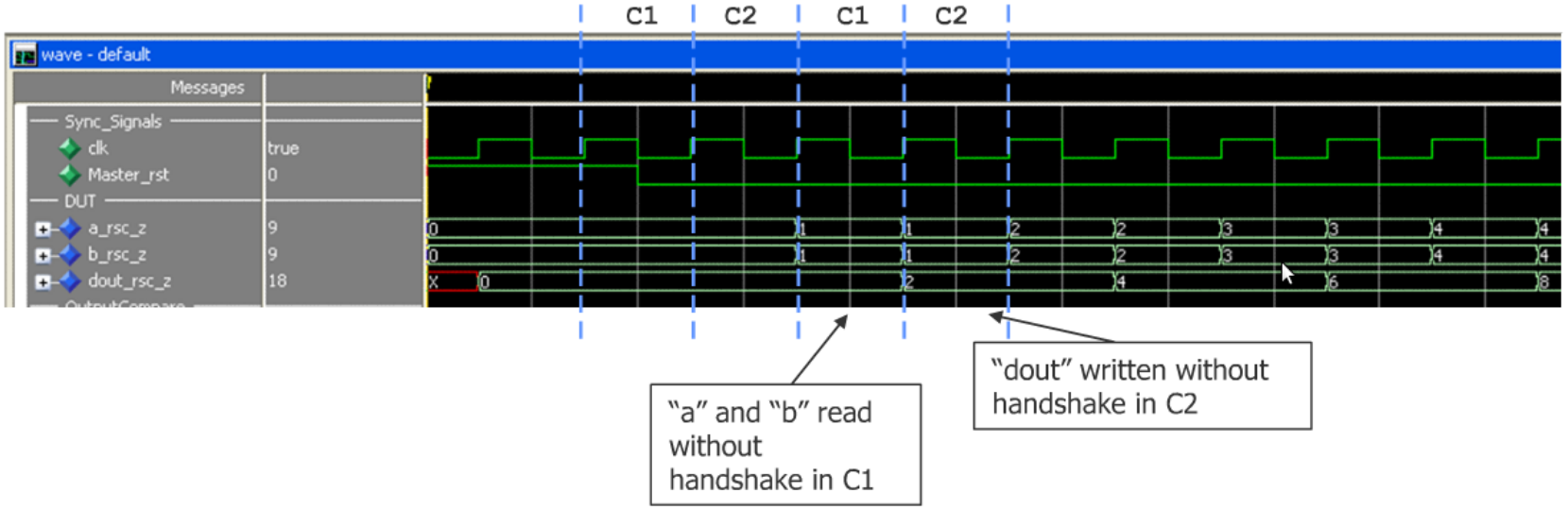


Rozhraní typu wired

- Implicitní typ rozhraní, bez pomocných signalů (enable, handshake)
- Návrhář je zodpovědný za korektní přísun vstupních dat a odběr výstupních dat na základě výsledku plánování

```
#pragma design top
void add(int a, int b, int &dout){
  dout = a+b;
}
```

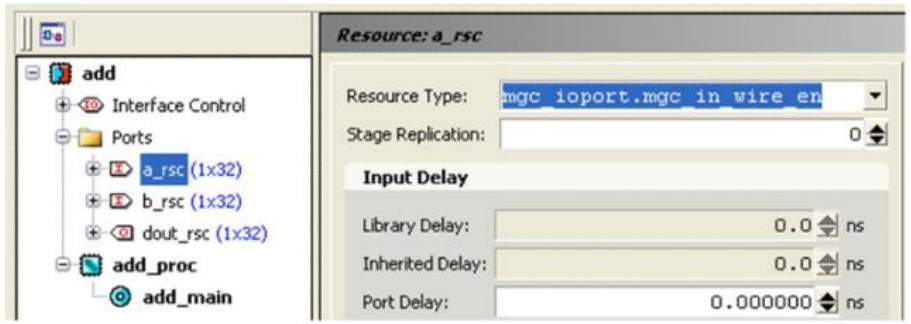


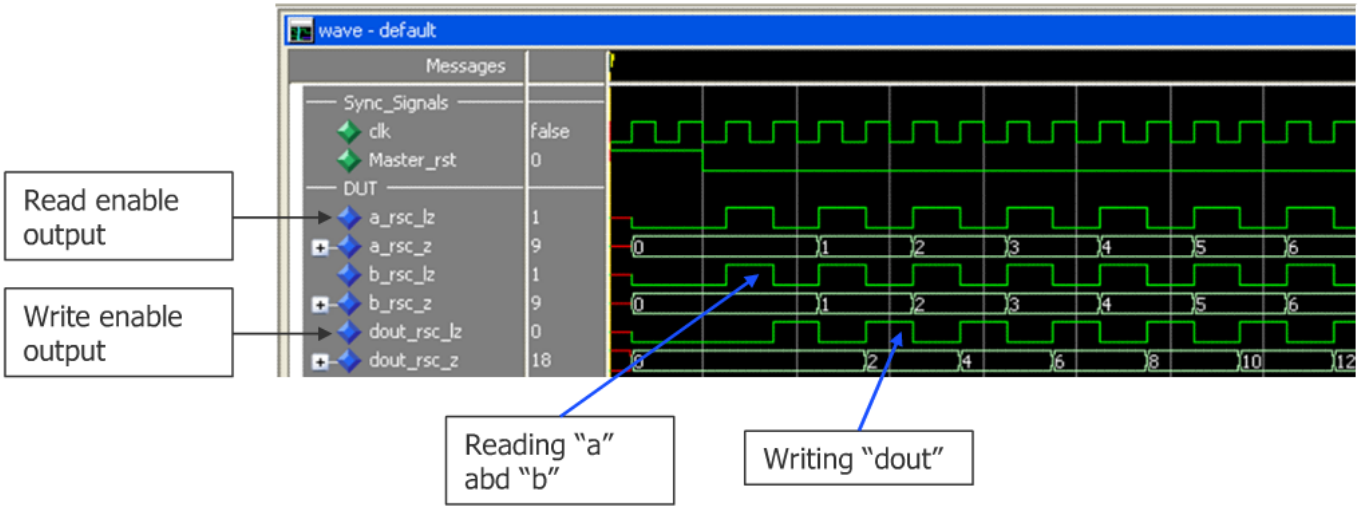


Rozhraní typu wired_en

- Do rozhraní je ke každému parametru doplněn enable signal s příponou *_lz
- Výsledný obvod sám aktivuje tyto enable signály v okamžiku, kdy vyžaduje čtení nebo zápis

```
#pragma design top
void add(int a, int b, int &dout){
  dout = a+b;
}
```

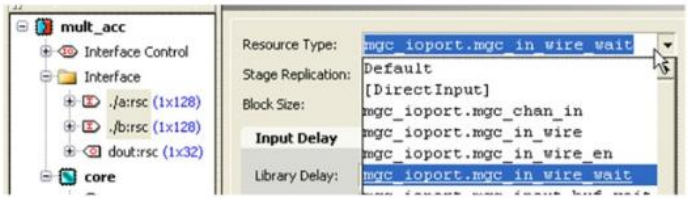


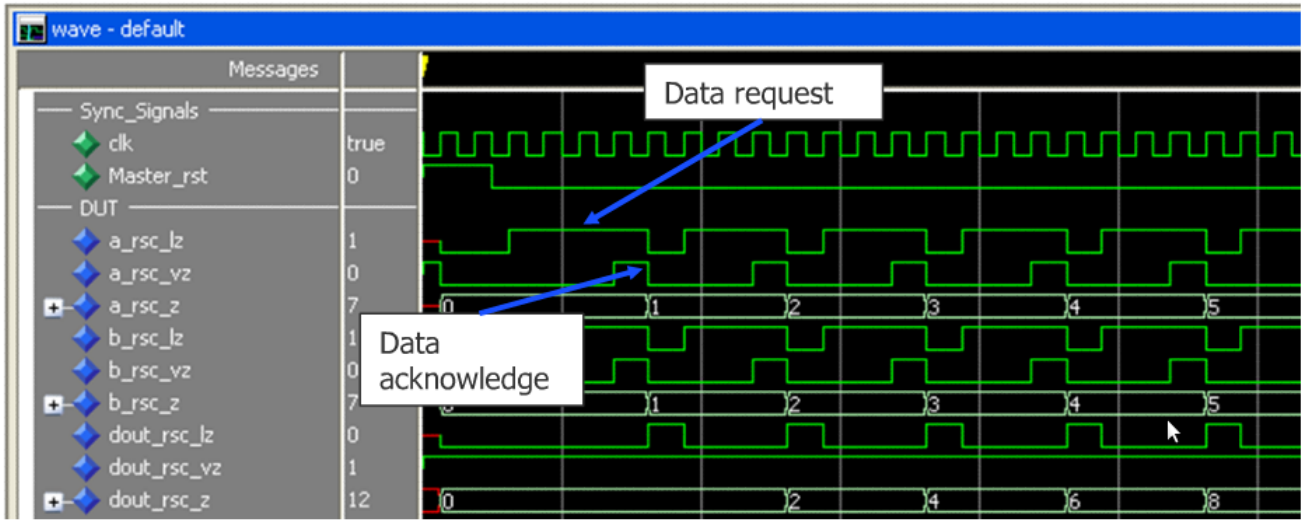


Rozhraní typu handshake

- Kromě enable signálů *_lz jsou do rozhraní doplněny také *_vz signály vyjadřující potvrzení dat ze strany externí komponenty
- Dokud nepřijde potvrzení zůstává enable signál aktivní a současně je jádro obvodu pozastaveno (podobně jako u zřetězené linky)

```
#pragma design top
void add(int a, int b, int &dout){
  dout = a+b;
}
```

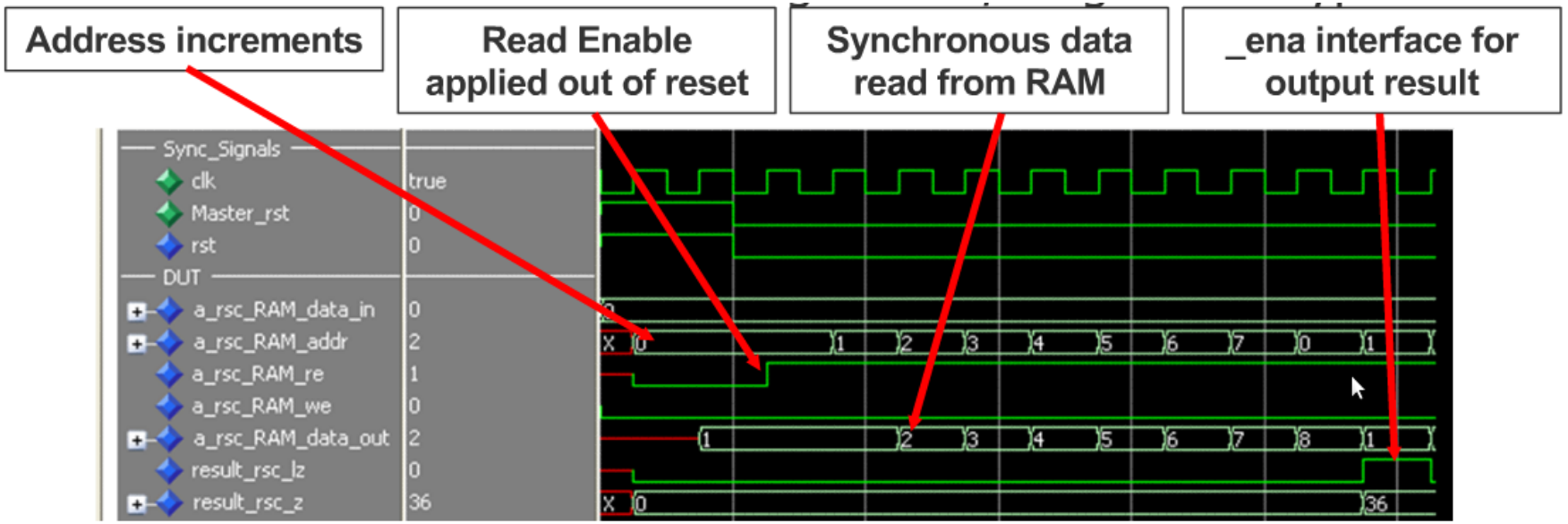




Paměťové rozhraní

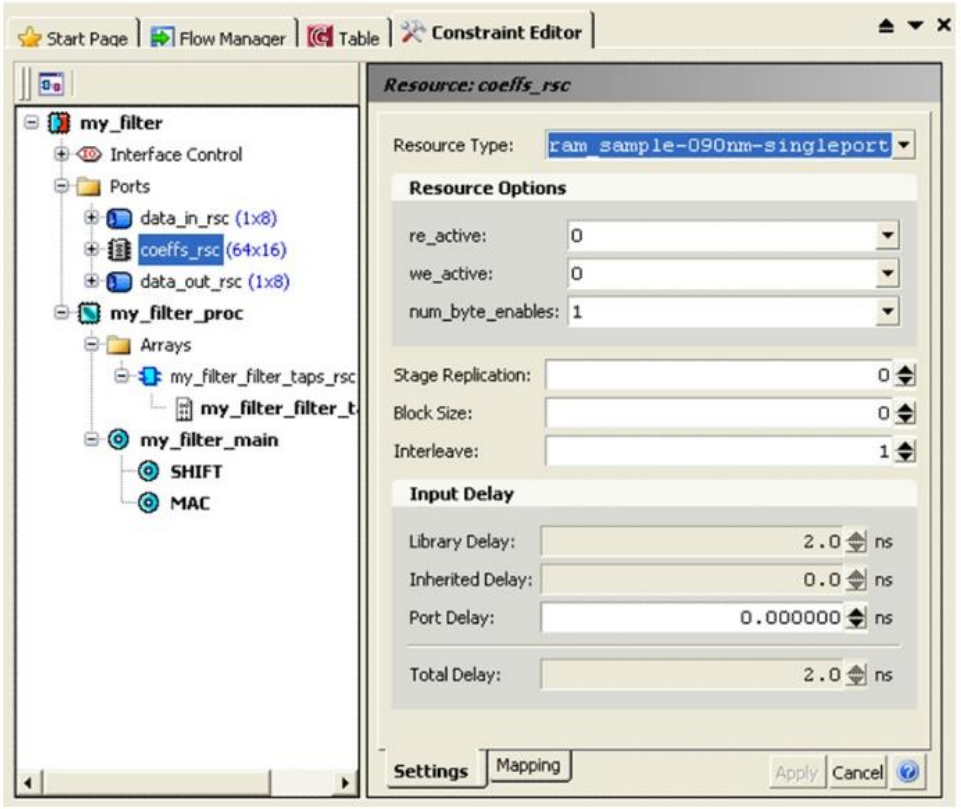
- Pokud je vstupem pole hodnot se sekvenčním přístupem, lze využít paměťové rozhraní
- Do rozhraní jsou doplněny signály *_data_in, *_data_out, *_addr, *_re, *_we
- Čtení i zápis probíhají synchronně

```
void test2 (
   int8 a[8],
   int13 &result
) {
   int13 acc =0;
   for (int i=0;i<8;i++)
      acc += a[i] ;
   result = acc ;
}
```



Paměťové rozhraní

- Konfigurovat lze:
  - Polaritu Read a Write Enable signálů
  - Přítomnost Byte Enable signálů
  - Vstupní a výstupní zpoždění používané paměti
  - Způsob uložení dat
    - Blokově
    - Prokládaně



Paměťové rozhraní

- Při rozbalování smyček vzrůstá obvykle požadavek na vstupně/výstupní propustnost
- V rámci paměťového rozhraní lze zvýšit propustnost blokovým přístupem nebo prokládáním – paměť rozdělena do několika bloků s paralelním přístupem

```
ac_int<8,false> a[16];
directive set /top_func/sub_func_proc/a_rsc -INTERLEAVE 4
```

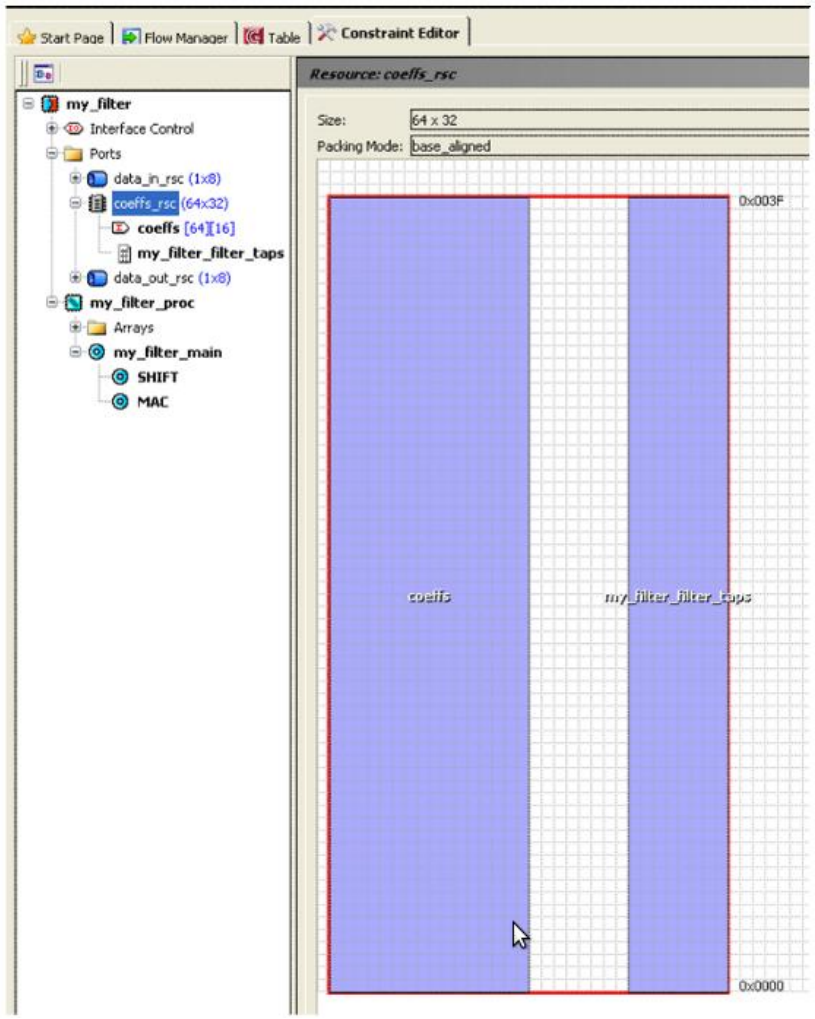
a_rsc_0	a_rsc_1	a_rsc_2	a_rsc_3
a[0]	a[1]	a[2]	a[3]
a[4]	a[5]	a[6]	a[7]
a[8]	a[9]	a[10]	a[11]
a[12]	a[13]	a[14]	a[15]

```
directive set /top_func/sub_func_proc/a_rsc -BLOCK_SIZE 4
```

a_rsc_0	a_rsc_1	a_rsc_2	a_rsc_3
a[0]	a[4]	a[8]	a[12]
a[1]	a[5]	a[9]	a[13]
a[2]	a[6]	a[10]	a[14]
a[3]	a[7]	a[11]	a[15]

Paměťové rozhraní

- Alternativně lze zvyšovat datovou šířku paměťového bloku a definovat umístění parametrů v rámci datového slova nebo bázové adresy
- Parametry:
  - Word Width
  - Base Address
  - Base Bit
- Příklad: sloučení polí Coeff a Taps do jedné paměti
  - Pole Coeff
    - Word Width: 16
    - Base Address: 0
    - Base Bit: 16
  - Pole Taps
    - Word Width: 8
    - Base Address: 0
    - Base Bit: 0



Osnova

- Základní schéma návrhu
- Datové typy s bitovou přesností
- Syntéza smyček
- Mapování rozhranní a pamětí
- Shrnutí

Shrnutí

- Zcela nový pohled na návrh obvodů od definice algoritmu až po výsledné RTL schéma
- Hlavní důraz kladen na práci s plánem v podobě data flow diagramu rozděleného do C-steps (na místo tvorby RTL)
- Vysoká produktivita - během pár kliknutí myší, lze vytvořit zcela nový obvod s jiným poměrem doby zpracování vs. množství výpočetních zdrojů
- Automatizovaný proces verifikace schopný porovnat funkčnost výsledného RTL obvodu oproti původnímu algoritmu napsanému v jazyce C++ a spuštěném na běžném procesoru

Shrnutí

- Rozhraní definována až v rámci aplikace omezujících podmínek, podpora nejčastěji používaných komunikačních protokolů
- Velmi propracovaná podpora mapování rozhraní a proměnných do paměťových bloků typu RAM, ROM
- Prozatím je vyžadována poměrně značná interakce s uživatelem při mapování smyček
- Do budoucna lze očekávat větší míru automatizace při rozbalování smyček a prohledávání stavového prostoru všech obvodů

Literatura

- Mike Fingeroff, ***High-Level Synthesis Blue Book***, January 2010
- Mentor Graphics, ***Catapult C Synthesis User's and Reference Manual***, July 2010

Konec

Děkuji za pozornost