Újrakonfigurálható technológiák nagy teljesítményű alkalmazásai

Magas szintű hardver leírás

Szántó Péter

BME MIT, FPGA Laboratórium

Hardver leírás

- **RTL (register transfer level) leírás – Verilog/VHDL**
  - A feladat újragondolását, manuális párhuzamosítását igényli
  - Viszonylag kényelmetlen tesztkörnyezet létrehozása
  - LASSÚ szimuláció
- **Magas szintű leírás**
  - C leírásból hardver szintézis
  - „Modell alapú”, grafikus fejlesztői környezetek
    - Xilinx System Generator (Simulink)
    - Synopsys Synplify DSP (Simulink)
    - NI LabView FPGA

C → hardver

- **Xilinx Forge: Java alapú HW leíró nyelv**
- **Celoxica Handel-C**
  - 1996-2000, Oxford University Computing Laboratory
  - {Celoxica, Catalytic} → Agility
  - 2009: Mentor Graphics megvásárolja a Handel-C-hez kapcsolódó jogokat, termékeket
- **Streams-C**
  - Los Alamos National Laboratory → Impulse C
- **Mitrionics Mitrion-C**
  - Elsősorban HPC területre
- **Calypto (ex-Mentor) Graphics Catapult-C**
- **Altera C2H, Xilinx AutoESL, Synopsys Synphony**

Celoxica Handel-C

- **ANSI-C szintaxis, néhány kiegészítő nyelvi elemmel**
- **Módosítatlan C kód → szekvenciális végrehajtás**
  - Párhuzamosítás explicit, a programozó feladata
  - par{ } kulcsszó
- **Kommunikáció a párhuzamos kódok között: channel**
  - FIFO jellegű felhasználás
  - Szinkronizáció
- **Kimenet**
  - Szintetizált EDIF file a támogatott FPGA-kra
  - Verilog/VHDL kód

Handel-C változók

- **Tetszőleges bitszélességű változók**
  - unsigned int 13 x; → 13 bites előjel nélküli
  - int 16 y; → 16 bites előjeles
- **Nincs automatikus cast/típus konverzió**
- **Van bit kiválasztás ([ ]) és konkatenálás (@)**

```
int 16 x;
int 14 y;
x = y[13] @ y[13] @ y;
```

```
macro expr copy(x, n) =
    select(n==1, x, (x @ copy(x, n-1)));

 macro expr extend(y, m) =
    copy(y[width(y)-1], m-width(y)) @ y;
```

- **width( ): visszaadja egy változó szélességét bit-ben**

Handel-C függvények

- **NINCS rekurzív függvény hívás**
- **A függvények paraméter listája konstans**
- **Paraméterek típusa ismert**
- **main( ) függvénynek nincs visszatérési értéke és nincsenek paraméterei**
  - Egy main( ) egy adott órajelről működik
  - Lehet több main( ), több órajel tartomány

Handel-C ciklusok

- **Ciklusok időbeli ciklikusságot írnak le, a végrehajtás szekvenciális**
- **Ciklus magot legalább egyszer végre kell hajtani**
  - Ha ez nem biztosítható, alternatív végrehajtás kell

```
if (rt != 0)
   for (i=0; i<rt; i++)
   {
        ……
   }
else
   delay;
```

Handel-C párhuzamosítás

- C kód végrehajtása alapvetően szekvenciális
- Párhuzamosítás: par{ } blokk – 1 változó 1 blokkból írható
- Par ciklus is létrehozható

```
x = 0;
y = 1;
z = 2;
```

3 órajel

```
seq{
   x = 0;
   y = 1;
   z = 2;
}
```

```
par{
    x = 0;
    y = 1;
    z = 2;
}
```

1 órajel

2 órajel

```
par{
    x = 0;
    seq{
       y = 1;
       z = 2;
    }
}
```

Handel-C channel

- **Szinkronizációhoz vagy adatcseréhez (FIFO)**
  - chanin/chanout: szimulációban file olvasás/írás

```
chan int 16 MyCh;
```

```
chan int 16 MyCh with {fifolength = 4};
```

  - Egy időben egy helyről írható/olvasható
    - Olvasás
    - Írás

```
MyCh ? x;
MyCh ! x;
```

Handel-C prialt

- Több channel kezelése egy process-ben

```
while(1)
   prialt
        {
             case chan1 ? y: break;
             case chan2 ? y: break;
             case chan3 ? y: break;
        }
```

- Prioritás: chan1 → chan3

ifselect

- Fordítási idejű elágazások létrehozása
- Pl. shift regiszter

```
int 16 q; int 16 shr[15];
par (i=0; i<16; i++)
{
   ifselect (i==0)         shr[0] = input;
   else ifselect (i==15)   q = shr[i-1];
   else                    shr[i] = shr[i-1];
}
```

Shared expression

- Alapértelmezés: minden értékadáshoz saját HW

```
seq{
    a = b * c;
    d = e * f;
    g = h * i;}
```

3 órajel (szekvenciális)
3 szorzó
1/3 kihasználtság

- Megosztott erőforrás: shared expression

```
shared expr mult(x, y) = x * y;

seq{
    a = mult(b, c);
    d = mult(e, f);
    g = mult(h, i);
}
```

3 órajel (szekvenciális)
1 szorzó

RAM/ROM

- Egy port-os RAM/ROM deklaráció

```
ram int 6 a[43];
static rom int 16 b[23] = {……};
```

- Több portos memória

```
mpram mpram_s{
    ram <signed 16> rw0[16];
    rom <signed 16> r1[16]; };

mpram mpram_s memory_inst;
```

- Címzés: megfelelő szélességű változóval!

Példa: FIR szűrő

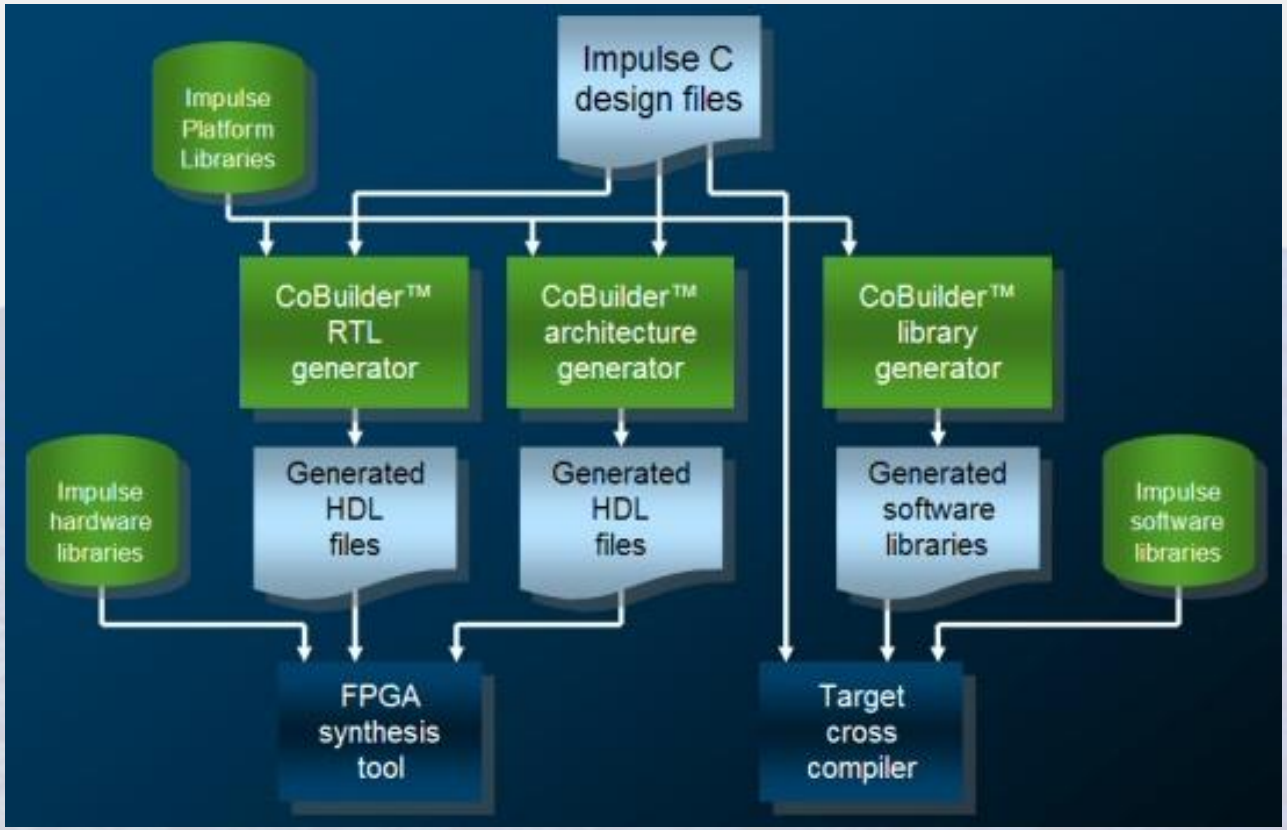
```
while(1){
par{
   InCh ? smpl_ram.rw0[sw_addr];
   accu = 0; c_addr = 0;
   sr_addr = sw_addr;sw_addr++;
}
while(c_addr<=7)
   par{
      accu = accu+
         ext(coeff[c_addr[2:0]],32)*ext(smpl_ram.rw0[sr_addr], 32)+
         ext(coeff[7-c_addr[2:0]],32)*ext(smpl_ram.r1[sr_addr-8], 32);
      sr_addr--; c_addr++;
   }
SimOut ! accu[30:15];
}
```

Impulse-C

- **Kiterjesztett C nyelvű leírás**
- **Process: független kódrészlet**
  - implementáció: SW vagy HW
- **Kommunikáció**
  - Stream: adatfolyam (FIFO)
  - Signal: 1:1 szinkronizáció
  - Szemafor: 1:több szinkronizáció
  - Register
  - Shared Memory: osztott memória a HW és SW között

Impulse-C

- **Platform: implementáció cél**
  - FPGA, processzor, kommunikációs busz, ….



Impulse-C

- **Egyszerű HW gyorsító fejlesztés Stream használatával**
- **Stream: két process közötti FIFO kapcsolat**
- **Használat:**
  - Stream létrehozása
  - Stream megnyitása mindkét process-ben (SW & HW)
  - Stream írása ill.
  - Stream olvasása
  - Stream bezárása

Impulse-C

- Stream létrehozása (név, típus, FIFO méret)

```
strm_image_value =
  co_stream_create("IMG_VAL",INT_TYPE(16),BUFSIZE);
```

- Stream megnyitás (mode: O_RDONLY vagy O_WRONLY)

```
co_stream_open(co_stream stream, mode_t mode,
  co_type type);
```

- Stream írás & olvasás (SW & HW is!)

```
co_error co_stream_write(co_stream stream, const
  void *buffer, size_t buffersize);
```

```
co_error co_stream_read(co_stream stream, void
  *buffer, size_t buffersize);
```

- SW oldalon van burst támogatás

HW generálás

- **„Instruction stage” – 1 órajel**
- **Új „instruction stage”-t okozó utasítások**
  - Vezérlés (if, swicth)
  - Ciklus
  - Memória/tömb újabb hozzáférés
- **Pl. 1 stage a ciklus magja:**

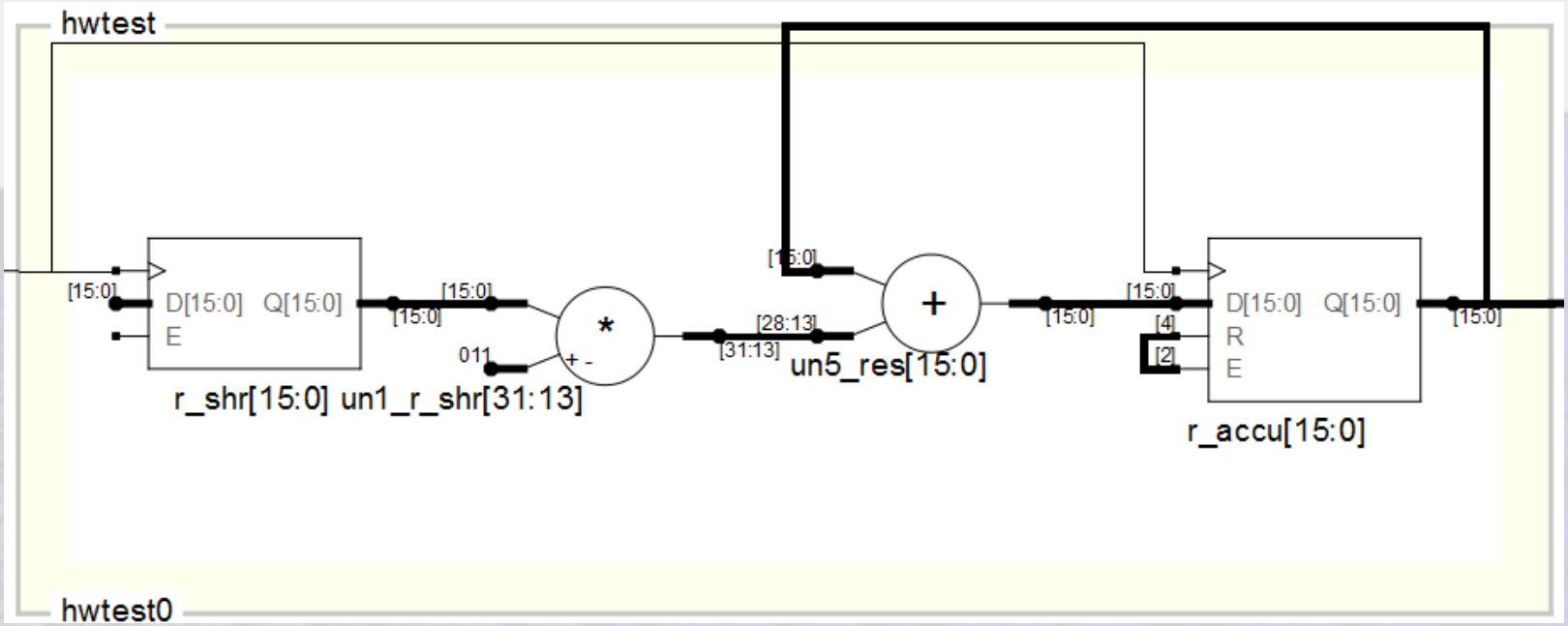
```
for (i=0; i<10; i++)
    sum += i << 1;
```

  - 1 órajel alatt 1 összeadás és shiftelés
  - Összesen 10 órajel

HW generálás

- #pragma nélküli C kód

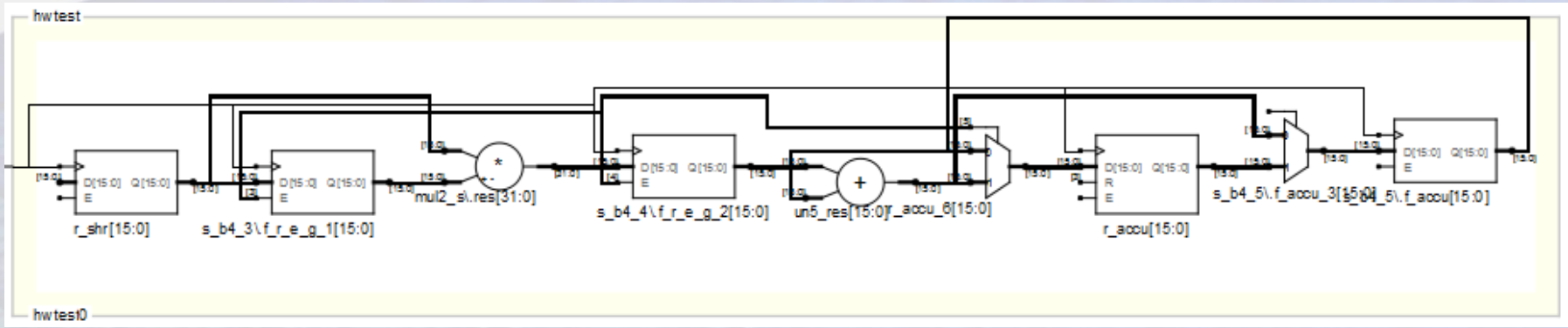
```
for (i=1; i<16; i++){
   accu = accu + shr[i]*3;
}
```



Pipeline

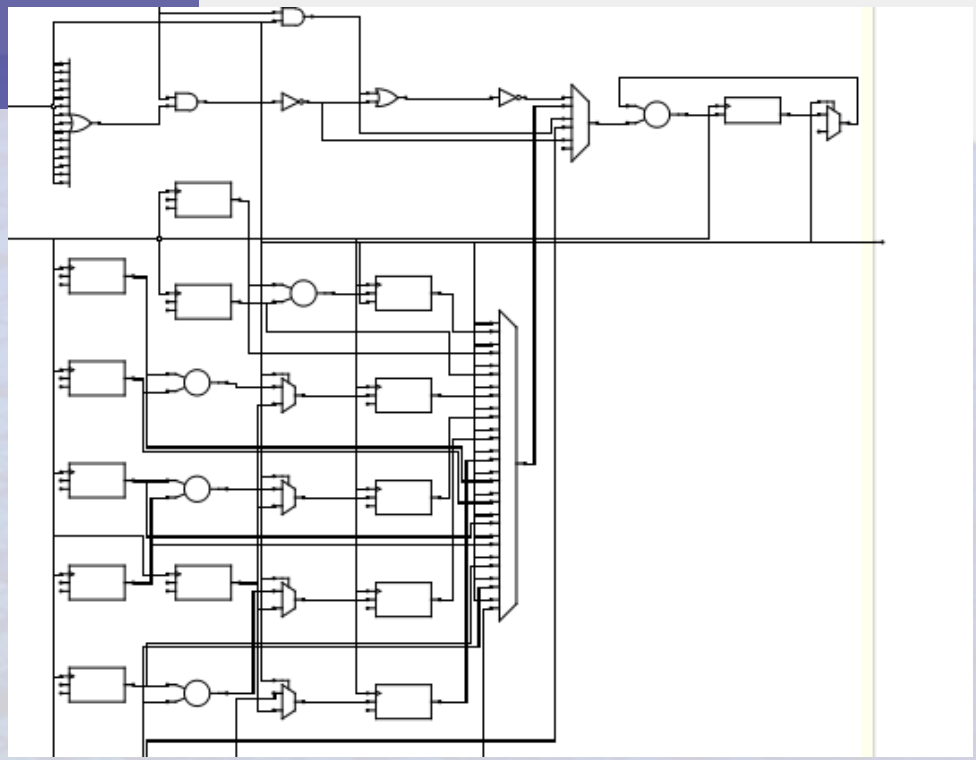
- #pragma CO PIPELINE

```
for (i=1; i<16; i++){
#pragma CO PIPELINE
#pragma CO SET stageDelay 16
   accu = accu + shr[i]*3;
}
```



Loop unroll

```
for (i=1; i<16; i++){
#pragma CO UNROLL
#pragma CO PIPELINE
#pragma CO SET stageDelay 16
   accu = accu + shr[i]*3;
}
```



Egyéb pragma-k

- **#pragma CO FLATTEN**
  - case & if-else szerkezetek „kiterítése” (1 órajel)
- **#pragma CO NONRECURSIVE**
  - Tömbök: az elérés nem rekurzív, szabadon pipeline-osíthstó
- **#pragma CO SET "mul2_s.XX" 1**
  - XX=lut, dsp, pipeline
- **Megkötések**
  - Csak teljes unroll támogatott
  - Pipeline cikluson belül nem lehet másik ciklus (!!!)

Mentor Catapult-C

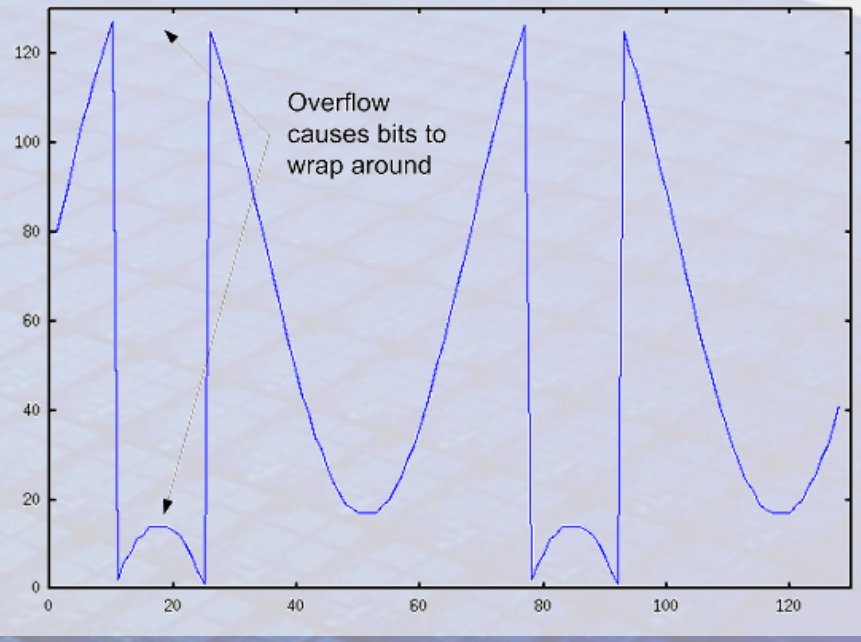
- **Időzítetlen C kódból HW generálás**
- **Algoritmus: C kód**
- **Architektúra: C kód**
  - Pl. cirkuláris buffer vagy shift regiszter
- **Mikroarchitektúra**
  - Implementációs lehetőségek a funkcionalitás megváltoztatása nélkül
  - Pl. 4 szorzás elvégzése 1/2/4 szorzóval
  - Megadás:
    - GUI
    - Direktívák és pragma-k

Catapult-C

- **Top-level hardver modul: C függvény**
- **Tetszőleges szélességű változók: AC adattípus**
  - Standard C könyvtár, szabadon felhasználható
  - http://www.mentor.com/products/esl/high_level_synthesis/ac_datatypes
- **Nem támogatott C elemek**
  - Dinamikus memória foglalás
  - Pointerek tömb indexeléshez
  - Union
  - float és double adattípusok
  - Rekurzió nem konstans rekurzió mélységgel

AC típusok

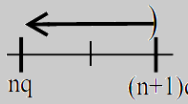
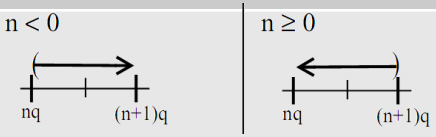
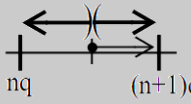
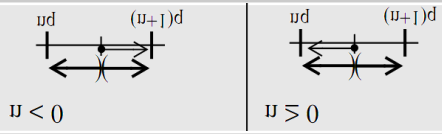
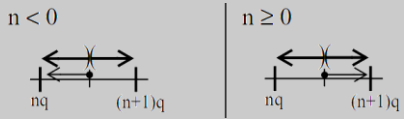
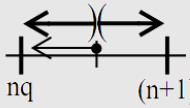
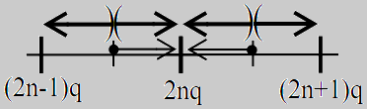
- **AC: Algorithmic C adattípus**
  - egész típus: ac_int.h
  - fixpontos típus: ac_fixed.h
- **Egész**
  - deklaráció: ac_int<W, false> x;
    - W: bit szélesség
    - false: pozitív
      true: kettes komplemens
  - Ábrázolási tartománynál nagyobb érték: túlcsordul



AC típusok

- **Fix pontos**
  - Deklaráció: ac_fixed<W,I,S,Q,O> x;
    - W: változó teljes bitszélessége
    - I: egészrész mérete
    - S: false: pozitív; true: kettes komplemens
    - Q: kvantálási mód
    - O: túlcsordulás mód
- **Példák**
  - ac_fixed<4, 4, false>: 0…15, felbontás: 1
  - ac_fixed<4, 6, true>: -32…28, felbontás: 4
  - ac_fixed<4,-1, false>: 0…15/32, felbontás: 1/32

AC – kvantálás

Mód		
AC_TRN (alapértelmezett)		Csonkolás -∞ felé
AC_TRN_ZERO		Csonkolás 0 felé
AC_RND		Kerekítés ∞ felé
AC_RND_ZERO		Kerekítés 0 felé
AC_RND_INF		Kerekítés ±∞ felé
AC_RND_MIN_INF		Kerekítés -∞ felé
AC_RND_CONV		Kerekeítés a legközelebbi páros q többszörösre

AC – túlcsordulás

Mód	
AC_WRAP (alapértelmezett)	Túlcsordulás
AC_SAT	Szaturáció MIN/MAX-ra
AC_SAT_ZERO	Szaturáció 0-ra
AC_SAT_SYM	Túlcsordulás vagy MIN érték esetén ±MAX

- **Kerekítés és szaturáció HW erőforrást igényel!**
- **Pl. 24 bites, előjeles, szaturált, kerekített változó**

```
ac_fixed<24, 1, 1, AC_RND_ZERO, AC_SAT> x;
```

AC – függvények

- **slc<W2>(int_type i) bit-vektor kiválasztás**
  - Verilog ekvivalens: x[W2-1+i : i]
  - visszatérési típus: ac_int<W2,S>

```
x = y.slc<2>(5);
```

- **set_slc(int_type i, ac_int<W2,S2> x): bit-vektor beállítás**
  - bits[W2-1+i : i] ← x

```
x.set_slc(7, y);
```

- **to_int(), to_uint(), to_long(), to_ulong(), to_int64(), to_uint64()**
- **length( ): visszaadja a változó szélességét**

AC – visszatérési típus

- **A bemenetek automatikusan kiterjesztődnek**
  - Előjel
  - Tört és egészrész
  - Kettedespontok illesztettek
- **Ez igaz a bit-operátorokra (|, &, ^) is!**
- **Relációs operátorok kimenete bool, bemenetre a fentiek igazak**

AC – visszatérési típus

- **Aritmetikai operátorok a művelet teljes végeredményét adják**

```
ac_int<8,false> a ;
ac_int<8,false> b ;
(a + b): 9-bites értéket ad
(a * b): 16-bites értéket ad
```

  - Integer konstans: ac_int<32,true> !!
- **Shift operátor a bemeneti vektor szélességének megfelelő értékkel tér vissza**

```
ac_int<8,false> x = 255;        // x =  "11111111"
ac_int<9,false> y = x;          // y = "011111111"
ac_int<9,false> z = x << 1;     // z = "011111110"
ac_int<9,false> q = y << 1;     // q = "111111110"
```

Bit kiválasztás

- Bool konverzió támogatott, így felhasználható bool-t igénylő esetben

```
while( y[k] && z[m] )  {}
z = y[k] ? a : b;
```

- Átadható int típus (LSB értékét kapja meg)

```
x[k] = n;
x[k] = (ac_int<1,false>) n;
x[k] = 1 & n;
```

AC complex

- **Definíció: ac_complex.h**
- **ac_complex<ac_fixed<16,8,true> > x (2.0, -3.0);**
  - valós és imaginárius rész is ac_fixed<16,8,true> típus
  - kezdőérték 2, -3
- **Az operátorok visszatérési típusa megegyezik a használt ac_ típusnál látottakkal**
- **A típusok keverhetőek a műveletekben**
  - Kivéve ac_ és float/double

AC Channel

- **„Streaming” jellegű interfész**
- **FIFO**
  - Top-level interfész
  - Interfész hardver modulok között
- **Több órajelet használó rendszereknél elkerülhetetlen**
- **Definíció: ac_channel.h**

AC Channel

- Hardver modul deklaráció

```
void hw_module (
   ac_channel< int > &data_in,
   ac_channel< int > &data_out
);
```

- Modulok között statikus channel

```
static ac_channel< int > channel0;
```

- AC típussal

```
ac_channel< ac_int<10,true> > &data_in;
```

AC Channel

- **available(n)**
  - Van-e n darab adat a FIFO-ban
  - CSAK szimulációban használható
    - Mindig igazzá szintetizálódik
- **Channel olvasás:**

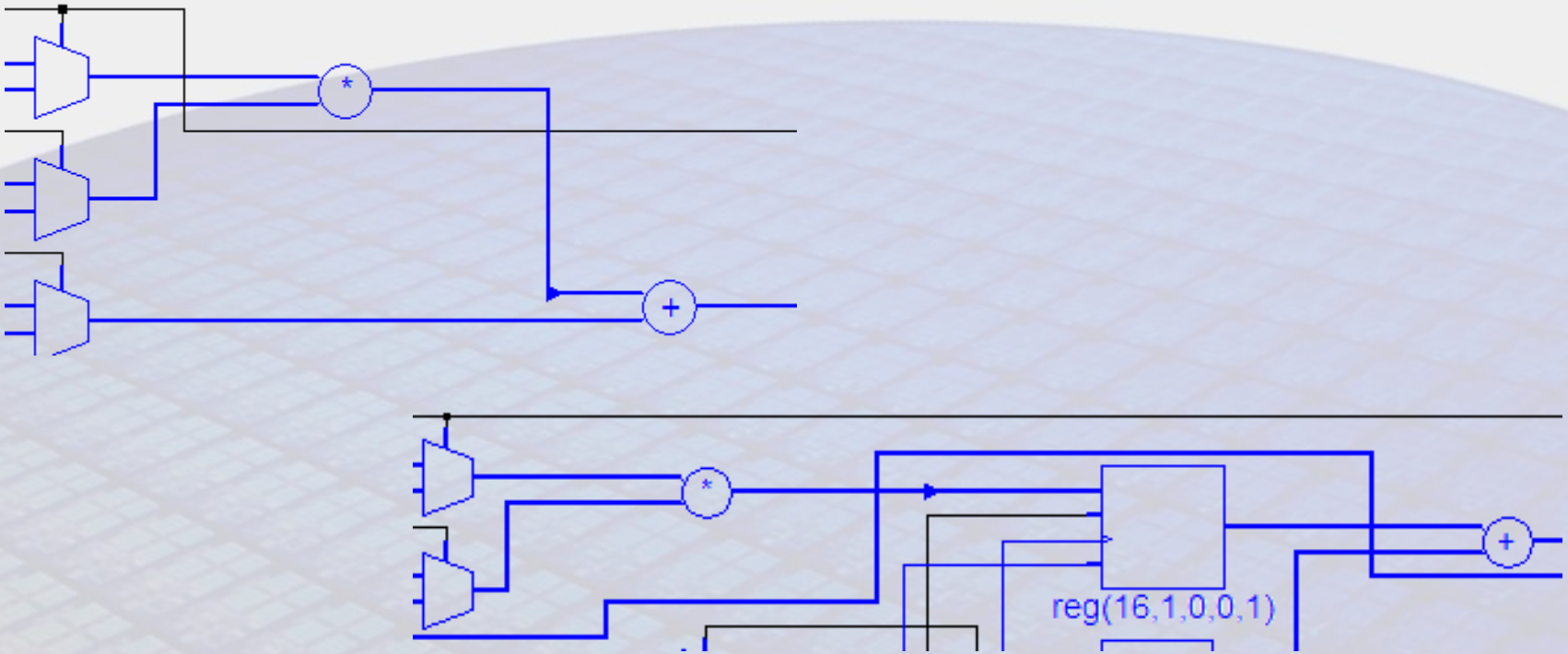
```
read_value = data_in.read();
```

- **Channel írás**

```
data_out.write( write_data );
```

Órajel constraint

- **Befolyásolja az architektúrát (FF → FF késleltetés), de nincs hatással a műveletek párhuzamosítási fokára**
  - műveletvégzők közötti regiszter szintek beillesztése automatikus, de ez NEM pipeline végrehajtás



Tömbök/memóriák

- **Tömb deklaráció:**
  - Ki-, ill. bemeneti változók
  - Belső változó
- **Ha engedélyezett a RAM primitívek használata, a tömbök automatikusan egy portos RAM-ba kerülnek!**
- **Egy tömbhöz rendelt memória adatszélessége lehet kisebb vagy nagyobb mint a tömbváltozó**
  - Egy órajel alatt részleges vagy több adat kiolvasása
- **Egy tömb több fizikai memória blokkra szétoszthető**
  - Block
  - Interleave

Tömbök – Block/Interleave

- **Egy tömb szétosztása 4 fizikai memóriába**
  - Block: egymást követő blokkok más-más memóriában

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]

  - Interleave: egymást követő címek más-más memóriában

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]

Tömbök összefogása

- **Több tömb elhelyezhető ugyanabban a fizikai memóriában**
  - Sidebyside
    - A tömbök eleminek konkatenálása
  - Compact
    - Tömbök folytonos egymás után helyezése
  - Base aligned
    - Tömbök egymás után helyezése, de minden tömb 2 hatvány címen kezdődik

Interfész szintézis – bemenet

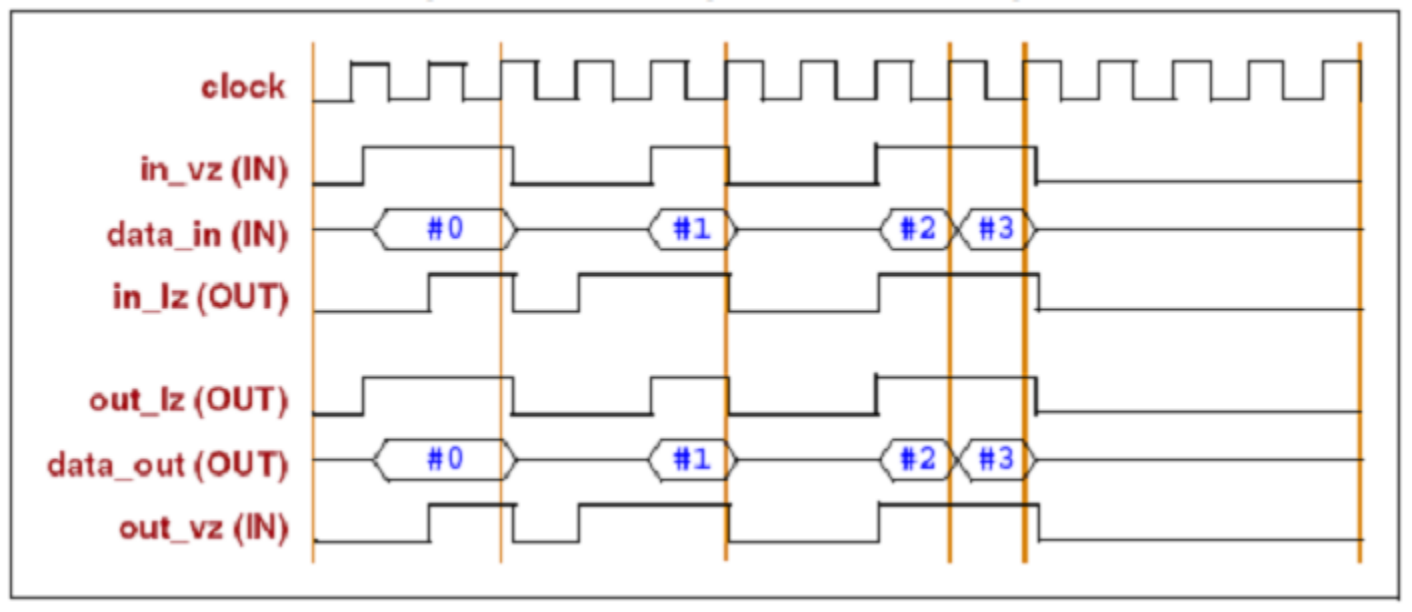
- **mgc_in_wire**
  - Csak adatbusz handshake nélkül
- **mgc_in_wire_en**
  - Adatbusz + „read acknowledge” kimenet
- **mgc_in_wire_wait**
  - Adatbusz + „read acknowledge” + „input valid”
- **mgc_out_stdreg**
  - Csak kimeneti adatregiszter

Interfész szintézis – kimenet

- **mgc_out_stdreg**
  - Csak kimeneti adatregiszter
- **mgc_out_stdreg_en**
  - Kimeneti adatregiszter + „output valid” (egyórajeles impulzus, egyébként a kimenet nem definiált)
- **mgc_out_stdreg_wait**
  - Mint az _en, plusz „output acknowledge” bemenet
- **mgc_out_buf_wait**
  - Plusz kimeneti regiszter → nem blokkolódik az „acknowledge” megérkezéséig
- **mgc_out_fifo_wait**

Interfész - handshake

- in_vz: „data in valid”
- in_lz: „data in read acknowledge”
- out_lz: „data out valid”
- out_vz: „data out acknowledge”



Ciklusok

- **Minden terv legalább egy „ciklust” tartalmaz**
  - main( ) függvény önmaga
- **Ciklusok az optimalizáció kulcspontjai**
  - merging: ciklusok párhuzamos ill. szekvenciális végrehajtása
  - unroll: ciklus kifejtése (párhuzamos végrehajtása)
  - pipeline: ciklus mag pipeline-osítása
    - „milyen gyakran futtatható le a ciklusmag újra”
- **Iterációk száma (annak maximuma) legyen ismert!**
  - Korábbi kilépés: break
  - Célszerű: legalább egyszer lefut (hátul tesztelő)
- **Ciklusok elnevezhetők → nagyon hasznos!**

Ismeretlen iteráció

```
void func (
  int a[256],
  int n,
  int result
) {
  int acc = 0 ;
  int i = 0 ;
  do {
    acc += a[i] ;
    i++;
  } while (i <= n);
  result = acc;
}
```

```
void func (
  int a[256],
  int n,
  int result
) {
  int acc = 0 ;
  for (int i=0; i<=n; i++)
    acc += a[i];
  result = acc;
}
```

Jó megoldás

```
void func (
  int a[256],
  int n,
  int result
) {
  int acc = 0 ;
  int i = 0 ;
  do {
    acc += a[i];
    i++;
    if (i==n) break;
  } while (i < 256);
  result = acc;
}
```

```
void func (
  int a[256],
  int n,
  int result
) {
  int acc = 0 ;
  for (int i=0; i<256; i++)
  {
    acc += a[i];
    if (i==n) break;
  }
  result = acc;
}
```

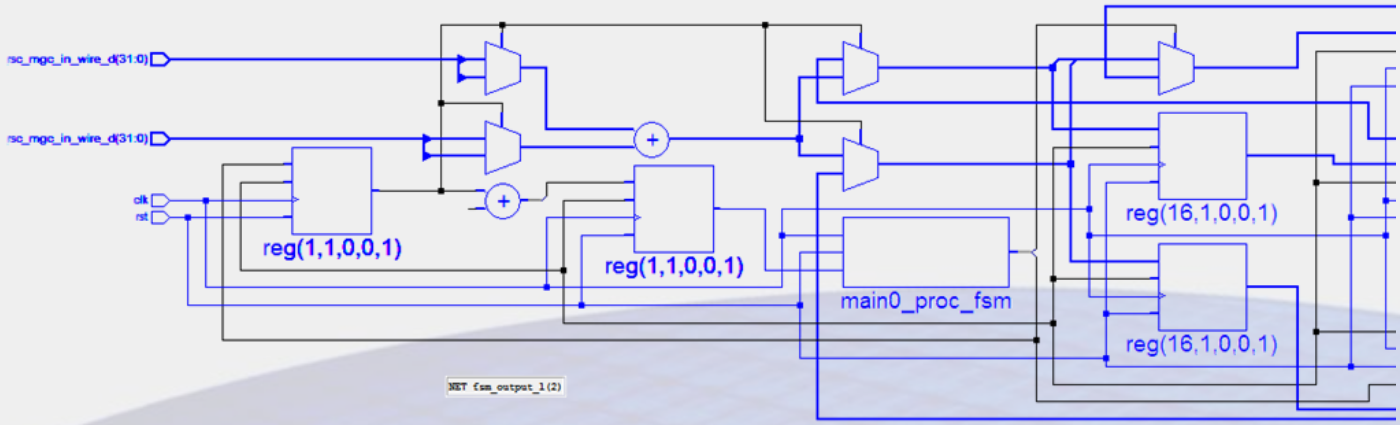
Unrolling

- **Több ciklus iteráció párhuzamos végrehajtása**
  - Nem lehet adatfüggőség
  - Bemeneti értékek rendelkezésre állása!
    - Esetleges sávszélesség limit
- **Lehetséges teljes és részleges unroll**
- **Értelemszerűen több erőforrást igényel**

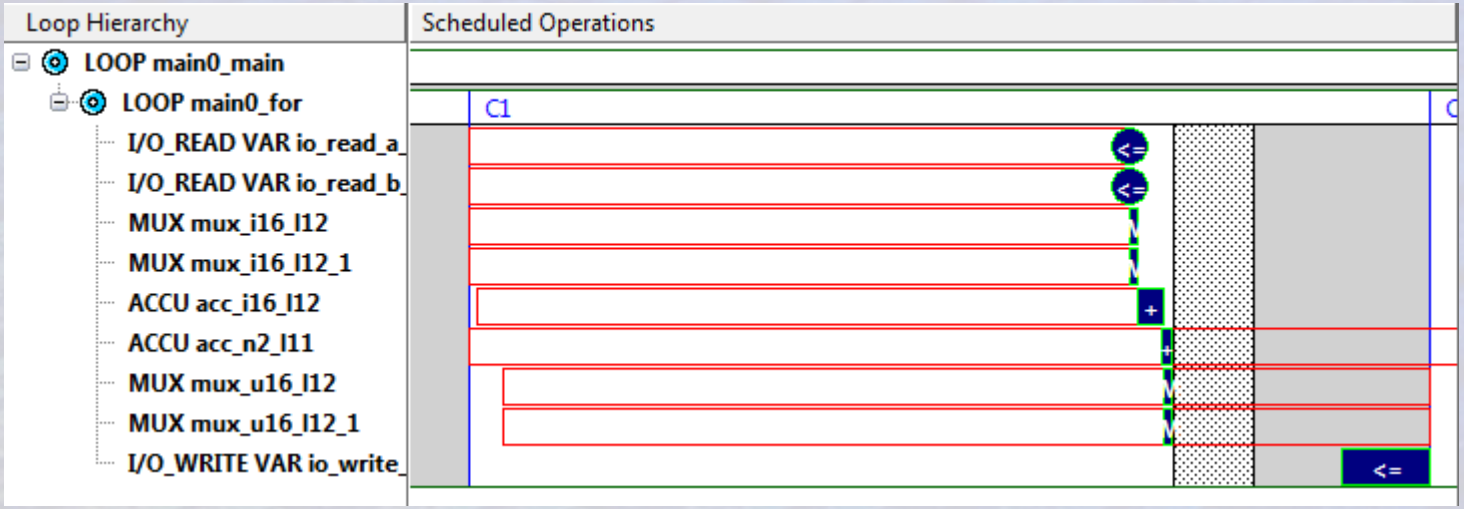
```
void main0 (
    d_type a[2], d_type b[2], d_type c[2]
)
{
    for (int i=0; i<2; i++)
        c[i] = a[i] + b[i];
}
```

Unroll kikapcsolva

- **Blokkvázlat**

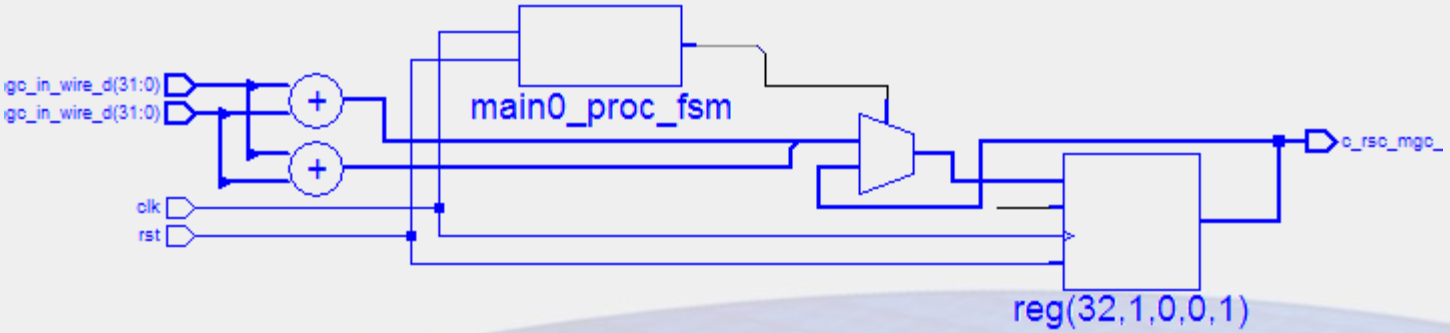


- **Időzítés**

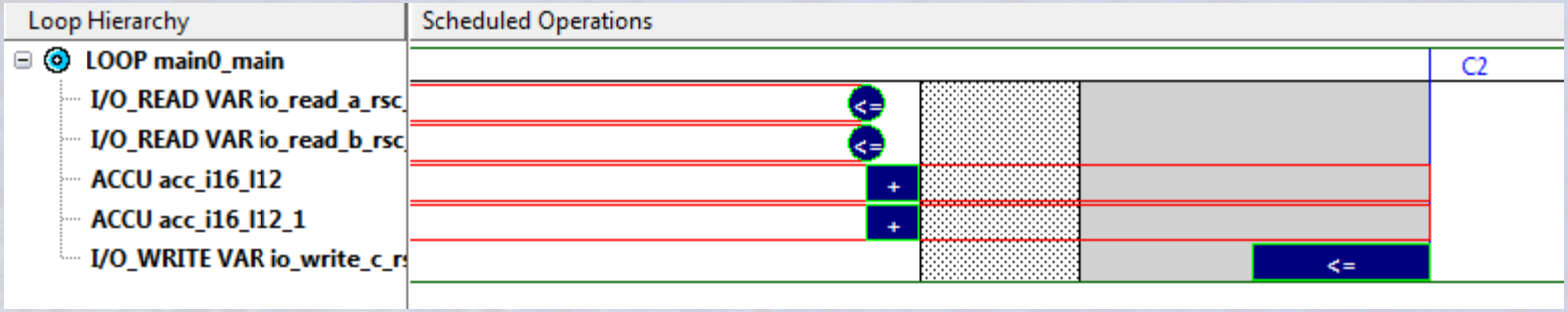


Unroll = 2

- **Blokkvázlat**



- **Időzítés**

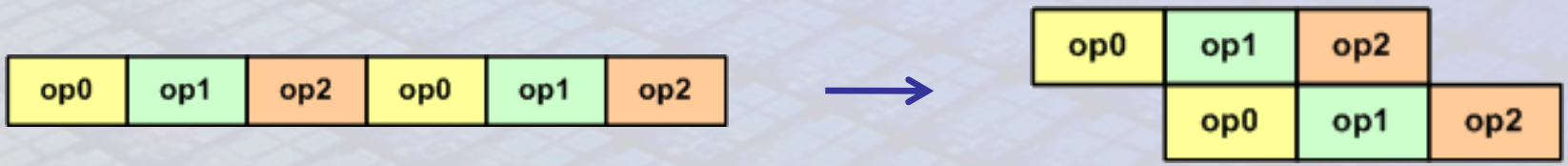


Unrolling – mit?

- **Egymásba ágyazott ciklusoknak a belső ciklusát (legalábbis először)**
- **Belső ciklusokat tartalmazó ciklus kibontása kerülendő**
  - Eredmény: szekvenciális ciklus, komplex vezérlés
- **Kifejtésre alkalmas**
  - Kis iterációs szám (vagy részleges unroll)
  - Nem túl bonyolult ciklusmag
  - Nem visszacsatolt
- **Fontos: megfelelő sávszélesség biztosítása**

Loop pipelining

- **A ciklusmag pipeline végrehajtása**
  - Következő iteráció azelőtt megkezdődik, mielőtt az előző befejeződött volna
- **Kis erőforrásigény-többlet (esetlegesen zéró)**
- **Limit**
  - Adat sávszélesség
  - Adatfüggőség az egyes iterációk között
- **Iteration Interval (II): iteráció elkezdésének periódusideje (órajelben)**
  - 1: minden órajelben újabb iteráció kezdődik

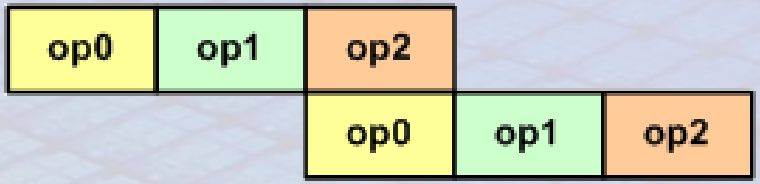


Loop pipelining

- 3 órajelet igénylő szekvenciális művelet (II = 3)

- II = 1

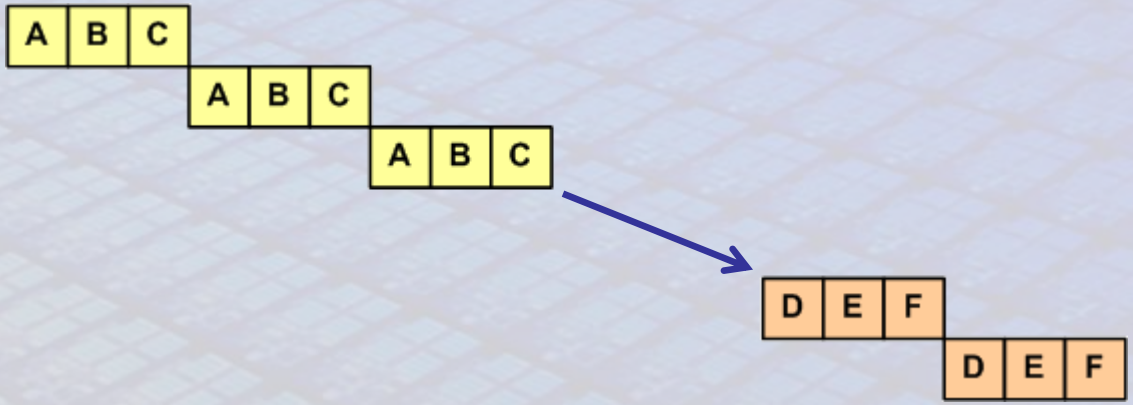
- II = 2



Egymásba ágyazott ciklusok

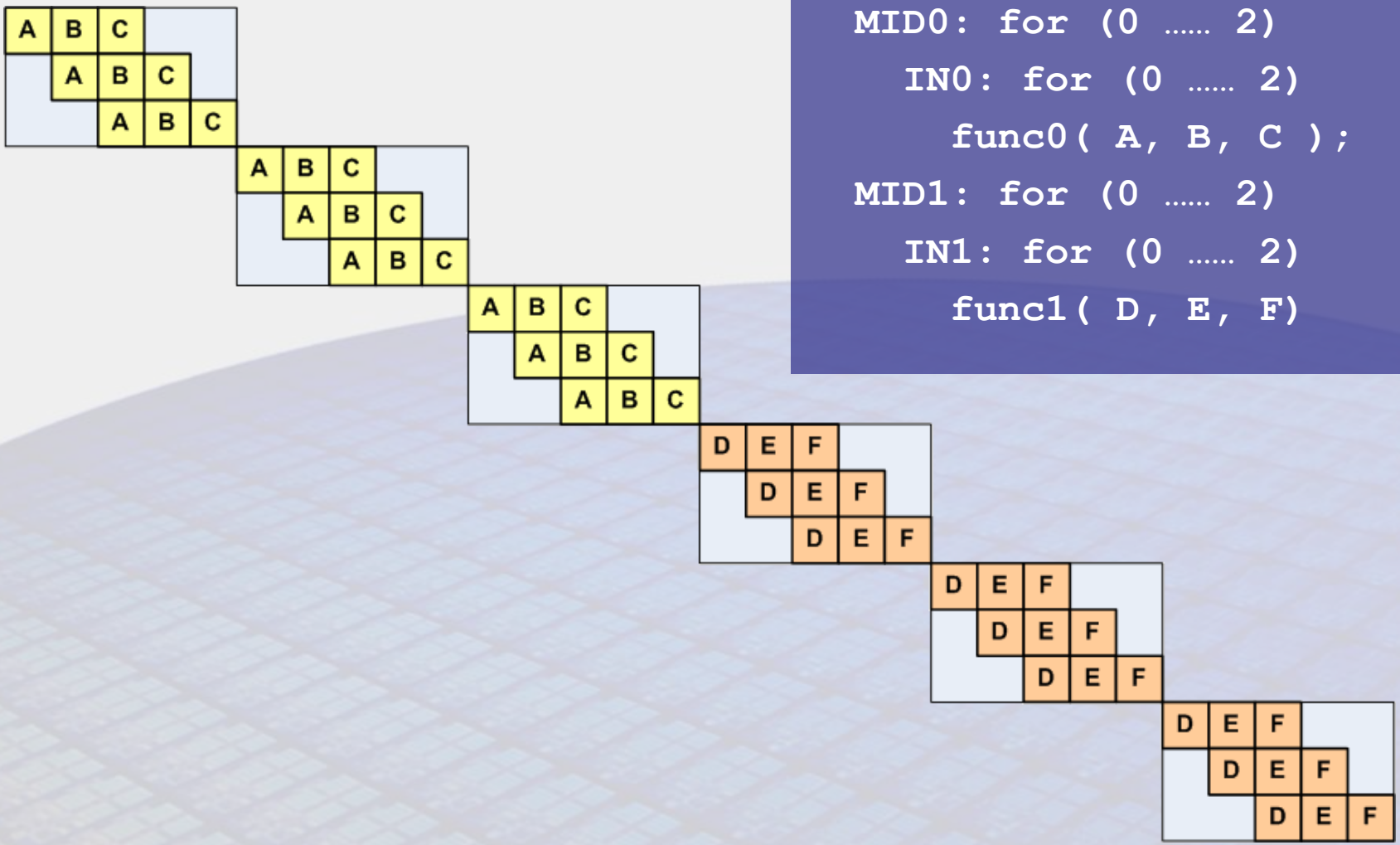
```
OUT: for (……)
  MID0: for (0 …… 1)
    IN0: for (0 …… 2)
      func0( A, B, C );
  MID1: for (0 …… 1)
    IN1: for (0 …… 2)
      func1( D, E, F)
```

- Nincs pipeline



Egymásba ágyazott ciklusok

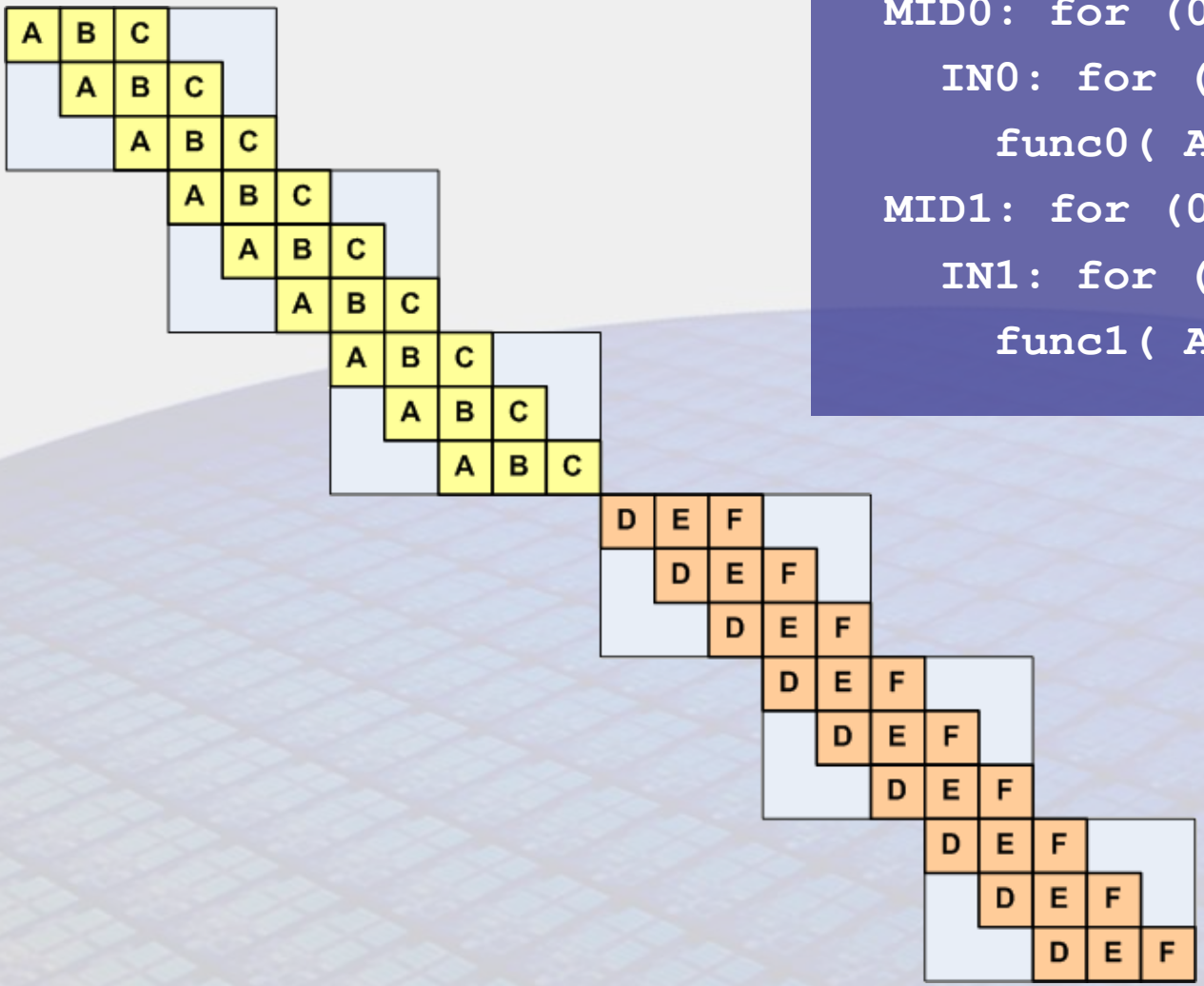
IN0 és IN1: II = 1




```
OUT: for (……)
  MID0: for (0 …… 2)
    IN0: for (0 …… 2)
      func0( A, B, C );
  MID1: for (0 …… 2)
    IN1: for (0 …… 2)
      func1( D, E, F)
```

Egymásba ágyazott ciklusok

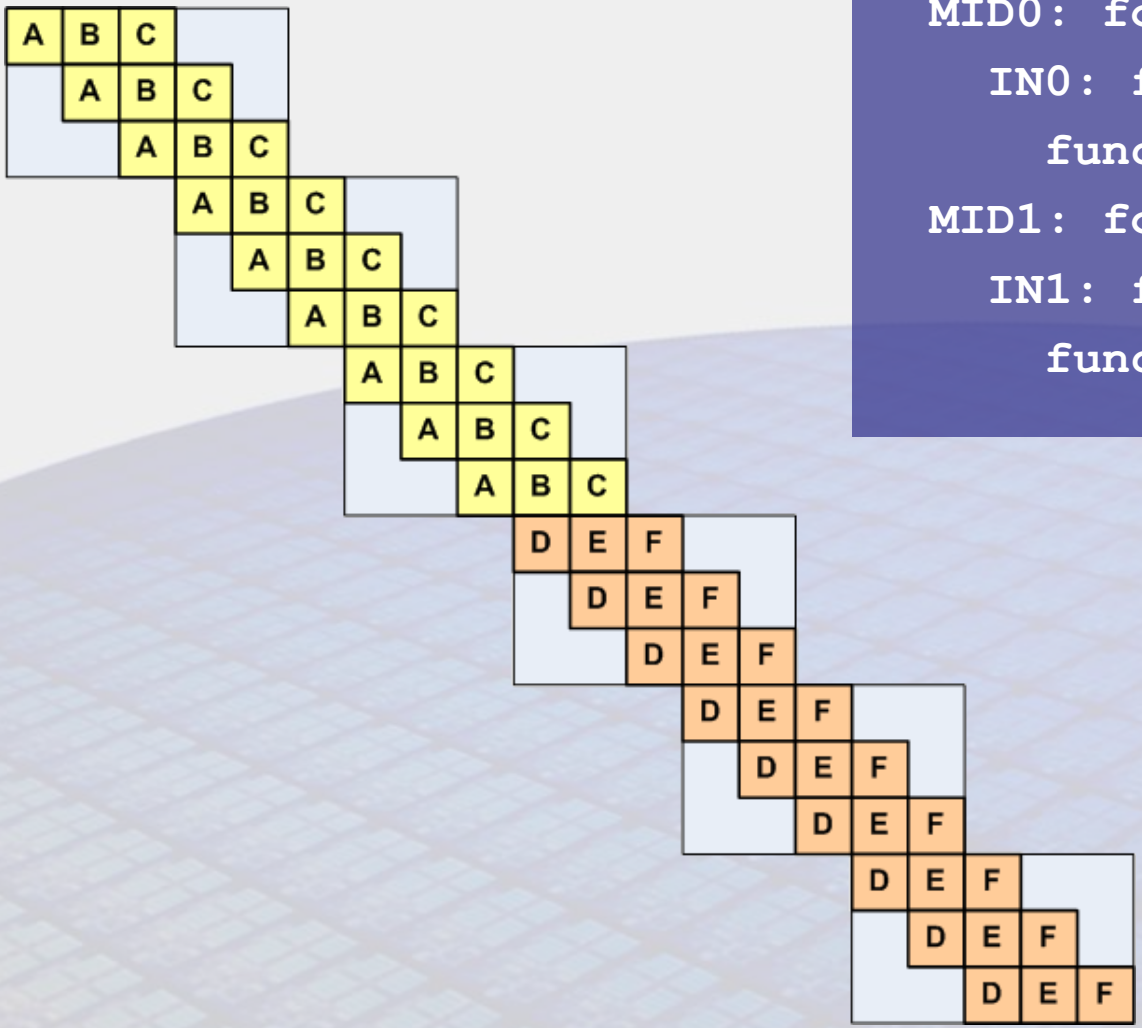
MID0 és MID1: II = 1




```
OUT: for (……)
  MID0: for (0 …… 2)
    IN0: for (0 …… 2)
      func0( A, B, C );
  MID1: for (0 …… 2)
    IN1: for (0 …… 2)
      func1( A, B, C)
```

Egymásba ágyazott ciklusok

OUT: II = 1




```
OUT: for (……)
  MID0: for (0 …… 2)
    IN0: for (0 …… 2)
      func0( A, B, C );
  MID1: for (0 …… 2)
    IN1: for (0 …… 2)
      func1( A, B, C)
```

Példa – design flow

- **Setup Design**
  - Cél eszköz, elérhető primitívek
  - Cél működési frekvencia
  - Reset beállítás
- **Architecture Constraints**
  - Interfész szintézis (handshake, tömb – RAM összerendelés)
  - Ciklus tulajdonságok beállítása (unroll, pipeline, …)
- **Schedule**
  - Ütemezés → eredmény Gantt táblában

Példa

```
void main0 (
    d_type a[4], d_type b[4], d_type c[4], d_type d[4],
    d_type q[4]
)
{
    for (int i=0; i<4; i++)
        q[i] = a[i] + b[i] + c[i] + d[i];
}
```

- órajel constraint hatása
- „wire” bemenet: szekvenciális, unroll
- „RAM” bemenet: szekvenciális, unroll
- RAM sávszélesség növelése

Példa: átlagolás

- Implementációra szánt függvény

```
#include "ac_fixed.h"
#include "ac_channel.h"
#include "type.h"
void avg(
   ac_channel< d_type > &data_in,
   ac_channel< d_type > &data_out
)
{
   if (data_in.available(2)) {
        d_type acc = 0 ;
        for (int i=0; i<2; i++)
                acc += data_in.read() ;
        data_out.write(acc >> 1) ;
   }
}
```

Mentor AC header file-ok

Saját típus deklaráció

Be- és kimeneti channel

Átlagolás
Channel írás

Példa: átlagolás

- Testbench

```
int main(){
   static ac_channel< d_type > data_in;
   static ac_channel< d_type > data_out;
   for (int i=0;i<8;i++){
        int z = i ;
        data_in.write(z) ;
        cout << "writing " << z << endl;
        avg(data_in, data_out) ;
   }
   cout << endl << "emptying stream" ;
   cout << endl << endl;
   do {
        cout << data_out.read() << endl;
   } while (data_out.available(1)) ;
}
```

!! include !!

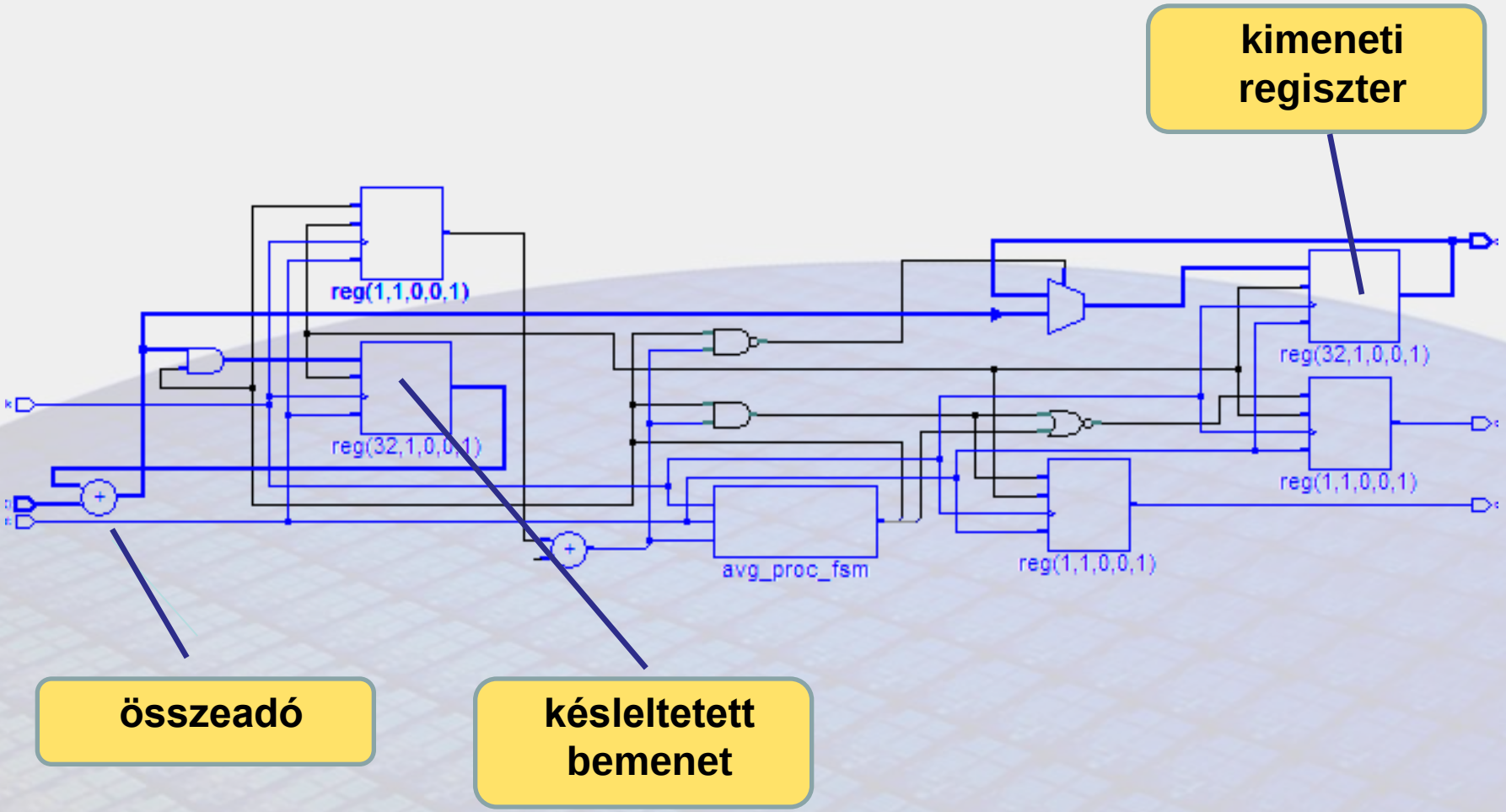
Channel deklarációk

Bemeneti adatok generálása
avg meghívása

Kimeneti stream kiírása

Példa: átlagolás

- Az avg függvényből generált hardver



1. gyakorlat

- **64 TAP-es FIR szűrő megvalósítása**
  - Lebegőpontos modell C-ben
  - Catapult-C implementációra szánt fix pontos verzió
  - C Testbench
    - Gerjesztés előállítása
    - Lebegőpontos & fixpontos verzió összehasonlítása
- **FIR szűrő**
  - Új bemeneti adat beolvasása shift regiszterbe
  - TAP számú MAC művelet végrehajtása akkumulátoron
  - Eredmény kiírása