BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

Konfigurálható processzorok

Fehér Béla
BME MIT

Adatfeldolgozó egységek

- **Hatékonyság ↔ Flexibilitás**
  - Egyedi HW :
    - Adott feladat igényei szerint kialakított műveletvégzők, adatutak, regiszterek
    - Jellemzően adatfolyam típusú működés
  - Általános célú adatfeldolgozó egységek:
    - Központi műveletvégző (ALU), regiszter tömb, I/O
    - Jellemzően utasítás sorozat végrehajtás
- **Cél: Általános struktúra + magas hatékonyság**

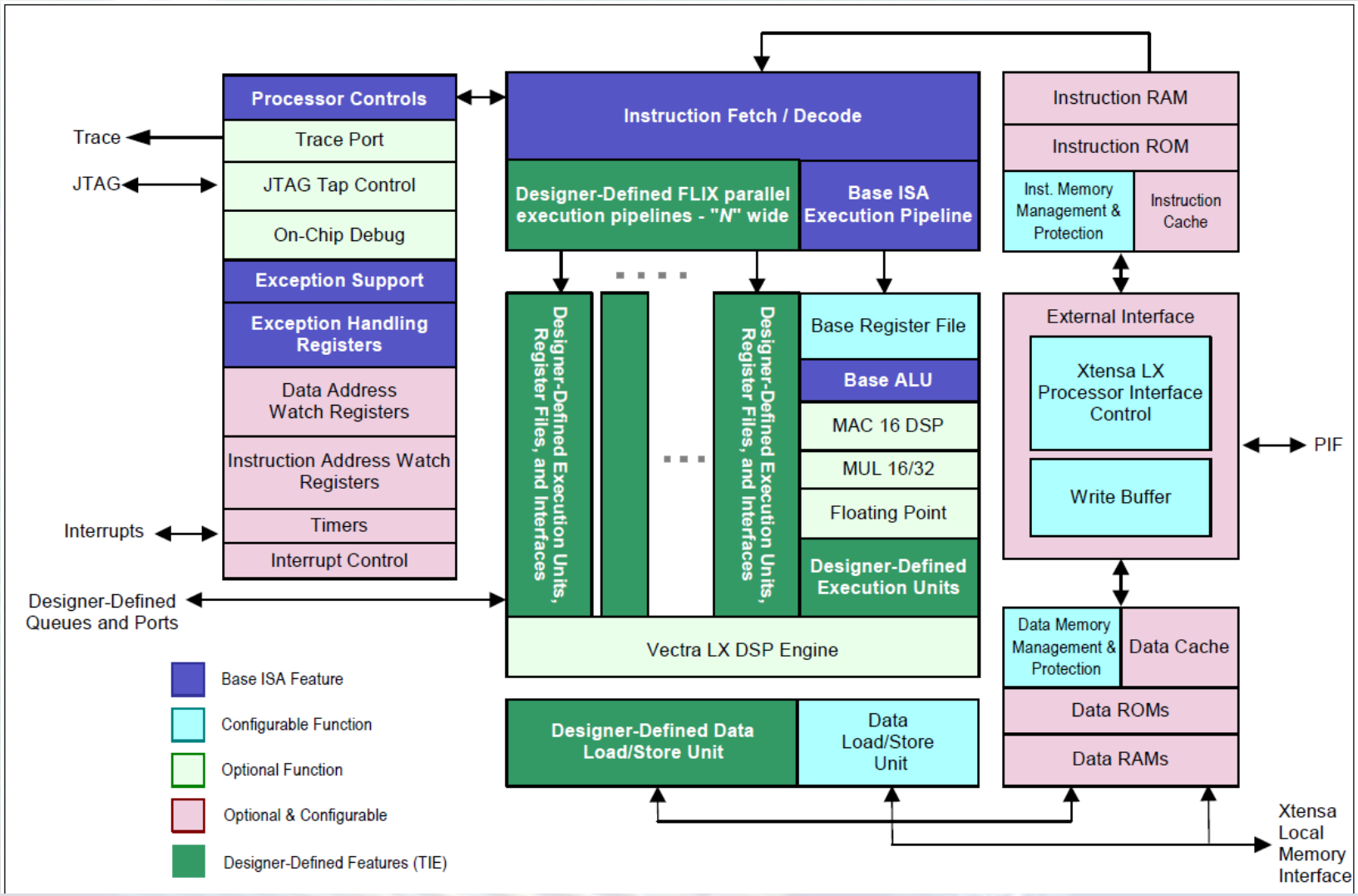
Mikroprocesszorok

- **Jó kompromisszum**
  - Tipikus feladatok közel optimális leképezése
  - Közepes igények megfelelő kiszolgálása
- **Magasabb követelmények esetén**
  - Komponensek többszörözése: multi – x
  - Speciális kiegészítések: HW bővítés (MUL, MAC)
  - Legfontosabb műveletek közvetlen beépítése, párhuzamosíthatóság támogatása
- **Nehézségek**
  - Nem lehet minden követelményt kielégíteni
  - Bizonyos területek egyedi megoldásokat, szolgáltatásokat igényelnének

Konfigurálható processzorok

- **Az általános jellemzők megválaszthatók**
  - Utasításkészlet
    - Egyedi utasítások (pl. POP_COUNT)
    - Ko-processzor jellegű egységek (pixel blokk, packet handler)
    - DSP és HPC (High Performance Computing) elemek, vektor/SIMD/lebegőpontos kiegészítés
  - Memória alrendszer
    - Utasítás és adat cache (I$/D$), utasítás és adat méret, védelem
  - Interfészek
    - Külső, belső elemek, sorok, több portos eszközök
  - Perifériák
    - Időzítők, IT vezérlő és kivételkezelés, fejlesztési támogatás

Tensilica Xtensa processzor



Alapvető jellemzők

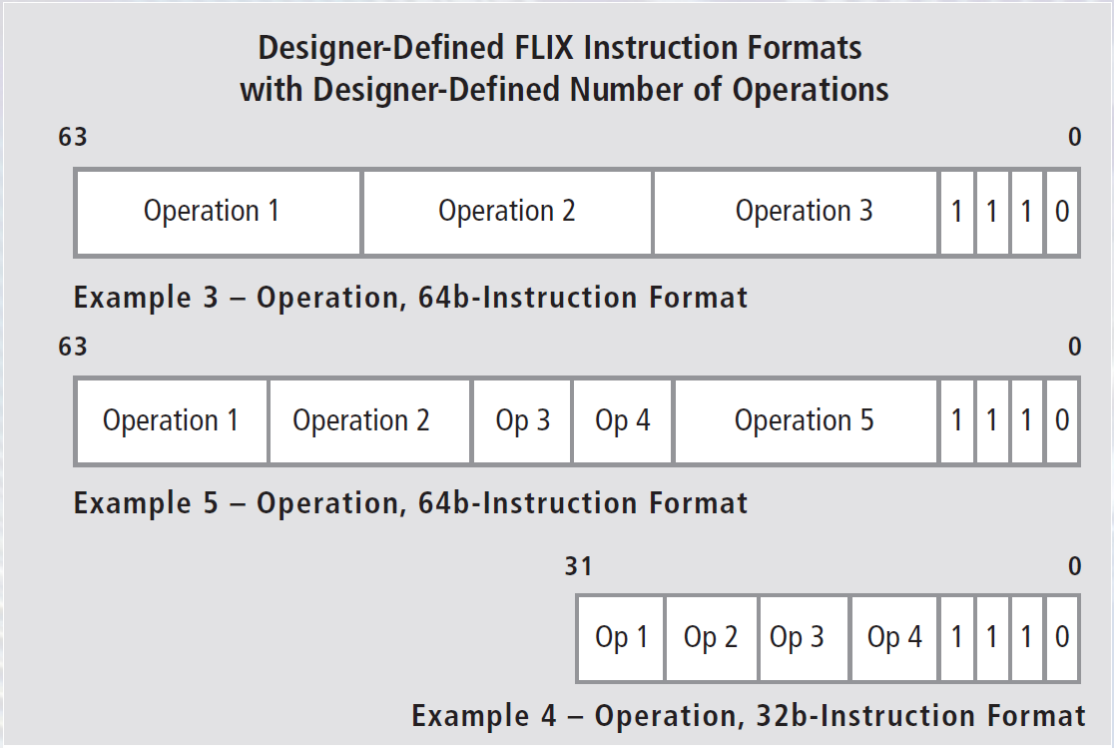
- **Normál processzor részegységek**
  - Utasítás kezelés (Fetch-Decode-Execute)
  - Általános vezérlés, kivételkezelés
  - Regiszterkészlet, általános utasítások
- **Konfigurálható funkciók**
  - Belső utasításmemória és tulajdonságai
  - Belső adatmemória és tulajdonságai
  - Külső processzor interfész
  - Load/Store egység
- **Opcionális, választható/elhagyható elemek**
  - ROM/RAM/Cache memória verziók
  - Debug támogatás

Speciális tulajdonság

- **TIE Tensilica Instruction Extension**
  - A Verilog nyelv általános tulajdonságaira épít
  - Néhány soros specifikációval speciális kiegészítések adhatók a processzor adatfeldolgozóhoz
    - Egyedi adatméretek a buszsávszélesség maximális kihasználásra
    - Utasítások egymásba olvasztása a hatékonyabb működéshez
    - SIMD regiszterek és műveletek definiálása
    - Több műveletes, megnövelt hosszúságú FLIX, Flexible Length Intsruction eXtensions) utasítások (VLIW)
    - Adatátviteli csatornák, pont-pont kapcsolatokhoz, sorok, FIFO-k, táblázatos hozzáférések

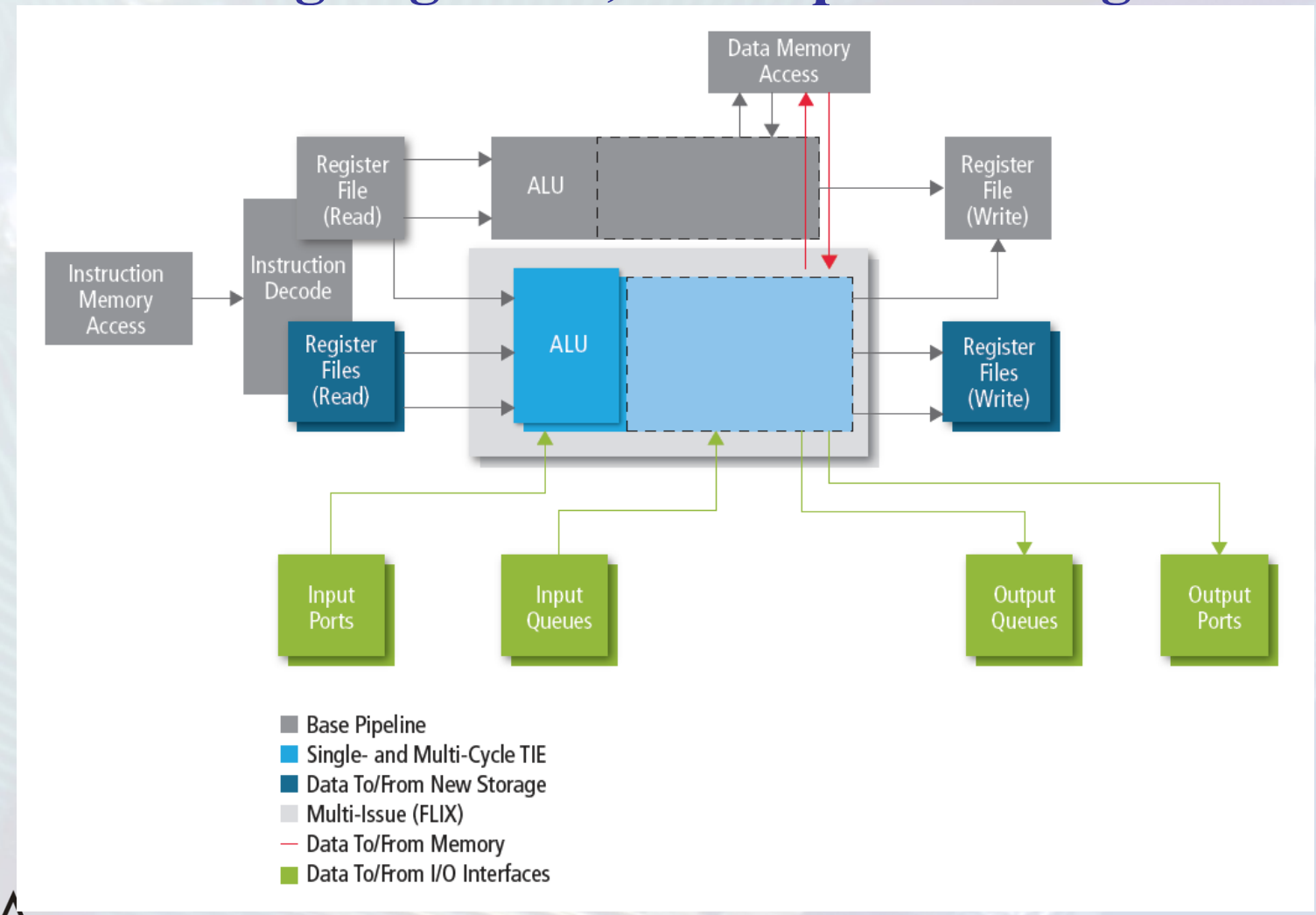
FLIX Flexible Length Instruction eXtensions

- **VLIW adatfeldolgozó egységek tervezhetősége**
  - 2 -30 db” tetszőleges típusú/számú feldogozó egység kezelése egyetlen (akár 128 bit hosszú) utasításban
  - Alaputasítások 16 -24 bitesek
  - Egységes kezelés, nincs „overhead”
  - A kiválasztott/ beépített elemek használata automatikus, a fordító kezeli



Alapelv

- **A CPU mag rugalmas, transzparens kiegészítése**

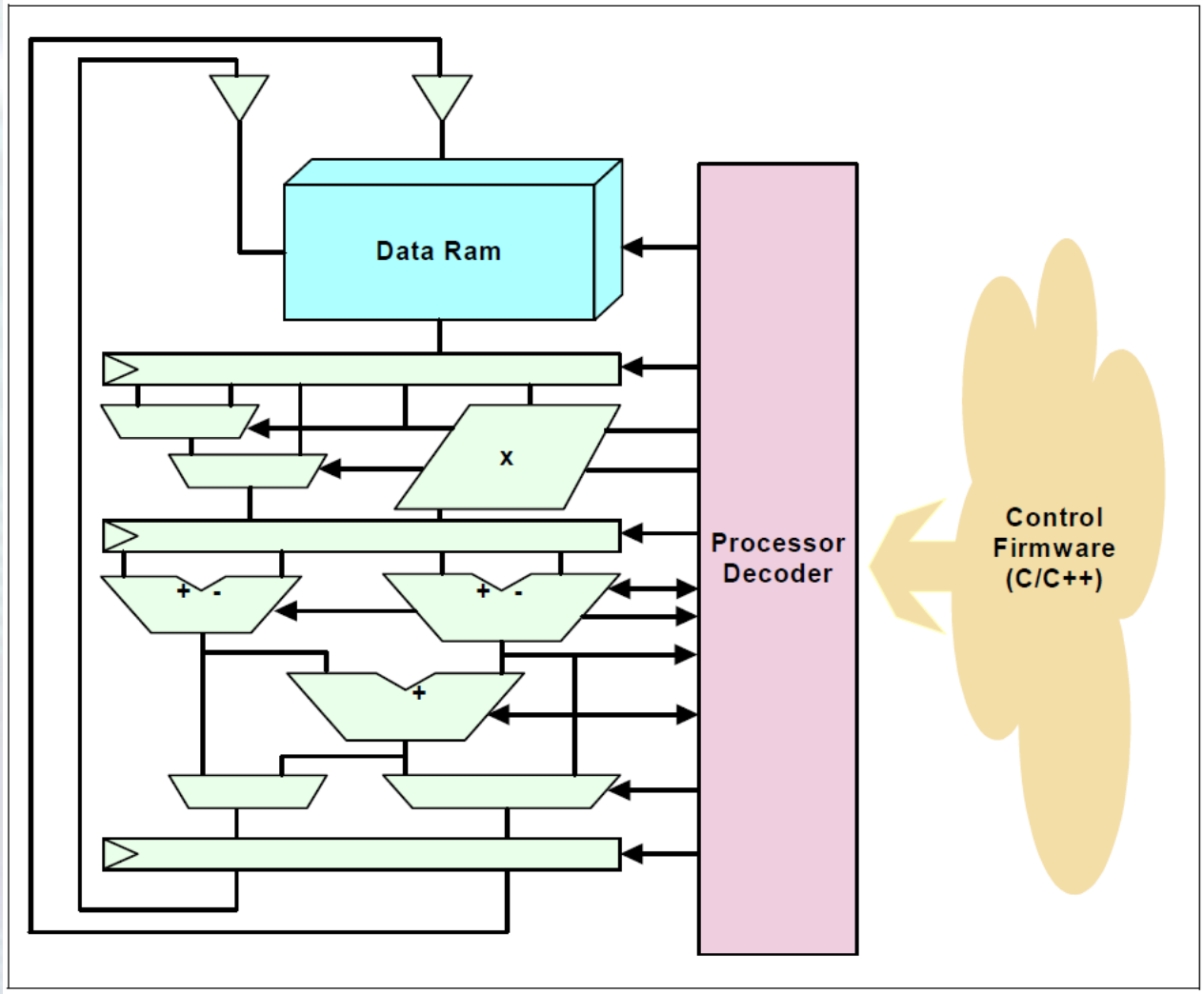


A TIE fejlesztési technológia

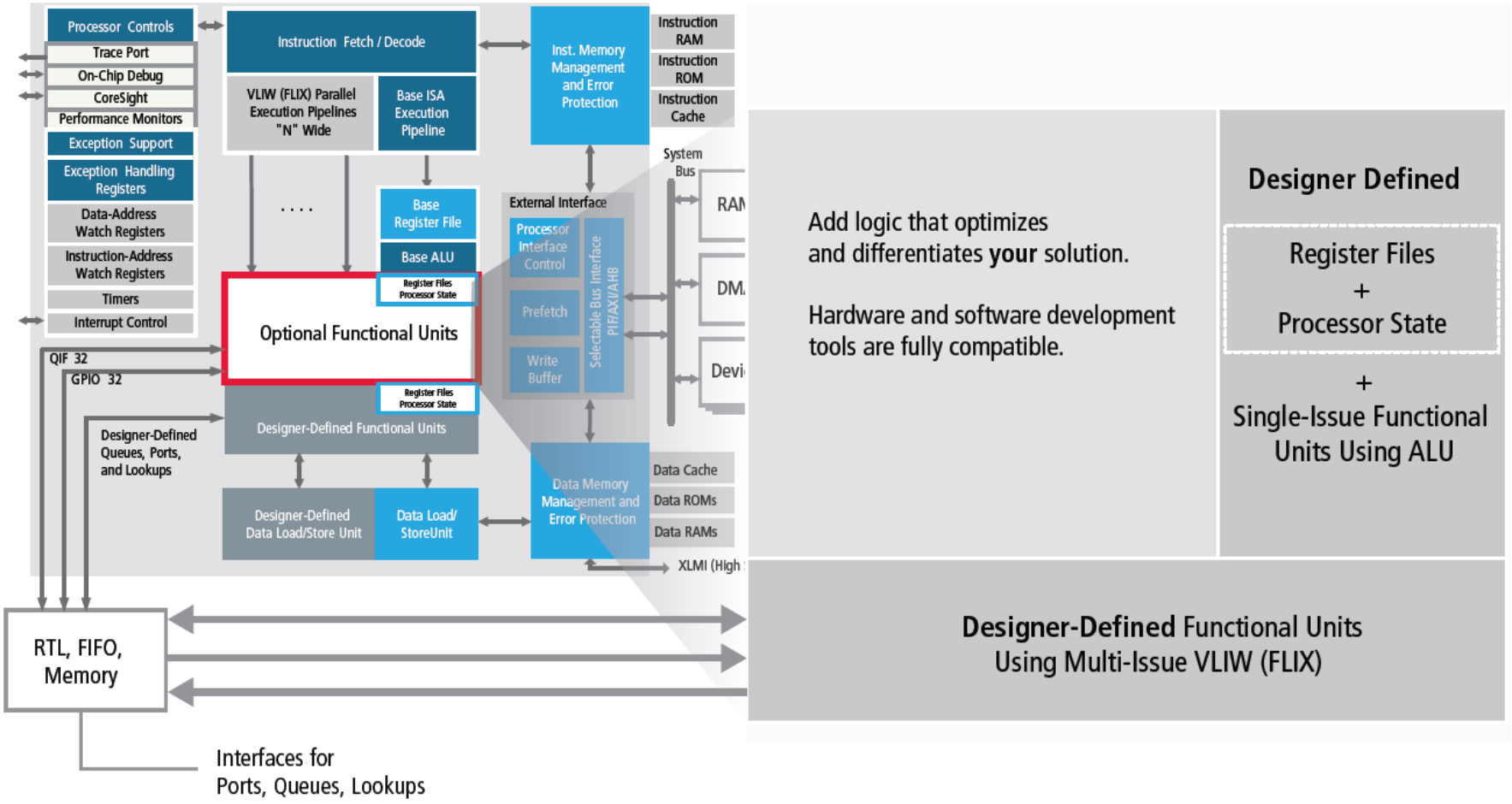
- A processzortervezés komoly felkészültséget igényel
- Az egyedi alacsonyszintű HW RTL egységek tervezése a HDL nyelvek használatával jó minőségben modellezhető, megvalósítható
- Alapvetően csak az adatfeldolgozó egységet kell feladatorientáltra szabni, a vezérlés beilleszthető, realizálható az általános modell szerint
- A szükséges kiegészítések automatikusan generálhatók és beilleszthetők
- A kiegészítő egységek hagyományos HW RTL blokkonként működnek, a vezérlésükről speciális firmware gondoskodik

A TIE fejlesztési technológia

- **Példa a speciális programozható rendszerre**



A TIE fejlesztési technológia

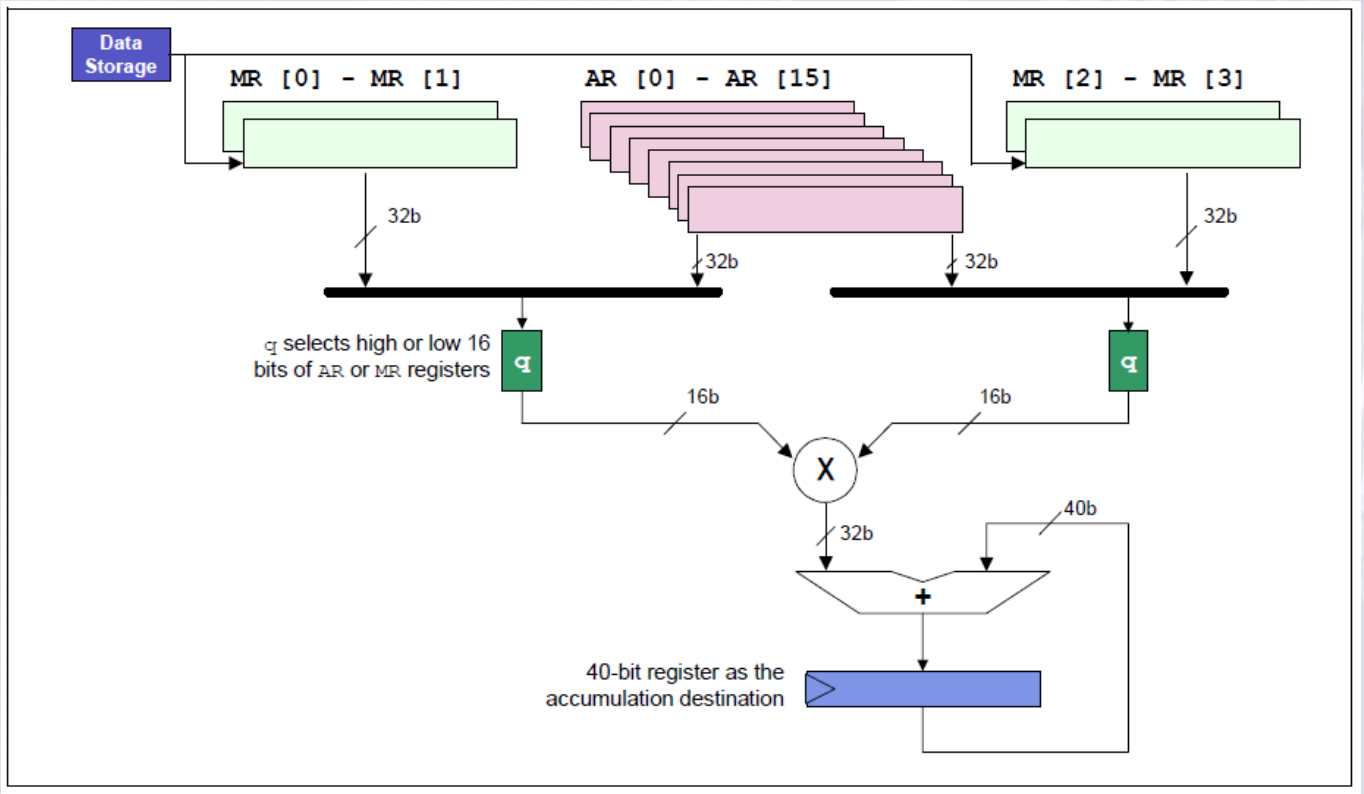


A TIE fejlesztési technológia

- **A programozott vezérlés előnyei:**
  - Rugalmasság, a felépített adatfeldolgozó HW-re bárki, CPU tervező/rendszerfejlesztő/végfelhasználó készíthet új alkalmazást firmware fejlesztéssel
  - Egyszerű szoftver alapú fejlesztési technológia
  - Gyors, könnyen ellenőrizhető, modellezhető
  - Egységesített SW/HW fejlesztői környezet, nem különül el a vezérlés és az adatfeldolgozás
  - Könnyű módosíthatóság
  - Gyors fejlesztési idő

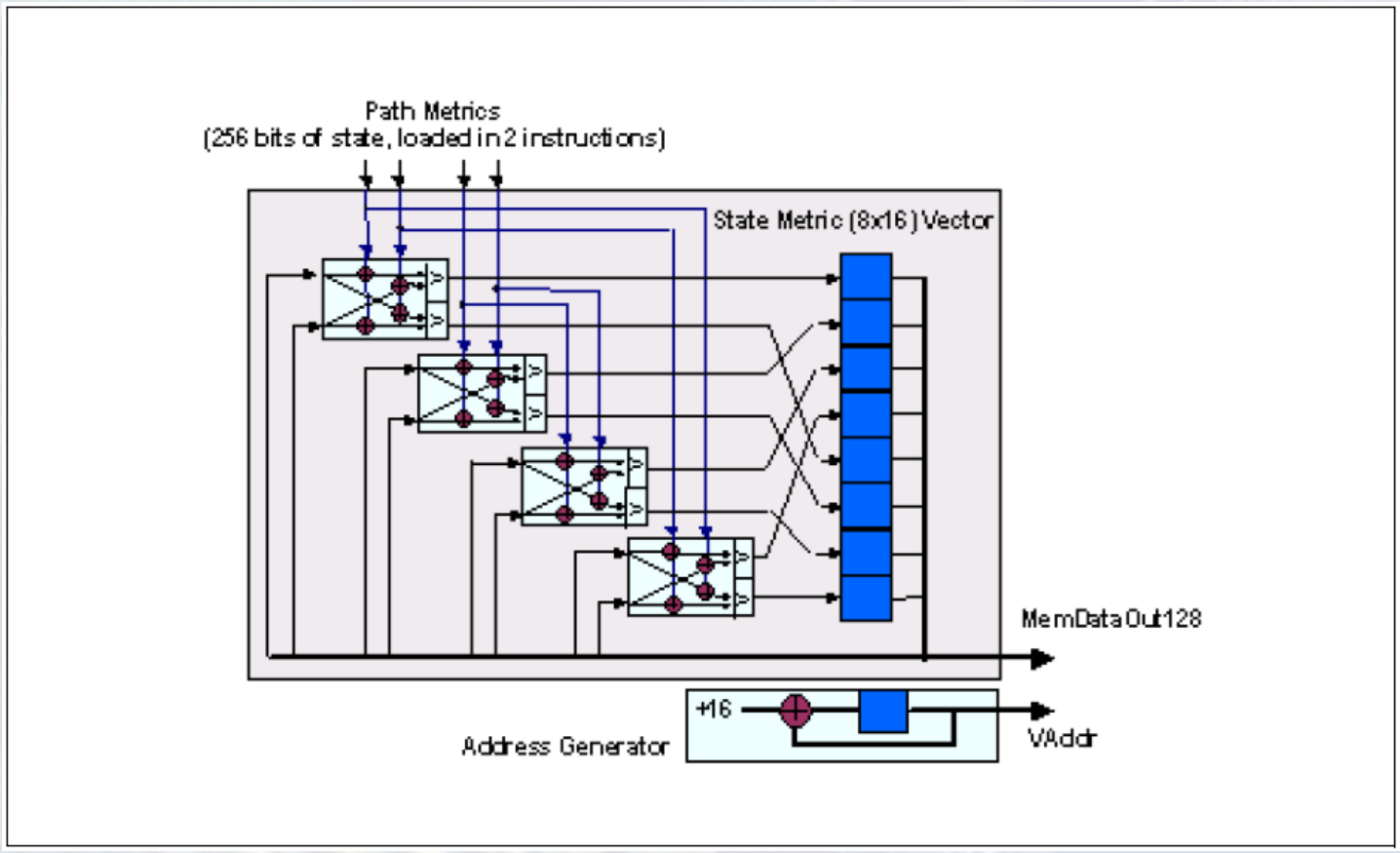
A TIE fejlesztési technológia

- **Példa 1: GSM audió kodek hangolása**
  - Előzmény: Szoftver kód analízis az alap CPU-n 80% végrehajtási idő a szorzás végrehajtására → HW szorzó szükséges 7x javulás



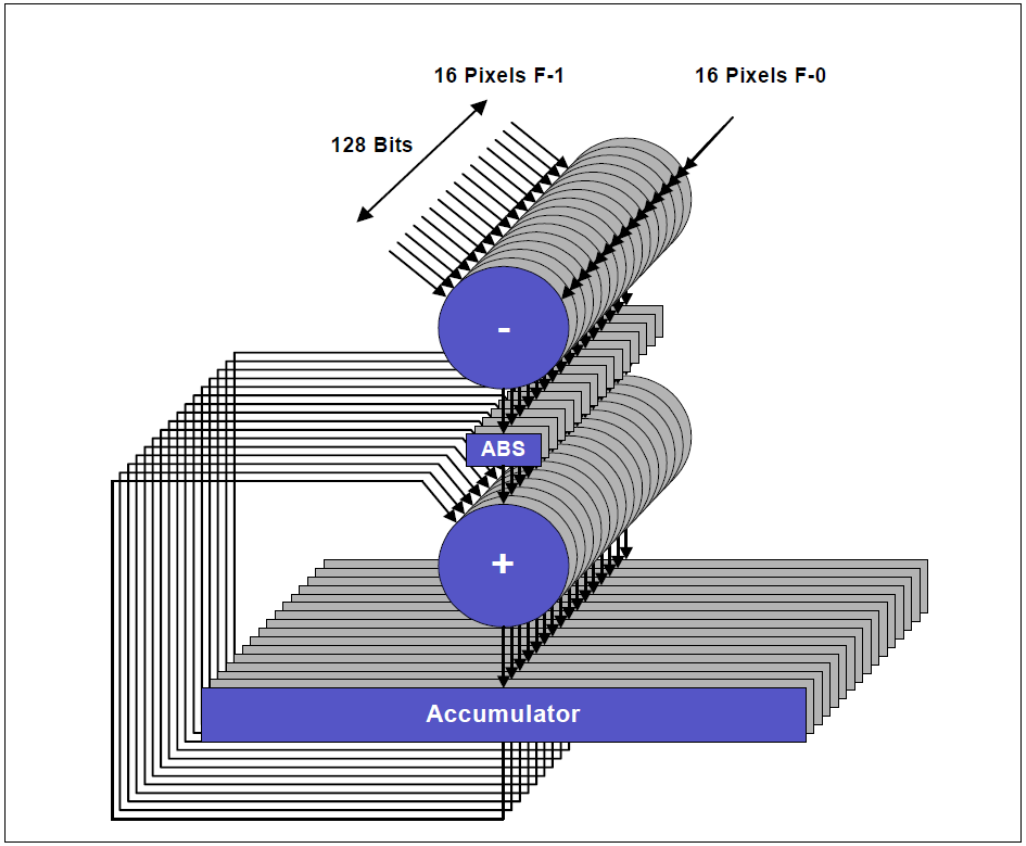
A TIE fejlesztési technológia

- **Példa 2: Viterbi dekódolás javítása**
  - Követelmény: a speciális „pillangó” blokk (4 ADD, 2 COMP, 2SEL) gyorsítása, párhuzamosítása 250x



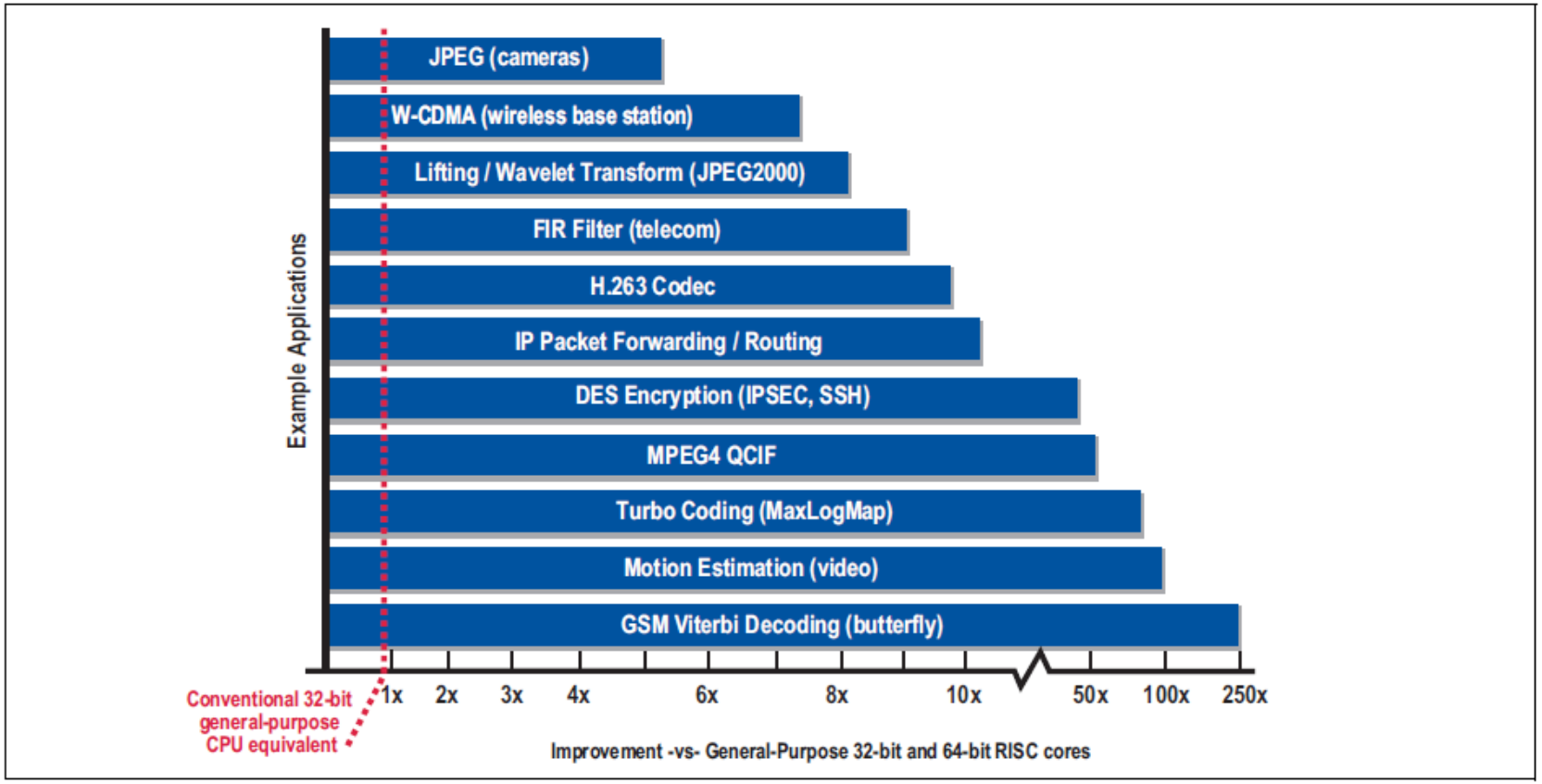
A TIE fejlesztési technológia

- **Példa 3: MPEG4 mozgás becslés gyorsítása**
  - Követelmény: Az SAD (SUM-of-ABS-DIF) párhuzamosítása SIMD műveletvégzővel 98% javulás

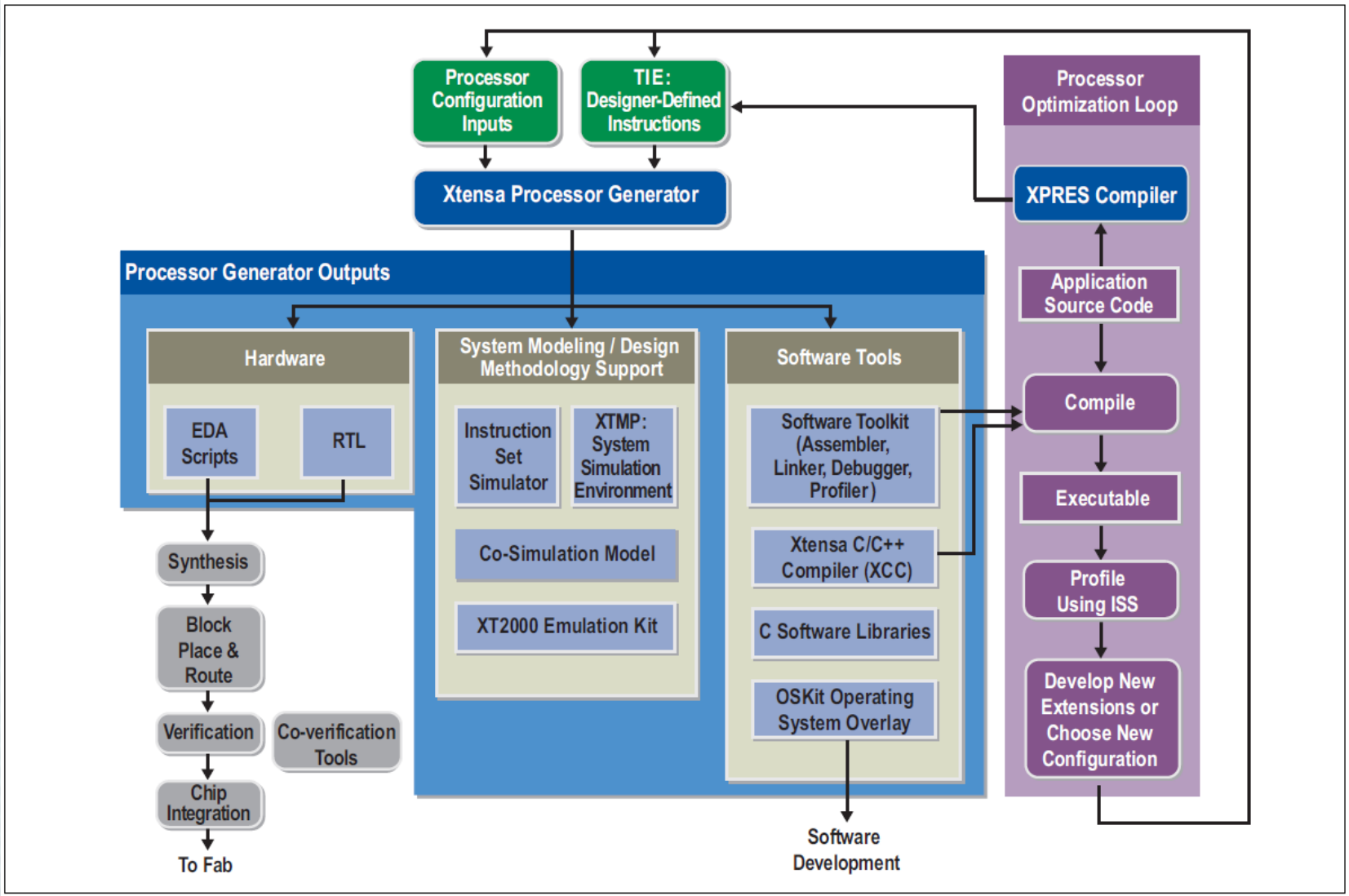


A TIE fejlesztési technológia

- **Összegzés:**
- **Egyszerű bővíthetőség, jelentős teljesítmény javulás**
- **Részben automatikus HW és SW fejlesztés**

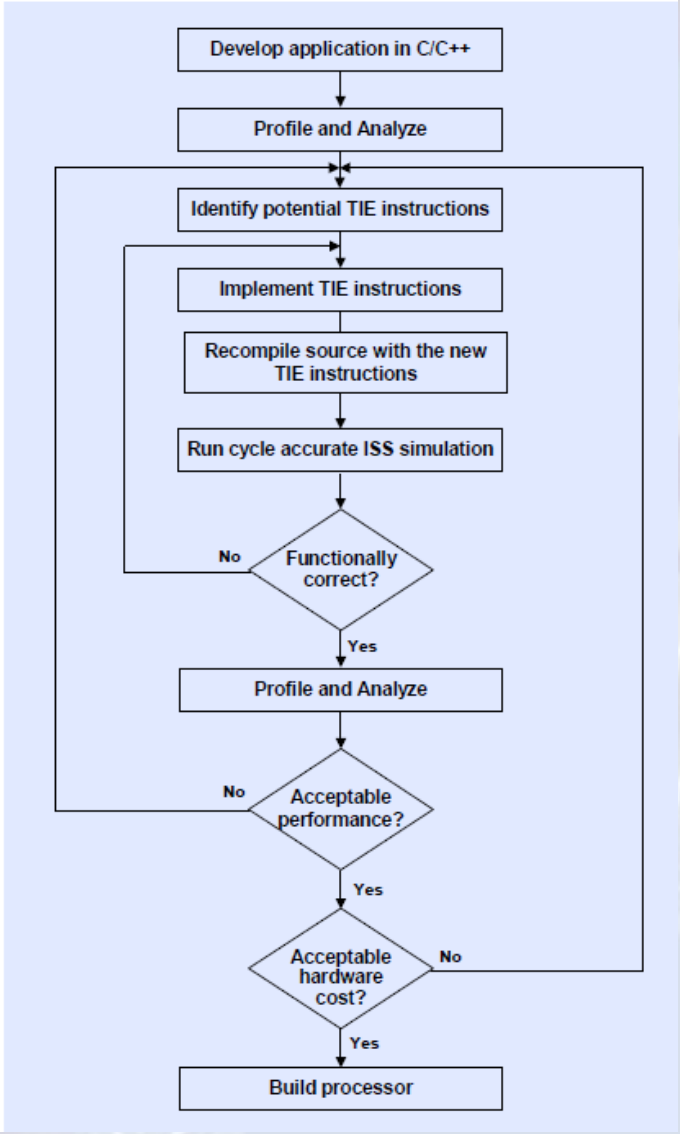


Xtensa Processzor Generator



A TIE fejlesztési technológia

- A TIE leíró nyelv segítségével a kritikus művelet hatékonysága javítható
- Iteratív ellenőrzési lehetőség, költség/teljesítmény elemzés
- Szükséges és elégséges HW kiegészítés beépítése
- Az adatfolyam jellegű HDL leírás alapján a HW generálása
- A megtervezett HW vezérlése az alap CPU utasítás készletének kiegészítése és a firmware funkciók elkészítése által



A TIE fejlesztési technológia

- **A fejlesztés az alkalmazás SW elkészítésével indul**
- **A műveleteket az alap CPU utasításival realizáljuk**
- **Igény szerint ezek a műveletek képezhetők le HW műveletvégzőkre**
- **Például több művelet összevonása egyetlen utasításba**
  - Két vektor átlagolása
  - Elemenként 2 utasítás

```
unsigned short *a, *b, *c;
...
for (i=0; i<n; i++)
     c[i] = (a[i] + b[i]) >> 1;
```

```
…
add.n   a9,a11,a10
srli    a9,a9,1
…
}
```

- **Ez a művelet HW-ben könnyen leírható, hatékonyan realizálható, a magja egyetlen ciklus alatt elvégezhető**

A TIE fejlesztési technológia

- **A művelet SW kódja (AR címző regiszter típus:**

```
operation addshift {out AR avg, in AR A, in AR B} {}
{
   assign avg = (A  + B) >> 1 ;
}
```

- **Ennek TIE nyelvű leírása:**

```
operation AVERAGE{out AR res, in AR input0, in AR input1} {} {
     wire [16:0] tmp = input0[15:0] + input1[15:0];
     assign res = tmp[16:1];
}
```

- **A fordítás után 1 CPU utasítás**

```
   …
   addshift  a12,a10,a8;
   …
   }
```

A TIE fejlesztési technológia

- **Adatszintű párhuzamosítás (SIMD)**
  - Két vektor összegzése
  - Az adatok 16 bitesek
  - Túlcsordulás nem lép fel

```
unsigned short
  A[VECLEN],
  B[VECLEN],
  sum[VECLEN] ;
for (i = 0; i < VECLEN; i++) {
   sum[i] = A[i] + B[i] ;
}
```

- **Cél: Párhuzamosítás a ciklusmag végrehajtására**
  - Egyidejűleg több operandus elérése: SIMD regfájl
  - Egyidejűleg több operandus összeadása: SIMD ADD

A TIE fejlesztési technológia

- **A generátor által előállított HW leírás:**

```
regfile simd64 64 16 v      // 16-entry register file that is 64 bits wide
operation vec4_add16 { out simd64 res, in simd64 A, in simd64 B } { }
{
    wire [15:0] rtmp1 = A[15: 0] + B[15: 0] ;
    wire [15:0] rtmp2 = A[31:16] + B[31:16] ;
    wire [15:0] rtmp3 = A[47:32] + B[47:32] ;
    wire [15:0] rtmp4 = A[63:48] + B[63:48] ;
    assign res = { rtmp4, rtmp3, rtmp2, rtmp1 } ;
}
```

- **Pontosan megvalósítva az előírt követelményeket**
  - Egyidejűleg több operandus elérése: SIMD regfájl
  - Egyidejűleg több operandus összeadása: SIMD ADD
  - A TIE egység használata a forráskódban

```
simd64 A[VECLEN/4]; // Input vectors
simd64 B[VECLEN/4]; // Input vectors
simd64 sum[VECLEN/4]; // Output vectors
for (i=0; i<VECLEN/4; i++){
   sum[i] = vec4_add16(A[i],B[i]);
}
```

A TIE fejlesztési technológia

- **Funkcionális párhuzamosítás leírása (FLIX)**
  - Több művelet egyetlen utasításba csomagolása
  - 32 – 128 bit széleségű utasítás kódok, 8 bites skálázással (32, 40, .. 88, 112, 120, 128, igény szerint)
  - Valódi VLIW utasításvégrehajtás, a fix méretű VLIW extrém nagy kódméretigénye nélkül
  - Példa kód:
    - Vektorok átlagolása
    - A teljes ciklus az alap CPU-n 8 utasításra fordul

```
for (i=0; i<n; i++)
  c[i]= (a[i]+b[i])>>2;
```

```
loop:
  …
  addi  a9,  a9,  4;
  addi  a11, a11, 4;
  l32i  a8,  a9,  0;
  l32i  a10, a11, 0;
  add   a12, a10, a8;
  srai  a12, a12, 2;
  addi  a13, a13, 4;
  s32i  a12, a13, 0;
```

A TIE fejlesztési technológia

- **FLIX használatával néhány utasítás párhuzamos végrehajtása is előírható**
  - 64 bites VLIW utasítás formátum
  - 3 funkció időbeli párhuzamosítása

```
format flix3 64 {slot0, slot1, slot2}

slot_opcodes slot0 {L16I, S16I}
slot_opcodes slot1 {ADDI}
slot_opcodes slot2 {ADD, SRAI}
```

  - A középső műveletvégző itt nem lesz 100%-ban kihasználva

A TIE fejlesztési technológia

- **FLIX használatával néhány utasítás párhuzamos végrehajtása is előírható**
- **A VLIW egység használata persze bonyolultabb**
  - Az ilyen párhuzamosítás a ciklusok pipe-line végrehajtásához az előkészítés és kifutás kezelését igényli (nincs részletezve)

```
loop:
{addi a9,a9,4;      add  a12,a10,a8;    l16i a8,a9,0   }
{addi a11,a11,4;    srai a12,a12,1;     l16i a10,a11,0 }
{addi a13,a13,4;    nop;                s16i a12,a13,0 }
```

  - A 3-szoros párhuzamosítás ciklusmagjának kódja összesen 3 utasításból áll, 8 helyett
  - A NOP itt üres utasítást jelent, de esetleg más funkció elhelyezhető lenne a helyén

A TIE fejlesztési technológia

- **Láttunk 4 módszert a vektorok átlagolására**
  - Az alap CPU utasításkészlet használata
  - Elemi utasítások egyesítése
  - SIMD jellegű vektorművelet
  - VLIW jhellegű párhuzamosítás

- **A legjobb lehetőség kiválasztása az alkalmazástól függhet**
  - Adat vektorok mérete
  - Adat vektorok elemeinek mérete
  - Elérhetőségük a memóriában

A TIE fejlesztési technológia

- **Végül egy szép példa**
  - POP_COUNT utasítás: Az adatban lévő 1 értékű bitek száma
  - A primitív megoldás, 32 iteráció kb. 70 utasítás ciklus

```
// unsigned int x contains input value

unsigned int y=0;
unsigned int i;
for(i = 0; i < 32; i++) {
  if((x&1) == 1) y++;                  // Increment counter if bit is set
  x = x >> 1;                          // Shift to prepare for next bit
}
// y contains result
```

  - Determinisztikus végrehajtás, biztos felesleges ciklusok az utolsó „1” feldolgozása után
  - További lehetőségek/trükkök?

A TIE fejlesztési technológia

- **Végül egy szép példa**
  - POP_COUNT utasítás: Az adatban lévő 1 értékű bitek száma
  - Egy ügyesebb megoldás
    - Csak annyi ciklus, amennyi „1” értékű bit van
    - Kevés „1”-es esetén szuper!
    - De nem rosszabb, mint az előző

```
// x contains input value
// count returns pop count

unsigned int x;
unsigned int count;
for (count=0; x!=0; count++) {
    x &= x-1;
}
// count contains result
```

- **A ciklusmag 2 utasítás SUB_AND, ezek egyesíthetők**
  - Ez további előnyt jelent

```
operation DECR_AND {out AR count, in AR x}{}{
    assign count = x & (x -1);
}
```

A TIE fejlesztési technológia

- **Végül egy szép példa**
  - De még ennél is jobb a HW POP_COUNT modul
  - A teljes függvény egy egységben realizálva

```
// unsigned int x contains input value

unsigned int y = POP _ COUNT(x); // single cycle
// y contains result
```

  - Ennek TIE leírása:

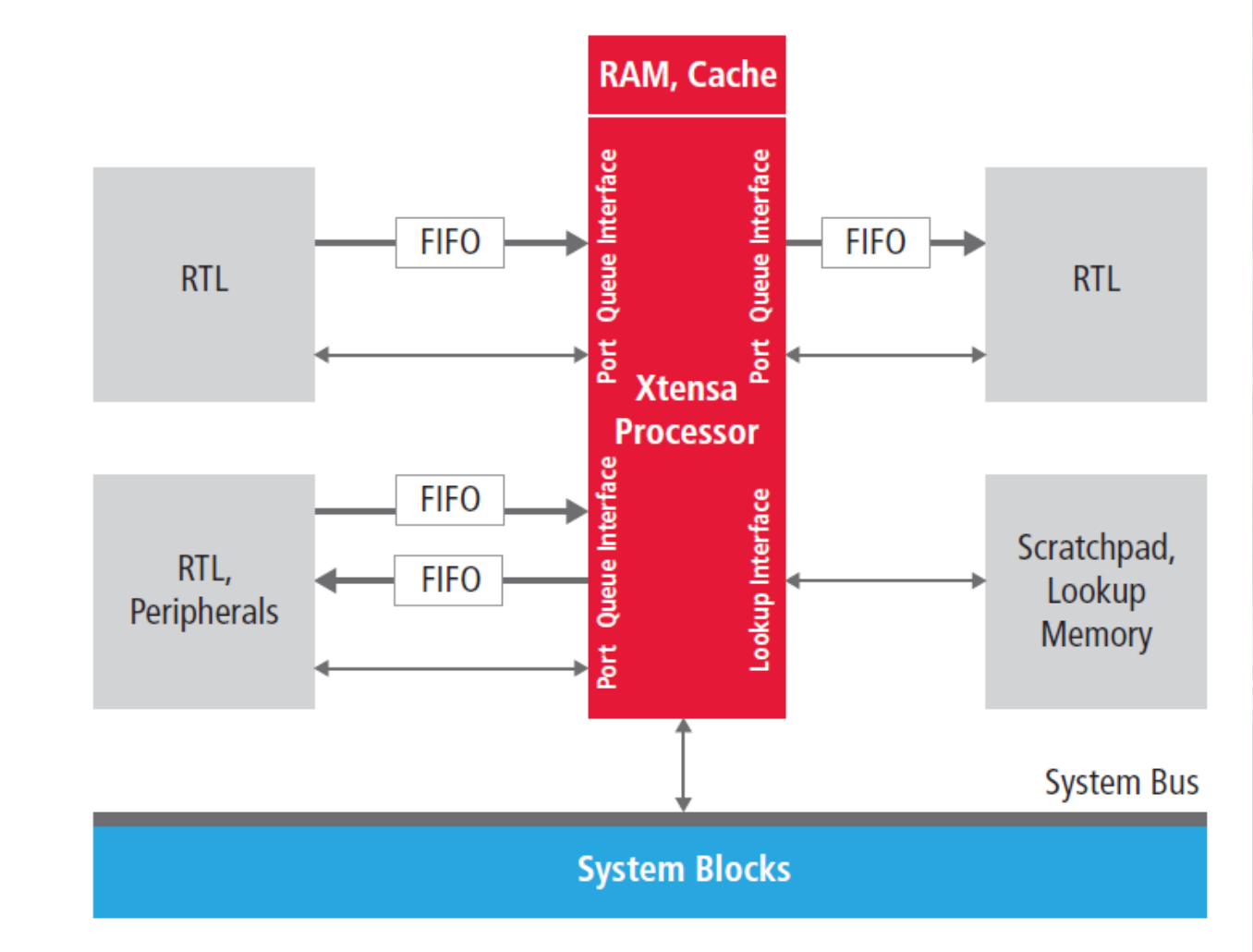
```
operation pop _ count {out AR co, in AR ci}{}{
    wire [3:0] a0 = ci[0] + ci[1] + ci[2] + ci[3] + ci[4] + ci[5] + ci[6] + ci[7];
    wire [3:0] a1 = ci[8]+ci[9] + ci[10] + ci[11] + ci[12] + ci[13] + ci[14] + ci[15];
    wire [3:0] a2 = ci[16]+ci[17]+ci[18] + ci[19] + ci[20] + ci[21] + ci[22] + ci[23];
    wire [3:0] a3 = ci[24]+ci[25]+ci[26] + ci[27] + ci[28] + ci[29] + ci[30] + ci[31];
    wire [5:0] sum = a0 + a1 + a2 + a3;

    assign co = {26'b0, sum};
}
```

További opciók

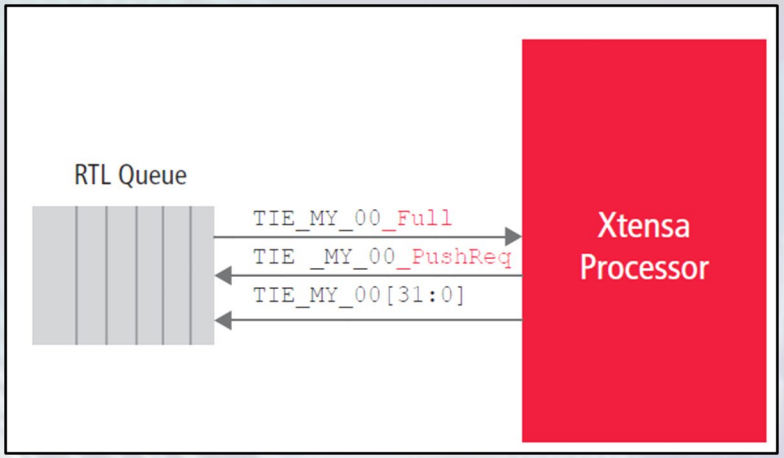
- A teljesítmény lehet sávszélesség korlátos
- Az adatregiszterek és memóriák elérése is javítható
- Sok esetben speciális adatelérési módszerek szükségesek
- Ezek SW-ben is megvalósíthatók, de jelentős késleltetés által

Xtensa processzor I/O interfészek



Portok beépítése

- A port segítségével az adatáramlás egyszerűen megvalósítható
- A specifikáció illeszkedik a modellhez
- A TIE leírás azt fogalmazza meg, amit használni szeretnénk
- Használata a forráskódban magától értetődő

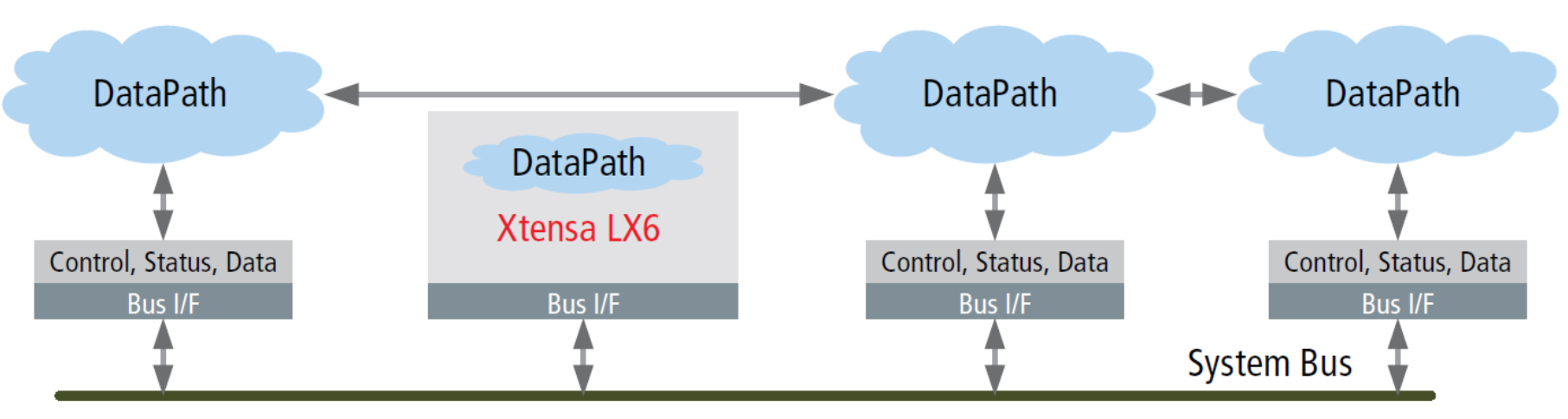



```
queue MY _ OQ 32 out
operation MY _ PushQ
   {in AR qdata}
   {out MY _ OQ} {
    assign MY _ OQ = qdata ;
   }
```

```
#include <xtensa/tie/outqueue.h>
…
int data = 12 ;
MY _ PushQ(data) ;
…
```

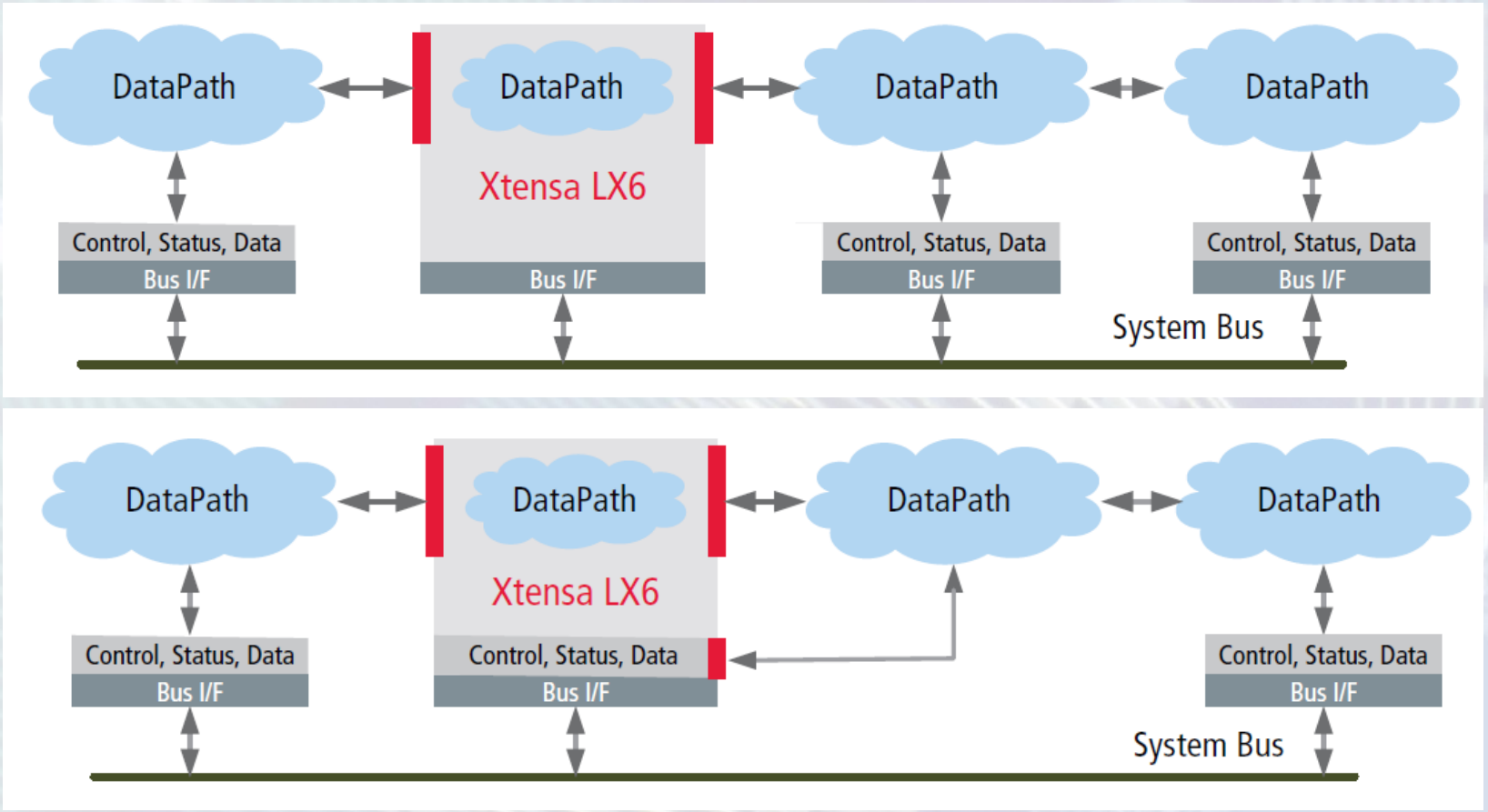
Portok beépítése

- **A portok segítségével a programozható CPU közvetlenül részt vehet az adatáramlási láncban**
  - Hagyományos megoldás, kommunikáció a rendszerbuszon keresztül



Portok beépítése

- **A portok segítségével a programozható CPU közvetlenül részt vehet az adatáramlási láncban**



Összegzés

- A Tensilica Xtensa konfigurálható processzor platform egy nagyon hatékony megvalósítás
- Az alkalmazás igényei szerint a processzor alaptulajdonságai kiegészíthetők, a követelményekhez hangolhatók.
- A kiegészítések kizárólag az adatfeldolgozó részt érintik, teljesítmény javulás alapvetően csak ott érhető el.
- Az alap és az egyedi CPU verziók hatékonysága profiling segítségével elemezhető, hangolható.
- Az eredmény egy gyártható SoC áramkör lesz

Összegzés

- A kialakított egyedi CPU-hoz azonnal generálódik magas szintű fordító
- A TIE ill. FLIX utasítások használata az alkalmazói szinten lehetséges, bárki (akár a végfelhasználó is) írhat új alkalmazásokat
- Az egyedi HW jellemzőket egy közbenső firmware réteg fedi el, gondoskodik az alacsonyszintű adatkezelési problémák megoldásáról
- Az Xtensa processzorok sok alkalmazásban bizonyították jó paramétériket
- De csak megfelelő darabszám esetén gazdaságos!