Reconfigurable Architectures & Hardware Acceleration

Mark McDermott

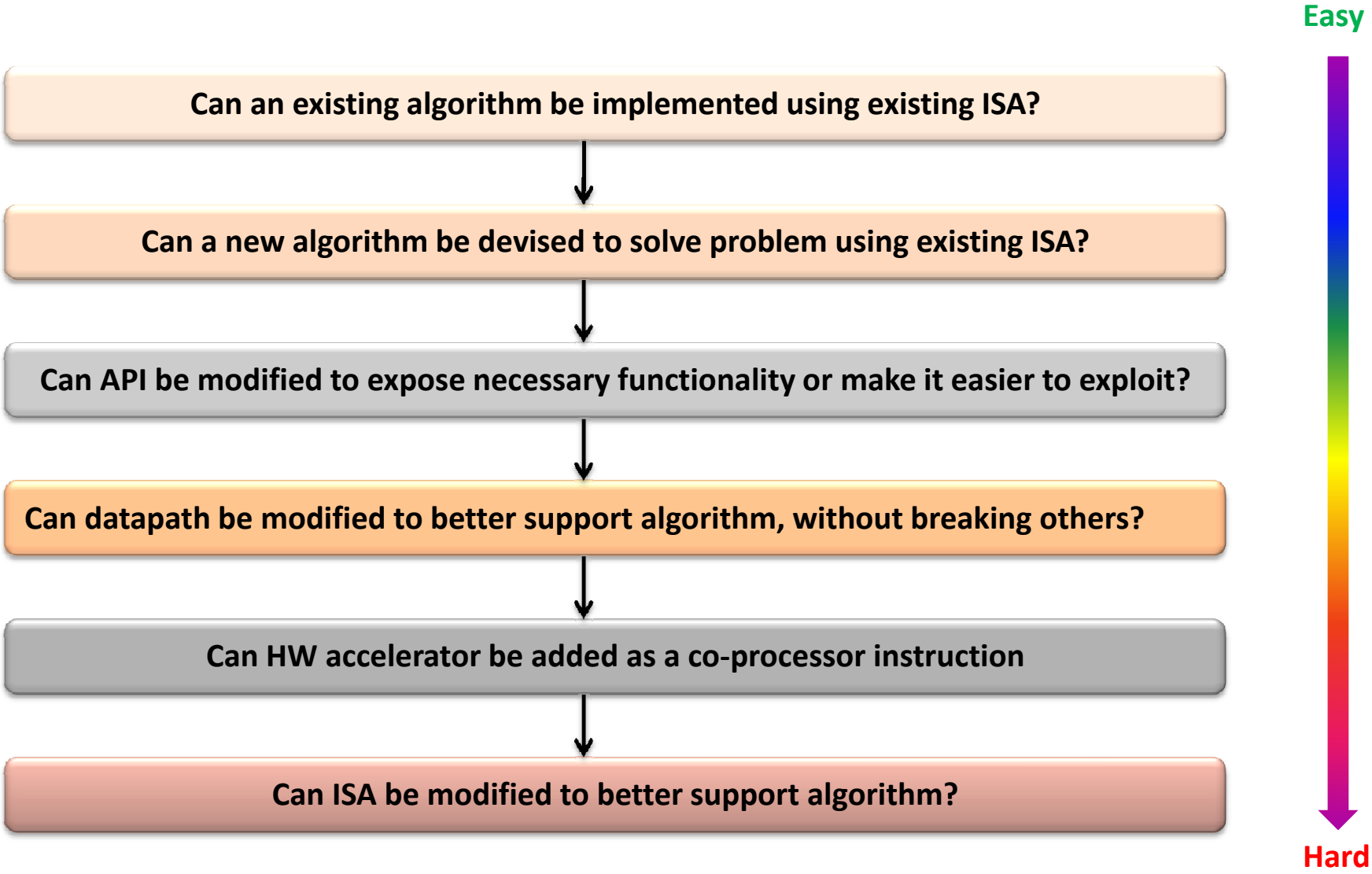
Spring 2009

Agenda

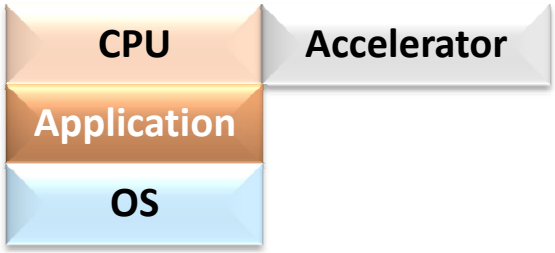
- Taxonomy of Hardware Acceleration
- ISA Acceleration:
  - HC12 Fuzzy Logic Acceleration
- Micro-coded Hardware acceleration
  - MC68332 Time Processing Unit
- Taxonomy of Reconfigurable Architectures
- Reconfigurable Instruction Set Architectures
  - Tensilica LX

Hardware Acceleration

Decision Tree: When do you use a hardware accelerator?

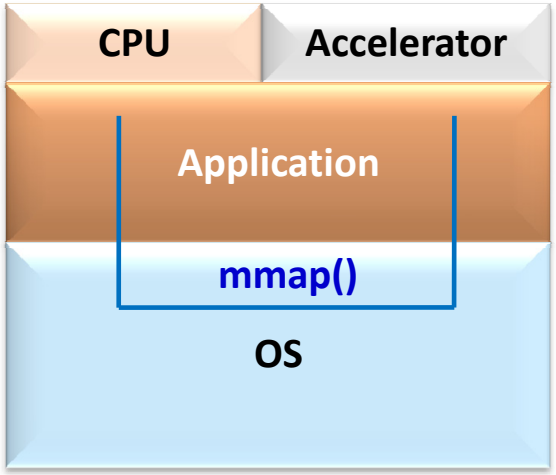


Four Fundamental Models of Accelerator Design

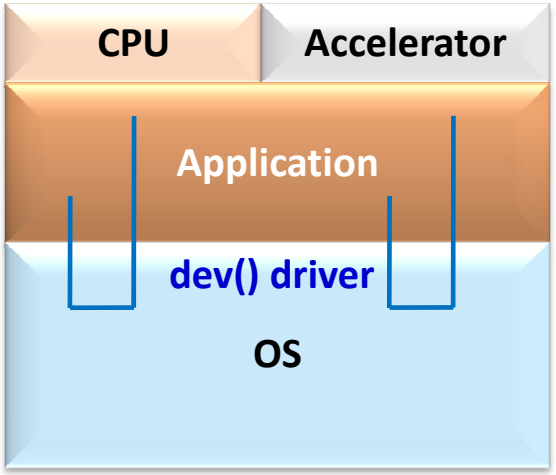


Base - HW I/F only

No OS Service (in simple embedded systems)



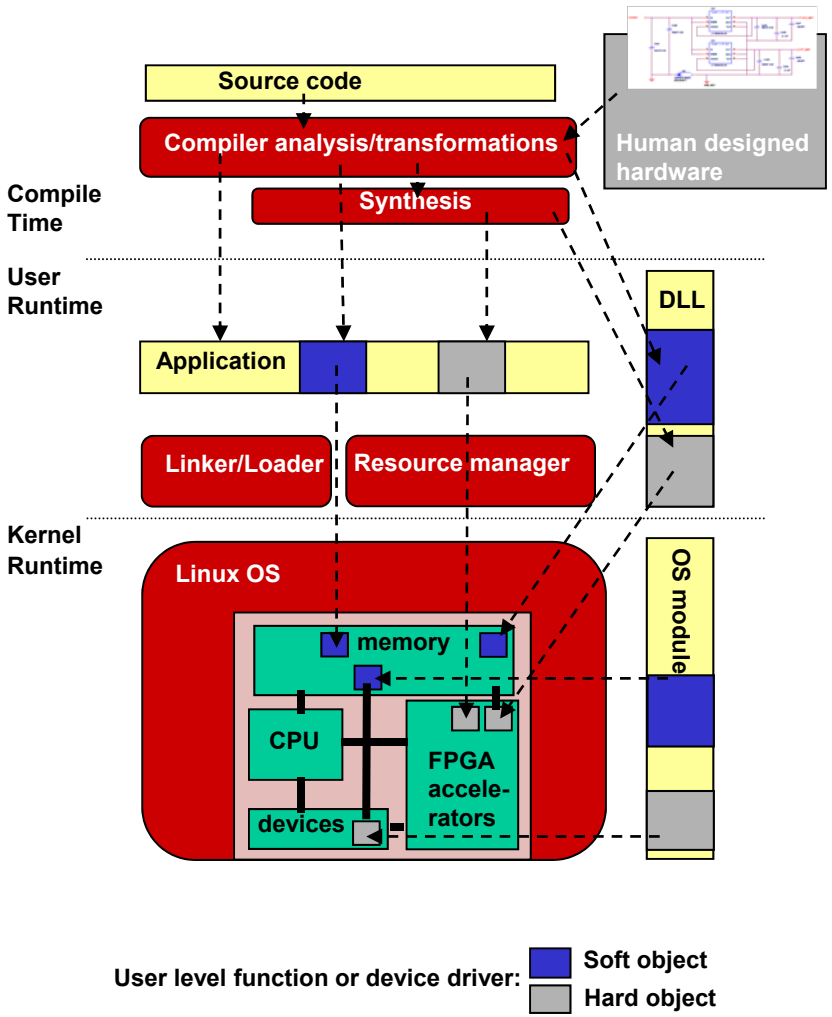
OS service – Accelerator accessed as a user space memory mapped I/O device



Virtualized Device with OS scheduling support

Perkowski, psu.edu

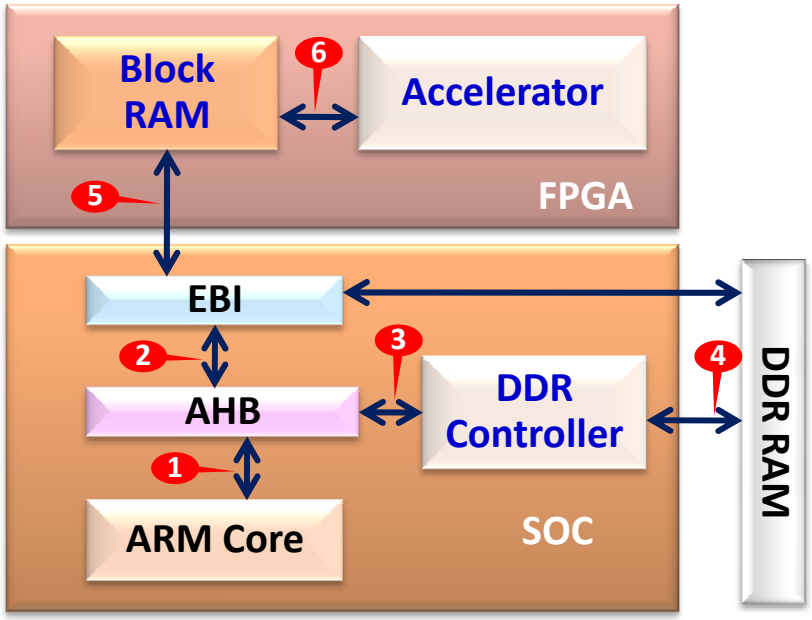
Hybrid Hardware/Software Execution Model



- Hardware Accelerator as a DLL
  - Seamless integration of hardware accelerators into the Linux software stack for use by mainstream applications
  - The DLL approach enables transparent interchange of software and hardware components
- Application level execution model
  - Compiler deep analysis and transformations generate CPU code, hardware library stubs and synthesized components
  - FPGA bitmaps as hardware counterpart to existing software modules.
  - Same dynamic linking library interfaces and stubs apply to both software and hardware implementation
- OS resource management
  - Services (API) for allocation, partial reconfiguration, saving and restoring the status, and monitoring
  - Multiprogramming scheduler can pre-fetch hardware accelerators in time for next use
  - Control the access to the new hardware to ensure trust under private or shared use

Perkowski, psu.edu

CPU-Accelerator Interconnect Options

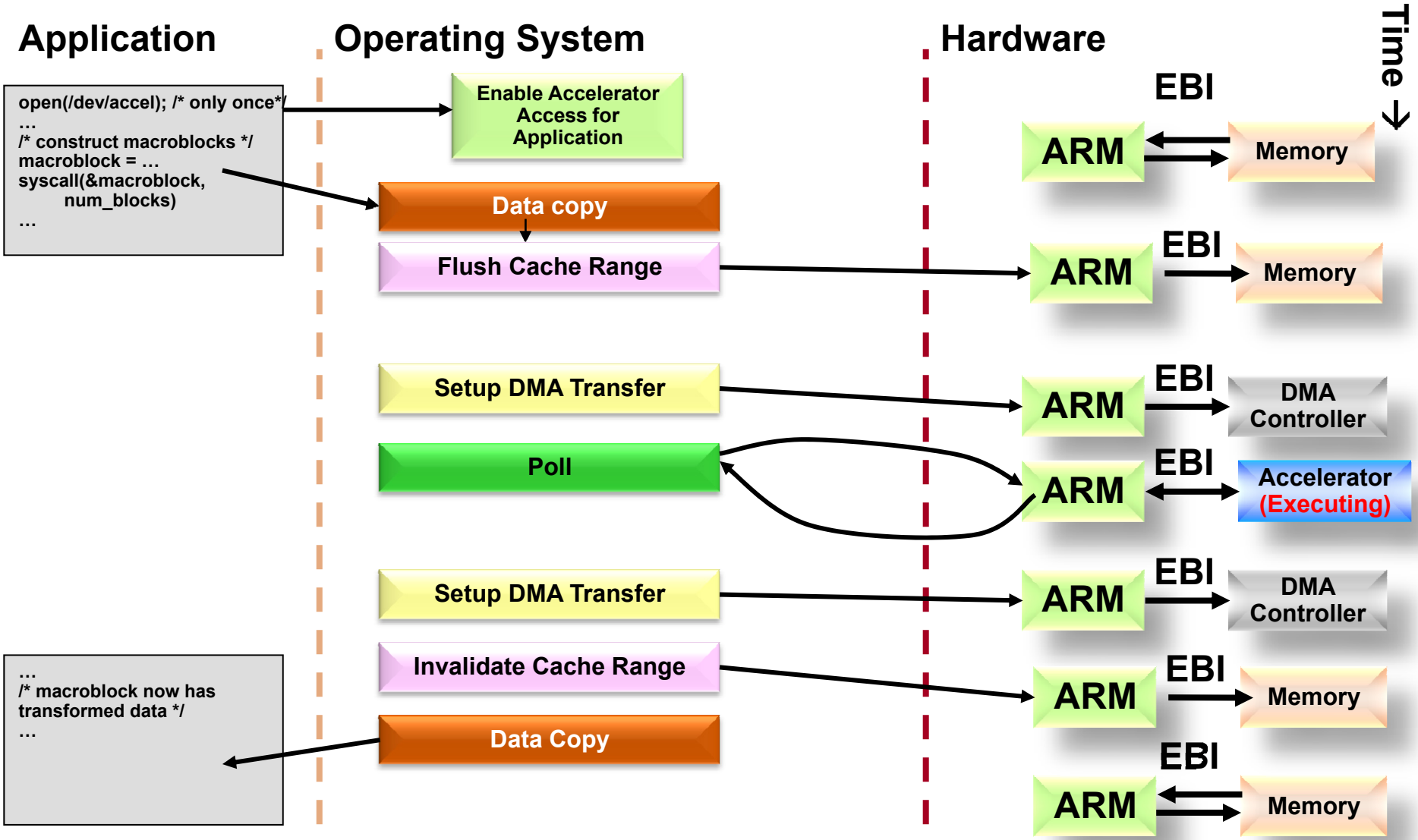


Bus	First Access		Pipelined Access		Arbitration
	Read	Write	Read	Write	
1 ARM → AHB	2	2	2	2	
2 AHB → EBI	8	8	3	3	5
3 AHB → DDRC	4	4	4	4	
4 DDRC → DRAM	8	9	3	3	5
5 EBI → BRAM	20	20	8	8	12
6 BRAM → ACC	2	2	2	2	

- **EBI (External Bus Interface)**
  - 32 bit Bus
  - Access to DRAM data & FPGA data
  - 1/4 CPU frequency
  - Big penalty if bus is busy during first attempt to access bus
- **AHB (AMBA High Speed Bus)**
  - 32 bit bus
  - Runs at CPU clock frequency
  - Access to DDR Controller to provide addresses to SDRAM

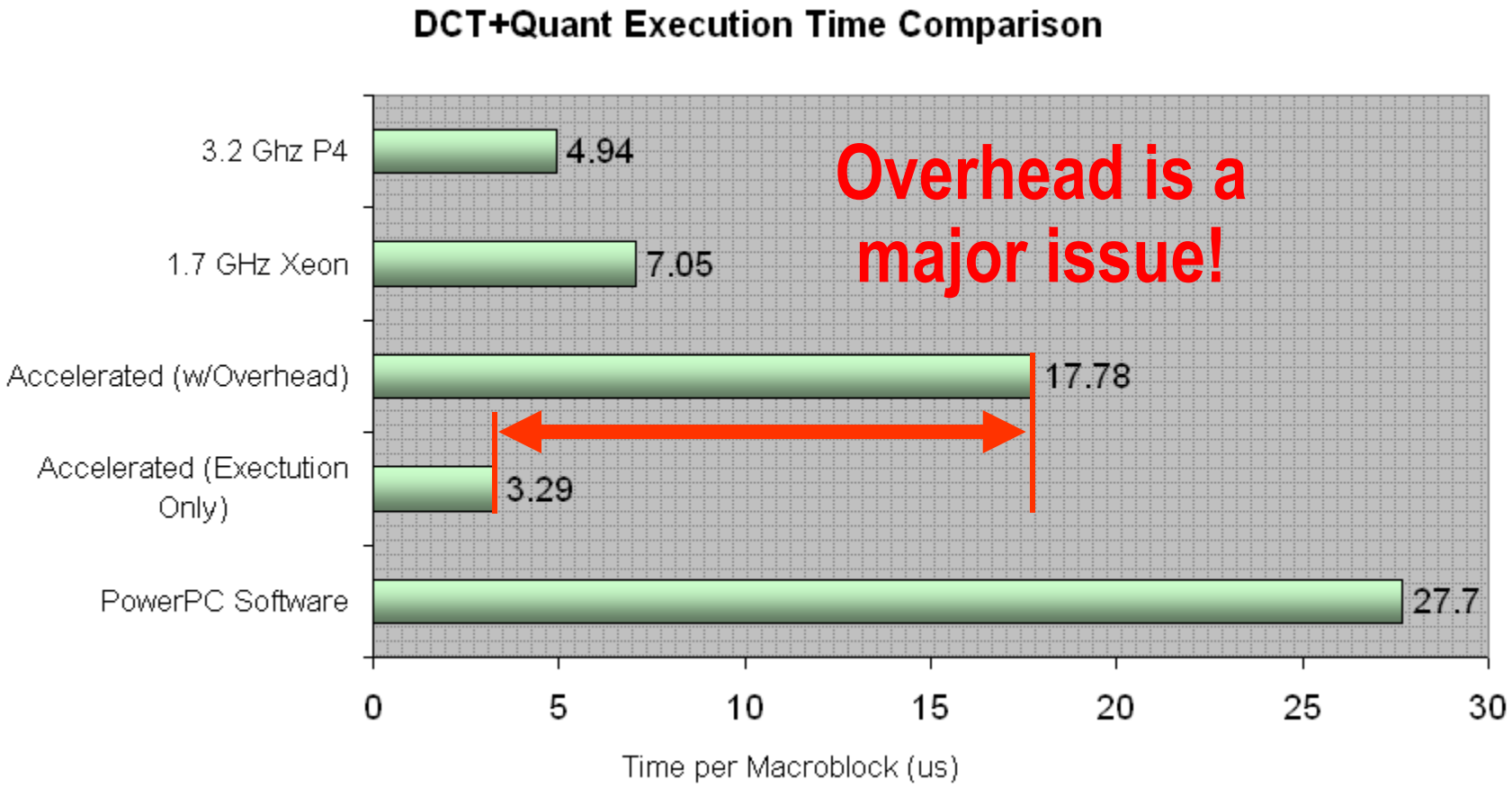
Perkowski, psu.edu

Typical CPU → Accelerator Transaction

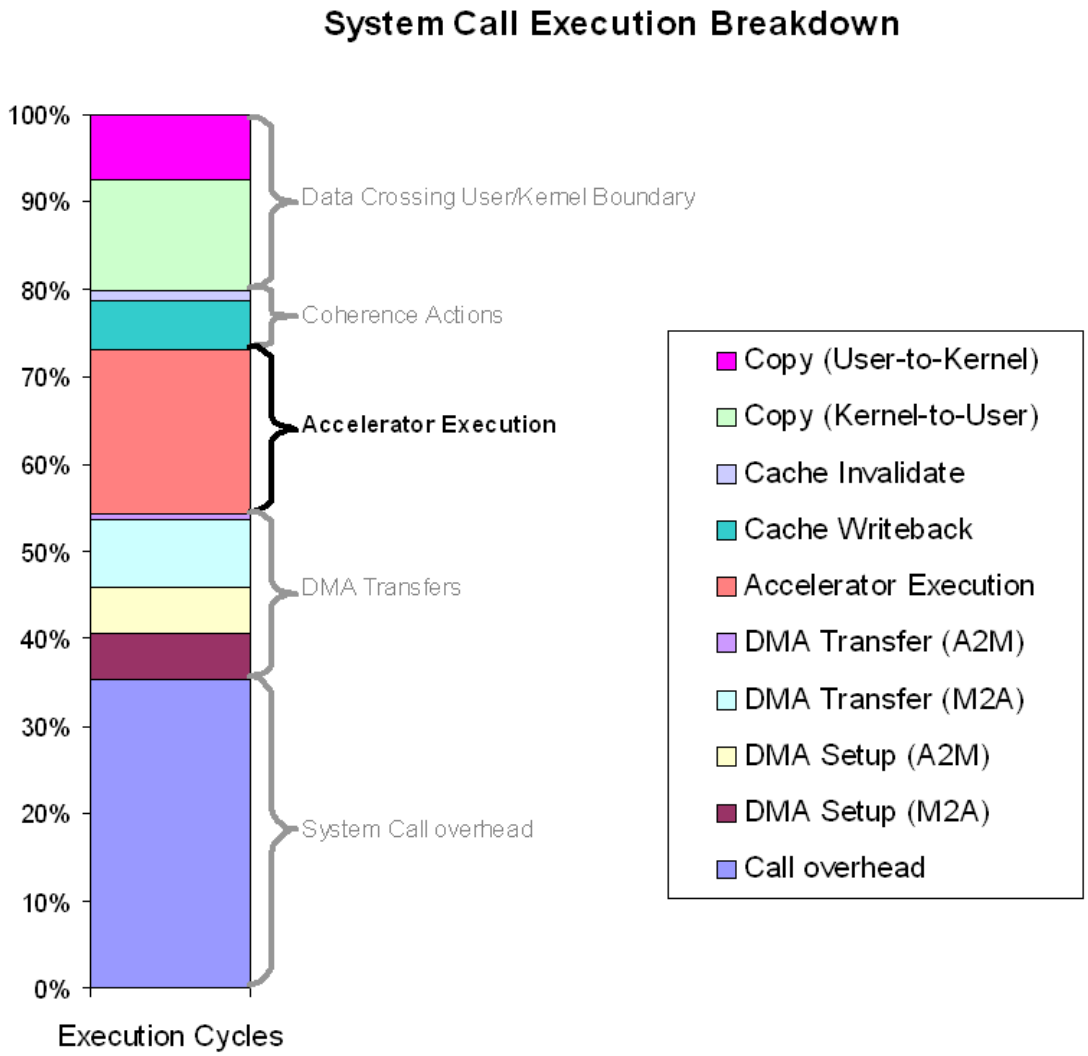


Perkowski, psu.edu

Software Versus Hardware Acceleration



Device Driver Access Cost



Perkowski, psu.edu

Fuzzy Logic Acceleration

Overview

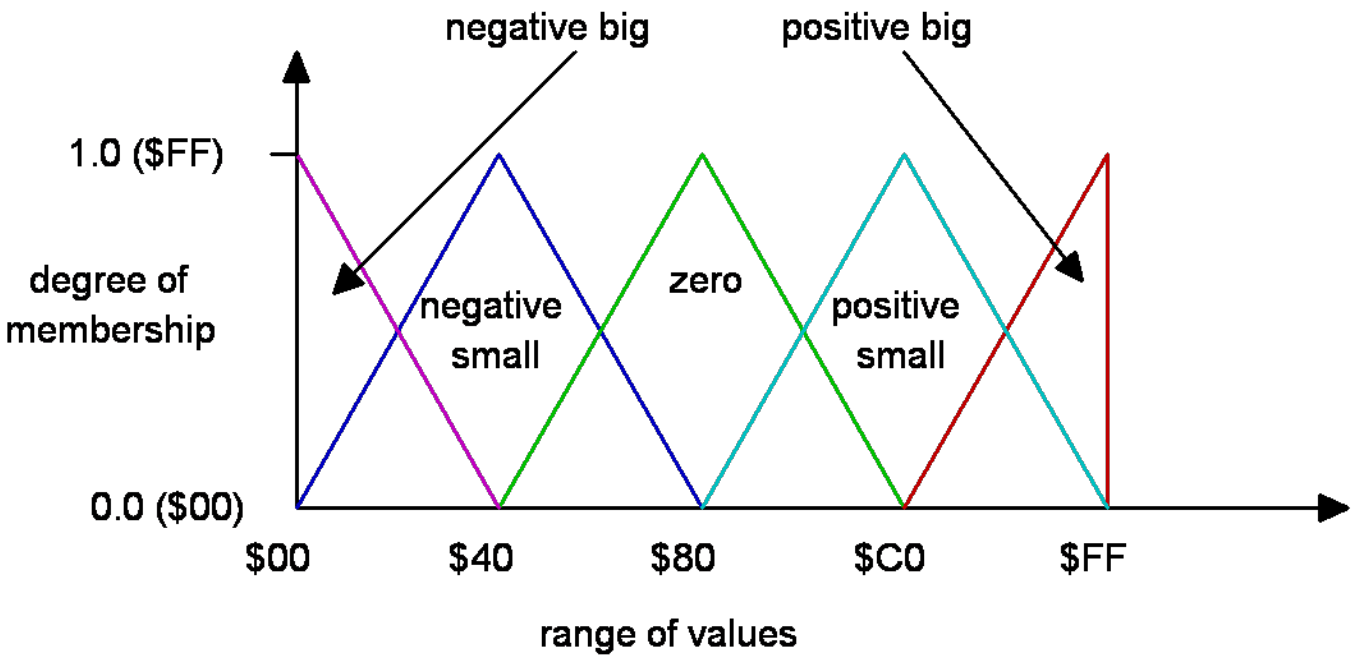
- **Fuzzy controllers provide a unique way of controlling complex systems.**
  - If written in software they can take a long time to execute, limiting their application in real-time systems.
  - Using dedicated logic and a few new assembler instructions, a microcontroller can be enhanced to execute a fuzzy controller quite efficiently.

Fuzzy Sets

- In the real world, very few things belong to a single classification and often the boundaries are not clear
- Fuzzy sets, then, is the extension of regular (Aristotelian) sets:
  - sets can overlap
  - members of a set have a degree of membership instead of just belonging to or not
- Fuzzy sets can be given linguistic labels such as warm, positive big, very cold, etc
- Fuzzy logic allows the sets to be computed
  - boundary conditions same as two-level logic

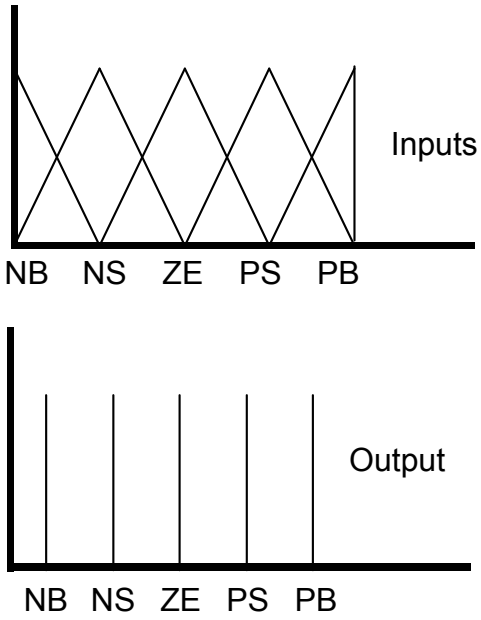
Input Memberships

- The degree to which an input belongs to a classification, is called its membership value
- The classifications can overlap so that an input value can belong to more than one, hence the ability to fuzzify the input



Motorola, Inc

Fuzzy Controller



Error

Change of Error	NB	NS	ZE	PS	PB
NB	ZE	PS	PB	PB	PB
NS	NS	ZE	PS	PS	PB
ZE	NB	NS	ZE	PS	PB
PS	NB	NS	NS	ZE	PS
PB	NB	NB	NB	NS	ZE

Output Adjustment

- error = setting - position
- change of error = error - previous error
- output is an adjustment to current output

Motorola, Inc

Fuzzy Rules

- IF error is positive big THEN output is much smaller
- IF error is small and change of error is positive big THEN output is smaller
- IF error is small and change of error is positive small THEN output is a little smaller
- IF error and change of error are zero THEN output is zero

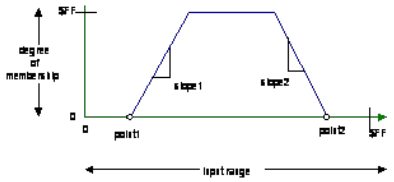
Motorola, Inc

Fuzzy Controller: Implementation

- 68HC12 microcontroller (Motorola)
  - Fuzzy logic instructions
- Simple fuzzy system response on 68HC11:
  - 750 ms
- Simple fuzzy system response on 68HC12:
  - 50 us (15,000 times faster)
    - An enabling technology

Motorola, Inc

Embedded Fuzzy

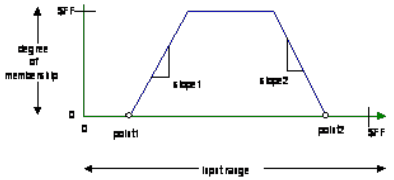


$$system\_output = \frac{\sum_{i=1}^{n} S_i F_i}{\sum_{i=1}^{n} F_i}$$

- Single chip 68HC12 microcontroller
- Native fuzzy instructions:
  - MEM; evaluate membership functions
  - REV; rule evaluation: IF a is x THEN b is y
  - WAV; weighted averaging
- Additional related instructions
  - MINA (place smaller of two unsigned 8-bit values in accumulator A)
  - EMIND (place smaller of two unsigned 16-bit values in accumulator D)
  - MAXM (place larger of two unsigned 8-bit values in memory)
  - EMAXM (place larger of two unsigned 16-bit values in memory)
  - TBL (table lookup and interpolate)
  - ETBL (extended table lookup and interpolate)
  - EMACS (extended multiply and accumulate signed 16-bit by16-bit to 32-bit)
- Orders of magnitude faster than fuzzy routines
- Includes: timers, PWM, A/D, Flash, RAM
- Small and low power

Motorola, Inc

68HC12: Fuzzy Innards



IF a is x
THEN b is y

$$system\_output = \frac{\sum_{i=1}^{n} S_i F_i}{\sum_{i=1}^{n} F_i}$$

program data

knowledge base

input membership functions

rule list

output membership functions

system inputs

assembler instructions

fuzzy inference kernel

fuzzification

mem

fuzzy inputs (in RAM)

rule evaluation

rev

fuzzy outputs (in RAM)

defuzzification

wav,ediv

system outputs

Motorola, Inc

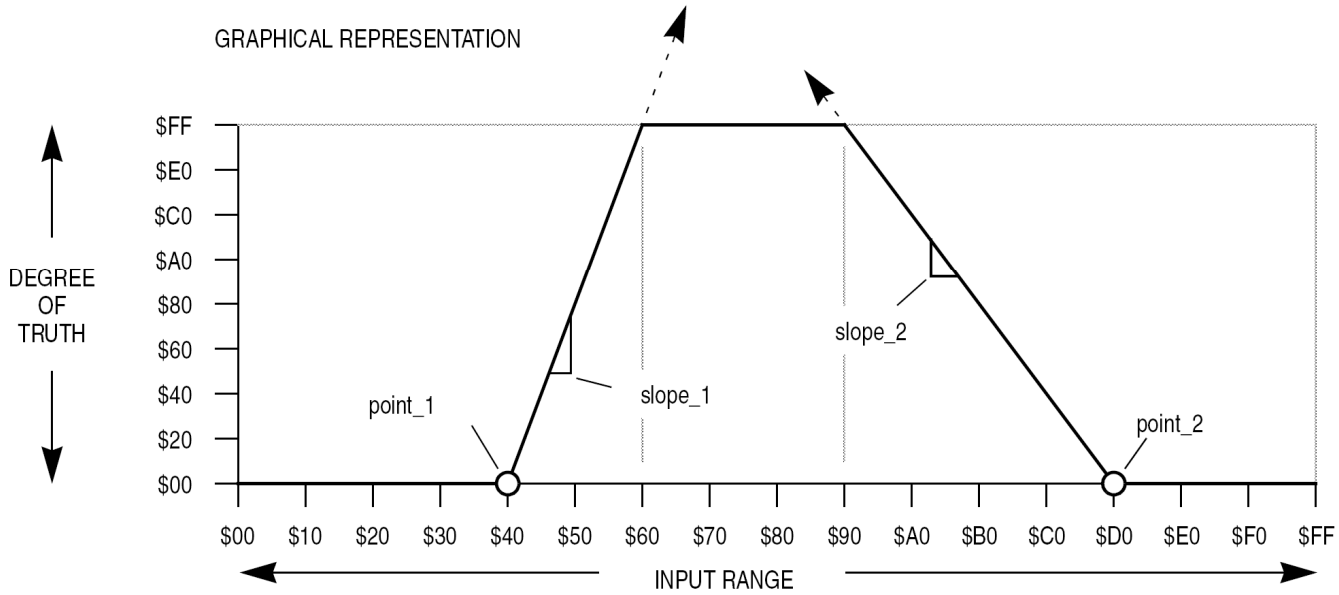
Chapman - ualberta.ca

A Closer Look

- **Each fuzzy instruction uses an efficient memory structure to maintain information.**
  - The MEM instruction uses an array of trapezoids for membership functions and writes to an array of bytes with the calculated membership values for each input.
  - The REV instruction use a byte array of offsets and flags for the rules antecedents and consequences and byte arrays for the outputs.
  - The REVW instruction uses an array of16 bit pointers and flags for antecedent and consequence values and a byte array for weights and outputs
  - The WAV instruction uses an byte array for outputs and singletons.

Motorola, Inc

Trapezoidal Parameters



MEMORY REPRESENTATION

ADDR	$40	X-POSITION OF point_1
ADDR+1	$D0	X-POSITION OF point_2
ADDR+2	$08	slope_1 ($FF/(X-POS OF SATURATION – point_1))
ADDR+3	$04	slope_2 ($FF/(point_2 – X-POS OF SATURATION))

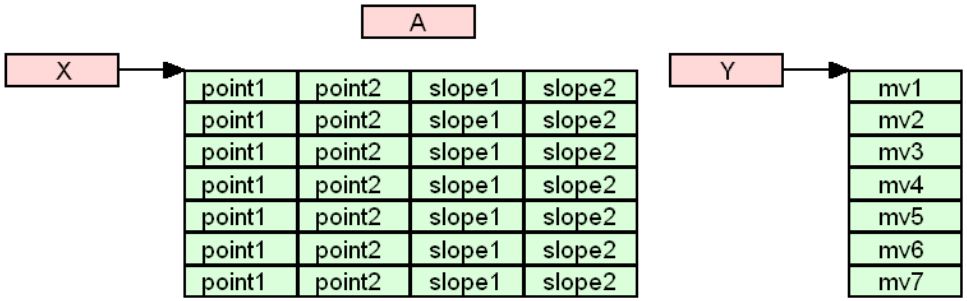
point1	point2	slope1	slope2
point1	point2	slope1	slope2
point1	point2	slope1	slope2
point1	point2	slope1	slope2
point1	point2	slope1	slope2
point1	point2	slope1	slope2
point1	point2	slope1	slope2

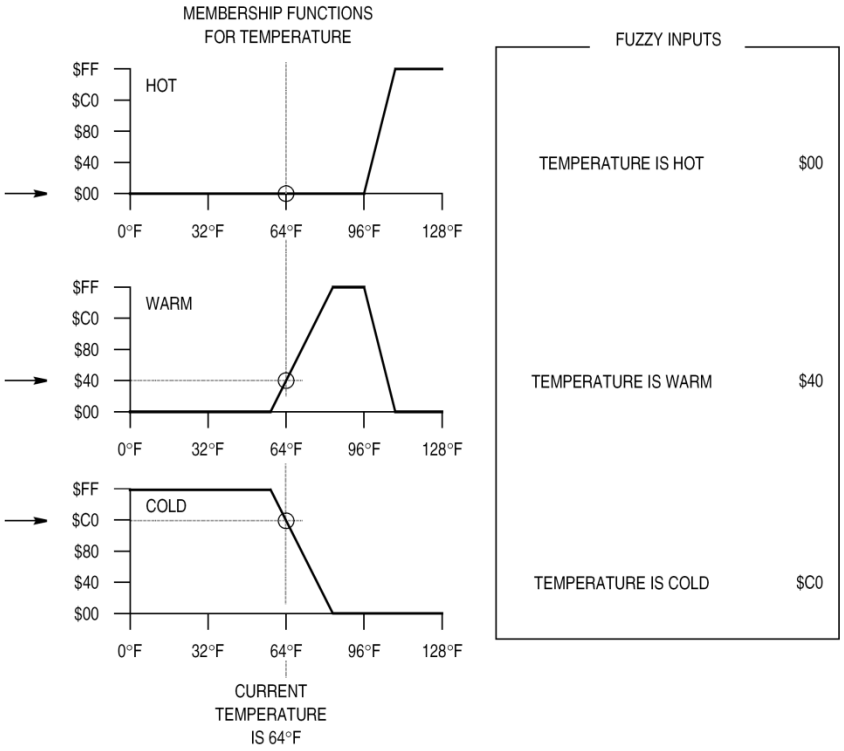
Motorola, Inc

Chapman - ualberta.ca

Fuzzify

```
fuzzify: ldx #input_mfs   ; point at membership functions
         ldy  #fuz_ins    ; point at fuzzy input table
         ldaa current_ins ; get first input values
         ldab #7          ; 7 labels per input
loop:    mem              ; evaluate one membership func.
         dbne b, loop     ; for 7 labels of one input
```





Motorola, Inc

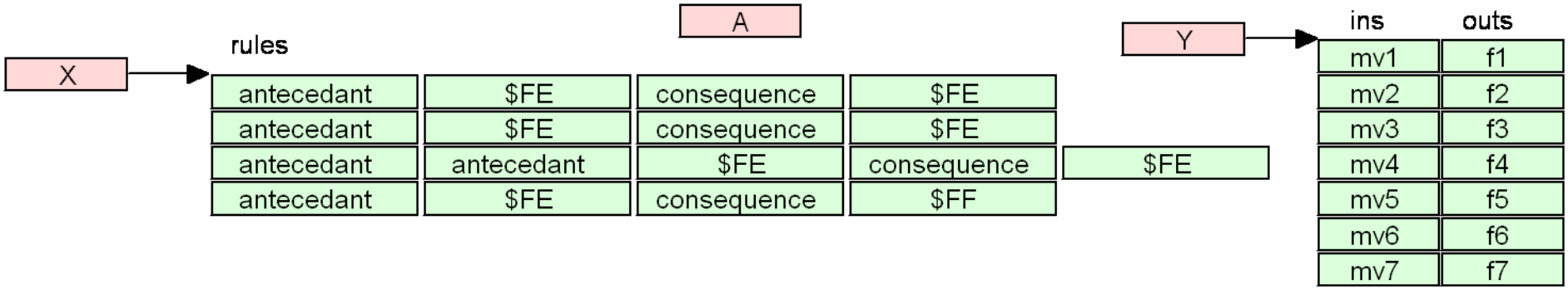
Chapman - ualberta.ca

Rule Evaluation

- **Rule evaluation is the central element of a fuzzy logic inference program.**
  - Processes a list of rules from the knowledge base using current fuzzy input values from RAM to produce a list of fuzzy outputs in RAM.
- **The CPU12 offers two variations of rule evaluation instructions:**
  - The REV instruction provides for unweighted rules (all rules are considered to be equally important).
  - The REVW instruction is similar but allows each rule to have a separate weighting factor which is stored in a separate parallel data structure in the knowledge base.
- **The fuzzy *and operator* corresponds to the mathematical minimum operation and the fuzzy *or* operation corresponds to the mathematical maximum operation.**

Evaluate Rules

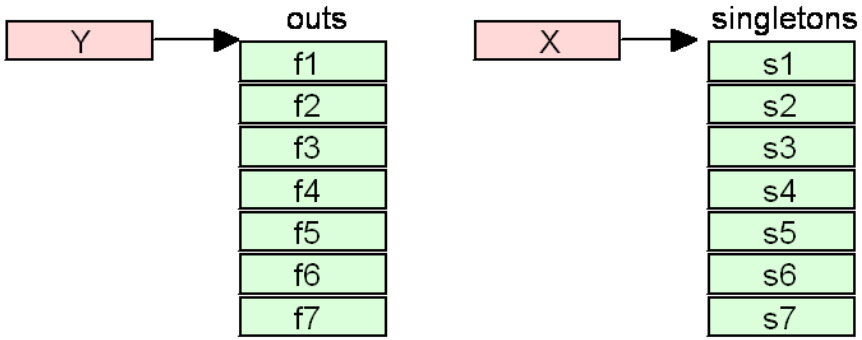
```
        ldab #7             ; loop count
eval:   clr  1,y+           ; clear a fuzzy out and inc pointer
        dbne b, eval        ; loop to clr all fuzzy outs
        ldx  #rule_start    ; point at first rule element
        ldy  #fuz_ins       ; point at fuzzy ins and outs
        ldaa #$ff           ; init A (and clears V-bit)
        rev                 ; process rule list
```



Motorola, Inc

De-Fuzzify

```
defuz:  ldy   #fuz_out      ; point at fuzzy outputs
        ldx   #sgltn_pos    ; point at singleton positions
        ldab  #7            ; 7 fuzzy outs per COG output
        wav                 ; calculate sums for wtd av
        ediv                ; final divide for wtd av
        tfr   y,d           ; move result to A:B
        stab  cog_out       ; store system output
```



$$system\_output = \frac{\sum_{i=1}^{n} S_i F_i}{\sum_{i=1}^{n} F_i}$$

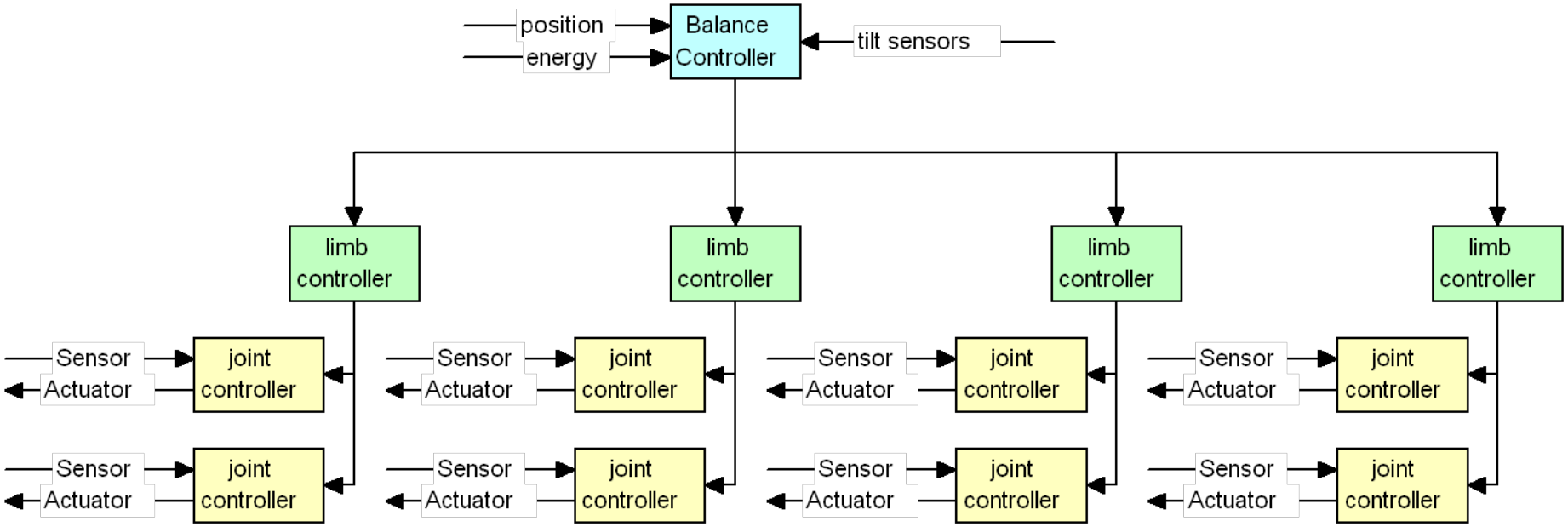
Motorola, Inc

Program Measurements

- **Assume: 2 inputs, 1 output, 25 non-conjunctive rules and 7 membership functions on ins and out.**
  - **Data structure costs: 160 bytes**
  - **Program structure costs: <100 bytes**
  - **Execution time: 75uS (response time) or about 13Khz maximum cycle time**

Motorola, Inc

Example: Multiple Fuzzy Controllers for a Quadruped Robot



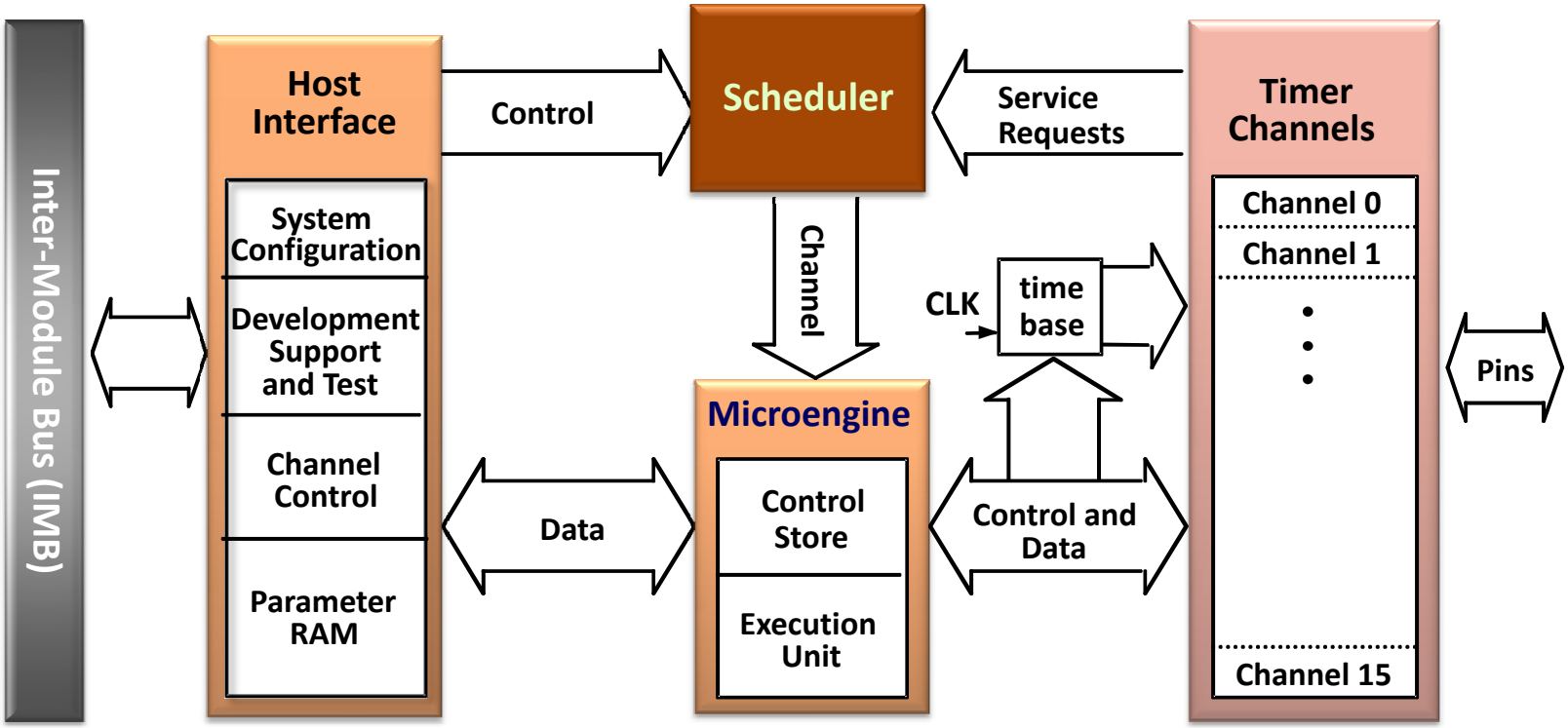
- hierarchy of 13 fuzzy controllers
- standing level on uneven surfaces
- maintaining balance dynamically

Motorola, Inc

MC 68332 Time Processing Unit

MC68332 Time Processing Unit

- The TPU3 can be viewed as a *special-purpose microcomputer* that performs a programmable series of two operations, match and capture.
- The microengine uses microcode to perform functions.



Time Processing Unit

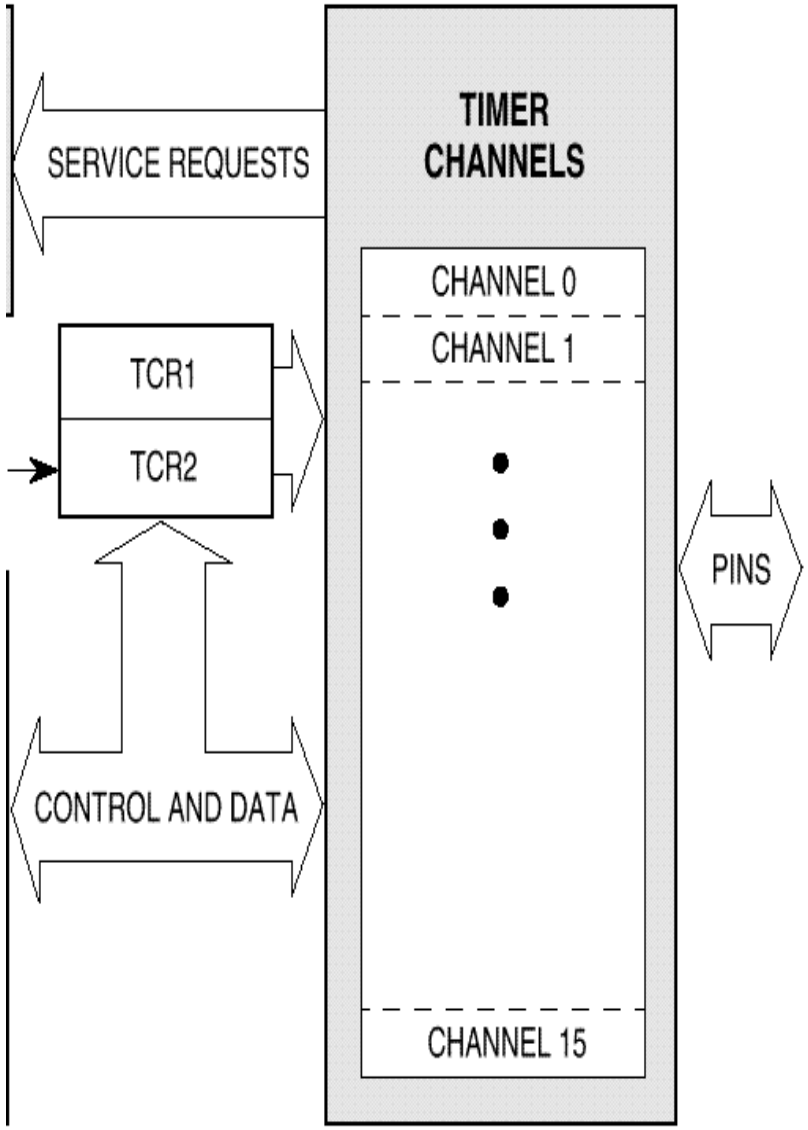
- Semi-autonomous microcontroller
- Operates concurrently with CPU
- Schedules tasks
- Processes ROM instructions
- Accesses shared data with CPU
- Performs Input/Output operations
- Programmable series of 2 operations
  - Match
  - Capture
- Each operation is called an ``event''
- A pre-programmed series of event is called a ``function''

TPU Preprogrammed Functions:

Function Number	Function Nickname	Function Name
0xF	PTA	Programmable Time Accumulator
0xE	QOM	Queued Output Match
0xD	TSM	Table Stepper Motor
0xC	FQM	Frequency Measurement
0xB	UART	Universal Asynchronous Receiver/Transmitter
0xA	NITC	New Input Capture/Input Transition Counter
9	COMM	Multiphase Motor Commutation
8	HALLD	Hall Effect Decode
7	MCPWM	Multi-Channel Pulse Width Modulation
6	FQD	Fast Quadrature Decode
5	PPWA	Period/Pulse Width Accumulator
4	OC	Output Compare
3	PWM	Pulse Width Modulation
2	DIO	Discrete Input/Output
1	SPWM	Synchronized Pulse Width Modulation
0	SIOP	Serial Input/output Port

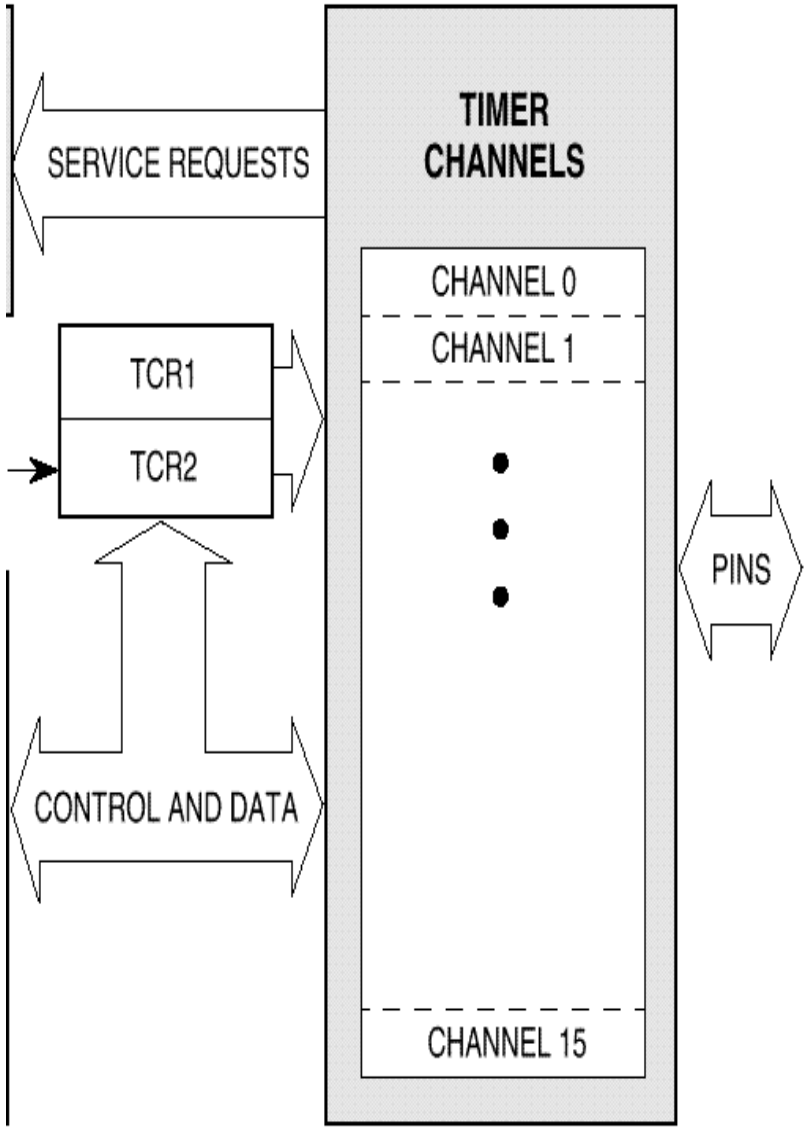
Time Bases

- Two sixteen-bit counters provide time bases for all
- Pre-scalers controlled by CPU via bit-fields in TPU module configuration register TPUCMR
- Current values accessible via TCR1 and TCR2 registers
- TCR1, TCR2 can be read/written by TPU microcode- not available to CPU
- TC1 qualified by system clock
- TC2 qualified by system clock or external clock



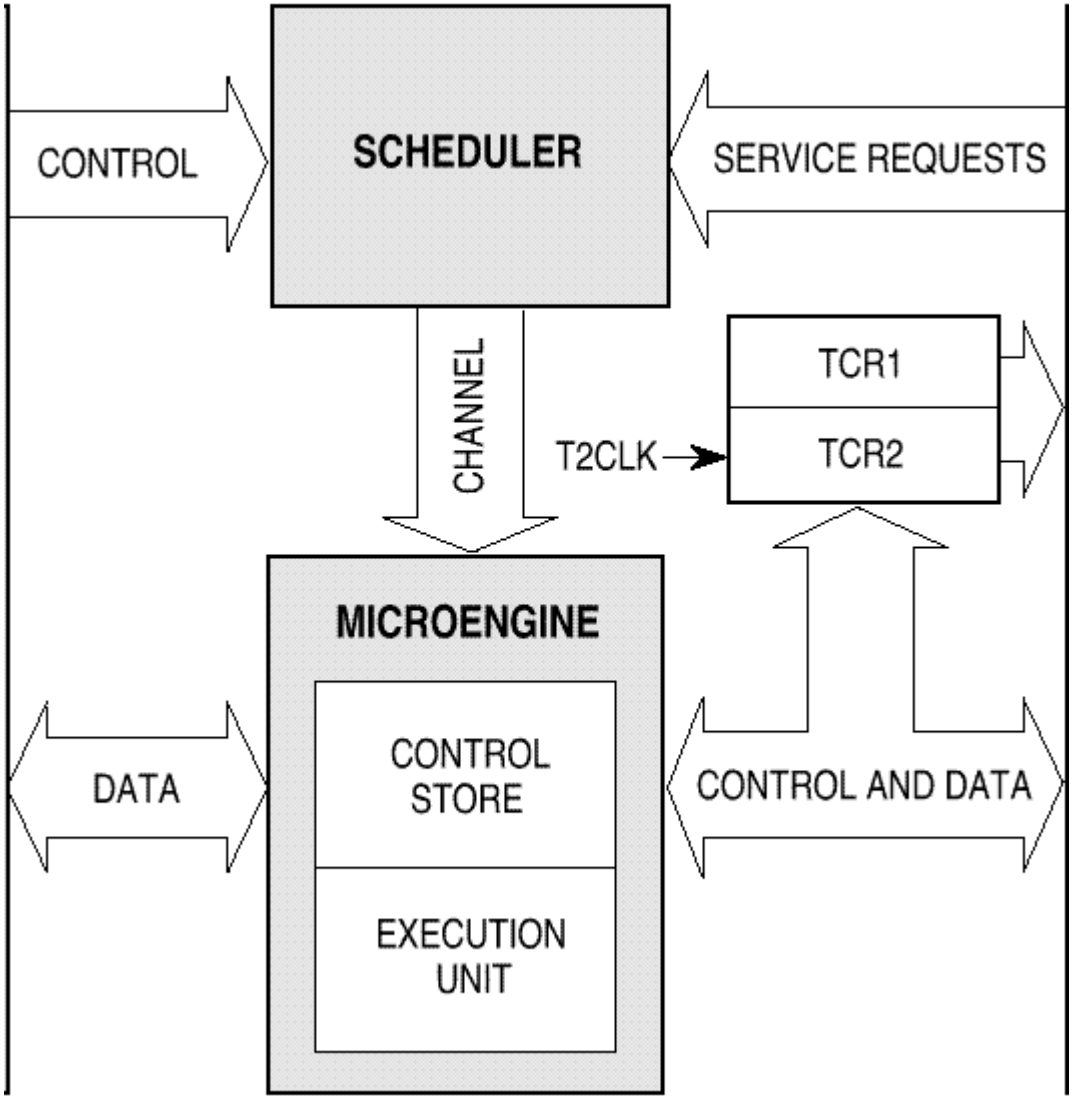
Timer Channels

- Sixteen channels
  - each one connect to a MCU pin
- Each channel has symmetric hardware:
- Event register
  - 16-bit capture register
  - 16-bit compare/match register
  - 16-bit comparator
- Pin control logic - pin direction determined by TPU microengine



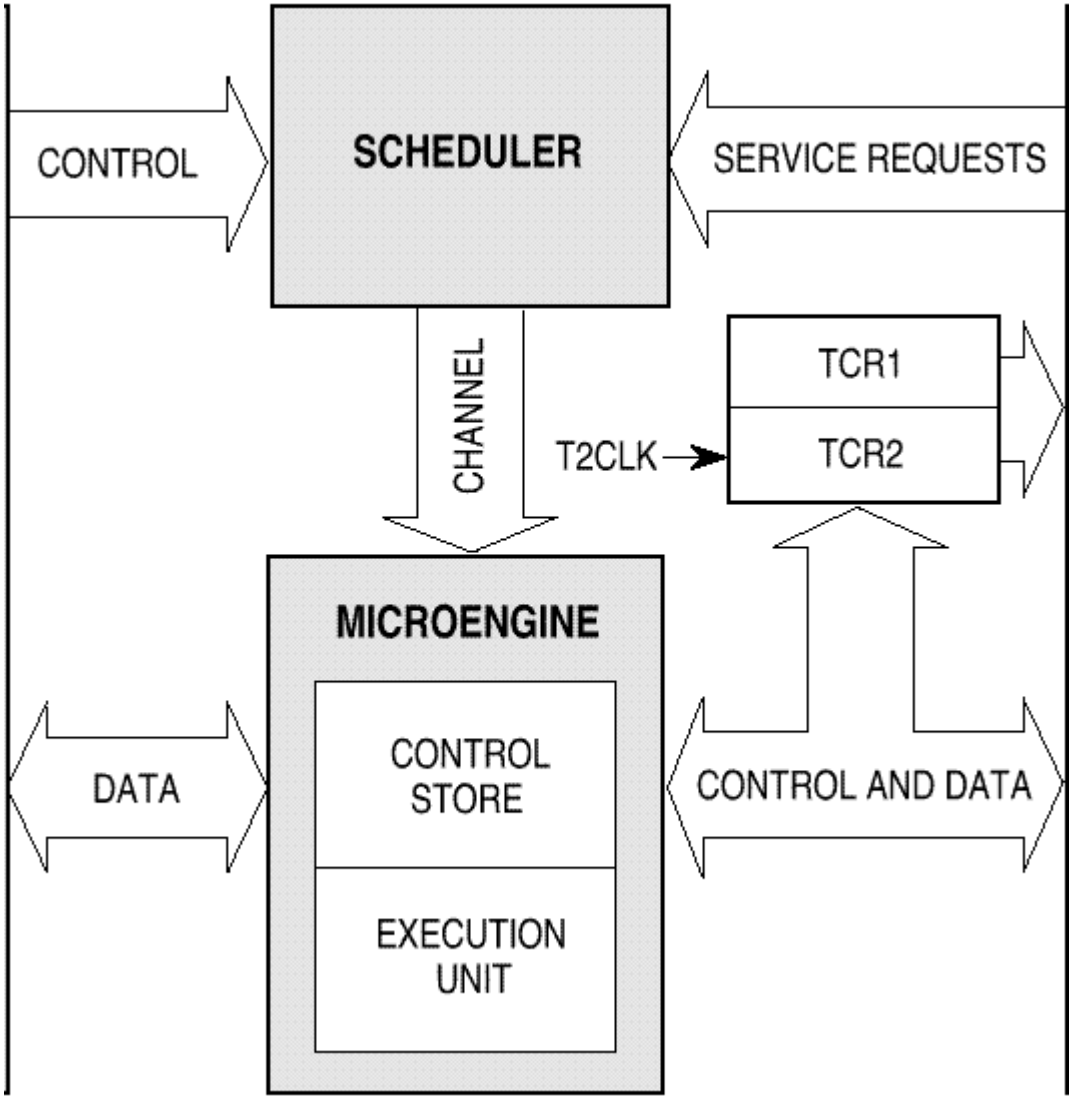
Scheduler

- Determines which of sixteen channels is serviced by the microengine
- Channel can request service for one of four reasons
  - host service
  - link to another channel
  - match event
  - capture event
- Host system assigns to each channel a priority
  - high
  - middle
  - low



Microengine

- Executes microcoded functions for selected channel.
- Returns control to scheduler when completed.

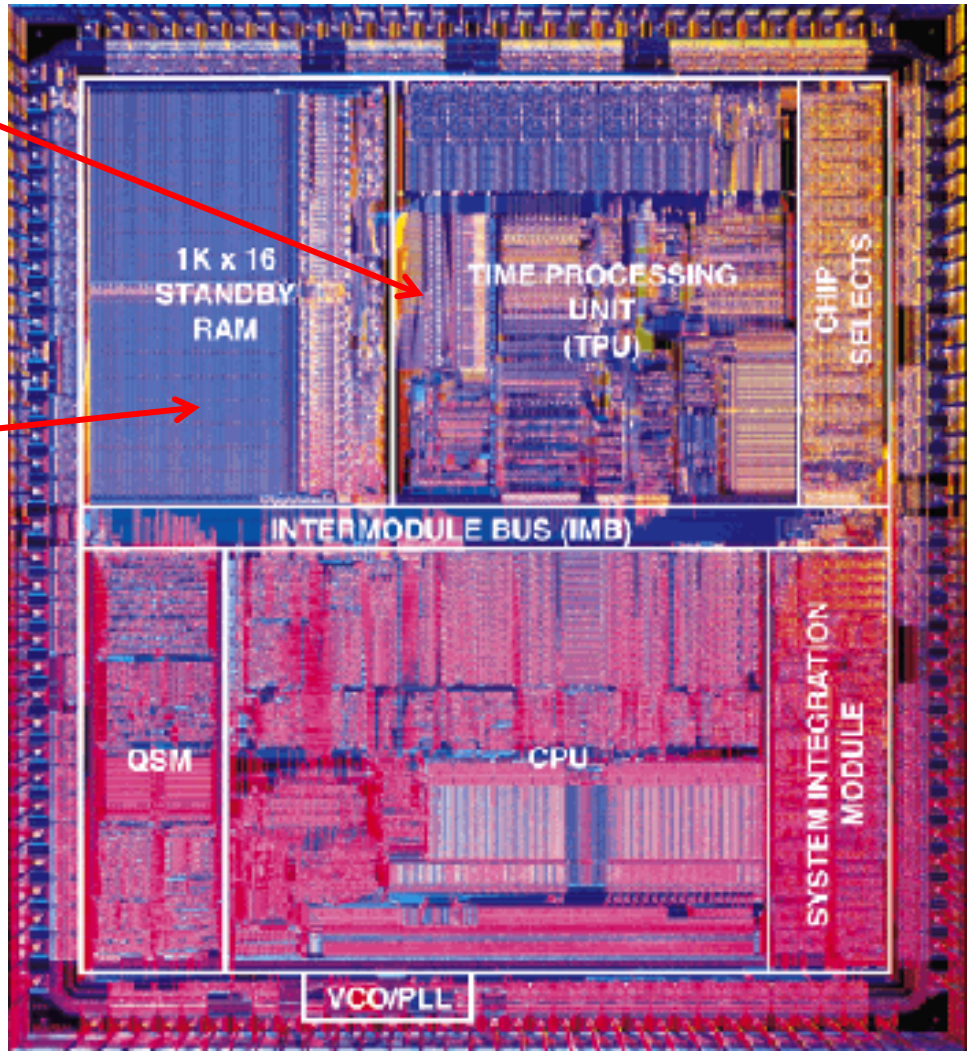


Motorola, Inc.

Kurt Keutzer UCB

WCS – Writeable Control Store

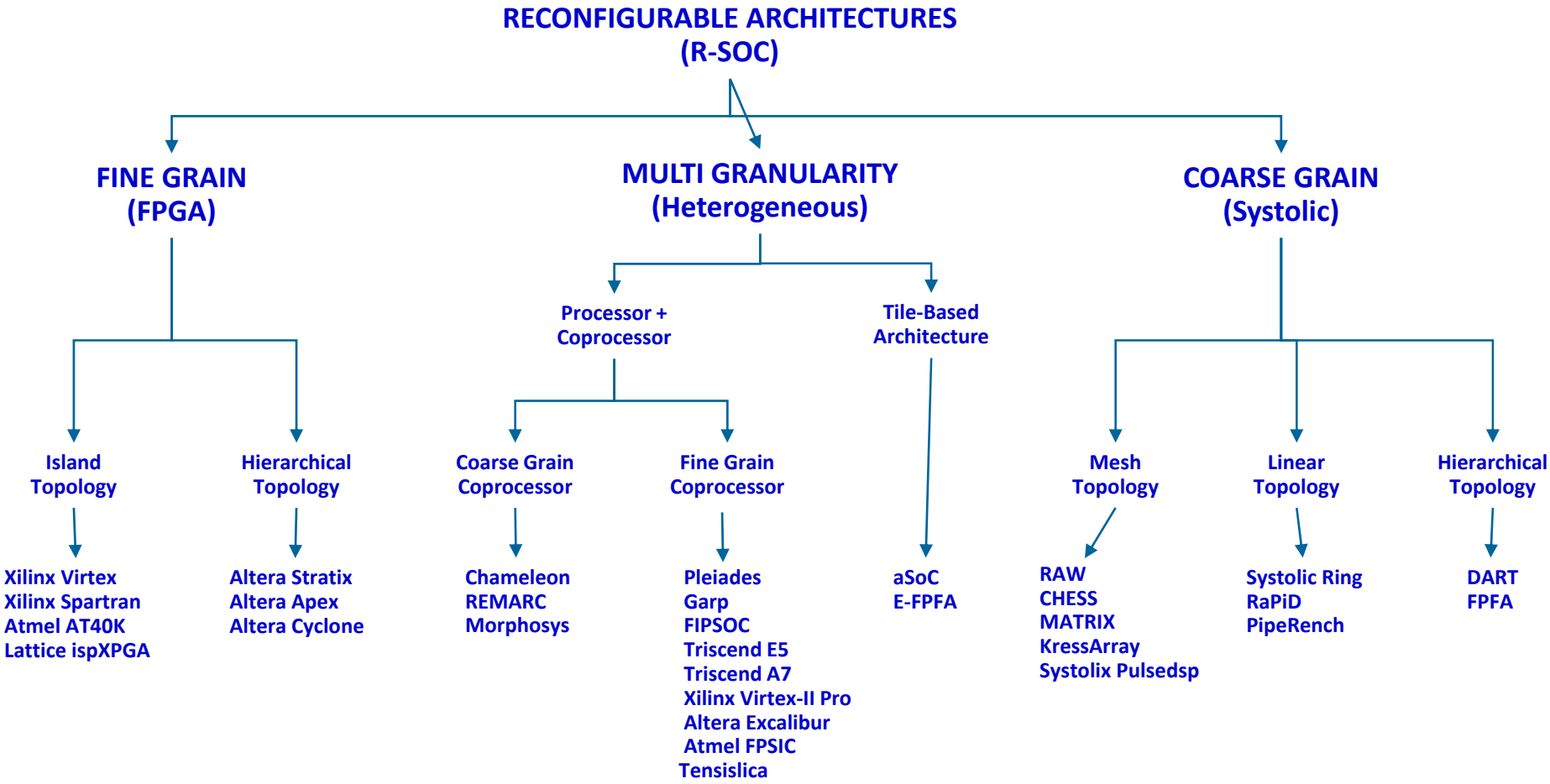
- The microcode in the TPU is hard coded into a mask programmable ROM.
- To facilitate microcode development and debug, a block RAM can be used to replace the ROM providing a "Writeable Control Store" capability.



68332 Die Photo

Reconfigurable Architectures: Tensilica LX

Taxonomy of Reconfigurable Architectures



Lilian Bossuet
LESTER Lab
Université de Bretagne Sud
Lorient, France

Xtensa LX – Basic Architecture

- **Processor Configuration**
  - Energy Usage: 76 μW/MHz , 47 μW/MHz (5 and 7 stage pipeline)
  - Clock Speed: 350 MHz, 400 MHz (5 and 7 stage pipeline)
  - Cache:
    - up to 32 KB and 1,2,3,4 way set associative cache
  - 64 general purpose physical registers (32-bits)
  - 6 special purpose registers
  - Extensible via use of TIE and FLIX instructions
  - Zero over head loops

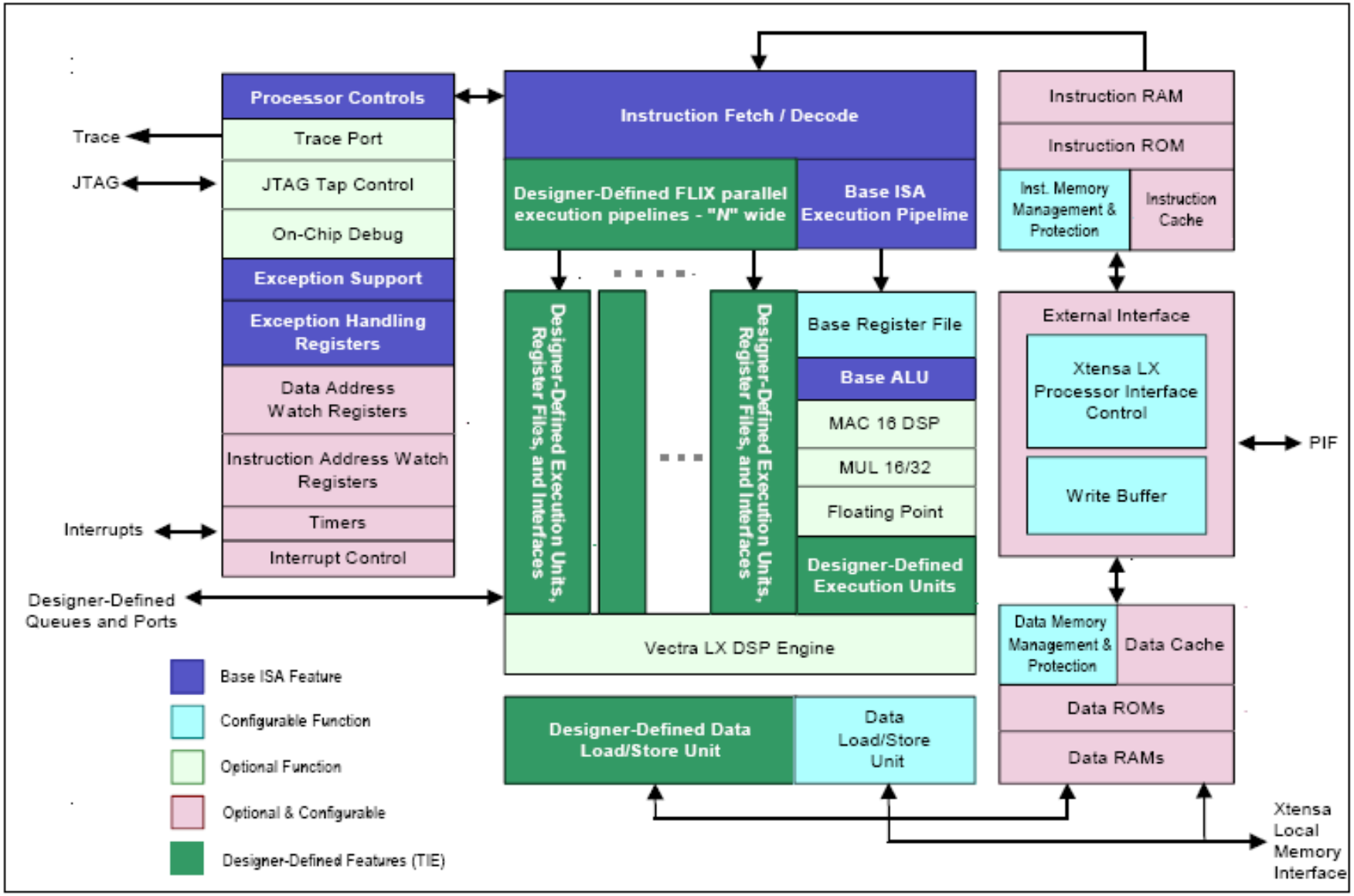
Courtesy Tensilica

Xtensa LX Architecture

- 32-bit ALU
- 1 or 2 Load/Store Model
- Registers
  - 32-bit general purpose register file
  - 32-bit program counter
  - 16 optional 1-bit Boolean registers
  - 16 optional 32-bit floating point registers
  - 4 optional 32-bit MAC16 data registers
  - Optional Vectra LX DSP registers
- General Purpose AR Register File
  - 32 or 64 registers
  - Instructions have access through “sliding window” of 16 registers. Window can rotate by 4, 8, or 12 registers
  - Register window reduces code size by limiting number of bits for the address and eliminated the need to save and restore register files

Courtesy Tensilica

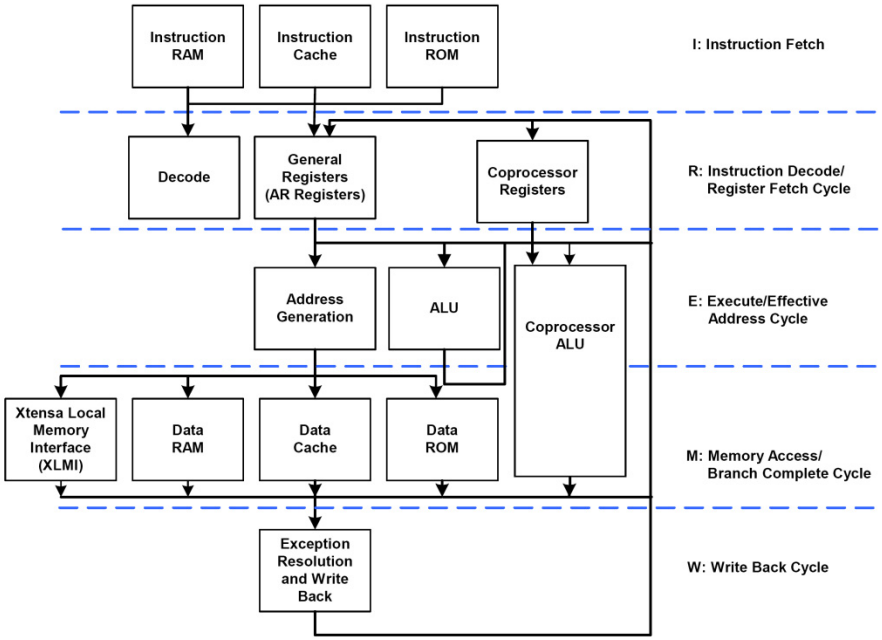
Xtensa LX Architecture



Courtesy Tensilica

Xtensa LX Pipelining

- **5 or 7 Stage Pipeline Design:**
  - 5 stage pipeline has stages: IF, Register Access, Execute, Data-Memory Access, and register writeback
  - 5 stage pipeline accesses memory in two stages. 7 stage pipeline is extended version of the 5 stage pipeline with extra IF and Memory Access stage. Extra stages provide more time for memory access. Designer can run at a higher clock speed while using slower memory to improve performance

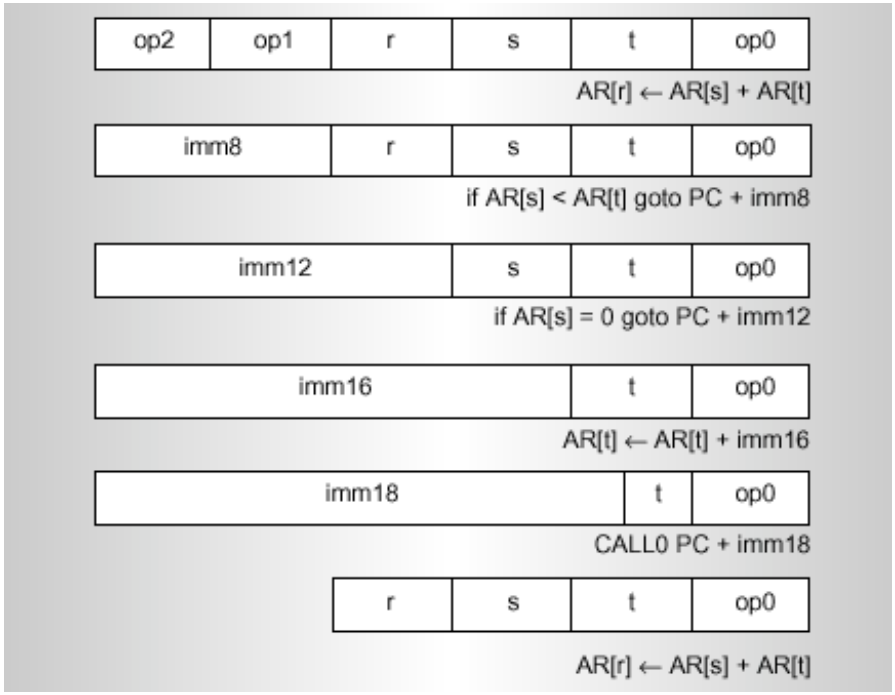


Name	Description
I	Instruction cache/RAM/ROM access Instruction cache tag comparison Instruction alignment
R	AR register file read Instruction decode, interlocking, and bypass Instruction cache miss recognition
E	Execution of most ALU-type instructions (ADD, SUB, etc.) Virtual address generation for load and store instructions Branch decision and address selection
M	Data cache/RAM/ROM access for load and store instructions Data cache tag comparison Data cache miss recognition Load data alignment
W	State writes (e.g. AR register file write)

Courtesy Tensilica

Xtensa LX Instruction Set

- The Xtensa ISA consists of 80 core instructions including both 16 and 24 bit instructions



Instruction Category	Instructions[1]
Load	L8UI, L16SI, L16UI, L32I, L32R
Store	S8I, S16I, S32I
Memory ordering	MEMW, EXTW
Jump, Call	CALL0, CALLX0, RET J, JX
Conditional branch	BALL, BNALL, BANY, BNONE BBC, BBCI, BBS, BBSI BEQ, BEQI, BEQZ BNE, BNEI, BNEZ BGE, BGEI, BGEU, BGEUI, BGEZ BLT, BLTI, BLTU, BLTUI, BLTZ
Move	MOVI, MOVEQZ, MOVGEZ, MOVLTZ, MOVNEZ
Arithmetic	ADDI, ADDMI, ADD, ADDX2, ADDX4, ADDX8, SUB, SUBX2, SUBX4, SUBX8, NEG, ABS
Bitwise logical	AND, OR, XOR
Shift	EXTUI, SRLI, SRAI, SLLI SRC, SLL, SRL, SRA SSL, SSR, SSAI, SSA8B, SSA8L
Processor control	RSR, WSR, XSR, RUR, WUR, ISYNC, RSYNC, ESYNC, DSYNC, NOP

Courtesy Tensilica

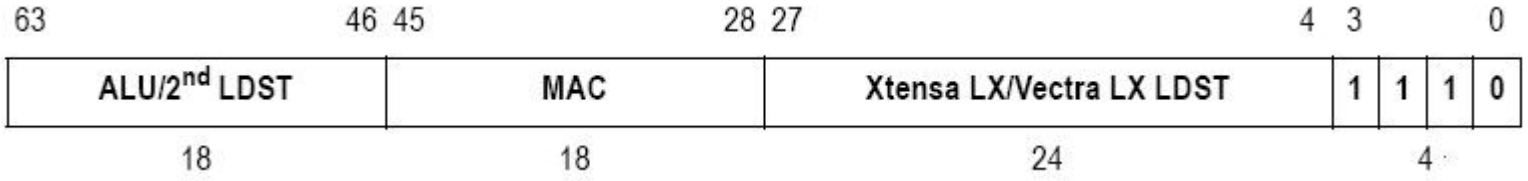
Xtensa LX ISA – Building Blocks

- **Floating Point Unit**
  - 32-bit, single precision, floating-point coprocessor
- **Vectra LX DSP Engine**
  - Optimized to handle Digital Signal Processing Applications

Courtesy Tensilica

Vectra LX DSP Engine

- FLIX (Flexible length instruction extension) Based
- Vectra LX instructions encoded in 64 bits.
  - Bits 0:3 of a Xtensa instruction determine its length and format, the bits have a value of 14 to specify it is a Vectra LX instruction
  - Bits 4:27 – contain either Xtensa LX core instruction or Vectra LX Load or Store instruction
  - Bits 28:45 – contains either a MAC instruction or a select instruction
  - Bits 46:63 – contains either ALU and shift instructions or a load and store instruction for the second Vectra LX load/store unit



Courtesy Tensilica

TIE (Tensilica Instruction Extension)

- **Method used to extend the processor's architecture and instruction set using the TIE compiler**

- **TIE Compiler**
  - Generates file used to configure software development tools so that they recognize TIE Extensions
  - Estimates hardware size of new instruction
  - You can modify application code to take advantage of the new instruction and simulate to decide if the speed advantage is worth the hardware cost

- **TIE Syntax**
  - Resembles Verilog
  - More concise than RTL (it omits all sequential logic, pipeline registers, and initialization sequences.
  - The custom instructions and registers described in TIE are part of the processor's programming model.

Courtesy Tensilica

TIE Queues and Ports

- New way to communicate with external devices
- Queues: data can be sent or read through queues. A queue is defined in the TIE and the compiler generates the interface signals required for the additional port needed to connect to the queue. Logic is also automatically generated
- Import-wire: processor can sample the value of an external signal
- Export-state: drive an output based on TIE

Courtesy Tensilica

TIE

- **TIE Combines multiple operations into one using:**
  - **Fusion**
  - **SIMD/Vector Transformation**
  - **FLIX**

Courtesy Tensilica

Fusion

- Allows you to combine dependent operations into a single instruction

Consider: computing the average of two arrays

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

Two Xtensa LX Core instructions required, in addition to load/store instructions

Courtesy Tensilica

Fusion

Fuse the two operations into a single TIE instruction

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

Essentially an add feeding a shift, described using standard Verilog-like syntax

Implementing the instruction in C/C++

```
#include <xtensa/tie/average.h>
unsigned short *a, *b, *c;
. . .
for( i = 0; i < n; i++)
        c[i] = AVERAGE(a[i] + b[i]);
```

Courtesy Tensilica

SIMD/Vector Transformation

- **Single Instruction, Multiple Data**
  - Fusing instructions into a "vector"
  - Allows replication of the same operation multiple times in one instruction

Consider: Computing four averages in one instruction

The following TIE code computes multiple iterations in a single instruction by combining Fusion and SIMD

```
regfile VEC 64 8 v

operation VAVERAGE{out VEC res, in VEC input0, in VEC input1} {} {
      wire [67:0] tmp = {input0[63:48] + input1[63:48],
                         input0[47:32] + input1[47:32],
                         input0[31:16] + input1[31:16],
                         input0[15:0]  + input1[15:0]   };
      assign res = {tmp[67:52], tmp[50:35], tmp[33:18], tmp[16:1]};
      }
```

Courtesy Tensilica

SIMD/Vector Transformation

- **Computing four 16-bit averages**
  - Each data vector must be 64 bits (4 x 16 bits)
- **Create new register file, new instruction**
  - VEC - eight 64-bit registers to hold data vectors
  - VAVERAGE - takes operands from VEC, computes average, saves results into VEC

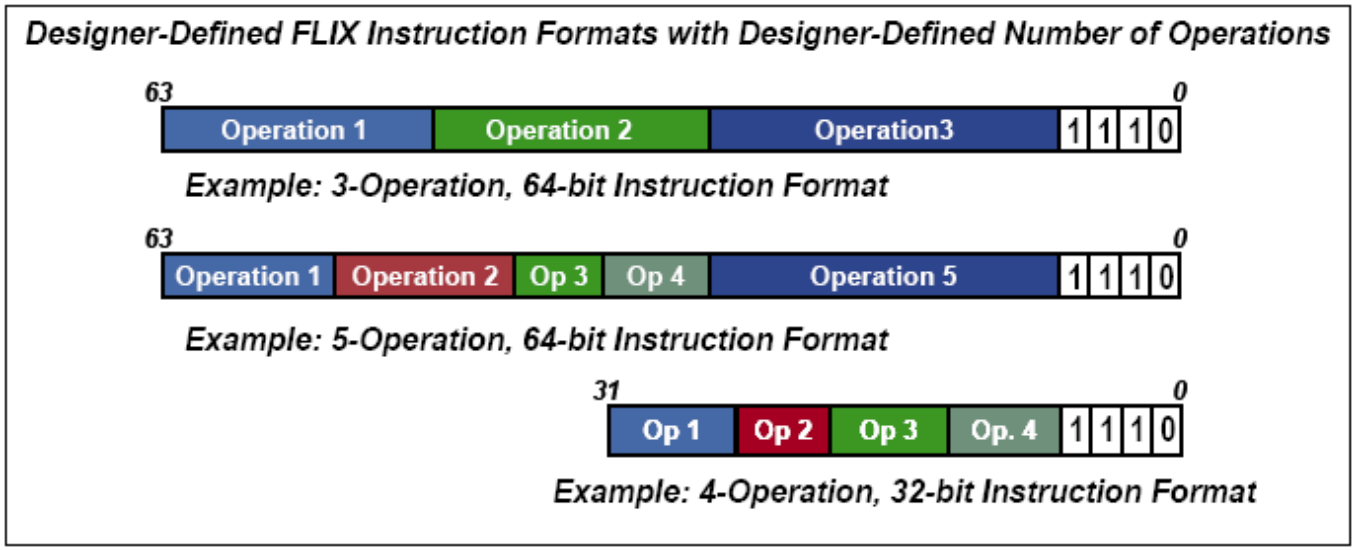
```
VEC *a, *b, *c;
for (i = 0; i < n; i += 4){
    c[i] = VAVERAGE( a[i], b[i] );}
```

- **New data type recognized**
  - TIE automatically creates new load, store instructions to move 64-bit vectors between VEC register file and memory

Courtesy Tensilica

FLIX

- Flexible length instruction extension
  - Key in extreme extensibility
  - Huge performance gains possible
  - Code size reduction without code bloat
- Similar to VLIW
- Created by XPRES Compiler



Courtesy Tensilica

FLIX - Usage

- Used selectively when parallelism is needed
- Avoids code bloat
- Used seamlessly with standard 16- and 24-bit instructions

Courtesy Tensilica

XPRES Compiler

- Powerful synthesis tool
  - Creates tailored processor descriptions
  - Run on native C/C++ code
- Three optimizations methods
- Returns optimal configurations along with pros and cons (tradeoffs)
- Analyzes C/C++ code
- Generates possible configurations
- Compares performance criteria to silicon size (cost)
- Returns possible configurations

Courtesy Tensilica

XPRES Compiler

- Analyzes C/C++ code
- Generates possible configurations
- Compares performance criteria to silicon size (cost)
- Returns possible configurations

Courtesy Tensilica