Embedded Connectivity Summit 2004

Hybrid MCU Architecture Introduction

Launched by Motorola

freescale semiconductor

Agenda

- 56800/E Family and Peripheral Introduction
  - ECS_FamilyAndPeripheralIntro.ppt
- 56800/E Development Tools Introduction
  - ECS_ToolsIntro.ppt
- 56800/E Core Introduction
  - ECS_56800ECoreIntro.ppt
- 56800/E 56F8300 Peripherals
  - ECS_56F8300_Peripherals.ppt
- 56800/E Competitive Differentiators
  - ECS_CompetitiveDifferentiators.ppt
- 56800/E Documentation and Support
  - ECS_Support.ppt
- 56800/E Summary
  - ECS_Summary.ppt

→ **Detailed Core Information**

- 56800 Core
  - ECS_56800Core.ppt
- 56800E Core
  - ECS_56800ECore.ppt

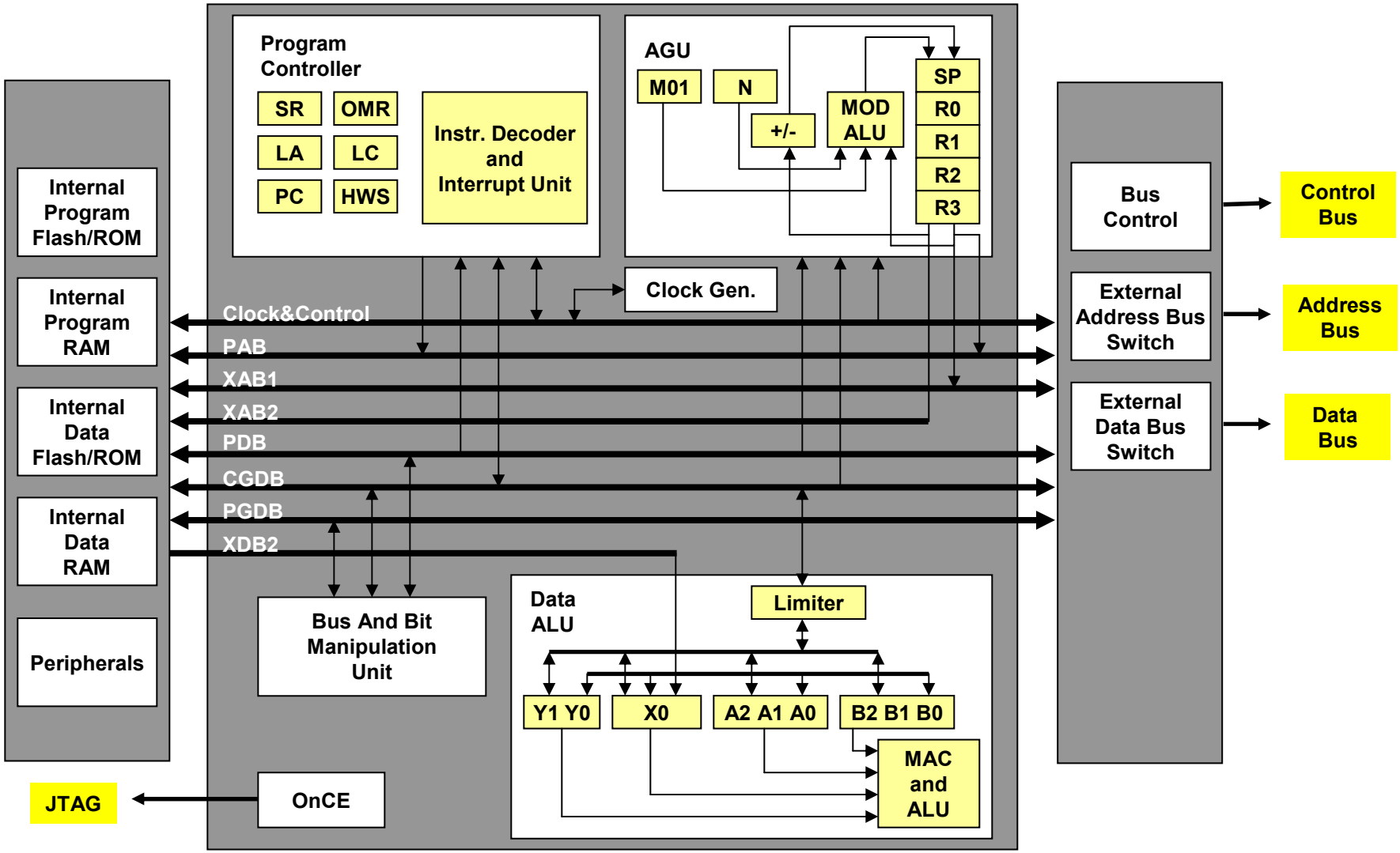
Thank You

Embedded Connectivity Summit 2004

Launched by Motorola
freescale™
semiconductor

Introduction to 56800 Architecture

DSP56800 Core Block Diagram



DSP56800 Core Features

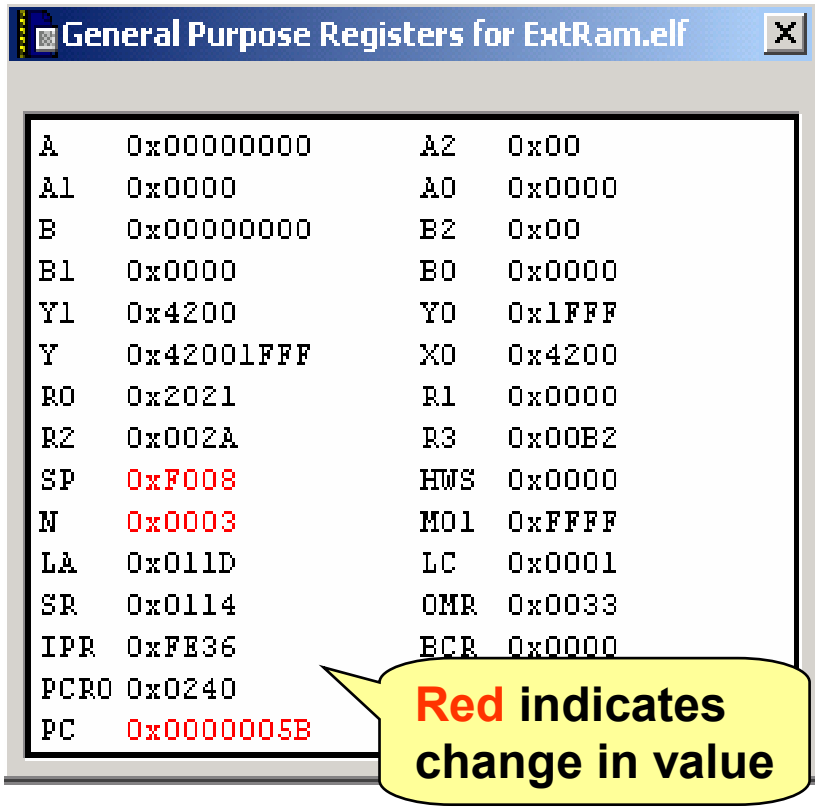
FEATURES	*BENEFITS*
• General Purpose, Register File based ALU	• Any of the ALU registers can be used as either source or destination of arithmetic operations; Code efficiency; Compiler efficiency; Programming ease
• 15 different addressing modes	• Compact code size; Efficient compiler performance; Programming ease
• True SW Stack	• Efficient compiler performance; Supports structured programming; Supports unlimited function calls; Supports local variable & parameter passing
• Parallel moves	• Compact code size; Efficient DSP operations
• Two levels of nestable interrupt support	• Flexible user-defined, multi-level interrupt priority support

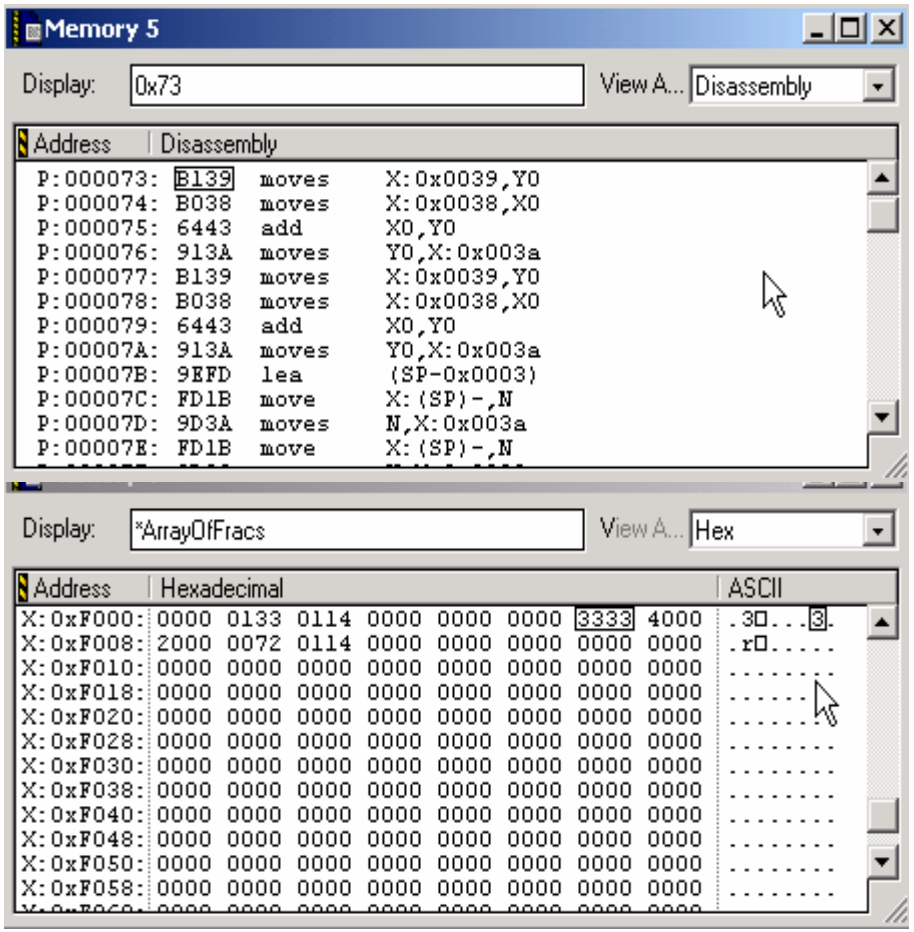
DSP56800 Core Features (2)

FEATURES	BENEFITS
• Nested, interruptible hardware and software DO loops	• Fast execution of iterative code loops and transparent support for real-time operation
• Bit Manipulation Unit	• Efficient control code and peripheral programming
• Software Programmable memory wait states.	• Supports a wide range of memory speed for performance/cost tradeoffs
• Intelligent Power Management	• Lower power consumption without user intervention or code overhead
• Multiple, user-selectable, low power modes	• Significant reduction in overall system power consumption
• 16-bit arithmetic	• Supports bit-exact standards

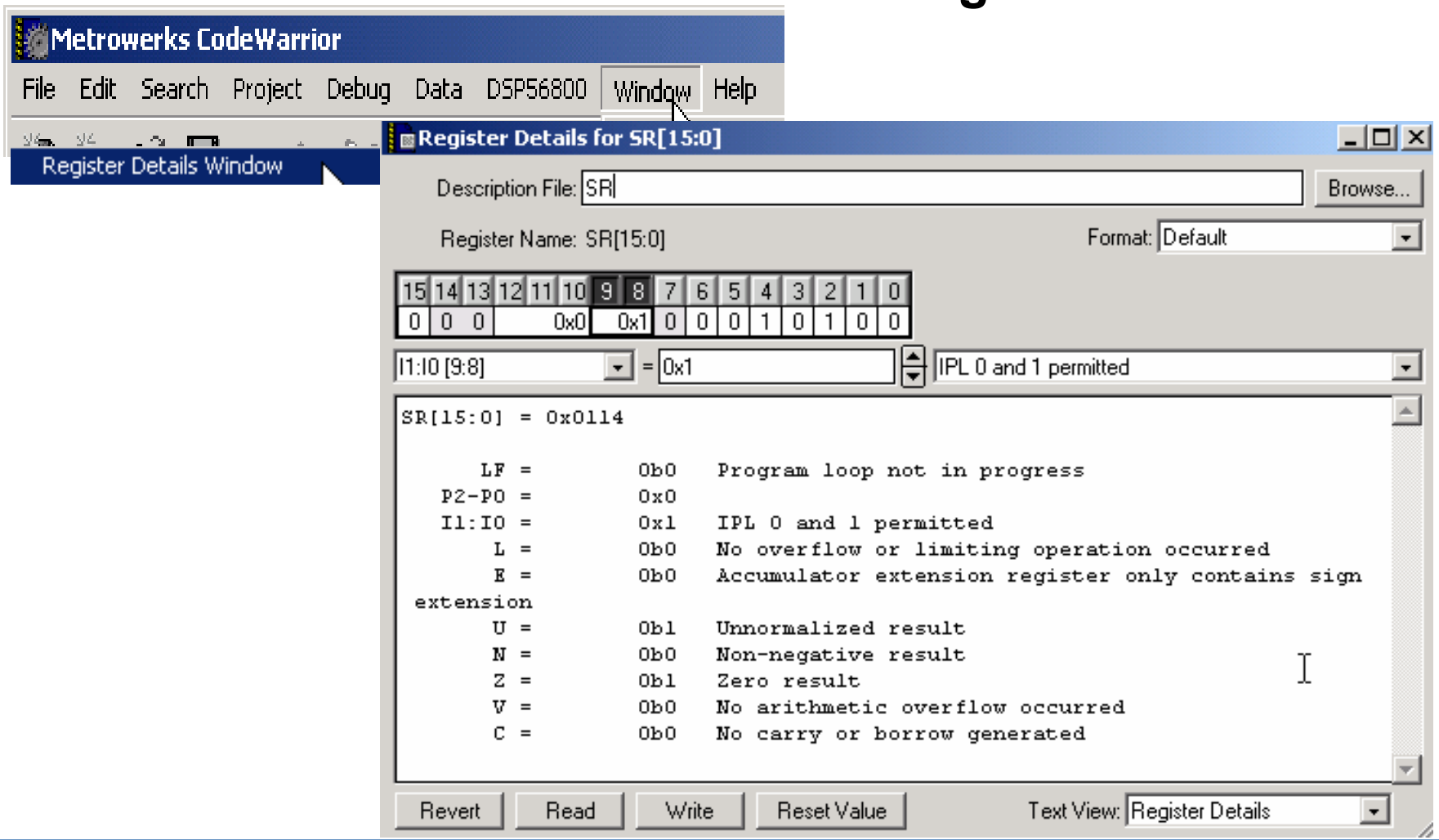
Launched by Motorola
freescale semiconductor

CodeWarrior™'s View of Core

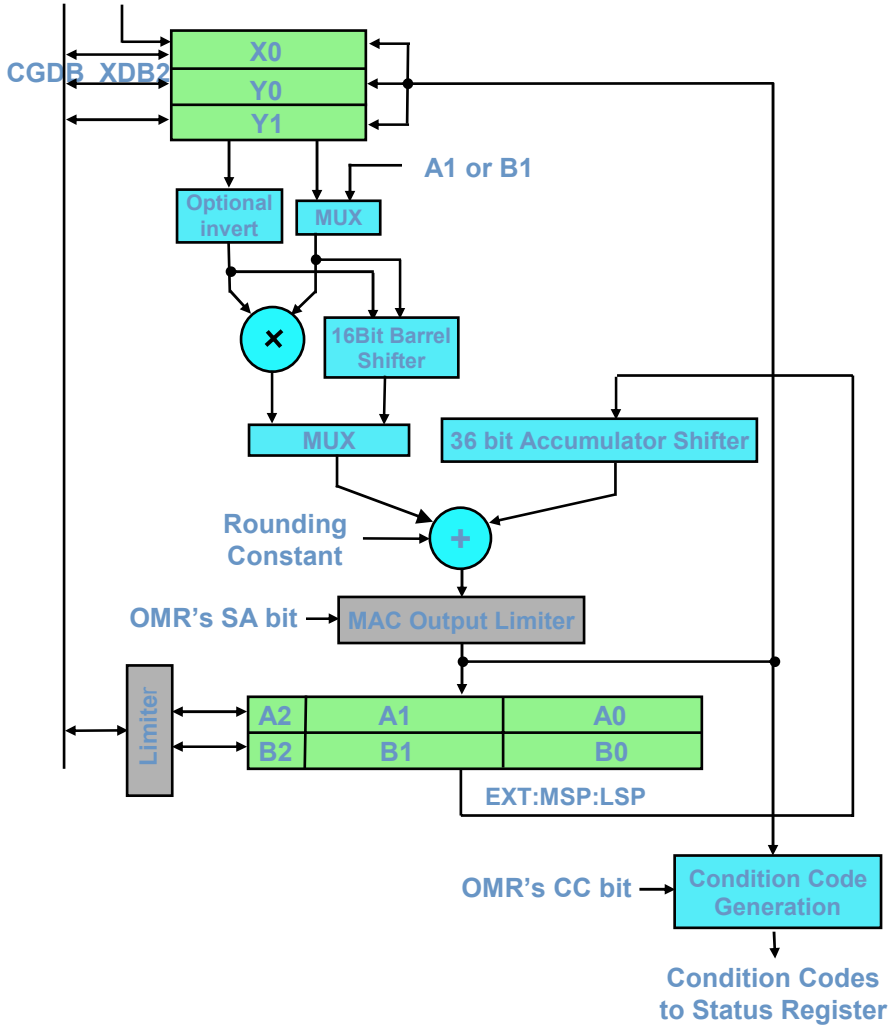




CodeWarrior's View of Register Details



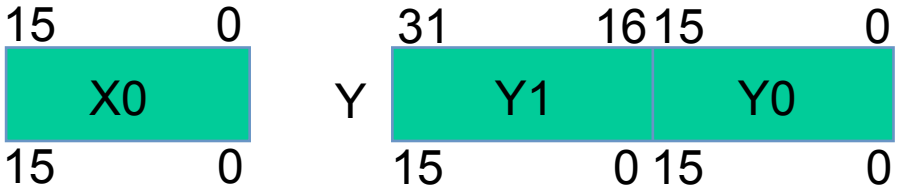
Data Arithmetic Logic Unit



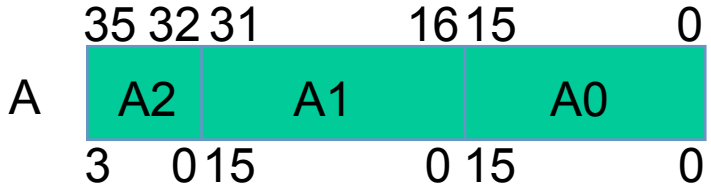
Data ALU

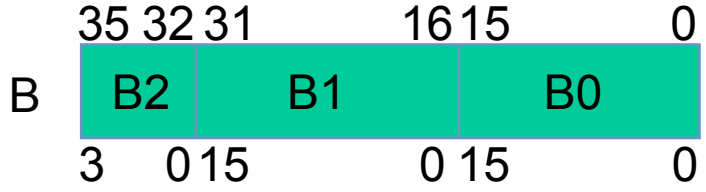
Data Arithmetic Logic Unit

Data ALU Input Registers



Accumulator Registers



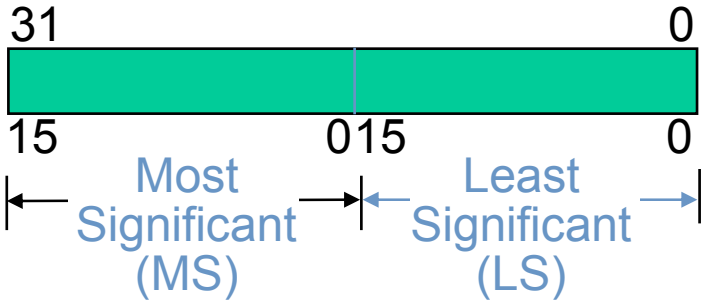


Operand Types

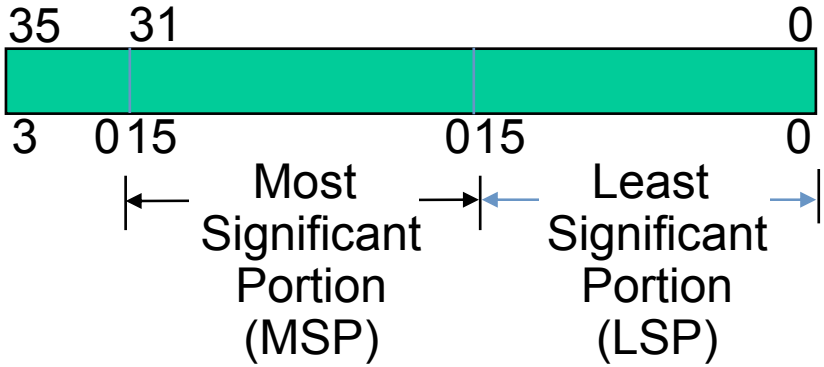
Operand Types:

Word (16 Bits)
(Integer or Fraction)

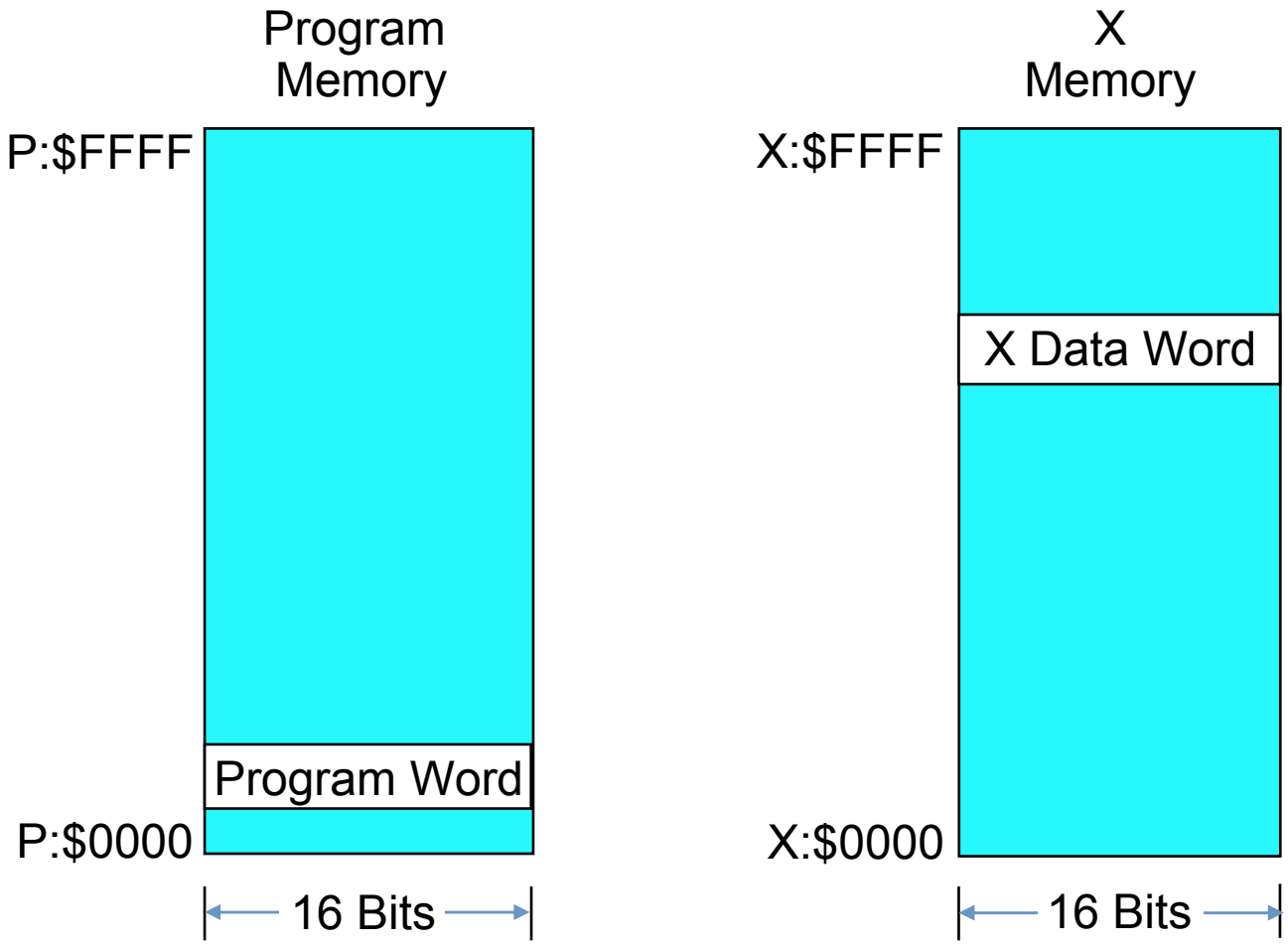
Long word (32 Bits)
(Fraction)



Accumulator (36 Bits)
(Integer/Fraction)

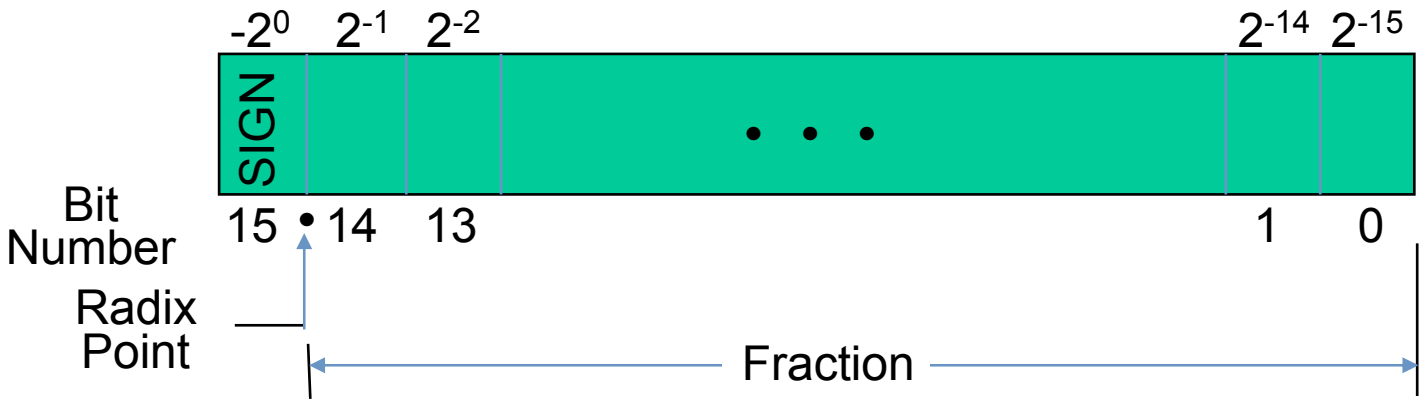


Memory Types



Two 64 K x 16-Bit Memory Spaces

Word Operand



Largest Value
$1 - 2^{-15}$ = $7FFF

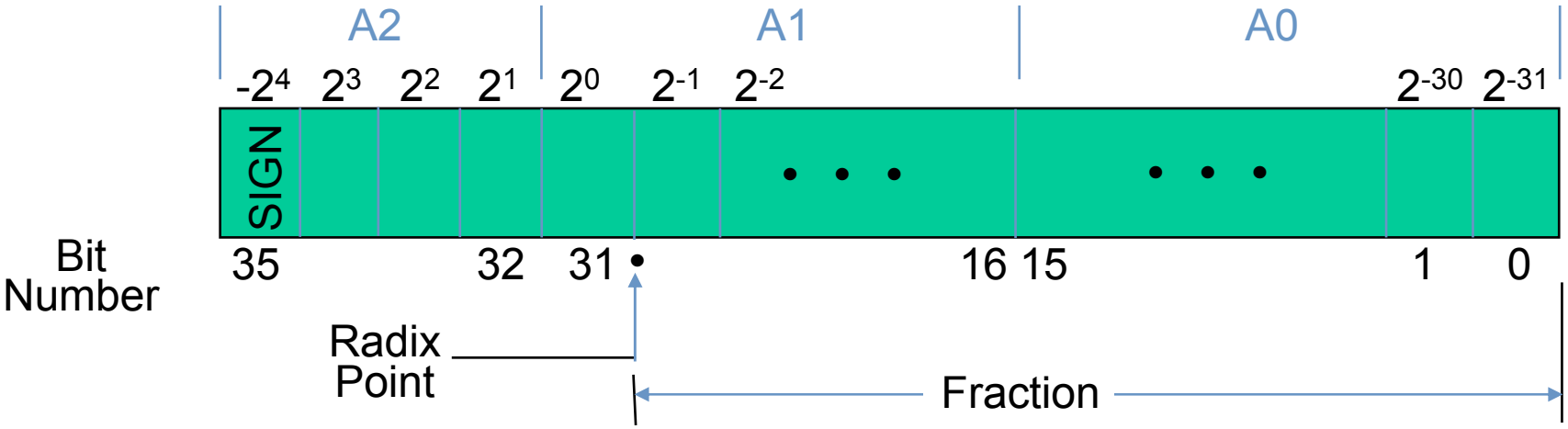
Smallest Value
-1 = $8000

Dynamic Range
96dB = $20\log_{10}(2^{16})$

Storage Locations

Program memory and X data memory
X0, Y1, Y0 Input/Destination Registers
A1, A0, B1, B0 Accumulators

Accumulator Operand



Largest Value
$16 - 2^{-15}$ = $7FFFFFFFF

Smallest Value
-16 = $800000000

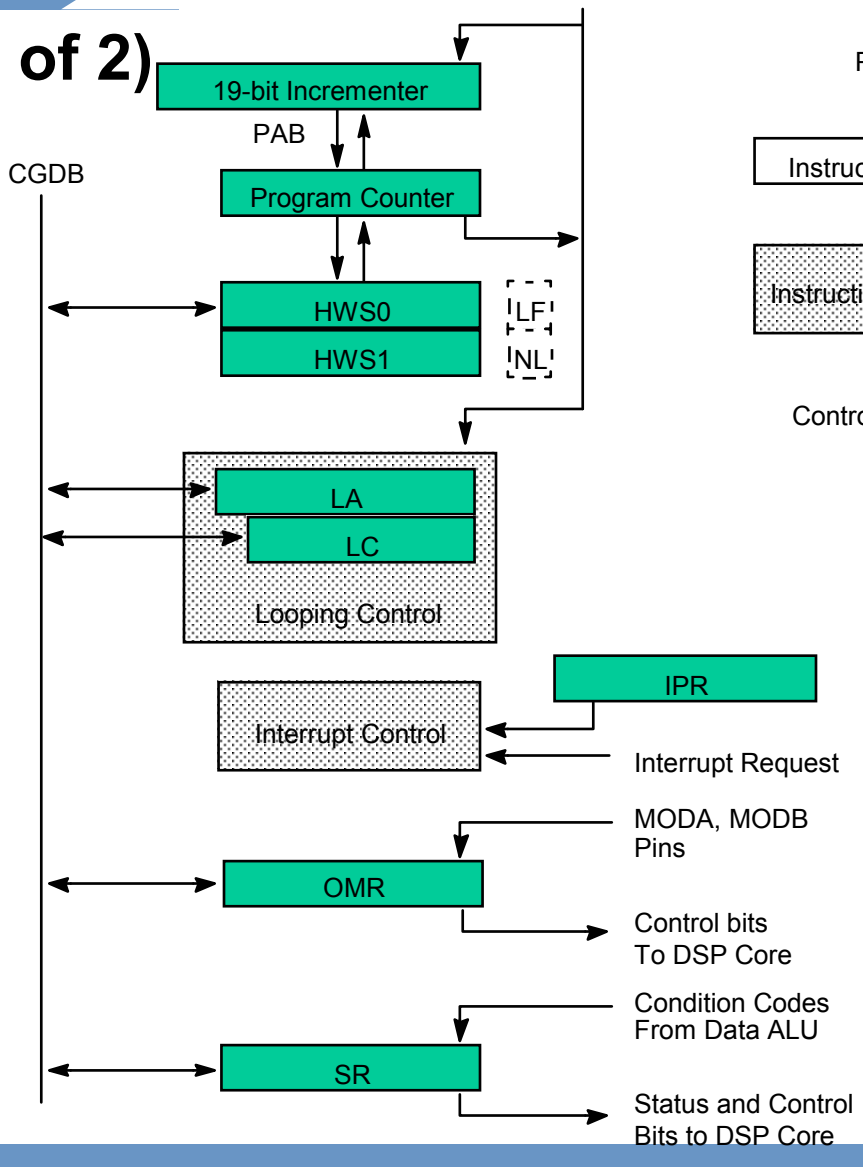
Dynamic Range
216 dB = $20\log_{10}(2^{36})$

Storage Locations
A, B Accumulators

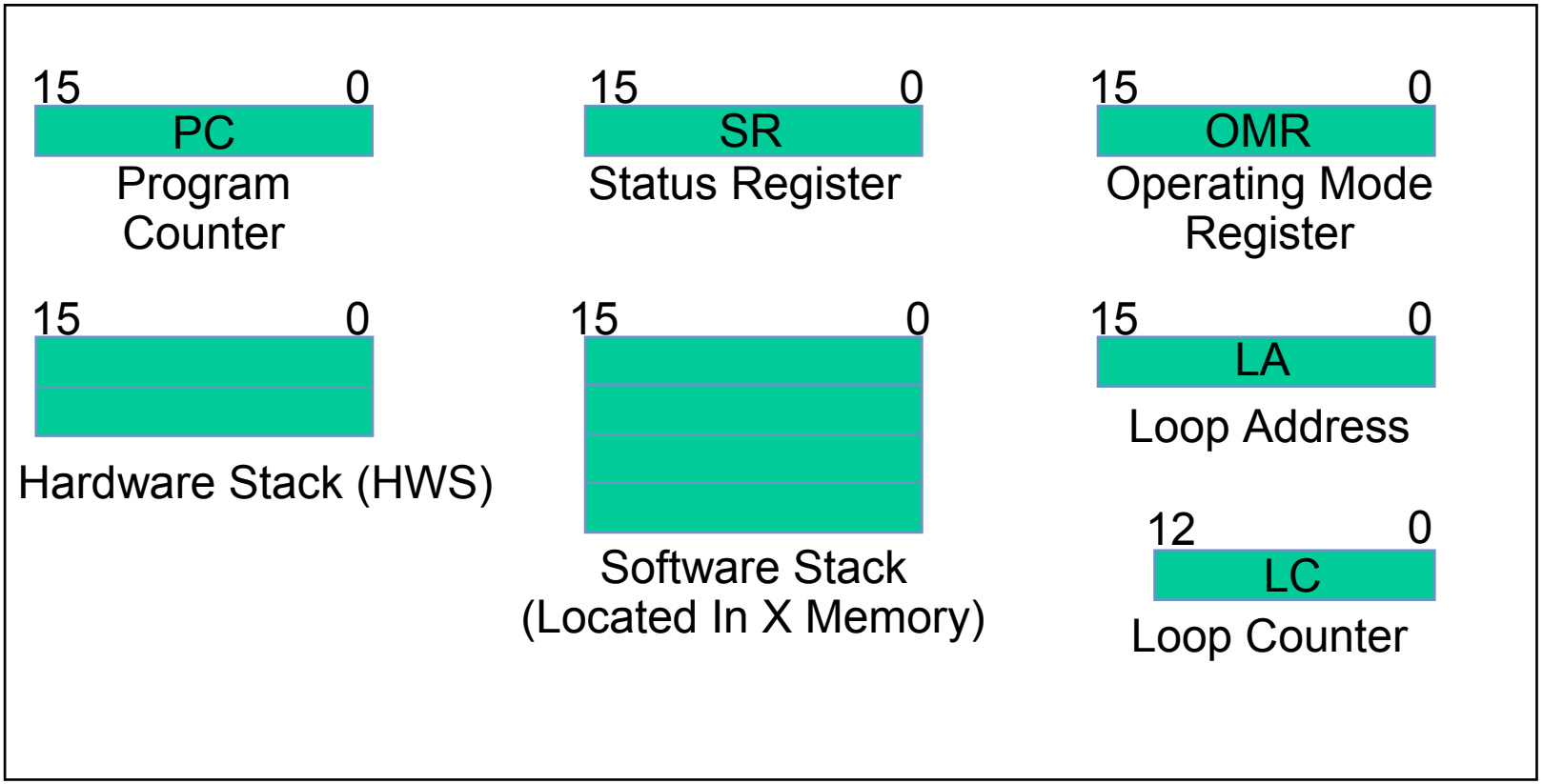
Fractional Examples

Decimal Fraction	16-Bit Binary Value	Hexadecimal Coefficient
0.750	0.1100000 00000000	$6000
0.500	0.1000000 00000000	$4000
0.250	0.0100000 00000000	$2000
0.125	0.0010000 00000000	$1000
-0.125	1.1110000 00000000	$F000
-0.750	1.0100000 00000000	$A000

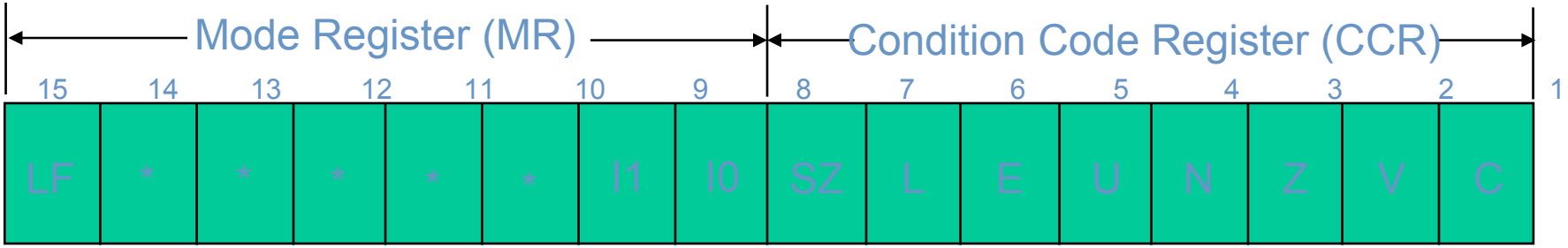
Program Controller (1 of 2)



Program Controller (2 of 2)



Status Register (SR)

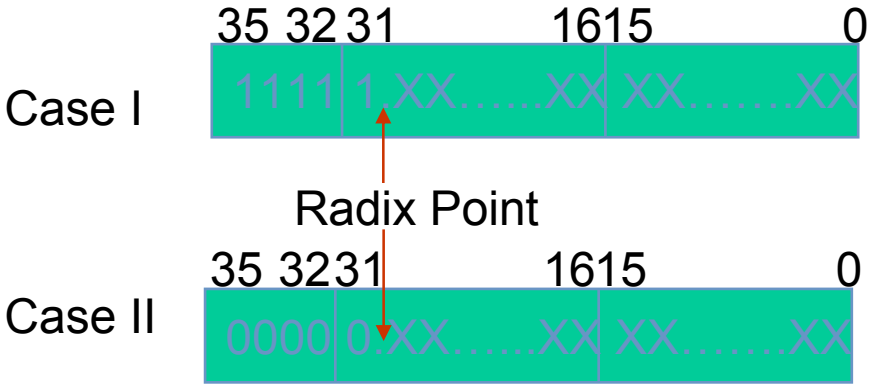


C -- Carry
V -- Overflow
Z -- Zero
N -- Negative
U -- Unnormalized*
E -- Extension*
L -- Limit*
SZ -- Size*
I0 and I1 -- Interrupt Mask
LF -- Loop Flag

Extension Bit

E -Extension bit:

- E = 0 If all the Bits of the Signed Integer Portion are the same.
  The number (n) is a fraction: +1 > n ≥ -1



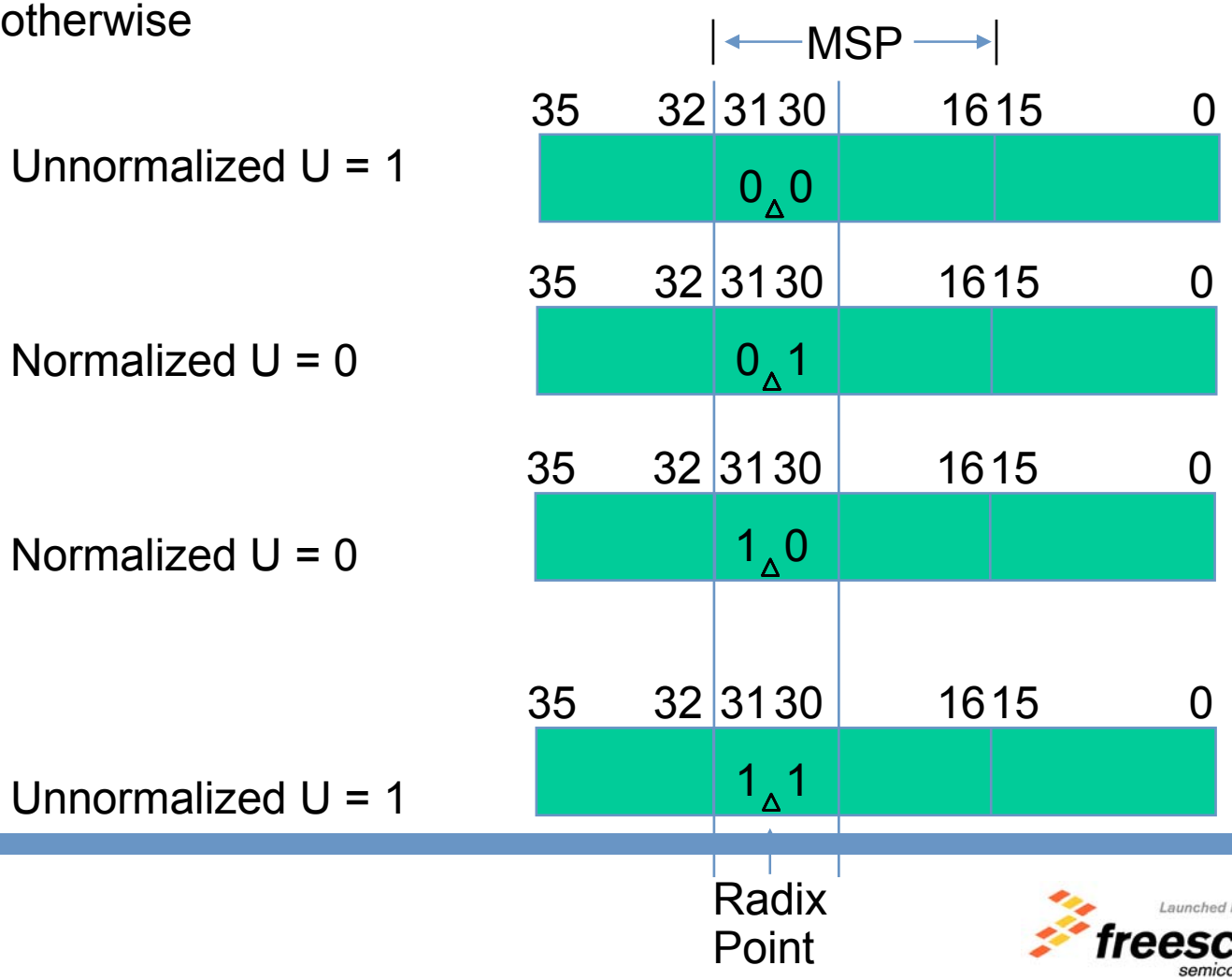
- E = 1

  For all other cases, the number (n) is an Integer plus Fraction:
  -16 ≤ n < -1 OR +1 ≤ n < +16

Unnormalized bit

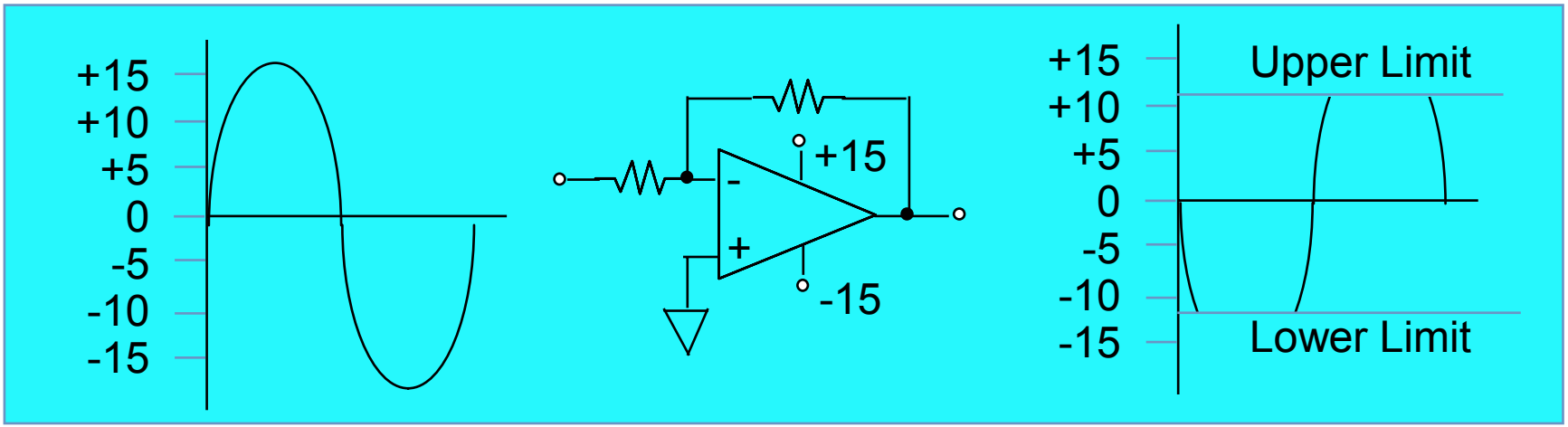
U -Unnormalized bit: When a number (n) is normalized:
$+ .5 \leq n < +1$ OR $-1 \leq n < -.5$

- U = 1 If the two most significant bits of the MSP are the same
  Cleared otherwise



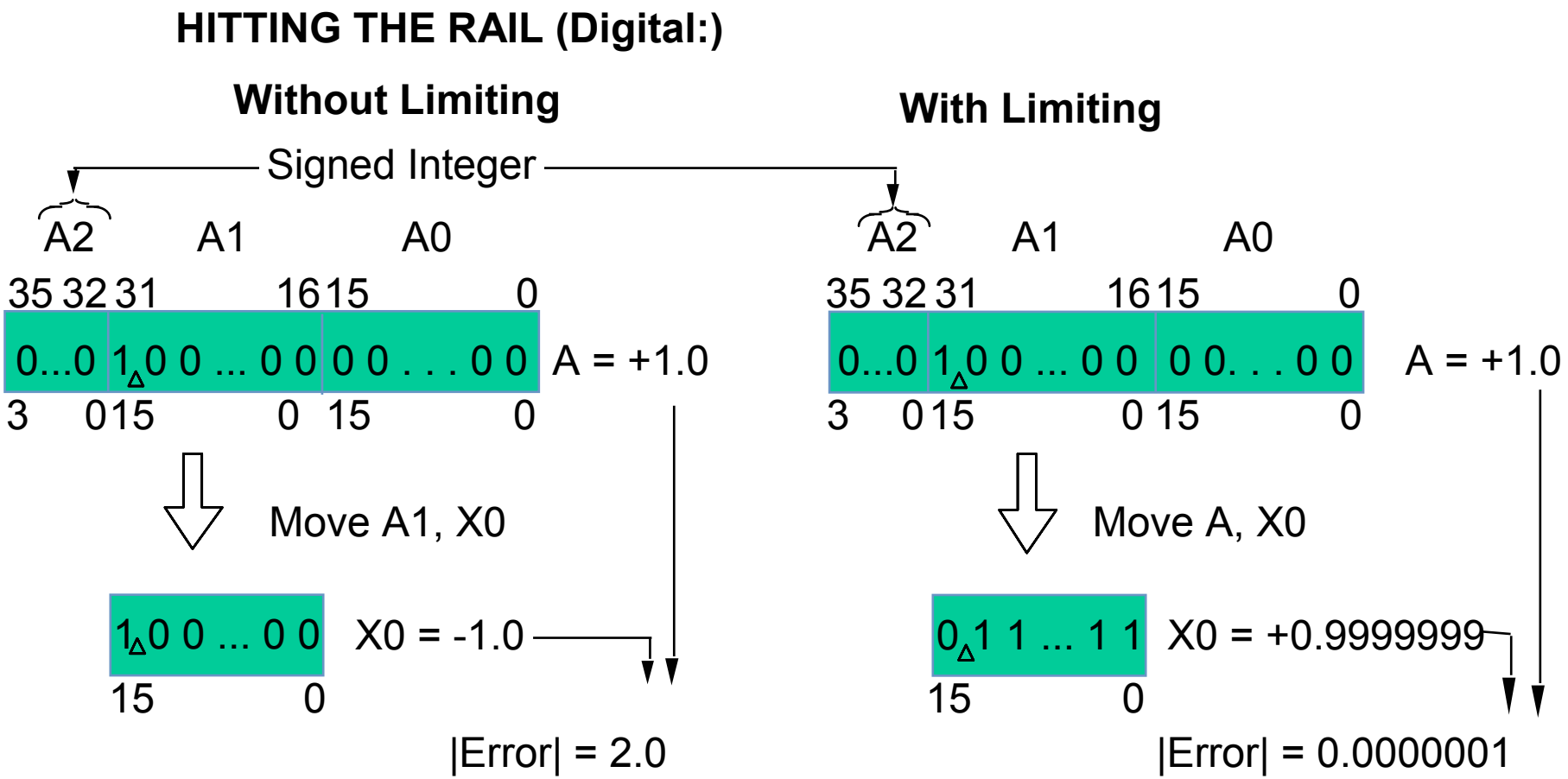
Saturation Arithmetic
(Analog)

HITTING THE RAIL (Analog:)

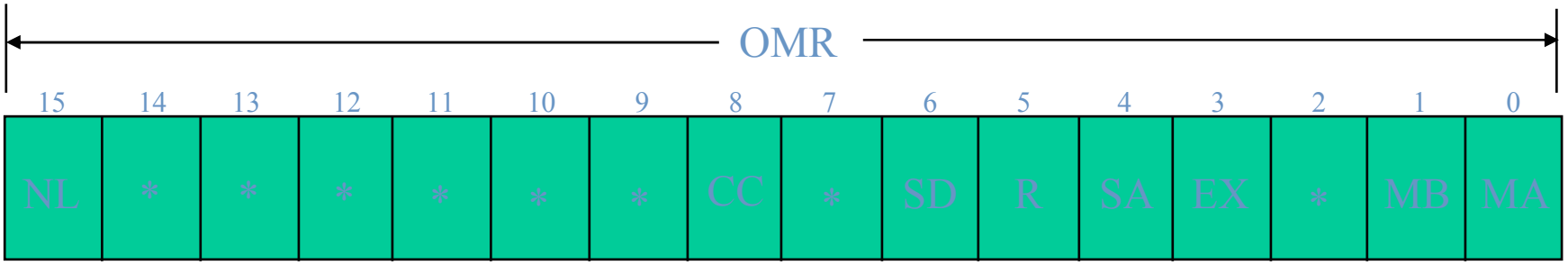


The OP-AMP has been driven into "SATURATION"

Saturation Arithmetic
(limiting)



Operating Mode Register (OMR)



MA, MB -- Operating Mode
EX -- External X (Data) Memory
SA -- Saturation
R -- Rounding
SD -- Stop Delay
CC -- Condition Code
NL -- Nested Looping

Do Loops

DO S,expr

HWS[0] 1st instruction in loop

LA last instruction word in loop

expr instruction

Software Loops

Data ALU Register Used for Loop Count

```
        MOVE #3,X0          ; Load loop count to execute the loop three times
LABEL                       ; Enters loop at least once
        (instructions)
        DECW X0
        BGT LABEL           ; Back to top-of-loop if positive and not 0
```

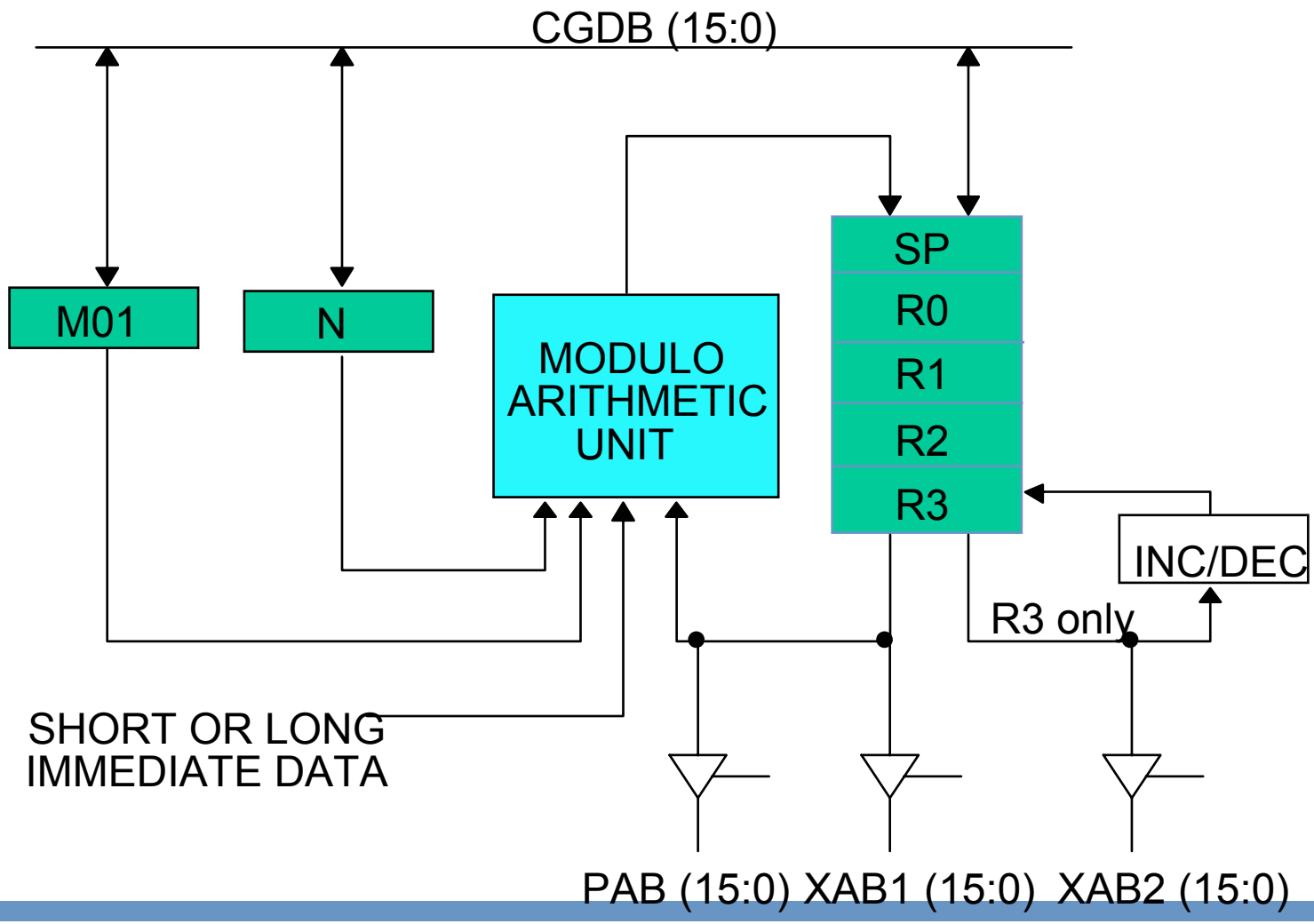
AGU Register Used for Loop Count

```
        MOVE #3-1,R2        ; Load loop count to execute the loop three times
LABEL                       ; Enters loop at least once
        (instructions)
        TSTW (R2)-
        BGT LABEL           ; Back to top-of-loop if positive and not 0
```

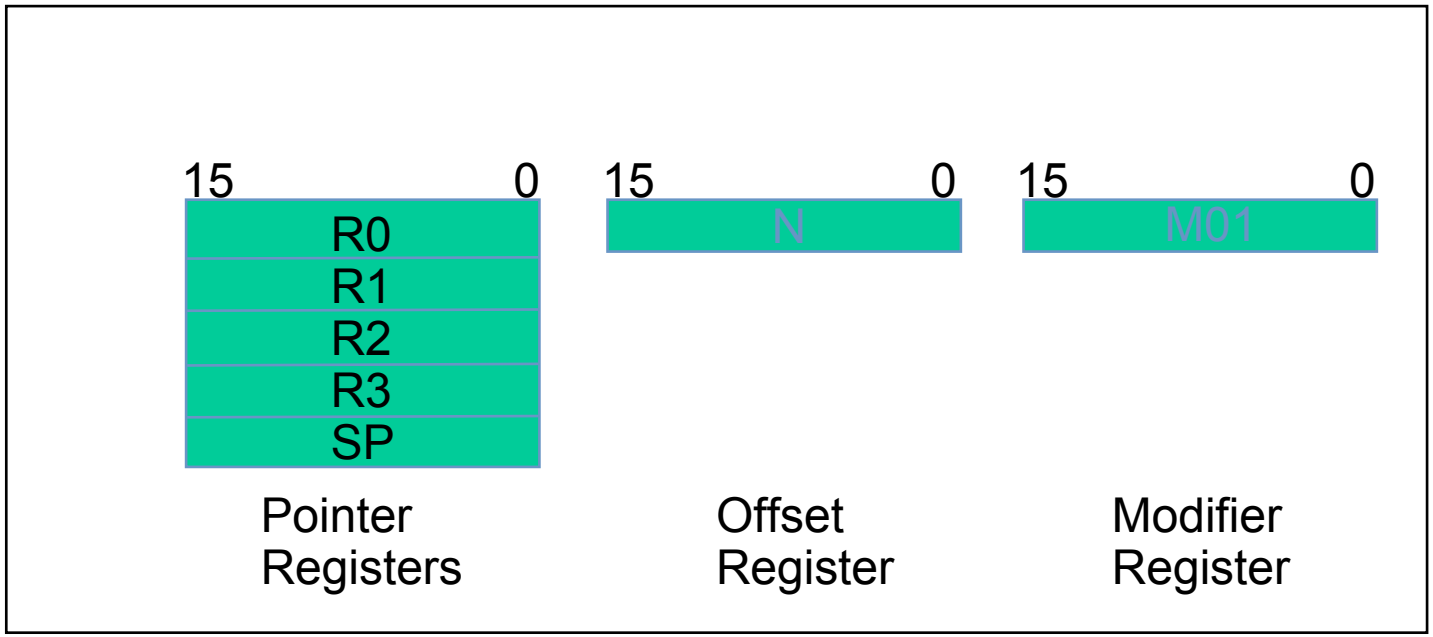
Memory Location (one of first 64 XRAM locations) Used for Loop Count

```
        MOVE #3,X:$7        ; Load loop count to execute the loop three times
LABEL                       ; Enters loop at least once
        (instructions)
        DECW X:$7
        BGT LABEL           ; Back to top-of-loop if positive and not 0
```

AGU Block Diagram

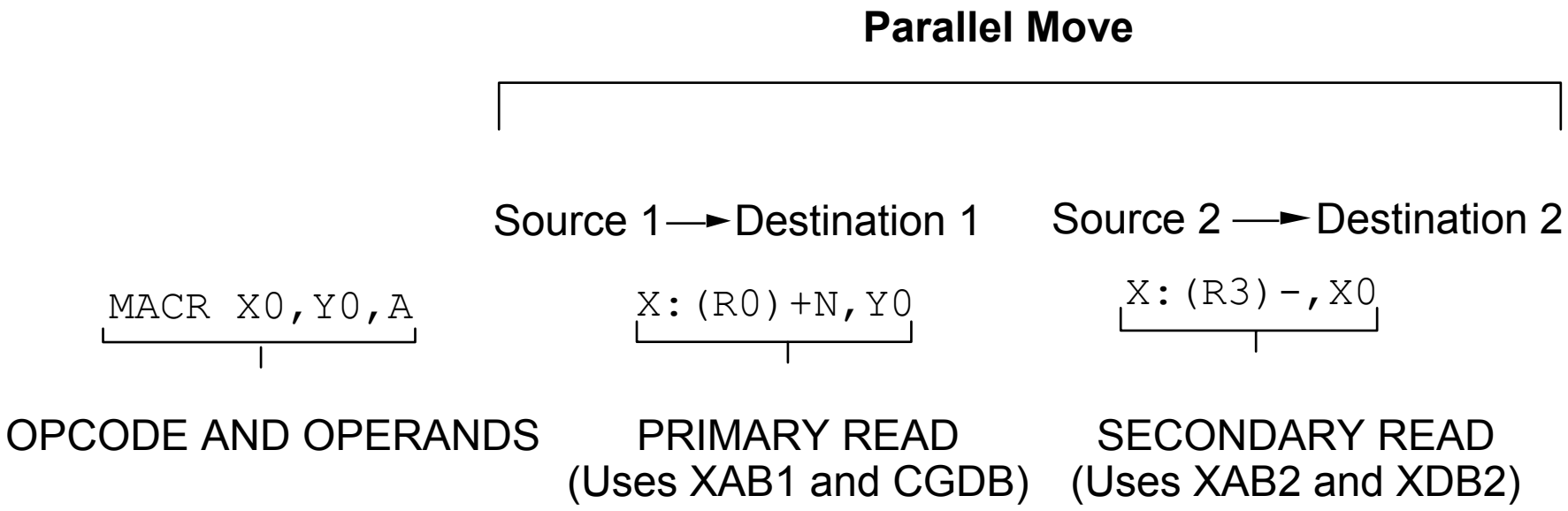


Address Generation Unit



Instruction Format

$$y(n) = \sum_{i=0}^{N-1} c(i)x(n-i)$$

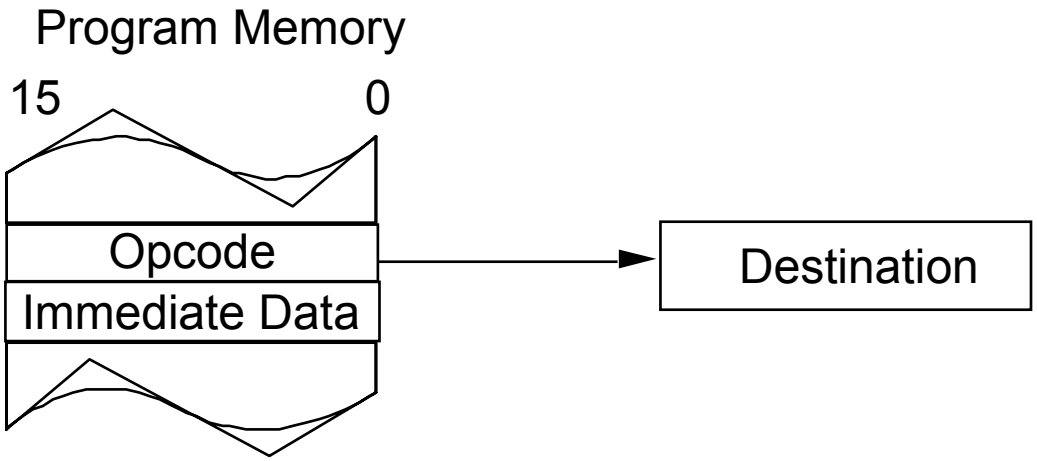


Special Addressing Modes

• Implicit	`RTS`	
• Register direct	`MOVE`	`A1,N`
• Immediate Data	`ADD`	`#$2000,X1`
• Immediate Short Data	`MOVE`	`#$001C,B1`
• Absolute Address	`SUB`	`X:$0800,A`
• Absolute Short Address	`MOVE`	`X:$001C,R2`
• I/O Short Address	`MOVE`	`X:$FFEC,R2`

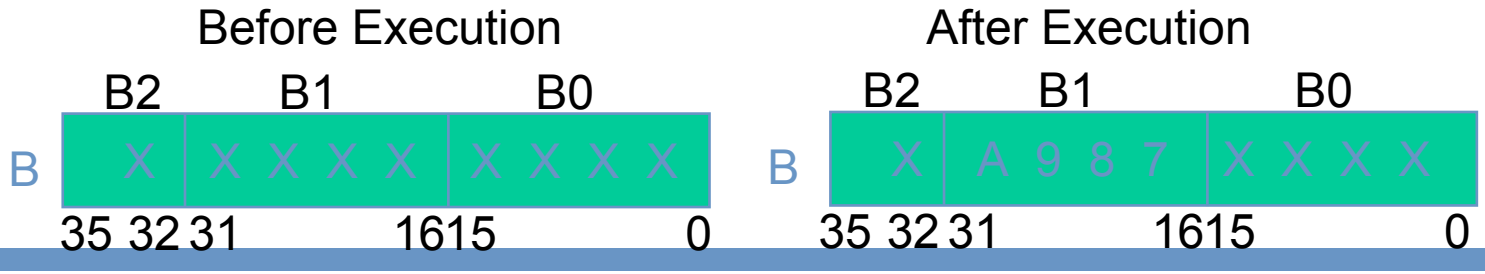
Immediate Data 1 of 2

Assembler Syntax: #xxxx
Operands Referenced: P Memory



16 Bit Destination

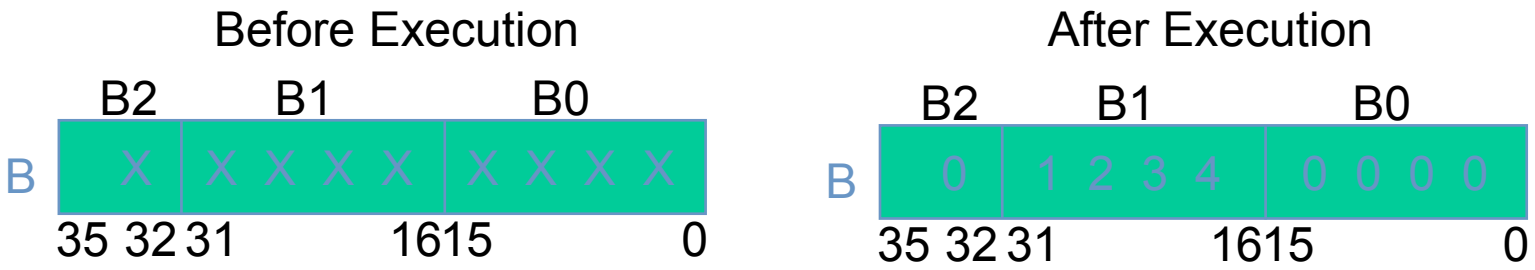
Immediate Into 16-bit Register Example: `MOVE #$A987,B1`



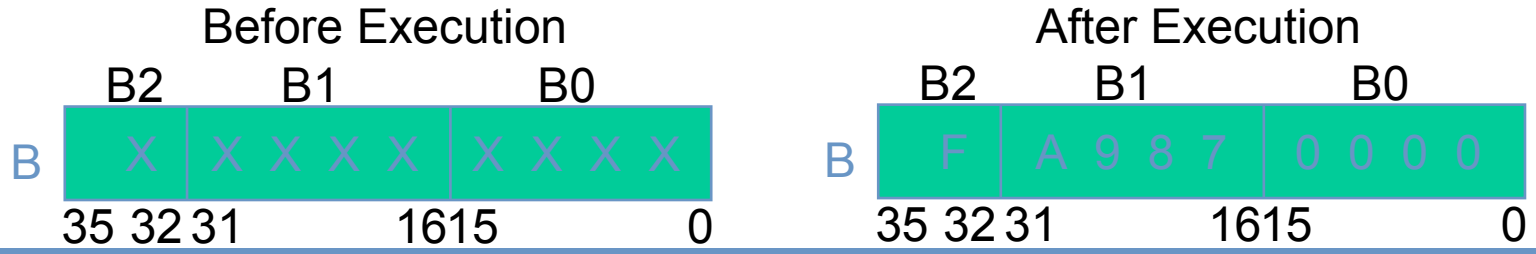
Immediate Data 2 of 2

IMMEDIATE DATA INTO ACCUMULATOR

Positive Immediate Into 36-bit Accumulator Example: `MOVE #$1234,B`

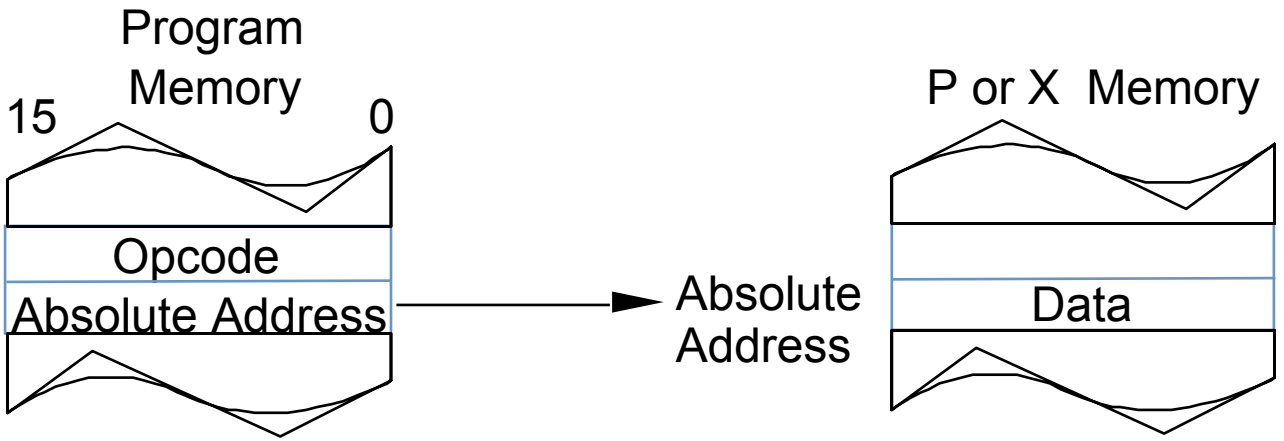


Negative Immediate Into 36-bit Accumulator Example: `MOVE #$A987,B`

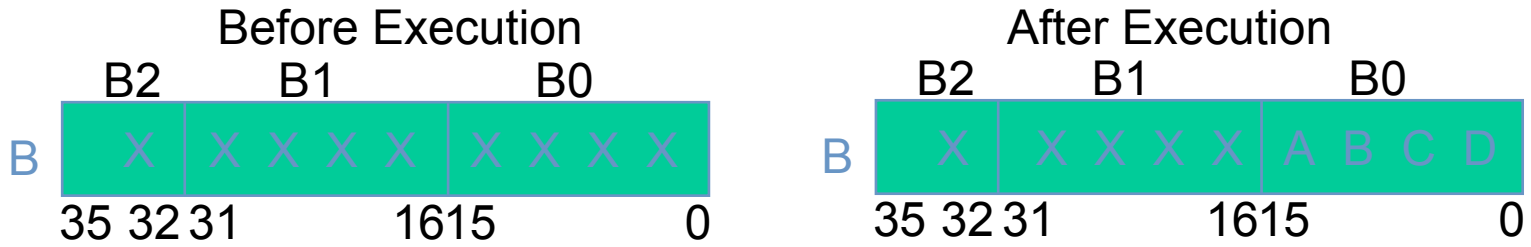


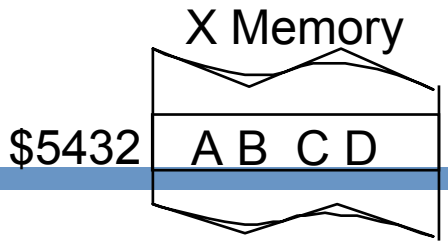
Absolute Addressing

Assembler Syntax: xxxx
Operands Referenced: P, X Memories



Example: `MOVE X:$5432,B0`



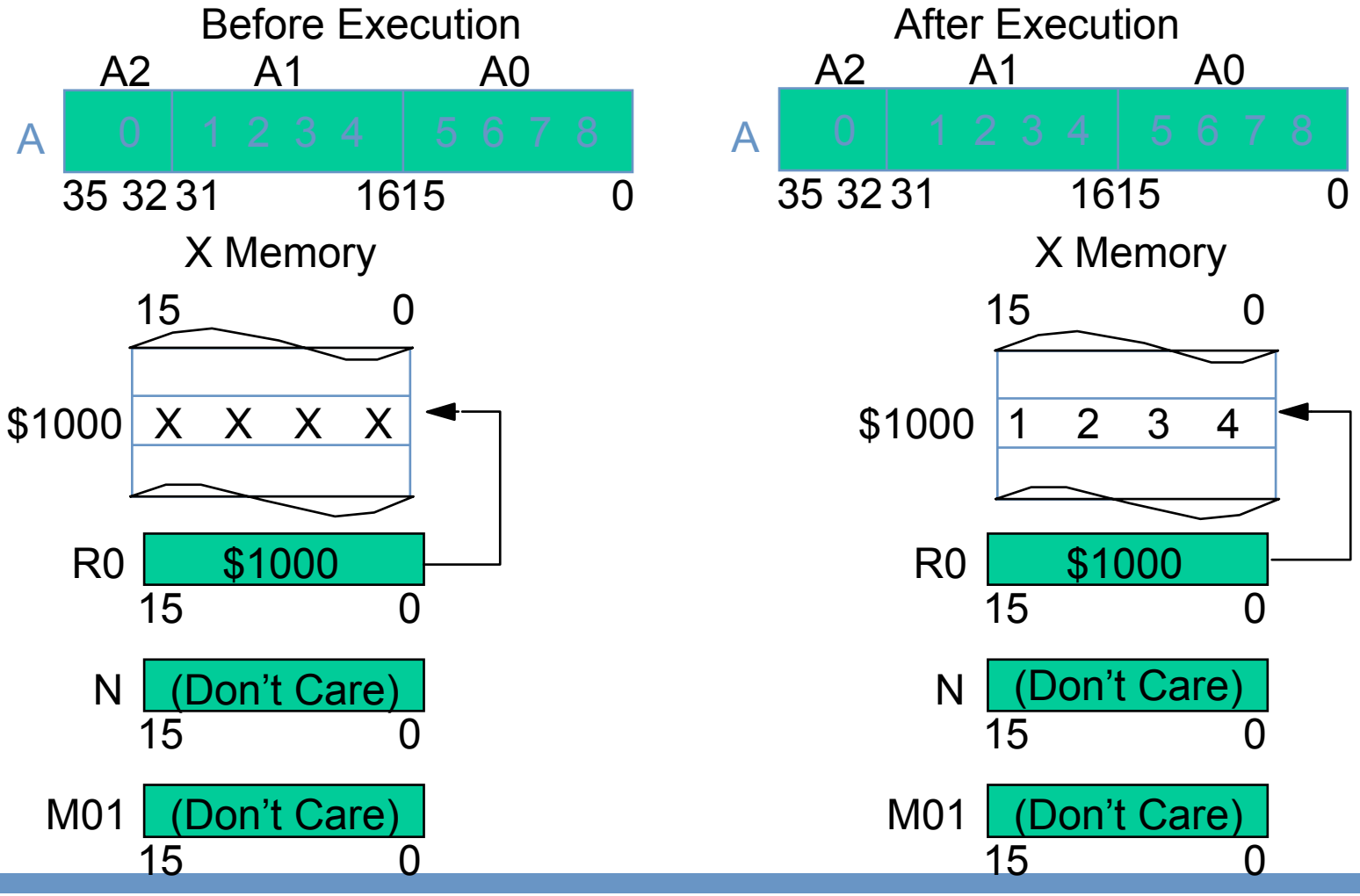


Address Register Indirect Addressing Modes

No Update	`MOVE A1,X:(Rn)`
Post Increment by One	`MOVE A1,X:(Rn)+`
Post Decrement by One	`MOVE X:(Rn)-,B1`
Post Update by N	`MOVE X0,X:(Rn)+N`
Indexed By Offset N	`MOVE X1,X:(Rn+N)`
Indexed by 16-bit Offset	`MOVE Y0,X:(Rn+xxxx)`
Indexed by 6-bit Offset for R2	`MOVE A0,X:(R2+xx)`
Indexed by 6-bit Offset for SP	`MOVE A0,X:(SP-xx)`

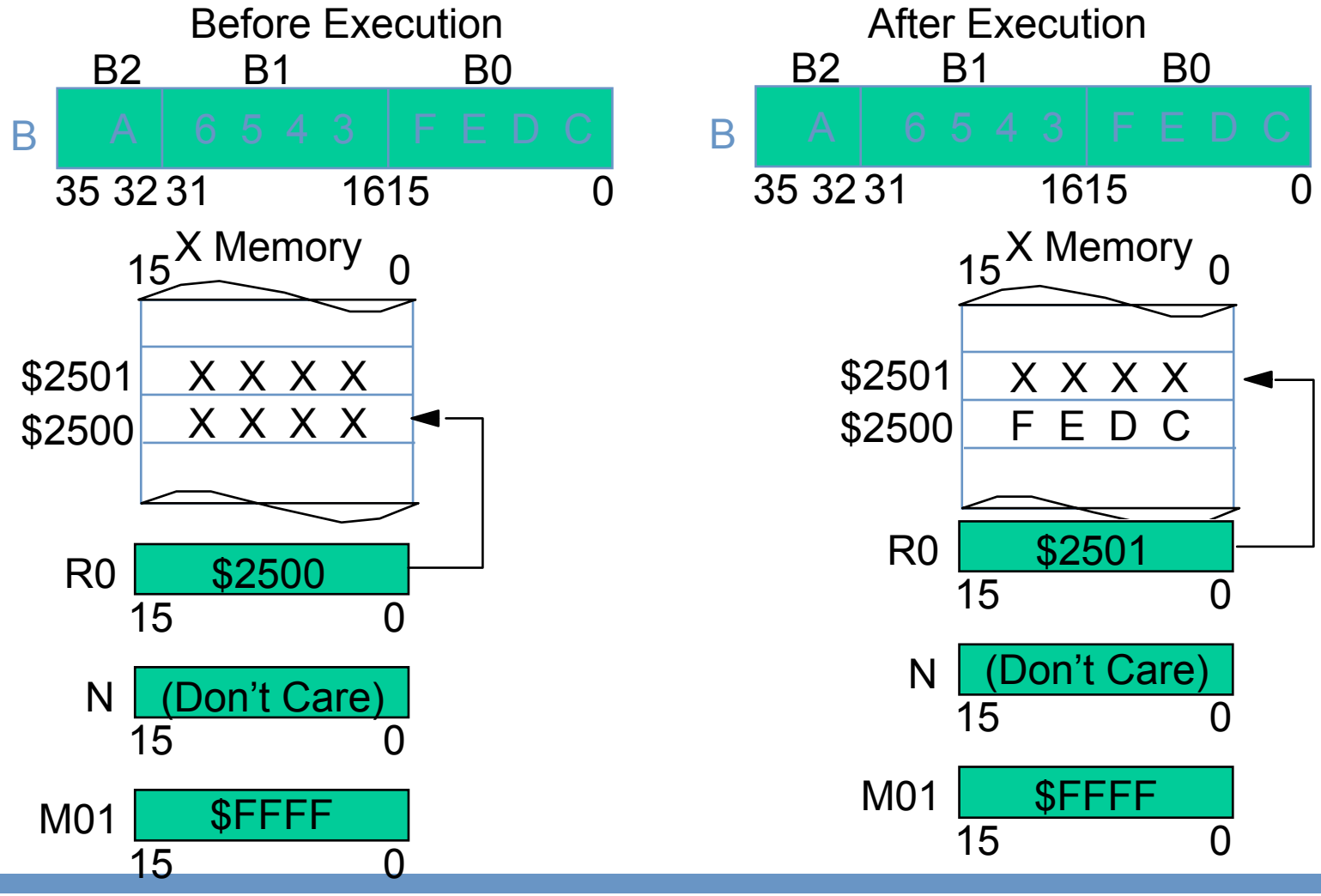
No Update

No Update Example: `MOVE A1,X:(R0)`



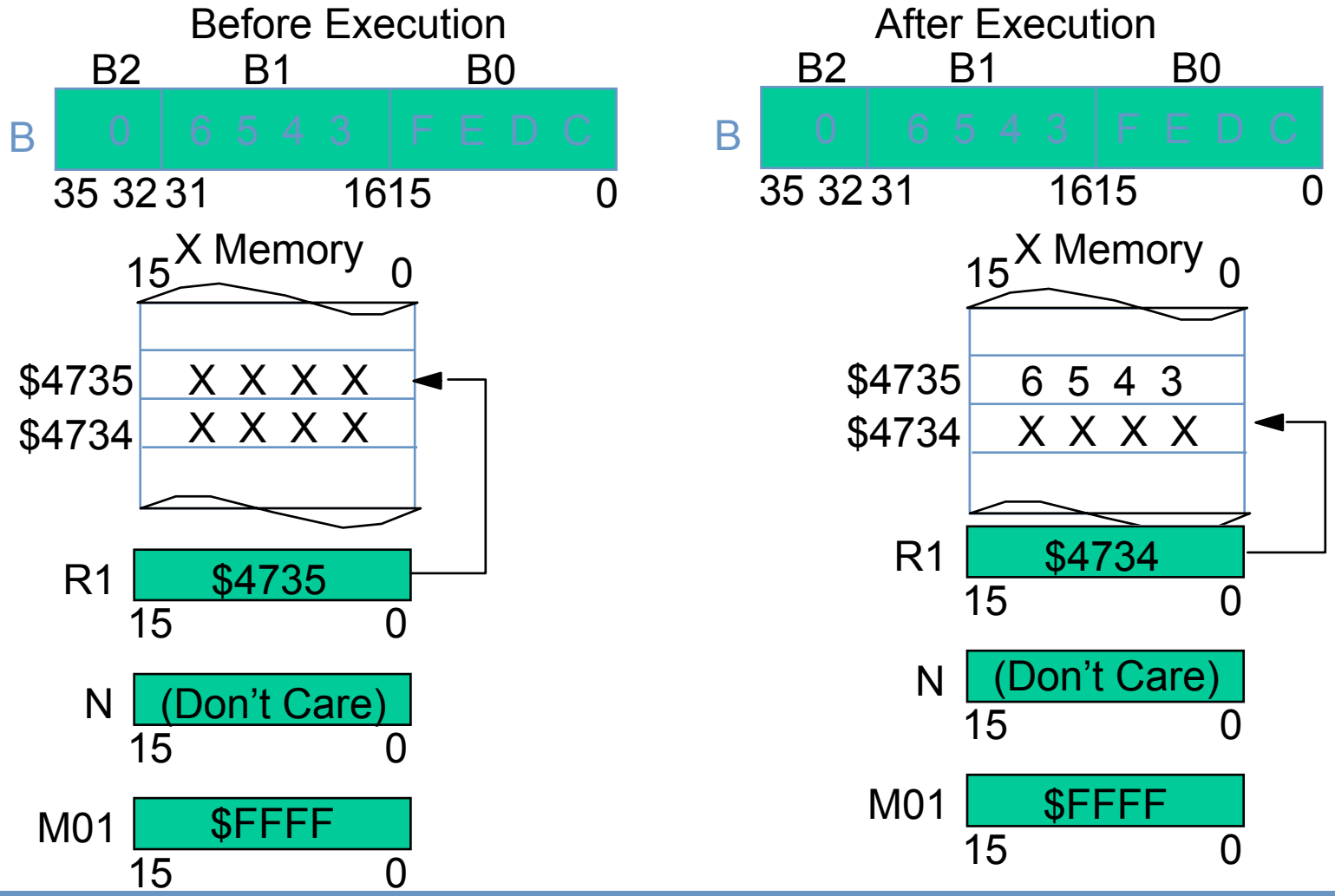
Post Increment by One

Post-increment Example: `MOVE B0,X:(R1)+`



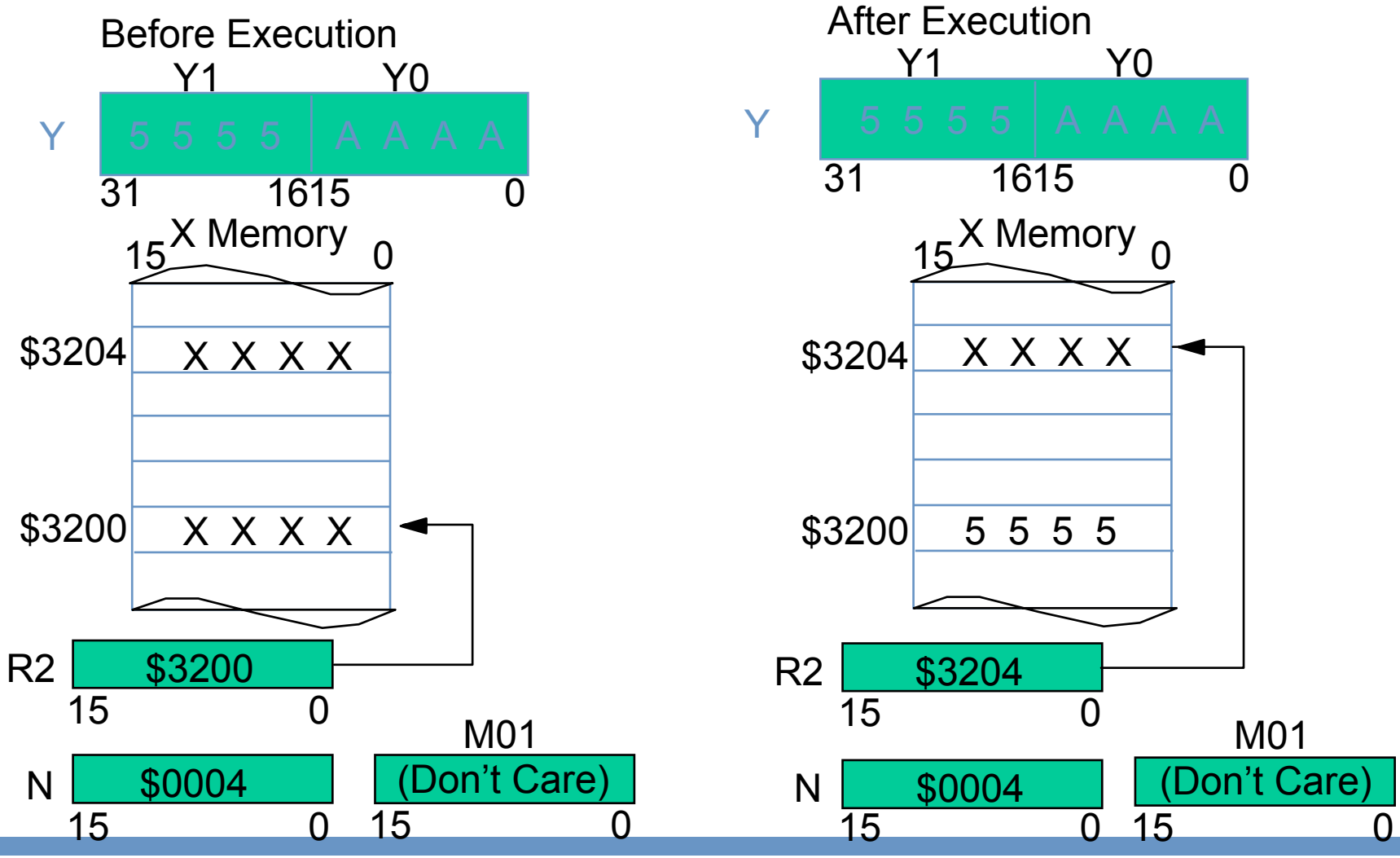
Post Decrement by One

Post-decrement Example: `MOVE B,X:(R1)-`



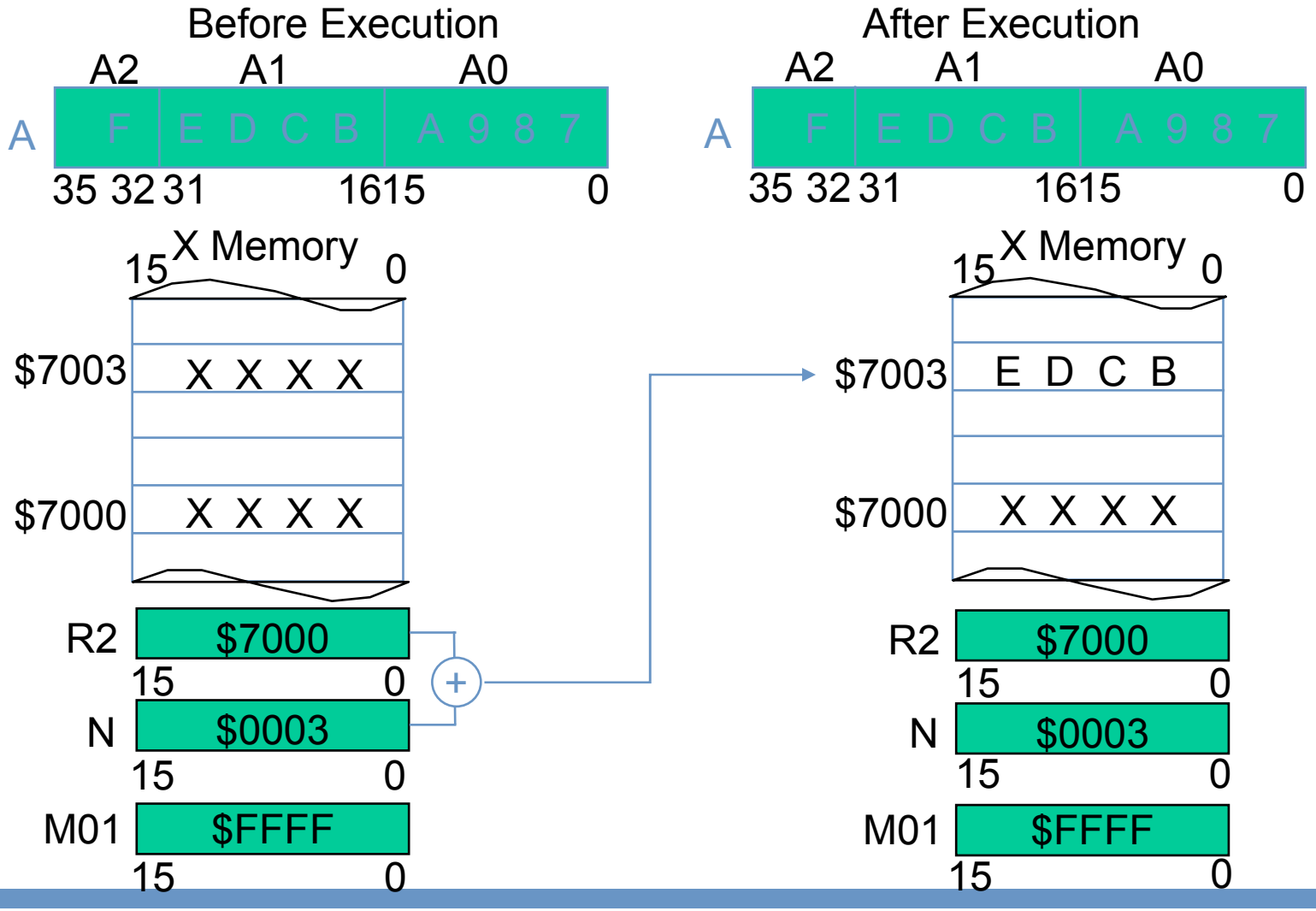
Post Update by N

Post-update By Offset N Example: `MOVE Y1,X:(R2)+N`



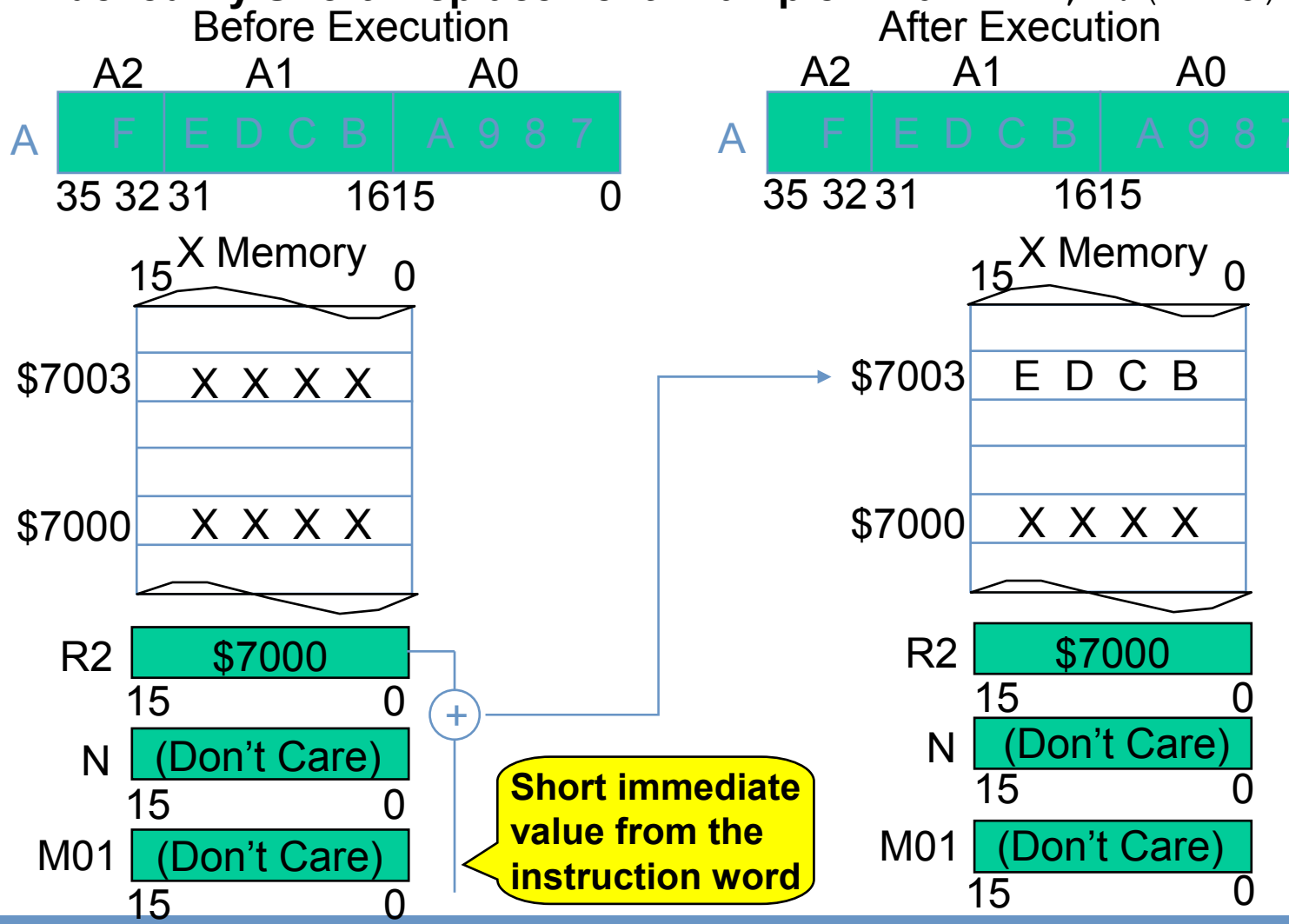
Index by Offset N

Indexed By Offset N Example: `MOVE A1,X:(R0+N)`



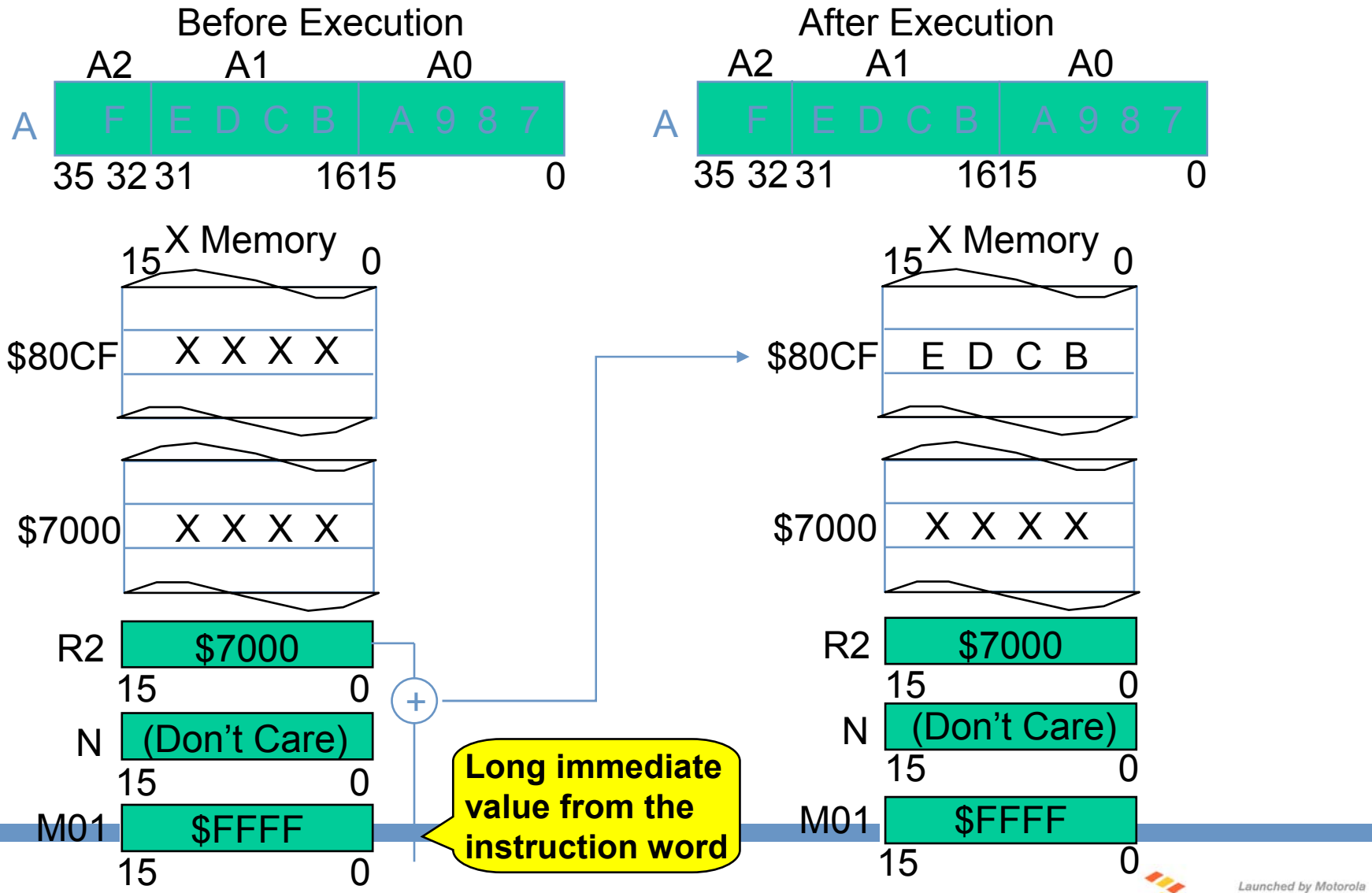
Index by Short Displacement

Indexed By Short Displacement Example: `MOVE A1,X:(R2+3)`



Index by Long Displacement

Indexed By Long Displacement Example: `MOVE A1,X:(R0+$10CF)`



Instruction Pipeline

Fetch	F1	F2	F3	F3e	F4	F5	F6	...
Decode		D1	D2	D3	D3e	D4	D5	...
Execute			E1	E2	E3	E3e	E4	...
Instruction Cycle	1	2	3	4	5	6	7	...

Pipeline Effects

A pipeline effect is encountered when an AGU register is written by one instruction and then used in the immediately following instruction as an address or in an LEA instruction.

```
; Example of AGU Pipeline Effect
    move        #$4,r0
    move        X:(r0),b    ; uses old value of r0, not "4"
    move        x0,y0
```

There are two ways to correct a pipeline effect — insert a NOP instruction between the two instructions, or rearrange the nearby instructions so that the conditions for the pipeline effect are no longer true:

```
; First Solution - Insert a NOP Instruction between the two
    move        #$4,r0
    nop                     ; inserted NOP instruction
    move        X:(r0),b    ; uses new value of r0, "4"
    move        x0,y0
```

```
; Second Solution - Rearrange instructions so no longer true
    move        #$4,r0
    move        x0,y0       ; placed between the instrs
    move        X:(r0),b    ; uses new value of r0, "4"
```

AGU Cases with Dependencies (1 of 2)

Case 1. Pipeline Dependency Caused by a Write to the N Register

```
MOVE    #$7,N              ; Write to the N register
MOVE    X:(R2)+N,X0        ; N register used in address arithmetic calculation
```

- The N register is used in the second instruction.
- This is true for using N to update R0-R3 as well as the SP register.
- The DSP core automatically stalls the pipeline by inserting one extra instruction cycle. Thus, this sequence is allowed.
- This dependency also exists for the(Rn+N) addressing mode.

Case 2. Pipeline Dependency Caused by a Bitfield Operation on the N Register

```
BFSET    #$7,N          ; Bitfield operation on the N register
MOVE     X:(R2)+N,X0    ; N register used in address arithmetic calculation
```

- The N register is used in the second instruction.
- This is true for using N to update R0-R3 as well as the SP register.
- Where a dependency is caused by a bitfield operation on the N register, this sequence is not allowed and is flagged by the assembler.
- May be fixed by rearranging the instructions or inserting a NOP between the two instructions.
- Only applies to the BFSET, BFCLR, or BFCHG instructions.
- There is no dependency for the BFTSTH, BFTSTL, BRCLR, or BRSET instructions.
- This dependency also exists for the (Rn+N) addressing mode.

AGU Cases with Dependencies (2 of 2)

Case3. Pipeline Dependency Caused by a Write to an Address Pointer Register (R0-R3, SP)

```
MOVE  #$7,R2          ; Write to the R2 register
MOVE  X:(R2)+,X0      ; R2 register used in address arithmetic calculation
```

- The address pointer register written in the first instruction is used in an address calculation in the second instruction.
- This sequence is not allowed and is flagged by the assembler.
- May be fixed by rearranging the instructions or inserting a NOP between the two instructions.

Case 4. Pipeline Dependency Caused by a Write to the Modifier Register (M01)

```
MOVE  #$7,M01         ; Write to the M01 register
MOVE  X:(R0)+,X0      ; M01 register used in address arithmetic calculation
```

- The M01 register written in the first instruction is used in an address calculation in the second instruction.
- This sequence is not allowed and is flagged by the assembler.
- May be fixed by rearranging the instructions or inserting a NOP between the two instructions.

AGU Cases With No Dependencies (1 of 2)

Case 1. No Pipeline Dependency

```
MOVE      #$7,N          ; Write to the N register
MOVE      X:(R2)+,X0     ; N not used in this instruction
```

Case 2. No Pipeline Dependency

```
MOVE      #$7,R1         ; Write to R1 register
MOVE      X:(R2)+N,X0    ; R1 not used in this instruction
```

Case 3. No Pipeline Dependency

```
MOVE      #$7,R1         ; Write to R1 register
MOVE      R1,X:$0004     ; R1 not used as a pointer or in an AGU calculation
```

AGU Cases With No Dependencies (2 of 2)

Case 4. No Pipeline Dependency

```
MOVE    #$7,R1                ; Write to R1 register
MOVE    X:(R1+$3456),X0       ; X:(Rn+xxxx) addressing mode using R1
```

This represents a special case. For the X:(Rn+xxxx) addressing mode, there is no pipeline dependency even if the same Rn register is written on the previous cycle. This is true for R0-R3 as well as the SP register. WHY????

Case 5. No Pipeline Dependency

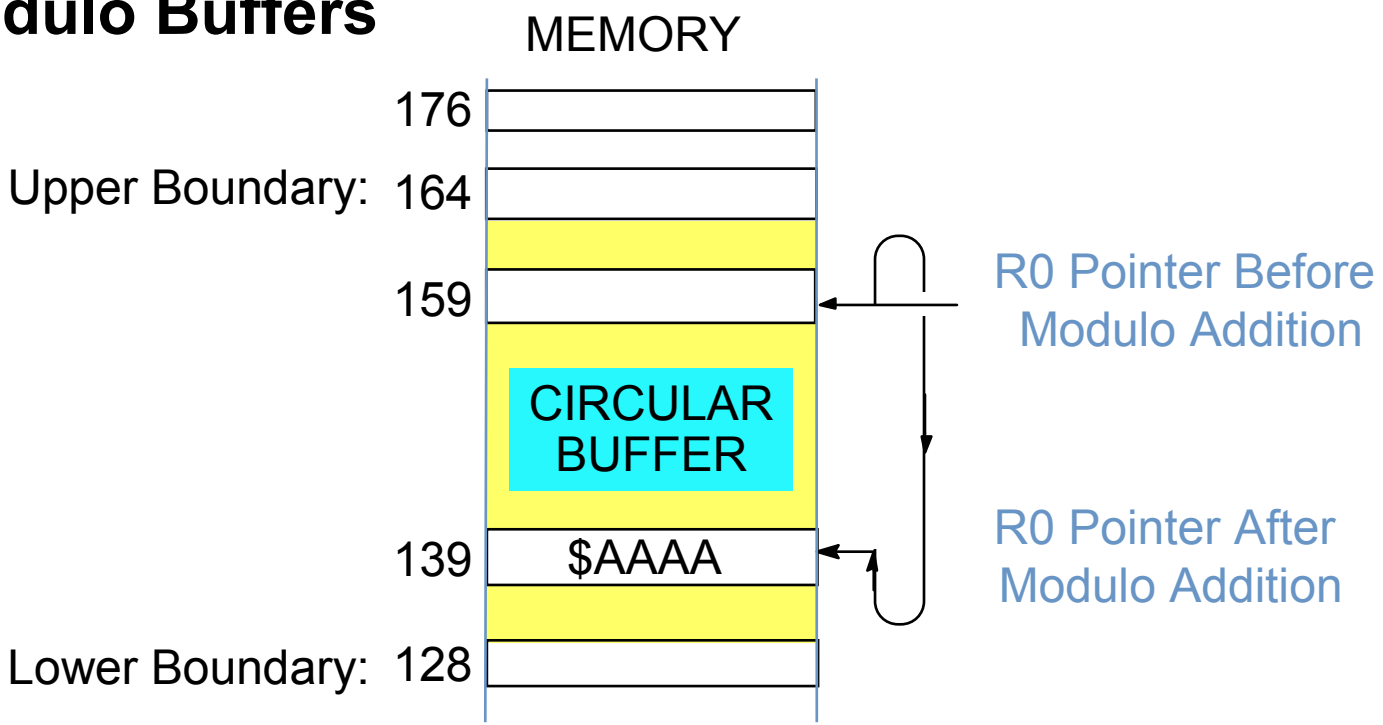
```
LEA     (R1)+N          ; Update the R1 register
MOVE    X:(R1)-,X0      ; R1 can be used in this  instruction
```

In this instruction sequence, there is no pipeline dependency since the LEA instruction is used to update the R1 register in the previous cycle. The LEA instruction is done as an address calculation and not as a MOVE so there is no pipeline dependency when the same pointer is used in the following instruction.

Question: **Is there any pipeline effect for following code?**

```
ORC     #$0008,OMR;
MOVE    X:$0100,A1;
```

Modulo Buffers




```
MOVE   #36,M01       ; M01 set up for a buffer size of 37
MOVE   #159,R0       ; R0 near upper boundary
MOVE   #17,N         ; N goes several locations past upper boundary
MOVE   #$AAAA,X0     ; Value to be written to Wrapped Address
MOVE   X0,X:(R0+N)   ; R0 wraps around to 139
```

Modulo Buffer Addition Example: 159 + 17 = 139 (i.e. wraps around)

Modulo Addressing

Addressing Mode Modifier Summary

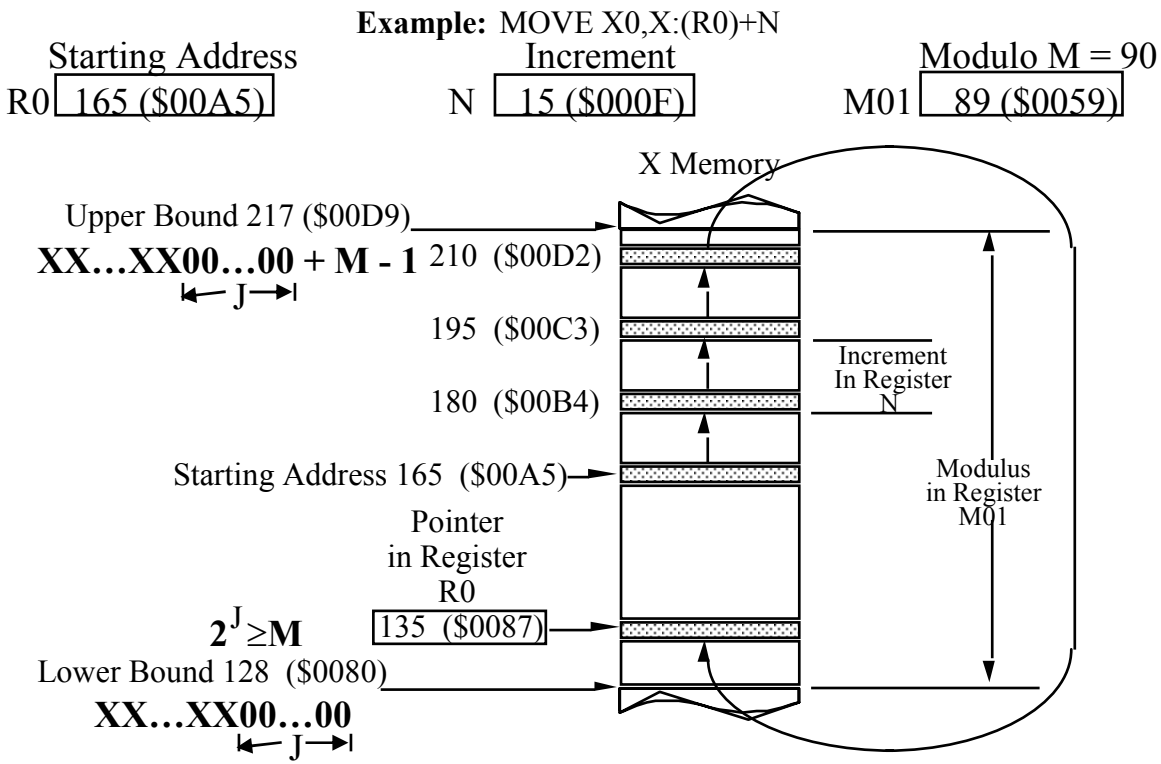
16-bit Modifier Register (M01) Contents	Address Calculation Arithmetic
0000 0000 0000 0000	Reserved
0000 0000 0000 0001	Modulo 2 (R0 only)
0000 0000 0000 0010	Modulo 3 (R0 only)
...	...
0011 1111 1111 1110	Modulo 16383 (R0 only)
0011 1111 1111 1111	modulo 16384 (R0 only)
0100 0000 0000 0000	Reserved
...	...
1000 0000 0000 0000	Reserved
1000 0000 0000 0001	Modulo 2 (R0 and R1)
1000 0000 0000 0010	Modulo 3 (R0 and R1)
...	...
1011 1111 1111 1110	Modulo 16383 (R0 and R1)
1011 1111 1111 1111	Modulo 16384 (R0 and R1)
1100 0000 0000 0000	Reserved
...	...
1111 1111 1111 1110	Reserved
1111 1111 1111 1111	Linear Arithmetic (R0 and R1)

Modulo Setup

MODULO M REGISTER SETUP

1. Modifier Register M01 = M -1, for R0 — MODULUS = M
   = M -1 + $8000, for R0 and R1
2. Lower Bound = XX…XX00…00 where $2^J \geq M$ — Beginning of Table
   $|\leftarrow J \rightarrow|$ — Minimize J
3. Upper Bound = XX…XX00…00 + M - 1 — End of Table
   $|\leftarrow J \rightarrow|$
4. Lower Bound $\leq$ Address Register Rn $\leq$ Upper Bound — Starting Point within Table
5. Offset Register N = Increment $\leq$ M — Desired Increment (if any)

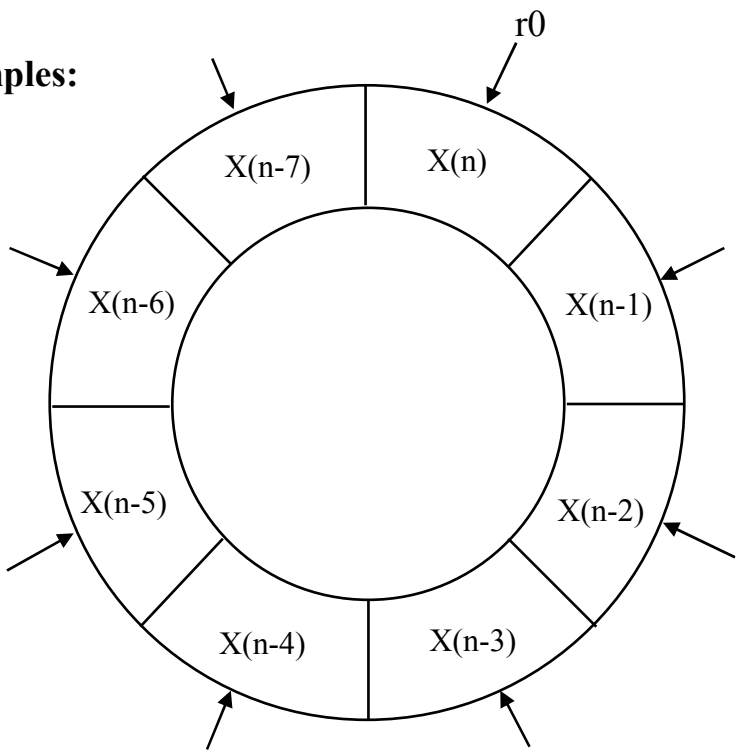
Modulo M Addressing Example

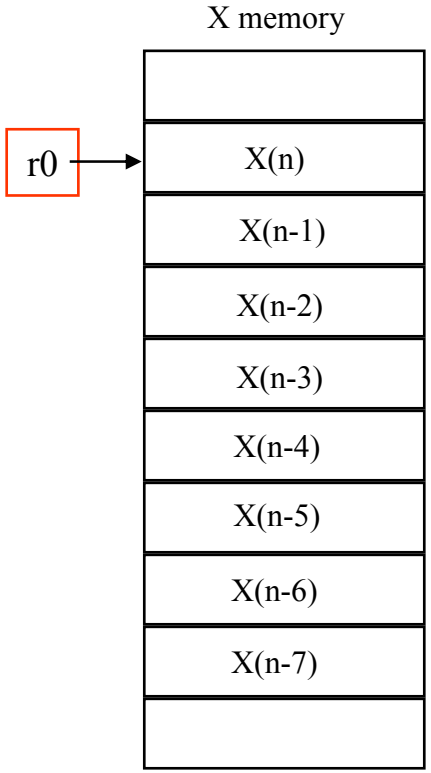


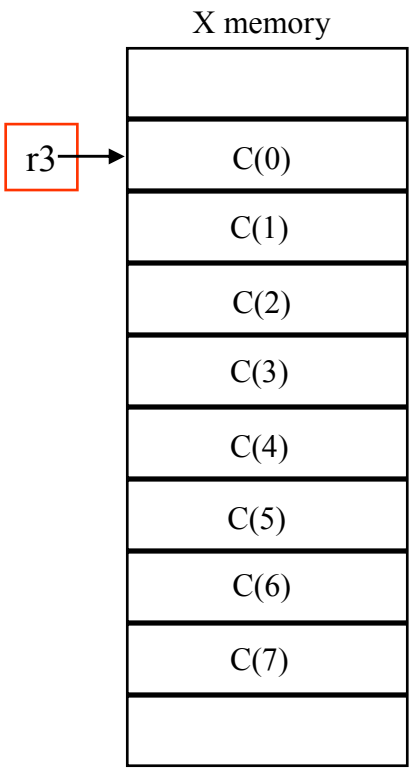
Accessing Coefficients & Samples

FIR equation: $Y(n) = \sum_{i=0}^{N-1} C(i) * X(n-i)$

A/D Samples:







Modulo Addressing Exercise

Write code to initialize R0 to point at samples,using modulo addressing, modulo 8, and R3 to point at coefficients using linear addressing. Also, move the first sample into register Y0 and the first coefficient into register X0, post incrementing both address registers. The 8 samples are stored in RAM beginning at X:$0080 and the 8 coefficients are stored in RAM beginning at X:$0000.

Write your program here:

```
npts    equ 8
        org X:$80
samples dsm npts
        org X:$00
coeff   dc .9,-.1,-.2,.5,.5,-.2,-.1,.9

        org P:$80
```

Suggested program steps:

Number of points in each buffer

Originate sample table at X:$80

Define storage for modulo 8 table

Originate coefficient table at X:$00

Define constant coefficients

Originate program at P:$80

1. Initialize r0 and r3

2. Initialize M01 register for Modulo 8

3. Move first sample and coefficient into Y0 and X0.

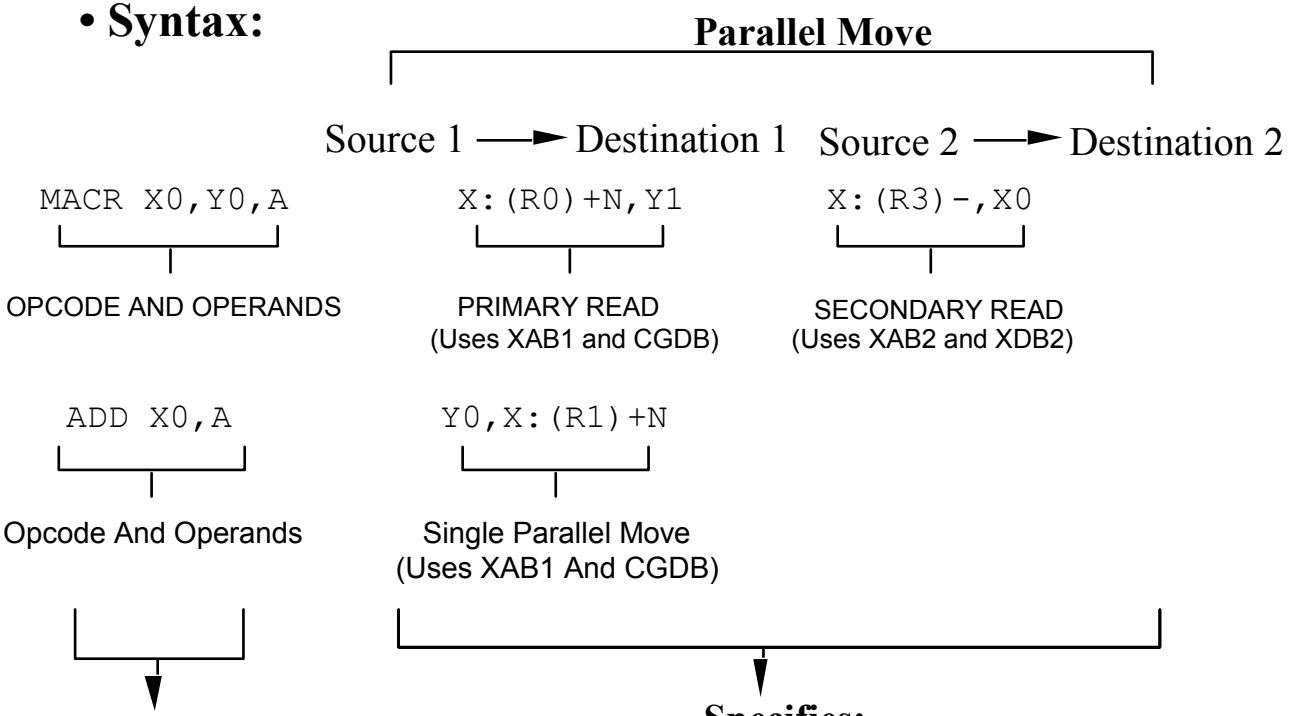
DSP56800 Instruction Set

- MPU like
- Makes pipeline invisible
- 6 groups
  - Move†
  - Arithmetic†
  - Logical
  - Bit Field Manipulation
  - Program Control
  - Loop

† Only arithmetic and Move instructions allow parallel data moves.

Parallel Data Move Instructions

- **Syntax:**



Specifies:

- Data ALU Operation
- Convergent or 2's Complement Rounding

Supports:

- Condition Code Bits 0 through 5

Specifies:

- Up to two optional data transfers
- Two different addressing modes
- Address space qualifiers [X:, XX:]

Supports:

- Limiting
- Sign extension and least significant zero fill
- Duplicate sources
- Duplicate destinations not allowed

Instruction Set Exercise

See DSP56800 Family Manual Pages 6-16 to 6-29

Data ALU Operation		First & Second Memory Reads		Destinations For Memory Reads	
Operation	**Operands**	**Read1**	**Read2**	**Dest1**	**Dest2**
MAC MPY MACR MPYR	X0,Y1,A X0,Y0,A Y1,Y0,A X0,Y1,B X0,Y0,B Y1,Y0,B	X:(R0)+ X:(R0)+N X:(R1)+ X:(R1)+N	X:(R3)+ X:(R3)-	Y0 Y1 This column lists the valid destination registers for the Read1 memory access.	X0 X0 This column lists the valid destination registers for the Read2 memory access.
ADD SUB	X0,A Y1,A Y0,A X0,B Y1,B Y0,B				
MOVE					

Based on the table above, find 6 reasons why the following instruction is not allowed:

ADD X0,Y1,A X:(R2)-,X0 X:(R3)+N,Y0

Move Instructions

Instruction	Description
LEA	Load effective address
POP	Pop a register from the software stack
MOVE	Move data
MOVE(C)	Move control register
MOVE(I)	Move immediate
MOVE(M)	Move program memory
MOVE(P)	Move peripheral data
MOVE(S)	Move absolute short

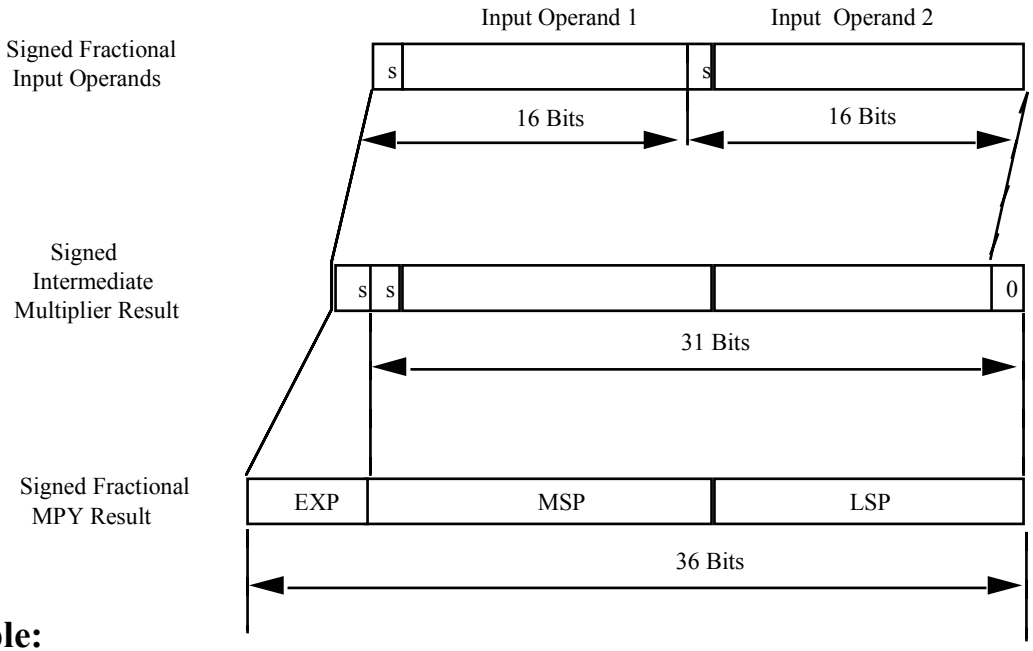
Arithmetic Instructions

Instruction	Description
ABS	Absolute value
ADC	Add long with carry[1]
ADD	Add
ASL	Arithmetic shift left (36-bit)
ASLL	Arithmetic multi-bit shift left[1]
ASR	Arithmetic shift right (36-bit)
ASRAC	Arithmetic multi-bit shift right with accumulate[1]
ASRR	Arithmetic multi-bit shift right[1]
CLR	Clear
CMP	Compare
DEC(W)	Decrement upper word of accumulator
DIV	Divide iteration[1]
IMPY(16)	Integer multiply[1]
INC(W)	Increment upper word of accumulator
MAC	Signed multiply-accumulate

Instruction	Description
MACR	Signed multiply-accumulate and round
MACSU	Signed/unsigned multiply-accumulate[1]
MPY	Signed multiply
MPYR	Signed multiply and round
MPYSU	Signed/unsigned multiply[1]
NEG	Negate
NORM	Normalize[1]
RND	Round
SBC	Subtract long with carry[1]
SUB	Subtract
Tcc	Transfer conditionally[1]
TFR	Transfer data ALU register to an accumulator
TST	Test a 36-bit accumulator
TST(W)	Test a 16-bit register or memory location[1].

Note: 1. These instructions do not allow parallel data moves

Fractional Multiplication



Example:

```
MPY     X0,Y1,A          ; multiply X0 by Y1
```

Before Execution

A2	A1	A0
0	1000	0000

X0	4000
Y1	F000

X0=$4000 (0.5)
Y1=$F000 (-0.125)

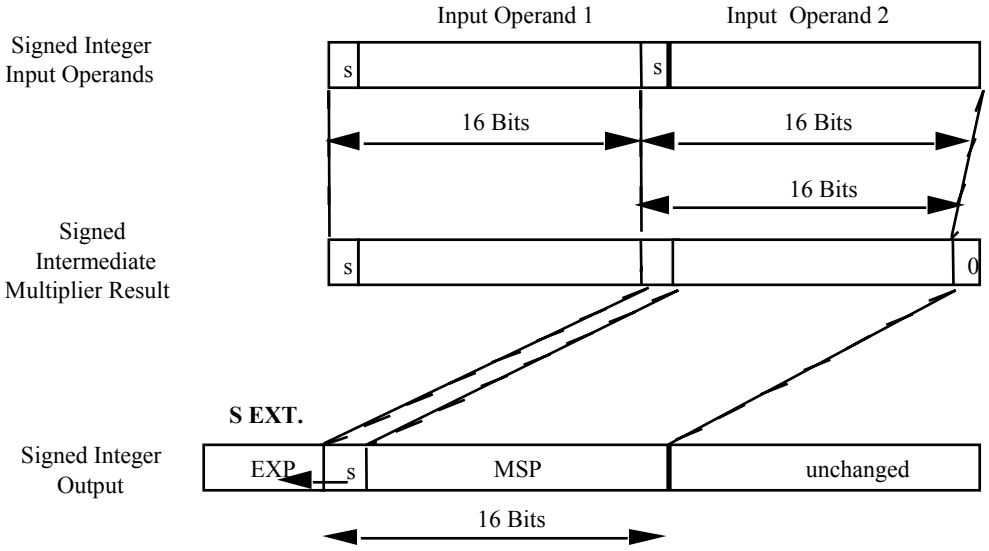
After Execution

A2	A1	A0
F	F800	0000

X0	4000
Y1	F000

A=$F:F800:0000 (-0.0625).

Integer Multiplication



Integer Arithmetic (IMPY)

Example:

```
IMPY    Y0,X0,A        ; form product
```

Before Execution

F	AAAA	789A
A2	A1	A0

X0	0003
Y0	0004

After Execution

0	000C	789A
A2	A1	A0

X0	0003
Y0	0004

- Result ($000C) is in A1
- A0 remains unchanged
- A2 is sign-extended.

Logical Instructions

Logical Instructions:

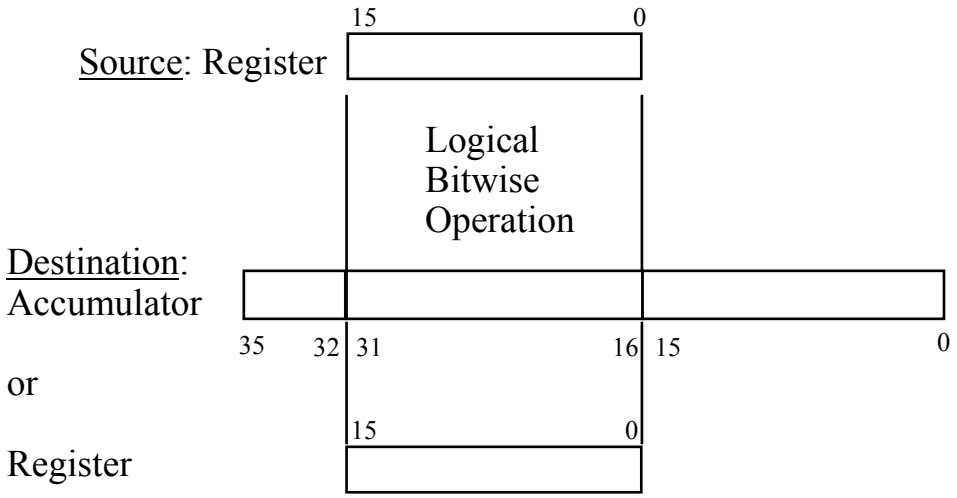
Instruction	Description
AND	Logical AND
EOR	Logical exclusive OR
LSL	Logical shift left
LSR	Logical shift right
NOT	Logical complement
OR	Logical inclusive OR
ROL	Rotate left
ROR	Rotate right

OPERATION	OPERANDS	COMMENTS
AND **EOR** **OR**	X0,F Y1,F Y0,F F1,X0 F1,Y1 F1,Y0 Y0,X0 Y1,X0 X0,Y0 Y1,Y0 X0,Y1 Y0,Y1	F = A or B F1 = A1 or B1

OPERATION	OPERAND
LSL **LSR** **NOT** **ROL** **ROR**	A B X0 Y1 Y0

Logical Instruction Examples (1 of 2)

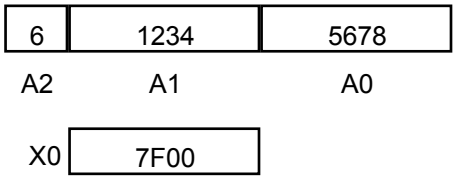
AND, EOR, OR:
INSTR S,D (no parallel move) If D is an accumulator, bits 16-31 are affected.



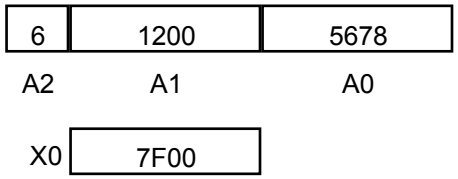
Example:

```
AND     X0,A        ; AND X0 with A1
```

Before Execution

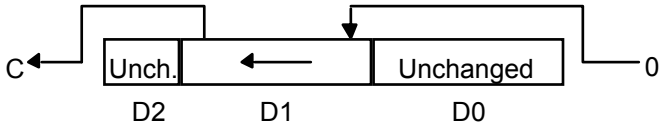


After Execution



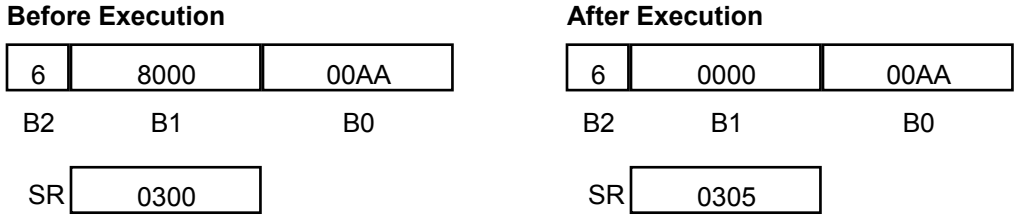
Logical Instruction Examples (2 of 2)

LSL, LSR, NOT, ROL, ROR:
INSTR D (no parallel move) If D is an accumulator, bits 16-31 are affected.



Example:

```
LSL     B       ; multiply B1 by 2 (B1 considered unsigned)
```



Multi-bit Shifting Instructions

Operation	Operands	Comments
LSRR ASRR ASLL	Y1,X0,F Y0,X0,F Y1,Y0,F Y0,Y0,F A1,Y0,F B1,Y1,F Y1,X0,DD Y0,X0,DD Y1,Y0,DD Y0,Y0,DD A1,Y0,DD B1,Y1,DD	Multi-bit Logical & Arithmetic Shifting First register is value to be shifted, second register is the shift amount (uses 4 LSBs)
LSLL	Y1,X0,DD Y0,X0,DD Y1,Y0,DD Y0,Y0,DD A1,Y0,DD B1,Y1,DD	Multi-bit Logical Left Shift First register is the value to be shifted, second register is the shift amount (uses 4 LSBs)
ASRAC LSRAC	Y1,X0,F Y0,X0,F Y1,Y0,F Y0,Y0,F A1,Y0,F B1,Y1,F	Multi-bit Arithmetic Shifting with Accumulation First register is the value to be shifted, second register is the shift amount (uses 4 LSBs)

Multi-bit Shifting Instruction Examples (1 of 3)

Example:

```
ASRR    Y1,X0,A    ; right shift of 16-bit Y1 by X0
```

Before Execution

A2	A1	A0
0	1234	5678

Y1	AAAA
X0	0004

After Execution

A2	A1	A0
F	FAAA	0000

Y1	AAAA
X0	0004

Note: Y1 is arithmetically shifted right 4 bits, the result placed in A1. Then A2 is sign extended and A0 is zero filled.

Multi-bit Shifting Instruction Examples (2 of 3)

Example:

```
LSRR    Y1,X0,A    ; right shift of 16-bit Y1 by X0
```

Before Execution

0	3456	3456
A2	A1	A0

Y1	AAAA

X0	0004

After Execution

0	0AAA	0000
A2	A1	A0

Y1	AAAA

X0	0004

Note: Y1 is logically shifted right 4 bits, the result placed in A1. Then A2 is sign extended and A0 is zero filled.

Multi-bit Shifting Instruction Examples (3 of 3)

Example:

```
ASRAC    Y1,X0,A    ; add right shifted Y1 to A
```

Before Execution

A2	A1	A0
0	0000	0099

Y1	C003
X0	0004

After Execution

A2	A1	A0
F	FC00	3099

Y1	C003
X0	0004

Note: Y1 is arithmetically shifted right 4 bits, the result added to A. Then A2 is sign extended.

Loop Instructions (DO)

Operation upon Executing DO Instruction:

HWS[0] → HWS[1]; #xx → LC
PC → HWS[0]; LF → NL; expr-1 → LA
1→ LF

Assembler Syntax:

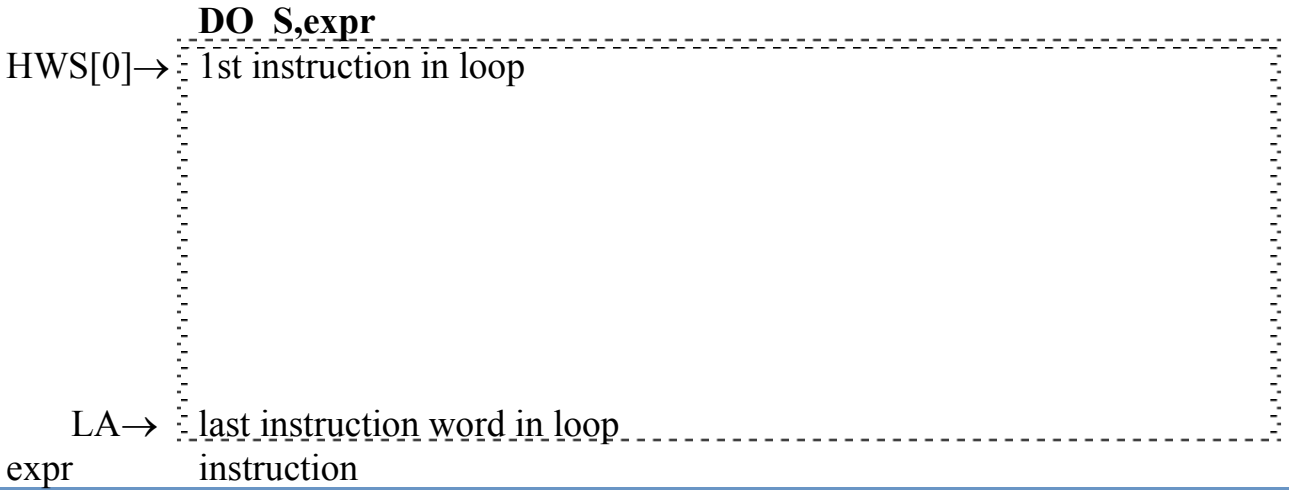
DO #xx,expr
#xx = 6 bit unsigned value

HWS[0] → HWS[1]; S → LC
PC → HWS[0]; LF → NL; expr-1 → LA
1→ LF

DO S,expr
S = any register except M01, SR, OMR, HWS (13 LSBs)

Operation when Loop Completes (End-of-loop Processing):

NL → LF
HWS[1] → HWS[0]; 0 → NL



Loop Instructions (ENDDO)

ENDDO:

Operation:	Assembler Syntax:
NL → LF	ENDDO
HWS[1] → HWS[0]; 0 → NL	

Example:

```
        DO      Y0,ENDLP    ; execute loop ending at ENDLP (Y0) times
        :
        MOVEC   LC,A        ; get current value of loop counter (LC)
        CMP     Y1,A        ; compare loop counter with value in Y1
        JNE     ONWARD      ; go to ONWARD if LC not equal to Y1
        ENDDO               ; LC equal to Y1, restore all DO registers
        JMP     ENDLP       ; go to ENDLP
ONWARD  :                   ; LC not equal to Y1, continue DO loop
        :                   ; (last instruction in DO loop)

ENDLP   MOVE    #$1234,X0   ; (first instruction AFTER DO loop)
```

Loop Instructions (REP)

REP:

Operation:	**Assembler Syntax:**
LC → TEMP; #xx → LC Repeat next instruction until LC = 1 TEMP → LC	REP #xx #xx = 6 bit unsigned value
LC → TEMP; S → LC Repeat next instruction until LC = 1 TEMP → LC	REP S S = any register except M01, SR, OMR, HWS (13 LSBs)

Notes:

- The REP instruction and the REP loop may not be interrupted once in progress until completion of the REP loop.
- A REP instruction can be used inside a DO loop

Normalization

Why Normalize:

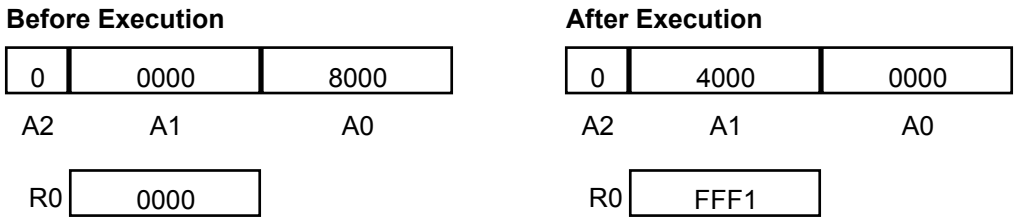
- Puts the value in the maximum resolution position
- Allows for growth while maintaining precision in an accumulator

Operation:				**Assembler Syntax:**	
If	(E • U • Z = 1)	then	ASL D and R0 - 1 → R0	NORM	R0,D
else if	(E = 1)	then	ASR D and R0 + 1→ R0	(D=A or B)	
else	NOP				

- Perform one normalization iteration on the destination operand (D)
- update the address register R0 based upon the results of that iteration
- This is a 36-bit operation.
- Since the operation of the NORM instruction depends on the E, U, and Z bits, these bits must correctly reflect the current state of the destination accumulator prior to executing the NORM instruction.

Example:

```
TST     A
REP     #31     ;maximum number of iterations (31) needed
NORM    R0,A    ;perform one normalization iteration
```



Program Control Instructions

Program Control Instructions:

Instruction	Description
Bcc	Branch conditionally
BRA	Branch Always
DEBUG	Enter debug mode
Jcc	Jump conditionally
JMP	Jump
JSR	Jump to subroutine
NOP	No operation
RTI	Return from interrupt
RTS	Return from subroutine
STOP	Stop processing (lowest power stand-by)
SWI	Software interrupt
WAIT	Wait for interrupt (low power stand-by)

Program Control Operation

Operation:	Assembler Syntax:
SP+1 → SP PC → X:(SP) SP+1 → SP SR → X:(SP) xxxx → PC	JSR xxxx

Operation:	Assembler Syntax:
X:(SP) → SR; SP-1→ SP X:(SP) → PC; SP-1→ SP	RTI

Operation:	Assembler Syntax:
SP-1→ SP X:(SP) → PC; SP-1→ SP	RTS

Operation:	Assembler Syntax:	
PC+displacement → PC	BRA	aa (aa= 7-bit signed value that is sign-extended to form the PC-relative offset.)

Example:

```
BRA     LABEL
INCW    A
INCW    A
LABEL
ADD     B,A
```

In this example, program execution skips the two INCW instructions and continues with the ADD instruction. The BRA instruction uses a PC-relative offset of +2 for this example.

Bit Manipulation Instructions

Instruction	Description
ANDC	Logical AND with immediate data
BFCLR	Bit-field test and clear
BFSET	Bit-field test and set
BFCHG	Bit-field test and change
BFTSTL	Bit-field test low
BFTSTH	Bit-field test high
BRSET	Branch if selected bits are set
BRCLR	Branch if selected bits are clear
EORC	Logical exclusive OR with immediate data
NOTC	Logical complement on memory location and registers
ORC	Logical inclusive OR with immediate data

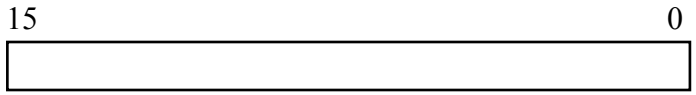
Notes:

ANDC equals and disassembles as BFCLR (with the mask inverted).
ORC equals and disassembles as BFSET (with the same mask).
EORC equals and disassembles as BFCHG (with the same mask).

Bit Manipulation Instructions Example

BFTSTL	Bit Field Test Low
BFTSTH	Bit Field Test High
BFCLR	Bit Field test and Clear
BFSET	Bit Field test and Set
BFCHG	Bit Field test and Change

Operand:
X:xxxx
X:I/O Short
X:Abs Short
X:(R2+xx)
X:(SP-xx)
Any register except HWS



- Test (and modify) up to 16 bits of the register or X:memory location. If all selected bits are set, C is set, except BFTSTL (If all bits are clear, C is set).
- i.e.: **BFCHG #$0310,X:<<$FFEC ;** tests and changes bits 4, 8, 9 in I/O Port B data register.

Example: Programming an FIR Filter

$$y(n) = \sum_{i=0}^{100-1} c(i) * x(n-i)$$

Expanded Equation:

y(n) = c(0) * x(n-0) + c(1) * x(n-1) + ... + c(99) * x(n-99)

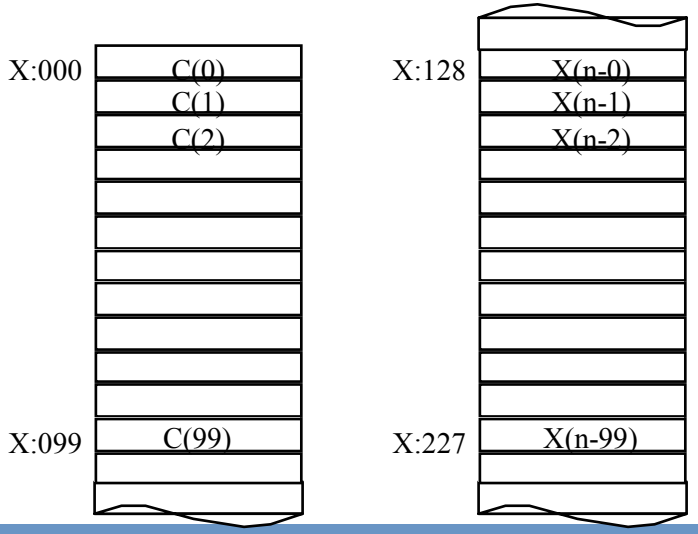
What do we need to calculate this?

- 100 storage locations for 100 coefficients (c(i)) ===> X memory
- 100 storage locations for 100 data samples (x(i)) ===> X memory
- Multiply Instruction
- Addition Instruction
- Move Instructions

MPY

ADD

MOVE



Example: Programming an FIR Filter

(continued)

```
        clr     a           ; Clear Accumulator
        move    #0,r3       ; Set up pointer to coeffs
        move    #128,r0     ; Set up pointer to data

        move    X:(r0)+,y0          X:(r3)+,x0

        do #100,label
        mac x0,y0,a         X:(r0)+,y0          X:(r3)+,x0
label
```

What's wrong with this code that will cause an assembler error?

Modulo Addressing Exercise Answer

Write code to initialize R0 to point at samples,using modulo addressing, modulo 8, and R3 to point at coefficients using linear addressing. Also, move the first sample into register Y0 and the first coefficient into register X0, post incrementing both address registers. The 8 samples are stored in RAM beginning at X:$0080 and the 8 coefficients are stored in RAM beginning at X:$0000.

Write your program here:

```
npts    equ 8
        org X:$80
samples dsm npts
        org X:$00
coeff   dc .9,-.1,-.2,.5,.5,-.2,-.1,.9

        org P:$80

        move    #samples,r0
        move    #coeff,r3

        move    #npts-1,m01

        nop

        move    x:(r0)+,y0      x:(r3)+,x0
```

Suggested program steps:

Number of points in each buffer

Originate sample table at X:$80

Define storage for modulo 8 table

Originate coefficient table at X:$00

Define constant coefficients

Originate program at P:$80

1. Initialize r0 and r3

2. Initialize M01 register for Modulo 8

3. Move first sample and coefficient into Y0 and X0.

Instruction Set Exercise Answer

Based on the table above, find 6 reasons why the following instruction is not allowed:

```
ADD    X0,Y1,A     X:(R2)-,X0    X:(R3)+N,Y0
```

1. The only operands accepted for ADD or SUB is X0,F, X1,F, and Y0,F, where F is either the A or B accumulator register. Thus, X0,Y1,A is an invalid entry.

2. The pointer R2 is not allowed for Read1.

3. The post-decrement addressing mode is not available for Read1.

4. The X0 register may not be a destination for Read1 because it is not listed in the Dest1 column.

5. The post-update by N addressing mode is not allowed for Read2.

6. The Y0 register may not be a destination for Read2 because it is not listed in the Dest2 column.

Example Answer
Programming an FIR Filter

```
                clr     a               ; Clear Accumulator
                move    #0,r3           ; Set up pointer to coeffs
                move    #128,r0         ; Set up pointer to data

                move    X:(r0)+,y0      X:(r3)+,x0

                do #100,label
                mac x0,y0,a             X:(r0)+,y0      X:(r3)+,x0
label
```

What's wrong with this code?

Answer:
Instruction 4 is an error because it uses r0, which was just initialized in instruction 3
solution 1:
Insert a NOP after instruction 3
solution2: (better)
Move instruction 1 after instruction 3

Processing States

- √ Reset: The DSP core is forced into a known reset states
- √ Normal: The state of The DSP core where instruction are normally executed.
- √ Exception: DSP core transfers program control from its current location to an Interrupt Service Routine (ISR) using the interrupt vector table.
- √ Wait: A low-power state where the DSP core is shut down but the peripherals and interrupt machine remain active.
- √ Stop: A lowest power consumption state where the DSP core, interrupt machine, and most of the peripherals are shut down.
- √ Debug: The state where the DSP core is halted and all registers in OnCE port of processor are accessible for program debug.

Reset Processing State

Reset Sources:

1) The external RESET pin is asserted (A low level is applied).
2) The Computer Operating Properly (COP) timer reaches ZERO.

Reset Process:

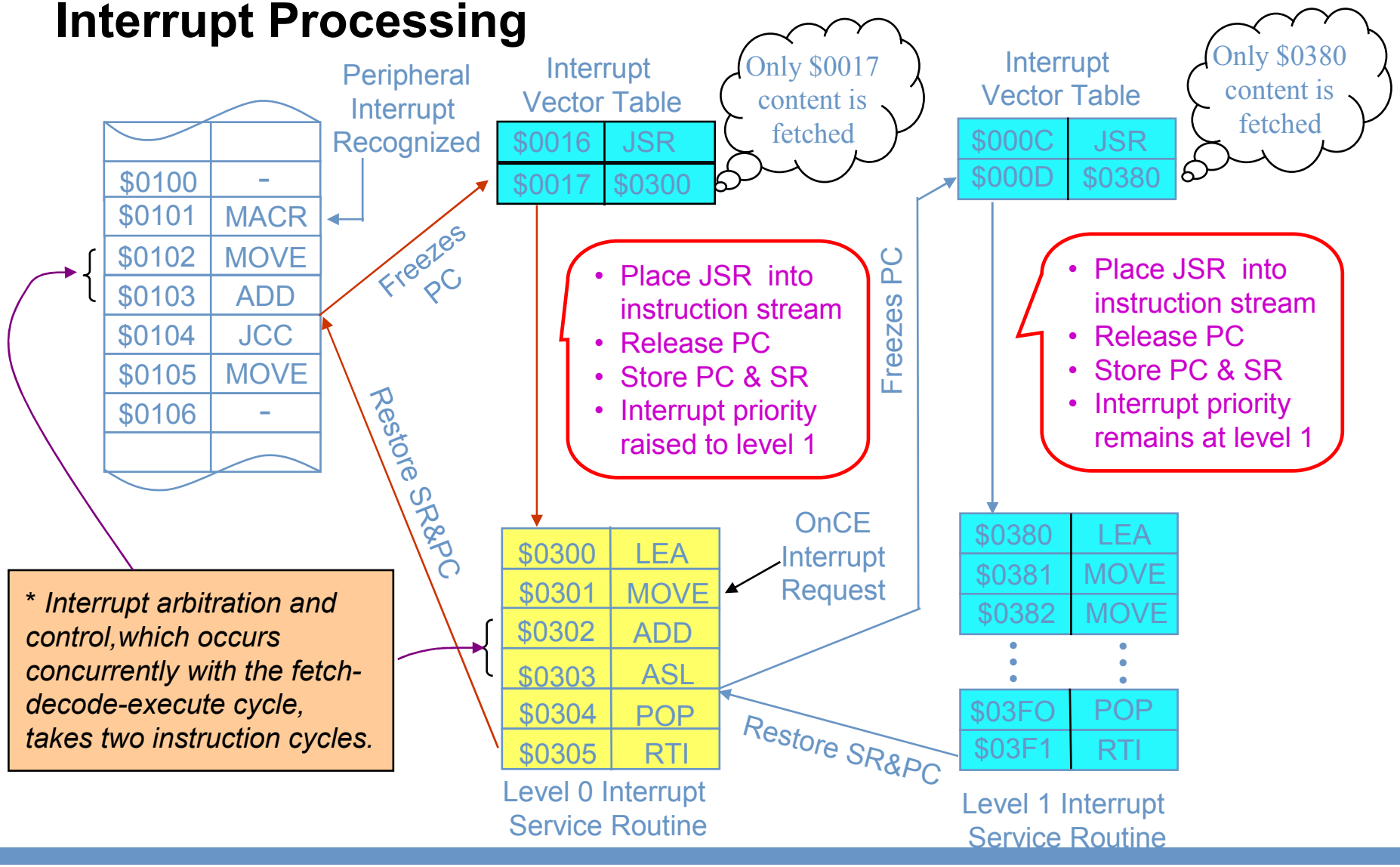
Reset Asserted

1) Reset internal peripheral devices.
2) Set the M01 modifier register to $FFFF.
3) Clears the interrupt priority register (IPR).
4) Sets the wait state fields in the bus control register (BCR) to their maximum value.
5) Clears SR's Loop Flag bit(LF) and condition code bits (SZ, L, E, U, N, Z, V, C,) and sets the interrupts mask bits (I1, I0) where masked level 0 interrupt.
6) Clears OMR's Nested Looping bit (NL); Condition Code bit (CC); Stop Delay bit (SD); Rounding bit (R); and External X Memory bit (EX).

Reset Deasserted

1) The Chip Operating Mode Bit (MA, MB) is loaded from external source.
2) A delay of 16 instruction cycles occurs to sync the local clock generator and state machine.
3) Fetch JMP instruction from appropriate reset vector table.

Interrupt Processing



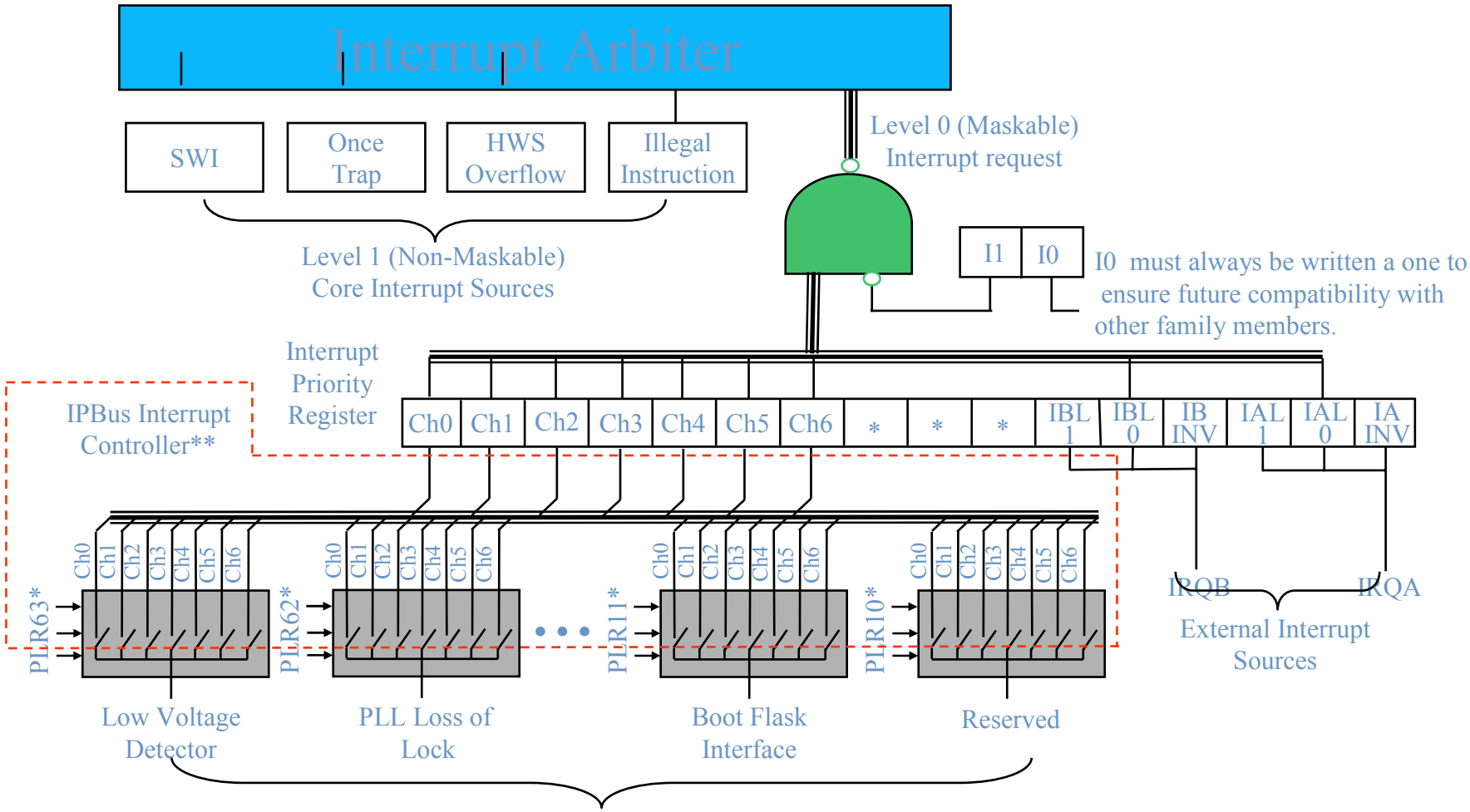
Exception Processing State

The exception processing state is the state where the DSP core recognizes and processes interrupts that can be generated by conditions inside the DSP or from external sources.

The Sequence of Events in the Exception Processing State

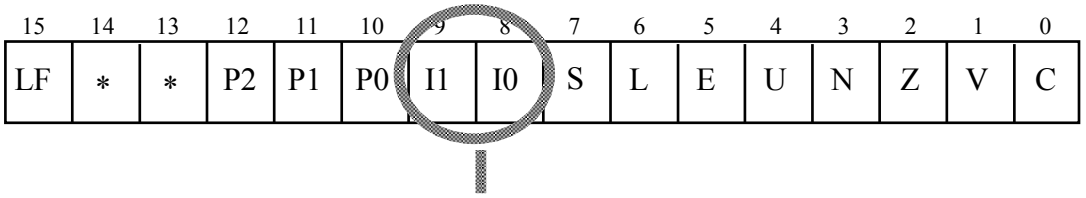
1) An interrupt request is generated either from external or from core or peripherals.
2) The request is either latched in an interrupt-pending flag or remain asserted .
3)The interrupt controller selects the highest interrupt priority to be processed
4)Interrupt controller then freezes the program counter (PC) and fetches the second word, which includes address information, located at the two interrupt vector addresses associated with the selected interrupt.
Note: It is required that the instruction located at the interrupt vector address must be a two-word JSR instruction.
5)The interrupt controller adds JSR opcode, and places this JSR instruction into instruction stream, and then releases the PC.
6) The execution of The JSR instruction stacks the PC and SR and jumps to the first instruction in the interrupt service routine. In addition, The core interrupt priority level is raised to level 1 to mask any level 0 interrupts.

Interrupt Arbitration



*If PLRxx is set to zero, that interrupt is disabled. If PLRxx is set to 1, that interrupt is assigned to Ch0
If PLRxx is set to 2, that interrupt is assigned to Ch1, and so on.
**DSP56824's peripheral interrupts are directly assigned to specific channel of Interrupt Priority Register.

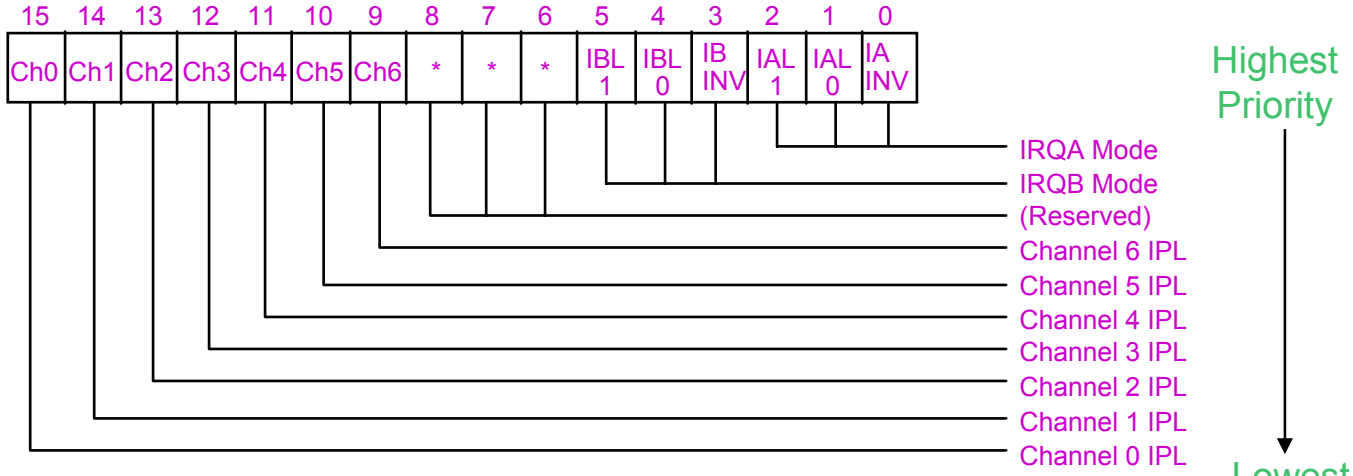
Interrupt Priority Structure



Status Register

Priority	I1	I0	Exceptions Permitted	Exceptions Masked
Low	0	0	Reserved	Reserved
↓	0	1	IPL 0,1	None
	1	0	Reserved	Reserved
High	1	1	IPL 1	IPL 0

- Set to Highest Level on Hardware or Software RESET
- Each interrupt source has a fix vector number, regardless of the level it has been assigned.
- The interrupt source with the highest vector has highest priority within a given level.



* Indicates reserved bits, read as 0 and written with 0 for future compatibility

IBL1 IAL1	IBINV IAINV	Trigger Mode
0	0	Low-level sensitive
0	1	High-level sensitive
1	0	Falling edge sensitive
1	1	Rising edge sensitive

IBL0 IAL0	Enabled?	IPL
0	No	–
1	Yes	0

Chx	Enabled?	IPL
0	No	–
1	Yes	0

Interrupt Vector Table

Interrupt Starting Address	Interrupt Priority level	Interrupt Source	Content	
Level 1 (Non-maskable Interrupt)				
$0000	-	Hardware Reset	JMP	$xxxx
$0002	-	COP Watchdog Reset	JMP	$xxxx
$0004	-	(Reserve)	-	
$0006	1	Illegal Instruction Trap	JSR	$xxxx
$0008	1	SWI	JSR	$xxxx
$000A	1	Hardware Stack Overflow	JSR	$xxxx
$000C	1	OnCe Trap	-	
$000E	1	(Reserve)	JSR	$xxxx
Level 0 (Maskable Interrupt)				
$0010	0	IRQA	JSR	$xxxx
$0012	0	IRQB	JSR	$xxxx
$0014	0	Vector Available for On-Chip Peripherals	JSR	$xxxx
$0016	0	Vector Available for On-Chip Peripherals	JSR	$xxxx
$0018	0	Vector Available for On-Chip Peripherals	JSR	$xxxx
$001A	0	Vector Available for On-Chip Peripherals	JSR	$xxxx
$001C	0	Vector Available for On-Chip Peripherals	JSR	$xxxx
$001E	0	Vector Available for On-Chip Peripherals	JSR	$xxxx
$0020	0	Vector Available for On-Chip Peripherals	JSR	$xxxx
•••	•••	•••	•••	
$007C	0	Vector Available for On-Chip Peripherals	JSR	$xxxx
$007E	0	Vector Available for On-Chip Peripherals	JSR	$xxxx

- Each entry (except RESETS) must be a two word JSR instruction. The Interrupt Controller fetches only the address of the jump. RESETS use JMP

Power Saving State

Wait Processing state

The WAIT instruction brings the processor into the wait processing state. In the wait state the internal clock is disabled from all internal circuitry except the internal peripherals. All internal processing is halted until an unmasked interrupt occurs or until the DSP reset.

Stop Processing State

The STOP instruction brings the processor into the stop processing state, which is lowest power consumption state. In the stop state the clock oscillator is gated off, whereas in the wait the clock oscillator remains active. The on-chip peripherals are held in their respective individual reset states while the processor is in the stop state.

Three actions will bring DSP back to Normal State:

1) A low level is applied to the IRQA pin.
2) A low level is applied to the RESET pin.
3) An on-chip timer reaches zero (56824's timer2).

Normal Processing State

The normal processing state is the typical state of the processor where it executes instructions in a three-stage pipeline. The three-stage pipeline allows most instruction to execute at a rate of one instruction per instruction cycle.

Debug Processing State

The debug processing state is a state where the DSP core is halted and under the control of the OnCE debug port. Serial date is shifted in and out of this port,and it is possible to execute single instructions from this processing state.

The End

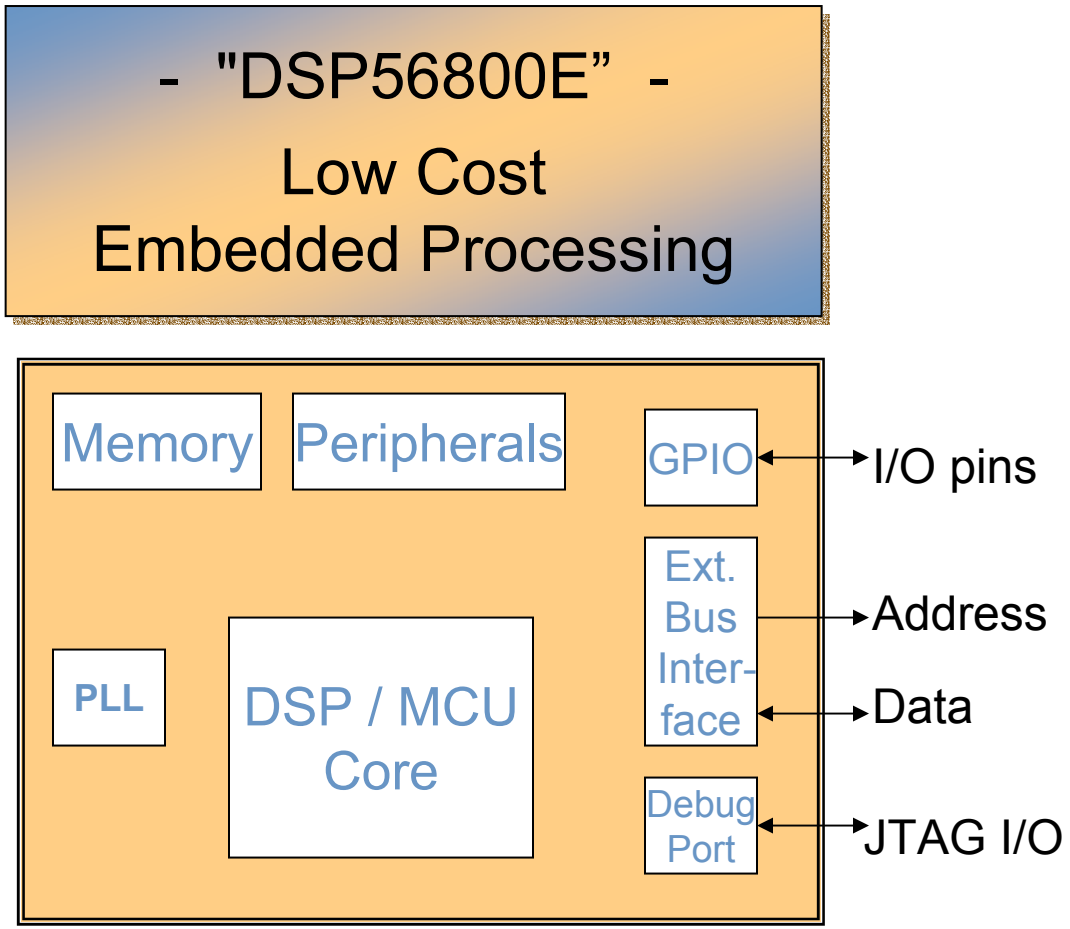
Embedded Connectivity Summit 2004

Launched by Motorola
freescale™
semiconductor

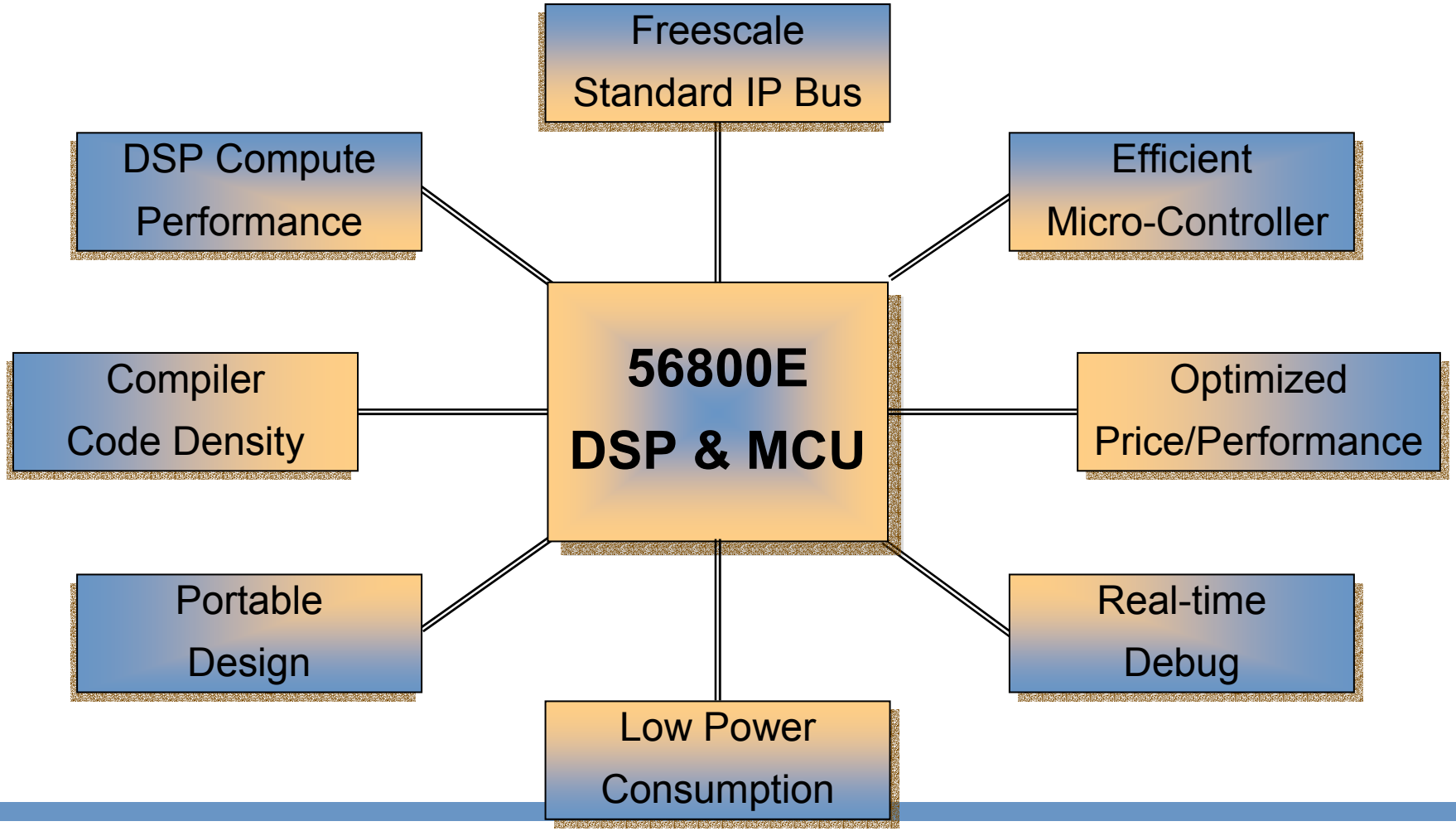
DSP56800E ARCHITECTURE

Launched by Motorola
freescale™
semiconductor

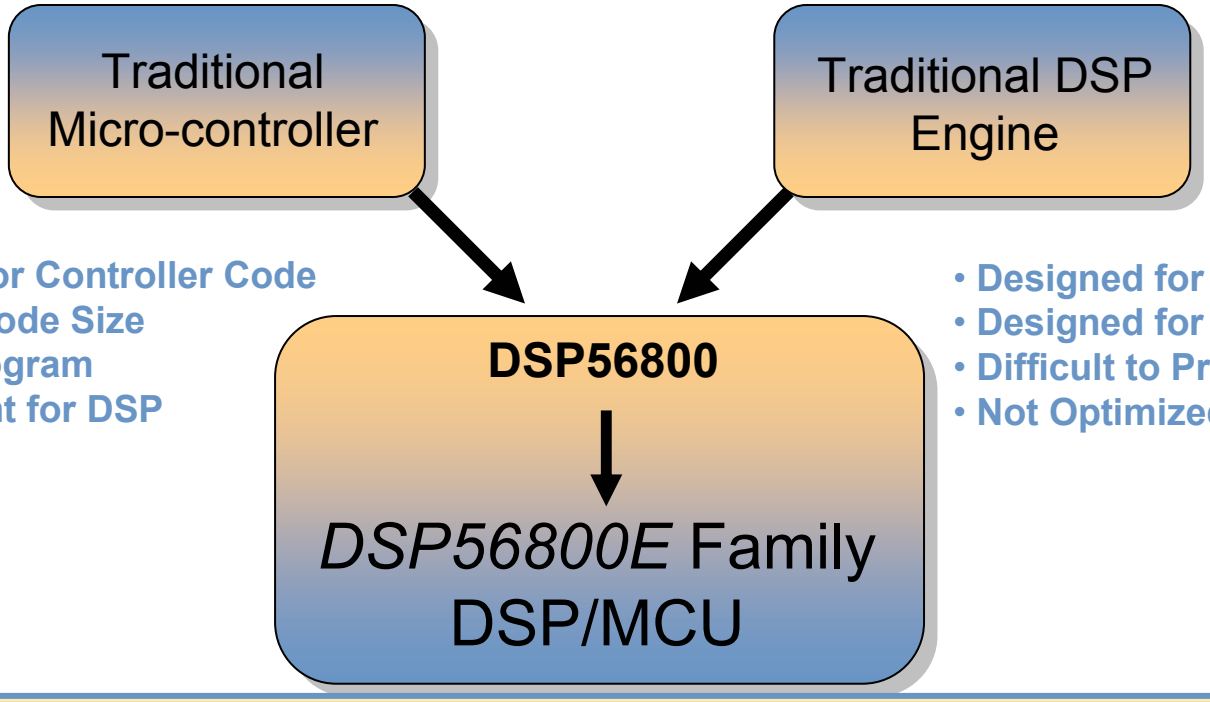
- “DSP56800E” -
Powerful, Low Cost DSP/MCU Engine



Bringing it All Together
The 56800E Family Foundation



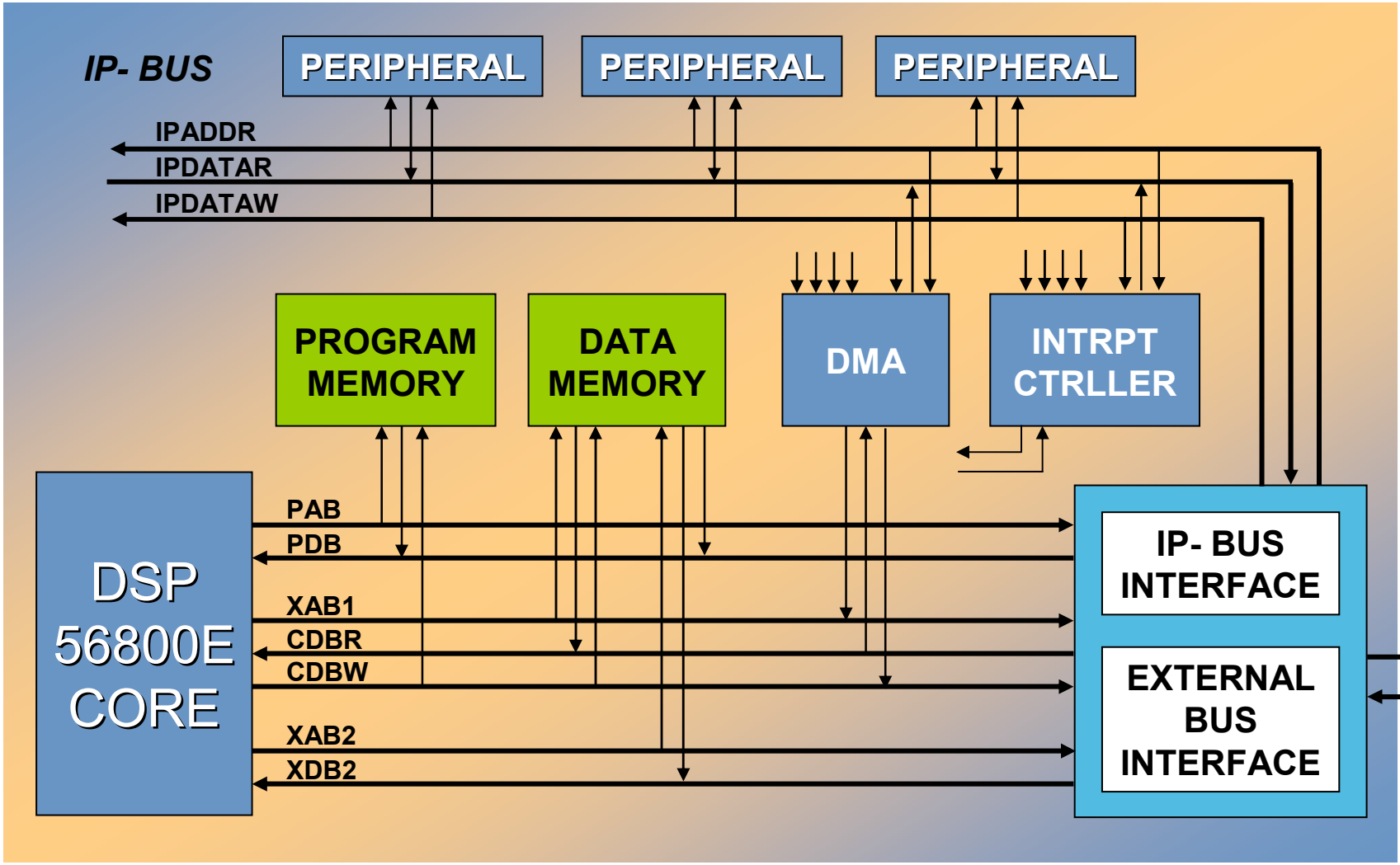
Technology Shift: Combined DSP and Controller Functionality



- Instructions Optimized for Controller Code, DSP, Matrix Operations
- Compact Assembly & "C" Compiled Code Size
- Easy to Program
- Adequate MIPS Headroom and Extended Addressing Space

Launched by Motorola freescale semiconductor

DSP56800E System Architecture



Hawk V2 Architectural Distinctives

High Level Abstraction of Application Software

Full Set of Data Types

High Code Density for Minimized Solution Cost

Large Address Spaces

Full Source Code Compatibility

Powerful Register Set

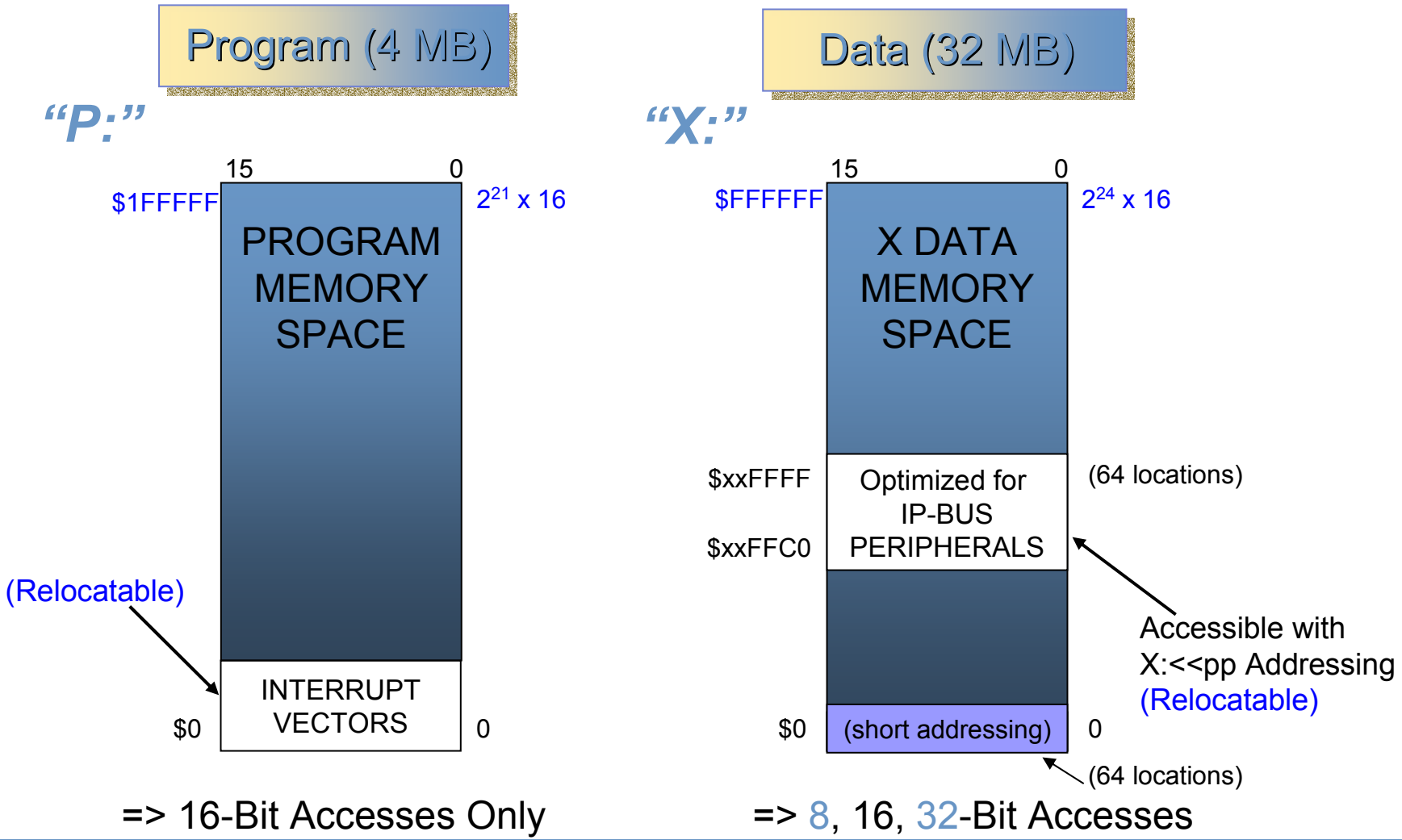
Improved Multitasking Support

Optimized Power Management

Efficient Peripheral Interfacing through Freescale's IP- BUS

Efficient Memory Interfacing

Program and Data Memories



DSP56800E General Purpose Registers

DATA ARITHMETIC LOGIC UNIT

DATA REGISTERS

	35 32	31 16	15 0
A	A2	A1	A0
B	B2	B1	B0
C	C2	C1	C0
D	D2	D1	D0

Y	Y1
	Y0
	X0

=> **7 Data Registers**

PROGRAM CONTROL UNIT

20 0
PC

PROGRAM COUNTER

15 0
OMR
SR

OPERATING MODE and STATUS

ADDRESS GENERATION UNIT

23 0
R0
R1
R2
R3
R4
R5
N
SP

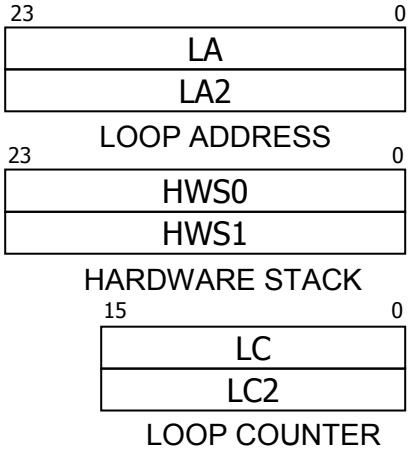
POINTER REGISTERS

=> **8 Address Registers**

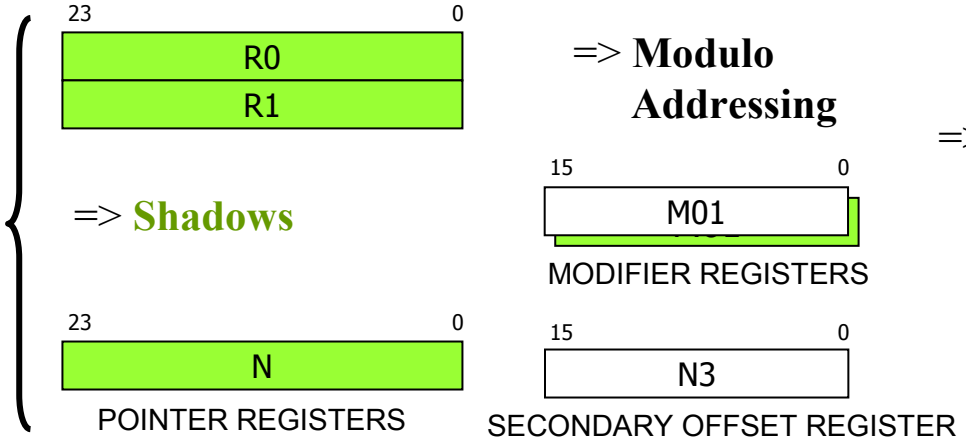
Registers with Dedicated Functionality

PROGRAM CONTROL UNIT

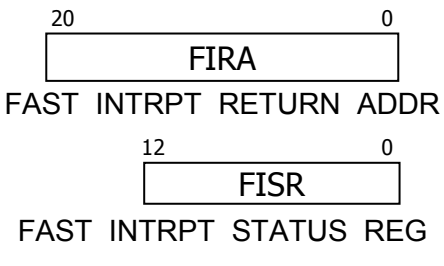
=> HW Looping
Nested 2 Deep



ADDRESS GENERATION UNIT



=> Fast
Interrupt



DSP56800E Programming Model

DATA ARITHMETIC LOGIC UNIT

DATA REGISTERS

	35 32	31 16	15 0
A	A2	A1	A0
B	B2	B1	B0
C	C2	C1	C0
D	D2	D1	D0

Y	Y1
	Y0
	X0

ADDRESS GENERATION UNIT

23 0

R0
R1
R2
R3
R4
R5

N

SP

POINTER REGISTERS

==> R0, R1, N, and M01 registers are shadowed

15 0

M01

MODIFIER REGISTERS

15 0

N3

SECONDARY OFFSET REGISTER

PROGRAM CONTROL UNIT

20 0

PC

PROGRAM COUNTER

15 0

OMR
SR

OPERATING MODE and STATUS

23 0

LA
LA2

LOOP ADDRESS

23 0

HWS0
HWS1

HARDWARE STACK

15 0

LC
LC2

LOOP COUNTER

20 0

FIRA

FAST INTERRUPT RETURN ADDRESS

12 0

FISR

FAST INTERRUPT STATUS REGISTER

Data Movement on the DSP56800E

Support of Compiler Data Types

- Longs (32-bits)
- Words (16-bits)
- Bytes (8-bits)

Parameters:

- 8, 16, or 32-bits
- Read or Write Operation
- Signed or Unsigned
- Addressing Mode

==> See the Instruction Set Summary for a Complete Set of Move Tables

Programming Model Comparison: ’800 vs ’800E

New for 800E

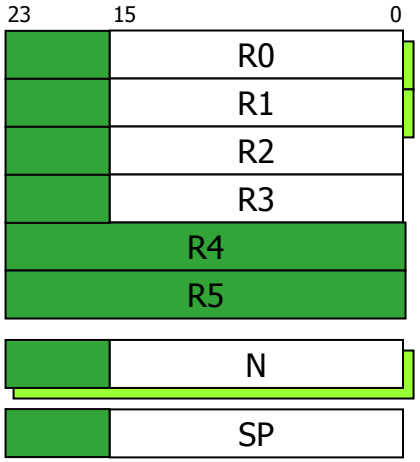
DATA ARITHMETIC LOGIC UNIT

DATA REGISTERS

	35 32	31 16	15 0
A	A2	A1	A0
B	B2	B1	B0
C	C2	C1	C0
D	D2	D1	D0

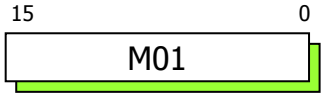
Y	Y1
	Y0
	X0

ADDRESS GENERATION UNIT

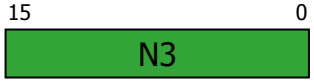


POINTER REGISTERS

==> R0, R1, N, and M01 registers are shadowed

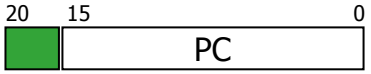


MODIFIER REGISTERS

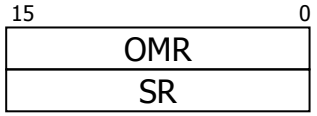


SECONDARY OFFSET REGISTER

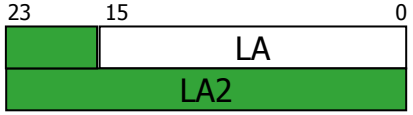
PROGRAM CONTROL UNIT



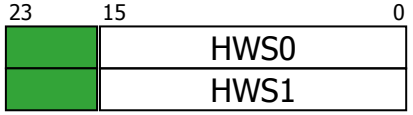
PROGRAM COUNTER



OPERATING MODE and STATUS



LOOP ADDRESS



HARDWARE STACK

LOOP COUNTER

FAST INTERRUPT RETURN ADDRESS

FAST INTERRUPT STATUS REGISTER

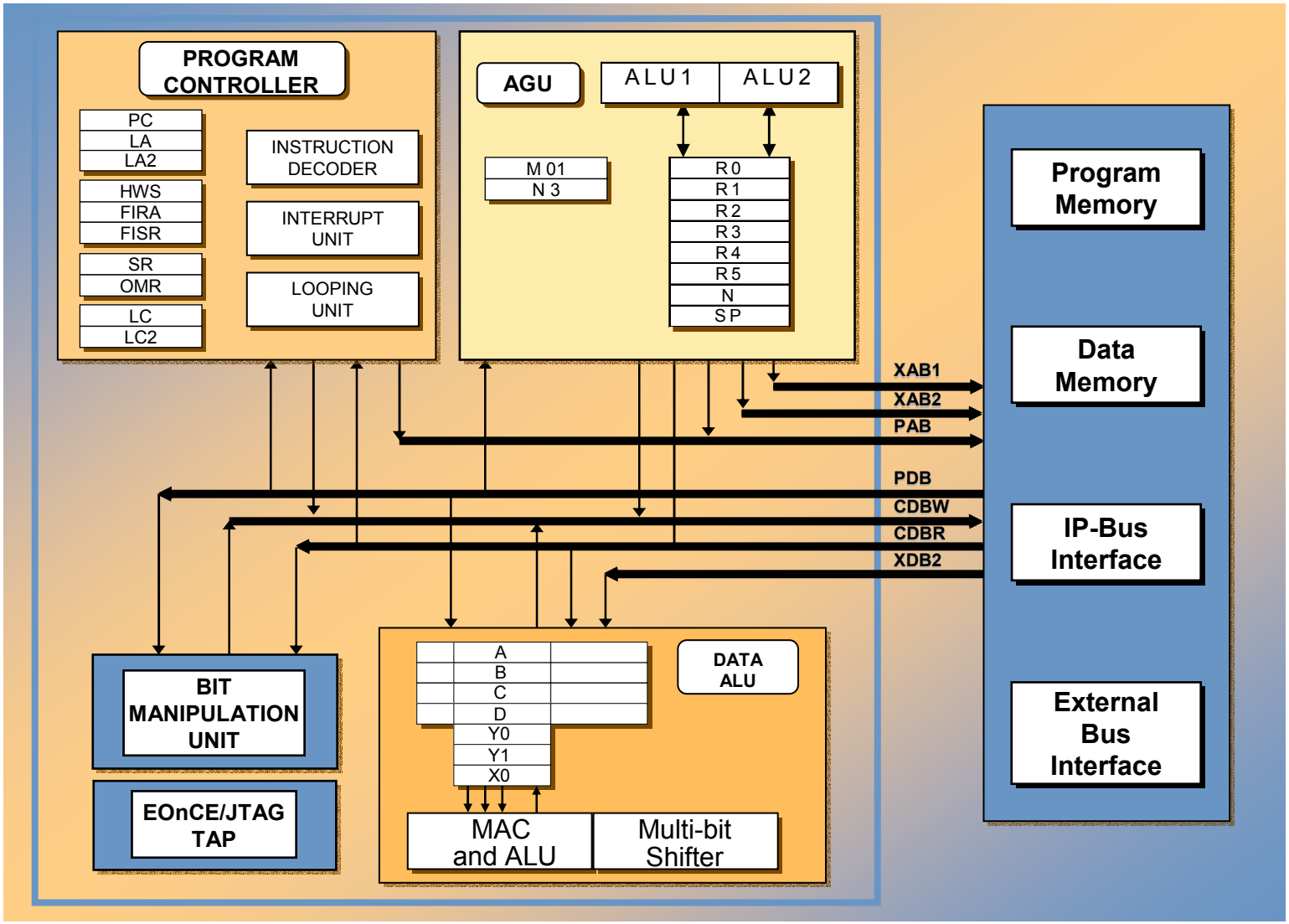
DSP56800E Improvement Summary

CPU	MIPS	Clocks perInstr	# Interrupt Levels	Registers	Data Types	Program Memory Adr Space	Data Memory Adr Space	Technology
DSP56800	20-40	2	2	5 Data 5 Address	16-bit	128 KB	128 KB	Semi-custom
DSP56800E	120-200	1	4	7 Data 8 Address	8-bit, 16-bit 32-bit	4 MB	32 MB	Fully Synthesizable & Scanable

Other Improvements

- Compiler Efficiency
- Real-time Debug
- Fast Interrupt
- Nested Hardware Looping
- Additional Addressing Modes
- Five (5) Software Interrupt Traps

DSP56800E Core Architecture



Instruction Fetch:

PAB - 21 bits
PDB - 16 bits

1st Data Access:

XAB1 - 24 bits
CDBR - 32 bits

2nd Data Access:

XAB2 - 24 bits
XDB2 - 16 bits

Operations Performed:

1st - PAB / PDB
2nd - XAB1 / CDBR-CDBW
3rd - XAB2 / XDB2

DSP56800E Parallel Move Instructions

Definition: One or two word sized moves occur in parallel with an arithmetic operation

The "Single Parallel Move"

```
ADD X0,A        Y0,X:(R1)+N
```

Arithmetic Operation — 16-Bit Move

Examples:

MPYR	X0,Y0,A	X:(R0)+,X0
MAC	-X0,Y0,A	Y0,X:(R1)+N
ADD	A,B	Y1,X:(R2)+
TFR	Y1,A	A,X:(R3)+
INC.W	A	X:(R0)+,B
ASL	A	X:(R1)+N,C

The "Dual Parallel Read"

```
MACR  X0,Y0,A    X:(R0)+N,Y1    X:(R3)+N3,X0
```

Arithmetic Operation — 1st 16-Bit Read — 2nd 16-bit Read

Examples:

MPY X0,Y0,A	X:(R0)+,Y0	X:(R3)+,X0
MACR X0,Y0,B	X:(R1)+,Y1	X:(R3)-,C
ADD X0,A	X:(R4)+N,Y0	X:(R3)+,X0
SUB Y1,B	X:(R4)+N,Y1	X:(R3)+N3,X0
MOVE.W	X:(R0)+N,Y0	X:(R3)+,C

Demonstration using Parallel Instructions

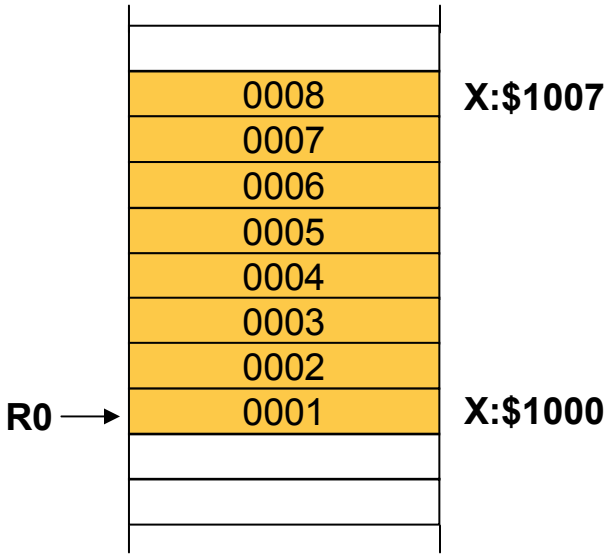
Initialize array of size 8 - direct addressing

Initialize pointer register R0: $1000

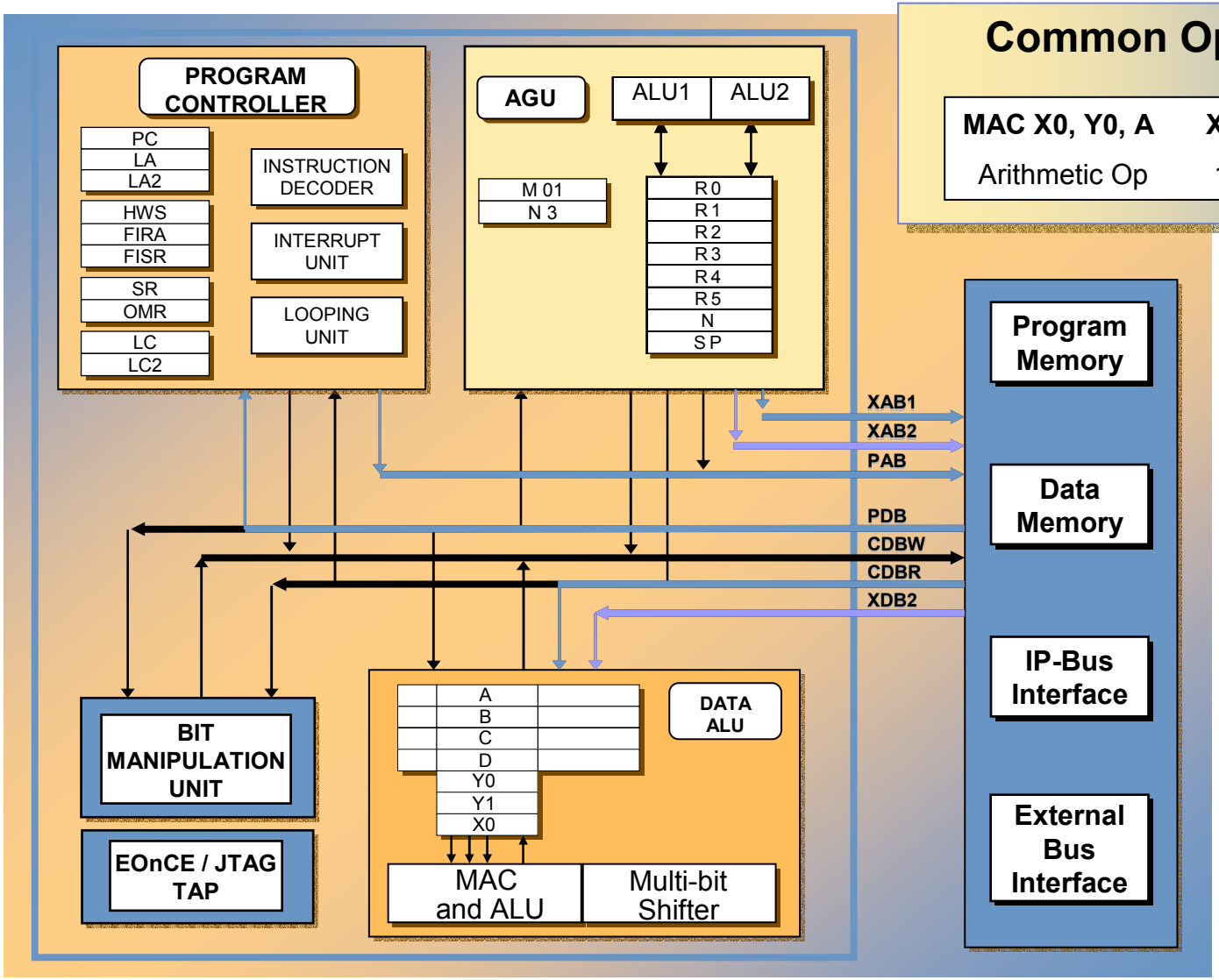
Clear accumulator A and read 1st value

Add value to A, while reading new value

Multiple right shift to generate average



Mapping the Architecture to DSP Algorithms



Common Operation in DSP

MAC X0, Y0, A	X:(R4)+, Y1	X:(R3)+, C
Arithmetic Op	1st Read	2nd Read

Operations Performed:

- Multiply-Accumulate
- 3 Memory Accesses
- 2 Address Additions

Instruction Fetch:

PAB - 21 bits
PDB - 16 bits

1st Data Access:

XAB1 - 24 bits
CDBR - 32 bits

2nd Data Access:

XAB2 - 24 bits
XDB2 - 16 bits

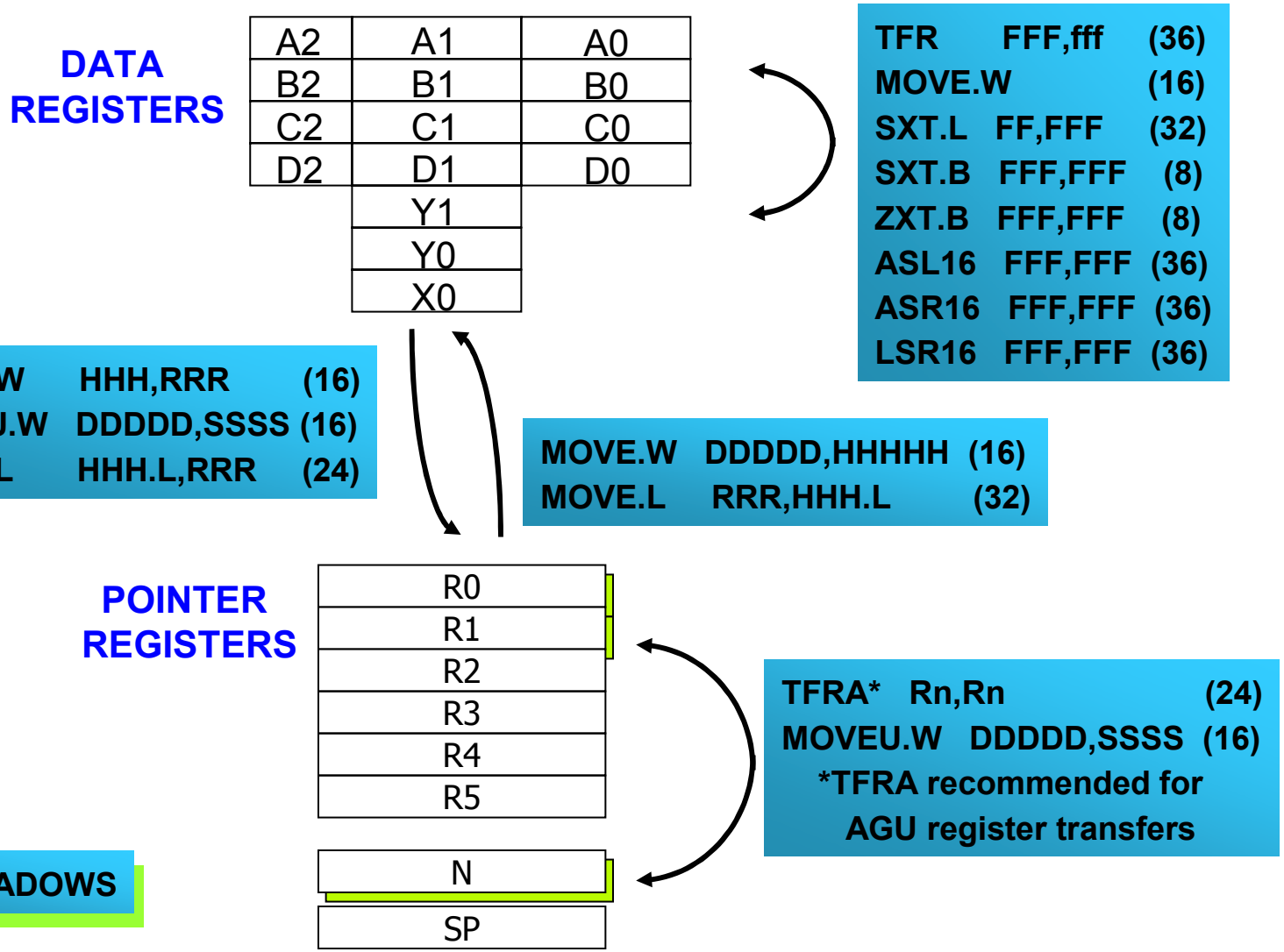
Single Parallel Move Instructions

Data ALU Operation		Parallel Memory Move	
Operation	Operands	Source	Destination
MAC MPY MACR MPYR	Y1,X0,F Y0,X0,F Y1,Y0,F Y0,Y0,F A1,Y0,F B1,Y1,F	X:(R_j)+ X:(R_j)+N	X0 Y1 Y0 A B C A1 B1
MAC MPY MACR	C1,Y0,F C1,Y1,F		
MAC	–C1,Y0,F –C1,Y1,F		
ADD SUB CMP TFR	X0,F Y1,F Y0,F C,F A,B B,A	X0 Y1 Y0 A B C A1 B1	X:(R_j)+ X:(R_j)+N
SAT	F,Y0		
EOR.L	C,F		
ABS ASL ASR CLR RND TST INC.W DEC.W NEG	F		
SUBL[1]	A,D,B	X:(R1)+	AD

Dual Parallel Read Instructions

Data ALU Operation		First Memory Read		Second Memory Read	
Operation	Operands	Source 1	Destination1	Source 2	Destination 2
MAC MPY MACR MPYR	Y1,X0,F Y1,Y0,F Y0,X0,F C1,Y0,F	X:(R0)+ X:(R0)+N X:(R1)+ X:(R1)+N	Y0 Y1	X:(R3)+ X:(R3)–	X0
		X:(R4)+ X:(R4)+N	Y0	X:(R3)+ X:(R3)+N3	X0
ADD SUB	X0,F Y1,F Y0,F A,B B,A	X(R0)+ X(R0)+N X(R4)+ X(R4)+N	Y1	X(R3)+ X(R3)+N3	C
TFR	A,B B,A				
CLR ASL ASR	F				
MOVE.W					

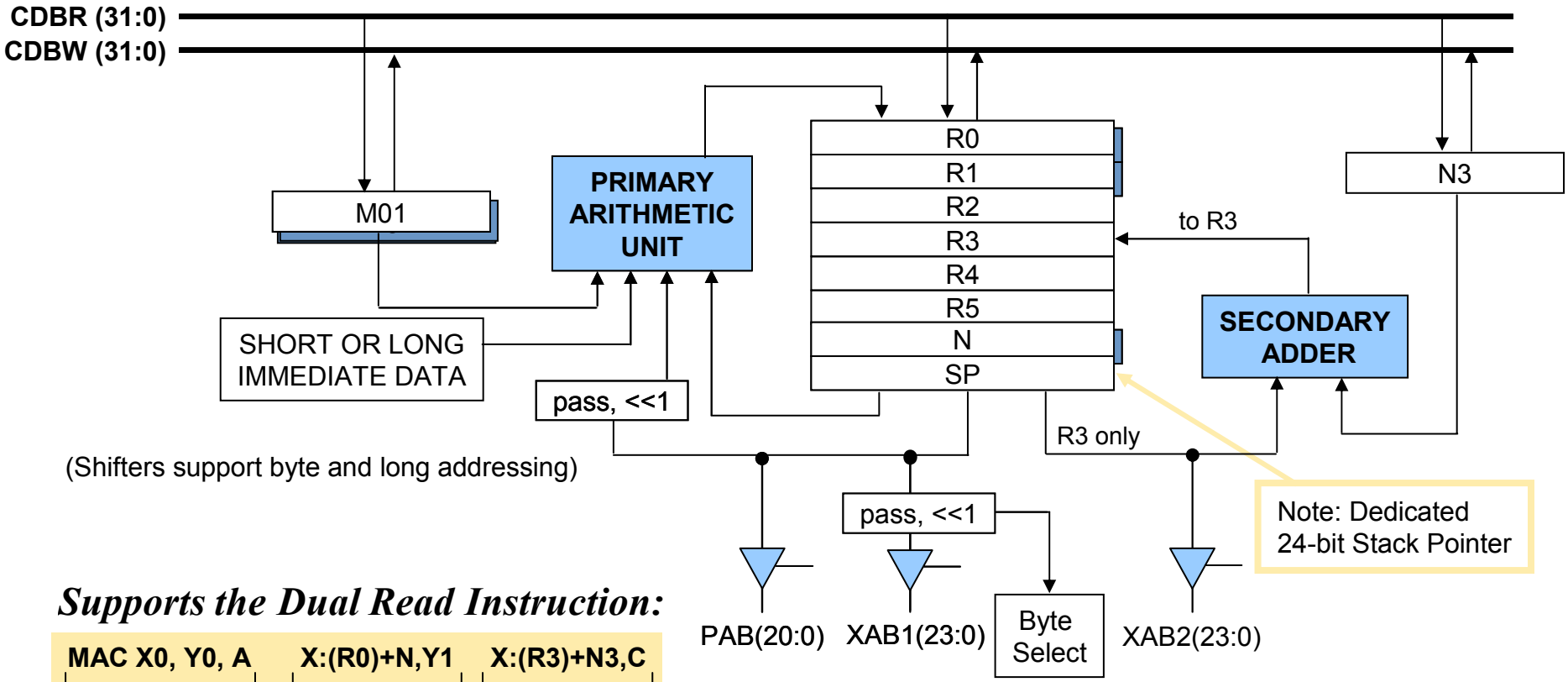
Powerful Set of Register to Register Moves



AGU Block Diagram

Provides up to 2 Data Memory Addresses per cycle

Performs up to 2 Address Calculations after Issuing Addresses



Supports the Dual Read Instruction:

MAC X0, Y0, A	X:(R0)+N,Y1	X:(R3)+N3,C
Data ALU	Uses XAB1	Uses XAB2

Powerful Set of Addressing Modes

Indirect

- X:(Rn) No Update
- X:(Rn)+ Post Increment
- X:(Rn)- Post Decrement
- X:(Rn)+N Post Update by Register

Indexed

- X:(Rn+x) Indexed:3-bit Offset
- X:(SP-xx) Indexed:6-bit Offset
- X:(Rn+xxxx) Indexed:16-bit Offset
- X:(Rn+xxxxxx) Indexed:24-bit Offset
- X:(Rn+N) Indexed: By a Register

Supports 8, 16, 32-bits

Supports Modulo Arithmetic

Immediate

- #x 5-bit “Long” Constant
- #xx 6-bit Loop Ct
- #xx 7-bit Short
- #xxxx 16-bit
- #xxxxxxxx 32-bit

Absolute

- X:aa 6-bit Absolute Short
- X:<<pp 6-bit Peripheral Direct
- X:xxxx 16-bit Absolute
- X:xxxxxx 24-bit Absolute

Other

- DDDDD Register Direct
- * Inherent

Examples of Addressing Modes

Addressing Modes for Move Instr.

```
move.w    <reg>,<reg>
move.w    #<-64,63>,<reg>
move.w    #xxxx,<reg>
move.w    X:xxxx,<reg>
move.w    X:xxxxxx,<reg>
move.w    X:(Rn),<reg>
move.w    X:(Rn)+,<reg>
move.w    X:(Rn)-,<reg>
move.w    X:(Rn+N),<reg>
move.w    X:(Rn)+N,<reg>
move.w    X:(Rn+xxxx),<reg>
move.w    X:(Rn+xxxxxx),<reg>
move.w    X:(SP-xx),<reg>
move.w    X:<<pp,<reg>
move.w    X:aa,<reg>
move.w    <reg>,X:(SP-xx)
move.w    <reg>,X:xxxx
move.w    <reg>,X:(Rn)
move.w    <reg>,X:(Rn)+
move.w    <reg>,X:(Rn)-
move.w    <reg>,X:(Rn+N)
move.w    <reg>,X:(Rn+xxxx)
.....     and many more !
```

Addressing Modes for ADD Instr.

```
add       <reg>,<reg>
add.w     #<0-31>,<reg>
add.w     #xxxx,<reg>
add.w     X:xxxx,<reg>
add.w     X:xxxxxx,<reg>
add.w     X:(Rn),<reg>
add.w     X:(Rn+xxxx),<reg>
add.w     X:(SP-xx),<reg>
add.w     <reg>,X:(SP-xx)
add.w     <reg>,X:xxxx
.....     and many more !
```

Launched by Motorola freescale semiconductor

Enhanced Set of AGU Arithmetic Instructions

Arithmetic:

- *ADDA*
- ADDA.L
- CMPA
- CMPA.W
- *DECA*
- DECA.L
- DECTSTA
- NEGA
- SUBA
- SXTA.B
- SXTA.W
- TSTA.B
- TSTA.W
- TSTA.L
- *TSTDECA.W*
- ZXTA.B
- ZXTA.W

Shifting and Moves:

- ASLA
- ASRA
- LSRA
- TFRA

AGU Instructions in 56800:

- ADDA == LEA (incr)
- DECA == LEA (decr)
- TSTDECA.W == TSTW (Rn)-

Structured Programming - The Software Stack

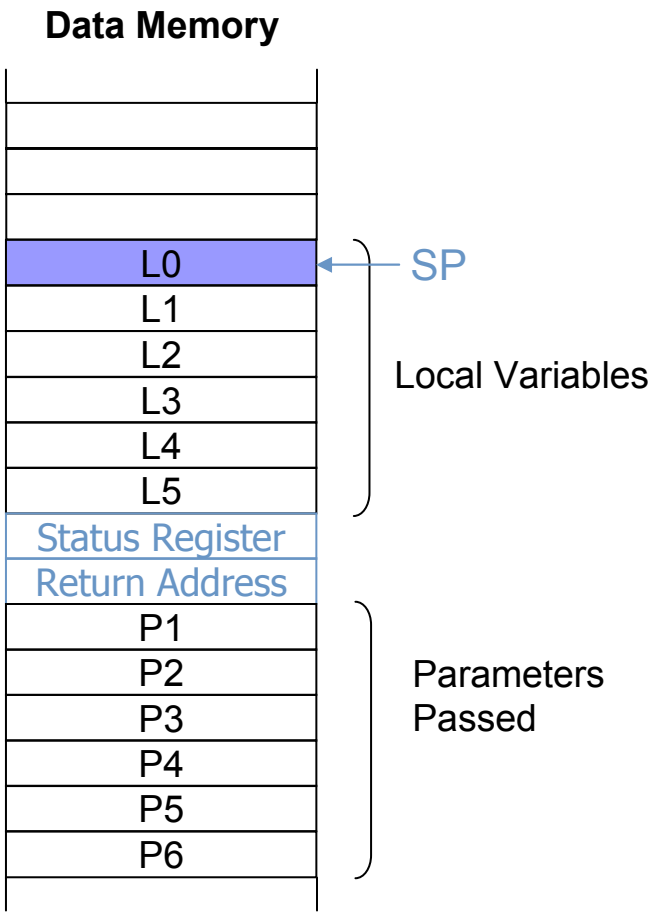
Software Stack support for structured programming

Supports Local Variables

Supports Parameter Passing to a Function

For both C and Assembly Code

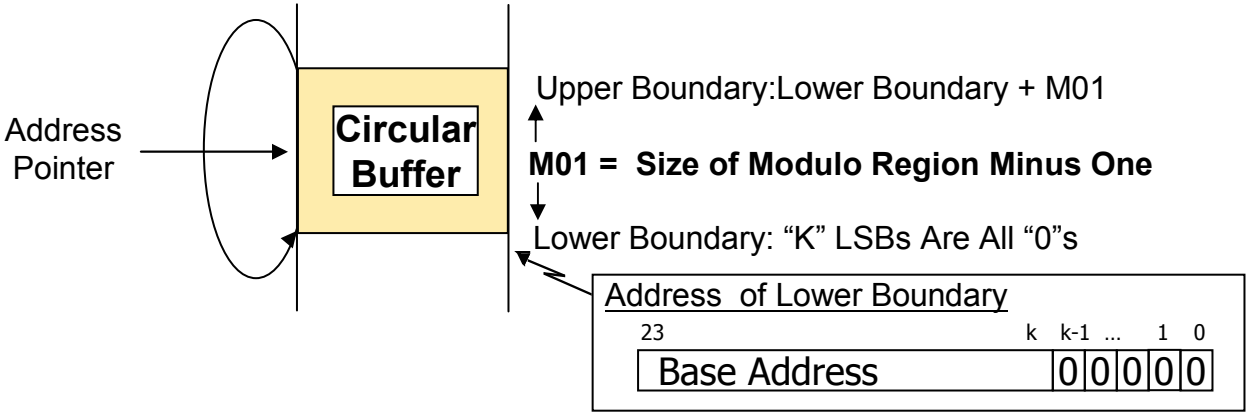
Utilizes strong set of SP Addressing Modes



Example: Local Variable "L5" accessed as X:(SP-5)

Also Note:
JSR and interrupts automatically stack PC and SR

AGU Modulo Addressing



Modulo Addressing Features:

- **Available for byte, word, and long accesses**
- **Available for the R0 and R1 pointers only. M01[15] = 1, then modulo on R0 & R1**
- **Occurs only when address arithmetic is performed to calculate effective address**
- **Supports buffer sizes from 2 locations to 16384 words (2 to 8192 for long values)**

The equations for modulo addressing are:

- **R0[23:k] = R0[23:k] (not modified)**
- **R0[k-1:0] = (R0[k-1:0] + offset) MOD (M01 + 1)**

Demonstration using Modulo and Linear Addressing

Initialize 2 arrays with the value of address

Linear Array - X:$1020

Circular Array - X:$1000, 5 elements

- **M01 = $0004; R0 (modulo)**

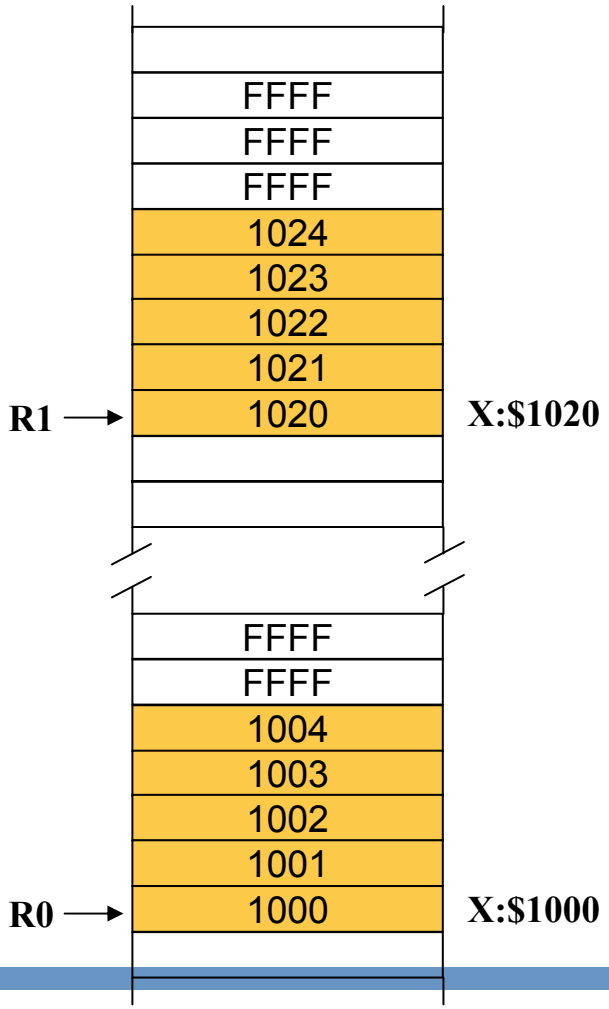
Initialize R0: start address of circular buffer

Initialize R1: start address of linear buffer

Write incrementing values to both arrays

R0 & R1 post-updated by N

After run: NEGA N, and repeat sequence



oke Metrowerks CodeWarrior

Modulo Arithmetic Example

Using the Modulo Buffer:

- **Used in post-update instructions: "MOVE.W X:(R0)+,X0"**
- **Used with AGU arithmetic instructions**
- **Example demonstrates usage; R0 = $000800:**

```
MOVE.W    X:(R0)+,X0  ; 1st time accesses location $0800
                      ; and bumps the pointer to location $0801
MOVE.W    X:(R0)+,X0  ; 2nd accesses at location $0801
MOVE.W    X:(R0)+,X0  ; 3rd accesses at location $0802
MOVE.W    X:(R0)+,X0  ; 4th accesses at location $0803
MOVE.W    X:(R0)+,X0  ; 5th accesses at location $0804
MOVE.W    X:(R0)+,X0  ; 6th accesses at location $0805
MOVE.W    X:(R0)+,X0  ; 7th accesses at location $0806
MOVE.W    X:(R0)+,X0  ; 8th accesses at location $0807
MOVE.W    X:(R0)+,X0  ; 9th accesses at location $0808
                      ; and bumps the pointer to location $0800

MOVE.W    X:(R0)+,X0  ; 10th accesses at location $0800
MOVE.W    X:(R0)+,X0  ; 11th accesses at ...
```

Example:

Buffer Size = 9

M01= $0008

R0: Modulo

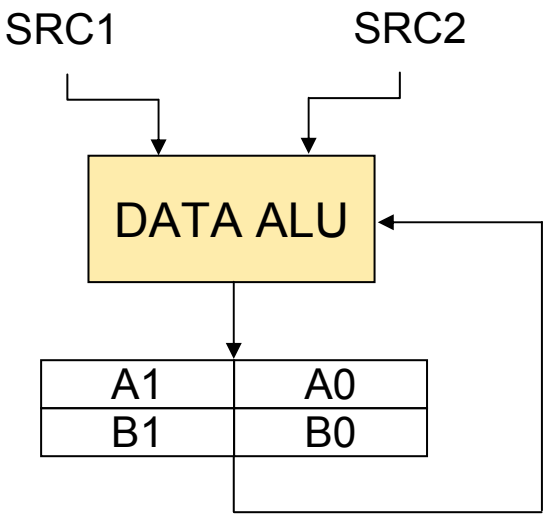
R1: Linear

Other Useful Features:

- **works for decrementing addressing modes too**
- **modulo operation works correctly even if the** pointer does not land exactly on upper or lower boundary
- **modulo buffer sizes are** not constrained to a power of two

Data ALU - General Purpose Register File

Conventional DSP

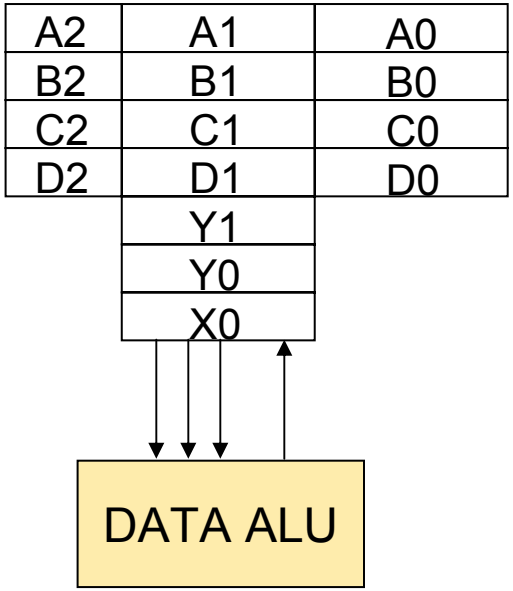


INC,DEC [A or B]
ASL,ASR

ADD, etc. [SRC1], [A or B]

"Accumulator Based"

DSP56800E

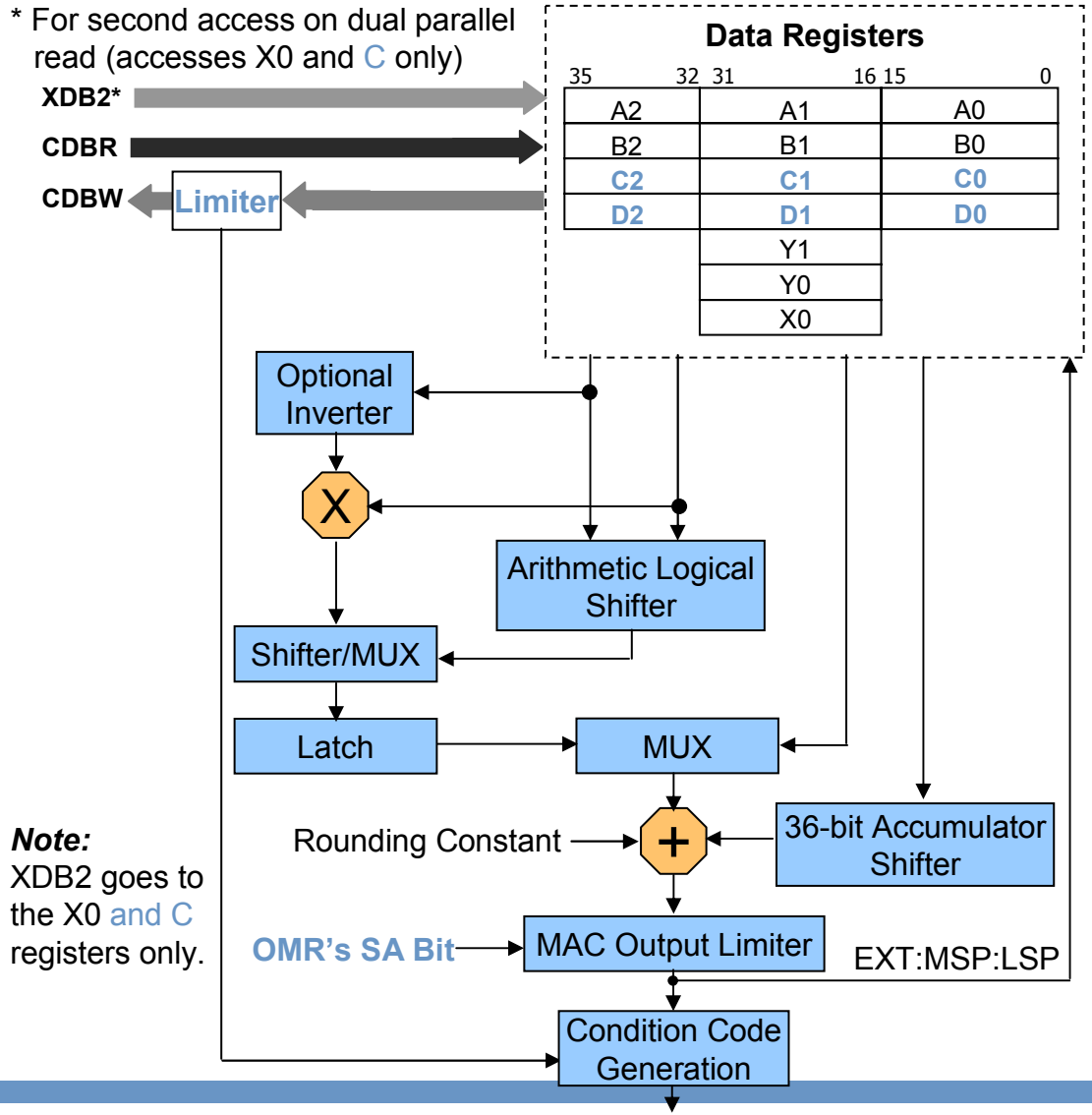


INC.W, DEC.W [FFF]
ASL, ASR

ADD, etc. [FFF], [FFF]

"GP Register File"

Data ALU Block Diagram

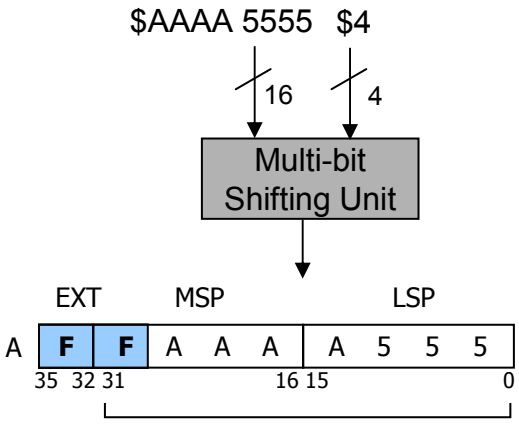


Capabilities

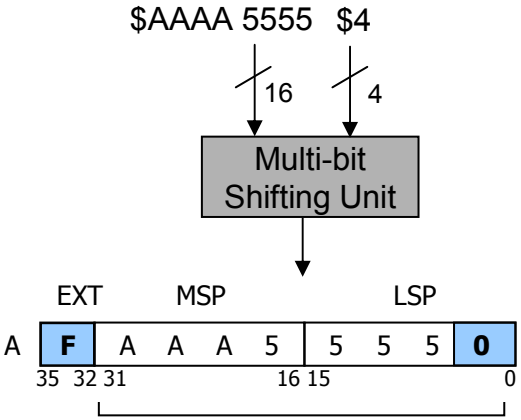
- Multiplication (w/ Rounding)
- Multiply-Accumulate (w/ Rounding)
- Multiprecision Multiplication Support
- Addition and Subtraction
- Increments and Decrements
- Tests and Compares (8, 16, 32, 36 bits)
- 16 and 32-bit Logical Operations
- 1's and 2's complement
- Single Bit Arithmetic & Logical Shifts
- 16-bit Arithmetic and Logical Shifts
- 32-bit Arithmetic and Logical Shifts
- Single Bit Rotates
- Rounding
- Absolute Value
- Sign/Zero Extension
- Limiting on Move Instructions
- Conditional Register Transfer
- Division Iteration Instruction
- Count Leading Bits
- Normalization

Shifting 32-Bit Long Words (Bidirectional)

EXAMPLE - RIGHT SHIFTING: ASRR.L #4,A



EXAMPLE - LEFT SHIFTING: ASLL.L #4,A



Operation	Operands	C	W	Comments
ASLL.L	#<0-31>,fff	2	1	arithmetic shift left by a 5-bit positive immediate integer
	EEE,FFF	2	1	**Bi-directional** arithmetic shift of destination by value in the first operand: positive -> left shift.
ASRR.L	#<0-31>,fff	2	1	arithmetic shift right by a 5-bit positive immediate integer
	EEE,FFF	2	1	**Bi-directional** arithmetic shift of destination by value in the first operand: positive -> right shift.
LSRR.L	#<0-31>,fff	2	1	logical shift right by a 5-bit positive immediate integer
	EEE,FFF	2	1	**Bi-directional** logical shift of destination by value in the first operand: positive -> right shift

Normalization

Normalization — left justify number, removing redundant sign bits

Performed in two instructions:

```
; Normalization Algorithm — 16-Bits
      CLB       A,X0
      ASLL.W    X0,A
```

```
; Normalization Algorithm — 32-Bits
      CLB       A,X0
      ASLL.L    X0,A
```

Example 1 - Normalization of a Positive Value

Before Execution: A = $0:0200:0000

A2	A1	A0
0000	0.000 0010 0000 0000	0000 0000 0000 0000

<< 5

After Execution: A = $0:4000:0000

A2	A1	A0
0000	0.100 0000 0000 0000	0000 0000 0000 0000

Example 2 - Normalization of a Negative Value

Before Execution: A = $F:E400:0000

A2	A1	A0
1111	1.110 0100 0000 0000	0000 0000 0000 0000

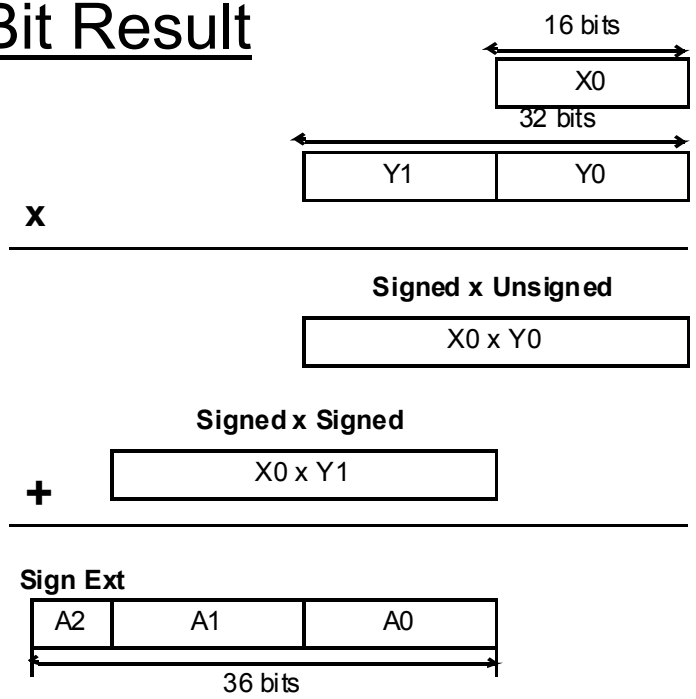
<< 2

After Execution: A = $F:9000:0000

A2	A1	A0
1111	1.001 0000 0000 0000	0000 0000 0000 0000

==> CLB Instruction also useful for finding 1st significant bit

Fractional 16 x 32=> 36-Bit Result



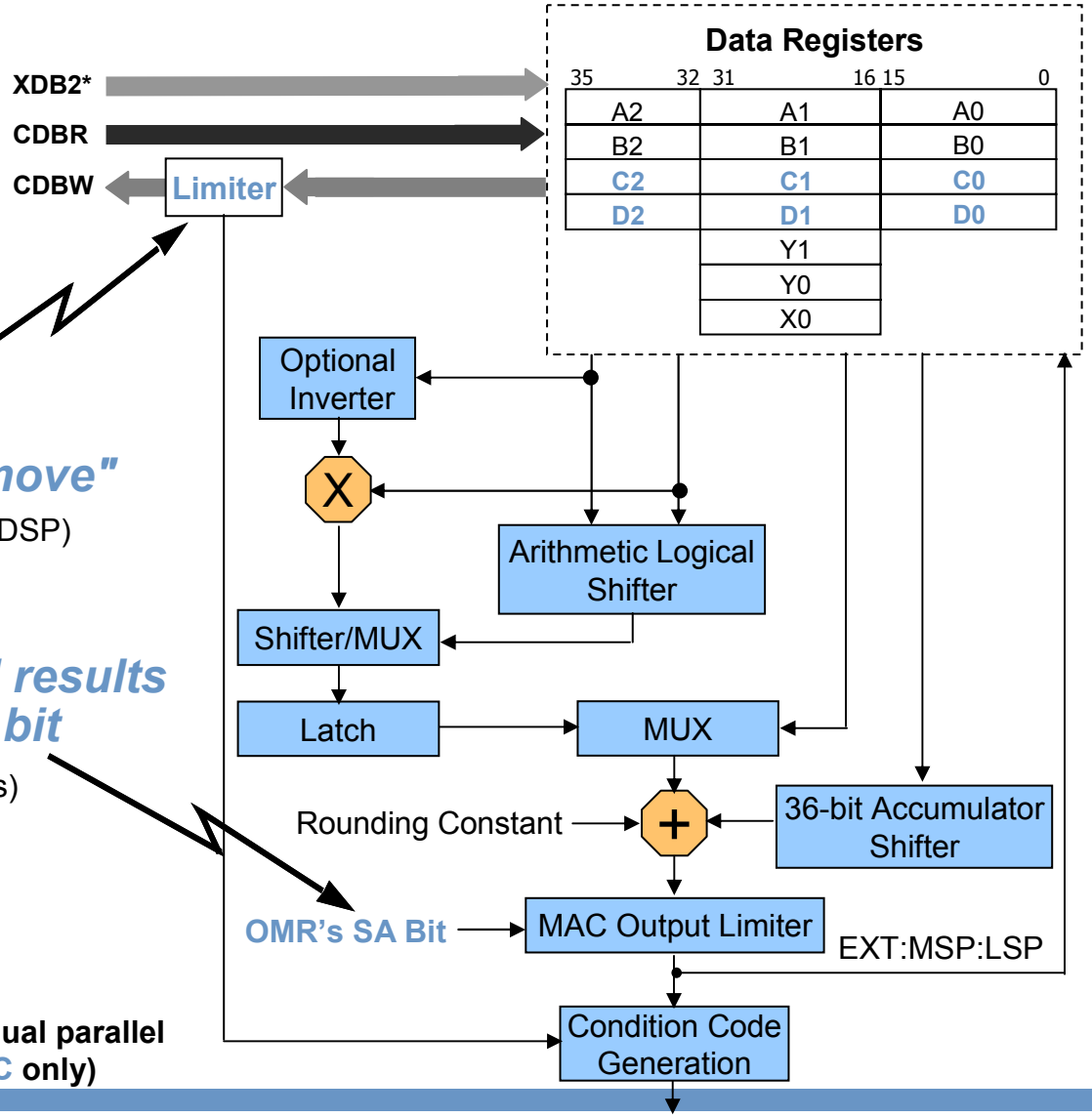
3 Words, 3 Cycles

16 x 32 Bit Fractional Multiplication — 36-Bit Result

FF2:FF1:FF0 = X0 x Y1:Y0
(Both Fractional Operands are Signed; 3 Cycles, 3 Words)

```
;Signed 16-Bit x Signed 32-Bit Fractional Multiplication
        MPYSU   X0,Y0,A      ; Y1:Y0 = signed X0 x unsigned Y0
        ASR16   A            ; Align 1st product
        MAC     X0,Y1,A      ; A2:A1:A0 = final 36-bit result
```

Two Supported Saturation Mechanisms



Saturation via "move"

(most typically used for DSP)

Saturate all Data ALU results via OMR control bit

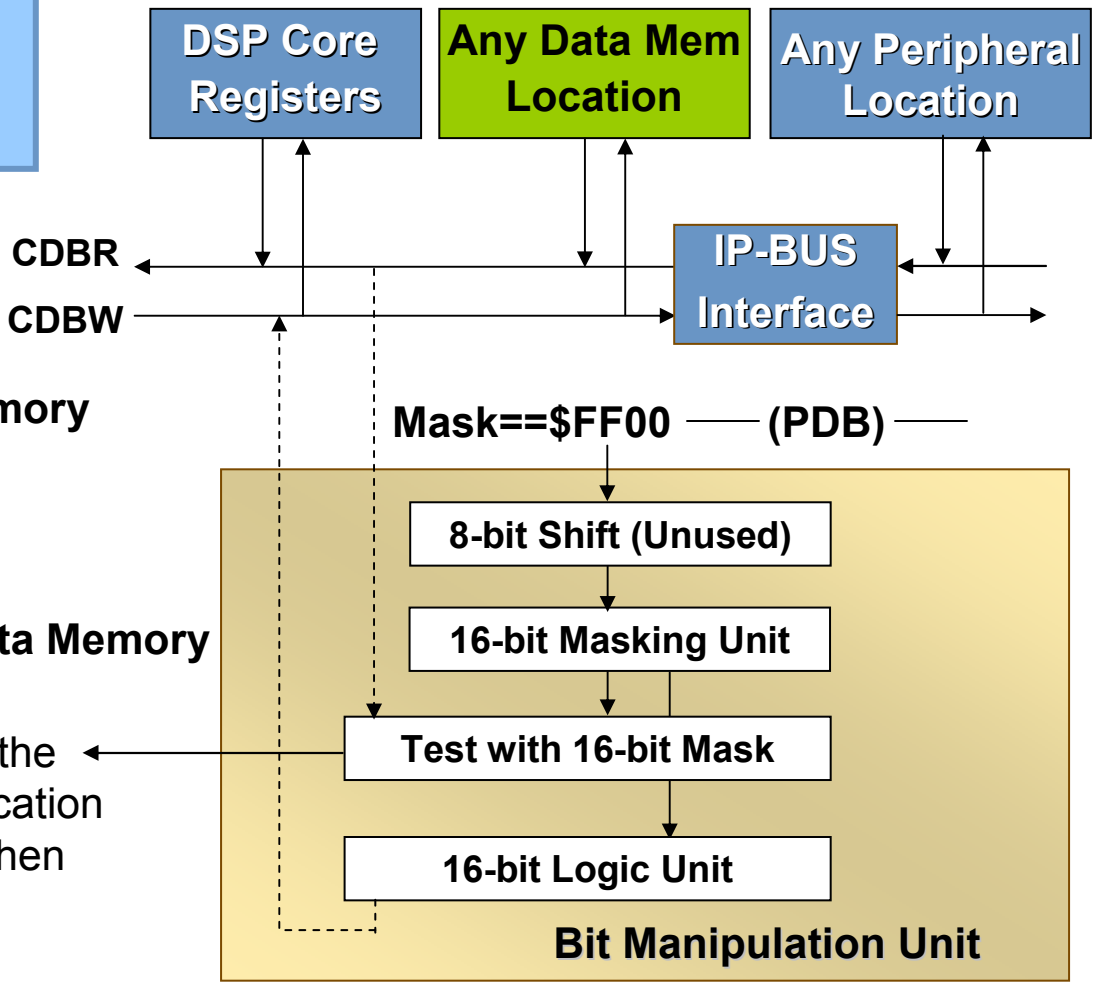
(for bit-exact applications)

*** For second access on dual parallel read (accesses X0 and C only)**

Flexible Bit Manipulation Instructions

BFCLR	#$FF00	X:(R0)
Operation (clear bits)	16-Bit Mask	Operand in Memory

OPCODE: 8040 FF00



Steps

1. **Read 16-bit Word from Data Memory**
2. **Test Masked (upper 8) Bits**
3. **Clear Masked (upper 8) Bits**
4. **Write modified Word back to Data Memory**

Carry bit set to “1” if all bits in the Upper Byte of the Memory Location were all “1”’s; otherwise “0”. Then clears all selected bits.

Summary: Bit Manipulation Capabilities

Strong Set of Bit Manipulation Instructions:

- BFSET: **Test and then set a field of bits in a word**
- BFCLR: **Test and then clear a field of bits in a word**
- BFCHG: **Test and then invert a field of bits in a word**
- BFTSTH: **Test a field of bits for all "1"s**
- BFTSTL: **Test a field of bits for all "0"s**
- BRSET: **Branch if a selected set of bits are all "1"s**
- BRCLR: **Branch if a selected set of bits are all "0"s**

AND, OR, and XOR w/ 16-bit values

Operates on any register or data memory location on the chip

Operates using a 16-bit mask

- **Except:** BRSET **and** BRCLR **only allow an 8-bit mask on upper or lower byte**

Other Bit Manipulation Instructions Performed Only in the Data ALU:

- **16-Bit Bidirectional Multi-bit Shifting (uses Data ALU registers)**
- **Arithmetic and Logical Shifts (uses Data ALU registers)**
- **Rotates (uses Data ALU registers)**
- **Increment and Decrement of Memory Locations**

Demonstration using Read-Modify-Write

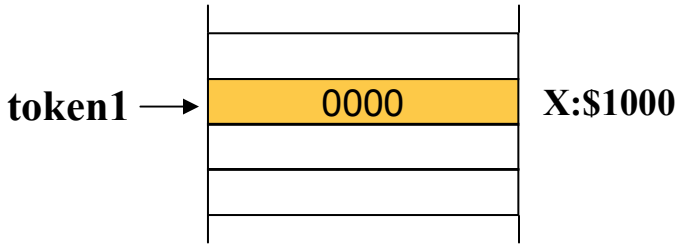
C program: Initialize token1 as "Available"

ASM program: read status of token1, return 1 if busy

- **token1: 0001 == TAKEN**
- **token1: 0000 == AVAILABLE**

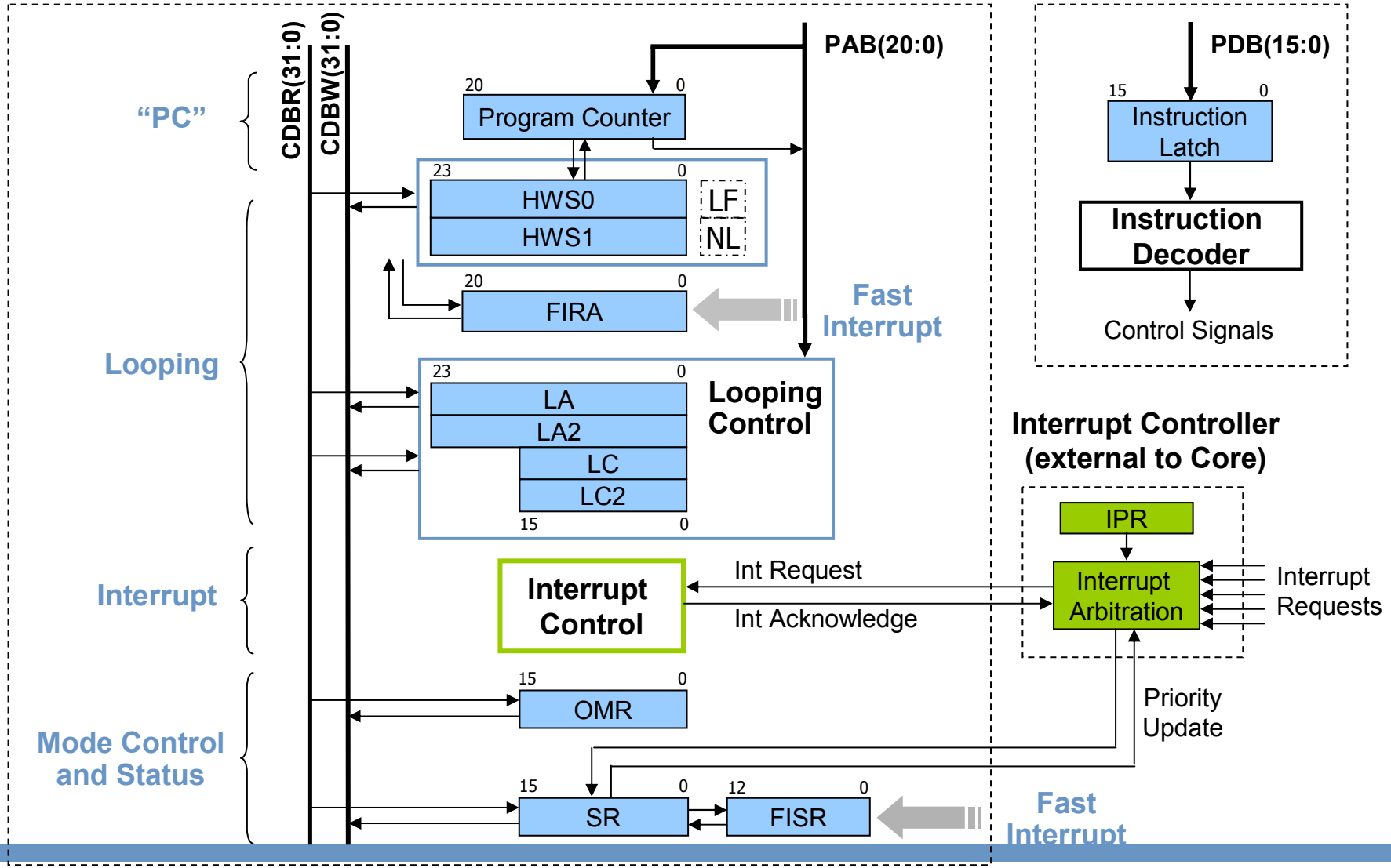
On iteration 5, clear status of token1

Recapture token1: read-modify-write in atomic (non-interruptible) sequence



oke Metrowerks™ CodeWarrior™

Program Controller Block Diagram



Nested Hardware Looping

Example - DSP56800E Assembly Code:

```
    CLR.W    A

    DO     #2,END_OUTR          ; Outer DO Loop
     DO     #20,END_INNR        ; Inner DO Loop
       MOVE.W   X:(R0)+,A       ;      (body of innermost loop)
       MOVE.W   X:(R1)+,B       ;      (body of innermost loop)
       ADD                      ;      (body of innermost loop)
       MOVE.W   A,X:(R3)+       ;      (body of innermost loop)
 END_INNR
     MOVE.W   A,X:(R5)+         ;
END_OUTR
```

Theoretical Best:
2*20*4 instrs = 160 Cycles

40 Loop Iterations in 172 Cycles
10 Program Words

Demonstration using Nested Hardware Looping

Initialize three 32-bit locations

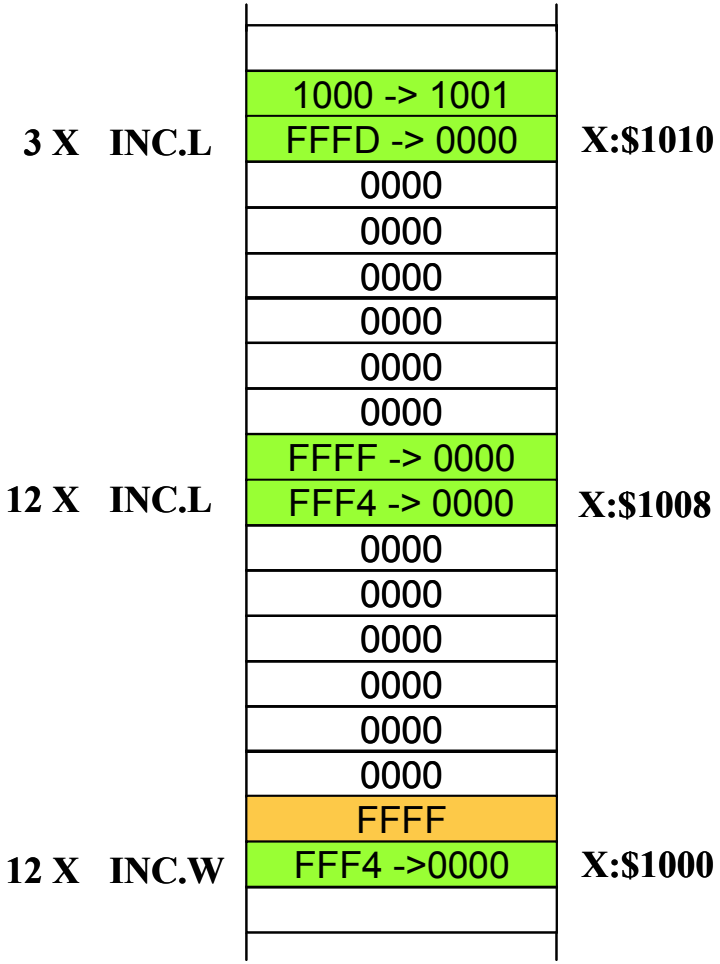
Inner loop repeats 4 times

Outer loop repeats 3 times

Use INC.W & INC.L to generate:

- **$1001 0000 @ X:$1010**
- **$0000 0000 @ X:$1008**
- **$0000 @ X:$1000**

```
DO  #3,OUTER_LOOP
    DO  #4,INNER_LOOP
        INC.W X:$1000        ; incr short mem word
        INC.L  X:$1008       ; incr  long mem word
INNER_LOOP:
    INC.L X:$1010            ; incr  long mem word
OUTER_LOOP:
```



oke Metrowerks CodeWarrior

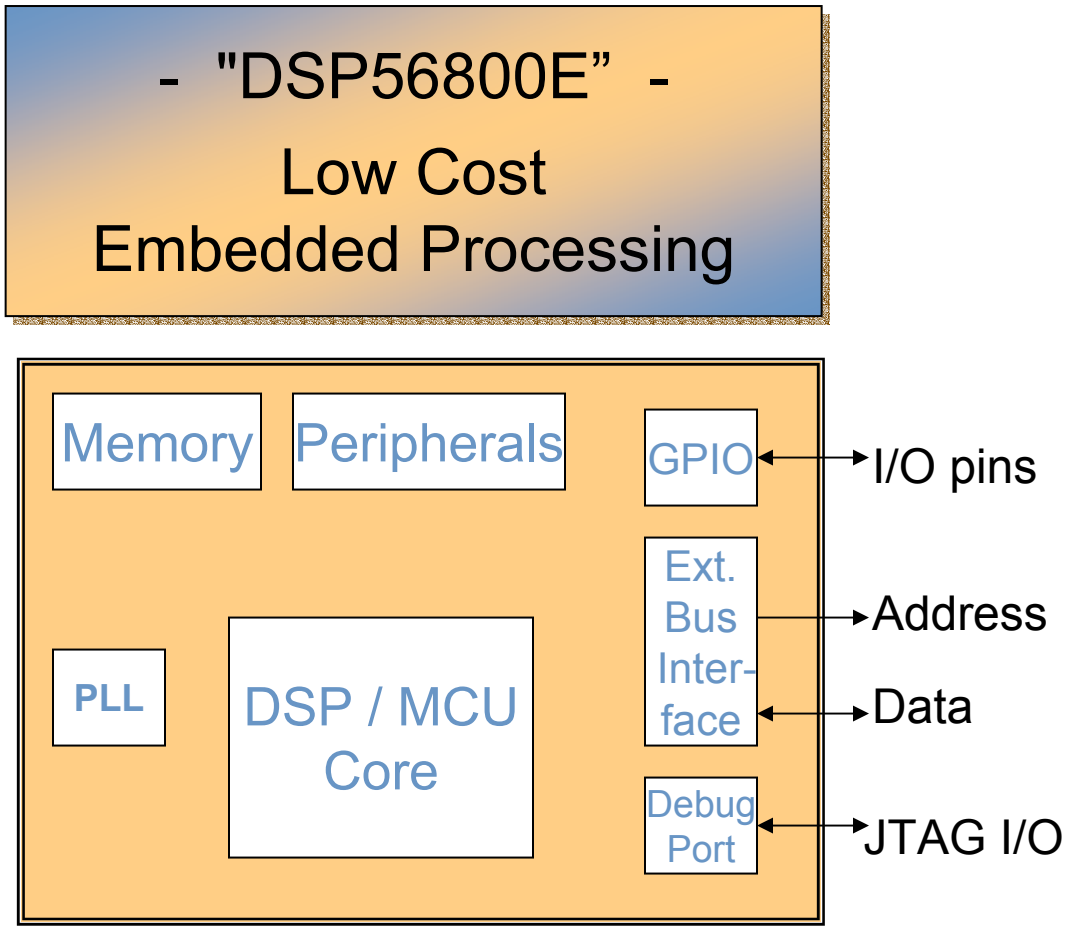
Embedded Connectivity Summit 2004

Launched by Motorola
freescale™
semiconductor

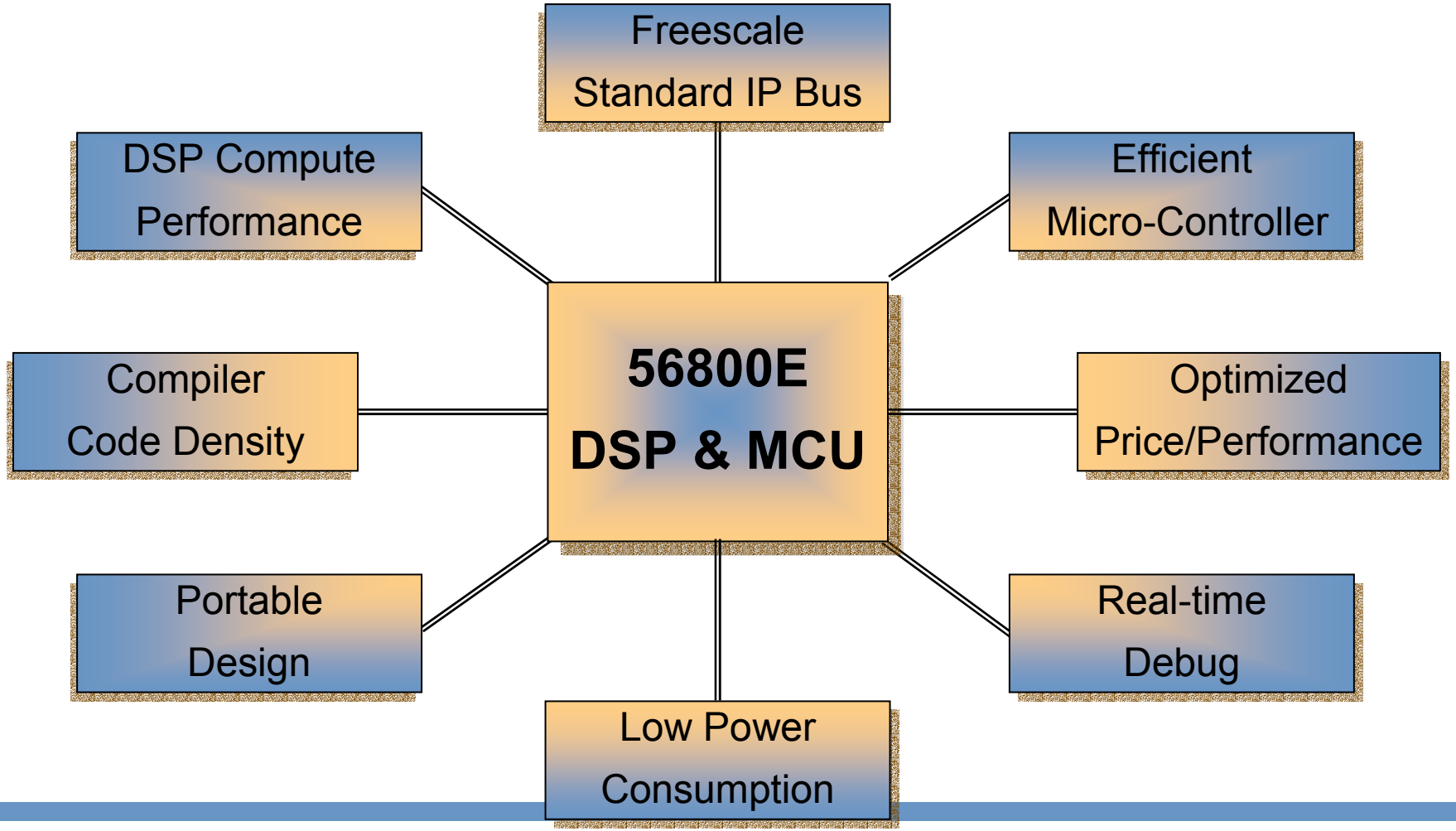
DSP56800E ARCHITECTURE

Launched by Motorola
freescale™
semiconductor

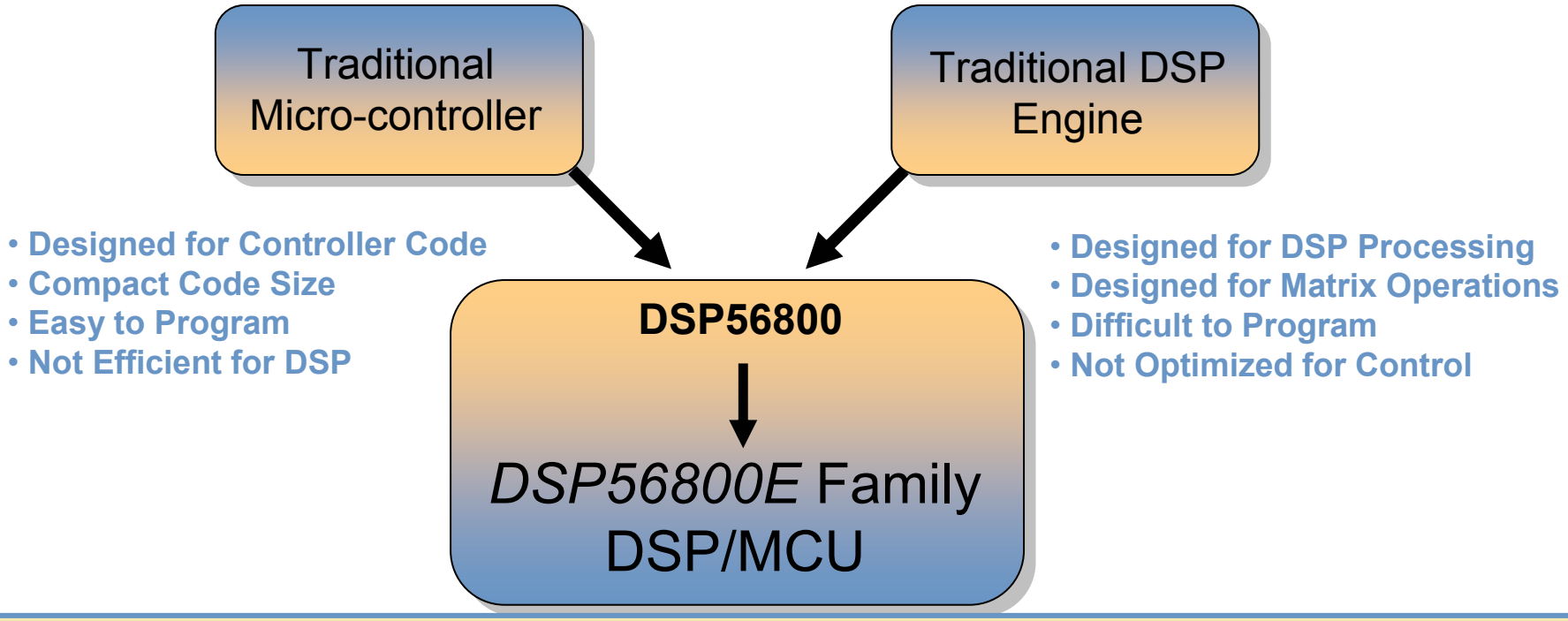
- “DSP56800E” -
Powerful, Low Cost DSP/MCU Engine



Bringing it All Together
The 56800E Family Foundation



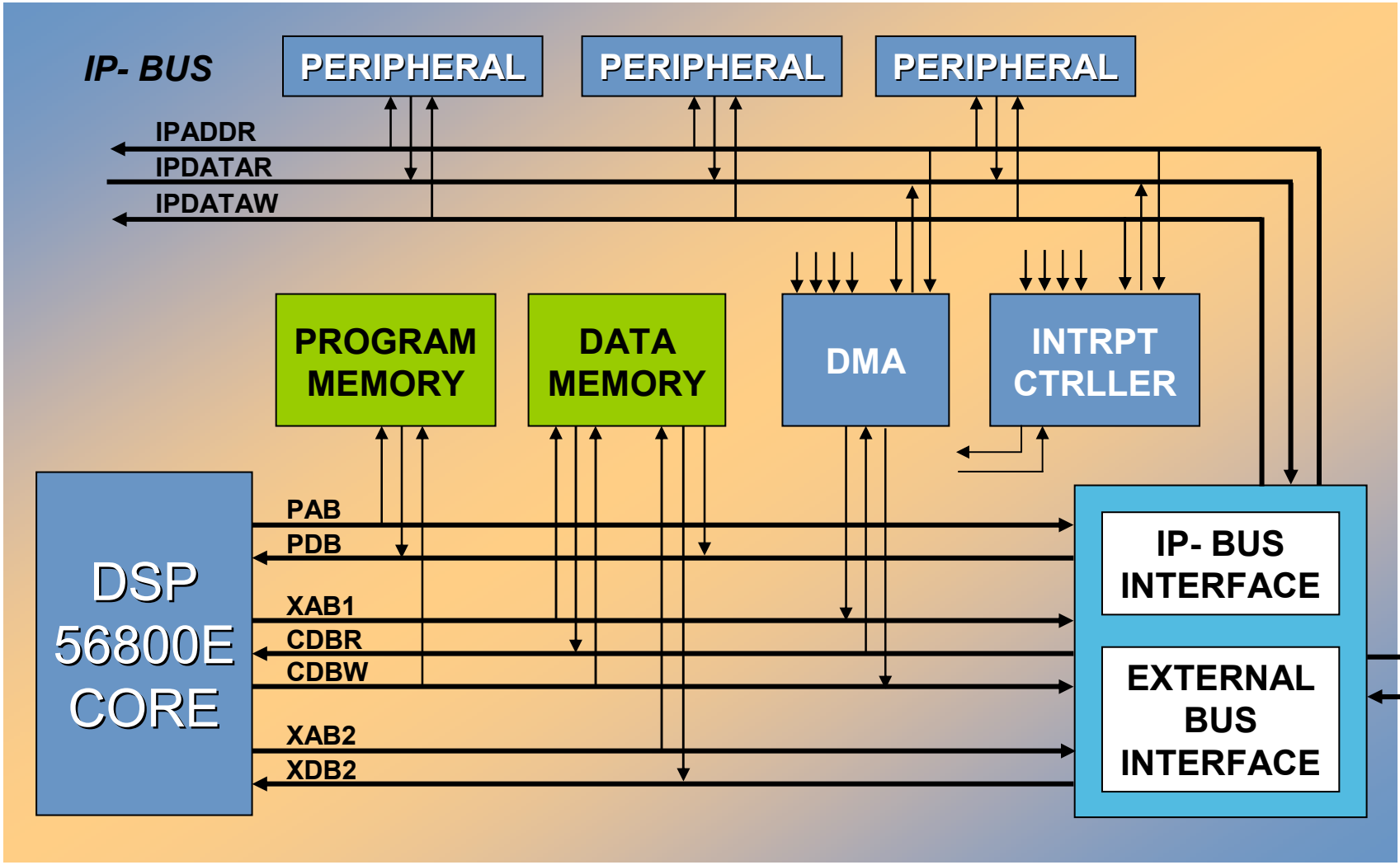
Technology Shift: Combined DSP and Controller Functionality



- Instructions Optimized for Controller Code, DSP, Matrix Operations
- Compact Assembly & "C" Compiled Code Size
- Easy to Program
- Adequate MIPS Headroom and Extended Addressing Space

Launched by Motorola freescale semiconductor

DSP56800E System Architecture



Hawk V2 Architectural Distinctives

High Level Abstraction of Application Software

Full Set of Data Types

High Code Density for Minimized Solution Cost

Large Address Spaces

Full Source Code Compatibility

Powerful Register Set

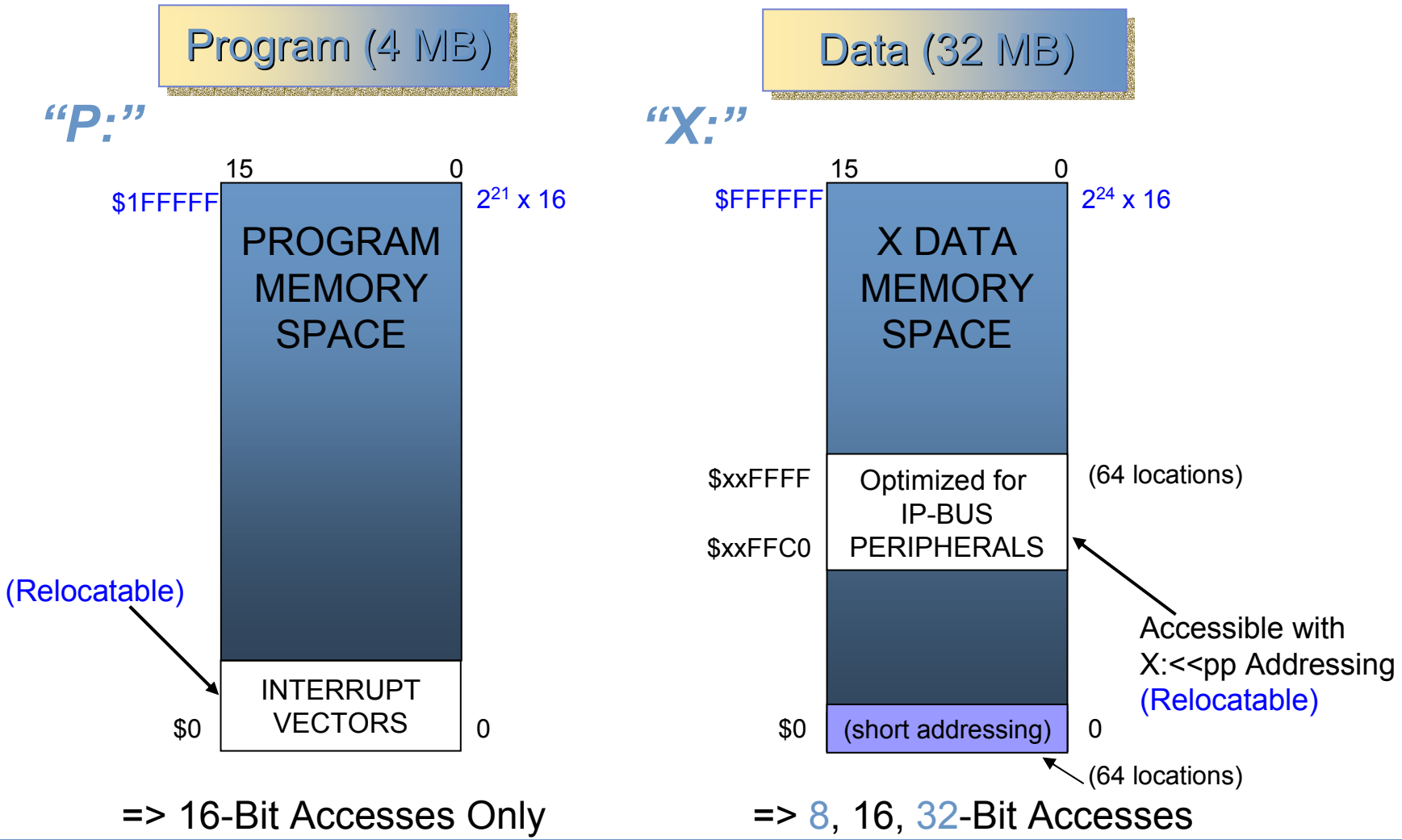
Improved Multitasking Support

Optimized Power Management

Efficient Peripheral Interfacing through Freescale's IP- BUS

Efficient Memory Interfacing

Program and Data Memories



DSP56800E General Purpose Registers

DATA ARITHMETIC LOGIC UNIT

DATA REGISTERS

	35 32	31 16	15 0
A	A2	A1	A0
B	B2	B1	B0
C	C2	C1	C0
D	D2	D1	D0

Y	Y1
	Y0
	X0

=> **7 Data Registers**

PROGRAM CONTROL UNIT

20 0
PC

PROGRAM COUNTER

15 0
OMR
SR

OPERATING MODE and STATUS

ADDRESS GENERATION UNIT

23 0
R0
R1
R2
R3
R4
R5
N
SP

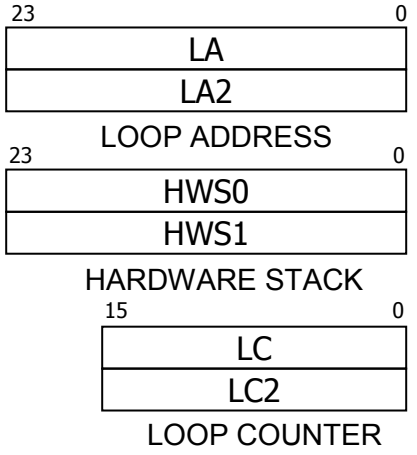
POINTER REGISTERS

=> **8 Address Registers**

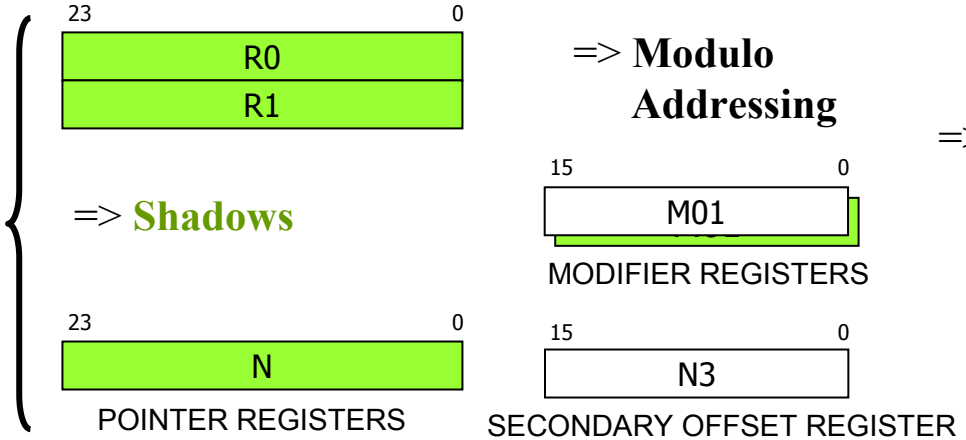
Registers with Dedicated Functionality

PROGRAM CONTROL UNIT

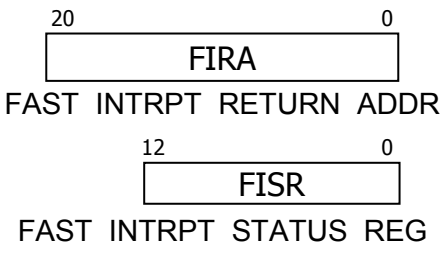
=> HW Looping
Nested 2 Deep



ADDRESS GENERATION UNIT



=> Fast
Interrupt



DSP56800E Programming Model

DATA ARITHMETIC LOGIC UNIT

DATA REGISTERS

	35 32	31 16	15 0
A	A2	A1	A0
B	B2	B1	B0
C	C2	C1	C0
D	D2	D1	D0

Y	Y1
	Y0
	X0

ADDRESS GENERATION UNIT

23 0

R0
R1
R2
R3
R4
R5

N

SP

POINTER REGISTERS

==> R0, R1, N, and M01 registers are shadowed

15 0

M01

MODIFIER REGISTERS

15 0

N3

SECONDARY OFFSET REGISTER

PROGRAM CONTROL UNIT

20 0

PC

PROGRAM COUNTER

15 0

OMR
SR

OPERATING MODE and STATUS

23 0

LA
LA2

LOOP ADDRESS

23 0

HWS0
HWS1

HARDWARE STACK

15 0

LC
LC2

LOOP COUNTER

20 0

FIRA

FAST INTERRUPT RETURN ADDRESS

12 0

FISR

FAST INTERRUPT STATUS REGISTER

Data Movement on the DSP56800E

Support of Compiler Data Types

- Longs (32-bits)
- Words (16-bits)
- Bytes (8-bits)

Parameters:

- 8, 16, or 32-bits
- Read or Write Operation
- Signed or Unsigned
- Addressing Mode

==> See the Instruction Set Summary for a Complete Set of Move Tables

Programming Model Comparison: ’800 vs ’800E

New for 800E

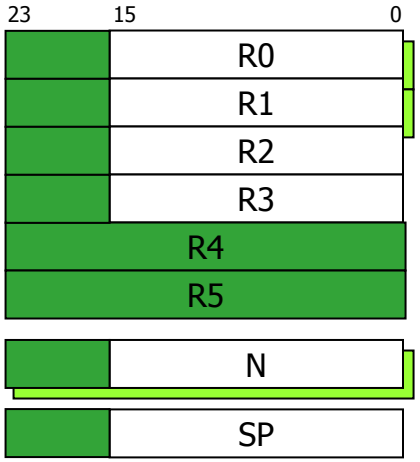
DATA ARITHMETIC LOGIC UNIT

DATA REGISTERS

	35 32	31 16	15 0
A	A2	A1	A0
B	B2	B1	B0
C	C2	C1	C0
D	D2	D1	D0

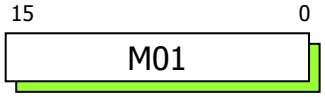
Y	Y1
	Y0
	X0

ADDRESS GENERATION UNIT

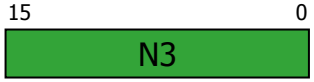


POINTER REGISTERS

==> R0, R1, N, and M01 registers are shadowed

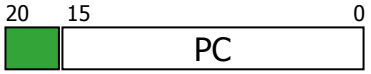


MODIFIER REGISTERS

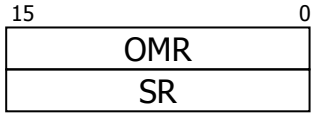


SECONDARY OFFSET REGISTER

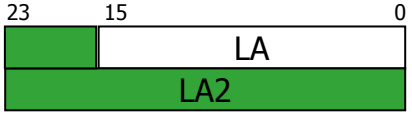
PROGRAM CONTROL UNIT



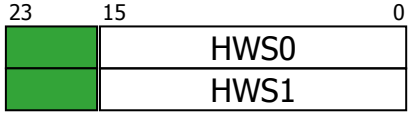
PROGRAM COUNTER



OPERATING MODE and STATUS



LOOP ADDRESS



HARDWARE STACK

LOOP COUNTER

FAST INTERRUPT RETURN ADDRESS

FAST INTERRUPT STATUS REGISTER

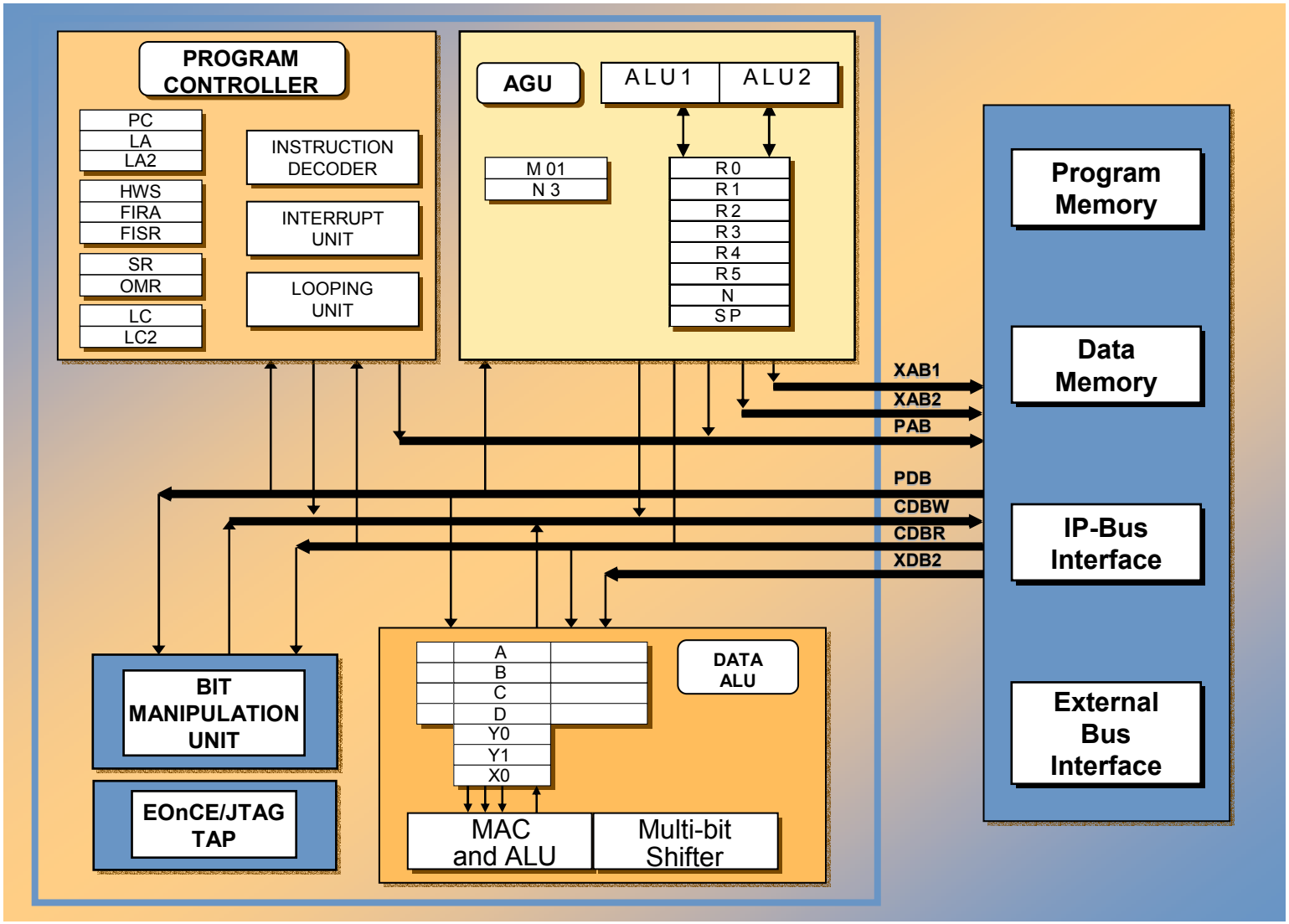
DSP56800E Improvement Summary

CPU	MIPS	Clocks perInstr	# Interrupt Levels	Registers	Data Types	Program Memory Adr Space	Data Memory Adr Space	Technology
DSP56800	20-40	2	2	5 Data 5 Address	16-bit	128 KB	128 KB	Semi-custom
DSP56800E	120-200	1	4	7 Data 8 Address	8-bit, 16-bit 32-bit	4 MB	32 MB	Fully Synthesizable & Scanable

Other Improvements

- Compiler Efficiency
- Real-time Debug
- Fast Interrupt
- Nested Hardware Looping
- Additional Addressing Modes
- Five (5) Software Interrupt Traps

DSP56800E Core Architecture



Instruction Fetch:

PAB - 21 bits
PDB - 16 bits

1st Data Access:

XAB1 - 24 bits
CDBR - 32 bits

2nd Data Access:

XAB2 - 24 bits
XDB2 - 16 bits

Operations Performed:

1st - PAB / PDB
2nd - XAB1 / CDBR-CDBW
3rd - XAB2 / XDB2

Launched by Motorola
freescale semiconductor

DSP56800E Parallel Move Instructions

Definition: One or two word sized moves occur in parallel with an arithmetic operation

The "Single Parallel Move"

```
ADD X0,A        Y0,X:(R1)+N
```

Arithmetic Operation | 16-Bit Move

Examples:

MPYR	X0,Y0,A	X:(R0)+,X0
MAC	-X0,Y0,A	Y0,X:(R1)+N
ADD	A,B	Y1,X:(R2)+
TFR	Y1,A	A,X:(R3)+
INC.W	A	X:(R0)+,B
ASL	A	X:(R1)+N,C

The "Dual Parallel Read"

```
MACR  X0,Y0,A    X:(R0)+N,Y1    X:(R3)+N3,X0
```

Arithmetic Operation | 1st 16-Bit Read | 2nd 16-bit Read

Examples:

MPY X0,Y0,A	X:(R0)+,Y0	X:(R3)+,X0
MACR X0,Y0,B	X:(R1)+,Y1	X:(R3)-,C
ADD X0,A	X:(R4)+N,Y0	X:(R3)+,X0
SUB Y1,B	X:(R4)+N,Y1	X:(R3)+N3,X0
MOVE.W	X:(R0)+N,Y0	X:(R3)+,C

Demonstration using Parallel Instructions

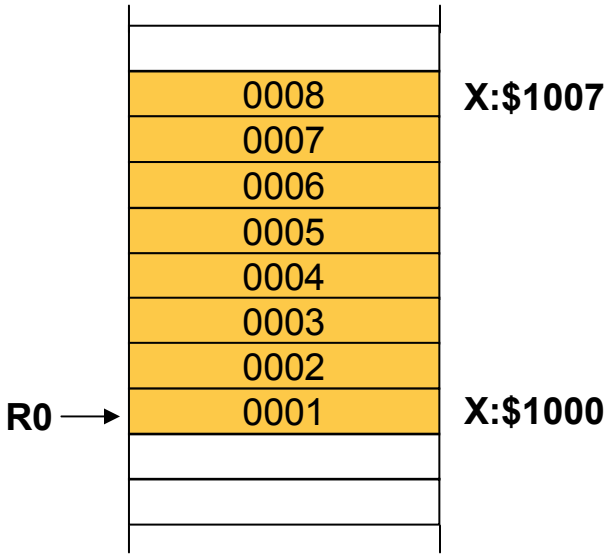
Initialize array of size 8 - direct addressing

Initialize pointer register R0: $1000

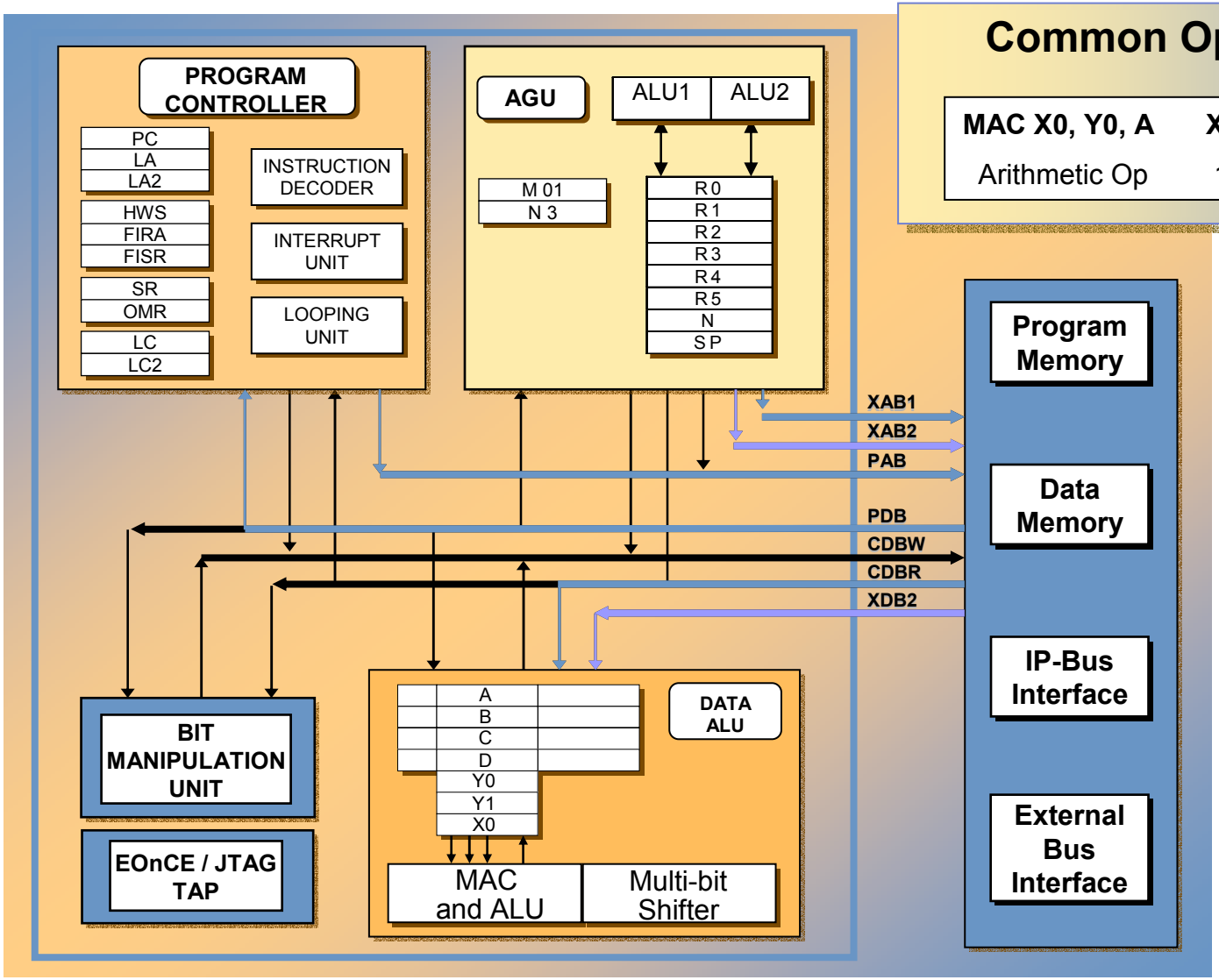
Clear accumulator A and read 1st value

Add value to A, while reading new value

Multiple right shift to generate average



Mapping the Architecture to DSP Algorithms



Common Operation in DSP

MAC X0, Y0, A	X:(R4)+, Y1	X:(R3)+, C
Arithmetic Op	1st Read	2nd Read

Operations Performed:

- Multiply-Accumulate
- 3 Memory Accesses
- 2 Address Additions

Instruction Fetch:

PAB - 21 bits
PDB - 16 bits

1st Data Access:

XAB1 - 24 bits
CDBR - 32 bits

2nd Data Access:

XAB2 - 24 bits
XDB2 - 16 bits

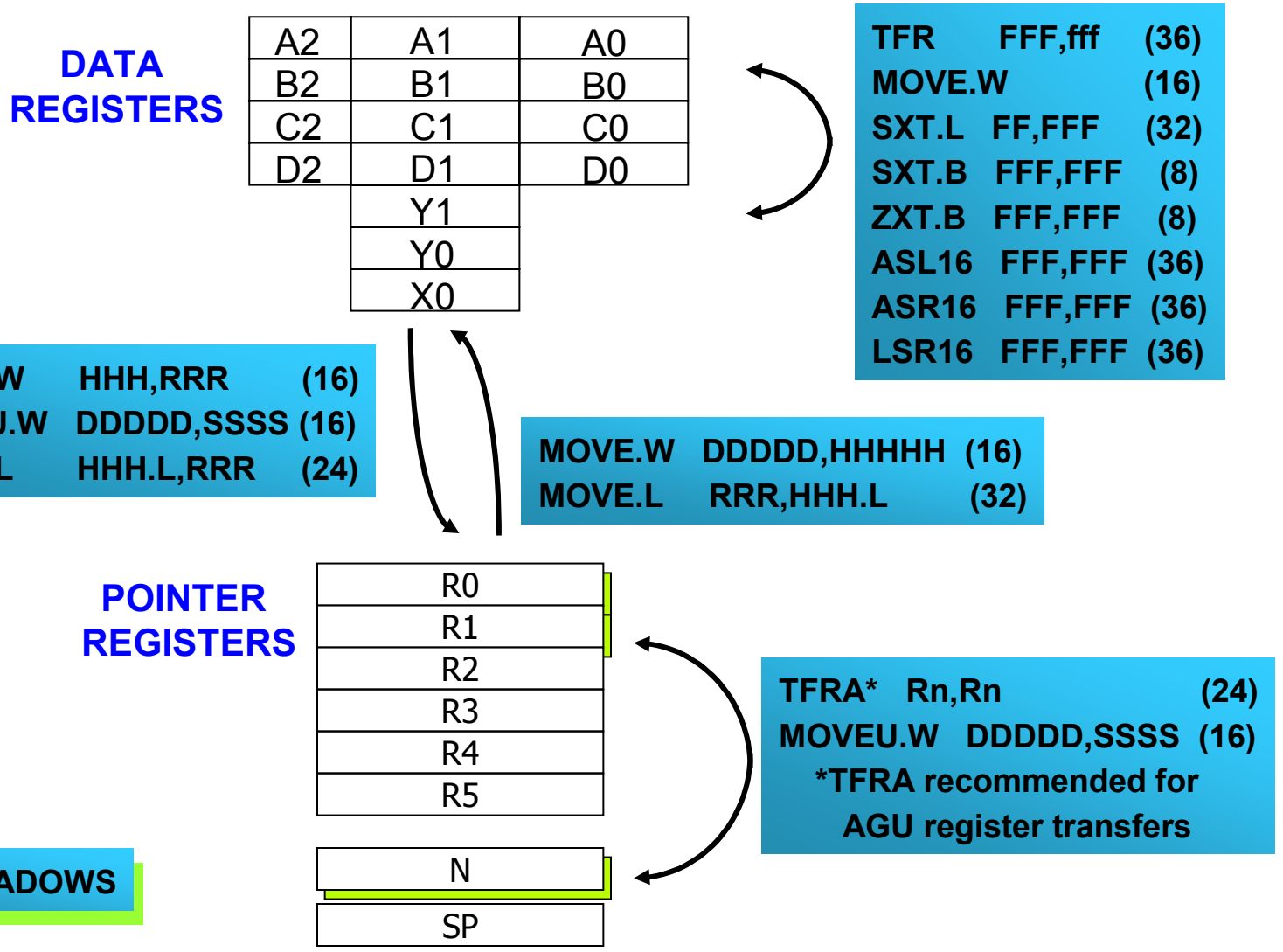
Single Parallel Move Instructions

Data ALU Operation		Parallel Memory Move	
Operation	Operands	Source	Destination
MAC MPY MACR MPYR	Y1,X0,F Y0,X0,F Y1,Y0,F Y0,Y0,F A1,Y0,F B1,Y1,F	$X:(R_j)+$ $X:(R_j)+N$	X0 Y1 Y0 A B C A1 B1
MAC MPY MACR	C1,Y0,F C1,Y1,F		
MAC	–C1,Y0,F –C1,Y1,F		
ADD SUB CMP TFR	X0,F Y1,F Y0,F C,F A,B B,A	X0 Y1 Y0 A B C A1 B1	$X:(R_j)+$ $X:(R_j)+N$
SAT	F,Y0		
EOR.L	C,F		
ABS ASL ASR CLR RND TST INC.W DEC.W NEG	F		
SUBL[1]	A,D,B	X:(R1)+	AD

Dual Parallel Read Instructions

Data ALU Operation		First Memory Read		Second Memory Read	
Operation	Operands	Source 1	Destination1	Source 2	Destination 2
MAC MPY MACR MPYR	Y1,X0,F Y1,Y0,F Y0,X0,F C1,Y0,F	X:(R0)+ X:(R0)+N X:(R1)+ X:(R1)+N	Y0 Y1	X:(R3)+ X:(R3)–	X0
		X:(R4)+ X:(R4)+N	Y0	X:(R3)+ X:(R3)+N3	X0
ADD SUB	X0,F Y1,F Y0,F A,B B,A	X(R0)+ X(R0)+N X(R4)+ X(R4)+N	Y1	X(R3)+ X(R3)+N3	C
TFR	A,B B,A				
CLR ASL ASR	F				
MOVE.W					

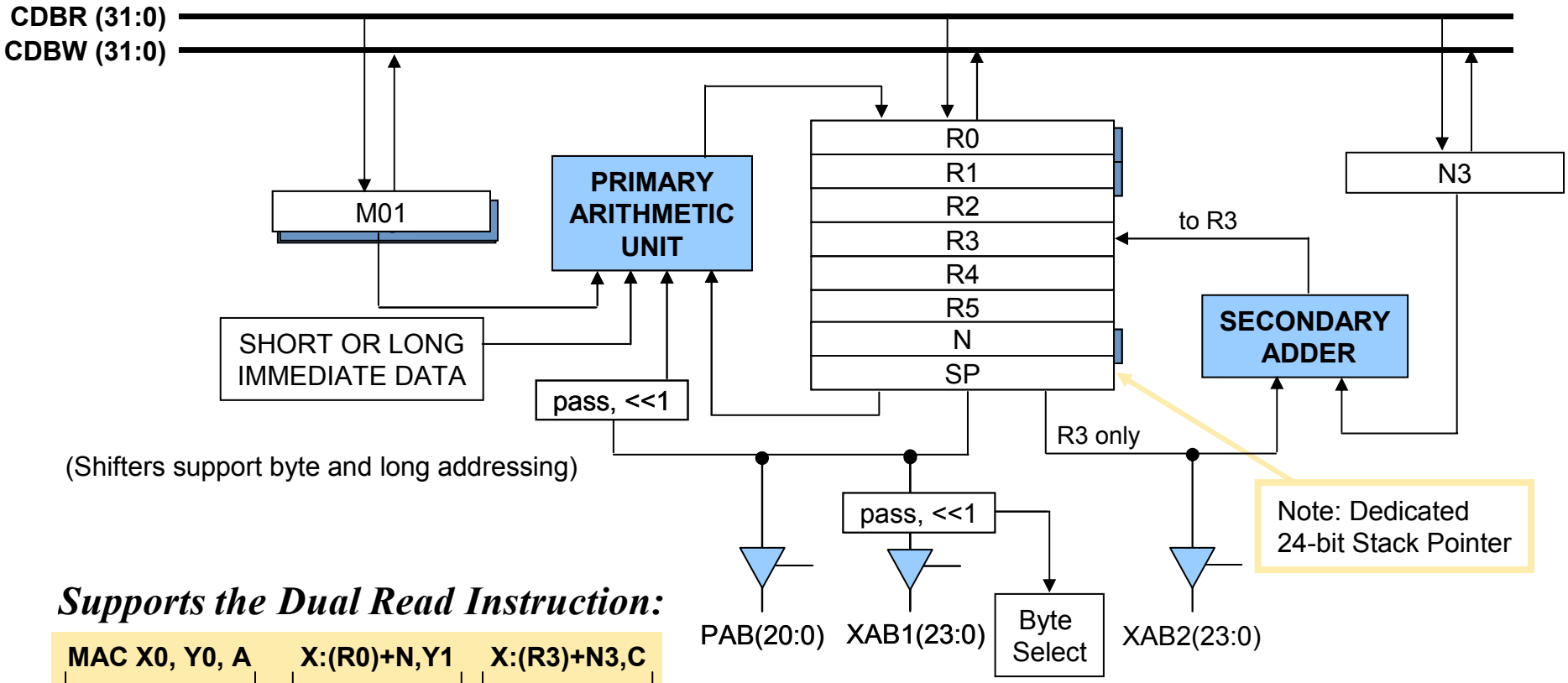
Powerful Set of Register to Register Moves



AGU Block Diagram

Provides up to 2 Data Memory Addresses per cycle

Performs up to 2 Address Calculations after Issuing Addresses



Supports the Dual Read Instruction:

MAC X0, Y0, A	X:(R0)+N,Y1	X:(R3)+N3,C
Data ALU	Uses XAB1	Uses XAB2

Powerful Set of Addressing Modes

Indirect

- X:(Rn) No Update
- X:(Rn)+ Post Increment
- X:(Rn)- Post Decrement
- X:(Rn)+N Post Update by Register

Indexed

- X:(Rn+x) Indexed:3-bit Offset
- X:(SP-xx) Indexed:6-bit Offset
- X:(Rn+xxxx) Indexed:16-bit Offset
- X:(Rn+xxxxxx) Indexed:24-bit Offset
- X:(Rn+N) Indexed: By a Register

Supports 8, 16, 32-bits

Supports Modulo Arithmetic

Immediate

- #x 5-bit “Long” Constant
- #xx 6-bit Loop Ct
- #xx 7-bit Short
- #xxxx 16-bit
- #xxxxxxxx 32-bit

Absolute

- X:aa 6-bit Absolute Short
- X:<<pp 6-bit Peripheral Direct
- X:xxxx 16-bit Absolute
- X:xxxxxx 24-bit Absolute

Other

- DDDDD Register Direct
- * Inherent

Examples of Addressing Modes

Addressing Modes for Move Instr.

```
move.w    <reg>,<reg>
move.w    #<-64,63>,<reg>
move.w    #xxxx,<reg>
move.w    X:xxxx,<reg>
move.w    X:xxxxxx,<reg>
move.w    X:(Rn),<reg>
move.w    X:(Rn)+,<reg>
move.w    X:(Rn)-,<reg>
move.w    X:(Rn+N),<reg>
move.w    X:(Rn)+N,<reg>
move.w    X:(Rn+xxxx),<reg>
move.w    X:(Rn+xxxxxx),<reg>
move.w    X:(SP-xx),<reg>
move.w    X:<<pp,<reg>
move.w    X:aa,<reg>
move.w    <reg>,X:(SP-xx)
move.w    <reg>,X:xxxx
move.w    <reg>,X:(Rn)
move.w    <reg>,X:(Rn)+
move.w    <reg>,X:(Rn)-
move.w    <reg>,X:(Rn+N)
move.w    <reg>,X:(Rn+xxxx)
.....     and many more !
```

Addressing Modes for ADD Instr.

```
add       <reg>,<reg>
add.w     #<0-31>,<reg>
add.w     #xxxx,<reg>
add.w     X:xxxx,<reg>
add.w     X:xxxxxx,<reg>
add.w     X:(Rn),<reg>
add.w     X:(Rn+xxxx),<reg>
add.w     X:(SP-xx),<reg>
add.w     <reg>,X:(SP-xx)
add.w     <reg>,X:xxxx
.....     and many more !
```

Launched by Motorola freescale semiconductor

Enhanced Set of AGU Arithmetic Instructions

Arithmetic:

- *ADDA*
- ADDA.L
- CMPA
- CMPA.W
- *DECA*
- DECA.L
- DECTSTA
- NEGA
- SUBA
- SXTA.B
- SXTA.W
- TSTA.B
- TSTA.W
- TSTA.L
- *TSTDECA.W*
- ZXTA.B
- ZXTA.W

Shifting and Moves:

- ASLA
- ASRA
- LSRA
- TFRA

AGU Instructions in 56800:

- ADDA == LEA (incr)
- DECA == LEA (decr)
- TSTDECA.W == TSTW (Rn)-

Structured Programming - The Software Stack

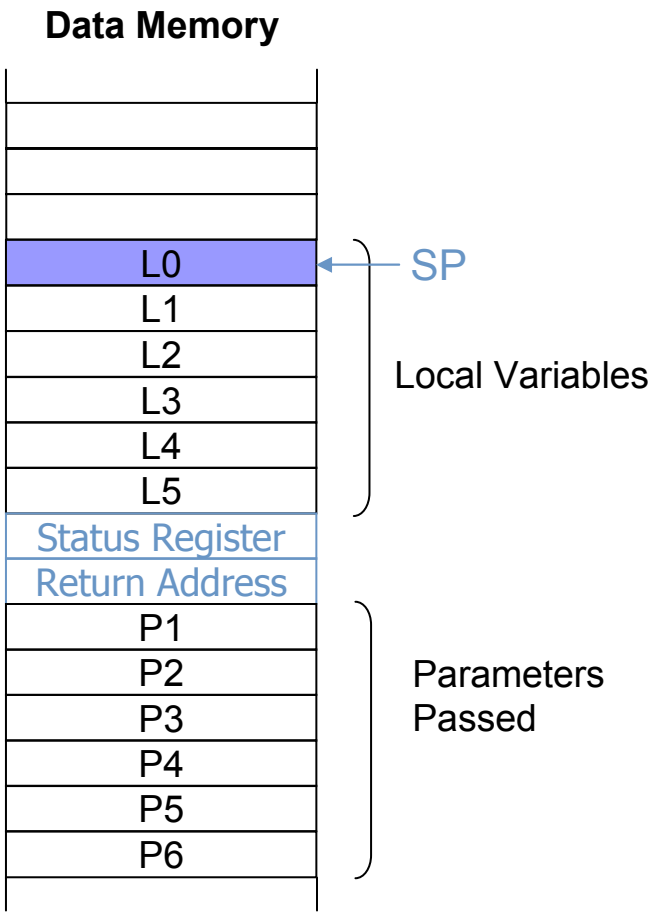
Software Stack support for structured programming

Supports Local Variables

Supports Parameter Passing to a Function

For both C and Assembly Code

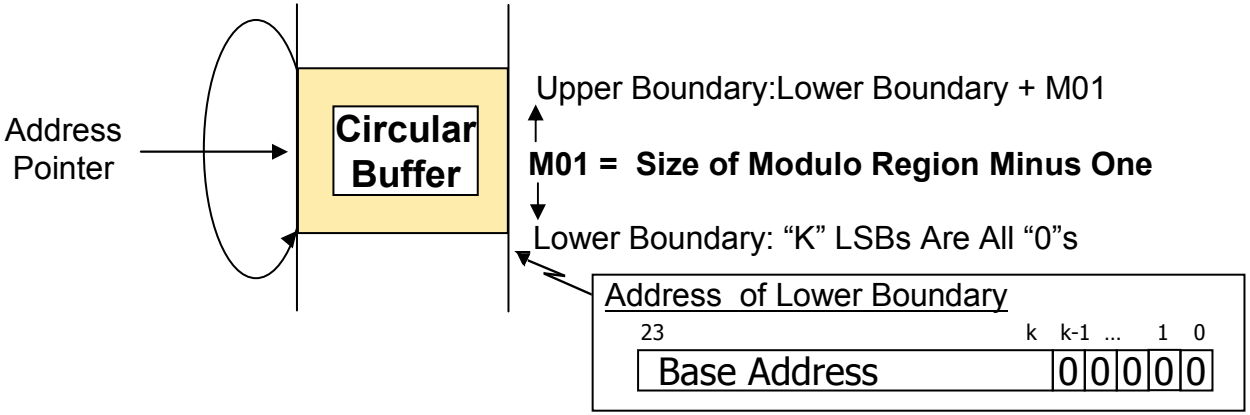
Utilizes strong set of SP Addressing Modes



Example: Local Variable "L5" accessed as X:(SP-5)

Also Note:
JSR and interrupts automatically stack PC and SR

AGU Modulo Addressing



Modulo Addressing Features:

- **Available for byte, word, and long accesses**
- **Available for the R0 and R1 pointers only. M01[15] = 1, then modulo on R0 & R1**
- **Occurs only when address arithmetic is performed to calculate effective address**
- **Supports buffer sizes from 2 locations to 16384 words (2 to 8192 for long values)**

The equations for modulo addressing are:

- **R0[23:k] = R0[23:k] (not modified)**
- **R0[k-1:0] = (R0[k-1:0] + offset) MOD (M01 + 1)**

Demonstration using Modulo and Linear Addressing

Initialize 2 arrays with the value of address

Linear Array - X:$1020

Circular Array - X:$1000, 5 elements

- **M01 = $0004; R0 (modulo)**

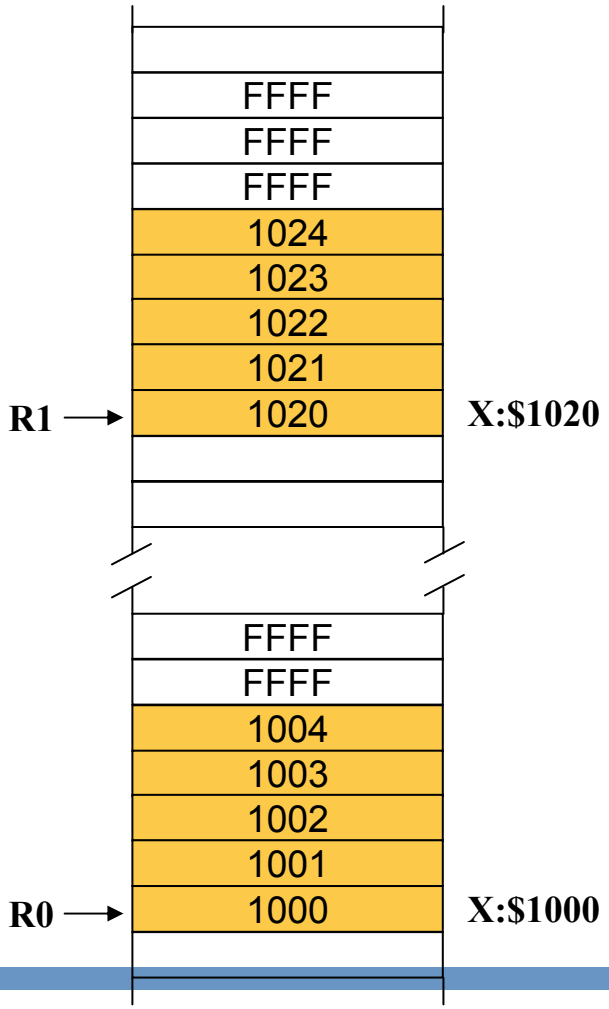
Initialize R0: start address of circular buffer

Initialize R1: start address of linear buffer

Write incrementing values to both arrays

R0 & R1 post-updated by N

After run: NEGA N, and repeat sequence



oke Metrowerks CodeWarrior

Modulo Arithmetic Example

Using the Modulo Buffer:

- **Used in post-update instructions: "MOVE.W X:(R0)+,X0"**
- **Used with AGU arithmetic instructions**
- **Example demonstrates usage; R0 = $000800:**

```
MOVE.W    X:(R0)+,X0  ; 1st time accesses location $0800
                      ; and bumps the pointer to location $0801
MOVE.W    X:(R0)+,X0  ; 2nd accesses at location $0801
MOVE.W    X:(R0)+,X0  ; 3rd accesses at location $0802
MOVE.W    X:(R0)+,X0  ; 4th accesses at location $0803
MOVE.W    X:(R0)+,X0  ; 5th accesses at location $0804
MOVE.W    X:(R0)+,X0  ; 6th accesses at location $0805
MOVE.W    X:(R0)+,X0  ; 7th accesses at location $0806
MOVE.W    X:(R0)+,X0  ; 8th accesses at location $0807
MOVE.W    X:(R0)+,X0  ; 9th accesses at location $0808
                      ; and bumps the pointer to location $0800

MOVE.W    X:(R0)+,X0  ; 10th accesses at location $0800
MOVE.W    X:(R0)+,X0  ; 11th accesses at ...
```

Example:

Buffer Size = 9

M01= $0008

R0: Modulo

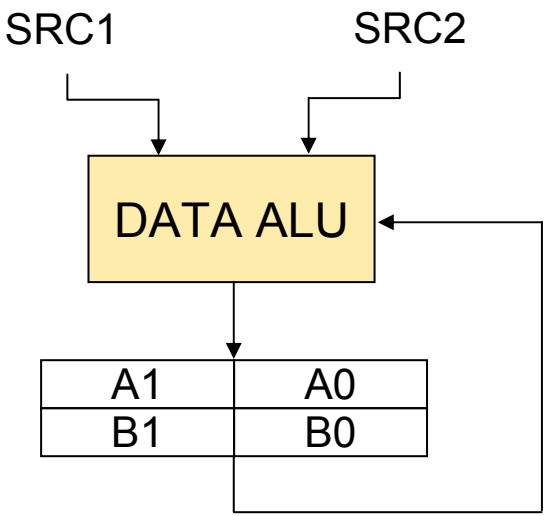
R1: Linear

Other Useful Features:

- **works for decrementing addressing modes too**
- **modulo operation works correctly even if the** pointer does not land exactly on upper or lower boundary
- **modulo buffer sizes are** not constrained to a power of two

Data ALU - General Purpose Register File

Conventional DSP

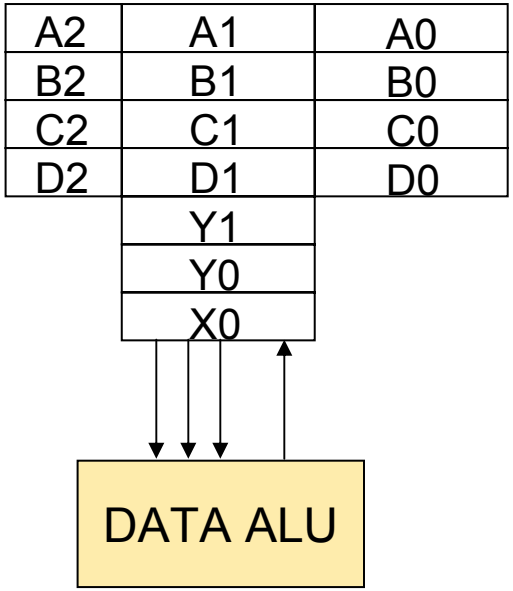


INC,DEC [A or B]
ASL,ASR

ADD, etc. [SRC1], [A or B]

"Accumulator Based"

DSP56800E

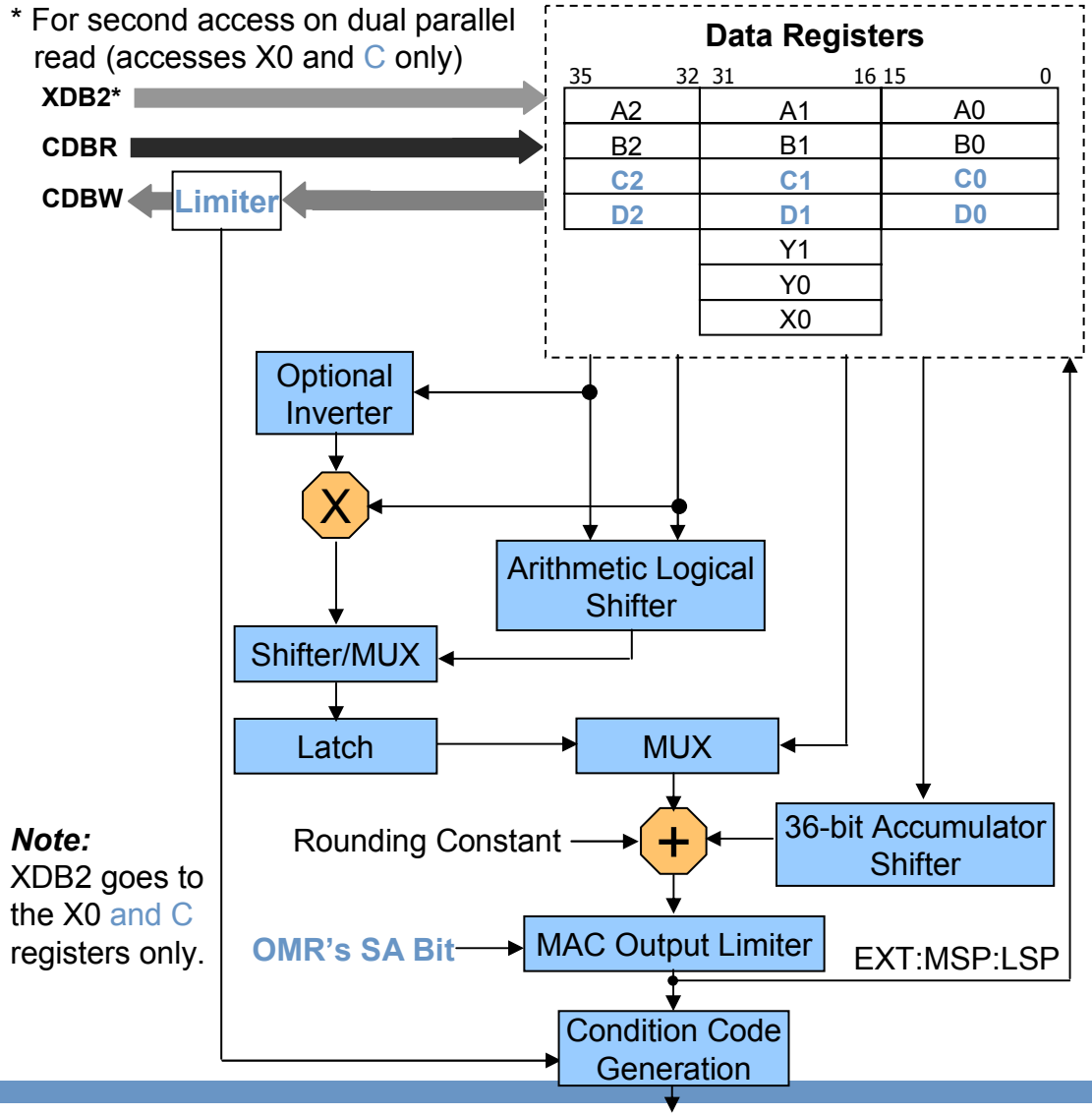


INC.W, DEC.W [FFF]
ASL, ASR

ADD, etc. [FFF], [FFF]

"GP Register File"

Data ALU Block Diagram

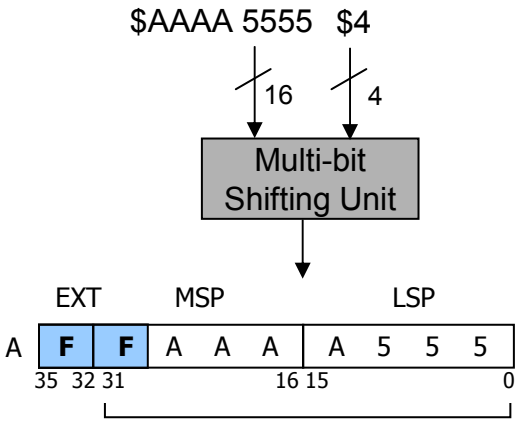


Capabilities

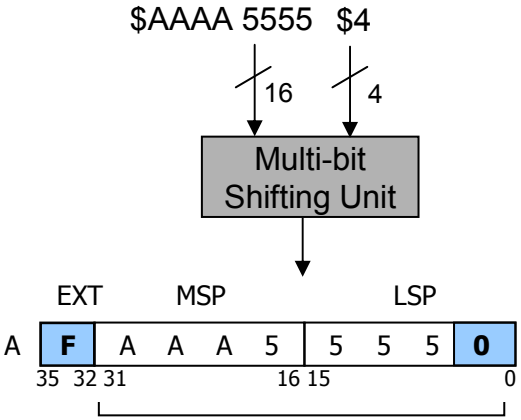
- **Multiplication (w/ Rounding)**
- **Multiply-Accumulate (w/ Rounding)**
- **Multiprecision Multiplication Support**
- **Addition and Subtraction**
- **Increments and Decrements**
- **Tests and Compares (8, 16, 32, 36 bits)**
- **16 and 32-bit Logical Operations**
- **1's and 2's complement**
- **Single Bit Arithmetic & Logical Shifts**
- **16-bit Arithmetic and Logical Shifts**
- **32-bit Arithmetic and Logical Shifts**
- **Single Bit Rotates**
- **Rounding**
- **Absolute Value**
- **Sign/Zero Extension**
- **Limiting on Move Instructions**
- **Conditional Register Transfer**
- **Division Iteration Instruction**
- **Count Leading Bits**
- **Normalization**

Shifting 32-Bit Long Words (Bidirectional)

EXAMPLE - RIGHT SHIFTING: ASRR.L #4,A



EXAMPLE - LEFT SHIFTING: ASLL.L #4,A



Operation	Operands	C	W	Comments
ASLL.L	#<0-31>,fff	2	1	arithmetic shift left by a 5-bit positive immediate integer
	EEE,FFF	2	1	**Bi-directional** arithmetic shift of destination by value in the first operand: positive -> left shift.
ASRR.L	#<0-31>,fff	2	1	arithmetic shift right by a 5-bit positive immediate integer
	EEE,FFF	2	1	**Bi-directional** arithmetic shift of destination by value in the first operand: positive -> right shift.
LSRR.L	#<0-31>,fff	2	1	logical shift right by a 5-bit positive immediate integer
	EEE,FFF	2	1	**Bi-directional** logical shift of destination by value in the first operand: positive -> right shift

Normalization

Normalization — left justify number, removing redundant sign bits

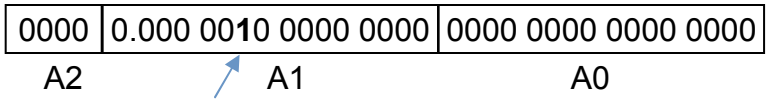
Performed in two instructions:

```
; Normalization Algorithm — 16-Bits
      CLB       A,X0
      ASLL.W    X0,A
```

```
; Normalization Algorithm — 32-Bits
      CLB       A,X0
      ASLL.L    X0,A
```

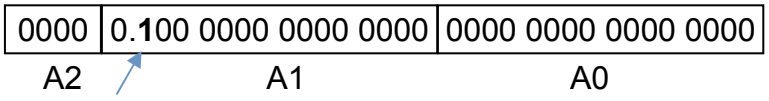
Example 1 - Normalization of a Positive Value

Before Execution: A = $0:0200:0000



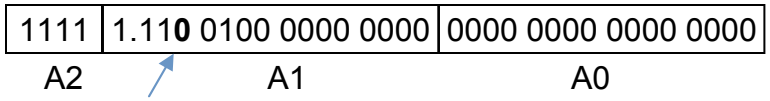
<< 5

After Execution: A = $0:4000:0000



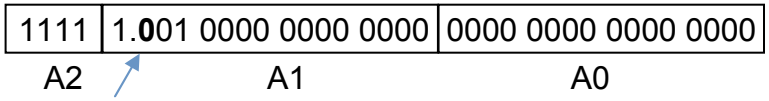
Example 2 - Normalization of a Negative Value

Before Execution: A = $F:E400:0000



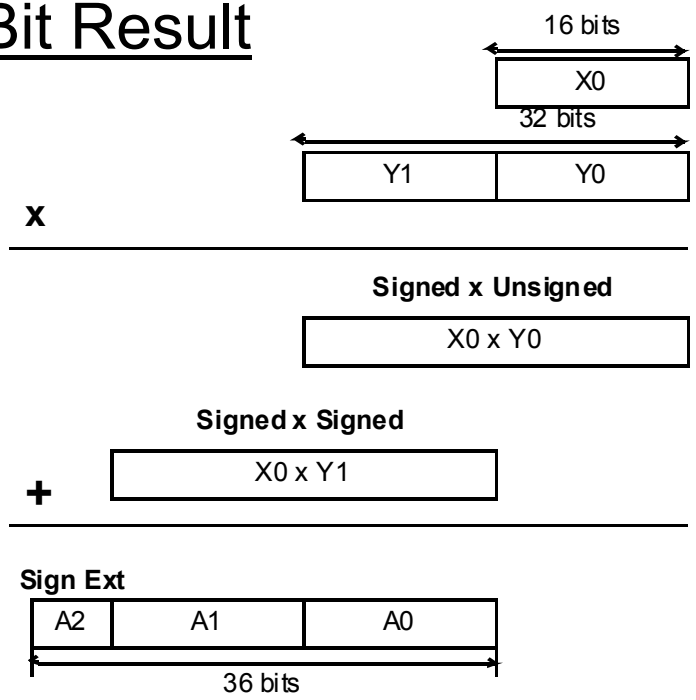
<< 2

After Execution: A = $F:9000:0000



==> CLB Instruction also useful for finding 1st significant bit

Fractional 16 x 32=> 36-Bit Result



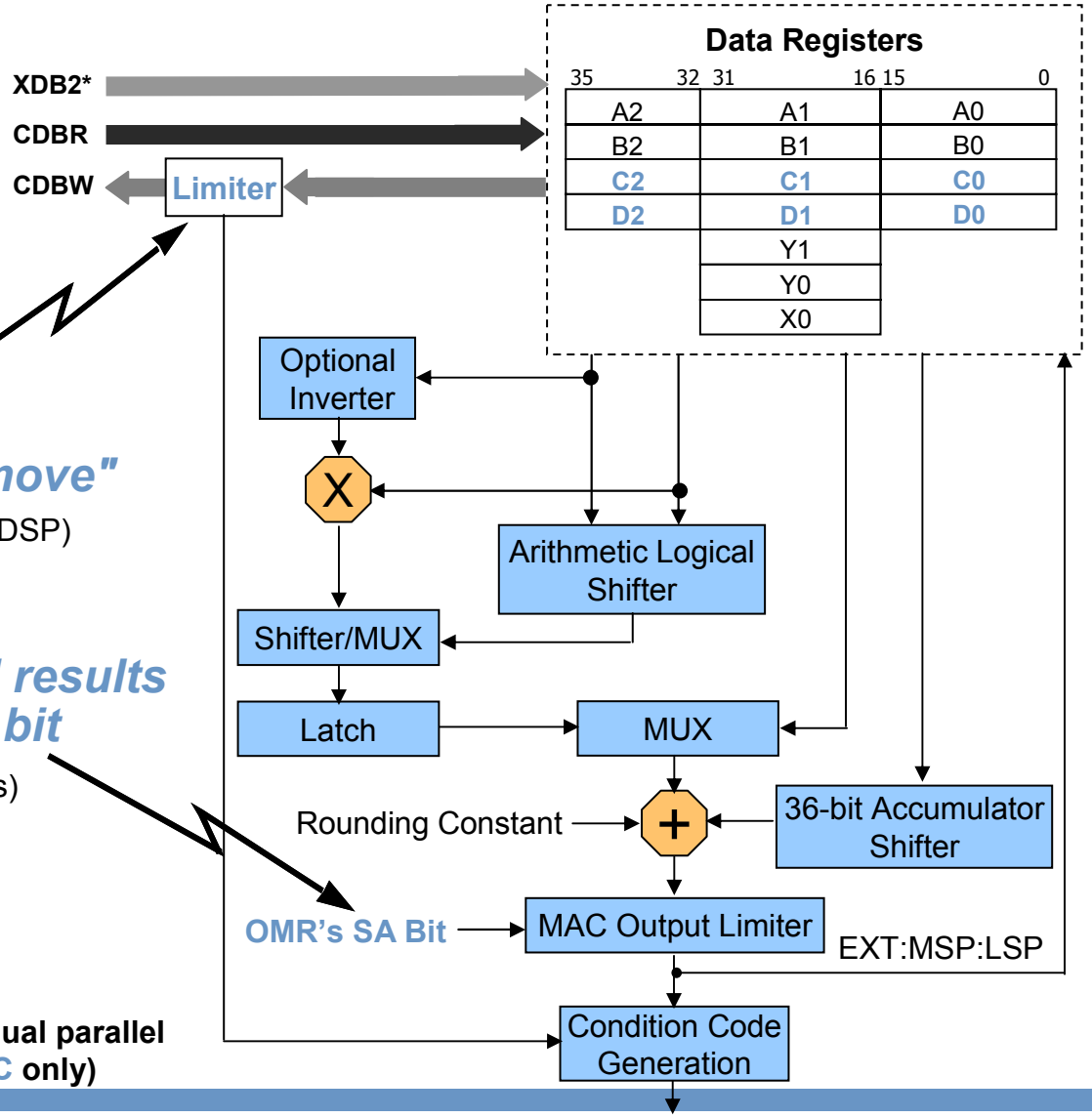
3 Words, 3 Cycles

16 x 32 Bit Fractional Multiplication — 36-Bit Result

FF2:FF1:FF0 = X0 x Y1:Y0
(Both Fractional Operands are Signed; 3 Cycles, 3 Words)

```
;Signed 16-Bit x Signed 32-Bit Fractional Multiplication
        MPYSU   X0,Y0,A      ; Y1:Y0 = signed X0 x unsigned Y0
        ASR16   A            ; Align 1st product
        MAC     X0,Y1,A      ; A2:A1:A0 = final 36-bit result
```

Two Supported Saturation Mechanisms



Saturation via "move"

(most typically used for DSP)

Saturate all Data ALU results via OMR control bit

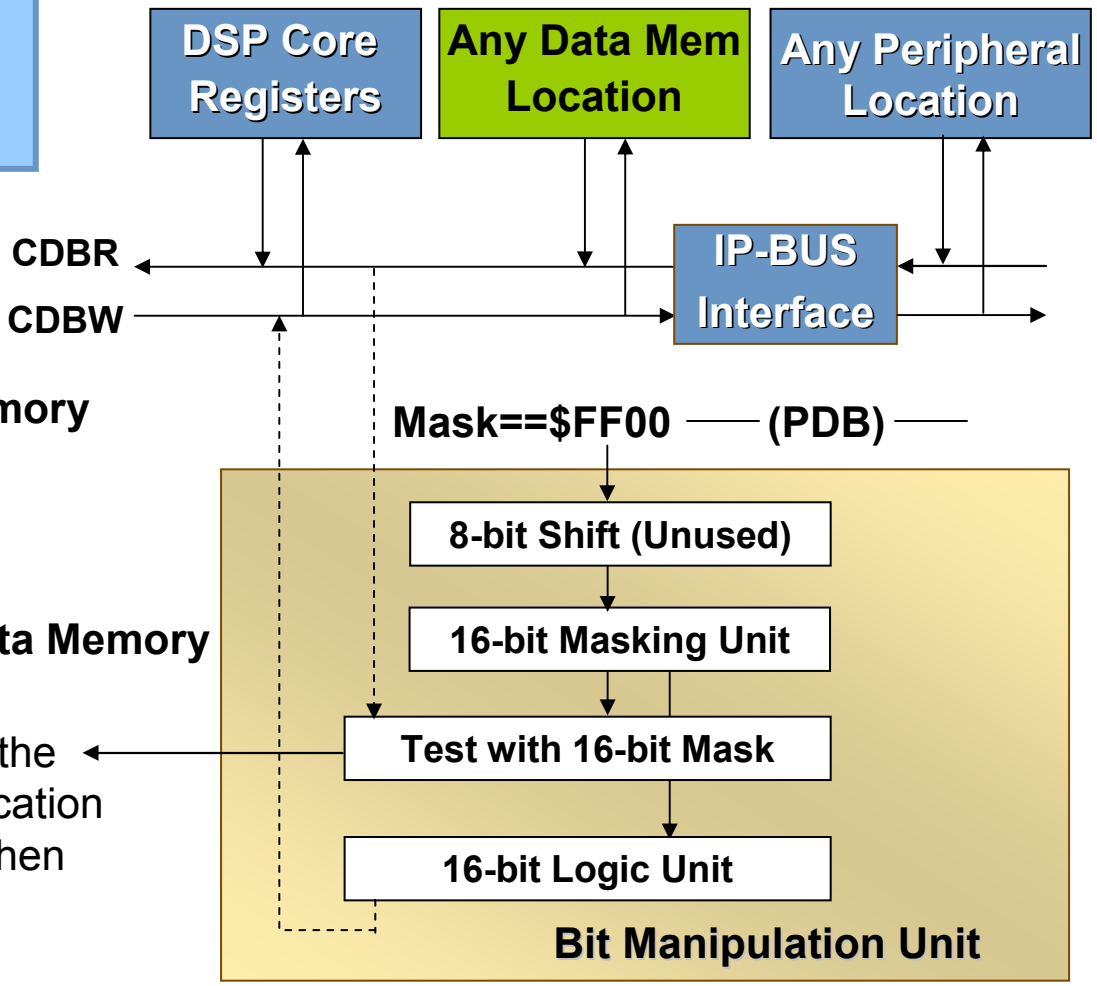
(for bit-exact applications)

*** For second access on dual parallel read (accesses X0 and C only)**

Flexible Bit Manipulation Instructions

BFCLR	#$FF00	X:(R0)
Operation (clear bits)	16-Bit Mask	Operand in Memory

OPCODE: 8040 FF00



Steps

1. **Read 16-bit Word from Data Memory**
2. **Test Masked (upper 8) Bits**
3. **Clear Masked (upper 8) Bits**
4. **Write modified Word back to Data Memory**

Carry bit set to “1” if all bits in the Upper Byte of the Memory Location were all “1”’s; otherwise “0”. Then clears all selected bits.

Summary: Bit Manipulation Capabilities

Strong Set of Bit Manipulation Instructions:

- BFSET: **Test and then set a field of bits in a word**
- BFCLR: **Test and then clear a field of bits in a word**
- BFCHG: **Test and then invert a field of bits in a word**
- BFTSTH: **Test a field of bits for all "1"s**
- BFTSTL: **Test a field of bits for all "0"s**
- BRSET: **Branch if a selected set of bits are all "1"s**
- BRCLR: **Branch if a selected set of bits are all "0"s**

AND, OR, and XOR w/ 16-bit values

Operates on any register or data memory location on the chip

Operates using a 16-bit mask

- **Except:** BRSET **and** BRCLR **only allow an 8-bit mask on upper or lower byte**

Other Bit Manipulation Instructions Performed Only in the Data ALU:

- **16-Bit Bidirectional Multi-bit Shifting (uses Data ALU registers)**
- **Arithmetic and Logical Shifts (uses Data ALU registers)**
- **Rotates (uses Data ALU registers)**
- **Increment and Decrement of Memory Locations**

Demonstration using Read-Modify-Write

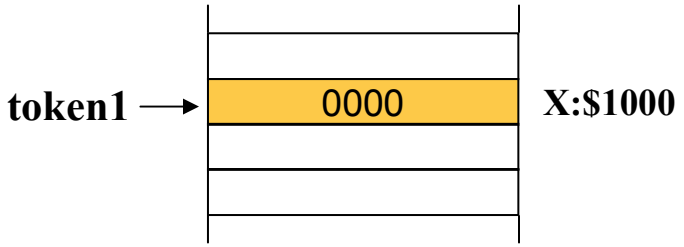
C program: Initialize token1 as "Available"

ASM program: read status of token1, return 1 if busy

- **token1: 0001 == TAKEN**
- **token1: 0000 == AVAILABLE**

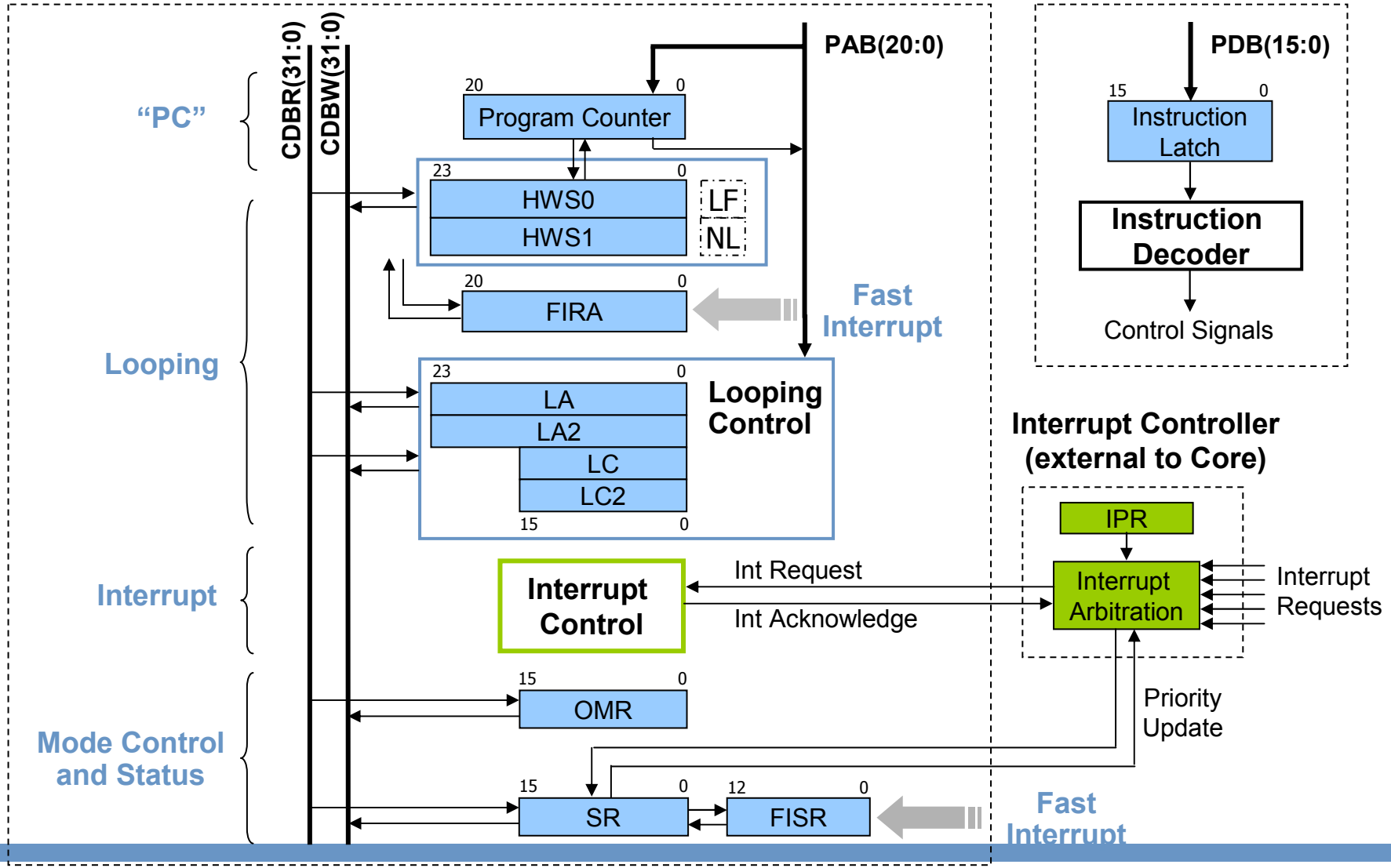
On iteration 5, clear status of token1

Recapture token1: read-modify-write in atomic (non-interruptible) sequence



oke Metrowerks™ CodeWarrior™

Program Controller Block Diagram



Nested Hardware Looping

Example - DSP56800E Assembly Code:

```
    CLR.W    A

    DO      #2,END_OUTR          ; Outer DO Loop
     DO      #20,END_INNR        ; Inner DO Loop
       MOVE.W   X:(R0)+,A        ;      (body of innermost loop)
       MOVE.W   X:(R1)+,B        ;      (body of innermost loop)
       ADD                       ;      (body of innermost loop)
       MOVE.W   A,X:(R3)+        ;      (body of innermost loop)
 END_INNR
      MOVE.W   A,X:(R5)+         ;
END_OUTR
```

Theoretical Best:
2*20*4 instrs = 160 Cycles

40 Loop Iterations in 172 Cycles
10 Program Words

Demonstration using Nested Hardware Looping

Initialize three 32-bit locations

Inner loop repeats 4 times

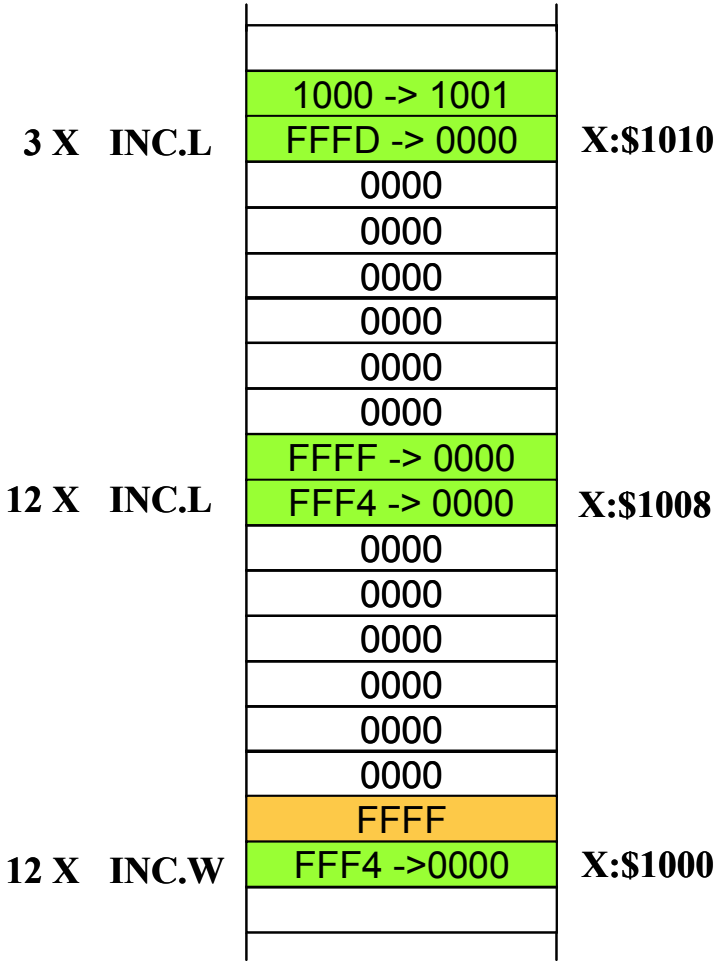
Outer loop repeats 3 times

Use INC.W & INC.L to generate:

- **$1001 0000 @ X:$1010**
- **$0000 0000 @ X:$1008**
- **$0000 @ X:$1000**

```
DO  #3,OUTER_LOOP
    DO  #4,INNER_LOOP
        INC.W X:$1000        ; incr short mem word
        INC.L  X:$1008       ; incr  long mem word
INNER_LOOP:
    INC.L X:$1010            ; incr  long mem word
OUTER_LOOP:
```

oke Metrowerks CodeWarrior



Standard Interrupt Processing

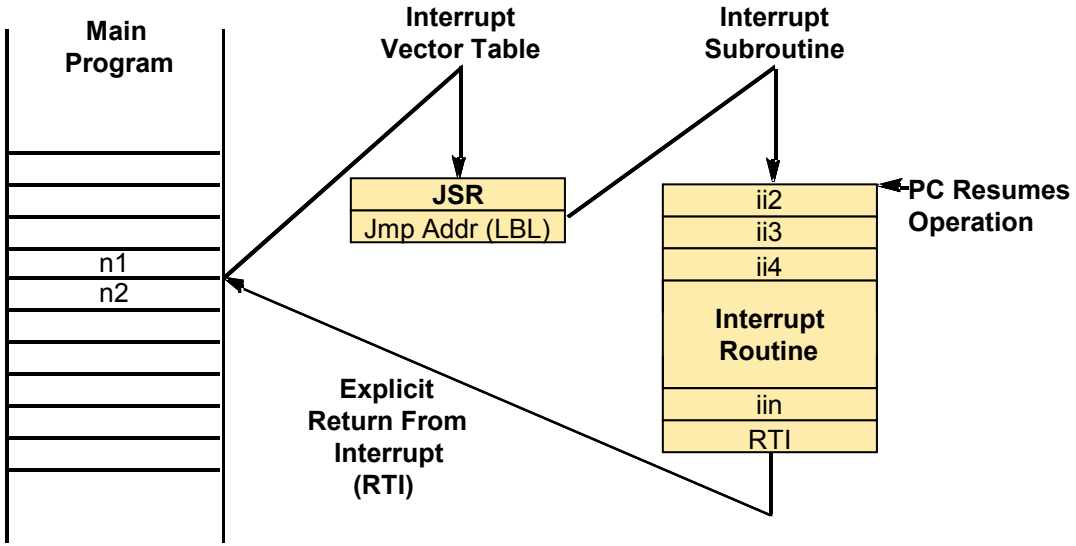
General Case:

Vectored Interrupts - Vectors may be located anywhere in Program Memory

4 Priority Levels - Highest is non-maskable

Software Traps at each priority level

One additional software trap (5th level) at lowest priority for O/S support



Interrupt Priority Structure

Interrupt Priority Level Summary

IPL	Description	Priority	Interrupt Sources
LP	**Maskable**	**Lowest**	**SWILP Instruction**
0	**Maskable**	*	**On-chip peripherals, IRQA and IRQB, SWI #0 Instruction**
1	**Maskable**	*	**On-chip peripherals, IRQA and IRQB, SWI #1 Instruction**
2	**Maskable**	*	**On-chip peripherals, IRQA and IRQB, SWI #2 Instruction**
3	**Non-maskable**	**Highest**	**Illegal instruction, hardware stack overflow, SWI instruction, EOnCE Interrupts, misaligned data access**

Current Core Interrupt Priority Levels

I1	I0	CCPL*	Exceptions Accepted	Exceptions Masked	Comments
0	**0**	**0**	**IPL 0,1,2,3, and SWILP**	**None**	**This interrupt controller accepts any unmasked interrupt, including the SWILP**
0	**1**	**1**	**IPL 1,2,3**	**IPL 0 and SWILP**	**This interrupt controller accepts all non-maskable interrupts and any unmasked interrupts that are programmed at level 1 or 2**
1	**0**	**2**	**IPL 2,3**	**IPL 0 , 1 and SWILP**	**This interrupt controller accepts all non-maskable interrupts and any unmasked interrupts that are programmed at level 1**
1	**1**	**3**	**IPL 3**	**IPL 0, 1, 2 and SWILP**	**This interrupt controller only accepts all non-maskable interrupts**

*** CCPL:**Current Core Interrupt Priority Level

Launched by Motorola freescale semiconductor

Demonstration Using Standard Interrupt

Read value from fixed memory location

Write value to a 3-word circular buffer

Use standard interrupt handler

Rewrite handler using SWAP **instruction**

```
        org             P:
swi2ISR:
        SWAP            SHADOWS

        MOVEU.W         X(R0),N
        MOVE.W          N,X:(R1)+

        SWAP            SHADOWS

        RTI
```

oke Metrowerks CodeWarrior

```
        org             P:
swi2ISR:
        ADDA            #2,SP
        MOVE.L          R1,X:(SP)+
        MOVE.W          Y0,X:(SP-1)
        MOVE.W          M01,X:(SP)

        MOVEU.W         X:$100A,M01
        MOVE.L          X:$1008,R1

        MOVE.W          X:$1000,Y0
        MOVE.W          Y0,X:(R1)+
        MOVE.L          R1,X:$1008

        MOVEU.W         X:(SP),M01
        MOVE.W          X:(SP-1),Y0
        SUBA            #2,SP
        MOVE.L          X:(SP)-,R1

        RTI
```

The "Fast Interrupt"

Fast Interrupt Case (Improved latency and throughput):

Vectors directly to service routine

Operates at Interrupt Level 2 - Highest "maskable" priority

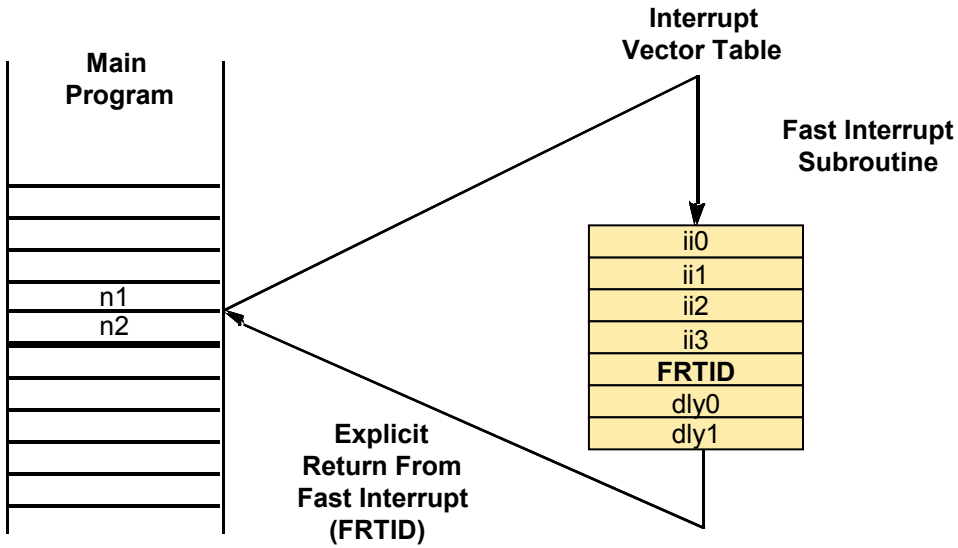
The Frozen PC is copied to FIRA, the status register and NL bit are copied to FISR

Automatically swaps registers with shadows: R0, R1, N, and M01

Automatically aligns SP and pushes the Y0 and Y1 registers onto the stack

Automatically advances the SP to an empty 32-bit location

Automatically restores above registers on exit, and restores original SP



Demonstration Using Fast Interrupt

```
; ============================================================
;  Interrupt Vock Metrowerks CodeWarrior
; ============================================================
InterruptVector                 ; starts at location $0000 Program Space

      JMP FSTART_               ;  0 at 0x0000  Hardware Reset
      JMP FSTART_               ;  1 at 0x0002  COP Reset
      JSR UnhandledInterrupt    ;  2 at 0x0004  Misaligned Long Word Access
      JSR UnhandledInterrupt    ;  3 at 0x0006  Illegal instruction
      JSR swiISR                ;  4 at 0x0008  SW interrupt 3
      JSR UnhandledInterrupt    ;  5 at 0x000A  HW Stack Overflow

; = FAST INTERRUPT HANDLER ==================================
      FRTID                     ;  6 at 0x000C  SW Interrupt 2
      MOVE.W X:(R0),Y0
      MOVE.W Y0,X:(R1)+
; ============================================================

      DC  $0000                 ;        0x000F  N/A
      JSR UnhandledInterrupt    ;  7  at 0x0010  Reserved
      JSR UnhandledInterrupt    ;  8  at 0x0012  Reserved
      JSR UnhandledInterrupt    ;  9  at 0x0014  Reserved
      JSR UnhandledInterrupt    ; 10 at 0x0016  Reserved
```

Example - Fast Interrupt for A/D

Read value from A/D into a Circular Buffer in Memory:

Initializing the Fast Interrupt:

The A/D's interrupt request is programmed for Level 2 (highest maskable level)

The shadow registers are initialized:

- **M01 <== SIZE - 1**
- **R0 <== A/D's memory mapped register**
- **R1 <== start address in Output Buffer**

Total Execution Time:
7 Cycles

The Fast Interrupt's Service Routine:

The first instruction must not be JSR or BSR

FRTID is used to return from interrupt (2 delay slots)

```
;Fast Interrupt Service Routine
     FRTID                    ; Return from interrupt - 2 delay slots
     MOVE.W   X:(R0),Y0       ; Read value from A/D peripheral
     MOVE.W   Y0,X:(R1)+      ; Write value to circular buffer in memory
```

Realtime Emulation Capabilities

System Level Debugging at one of 3 levels:

- **Non-intrusive Realtime Debug**
- **Minimally Intrusive Realtime Debug**
- **Breakpoint and Step Mode — Core is halted**

Nexus Level 0 Compliant

A New Specification for on-chip Debugging Interface

Data Upload / Download through the JTAG port

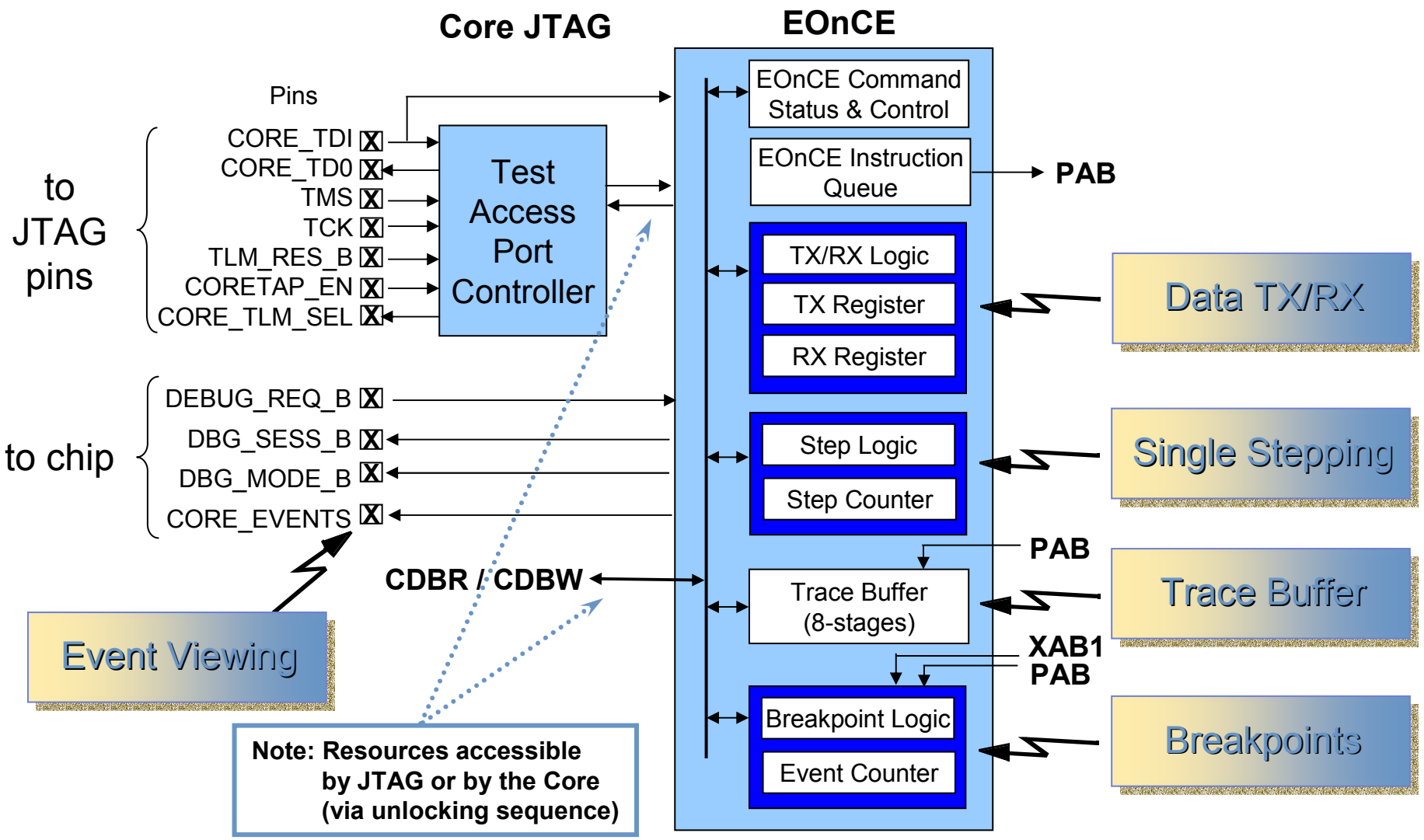
Advanced Breakpoint Capability

Change of Flow Buffer

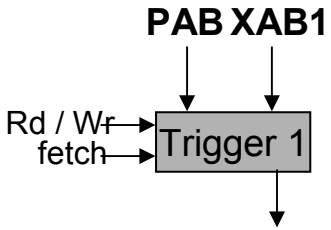
Event Viewing through a terminal

Resources accessible through JTAG or through the Core

Core JTAG / EOnCE Interface Diagram



DSP56800E Breakpoints - Basic



Basic Triggers Available:

on instruction:
PAB==$000080

---

on *write to* data address:
XAB1==$0C0000

on *read from* data address:
XAB1==$0C0000

on *access to* data address:
XAB1==$0C0000

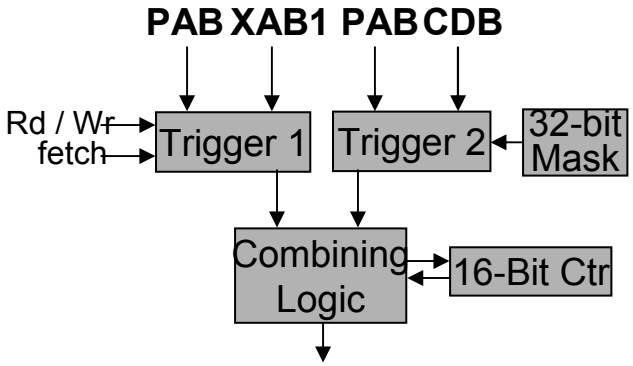
---

on *write to* program address:
PAB==$00E000

on *read from* program address:
PAB ==$00E000

on *access to* program address:
PAB ==$00E000

DSP56800E Breakpoints - Combinations



Example Triggers Available:

on nth occurrence of instruction:
500 occurrences of PAB==$008794

---

on either of two instructions:
PAB==$3792 || PAB==$7E45

on sequence of two instructions:
PAB==$3792 => PAB==$7E45

on 1037 occurrences of PAB==$394
followed by PAB==$7E45

---

on *write of value to* data address:
XAB1==$00FFE7 && CDB==$AAAA

on *read from* data address:
XAB1==$00FFEA && CDB==$5555

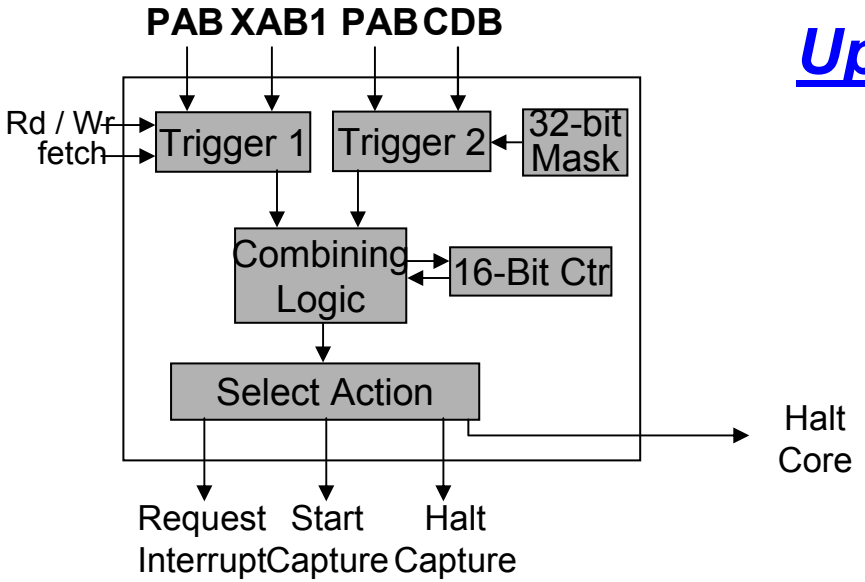
---

on *read from* data address:
XAB1==$00FFEA && CDB!=$5555

---

on *write of bits to* data address:
XAB1==$00FFE7 && CDB[2:0]==$3

Breakpoint Actions



Upon detecting a final trigger:

Halt Core

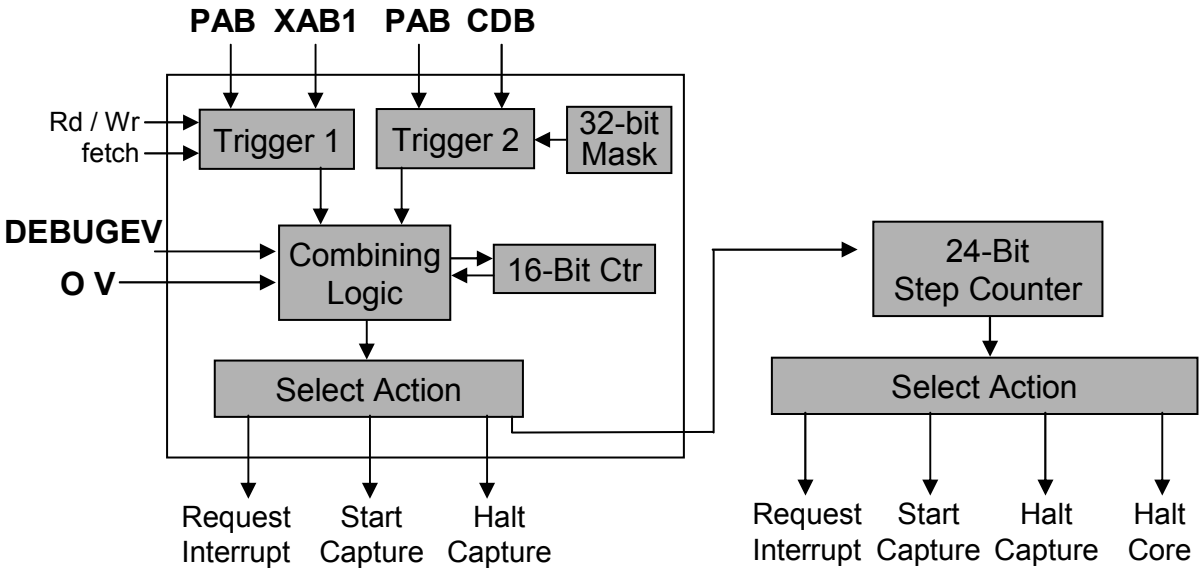
Generate Interrupt Request

Start Trace Buffer Capture

Halt Trace Buffer Capture

Toggle "Event Terminal"

DSP56800E Breakpoints - Advanced



"DEBUGEV" refers to execution of a DEBUGEV instruction.

"OV" refers to an overflow or saturation operation.

Example Triggers Available:

on: (PAB==$3792 || PAB==$7E45 || DEBUGEV) => 4000 instructions

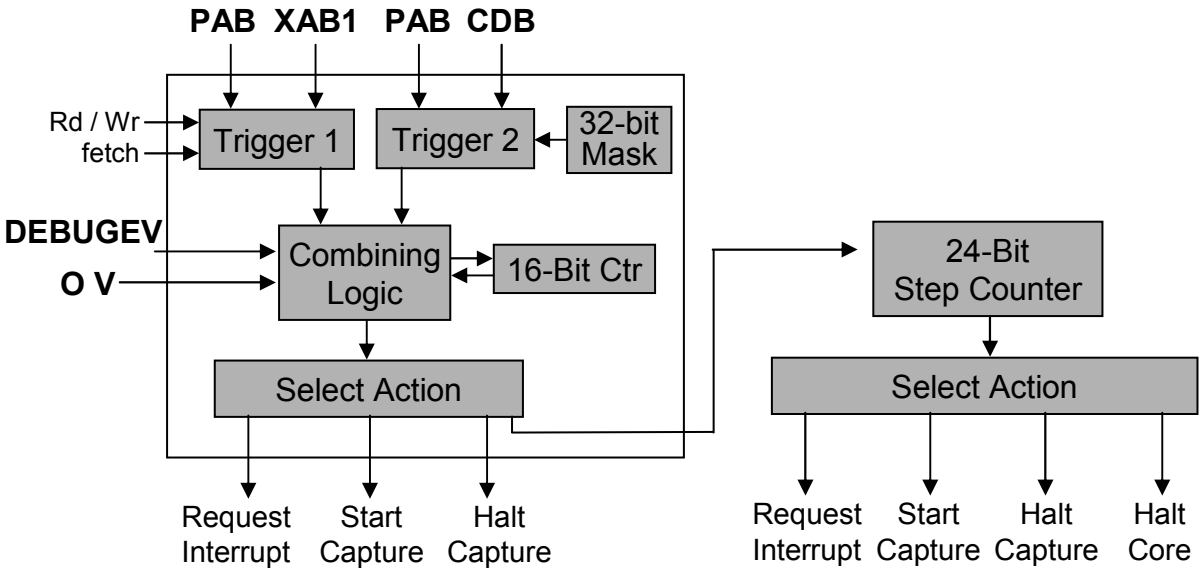
on: PAB==$3792 => PAB==$7E45 => DEBUGEV => 30 instructions

on: (PAB==$394: 900 occurrences) => PAB==$7E45 => OV => 9 instructions

on: (XAB1==$00FFE7 && CDB[14:12]!=$3) => 20,000 instructions

on: (XAB1==$00FFE7 && CDB[14:12]!=$3):400 occurrences => 350 instructions

Advanced Breakpoint Actions



Note: The "DEBUGHLT" instruction allows for halting the core independent of the Step Counter & Breakpoint logic.

Upon detecting a final trigger :

Halt Core

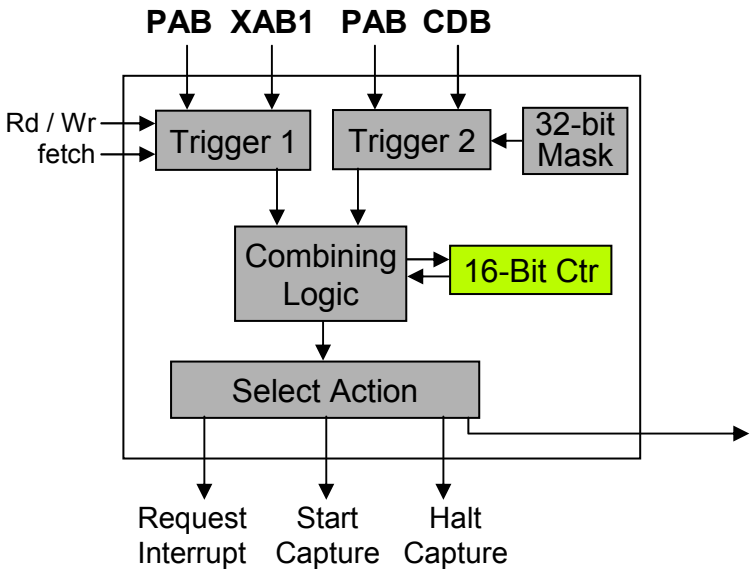
Generate Interrupt Request

Halt Trace Buffer Capture

Start Trace Buffer Capture when Counter begins; Halt Capture when Counter expires.

Start Trace Buffer Capture on PAB Trigger 1; Halt Capture on PAB Trigger 2.

Other Uses of the Breakpoint Counter



Configuring the Counter for Other Uses:

16-bit counter can be assigned to tasks other than trigger generation

Configurable to count the following:

- **Clock cycles - includes wait modes**
- **Clock cycles - not including Wait modes**
- **Instructions executed**
- **Writes to the Trace Buffer**
- **Event counting - counts triggers**

Can be cascaded with 24-bit step counter for 40-bit counter operation

Example Counter Usage:

Count clocks / instructions between: PAB Trigger 1 and PAB Trigger 2

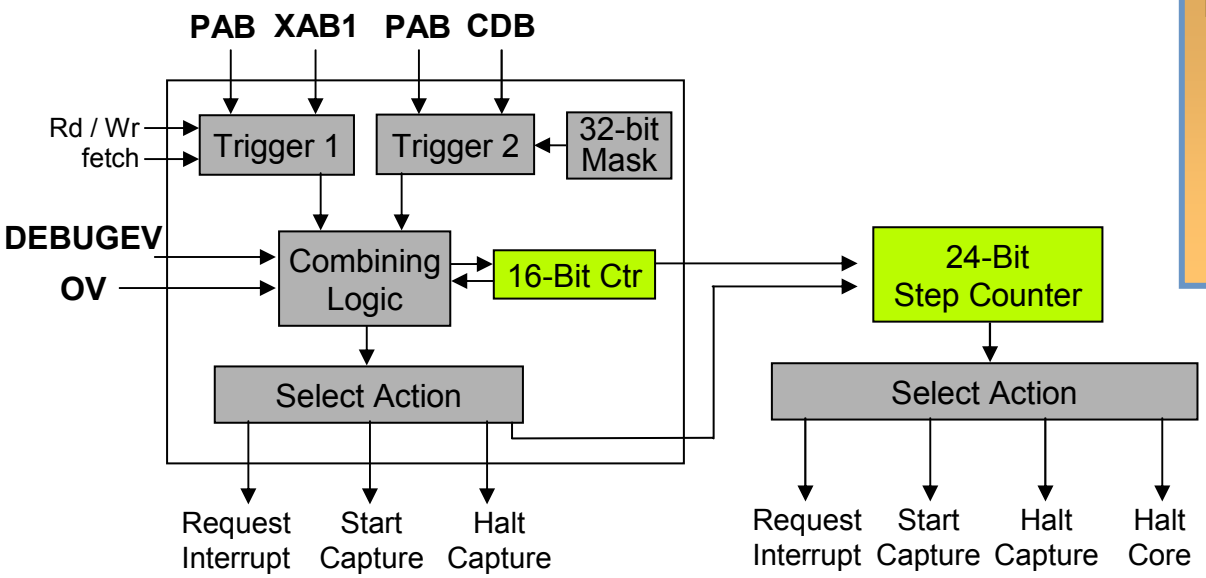
Count clocks / instructions between: Breakpoint trigger and Entering debug state

Count clocks / instructions between: Exiting debug state and Breakpoint trigger

Count clocks / instructions between: DEBUGEV and Breakpoint trigger

Count Breakpoint triggers (event counting)

Advanced Counter Actions



Note: The 24-bit Step Counter can be cascaded with the 16-bit counter to form a 40-bit counter for longer time measurements.

When used in this fashion, it is no longer available to the breakpoint logic.

Advanced Actions :

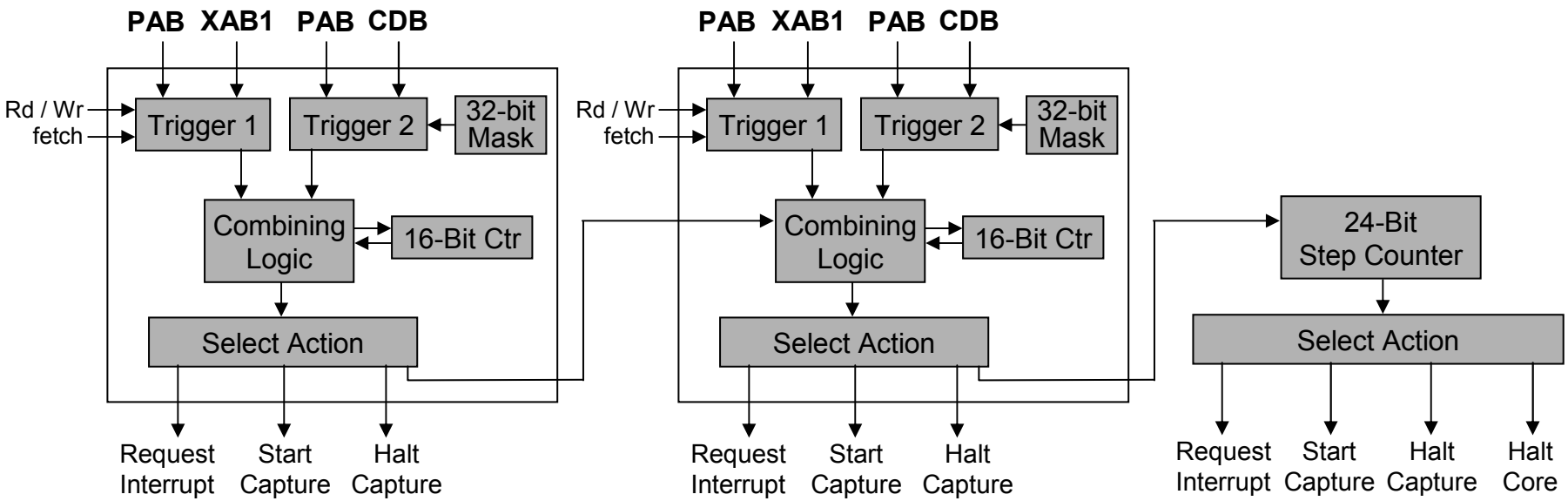
Halt Core or Generate Interrupt Request when:

- **Counter==0**
- **Counter==0 *before* Counter STOP trigger**
- **Counter STOP trigger occurs *before* Counter==0**

Halt Core, Generate Interrupt Request, or Halt Capture when:

- **Counter==0 *before* Counter STOP trigger => 4500 instructions (counters not cascaded)**
- **Counter STOP trigger occurs *before* Counter==0 => 30 instructions (counters not cascaded)**

Dual Breakpoint Units



Example Triggers Available:

on: (PAB==... || PAB==... || PAB==... || PAB==... || DEBUGEV) => 4000 instrs

on: PAB==... => PAB==... => PAB==... => PAB==... => DEBUGEV => 30 instrs

on: PAB==… => PAB==… => (XAB1==$00FFE7 && CDB[14:12]!=$3) => 10 instrs

1st unit used for interrupt generation, 2nd unit used for counter capabilities

1st unit used to start counter, 2nd unit used to stop counter

Programmable Trace Buffer

Circular buffer captures change of flow instructions

Programmable Start Capture and Stop Capture

Programming the "Instruction Capture" (any combination of the following):

- **Interrupt ----------Address of the interrupt vector and Target address of returns**
- **Subroutine ----------Target address of JSR or BSR instructions**
- **Conditional Forward Branch -----Target for Bcc, Jcc*, BRSET, BRCLR**
- **Conditional Backward Branch ---Target for Bcc, BRSET, BRCLR**
- **Conditional Branch not taken ----Target for Bcc, Jcc, BRSET, BRCLR**

Action Performed on "Buffer Full"

- **No Action ------ Continues to capture change of flows**
- **Halt Buffer ----- Capture is stopped and the TBH bit is asserted**
- **Enter Debug --- Enters debug mode**
- **Interrupt -------- Capture is stopped and an interrupt occurs**

Viewable EOnCE Events

Final Trigger from Breakpoint Unit 0 (*hardware watchpoint*)

Core has entered the Debug state

Trace buffer capture started

Trace buffer capture stopped

Trace buffer is full

24-bit Step counter started

24-bit Step counter expired

16-bit counter started

16-bit counter expired (before stopped by breakpoint)

16-bit counter stopped by breakpoint

Transmit data is full

Receive data is empty

Event Viewing - One of the above events can be selected for viewing

Two Styles of Coding Possible

Classical Control Code

```
; Example - Control Code Segment
 ;      if (g_var1 & 0x0400)   {
                brclr      #$0400,X:g_var1,label1

 ;          g_var1 &= 0x05c3;
                bfclr      #$FA3C,X:g_var2

 ;          g_32bit_var = 0x8000;
                move.l     #$8000,X:g_32bitvar

 ;          local_32bit_var <<= 7;
                asll.l     #7,a

 ;          local_16bit_var ^= CONST2;
                bfchg      #CONST2,X:(sp-12)

 ;          if (local_byte1)   {
                tst.b      X:(sp-3)
                bne        label2

 ;          local_16bit_var = CONST1;
                move.w  #CONST1,X:(sp-12)
  ;         }
label2
```

16 Program Words

Classical DSP Code

```
; Example - 2 Cascaded Biquads
        move.w          x:(r0)+,y0      x:(r3)+,x0

 ; ---1st Biquad---
        mac y0,x0,b     x:(r0)+,y1      x:(r3)+,x0
        mac y1,x0,b     y1,x:(r2)+
        tfr   b,a                       x:(r3)+,x0
        asl   a
        mac y0,x0,b                     x:(r3)+,x0
        mac y1,x0,b     x:(r0)+,y0      x:(r3)+,x0

 ; ---2nd Biquad---
        mac y0,x0,b     x:(r0)+,y1      x:(r3)+,x0
        mac y1,x0,b     a,x:(r2)+
        tfr   b,a       y1,x:(r2)+
        asl   a                         x:(r3)+,x0
        mac y0,x0,b                     x:(r3)+,x0
        mac y1,x0,b     a,x:(r2)+
```

13 Program Words

Assembly Code

True MCU Processing:
MCU Features on DSP56800E

True Stack Pointer

16-Bit Program Word for Optimal Code Density

General Purpose Register Files

Orthogonal Instructions available to the Data and Address Register Files

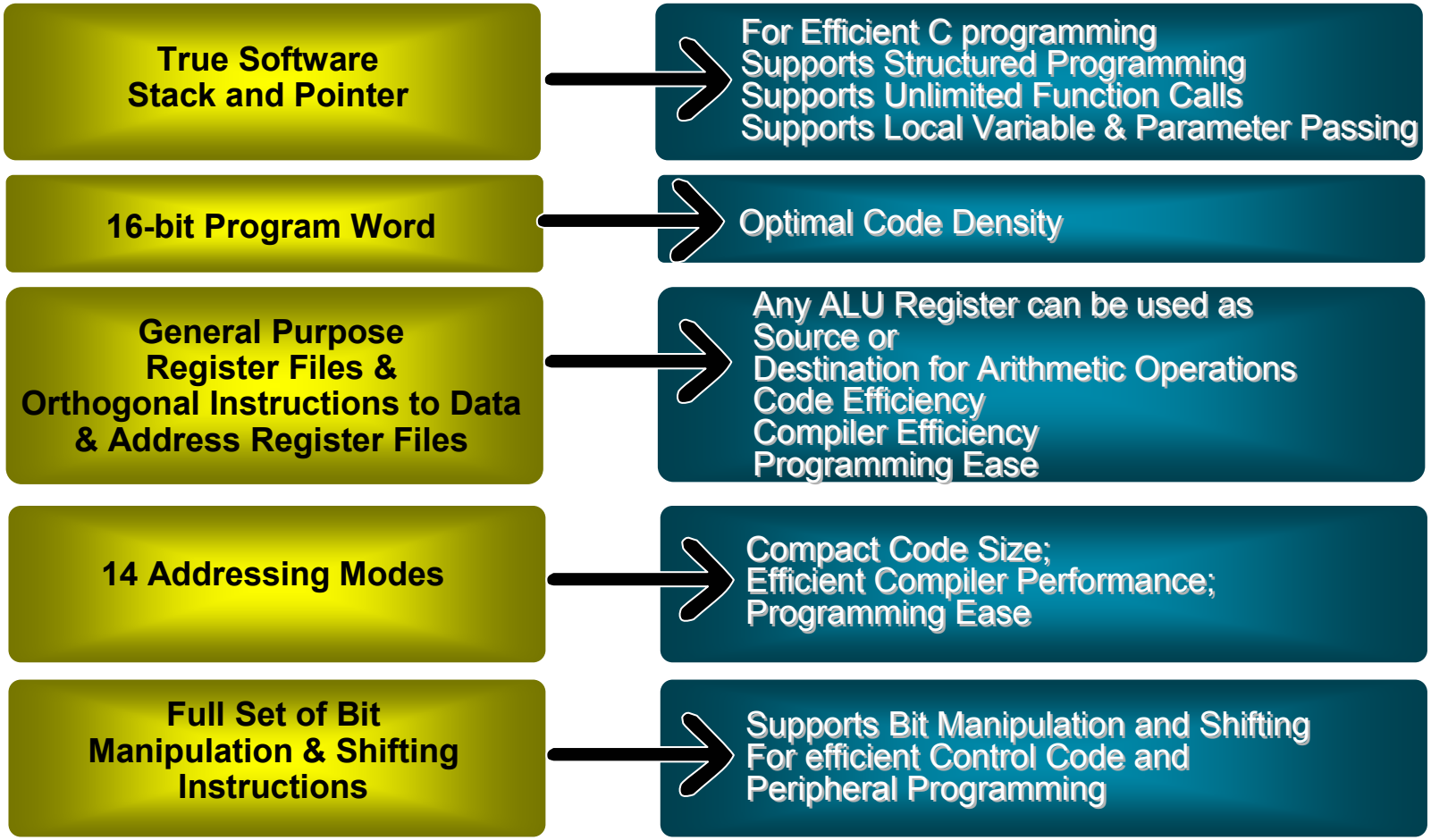
8, 16, and 32-bit Data Types

Atomic Read-Modify-Write instructions

Full set of bit manipulation instructions

16 and 32-bit shifting

True MCU Processing:
MCU Features on DSP56800E



True DSP Processing: DSP Features on DSP56800E

Multiplier - Accumulator

Single and Dual Parallel Move Instructions

No Overhead Hardware Looping

Nested Looping Capability

Modulo Arithmetic (for circular buffers)

Integer and Fractional Arithmetic Support

Fast Interrupt Support

Two Types of Saturation Arithmetic

- **Mode Selectable**
- **Instruction Based**

True DSP Processing:
DSP Features on DSP56800E

Multiplier - Accumulator (MAC) Single And Dual Parallel Move Instructions	→ Features DSP Programmers Demand For Executing True Signal Processing Algorithms
No Overhead Hardware Looping Nested Looping Capability	→ Flexible User Defined, Multi-level Interrupt Priority Support
Modulo arithmetic (For Circular Buffers) Integer And Fractional Arithmetic Support	→ Supports Traditional DSP Mathematical Functions For Executing Complicated Computations
Two levels of Nested Interrupts	→ Gives The Programmer More Control In Interrupt-driven Applications

Is it a 16 or 32-Bit MCU?

Five 32-bit Data Registers

Eight 24-bit Address Registers

32-bit ALU Operations

— Add, Subtract, Test, Compare, Logical, etc.

32-bit shifting

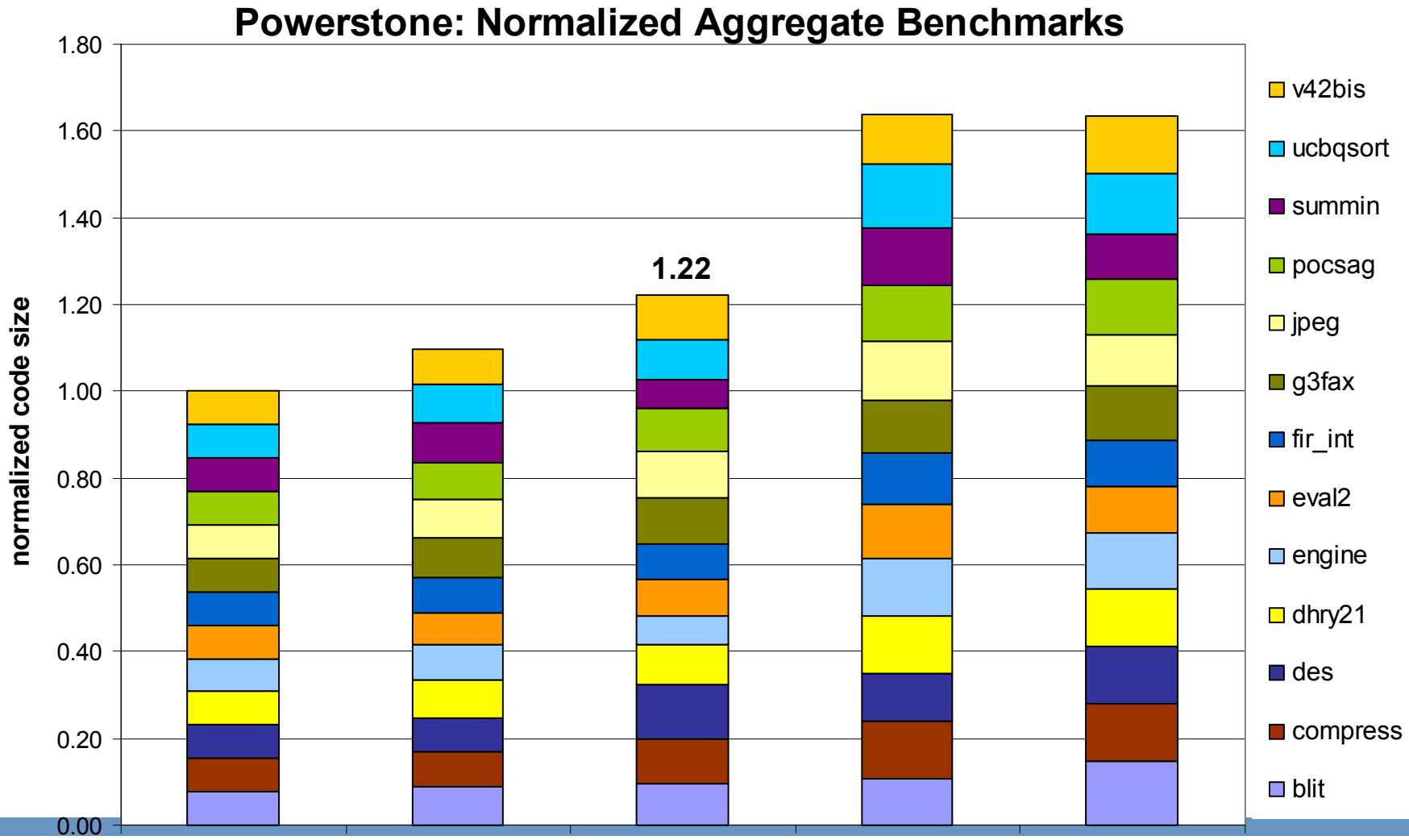
24-bit pointers

24-bit pointer arithmetic

==> 24-bit pointers (not 32-bit)

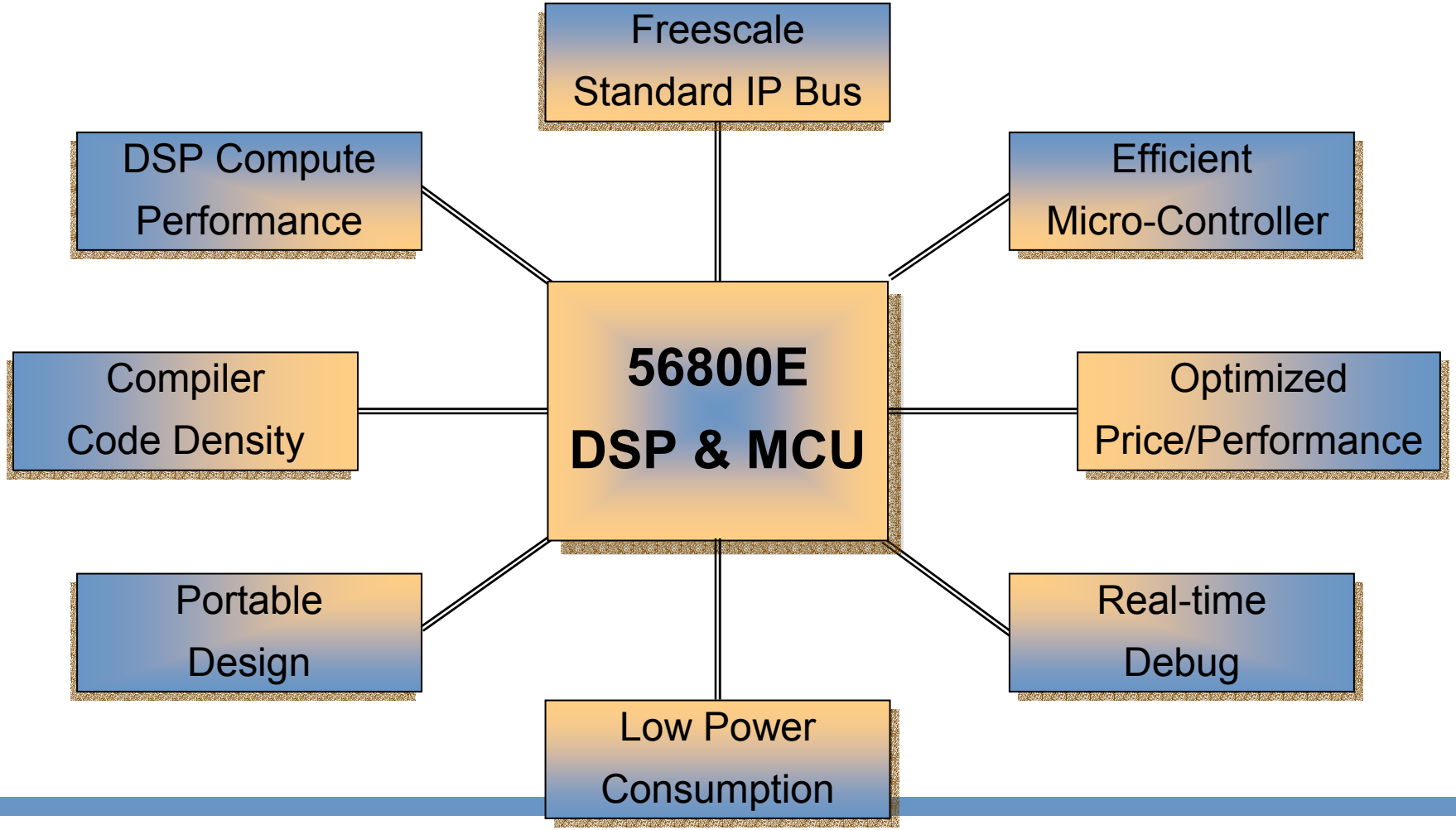
==> Note: Can't perform 32x32 multiplication in single instruction

Powerstone C code size Benchmarks
56800E - Competitive with micros and DSPs



Launched by Motorola
freescale semiconductor

Bringing it All Together
The 56800E Family Foundation



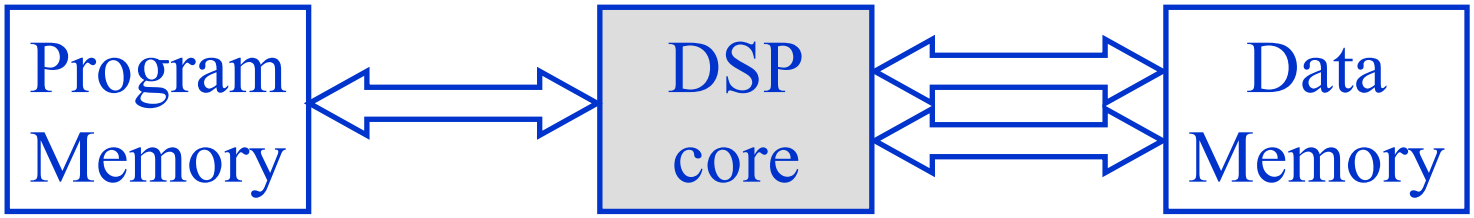
Hybrid DSP

The End

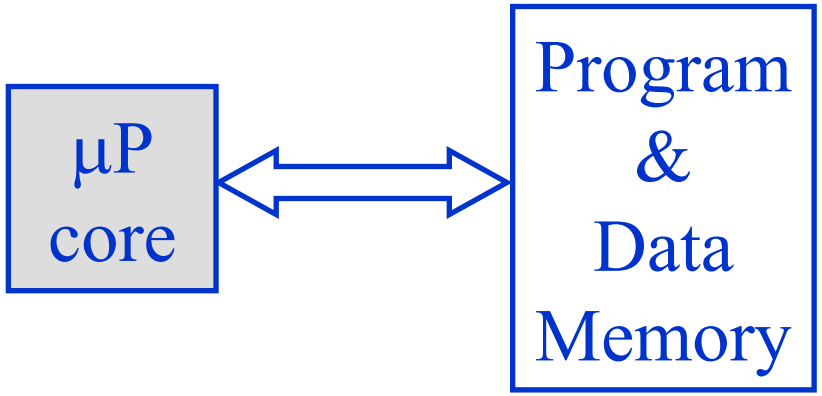
Hybrid MCU Architecture Introduction

56800/E Core Introduction

DSP vs. Microprocessor Architectures



Modified Harvard Architecture

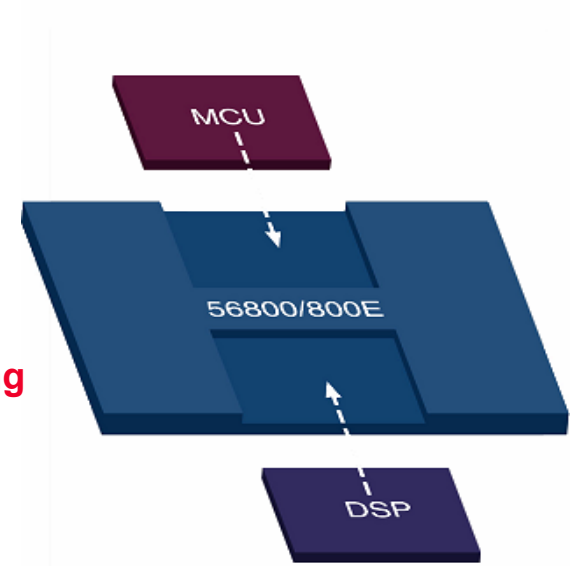


Von Neumman Architecture

56800/E Family Combining Signal Processing and Controller Functionality

Traditional Microcontroller

- Design for Controller Code
- Compact Code Size
- Easy to Program
- Inefficient Signal Processing



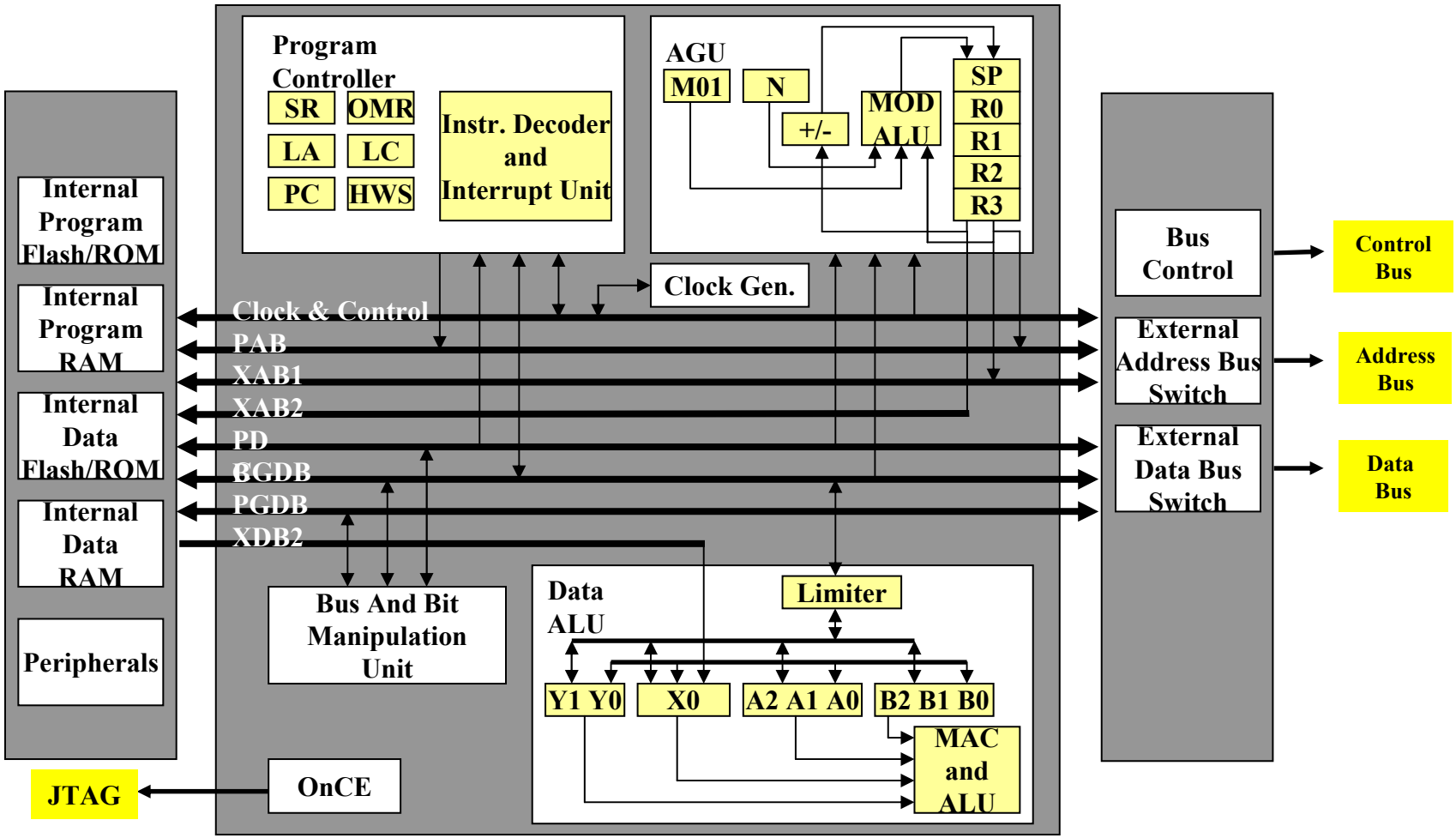
Traditional DSP Engine

- Designed for DSP Processing
- Designed for Matrix Operations
- Complex Programming
- Less Suitable for Control

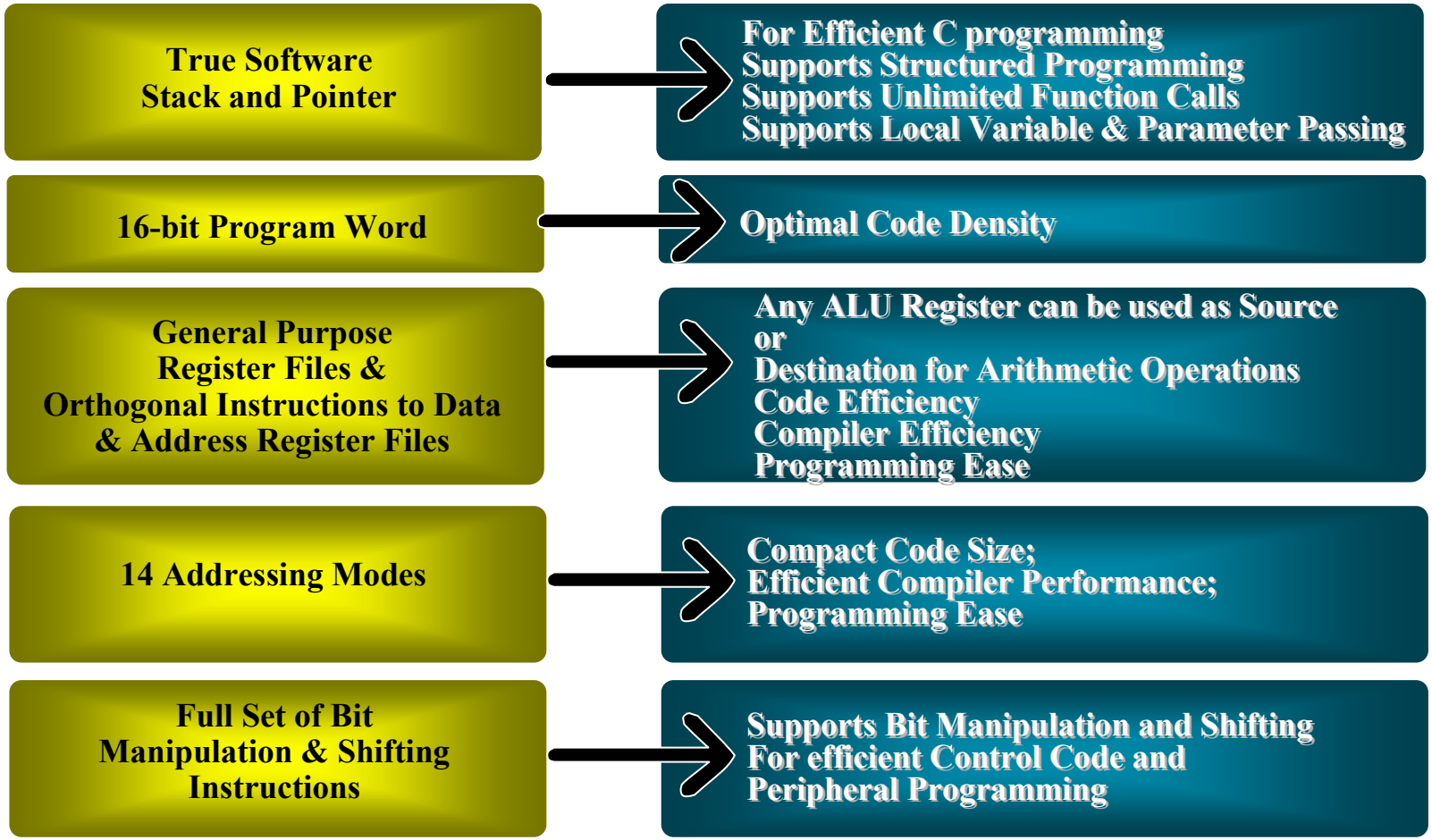
- Instructions Optimized for Controller Code, DSP, Matrix Operations
- Compact Assembly and “C” Compiled Code Size
- Easy to Program
- Additional MIPS Headroom and extended addressing space

Launched by Motorola freescale semiconductor

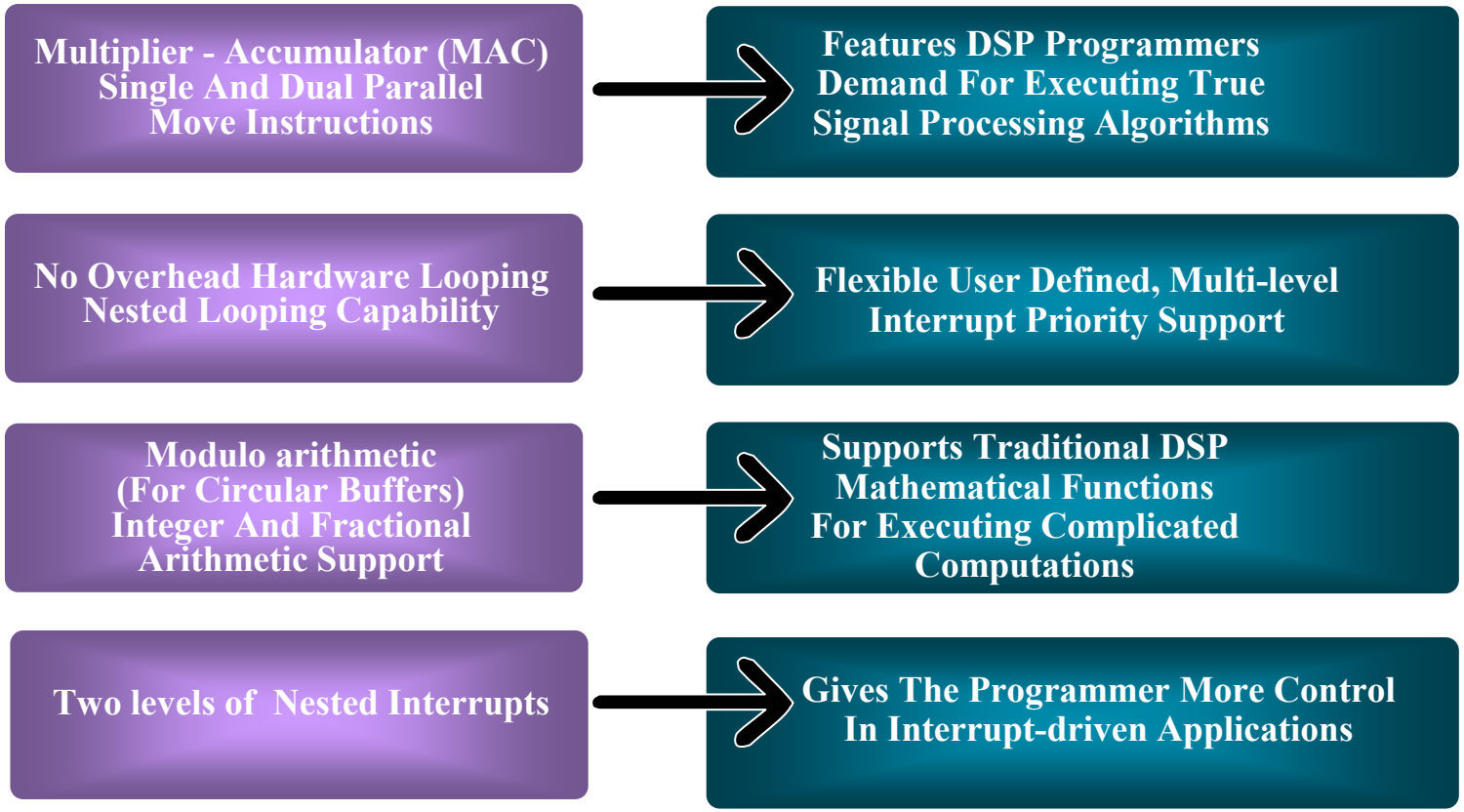
56800 Core Block Diagram



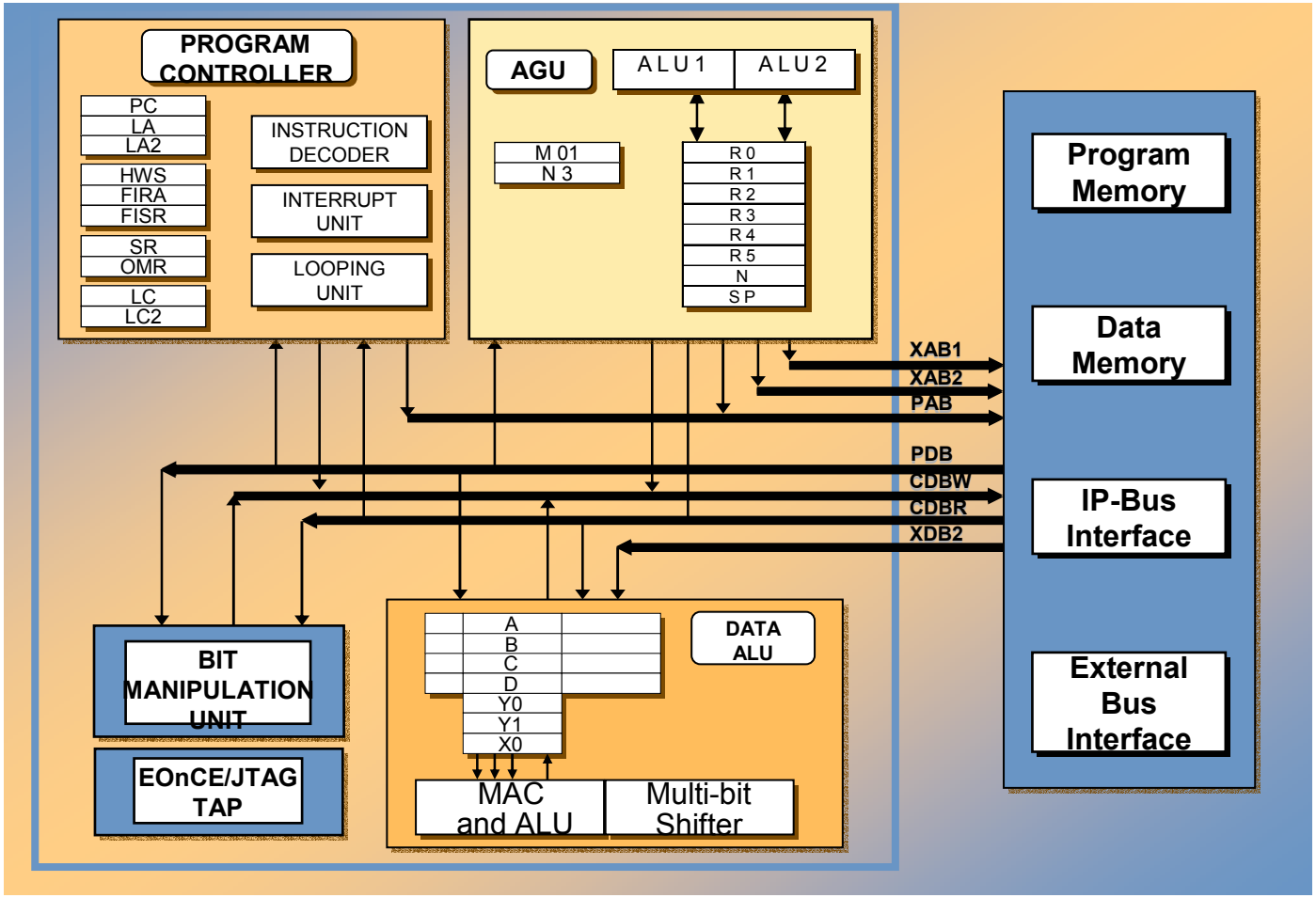
56800 Controller Functionality



56800 Signal Processing Functionality



56800E Core Architecture



Instruction Fetch:

PAB - 21 bits
PDB - 16 bits

1st Data Access:

XAB1 - 24 bits
CDBR - 32 bits
CDBW - 32 bits

2nd Data Access:

XAB2 - 24 bits
XDB2 - 16 bits

Operations Performed:

1st - PAB / PDB

2nd - XAB1 / CDBR-CDBW

3rd - XAB2 / XDB2

MAC and Dual Parallel Read

MAC — Multiply-Accumulate — MAC

Operation:

D + (S1 × S2) → D (no parallel move)
D + (S1 × S2) → D (one parallel move)
D + (S1 × S2) → D (two parallel reads)

Assembler Syntax:

MAC	(±)S1,S2,D	(no parallel move)
MAC	(±)S1,S2,D	(one parallel move)
MAC	S1,S2,D	(two parallel reads)

Description: Multiply the two signed 16-bit source operands, and add or subtract the 32-bit fractional product to or from the destination (D). Both source operands must be located in the FF1 portion of an accumulator or in X0, Y0, or Y1. The fractional product is first sign extended before the 36-bit addition (or subtraction) is performed. If the destination is one of the 16-bit registers, it is first sign extended internally and concatenated with 16 zero bits to form a 36-bit operand before the operation to the fractional product;

Instruction Fields:

One clock cycle!

Operation	Operands	C	W	Comments
MAC	(±)FFF1,FFF1,FFF	1	1	Fractional multiply-accumulate; multiplication result optionally negated before accumulation.

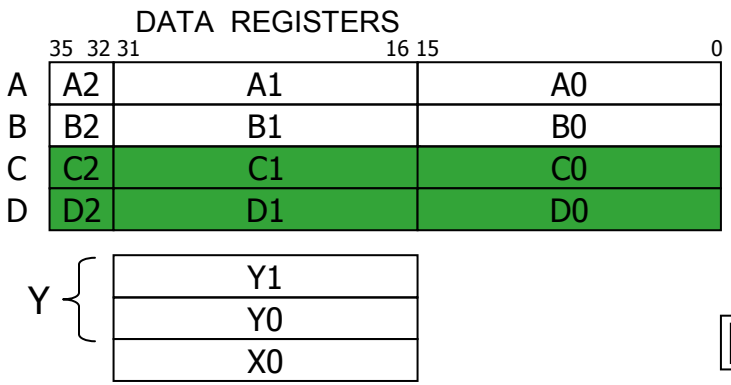
Example:

```
MAC     Y0,X0,A   X:(R0)+,Y0   X:(R3)+,X0   ; fractional MAC, two reads
```

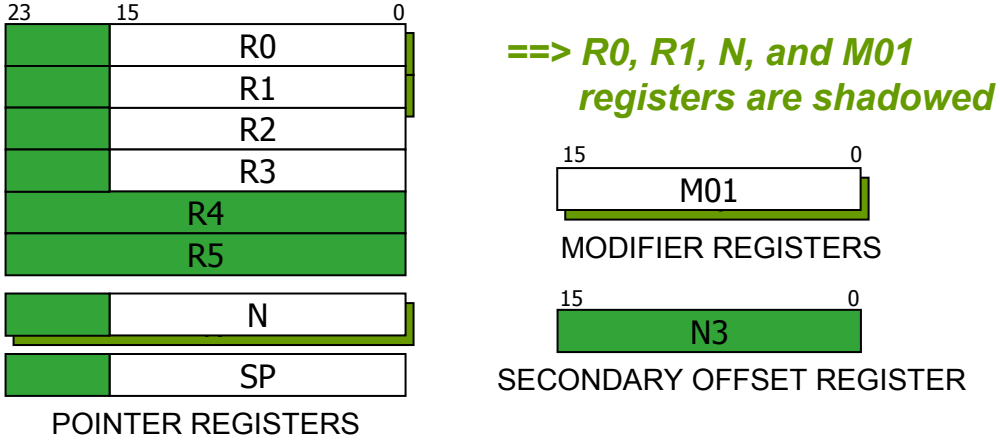
What's Happening: A+=C[n]*X[n] **Move *X*[*n*+1]** **R0++** **R0 wrapped** **Move *C*[*n*+1]** **R3++** } **All in One Instruction Cycle**

Programming Model Comparison: 56800 vs 56800E

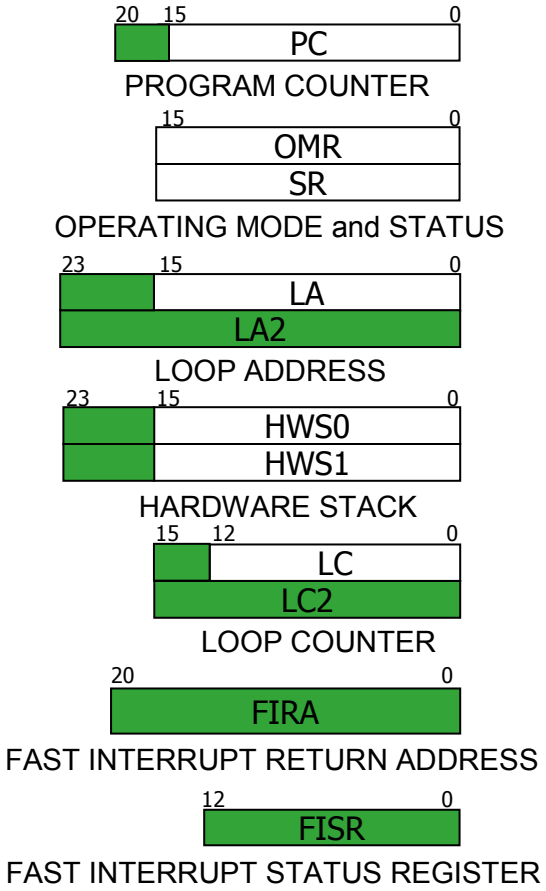
DATA ARITHMETIC LOGIC UNIT



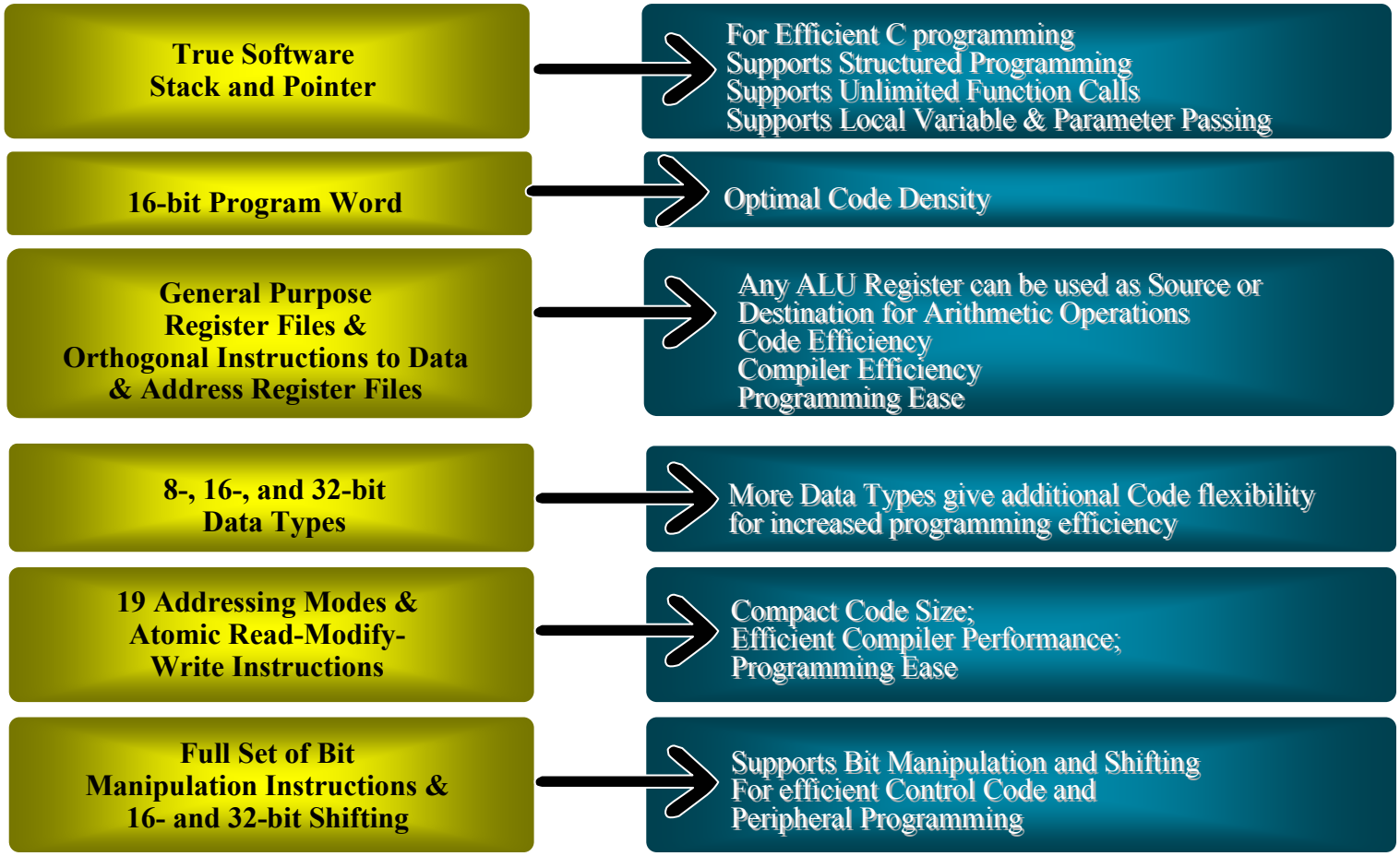
ADDRESS GENERATION UNIT



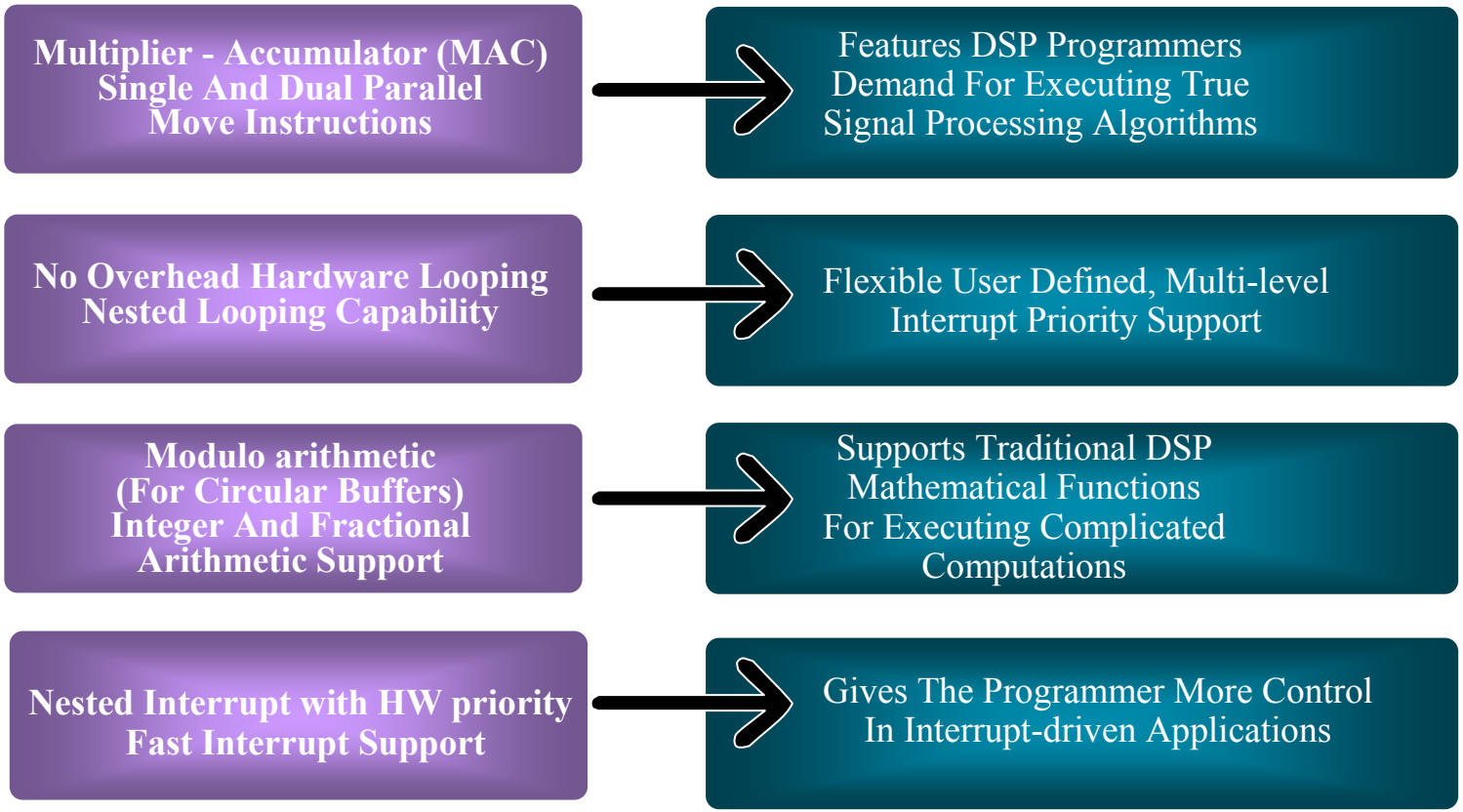
PROGRAM CONTROL UNIT



56800E Controller Functionality



56800E Signal Processing Functionality



56800/E Core Comparisons

Core	MIPS (Max)	Clocks per Instr	# Interrupt levels	Registers	Data Types	Program Memory Addr Space	Data Memory Addr Space	Technology
56800	40	2	2	5 Data 5 Address	16-bit	128 KB	128 KB	Semi-custom
56800E	200	1	5	7 Data 8 Address	8-bit, 16-bit 32-bit	4 MB	32 MB	Fully Synthesizable & Scanable

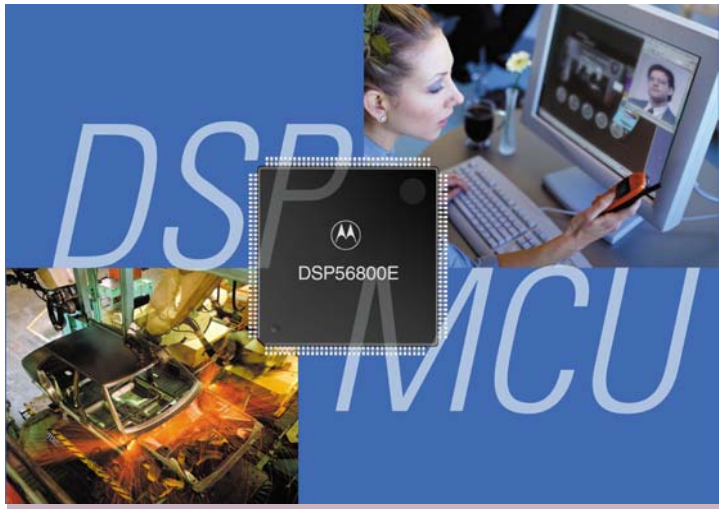
56800E Enhancements

- Compiler Efficiency
- Real-time Non-Intrusive Debug
- Fast Interrupt
- Nested Hardware Looping
- Additional Addressing Modes
- Enhanced OnCE/JTAG

56800/E Core Differentiators

Q: Name at least two distinguishing features of the 56800E Core that contribute to its exceptional performance?

- Harvard Architecture contains separate Program and Data buses
- Single cycle MAC and dual parallel data reads provide enhanced DSP algorithm performance
- No overhead interruptible hardware DO loops
- Independent Functional units (Program Controller, AGU, Data ALU, Bit Manipulation) support parallel processing
- Fast Interrupts provide low overhead and low latency

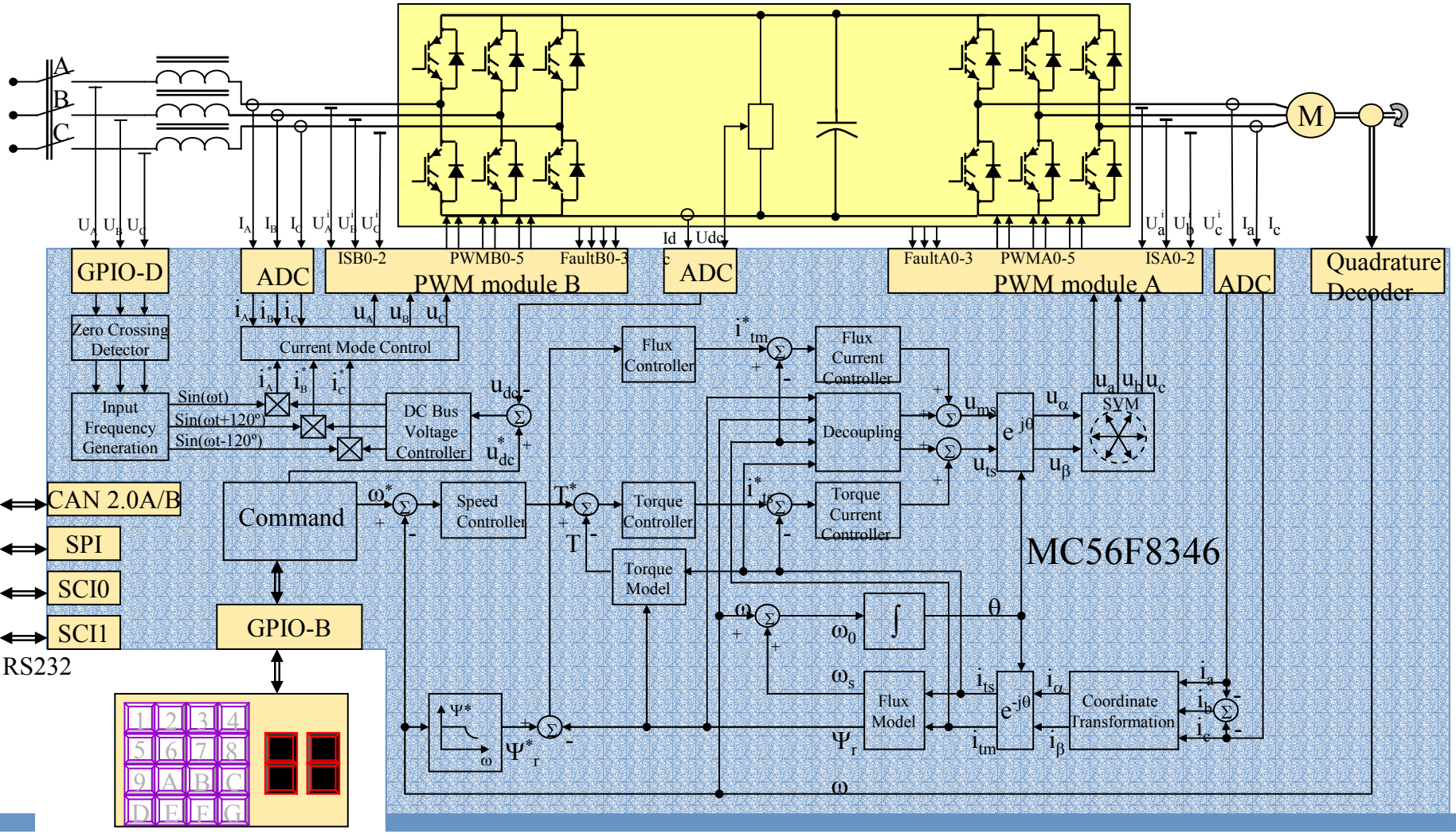


Hybrid MCU Architecture I

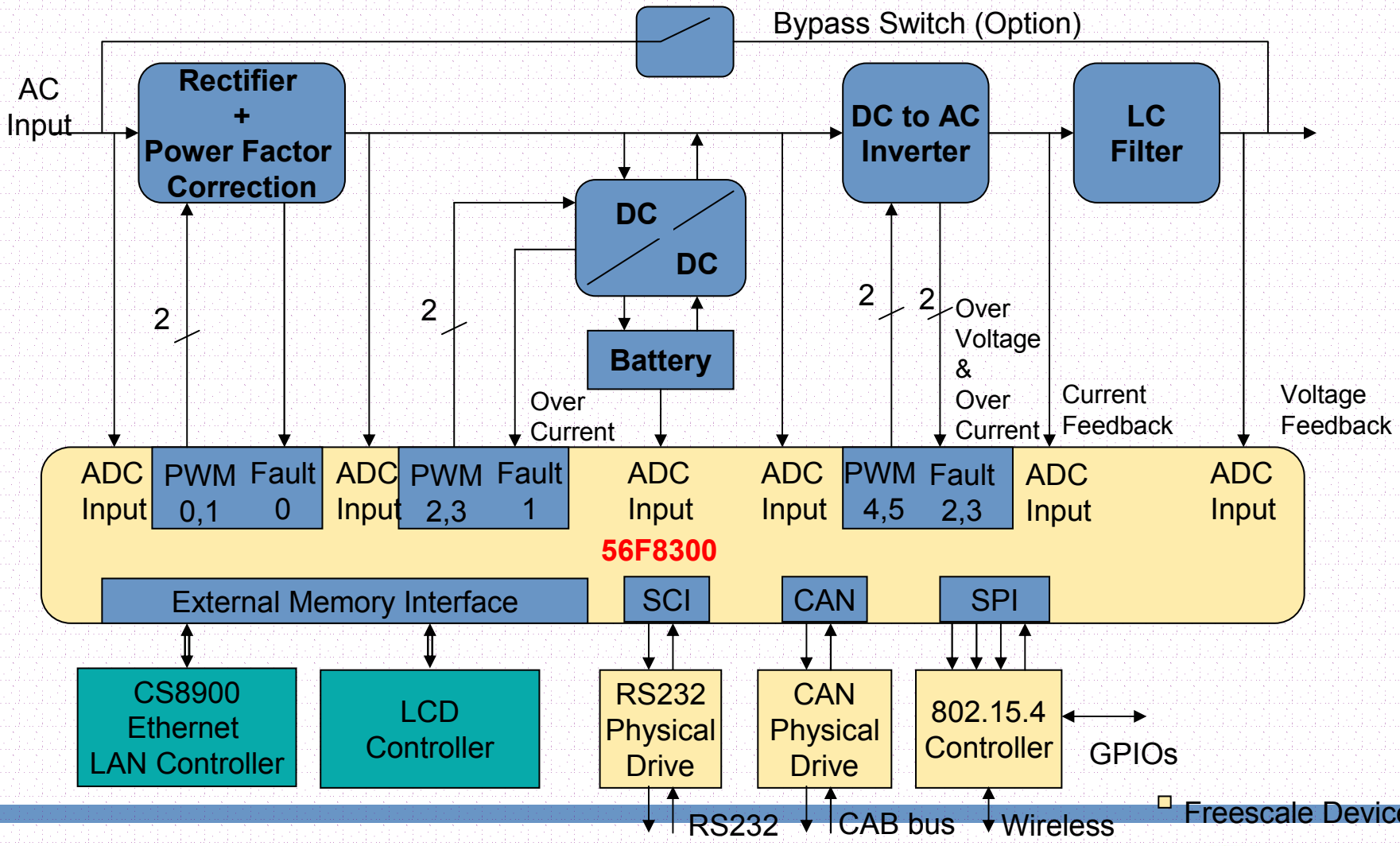
56F8300 Integrated Peripherals

Launched by Motorola
freescale™
semiconductor

Field Oriented Motor Control Application



UPS Application



Analog to Digital Converter

12 bit resolution

Up to two ADC modules
- Two ADCs per Module
- **6 to 8 Analog inputs per ADC Module**

Sampling rate up to
1.66 million samples per second

Sequential conversions: first 1.7 μsec subsequent 1.2 μsec

Simultaneous conversions: 8 in 5.3 μsec

Can be synchronized with
Pulse Width Modulators (PWM)

Simultaneous or sequential sampling

Eight words result buffer

Sample correction via programmable offset

Current injection protection

new

Software self calibration capability
- Removes gain and offset errors

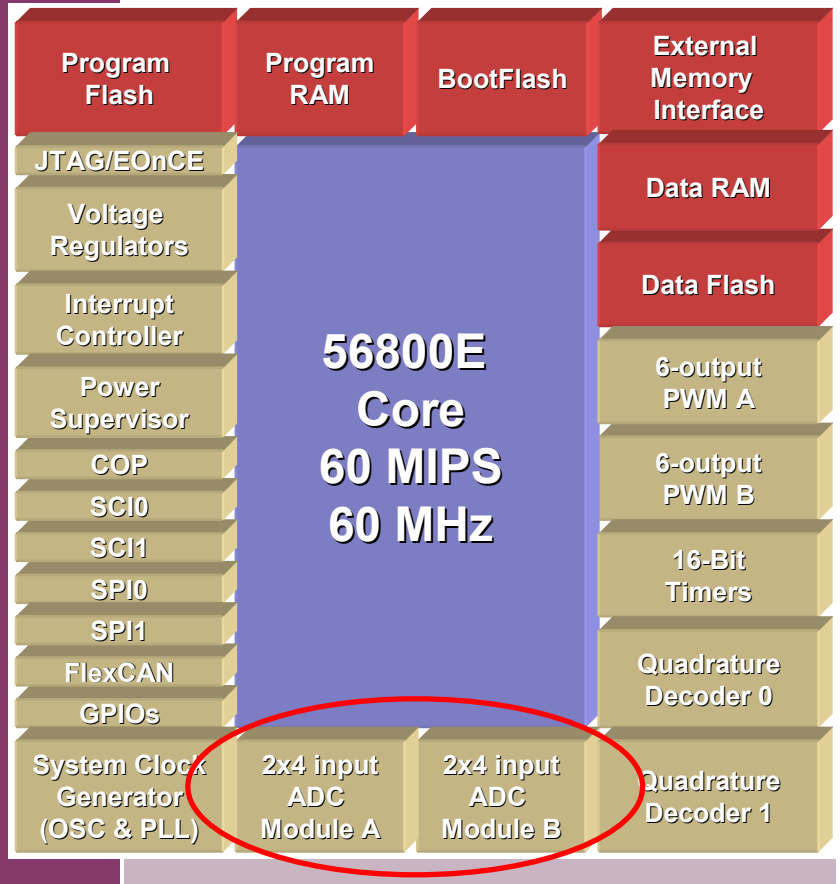
new

Interrupt generating capabilities
- End-of-Scan; Zero crossing; High/Low limit check

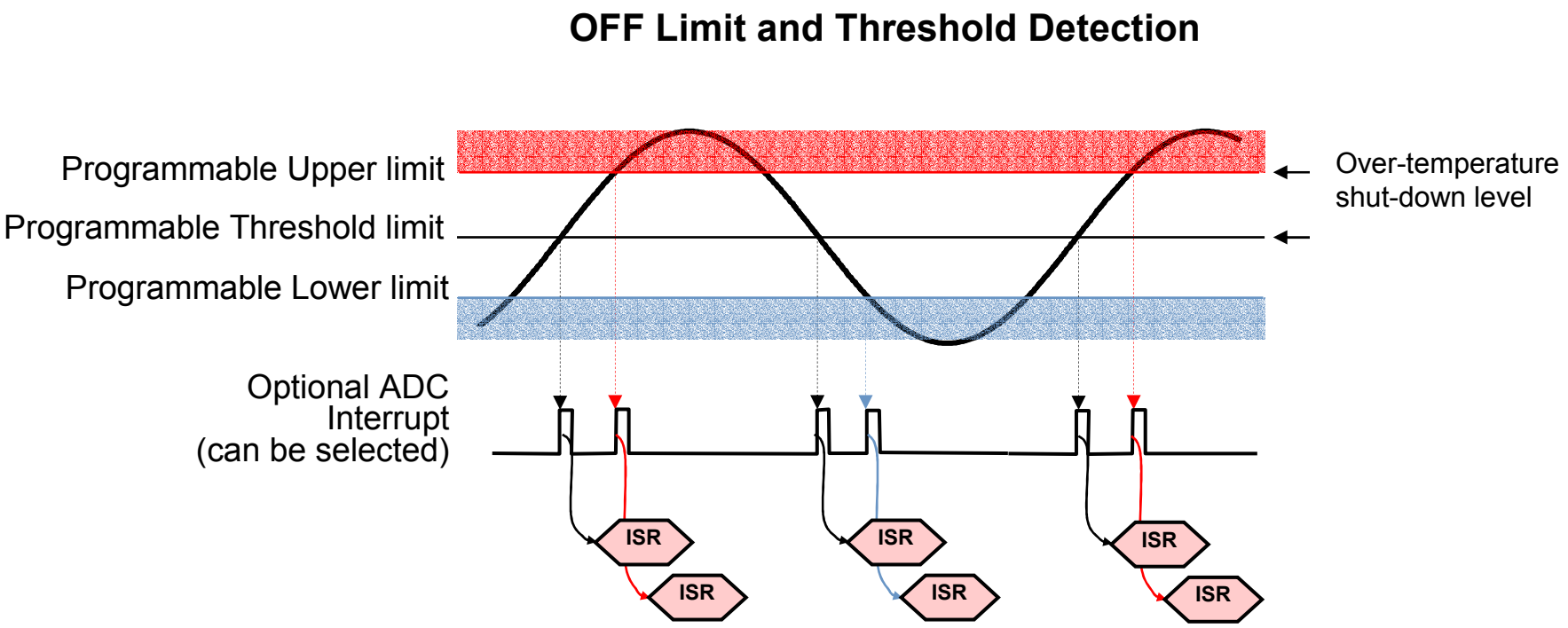
Two outputs formats available
- Two's complement
- Unsigned

Power down and power savings modes
- Explicit power down
- Intelligent power savings mode: powers up only when needed

new

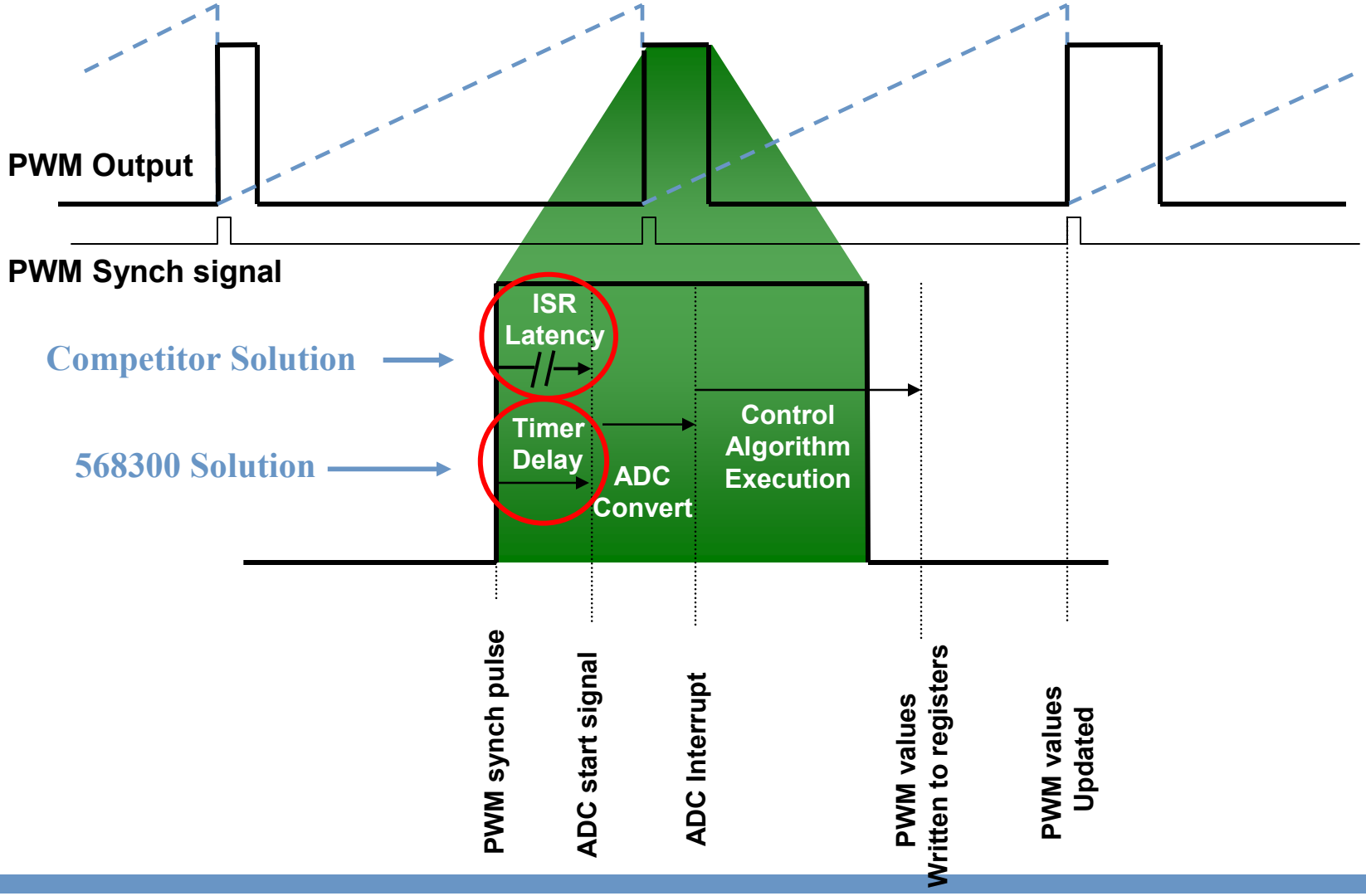


ADC Sampling Process



- ✓ The ADC can perform limit checking and zero crossing detection with NO CPU intervention.
- ✓ Each channel has its own upper, lower, and threshold comparators.

ADC Synchronization with the PWM



ADC Modes

Once

- **The ADC starts to sample just one time whether you use the START bit or by a sync pulse. This mode must be re-armed by writing to the ADCR1 register again if you want to go capture another scan**

Triggered

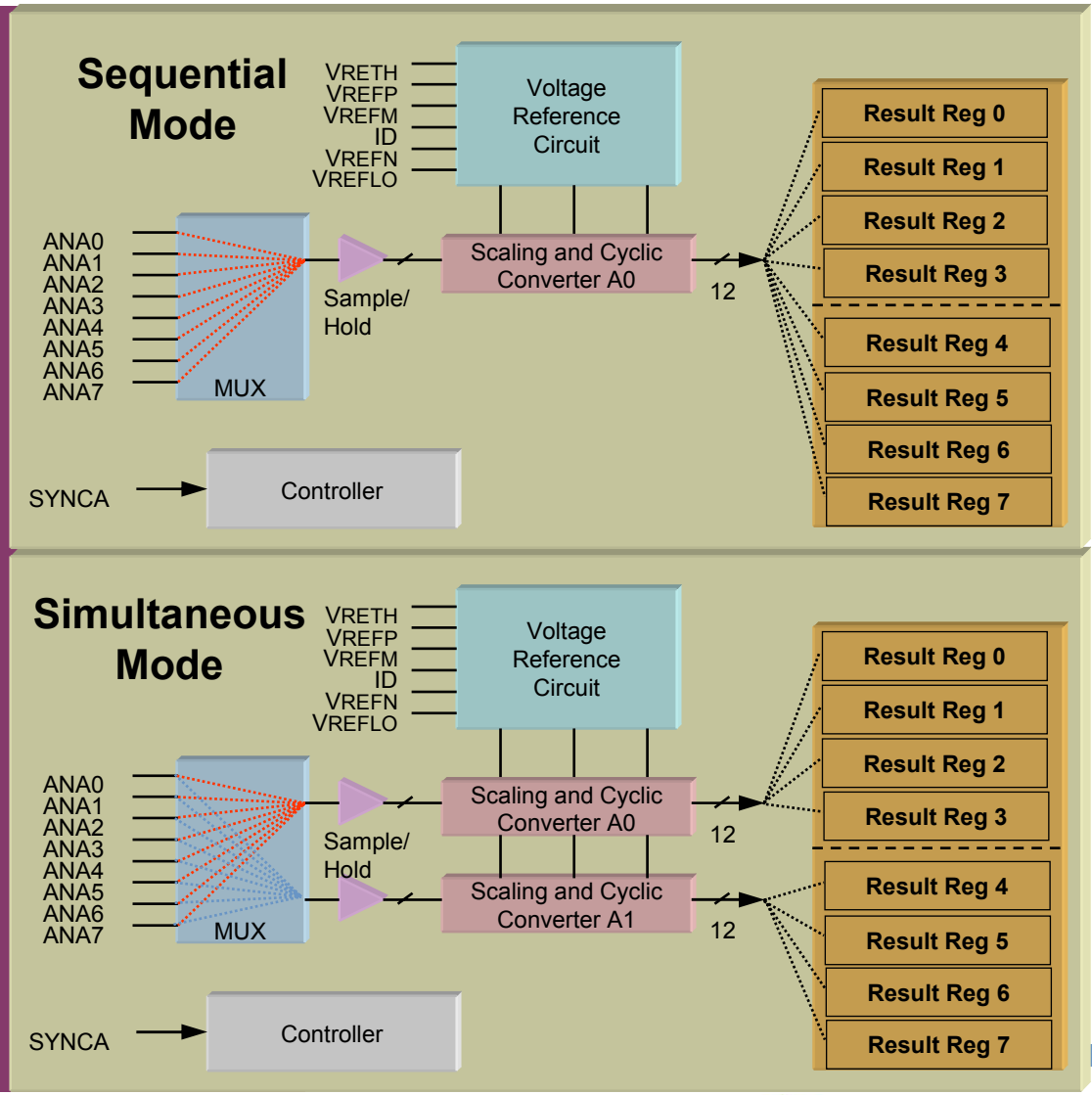
- **Sampling begins with every recognized START command or sync pulse**

Loop

- **The ADC continuously take samples as long as power is on and the STOP bit has not been set**

Each mode can be sequential or simultaneous

- **Sequential will sample SampleN one after another, while simultaneous can sample SampleN from Group1 and SampleN from Group 2 at the same time.**



56F8300 Differentiators

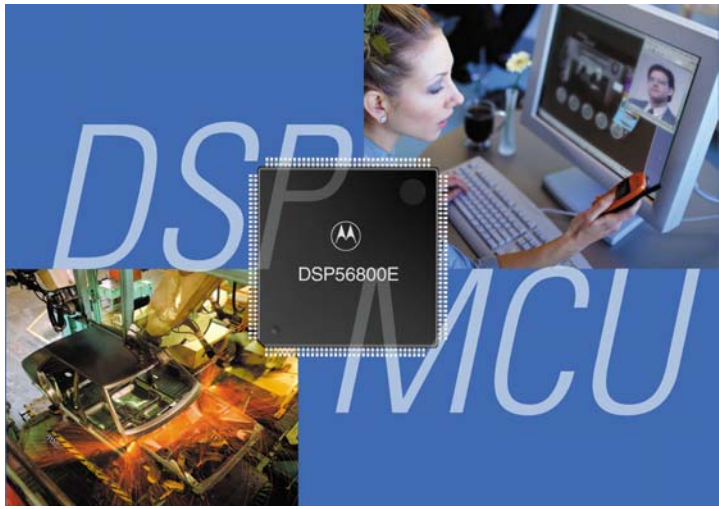
Q: Name the 4 types of ADC interrupts?

End of Scan

Zero Crossing

Upper Limit

Lower Limit



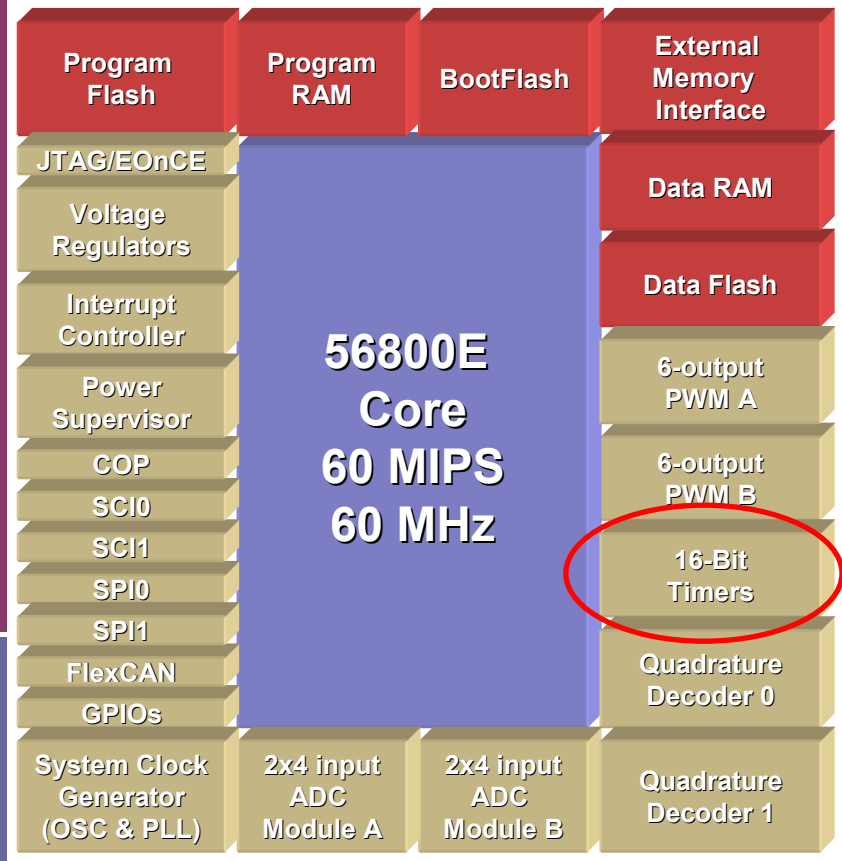
Launched by Motorola
freescale semiconductor

Quad Timers

- Four16-bit general purpose up/down timers per module
- Individually programmable
  - Input capture trigger
  - Output compare capture
  - Clock source
- Pins available as general I/O when timer(s) not in use
- Input pins are shareable within a timer module (Quad)
- Compare preload feature
- Counters in module can be daisy-chained to yield longer counter lengths

Operation Modes

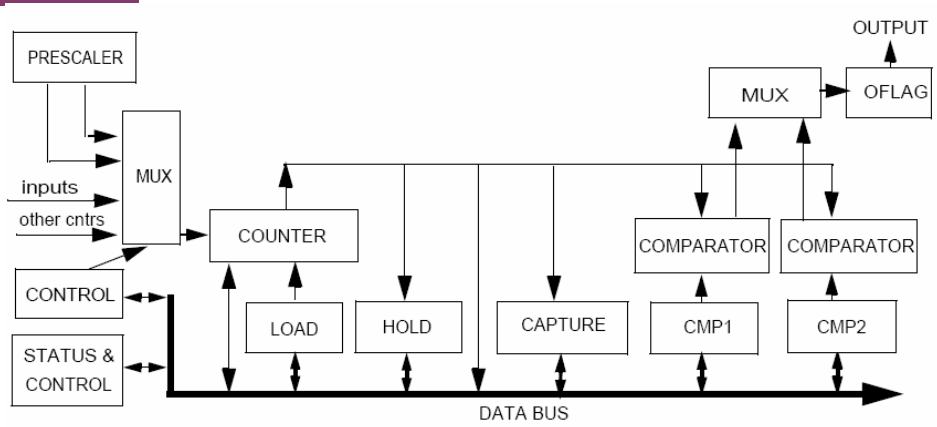
- Fixed Frequency PWM mode
- Variable Frequency PWM Mode
- Stop Mode
- Count Mode
- Edge Count Mode
- Gated Count Mode
- Quadrature Count Mode
- Signed Count Mode
- Triggered Count Mode
- One-Shot Mode
- Cascade Count Mode
- Pulse Output Mode

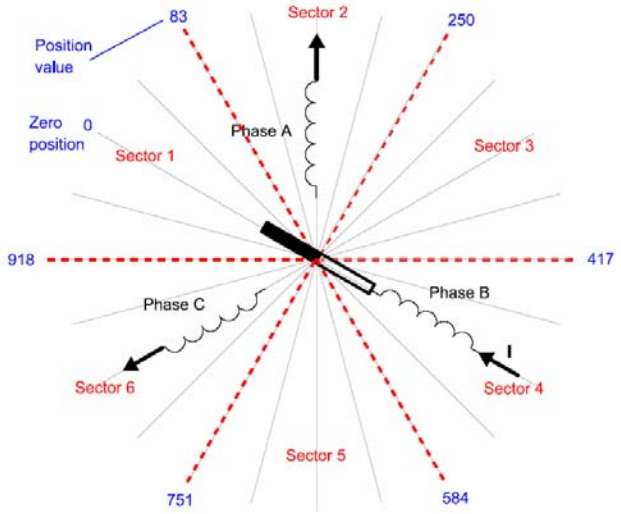


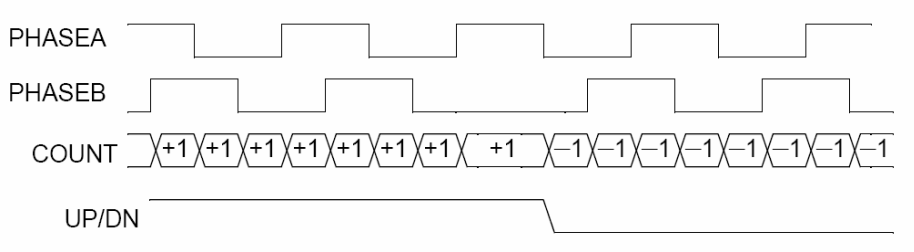
Motor Control – Quadrature Mode

Decodes primary and secondary inputs as quadrature encoder signals

Comparators are utilized to signal commutation transition



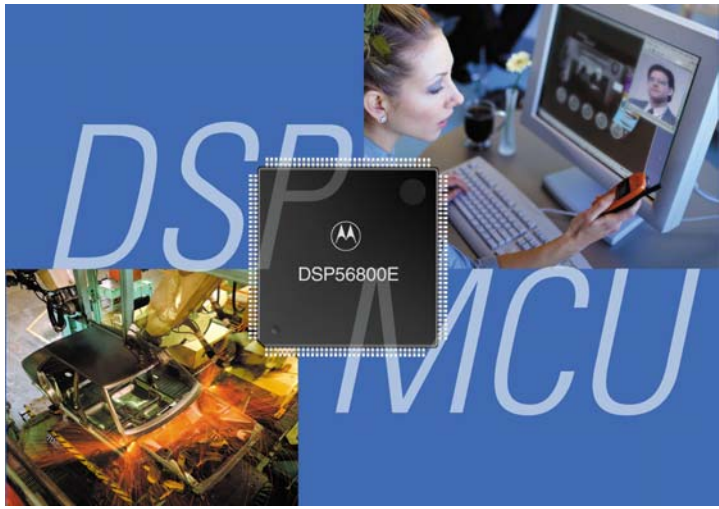




56F8300 Differentiators

Q: What is the maximum count available within a Quad Timer module?

All four 16-bit timers can be daisy chained to provide a count value up to 2^{64}

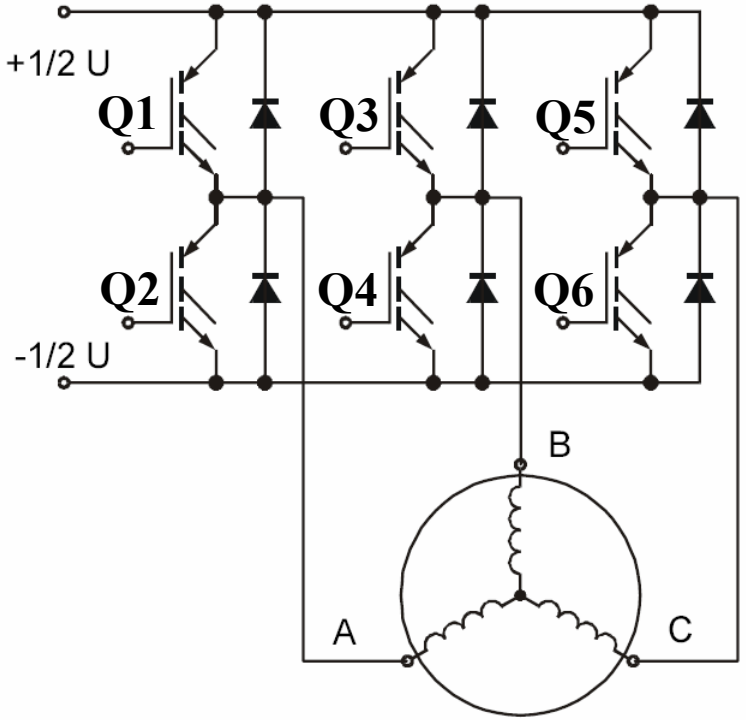


PWM – BLDC Background

Turn BLDC motor by performing commutation

- **Apply voltage to phases A & B, then B & C, then A & C, etc:**

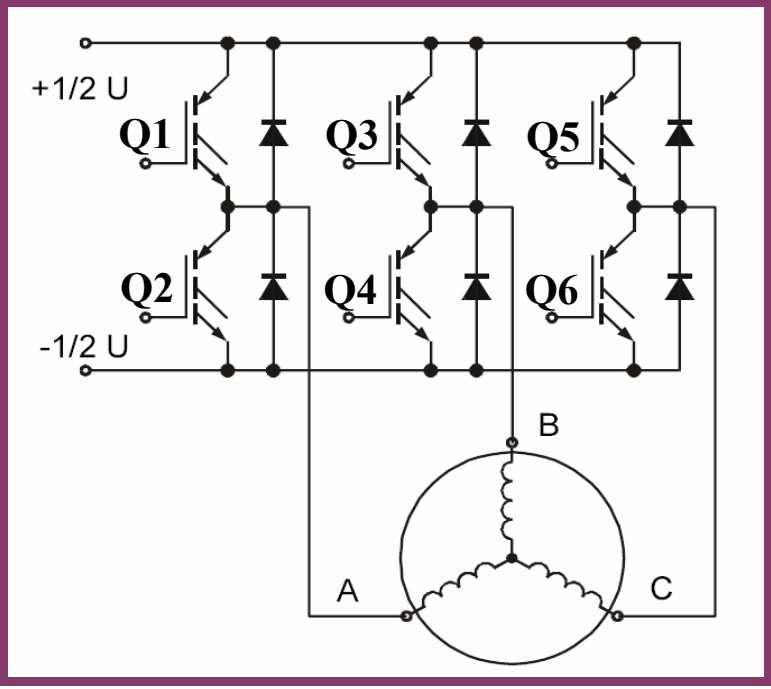
Phase A	Phase B	Phase C
$-V_{DCB}$	$+V_{DCB}$	NC
NC	$+V_{DCB}$	$-V_{DCB}$
$+V_{DCB}$	NC	$-V_{DCB}$
$+V_{DCB}$	$-V_{DCB}$	NC
NC	$-V_{DCB}$	$+V_{DCB}$
$-V_{DCB}$	NC	$+V_{DCB}$



PWM – BLDC Background

Control speed by changing applied voltage

- Change voltage by utilizing “switch” technique – Duty Cycle



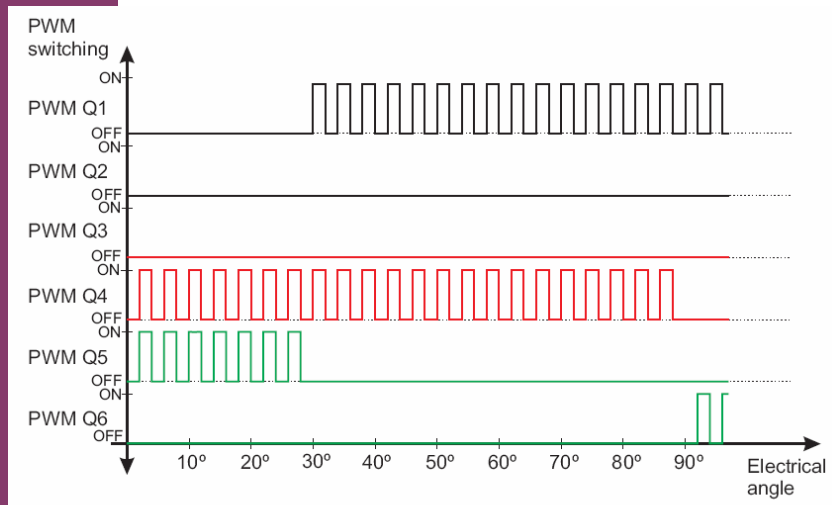


Figure 3-4. Independent Switching of Power Transistors

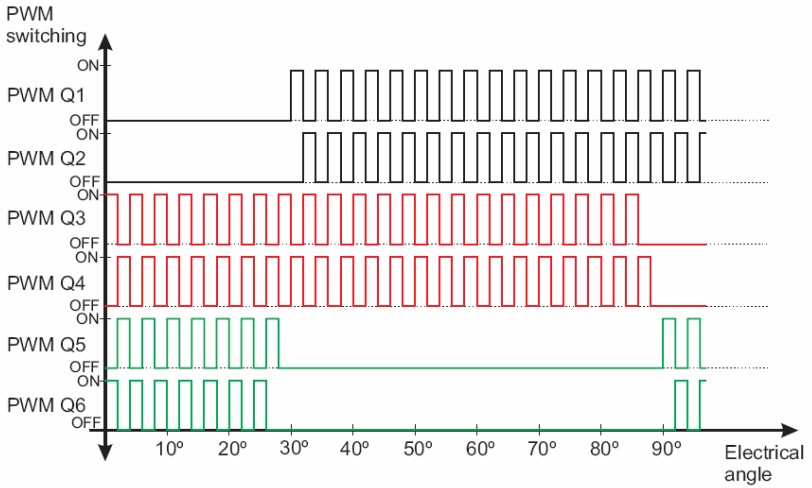


Figure 3-5. Complementary Switching of Power Transistors

PWM

Three complementary signal pairs or six independent signals

Complementary channel operation

- Deadtime insertion
- **Separate top and bottom pulse width correction via current sensing or software**
- **Separate top and bottom polarity control**

Edge-aligned or center-aligned signals

15-bits of resolution

Half-cycle reload capability

Asymmetric mode of operation (for phase shifting)

Programmable integral reload rates (half to 16)

Individually software-controlled PWM outputs

ADC synchronization

Programmable fault inputs

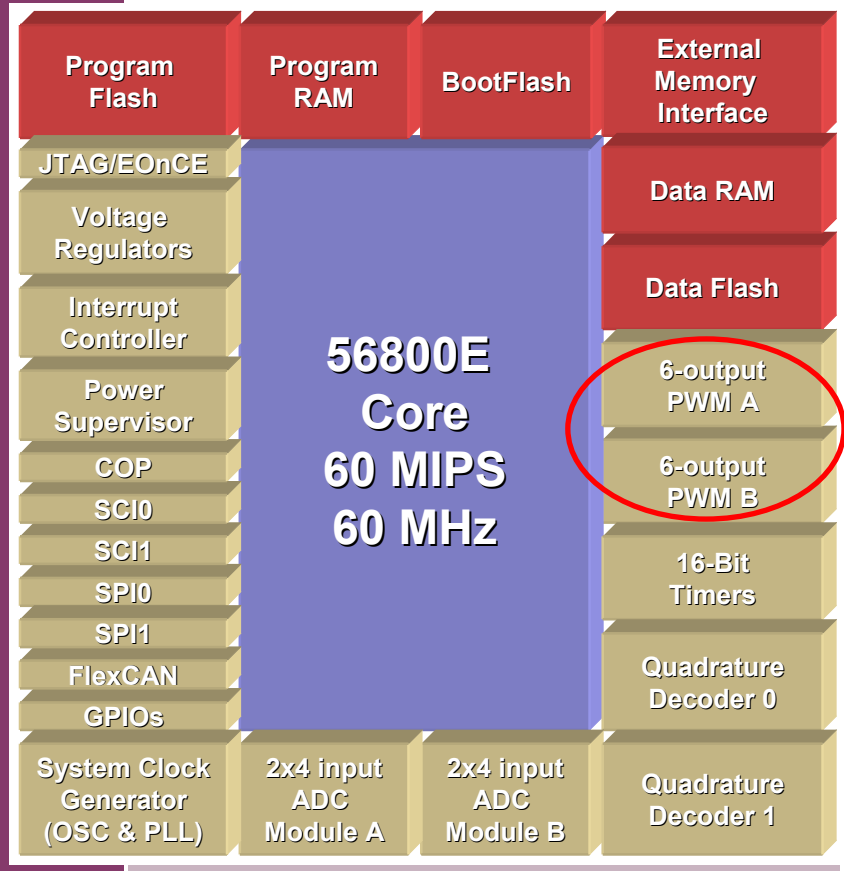
16 mA current sink capability
10 mA current source

Output Polarity Control

Write protected registers

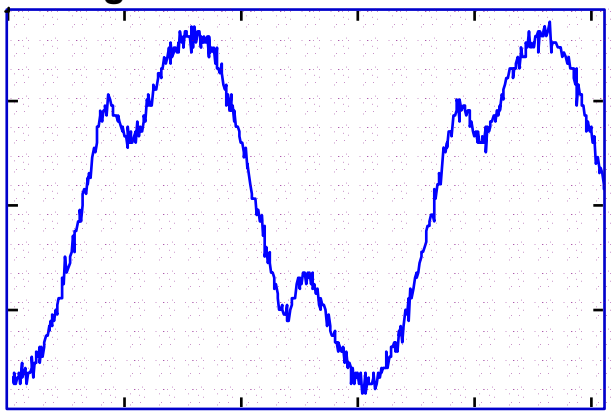
Double-buffered PWM register

Wait/Debug mode operation



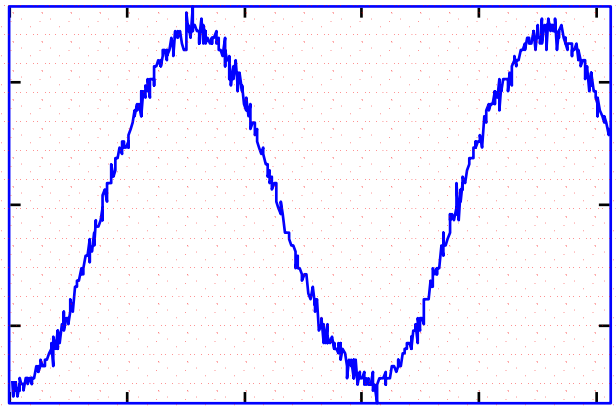
PWM Distortion Correction

Voltage with Correction Disabled



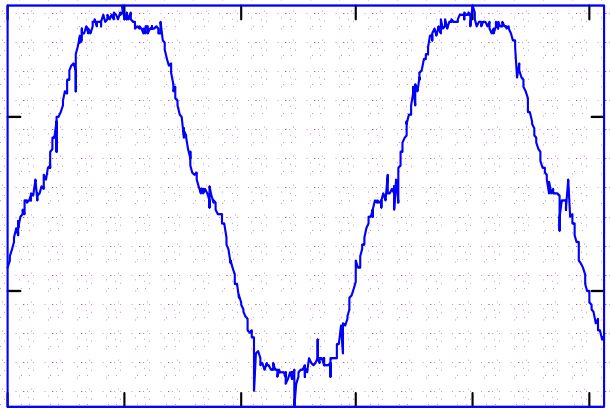
Motor Voltage

Voltage with Correction Enabled



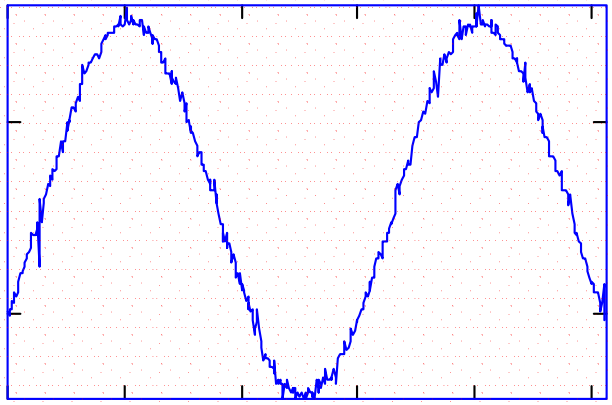
Quieter operation
Smoother operation
Less motor harmonic losses

Current with Correction Disabled



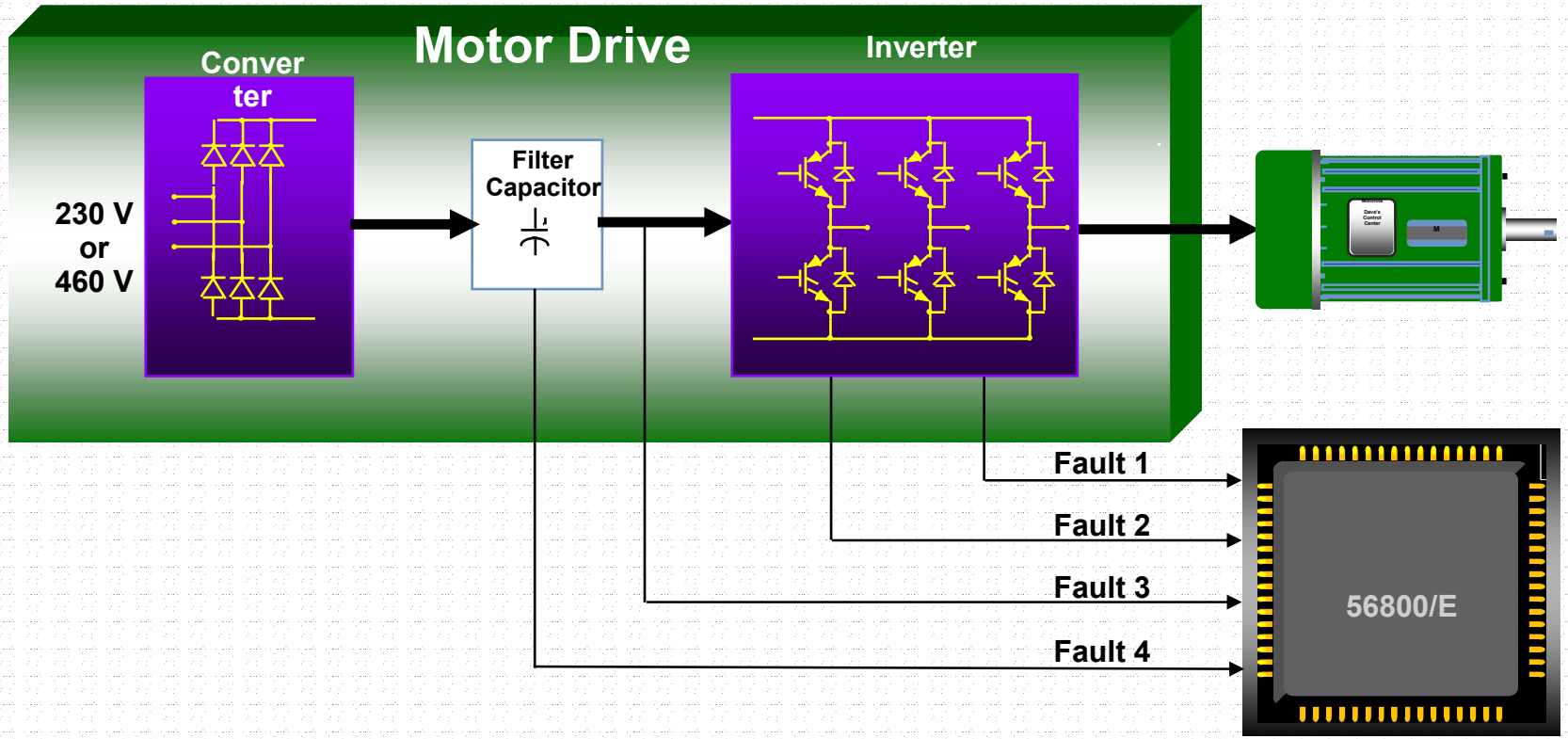
Motor Current

Current with Correction Enabled



Actual waveforms taken on a 1/2 horsepower motor

PWM – Multiple Fault Inputs

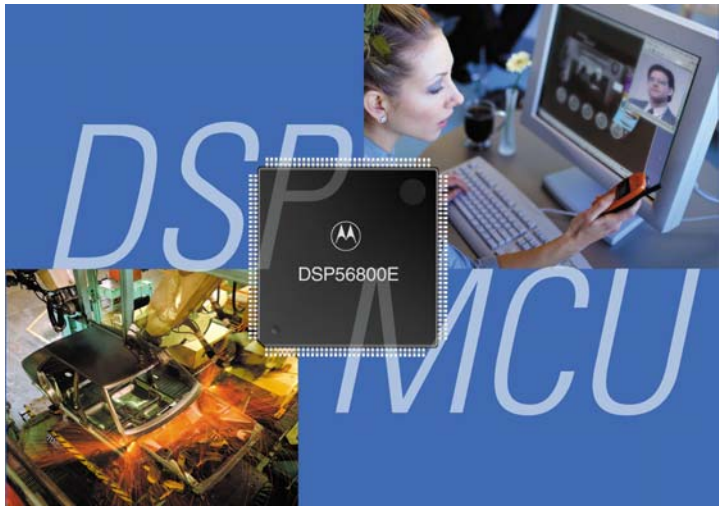


- Fault inputs can independently monitor critical system parameters, and generate an interrupt when asserted.
- Each input is mappable to immediately disable any or all PWMs
- Each input is programmable to allow Automatic or Manual PWM restart

56F8300 Differentiators

Q: How many independent fault inputs are available in the PWM?

Four programmable Fault inputs provide independent handling of faults within the system.



Launched by Motorola
freescale semiconductor

Quadrature Decoder

Four encoder inputs per decoder
- Phase A; Phase B; Index; Home

Captures all four transitions on two-phased inputs
- extracts actual shaft position and direction
- 32-bit position counter - initialized by software or external events
- Pre-loadable 16-bit revolution register

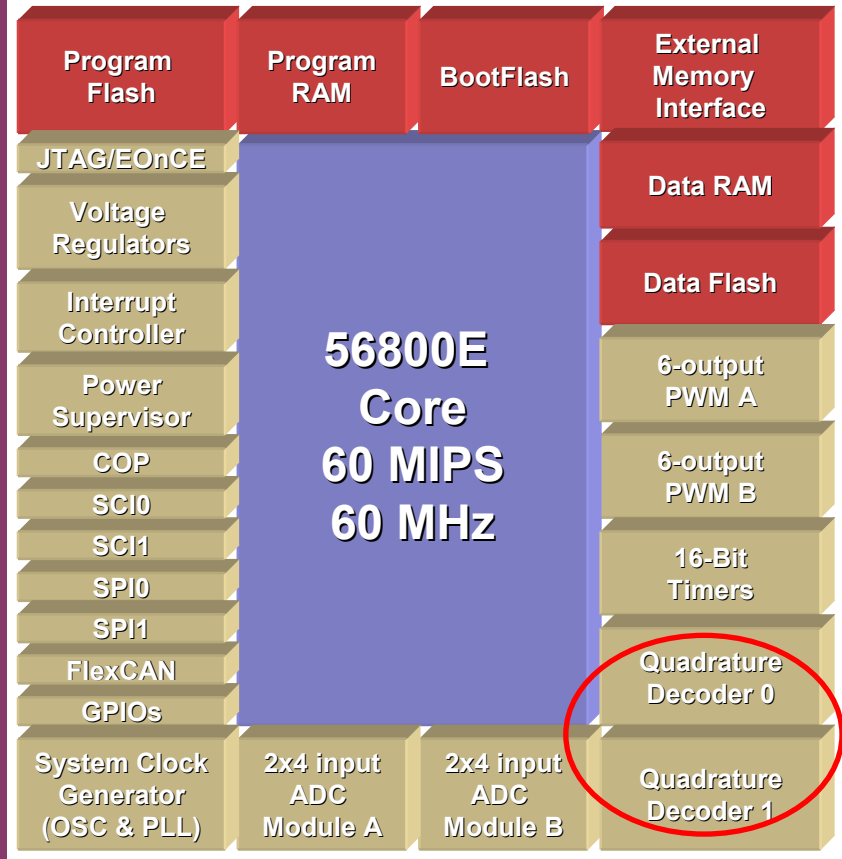
Index input
- resets the current integration value
- begins integrating a new revolution value

Home input
- Initializes position counter

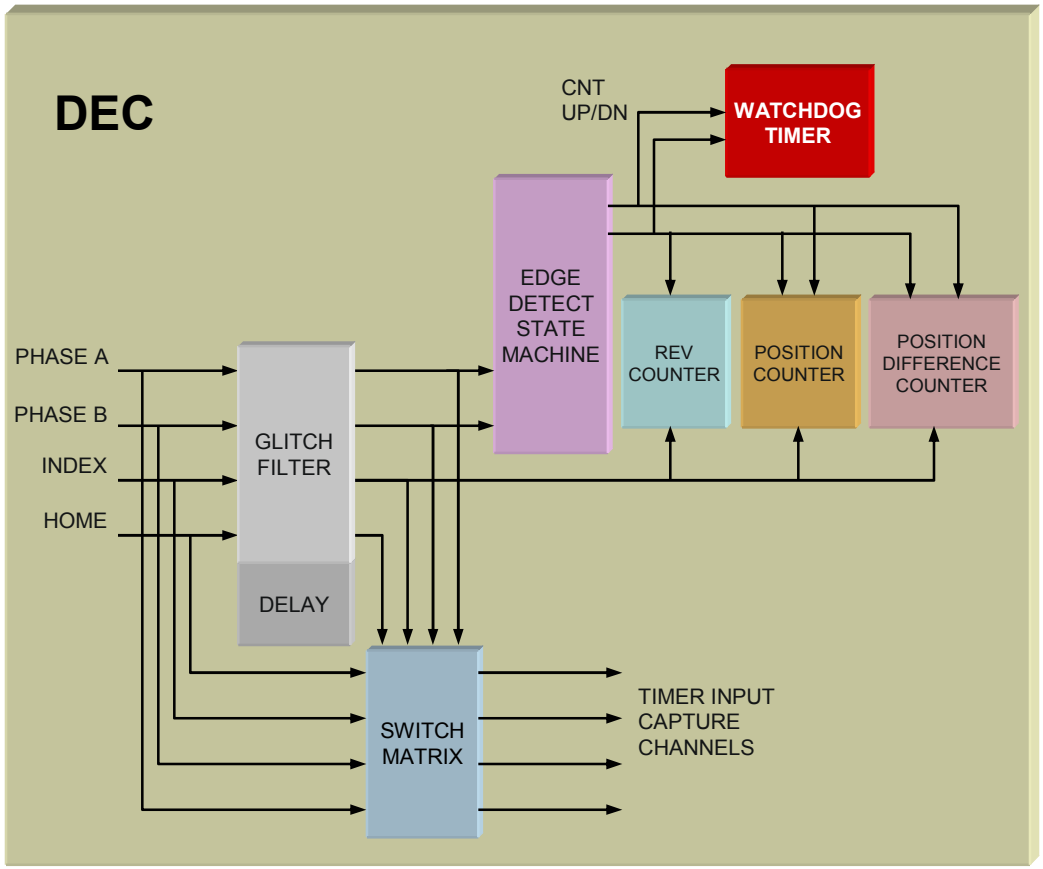
Configurable glitch filter for inputs

Can operate as single-phase pulse accumulators

Watchdog timer detects non-rotating shaft condition



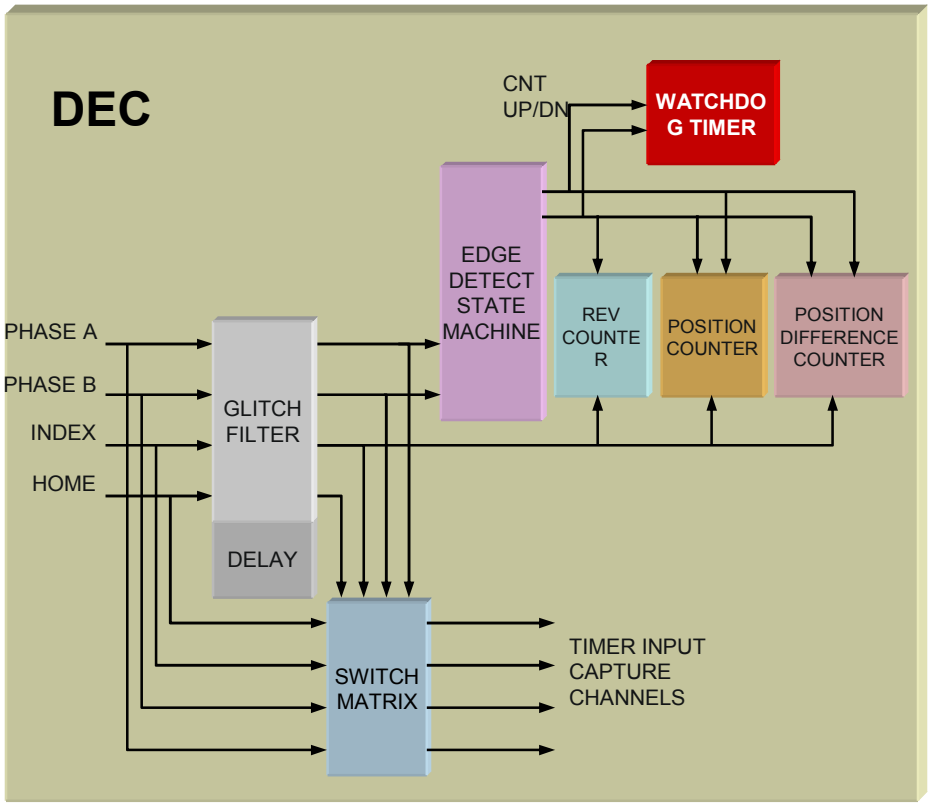
Quadrature Decoder - Block Diagram

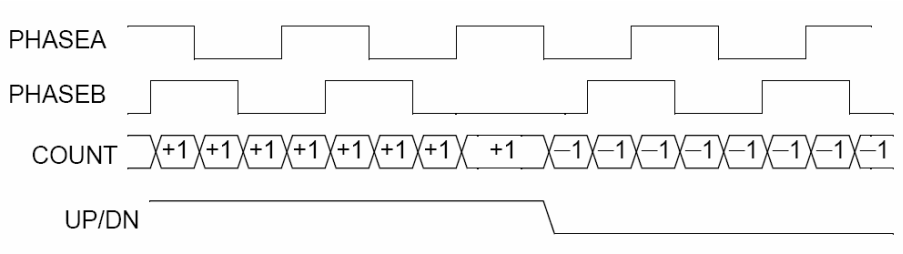


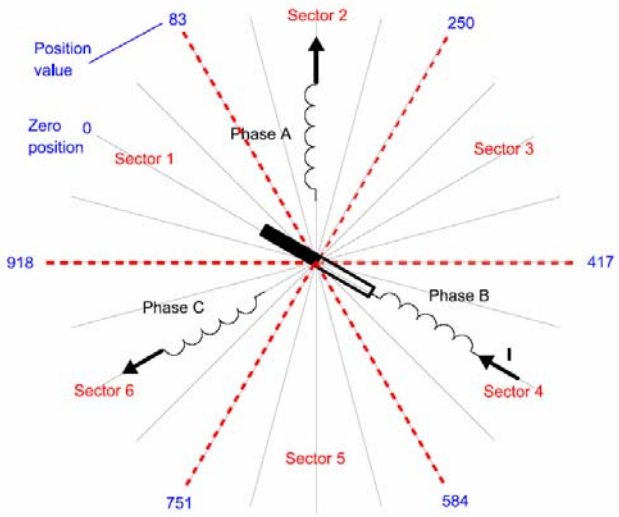
- ✓ Hold registers are associated with three counters:
  1. **Position**
  2. **Position difference**
  3. **Revolution**

  When any of the counter registers are read, the contents of each is written to its hold register. Taking a "snapshot" of the counters' values provides a consistent view of a system position and a velocity.
- ✓ **Glitch Filter**
  - ➢ This filter samples four time points on the signal and verifies the majority of samples are at a new state.
  - ➢ The sample rate of this filter can be adapted to a variety of signal bandwidths.
- ✓ **Edge Detect State Machine**
  - ➢ Identifies changes in the four possible states of the filtered PHASEA/B inputs, calculating the direction of motion.
  - ➢ These signals are routed into three up/down counters.
- ✓ **WATCHDOG Timer**
  - ➢ Detection of no rotation.
  - ➢ Two successive counts indicate proper operation and will reset the timer.

Quadrature Decoder for Motor Control



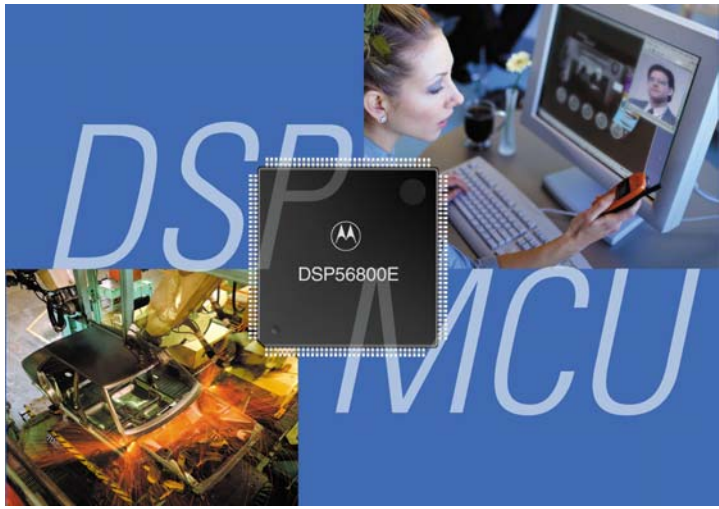




56F8300 Differentiators

Q: What is the differentiating feature of the Quad Decoder module?

The watchdog provides an interrupt after detection of two consecutive cycles of the non-rotating shaft condition.



Launched by Motorola
freescale semiconductor

CAN

Version 2.0B compliant

- **Standard and extended data frames**
- **0-8 bytes data length**
- **Programmable bit rate up to 1Mbps**
- **Support for remote frames**

"Time Stamp", based on 16-bit free-running timer

new **Two Serial Message Buffers for Buffer Frame**

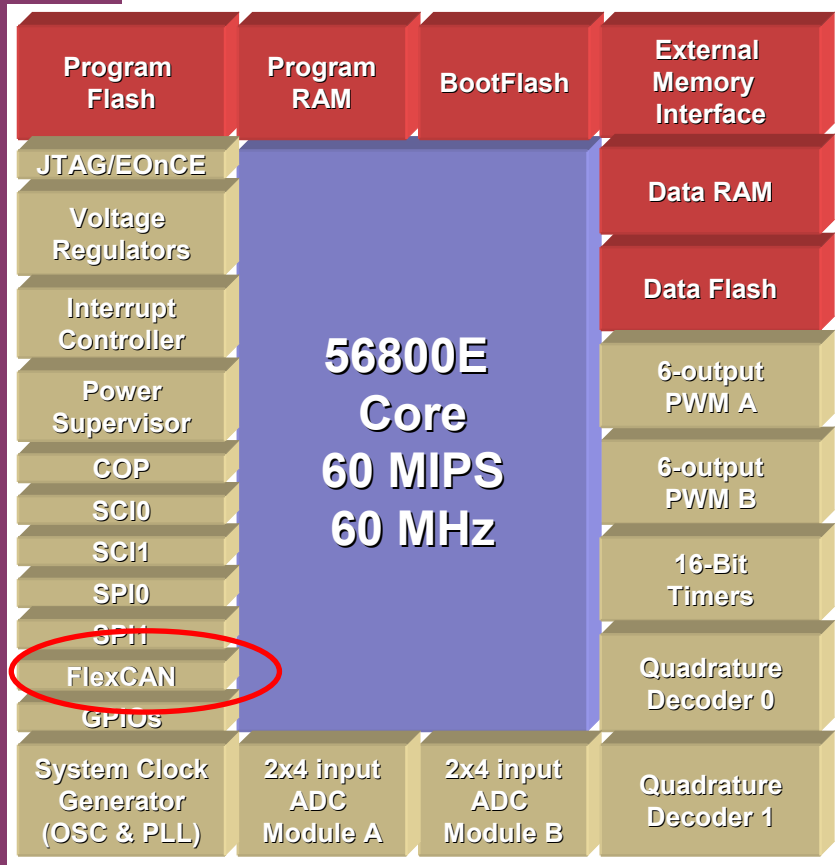
Sixteen Flexible Message Buffers of 0-8 bytes Data Length, each configurable as RX or TX, all support Standard and Extended Messages

new **Flexible maskable identifier filter**

Programmable wake-up functionality with integrated low-pass filter

Separate signaling and interrupt capabilities for all CAN RX/TxXrror states

Three low power modes



SCI

Operates as a Universal Asynchronous Receive and Transmitter (UART)

Full duplex operation provides simultaneous data transmit and receive

Half duplex operation allows data transmit and receive via single wire.

Separately enabled transmitter and receiver

13-bit baud rate selection

Standard mark/space non-return-to-zero (NRZ) format:

- Programmable 8-bit or 9-bit data format

Separate receiver & transmitter CPU interrupts

Programmable polarity for transmitter and receiver

Two receiver wakeup methods:

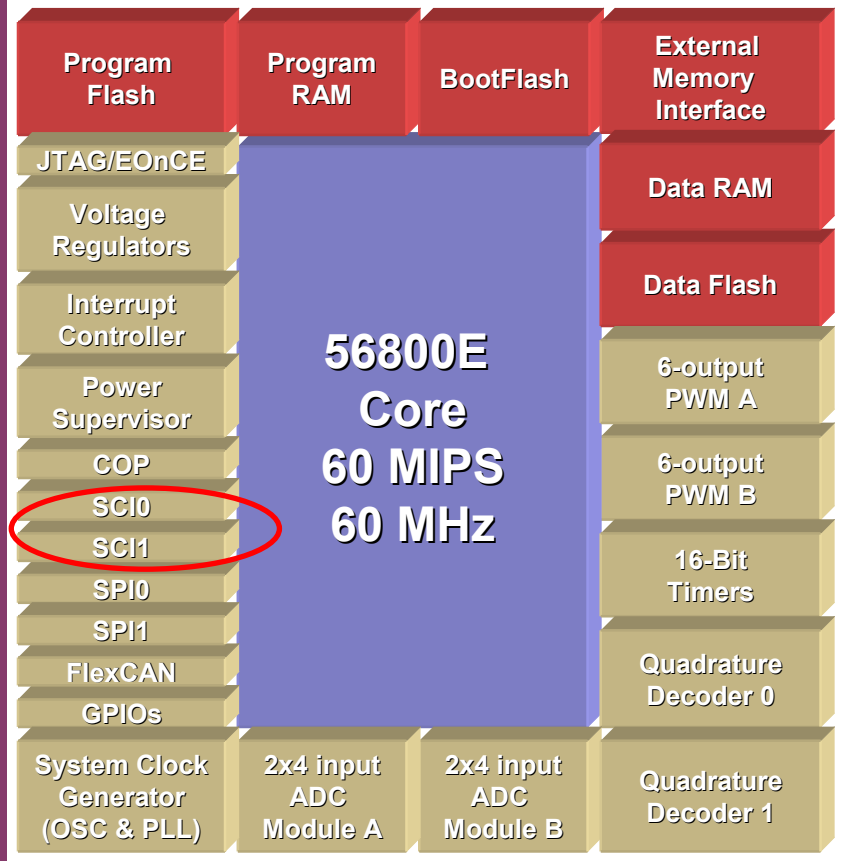
- Idle Line
- Address Mark

Interrupt-driven operation with eight flags

Receiver framing error detection

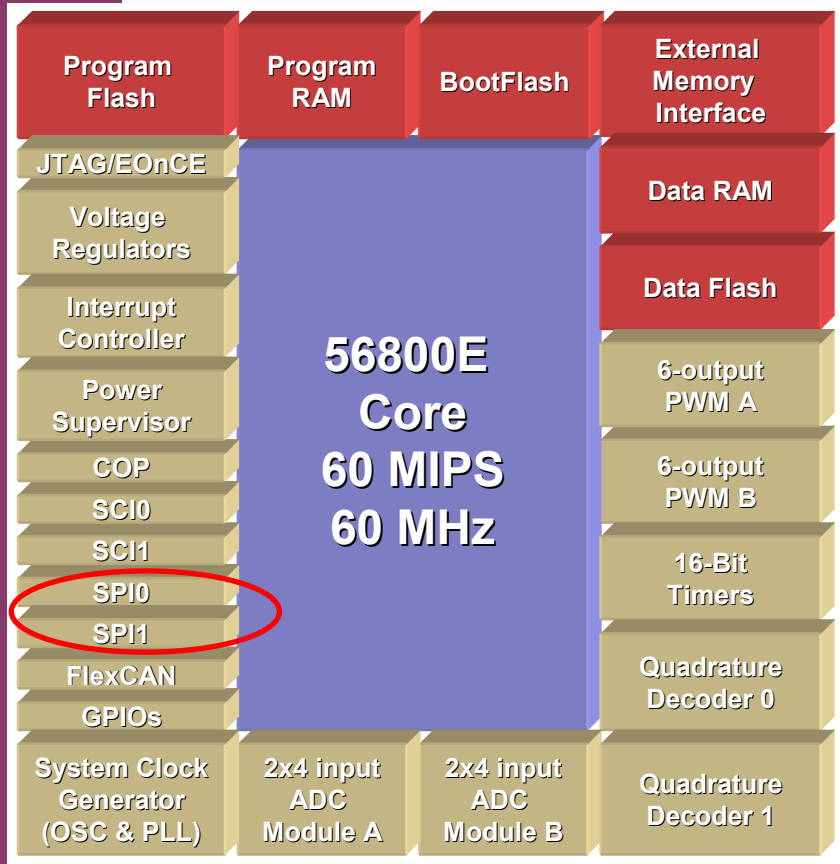
Hardware parity checking

1/16 bit-time noise detection



SPI

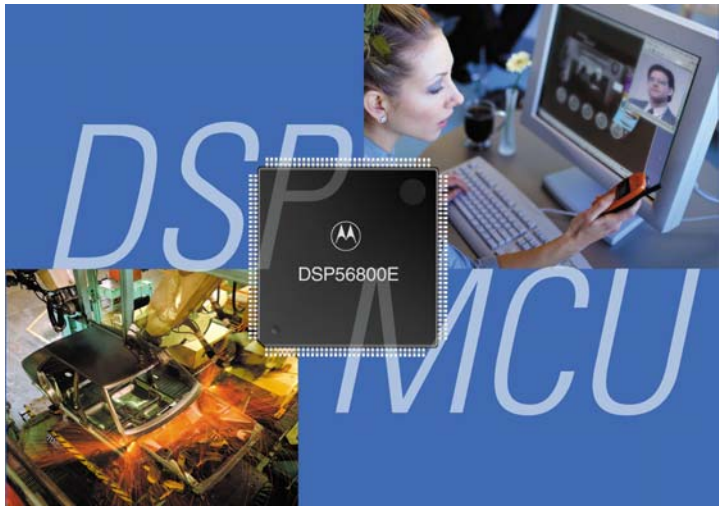
- Supports LCD drivers, A/D subsystems, & MCU systems
- Supports inter-processor communications in a multiple master system
- Supports demand-driven master or slave devices with high data rates
- Full-Duplex Operation
- Double-buffered Operation with separate transmit and receive registers
- Programmable length transmissions, 2 to 16 bits
- Programmable transmit and receive shift order, MSB or last bit transmitted
- Eight master mode frequencies (maximum = bus frequency ÷ 2)
- Maximum slave mode frequency = bus frequency
- Serial clock with programmable polarity and phase
- Two separately enabled interrupts
  - Receiver Full
  - Transmitter Empty
- Mode Fault and overflow error flag with interrupt capability
- Easy interface to Motorola's MCUs, Analog, and Sensors



56F8300 Differentiators

Q: What are the three types of serial communication peripherals available?

- CAN
- SCI
- SPI



Launched by Motorola
freescale semiconductor

Voltage Management & Power Supervisor

I/O drivers designed to interface at TTL compatible 3.3 V (5 V tolerant)

Two internal regulators available

- **One for device core (can be bypassed)**
- **One for analog circuitry (always enabled)**

Regulators converts 3.3 V input to 2.5 V core operating voltage

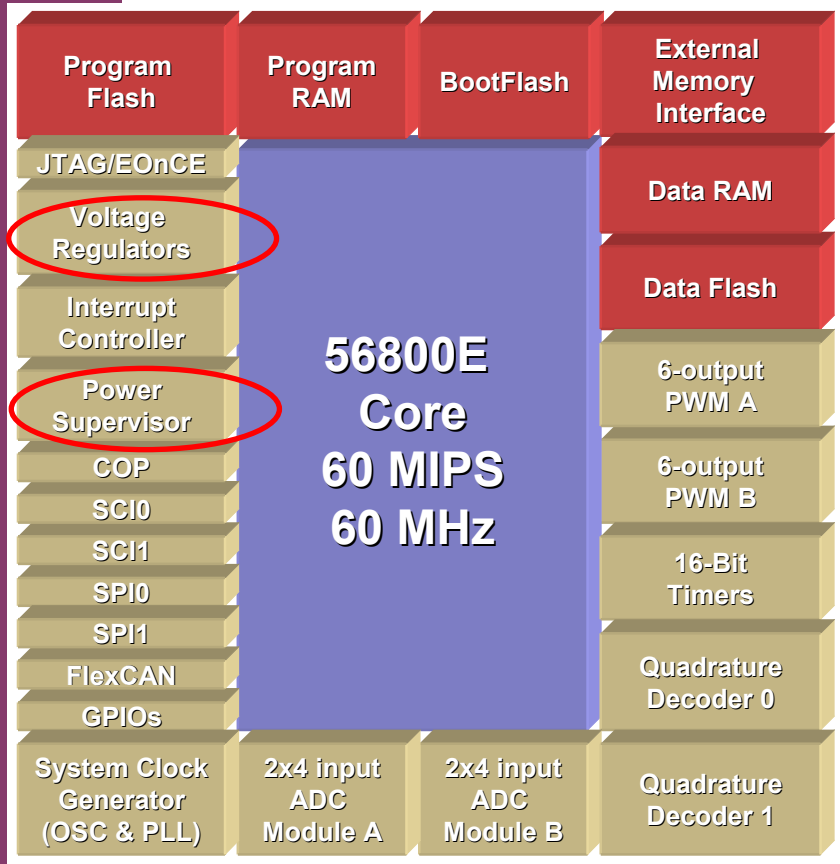
- **Reduces overall system cost**
- **Controls power usage**
- **Controls system noise floor**

Power Supervisor holds device in reset until there is enough voltage for on-chip logic to operate at the oscillator frequency

- **Precludes any problems associated with false restart**
- **Eliminates need for external power monitor**

Two Low Voltage Detect high-priority interrupts

- **Low voltage detect signals used to initiate a software controlled shutdown when the supply voltage drops below acceptable levels**



On Chip Clock Synthesis (OCCS)

Four different, dynamically selectable system clock sources available

- Crystal Oscillator - driven by external crystal or clock source
- Relaxation Oscillator – On-chip Oscillator (56F8322/323 only)
- Programmable 4-bit Prescaler - divide-by of the IP Bus clock
- Phase Locked Loop (PLL) - generates output frequencies of up to 60 MHz

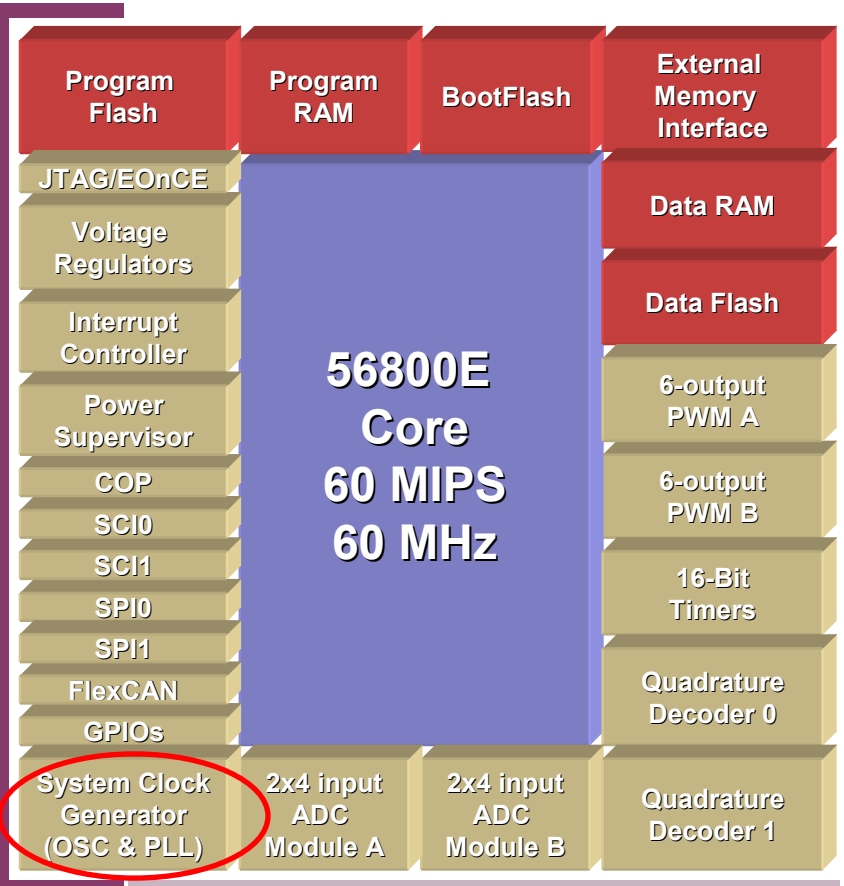
Selectable Phase Locked Loop input clock source

- Crystal Oscillator
- Relaxation Oscillator (+/- 2.5% accuracy over temperature after trimmed)
- Programmable 4-bit Prescaler

Dynamically programmable PLL allows configurable power/speed options

Generates an interrupt if either loss of clock ,or loss of lock, or both

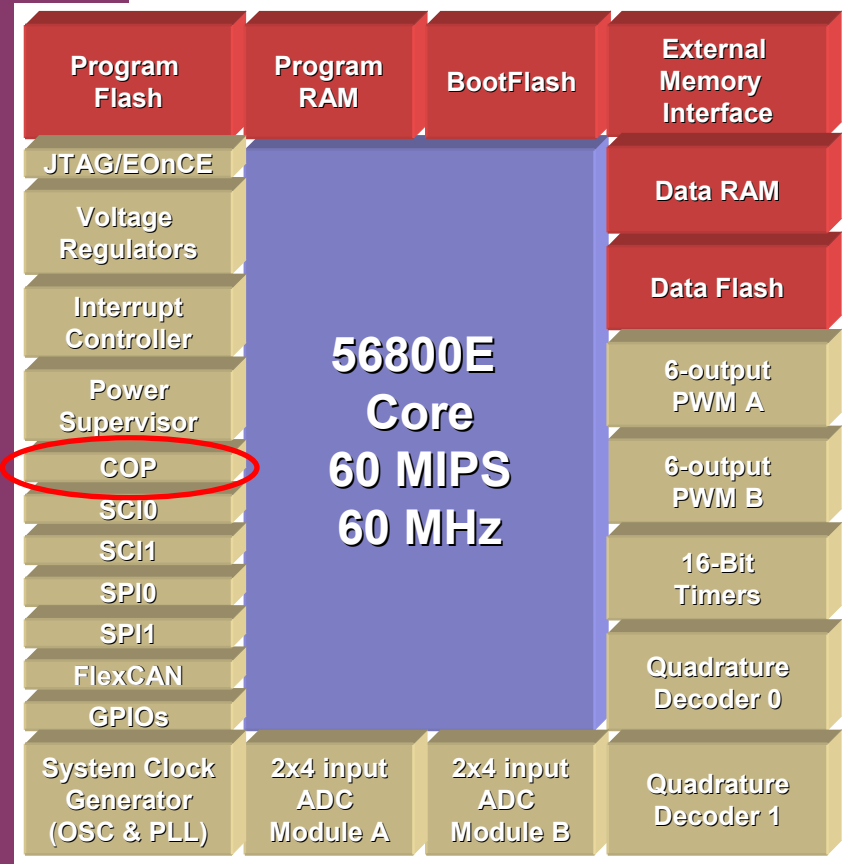
Five clock output pins



Computer Operating Properly (COP)

Allows for detection of application software that may be operating incorrectly.

Resets the part if not properly serviced.

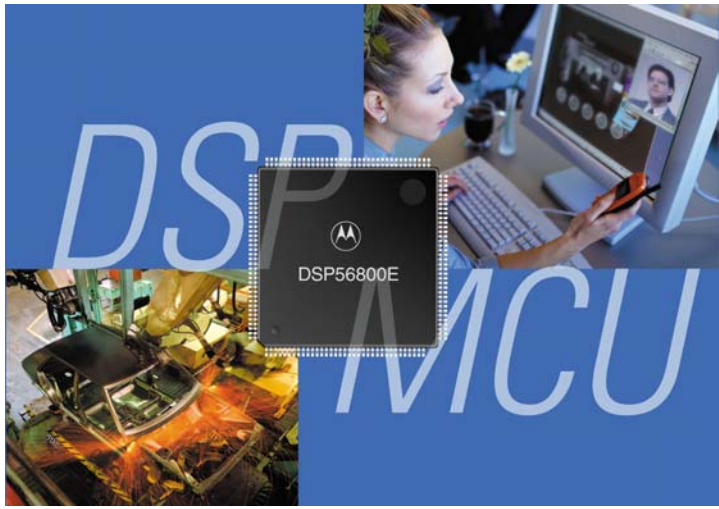


56F8300 Differentiators

Q: How can the Voltage Management and Power Supervision modules reduce your system costs?

No need to supply multiple source voltage regulation circuitry

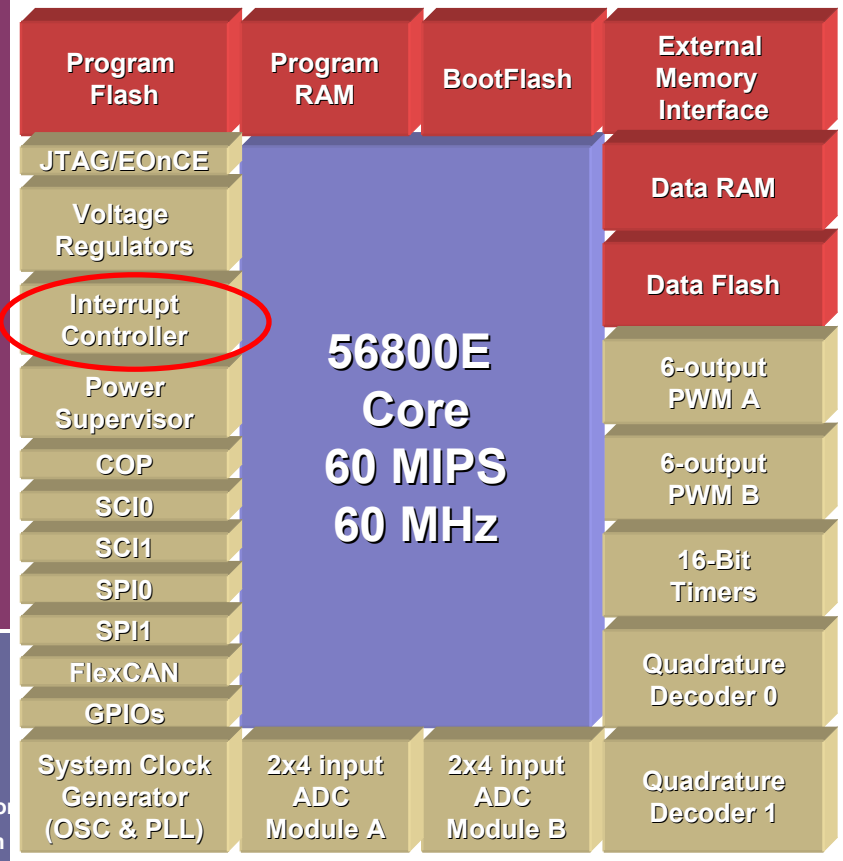
No need for external POR circuitry



Interrupt Controller

- Arbitrates peripheral interrupt requests
- Signals core when an interrupt of sufficient priority exists
- Provides ISR address to service each interrupt
- Relocatable Interrupt Vector Table
- Supports 128 interrupt sources
- Supports 5 interrupt priority levels with HW nesting
  - Level # 3 (highest) for core interrupts
  - Levels # 2, # 1, and # 0 for peripherals
  - Level # -1 (lowest) for SW interrupt

Self-paced training available on Interrupt Handling



new

Two Fast Interrupts

Available for Level 2 interrupts

Dedicated context registers:

- R0, R1, N, M01 – Shadow Regs
- Y – Data Reg
- FIRA – Fast Interrupt Return Address Reg saves PC
- FISR – Fast Interrupt Status Reg saves SR and NL bit

Lower latency

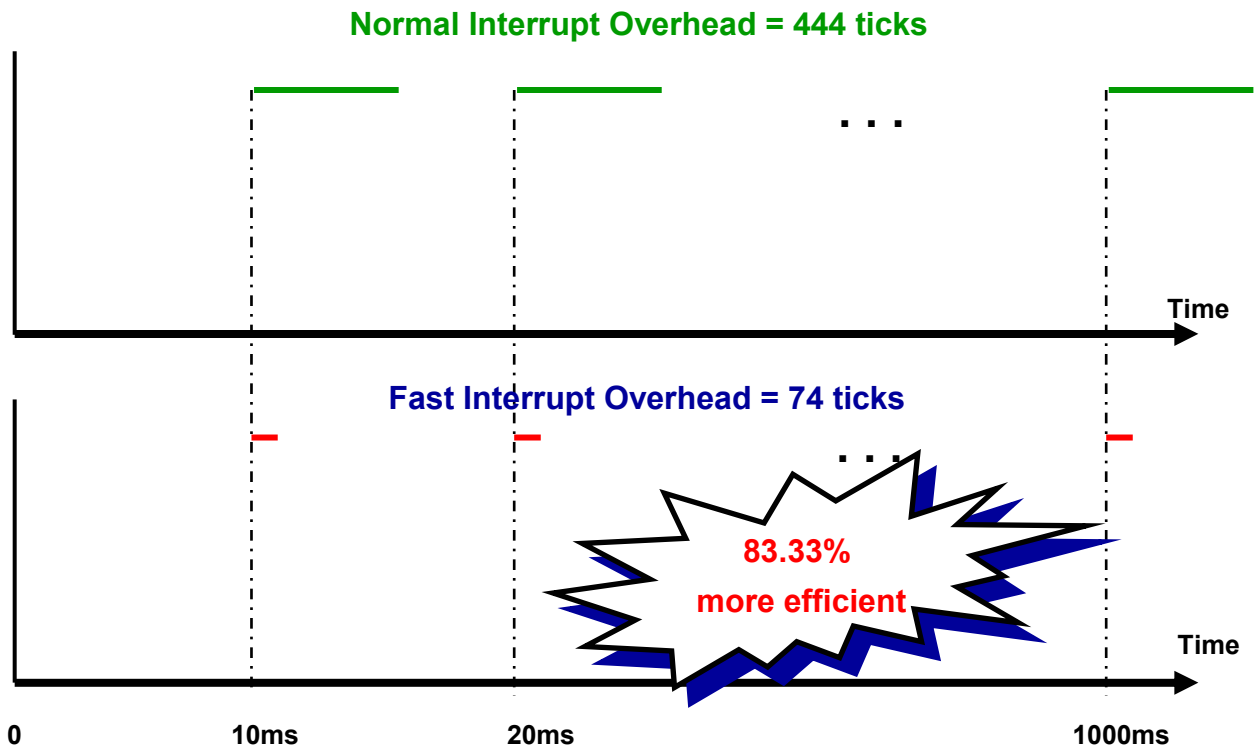
- Bypass jump table
- Bypass context save/restor
- FRTID - lower latency than

Launched by Motorola freescale semiconductor

Fast Interrupts Demonstration

Execute “FastIntDemo.mcp” project (for 56800E devices only)

Applications executes Fast and Normal interrupt every 10ms, and calculates associated CPU overhead in clock ticks



Fast Interrupt takes 83.33% less time to complete.

More time for application processing!

(444 – 74) / 444 = .8333

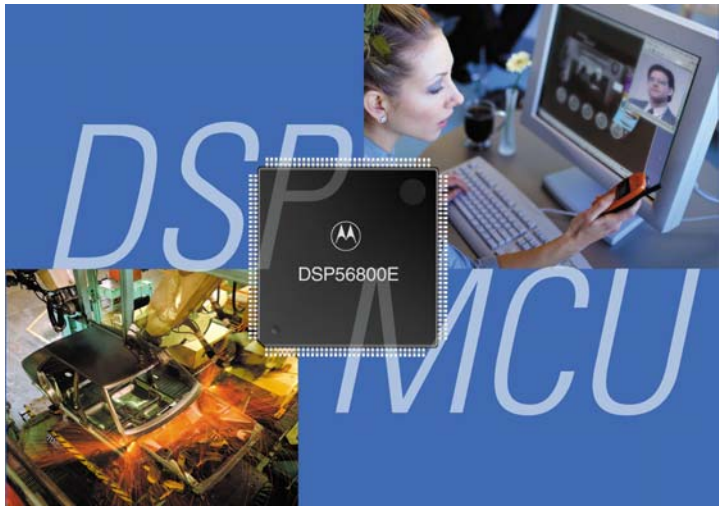
56F8300 Differentiators

Q: Why are Fast Interrupts faster than normal interrupts?

Shadow Registers automatically loaded and restored by hardware; no need to perform a context save

By-Passes the Vector Table to begin ISR execution.

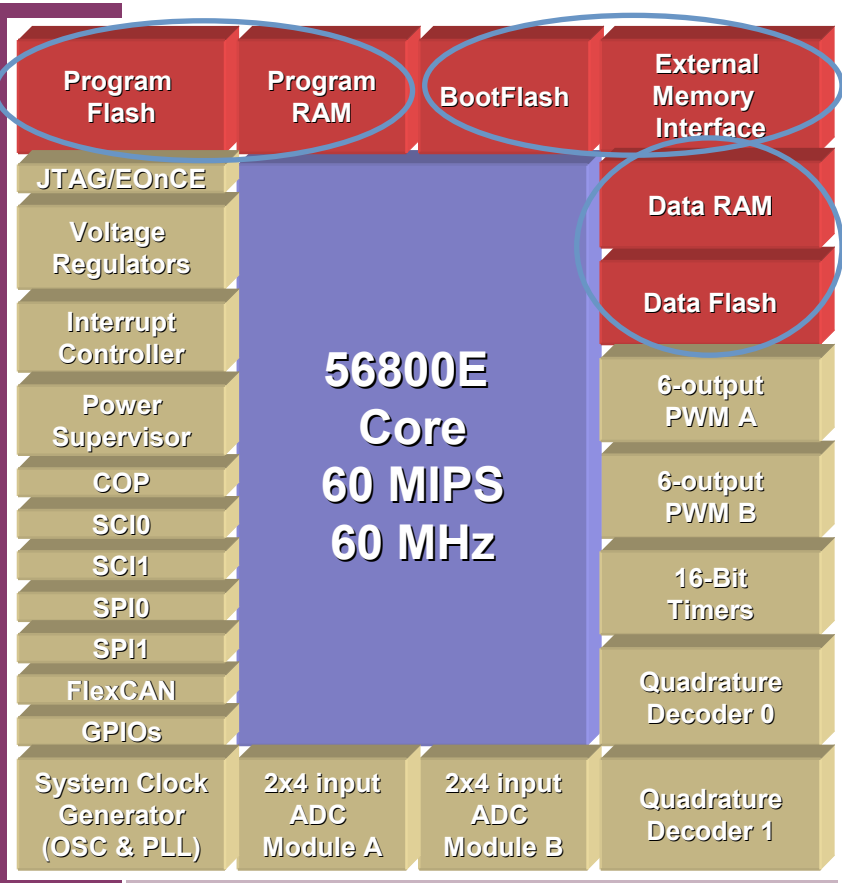
Dedicated hardware registers (FIRA, FISR)



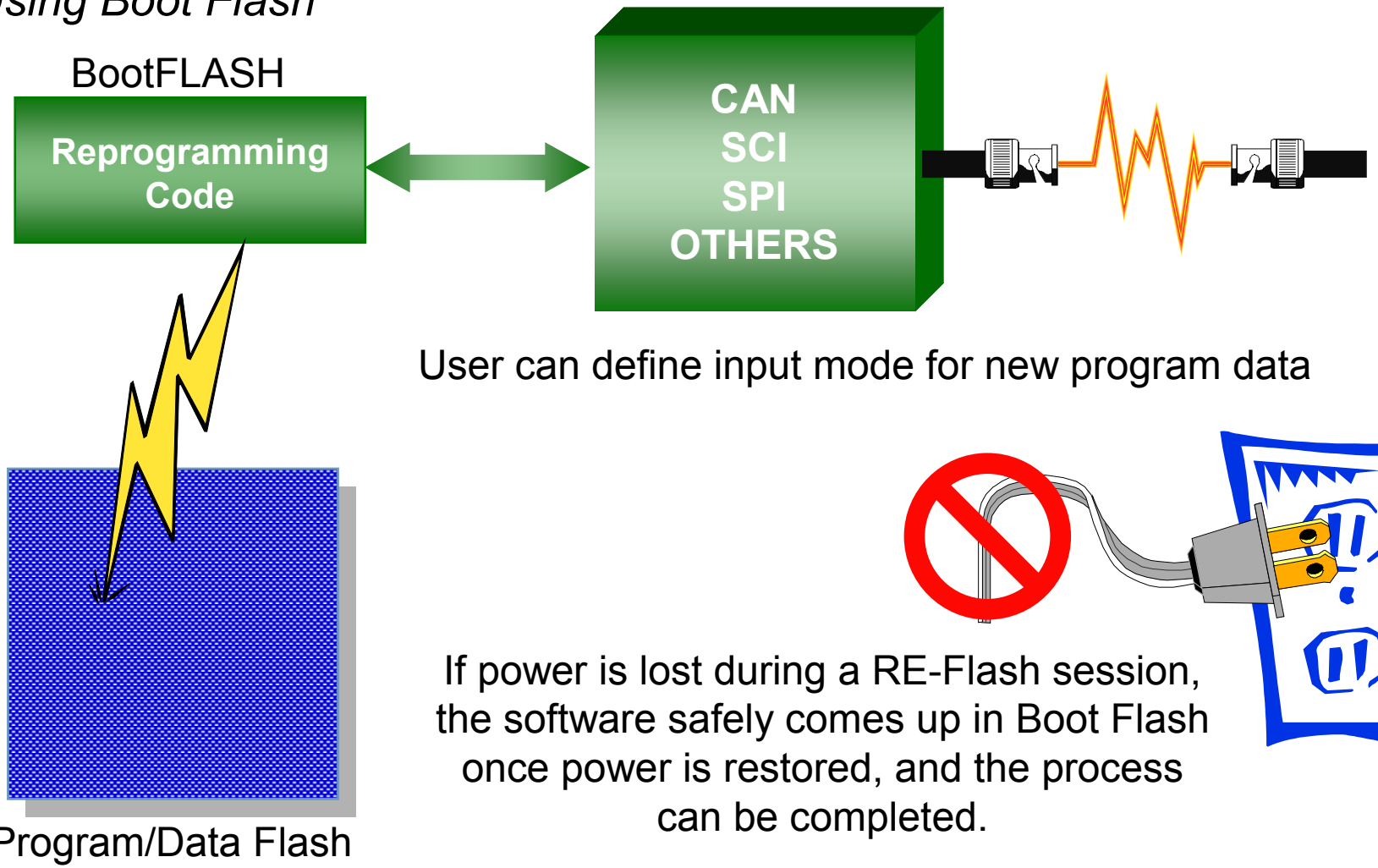
Launched by Motorola
freescale semiconductor

Memory

- On chip Harvard Architecture
  - Separate program and data buses
  - Permits up to three simultaneous accesses to program AND data memory
- On chip and Off-chip memory
- 8K bytes of BootFLASH™
- Programmable “Code Protection” feature
- Programmable “Code Security” feature
- Flash with 256/512 word page size for data/program
- Flash memory programmable via JTAG/OnCE interface or user defined programming (such as SPI, SCI, CAN)
- Programmable wait states for low cost system memory solutions
- Can program one word at a time
- 60MHz operation at 125ºC
- EEPROM emulation (HW & SW support)



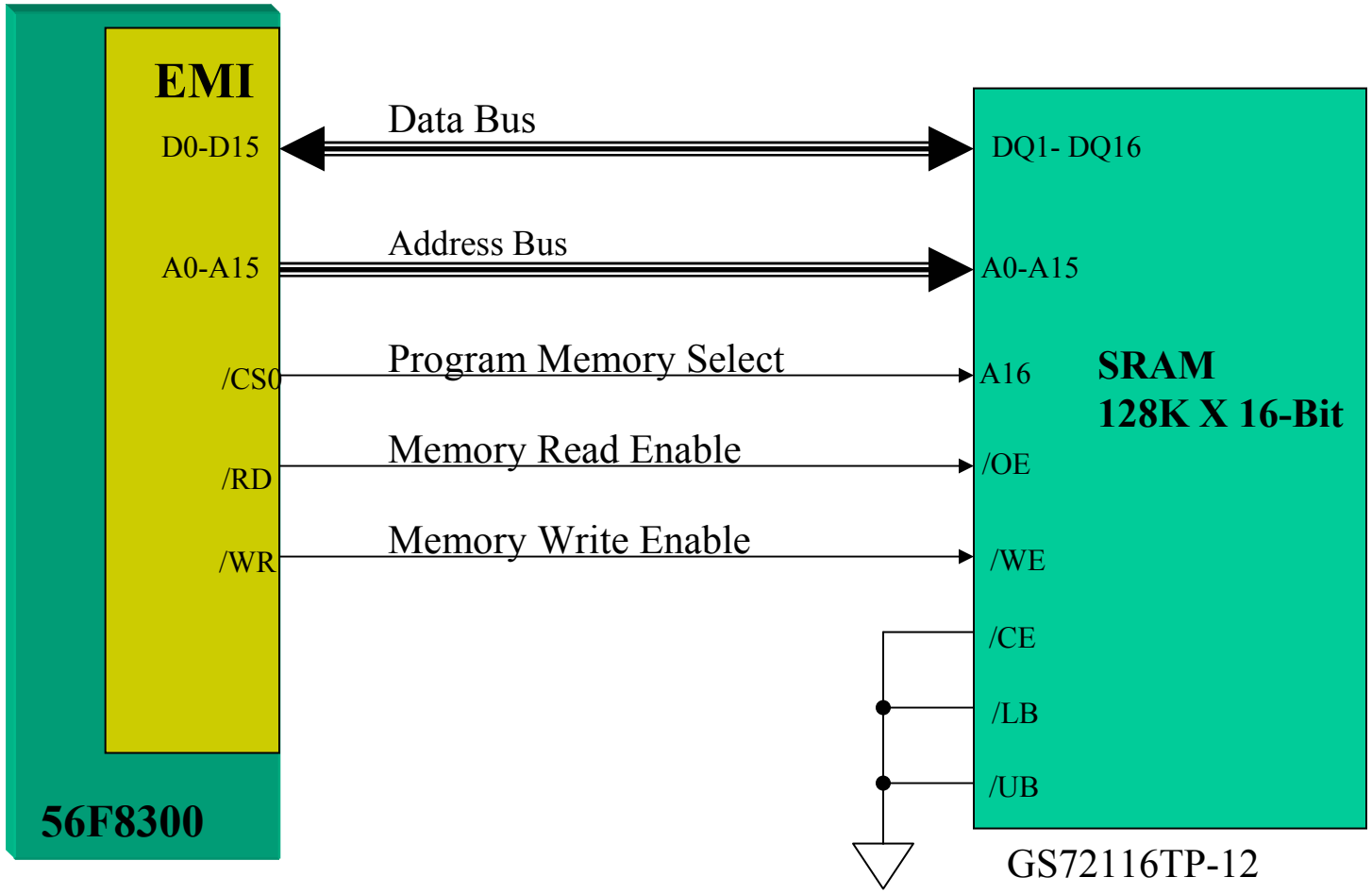
Using Boot Flash



User can define input mode for new program data

If power is lost during a RE-Flash session, the software safely comes up in Boot Flash once power is restored, and the process can be completed.

External Memory Interface



*First 64 K words assigned for program memory
*Second 64 K words assigned for data memory

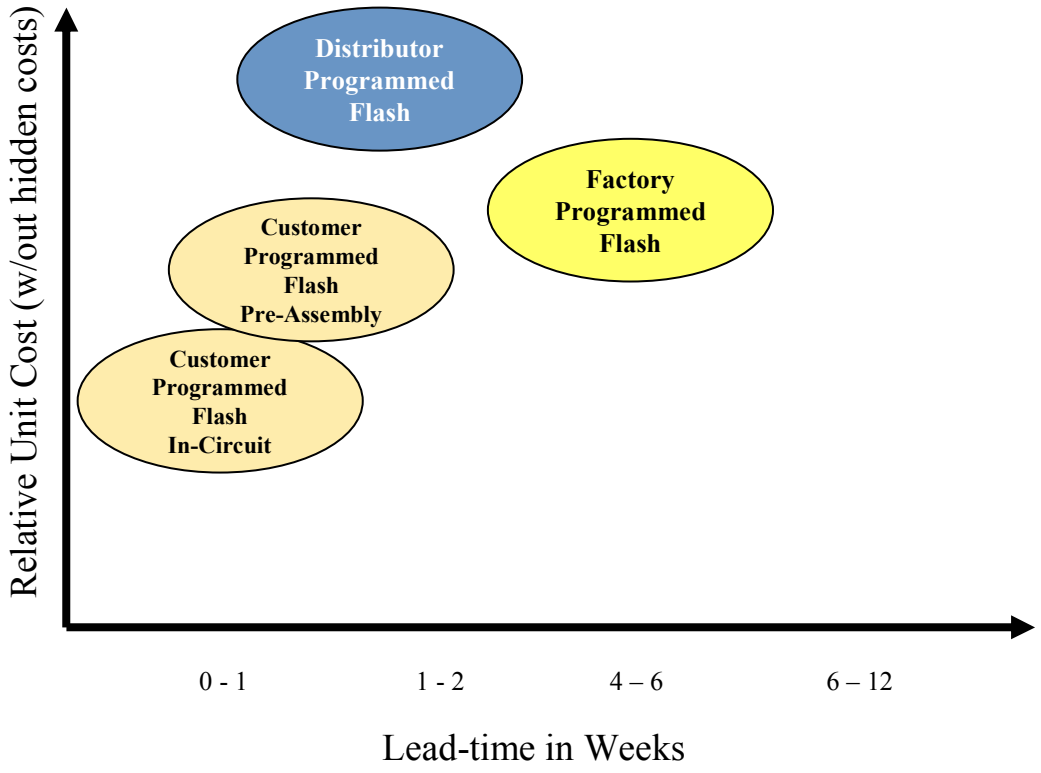
Flash Programming Solutions

JTAG/EOnCE Port

- 800 W/sec
- MetroWerks CodeWarrior
  - Parallel port
- BP Microsystems
  - Bulk Device Programmer
- Domain Technologies
  - Assortment of communication interfaces
  - Custom off-chip Flash
- Flash over JTAG Utility
  - Source code available

Distributor Programming

- Preassembly programming
- See local distributor



Serial/CAN Bootloader

- Serial: 2.6 K W/sec
- CAN: 3.7 K W/sec
- Source code available

Parallel Programming Mode

- 91 K W/sec (Program)
- 47 K W/sec (Boot/Data)
- Requires NDA and factory support

Factory Programming

- Lead Time
- Fee based
- Min qty req

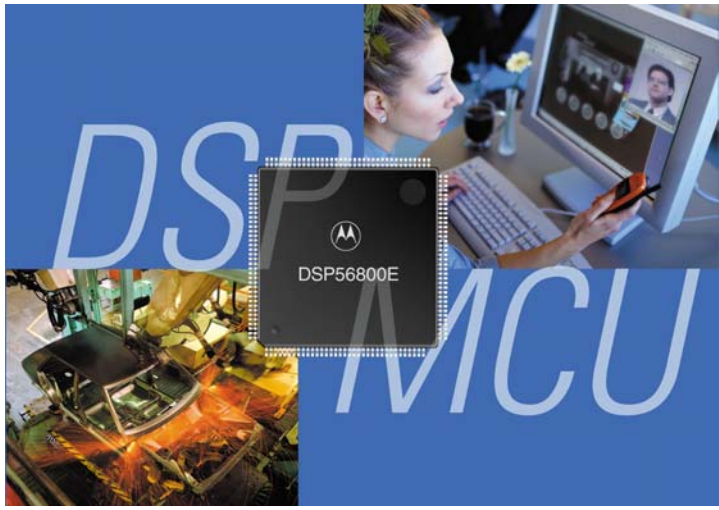
Launched by Motorola freescale semiconductor

56F8300 Differentiators

Q: What is the difference between flash protection and flash security?

Flash Protection prevents the protected flash sections from being erased/programmed

Flash Security prevents the contents of flash from being viewed to protect customer IP.



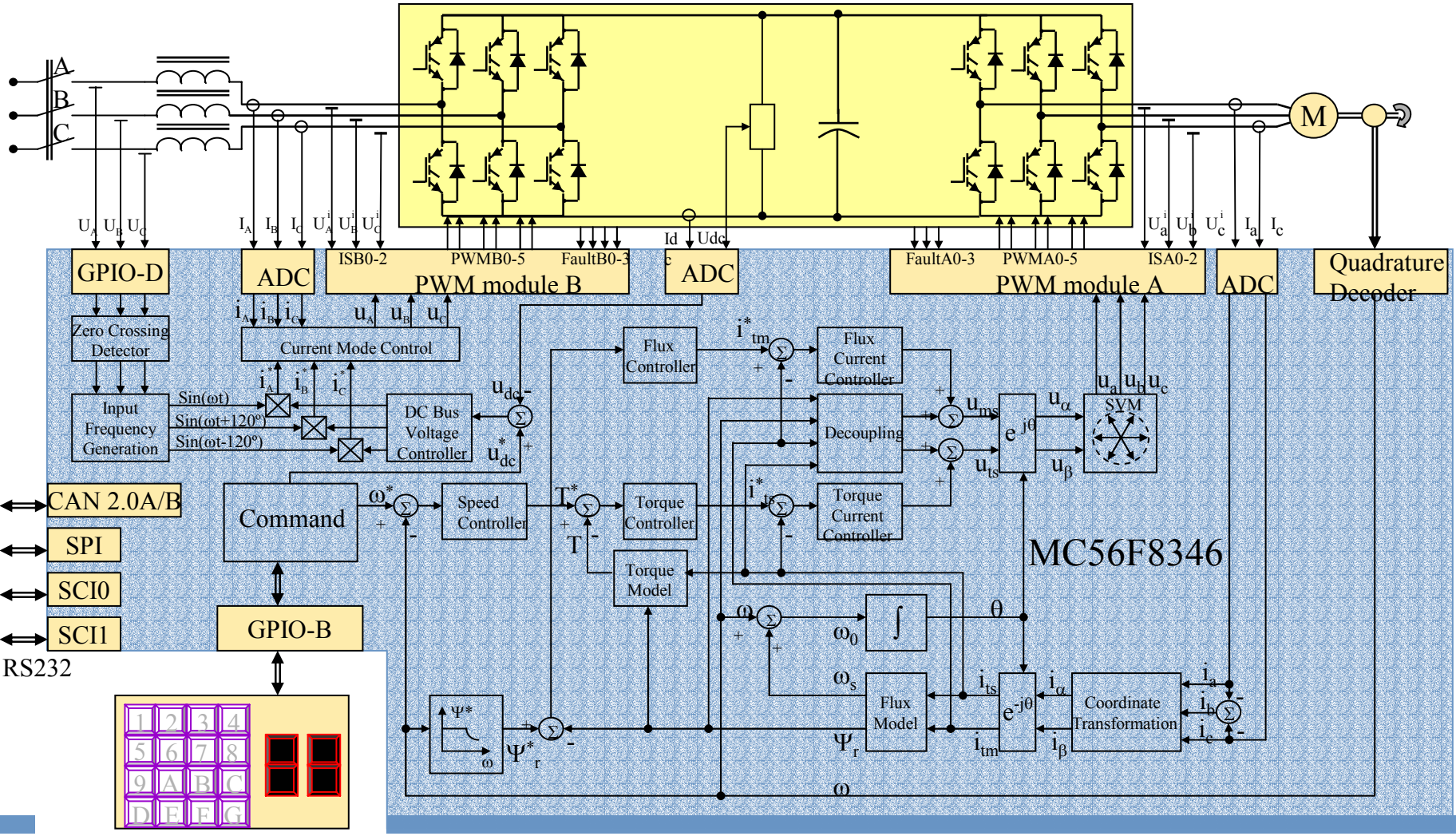
Launched by Motorola
freescale semiconductor

Hybrid MCU Architecture I

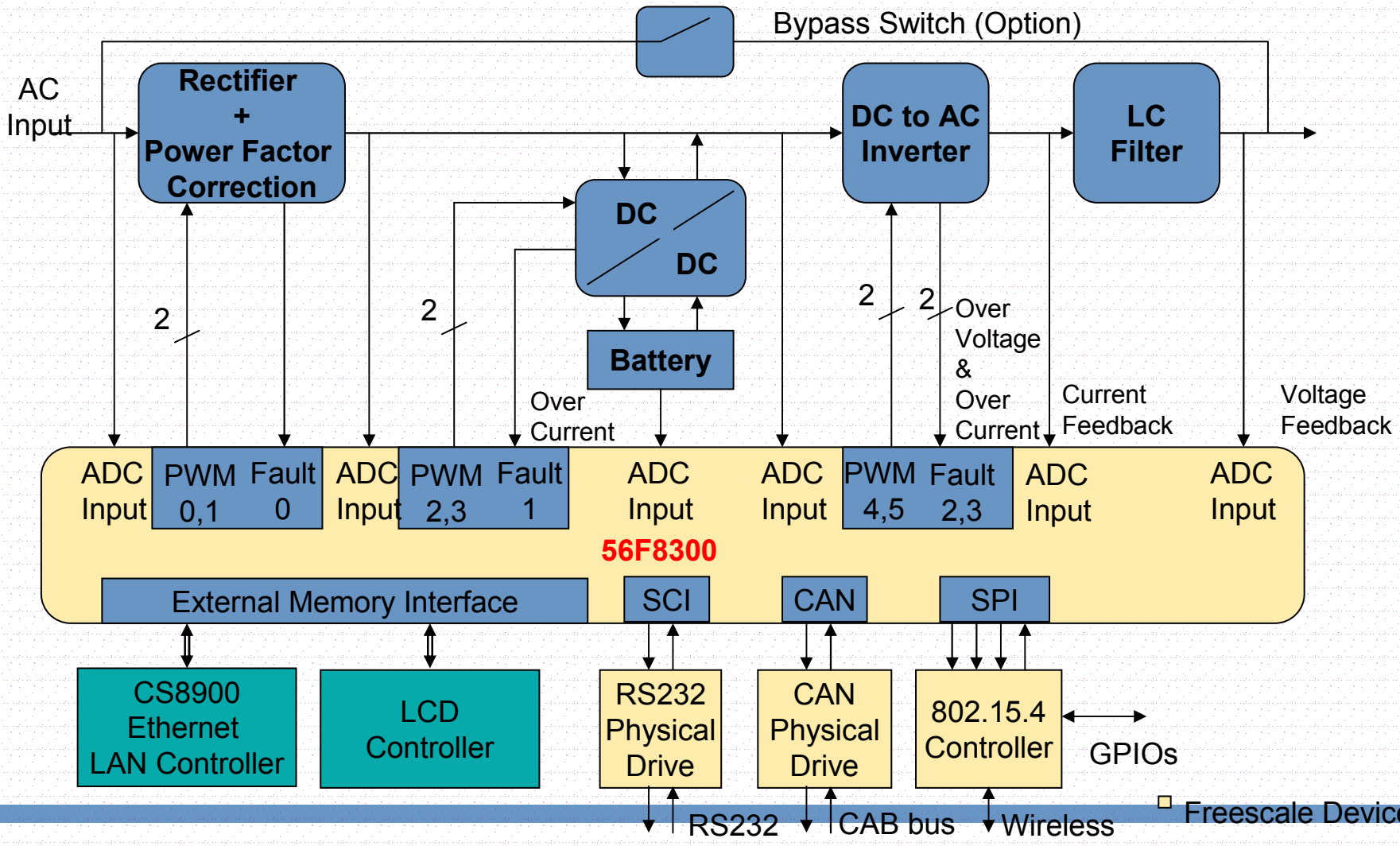
56F8300 Integrated Peripherals

Launched by Motorola
freescale™
semiconductor

Field Oriented Motor Control Application



UPS Application



Analog to Digital Converter

12 bit resolution

Up to two ADC modules
- Two ADCs per Module
- **6 to 8 Analog inputs per ADC Module**

Sampling rate up to
1.66 million samples per second

Sequential conversions: first 1.7 μsec subsequent 1.2 μsec

Simultaneous conversions: 8 in 5.3 μsec

Can be synchronized with
Pulse Width Modulators (PWM)

Simultaneous or sequential sampling

Eight words result buffer

Sample correction via programmable offset

Current injection protection

new

Software self calibration capability
- Removes gain and offset errors

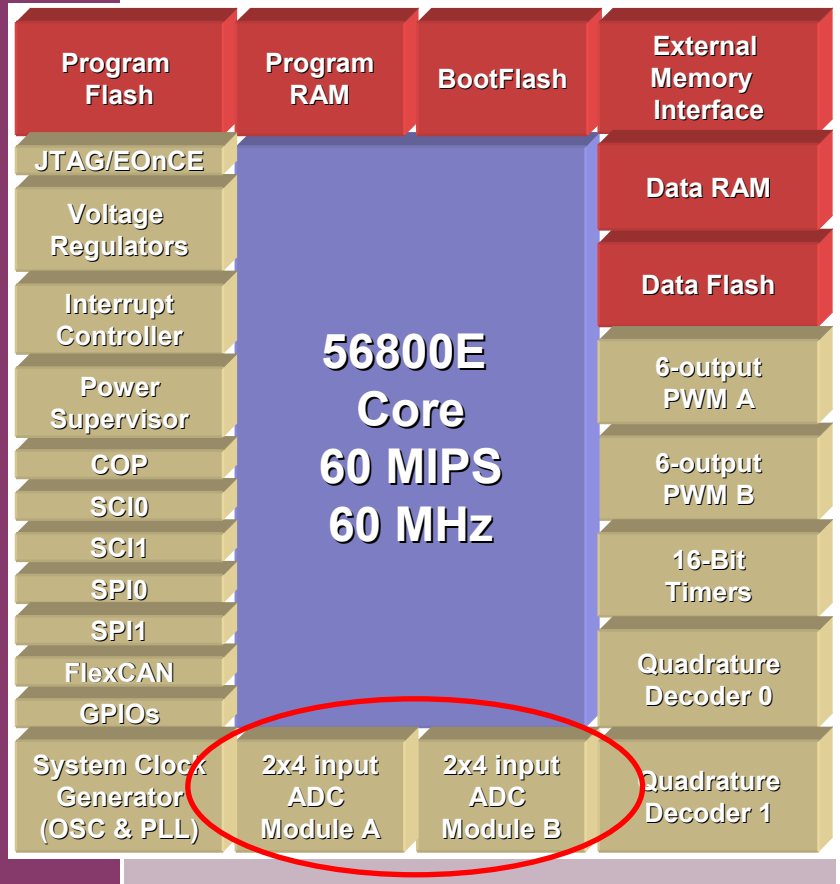
new

Interrupt generating capabilities
- End-of-Scan; Zero crossing; High/Low limit check

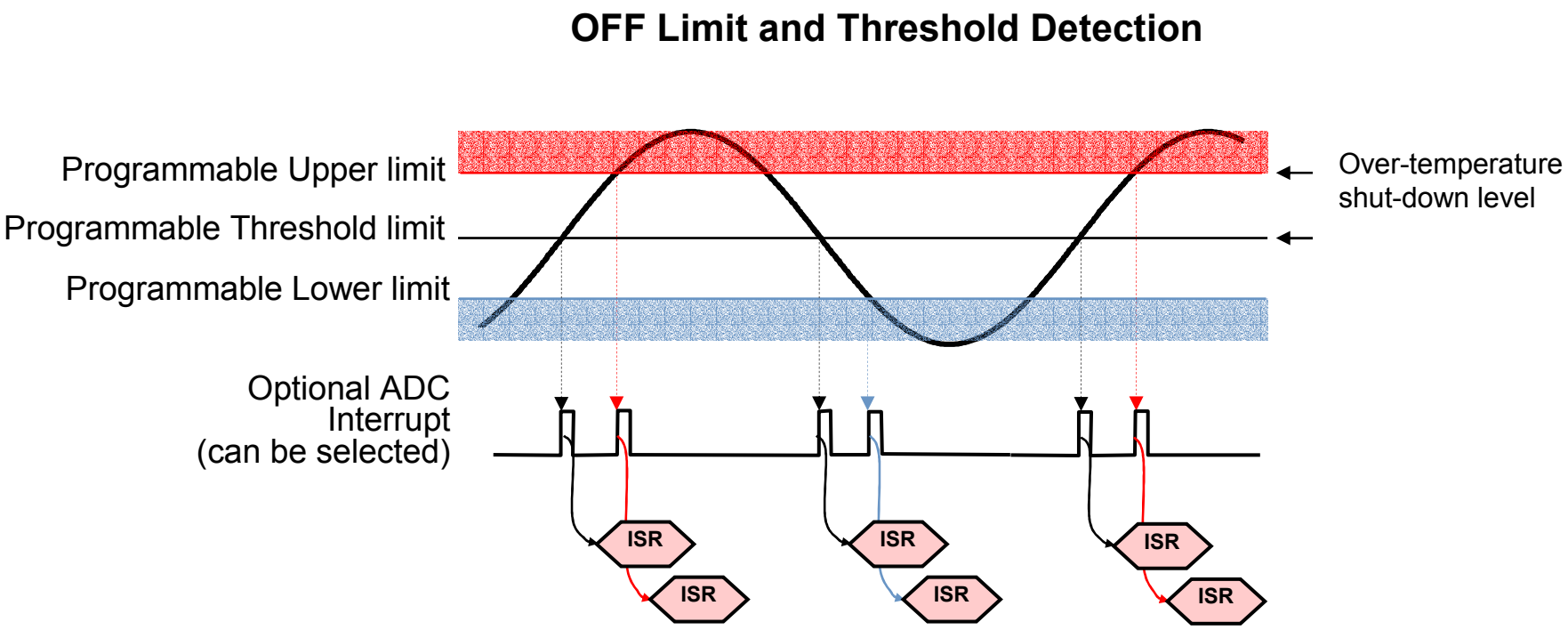
Two outputs formats available
- Two's complement
- Unsigned

Power down and power savings modes
- Explicit power down
- Intelligent power savings mode: powers up only when needed

new

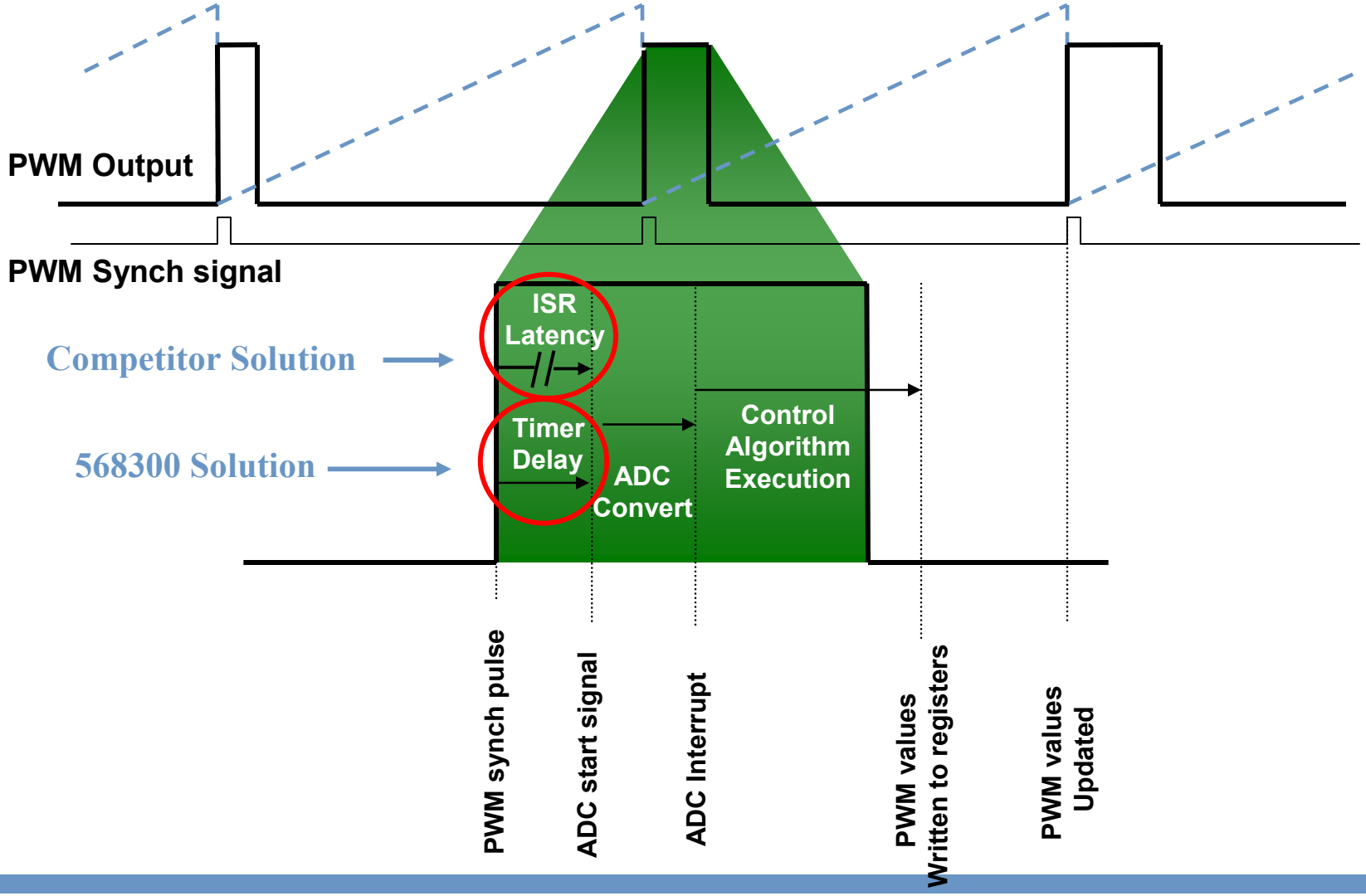


ADC Sampling Process



- ✓ The ADC can perform limit checking and zero crossing detection with NO CPU intervention.
- ✓ Each channel has its own upper, lower, and threshold comparators.

ADC Synchronization with the PWM



ADC Modes

Once

- **The ADC starts to sample just one time whether you use the START bit or by a sync pulse. This mode must be re-armed by writing to the ADCR1 register again if you want to go capture another scan**

Triggered

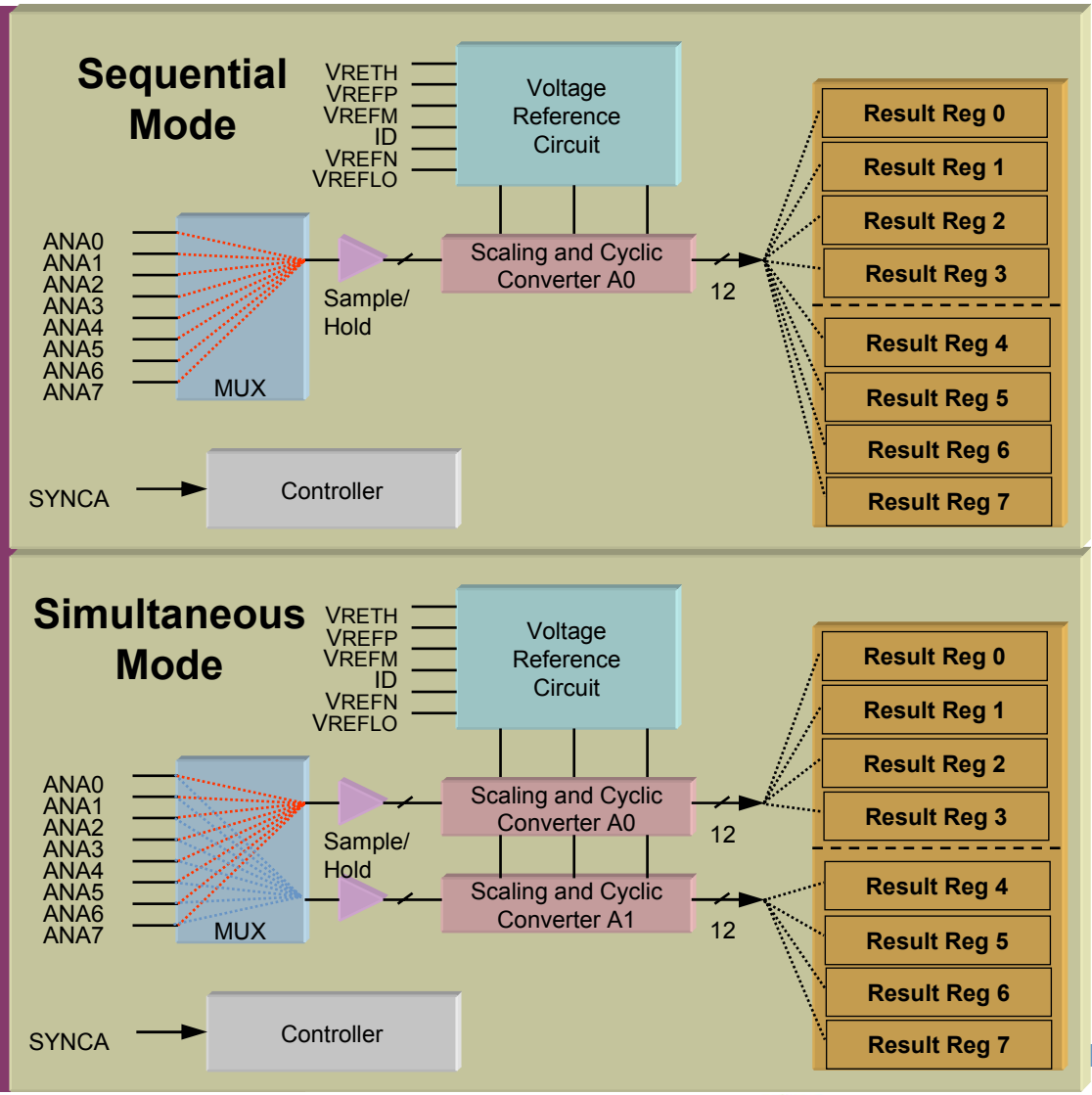
- **Sampling begins with every recognized START command or sync pulse**

Loop

- **The ADC continuously take samples as long as power is on and the STOP bit has not been set**

Each mode can be sequential or simultaneous

- **Sequential will sample SampleN one after another, while simultaneous can sample SampleN from Group1 and SampleN from Group 2 at the same time.**



56F8300 Differentiators

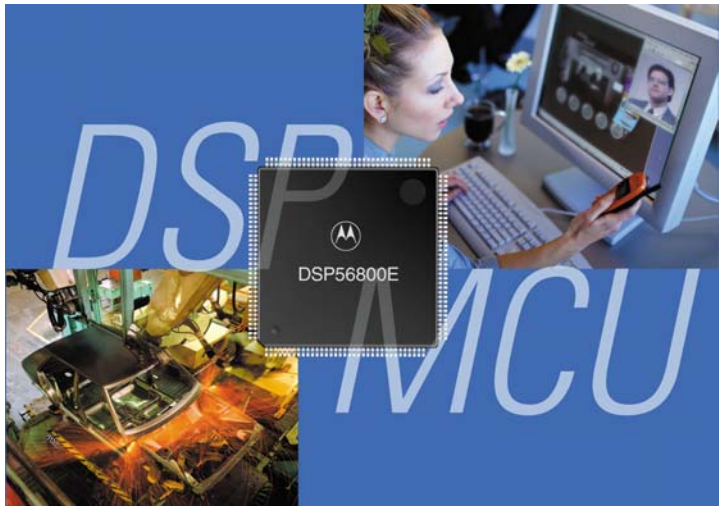
Q: Name the 4 types of ADC interrupts?

End of Scan

Zero Crossing

Upper Limit

Lower Limit

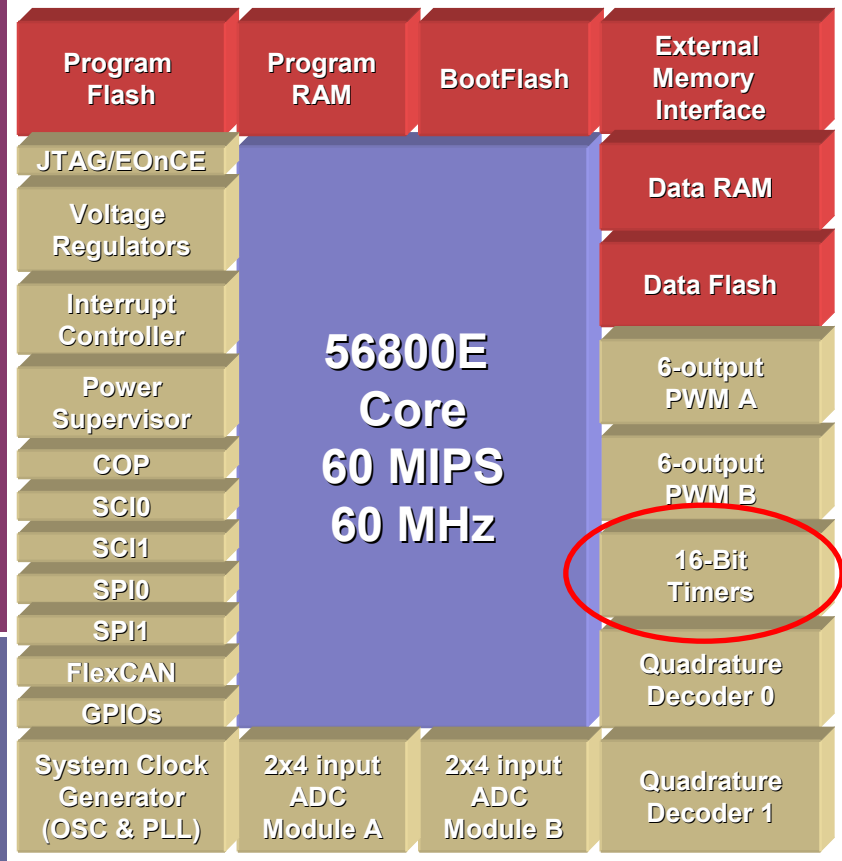


Quad Timers

- Four16-bit general purpose up/down timers per module
- Individually programmable
  - Input capture trigger
  - Output compare capture
  - Clock source
- Pins available as general I/O when timer(s) not in use
- Input pins are shareable within a timer module (Quad)
- Compare preload feature
- Counters in module can be daisy-chained to yield longer counter lengths

Operation Modes

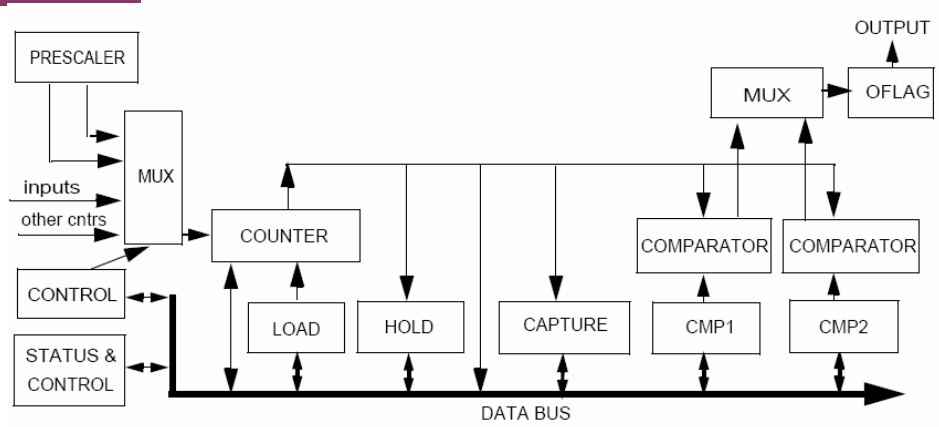
- Fixed Frequency PWM mode
- Variable Frequency PWM Mode
- Stop Mode
- Count Mode
- Edge Count Mode
- Gated Count Mode
- Quadrature Count Mode
- Signed Count Mode
- Triggered Count Mode
- One-Shot Mode
- Cascade Count Mode
- Pulse Output Mode

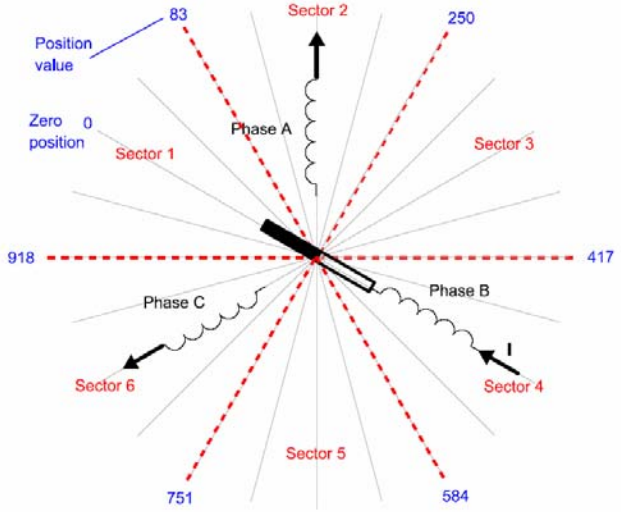


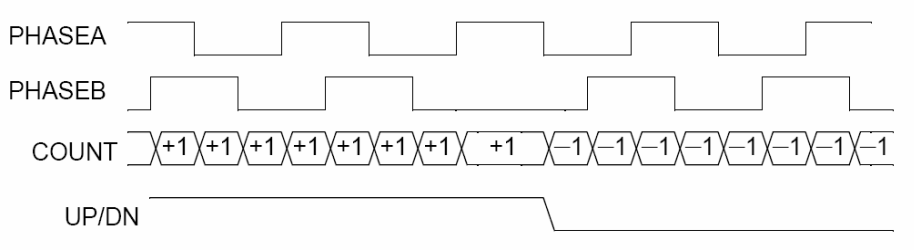
Motor Control – Quadrature Mode

Decodes primary and secondary inputs as quadrature encoder signals

Comparators are utilized to signal commutation transition



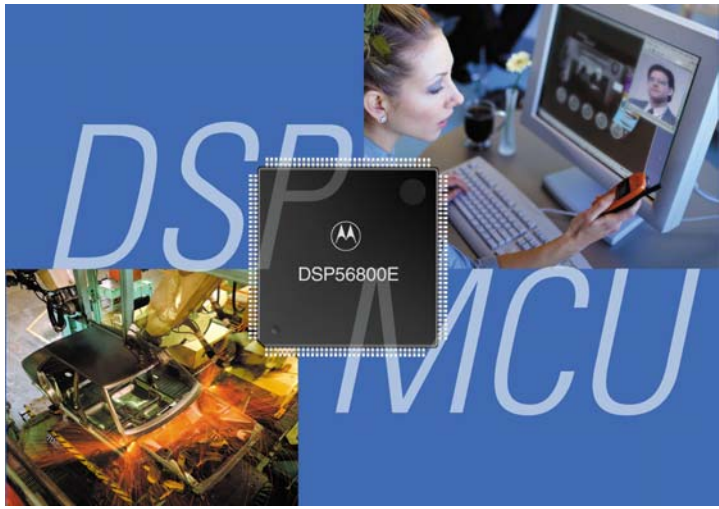




56F8300 Differentiators

Q: What is the maximum count available within a Quad Timer module?

All four 16-bit timers can be daisy chained to provide a count value up to 2^{64}



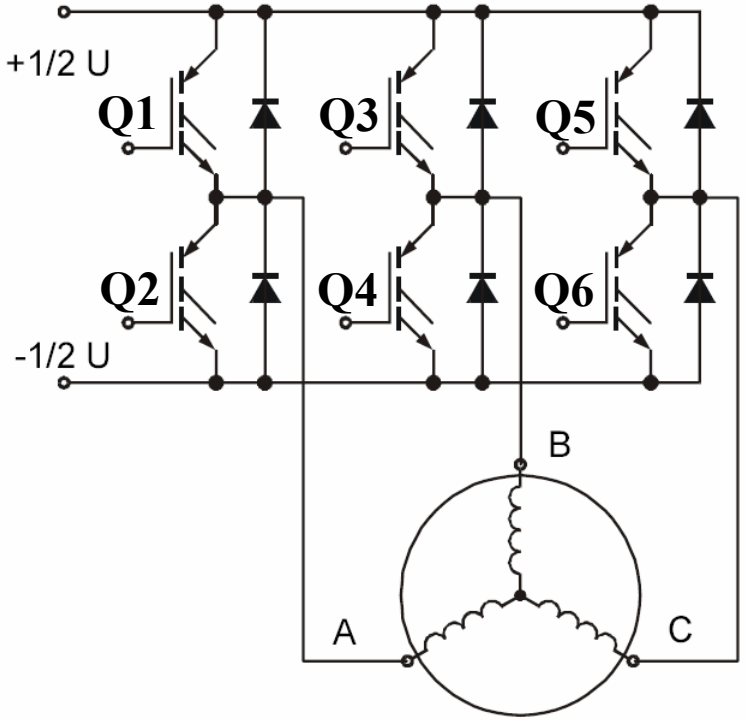
Launched by Motorola
freescale semiconductor

PWM – BLDC Background

Turn BLDC motor by performing commutation

- **Apply voltage to phases A & B, then B & C, then A & C, etc:**

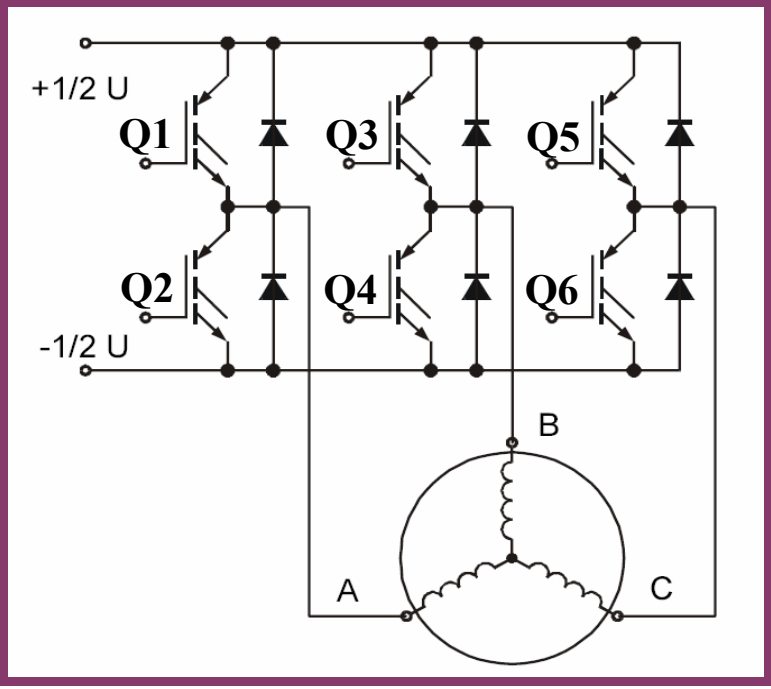
Phase A	Phase B	Phase C
$-V_{DCB}$	$+V_{DCB}$	NC
NC	$+V_{DCB}$	$-V_{DCB}$
$+V_{DCB}$	NC	$-V_{DCB}$
$+V_{DCB}$	$-V_{DCB}$	NC
NC	$-V_{DCB}$	$+V_{DCB}$
$-V_{DCB}$	NC	$+V_{DCB}$



PWM – BLDC Background

Control speed by changing applied voltage

- Change voltage by utilizing “switch” technique – Duty Cycle



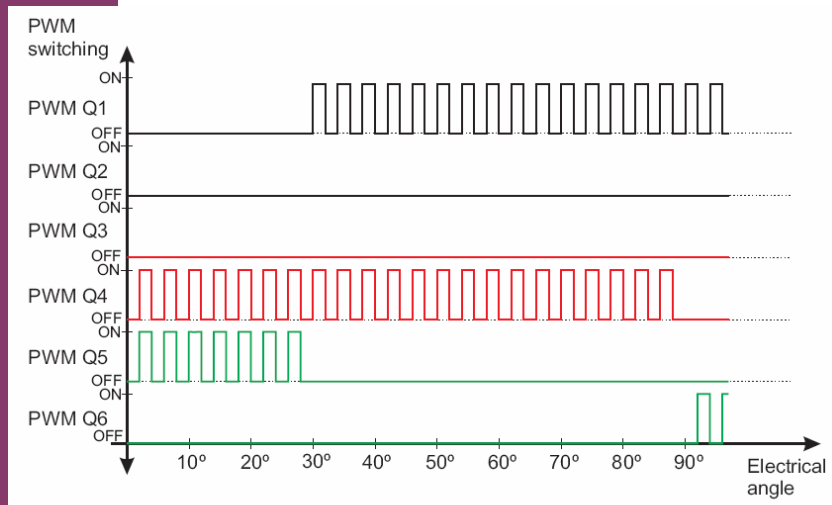


Figure 3-4. Independent Switching of Power Transistors

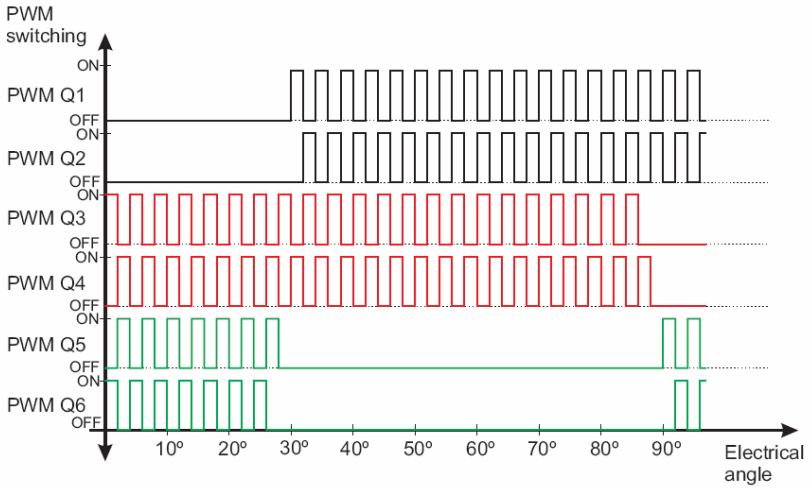
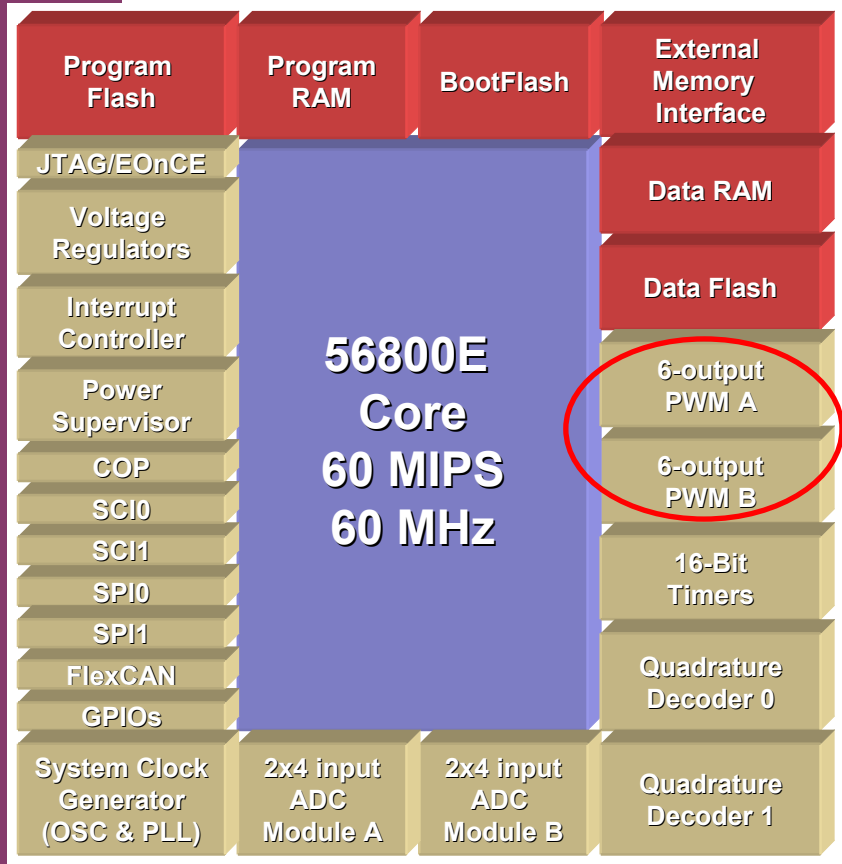


Figure 3-5. Complementary Switching of Power Transistors

Launched by Motorola
freescale semiconductor

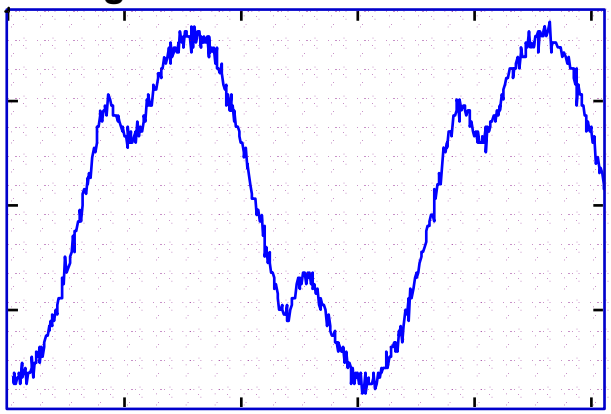
PWM

- **Three complementary signal pairs or six independent signals**
- **Complementary channel operation**
  - Deadtime insertion
  - **Separate top and bottom pulse width correction via current sensing or software**
  - **Separate top and bottom polarity control**
- **Edge-aligned or center-aligned signals**
- **15-bits of resolution**
- **Half-cycle reload capability**
- Asymmetric mode of operation (for phase shifting)
- **Programmable integral reload rates (half to 16)**
- **Individually software-controlled PWM outputs**
- ADC synchronization
- Programmable fault inputs
- **16 mA current sink capability**
  10 mA current source
- **Output Polarity Control**
- **Write protected registers**
- Double-buffered PWM register
- Wait/Debug mode operation



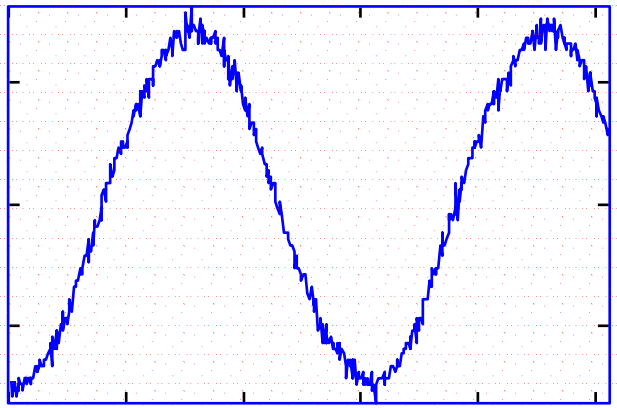
PWM Distortion Correction

Voltage with Correction Disabled



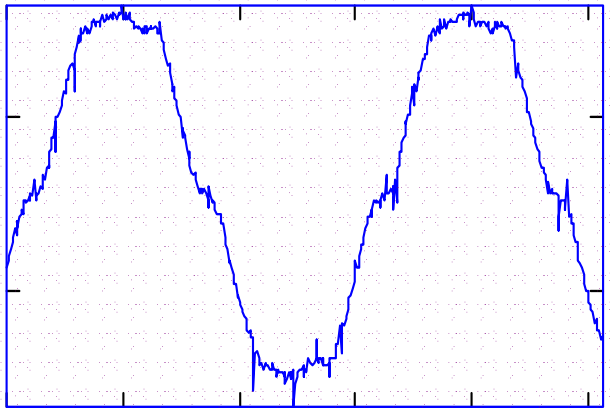
Motor Voltage

Voltage with Correction Enabled



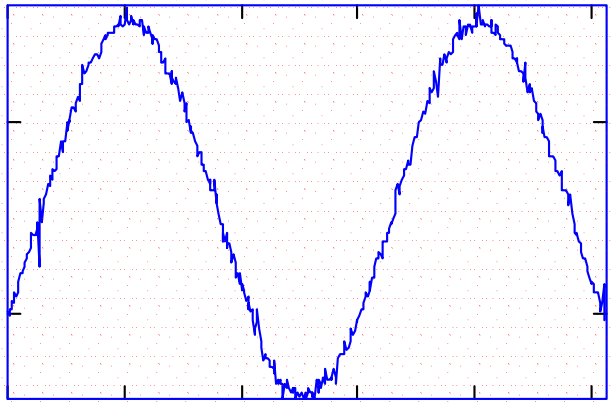
Quieter operation
Smoother operation
Less motor harmonic losses

Current with Correction Disabled



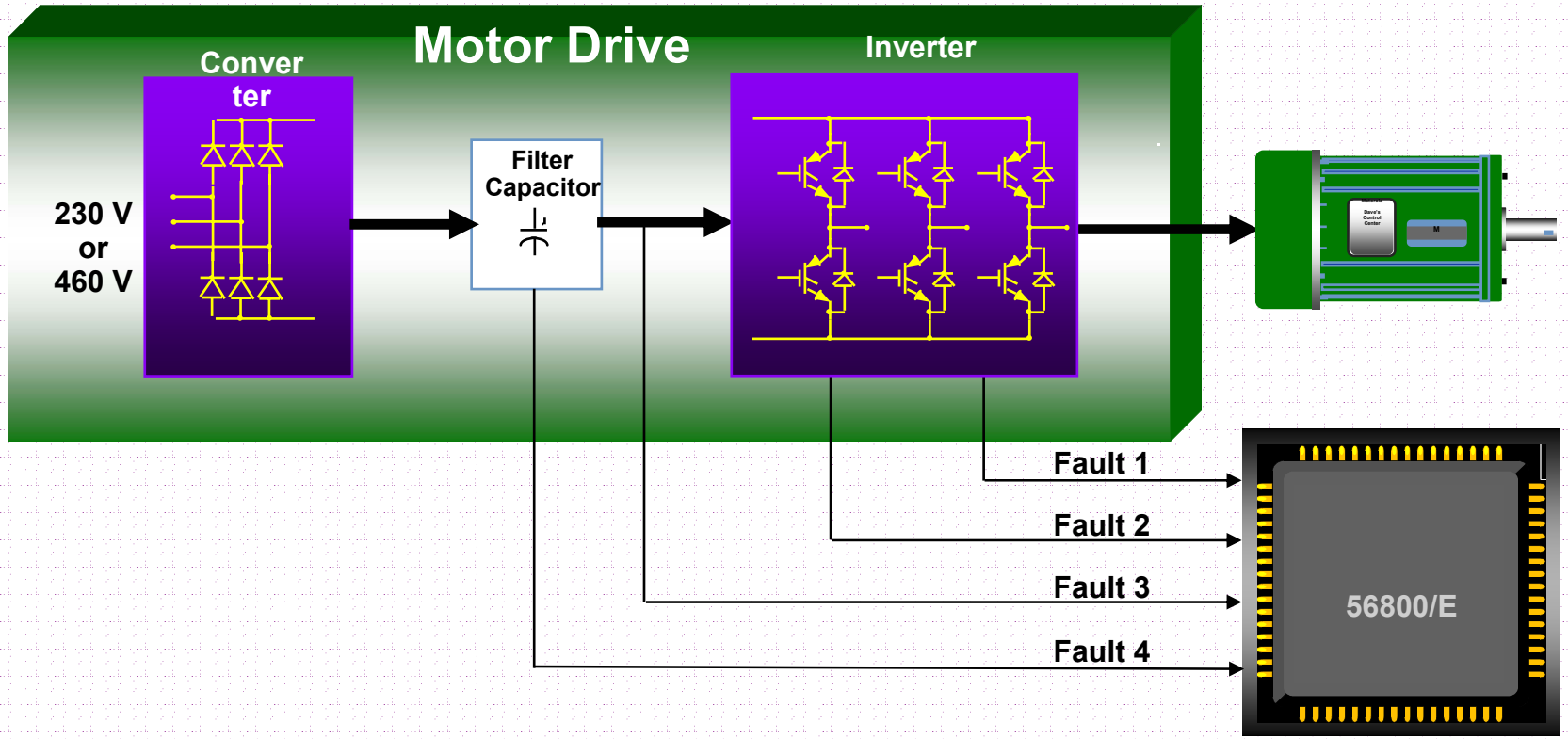
Motor Current

Current with Correction Enabled



Actual waveforms taken on a 1/2 horsepower motor

PWM – Multiple Fault Inputs

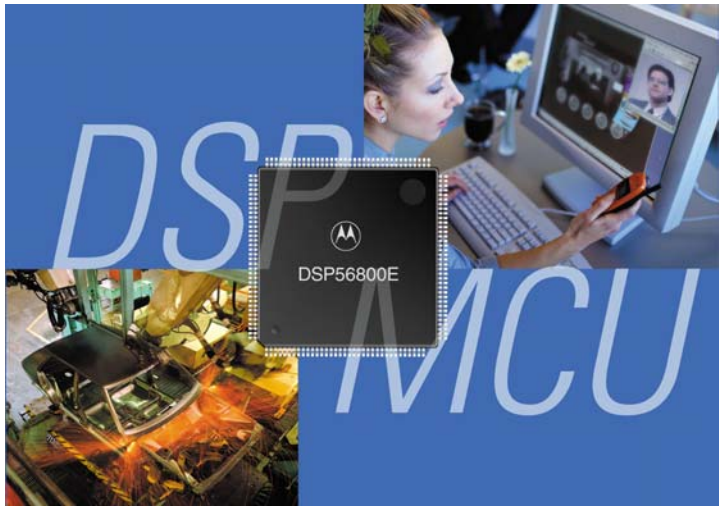


- Fault inputs can independently monitor critical system parameters, and generate an interrupt when asserted.
- Each input is mappable to immediately disable any or all PWMs
- Each input is programmable to allow Automatic or Manual PWM restart

56F8300 Differentiators

Q: How many independent fault inputs are available in the PWM?

Four programmable Fault inputs provide independent handling of faults within the system.



Launched by Motorola
freescale semiconductor

Quadrature Decoder

Four encoder inputs per decoder
- Phase A; Phase B; Index; Home

Captures all four transitions on two-phased inputs
- extracts actual shaft position and direction
- 32-bit position counter - initialized by software or external events
- Pre-loadable 16-bit revolution register

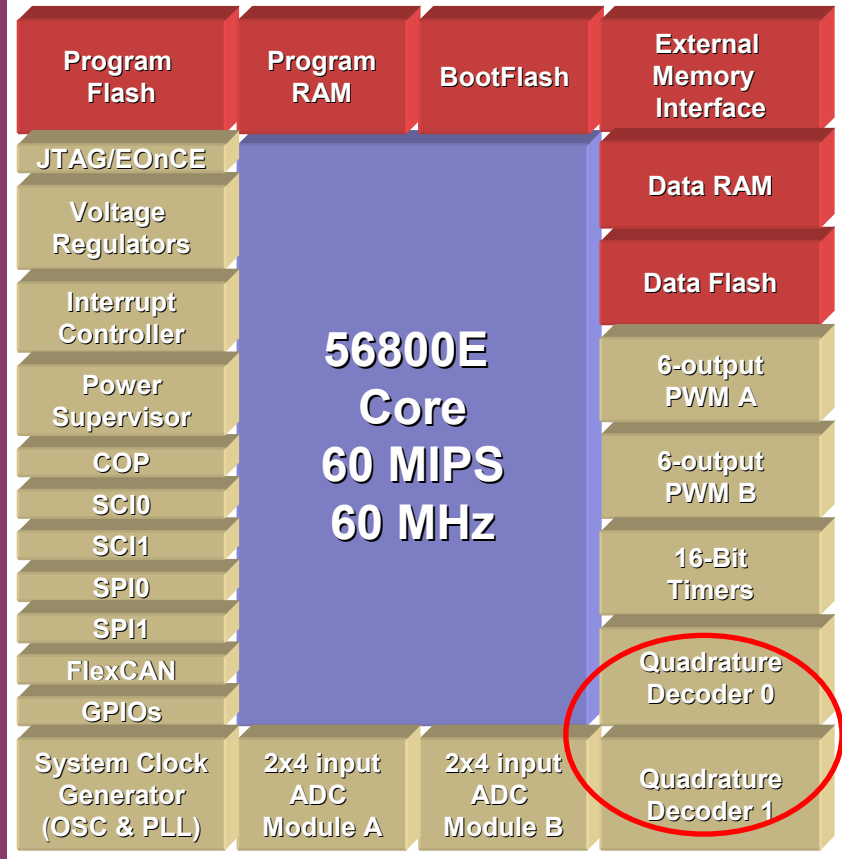
Index input
- resets the current integration value
- begins integrating a new revolution value

Home input
- Initializes position counter

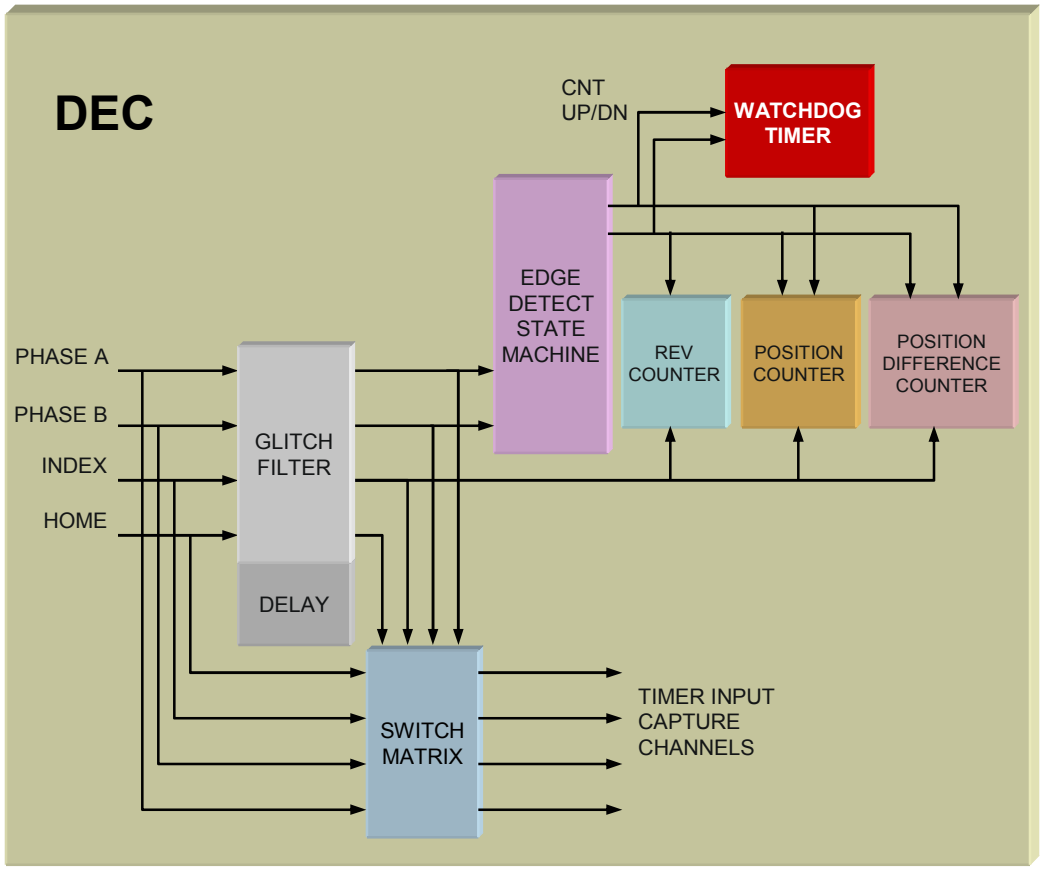
Configurable glitch filter for inputs

Can operate as single-phase pulse accumulators

Watchdog timer detects non-rotating shaft condition



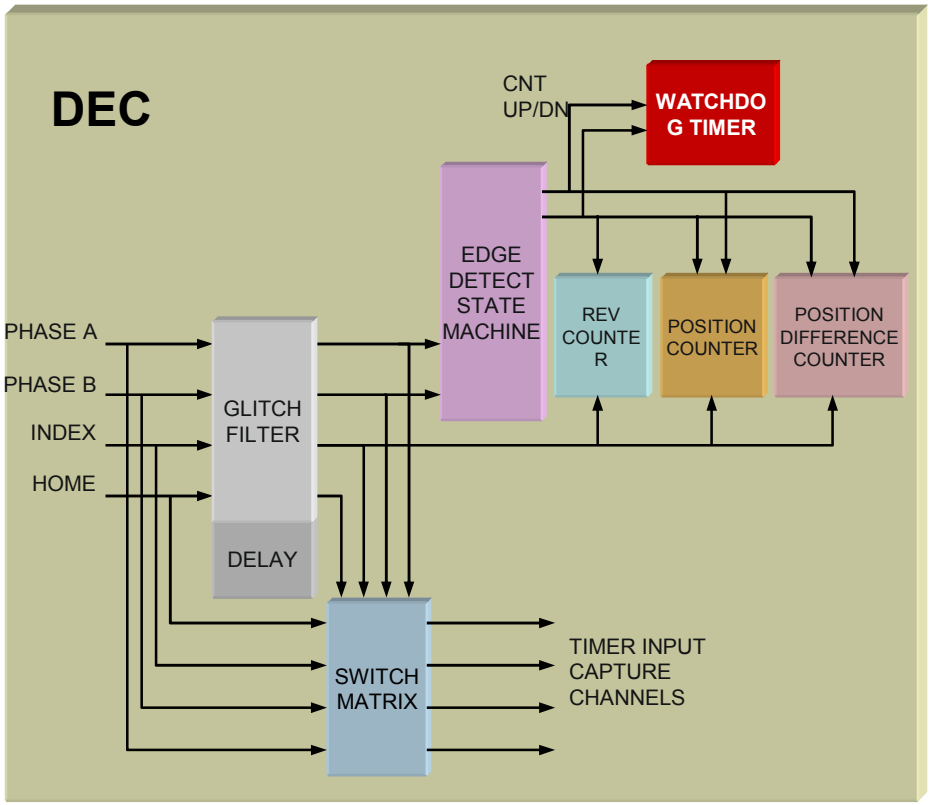
Quadrature Decoder - Block Diagram

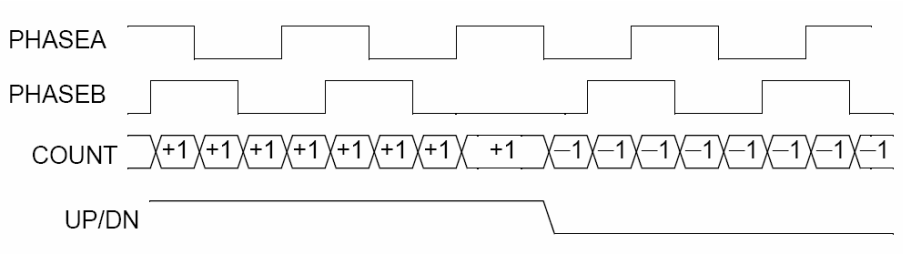


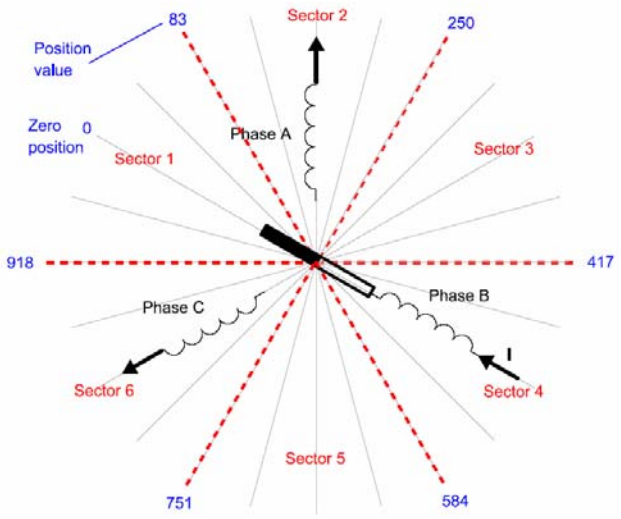
- ✓ Hold registers are associated with three counters:
  1. **Position**
  2. **Position difference**
  3. **Revolution**

  When any of the counter registers are read, the contents of each is written to its hold register. Taking a "snapshot" of the counters' values provides a consistent view of a system position and a velocity.
- ✓ **Glitch Filter**
  - ➢ This filter samples four time points on the signal and verifies the majority of samples are at a new state.
  - ➢ The sample rate of this filter can be adapted to a variety of signal bandwidths.
- ✓ **Edge Detect State Machine**
  - ➢ Identifies changes in the four possible states of the filtered PHASEA/B inputs, calculating the direction of motion.
  - ➢ These signals are routed into three up/down counters.
- ✓ **WATCHDOG Timer**
  - ➢ Detection of no rotation.
  - ➢ Two successive counts indicate proper operation and will reset the timer.

Quadrature Decoder for Motor Control



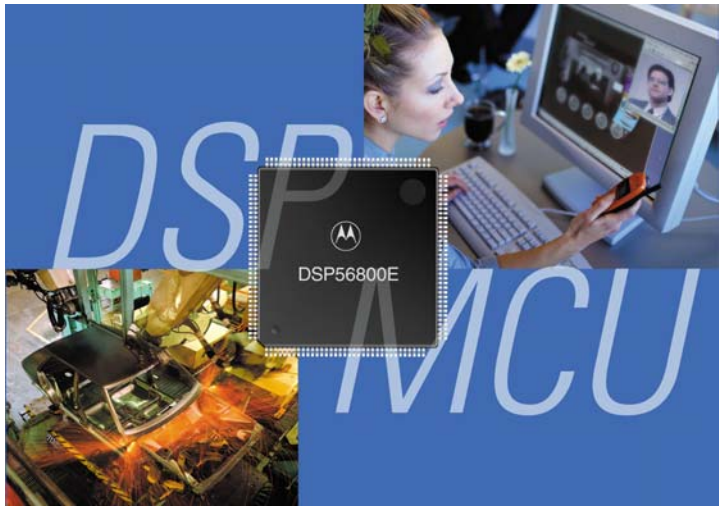




56F8300 Differentiators

Q: What is the differentiating feature of the Quad Decoder module?

The watchdog provides an interrupt after detection of two consecutive cycles of the non-rotating shaft condition.



Launched by Motorola
freescale semiconductor

CAN

Version 2.0B compliant

- **Standard and extended data frames**
- **0-8 bytes data length**
- **Programmable bit rate up to 1Mbps**
- **Support for remote frames**

"Time Stamp", based on 16-bit free-running timer

new **Two Serial Message Buffers for Buffer Frame**

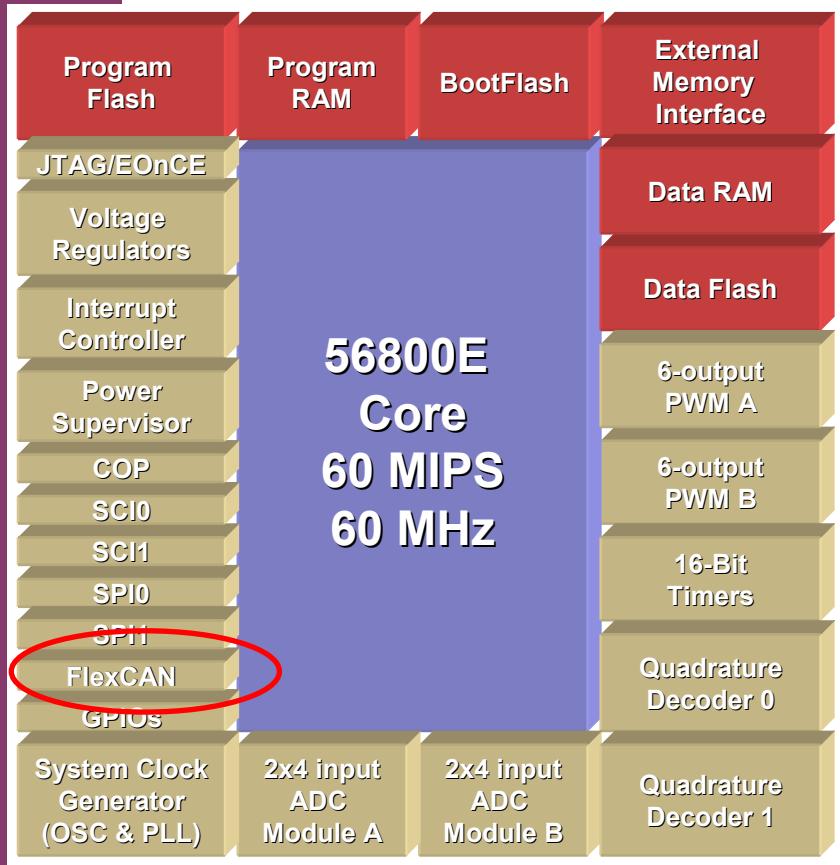
Sixteen Flexible Message Buffers of 0-8 bytes Data Length, each configurable as RX or TX, all support Standard and Extended Messages

new **Flexible maskable identifier filter**

Programmable wake-up functionality with integrated low-pass filter

Separate signaling and interrupt capabilities for all CAN RX/TxXrror states

Three low power modes



SCI

Operates as a Universal Asynchronous Receive and Transmitter (UART)

Full duplex operation provides simultaneous data transmit and receive

Half duplex operation allows data transmit and receive via single wire.

Separately enabled transmitter and receiver

13-bit baud rate selection

Standard mark/space non-return-to-zero (NRZ) format:

- Programmable 8-bit or 9-bit data format

Separate receiver & transmitter CPU interrupts

Programmable polarity for transmitter and receiver

Two receiver wakeup methods:

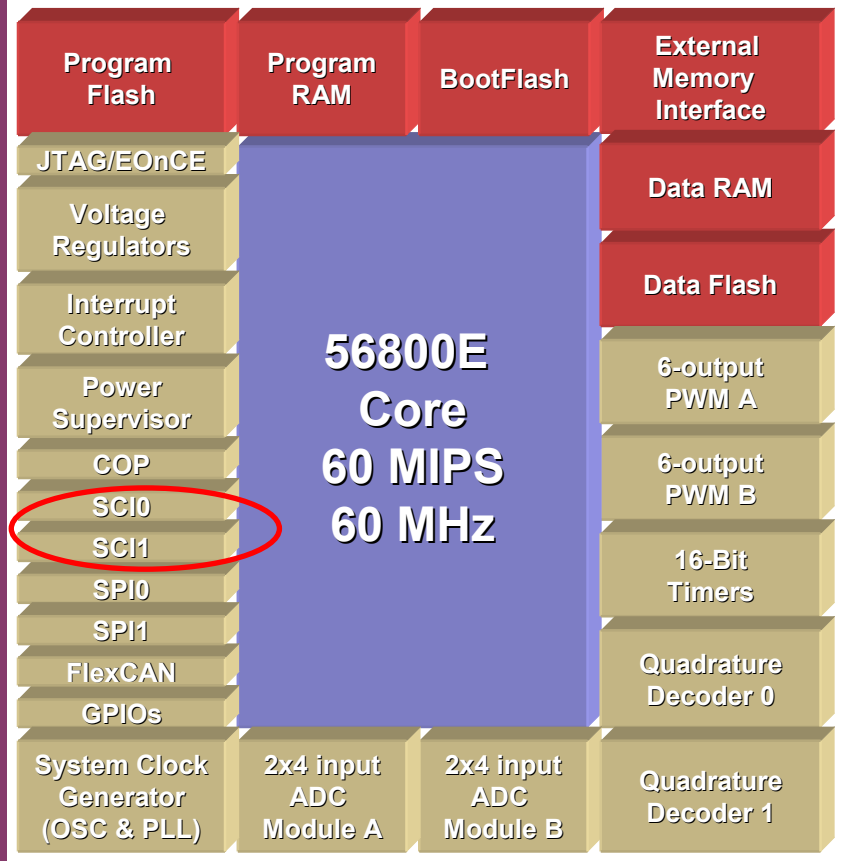
- Idle Line
- Address Mark

Interrupt-driven operation with eight flags

Receiver framing error detection

Hardware parity checking

1/16 bit-time noise detection



SPI

Supports LCD drivers, A/D subsystems, & MCU systems

Supports inter-processor communications in a multiple master system

Supports demand-driven master or slave devices with high data rates

Full-Duplex Operation

Double-buffered Operation with separate transmit and receive registers

Programmable length transmissions, 2 to 16 bits

Programmable transmit and receive shift order, MSB or last bit transmitted

Eight master mode frequencies (maximum = bus frequency ÷ 2)

Maximum slave mode frequency = bus frequency

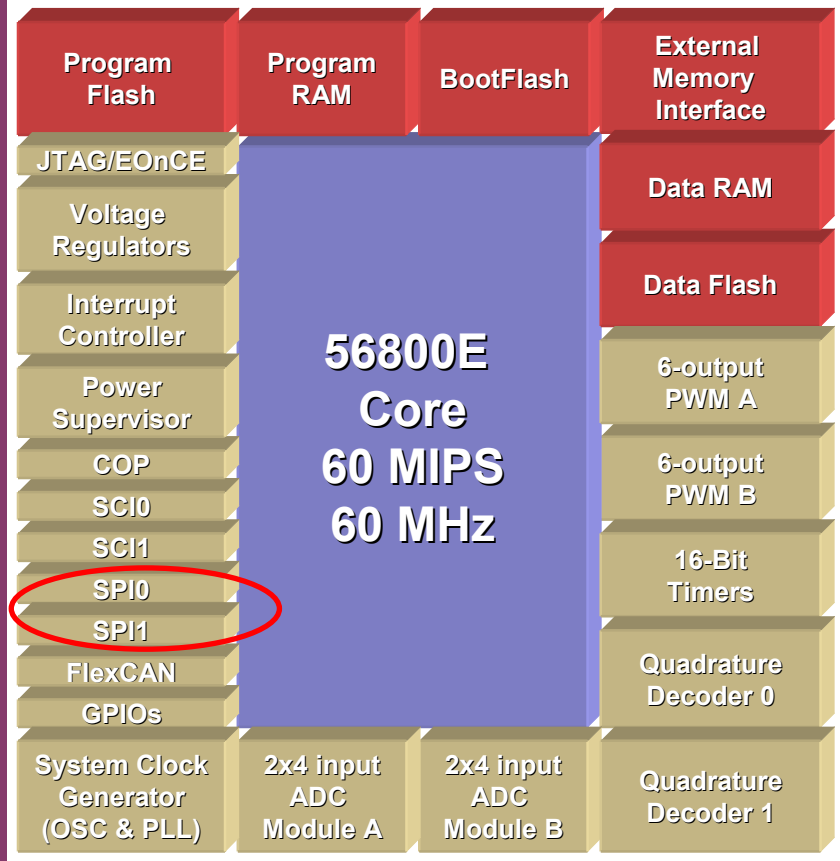
Serial clock with programmable polarity and phase

Two separately enabled interrupts

- Receiver Full
- Transmitter Empty

Mode Fault and overflow error flag with interrupt capability

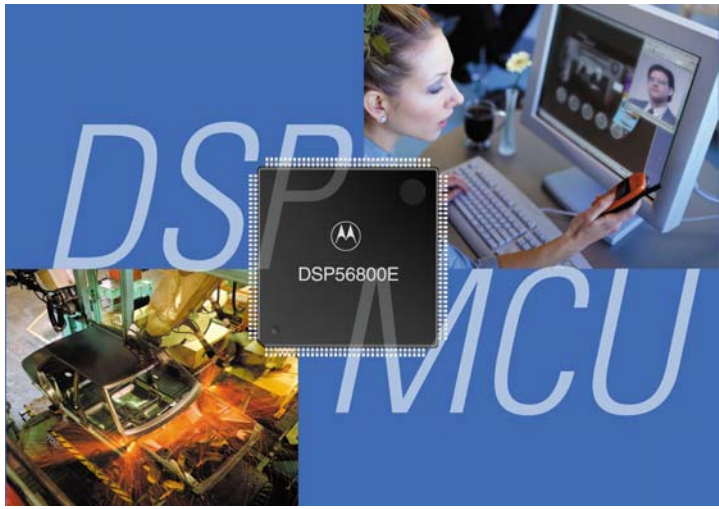
Easy interface to Motorola's MCUs, Analog, and Sensors



56F8300 Differentiators

Q: What are the three types of serial communication peripherals available?

- CAN
- SCI
- SPI



Launched by Motorola
freescale semiconductor

Voltage Management & Power Supervisor

I/O drivers designed to interface at TTL compatible 3.3 V (5 V tolerant)

Two internal regulators available

- **One for device core (can be bypassed)**
- **One for analog circuitry (always enabled)**

Regulators converts 3.3 V input to 2.5 V core operating voltage

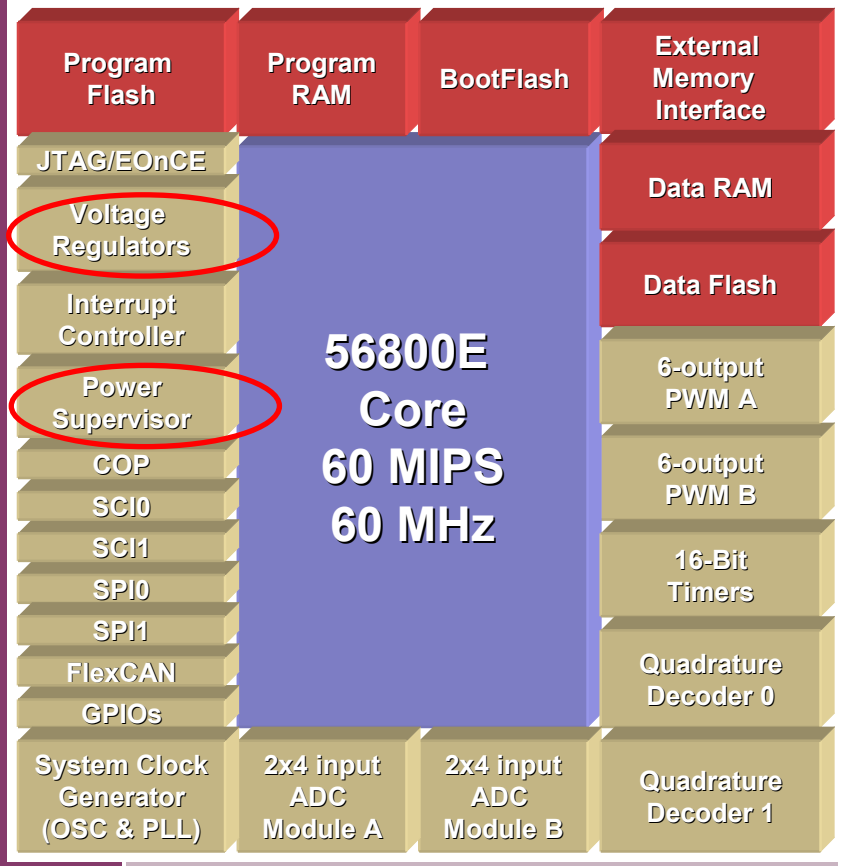
- **Reduces overall system cost**
- **Controls power usage**
- **Controls system noise floor**

Power Supervisor holds device in reset until there is enough voltage for on-chip logic to operate at the oscillator frequency

- **Precludes any problems associated with false restart**
- **Eliminates need for external power monitor**

Two Low Voltage Detect high-priority interrupts

- **Low voltage detect signals used to initiate a software controlled shutdown when the supply voltage drops below acceptable levels**



On Chip Clock Synthesis (OCCS)

Four different, dynamically selectable system clock sources available

- Crystal Oscillator - driven by external crystal or clock source
- Relaxation Oscillator – On-chip Oscillator (56F8322/323 only)
- Programmable 4-bit Prescaler - divide-by of the IP Bus clock
- Phase Locked Loop (PLL) - generates output frequencies of up to 60 MHz

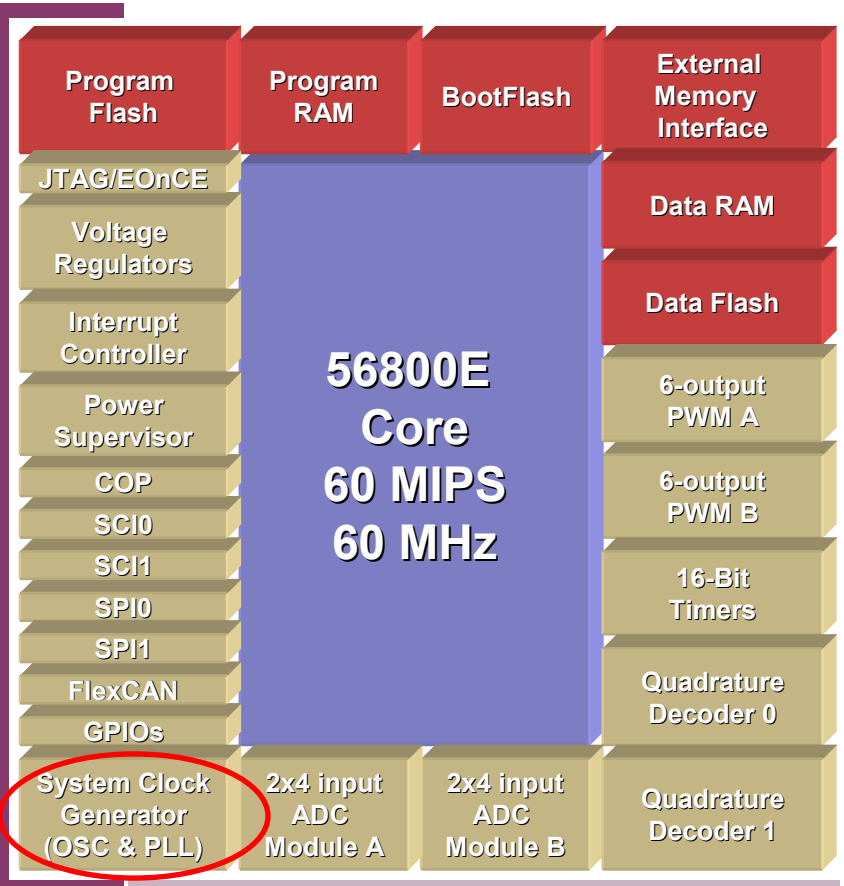
Selectable Phase Locked Loop input clock source

- Crystal Oscillator
- Relaxation Oscillator (+/- 2.5% accuracy over temperature after trimmed)
- Programmable 4-bit Prescaler

Dynamically programmable PLL allows configurable power/speed options

Generates an interrupt if either loss of clock ,or loss of lock, or both

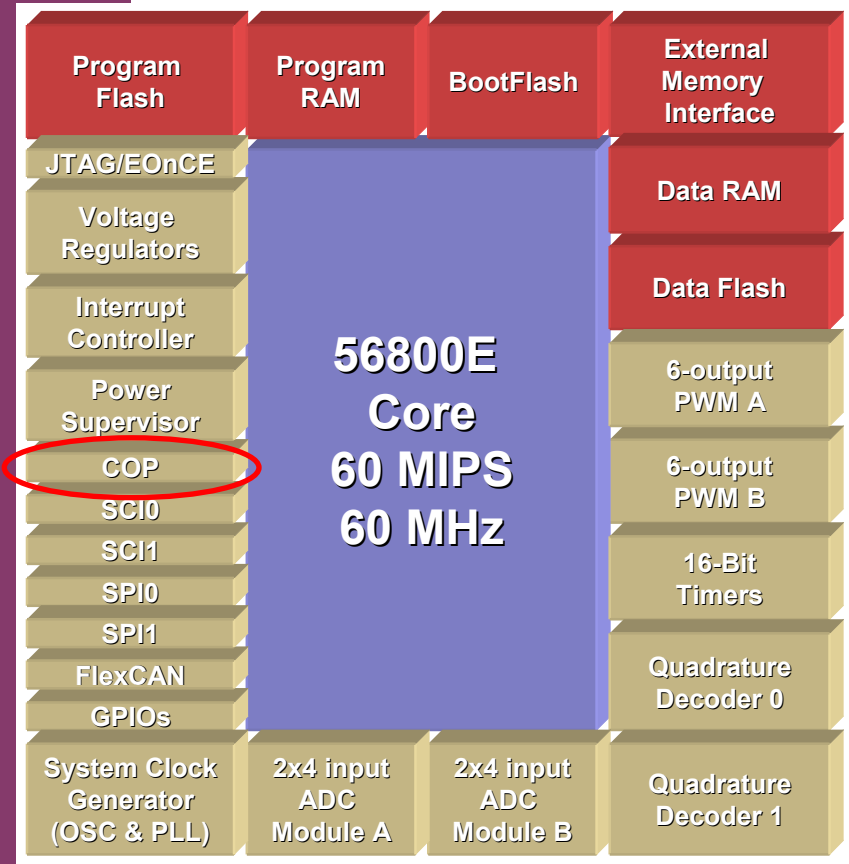
Five clock output pins



Computer Operating Properly (COP)

Allows for detection of application software that may be operating incorrectly.

Resets the part if not properly serviced.

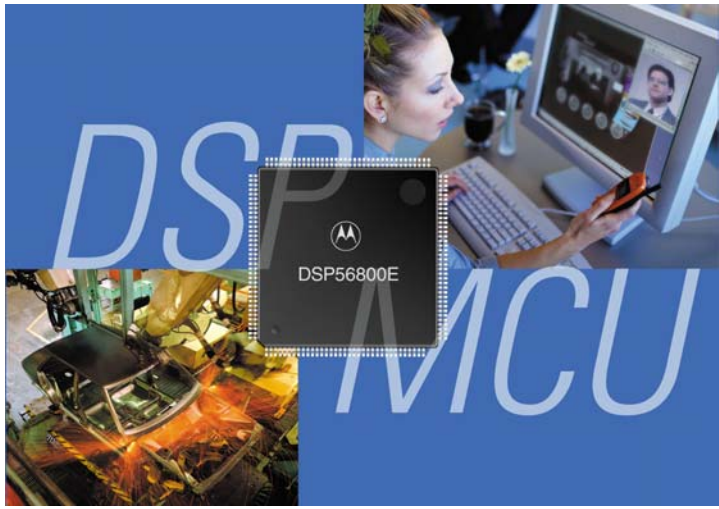


56F8300 Differentiators

Q: How can the Voltage Management and Power Supervision modules reduce your system costs?

No need to supply multiple source voltage regulation circuitry

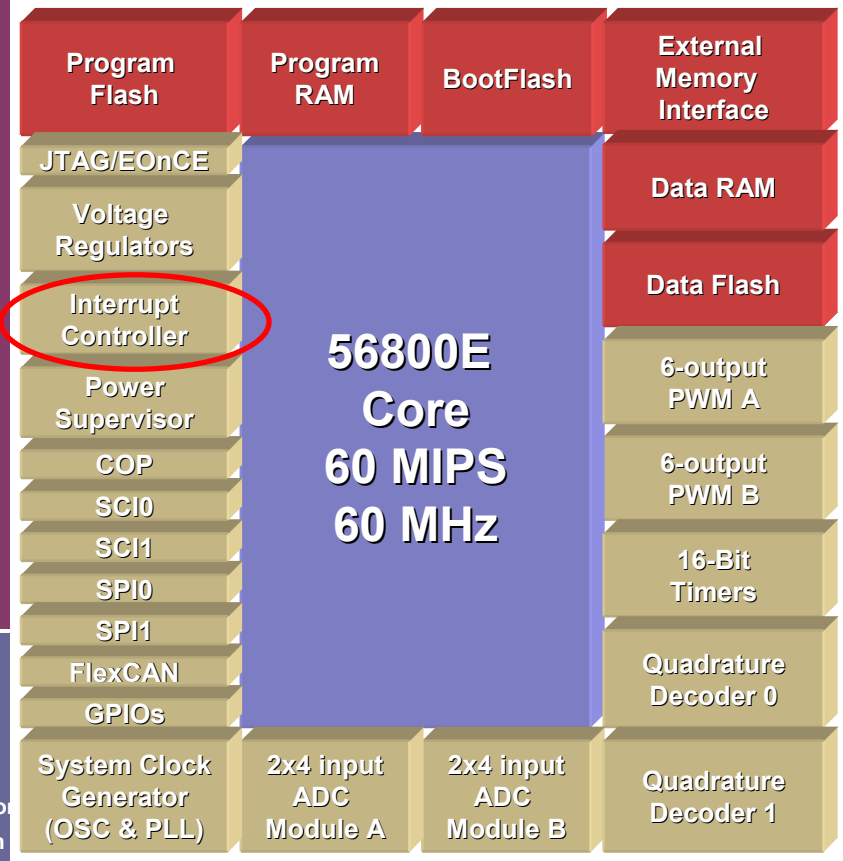
No need for external POR circuitry



Interrupt Controller

- **Arbitrates peripheral interrupt requests**
- **Signals core when an interrupt of sufficient priority exists**
- **Provides ISR address to service each interrupt**
- Relocatable Interrupt Vector Table
- **Supports 128 interrupt sources**
- **Supports 5 interrupt priority levels with HW nesting**
  - **Level # 3 (highest) for core interrupts**
  - **Levels # 2, # 1, and # 0 for peripherals**
  - **Level # -1 (lowest) for SW interrupt**

Self-paced training available on Interrupt Handling



new

Two Fast Interrupts

Available for Level 2 interrupts

Dedicated context registers:

- R0, R1, N, M01 **– Shadow Regs**
- Y **–** **Data Reg**
- FIRA **– Fast Interrupt Return Address Reg saves PC**
- FISR **– Fast Interrupt Status Reg saves SR and NL bit**

Lower latency

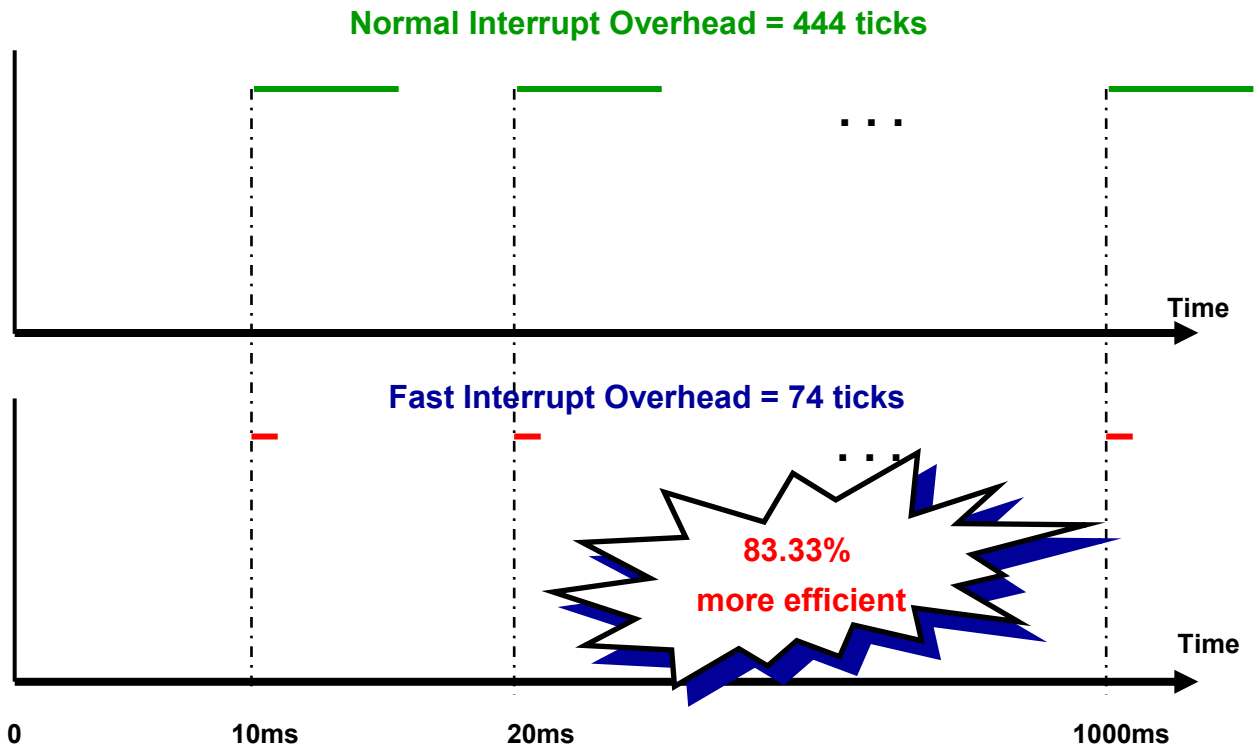
- **Bypass jump table**
- **Bypass context save/restor**
- **FRTID - lower latency than**

Launched by Motorola freescale semiconductor

Fast Interrupts Demonstration

Execute “FastIntDemo.mcp” project (for 56800E devices only)

Applications executes Fast and Normal interrupt every 10ms, and calculates associated CPU overhead in clock ticks



Fast Interrupt takes 83.33% less time to complete.

More time for application processing!

(444 – 74) / 444 = .8333

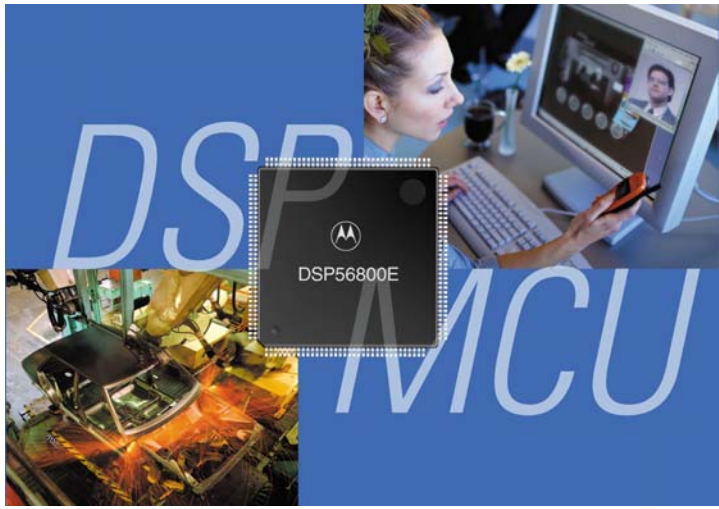
56F8300 Differentiators

Q: Why are Fast Interrupts faster than normal interrupts?

Shadow Registers automatically loaded and restored by hardware; no need to perform a context save

By-Passes the Vector Table to begin ISR execution.

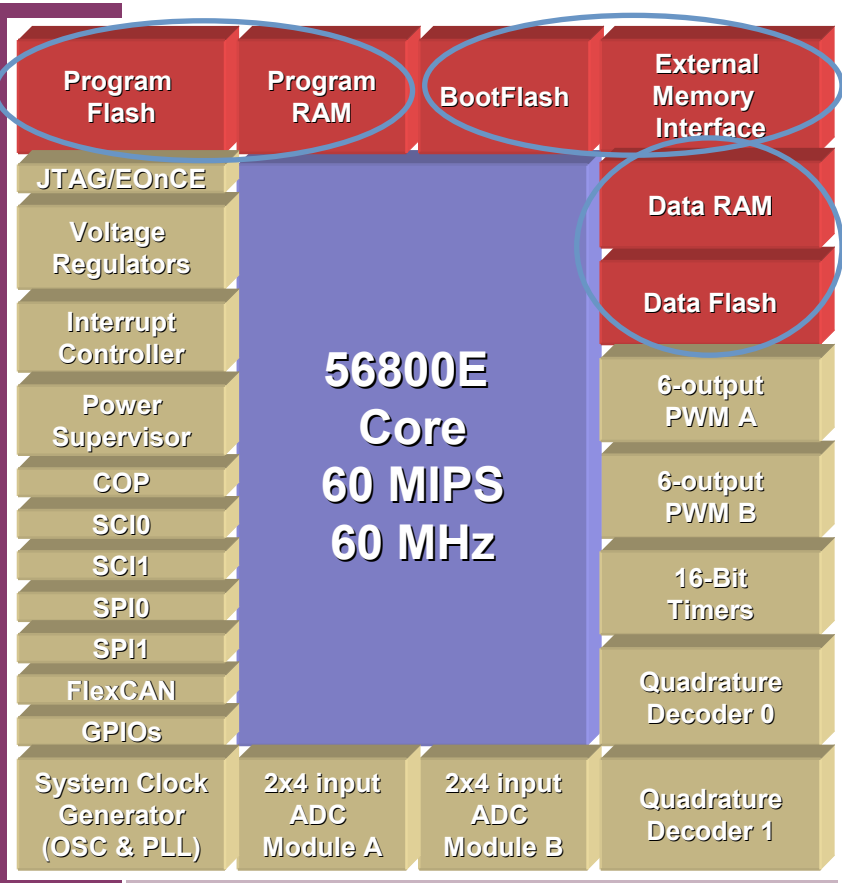
Dedicated hardware registers (FIRA, FISR)



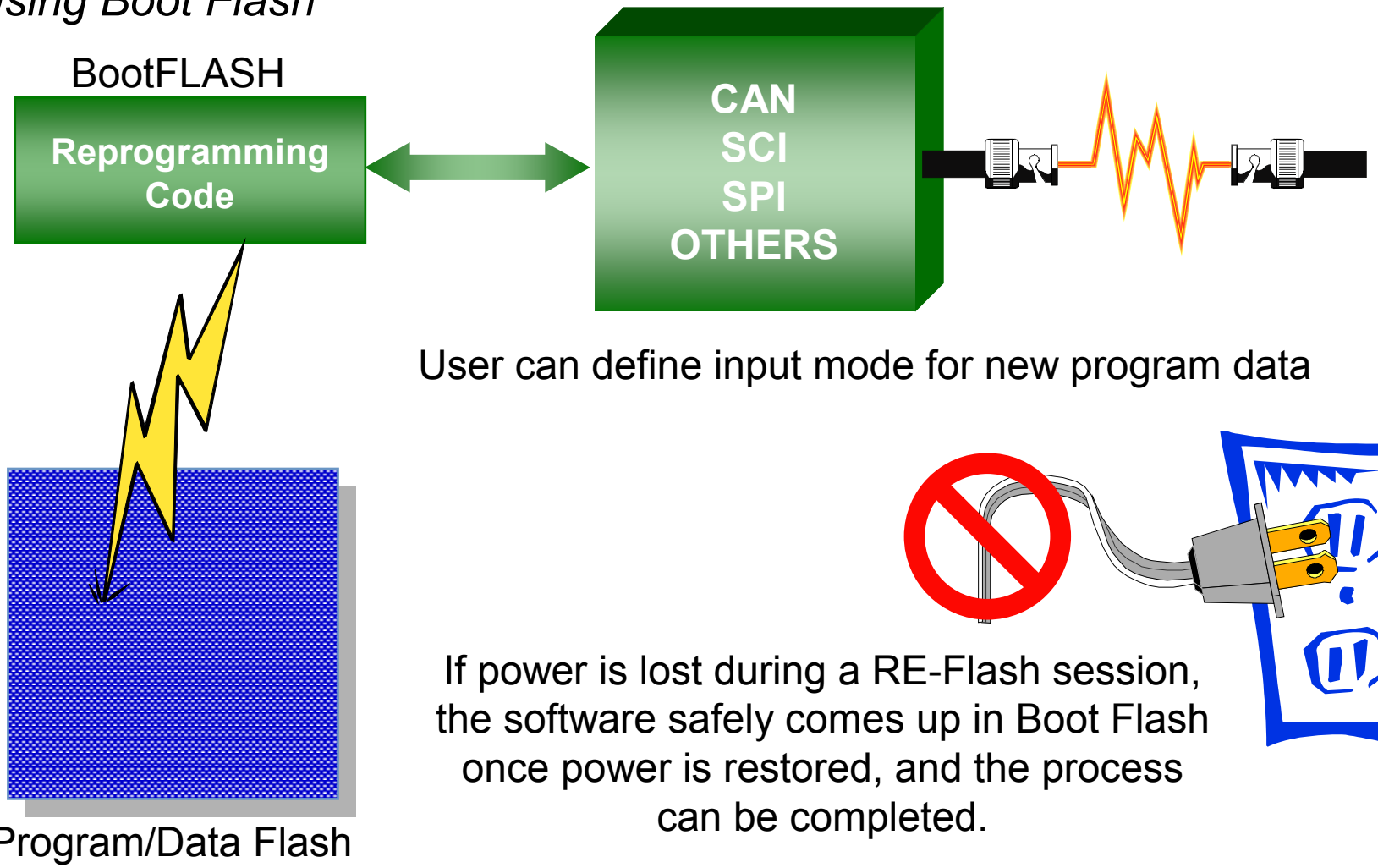
Launched by Motorola
freescale semiconductor

Memory

- On chip Harvard Architecture
  - Separate program and data buses
  - Permits up to three simultaneous accesses to program AND data memory
- On chip and Off-chip memory
- 8K bytes of BootFLASH™
- Programmable “Code Protection” feature
- Programmable “Code Security” feature
- Flash with 256/512 word page size for data/program
- Flash memory programmable via JTAG/OnCE interface or user defined programming (such as SPI, SCI, CAN)
- Programmable wait states for low cost system memory solutions
- Can program one word at a time
- 60MHz operation at 125ºC
- EEPROM emulation (HW & SW support)



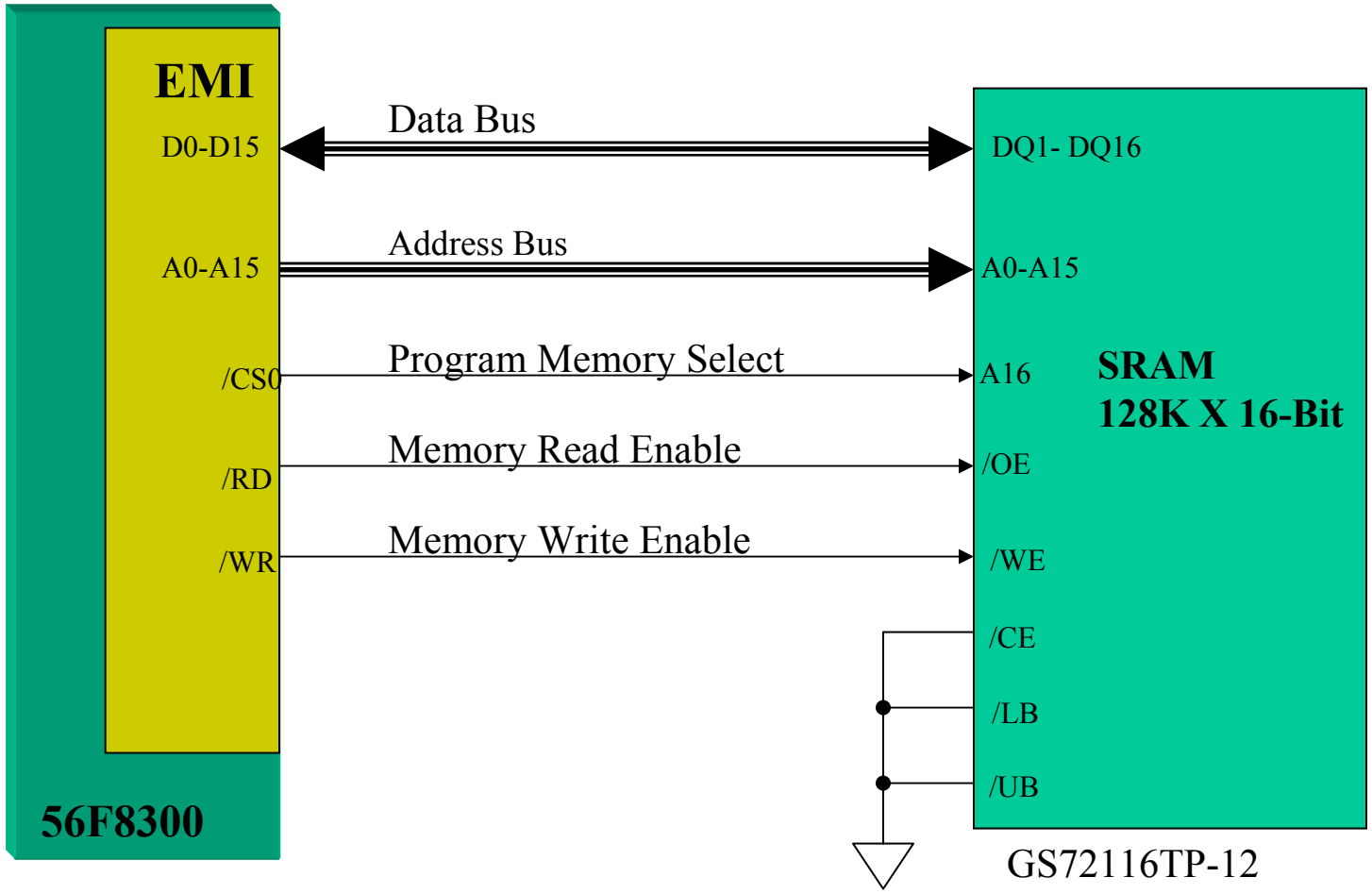
Using Boot Flash



User can define input mode for new program data

If power is lost during a RE-Flash session, the software safely comes up in Boot Flash once power is restored, and the process can be completed.

External Memory Interface



*First 64 K words assigned for program memory
*Second 64 K words assigned for data memory

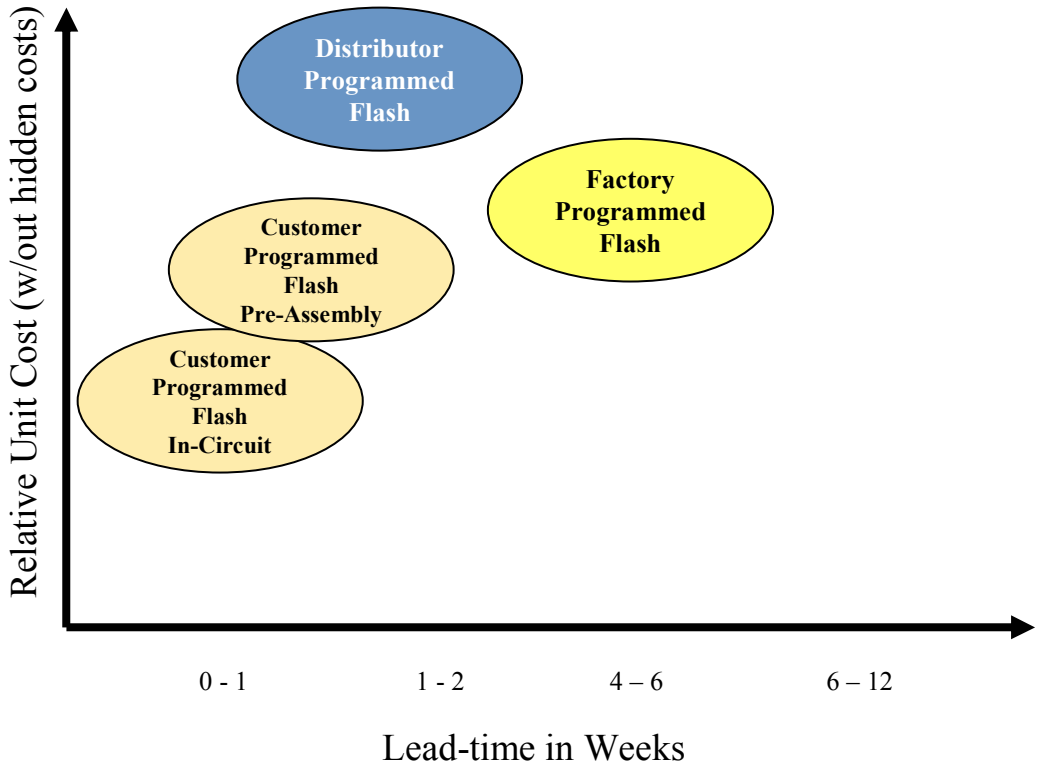
Flash Programming Solutions

JTAG/EOnCE Port

- 800 W/sec
- MetroWerks CodeWarrior
  - Parallel port
- BP Microsystems
  - Bulk Device Programmer
- Domain Technologies
  - Assortment of communication interfaces
  - Custom off-chip Flash
- Flash over JTAG Utility
  - Source code available

Distributor Programming

- Preassembly programming
- See local distributor



Serial/CAN Bootloader

- Serial: 2.6 K W/sec
- CAN: 3.7 K W/sec
- Source code available

Parallel Programming Mode

- 91 K W/sec (Program)
- 47 K W/sec (Boot/Data)
- Requires NDA and factory support

Factory Programming

- Lead Time
- Fee based
- Min qty req

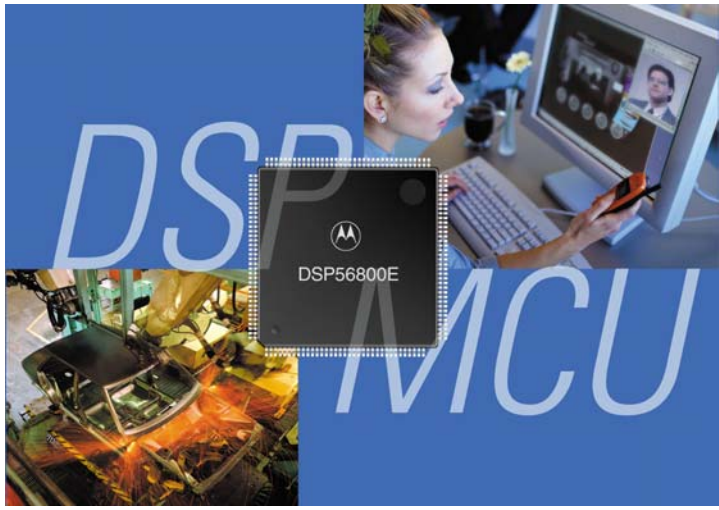
Launched by Motorola freescale semiconductor

56F8300 Differentiators

Q: What is the difference between flash protection and flash security?

Flash Protection prevents the protected flash sections from being erased/programmed

Flash Security prevents the contents of flash from being viewed to protect customer IP.



Launched by Motorola
freescale semiconductor

Hybrid MCU Architecture Introduction

56800/E Family and Peripheral Introduction

56800/E Hybrid Controller Roadmap

56850 Series
Telecom/voice, RAM-based, 120 MMACS, 81–144 pins

56F8400 Series
Automotive, industrial, Flash-based,120 MMACS, Flex-Ray, 2xCan, Improved PWM&ADC, low power, 48–160 pins

56F8500 Series
Automotive, industrial, Flash-based,150 MMACS, Flex-Ray, 2xCan, Improved PWM & ADC, low power,48- 160 pins

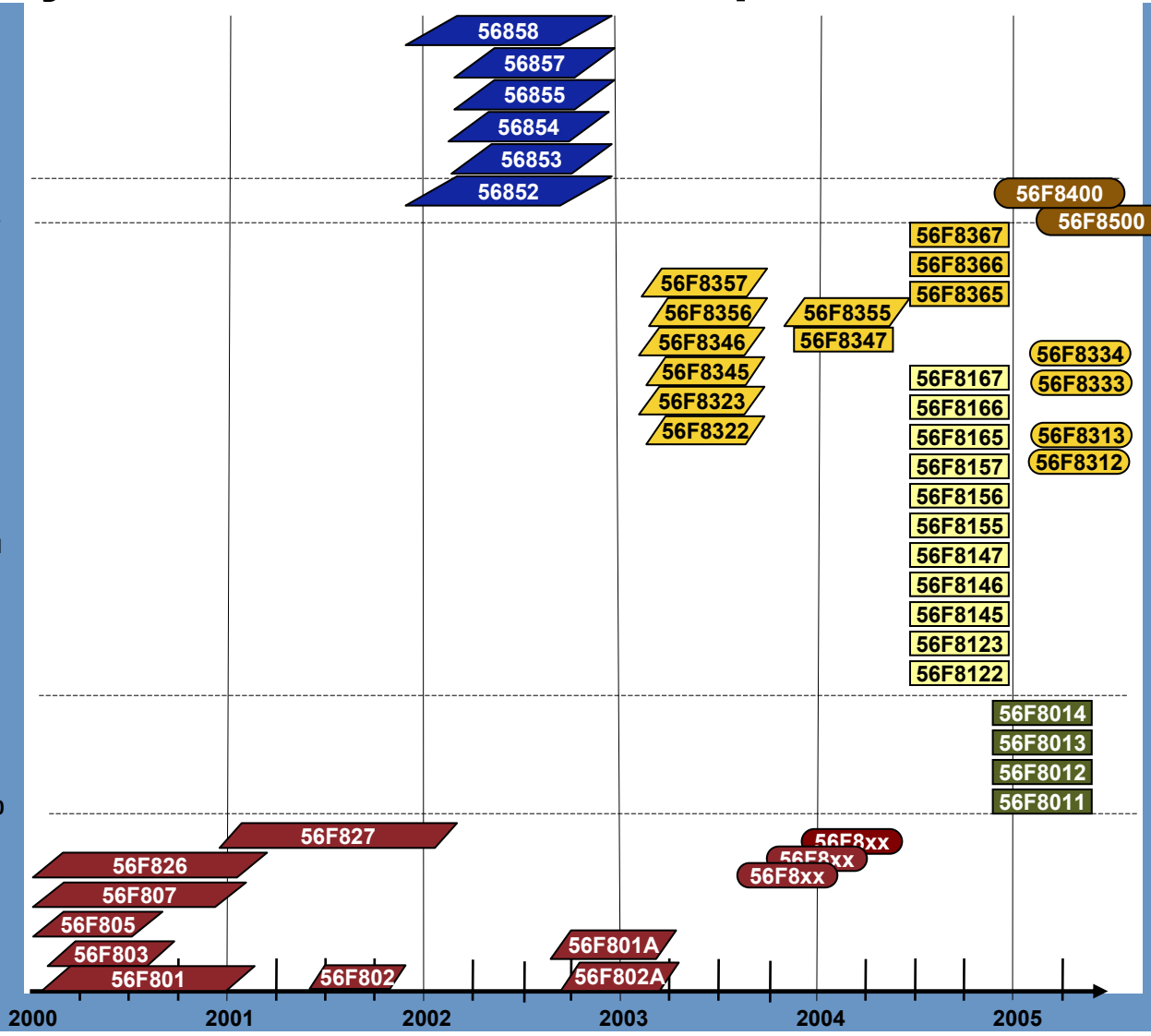
56F8300 Series
Automotive&Industrial, Flash-based, 60 MMACS, 16-512KB PFlash, 48–160 pins

56F8100 Family
Mid-Range Industrial and General Purpose, Flash Based, 40MMACs 48-144 pins

56F8000 Series
Low end industrial and automotive control, Flash-based, 32 MMACS, 28-32 pins

56F820 Series
General Purpose, Flash-based, 40 MMACS, 100–128 pins

56F80x Series
Industrial controllers, Flash-based, 40 MMACS, 32–160 pins



Proposal
Planning
Execution
Production

0.1Xμ, 56F8400 120 MMACS
0.1Xμ, 56F8500 150 MMACS
0.18μ, 56800E 120 MMACS
0.25μ, 56800E 60 MMACS
0.25μ, 56800E 40 MMACS
0.25μ, 56800 30/40 MMACS
0.25μ, 56800E 32 MMACS

Launched by Motorola
freescale semiconductor

56F800 Series Overview

Launched by Motorola
freescale™
semiconductor

56F800 Series

- Product Offerings
- Functionality
- Application Examples
- Why 56F800?

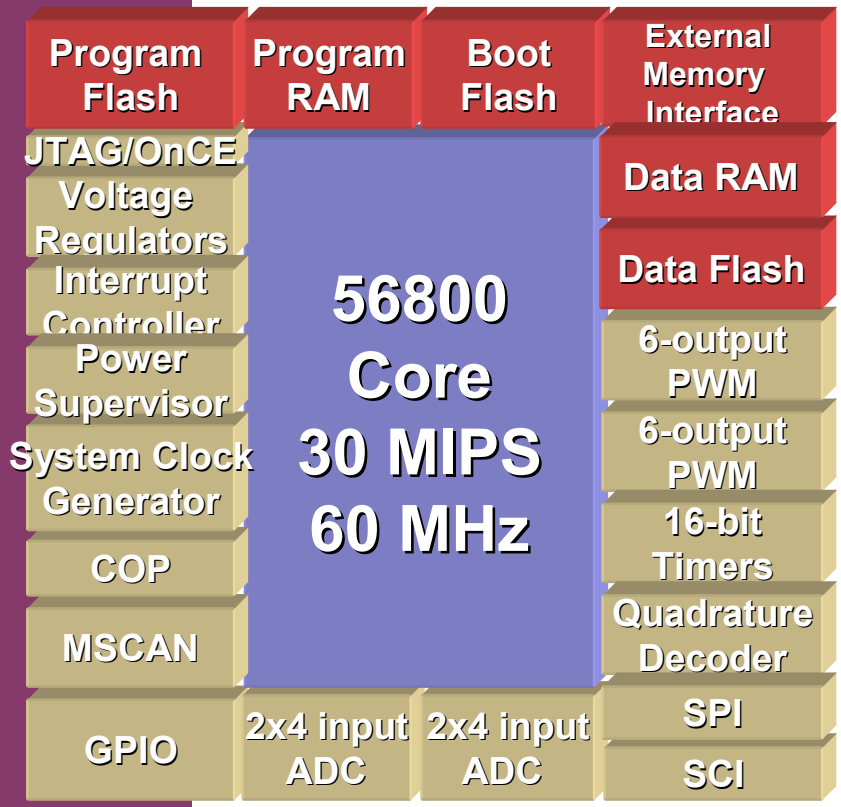
56F800 Series

	56F801	56F802	56F803	56F805	56F807	56F826	56F827
Performance	80MHz/40MIPS 60MHz/30MIPS	80MHz/40MIPS 60MHz/30MIPS	80MHz/40MIPS	80MHz/40MIPS	80MHz/40MIPS	80MHz/40MIPS	80MHz/40MIPS
Temp. Range	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C
Voltage	3.3V	3.3V	3.3V	3.3V	3.3V	2.5 & 3.3V	2.5 & 3.3V
On-Chip Flash	**12K x 16**	**12K x 16**	**38K x 16**	**38K x 16**	**70K x 16**	**35.5K x 16**	**67K x 16**
Program Flash	8K x 16	8K x 16	32K x 16	32K x 16	60K x 16	31.5K x 16	63K x 16
Data Flash	2K x 16	2K x 16	4K x 16	4K x 16	8K x 16	2K x 16	4K x 16
Boot Flash	2K x 16	2K x 16	2K x 16	2K x 16	2K x 16	2K x 16	Via Program Flash
On-Chip RAM	**2K x 16**	**2K x 16**	**2.5K x 16**	**2.5K x 16**	**6K x 16**	**4.5 X 16**	**5K x 16**
Program RAM	1K x 16	1K x 16	512 x 16	512 x 16	2K x 16	512 x 16	1K x 16
Data RAM	1K x 16	1K x 16	2K x 16	2K x 16	4K x 16	4K x 16	4K x 16
Ext. Memory Interface	-	-	Yes	Yes	Yes	Yes	Yes
PLL	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Watchdog Timer	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Interrupt Controller	Yes	Yes	Yes	Yes	Yes	Yes	Yes
16-bit Timers	8	8	8	16	16	4	4
Quadrature Decoder	-	-	1 x 4ch	2 x 4ch	2 x 4ch	-	-
PWM	1 x 6ch	1 x 6ch	1 x 6ch	2x 6ch	2 x 6ch	-	-
PWM Fault Input	1	1	3	4 + 4	4 + 4	-	-
PWM Current Sense Pins	0	0	3	3+ 3	3 + 3	-	-
12-bit ADC	2 x 4ch	5ch	2 x 4ch	2 x 4ch	4 x 4 ch	-	10ch
CAN 2.0 A/B	-	-	1	1	1	-	-
SCI (UART)	1	1	1	2	2	2	3
SPI (Synchronous)	1	-	1	1	1	2	2
SSI	-	-	-	-	-	1	1
GPIO (Ded./Shrd/Tot)	0 / 11 / 11	0 / 4 / 4	0 / 16 / 16	14 / 18 / 32	14 / 18 / 32	16 / 30 / 46	16 / 48 / 64
JTAG/OnCE	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Packages	48LQFP	32LQFP	100LQFP	144LQFP	160LQFP 160MBGA	100LQFP	128LQFP
Availability	Now	Now	Now	Now	Now	Now	Now

Launched by Motorola freescale semiconductor

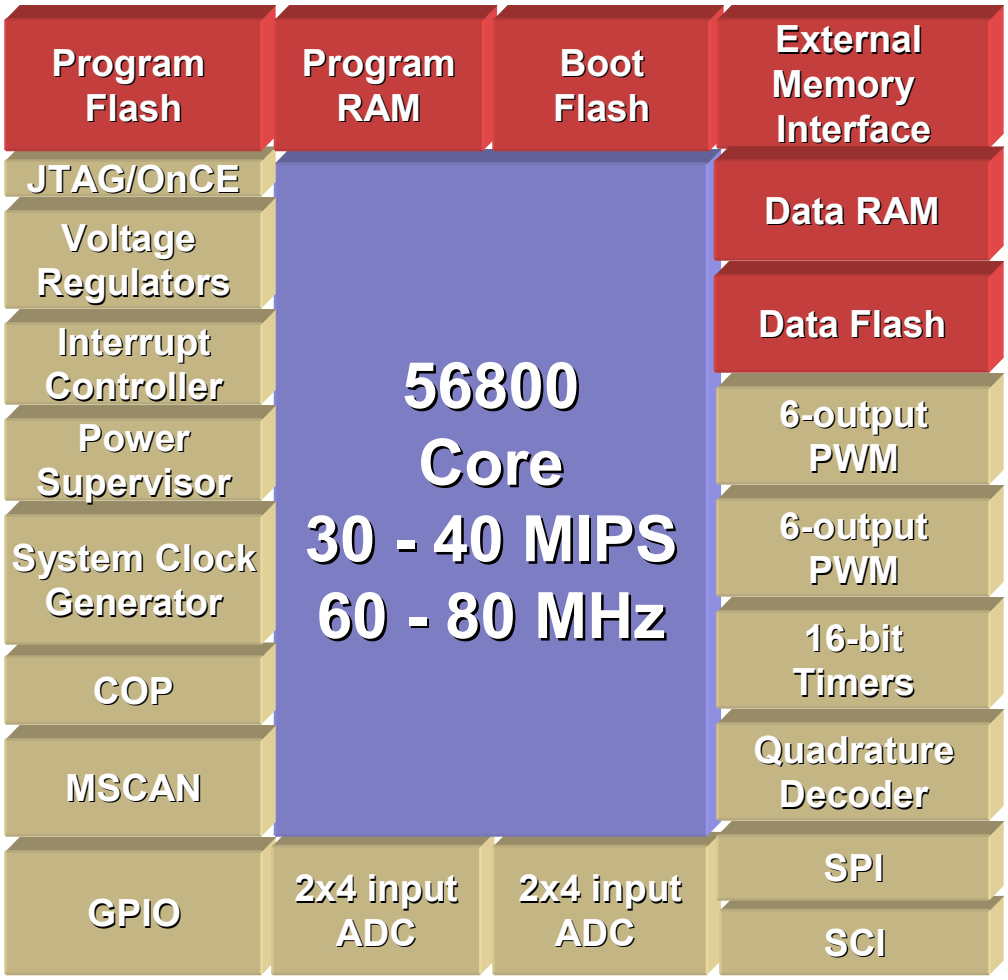
Low Cost Solutions 56F801/802 60 MHz/30 MIPS

- 30 MIPS Performance
- Program Memory
  - Up to 8 K x 16 FLASH
  - Up to 1 K x 16 RAM
  - 2K x 16 BootFLASH ™
- Data Memory
  - Up to 2 K x 16 FLASH
  - Up to 1 K x 16 RAM
- Serial Ports: SCI and SPI
- Dual, 4-channel, 12-bit ADC
- 6-Output PWM Module
- Eight 16-bit Timers (3 with external pins)
- COP/Watchdog Timer
- Up to 11 GPIO
- System Clock Generator
- On-chip Voltage Regulator & Power Supervisor
- Vectored Interrupt Controller
- JTAG/OnCE™ Debug Port
- Order Part Number
  - DSP56F801FA60 or Samples SPAK56F801FA60
  - DSP56F802TA60 or Samples SPAK56F802TA60
- Pricing from $2.50 - $5 suggested resale in volume



Launched by Motorola freescale semiconductor

56F800 Features



- ✓ 8 K-60 K Words Program FLASH, 512-2 K Words Program RAM
- ✓ 2 K-8 K Words Data FLASH, 1 K-4 K Words Data RAM
- ✓ 2 K Words BootFLASH™
- ✓ External Memory Interface
- ✓ Voltage regulator
- ✓ Software Programmable Phase Lock Loop
- ✓ On-Chip Relaxation Oscillator
- ✓ Power Supervisor
- ✓ COP Timer
- ✓ MSCAN Module – CAN 2.0A/B Compliant
- ✓ Up to two 6-Output PWM Modules
- ✓ Up to two Quadrature Decoders
- ✓ Up to four 4-Input 12-bit ADCs
- ✓ Up to sixteen 16-Bit General Purpose Timers
- ✓ Event Synchronizer
- ✓ Multiple Serial Ports (SCI, SPI)
- ✓ Up to 32 General Purpose I/O Pins
- ✓ Vectored Interrupt Controller
- ✓ JTAG/OnCE™ Debug Port
- ✓ 32 – 160 LQFP, 160 MBGA Packages
- ✓ 10Ku Pricing $3.16-$11.61 suggested resale

32 – 160 pins

Launched by Motorola freescale semiconductor

56800 Example Applications

General

- Medical Scanners
- Remote Monitoring
- Cable Test Equipment
- Noise Cancellation
- ID Tag Readers
- Traffic Light Control
- Underwater Acoustics

Industrial

- HVAC Blowers & Fans
- Lifts / Elevators / Cranes
- Frequency Inverters
- Variable Speeds Pumps
- Uninterruptible Power Supplies

Appliance

- Compressors
- Smart Appliances
- General Purpose Drives

Office / Home

- Printers / Fax / Scanners
- Exercise Equipment
- Electric Lawn Equipment
- Temperature Control

Why 56F800?

- Hybrid Architecture providing controller functionality and signal processing functionality and performance
- 30 & 40 MIPS FLASH performance
- Highly integrated features reduce system costs
- Priced to meet low cost application design goals
- Proven success - hundreds of customers in production today
- New SW tools strategy for low cost development
- Continual investment into family

56580 Series Overview

Launched by Motorola
freescale™
semiconductor

56850 Series

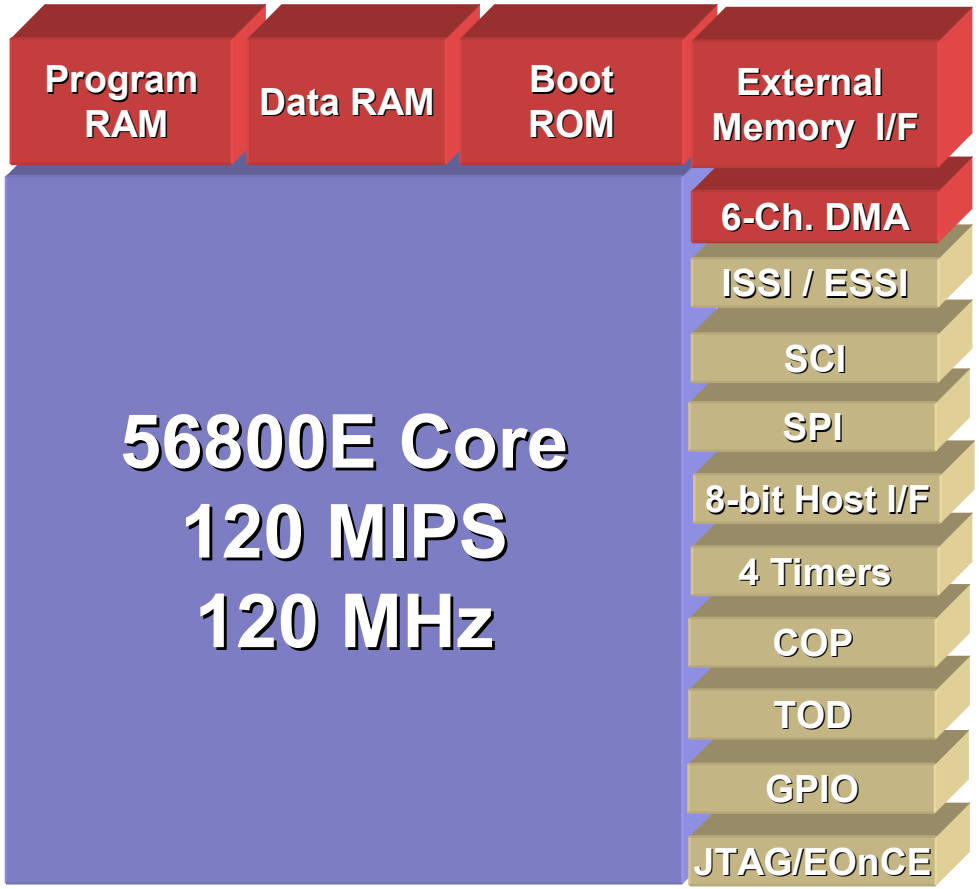
- Product Offerings
- Functionality
- Application Examples
- Why 56850?

56850 Series

	56852	56853	56854	56855	56857	56858
Performance	120MHz/120MIPS	120MHz/120MIPS	120MHz/120MIPS	120MHz/120MIPS	120MHz/120MIPS	120MHz/120MIPS
Temp. Range	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C	(-40, +85)°C
I/O Voltage	3.3V	3.3V	3.3V	3.3V	3.3V	3.3V
Core Voltage	1.8V	1.8V	1.8V	1.8V	1.8V	1.8V
On-Chip RAM	**22KB**	**34KB**	**66KB**	**58KB**	**130KB**	**130KB**
Program RAM	12KB	24KB	32KB	48KB	80KB	80KB
Data RAM	8KB	8KB	32KB	48KB	48KB	48KB
Boot RAM	2KB	2KB	2KB	2KB	2KB	2KB
Ext. Memory Expansion	4MB/12MB	4MB/16MB	4MB/16MB	4MB/16MB	-	4MB/16MB
Oscillator	Yes	Yes	Yes	Yes	Yes	Yes
PLL	Yes	Yes	Yes	Yes	Yes	Yes
Watchdog Timer	Yes	Yes	Yes	Yes	Yes	Yes
Interrupt Controller	Yes	Yes	Yes	Yes	Yes	Yes
16-bit Timers	4	4	4	4	4	4
SPI (Synchronous)	1	1	1	-	1	1
SCI	1	2	2	2	2	2
ISSI	1	-	-	-	-	-
ESSI	-	1	1	1	2	2
Parallel Host Interface	-	8-bit	8-bit	-	8-bit	8-bit
DMA	-	6-ch	6-ch	6-ch	6-ch	6-ch
GPIO (Max)	11	41	41	18	47	47
JTAG/EOnCE	Yes	Yes	Yes	Yes	Yes	Yes
Packages	81 MBGA	128 LQFP	128 LQFP	100 LQFP	100 LQFP	144 LQFP 144 MBGA
Availability	Now	Now	Now	Now	Now	Now

Launched by Motorola
freescale semiconductor

56850 Features



81 – 144 Pins

- ✓ 120 MIPS at 120 MHz
- ✓ 12 K-80 KBytes Program RAM
- ✓ 8 K-48 KBytes Data RAM
- ✓ 2 K Words Boot ROM
- ✓ 21 External Memory Address lines, 16 data lines and four programmable chip selects
- ✓ Software Programmable Phase Lock Loop
- ✓ Six independent channels of DMA
- ✓ Improved Synchronous Serial Interface (ISSI)
- ✓ Enhanced Synchronous Serial Interface (ESSI)
- ✓ Serial Port Interface (SPI)
- ✓ Serial Communication Interfaces (SCI)
- ✓ 8-bit Parallel Host Interface
- ✓ Four General Purpose 16-bit Timers
- ✓ JTAG/Enhanced On-Chip Emulation (EOnCE™) for unobtrusive, real-time debugging
- ✓ Computer Operating Properly (COP/Watchdog)
- ✓ Time of Day
- ✓ Up to 48 GPIO
- ✓ 10Ku Pricing $4.17-$7.34 suggested resale

56850 Example Applications

Telecommunications
- VoIP
- Low end Networking

Consumer
- Feature Phones
- IP Phones
- Karaoke
- Toys

Vehicle
- Hands-free
- Embedded Video

General
- Medical applications
- Security
- Voice recognition systems

Why 56850?

- Utilizes DSP Capability For High performance (120 MIPS) RAM Based Solutions
- Hybrid Architecture allows customers to implement both Voice and Communications Processing on one single device.
- Peripheral Set optimized for telecommunication
- Software modules library
- 32-bit performance with 16-bit code density

56F8300 Series Overview

Launched by Motorola
freescale™
semiconductor

56F8300 SERIES

- Product Offerings
- Low Cost 56F8300 Products
- Pin Compatibility
- Functionality
- Safety Features
- Application Examples
- Why 56F8300?

56F8300 Series

	56F8322	56F8323	56F8333	56F8334
Performance	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS
Temp. Range	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C
Voltage (Core / I/O)	2.5/3.3V	2.5/3.3V	2.5/3.3V	2.5/3.3V
On-Chip Flash	**48KB**	**48KB**	**80KB**	**80KB**
Program Flash	32KB	32KB	64KB	64KB
Data Flash	8KB	8KB	8KB	8KB
Boot Flash	8KB	8KB	8KB	8KB
On-Chip RAM	**12KB**	**12KB**	**12KB**	**12KB**
Program RAM	4KB	4KB	4KB	4KB
Data RAM	8KB	8KB	8KB	8KB
Flash Security	Yes	Yes	Yes	Yes
Ext. Memory Interface	-	-	-	Yes
Internal Voltage Regulator	On-Chip	On/Off-Chip	On/Off-Chip	On/Off-Chip
On-Chip Relaxation Osc.	Yes	Yes	Yes	Yes
16-bit Timers	8	8	16	16
Quadrature Decoder	1 x 4ch	1 x 4ch	1 x 4ch	1 x 4ch
PWM	1 x 6ch	1 x 6ch	1 x 6ch	1 x 6ch
PWM Fault Input	1	3	3	3
PWM Current Sense Pins	0	3	3	3
12-bit ADC	2 x 3ch	2 x 4ch	2 x 4ch	2 x 4ch
Temperature Sensor	YES	Optional	Optional	Optional
CAN	FlexCAN	FlexCAN	FlexCAN	FlexCAN
SCI (UART)	2	2	2	2
SPI (Synchronous)	2	2	2	2
GPIO (Ded./Shrd/Tot)	0 / 21 / 21	0 / 27 / 27	0 / 27 / 27	0/61/61
JTAG/EOnCE	Yes	Yes	Yes	Yes
Package	48LQFP	64LQFP	64LQFP	100LQFP

Launched by Motorola
freescale semiconductor

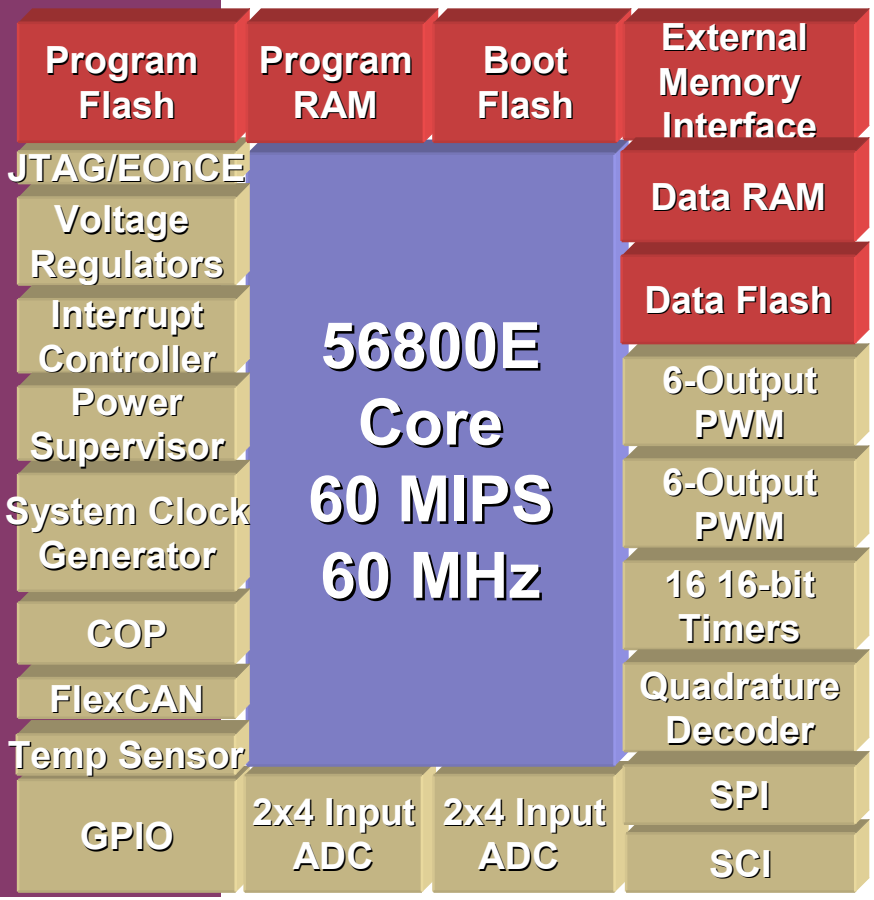
56F8300 Series

	56F8345	56F8346	56F8347	56F8355	56F8356	56F8357	56F8365	56F8366	56F8367
Performance	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS	60MHz/MIPS
Temp. Range	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C	(-40, +125)°C
Voltage (Core / I/O)	2.5/3.3V	2.5/3.3V	2.5/3.3V	2.5/3.3V	2.5/3.3V	2.5/3.3V	2.5/3.3V	2.5/3.3V	2.5/3.3V
On-Chip Flash	**144KB**	**144KB**	**144KB**	**280KB**	**280KB**	**280KB**	**560KB**	**560KB**	**560KB**
Program Flash	128KB	128KB	128KB	256KB	256KB	256KB	512KB	512KB	512KB
Data Flash	8KB	8KB	8KB	8KB	8KB	8KB	32KB	32KB	32KB
Boot Flash	8KB	8KB	8KB	16KB	16KB	16KB	16KB	16KB	16KB
On-Chip RAM	**12KB**	**12KB**	**12KB**	**20KB**	**20KB**	**20KB**	**36KB**	**36KB**	**36KB**
Program RAM	4KB	4KB	4KB	4KB	4KB	4KB	4KB	4KB	4KB
Data RAM	8KB	8KB	8KB	16KB	16KB	16KB	32KB	32KB	32KB
Flash Security	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Ext. Memory Interface	-	Yes	Yes	-	Yes	Yes	-	Yes	Yes
Internal Voltage Regulator	On/Off-Chip	On/Off-Chip	On/Off-Chip	On/Off-Chip	On/Off-Chip	On/Off-Chip	On/Off-Chip	On/Off-Chip	On/Off-Chip
On-Chip Relaxation Osc.	No	No	No	No	No	No	No	No	No
16-bit Timers	16	16	16	16	16	16	16	16	16
Quadrature Decoder	2 x 4ch	2 x 4ch	2 x 4ch	2 x 4ch	2 x 4ch	2 x 4ch	2 x 4ch	2 x 4ch	2 x 4ch
PWM	2 x 6ch	2 x 6ch	2 x 6ch	2 x 6ch	2 x 6ch	2 x 6ch	2 x 6ch	2 x 6ch	2 x 6ch
PWM Fault Input	4 + 4	3 + 4	3 + 4	4 + 4	3 + 4	3 + 4	4 + 4	3 + 4	4 + 4
PWM Current Sense Pins	3 + 3	3 + 3	3 + 3	3 + 3	3 + 3	3 + 3	3 + 3	3 + 3	3 + 3
12-bit ADC	4 x 4 ch	4 x 4 ch	4 x 4 ch	4 x 4ch	4 x 4ch	4 x 4ch	4 x 4 ch	4 x 4ch	4 x 4ch
Temperature Sensor	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional	Optional
CAN	FlexCAN	FlexCAN	FlexCAN	FlexCAN	FlexCAN	FlexCAN	FlexCAN (2)	FlexCAN (2)	FlexCAN (2)
SCI (UART)	2	2	2	2	2	2	2	2	2
SPI (Synchronous)	2	2	2	2	2	2	2	2	2
GPIO (Ded./Shrd/Tot)	21/ 28 / 49	0 / 62 / 62	0 / 76 / 76	21 / 28 / 49	0 / 62 / 62	0 / 76 / 76	21 / 28 / 49	0 / 62 / 62	0 / 76 / 76
JTAG/EOnCE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Package	128LQFP	144LQFP	160LQFP	128LQFP	144LQFP	160LQFP	128LQFP	144LQFP	160LQFP

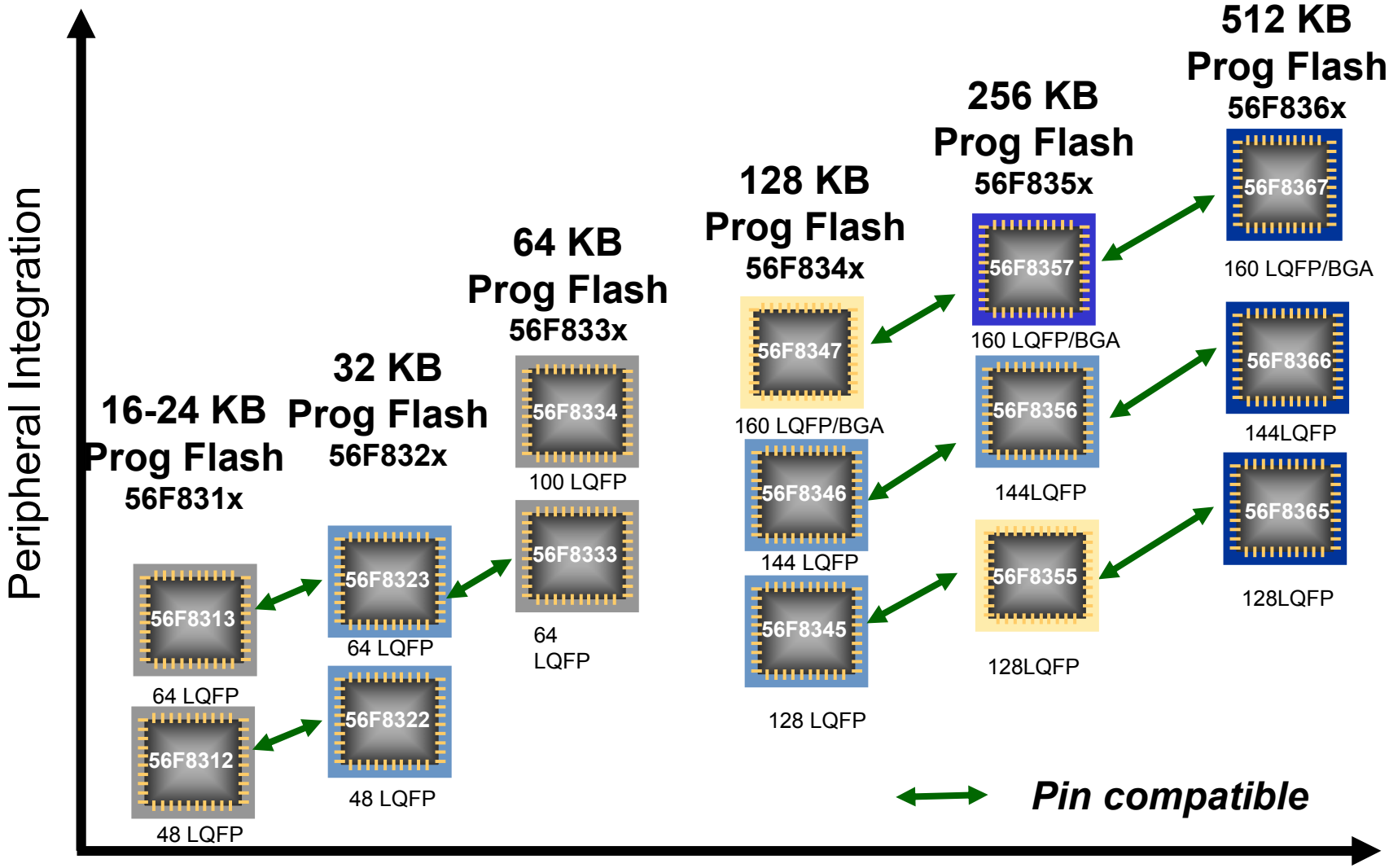
Launched by Motorola
freescale semiconductor

Low Cost 56F8300 Solutions - 56F8322/F8323 60 MHz/60 MIPS

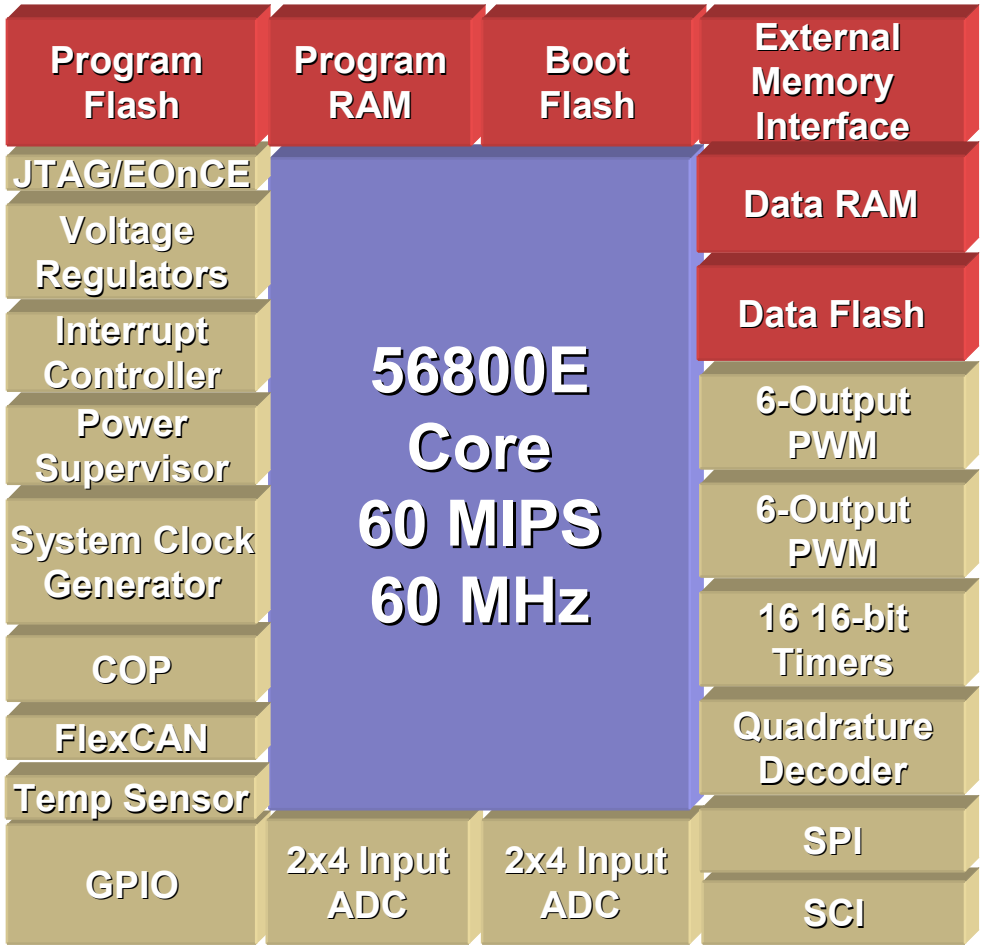
- 60 MIPS Performance
- Program Memory
  - Up to 32 KBytes x 16 FLASH
  - Up to 4 KBytes x 16 RAM
  - 8 KBytes x 16 BootFLASH ™
- Data Memory
  - Up to 8 KBytes x 16 FLASH
  - Up to 8 KBytes x 16 RAM
- Serial Ports: SCI and SPI
- Dual, 3-channel, 12-bit ADC or Dual, 4-channel, 12-bit ADC
- 6-Output PWM Module
- Eight 16-bit Timers
- COP/Watchdog Timer
- Up to 27 GPIO
- System Clock Generator
- On-chip Voltage Regulator andPower Supervisor
- Vectored Interrupt Controller
- JTAG/OnCE™ Debug Port
- Order Part Number
  - MC56F8322VFA60 or Samples SPAK56F8322FA60
  - MC56F8323VFB60 or Samples SPAK56F8323FB60
- Pricing from $5.50-5.75 suggested resale in volume



56F8300 Series Products



56F8300



48 - 160 Pins

- ✓ 60 MIPS Harvard Architecture Core
- ✓ 32 K-512 KBytes Program Flash, 4K Bytes Program RAM
- ✓ 8 K-32KBytes Data RAM, 8K-32K Bytes Data Flash
- ✓ 8 K-16 KBytes BootFLASH™
- ✓ Code Security Interlock feature for Flash Memory
- ✓ External Memory Interface with 24 Address Lines, 16 Data Lines and 8 Programmable Chip Selects
- ✓ On chip Voltage regulator and ADC reference
- ✓ System Clock Generator - Dynamically Program System Clock Frequency
- ✓ Power Supervisor – Power on Reset and Low Voltage detection.
- ✓ Computer Operating Properly Timer
- ✓ Up two FlexCAN Module – CAN 2.0 A/B Compliant
- ✓ Up to two 6-Output PWM Modules
- ✓ Up to four 4-Input 12-bit ADC
- ✓ Up to two Quadrature Decoders
- ✓ Up to sixteen 16-Bit General Purpose Timers
- ✓ Multiple Serial Ports – SCI & SPI
- ✓ Up to 77 GPIO
- ✓ JTAG/EOnCE™ Debug Port
- ✓ **10Ku Pricing $7.20-$22.00 suggested resale**

Launched by Motorola
freescale semiconductor

56F8300 Safety Features and Benefits

Power Supervisor – Power on Reset (POR)

✓ Facilitates graceful system shutdown and restart in the event of a power supply failure

Power Supervisor – Low Voltage Interrupt (LVI)

✓ Monitors input voltage and generates an interrupt to allow shutdown of the chip if it falls below pre-defined levels. This helps to prevent continued operation that results in damage or incorrect results due to low-voltage levels.

On-chip Clock Synthesis (OCCS)

OCCS - Loss of Clock (LOC) Detection

✓ Enables systems to pass the “Cut Crystal Test”. This insures graceful shutdown if something happens to the crystal (physical damage, EMC, etc) during normal operation

OCCS – Loss of PLL Lock (LOL) Detection

✓ Monitors the stability of the generated clock to insure it is in sync with the reference clock.

OCCS – Generates an Interrupt on LOC,LOL, or both

✓ Each of the above conditions notifies the processor so that it can respond appropriately if the condition occurs, i.e. gracefully shutdown the system.

OCCS – Safety shutdown feature available in the event the PLL Reference clock disappears

✓ As in each of the above, enables graceful shutdown to prevent damage to the system.

56F8300 Safety Features & Benefits

Temperature Sensor – Device temperature monitoring using ADC channel

Measures temperature of die & using an ADC informs system that the temperature has exceeded limits.

- ✓ This enables the processor to adjust operations and or shutdown operations as appropriate when warning or excessive temperature levels are detected.
- ✓ Allows engineers to more closely identify actual operating temperature ranges enabling system cost savings through optimized thermal designs and component selection.

Computer Operating Properly (COP) – run-away code recovery

- ✓ Insures recovery from runaway code conditions to prevent incorrect results and/or resulting damage to the system.

Quad Decoder – Zero speed watchdog

- ✓ Detects that the motor has stopped spinning so that an alert may be generated or adjustments made.
- ✓ Also monitors the connection between the controller and the encoder to insure that position information is being transferred

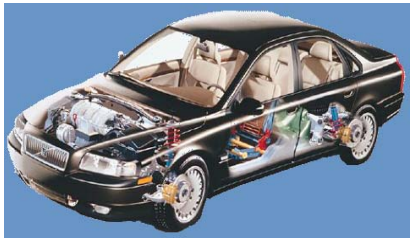
FLASH Security

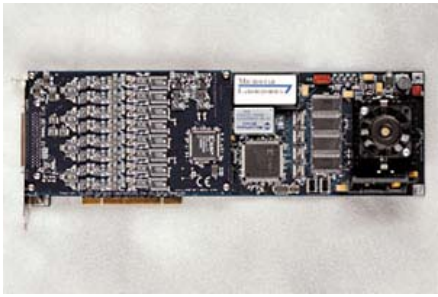
56F8300 Example Applications

Industrial
- Uninterruptible Power Supplies
- Power Supplies
- Frequency Inverters
- Protection Relay
- Sensorless Control
- Valve Actuators

Vehicle
- EPAS
- Braking
- Transmission
- Active Suspension
- Valve Actuators

General
- Medical applications
- Currency (bill) validators
- Medical instrumentation
- Engine performance modules
- Exercise Equipment
- Security and Safety Systems
- Vending machines
- Home automation
- Intelligent toys
- Metering
- Retail scanners





Why 56F8300?

- Highest performance Flash (60 MIPS) in the hybrid controller series
- Natural progression to the 56F800 family (Controller Continuum)
- Flash Security
- 32-bit performance with 16-bit code density
- Driving Development Tools Strategy

Launched by Motorola
freescale semiconductor

56800/E Family and Peripheral Summary

- Wide range of devices offering different levels of performance, memory, and pin packages so you buy only what you need

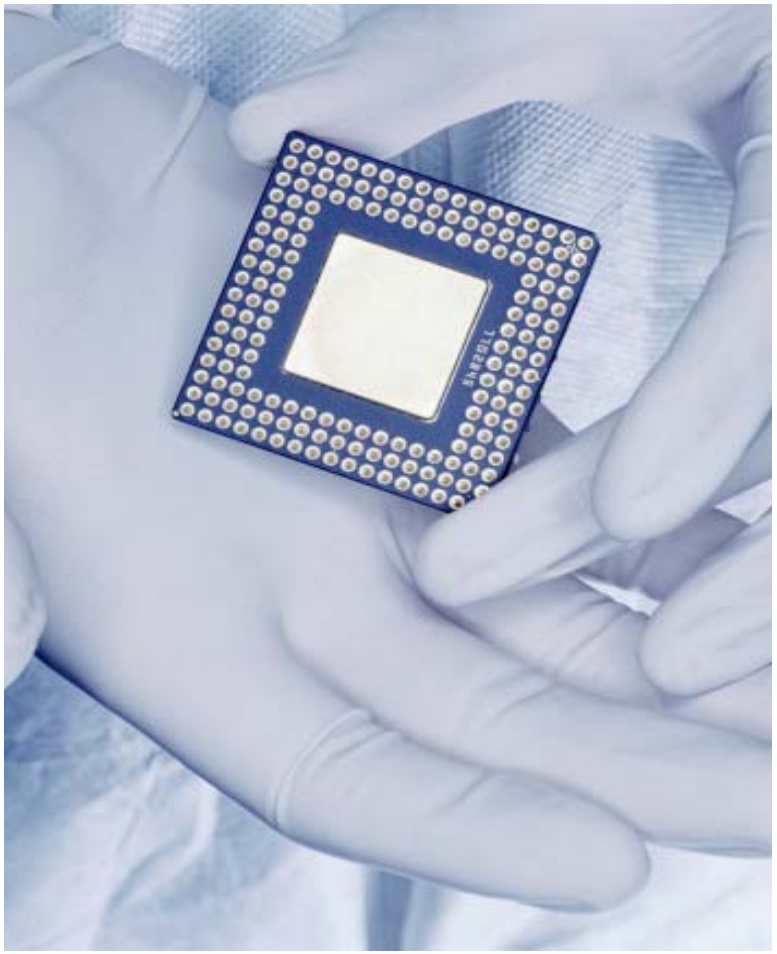
- Feature rich integrated peripherals reducing your system costs

Hybrid MCU Architecture Introduction

56800/E Summary

Leading the Way with the Controller Continuum

- Broad range of multi-function productions
  - Leveraging products into a wide range of general markets: consumer, industrial, vehicle
  - Providing a performance migration with new and innovative products
  - Providing solutions from low end 8-bit to 32-bit
- Premiere supplier in connectivity
- Performance/price points to fit application needs
- Leadership in embedded memory
- Service and support
  - Complementary tools across the families
  - On-line training
  - Workshops

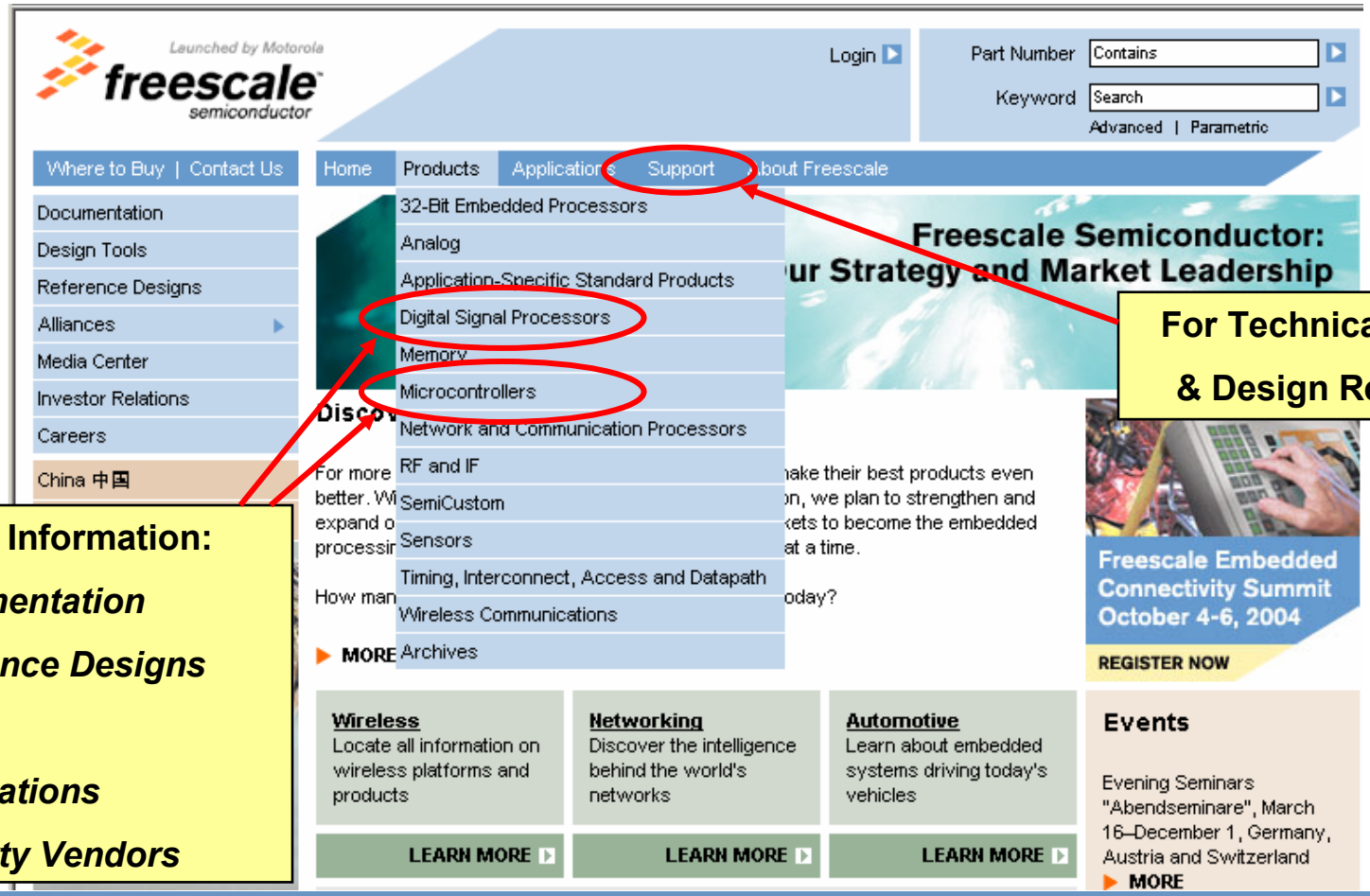


56800/E Documentation and Technical Support

Hybrid MCU Architecture Introduction

Documentation and Technical Support

All Located at: www.freescale.com



For Technical Support & Design Resources

56800/E Information:

- ***Documentation***
- ***Reference Designs***
- ***Tools***
- ***Applications***
- ***3rd Party Vendors***

Launched by Motorola freescale semiconductor

Reference Material Reorganization

56800 Core

- 56800 Family Manual

56F800 Series

- 56F800 User's Manual
- 56F8xx Data Sheets

The Rest…

- Product Briefs
- Application Notes
- Device Errata

Family

Device

Device

56F8300 Core

- 56F8300 Reference Manual

56F8300 Series

- 56F8300 Peripherals Manual
- 56F8300 Data Sheets

The Rest…

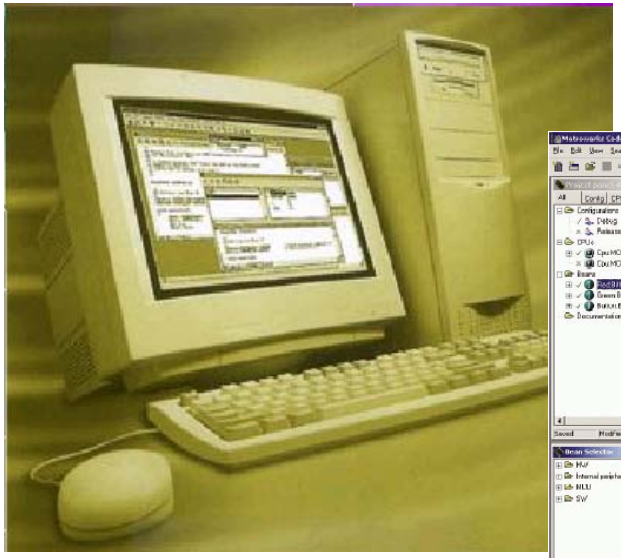
- Product Briefs
- Application Notes
- Device Errata

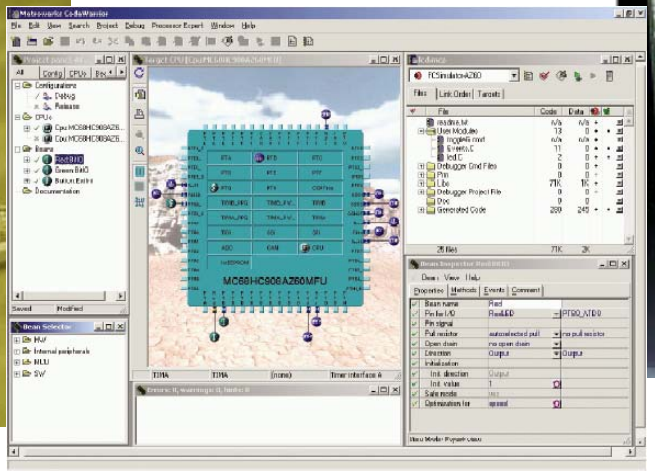
56800/E Development Tools Introduction
Hybrid MCU Architecture Introduction

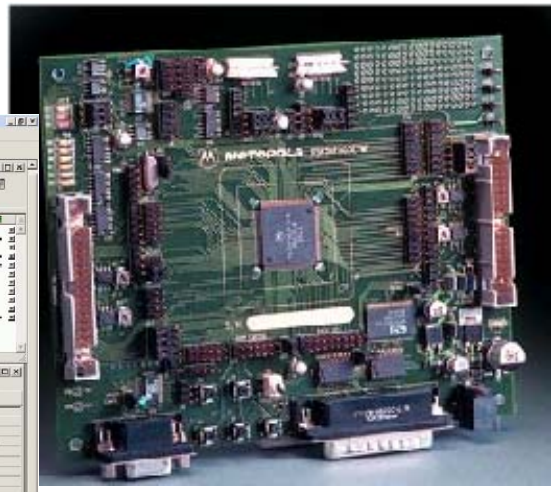
56800/E Development Environment

- The Complete Solution
- CodeWarrior Overview
- Processor Expert Overview
- EVB (Evaluation Board) Kits
- Low Cost Demo Kits
- Demos and Reference Designs
- 8/16-Bit Products Division Tools Strategy
- Tools Pricing
- Hybrid Controller Third Party Support

The Complete Development Environment







CodeWarrior for 56800/E

CodeWarrior for Freescale 56800/E is a windows based visual IDE that includes an optimizing C compiler, assembler and linker, project management system, editor and code navigation system, debugger, simulator, scripting, source control, and third party plug in interface.

Processor Expert

Processor Expert (PE) provides a Rapid Application Design (RAD) tool that combines easy-to-use component-based software application creation with an expert knowledge system. PE is fully integrated with the CodeWarrior for 56800/E.

Hardware Tools

The 56800/E solutions are supported with a complete set of evaluation modules which supply all required items for rapid evaluation and software and hardware development. In addition several command converter options exist for customer target system debugger connection.

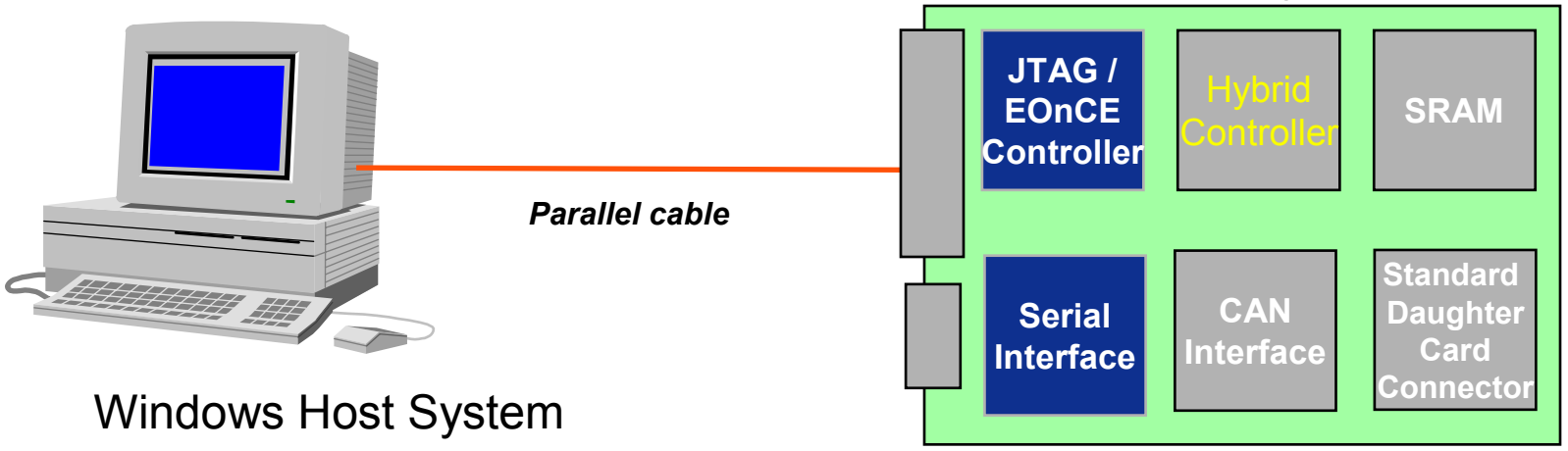
56800/E Evaluation Board (EVB)

Evaluation Board (EVB) Kit

The EVB kit includes everything required to start developing code immediately. It includes the evaluation board, all documentation, required cabling, power supply, CodeWarrior IDE, Processor Expert, and the training CD.

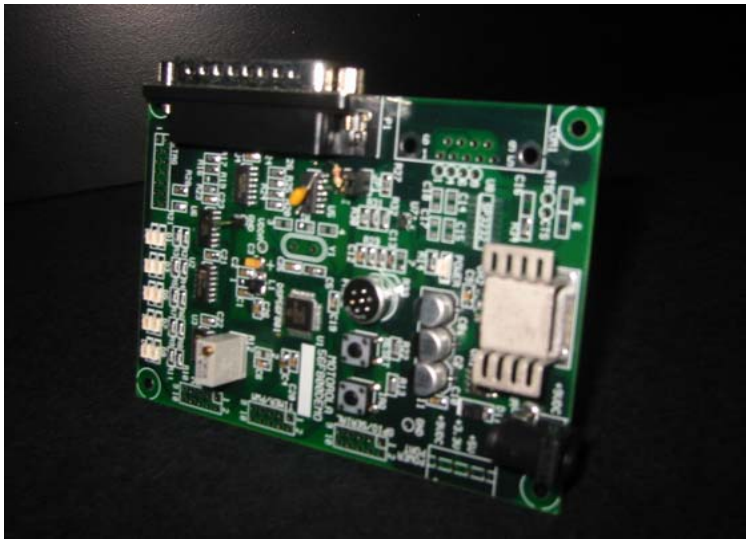
Standard Features:

- ✓ Parallel port Connection to Host PC
- ✓ Non intrusive debug via OnCE/EOnCE port
- ✓ JTAG Connector
- ✓ RS232 Serial connector
- ✓ Expansion Memory
- ✓ Standard daughter card connection
- ✓ CAN PHY layer
- ✓ Universal Power Supply
- ✓ Code Warrior CD w/30 day evaluation license
- ✓ Processor Expert
- ✓ Hands on training CD



56F800 Demonstration Kit

- Demo Board
- Complimentary CodeWarrior™ License
  - 16 KBytes Program Memory Limit
- Parallel Cable
- Directions for Kit
- Training CD
- Ordering Part Number and Price:
  - DSP56F800DEMO
    - --$49.95 (US power supply) suggested resale
  - DSP56F800DEMO-E
    - --$64.95 (International power supply) suggested resale



56F800 Demonstration Application Examples

Functionality Demos included with Demo Kit:

Quad Timer Flashing LEDs
- Uses Quad Timer and Quad Timer Compare Capabilities

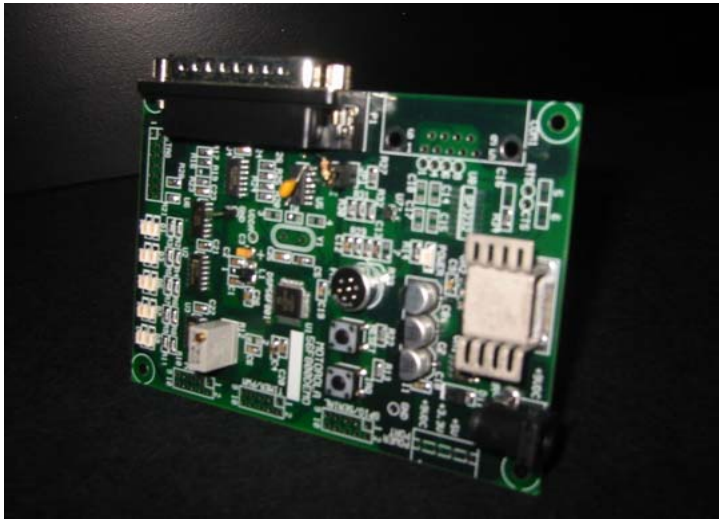
Timer Driver Flashing LEDs
- Uses Timer Peripherals and Clock Capabilities

Frequency Spectrum with Amplitude
- Uses ADC, PWM, and GPIO to Perform FFT on Sampled Data to Determine and Display Frequency Content and Amplitude of Analog Signal

Frequency Spectrum without Amplitude
- Uses ADC, PWM, and GPIO to Perform FFT on Sampled Data to Determine Frequency Content of Analog Signal

Frequency Detector
- Exercises ADC, PWM and GPIO to perform FFT on Sampled Data to Determine Frequency Content of Analog Signal. Uses LEDs Connected to PWM and GPIO to Display Binary Representation of the Strongest Frequency Detected.

Potentiometer Controlled LEDs
- Uses ADC, PWM, and GPIO. Voltage Divider Controlled by a Potentiometer Samples ADC and via LEDs Connected to PWM and GPIO Displays Amplitude of Signal.



56F8300 Developer's Starter Kit

- Everything required to start developing code immediately
- All documentation, required cabling, power supply and more
- Parallel port connection to host PC
- JTAG connector
- CAN PHY layer
- Non-intrusive debug via EOnCE port
- On-board MC33794 E-Field sensor
- Universal power supply
- Code Warrior CD with Processor Expert

56800/E Demos and Reference Designs

Vehicle

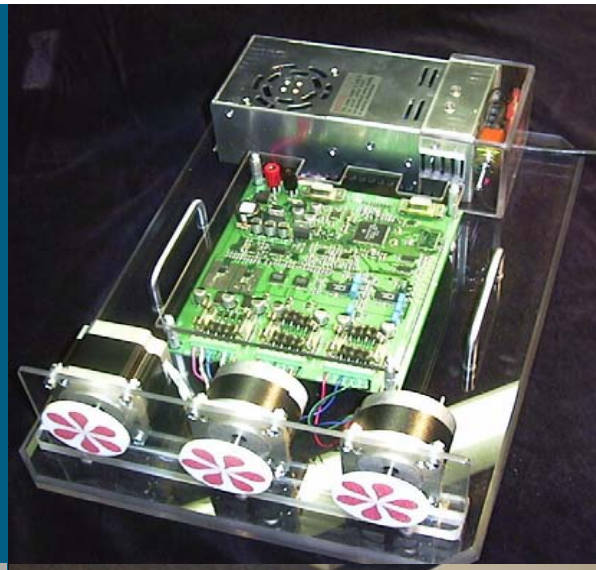
- Electronic Power Assisted Steering (EPAS) Demo
  - 56F805 Version and 56F8345 Version
- Hybrid Braking Demo in Definition

Industrial

- UPS Reference Design in Development
- Powerline Modem
- Switch Mode Power Supply in Definition
- Low Cost 56F800
- Motor Control Demos
  - BLDC
  - Switch Reluctance
  - Sensorless ACIM
  - Stepper Motors

Voice

- Speaker Feature Phone
- Hands-free (AEC/NS)
- Voice Recognition



Comprehensive Development Tools

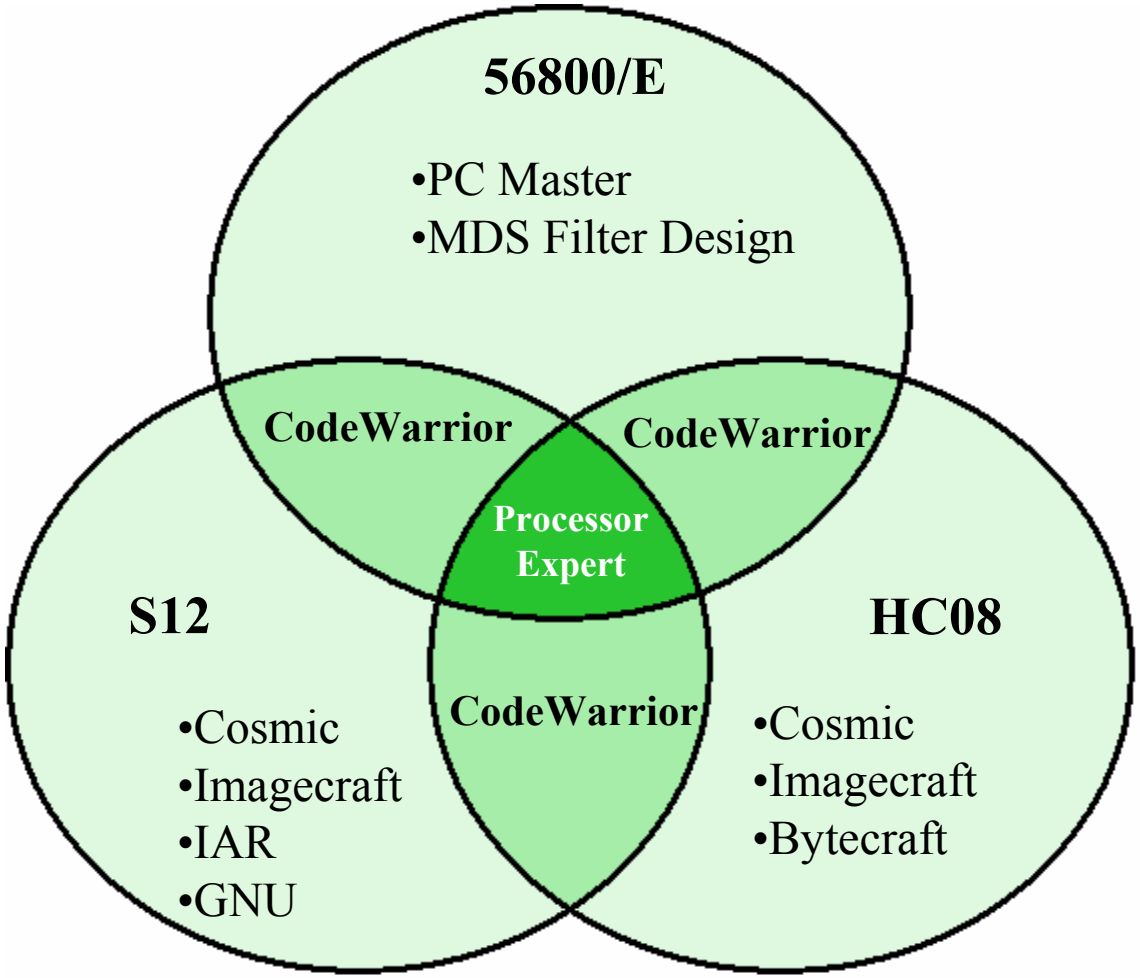
Single tools solution across Freescale's 16-bit controllers

- Invest in one tool set and develop across multiple hardware platforms: from legacy MCUs to 56F8300
- Broad software support includes motor control, industrial, automotive, and general purpose applications
- CodeWarrior by Metrowerks: highly visual development, code generation

Evaluation Boards (EVB) Kits

- Exceptional "out of box" experience
- Full featured processor evaluation boards with full emulation
- Extensive line of add-on boards enable specific motion and industrial control application development

8 and 16-Bit Products Division Software Development Tools



8/16-Products Compiler Offerings

	FREE	$495	$995	Unlimited
HC(S)08	4 KBytes	32 KBytes	64 KBytes	$1995
HCS12	12 KBytes	32 KBytes	64 KBytes	$2495
56800/E	16 KBytes	64 KBytes	128 KBytes	$1495

Third Party Hybrid Controller Support

- **Domain Technologies: www.domaintec.com**
  - Development systems: Emulators, debuggers,interfaces and libraries.
- **Lauterbach: www.lauterbach.com**
  - Modular development tools ranging from In-Circuit Emulators and Logic Analyzers to Pattern and Stimulus Generators
- **Macraigor: www.macraigor.com**
  - Host to target connections including parallel port, ISA bus, PCI bus, Serial and Ethernet.
- **P & E Micro:**
  - In-Circuit Debugger (ICD) Software, Parallel-Port Interface, Cable (DSP Cable), Flash Programmer (PROG) Software and Register File Viewer/Editor
- **System General: www.sg.com.tw**
  - Manual and Automated Device Flash Programmers

Third Party Hybrid Controller Support

Motor Boards & Development Systems

- **Micromint: www.micromint.com**
  - Motorman GUI Configured Embedded Motion Control Module
- **New Micros: www.newmicros.com**
  - NMIN-0803 & IsoPod Single Board Motion Controllers

RTOS/Network Stacks

- **Unicoi (DSP OS): www.unicoi.com**
  - DSPOS RTOS; Ethernet Daughtercards; Softworks Fusion Internet (TCP/IP stack), LAN, network management, Web software
- **Micrium: www.micrium.com**
  - uCOS RTOS
- **Quadros: www.quadros.com**
  - RTXC Quadros RTOS

Voice Solutions

- **Clarity: www.clarityco.com**
  - CVC ClearVoice Capture Noise Suppression Solutions