Embedded Connectivity Summit 2004

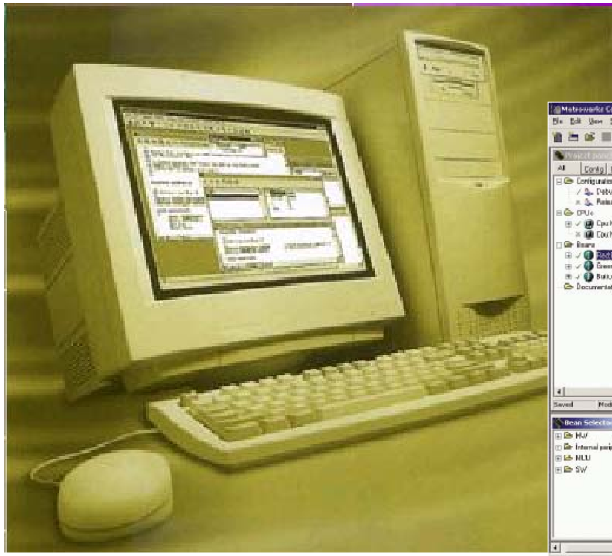
Efficient C for 56800E

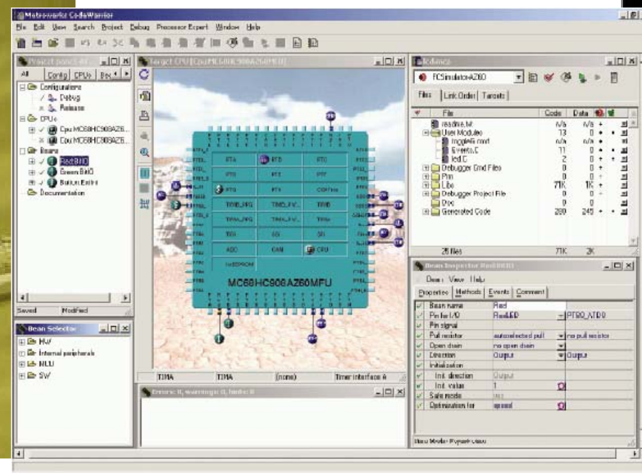
Launched by Motorola
freescale™
semiconductor

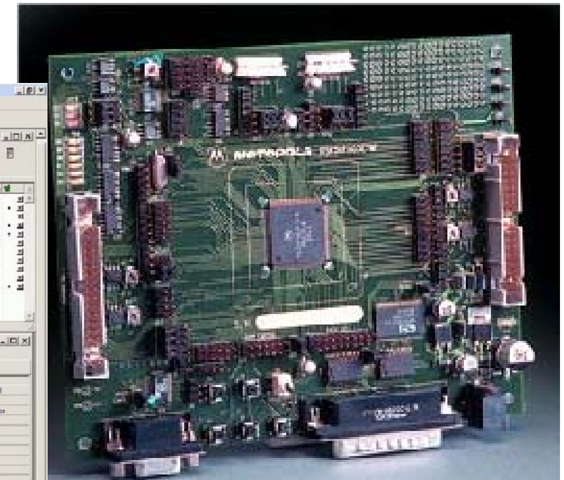
Agenda

- Tools Overview
- Compiler Efficiency
- 56800/E Core Dependencies
  - Memory Models
  - 8/16/32 bit data types
- Misc. Techniques
- Utilizing HW Features
  - Do Loops
  - Intrinsic Functions
- Optimized SW Libraries
- Utilizing Processor Expert and PESL
- From Efficiency to Safety

The Complete Development Environment







CodeWarrior for 56800/E

CodeWarrior for Motorola 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.

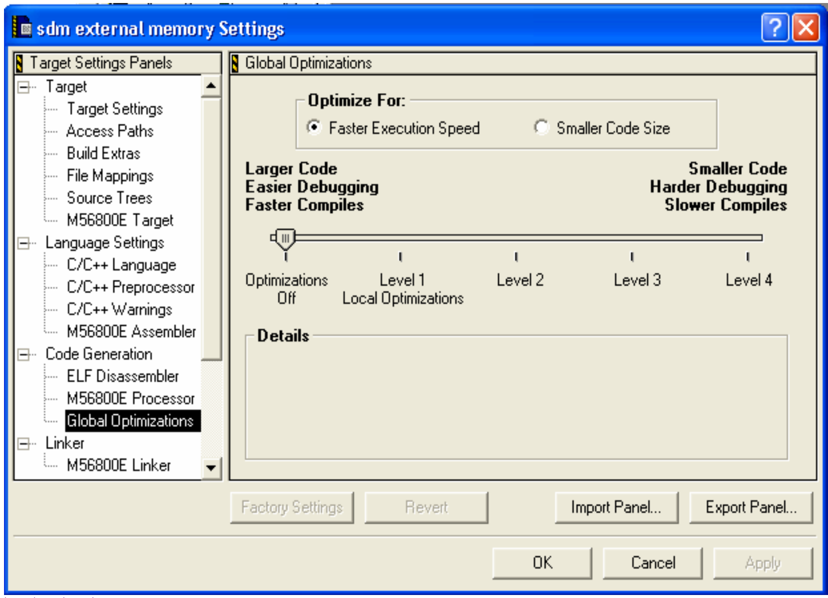
Hardware Tools

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.

Compiler Code Efficiency

- **Use Level 4 Optimizations**
  Efficiency of hand-coded assembly can be reached

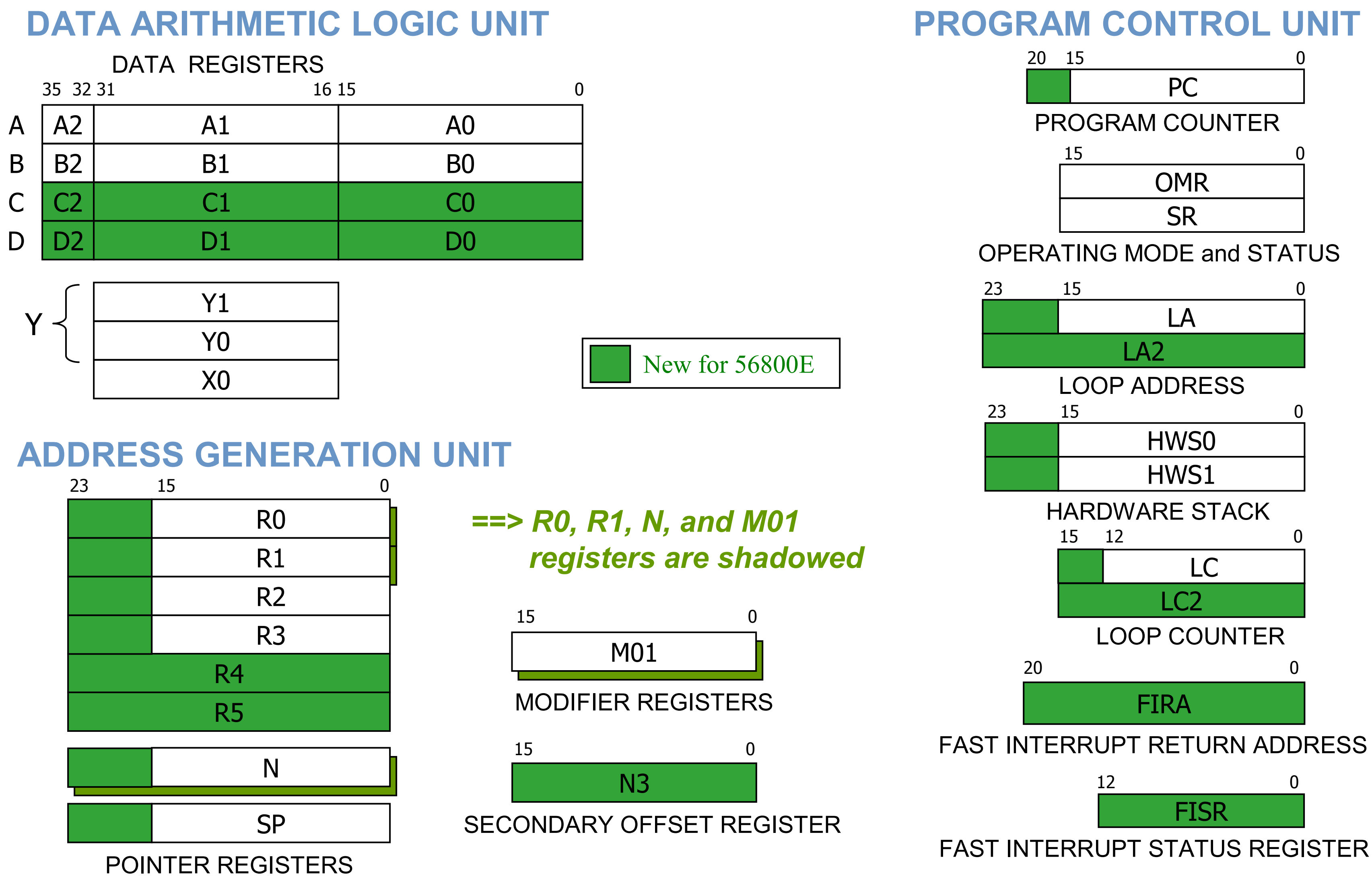
- **Advanced techniques that assist the compiler.**
  Calling convention considerations
  Effects of type casting
  Large memory model options



MCU Func ->	sort	Bits	max	search	CRC	init
asm	23	30	16	14	31	12
C - 0	67	50	32	31	76	34
C - 1						
C - 2						
C - 3						
C - 4	23	31	16	14	31	14
C overhead	**0.0%**	**3.3%**	**0.0%**	**0.0%**	**0.0%**	**16.7%**

Efficiency Hint:
Use CW Optimizations

Small / Large Memory Models



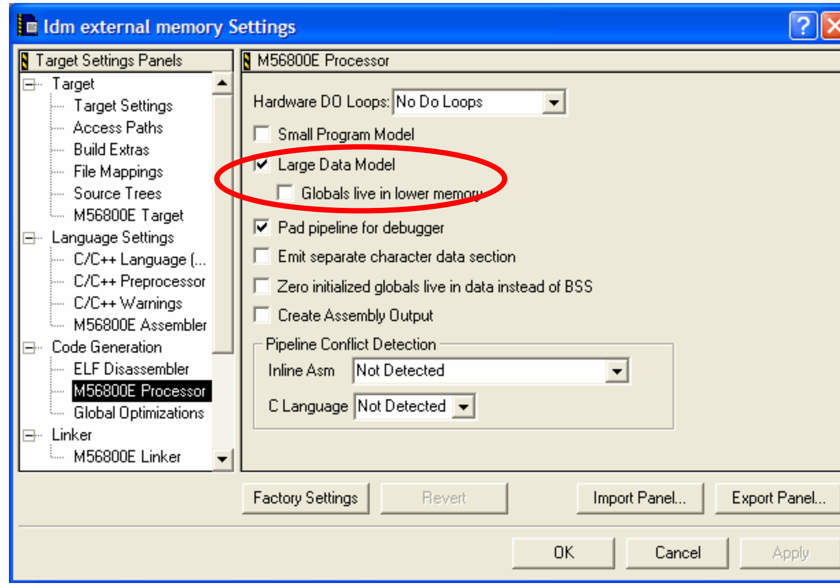
Small / Large Memory Models

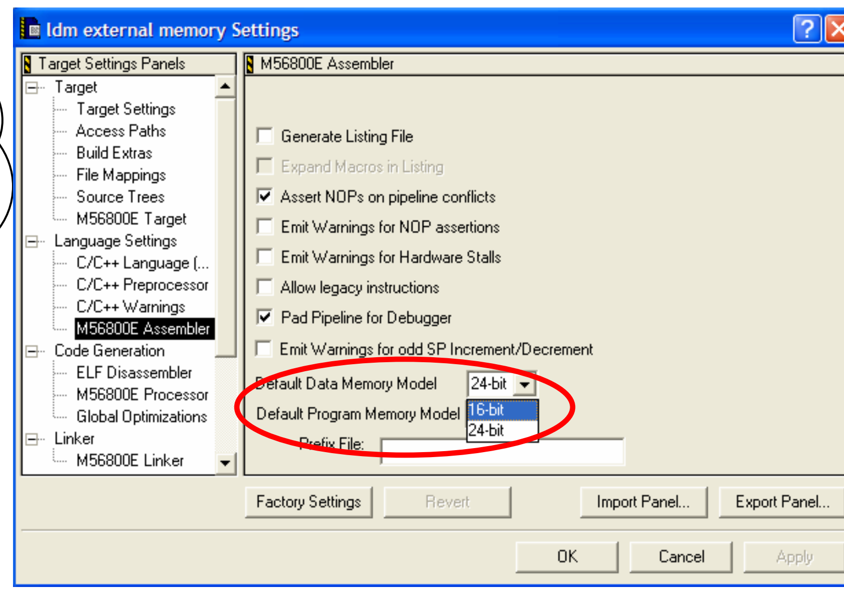
Memory model comparison (Bubble Sort)

```
int vector[] = { 3,7,6,1,2,5 };
int next;
int main()
{
    int i=0, j=0;
    int sz = sizeof(vector)/sizeof(int);

    for (i=0; i<sz; i++){
        for(j=0; j<sz-i; j++){
            if (vector[j]>vector[j+1]) {
                next=vector[j];
                vector[j]=vector[j+1];
                vector[j+1]=next;
            }
        }
    }

}
```

Efficiency Hint:
Use appropriate memory model

Small Data Model	Large Data Model	Large Data Model + Globals live in lower memory
579 cycles	760 cycles	729 cycles





Casting with Non 16-bit Data Types

Efficiency Hint: Avoid casting to/from char and long

- int to long

```
int ls;
long ll;

ll = (long)ls;
```

```
move.w    X:(SP-2),A;
asr16     A,A
move.l    A10,X:(SP-4)
```

- int to char

```
char lc;
int ls;

lc = (char)ls;
```

```
move.w    X:(SP-2),A
sxt.b     A,A
move.b    A1,X:(SP)
```

- char to long

```
long ll;
char lc;

ll = (long)lc;
```

```
moveu.b   X:(SP),A
sxt.b     A,A
asr16     A,A
move.l    A10,X:(SP-4)
```

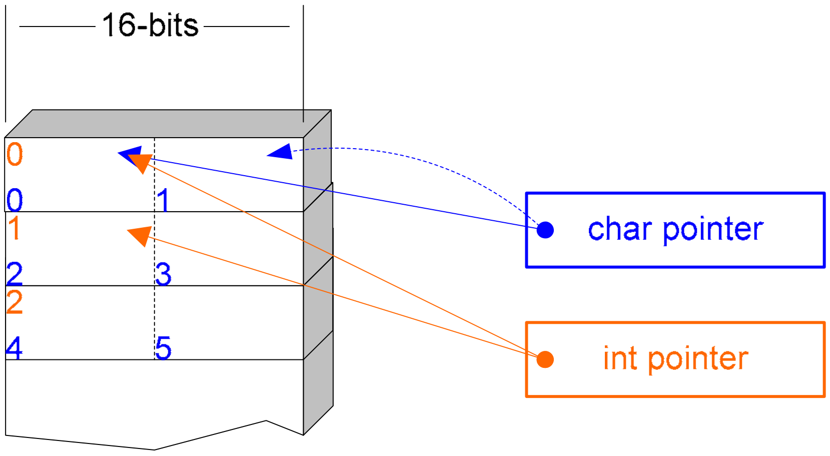
Casting with Void & Byte Pointers

Efficiency Hint: Limit casting to/from char & void pointers

```
void * pvoid;
int    vint;
int  * pint;
char * pchar;


pint = (int *)&vint;
adda      #-5,SP,R0
move.w    R0,X:(SP-6)

pvoid = (void *)pint;
moveu.w   X:(SP-6),R0
asla      R0,R0
move.w    R0,X:(SP-4)

pchar = (char *)pint;
move.w   X:(SP-6),R0
asla      R0,R0
move.w    R0,X:(SP-7)

pint = (int *)pvoid;
moveu.w   X:(SP-4),R0
lsra      R0
move.w    R0,X:(SP-6)
```

Casting penalty:

1 word, 1 cycle



Other Efficiency Hints

- Initialize local arrays and structures at declaration time, if possible. Local arrays and structures are initialized optimally by the compiler.
- Functions with a large number of parameters will probably have to pass some parameters on the stack causing costly memory accesses. Make sure that frequently called functions pass their parameters in registers. Chapter 6 of the targeting manual documents the parameter passing rules for the DSP56800E C Compiler.
- Loading frequently used global variables into local temporary variables sometimes has a positive effect on code size and performance, since accessing variables through registers is more efficient that absolute addressing modes.

Example: Loading Global Variable into Local Temp

Code Using Globals

(98 cycles, 20 words)

```
#define ARRAY_SIZE 5

static struct s1
{
   unsigned char value_a;
   unsigned char value_b;
   unsigned char value_c;
} s_s1[ARRAY_SIZE];

unsigned int r1;

int main()
{
   int i;
   for (i = 0; i < ARRAY_SIZE; i++)
   {
      r1 += s_s1[i].value_a;
      r1 += s_s1[i].value_b;
      r1 += s_s1[i].value_c;
   }
   return (r1);
}
```

Code Using Locals

(57 cycles, 13 words)

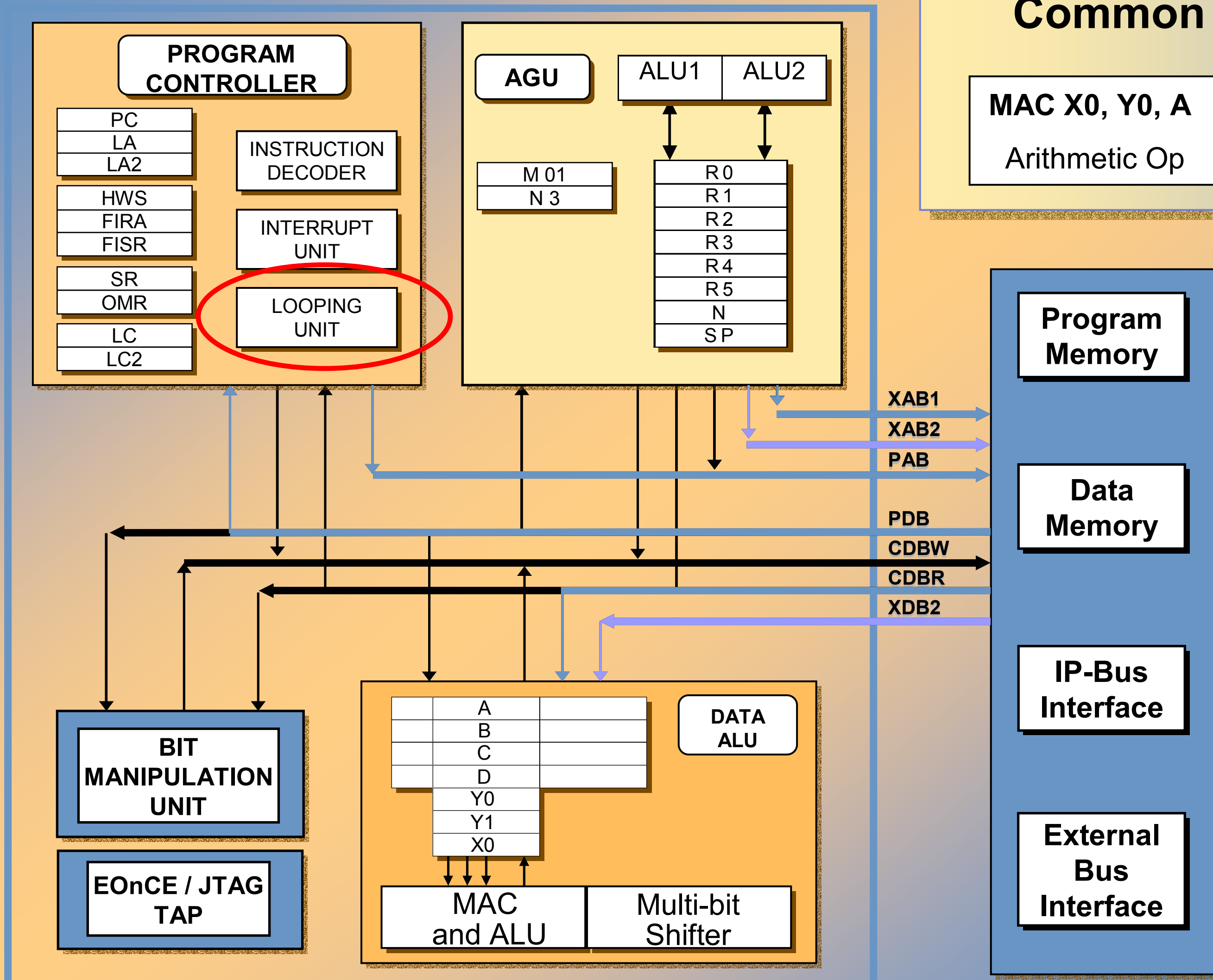
```
int main()
{
   int i;
   unsigned int local_var;

   local_var = r1;
   for (i = 0; i < ARRAY_SIZE; i++)
   {
      local_var += s_s1[i].value_a;
      local_var += s_s1[i].value_b;
      local_var += s_s1[i].value_c;
   }

   r1 = local_var;

   return (r1);
}
```

Efficiency Hint:
Use local variable substitution

Utilizing HW Do Loops

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:

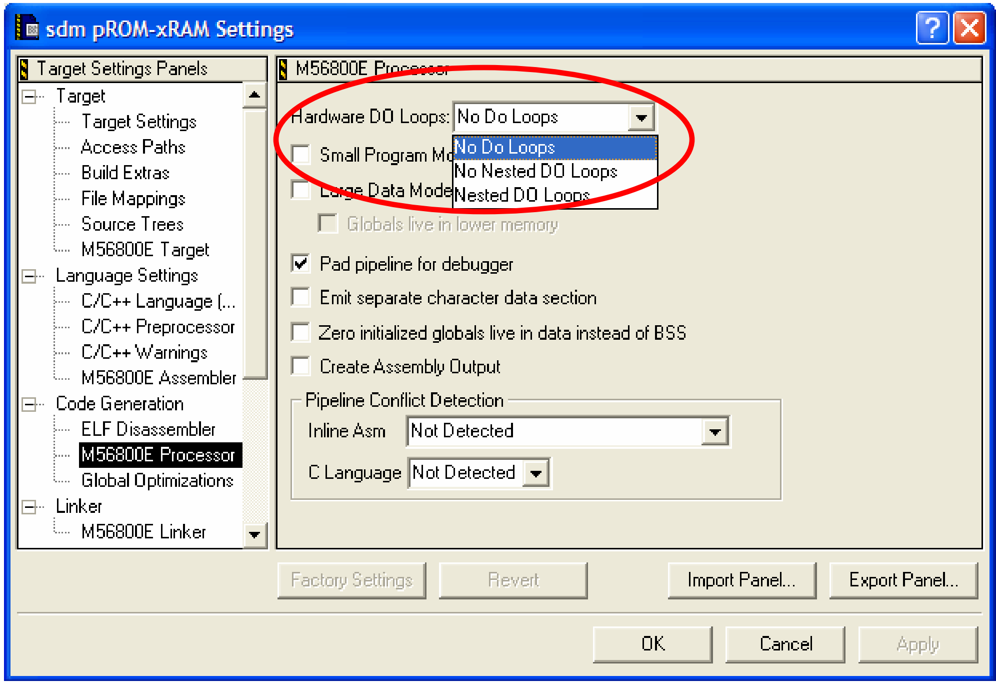
XAB1 - 24 bits
CDBR - 32 bits

2nd Data Access:

XAB2 - 24 bits
XDB2 - 16 bits

Launched by Motorola
freescale semiconductor

Utilizing HW DO Loops



Example C Code

```
for(i=0; I < 25; i++)
{
   y = i + 5;
}
```

Without DO Loops:
226 cycles

```
move.w   #0,B
move.w   B1,A
add.w    #0x000005,A
add.w    #0x000001,B
cmp.w    #0x000019,B
blt      main+0x3 (0xe3)
move.w   A1,X:0x000000
```

With DO Loops:
130 cycles

```
move.w   #0,B
do       #0x000019,0x0000eb
move.w   B1,A
add.w    #0x000005,A
move.w   A1,X:0x000000
add.w    #0x000001,B
```

Efficiency Hint:
Use Do Loops, but be aware of limited resources

56800E CodeWarrior™ Intrinsic Functions

Math support

Category	Function	Category (cont.)	Function (cont.)
Absolute/Negate	abs_s	Multiplication/MAC	mac_r
	negate		msu_r
	L_abs		mult
	L_negate		mult_r
Addition/ Subtraction	add		L_mac
	sub		L_msu
	L_add		L_mult
	L_sub		L_mult_ls
Control	stop	Normalization	ffs_s
	wait		norm_s
	turn_off_conv_rndg		ffs_l
	turn_off_sat		norm_l
	turn_on_conv_rndg	Rounding	round
	turn_on_sat	Shifting	shl
Deposit/Extract	extract_h		shlftNs
	extract_l		shlfts
	L_deposit_h		shr
	L_deposit_l		shr_r
Division	div_s		shrtNs
	div_s4q		L_shl
	div_ls		L_shlftNs
	div_ls4q		L_shlfts
			L_shr
			L_shr_r
			L_shrtNs

Modulo Addressing Support

- __mod_init
- __mod_initint16
- __mod_start
- __mod_access
- __mod_update
- __mod_stop
- __mod_getint16

Efficiency Hint:
Utilize intrinsic functions to target specific instructions

Example: FIR Filter in C with Intrinsic Functions

Code Using L_mult

(4 instr / tap)

```
Acc1 += L_mult(0x1000, Buff[0]);

move.w      X:(SP-3),B
move.w      #4096,Y0
mpy         B1,Y0,B
add         B,A

Acc1 += L_mult(0x2000, Buff[1]);

move.w      X:(SP-2),B
move.w      #8192,Y0
mpy         B1,Y0,B
add         B,A
```

Code Using L_mac

(3 instr / tap)

```
Acc1 = L_mac(Acc1, 0x1000, Buff[0]);

move.w      X:(SP-3),B
move.w      #4096,Y0
mac         B1,Y0,A

Acc1 = L_mac(Acc1, 0x2000, Buff[1]);

move.w      X:(SP-2),B
move.w      #8192,Y0
mac         B1,Y0,A
```

Efficiency Hint:
Use most efficient intrinsic functions

Application Specific Algorithm Libraries

Memory Manager
- Dynamic allocation

Feature Phone Library
- CallerID type 1&2, CallerID Parser, Generic Echo Cancellor

DSP Library
- FIR, IIR, FFT, Auto Correlation, Bit Reversal

Telephony Libraries
- AEC, AGC, Caller ID,
- CAS, CPT, CTG, DTMF
- G165, G168, G711
- G723, G726, G729

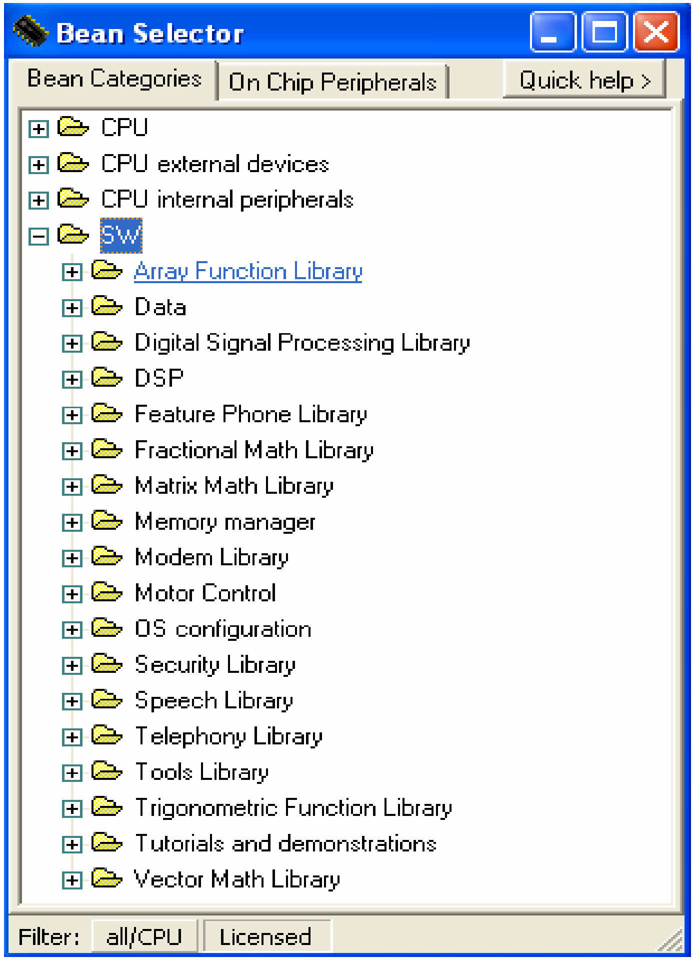
Modem Libraries
- V.8bis, V.21, V.22bis, V.42bis

Security Libraries
- RSA, DES, 3DES,

Motor Control
- BLDC, ACIM, SR motor specific algorithms
- General purpose algorithms

Math Libraries
- Matrix, Fractional, Vector
- Trigonometric

Tools Library
- Cycle Count, FIFO, FileIO, Test



C-callable fixed point FIR from Processor Expert™

11.3.19 *fir* - Finite Impulse Response Filter

Call(s):

```
void dfr16FIR (dfr16_tFirStruct *pFIR, Frac16 *pX, Frac16 *pZ, UInt16 n);
```

Arguments:

Table 11-32. *fir* arguments

pFIR	in	Pointer to a data structure containing private data for the *fir* filter; created by a call to *firCreate*
pX	in	Pointer to the input vector of *n* data elements
pZ	inout	Pointer to the output vector of *n* data elements
n	in	Length of the input and output vectors

Description: Computes a Finite Impulse Response, (FIR), filter for a vector of fractio to any call to *fir*, the FIR filter must be initialized via a call to *firCreate*; the FIR f passed to that *firCreate* call. The function *fir* uses the private data structure establis maintain the past history of data elements required by the FIR filter computation.

Efficiency Hint:
Use Processor Expert Libraries

Processor Expert Overview

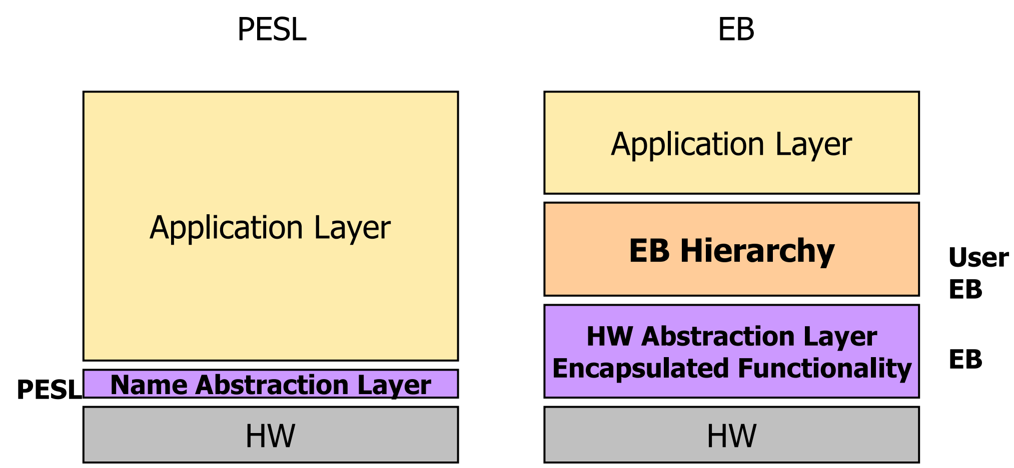
Efficiency Hint:
PE provides efficiency, portability, and rapid development

Processor Expert™

- Supports rapid application development
- Enables component oriented programming
- Provides expert advice if necessary
- Delivers instant functionality of generated code
- Provides tested ready-to-use code

Key Abstraction Technologies

- **PESL**
  - Processor Expert System Library
  - Peripheral oriented
- **EB – an abstraction provider**
  - Embedded Beans
  - Functionality oriented
  - Real *components* for building of an application

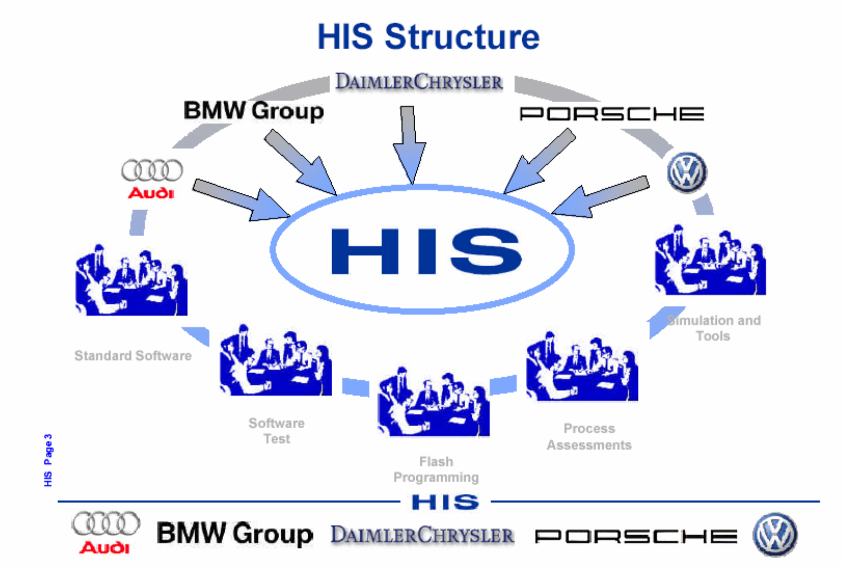


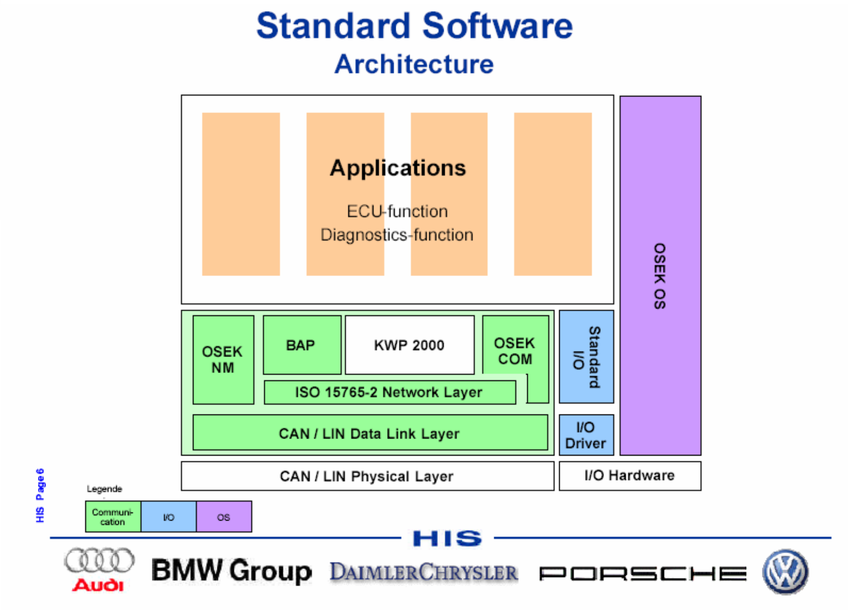
How Features of PE are Achieved

- Developed by experienced programmers of embedded systems
- Expert knowledge system is working on the background of PE and checks all the settings
- Provides context help and access to CPU/MCU vendor documentation
- All EB delivered by UNIS are tested according to ISO testing procedures (UNIS is ISO certified company)

Launched by Motorola freescale semiconductor

Processor Expert and Safety Critical Applications



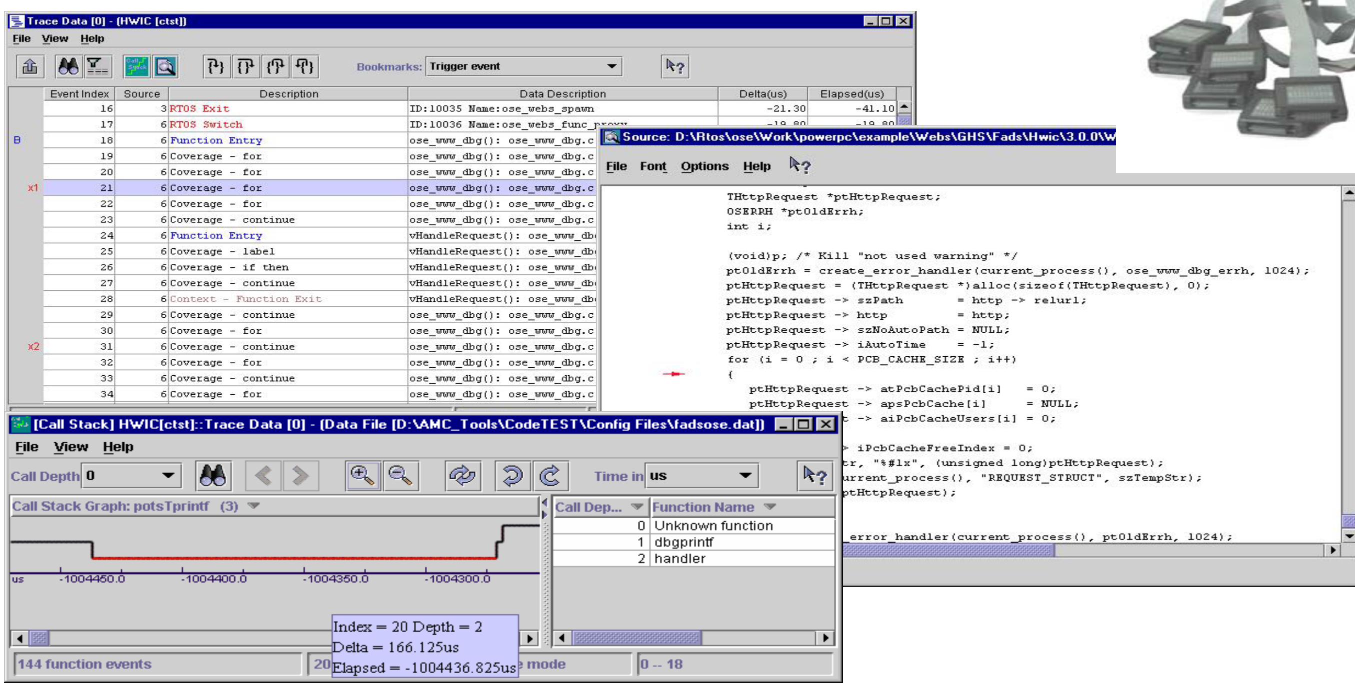


Additional Efficiency and Safety Considerations

- Use Lint tools
  - CodeWarrior provides plug-in for CP-Lint
  - Lint greatly reduces errors and improves efficiency

- Use MISRA C tools
  - Subset ANSI-C standard with focus on safety
  - Usually provided as part of Lint tools

Profile Your Code for Better Optimizations



Launched by Motorola
freescale semiconductor

In Summary

- Use CodeWarrior optimizations
- Tune your code with custom settings
- Help compiler to help you
- Use intrinsic functions
- Use Processor Expert production ready code
- Utilize profiler tools
- Utilize 3rd party tools