Embedded C
for
High Performance DSP Programming
with the
CoSy Compiler Development System

Marcel Beemster/Yoichi Sugiyama
ACE Associated Compiler Experts/Japan Novel Corporation
contact: yo_sugi@jnovel.co.jp
ACE Associated Compiler Experts

- Home of CoSy
  the compiler development system
  - Compiler Generator System
  - Modular design
  - Configurable
  - Retargetable
  - Robust
  - Extensible
  - High Quality
  - Highly optimising
Japan Novel and CoSy

• Japan Novel is an exclusive agent in Japan for ACE
• Japan Novel provides a products and services to improve the quality of today’s complex embedded software
  – Compiler evaluation services
  – Automated test&evaluation system - Quality Commander
  – C/C++ comformance test suites - PlumHall’s products
• Compiler evaluation services provide a thorough testing of C/C++, Embedded - C, DSP-C compilers
• With it’s high reliability, CoSy compiler development system contributes to the embedded system development in Japan
Programming
DSPs in C has many advantages

• Code portability allows switching to another DSP >>> *No CPU lock-in*

• Software engineering quality is much better than assembly >>> *Faster time to market*

• No dependency on specialist assembly programmers >>> *Lower Cost/Time to market*

...But still DSP assembly programming is used very often because of

Performance Requirements
DSP Processors are Adapted to the Application

APP  →  DSP  →  C

But C is not
RISC vs. DSP Architecture

RISC

REG → ALU → LD/ST → MEM

DSP

X MEM → REG → Y MEM → MAC +FP+Sat → ACCU

16 → LD/ST → MEM
Enter *Embedded C*
Embedded C

- Based on the DSP-C extension, an industry standard by ACE
- Unlocks the performance potential of embedded processor features to the high level language programmer
- Promotes portability
- Adds fixed-point, saturation, named address spaces and I/O access to C
Status of Embedded C

- Defined in a technical report by ISO (TR18037)
- Ratified in February 2004
- www.open-std.org/jtc1/sc22/wg14/
Fixed-Point Type: _Fract

- Range of \([-1.0, 1.0]\)
- Has a fractional part only, no integer part
- Efficiently implemented on top of two’s complement arithmetic
- Variants for short and long
- Accuracy defined by implementation/architecture
Accum Type

- Also has an integer part, for example a range of \([-256.0, 256.0]\)
- Designed as an intermediate value for fixed-point arithmetic
- Not meant to be a storage type
- Also short and long variants with accuracy matching the \_Fract variants
Saturation: _Sat Qualifier

- Makes computations saturate:
  \[-0.75r + -0.75r = -1.0r\]
- No storage implications
- Needed because signal processing applications often operate at the boundary of the range to get best S/N
Named Address Spaces

- No predefined keywords
- Examples:
  
  ```
  X int a[25] ;
  X int * Y p ;
  ```

- Restrict access to a specific part of the address space

- Unrestricted (general) pointers can access all address spaces
Example

```
X fract coeff[N] = { 0.7r, ... };

fract fir( Y fract *inp ) {
    int i;
    accum sum = 0.0k;
    for( i = 0 ; i < N ; i++ ) {
        sum += coeff[i] * (accum)*inp++ ;
    }
    return (sat fract)sum ;
}
```
Performance
Comparison Based on DSP-C

• Using the CoSy DSP-C compiler for internal Ericsson telecom processor (16-bit, VLIW, dual MAC, dual 32-bit ld/st)
• Using the CoSy DSP-C compiler for the NEC µPD77016 processor (16-bit standard DSP)
• Compare:
  – Basic operations
  – NEC applications
  – MiBench loop

Thanks to Ericsson, NEC and University of Michigan
Basic Operations

- Using the Ericsson compiler, highest performance settings

<table>
<thead>
<tr>
<th></th>
<th>ISO C Cycles/Size</th>
<th>CoSy DSP-C Cycles/Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saturation</td>
<td>10/46</td>
<td>0*/0*</td>
</tr>
<tr>
<td>FIR-filter</td>
<td>2/74</td>
<td>1*/26</td>
</tr>
<tr>
<td>Array Copy</td>
<td>2/12</td>
<td>1*/12</td>
</tr>
</tbody>
</table>

Cycles: in inner loop, except for saturation
Results marked with * are optimal
NEC Applications

Using the NEC µPD77016 CoSy compiler
Reporting total clock cycles for application

<table>
<thead>
<tr>
<th>Control</th>
<th>ISO C</th>
<th>CoSy DSP C</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>3946</td>
<td>3890</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>DSP 1</td>
<td>5144</td>
<td>550</td>
<td>9.4</td>
</tr>
<tr>
<td>DSP 2</td>
<td>168546</td>
<td>48064</td>
<td>3.5</td>
</tr>
<tr>
<td>DSP 3</td>
<td>2822</td>
<td>349</td>
<td>8.1</td>
</tr>
</tbody>
</table>

Near 10 times improvement! How?
MiBench Original Code

• Taking procedure from telecom/gsm: Short_term_analysis_filtering

• Inner loop:

```c
for (i = 0; i < 8; i++) { /* YYY */
    ui    = u[i];
    rpi   = rp[i];
    u[i]  = sav;
    zzz   = GSM_MULT_R( rpi, di  );
    sav   = GSM_ADD   ( ui,  zzz );
    zzz   = GSM_MULT_R( rpi, ui  );
    di    = GSM_ADD   ( di,  zzz );
}
```
MiBench DSP-C Code + Accum

- Rewritten in DSP-C, with accumulator use

```c
for (i = 0; i < 8; i++) { /* YYY */
    ui   = u[i];
    rpi  = rp[i];
    u[i] = sav;
    sav  = ((long accum)ui) + rpi * di ;
    di   = ((long accum)di) + rpi * ui ;
}
```
MiBench Loop

- Using the Ericsson compiler, highest performance settings, report cycles in inner loop

<table>
<thead>
<tr>
<th></th>
<th>ISO C Cyc./Size</th>
<th>CoSy DSP-C Cyc./Size</th>
<th>DSP-C+ accum Cyc./Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_t_a_f</td>
<td>32/228</td>
<td>4/112</td>
<td>3*/112</td>
</tr>
</tbody>
</table>

Results marked with * are optimal
MiBench Loop Results Explained

- Original code uses *macros* to write maintainable code, but results in unnecessary saturation in inner loop
- Fixed point emulation code in plain ISO C requires more registers; with a limited register file this results in *spill-code*
- Cumulative effect explains factor 10 overhead in the inner loop
Embedded C in CoSy

- Full front-end support for Embedded C
- Configurable data type sizes
- Fully integrated with Compiler IR
- Optimized by CoSy optimization modules
- Full emulation and support library included
- Example emulation compiler included
- Compiler generation for Windows, Linux, Solaris
- DSP-C available in CoSy since 1998
- Embedded C available in CoSy 2005
The Embedded C world

• C compilers with DSP extensions have been developed for: Philips REAL, Adelante Saturn, NEC µPD77111/210, TI TMS320C54x, Analog Devices SHARC, MMDSP+, META RISC/DSP and many more …

• Used and supported by: ACE, Ericsson, AbsInt, NullStone, Mentor Graphics XRAY, Japan Novel, …
Benefits of Embedded C

- Enables high level language code for embedded processors that runs efficiently
- Standardizes the notation of common performance features in DSP architectures
- Standardizes I/O hardware access
- Thereby considerably improving the software engineering and portability of embedded applications
- Economic advantages: time to market, flexibility
Use CoSy with Embedded C!

marcel@ace.nl/yo_sugi@jnovel.co.jp
www.embedded-c.org
Implicit Promotions

_Fract f = (_Fract)(0.1r * 2.5) ;

- **Rank order:** int, _Fract, _Accum, float
- **New concept (for C), mixed type arithmetic:**
  \[ f = 3 \times 0.1r \]

Only for mixing int and fixed point
unsigned Fixed-Point Variants

- **Range:** [0.0, 1.0>
- Useful for image processing
- Not implemented by all DSP processors (saturation)
Features That Did Not Make It

- Circular buffers
- `__Modwrap` qualifier for modulo fixed-point arithmetic
- Complex fixed-point type
- BCD data types
Performance vs. Portability of Embedded C (1)

- Accuracy of fixed-point types is not guaranteed, while most DSP algorithms rely on a particular accuracy
- Memory qualifier keywords are not predefined by Embedded C
- Similar to existing practice in C

So, 100% portability of Embedded C programs is not guaranteed
Portability of Embedded C (2)

• The Embedded C implementation must match the target processor
• Example: A 24-bit accuracy DSP application can not run efficiently on a 16-bit DSP processor
• Embedded C implementation allows for adaptation specific to the architecture
## NEC Code Size Comparison, Bytes

<table>
<thead>
<tr>
<th></th>
<th>ISO C</th>
<th>DSP-C</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>442</td>
<td>414</td>
<td>1.1</td>
</tr>
<tr>
<td>DSP 1</td>
<td>350</td>
<td>90</td>
<td>3.9</td>
</tr>
<tr>
<td>DSP 2</td>
<td>2152</td>
<td>1155</td>
<td>1.9</td>
</tr>
<tr>
<td>DSP 3</td>
<td>3807</td>
<td>2781</td>
<td>1.4</td>
</tr>
</tbody>
</table>
I/O Hardware Addressing

- Aims to allow for portable device driver source code
- Based on a three level model

- Portable driver source code
- I/O register definitions
- Vendor IOHW implementation

- Still allows for minimal overhead implementation