Using the GNU Toolchain to Build, Debug, and Simulate ARM Applications

CodeSourcery, Inc.
Mark Mitchell
mark@codesourcery.com
Outline

- ARM Toolchain
- Using NEON Vector Instructions
- Thumb-2 Code Size Optimization
- Improved “Bare Metal” Support
- Demonstration
## Target Matrix

<table>
<thead>
<tr>
<th>Operating System</th>
<th>C Library</th>
<th>Run-Time Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>EABI</td>
<td>Newlib</td>
<td>- ARMV5TE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ARMV5TE (Thumb)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ARMV6M (Thumb)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ARMV7 (Thumb-2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Feroceon</td>
</tr>
<tr>
<td>uClinux</td>
<td>uClIBC</td>
<td>- ARMV5TE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ARMV6M (Thumb)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ARMV7 (Thumb-2)</td>
</tr>
<tr>
<td>GNU/Linux</td>
<td>GLIBC</td>
<td>- ARMV4T</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ARMV5TE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ARMV7 (Thumb-2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Feroceon</td>
</tr>
<tr>
<td>SymbianOS</td>
<td></td>
<td>- ARMV5TE</td>
</tr>
</tbody>
</table>
## Key Facts

<table>
<thead>
<tr>
<th>Components</th>
<th>Editions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC 4.2.1</td>
<td>Lite</td>
</tr>
<tr>
<td>Binutils 2.18</td>
<td>Command-Line Tools</td>
</tr>
<tr>
<td>GDB 6.6</td>
<td>Semi-Annual releases</td>
</tr>
<tr>
<td>GLIBC 2.5</td>
<td>Sponsored by ARM, Ltd.</td>
</tr>
<tr>
<td>uClibc</td>
<td></td>
</tr>
<tr>
<td>Newlib</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Personal</td>
</tr>
<tr>
<td></td>
<td>Includes IDE</td>
</tr>
<tr>
<td></td>
<td>Professional</td>
</tr>
<tr>
<td></td>
<td>Includes unlimited support</td>
</tr>
</tbody>
</table>

**Hosts**

- IA32 Windows
- IA32 GNU/Linux

---

**CodeSourcery**

---

**ARM Developers’ Conference & Design Pavilion 2007**
New Features

- GCC 4.2.1
  - Significant overall improvements to compiler infrastructure
- Optimization for latest ARM cores
  - Cortex-A8
  - Cortex-R4
  - Cortex-M1 (FPGA)
- uClinux Support
  - Just add kernel and stir
- Thumb-2
  - GLIBC & uClibc precompiled for Thumb-2
  - Significant code-size improvements
Outline

- ARM Toolchain
- Thumb-2 Code Size Optimization
- Using NEON Vector Instructions
- Improved “Bare Metal” Support
- Demonstration
### Code Size: Prologues

- **Input:**
  ```c
  void f(void);
  int main () {
    f ();
    return 0;
  }
  ```

- **Before:**
  ```asm
  push {lr}
  sub sp, sp, #4
  bl f
  movs r0, #0
  add sp, sp, #4
  pop {pc}
  ```

- **After:**
  ```asm
  push {r4, lr}
  bl f
  movs r0, #0
  pop {r4, pc}
  ```

- **Input:**
  ```c
  void f(int *);
  int main() {
    int tmp[2];
    f(tmp);
    return 0;
  }
  ```

- **Before:**
  ```asm
  push {lr}
  sub sp, sp, #12
  mov r0, sp
  bl f
  movs r0, #0
  add sp, sp, #12
  pop {pc}
  ```

- **After:**
  ```asm
  push {r0, r1, r2, lr}
  mov r0, sp
  bl f
  movs r0, #0
  pop {r1, r2, r3, pc}
  ```

---

**ARM, Thumb, Thumb-2**

**ARM Developers’ Conference & Design Pavilion 2007**
Code Size: Branches vs. Conditional Execution

void foo(int a, int b) {
    if (a < 5 || b > 42)
        bar();
}

Conditional Execution

cmp r1, #42
ite le
movle r1, #0
movgt r1, #1
cmp r0, #4
it le
orrle r1, r1, #1
cbz r1, .L4
bl bar
.L4:

Branches

cmp r0, #4
ble .L2
cmp r1, #42
ble .L5
.L2:
    bl bar
.L5:
## Code Size: Miscellany

### Use High Registers
- In Thumb-1, usually a loss
- In Thumb-2, a win
  - … if using a 32-bit instruction anyhow

### Do Not Align Strings
- Aligned strings permit single-word loads …
- … but waste padding bytes.
Code Size: Results

Methodology
- Dynamically linked
- Size does not include GLIBC
- Measured with size

4.6% Improvement
Outline

- ARM Toolchain
- Thumb-2 Code Size Optimization
- Using NEON Vector Instructions
- Improved “Bare Metal” Support
- Demonstration
NEON Vectors: Introduction

Overview

- NEON CPUs have SIMD operations
  - Load multiple values at once
  - Add multiple values at once
  - Store multiple values at once
  - ...

Programming Approaches

- Assembly code
- Machine-dependent vector intrinsics
- Machine-independent vector extensions
Example

C Code

```c
void f (int N,
    float a[], float b[],
    float c[], float d[]){
  for (int i = 0; i < N; i++)
    a[i] = b[i] + c[i] + d[i];
}
```

Unvectorized Code

```
.L4:
    add     r5, ip, r2
    flds    s15, [r5, #0]
    add     r5, ip, r3
    flds    s14, [r5, #0]
    fadds   s15, s15, s14
    add     r5, ip, r4
    flds    s13, [r5, #0]
    add     lr, lr, #1
    cmp     lr, r0
    fadds   s15, s15, s13
    fmrs    r5, s15
    str     r5, [ip, r1]
    add     ip, ip, #4
    bne     .L4
```

14N Instructions

14N x N Iterations

ARM Developers’ Conference & Design Pavilion 2007
Vectorization I: Assembly Code

Assembly Code

.L4:

add r3, r4, ip
add r2, r1, ip
vldmia r3, {d6-d7}
vldmia r2, {d2-d3}
add r3, r6, ip
add lr, lr, #1
vldmia r3, {d4-d5}
vadd.f32 q3, q3, q1
cmp lr, r0
add r3, r5, ip
vadd.f32 q3, q3, q2
vstmia r3, {d6-d7}
add ip, ip, #16
bne .L4

14 Instructions
x N/4 Iterations
4x Faster

Loads
Adds
Store

ARM Developers’ Conference & Design Pavilion 2007
#include <arm_neon.h>

void f(int N,
       float a[], float b[], float c[], float d[]) {
    for (int i = 0; i < N; i += 4)
    *(float32x4_t *) (a + i) =
        vaddq_f32 (vaddq_f32
                      *(float32x4_t *) (b + i),
                      *(float32x4_t *) (c + i)),
        *(float32x4_t *) (d + i));
}
Vectorization III: Generic Vectors

typedef __attribute__((vector_size (16))) float quad;

void f(int N,  
    float a[], float b[], float c[], float d[]) {  
    for (int i = 0; i < N / 4; ++i)  
        ((quad*) a)[i] =  
            ((quad*) b)[i] + ((quad*) c)[i]  
            + ((quad*) d[i]);
}
## Vectorization: Compare and Contrast

<table>
<thead>
<tr>
<th>Assembly</th>
<th>Generic Vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Total control of instruction sequence</td>
<td>+ Simple syntax</td>
</tr>
<tr>
<td>+ Maximum performance</td>
<td>+ Portable to all CPUs</td>
</tr>
<tr>
<td>+ Hard</td>
<td>+ Vectorized on CPUs with vector support</td>
</tr>
<tr>
<td>- Not portable</td>
<td>+ Compiler can schedule instructions</td>
</tr>
<tr>
<td>- Performance varies with CPU</td>
<td>- Least control of instruction sequence</td>
</tr>
</tbody>
</table>

### Intrinsics

| + Good control of instruction sequence         | + Easier than assembly                                |
| + Easier than assembly                         | + Compiler can schedule instructions                  |
| + Compiler can schedule instructions          | + Portable to RealView                                |
| + Portable to RealView                        | - Not portable to other CPUs                          |
| - Not portable to other CPUs                  |                                                       |
Outline

- ARM Toolchain
- Thumb-2 Code Size Optimization
- Using NEON Vector Instructions
- Improved “Bare Metal” Support
- Demonstration
CS³

- Uniform – but replaceable – startup routines on all targets
  - Hardware initialization
  - Assembly initialization
  - C initialization
- Simplified interrupt handling
  - Declare ISRs with interrupt attribute
  - Functions automatically inserted in ISR table
- Reduced footprint
  - Avoid requiring run-time library support unless absolutely required

---

**Hardware**

__cs3_reset_sys

- Memory controller
- Peripherals

**Assembly**

__start

- Initialize stack

**C**

__cs3_start_c

- Clear BSS, initialize data
- Run constructors
- Initialize heap

**main**
CS\textsuperscript{3} Interrupts

Simple Set Up

- ISRs have documented names
- Default implementations do nothing, halt CPU, etc.
- Provide your own implementation for custom behavior

Example

```c
void __cs3_isr_systick()
    __attribute__((interrupt))
{
    /* Do something. */
}
```

Retaining Control

- `__cs3_interrupt_vector` is table of pointers to ISRs
  - Provide your own `__cs3_interrupt_vector` to override completely

- Linker script places `__cs3_interrupt_vector`
  - Provide your own linker script to control placement
CS\textsuperscript{3} Footprint: Avoiding Cost of Exit

### Classic Approach

- **Startup code:**
  ```c
  /* Run constructors. */
  _init ();
  /* Destructors. */
  atexit (_fini);
  /* Main. */
  exit (main());
  ```
- **atexit**
  - Calls `malloc`

### CS\textsuperscript{3} Approach

- **Startup code:**
  ```c
  _init ();
  main();
  ```
- **Module containing `exit` contains constructor that does:**
  ```c
  atexit (_fini)
  ```
  - Included only if application calls `exit`
- **atexit**
  - Contains static buffer for first few calls
  - Uses `malloc` only if otherwise used in application
CS³ Footprint: Avoiding Cost of Exit

**Classic Approach**

- **Startup code:**
  
  ```c
  /* Run constructors. */
  _init();
  /* Destructors. */
  atexit(_fini);
  /* Main. */
  exit(main());
  ```

- atexit
  - **Calls** malloc

**CS³ Approach**

- **Startup code:**
  
  ```c
  _init();
  main();
  ```

- Module containing `exit` contains constructor that does:
  ```c
  atexit(_fini)
  ```
  - Included only if application calls `exit`

- atexit
  - Contains static buffer for first few calls
  - Uses `malloc` only if otherwise used in application

---

**Graphic Data: Empty C Program**

- **2007Q1**
  - Empty text
  - Empty data

- **2007Q3**
  - Empty text
  - Empty data

---

**ARM Developers’ Conference & Design Pavilion 2007**
Footprint: Destructors & Exceptions

- **Example:**
  ```
  struct S {
    virtual ~S();
  };
  S s;
  ```
  Virtual destructors call operator delete to support:
  ```
  struct Derived : public S { ... };
  S* p = new Derived;
  delete p;
  ```

- **Implementation:**
  ```
  void operator delete(void *p) {
    free (p);
  }
  ```

- **Problem:**
  - Compiling with `-fno-exceptions` still pulls in exception-handling!

- **Tell compiler that most C library functions cannot throw exceptions**
  - Place explicit `throw ()` declarations in headers
  - Allows compiler to avoid generating code to support exceptions

- **Refactor runtime library so that unrelated functions are in separate object files**
  - Linker always pulls in an entire object file

- **Teach compiler not to generate any ARM EH data for functions not using exceptions**
  - EH data references personality routines

---

ARM Developers’ Conference & Design Pavilion 2007
Footprint: Destructors & Exceptions

- **Example:**
  
  ```c++
  struct S {
    virtual ~S();
  };
  S s;
  ```

- **Virtual destructors call operator delete to support:**
  
  ```c++
  struct Derived : public S { ... };
  S* p = new Derived;
  delete p;
  ```

- **Implementation:**
  
  ```c++
  void operator delete(void *p) {
    free (p);
  }
  ```

- **Problem:**
  - Compiling with `-fno-exceptions` still pulls in exception-handling!

- **Tell compiler that most C library functions cannot throw exceptions**
  - Place explicit `throw()` declarations in headers
  - Allows compiler to avoid generating code to support exceptions

- **Refactor runtime library so that unrelated functions are in separate object files**
  - Linker always pulls in an entire object file

- **Teach compiler not to generate any ARM EH data for functions not using exceptions**
  - EH data references personality routines

---

**ARM Developers’ Conference & Design Pavilion 2007**
System Calls

## Semihosting

- Use host system to implement system calls
  - System calls implemented on target as trap instructions
  - Debugger recognizes trap instructions and calls corresponding functions on host
- Useful for debugging
  - Print debugging messages
  - Write to log file on host system
- Useful for prototyping
  - Write computational kernel on target
  - Use semihosting to transfer data to/from host

## Real Implementations

- Define appropriate functions, e.g.:
  - _write (int fd, char *ptr, int len)
- User definitions automatically override semihosting implementations
- Other functions use system calls
  - For example, printf calls _write
Outline

- ARM Toolchain
- Thumb-2 Code Size Optimization
- Using NEON Vector Instructions
- Improved “Bare Metal” Support
- Demonstration
### QEMU

<table>
<thead>
<tr>
<th>Overview</th>
<th>CodeSourcery Enhancements</th>
</tr>
</thead>
<tbody>
<tr>
<td>System emulator</td>
<td>ARMV6 and ARMV7</td>
</tr>
<tr>
<td>- CPU emulation</td>
<td>- Including Thumb-2</td>
</tr>
<tr>
<td>- Peripheral emulation</td>
<td>- Numerous bug fixes</td>
</tr>
<tr>
<td>- Can even boot GNU/Linux</td>
<td>- Integrated with Eclipse</td>
</tr>
<tr>
<td>Dynamic translation</td>
<td></td>
</tr>
<tr>
<td>- Convert target instructions to host instructions</td>
<td></td>
</tr>
<tr>
<td>- Runs fast …</td>
<td></td>
</tr>
<tr>
<td>- … but not cycle-accurate.</td>
<td></td>
</tr>
</tbody>
</table>

ARM Developers’ Conference & Design Pavilion 2007
ICE/JTAG Debugging

Via Sprite

- ARM RDI
  - ARMulator, MultiICE
- Cortex-M3 USB
  - Luminary Micro Stellaris
- Actel FlashPRO
  - Cortex-M1
- Keil ULINK2
  - Most ARM processors

Via GDB Protocol

- QEMU
- ARM RealView ICE
- Abatron BDI-2000
- ...
References

- Sourcery G++ Lite Edition:
  - [http://www.codesourcery.com/gnu_toolchains/arm](http://www.codesourcery.com/gnu_toolchains/arm)
- Sourcery G++:
  - [http://www.codesourcery.com/sgpp](http://www.codesourcery.com/sgpp)
- GCC Home Page:
  - [http://gcc.gnu.org](http://gcc.gnu.org)