Navigating the ABI for the ARM Architecture

Peter Smith
Agenda

● Introduction to the ABI, and its history
● The structure of the ABI and how it fits together with other standards
● Expectations of compatibility
● The 64-bit ABI and how it differs from the 32-bit
My background with the ABI

● Worked in the proprietary ARM Compiler toolchain from 2000 - 2016
  ○ ADS 1.0, RVCT, ARM Compiler 5, ARM Compiler 6
  ○ Main focus is on embedded systems
  ○ Limited intersection with ARM Linux

● Specializing in non-compiler tools, such as the linker, assembler and object processing tools
  ○ Armlink, armasm, fromelf

● Involved in implementing, rather than specifying the ABI
Definitions

- **Application Programming Interface API**
  - Interface at the source code level

- **Application Binary Interface ABI**
  - Interface between executables, shared libraries and operating systems

- **Embedded Application Binary Interface**
  - Interface between relocatable objects

- **Platform**
  - A software platform running a sophisticated OS that can run applications
  - Examples include Linux, *BSD, Symbian

- **Bare Metal**
  - An embedded application running without an OS, or at most an RTOS

- **Quality of Implementation Q-o-I**
  - Additional functionality over and above the minimum required for conformance, or permitted collusion between components made by same vendor
What is the ABI?

- What does an ABI define and why should I care?
- ARM in early 2000s and the influence on the ABI
- Motivations and principles behind the ABI
What does an ABI define?

- Procedure calling standard
- Sizes, layout and alignment of types
- C++ name mangling
- Exception handling
- Object file and library file format
- Debug information format
- Thread local storage?
- Compiler Helper Functions and runtime library?
- Dynamic Linking?
- System call interface?

Mandatory for a C/C++ ABI

Depends on scope of ABI
Why should I care about the ABI?

- Majority of the ABI hidden from you by development tools
- Necessary if you are implementing development tools
  - Not just large tools like compiler, linker, assembler and debugger
  - Custom object file processors
- Developing or distributing cross platform binaries
  - What level of interoperability can you expect between toolchains?
- Understanding the different calling conventions available
ARM in early 2000s

- The ABI for the ARM Architecture was released on 30th October 2003
- ARM11 family (v6) released to partners late 2002, ARM7 and ARM9 popular
- ARM’s largest market by far was Mobile with custom ASICs
- Constrained devices, with expensive flash prices
- Software mostly embedded, device specific and not upgradeable
  - SW Could be targeted at, and optimized for a specific device
- Fragmented market in tools and operating systems
  - Upwards of 20 available toolchains
- Early signs of commercial interest in Linux
  - Consumer Electronics Linux Forum founded in June 2003
ARM consumer products from early 2000s
ABI motivation and goals

- IA64 C++ ABI recently available on both GCC and EDG
- Linux on ARM showing signs of gathering momentum
- RTOS vendors needing to ship a different binary package for every toolchain
- Enable independent development tool chains to support inter-operation between portable binary packages
  - Use any application library in any ARM environment
  - Use any object producer with any ARM-based platform
ABI for the ARM Architecture Principles

- Must be read in conjunction with other documents such as the ARM ARM
- ABI builds upon industry standards such as ELF and Dwarf
- Platform owners expected to define their own ABI when required
- Conformance follows the “as if” rule, if a conforming external observer cannot detect non-conformance there is no need to conform
- Tools vendors must be free to differentiate on Q-o-I even at ‘extern’ interfaces
  - Components have an exported interface with ‘contractual’ guarantees of conformance
  - An exported interface is ‘extern’ but not all ‘extern’ interfaces are exported
- Multiple options when consensus not reachable
ABI Structure

- The ABI documents and how they relate to generic and platform standards
### ABI for the ARM Architecture structure

Instead of a System V ABI ARM Processor Supplement, the ABI is split up into several documents:

- ABI documents in gray
- Generic industry documents in cyan
- BPABI is the interface to platforms

<table>
<thead>
<tr>
<th>AAPCS Procedure Call Standard</th>
<th>CPPABI C++ ABI</th>
<th>AAELF ARM ELF</th>
<th>AADWARF ARM Dwarf</th>
<th>BPABI Base Platform ABI</th>
</tr>
</thead>
<tbody>
<tr>
<td>EHABI C++ exceptions</td>
<td>IA64 C++ ABI</td>
<td>TIS ELF</td>
<td>Dwarf 3.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**BPABI**

- **AAPCS**:
  - Procedure Call Standard
- **CPPABI**:
  - C++ ABI
- **AAELF**:
  - ARM ELF
- **AADWARF**:
  - ARM Dwarf
- **CLIBABI**:
  - C-lib
- **RTABI**:
  - run-time
- **AR format**
ABI for the ARM Architecture structure

4 types of platform are supported by the base platform ABI, with an example of each one.

<table>
<thead>
<tr>
<th>RTOS</th>
<th>Palm OS</th>
<th>Symbian OS</th>
<th>Linux, *BSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bare metal</td>
<td>DLL-Like</td>
<td>DLL-Like</td>
<td>SVr4-like</td>
</tr>
<tr>
<td>No dynamic loading,</td>
<td>Multiple processes</td>
<td>DLL mapped at same</td>
<td>DSO mapped at a</td>
</tr>
<tr>
<td>RTOS statically</td>
<td>thread DLL in one</td>
<td>address in multiple</td>
<td>different address in</td>
</tr>
<tr>
<td>linked</td>
<td>address space</td>
<td>address space</td>
<td>each process</td>
</tr>
</tbody>
</table>

| Single Address Space | Multiple Address space   |

**BPABI** Base Platform ABI
Layering of Standards, ELF example

**Generic**
- Concepts common to all uses of ELF
- Extension points for processor and platform (OS)

Often defined by the SystemV ABI document for the architecture

**Processor Specific**
- Concepts specific to ARM’s interpretation of ELF
- Relocation directives
- Flags and types in the processor specified range

**Platform Specific**
- Concepts specific to a platform’s interpretation of ELF
- Dynamic Relocation directives, TLS Model
- Flags and types in the OS specified range
ABI Documents

- The core of the ABI is related to relocatable object compatibility
  - ARM Procedure call standard
    - Includes base standard and variants such as VFP
  - ELF for the ARM Architecture, ARM processor specific supplement psABI
    - Includes relocations for all platforms
  - Dwarf for the ARM Architecture
    - Mapping of register numbers
  - Exception handling ABI for the ARM Architecture
    - Table based but specific to ARM
  - C++ ABI for the ARM Architecture
    - Deviations from the Itanium C++ ABI standard
  - Run time ABI for the ARM Architecture
    - Compiler helper functions and built-ins
  - C-library ABI for the ARM Architecture
    - Compatibility model for C-library interoperation
  - Errata and Addenda to the ARM Architecture (TLS and build attributes)
Relocatable Object Compatibility

- What can I expect for C and C++ objects produced by different toolchains?
- How to check binary object properties?
C-Library compatibility model

- `<Header1.h>`
  - Compiler1 and system headers from same toolchain
  - Compiler1
  - Object, inc private helpers
  - Linker 1
  - Executable
  - Run-time lib
  - Functions outside AEABI
  - AEABI Functions
  - Functions outside AEABI
  - AEABI

- "Header1.h"
  - Source.c
  - Compiler 2
  - Object, inc private helpers
  - Linker 2
  - Executable
  - platform standards
  - Run-time lib
  - AEABI Functions

- `<Header2.h>`
  - Compiler2 and system headers from same toolchain
  - Compiler2
  - Object, inc private helpers
  - Linker 2
  - Executable
  - AEABI Functions
Q-o-I Managing compatibility between objects

- Build Attributes capture the intention to use short or int enums, a potentially incompatible choice if each side of the API chooses differently.
- Linker can check for clashes in the attributes

```c
typedef enum {val1 = 0, val2 = 1} Vals;
extern void api_function(Vals* v);
static void internal_function(Vals* v) { ... }
```

- `-fno-short-enums`
- `-fshort-enums`

Tag_ABI_enum_size: int

warning: t2.o uses variable-size enums yet the output is to use 32-bit enums; use of enum values across objects may fail

Tag_ABI_enum_size: short
Base Platform ABI for the ARM Architecture

Bare-Metal
- ELF objects
- Linker
- ELF exe
- Flash/ROM burner
- Outside scope of ABI

DLL-Like
- ELF objects
- Linker
- ELF BPABI exe or dll
- Post Linker
- ELF BPABI exe or dll

SVr4-Like
- ELF objects
- Linker
- SVr4 ELF exe or .so
- For SVr4 post-linker not needed

For SVr4 post-linker not needed
Navigating the ARM ABI

● How to find the information you need in the set of documents making up the ABI?
How to navigate the ABI in general

- The “Application Binary Interface for the ARM Architecture The Base Standard”
  - Referred to as Introduction in infocenter.arm.com
- ABI is available under ARM Software Development Tools
How to navigate the ABI as a programmer

- In an ideal world you won’t have to, goal of the ABI is that most things should just work
- When things don’t just work the ABI can be a good source of information to help track down the problem
  - Diagnosing compatibility problems between tools
  - Understanding error messages from low-level tools such as assemblers and linkers
- ELF for the ARM Architecture [AAELF] is usually the first port of call for linker and non-syntax assembler error messages
- Addenda to and errata in the ABI for the ARM Architecture [ADDENDA] contains the Build Attributes values that may explain link time compatibility messages.
- [AAPCS] For how to call a C/C++ function from Assembler
Concluding thoughts

• Looking back at the 32-bit ABI
• Influences on the 64-bit ABI
• References
Looking back on the 32-bit ABI

- Recall that back in the early 2000s the hardware and software landscape looked very different
- Large amount of consolidation has occurred at all levels
- Much more sharing done at the source code level
- The problem of the RTOS having to ship 20 different binaries for 20 different toolchains is much reduced
The 64-bit ABI for the ARM Architecture

- A chance to start from scratch without baggage of existing implementations
  - AAELF, AADWARF, AAPCS and CPPABI minimum necessary building blocks for SW tools to aim for

- AArch64 only available in the A profile
  - Reusing existing industry standards with minimal changes acceptable.

- Primary use case is to run rich software platforms such as Linux and Android
  - Unit of sharing is either shared-libraries with C-like exported interfaces or source code

- Number of developers writing applications for a platform vastly outnumber OS and bare-metal developers
  - Platform specific standards and interfaces more important than defining a base platform ABI with post-linking
Conclusion

- The 32-bit ABI for the ARM Architecture is an embedded ABI
  - Interoperability at the binary package level
- Platforms may build their own standards on top of the ABI
  - Tends to be the dominant toolchain on a particular platform rather than documentation
- The 64-bit ABI is more traditional, concentrating on what platforms need to build upon.
References

● ARM published
  ○ ABI for the ARM 32-bit Architecture
  ○ ABI for the ARM 64-bit Architecture
    ■ Release 1.0
      ● Incorporating PCS, ELF, Dwarf and C++
    ■ Release 1.1 Beta
      ● PCS and ELF changes for ILP32

● Generic Standards
  ○ Itanium C++ ABI [https://mentorembedded.github.io/cxx-abi/abi.html](https://mentorembedded.github.io/cxx-abi/abi.html)
  ○ Thread local storage [https://www.akkadia.org/drepper/tls.pdf](https://www.akkadia.org/drepper/tls.pdf)
  ○ Thread local storage descriptors [https://www.fsfla.org/~lxoliva/writeupsTLSRFC-TLSDESC-ARM.txt](https://www.fsfla.org/~lxoliva/writeupsTLSRFC-TLSDESC-ARM.txt)
Thank You

#BUD17
For further information: www.linaro.org
BUD17 keynotes and videos on: connect.linaro.org