Bound-T timing analysis tool

Technical Note

AOMF with Keil C51 extensions as input to Bound-T

TR-TN-AOMF-001
Version 1
2007-09-27
Tidorum Ltd.

Tidorum Ltd
www.tidorum.fi
Tiirasaarentie 32
FI-00200 Helsinki
Finland

This document was written and is maintained by Niklas Holsti at Tidorum Ltd.

Document reference: TR-TN-AOMF-001
Document issue: Version 1
Document issue date: 2007-09-27
Bound-T version: Various, depending on target processor.
Web location: *http://www.bound-t.com/tech_notes/tn-aomf.pdf*

Credits:
This document was created with the free OpenOffice.org software, *http://www.openoffice.org/*.

Relevant Bound-T source-file versions:

File	*Revision*
formats-aomf_keil.adb	1.4
formats-aomf_keil.ads	1.5
formats-aomf_keil-opt.ads	1.4
formats-aomf_keil-parsing.adb	1.8
formats-aomf_keil-parsing.ads	1.4
formats-aomf_keil-text.adb	1.4
formats-aomf_keil-text.ads	1.2

The information in this document is believed to be complete and accurate when the document is issued. However, Tidorum Ltd. reserves the right to make future changes in the technical specifications of the product Bound-T described here. For the most recent version of this document, please refer to the web address *http://www.tidorum.fi/*.

If you have comments or questions on this document or the product, they are welcome via electronic mail to the address *info@tidorum.fi*, or via telephone, fax or ordinary mail to the address given below.

Please note that our office is located in the time-zone GMT + 2 hours (+3 hours in the summer) and office hours are 9:00 -16:00 local time.

Cordially,

Tidorum Ltd.

Telephone: +358 (0) 40 563 9186
Web: http://www.tidorum.fi/
http://www.bound-t.com/
Mail: info@tidorum.fi

Post: Tiirasaarentie 32
FI-00200 HELSINKI
Finland

Credits

The Bound-T tool was first developed by Space Systems Finland Ltd (*http://www.ssf.fi*) with support from the European Space Agency (ESA/ESTEC). Free software has played an important role; we are grateful to Ada Core Technology for the Gnat compiler, to William Pugh and his group at the University of Maryland for the *Omega* system, to Michel Berkelaar for the *lp-solve* program, to Mats Weber and EPFL-DI-LGL for Ada component libraries, and to Ted Dennison for the *OpenToken* package. Call-graphs and flow-graphs from Bound-T are displayed with the *dot* tool from AT&T Bell Laboratories.

Contents

1 INTRODUCTION

1.1 Purpose and Scope

Bound-T is a tool for computing bounds on the worst-case execution time of real-time programs; see reference [1]. Bound-T applies static analysis to the machine-code program in its compiled, linked and executable form. Bound-T must therefore start by reading in the machine-code program from a file, for example a file in the ELF format. Bound-T can read and understand several program-file formats. In addition to the basic machine code (memory load image) a program file usually also contains symbolic debugging information for the program (and this is usually the complex part of the format).

There are different versions of Bound-T for different target processors. Each version supports a set of program formats that are commonly used with that target processor and its cross-compilers. The Bound-T Application Note for each target explains which formats are supported.

This Technical Note supplements the Bound-T User Manual [1] and the target-specific Application Notes by describing how Bound-T reads and models programs represented in the *Absolute Object Module Format* (AOMF) as defined by Intel [3] and extended by Keil [4, 2]. This format is often used for programs that run on MCS-51 microcontrollers, also known as “8051” processors, but could conceivably be used for other targets, too.

This Technical Note describes Bound-T AOMF support in a generic way, without considering a particular target processor. When a version of Bound-T for a target processor supports AOMF, the Application Note for that target processor may give additional target-specific details, for example on the modeling of variables held in the processor's registers.

1.2 Overview

The reader is assumed to be familiar with the general principles and usage of Bound-T, as described in the Bound-T User Manual [1]. The user manual also contains a glossary of terms, some of which may be used in this Technical Note. You may also may find it useful to first read the Bound-T Application Note for your target processor.

The remainder of this document is structured as follows:

- Chapter 2 describes the main features of the AOMF format and how they relate to the functions of Bound-T. The chapter also gives an overview of how Bound-T reads and uses an AOMF file.
- Chapter 3 explains the warning messages that Bound-T may emit if it finds some problems or unsupported features in an AOMF file.
- Chapter 4 explains the possible error messages similarly.

1.3 References

[1] Bound-T User Manual.
Tidorum Ltd, Doc. ref. TR-UM-001.
http://www.bound-t.com/user-manual.pdf.

[2] Keil – an ARM company. http://www.keil.com/.

[3] External Product Specification for the MCS-51 Object Module Format. Intel Corporation, V5.0, Sept 05, 1982.

[4] Additions to the 8051 Object Module Format (OMF-51). Keil Elektronik GmbH, 05/07/2000.

1.4 Typographic Conventions

We use the following fonts and styles to show the role of pieces of the text:

-option	A command-line option for Bound-T or other tools.
symbol	A mathematical symbol or variable.
`text`	Text quoted from a text / source file or command.
Record Field	The name of an AOMF record (type), or the name of a field in an AOMF record.

1.5 Abbreviations and Acronyms

See also reference [1] for abbreviations specific to Bound-T and Appendix B of [3] for a glossary of AOMF terms.

AOMF	Absolute Object Module Format
BL51	Keil Banking Linker
OMF	Object Module Format
PL/M	Programming Language for Microcomputers
WCET	Worst-Case Execution Time

2 THE AOMF FORMAT

2.1 AOMF Features

AOMF and OMF

The Absolute Object Module Format AOMF is a subset of the Object Module Format or OMF [3]. OMF can represent relocatable code (compiled but not yet linked) but AOMF, as defined in Appendix C of [3], is limited to absolute code that is both compiled and linked, that is, placed at absolute addresses in memory. Bound-T analyses only absolute code and therefore reads only AOMF, not OMF.

AOMF record types

AOMF as Intel defined it in [3] is a *binary* file format – not text. An AOMF file is a sequence of *records* and each record is a sequence of 8-bit *octets*. There are several *types* of record. In each type the octets are grouped into data *fields* in different ways. The names of the record types and data fields reflect the fact that OMF and AOMF were first used with the PL/M programming language.

Each AOMF record begins with a *type octet* that gives the record type. This is followed by a two-octet *record-length* field, followed by type-specific fields. The last octet in each record is a *check-sum* of the record's contents. The length field lets a reader skip records without reading their contents or even knowing the structure of the contents.

A basic AOMF file can contain the following record types [3]:

- Module Header Record: Starts the file and defines the name of the module.
- Content Record: Defines a part of the program by giving code (or data) octets to be loaded at a given address in a given program "segment".
- Scope Definition Record: Names the procedure or block that contains the symbols defined in following Debug Items Records. Another form of Scope Definition Record marks the end of a procedure or block. Scopes can be nested.
- Debug Items Record: Defines the address (or register) assigned to the symbol for a procedure, label, or data variable. The name of the symbol is also given, of course. Another form of Debug Items Record connects source-code line-numbers to the corresponding machine-code addresses.
- Module End Record: Ends the file.

Keil extensions

The company Keil (now a part of ARM [2]) extended AOMF with more symbolic debugging information and introduced the following record types [4] that Bound-T can read and use:

- Source Name Record: Provides the name of the source-code file that was compiled or assembled to generate the following Content Records and Debug Items Records.

- Type Definition Record: Defines the types of symbols mentioned in Debug Items Records. The full, recursive type structure of the C language is supported. However, at present Bound-T makes very little use of type infomation.
- Extended Debug Items Record: Associates type information from Type Definition Records with each symbol.

Keil also extended AOMF to support "banked" code by means of the following new or extended record types [4] that Bound-T does *not* support:

- BL51 Bank Head Record
- Banked Content Record
- Banked Scope Definition Record
- Banked Debug Items Record
- Banked Extended Debug Items Record.

According to [4] Keil also added, or will add, further record types for "source browse" and other features. Bound-T tries to tolerate such records by skipping them.

Overall AOMF structure

The records in an AOMF file occur in an order that obeys the syntax defined in [3] and [4]. We give only an outline here.

For non-banked code the file starts with a Module Header Record and ends with the corresponding Module End Record. Between these records is a mixture of Content Records, Scope Definition Records, Type Definition Records, Source Name records, and (Extended) Debug Items Records.

Scope Definition Records occur in pairs. The first record opens a scope and the second record closes the scope. Such scopes can be nested.

A Source Name record can come immediately after a scope-starting Scope Definition Record, and only there.

The initial Module Header Record and the final Module End Record are also held to define a scope – the module scope.

Banked AOMF structure

An AOMF file for banked code begins by a BL51 Bank Head Record which is followed by the Module Header Record and other records as in a non-banked file, but using the extended, Banked form of each record type.

As already said Bound-T does not support banked code. However, Bound-T will make an attempt to read banked AOMF in the hope that the banking is irrelevant to the particular analysis that is performed.

2.2 How Bound-T Reads an AOMF File

Auto-detecting AOMF

A given version of Bound-T can often read several forms of program files. If the user does not specify the file-format with a command-line option Bound-T will try to determine the format from the given program file itsef. This is called "auto-detecting" the format.

To determine if a given file is an AOMF file Bound-T tries to read one AOMF record from the start of the file. Bound-T considers the file to be AOMF if it finds a valid Module Header Record, BL51 Bank Head Record, or Keil "source browse" record at the start of the file.

Reading and loading an AOMF file

Bound-T reads an AOMF file in one pass from the first record (Module Header or BL51 Bank Head) to the last record (Module End). Any data in the file after the Module End Record is ignored. While reading the file, Bound-T:

- checks the check-sum field of each record,
- checks that scope-starting Scope Definition Records pair up with matching scope-ending Scope Definition Records, with nested scopes started and ended in last-in-first-out order,
- loads code/data bytes from Content Records into a memory-image data structure that Bound-T will later analyse as the code and initial data of the target program,
- loads type definitions from Type Definition Records into a type data-base for use by later symbol definitions,
- records source-file names from Source Name records for use by later source-line/code-address connections,
- loads symbol definitions and source-line/code-address connections from (Extended) Debug Items Records into a symbol-table data structure that Bound-T will use to display analysis results in source-code terms and to translate assertions expressed in source-code terms into assertions on machine-code entities.

The following sections explain some details of these steps and actions.

2.3 Scopes for Symbols and Line Numbers

Constructing Bound-T scopes for symbols

The Bound-T symbol table attaches a static, lexical *scope* to each symbol (the name of a subprogram or a variable) and to each connection between a source-code line number and a code address. The scope is a list of zero or more strings that are the names of the program parts (modules, subprograms, *etc.*) that lexically contain the symbol. The strings in the list are called the *levels* of the scope.

When Bound-T reads an AOMF file it maintains a *current symbol scope* that reflects the current nesting of Scope Definition Records. The scope levels are simply the Block Name fields of the nested Scope Definitions from outermost to innermost.

For example, symbols defined in a Debug Items Record that is not within a Scope Definition have a null Bound-T scope. Symbols defined in a Debug Items Record that is within a Scope Definition with Block Name = "`Foo`" have the Bound-T scope "`Foo`". If there is a nested Scope Definition with Block Name = "`Blk1`" then symbols defined in this scope have the Bound-T scope "`Foo|Blk1`" where the '|' separates levels in the scope.

The module name, as defined in the Module Header Record, is not usually entered in the Bound-T scope for a symbol.

The above explains the general or default construction of scopes for symbols. This default can be overriden in versions of Bound-T for some target processors. More below on target-specific parts of AOMF processing.

Constructing Bound-T scopes for source-line numbers

The Bound-T symbol table attaches a static *scope* to each connection between a source-code line number and a code address. These scopes differ from the scopes for symbols. The scope for a line-number/code-address connection usually has two levels where the first (outer) level is the name of the source-code file and the second (inner) level is the name of the subprogram that contains the source line.

When Bound-T reads an AOMF file it creates a *current line scope* that reflects the content of the Module Header Record, the most recent Source Name Record, and the most recent scope-starting Scope Definition Record for a procedure (a subprogram), as follows:

- First (outermost) scope level (~ source file name):
  - the source-file name from the most recently preceding Source Name record, if there is such a record; otherwise
  - the module name from the Module Header Record, if not null; otherwise
  - a null string (and a warning message).
- Second (next inner) scope level (~ subprogram name):
  - the procedure name (Block Name field) from the most recently preceding Scope Definition Record that starts a procedure scope, if there is such a record; otherwise
  - a null string (and a warning message).

This construction of scopes for line-numbers is fixed and cannot be overridden in a target-specific way.

2.4 Target-Specific Parts of AOMF Processing

The process described above for reading and loading an AOMF file invokes some operations that are specific to the target processor for which the AOMF program is compiled and linked. The actions that can be defined in a target-specific way are the following:

- Deciding whether a Content Record for a given segment should be loaded into the Bound-T memory image, and if so, in what octet order (big-endian, little-endian, word size).

- Deciding whether a data symbol (variable) defined in an (Extended) Debug Items Record should be loaded into the Bound-T symbol table, and whether it should be modelled as an arithmetic “storage cell” that can hold an integer value. For example, floating-point variables are usually not loaded or modelled.
- Defining the Bound-T scope for symbols. Target-specific operations can override the default symbol-scope construction explained above.
- Translating the 16-bit memory addresses in (Extended) Debug Items Records to the kind of code/data address used in the target processor. Likewise for data register names.

These steps in AOMF processing should be explained in the Bound-T Application Note for the relevant target processor, as should be the warning or error messages that may issue from these steps.

3 AOMF WARNING MESSAGES

The following table lists the warning messages that Bound-T may emit to highlight some problems or unsupported features in an AOMF file. The messages are listed in alphabetical order, perhaps slightly altered by variable fields in the message; such fields are indicated by *italic* text. The Bound-T User Manual [1] explains the general form of warning messages. The Bound-T Application Note for the relevant target processor may describe additional warning messages relating to the target-specific steps in AOMF processing.

The probable reason for any of these warnings is either a damaged AOMF file, or a file that uses a version of AOMF that Bound-T does not support. To correct the problem you should obtain an undamaged program file, in a supported version of AOMF, or in some other format that Bound-T supports for your target processor.

Table 1: Warning messages for AOMF

Warning Message	*Meaning*
AOMF record with code *C* (hex) skipped; end-of-scope out of context	The file contains a scope-ending Scope Definition Record at the outermost scope level. Thus the record cannot and does not pair up with any earlier scope-starting Scope Definition Record.
AOMF record with code *C* (hex) skipped; invalid in a module	The file contains, between the initial Module Header Record and the final Module End Record, but not within a scope defined by a pair of Scope Definition Records, a record that is not allowed at such a place, namely another Module Header Record, a Source Name record, a BL51 Bank Head Record, or a Keil “source browse” record.
AOMF record with code *C* (hex) skipped; invalid in a scope	The file containts, between a scope-starting Scope Definition Record (after the optional Source Name Record for this scope) and the matching scope-ending Scope Definition Record, but not within a nested scope, a record that is not allowed at such a place, namely another Source Name Record, a Module Header Record, a BL51 Bank Head Record, a Module End Record, or a Keil “source browse” record.
AOMF record with code *C* (hex) skipped; unknown content	The DEF TYP field in this Debug Items Record has a value that does not represent a known kind of debug items: local symbols, public symbols, segment symbols, or source-line numbers.
AOMF source-file name unknown for line numbers	This Debug Items Record contains source-line number/code-address connections, but Bound-T finds no information on which source file contains these source lines because there is no preceding Source Name Record and the Module Header Record defines no module name. Therefore the source-file part of the Bound-T scope (see [1]) will be a null string for these source-line numbers.
AOMF subprogram name unknown for line numbers	This Debug Items Record contains source-line number/code-address connections, but Bound-T finds no information on which subprogram contains these source lines because there is no preceding Scope Definition Record that starts a procedure. Therefore the subprogram-name part of the Bound-T scope (see [1]) will be a null string for these source-line numbers.
Skipping AOMF BL51 Bank Head record; not implemented	The AOMF file begins with a BL51 Bank Head Record. Bound-T does not support banked code, but continues reading the file in the hope that banking is not relevant to this analysis.

Warning Message	*Meaning*
Unknown AOMF Record Type = *T*	The file contains a record with a record-type field that has a value (*T*, in hex) that Bound-T does not know about.
Unknown Pointer Kind = *K*	This Type Definition Record defines a Generic Pointer Descriptor [4], but the field that denotes the kind of pointer (pSpec_8) contains the number *K* (in decimal) that Bound-T does not know about.
Unknown Pointer Space = *S*	This Type Definition Record defines a Spaced Pointer Descriptor or a Generic Pointer Descriptor [4], but the field that denotes the kind of memory space that the pointer points to (Mspace_C51_n8 or MSpace_8) contains the number *S* (in decimal) that Bound-T does not know about.

4 AOMF ERROR MESSAGES

The following table lists the error messages that Bound-T may emit to highlight severe problems or unsupported features in an AOMF file. The messages are listed in alphabetical order, perhaps slightly altered by variable fields in the message; such fields are indicated by *italic* text. The Bound-T User Manual [1] explains the general form of error messages. The Bound-T Application Note for the relevant target processor may describe additional error messages relating to the target-specific steps in AOMF processing.

The probable reason for any of these errors is either that the AOMF file is damaged or that the file uses a version of AOMF that Bound-T does not support. To correct the problem you should obtain an undamaged program file, in a supported version of AOMF, or in some other format that Bound-T supports for your target processor.

Table 2: Error messages for AOMF

Error Message	*Meaning*
AOMF level 0 scope should be BEGIN_MODULE, not *S*	The file contains an outermost Scope Definition Record (one that is not nested in another scope) in which the BLK TYP field denotes the start of some nested kind of scope, *S*, instead of a module scope as expected at level 0.
AOMF level 1 scope should be BEGIN_PROCEDURE, not *S*	The file contains a level-1 Scope Definition Record (one that is nested only within the module, not in a procedure or a block) in which the BLK TYP field denotes the start of some deeper nested kind of scope, *S*, instead of a procedure scope as expected at level 1.
AOMF level *L* scope should be BEGIN_DO, not *S*	The file contains a Scope Definition Record nested at level $L > 1$ in which the BLK TYP field denotes the start of some other kind of scope, *S*, instead of a DO-block scope as expected at this level.
AOMF Module_Header.Name /= Module_End.Name	The Block Name field in the Module End Record is not equal to the Block Name field in the Module Header Record.
AOMF *kind* record invalid or out of context. *Detail*.	The file contains a record of the given *kind* that is invalid in some way (usually reported by preceding error messages) or appears in the wrong context for this kind of record, according to the overall syntax of AOMF. The *Detail* field gives more information on the context.
AOMF record has wrong check-sum	The trailing check-sum octet in a record does not match the check-sum computed from the record's other content.
Unknown AOMF Compound Type Tag = *T*	The file contains a Type Definition Record with a value *T* (hex) in the TI field that is in the range of compound type descriptors but is not a valid such value.
Too short AOMF Content.Length = *L*	The value *L* (octets, in decimal) in the Record Length field of a Content Rcord is shorter than the least possible length for this type of record.
Unknown AOMF Debug_Items.Def_Type = *B*; skipping record	The file contains a Debug Items Record [3] with an invalid value *B* (decimal) in the DEF TYP field.
Unknown AOMF Scope_Definition.Block_Type =*B*	The file contains a Scope Definition Record with an invalid value *B* (decimal) in the BLK TYP field.

Error Message	*Meaning*
Unknown AOMF Segment_Info.Seg_Type = *S*	This Debug Items Record defines segment symbols but the SEG TYPE part of the SEG INFO field contains the number *S* (in decimal), a number unknown to Bound-T for this field.
Unknown AOMF Symbol_Info.Usage = *U*	This Debug Items Record defines local, public or segment symbols, but the USAGE TYPE part of the SYM INFO field contains the number *U* (in decimal), a number unknown to Bound-T for this field.

Tidorum Ltd

Tiirasaarentie 32
FI-00200 Helsinki, Finland
www.tidorum.fi
Tel. +358 (0) 40 563 9186
VAT FI 18688130