Bound-T
Application Note
MCS®-51 (8051) Family

Version 1

March 2001

Space Systems Finland Ltd
www.ssf.fi
Kappelitie 6
FIN-02200 ESPOO
Finland

This document was written at Space Systems Finland Ltd. by Ville Sipinen.

The document is currently maintained by the same person(s).

Document reference: CWT-SSF-MA-002
Document issue: Draft
Document issue date: March 2001
Bound-T version: 1
Web location: none

Trademarks:

Preface

The information in this document is believed to be complete and accurate when the document is issued. However, Space Systems Finland 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.ssf.fi/boundty/*.

If you have comments or questions on this document or the product, they are welcome via electronic mail to the address *bound-t@ssf.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, and office hours are 9:00 -16:00 local time.

Cordially,

Space Systems Finland Ltd.

Telephone:	+358 9 6132 8600
Fax:	+358 9 6132 8699
Web:	*http://www.ssf.fi* *http://www.ssf.fi/boundty/*
Mail:	Kappelitie 6 FIN-02200 ESPOO Finland

NOTICE FOR PRELIMINARY DOCUMENT VERSION:

Notwithstanding the above preface, the present version of this document is entirely preliminary and distributed for your information only. The distribution of this document shall not be understood as placing Space Systems Finland (SSF) under any obligation to create or supply a product of the type described here. SSF is developing this product with the support of the European Space Agency and others, but SSF has no obligations to other parties with respect to this development.

This page is blank on purpose

CONTENTS

LIST OF TABLES

This page is blank on purpose

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]. There are different versions of Bound-T for different target processors. This Application Note supplements the Bound-T User Manual [1] by giving additional information and advice on using Bound-T for one particular target processor family, the Intel MCS(R)-51 (8051) Family.

Some information in Chapters 4 and 5 of this Application Note applies only when the target-program executable is generated with the Keil 8051 C-compiler or assembler. This information could have been the subject of an independent Application Note but was included here because the Keil tools are very commonly used for this processor family.

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, many of which will be used in this Application Note.

In a nutshell, here is how Bound-T bounds the worst-case execution time (WCET) of a subprogram: Starting from the executable, binary form of the program, Bound-T decodes the machine instructions, constructs the control-flow graph, identifies loops, and (partially) interprets the arithmetic operations to find the "loop-counter" variables that control the loops, such as *n* in "*for* (*n* = 1; *n* < 20; *n*++) { ... }".

By comparing the initial value, step and limit value of the loop-counter variables, Bound-T computes an upper bound on the number of times each loop is repeated. Combining the loop-repetition bounds with the execution times of the subprogram's instructions gives an upper bound on the worst-case execution time of the whole subprogram.

This Application Note explains how Bound-T has been adapted to the architecture of the MCS(R)-51 (8051) Family processors and how to use Bound-T to analyse programs for these processors. To make full use of this information, the reader should be familiar with the register set and instruction set of this processor, as presented in reference [2].

The remainder of this Application Note is structured as follows:

- Chapter 2 describes the main features of the 8051 architecture and how they relate to the functions of Bound-T.
- Chapter 3 defines in detail the set of 8051 instructions and registers that is supported by Bound-T.

- Chapter 4 explains those Bound-T command arguments and options that are wholly specific to the 8051 processors or that have a specific interpretation for these processors.
- Chapter 5 addresses the user-defined assertions on target program behaviour and explains the possibilities and limitations in the context of the 8051.

1.3 References

[1] Bound-T User Manual.
Space Systems Finland Ltd., Doc.ref. DET-SSF-MA-001.

[2] 8-bit Embedded Controller Handbook.
Intel ® 1990

[3] C51 Compiler User's Guide 01.97
Keil Software Inc

1.4 Abbreviations and Acronyms

See also reference [1] for abbreviations specific to Bound-T and reference [2] for the mnemonic operation codes and register names of the MCS(R)-51 (8051) Family.

Effort	Describes the execution time of an instruction in processing cycles and the number of memory reads and writes performed by it.
LSB	Least Significant Byte
MSB	Most significant Byte
Scope	Presents the context of an object of the program (for example when name only is not sufficient)
WCET	Worst-Case Execution Time

2 The 8051 and Timing Analysis

2.1 The 8051

The 8051 [2] is an 8-bit micro-controller. It has a “Harvard” architecture (separated program and data memories). Instructions can be 8, 16 or 24 bits wide. Data can also be read from the program memory. Data memory is divided into internal and external with respect to the processor. All accesses to the program memory and external data memory are addressed indirectly with dedicated registers defining the actual address.

All arithmetic integer operations are supported in hardware, but floating point operations are not supported at all. No standard floating point type is defined.

An on-chip stack in the internal data memory contains the return addresses from subroutines and data pushed by PUSH instructions. Since the internal memory is at most 256 bytes, and includes the banked registers (see section 3.6), the stack must be less than 256 bytes.

2.2 Static Execution Time Analysis on the 8051

The 8051 architecture is very regular and quite fitting for static analysis by Bound-T. Instruction timing in no case depends on the data being processed, but only on the control flow.

The automatic analysis of the loop counters is limited to unsigned 8-bit computation (see section 3.1).

3 Supported MCS(R)-51 (8051) Family Features

3.1 Overview

This section specifies which 8051 instructions, registers and status flags are supported by Bound-T. We will first describe the extent of support in general terms, with exceptions listed later. Note that in addition to the specific limitations concerning the 8051, Bound-T also has generic limitations as described in the User Manual [1]. For reference, these are briefly listed in section 3.4.

General support level

In general, when Bound-T is analysing a target program for the 8051, it can decode and correctly time all instructions.

Bound-T can construct the control-flow graphs and call-graphs for all instructions, unless they contain unresolved jumps to dynamically defined destinations.

When analysing loops to find the loop-counter variables, Bound-T is able to track all the 8-bit additions and subtractions assuming unsigned variables. Bound-T correctly detects when this integer computation is overridden by other computations, such as multiplications in the same registers.

However there is one considerable limitation specific to 8051 processors: they don't have any dedicated machine instructions to handle signed numbers and cannot for example directly compare signed numbers. Because of this limitation the program code processing signed integers often contains operations which Bound-T 8051 cannot support at *arithmetic level* (see section 3.2) and therefore automatic loop bounding is possible only if loop counters and limits are unsigned 8-bit numbers. However counter steps can be positive or negative.

Furthermore because all registers (except the data pointer) are 8 bits wide and all arithmetic operations are performed with 8-bit entities, the processing of bigger variables requires several arithmetic operations to several registers or memory locations. Currently Bound-T does not understand that these operations actually process single variables and cannot find and bound loop counters that are bigger than 8-bit variables. In 'C' terms loop counters should to be "unsigned char".

Loops with signed counters or 16-bit or larger counters can be bounded only by user-given assertions.

Before detailing the exceptions to the general support, some terminology needs to be defined concerning the levels of support.

3.2 Levels of Support

Four levels of support can be distinguished, corresponding to the four levels of analysis used by Bound-T:

1. *Instruction decoding*: are all instructions correctly recognised and decoded? Is the execution time of each instruction correctly and exactly included in the WCET, or only approximately?
2. *Control-flow analysis*: are all jump and call instructions correctly traced to their possible destinations? Are there other instructions that could affect control flow, and are they correctly decoded and entered in the control-flow graph?
3. *Definition analysis*: does Bound-T correctly trace the effect of each instruction on the data flow, in terms of which "cells" (registers, memory locations) are defined (written, modified) by the instruction?
4. *Arithmetic analysis*: to what extent are the arithmetic operations of instructions mastered, so that the range of the results can be bounded?

These levels are hierarchical in the sense that a feature is considered to be supported at one level only if it is also supported at all the lower levels, with arithmetic analysis as the highest level.

Opaque values

When an operation is supported at the definition level, but not at the arithmetic level, then Bound-T's arithmetic analysis considers the operation's results to be "unknown" or *opaque*.

When an opaque value is stored in a register or memory location, the store is understood to destroy the earlier (possibly non-opaque) value and replace it with the opaque value. For arithmetic analysis, an opaque value represents an unconstrained value from the set of possible values of the storage cell (8 bits - or 16 bits in case of data pointer - for a general register, 1 bit for a flag).

The difference between definition analysis and arithmetic analysis is crucial to Bound-T's ability to bound the worst-case times of loops. To illustrate this difference, the following table lists some 8051 instructions in the leftmost column and their definition-analysis and arithmetic analysis in the two other columns. The instructions are assumed to be executed in sequence. The analysis contains just the aspects supported by Bound-T.

Table 1: Definition Analysis vs Arithmetic Analysis

Instruction	Definition analysis	Arithmetic analysis
MOV A,#80H	Accumulator gets new value	Accumulator gets value 128
ADD A,#90H	Accumulator gets new value	Accumulator gets value 16 (8 LSB bits of sum 80H + 90H) and carry flag gets value 1 since sum did not fit in 8 bits.

Table 1: Definition Analysis vs Arithmetic Analysis

Instruction	Definition analysis	Arithmetic analysis
MOV DPTR,#0	Data pointer gets new value.	Data pointer gets value 0
MOVX @DPTR,A	External memory location pointed by the data pointer gets new value	External memory location 0 gets value 16
INC DPTR	Data pointer gets new value	Data pointer gets value 1 (previous value 0 incremented by one)
MOVX A,@DPTR	Accumulator gets new value	Accumulator gets opaque value, because the memory location 1 has unknown value (there is no instruction that would have been set some known value to it).
MOV R0, PSW	R0 gets new value	R0 gets opaque value, because state of PSW is considered to be always unknown.
SETB F0	No effect, because F0 bit (of PSW) is not tracked	No effect

Arithmetic analysis tracks the formulae, not the values; the values (or value ranges) are then calculated from the formulae when needed.

3.3 Implications of Limited Support

Looking at the support levels from the Bound-T user's point of view, the following implications arise when the target program uses some 8051 feature which is *not* supported at some level.

- *Arithmetic analysis*: If a feature is supported at all levels except arithmetic analysis, then using this feature in any loop-counter computation will keep Bound-T from identifying the loop-counters (due to opaque values) so these loops cannot be bounded automatically. However, the other results from Bound-T stay valid.

For example, if the initial value of a loop-counter is read from a memory location or a register whose value is unknown, then Bound-T cannot compute bounds for the initial value and thus cannot bound the loop (without a user-given assertion).

- *Definition analysis*: If a feature is not supported in definition analysis, then in addition to the preceding impact, using this feature implies a risk of invalidating the arithmetic analysis, and thus a risk of incorrect results from Bound-T. Few 8051 features are at this level of non-support, and Bound-T will warn if they are used. The switching of the register bank is one example (see section 3.6).
- *Control-flow analysis*: If a feature is not supported in control-flow analysis, then Bound-T can produce arbitrary (correct or incorrect) results when this feature is used in the target program, because the correct control-flow graphs cannot be determined. Again, Bound-T will warn of such usage.

- *Instruction decoding*: If a feature is not supported even for decoding, then it is useless to run Bound-T on a target program that uses this feature, since the only reliable result will be error messages. However, all 8051 features are supported at this level.

3.4 Reminder of Generic Limitations

To help the reader understand which limitations are specific to the 8051 architecture, the following compact list of the generic limitations of Bound-T is presented.

Table 2: Generic Limitations of Bound-T

Generic Limitation	Remarks for 8051 target
Understands only integer operations in loop-counter computations.	Loop counter analysis can succeed only if loop counters and limits are unsigned 8-bit variables.
Understands only addition, subtraction and multiplication by constants, in loop-counter computations.	The MUL instruction and logical/arithmetic shifts must not be used in loop counting.
Assumes that loop-counter computations never suffer overflow.	Loop counter analysis can succeed only if the loop repeats less than 256 times. A loop that repeats 256 times can be built when an overflow of a 8-bit variable is used together with a suitable overflow option (see section 4.2 for details about 8051 specific options).
Can bound only counter-based loops.	No implications specific to the 8051.
Can analyse only reducible control-flow graphs	No implications specific to the 8051.
May not resolve aliasing in dynamic memory addressing.	No implications specific to the 8051.

3.5 Support Synopsis

The following table gives a synoptical view of the level of support for 8051 features. A ‘X’ in a cell means that the feature corresponding to the table row is supported on the level corresponding to the table column. A shaded cell indicates lack of support.

Table 3: Synopsis of 8051 Support

8051 registers, instructions, or other features	Decoding	Control flow	Definition	Arithmetic
Program Status Word (PSW)	X	X	X	
Carry flag (C)	X	X	X	X
Accumulator (Acc)	X	X	X	X
B-register	X	X	X	
Data Pointer (DPTR)	X	X	X	X
Special Function Registers	X	X	X	
DPH & DPL (MSB & LSB of Data pointer)	X	X	X	
Register bank (registers R0 .. R7)	X	X	X	X
Register bank switching	X	X		
Indirect addressing	X	X	X	X
Bit addressing	X	X	X	
Paged addressing	X	X	X	X
Addition & Subtraction	X	X	X	X
Multiplication	X	X	X	
Division	X	X	X	
Logical operations (bitwise AND, OR and XOR)	X	X	X	
Rotation operation (left & right)	X	X	X	
Swapping of nibbles	X	X	X	
Data transfer (MOV, MOVX, MOVC)	X	X	X	X
Boolean variable manipulation	X	X	X	
Decimal adjust and digit exchange	X	X	X	
Arithmetic effects of branch instructions (CJNE, DJNZ)	X	X	X	X

3.6 Registers and Memory Accesses

The 8051 contains several Special Function Registers with different roles and several general purpose registers whose location in the memory depends on the effective register bank selection. This section explains how Bound-T supports these registers.

Banked registers R0 - R7

Banked registers are located in the beginning of the processor's internal memory in the locations defined by the effective register bank selection. There are four possible register bank selections: bank 0, 1, 2 and 3. When bank 0 is selected the banked registers correspond to memory locations 0 .. 7, to locations 8 .. 15 when bank 1 is selected an so on. Bound-T does not track the changes of the register bank selections, but assumes that the register bank selection stays unchanged in the analysed processing thread. If an instruction which would change the register bank is detected, it is ignored, but a warning message is printed. The one effective register bank selection can be set with a target specific option described in section 4.2.

DPH & DPL Registers

The DPH and DPL registers contain the MSB and LSB of the Data Pointer respectively. The values of these registers are not tracked, but manipulation of these registers affects the Data Pointer and therefore writing to either of these registers leads generally to an opaque value of the Data Pointer. The only exception is when the instructions MOV DPH,#immediate8 and MOV DPL,#immediate8 are in consecutive code addresses in either order. In this case these instructions are effectively merged to one MOV DPTR,#immediate16 instruction that loads a 16 bit constant to the Data Pointer. The effort of the “merged” instruction is however of course the sum of the efforts of the original instructions.

Program Status Word

The value of this register as a whole is not tracked and therefore any reading from it yields an opaque value. However when a writing to it is detected, it is checked whether the new value would change the register bank selection (see banked registers). The value of the Carry flag included in this register is anyway tracked.

Special Function Registers

The only Special Function Registers whose values are tracked are: the Accumulator, the Data Pointer and the page register (P2) for the paged addressing mode. Any reading of other Special Function Registers always yields an opaque value.

3.7 Keil C-51 Calling Protocol

Because of its very limited size the processor's internal stack is not used for parameter passing. The banked registers are the primary method, but with them it is possible to pass only very few parameters (see reference [3]).

If the called subroutine is not re-entrant those parameters that do not fit in the registers are passed in statically assigned memory locations. For re-entrant subroutines a simulated stack in the external data memory is used (see reference [3]). However currently Bound-T does not implement parameter passing through the simulated stack and therefore if a parameter affects one or more loop bounds, it should be passed within those parameters that are passed in registers (see reference [3]).

Bound-T treats the banked registers and memory locations equally as data cells whose values are passed to the called subroutine when necessary. This applies also for the data cells corresponding to global variables used by both the caller and the callee.

3.8 Instructions

Whether or not a computational operation is supported on the arithmetic analysis level depends exclusively on the generic abilities of Bound-T; the only concern here is to map these abilities onto the 8051 instruction set.

Arithmetic operations

The supported arithmetic operations are addition (ADD, ADDC), subtraction (SUBB), increment (INC) and decrement (DEC). Except for increment of DPTR (INC DPTR) all of these operate on 8-bit values. The data pointer is the only 16-bit register and the only arithmetic operation for it is the increment.

Because the range for the 8-bit variables is so small, the tracking of overflows may be necessary. The 8051 version of Bound-T offers options for that (see section 4.2). One of the options is to always set the target of an arithmetic operation possibly causing overflow to an unknown state. With that option automatic loop bounding is not possible and all loop bounds have to be asserted.

The targets of all unsupported arithmetic operations (logical operations etc.) are set to an unknown state.

Rotate operations

The bit rotation operations of the 8051 are not tracked, but the targets of these operations are set to an unknown state.

Branch instructions

All jump and call instructions are supported on all levels. However, there are generic limitations on the control-flow analysis of indirect jumps and calls.

All return instructions are supported on all levels.

Loops

On the machine code level there is only one instruction that can be considered as being targeted for loop structures: decrement by one and jump if not zero (DJNZ). This instruction is fully supported.

Bound-T does not generally handle overflows of loop counters properly, but in some situations this limitation can be tolerated or even taken adavantage of. For example when using the “overflow_off”-option (see section 4.2) the following kind of loop structure can be used for a loop reapeating 256 times (usually only loops that repeats less than 256 times can be automatically be bounded, because of 8-bit limit of loop counters):

```
       MOV    R0, #255
       INC    R0        ; R0 becomes 0 in reality, but when Bound-T
                        ; ignores overflows, R0 is assumed to become 256
Head:  DJNZ   R0, Head  ; R0 decremented and jumps if result <> 0
```

Moves and miscellanea

All “move” instructions (MOV) are supported on the arithmetic level when the source and target are byte registers or byte variables in static memory locations, or in case of loading of the data pointer register a 16-bit register or a 16-bit constant. When the source or target are bit registers or bit variables in static memory locations, support is reduced to the definition level. The only bit manipulation instructions that are supported on the arithmetic level are clearing, setting and complementing the carry flag (CLR C, SETB C and CPL C).

Exchange of two byte variables (XCH) is supported on the arithmetic level, but the exchange digit instruction (XCHD) is supported only on the definition level.

Decimal adjust (DA) and swap nibble (SWAP) of accumalator instructions are supported only on the definition level.

The NOP operation is supported on all levels (well it’s not very hard is it!).

3.9 Time Approximations

The execution times of all 8051 instructions are always constant and therefore there is no need for any approximations of the instruction execution times.

4 Using Bound-T 8051

4.1 Input Formats

The target program executable file must be supplied in the Intel defined AOMF format produced for example by the Keil BL-51 banked linker. Some other developing environments (compilers & linkers) have been tested, but not extensively.

4.2 Command Arguments and Options

The generic Bound-T command format and arguments apply without modification to the 8051 version of Bound-T.

There are specific options as explained in the table below. All the generic Bound-T options apply also.

Option	Meaning and default value	
-bank0	*Function*	Selects register bank 0: registers R0 .. R7 are located in internal RAM locations 0 .. 7.
	Default	Yes.
-bank1	*Function*	Selects register bank 1: registers R0 .. R7 are located in internal RAM locations 8 .. 15.
	Default	No.
-bank2	*Function*	Selects register bank 2: registers R0 .. R7 are located in internal RAM locations 16 .. 23.
	Default	No.
-bank3	*Function*	Selects register bank 3: registers R0 .. R7 are located in internal RAM locations 24 .. 31.
	Default	No.
-overflows_on	*Function*	Sets the overflow tracking on, with exact effects for operations possibly generating overflows. Currently this option prevents the success of the automatic loop bounding.
	Default	No.
-overflows_ignore	*Function*	Sets the overflow tracking to ignore the possible overflows. This means that for example adding 1 to 255 gives result 256 instead of 0 and overlfow.
	Default	Yes.
-overflows_off	*Function*	Sets the overflow tracking off, giving opaque values to targets of operations possibly generating overflows. With this option it is not possible to bound loops automatically and therefore all loop bounds need to be asserted.
	Default	No.

4.3 The Keil Library Routines

Some of the Keil library routines contains irreducible flow graphs or unresolvable dynamic jumps, or they do not follow the standard calling protocol. For these reasons Bound-T cannot analyse these subroutines and when they are called from the program being analysed, the execution time of them has to be asserted.

4.4 Analysing Programs that Use RTX-51

The Keil RTX-51 is a real time kernel for 8051 processors. It contains some functions that cannot be analysed with Bound-T, because they contain irreducible flow graphs or unresolvable dynamic jumps. Some of the functions can also switch tasks and their operation does not entirely concern the scope of determination of the worst case execution time for a single processing thread. The table below lists RTX-51 functions that cannot be analysed and their execution times (courtesy of Keil). When these operations are used in the program being analysed, assertions for them have to be written.

RTX-51 Function	Execution Time (cycles) (*)
isr_recv_message (with message in mailbox)	71
os_attach_interrupt	199
os_create_task	312
os_send_message (to standard task)	459
os_send_message (to fast task)	361
os_send_signal (to standard task)	425
os_send_signal (to fast task)	335
os_start_system	6096
os_wait (on already set signal)	72

(*) The execution times are average values provided by the Keil Software, Inc. Unfortunately the worst case values have not been available.

4.5 HRT Analysis

For HRT analysis, the 8051 is usually run with the RTX-51 kernel from Keil Software. See reference [1] for details about HRT analysis.

The memory reads and writes reported in the HRT analysis output are interpreted to concern only access of external memory. Therefore programs whose variables are located entirely in the processor's internal memory will have zero memory reads and memory writes unless there are some reads from external program memory.

4.6 Choice of Calling Protocol

Currently Bound-T handles all subroutine calls equally and does not for example separate calls to re-entrant and non re-entrant subroutines.

4.7 Basic Output Format Limitations

Most Bound-T outputs, including warning and error messages, follow a common, basic format that contains the source-file name and source-line number that are related to the message.

4.8 Warning Messages

The following lists the Bound-T warning messages that are specific to the 8051 or that have a specific interpretation for this processor. The messages are listed in alphabetical order. The Bound-T User Manual [1] explains the generic warning messages, all of which may appear also when the 8051 is the target.

The specific warning messages refer mainly to unsupported or approximated features of the 8051.

Warning Message		Meaning and Impact
Attempt to change the register bank ignored	*Reasons*	Bound-T expects that only one register bank is used through out the analysed processing thread and switching the register bank is not supported. This warning is printed when a machine code instruction which would change the register bank selection is detected.
	Impact	The ignored attempt for register bank switching can lead to incorrectly decoded arithmetic effects, because after the ignored bank switch the operations with register addresses point to the wrong memory addresses and that would lead wrong results at least after the bank would be switched back to the original. This warning can be ignored when it concerns bank switching at the beginning of the interrupt service.
Call to address zero replaced by return at	*Reasons*	Because after processor reset the execution of any program starts at zero address, this would correspond to rebooting of the program.
	Impact	The flow stops at the return and if the call was at the worst case execution path, the analysis concerns the worst case execution time until the reboot.
Dynamic control flow unbounded at	*Reasons*	Destination address of a dynamic jump could not be bounded.
	Impact	The call is replaced by a return which means that the target of the jump is not included in the flow graph and is therefore ignored in the analysis.

Warning Message		Meaning and Impact
Idle loop (jump to self) replaced by return at	*Reasons*	The idle loop would lead to infinite execution time estimate and therefore it has to be replaced by something else. Here it it is replaced by return, because the execution time of the operation could not be analysed anyhow.
	Impact	Analysis stops at the iternal loop and the result concerns only worst case execution time until the loop.
Scope not closed	*Reasons*	The scope end was not found when expected.
	Impact	The target program file may be corrupted and impossible to analyse.
Unbounded dynamic memory access	*Reasons*	The address of the dynamic memory access could not be bounded.
	Impact	If the dynamic access concerned memory read, the target of the operation gets opaque value. The write operation is ignored and can lead to opaque value of the target of some read operation.
Unknown scope record	*Reasons*	The input file containing the target program includes a scope record with unknown structure.
	Impact	The target program file may be corrupted and impossible to analyse.

4.9 Error Messages

The following lists the Bound-T error messages that are specific to the 8051 or that have a specific interpretation for this processor. The messages are listed in alphabetical order. The User Manual explains the generic error messages, all of which may appear also when the 8051 is the target.

Error Message		Meaning and Impact
Address string not in valid format	*Problem*	String describing a variable or subprogram address could not be converted to a numerical value.
	Reasons	The assertion file contains an invalid address string.
	Solution	Correct the address string in the assertion file.
Address string too short	*Problem*	String describing a variable or subprogram address is too short to contain a valid address.
	Reasons	The assertion file contains an invalid address string.
	Solution	Correct the address string in the assertion file.

Error Message		Meaning and Impact
Bit space not supported	*Problem*	Bit variables are not supported.
	Reasons	The assertion file contains an address string defining a bit address (“B:xx”).
	Solution	Change the memory space of the address or remove the assertion containing the bit address.
Cannot read file	*Problem*	The file containing the target program cannot be read.
	Reasons	The target program file may not have read permission.
	Solution	Give read permission to the target program file.
Could not read code byte at offset	*Problem*	There does not exist a code byte at the requested offset.
	Reasons	The target program file may be corrupted.
	Solution	Try to generate a new target program file. If that does not help, there may be an internal error in Bound-T.
File not found	*Problem*	Specified target program file was not found.
	Reasons	The name of the target program file was wrong or the path of it was wrong.
	Solution	Correct the name or path of the target program file, or copy it to proper place.
Illegal instruction at	*Problem*	The instruction being decoded is not a valid 8051 instruction.
	Reasons	The target program file may be corrupted.
	Solution	Try to generate a new target program file. If that does not help, there may be an internal error in Bound-T.
Invalid direct bit address in instruction at	*Problem*	The instruction being decoded has an invalid bit address argument.
	Reasons	The target program file may be corrupted.
	Solution	Try to generate a new target program file. If that does not help, there may be an internal error in Bound-T.
Invalid direct data address in instruction at	*Problem*	The instruction being decoded has an invalid data address argument.
	Reasons	The target program file may be corrupted.
	Solution	Try to generate a new target program file. If that does not help, there may be an internal error in Bound-T.
Invalid immediate byte argument in instruction at	*Problem*	The instruction being decoded has an invalid immediate byte argument.
	Reasons	The target program file may be corrupted.
	Solution	Try to generate a new target program file. If that does not help, there may be an internal error in Bound-T.

Error Message		Meaning and Impact
Invalid immediate word argument in instruction at	*Problem*	The instruction being decoded has an invalid immediate word argument.
	Reasons	The target program file may be corrupted.
	Solution	Try to generate a new target program file. If that does not help, there may be an internal error in Bound-T.
Memory space not recognized	*Problem*	String describing a variable or subprogram address contains an unrecognized memory space indicator.
	Reasons	The assertion file contains an invalid address string.
	Solution	Correct the address string in the assertion file.
Record checksum mismatch	*Problem*	The checksum of one (or more) of the records included in the target program file is wrong.
	Reasons	The target program file may be corrupted.
	Solution	Try to generate a new target program file. If that does not help, there may be an internal error in Bound-T.
Subprogram address not in code space	*Problem*	The address string describing an address of a subprogram defines an address in a memory space other than the code space.
	Reasons	The address string in the assertion file contains some other space indicator than “C:”.
	Solution	Change the space indicator of the address string to “C:”.
Unexpected end of file	*Problem*	The target program file ended unexpectedly.
	Reasons	The target program file may be corrupted.
	Solution	Try to generate a new target program file. If that does not help, there may be an internal error in Bound-T.
Variable address in code space	*Problem*	The address string describing an address of a variable defines an address in the code space. This is an error, because variables cannot be located in the code space. Only constants can be located there.
	Reasons	The address string in the assertion file contains space indicator “C:”.
	Solution	Change the space indicator to “D:” or “X:”.

4.10 Output of Option “-trace effect”

There are two trace options that provide information about the target program’s decoding process: “-trace decode” and “-trace effect”. Both provide otherwise identical information, except that the latter outputs the decoded effect of the instruction and the former does not. So the description below applies to both options, except that “-trace decode” does not provide the instruction effects.

The output contains the following columns:

Address Instruction Mnemonic / Effect Effort Steps Remarks

These columns contain the following information:

- Address: code offset of the decoded instruction
- Instruction: numeric values in hexadecimal format of the instruction bytes forming the instuction

Mnemonic: representation of the instruction containing the type(s) of the possible parameters and instruction mnemonics corresponding to the symbols used in the reference [2] with the following additions to help the internal operation of the decoder:

CPLC	= CPL	C, complements the carry flag
GET	= MOV	<A|addr>, @R<0|1>
GETC	= MOVC	A, @A+PC or MOVC A, @A+DPTR
GETX	= MOVX	A, @DPTR or MOVX A, @R<0|1>
INC16	= INC	DPTR
MOVB	= MOV	<dest_bit>, <src_bit>, copy bit value
PUT	= MOV	@R<0|1>, <A|addr>
PUTX	= MOVX	@DPTR, A or MOV @R<0|1>, A

- Effect: (on its own line) the decoded effect of the instruction
- Effort: the number of cycles to execute the instruction
- Steps: the step numbers in the general control-flow graph associated with the instruction
- Remarks: additional notes related to the instructions.

5 Writing Assertions

This chapter explains any specific limitations and possibilities for user-specified assertions when Bound-T is used with 8051 programs. In fact, these issues are not caused by the 8051 as target processor, but by the Keil-PK51 development tools.

The issues concern the naming of subprograms, variables and source lines (via line numbers).

The special properties that are defined for the 8051 are also listed in the end of this chapter.

5.1 Using Scopes

The scope of a "C" symbol is defined in the following way:

module|subprogram|name,

where the "module" corresponds to the file where the symbol is defined, the "subprogram" to the name of the subprogram containing the symbol and the "name" is the name of the symbol. If the symbol is not local to any subprogram, the "subprogram" part is naturally excluded from the scope.

The scope of an assembler symbol is defined in the following way:

segment|name,

where the "segment" corresponds to the segment defined in the assembler file and containing the symbol and the "name" is the name of the symbol.

5.2 Naming C Subprograms

The Keil compiler seems to follow the following principles for names of "C"-subprograms:

- If the subprogram has parameters and/or returns some value, its name is prefixed with "_".
- If the subprogram has parameters and/or returns some value, and if it is reentrant, its name is prefixed with "_?".

5.3 Naming Assembler Subprograms

The Keil assembler seems to change the names of assembler subroutines to contain only upper-case letters.

5.4 Naming C Variables

The names of C variables seem to remain unchanged.

5.5 Naming Assembler Variables

The names of assembler variables seem to be changed to contain only upper-case letters.

5.6 Specifying Variable and Subprogram Addresses

Addresses of variables and subprograms can be specified with the following kind of strings:

"M:XXXX" or "M:XXXXH",

where the "M" indicates the memory space and has to be one of the following:

- "C", for code memory space
- "X", for external data memory space
- "D", for internal data memory space
- "B", for bit memory space

If the string ends with "H" the address value "XXXX" is interpreted as a hexadecimal number, otherwise as a decimal number. For example:

X:1000H	= external data address 4096
X:15000	= external data address 15000
D:20H	= internal data address 32
C:0200	= code address 200

The code address string has to be at least 4 characters in addition to the memory space indicator and other address strings at least 2 characters.

5.7 Properties

The special properties for 8051 and their meaning is listed in the following table.

Property name	Meaning, value type and default value	
Aregs	*Function*	Allows a subprogram to use absolute register addressing
	Value type	
	Default	
Reentrant	*Function*	Subprogram is reentrant
	Value type	
	Default	

Appendix A: Variant Support

There are many vartiants of the 8051 processor. Currently Bound-T has been ported in particular for the basic 8051 processor. Programs written for the other variants can also be analysed as long as they don't use such currently unhandled features that would affect the results or even disable the analysis (for example because of unrecognized machine instructions). The table below lists some common variants of the 8051, their main differences with respect of the basic 8051 and foreseen problems and limitations for Bound-T usage. Many of the differences in the variants do not affect the analysis, because they are often related to the memory sizes, special function registers, timers etc. which are not relevant for the Bound-T analysis. Generally the diffences are relevant to Bound-T only if they include changes in the instruction set and/or addressing modes of the machine code.

The information in the table below has been taken from the Intel 8-bit controller handbook [2] and internet web-sites of Intel, Philips and Siemens. In some cases clear information about the compatibility of the instruction sets was not found, but then it was assumed that differences do not exist.

Table 4: Support of 8051 variants

Variant	Main Differences	Bound-T Limitations
8031, 80C31	No on-chip ROM	None
8032, 80C32	No on-chip ROM, 3 16-bit timers (instead of 2 timers of basic 8051)	None (additional timer does not affect the analysis)
8044AH	Serial Interface Unit (SIU) and additional special function registers to control it	None (additional special function registers do not affect analysis)
80C51	CMOS version of 8051	None
80C51FA/FB	3 16-bit timers, programmable counter array, 7 interrupt sources (instead of 5), serial interface with framing error detection and automatic address recognition	None (additions do not affect analysis)

Table 4: Support of 8051 variants

Variant	Main Differences	Bound-T Limitations
80C51GB	3 16-bit timers, watchdog counter, 2 programmable counter arrays, 8-bit 8-channel A/D, serial channel with framing error detection and automatic address recognition, serial expansion port,15 interrupt sources (7 external, 8 internal) with 4 priority levels (instead of 2)	None (additions do not affect analysis)
80C52	3 16-bit timers (instead of 2 timers of basic 8051)	None (additional special function registers do not affect analysis)
8344AH	Same as 8044AH but without ROM	None (additional special function registers do not affect analysis)
83C51FA/FB	Same as 80C51FA/FB except with factory masked programmable PROM	None (additions do not affect analysis)
83C51GB	Same as 80C51GB except with factory programmable ROM.	None (additions do not affect analysis)
8744AH	Same as 8344AH except with EPROM	None (additional special function registers do not affect analysis)
8751, 87C51	On-chip EPROM	None
87C51FA/FB/FC	Same as 80C51FA/FB except with EPROM	None (additions do not affect analysis)
87C51GB	Same as 80C51GB except with OTP ROM	None (additions do not affect analysis)
80C152	Global Serial Channel, 2 channels for DMA transfers, new I/O port, several new special function registers (the instruction set is however the same)	Bound-T does not understand the effects of the DMA transfers, since a DMA transfer can change memory data without specific MOV instructions. Thus no loop counter data or variables should be subject to DMA input.
80C251	3-stage pipeline, 40 bytes general purpose Register File accessible as 16 8-bit, 16 16-bit or 10 32-bit registers, 24-bit linear code and data addressing, 64 kBytes stack space, new instructions and addressing modes, 64 interrupt sources with 4 interrupt levels	Bound-T can decode only native 8051 instructions corresponding to the instructions without escape code “A5H” in the binary compatibility mode.
80C451	7 I/O ports and 4 additional special function registers.	None (additions do not effect analysis)
80515, 80C515	6 I/O ports, 3 timers, 8 bit AD-converter, watchdog timer, 12 interrupt sources with 4 priority levels	None (additions do not effect analysis)

Table 4: Support of 8051 variants

Variant	Main Differences	Bound-T Limitations
80C517	4 timers, 16 bit compare/capture unit, MUL/DIV unit, 8 data pointers (1 active selected with a specific additional special function register), 14 interrupt vectors	Bound-T cannot handle the change of selected data pointer and therefore external memory addressing with more than one data pointer is not supported.
80C528	3 timers, watchdog timer, bit level I^2C-bus serial I/O port, 7 interrupt sources,	None (additions do not effect analysis)
80535, 80C535	Same as 80515, but without on-chip ROM	None (additions do not effect analysis)
80C537	Same as 80C517, but without on-chip ROM	See 80C517
83C152	Same as 80C152 except with on-chip ROM	Bound-T does not understand the effects of the DMA transfers.
83C251	Same as 80C251 except with on-chip ROM	See 80C251
83C451	Same as 80C451 except with on-chip ROM	None (additions do not effect analysis)
83C524	Same as 80C528 except with on-chip ROM	None (additions do not effect analysis)
83C528	Same as 80C528 except with on-chip ROM	None (additions do not effect analysis)
87C152	Same as 80C152 except with on-chip EPROM	Bound-T does not understand the effects of the DMA transfers.
87C251	Same as 80C251 except with on-chip EPROM	See 80C251
87C451	Same as 80C451 except with on-chip EPROM	None (additions do not effect analysis)
83C524	Same as 80C528 except with on-chip EPROM	None (additions do not effect analysis)
83C528	Same as 80C528 except with on-chip EPROM	None (additions do not effect analysis)