ON Semiconductor®

Application Note **AND9370/D**

AX8052 Debugger Software Manual

Revision 2

Table of Contents

1. Introduction

The ON Semiconductor AX8052 line of fully integrated embedded microcontrollers feature advanced debug features that significantly ease the task of writing firmware compared to other 8052 compatible microcontrollers. The ON Semiconductor AX8052 Debug System consists of the following components:

- The ON Semiconductor AX8052 Debug Interface. This device connects the AX8052 Microcontroller Debug Interface, consisting of the Signals RESET_N, DBG_EN, PB6, PB7, GND, VCCIO to a standard PC USB Interface.

- The AXSDB command line debugger processes commands and executes them on the Microcontroller using the Debug Interface. The AXSDB debugger can be directly used, or through the AxCode::Blocks IDE. It is fully scriptable thanks to its built-in Tool Control Language (TCL) interpreter.

- The AxCode::Blocks IDE. AxCode::Blocks is a customized version of the popular Code::Blocks IDE. It is documented elsewhere, see AxCodeBlocks.pdf for an introduction. Users wishing to develop with the AxCode::Blocks IDE need not be familiar with axsdb debugger commands, and can skip the remainder of this document.

2. AXSDB

AXSDB is the ON Semiconductor AX8052 Symbolic Command Line debugger. It is fully scriptable, thanks to its built-in Tool Control Language (TCL) scripting engine. It is suggested that the reader consults the Documentation section on the TCL homepage, http://www.tcl.tk, for information on the standard TCL commands. The remainder of this document will describe the AX8052 specific commands AXSDB adds to the standard TCL commands. AX8052 specific Commands come in two flavours:

- Core commands implemented in the Debugger DLL, libaxsdb-0.dll

- Convenience Commands implemented in TCL on top of the core commands, contained in axsdb.tcl. These commands can be changed by the user by changing axsdb.tcl.

All commands, variables and channels are defined in the namespace axsdb. axsdb.tcl imports them into the global namespace.

2.1. State Handling

When AXSDB is started, it responds with a prompt. The debugger is then ready to accept user commands. It does however not yet have access to any hardware.

First, AXSDB must be connected to any AX8052 Debug Interface, connected to an USB port of the Computer running AXSDB. The commands `list_targets`, `connect_target`, and `disconnect_target` manage AXSDB's connections to the active Debug Interface. Note that the default axsdb.tcl automatically connects to the Debug Interface if it finds exactly one connected to the PC. Once AXSDB has connected to a Debug Interface, it is configured to be inactive, i.e. RESET_N is driven high, DBG_EN is driven low, and PB6 and PB7 are set to high-impedance. In this state a target board may be connected to the debug interface without disturbing the target microprocessor.

In order to actually start debugging, AXSDB needs to be connected to the microcontroller. The commands `connect` and `disconnect` manage the connection to the microcontroller hardware debug interface.

Once connected to the microcontroller, commands that control the microcontroller state can be used, such as `run`, `step`, `stop`, etc.

2.2. Command Line Arguments

––norc	Disable the processing of the normal startup script axsdb.tcl; Convenience Commands documented below will not be available.
––script <tclscript>	Add the given script to the TCL scripts evaluated at debugger startup.
––listserials	List the connected debug interface serial numbers adn exit.
––serial <serial>	Connect to debug interface with the given serial number.
––flashprog <file>	Program the microcontroller flash with the given file and exit.
––ignorecalibration	Normally, flashprog saves Microcontroller Calibration data if present in the last 1k sector of the FLASH memory. If this option is given, Calibration data is erased.
––oldkeys <keylist>	This option specifies the old debugger keys to try in order to read out (and

preserve) Microcontroller Calibration data.

––newkey <key>	When ––flashprog is given, use this key to protect the newly flashed firmware
––hwreset	Perform a hardware reset (pulse RESET_N low)
––debuglink	Start as DebugLink relay. Standard input is copied to the debug link, and debug link is copied to standard output. May be combined with ––hwreset. Terminates when standard input is closed.
––savecalib <file>	Save calibration data (if present in the last 1k sector of the FLASH memory) into the file given
––loadcalib <file>	Load calibration data from the file given into the last 1k sector of the FLASH memory
––version	Print the version number and exit
––installdir	Print the installation directory and exit
––help	Display help and exit.

2.3. Core Commands

2.3.1. traceio

The `traceio` command allows an event log to be written into a file; this is intended to help debugging axsdb.

Arguments:

––off	turn logging off
––error	log only error events
––normal	log normal and error events

––poll log polling in addition to normal and error events

––lowlevelio log everything, including low level IO operations

––stderr Log to standard error instead of a supplied file name

––stdout Log to standard error instead of a supplied file name

<filename> Open the given file for writing and use it as log file

2.3.2. LIST_TARGETS

`list_targets` returns a TCL list containing the serial numbers of all connected AX8052 Debug Interfaces.

2.3.3. DISCONNECT_TARGET

`disconnect_target` disconnects axsdb from the currently connected AX8052 Debug Interface. The Debug Interface is set such that it does not interfere with running a connected microcontroller.

2.3.4. CONNECT_TARGET

`connect_target` connects axsdb to the specified AX8052 Debug Interface.

Arguments:

<serial> The serial number of the debug interface to connect to.

2.3.5. TARGET_SERIAL

`target_serial` returns the serial number of the currently connected AX8052 Debug Interface, or an empty string if no Debug Interface is currently connected.

2.3.6. READ_MEM

`read_mem` reads one or more bytes from microcontroller memory. It can only be issued if the microcontroller is in halt state. The read results are returned in a TCL list.

Arguments:

––code, –c	Read from code address space
––direct, –d	Read from direct address space; addresses below 128 address the internal RAM (addresses from 0 to 31 address the four banks of R0–R7 registers), addresses above or equal 128 address the on chip special function registers.
––indirect, –i	Read from indirect address space; this option adresses the internal RAM
––external, –e	Read from external address space
––flash, –f	Read from flash
––sfr, –s	Read from sfr address space
––pagedexternal, –p	Read from paged external address space; this is the same address space as external, however only the low address byte is specified. The high byte is taken from the XPAGE special function register
<address>	The address to read from
<length>	The number of bytes to read; if the length is omitted, one is assumed

2.3.7. WRITE_MEM

`write_mem` writes one or more bytes to microcontroller memory. It can only be issued if the microcontroller is in halt state.

Arguments:

––code, –c	Write to code address space
––direct, –d	Write to direct address space; addresses below 128 address the internal RAM

	(addresses from 0 to 31 address the four banks of R0−R7 registers), addresses above or equal 128 address the on chip special function registers.
−−indirect, −i	Write to indirect address space; this option adresses the internal RAM
−−external, −e	Write to external address space
−−flash, −f	Write to flash
−−sfr, −s	Write to sfr address space
−−pagedexternal, −p	Write to paged external address space; this is the same address space as external, however only the low address byte is specified. The high byte is taken from the XPAGE special function register
<address>	The address to write to
<data...>	The data bytes to write; multiple bytes may be specified

2.3.8. FILL_MEM

`fill_mem` writes a single data byte into one or more consecutive bytes of microcontroller memory. It can only be issued if the microcontroller is in halt state.

Arguments:

−−code, −c	Write to code address space
−−direct, −d	Write to direct address space; addresses below 128 address the internal RAM (addresses from 0 to 31 address the four banks of R0−R7 registers), addresses above or equal 128 address the on chip special function registers.
−−indirect, −i	Write to indirect address space; this option adresses the internal RAM
−−external, −e	Write to external address space
−−flash, −f	Write to flash

––sfr, –s Write to sfr address space

––pagedexternal, –p Write to paged external address space; this is the same address space as external, however only the low address byte is specified. The high byte is taken from the XPAGE special function register

<address> The address to write to

<length> The number of data bytes to write

<data> The data byte to write; it is optional. If not given, the default is to write 0xff into code and flash address space, and 0x00 otherwise.

2.3.9. READ_PC

`read_pc` returns the program counter of the microcontroller. Returned values may be unreliable unless the microcontroller is in halt state.

2.3.10. WRITE_PC

`write_pc` sets the program counter of the microcontroller. It can only be issued if the microcontroller is in halt state.

2.3.11. CPUSTATE

`cpustate` returns a list or the current state of the microcontroller.

Arguments:

––all, –a Return a TCL list of all recent state transitions. Each list element is in itself a list, containing the state (as string) and a timestamp.

––last, –l Return the current state as string.

––text, ––iso8601, –tReturn timestamps as ISO8601 strings

––numeric, –n Return timestamps as Unix time (number of seconds since 1970).

2.3.12. CONNECT

`connect` connects axsdb to the microcontroller, i.e. it causes axsdb to start controlling the microcontroller.

Arguments:

<unlockkeys>	Since the debug interface can potentially reveal sensitive information (such as the firmware), it can be protected from unauthorized use by a 64-bit access key. If the microcontroller is protected, then the key must be supplied to connect. If multiple keys are given, they are tried in sequence. If the microcontroller is unprotected (i.e. it has a key of 0xffffffffffffffff), then no key needs to be supplied.

2.3.13. DISCONNECT

`disconnect` disconnects axsdb from the microcontroller, i.e. the microcontroller is released to run on its own.

2.3.14. HWRESET

`hwreset` controls the RESET_N line from the debug interface to the microcontroller.

Arguments:

––pulse, –p	RESET_N is toggled low and then high again; the microcontroller is disconnected.
––off, –f	RESET_N is driven high (inactive)
––on, –o	RESET_N is driven low (active); the microcontroller is disconnected

2.3.15. RUN

`run` causes the microcontroller to start executing at the current PC value. Temporary breakpoint addresses may be given; these breakpoints will only be active during run and will be deleted as soon as the CPU stops.

Arguments:

––setaddr, –a <addr>	Set the breakpoint address
––symbol, –s <sym>	Set the breakpoint address to the address of the symbol <sym>. The symbol must be located in code address space.
––sourceline, –l <sl>	Set the breakpoint address to the source line <sl>

2.3.16. STOP

`stop` halts microcontroller instruction execution.

2.3.17. RESET

`reset` performs a (software) reset of the microcontroller.

2.3.18. STEP

`step` causes the microcontroller to execute the instruction at the current PC (or schedule an enabled interrupt), but halt again after executing one instruction.

2.3.19. STEPLINE

`stepline` causes the microcontroller to execute instructions until the current C language source line completes execution. It steps through function calls if embedded in the current C language source line. The debugger steps instructions, so execution is significantly slower than real-time.

2.3.20. STEPINTO

`stepinto` causes the microcontroller to execute the instructions until the PC leaves the current C language source line. Function calls stop the execution. The debugger steps instructions, so execution is significantly slower than real-time.

2.3.21. STEPOUT

`stepout` causes the microcontroller to complete execution of the current C language function. The debugger steps instructions, so execution is significantly slower than real-time.

2.3.22. WRITEBACK

In order to speed up operation of the debugger, axsdb contains caches of all memory of the microcontroller that can safely be cached. Consequently, `write_mem`, `load` and other commands only directly modify the caches. `writeback` causes the dirty caches to be written to the chip, for example the program loaded by `load`. `writeback` can only be issued if the microcontroller is in halt state.

2.3.23. BULKERASE

`bulkerase` causes the microcontroller to be safely erased. All FLASH content is lost.

Arguments:

––ignorecal, –i	Normally, if calibration data is available in the calibration sector, it is saved before the bulk erase and restored after the bulk erase. Specifying this option discards the calibration sector.
––keys, –k <unlockkeylist>	The old key to be used to access the calibration sector. Multiple keys may be given, in which case they are tried in sequence.

2.3.24. WRITEKEY

Since the debug interface allows access to sensitive information (like the firmware), it can be protected from unauthorized use by a 64-bit key. `writekey` writes the key into the microcontroller.

Arguments:

<unlockkey>	The key to be requested before granting debug interface access

2.3.25. WRITEPROTECT

The FLASH is organised as 64 1kByte sectors. FLASH contents can be protected with sector granularity. `writeprotect` protects the contents of a FLASH sector from overwriting. The only way to restore writes to protected sectors is by completely erasing the device by issuing a bulk erase.

Arguments:

<address> An address that lies within the sector to be protected

2.3.26. ERASEPROTECT

The FLASH is organised as 64 1kByte sectors. FLASH contents can be protected with sector granularity. `eraseprotect` protects the contents of a FLASH sector from erasing. The only way to restore erase functionality of protected sectors is by completely erasing the device by issuing a bulk erase.

Arguments:

<address> An address that lies within the sector to be protected

2.3.27. LOAD_MEM

`load_mem` reads a file containing binary code and/or debugging information into the debugger.

Arguments:

––debug, ––symbols, –d When used together with ––omf51, only load the symbolic debug information from the OMF51 file, and discard the binary code.

––omf51, –o Load an OMF51 format file

––hex, ––ihex, –i Load an Intel Hex format file

––cdb, –c Load a CDB format file

2.3.28. BREAKPOINT

`breakpoint` without argument returns a list of currently set breakpoints, their status (i.e. whether they are enabled or disabled), their count and their associated TCL script.

`breakpoint` with the following arguments manipulate the breakpoint list.

Arguments:

––disable, –d	Disable the breakpoint
––enable, –e <num>	Enable the breakpoint if <num> is nonzero or absent, disable otherwise
––setaddr, –a <addr>	Set the breakpoint address
––symbol, –s <sym>	Set the breakpoint address to the address of the symbol <sym>. The symbol must be located in code address space.
––sourceline, –l <sl>	Set the breakpoint address to the source line <sl>
––count, –c <count>	Ignore Breakpoint for <count> times before stopping the microcontroller
––script, –S <script>	Execute the TCL script <script> when hitting the breakpoint. The Script is executed in the global context.
––new, –n	Create a new breakpoint
––index, –i <nr>	Manipulate Breakpoint Number <nr>
––delete, –D <nr>	Delete Breakpoint Number <nr>

2.3.29. DISASS

`disass` disassembles one or multiple instructions and returns the result as a list (if one instruction is disassembled), or as a list of lists. Each instruction is described by the following list elements: the address (numeric), the opcode (as hex string), the symbol (with or without offset) closest to the address, the source line (with or without offset) closest to the address, and the disassembled instruction string.

Arguments:

––symbol, –s	The address argument is a symbol
––sourceline, –L	The address argument is a source line
––lines, –l <ln>	Disassemble <ln> instructions. If this argument is absent, disassemble just one.
<address>	The address argument. It must be numeric, unless –s or –L or their long forms is given, in which case it must be a string. If the address is omitted, the current PC is taken

2.3.30. MODULES

`modules` returns a list of the source code modules.

Arguments:

––asm, –a	Return the assembly modules. If absent, return the C modules

2.3.31. SOURCELINES

`sourcelines` returns a list of the source code lines.

Arguments:

––asm, –a	Return only assembly source lines
–c	Return only C source lines
<addr>	Return the source line that contains this address

2.3.32. SYMBOLS

`symbols` returns a list of the symbols.

2.3.33. QUIT

`quit` exits the debugger.

Arguments:

<exitcode>	Return with this exit code. Optional.

2.3.34. REGISTERS

`registers` returns a list of the chip registers.

2.3.35. CHIPS

`chips` sets or returns the currently selected chip(s)

Arguments:

––autodetect, –a	Autodetect the chip connected to the debugger.
––clear, –c	Clear the chip. This will clear the register list.
––set, –s	Manually set the chip
––all, –A	Returns all available chip models
––current, –C	Returns the currently selected chip(s).

2.3.36. PINEMUL

`pinemul` controls the pin emulation feature. While debugging, PB6 and PB7 are not available as GPIO, they are used by the debug interface. The pin emulation feature however still allows the GPIO state of the PB6 and PB7 pins to be read and controlled through the debugger software.

Arguments:

––script, –s	Set the script to be evaluated whenever the pin emulation state changes. The script may be deleted by setting it to an empty string.
––getscript, –g	Return the script that is evaluated whenever the pin emulation state changes.
––set-b6	Set the PB6 drive value
––clear-b6	Set the PB6 drive value to zero
––set-b7	Set the PB7 drive value
––clear-b7	Set the PB7 drive value to zero
––enable	Enable the pin emulation feature
––disable	Disable the pin emulation feature

Return value:

Unless the script is set or requested, `pinemul` returns a list with the following seven entries:

PORTB.6, PORTB.7, DIRB.6, DIRB.7, Debugger Drive PB6, Debugger Drive PB7, Enable

2.3.37. CPUTRACE

`cputrace` returns the CPU trace buffer.

Arguments:

––length, –l	Set or return the length of the trace buffer.

Return value:

If ––length is given, `cputrace` returns the length of the trace buffer, otherwise it returns the trace buffer entries accumulated since the last call to `cputrace`.

2.3.38. PROFILE

`profile` controls the profiler.

Arguments:

––disable, –d	Disable the profiler.
–c	Enable profiling of C source lines
––asm, –a	Enable profiling of assembly source lines

Return value:

If no argument is given, `profile` returns and clears the accumulated profile buffer.

2.4. VARIABLES

2.4.1. COMPILERVENDOR

There is no standard Application Binary Interface (ABI) in the 8052 ecosystem. Different compiler use different representations of data elements, especially “generic” pointers (pointers containing an address space tag in addition to the actual address). In order for the debugger to be able to access symbolic information, it needs to know which compiler generated the code in question.

Keil is selected by default, unless a cdb file is loaded, in which case the default is sdcc.

Valid values:

sdcc	Small Devices C Compiler (http://sdcc.sourceforge.net)
keil	Keil (http://www.keil.com/)
iar	IARSystems (http://www.iar.se)
wickenhaeuser	Wickenhäuser (http://www.wickenhaeuser.de/)

noice NoICE

2.5. TCL I/O Channels

AXSDB provides two TCL I/O Channels.

2.5.1. cpustat

Reading a single character from `cpustat` returns the state of the microprocessor. The channel issues a read event if the microprocessor status changes. The channel is not writeable.

2.5.2. dbglink

`dbglink` is the interface to the microprocessor DebugLink UART. Characters written to `dbglink` can be read by the microprocessor from the DebugLink UART, while characters written by the microprocessor to the DebugLink UART are returned to the TCL script via the `dbglink` channel.

2.6. Convenience Commands

Convenience Commands are defined in axsdb.tcl and implemented as TCL procedures.

2.6.1. inforeg

`inforeg` prints the most important microprocessor registers and the current instruction.

2.6.2. ir

`ir` stops the microprocessor, and then prints the same information as `inforeg`.

2.6.3. SR

`ir` prints the same information as `inforeg`, then steps the microprocessor, and then prints the same information as `inforeg`.

2.6.4. RI

`ri` stands for "run interactive". `ri` first prints the microprocessor state (same as `inforeg`), then runs the microprocessor. After that, `ri` implements a simple terminal program. Key presses are sent to the DebugLink UART on the processor, while characters the microprocessor transmits on the DebugLink UART are printed on the screen. The terminal terminates if the microprocessor hits a breakpoint, or CTRL-A or CTRL-C is pressed. CTRL-C halts the microprocessor, while CTRL-A keeps it running. At the end, `ri` prints the new microprocessor state (same as `inforeg`)

2.6.5. ALOAD

`aload` is a convenience `load_mem` wrapper. It determines file types from the file extensions, and autoloads an sdb file if one is found with the same base filename.

2.6.6. RLOAD

`rload` is a convenience `aload` wrapper. It stops the microcontroller, then resets it and calls `aload` with the given argument.

2.6.7. BERASE

`berase` is a convenience `bulkerase` wrapper. It starts the bulk erase, waits until it finishes (or times out), stops and resets the processor. It returns either "done" or "failed".

2.6.8. WAITCPUSTATE

`waitcpustate` waits until the CPU state matches the supplied glob-like pattern. See the description of the TCL `string match` command for a description of the pattern syntax.

2.6.9. WAITCPUSTOPPED

`waitcpustopped` waits until the CPU is stopped.

2.6.10. WAITCPURUNNING

`waitcpurunning` waits until the CPU is running.

3. Command Line FLASH Programming

Besides the TCL scriptable command interpreter, AXSDB also provides command line parameters to easily program the FLASH from a script. This can be useful for production programming.

The basic command to program the microcontroller FLASH memory is as follows:

```
axsdb.exe --oldkeys key --newkey key --flashprog file
```

file is the file name (including the path) to the file containing the microcontroller code. It may either be an Intel Hex file (extension .hex), an OMF-51 file (extension .omf), or an UBROF 10 file (extension .ubr). The file is usually located in the bin\Release subdirectory of the AxCodeBlocks project. If using SDCC, either the .hex or the .omf file may be used interchangeably. If using IAR ICC, then only the .ubr file is generated.

key is a 64 bit hexadecimal number (format 0x0123456789abcdef). This option locks the debug interface to unauthorized access. After this command succeeds, the debug interface may no longer be accessed unless the key number is known. It is strongly recommended that customer chooses a random number for key and keeps it secret.

The command returns success / failure status as exit code. The exit code is stored in the pseudo variable %errorlevel%. It is 0 on success and 1 on failure.

Another useful command is the following, which sends a reset pulse to the microcontroller:

```
axsdb.exe --hwreset
```

If it is desired to reset the key of a locked microcontroller, the following command can be used:

```
axsdb.exe --oldkeys key --newkey 0xffffffffffffffff --flashprog file
```

It is important that whenever the flash is programmed, `--oldkeys key1,key2...` is given with all possible keys the microcontroller could be locked with. Otherwise, calibration data is lost.

PUBLICATION ORDERING INFORMATION
LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303-675-2175 or 800-344-3860 Toll Free USA/Canada
Fax: 303-675-2176 or 800-344-3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800-282-9855 Toll Free USA/Canada.
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910
Japan Customer Focus Center
Phone: 81-3-5817-1050

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative

ON Semiconductor Website: **www.onsemi.com**