

User Manual

AxCode::Blocks



ON Semiconductor®

TABLE OF CONTENTS

TABLE OF CONTENTS	2
1. Introduction	4
2. Installing AxCode::Blocks	7
3. Connecting the Hardware	8
4. Creating a New Project	9
4.1. AX8052	9
4.2. AXM0F243	12
5. Adding and Editing Files	17
6. Compiling the Project	18
7. Programming and Debugging the Project	19
7.1. Programming the project	19
7.2. Debugging the project	19
7.3. OPEN OCD Commands	22
8. Debugging Windows	25
8.1. Breakpoints	25
8.2. CPU Registers	25
8.3. Disassembly	28
8.4. Memory dump	29
8.5. Watches	30
8.6. Pin Emulation	30
8.7. Debug link	31
8.7.1. General Serial terminal	31
8.7.2. Windows power shell as Serial terminal	32
8.8. Call stack	34
8.9. Running threads	34
9. Advanced Debugger Configuration	35
9.1. AX8052	35
9.2. AXM0F243	37
10. Onsemi Project Wizard	38
10.1. AX8052	38
10.2. AXM0F243	40
11. Troubleshooting Guide	41
11.1. Compiler Auto-detection fails on first start	41
11.2. Remove all saved User Settings	46
11.3. SDCC Project does not compile	46
11.4. IAR Project does not compile	50

11.5. GCC Project does not compile	53
11.6. Project compiles, but debugging does not work	55
12. History	65
13. Contact Information	66

1. INTRODUCTION

Figure 1 and 2 shows a diagram of the ON Semiconductor Development System Architecture.

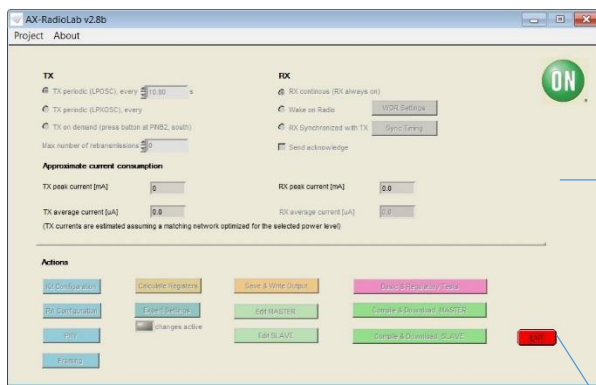
Radio Link parameters are set using the AX-RadioLAB GUI. AX-RadioLAB produces source code, compiles it and downloads it into the target board.

AxCode::Blocks is the graphical Integrated Development Environment (IDE) for AX8052 and AXM0F243 projects. It is a customized version of the popular Code::Blocks IDE. It can be used to further customize the AX-RadioLAB generated code, or it can be used to create new projects (such as those that do not involve a radio link).

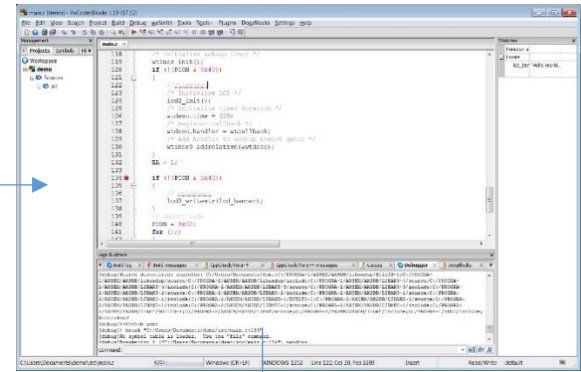
Both AX-RadioLAB and AxCode::Blocks talk to the ON Semiconductor Symbolic (command line) Debugger (AXSDB) for programming and debugging the AX8052 microcontroller. Normally, Users need not directly interact with AXSDB. AXSDB can however be useful for automated or scripted tasks, thanks to its command line and TCL scripting features.

For AXM0F243 microcontroller, OpenOCD debug interface software is used for programming and debugging using the GDB (GNU) debugger.

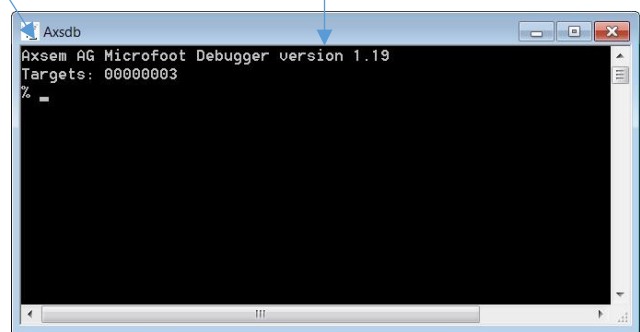
The Debug Adapter provides the link between the developer's workstation and the target board.



AX-RadioLab



AxCode::Blocks IDE



AXSDB Command Line Debugger

User Scripts



Target Board



Debug Adapter

Figure 1: ON Semiconductor Development System Architecture for AX8052

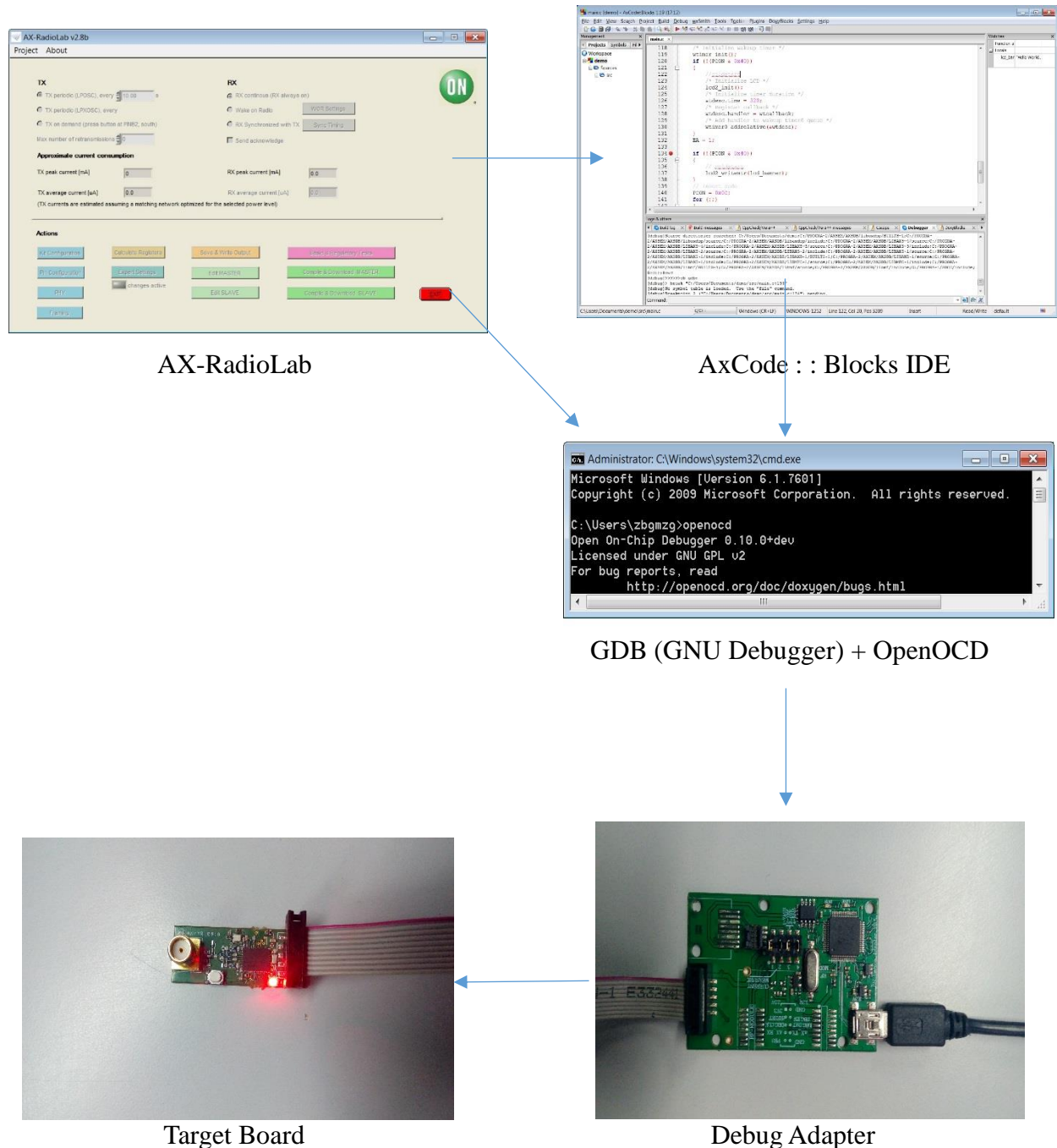


Figure 1: ON Semiconductor Development System Architecture for AXM0F243

This document should guide the reader through the installation of AxCode::Blocks and its use to create, compile and debug a little project.

AX-RadioLAB is documented in a separate document.

For general issues regarding Code::Blocks, please refer to its manual:

http://www.codeblocks.org/docs/manual_en.pdf

2. INSTALLING AxCode::BLOCKS

The AX-IDE installer contains everything you need: the SDCC compiler, GNU GCC Compiler for ARM, AX-RadioLab, debug adapter drivers, the AXSDB debugger, OpenOCD software, example files and libraries and, of course, AxCode::Blocks.

- Launch the installer.
- After accepting the terms of agreement you are asked to select the components to be installed. We strongly suggest to install all components. Failure to do so could result in missing links in your toolchain.
- Choose where to install AXSDB (it is recommended to keep the default settings). Hit *Install*.
- When asked if you want to install AxCode::Blocks too, click *Yes*: the corresponding setup wizard is started. This one is quite similar to the previous one: go through it.
- Next, you want to install the SDCC compiler and GNU GCC Compiler for ARM. Again, the installation is pretty intuitive. Be sure to tick the option for adding the compiler directory to the system path.
- When asked if you want to install AX-RadioLab, click *Yes*. At the end, you will be asked to reboot your computer. This is not necessary.
- Wait until the process finishes and you are done!

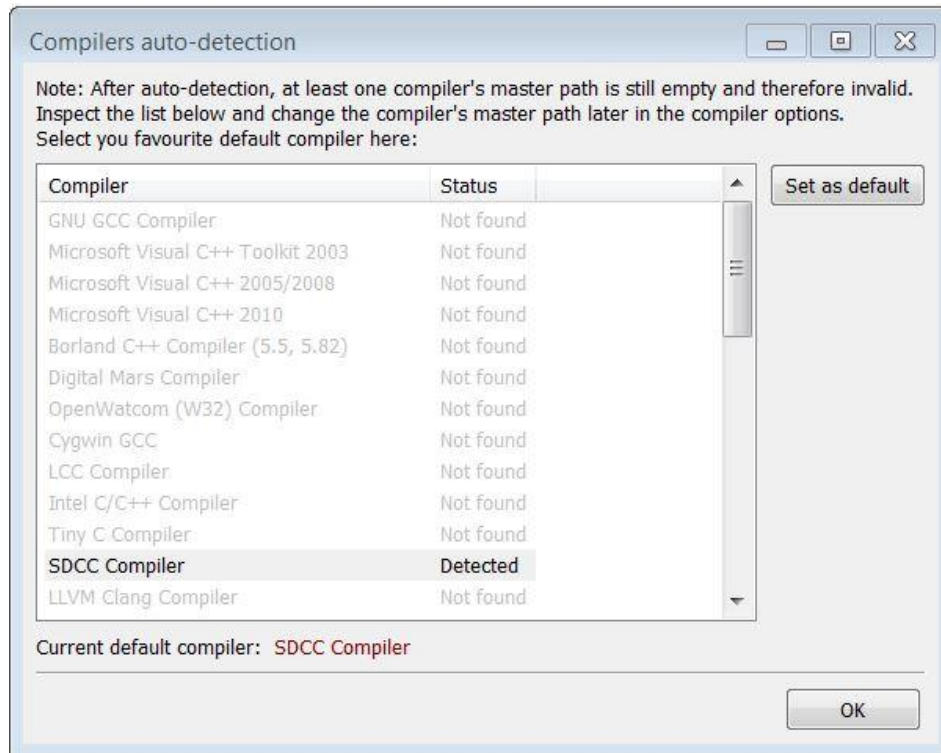
3. CONNECTING THE HARDWARE

Please refer to the application note available from the ON Semiconductor website:
<http://www.onsemi.com>

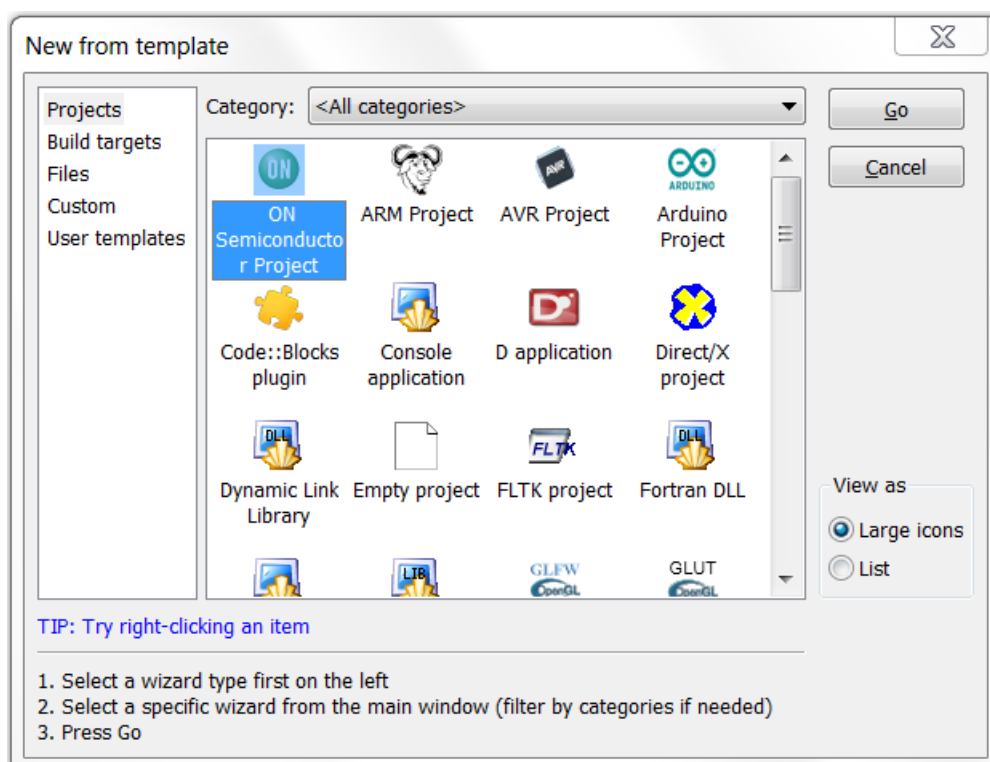
4. CREATING A NEW PROJECT

4.1. AX8052

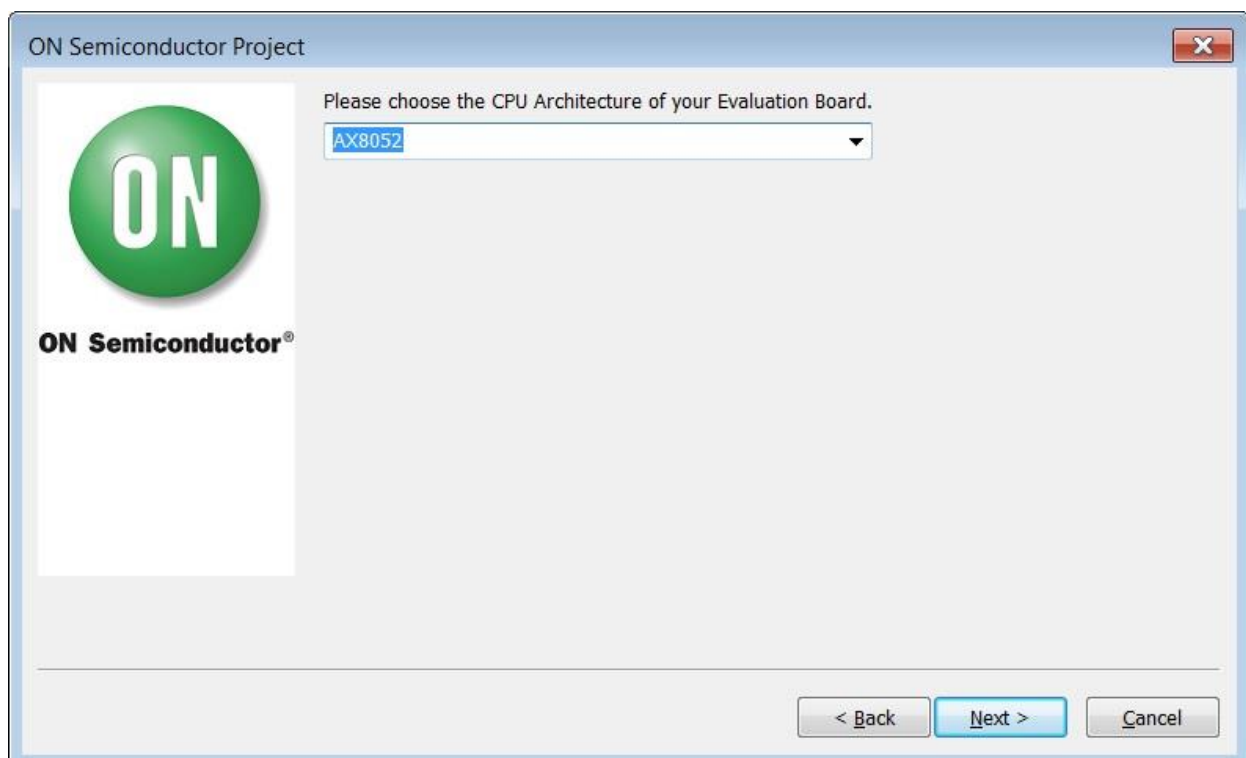
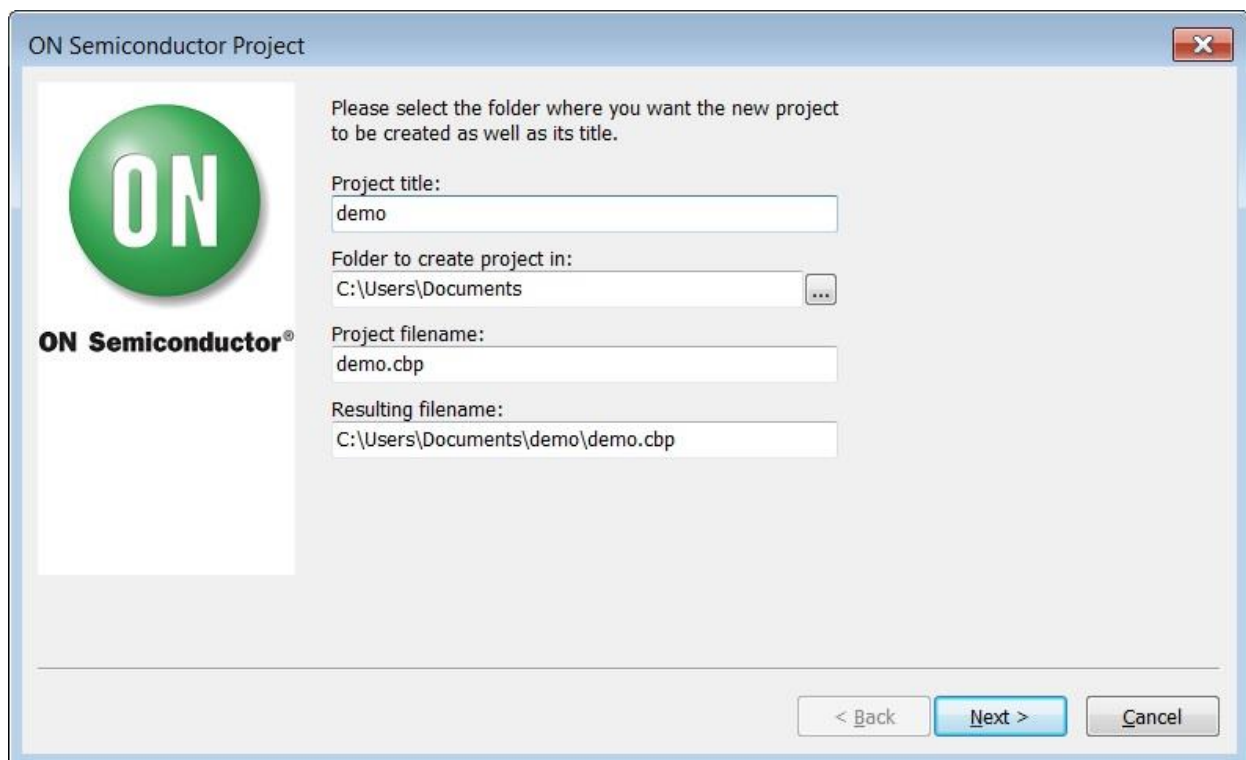
Start AxCode::Blocks. The first time AxCode::Block starts, it scans for installed compilers and presents a list of the compilers found. Select SDCC as default.

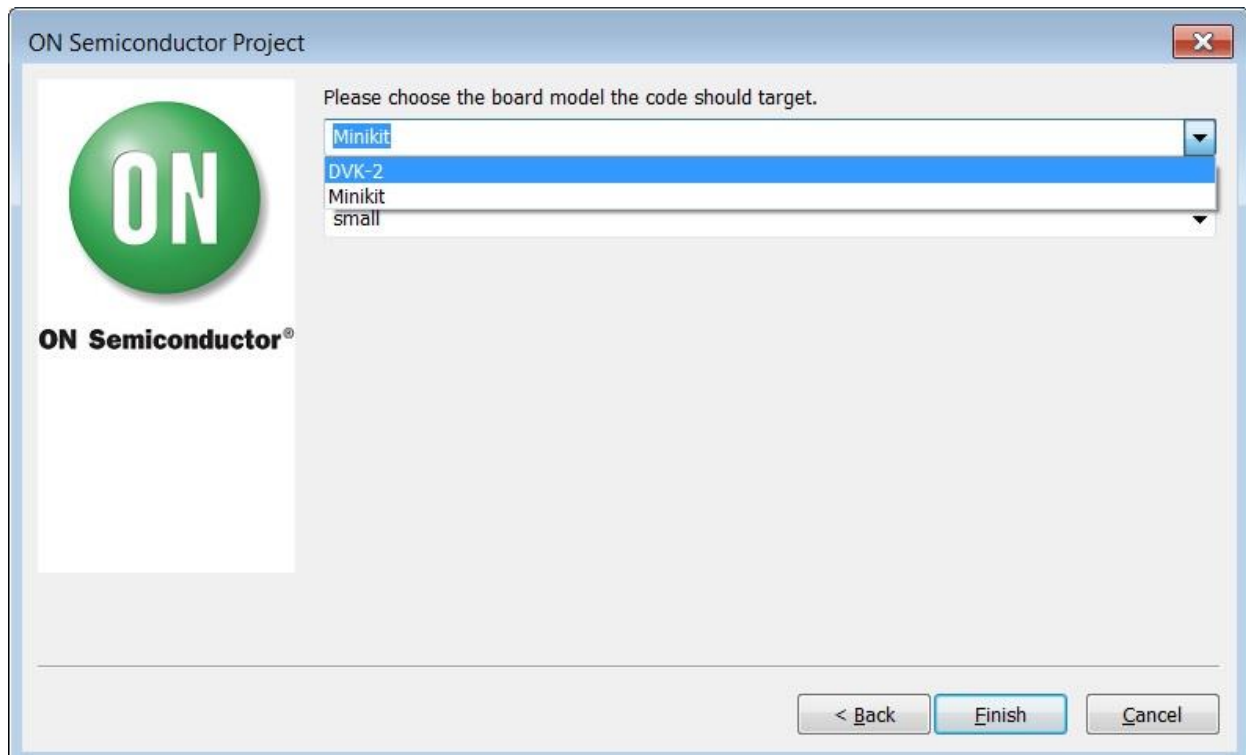
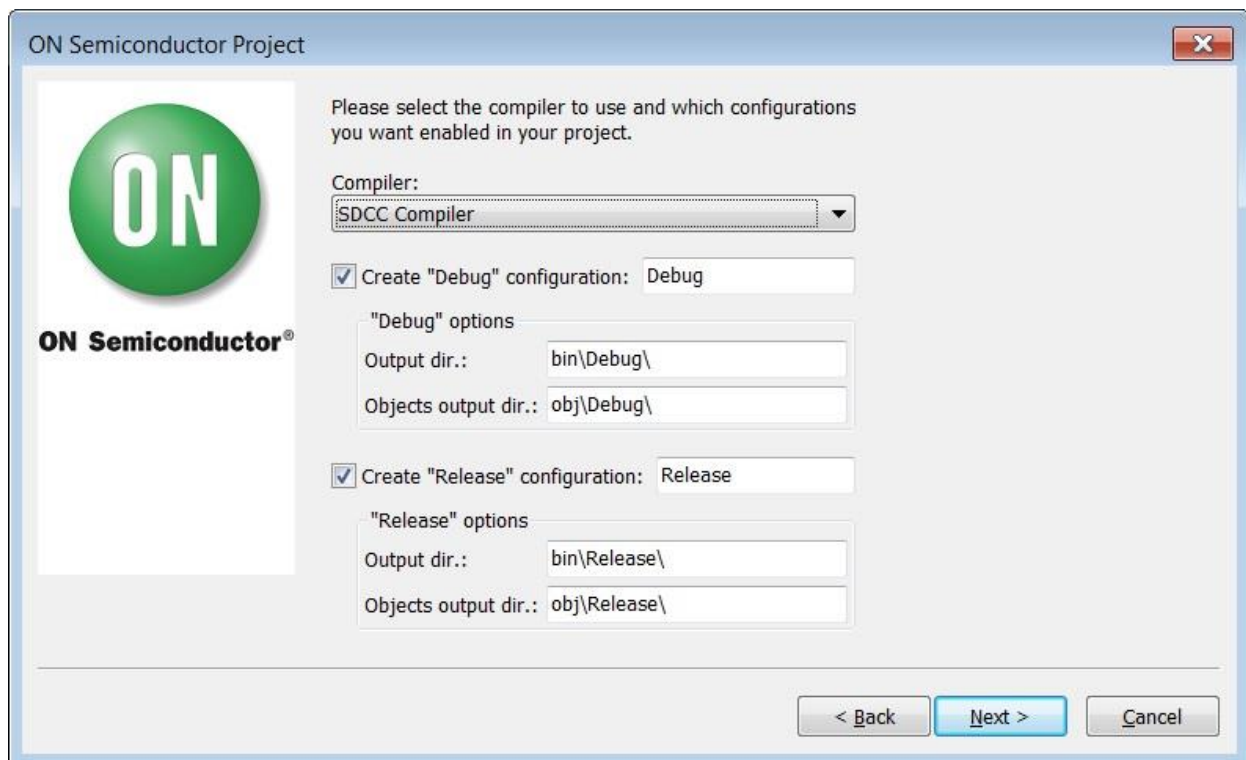


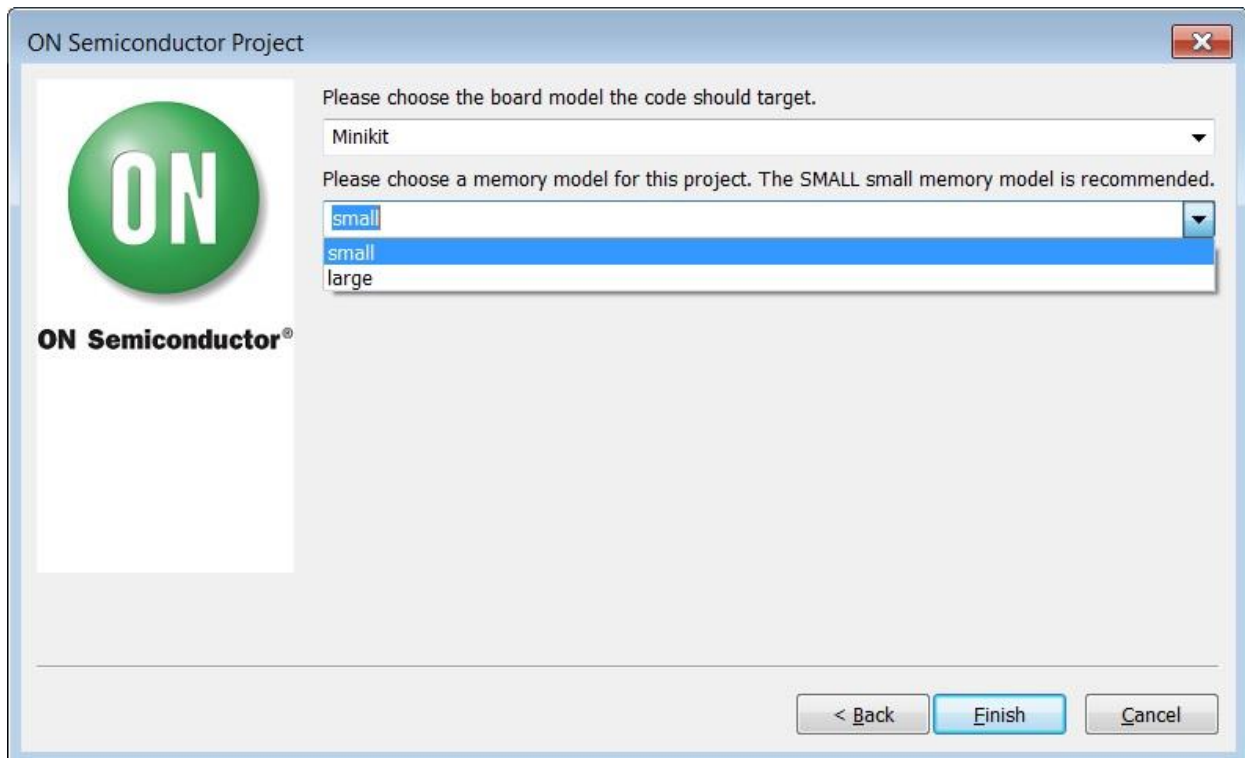
Click on *File* → *New* → *Project* and Choose ON Semiconductor Project:



A dialog will pop-up. Go through it and change the settings if needed. The following screenshots are intended as examples.



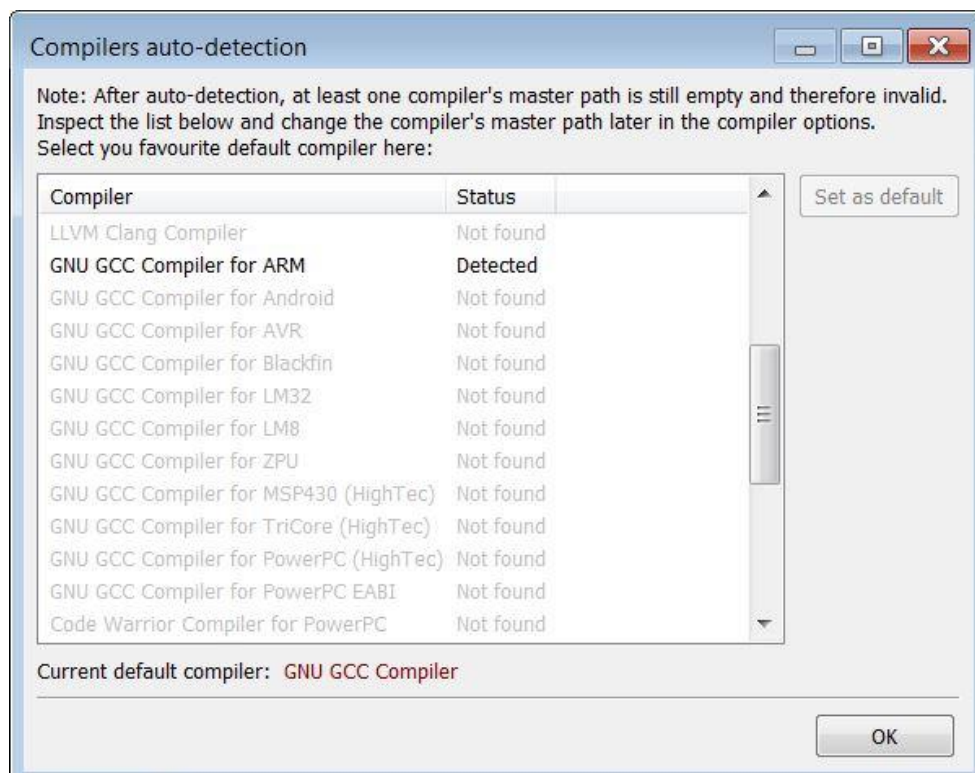




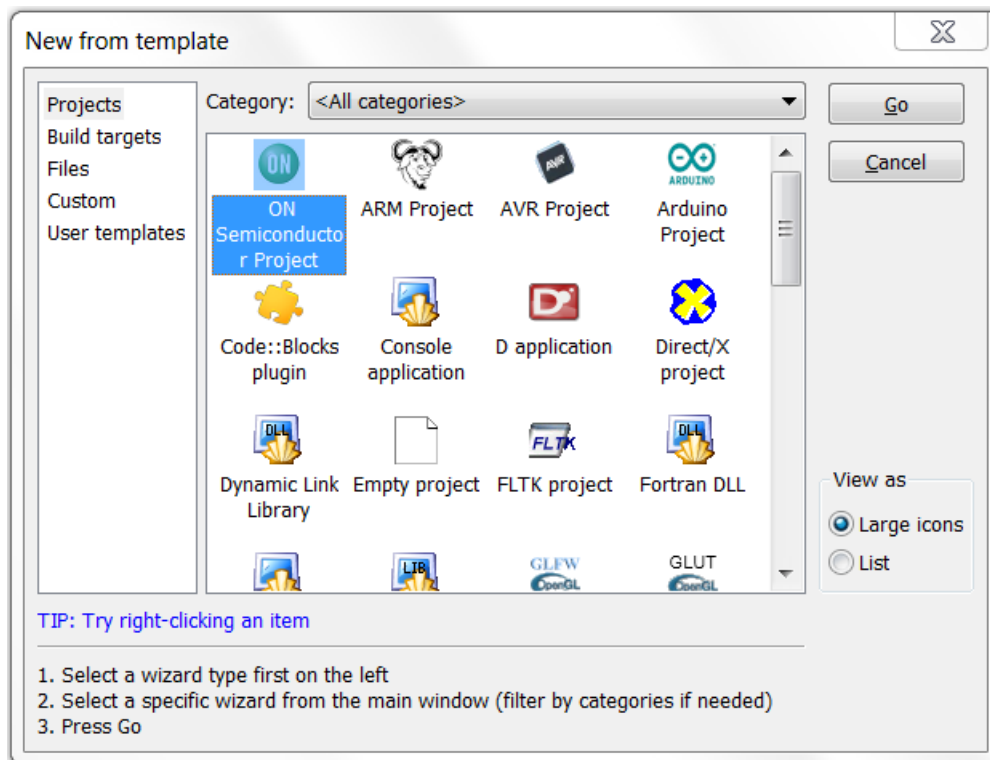
Clicking on *Finish* will create the new project. For your convenience the most important build options and compiler preferences are set automatically.

4.2. AXM0F243

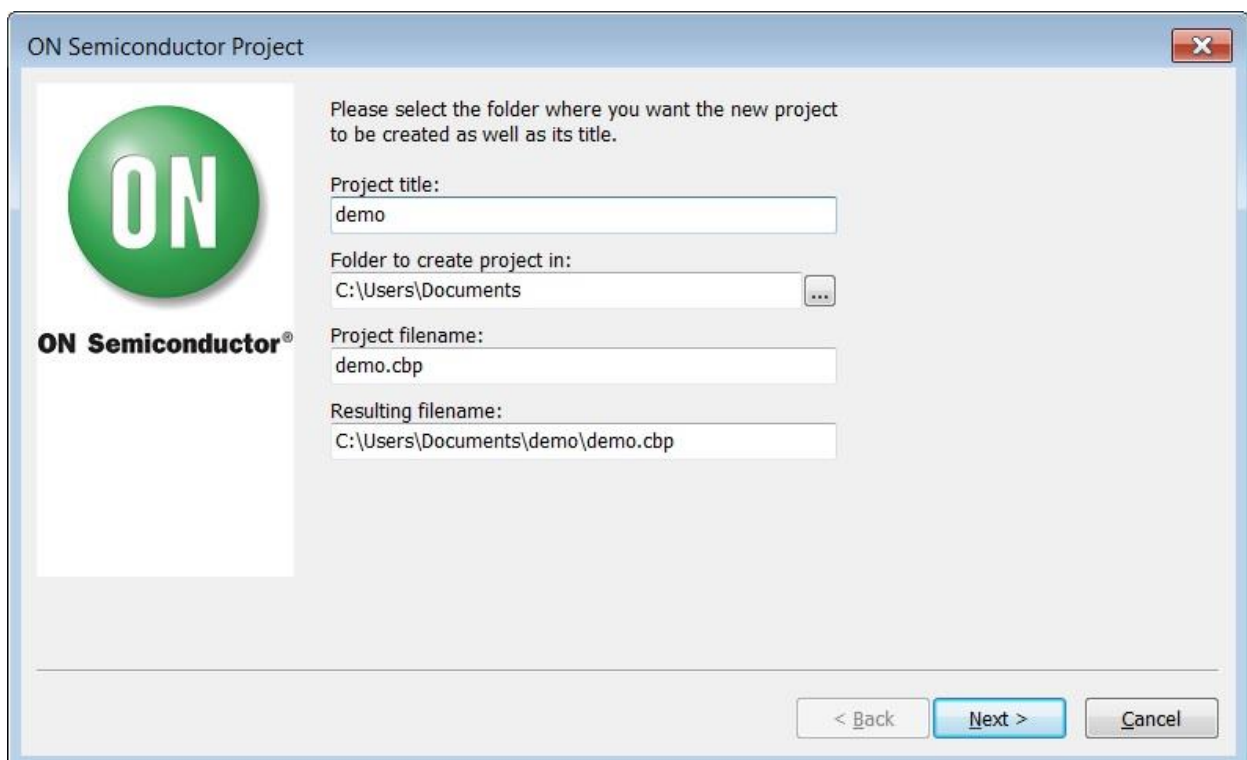
Start AxCode::Blocks. The first time AxCode::Block starts, it scans for installed compilers and presents a list of the compilers found. Select GNU GCC Compiler for ARM as default.

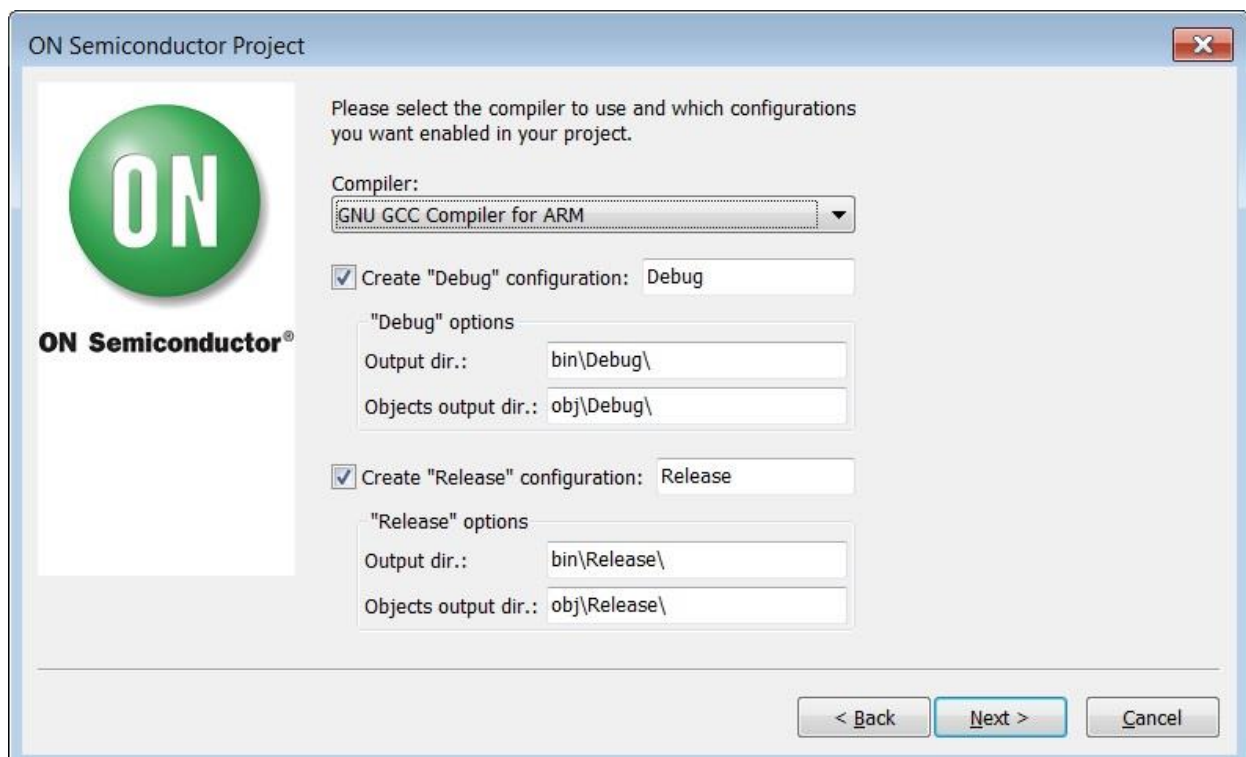
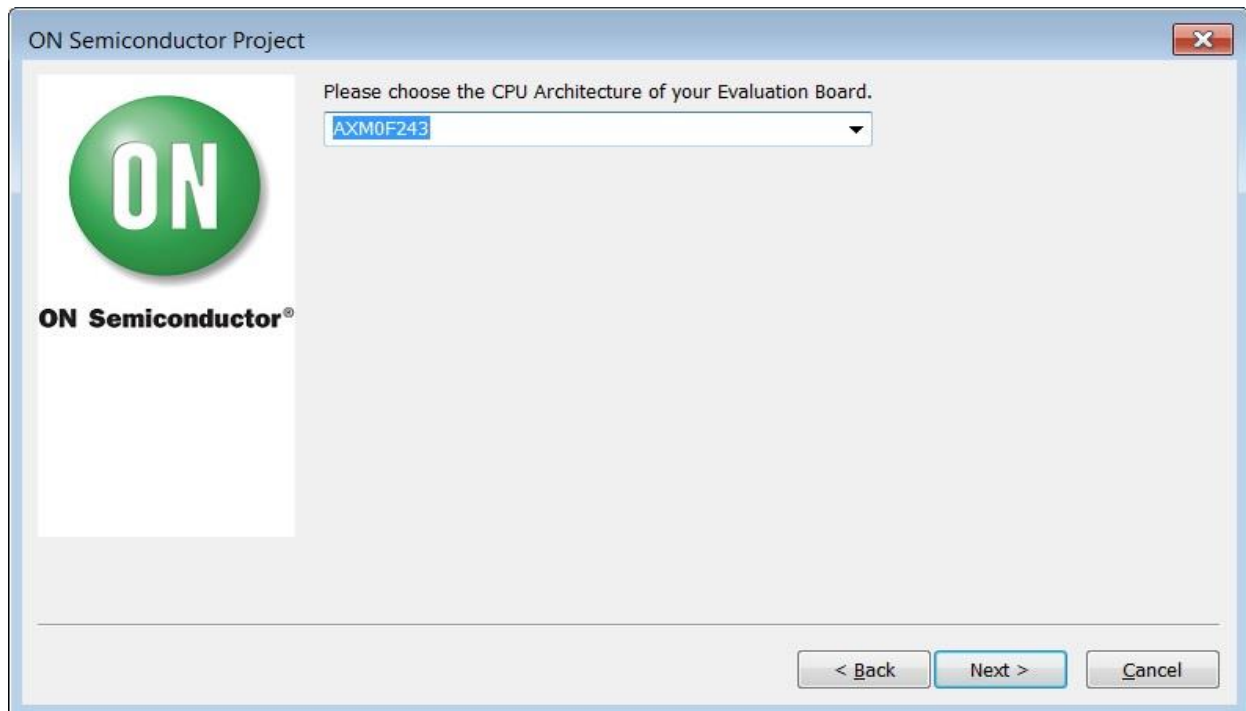


Click on *File* → *New* → *Project* and Choose ON Semiconductor *Project*:

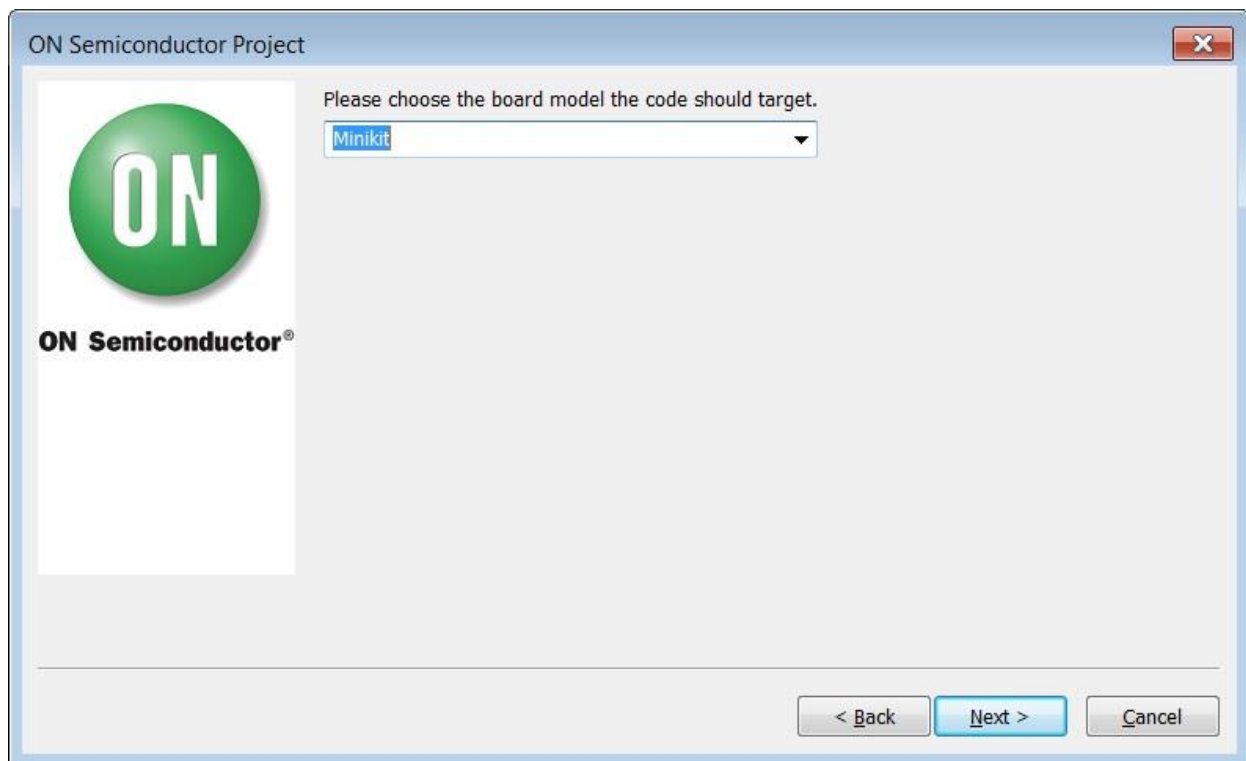


A dialog will pop-up. Go through it and change the settings if needed. The following screenshots are intended as examples.

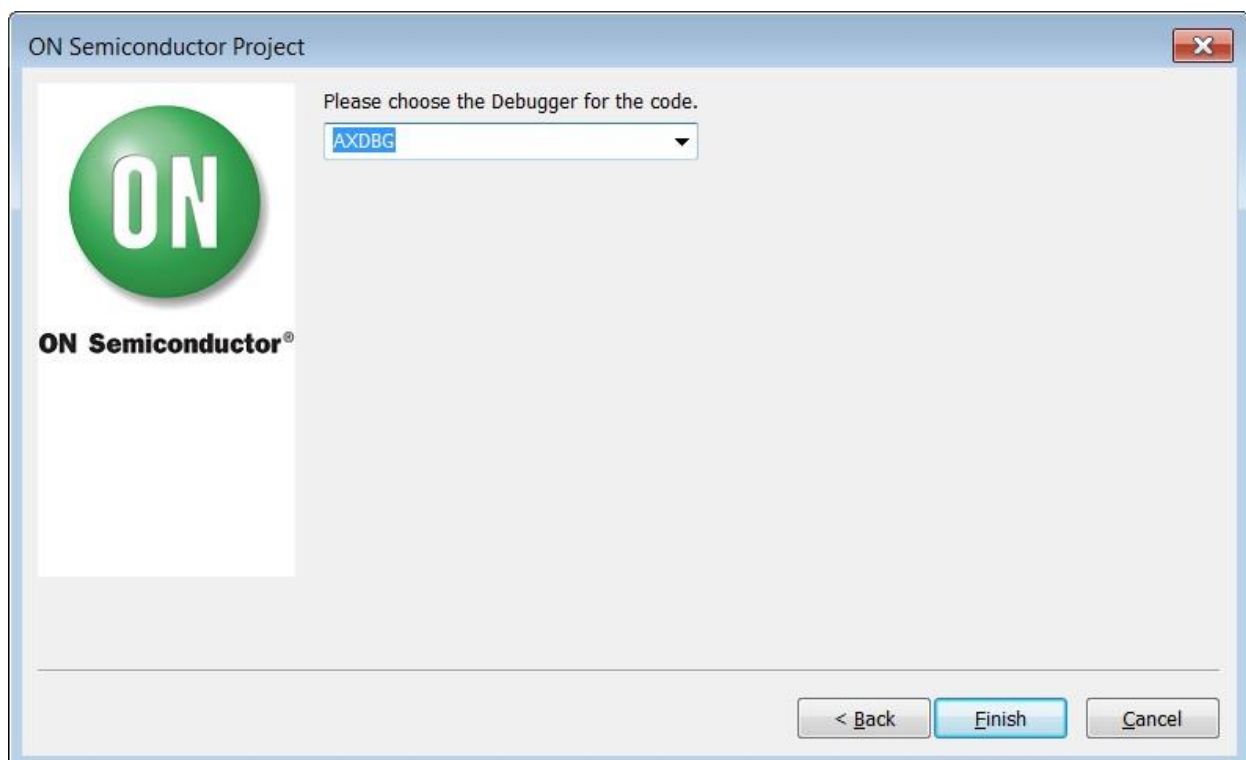




Choose the board model as *Minikit* for AXM0F243

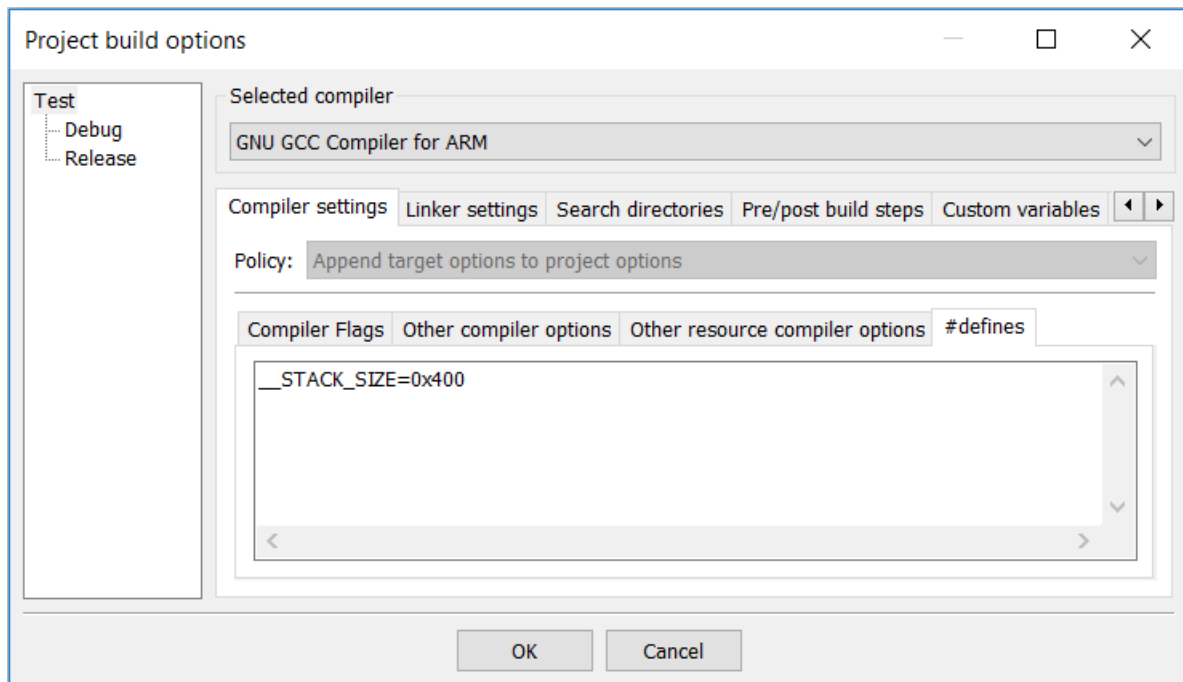


Choose the Debugger as *AXDBG*



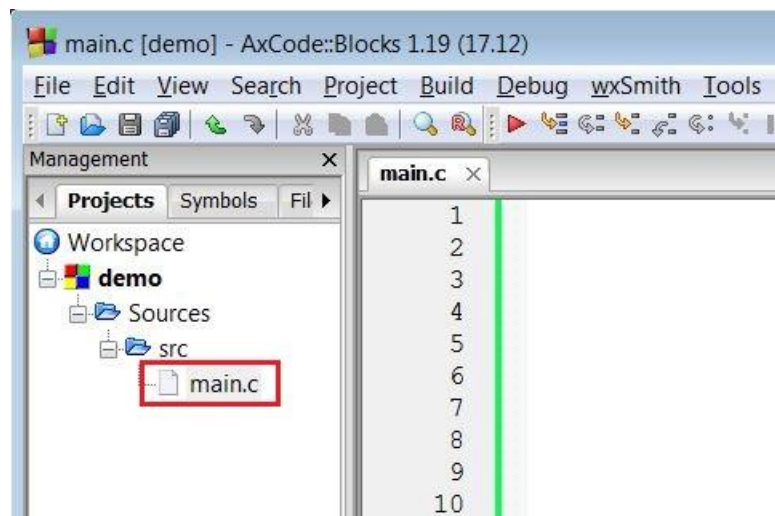
Clicking on *Finish* will create the new project. For your convenience the most important build options and compiler preferences are set automatically.

After creating the project, user has the option to modify the stack size for AXM0F243 firmware using the macro “**__STACK_SIZE**” which is defined in “Project build options” window under compiler settings.



5. ADDING AND EDITING FILES

An example source file has been included. You can open it by double-clicking its name on the project tree:



You can add existing files to the project using the menu entry *Project* → *Add Files*.
You can add new files to the project using the menu entry *File* → *New* → *File*.

Open files are shown on the right pane and can be edited.

6. COMPILING THE PROJECT

The most important function can be accessed through the compiler toolbar:

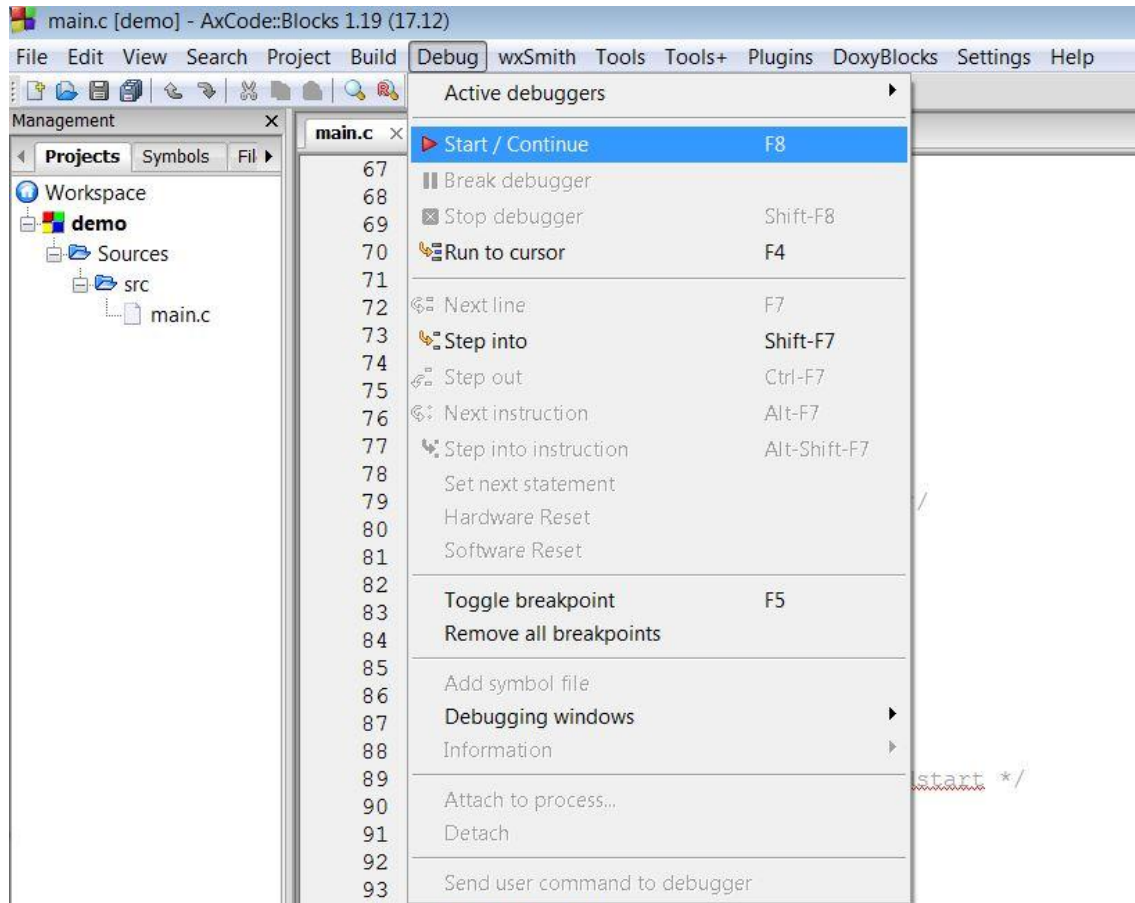


- | | | |
|---|--------------|---|
| ① | <i>Build</i> | Compile and link the project |
| ② | Rebuild | Delete existing files and build |
| ③ | Abort | Stop the building process |
| ④ | Build target | The Debug target generates automatically debug informations |

7. PROGRAMMING AND DEBUGGING THE PROJECT

7.1. PROGRAMMING THE PROJECT

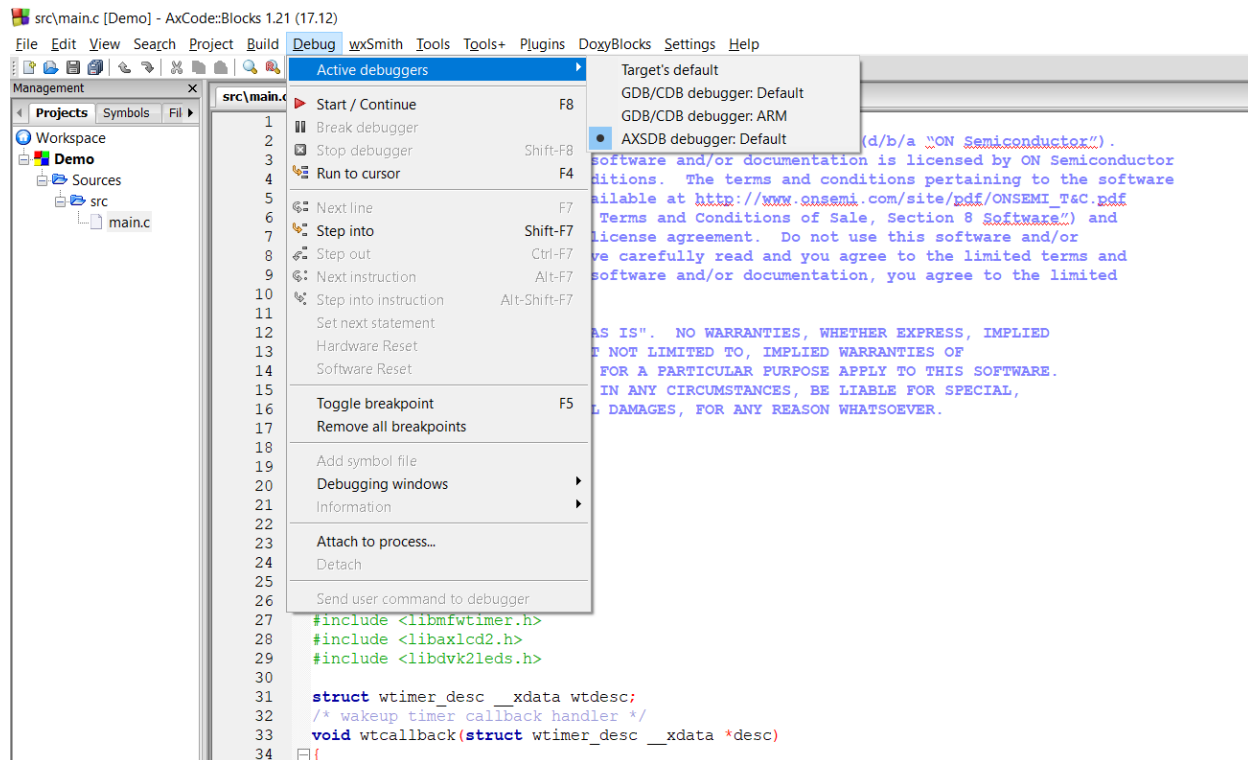
For just programming the device, choose **Start / Continue** option under *Debug* menu as shown in the image below or hit the option ① in the debugger tool bar.



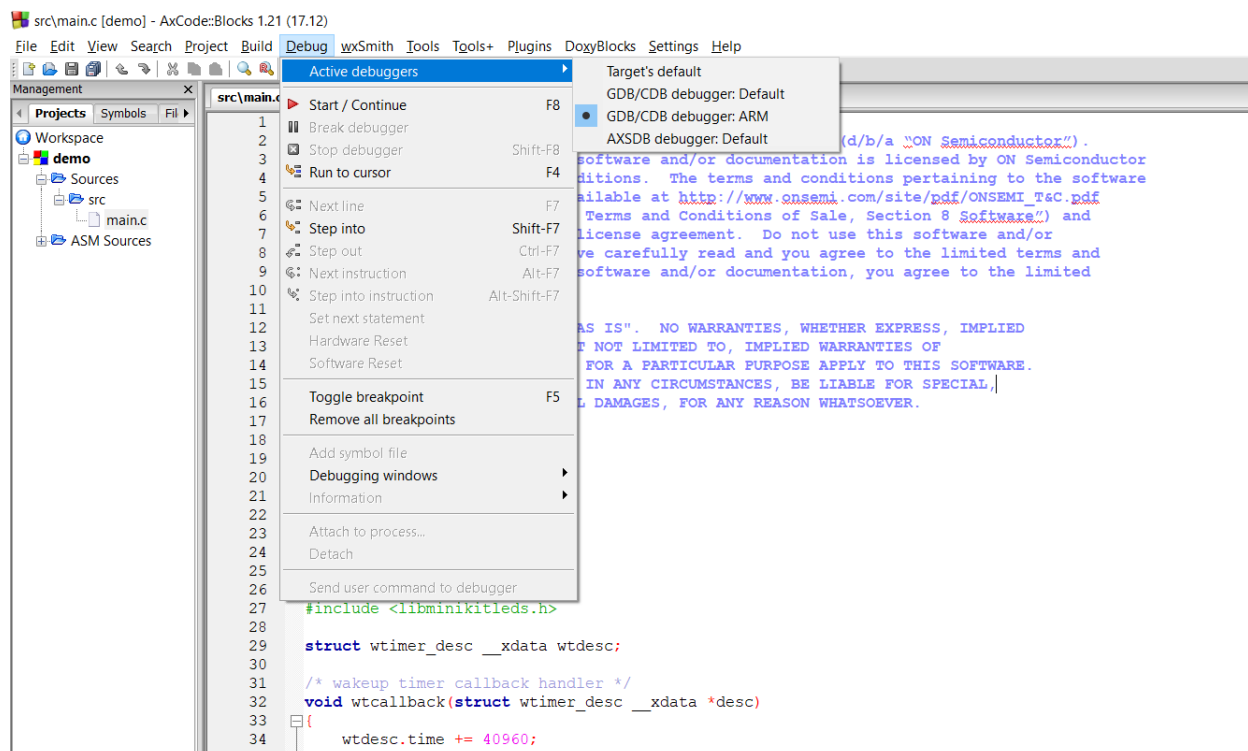
7.2. DEBUGGING THE PROJECT

Select the Debugger under *Debug* → *Active debuggers* and make sure the corresponding toolbar is visible (*View* → *Toolbars* → *Debugger*).

For AX8052, choose either *AXSDB debugger: Default* option or *Target's default* option under *Debug* → *Active debuggers* as shown below.



For AXM0F243, choose either *GDB/CDB debugger: ARM* option or *Target's default* option under *Debug* → *Active debuggers* as shown below.



Before the debugging process is started, the toolbar looks like this:



By hitting ① the debugger is started. If no devices are found, an error message is issued¹. If exactly one device is found, the device is automatically connected.

If changes are made to the project since the last build, the project is automatically compiled.

When debugging is started and once the device is programmed, the toolbar changes its appearance to



The pause button ② stops the execution of the program to examine its state. The cursor inside the editor is moved to the line corresponding to the current instruction. For AX8052, Button ③ does not stop the execution, but disconnects the device and exits the debugger. For AXM0F243, Button ③ stops the execution and exits the debugger. ④ and ⑤ are used to reset the microcontroller. Options ④ and ⑤ are available only for the AX8052 microcontroller.

When a breakpoint in the code is hit or after hitting the pause button ② other functions become enabled:



- | | | |
|---|-------------------------|--|
| ⑥ | <i>Run to cursor</i> | Execution is stopped on the selected line |
| ⑦ | <i>Next line</i> | Execute the next line in the source code |
| ⑧ | <i>Step into</i> | Execute the next line, if it's a function step into it |
| ⑨ | <i>Step out</i> | Continue execution until the end of the current frame |
| ⑩ | <i>Next instruction</i> | Execute the next assembly instruction |

Due to limitations in the debug information of 8052 compilers, the stepping commands ⑦, ⑧ and ⑨ require the microcontroller to be single-stepped. It can therefore take a long time until these commands terminate. It is always possible to stop one of these commands

¹ If you get an error message although a device is connected, you likely are missing the drivers. Open the *Control Panel* and navigate to the *Device Manager*. Find the unrecognized devices (look for the exclamation marks) named "USB Serial Converter A" or "USB Serial Converter B" or "USB Composite Device", right-click them and choose to install or update the driver. Do not search online for the driver. As search directory give the directory where you installed AXSDB followed by "\ftdi", e.g. "C:\Program Files (x86)\ON Semiconductor\AXSDB\ftdi". Eventually, you need to disconnect the device and restart AxCode::Blocks.

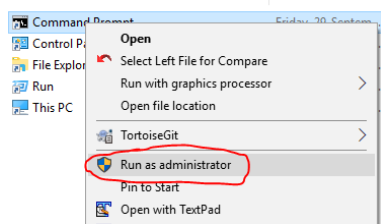
prematurely by hitting the pause button ② Usually, it is preferable to set breakpoints or use the Run-To-Cursor feature over the stepping commands.

The drop-down menu ⑪ can be used to open the debugging windows described in the next chapter.

7.3. OPEN OCD COMMANDS

This section will outline on how to start GDB server, connect device and execute Open OCD commands on the device. For AXM0F243 microcontroller, OpenOCD debug interface software is used for programming and debugging using the GDB (GNU) debugger.

1. Open a command prompt with 'Run as administrator'



2. Go to working directory

```
$ cd "C:\Program Files (x86)\ON Semiconductor\AXSDB\bin"
```

3. Start gdb sever

```
$ "C:\Program Files (x86)\GNU Tools ARM Embedded\5.4 2016q3\bin\arm-none-eabi-gdb.exe"
```

```
C:\Program Files (x86)\ON Semiconductor\AXSDB\bin>"C:\Program Files (x86)\GNU Tools ARM Embedded\5.4 2016q3\bin\arm-none-eabi-gdb.exe"
GNU gdb (GNU Tools for ARM Embedded Processors) 7.10.1.20160923-cvs
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb)
```

4. Launch open OCD

```
(gdb) target remote | openocd.exe -p -l openocd.log -f ../share/openocd/scripts/board/axm0f2_axdbg.cfg -d3
```

```
C:\Program Files (x86)\ON Semiconductor\AXSDB\bin>"C:\Program Files (x86)\GNU Tools ARM Embedded\5.4 2016q3\bin\arm-none-eabi-gdb.exe"
GNU gdb (GNU Tools for ARM Embedded Processors) 7.10.1.20160923-cvs
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target remote | openocd.exe -p -l openocd.log -f ../share/openocd/scripts/board/axm0f2_axdbg.cfg -d3
Remote debugging using | openocd.exe -p -l openocd.log -f ../share/openocd/scripts/board/axm0f2_axdbg.cfg -d3
Open On-Chip Debugger 0.10.0+dev
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
0x00004ca0 in ?? ()
(gdb)
```

5. Open OCD commands to control and monitor device and flash

Device commands

Device reset halt

(gdb) monitor reset halt

Device halt

(gdb) monitor halt

Device reset

(gdb) monitor reset

```
msp (/32): 0x20002000
(gdb) monitor reset
SWD DPIDR 0x0bc11477
(gdb) monitor reset halt
target halted due to debug-request, current mode: Thread
xPSR: 0xa1000000 pc: 0x10000040 msp: 0x20001fe8
pc (/32): 0x10000040
pc (/32): 0x00003354
msp (/32): 0x20002000
```

Flash commands

Flash Erase (Erase command is not supported as auto erase is enabled for the flash write)

(gdb) monitor flash erase_address <address> <length>

Flash Read

(gdb) monitor flash read_bank <bank_num> <file_name.bin> <address_offset> <length>

E.g.: monitor flash read_bank 0 filename.bin 0x0 0x100

Flash Write

(gdb) monitor flash write_bank <bank_num> <file_name.bin> <address_offset>

(gdb) monitor flash write_image <filename.elf>

E.g.: monitor flash write_image D:/test/MASTER_AXM0f2-GNU.elf

Flash verify

(gdb) monitor flash verify_bank <bank_num> <file_name.bin> <address_offset>

E.g.: monitor flash verify_bank 0 filename.bin 0x0

Flash info

(gdb) monitor flash info <bank_num>

Flash probe

(gdb) monitor flash probe <bank_num>

Flash banks list

(gdb) monitor flash banks

Flash fill

(gdb) monitor flash fillw <addr> <val> <len>

```
(gdb) monitor flash write_image D:/test/MASTER_AXM0F2-GNU.elf
Flash Download Started
Downloaded 4224 of 26850 bytes
Downloaded 8576 of 26850 bytes
Downloaded 13056 of 26850 bytes
Downloaded 17536 of 26850 bytes
Downloaded 22016 of 26850 bytes
Downloaded 26496 of 26850 bytes
Flash Download Completed
wrote 26850 bytes from file D:/test/MASTER_AXM0F2-GNU.elf in 6.176000s (4.246 KiB/s)
(gdb)
```

Note: As AXM0F243 has single flash bank, <bank_num> will always be 0 (zero).

More detailed description of the open OCD commands can be found at

<http://openocd.org/doc/pdf/openocd.pdf>

8. DEBUGGING WINDOWS

8.1. BREAKPOINTS

This window simply displays a list of the breakpoints set. In order to set a breakpoint, click on the left of the corresponding line in the editor or hit *F5*. The same procedure removes an already set breakpoint.

8.2. CPU REGISTERS

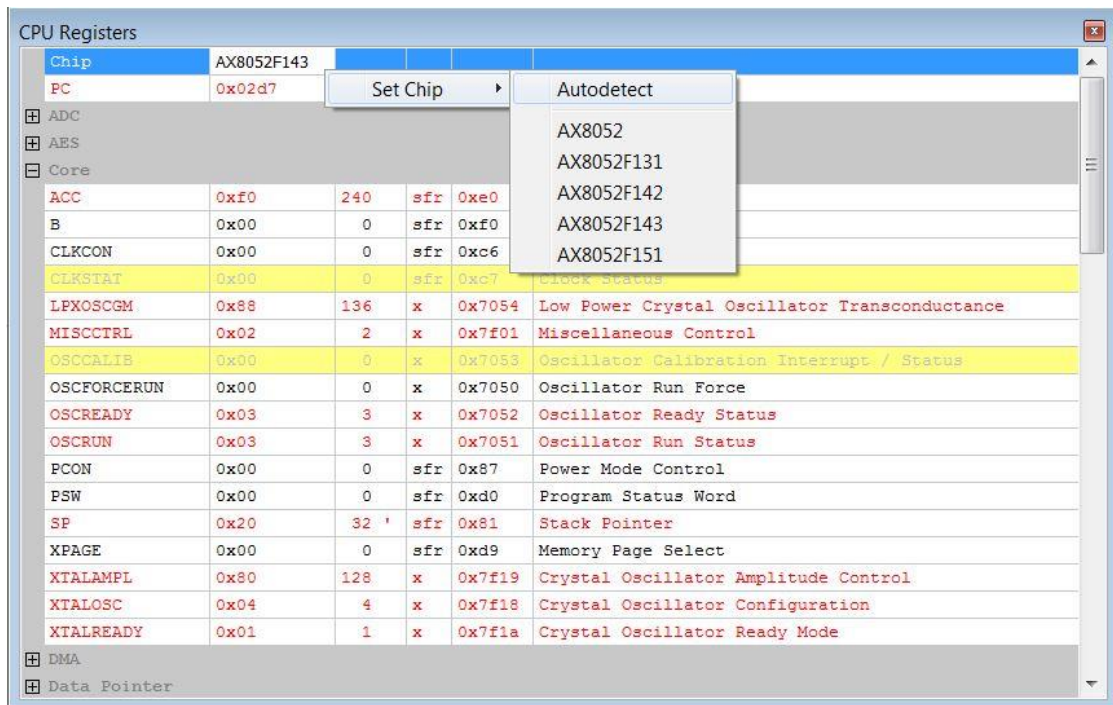
AX8052:

For Ax8052 microcontroller, this window shows the register contents while the microcontroller is stopped. Registers of radio Chips or SoC functions are also displayed.

Registers are grouped into sections. Light gray bars show the section name, the + or – sign to the left of the section name allows to show or hide the section.

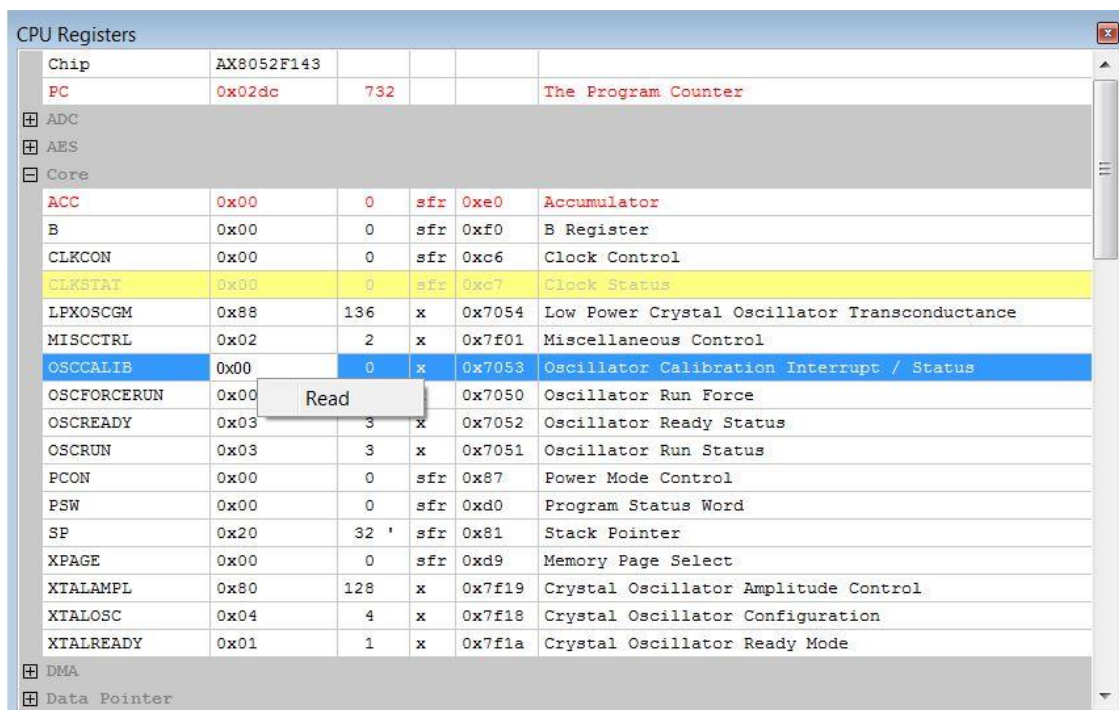
CPU Registers					
Chip	AX8052F143				
PC	0x02d7	727			The Program Counter
+ ADC					
+ AES					
+ Core					
ACC	0xf0	240	sfr	0xe0	Accumulator
B	0x00	0	sfr	0xf0	B Register
CLKCON	0x00	0	sfr	0xc6	Clock Control
CLKSTAT	0x00	0	sfr	0xc7	Clock Status
LPXOSCGM	0x88	136	x	0x7054	Low Power Crystal Oscillator Transcon
MISCCTRL	0x02	2	x	0x7f01	Miscellaneous Control
OSCCALIB	0x00	0	x	0x7053	Oscillator Calibration Interrupt / S
OSCFORCERUN	0x00	0	x	0x7050	Oscillator Run Force
OSCREADY	0x03	3	x	0x7052	Oscillator Ready Status
OSCRUN	0x03	3	x	0x7051	Oscillator Run Status
PCON	0x00	0	sfr	0x87	Power Mode Control
PSW	0x00	0	sfr	0xd0	Program Status Word
SP	0x20	32	sfr	0x81	Stack Pointer
XPAGE	0x00	0	sfr	0xd9	Memory Page Select
XTALAMPL	0x80	128	x	0x7f19	Crystal Oscillator Amplitude Control
XTALOSC	0x04	4	x	0x7f18	Crystal Oscillator Configuration
XTALREADY	0x01	1	x	0x7f1a	Crystal Oscillator Ready Mode

The first line of the dialog shows the name of the chip. The chip is normally auto-detected. Should auto-detection fail, the chip type can be manually set by right-clicking on the chip name.

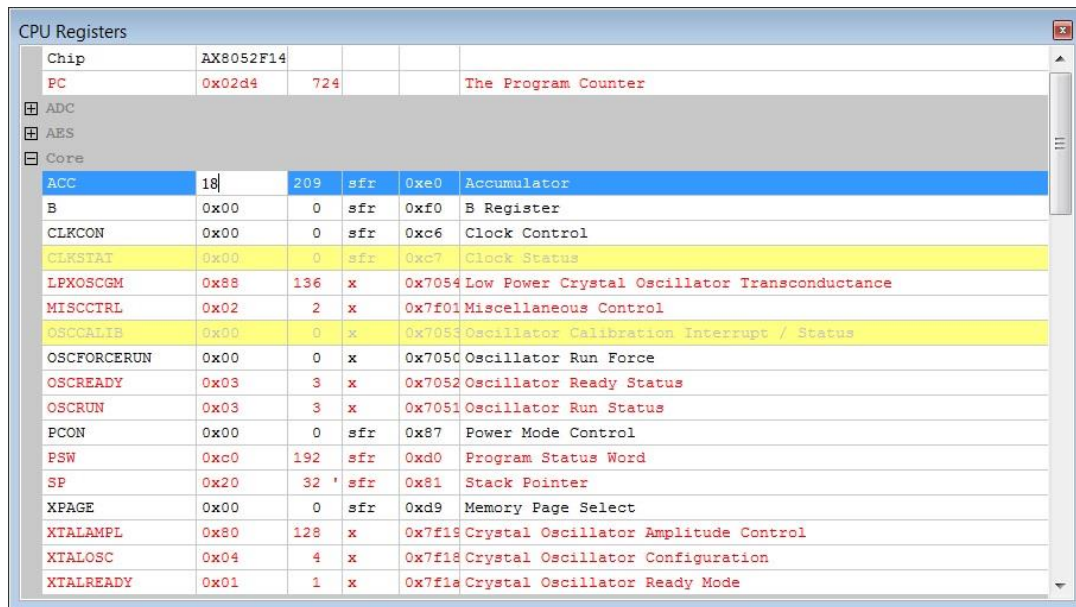


Registers and their contents are shown in the remaining rows. The first column shows the register name. The second column shows the current register contents as a hexadecimal number. The third column displays the register contents as a decimal number; if this number is greater or equal 32, it is also displayed as ASCII character. The fourth and fifth columns display the address space and the address of the register, while the sixth column displays a short description of the register.

Registers that have changed since the last processor break are displayed in red. Registers that can cause side effects when read are not automatically read. They are displayed with a light yellow background. Their values are also shown in light-gray, if they have not yet been read.



Registers can be (re-)read by either right-clicking into the row and selecting Read, or by selecting the row and pressing the space-bar.

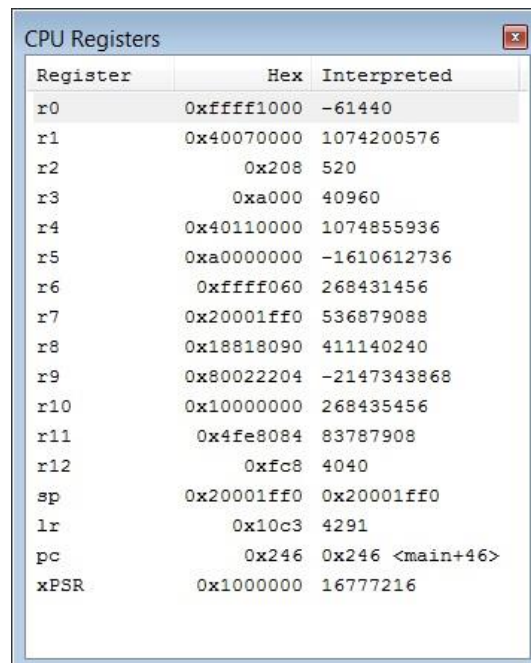


Register	Value	Address	Type	Description
Chip	AX8052F14			
PC	0x02d4	724		The Program Counter
ADC				
AES				
Core				
ACC	18	209	sfr	0xe0 Accumulator
B	0x00	0	sfr	0xf0 B Register
CLKCON	0x00	0	sfr	0xc6 Clock Control
CLKSTAT	0x00	0	sfr	0xc7 Clock Status
LPXOSCGM	0x88	136	x	0x7054 Low Power Crystal Oscillator Transconductance
MISCCCTRL	0x02	2	x	0x7f01 Miscellaneous Control
OSCCALIB	0x00	0	x	0x7053 Oscillator Calibration Interrupt / Status
OSCFORCERUN	0x00	0	x	0x7050 Oscillator Run Force
OSCREADY	0x03	3	x	0x7052 Oscillator Ready Status
OSCRUN	0x03	3	x	0x7051 Oscillator Run Status
PCON	0x00	0	sfr	0x87 Power Mode Control
PSW	0xc0	192	sfr	0xd0 Program Status Word
SP	0x20	32	sfr	0x81 Stack Pointer
XPAGE	0x00	0	sfr	0xd9 Memory Page Select
XTALAMPL	0x80	128	x	0x7f19 Crystal Oscillator Amplitude Control
XTALOSC	0x04	4	x	0x7f18 Crystal Oscillator Configuration
XTALREADY	0x01	1	x	0x7f1a Crystal Oscillator Ready Mode

Register values can be changed by clicking into the second column and entering a number. Numbers can be entered both in decimal format (eg. 18), as well as hexadecimal format (eg. 0x12).

AXM0F243:

For AXM0F243 microcontroller, this window shows the Processor core register contents while the microcontroller is stopped.



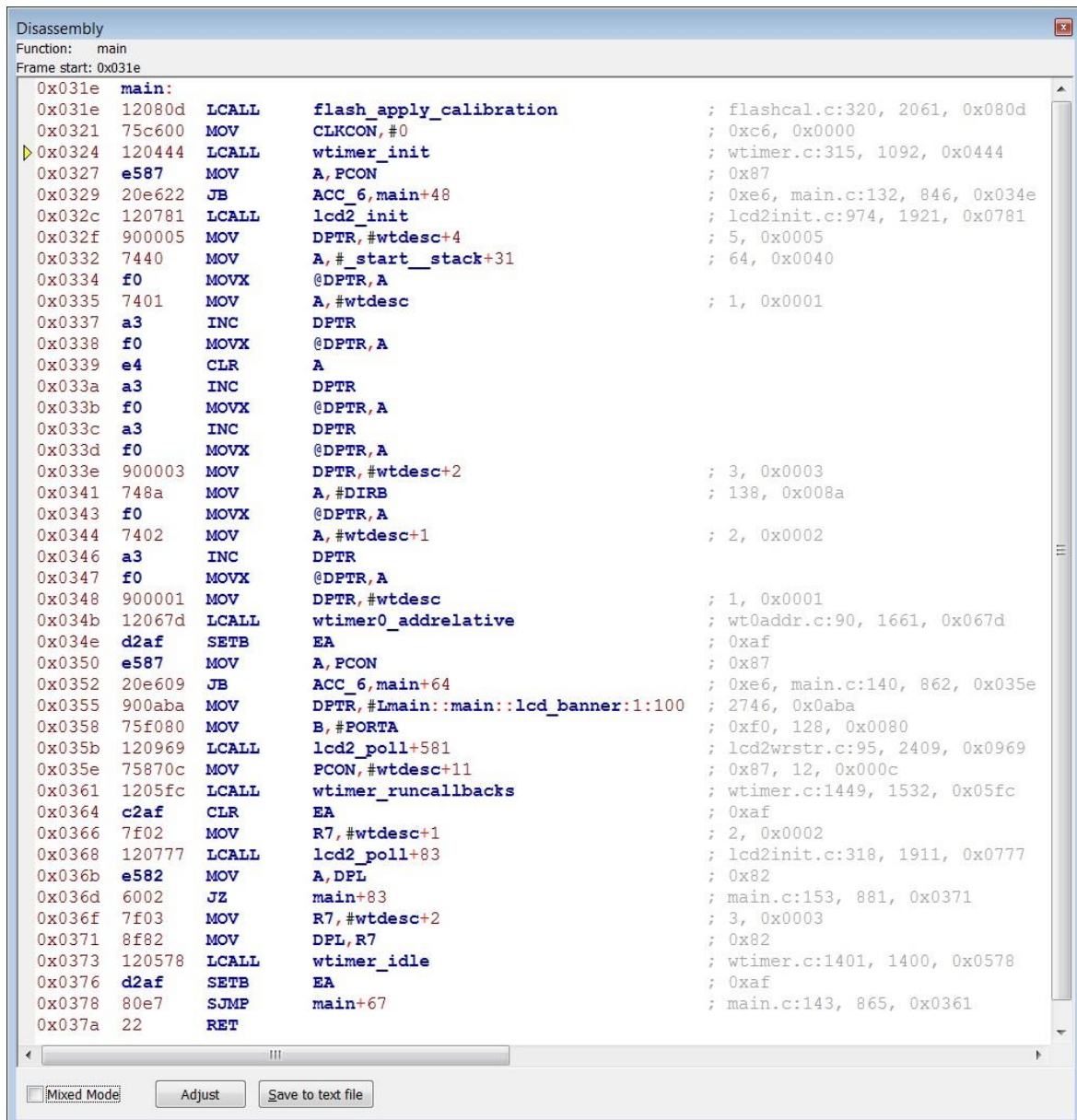
Register	Hex	Interpreted
r0	0xffff1000	-61440
r1	0x40070000	1074200576
r2	0x208	520
r3	0xa000	40960
r4	0x40110000	1074855936
r5	0xa0000000	-1610612736
r6	0xffff060	268431456
r7	0x20001ff0	536879088
r8	0x18818090	411140240
r9	0x80022204	-2147343868
r10	0x10000000	268435456
r11	0x4fe8084	83787908
r12	0xfc8	4040
sp	0x20001ff0	0x20001ff0
lr	0x10c3	4291
pc	0x246	0x246 <main+46>
xPSR	0x1000000	16777216

The IDE uses OpenOCD debug interface software for programming and debugging the AXM0F243 microcontroller. For more information on the OpenOCD software, refer <http://openocd.org/>

8.3. DISASSEMBLY

The disassembly window shows the disassembly of the current function. A yellow triangle indicates the next instruction that is executed when the microcontroller is stepped or run. If mixed mode is selected, assembly instructions are interspersed by C source lines.

AX8052 disassembly window is shown in the below image



AXM0F243 disassembly window is shown in the below image

```

Disassembly
Function:
Frame start: 0x20002000
0x022c  push    {r7, lr}
0x022e  sub     sp, #8
0x0230  add     r7, sp, #0
0x0232  bl      0x904 <wtimer_init>
0x0236  bl      0x618 <get_startcause>
0x023a  subs    r3, r0, #0
0x023c  bne.n   0x254 <main+40>
0x023e  ldr     r3, [pc, #64] ; (0x280 <main+84>)
0x0240  movs    r2, #160 ; 0xa0
0x0242  lsls    r2, r2, #8
0x0244  str     r2, [r3, #8]
0x0246  ldr     r3, [pc, #56] ; (0x280 <main+84>)
0x0248  ldr     r2, [pc, #56] ; (0x284 <main+88>)
0x024a  str     r2, [r3, #4]
0x024c  ldr     r3, [pc, #48] ; (0x280 <main+84>)
0x024e  movs    r0, r3
0x0250  bl      0xaec <wtimer0_addrelative>
0x0254  cpsie   i
0x0256  bl      0x618 <get_startcause>
0x025a  bl      0x9b8 <wtimer_runcallbacks>
0x025e  cpsid   i
0x0260  adds    r3, r7, #7
0x0262  movs    r2, #2
0x0264  strb    r2, [r3, #0]
0x0266  adds    r3, r7, #7
0x0268  adds    r2, r7, #7
0x026a  ldrb    r2, [r2, #0]
0x026c  movs    r1, #1
0x026e  orrs    r2, r1
0x0270  strb    r2, [r3, #0]
0x0272  adds    r3, r7, #7
0x0274  ldrb    r3, [r3, #0]
0x0276  movs    r0, r3
0x0278  bl      0x938 <wtimer_idle>
0x027c  cpsie   i
0x027e  b.n     0x25a <main+46>
0x0280  lsls    r0, r2, #2
0x0282  movs    r0, #0
0x0284  lsls    r1, r5, #5
0x0286  movs    r0, r0
  
```

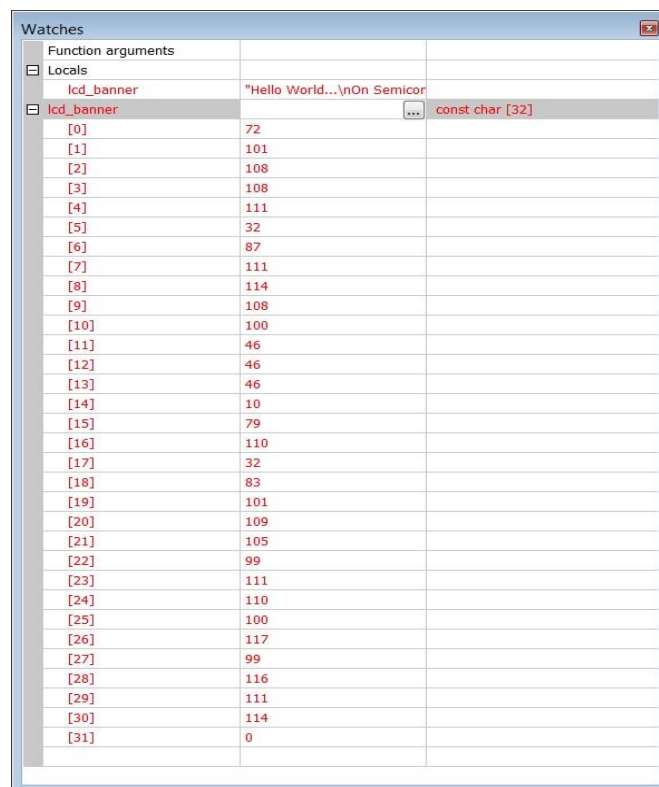
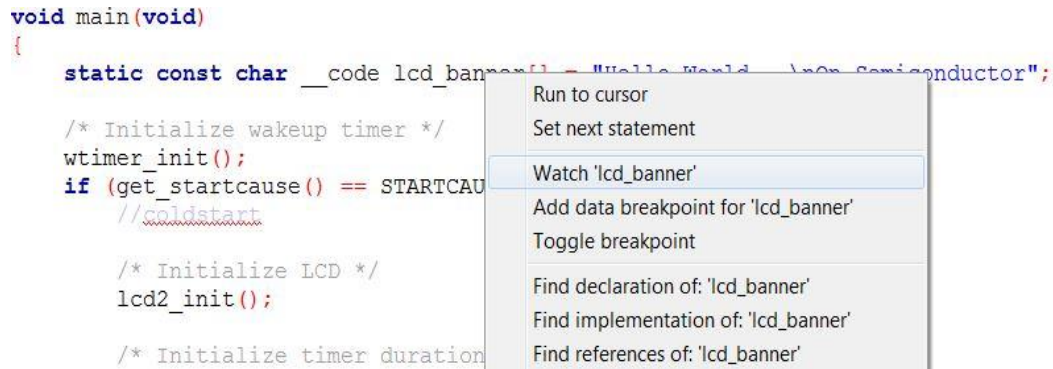
☐ Mixed Mode

8.4. MEMORY DUMP

This dialog allows to read a range in the memory of the microcontroller unit and displays it.

8.5. WATCHES

Watches allow the user to monitor the content of a variable which may be defined only in the high-level programming language. The easiest way to add a watch is to right click on the name of the variable and to choose the corresponding menu entry:



The watch window is able to display complex variable types. The last line is empty; its purpose is to allow adding watches simply by typing a name into the first column of the last empty row. Watch variables can be deleted by right-clicking into the name field of the watch to be deleted. A context menu then allows to rename or delete the watch.

8.6. PIN EMULATION

This function emulates the two pins of the microcontroller occupied by the debugger. The direction (input or output) is automatically detected. For both directions, the logic state of the pin is shown. Additionally, the state of inputs can be toggled. This feature is available only for AX8052 microcontroller.

8.7. DEBUG LINK

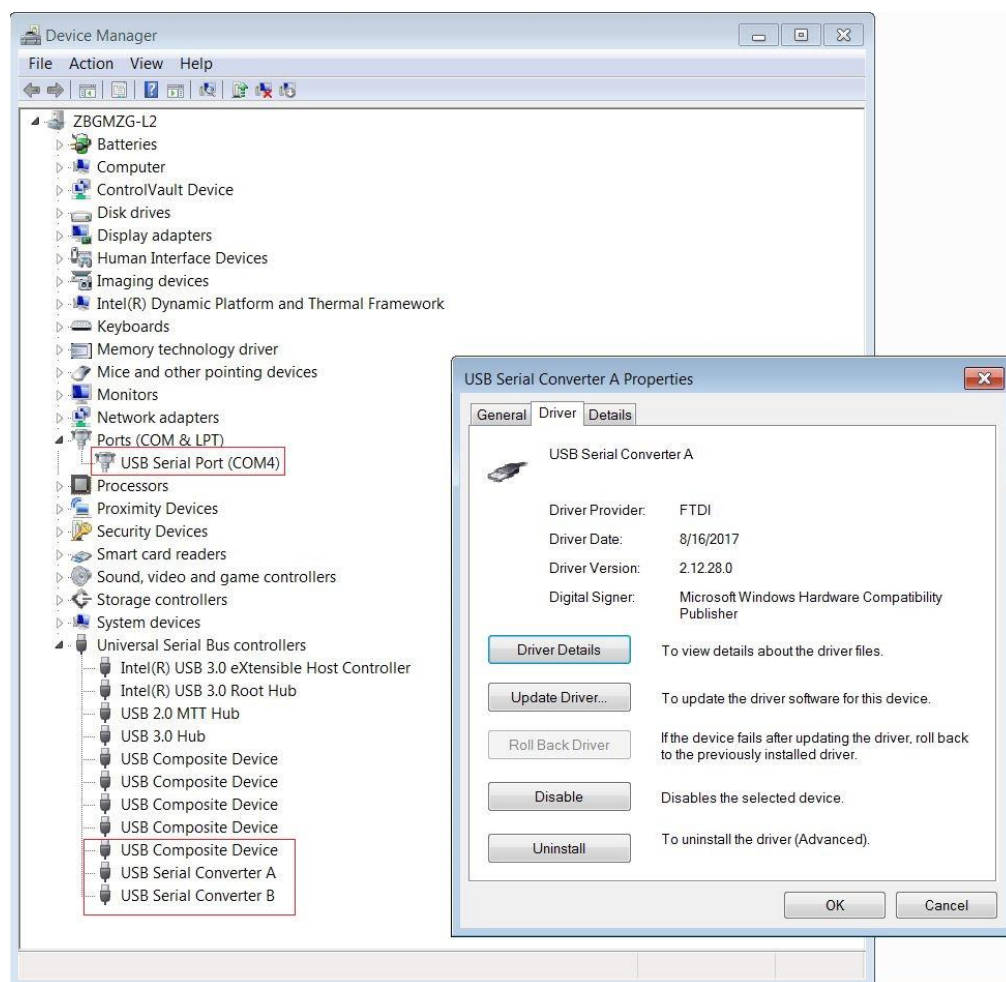
AX8052:

This window provides a graphical front-end to the debug link, a console-like input and output interface to the microcontroller unit. Local echo can be turned on for your convenience when the firmware is not providing a feedback. Please notice that the entry field is only active when the debugger is running.

AXM0F243:

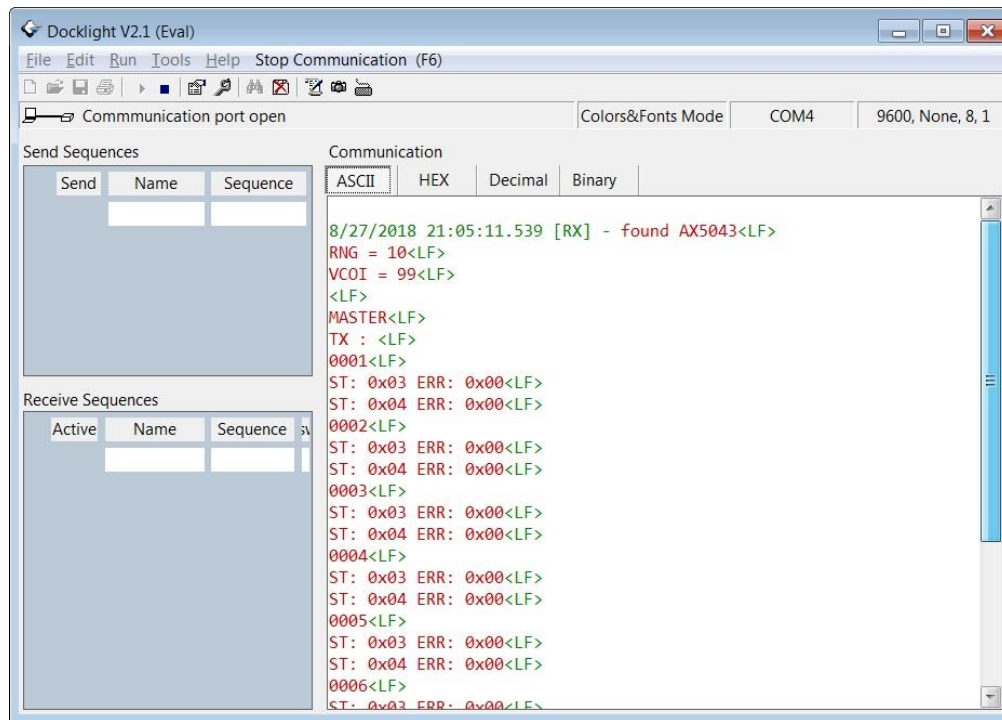
8.7.1. GENERAL SERIAL TERMINAL

For AXM0F243 microcontroller, the debug link data can be seen using any serial terminal software. Get the COM port (USB Serial Port) of the device from the device manager.



On the serial terminal software, set baud rate as 9600, Data bits as 8, Stop Bits as 1 and Parity as None.

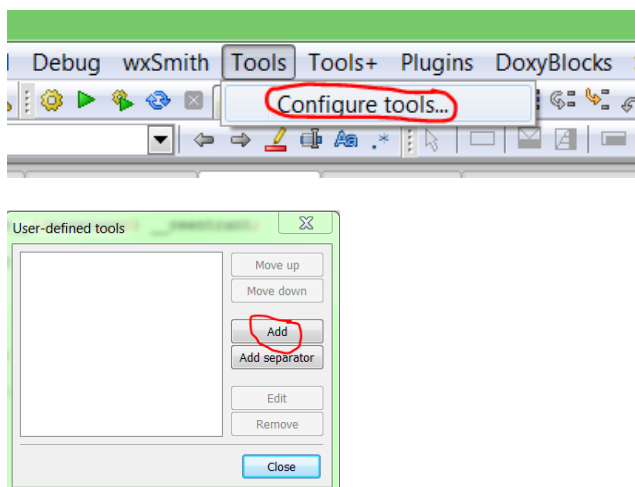
If the debug link options are enabled in the project, then debug messages can be seen on the serial terminal software as shown below.



8.7.2. WINDOWS POWER SHELL AS SERIAL TERMINAL

Configure Serial Terminal on codeblocks menu in order to open serial port and receive debug information from the device on 'Windows power shell' window (Windows-7 built-in).

Step 1: Open Tools Dialog using Code::Blocks->Tools->Configure Tools->Add



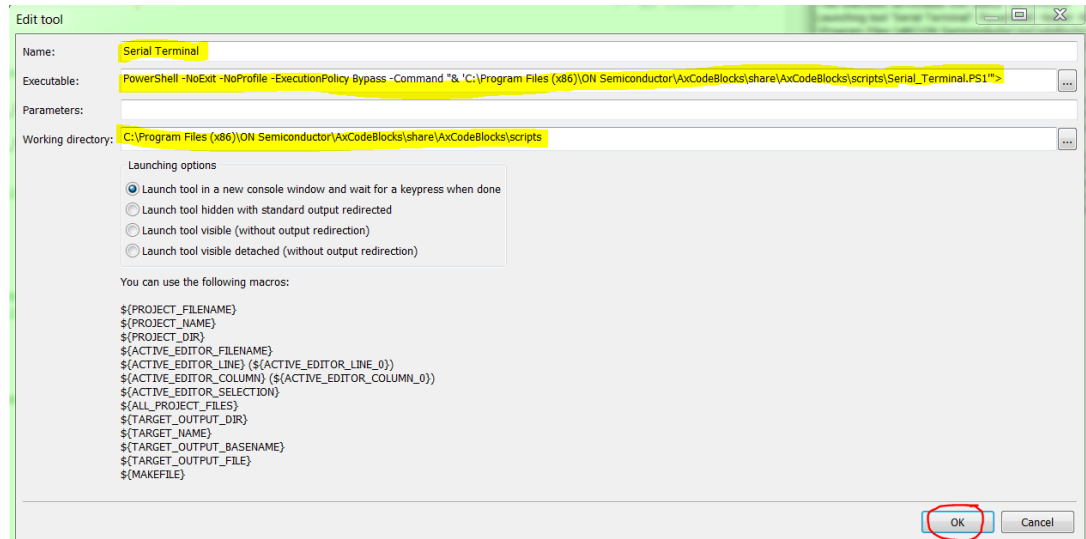
Step 2: Add below details at respective fields

Step 2.1: Name: <Serial Terminal>

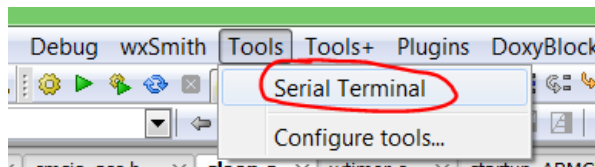
Step 2.2: Executable: <PowerShell -NoExit -NoProfile -ExecutionPolicy Bypass -Command "& 'C:\Program Files (x86)\ON Semiconductor\AxCodeBlocks\share\AxCodeBlocks\scripts\Serial_Terminal.PS1'">

Step 2.3: Working directory: <C:\Program Files (x86)\ON Semiconductor\AxCodeBlocks\share\AxCodeBlocks\scripts>

Step 2.4: Click OK



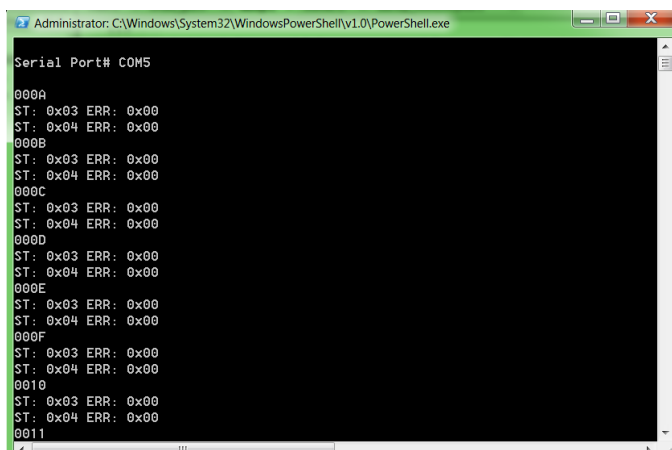
This will create 'Serial Terminal' item under the tools menu



Step 3: Open Serial Terminal window using Menu item

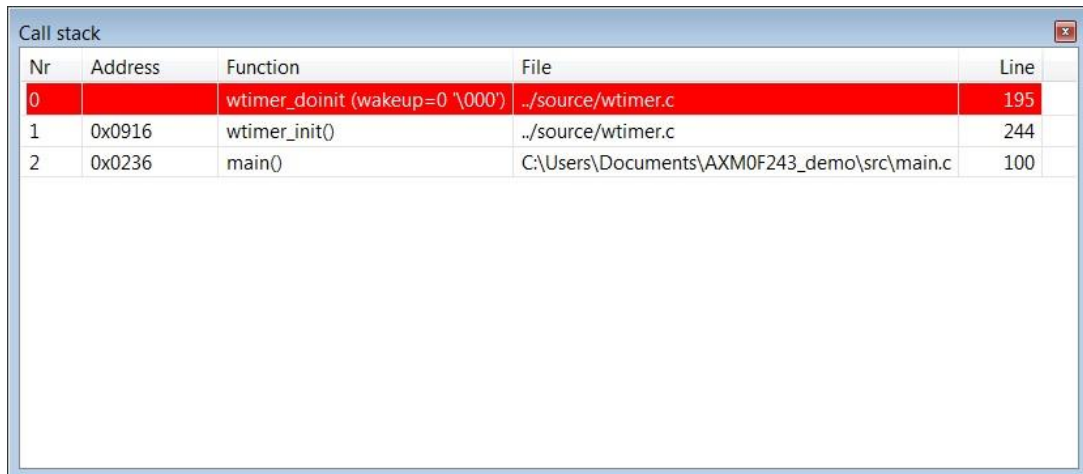
Code::Blocks->Tools->Serial Terminal

This will open 'windows power shell' window, opens serial port and receives Serial data from the device.



8.8. CALL STACK

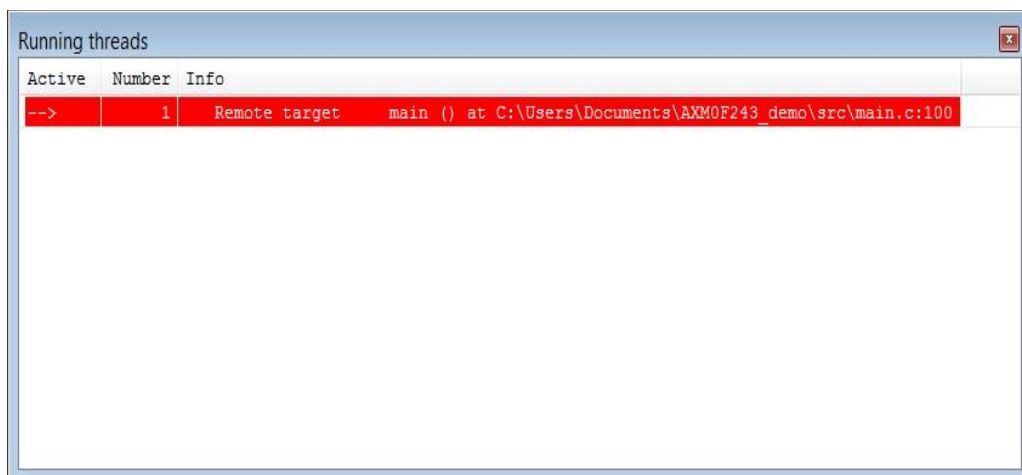
Whenever the program is paused by a breakpoint or during manual stepping, the current stack is displayed in the call stack window. The call stack window displays the name of each function and the file name in which the function is present. This feature is supported only for AXM0F243 microcontroller. The call stack window is shown in the below image.



Nr	Address	Function	File	Line
0		wtimer_doint (wakeup=0 '\000')	../source/wtimer.c	195
1	0x0916	wtimer_init()	../source/wtimer.c	244
2	0x0236	main()	C:\Users\Documents\AXM0F243_demo\src\main.c	100

8.9. RUNNING THREADS

In the running thread window, the name of function that is currently executing will be shown. This feature is available only for AXM0F243 microcontroller. The running threads window is shown in the below image.

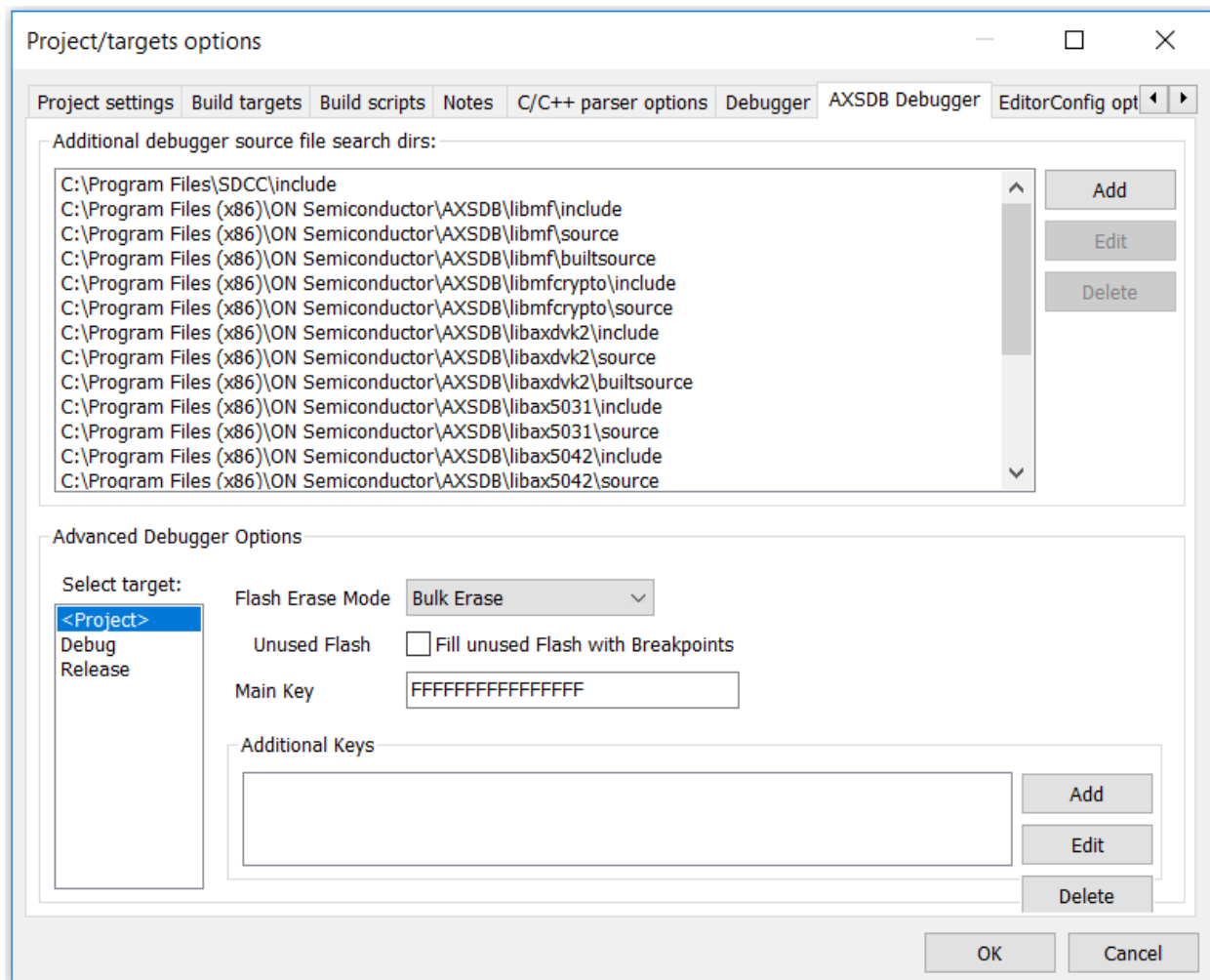


Active	Number	Info
-->	1	Remote target main () at C:\Users\Documents\AXM0F243_demo\src\main.c:100

9. ADVANCED DEBUGGER CONFIGURATION

9.1. AX8052

The advanced debugger configuration window can be reached by selecting Project→Properties... from the menu bar, and then clicking on the “AXSDB Debugger” tab.



Whenever the CPU stops, for example because it hits a breakpoint or the user hits the pause button, the debugger reads the current PC and looks up the file name and line number corresponding to that PC in the debug information. If this file is found neither in the open editor tabs, nor in the project directories, the debugger searches the directories listed under “*Additional debugger source file search dirs*”. It is recommended to add the source and include directories of all used libraries (such as libmf) to the directory list to enable source level debugging even in library code.

Flash Erase Mode specifies the strategy the debugger uses to erase the flash memory before reprogramming. *Bulk Erase*, the default, sends a bulk erase command to the microcontroller, thus erasing the complete memory. If the debugger knows the device key, it saves and restores the calibration data in the last flash sector. Bulk erase may be issued even without knowing the device key; the calibration data will be lost however. The debugger warns you if

it does not know the correct key and asks you to confirm to continue. After a Bulk Erase, the device key is set to the *Main Key* configured below.

All Sectors Erase instructs the debugger to issue individual page erase commands to all flash sectors that need to be reprogrammed. Flash sectors not needed by the program to be loaded are erased if they contain data. The device key cannot be changed, unless the device was using the default key. This option may be faster than *Bulk Erase* if only little changes between download cycles.

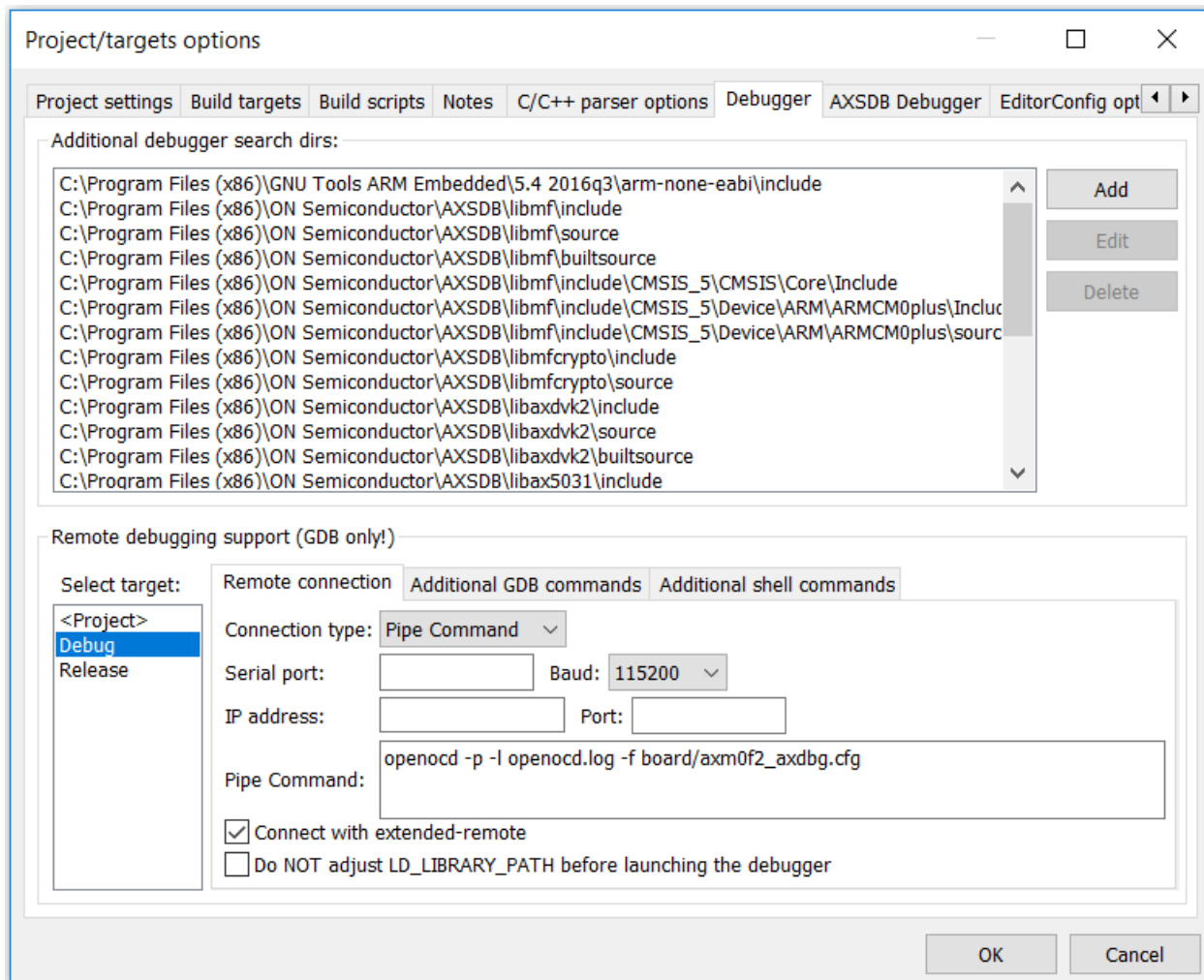
Needed Sectors Erase instructs the debugger to issue individual page erase commands to all flash sectors that need to be reprogrammed. Flash sectors not needed by the program will be left as-is. The device key cannot be changed, unless the device was using the default key. This option may be the fastest if only little changes between download cycles, however left-over contents are not necessarily cleared.

In order to protect the intellectual property of the customer while still allowing full debugging capability, use of the debug interface is protected by a 64 bit key. The debug interface can only be used if the device key is known. The default key (as shipped by ON Semiconductor, and after a bulk erase) is FFFFFFFFFFFFFFFF. *Main Key* specifies the key the debugger should set to protect the debug interface.

Additional Keys lists keys that are tried as well when connecting the device when the main key does not unlock the device. This is useful if multiple projects use different keys, and devices from other projects should be used. If their keys are listed under *Additional Keys*, the debugger will be able to retain the calibration data and reprogram the key to the *Main Key*, when the *Flash Erase Mode* is set to Bulk Erase.

9.2. AXM0F243

The advanced Debugger configuration window can be reached by selecting Project→Properties... from the menu bar, and then clicking on the “Debugger” tab.



For AXM0F243, the programming and debugging happens with the GDB/CDB (GNU) debugger using OpenOCD commands at the background.

10. ONSEMI PROJECT WIZARD

10.1. AX8052

The new project wizard supports the creation of a skeleton project template for AX8052. For AX8052 microcontroller, both SDCC and IAR compilers are fully supported, with selectable code models.

The code structure of the AX8052 example project's main.c looks like this:

```
#if defined(__ICC8051__)

#define coldstart 1

#define warmstart 0

//

// If the code model is banked, low_level_init must be declared

// __near_func else a ?BRET is performed

//

#if (__CODE_MODEL__ == 2)

__near_func __root char

#else

__root char

#endif

__low_level_init(void) @ "CSTART"

#else

#define coldstart 0

#define warmstart 1

uint8_t _sdcc_external_startup(void)

#endif

{

    DPS = 0;

    . . .

    GPIOENABLE = 1;

    /* Check for warmstart or coldstart */

    if (PCON & 0x40)

        return warmstart;

    return coldstart;

}

#undef coldstart
```

```
#undef warmstart

#if defined(SDCC)
extern uint8_t __start__stack[];
#endif

void main(void)
{
    #if !defined(SDCC) && !defined(__ICC8051__)
        _sdcc_external_startup();
    #endif

    #if defined(SDCC)
        __asm
        G$__start__stack$0$0 = __start__stack
        .globl G$__start__stack$0$0
        __endasm;
    #endif

    ...
}
```

We need to distinguish the different compilers.

☐ SDCC

If available, SDCC arranges for a function named `_sdcc_external_startup` to be called before `main`. The return value of this function determines whether static variables should be initialized. The example code uses this to avoid overwriting static variables on a wake-up from sleep.

Stack: The two `ifdefs` enclosing `__start__stack` define a global symbol so that the debugger knows where the stack starts, and can find the call stack.

☐ IAR

If available, SDCC arranges for a function named `__low_level_init` to be called before `main`. The return value of this function determines whether static variables should be initialized. The function needs to reside in segment `CSTART` and its calling convention is determined by the code model selected. The example code uses this to avoid overwriting static variables on a wake-up from sleep.

☐ Keil

Keil does not call any startup routine, its runtime library does not support bypassing static variable initialization. Therefore, the example code calls it explicitly if the compiler is Keil.

10.2. AXM0F243

The new project wizard supports the creation of a skeleton project template for AXM0F243. For AXM0F243 microcontroller, GNU GCC Compiler for ARM is supported.

The code structure of the AXM0F243 example project's main.c looks like this:

```
#define coldstart 0

#define warmstart 1


uint8_t _axm0f2_external_startup(void)
{
    . . .

    return get_startcause() == STARTCAUSE_COLDSTART ? coldstart : warmstart;
}

#undef coldstart
#undef warmstart

void main(void)
{
    /* Initialize wakeup timer */
    wtimer_init();

    if (get_startcause() == STARTCAUSE_COLDSTART) {
        . . .
    }

    for (;;)
    {
        wtimer_runcallbacks();

        . . .

        uint8_t flg = WTFLAG_CANSTANDBY;

        flg |= WTFLAG_CANSLEEP;

        wtimer_idle(flg);

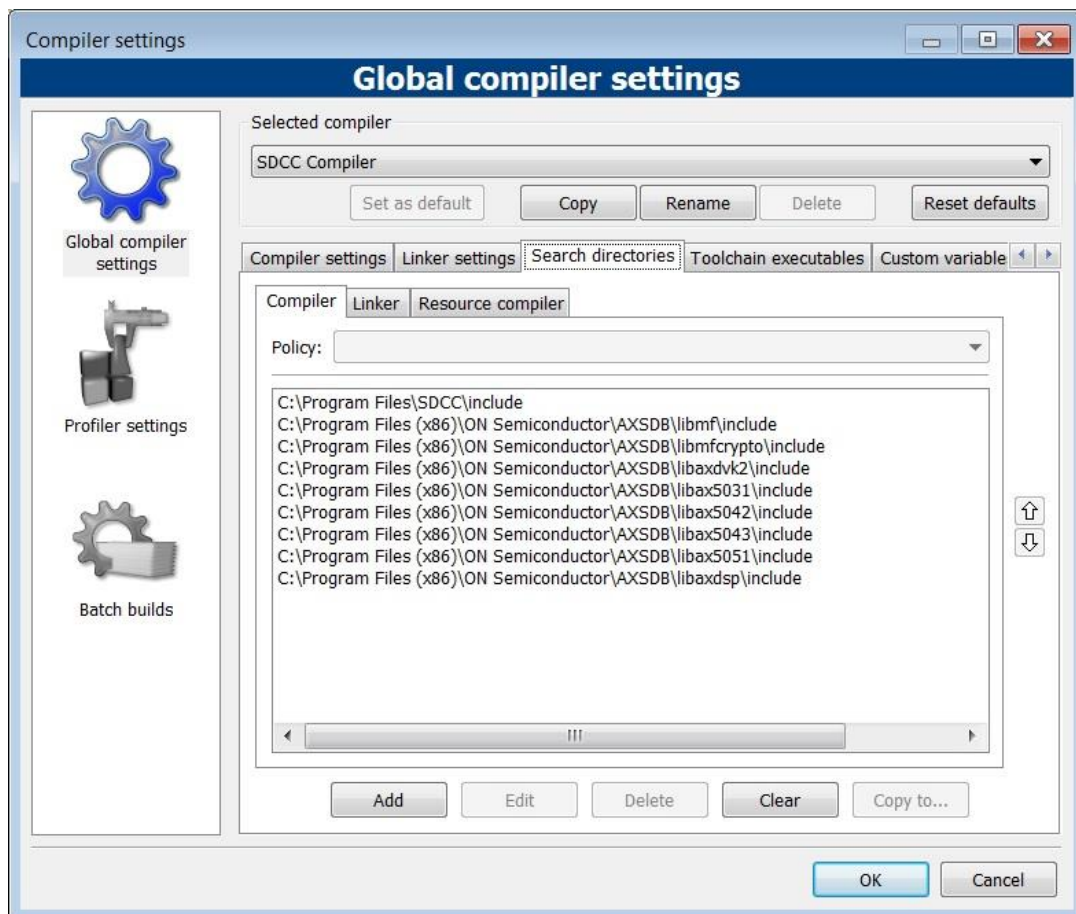
        . . .
    }
}
```


11. TROUBLESHOOTING GUIDE

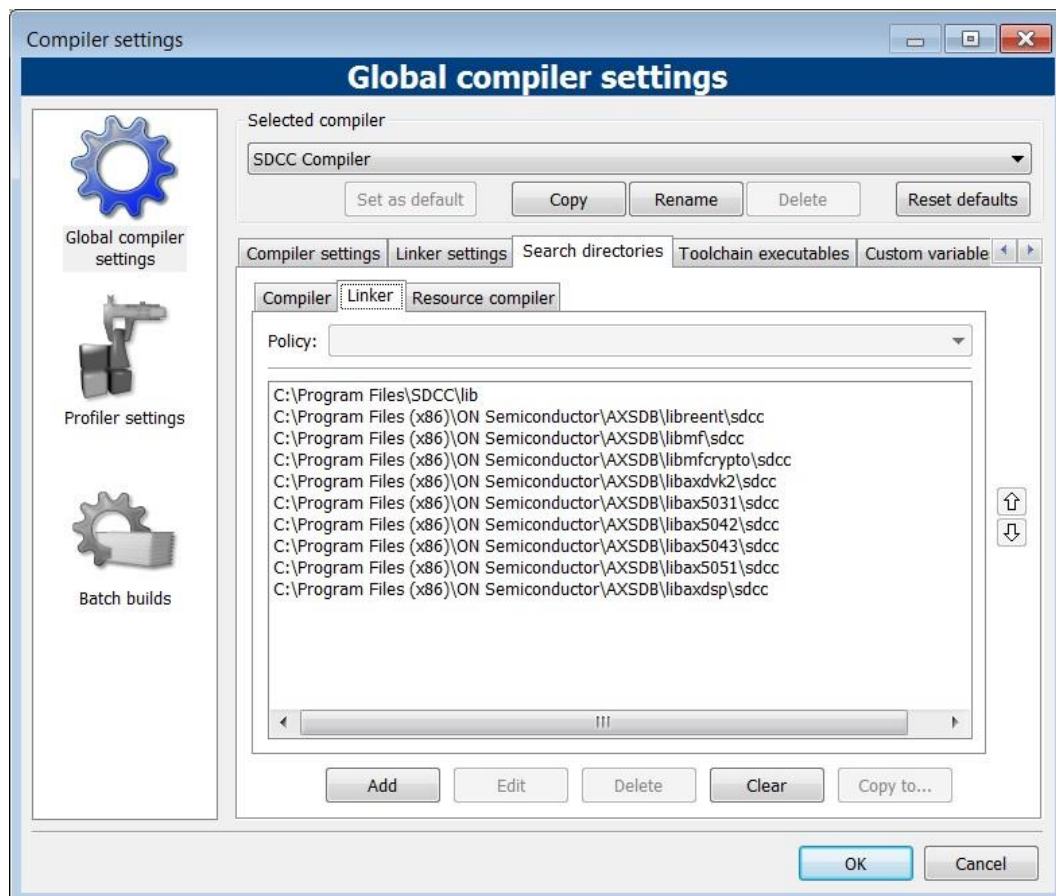
11.1. COMPILER AUTO-DETECTION FAILS ON FIRST START

AX8052:

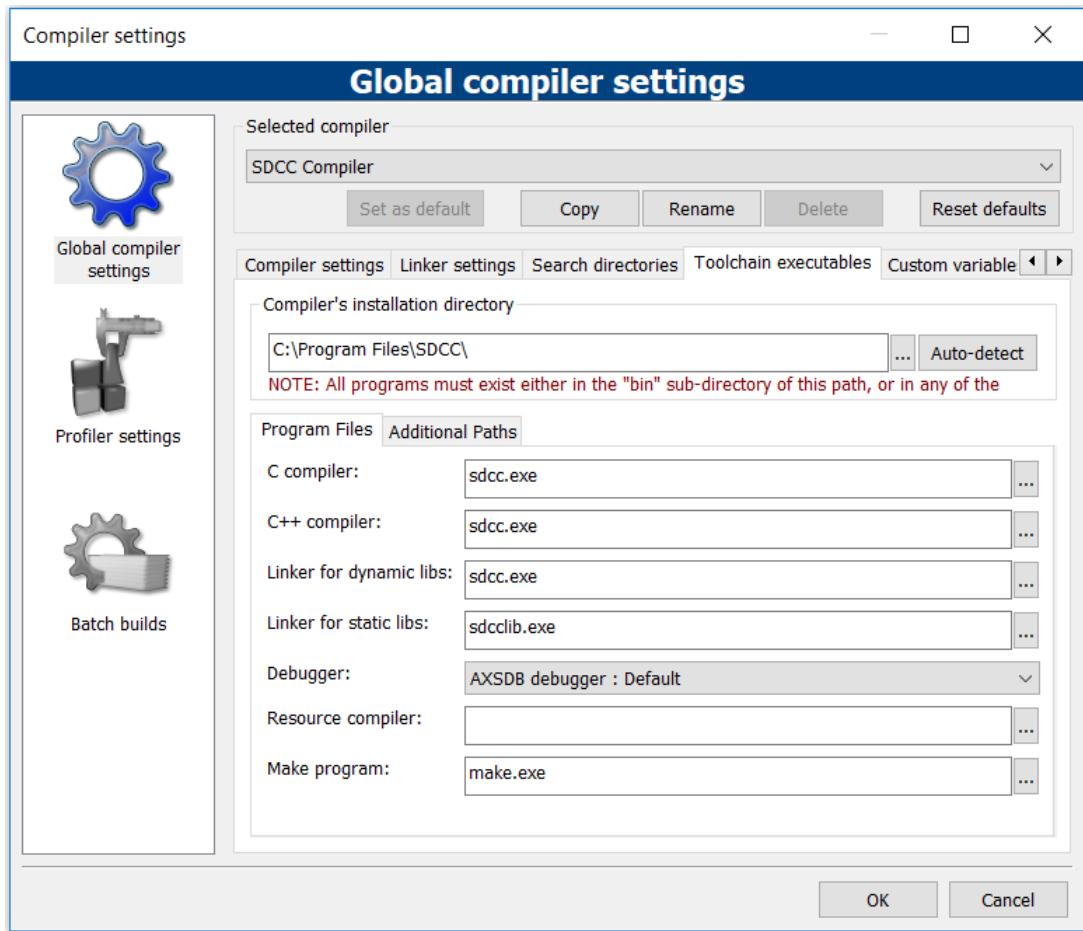
When AxCode::Blocks is run for the first time, it tries to auto-detect the compilers. If the auto-detection fails, configure the compiler (SDCC) for Ax8052 manually as shown in the screen shots below. The path C:\Program Files\ must be replaced by the actual installation path if a non-default installation has been selected.



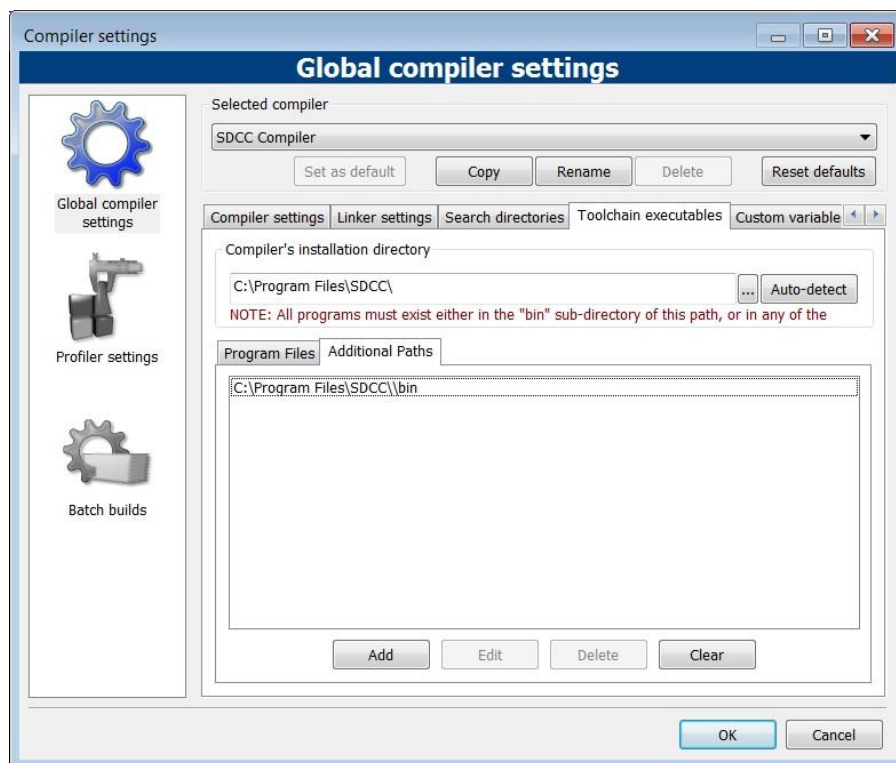
Open the compiler settings dialog by clicking on Settings→Compiler. Select SDCC (under selected compiler). Click on "Set as default". Click on "Search Directories" and "Compiler", and enter the directories as in the picture above.



Click on "Linker", and enter the directories as in the picture above.



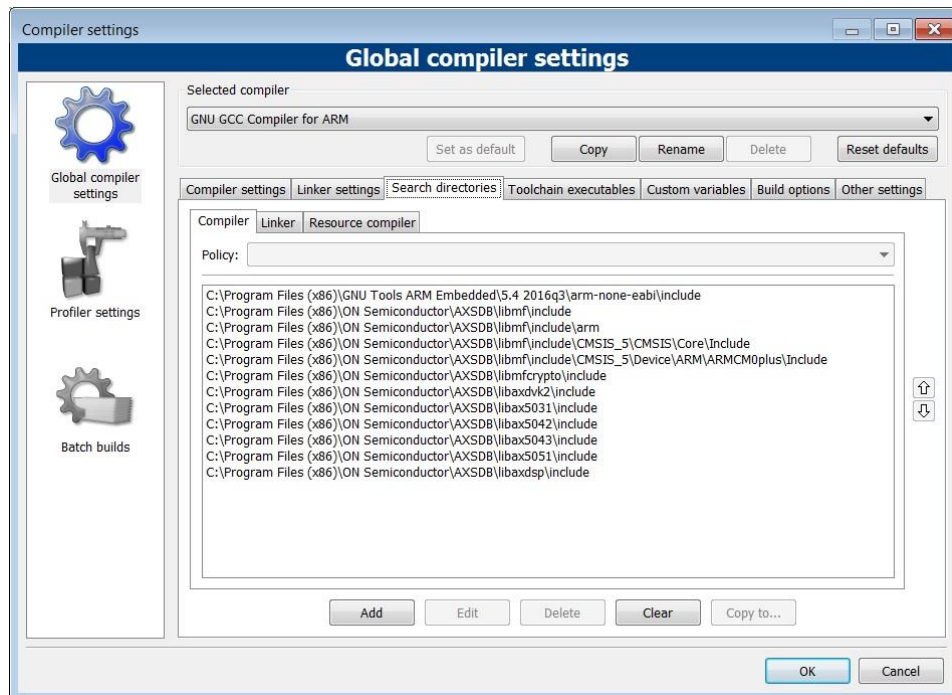
Click on "Toolchain Executables", and verify the "Compiler's Installation directory".



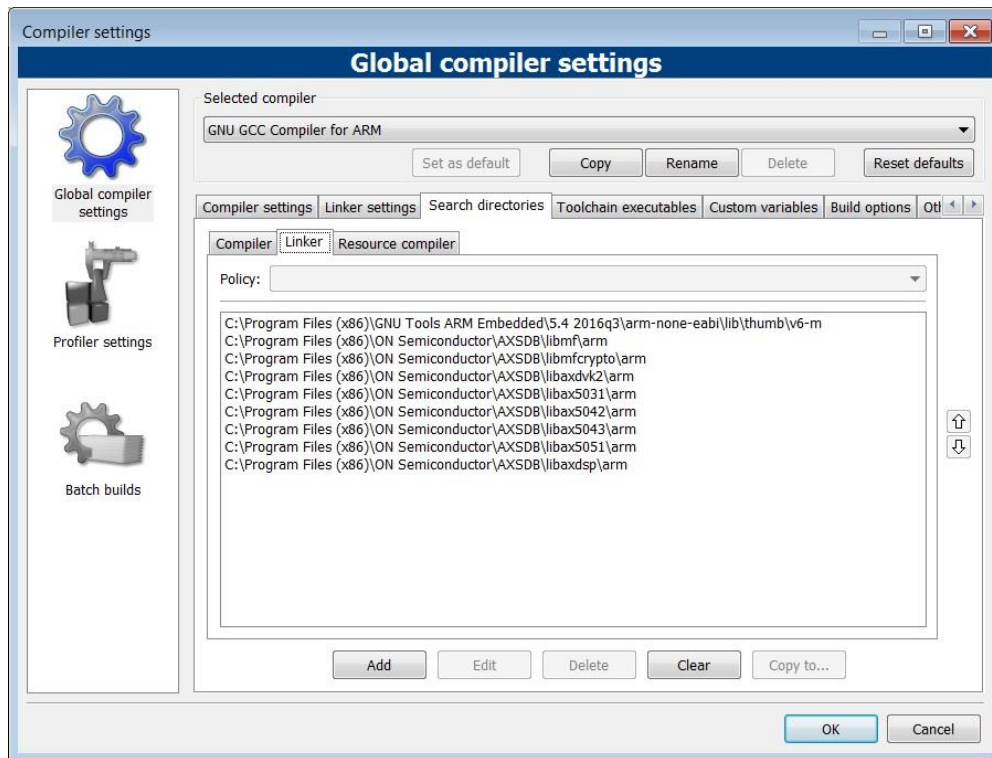
Click on "Additional Paths", and enter the path as shown above.

AXM0F243:

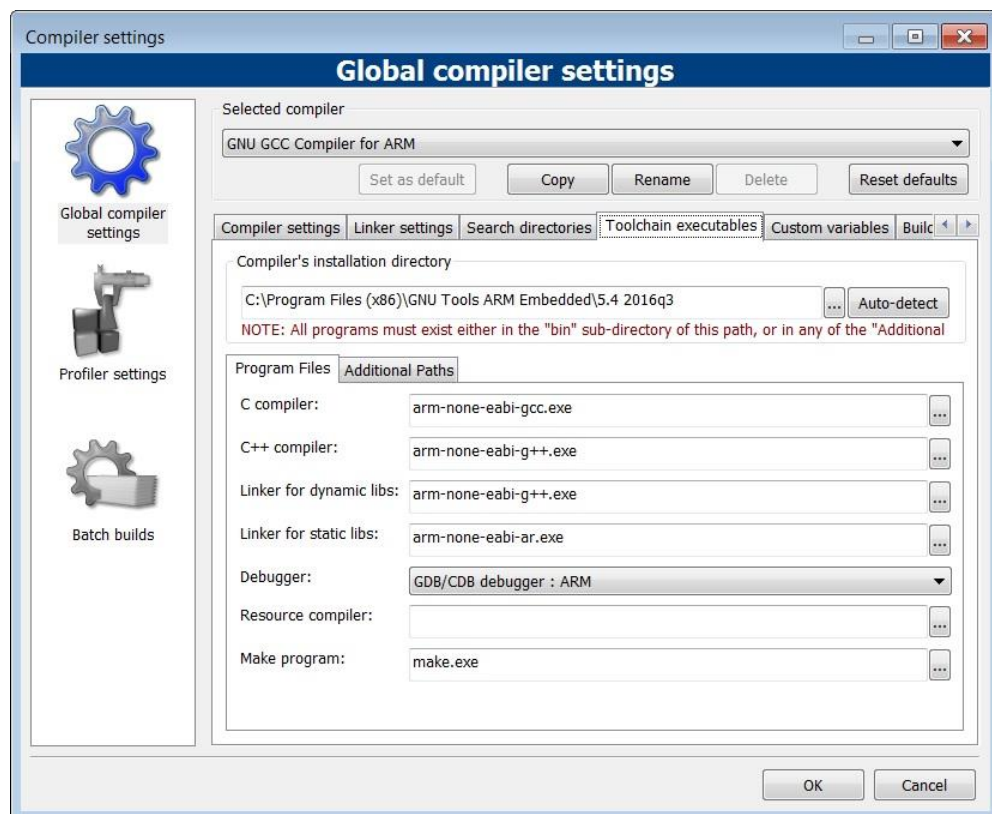
When AxCode::Blocks is run for the first time, it tries to auto-detect the compilers. If the auto-detection fails, configure the compiler (GNU GCC Compiler for ARM) for AXM0F243 manually as shown in the screen shots below. The path C:\Program Files (x86)\ must be replaced by the actual installation path if a non-default installation has been selected.



Open the compiler settings dialog by clicking on Settings→Compiler. Select GNU GCC Compiler for ARM (under selected compiler). Click on "Set as default". Click on "Search Directories" and "Compiler", and enter the directories as in the picture above.



Click on "Linker", and enter the directories as in the picture above.



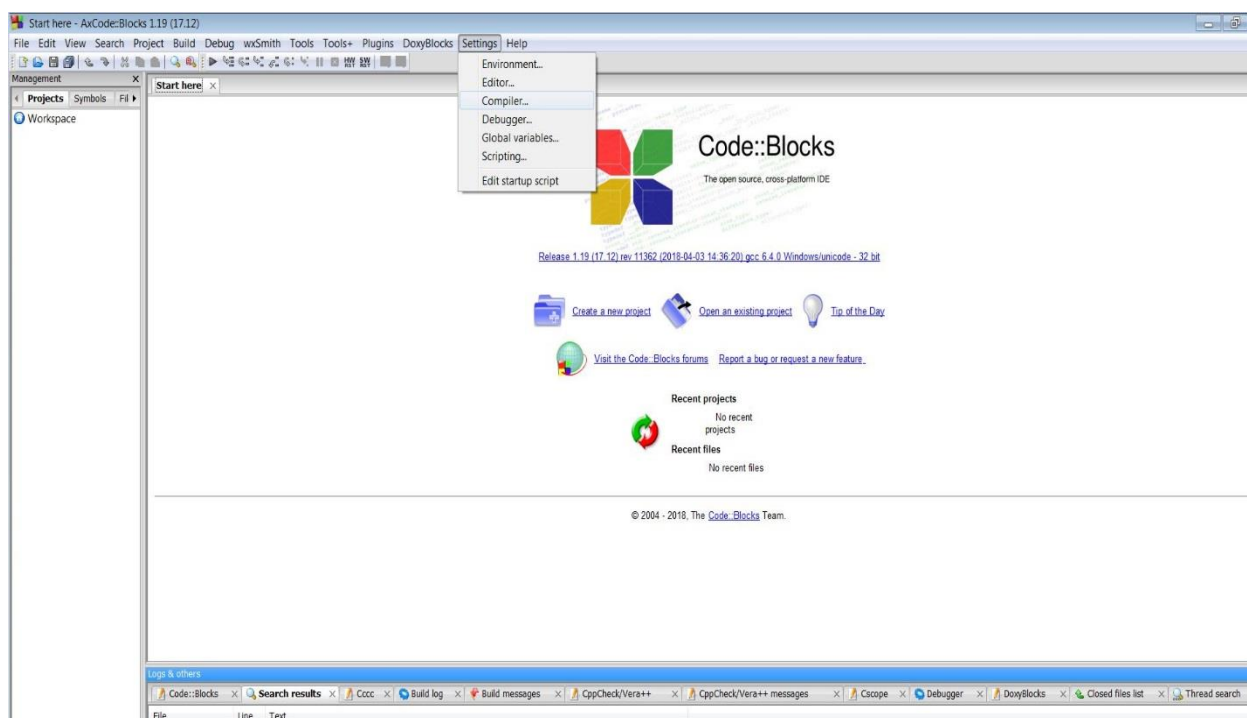
Click on "Toolchain Executables", and verify the "Compiler's Installation directory".

11.2. REMOVE ALL SAVED USER SETTINGS

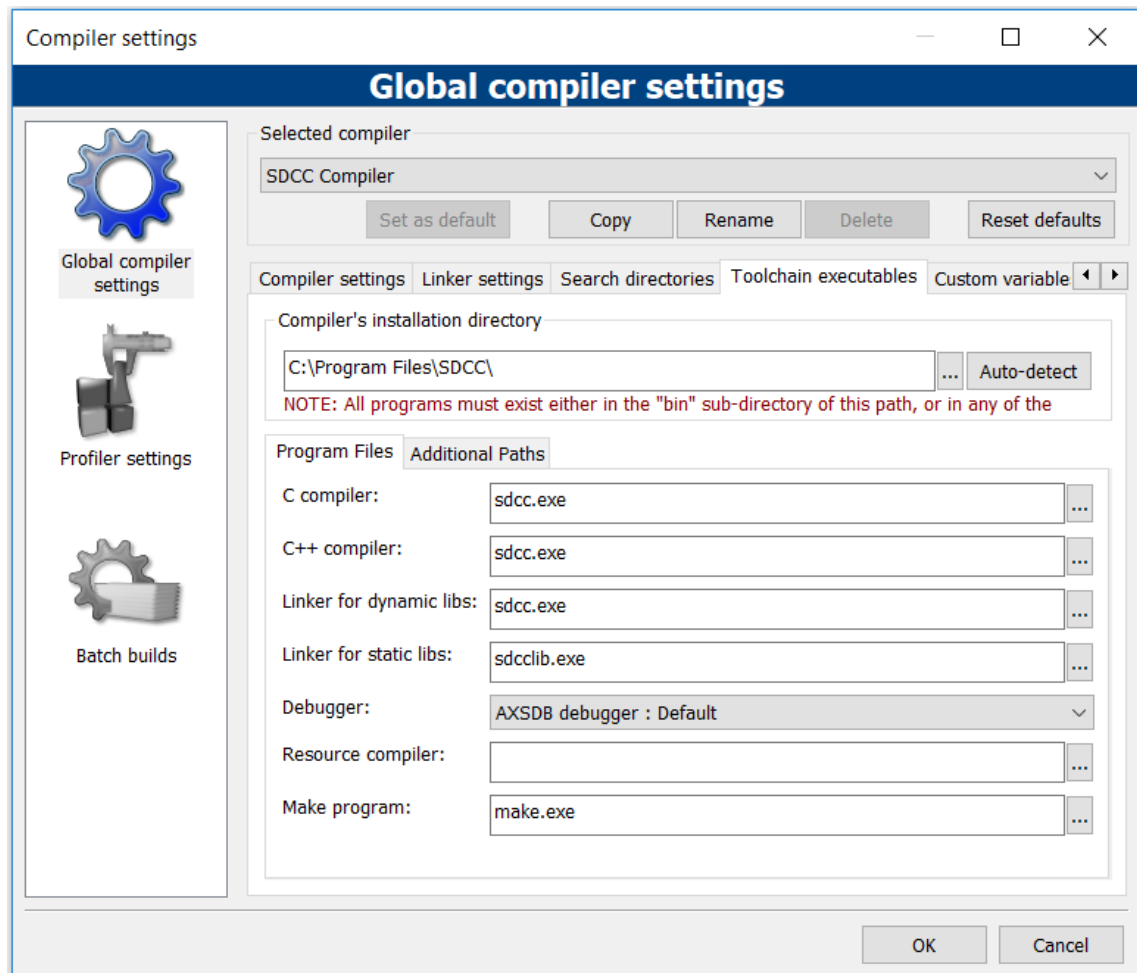
Open an explorer. Type "%APPDATA%" (without double quotes) into the address bar. Type enter. The directory now displayed should contain a folder (subdirectory) named "axCodeBlocks". This is where AxCode::Blocks stores per-user configuration settings. Delete that subdirectory.

11.3. SDCC PROJECT DOES NOT COMPILE

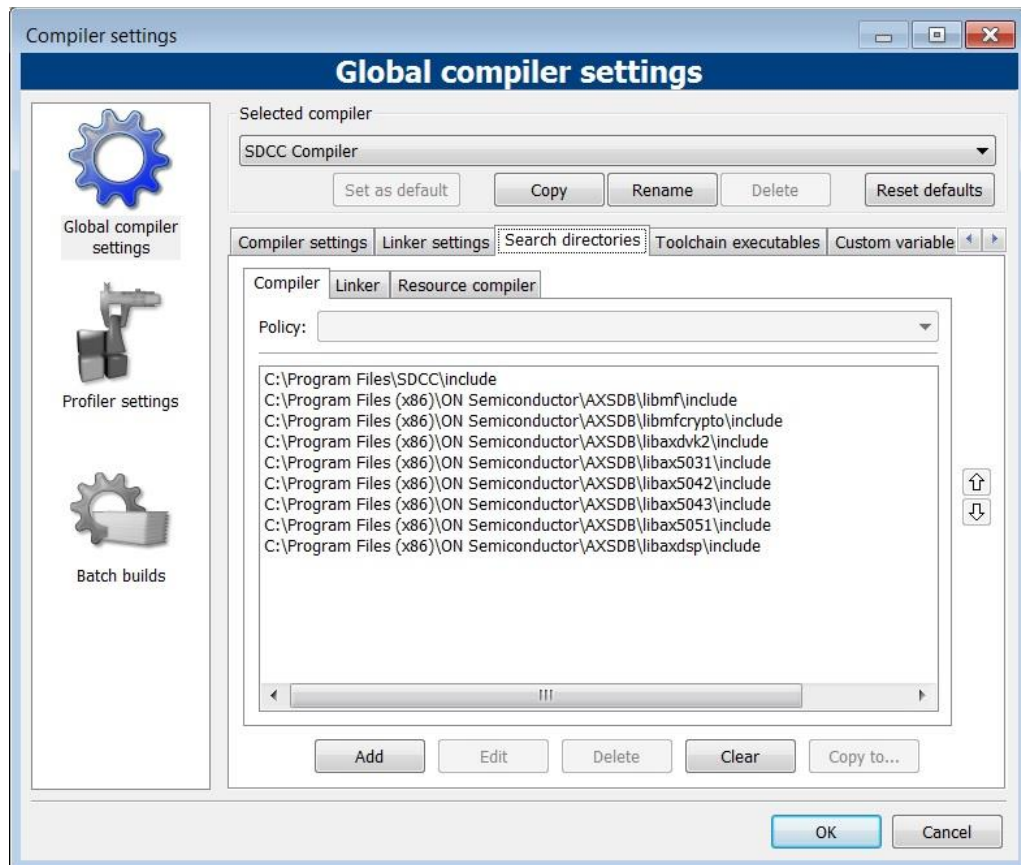
First, make sure that AxCode::Blocks finds the compiler executable. Open the compiler settings window.



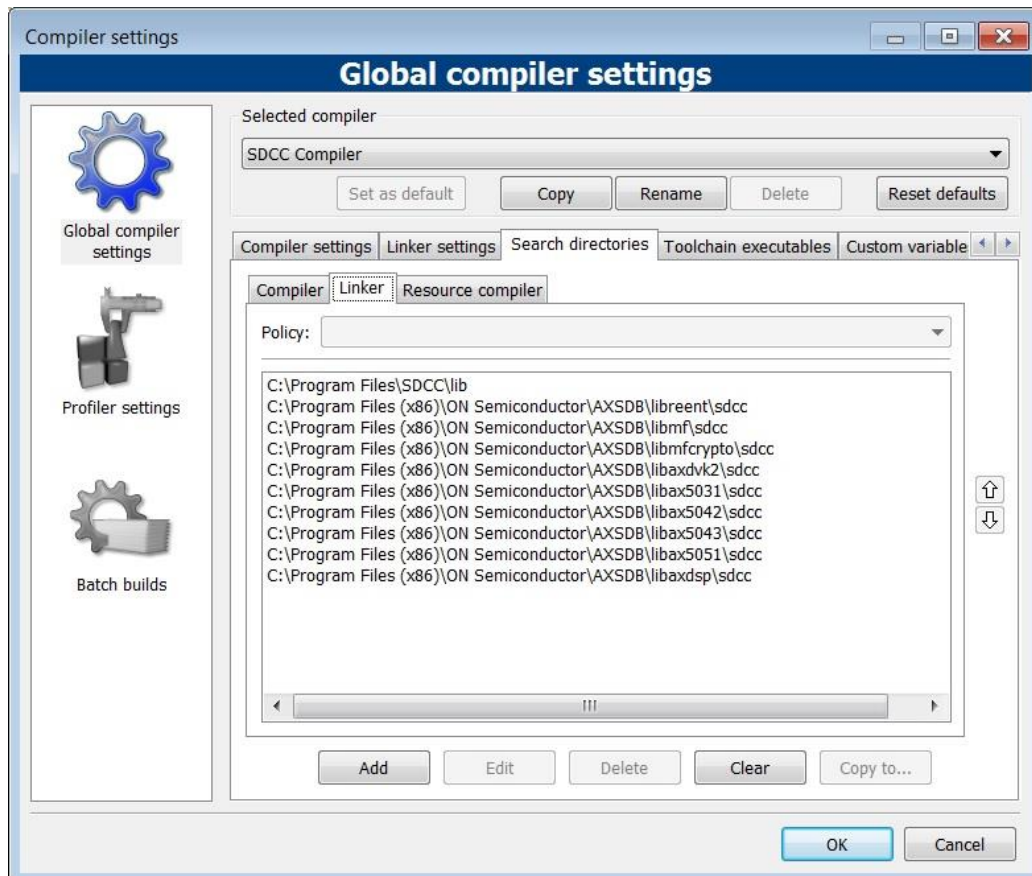
Check that SDCC is the default compiler and that the Compiler's Installation directory points to SDCC's installation location. Normally, this is automatically set up correctly, but if you manually move the SDCC directory, or re-install it at another location, you must manually update the location.



Make sure that SDCC finds all required header files. Check that the compiler's default include paths contain SDCC's own includes, as well as the AX8052 convenience library (such as libmf) includes.



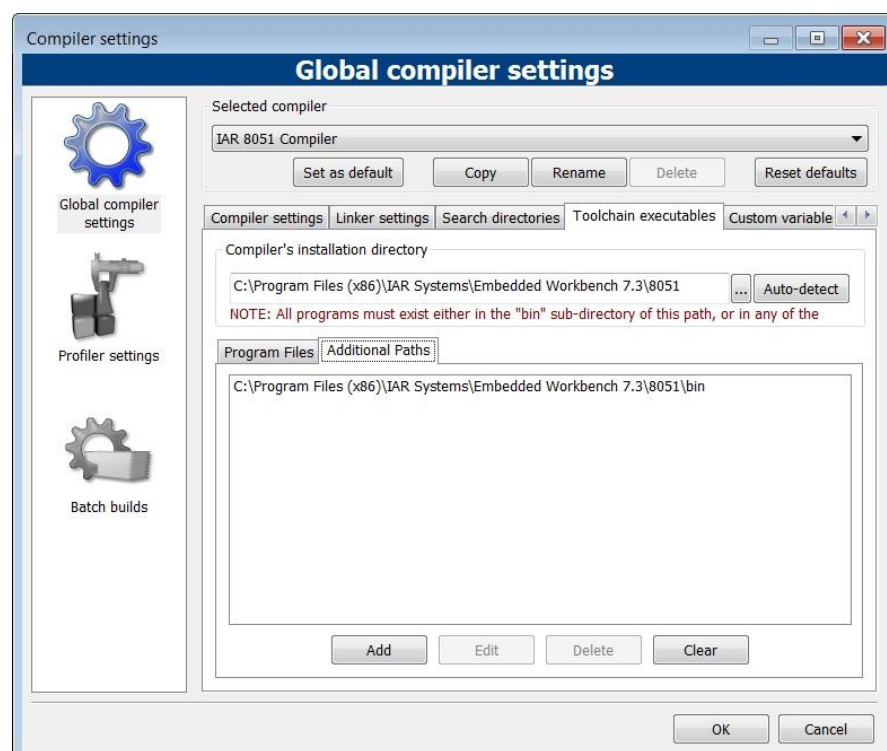
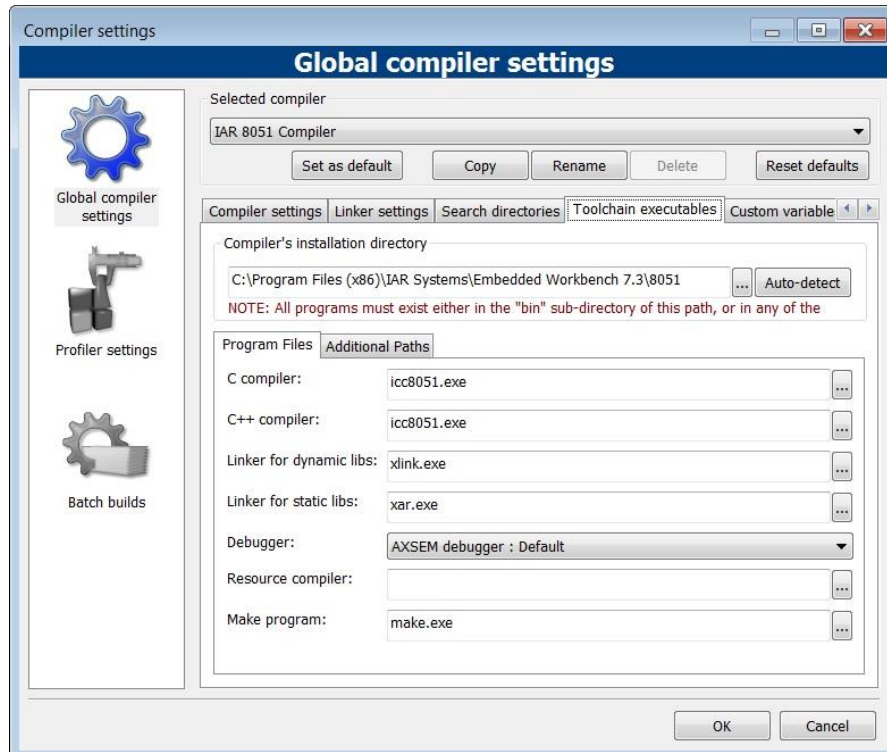
Check that the linker's default paths contain SDCC's own libraries path, as well as the AX8052 convenience library (such as libmf) path.



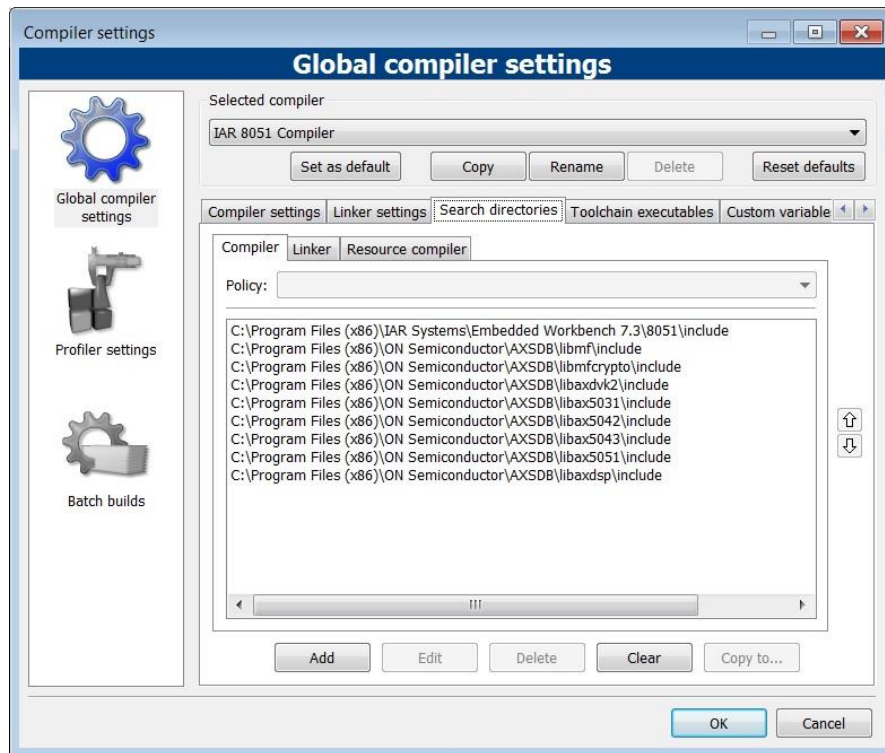
11.4. IAR PROJECT DOES NOT COMPILE

First, make sure that AxCode::Blocks finds the compiler executable. Open the compiler settings window. Then select "IAR 8051 Compiler" in the "Selected Compiler" combobox.

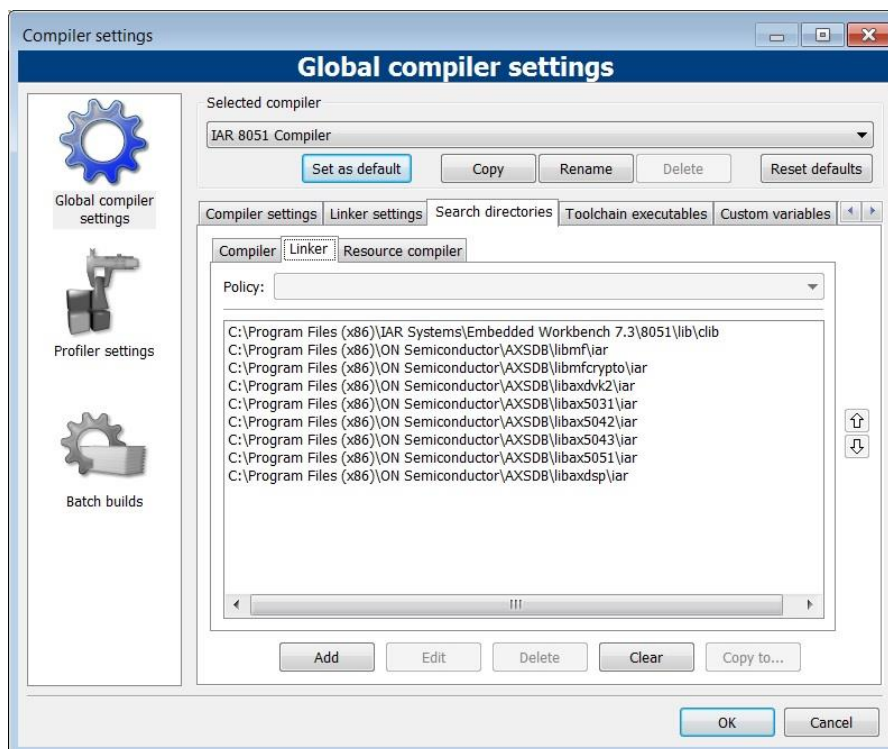
Check that the Compiler's Installation directory points to IAR's installation location. Normally, this is automatically set up correctly, but if you manually move the IAR directory, or re-install it at another location, or update it to a new version, you must manually update the location.



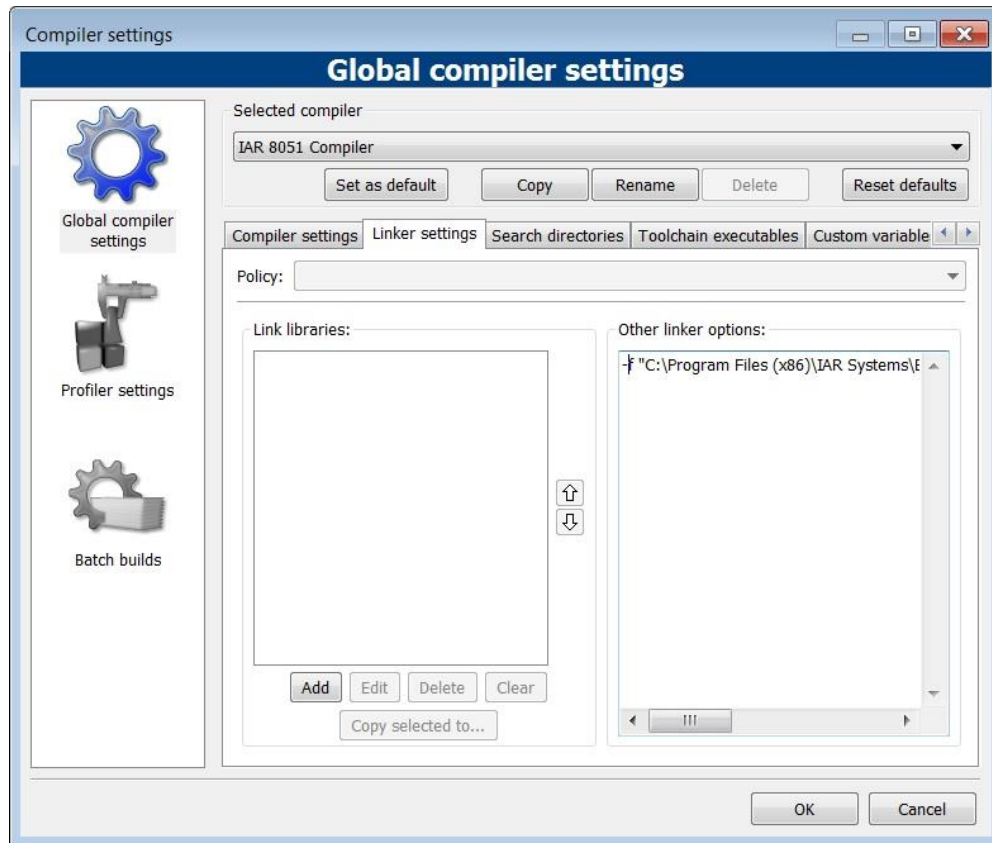
Make sure that IAR finds all required header files. Check that the compiler's default include paths contain IAR's own includes, as well as the AX8052 convenience library (such as libmf) includes.



Check that the linker's default paths contain IAR's own libraries path, as well as the AX8052 convenience library (such as libmf) path.

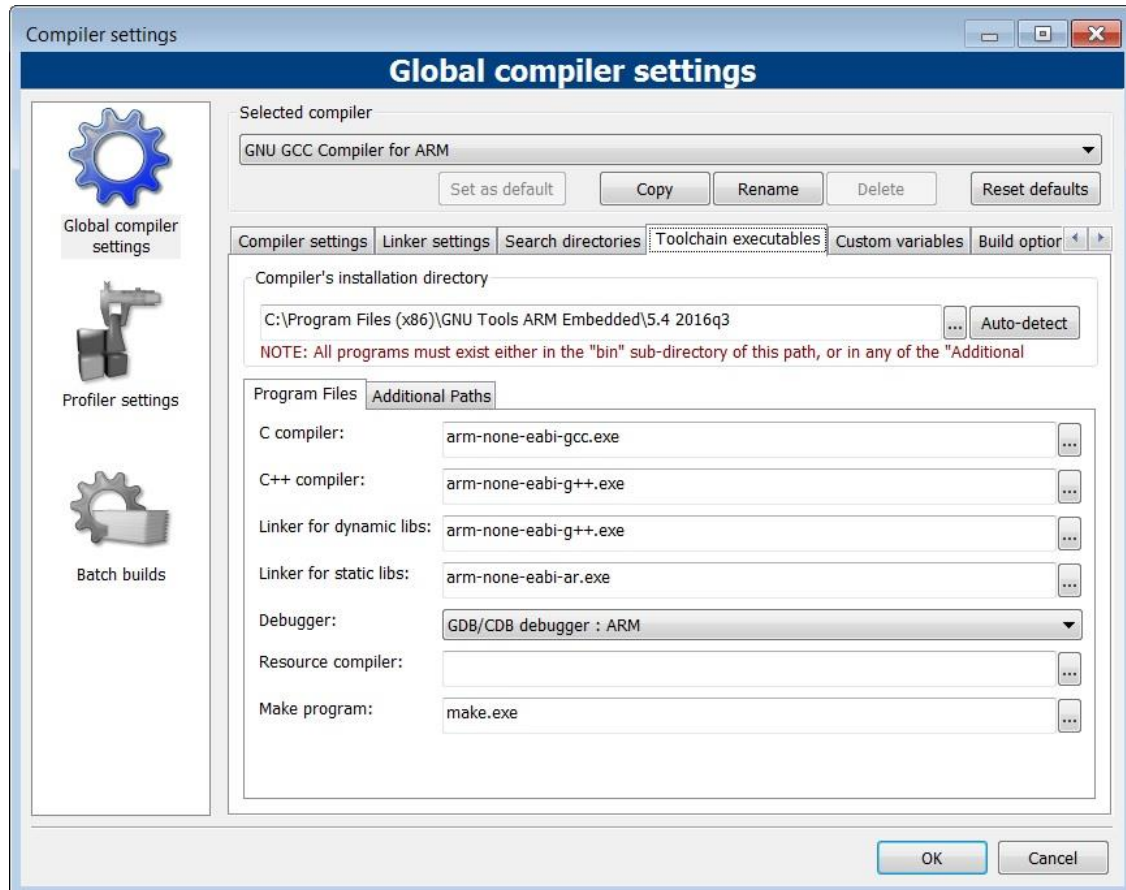


Make sure the correct linker script appears under "Other linker options"; the correct entry is
 -f "C:\Program Files (x86)\IAR Systems\Embedded Workbench 7.3\8051\config\devices_generic\
 lnk51ew_plain.xcl"



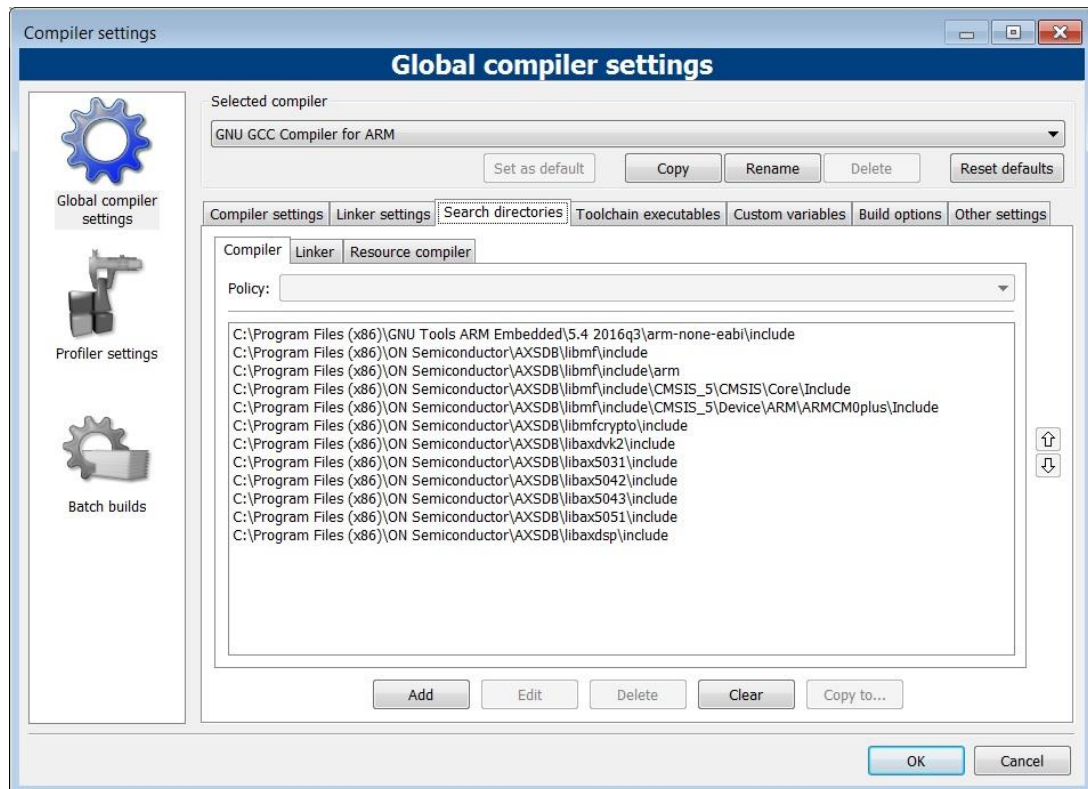
11.5. GCC PROJECT DOES NOT COMPILE

First, make sure that AxCode::Blocks finds the compiler executable. Open the compiler settings window. Then select "GNU GCC Compiler for ARM" in the "Selected Compiler" combobox.

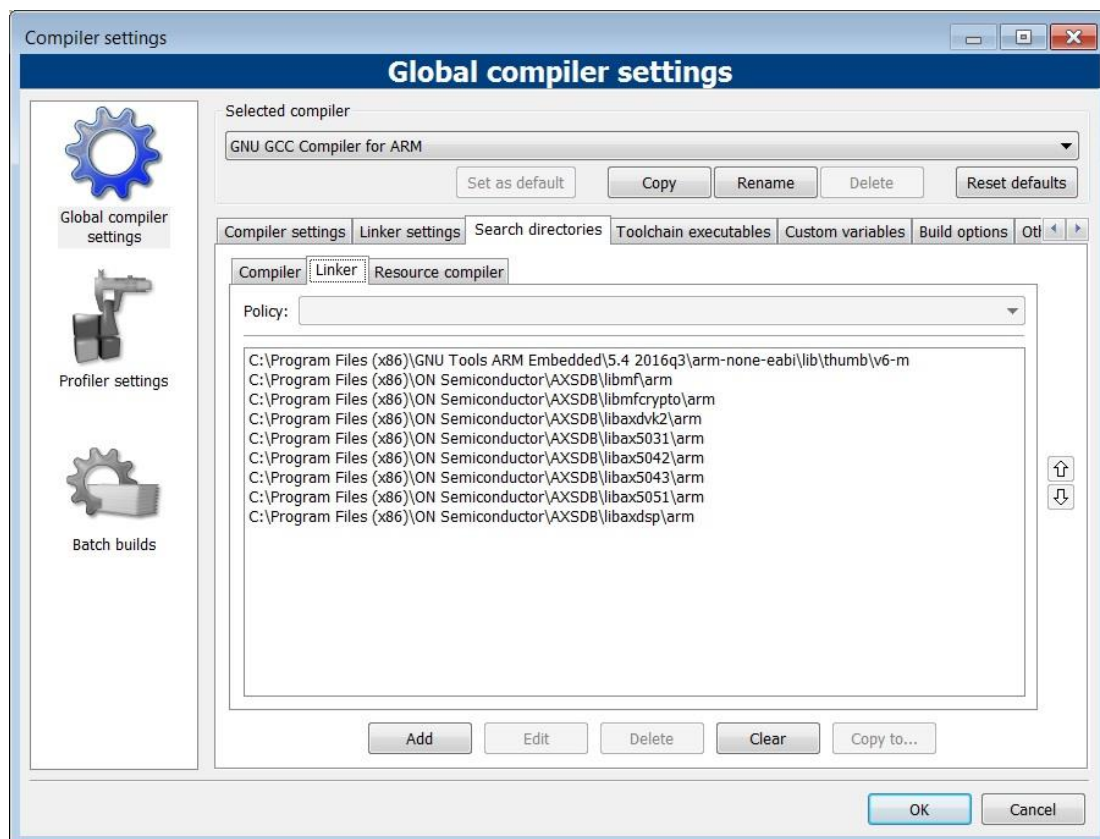


Check that GNU GCC Compiler for ARM is the default compiler and that the Compiler's Installation directory points to compiler's installation location. Normally, this is automatically set up correctly, but if you manually move the compiler directory, or re-install it at another location, you must manually update the location. Also make sure that the compiler's installation location is available in the system path. If it's not available, add the same to the system path.

Make sure that GCC compiler finds all required header files. Check that the compiler's default include paths contain GCC compiler's own includes, as well as the AXM0F243 convenience library (such as libmf) includes.

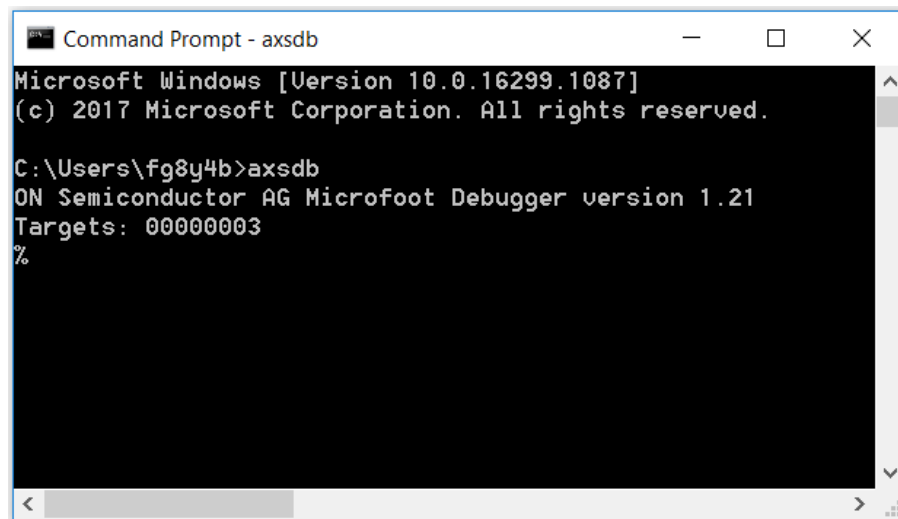


Check that the linker's default paths contain GCC compiler's own libraries path, as well as the AXM0F243 convenience library (such as libmf) path.



11.6. PROJECT COMPILES, BUT DEBUGGING DOES NOT WORK

First check whether the ON Semiconductor Command Line Debugger, AXSDB, works. Start AXSDB (the installer places a link into the program section of the start menu).

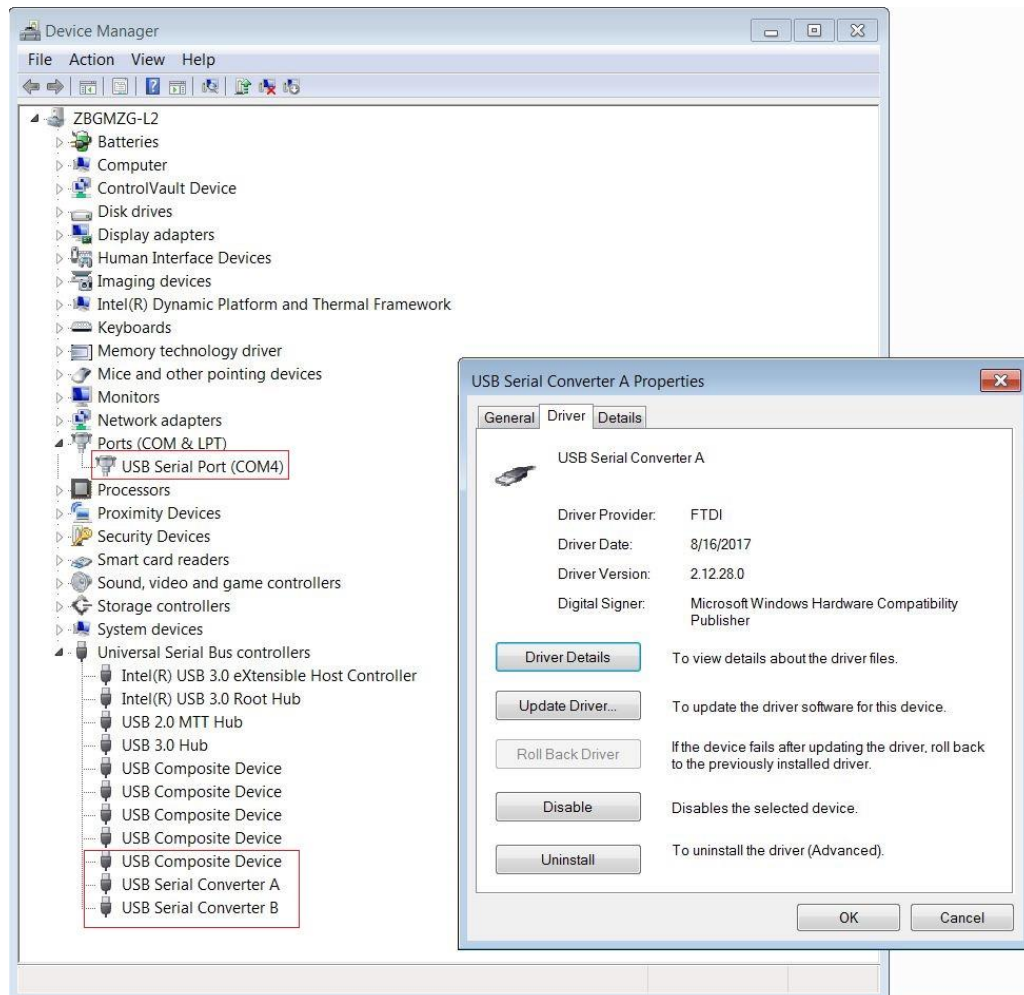


```
Microsoft Windows [Version 10.0.16299.1087]
(c) 2017 Microsoft Corporation. All rights reserved.

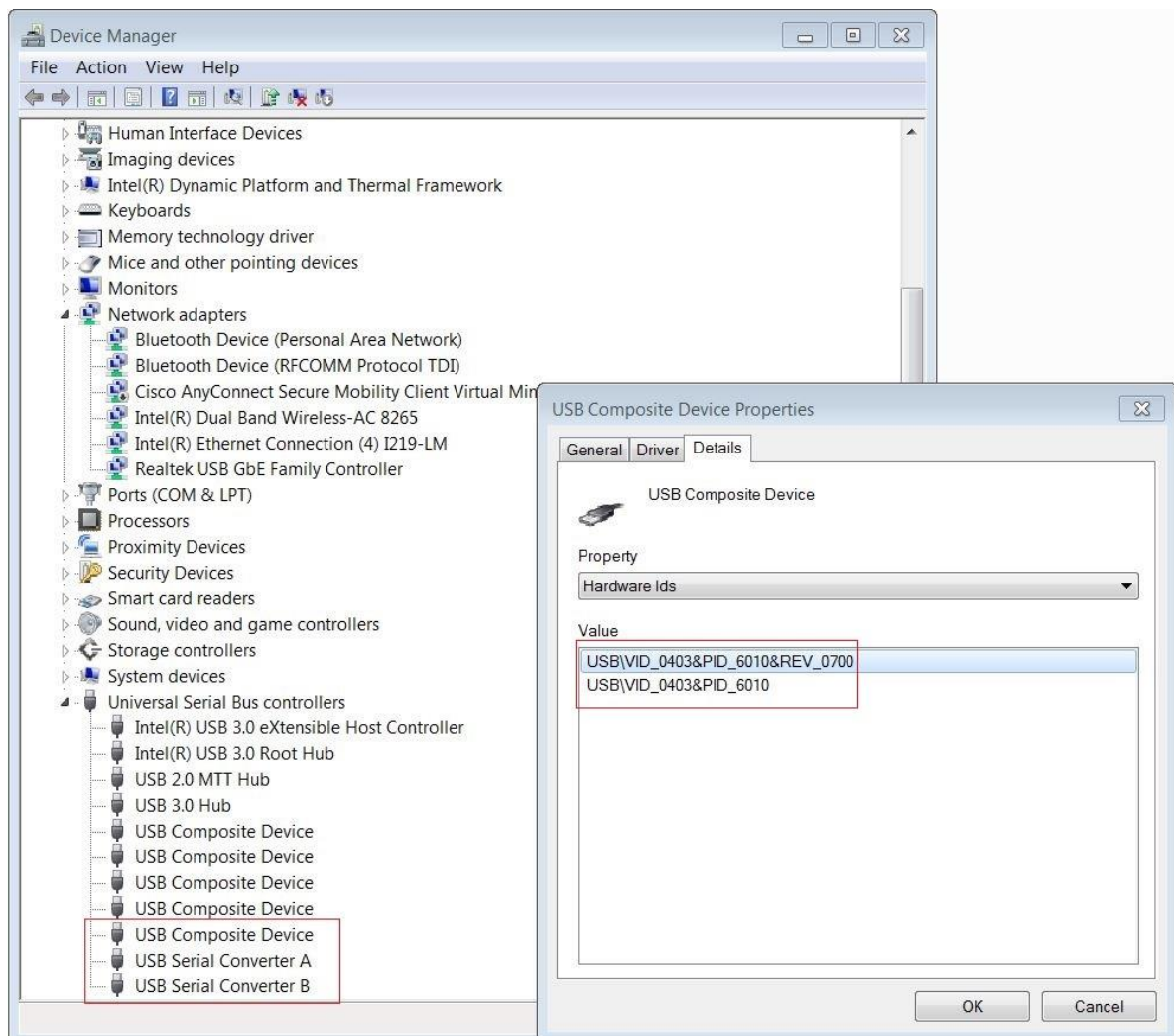
C:\Users\fg8y4b>axsdb
ON Semiconductor AG Microfoot Debugger version 1.21
Targets: 00000003
%
```

AXSDB should print the serial number of a target after the “Targets:” prompt. The top two reasons for an empty line after “Targets:” are that no debug adapter is connected to a working USB port, or that the USB drivers have not been correctly installed.

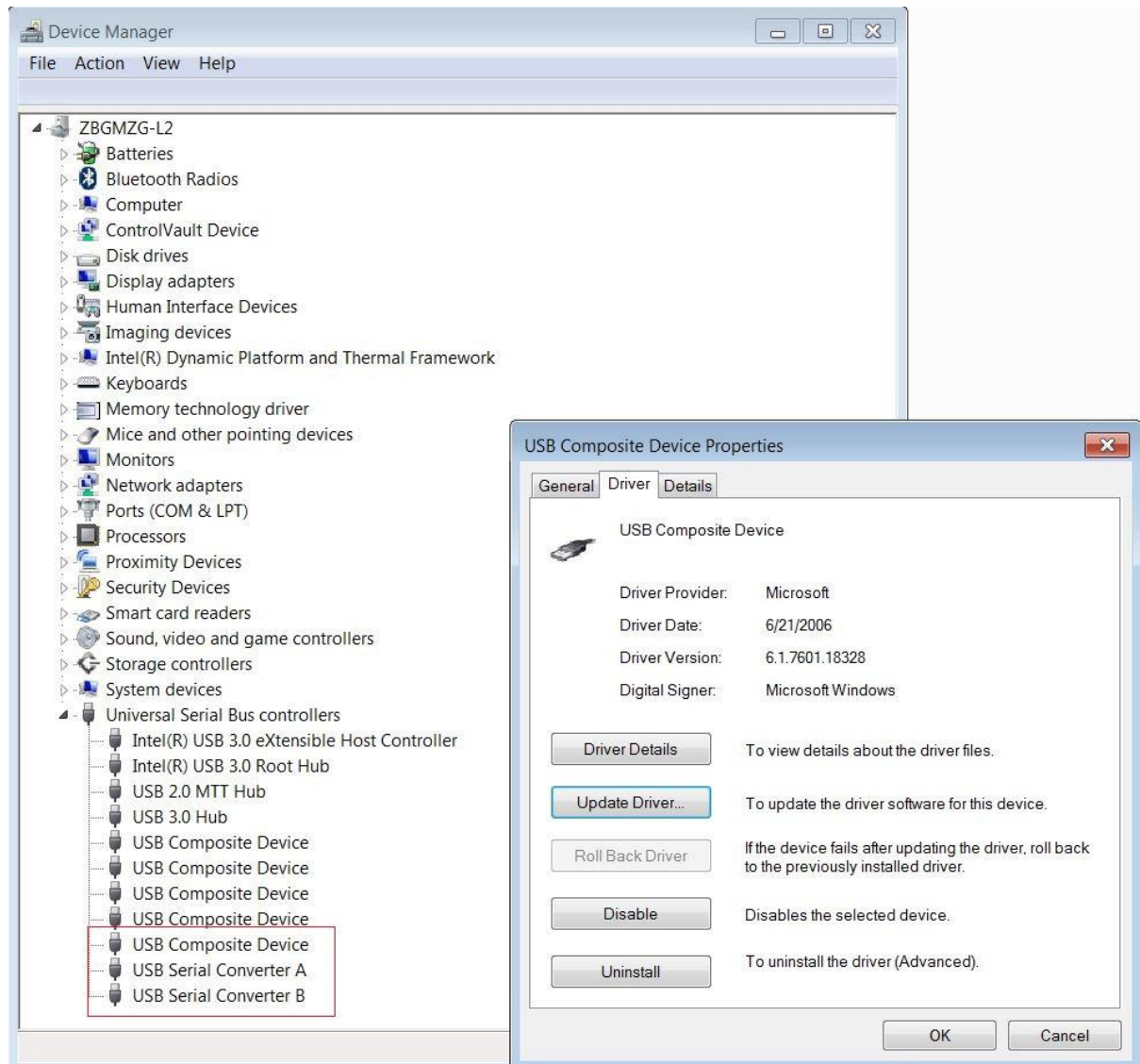
To check whether the drivers are correctly installed, open the device manager (The driver installation is common for both AX8052 and AXM0F243)



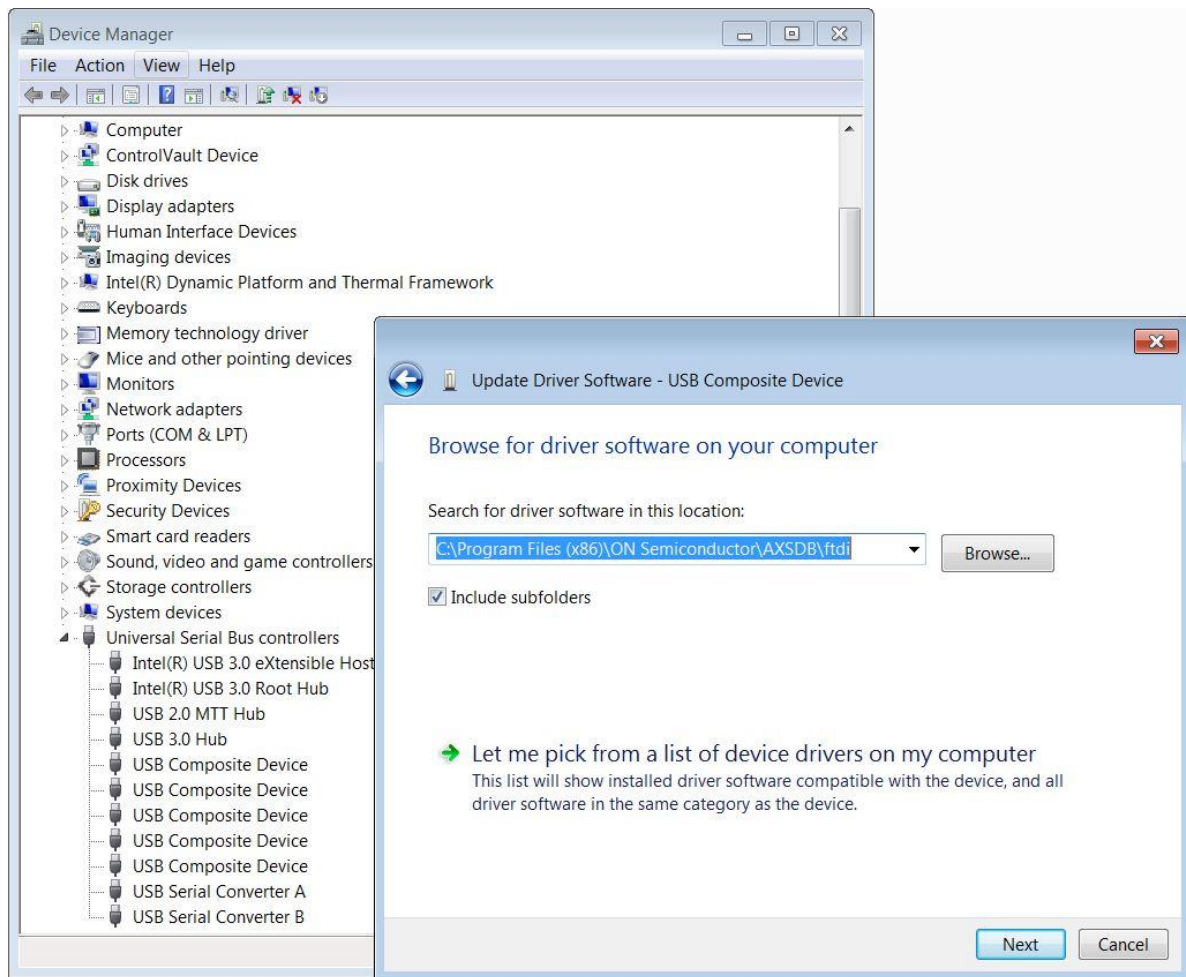
For the AXDBG debug adapter USB device, the Vendor ID (VID) is 0403 and the Product ID (PID) is 6010 as shown below.



If the drivers are not correctly installed, there will be a yellow exclamation mark on "USB Serial Converter A" or "USB Serial Converter B" or "USB Composite Device" which indicates that the driver has not been correctly installed. Right-click on the device with the exclamation mark, and choose Update driver.



Newer Versions of Windows, if connected to the internet, offer the option to search Windows Update for a suitable driver. You can choose this option; it is the easiest option, though may take some time. As an alternative, the installer also puts suitable drivers into C:\Program Files (x86)\ON Semiconductor\AXSDB\ftdi. You can choose to install from this directory and its subdirectories as well.



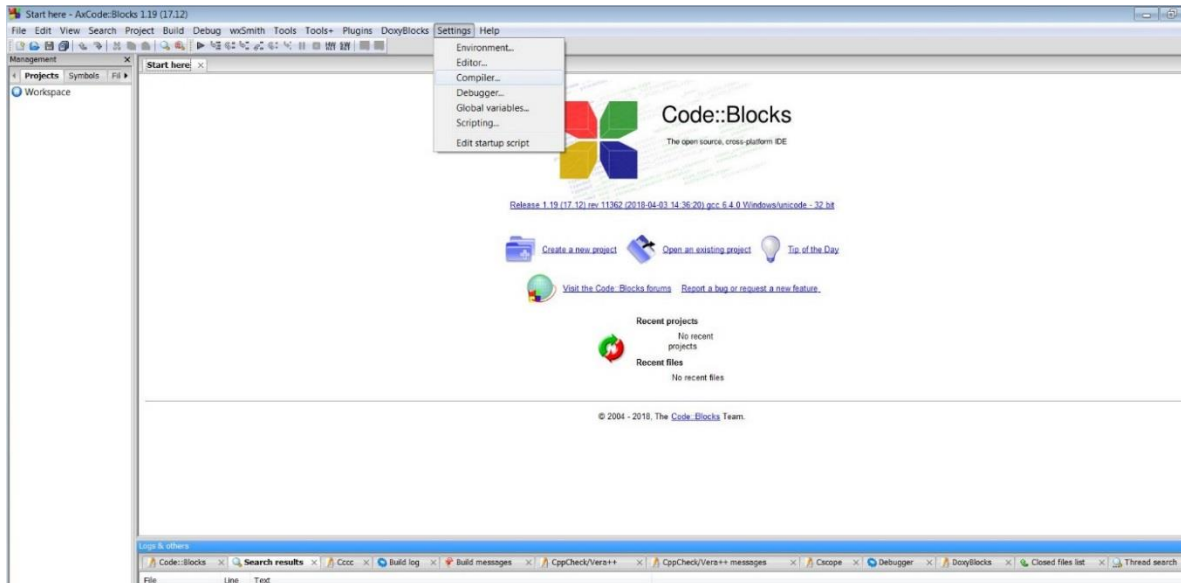
After successfully re-installing the drivers, there should not be any yellow exclamation marks on "USB Serial Converter A" or "USB Serial Converter B" or "USB Composite Device". After installing the drivers, you should reboot Windows.

Check that AXSDB now recognizes the target.

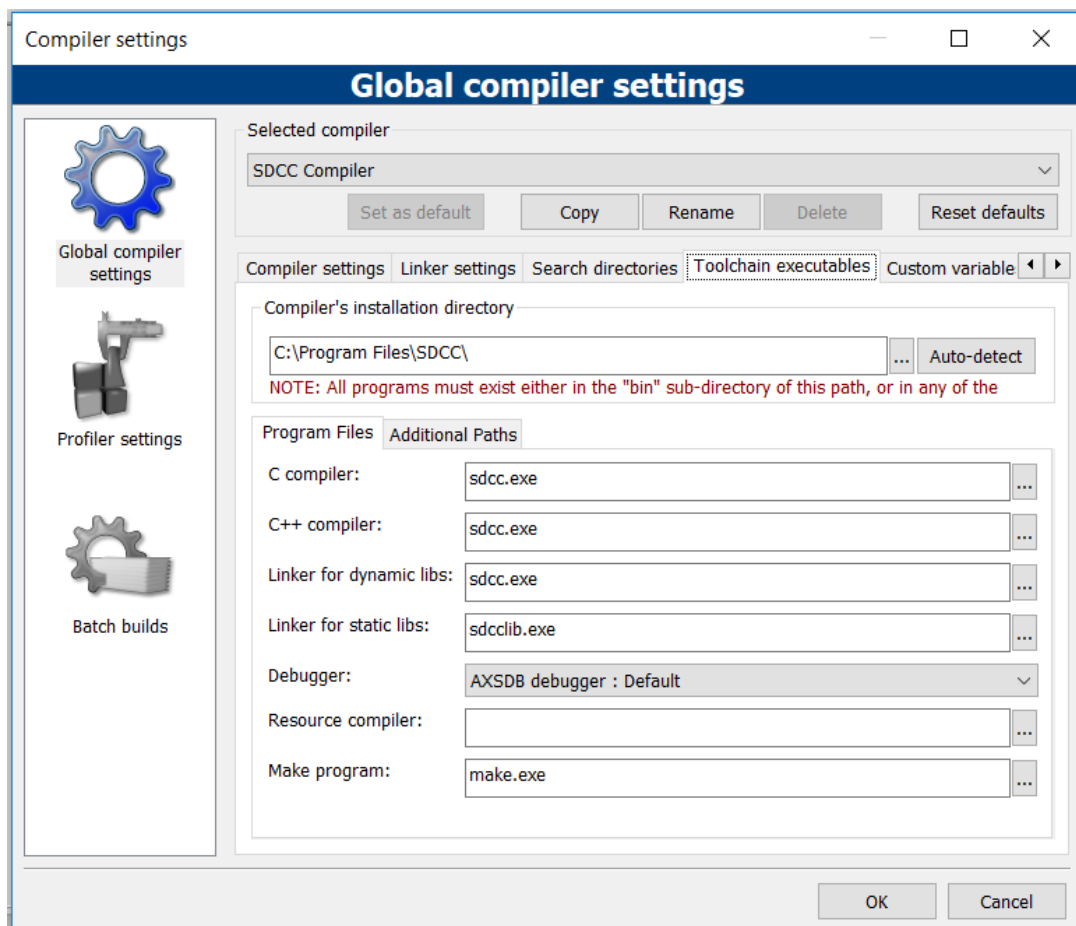
AX8052:

If the debugger plugin is not correctly configured, see below for how to configure it for AX8052 microcontroller.

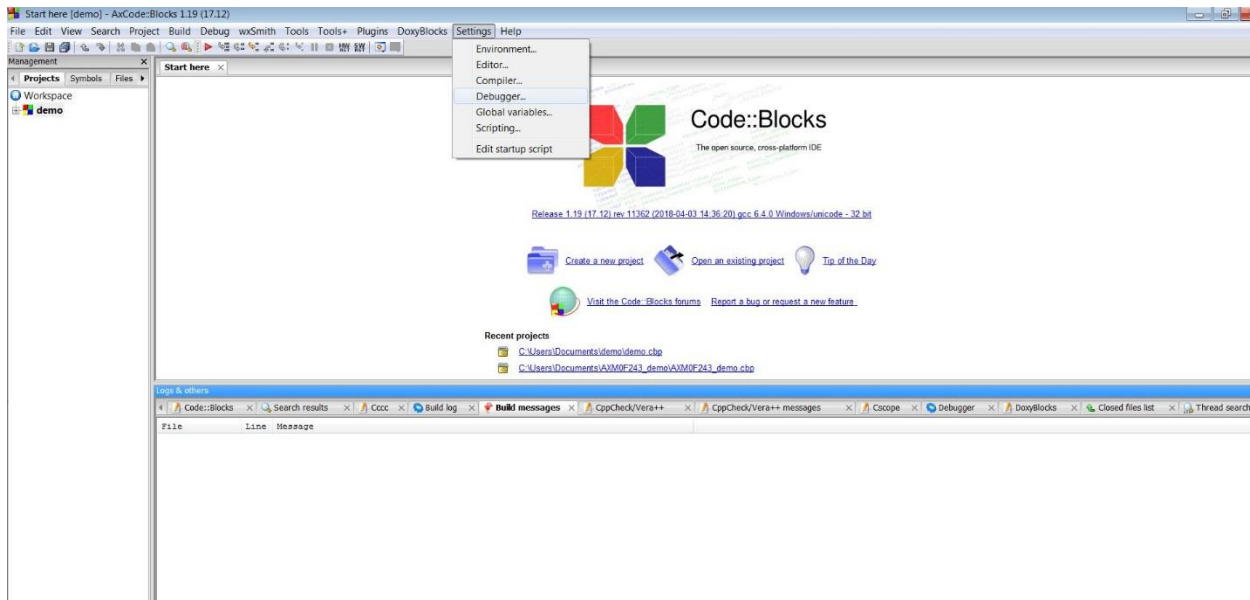
Start AxCode::Blocks. Verify that the correct debugger plugin is selected. To do this, open the compiler settings window.



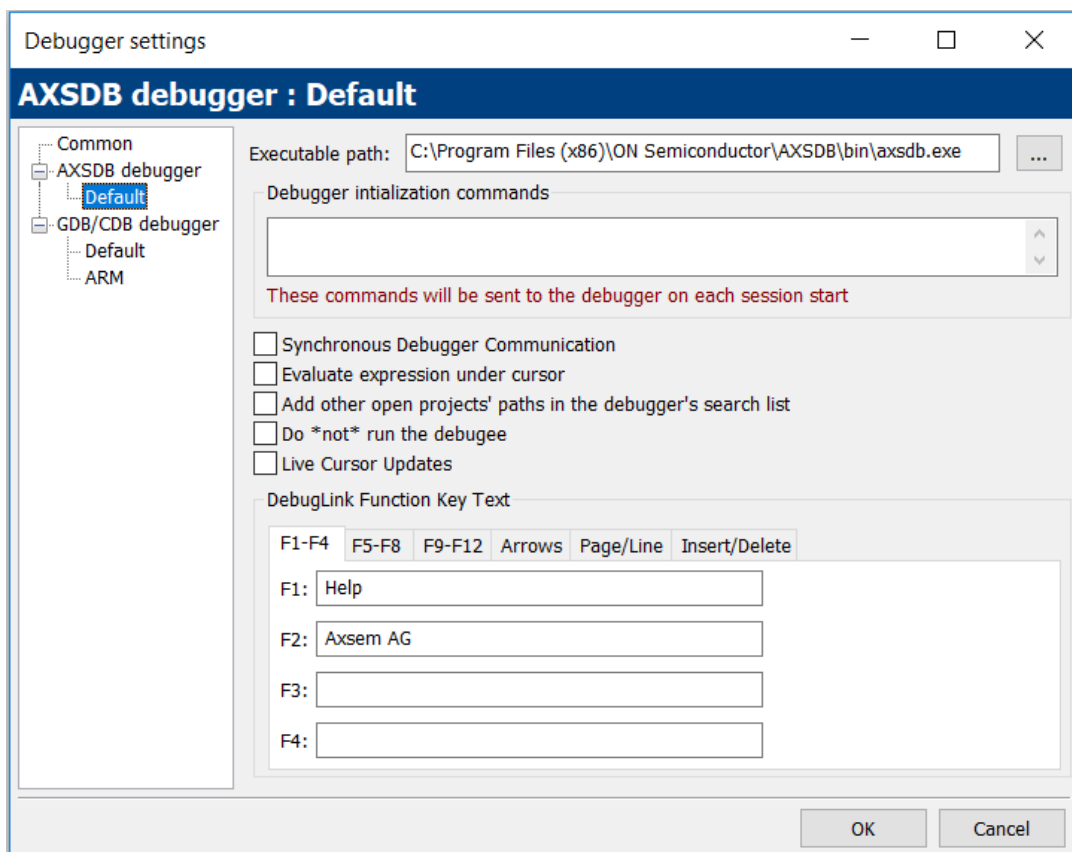
Check that the debugger plugin named "AXSDB debugger: Default" is selected.



Now check that the correct axsdb binary is configured in the plugin configuration. To do this, open the debugger settings.



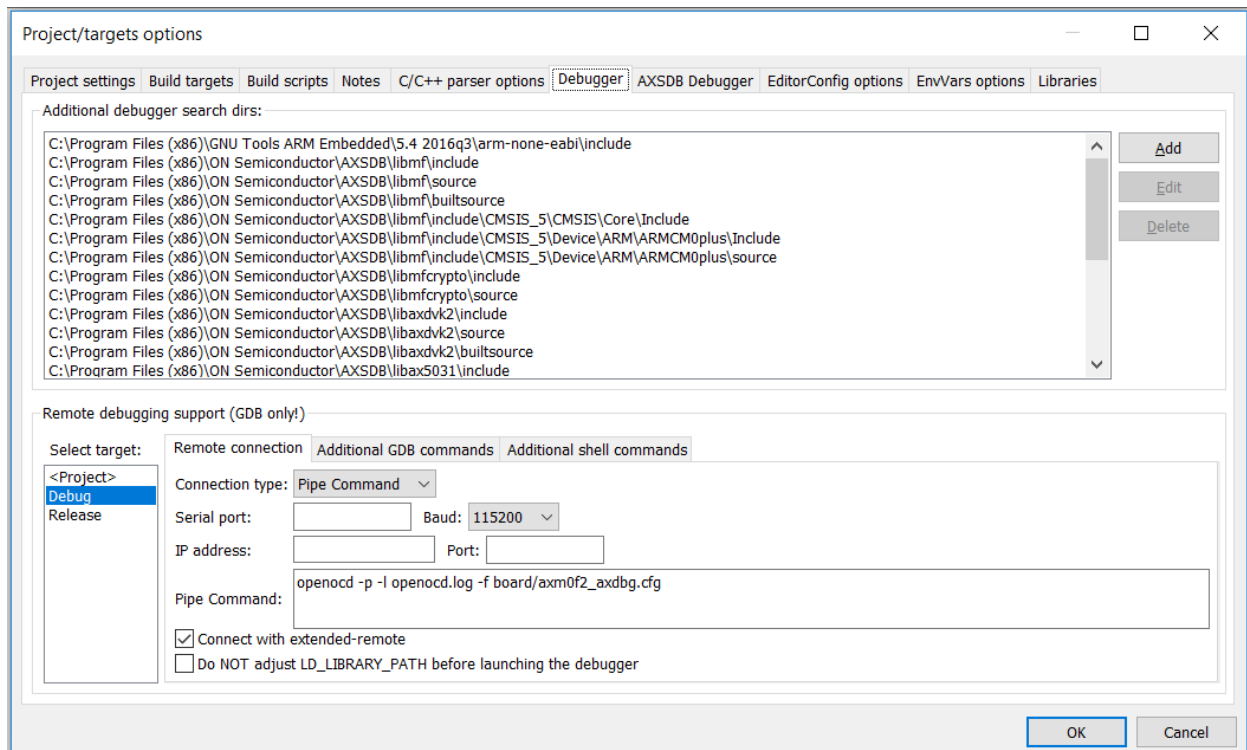
Select AXSDB debugger – default. Check that the Executable path is correct.



AXM0F243:

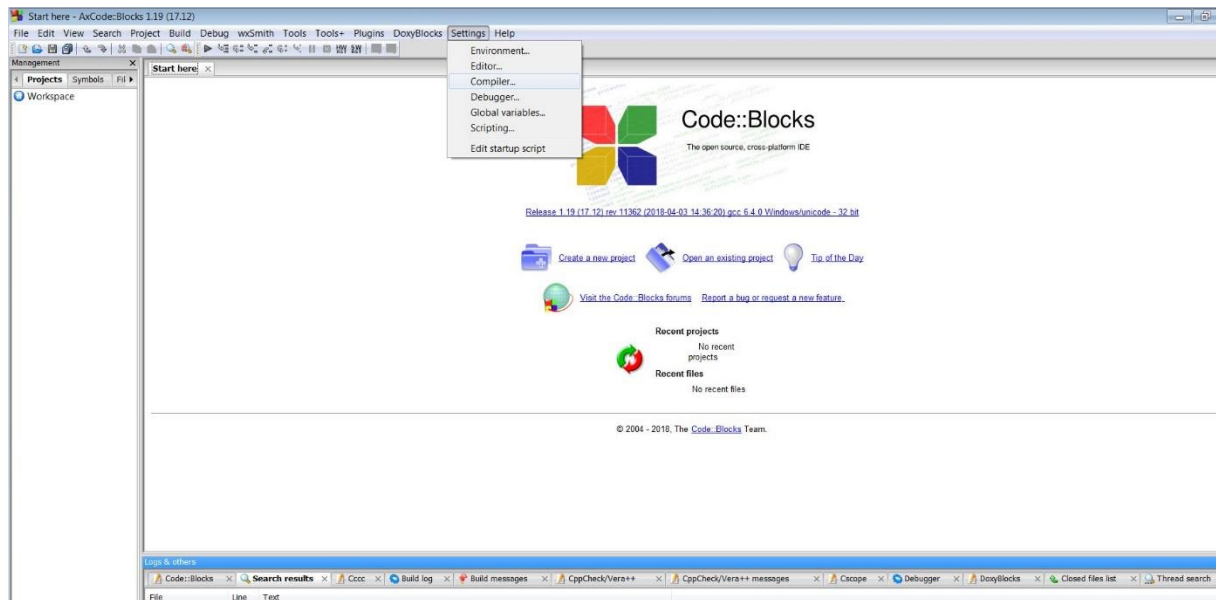
Check that the OpenOCD.exe debug interface software is available in the installation directory where the you have installed the AXSDB followed by "\\bin" ("C:\\Program Files (x86)\\ON Semiconductor\\AXSDB\\bin")

Make sure that the name of the configuration file (.cfg) and the path to the file is given correctly. By default, the configuration file for AXM0F243 microcontroller is available in the directory "C:\\Program Files (x86)\\ON Semiconductor\\AXSDB\\share\\openocd\\scripts\\board"

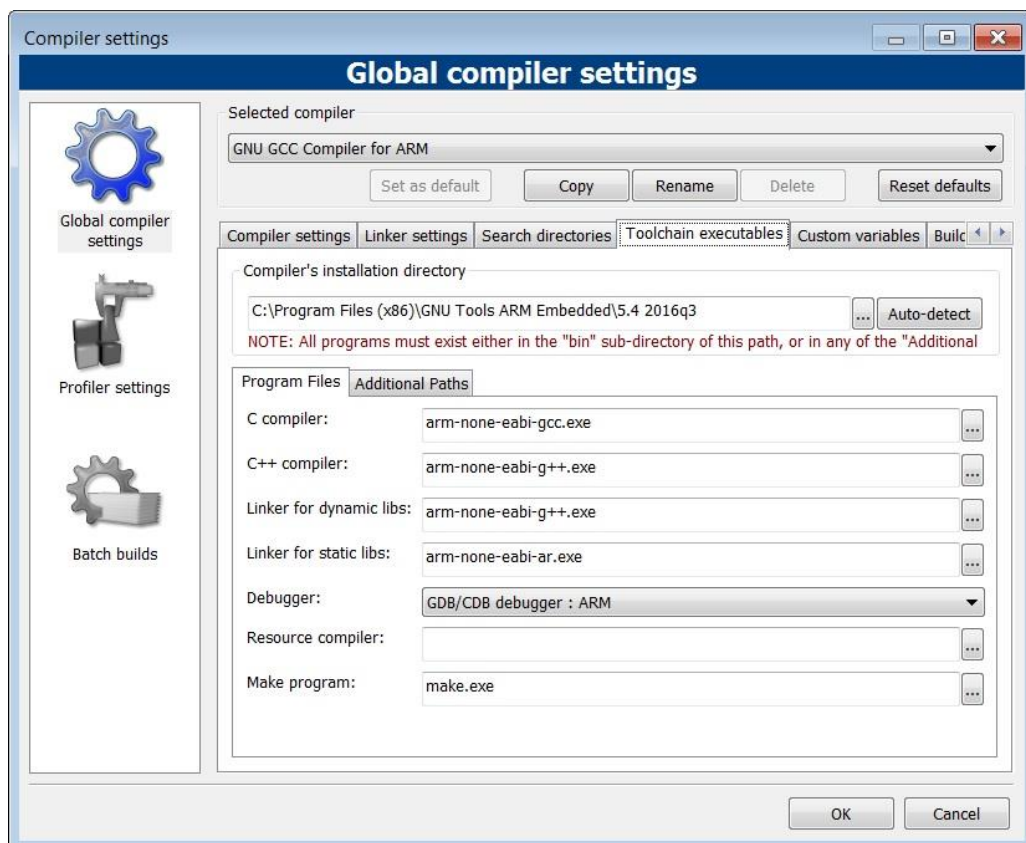


If the debugger plugin is not correctly configured, see below for how to configure it for AXM0F243 microcontroller.

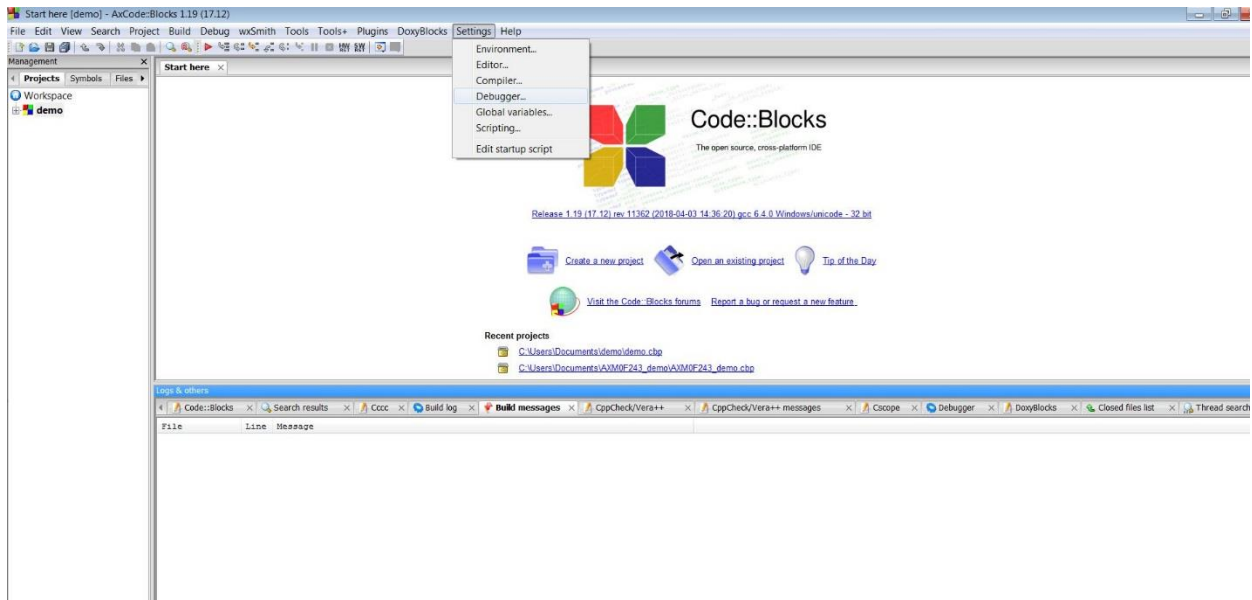
Start AxCode::Blocks. Verify that the correct debugger plugin is selected. To do this, open the compiler settings window.



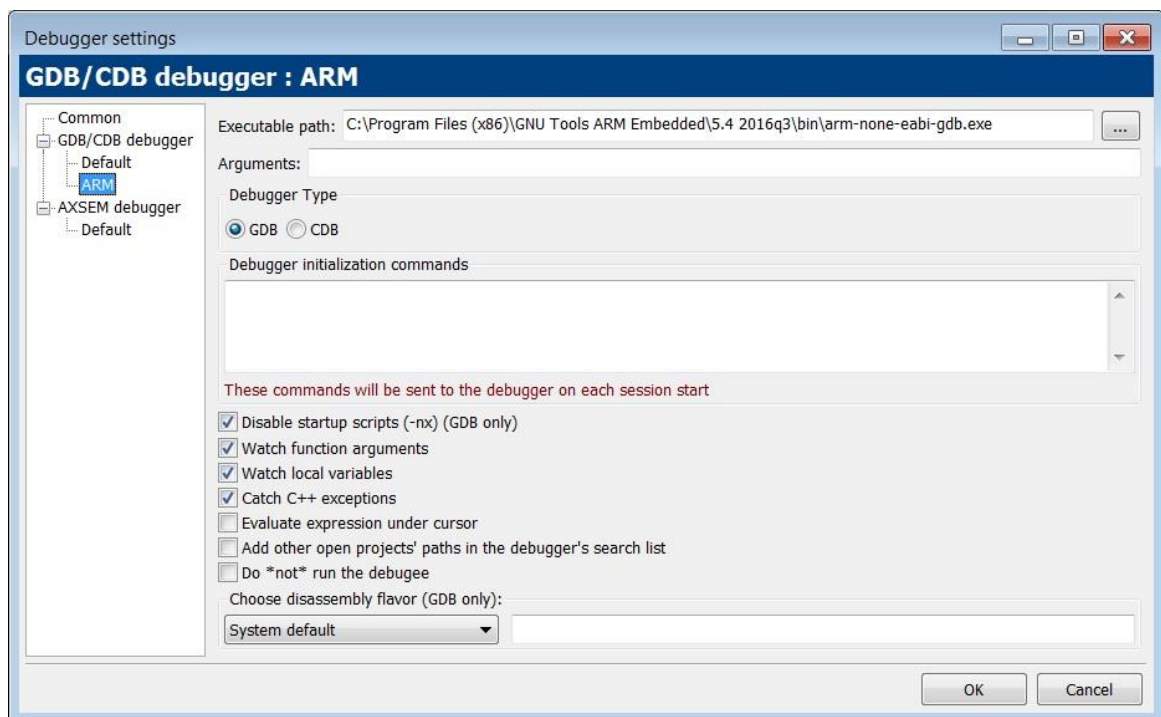
Check that the debugger plugin named "GDB/CDB debugger : ARM" is selected.



Now check that the correct binary is configured in the plugin configuration. To do this, open the debugger settings.



Select GDB/CDB debugger: ARM. Check that the Executable path is correct.



12. HISTORY


Version	Date	Comments
1.00	28-Aug-18	Draft version
1.01	28-Aug-18	Added AX8052 and AXM0F243 details
1.02	24-May-19	Updated section 7.3 Open OCD commands details

13. CONTACT INFORMATION

ON Semiconductor
Oskar-Bider-Strasse 1
CH-8600 Dübendorf
SWITZERLAND

Phone +41 44 882 17 07
Fax +41 44 882 17 09
Email sales@onsemi.com
www.onsemi.com

For further product, related or sales information please visit our website or contact your local representative.

ON Semiconductor and  are trademarks of Semiconductor Components Industries, LLC dba ON Semiconductor or its subsidiaries in the United States and/or other countries. ON Semiconductor owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of ON Semiconductor's product/patent coverage may be accessed at www.onsemi.com/site/pdf/Patent-Marking.pdf. ON Semiconductor reserves the right to make changes without further notice to any products herein. ON Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does ON Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using ON Semiconductor products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by ON Semiconductor. "Typical" parameters which may be provided in ON Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. ON Semiconductor does not convey any license under its patent rights nor the rights of others. ON Semiconductor products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use ON Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold ON Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that ON Semiconductor was negligent regarding the design or manufacture of the part. ON Semiconductor is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.