ON Semiconductor®

Application Note **AND9371/D**

AxCode::Blocks Quick Guide

Revision 2

Table of Contents

1. Introduction

Figure 1 shows a diagram of the ON Semiconductor AX8052 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 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). 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.

The Debug Adapter provides the link between the developers workstation and the target board.

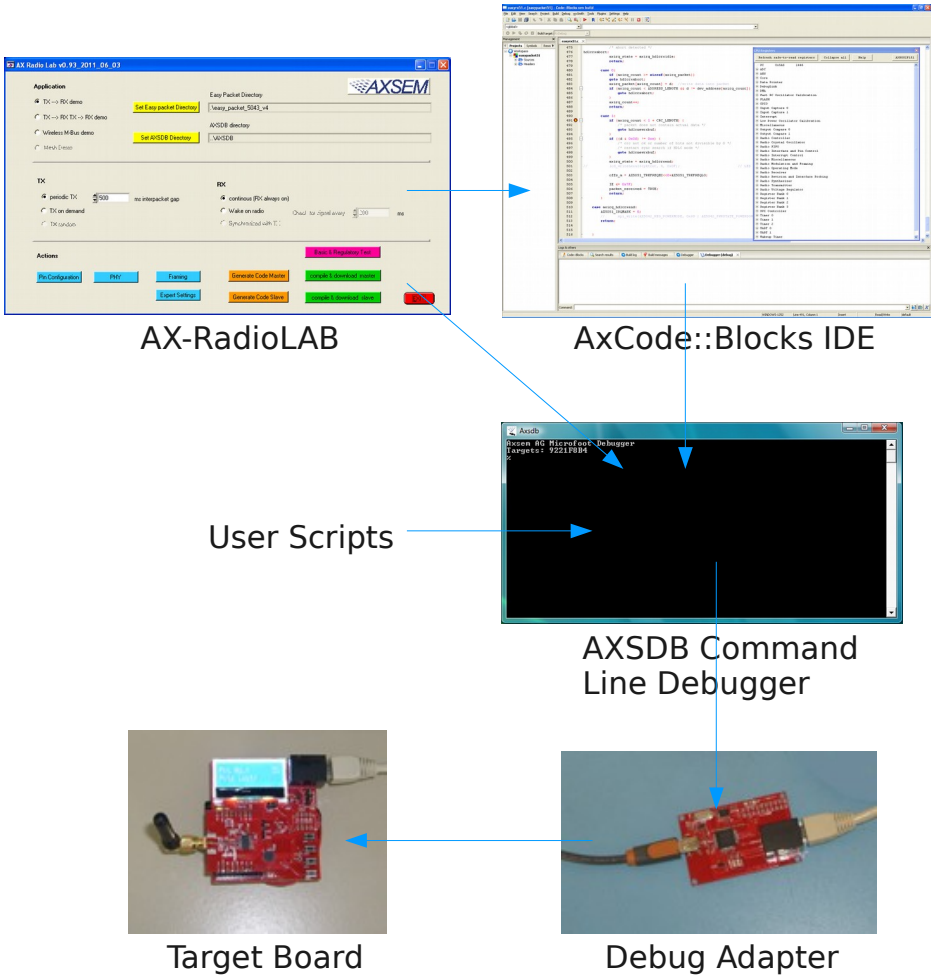


Figure 1: ON Semiconductor AX8052 Development System Architecture

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 installer contains everything you need: the SDCC compiler, the AXSDB debugger, example files and libraries and, of course, AxCode::Blocks.

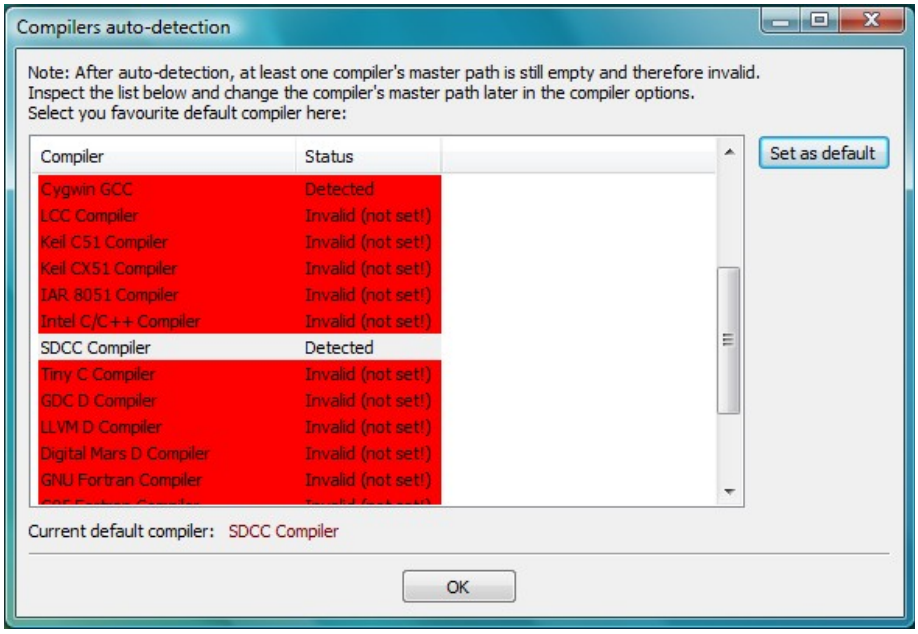
- Launch the installer.
- After accepting the terms of agreement you are asked to select the components to be installed. We strongly suggest to install all components. Failure to do so could result in missing links in your toolchain.
- Choose where to install AXSDB (it is recommended to keep the default 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. Again, the installation is pretty intuitive. Be sure to tick the option for adding the SDCC directory to the system path.
- Wait until the process finishes and you are done!

3. Connecting the Hardware

Please refer to the application note *AX-RF-DVK2 Quick Start Instructions* available from the ON Semiconductor website: http://www.onsemi.com

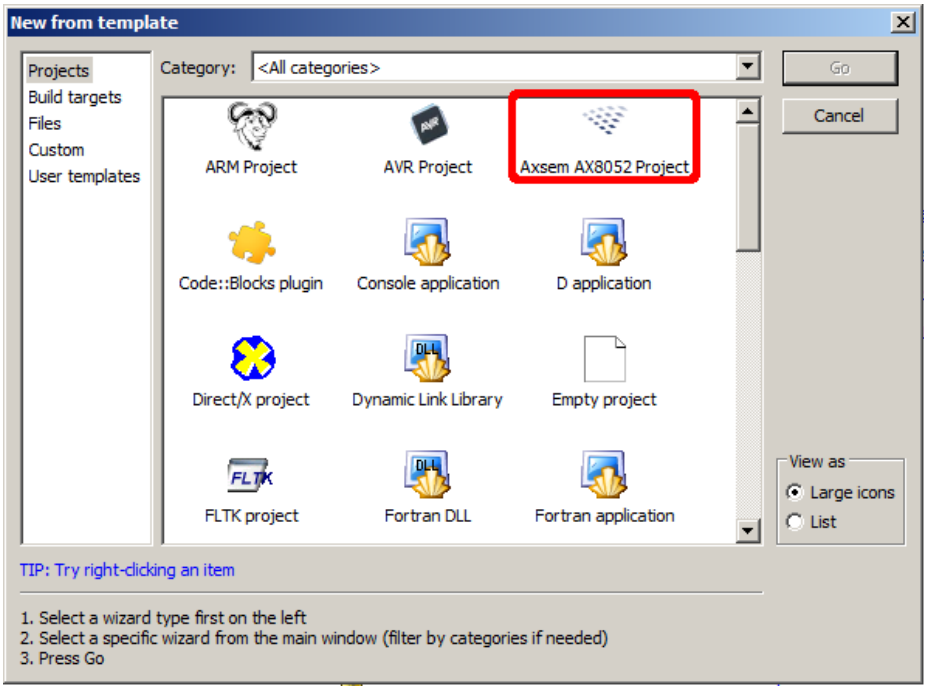
4. Creating a New Project

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.

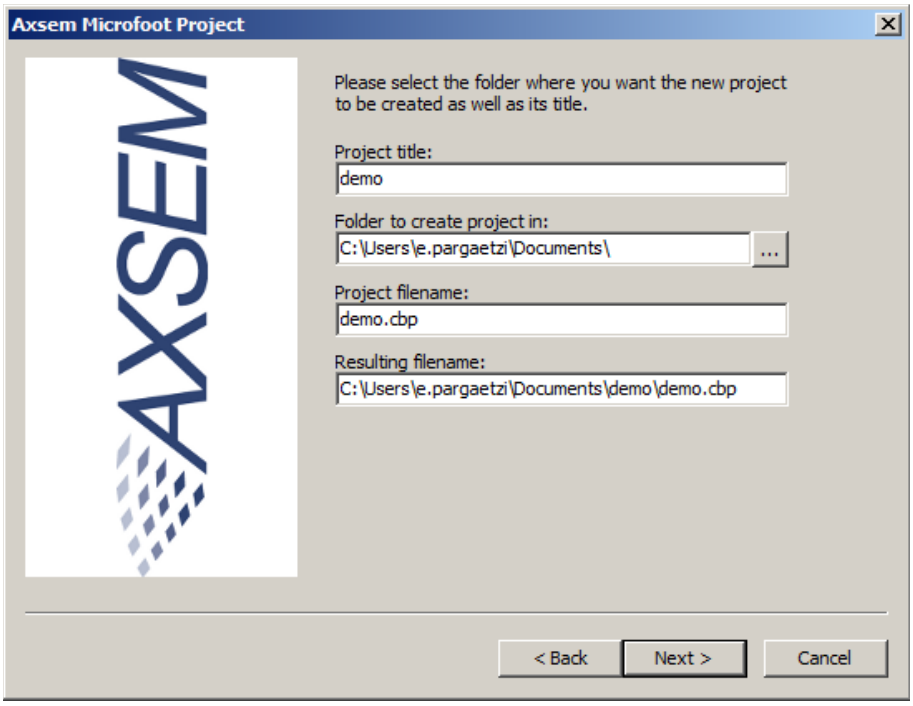


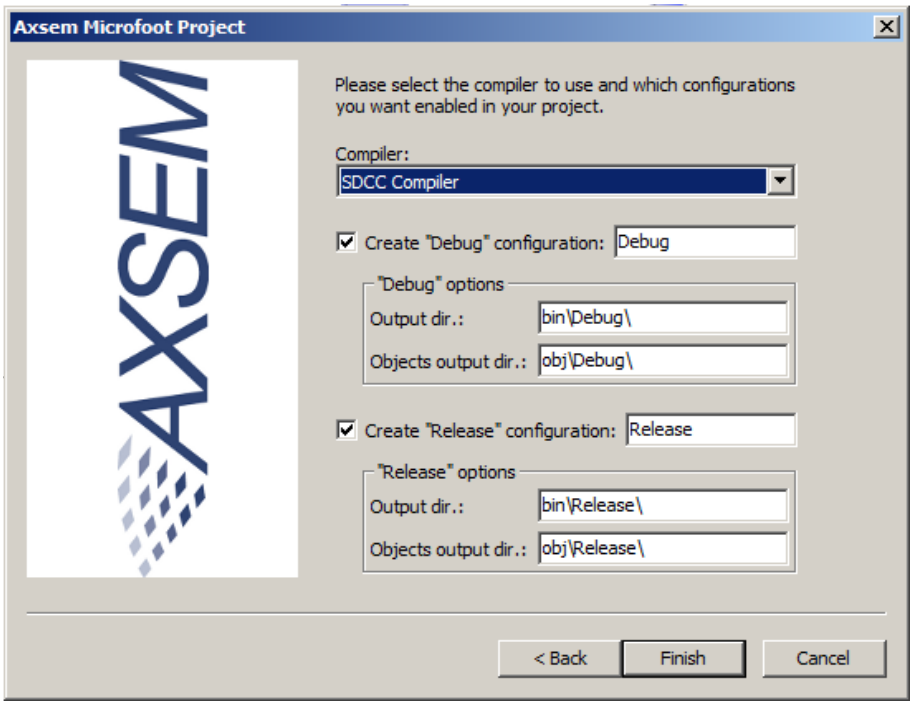
Click on *File → New → Project.*

Choose *Axsem AX8052 Project:*



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



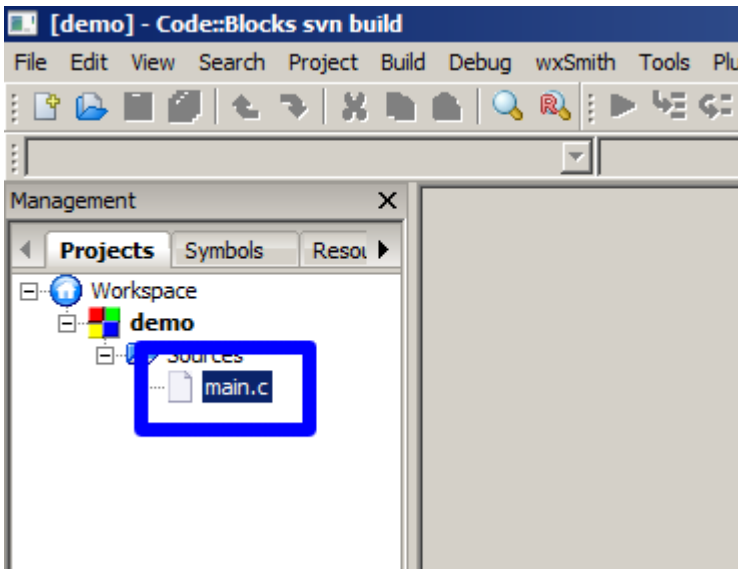


Clicking on *Finish* will create the new project. For your convenience the most important build options and compiler preferences are set automatically.

ON Semiconductor also distributes example projects. Check the ON Semiconductor homepage for the available examples.

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:



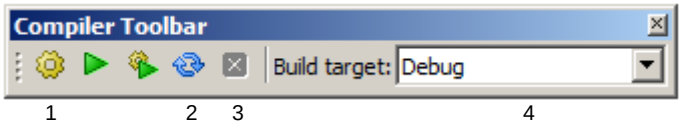
You can add existing files to the project using the menu entry *Project → Add Files.*
You can add new files to the project using the menu entry *File → New → File.*

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

Warning: SDCC seems to have a bug. Do not use hyphens in your filenames. Underscores are fine.

6. Compiling the Project

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



①	*Build*	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. Debugging the Project

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

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 issued[1]. If exactly one device is found, the device is automatically connected. If more than one device is found, the user is prompted to select one.

Furthermore, the user can decide to load the firmware to the device or to use the firmware already stored in it.

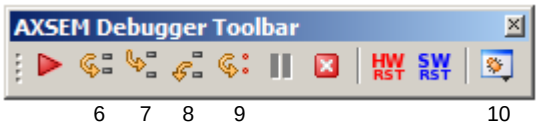
If changes are made to the project since the last build, the project is automatically compiled.

When running, the toolbar changes it's appearance to

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 “Microfoot Debug Interface”, 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\AXSEM\AXSDB\ftdi”. Eventually, you need to disconnect the device and restart AxCode::Blocks.

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. Button ③ does not stop the execution, but disconnects the device and exits the debugger. ④ and ⑤ are used to reset the microcontroller.

After hitting the pause button ②, other functions become enabled:



⑥ *Next line* Execute the next line in the sourcecode
⑦ *Step into* Execute the next line, if it's a function step into it
⑧ *Step out* Continue execution until the end of the current frame
⑨ *Next instruction* Execute the next assembly instruction

Due to limitations in the debug information of 8052 compilers, the stepping commands ⑥, ⑦ and ⑧ 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 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 ⑩ can be used to open the debugging windows described in the next chapter.

8. Debugging Windows

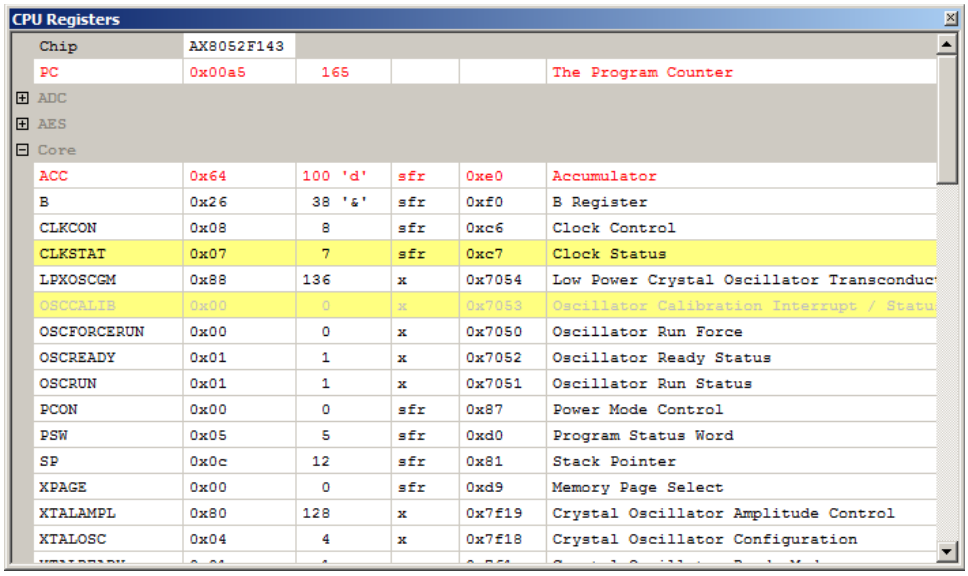
8.1. Breakpoints

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 bookmark.

8.2. Registers

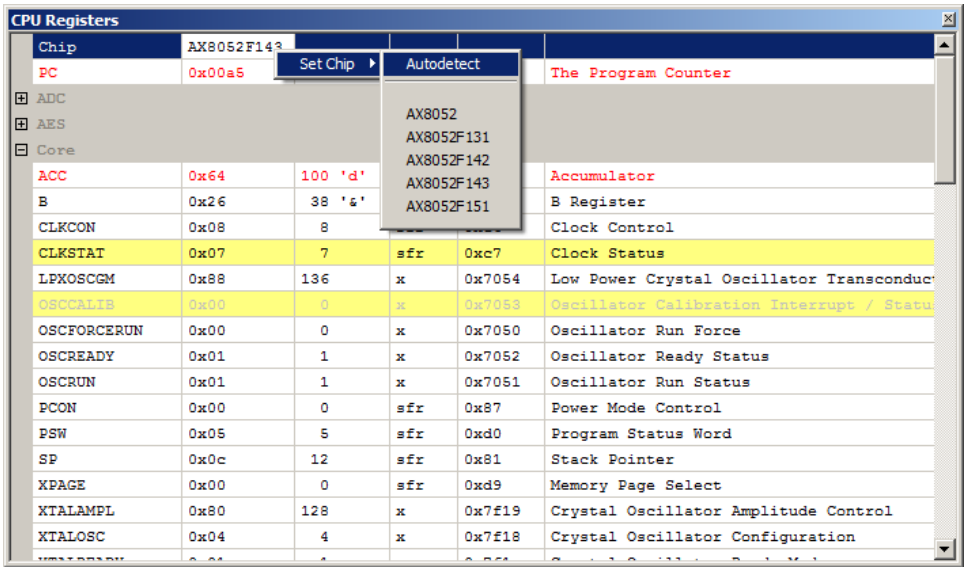
This window shows Microcontroller register contents while the Microcontroller is stopped. Registers of ON Semiconductor Radio Chips or SoC functions are also displayed.

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.



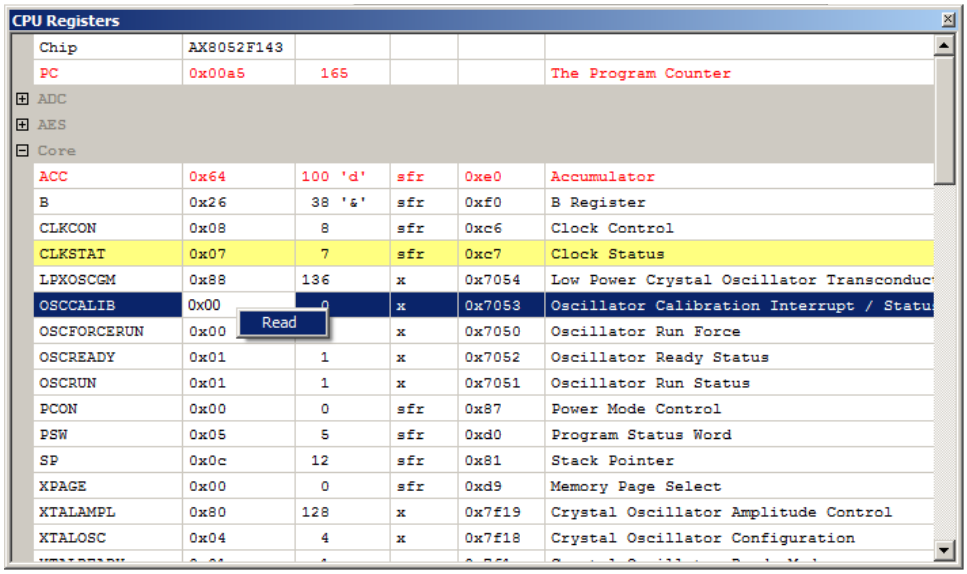
Chip	AX8052F143				
PC	0x00a5	165			The Program Counter
⊞ ADC					
⊞ AES					
⊟ Core					
ACC	0x64	100 'd'	sfr	0xe0	Accumulator
B	0x26	38 '&'	sfr	0xf0	B Register
CLKCON	0x08	8	sfr	0xc6	Clock Control
CLKSTAT	0x07	7	sfr	0xc7	Clock Status
LPXOSCGM	0x88	136	x	0x7054	Low Power Crystal Oscillator Transconduc
OSCCALIB	0x00	0	x	0x7053	Oscillator Calibration Interrupt / Statu
OSCFORCERUN	0x00	0	x	0x7050	Oscillator Run Force
OSCREADY	0x01	1	x	0x7052	Oscillator Ready Status
OSCRUN	0x01	1	x	0x7051	Oscillator Run Status
PCON	0x00	0	sfr	0x87	Power Mode Control
PSW	0x05	5	sfr	0xd0	Program Status Word
SP	0x0c	12	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

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.

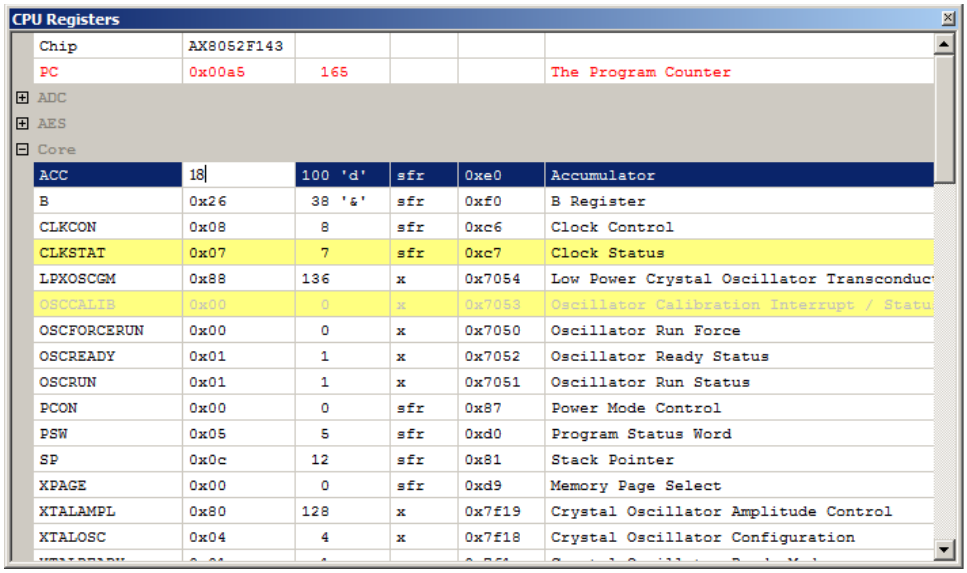


Registers and their contents are shown in the remaining rows. The first column shows the register name. The second column shows the current register contents as a hexadecimal number. The third column displays the register contents as a decimal number; if this number is greater or equal 32, it is also 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.

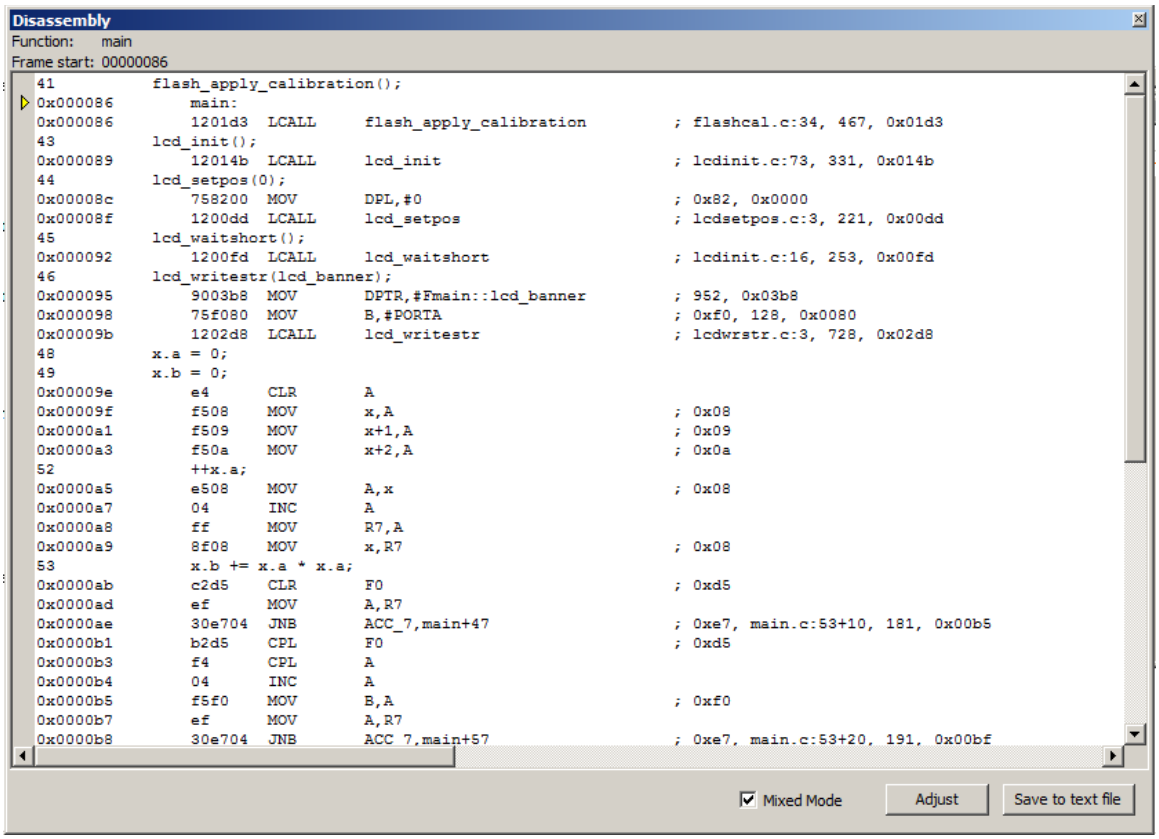


Registers can be (re-)read by either right-clicking into the row and selecting Read, or by selecting the row and pressing the space-bar.



Register 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).

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.

8.4. Memory dump

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

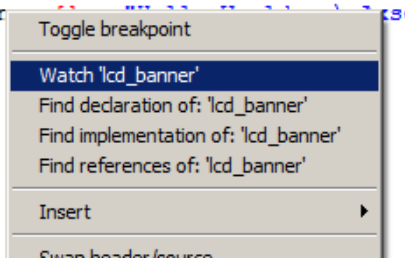
8.5. Watches

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:

```
d main(void)

 static const char __code lcd_bar
                                                    sem
ndef SDCC
 _sdcc_external_startup();
dif

 lcd_init();
 lcd_setpos(0);
 lcd_waitshort();
```



Watches (new)

⊟ x		struct { char a; unsigned short b; }	iramlow	0x0009
a	99	char		
b	36254	unsigned short		
⊟ lcd_banner	Hello Wo ...	char[21]	codestatic	0x0252
[0]	72	char		
[1]	101	char		
[2]	108	char		
[3]	108	char		
[4]	111	char		
[5]	32	char		
[6]	87	char		
[7]	111	char		
[8]	114	char		
[9]	108	char		
[10]	100	char		
[11]	46	char		
[12]	46	char		
[13]	46	char		
[14]	10	char		
[15]	65	char		
[16]	120	char		
[17]	115	char		
[18]	101	char		
[19]	109	char		
[20]	0	char		

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

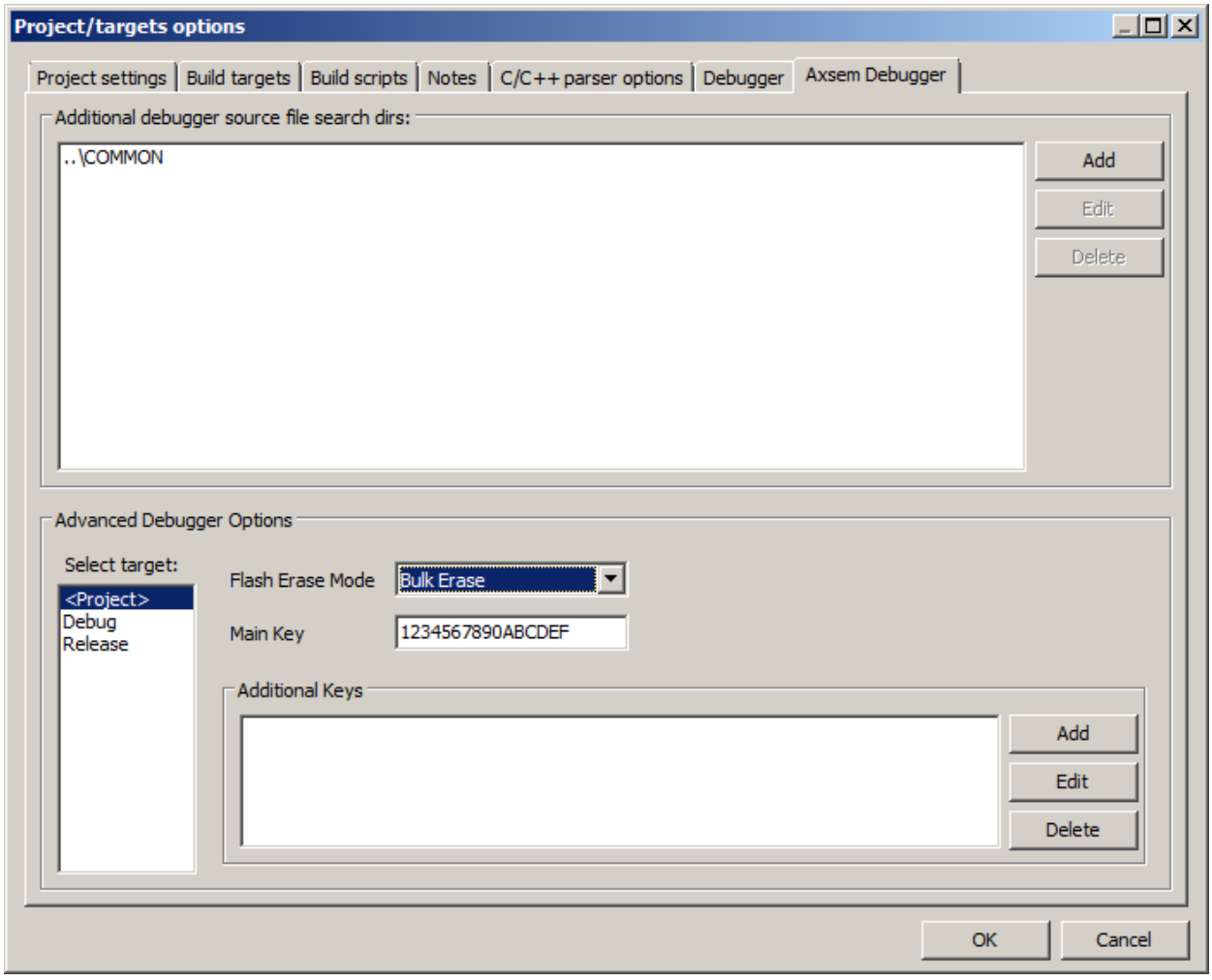
This function emulates the two pins of the microcontroller occupied by the debugger. The direction (input or output) is automatically detected. For both directions, the logic state of the pin is shown. Additionally, the state of inputs can be toggled.

8.7. Debuglink

This window provides a graphical front-end to the debuglink, 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.

9. Advanced Debugger Configuration

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

10. ON Semiconductor Project Wizard

The new project wizard supports the creation of a skeleton project template for ON Semiconductor microcontroller projects. Both SDCC and IAR compilers are fully supported, with selectable code models.

The code structure of the 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 elsa 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;

   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.

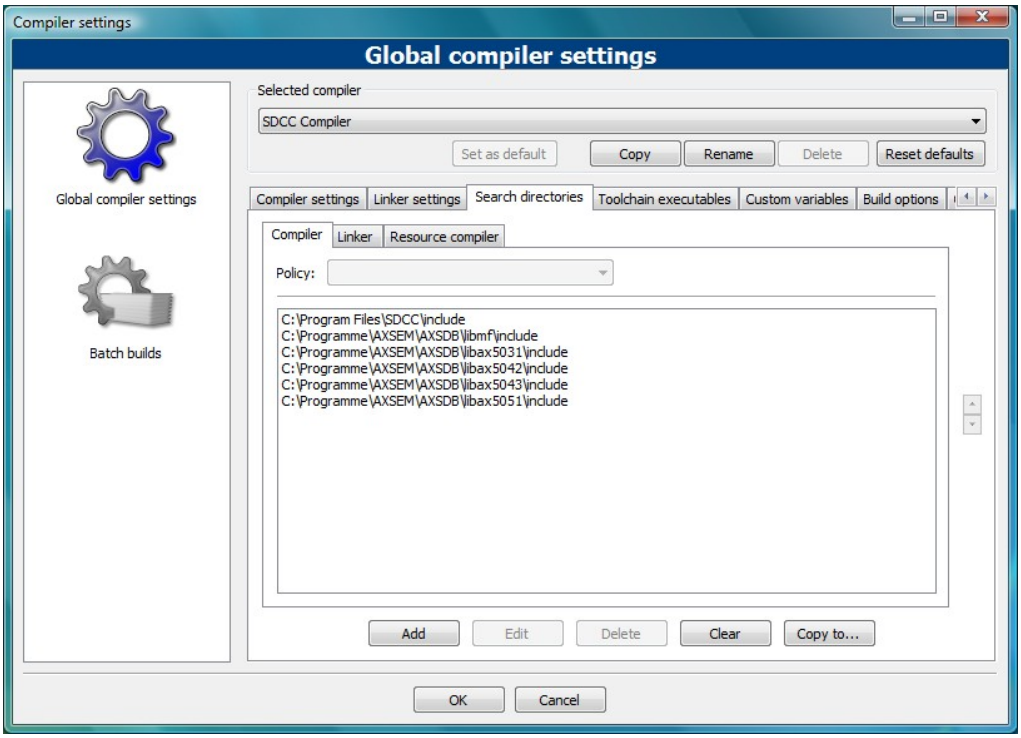
- Keil

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.

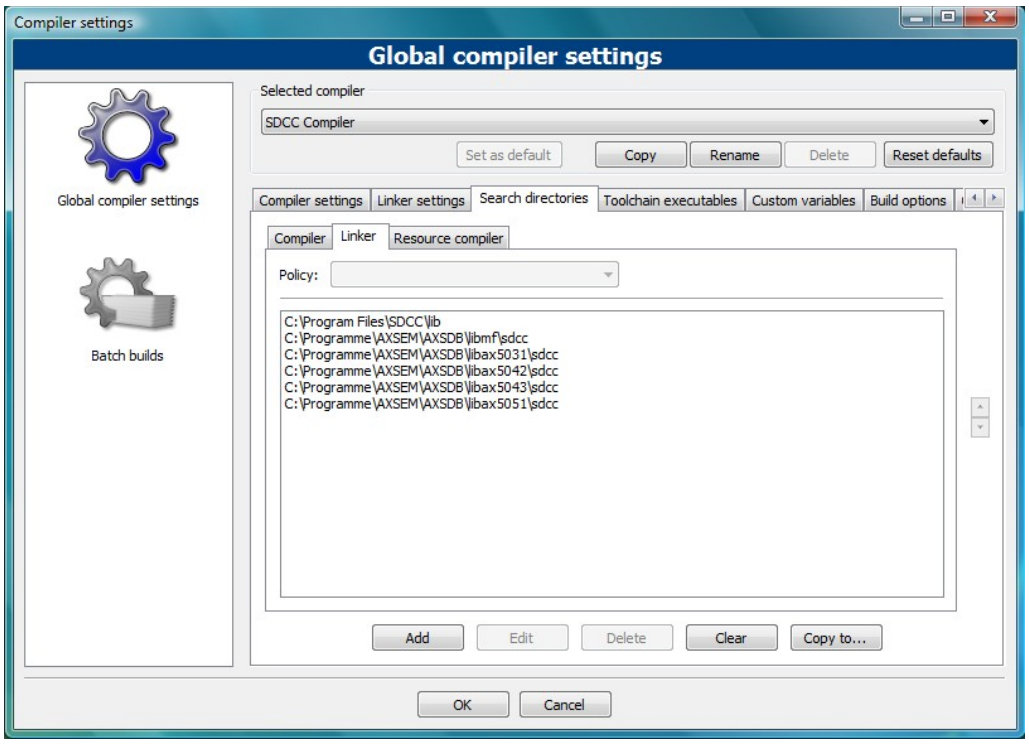
11. Troubleshooting Guide

11.1. Compiler Auto-detection fails on first start

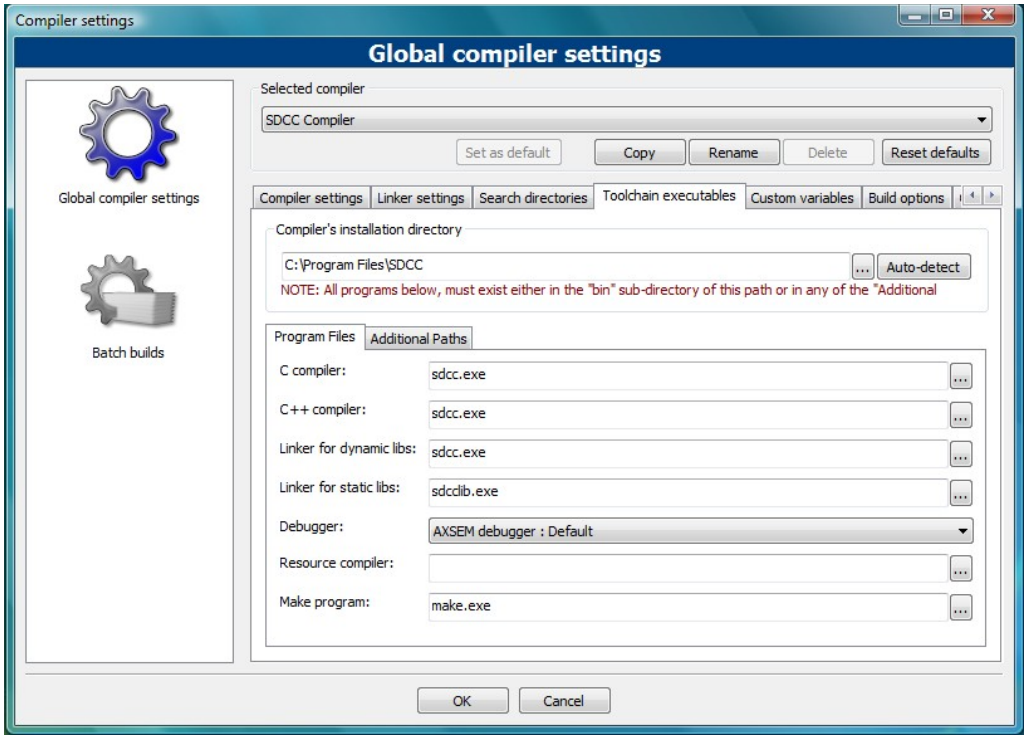
When AxCode::Blocks is run for the first time, it tries to auto-detect the installers. If the auto-detection fails, configure the compiler (SDCC) manually as shown in the screen shots below. The path C:\Program Files\ must be replaced by the actual installation path if a non-default installation has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



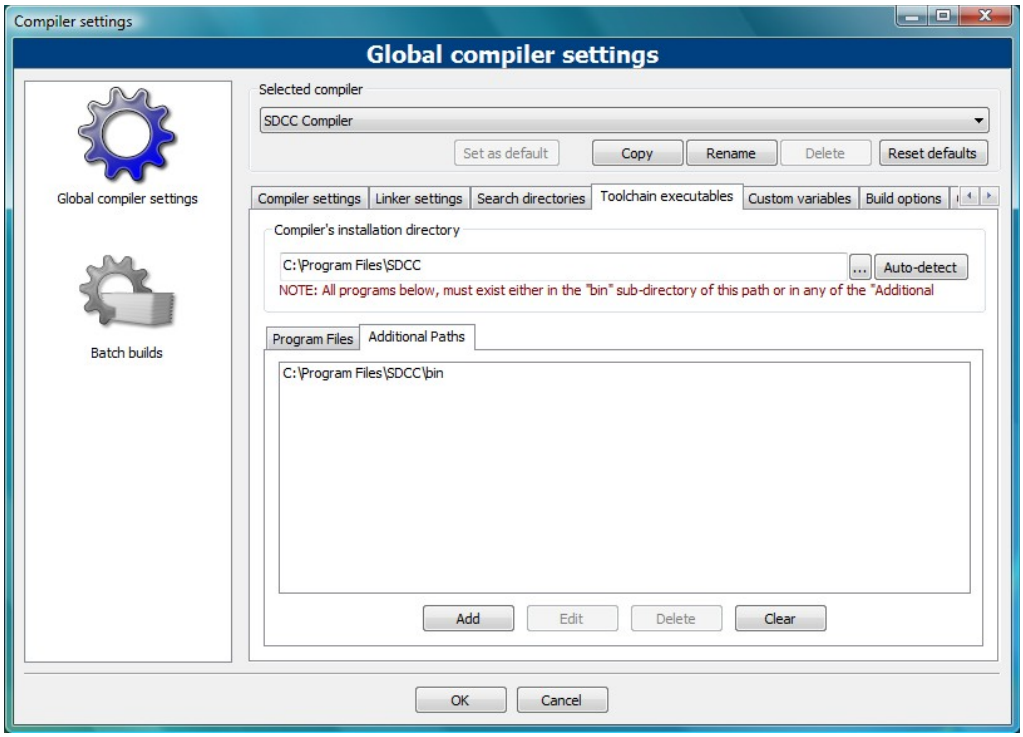
Open the compiler settings dialog by clicking on Settings→Compiler. Select SDCC (under selected compiler). Click on “Set as default”. Click on “Search Directories” and “Compiler”, and enter the directories as in the picture above.



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



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



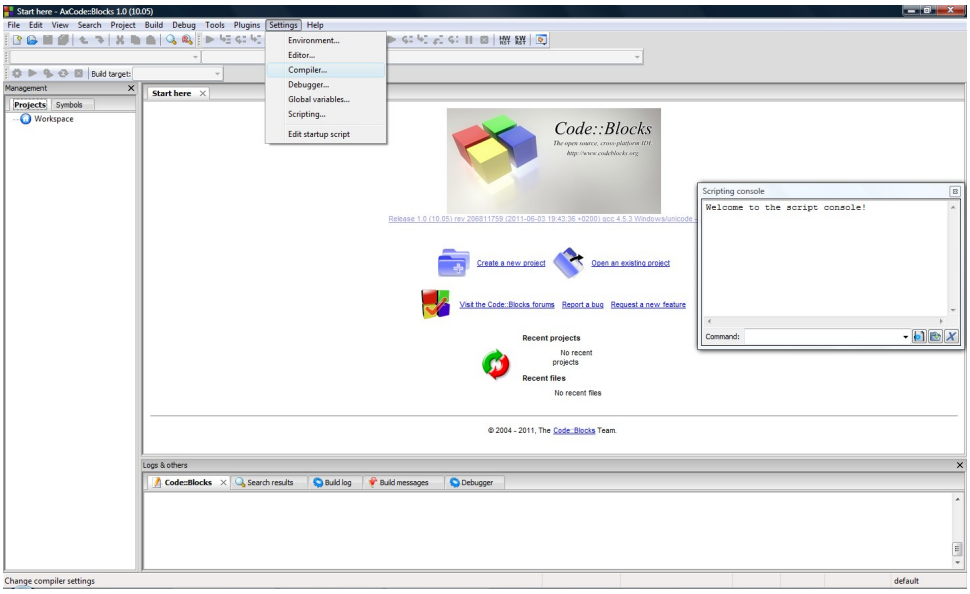
Click on “Additional Paths”, and enter the path as shown above.

11.2. Remove all saved User Settings

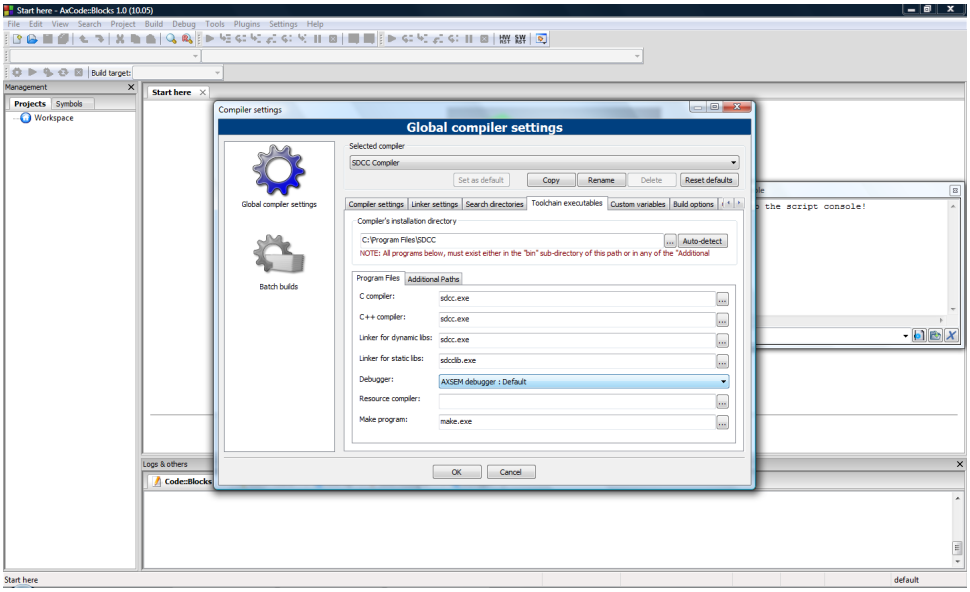
Open an explorer. Type “%APPDATA%” (without double quotes) into the address bar. Type enter. The directory now displayed should contain a folder (subdirectory) named “axCodeBlocks”. This is where AxCode::Blocks stores per-user configuration settings. Delete that subdirectory.

11.3. SDCC Project does not compile

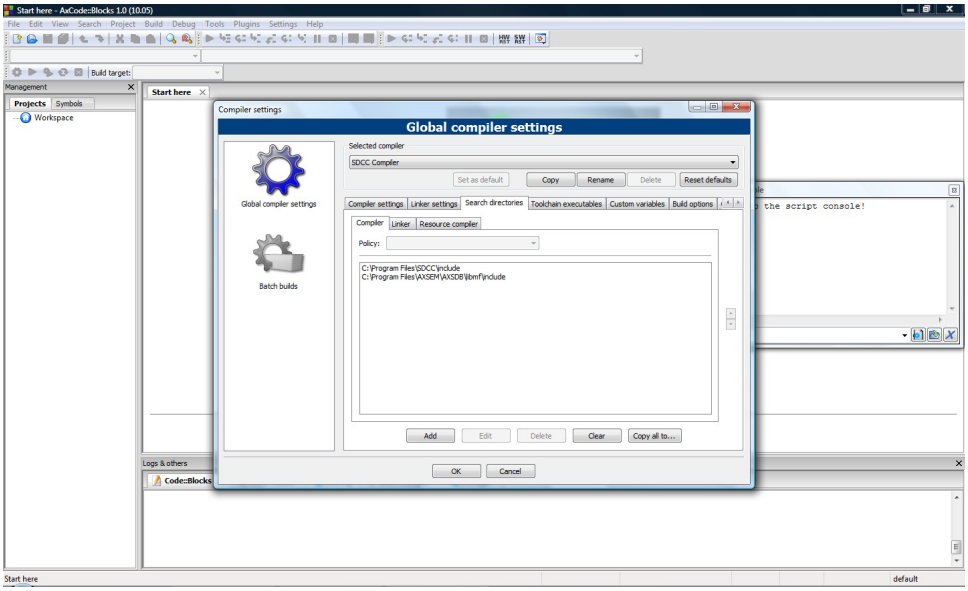
First, make sure that AxCode::Blocks finds the compiler executable. Open the compiler settings window.



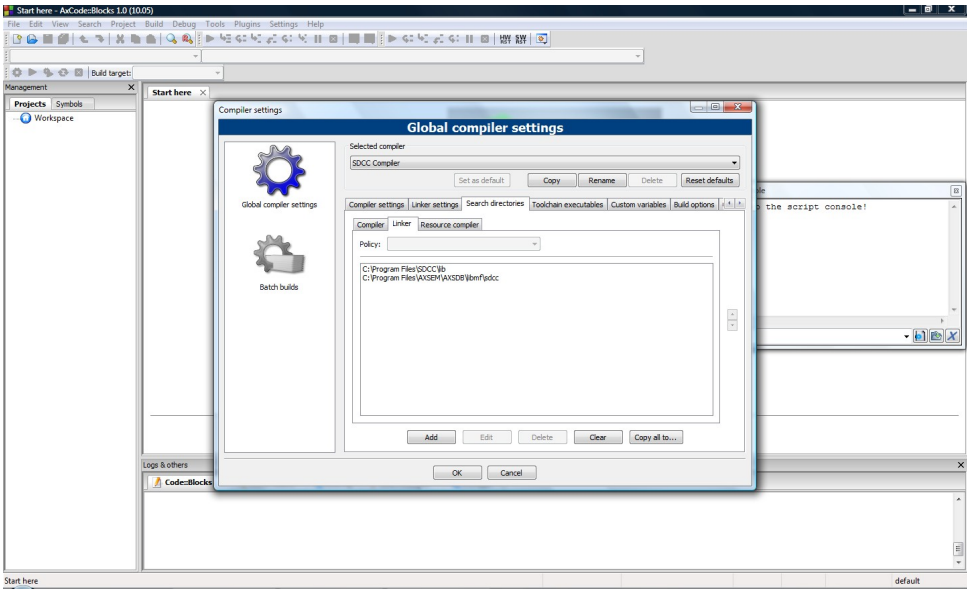
Check that SDCC is the default compiler and that the Compiler's Installation directory points to SDCC's installation location. Normally, this is automatically set up correctly, but if you manually move the SDCC directory, or re-install it at another location, you must manually update the location.



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.



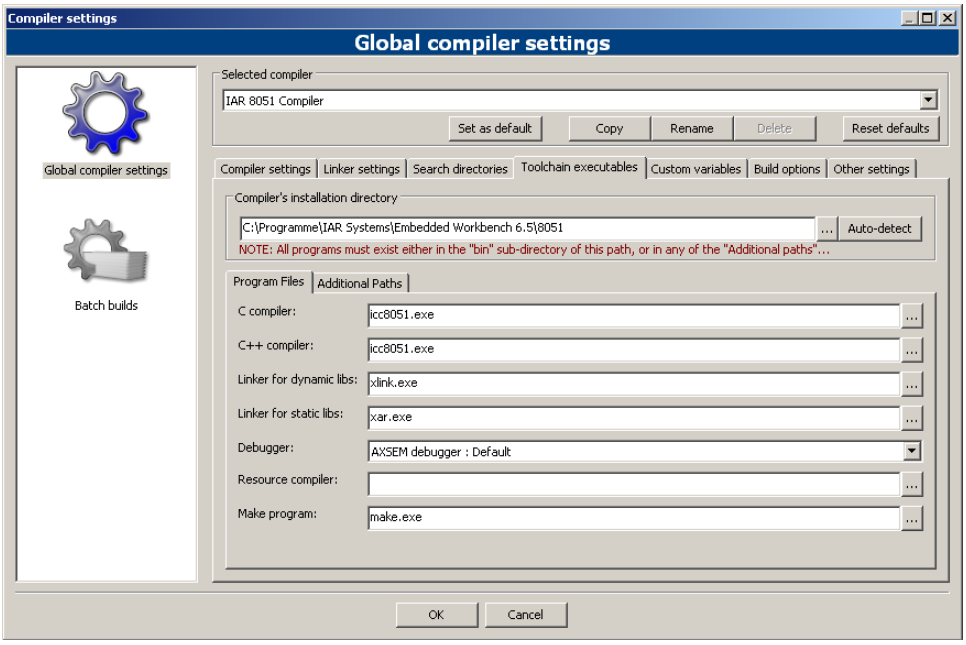
Check that the linker's default paths contain SDCC's own libraries path, as well as the AX8052 convenience library (such as libmf) path.

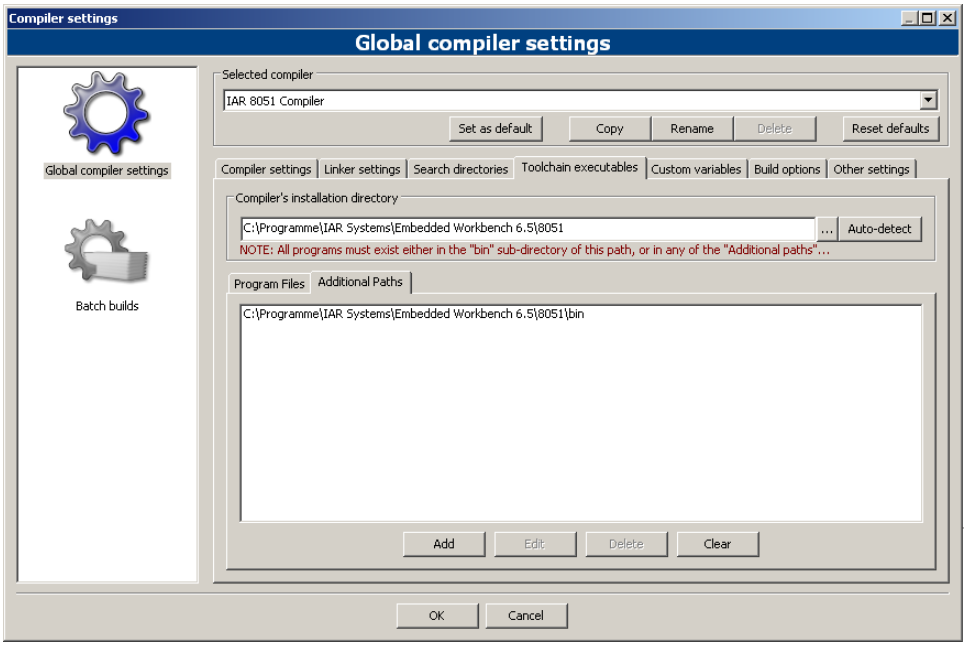


11.4. IAR Project does not compile

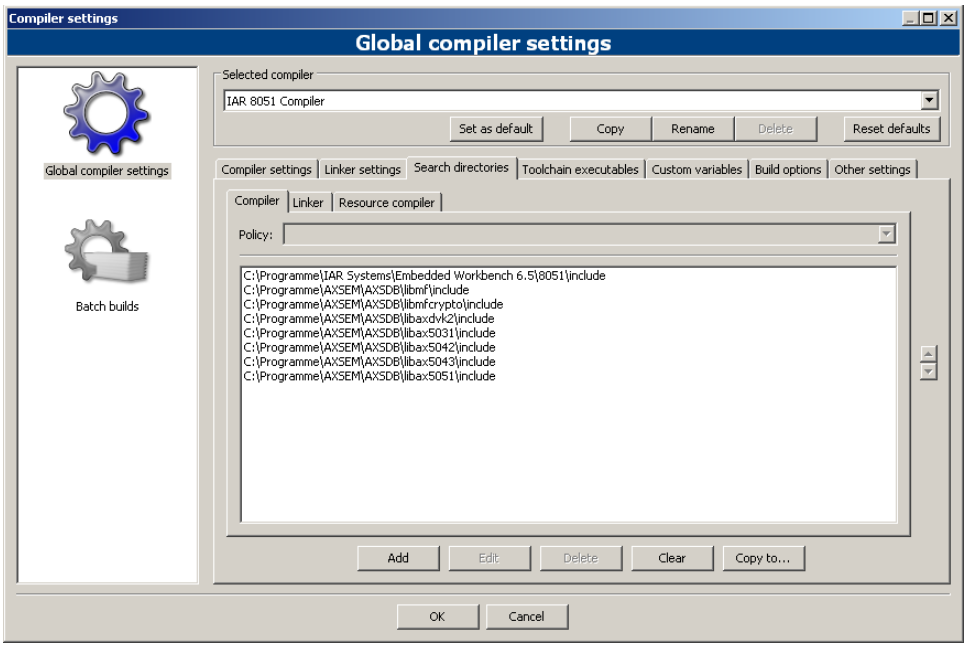
First, make sure that AxCode::Blocks finds the compiler executable. Open the compiler settings window. Then select “IAR 8051 Compiler” in the “Selected Compiler” combobox.

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.

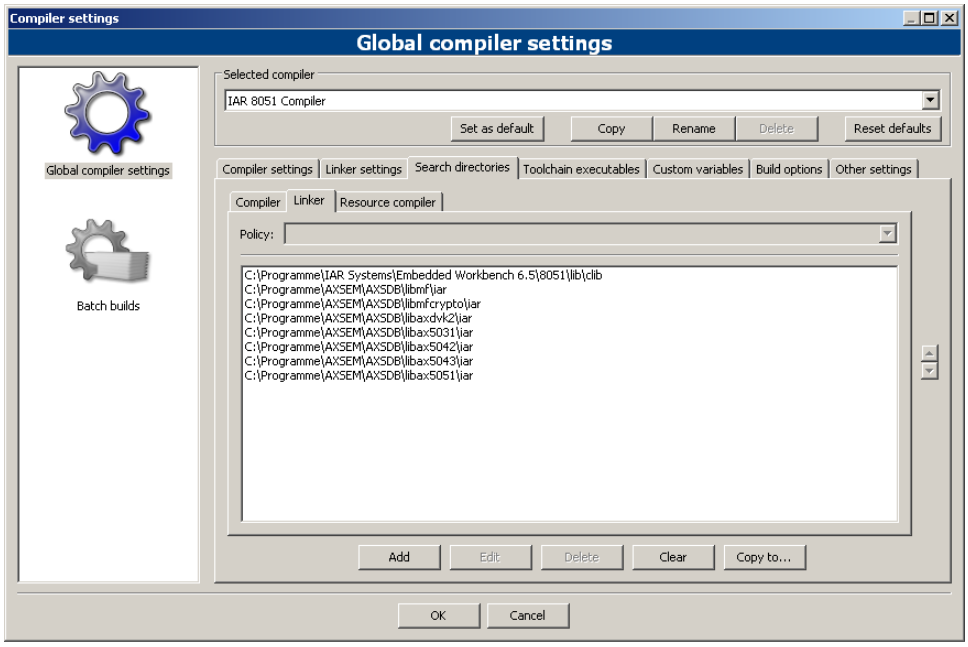




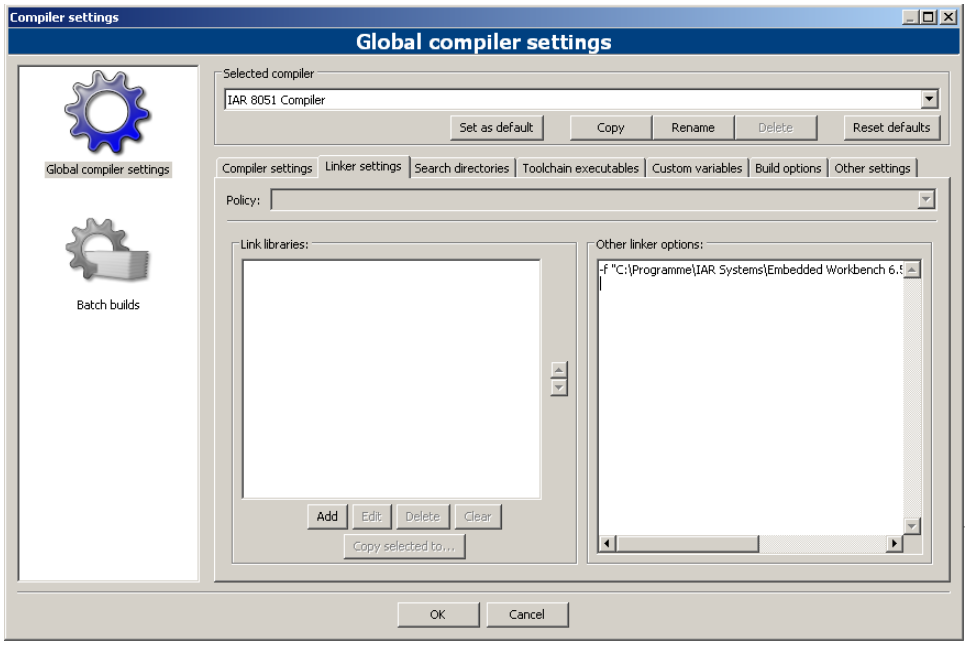
Make sure that IAR finds all required header files. Check that the compiler's default include paths contain IAR's own includes, as well as the AX8052 convenience library (such as libmf) includes.



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

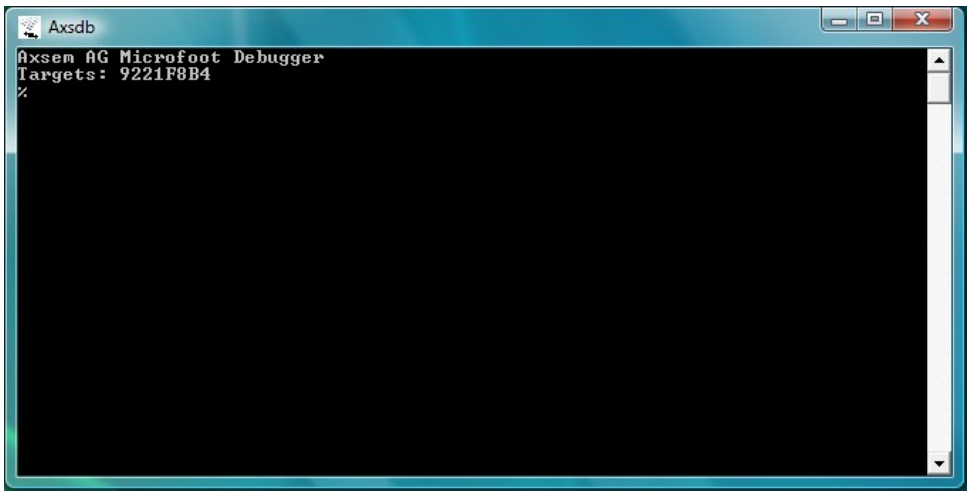


Make sure the correct linker script appears under “Other linker options”; the correct entry is –f "C:\Programme\IAR Systems\Embedded Workbench 6.5\8051\config\devices_generic\lnk51ew_plain.xcl"



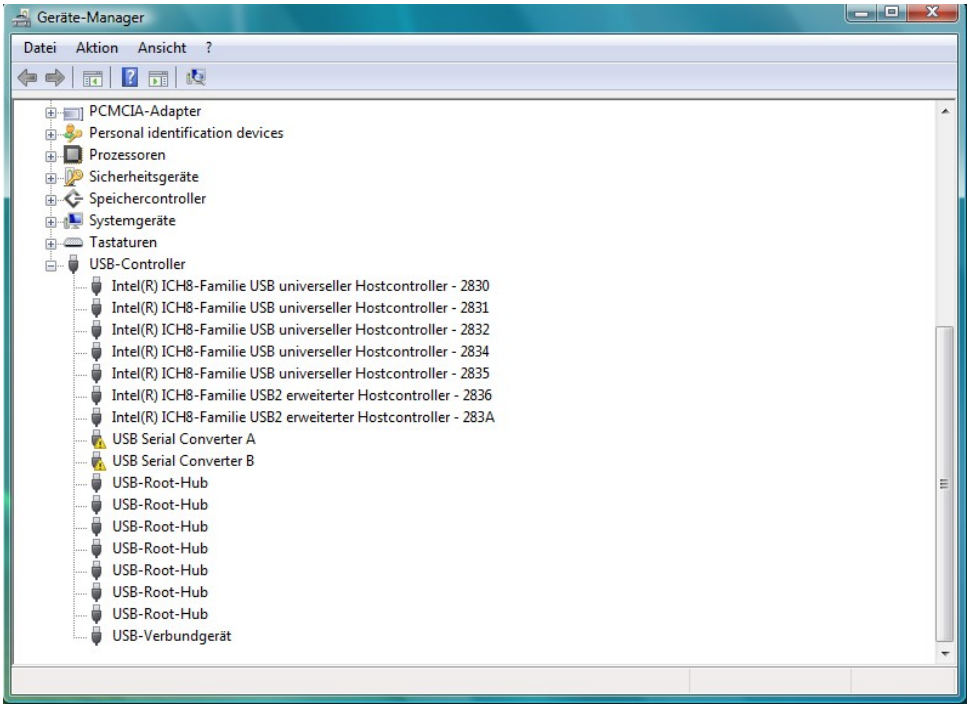
11.5. Project compiles, but debugging does not work

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

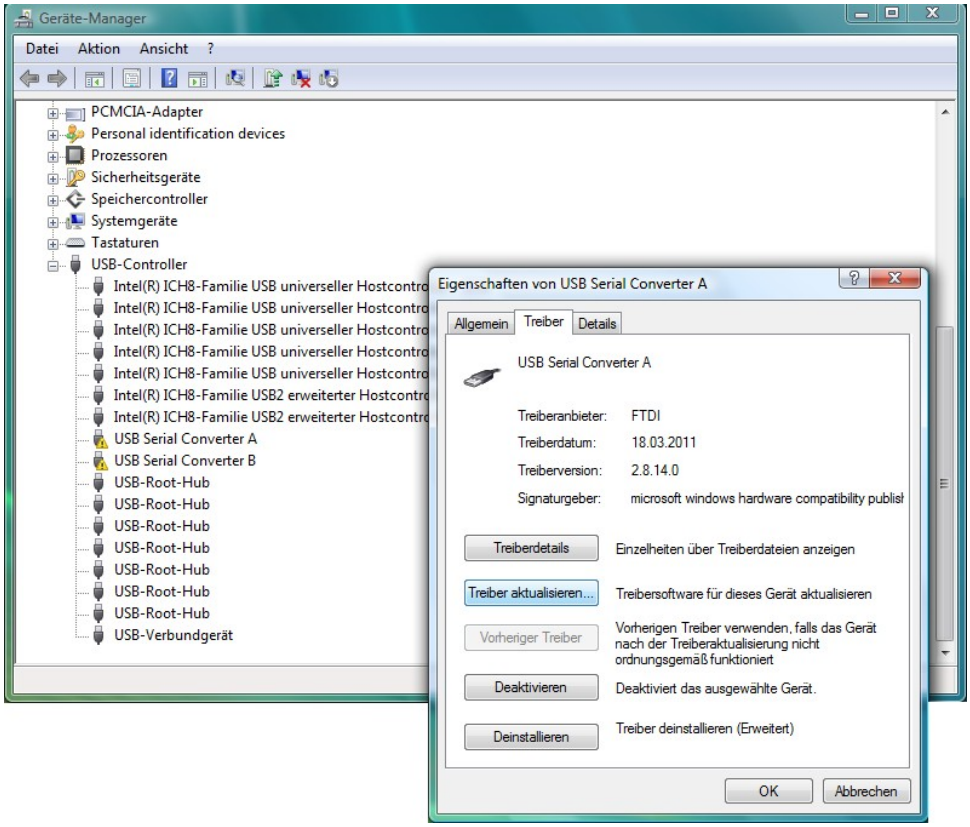


AXSDB should print the serial number of a target after the "Targets:" prompt. The top two reasons for an empty line after "Targets:" are that no debug adapter is connected to a working USB port, or that the USB drivers have not been correctly installed.

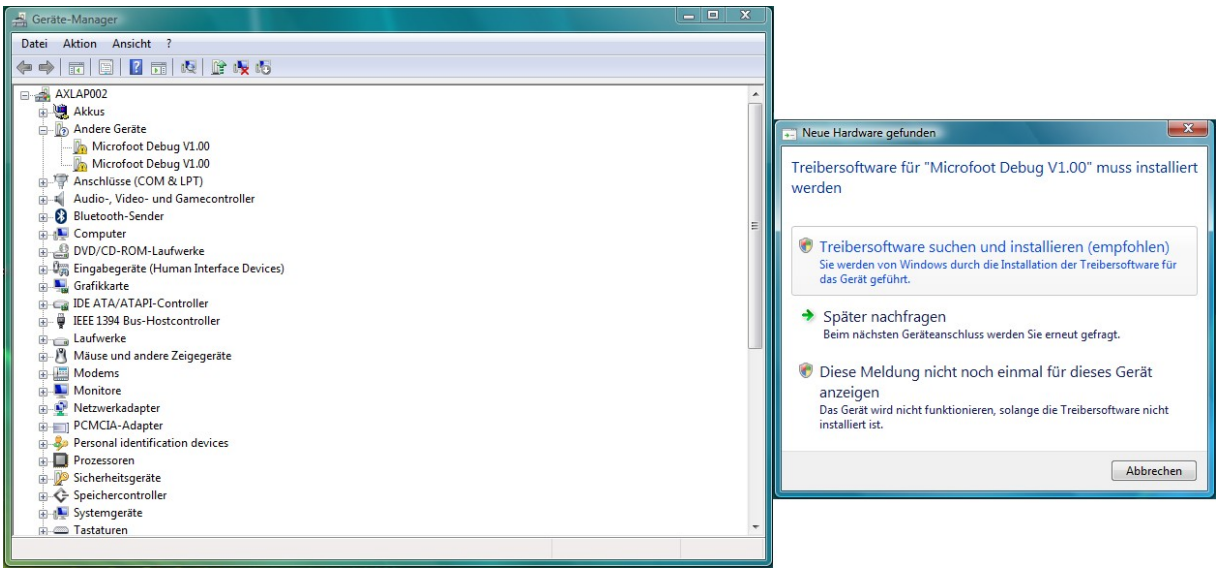
To check whether the drivers are correctly installed, open the device manager.



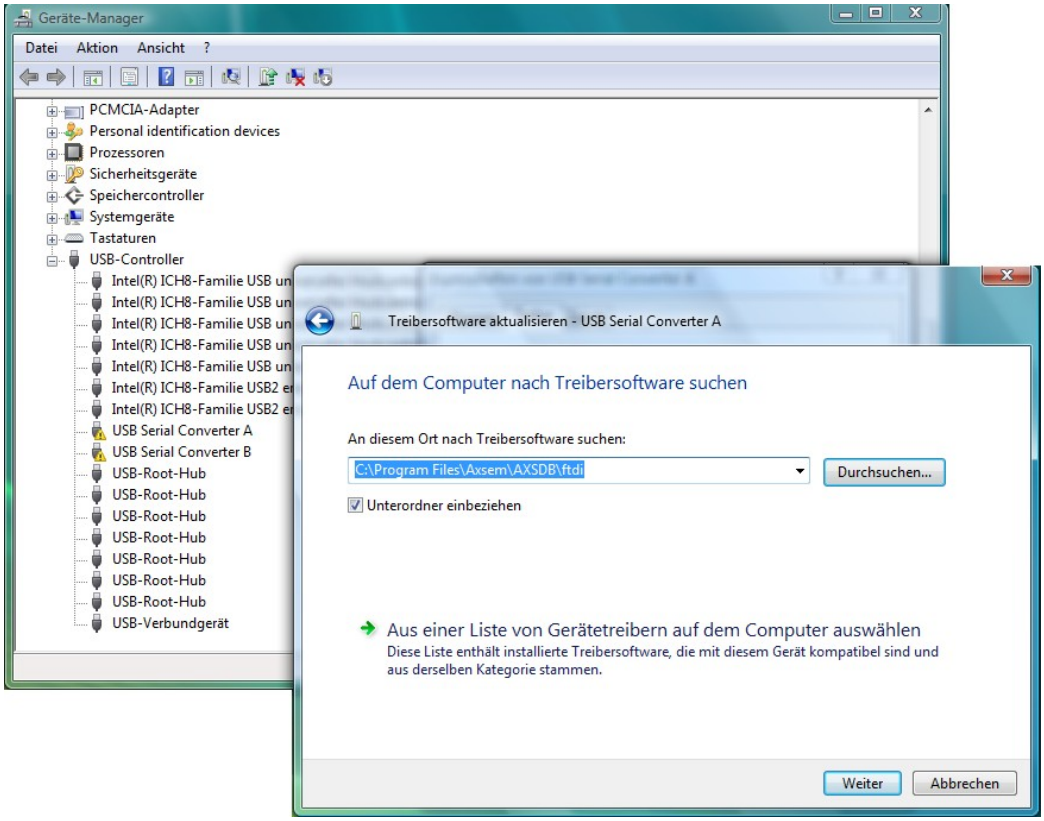
Yellow exclamation marks on “USB Serial Converter A” or “... B” or “Microfoot Debug Adapter V1.00” indicate that the driver has not been correctly installed. Right-click on the device with the exclamation mark, and choose Reinstall driver.



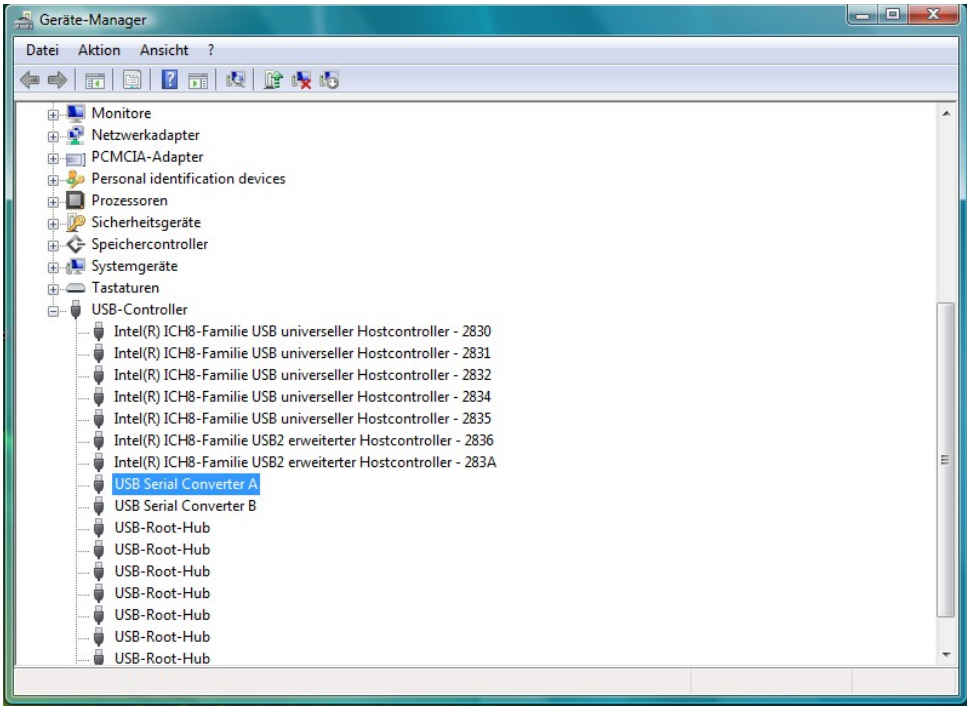
The first time a Debug Adapter is installed, Windows may prompt for the installation of a driver; choose to install a driver now.



Newer Versions of Windows (starting with XP), 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\Axsem\AXSDB\ftdi. You can choose to install from this directory and its subdirectories as well.

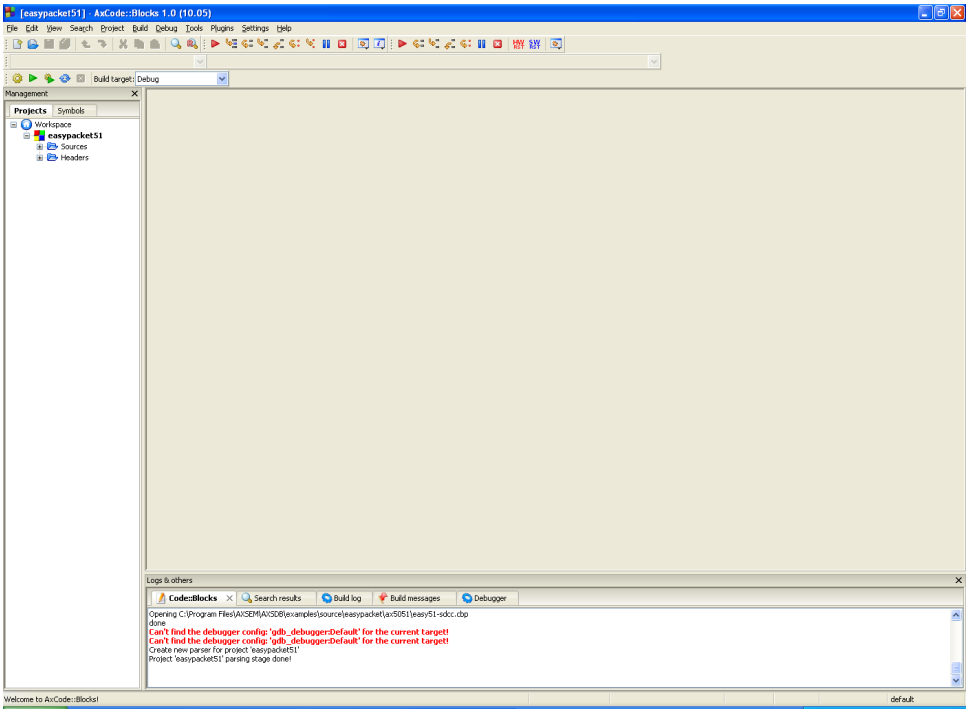


After successfully re-installing the drivers, the device manager should look as follows. After installing the drivers, you should reboot Windows.

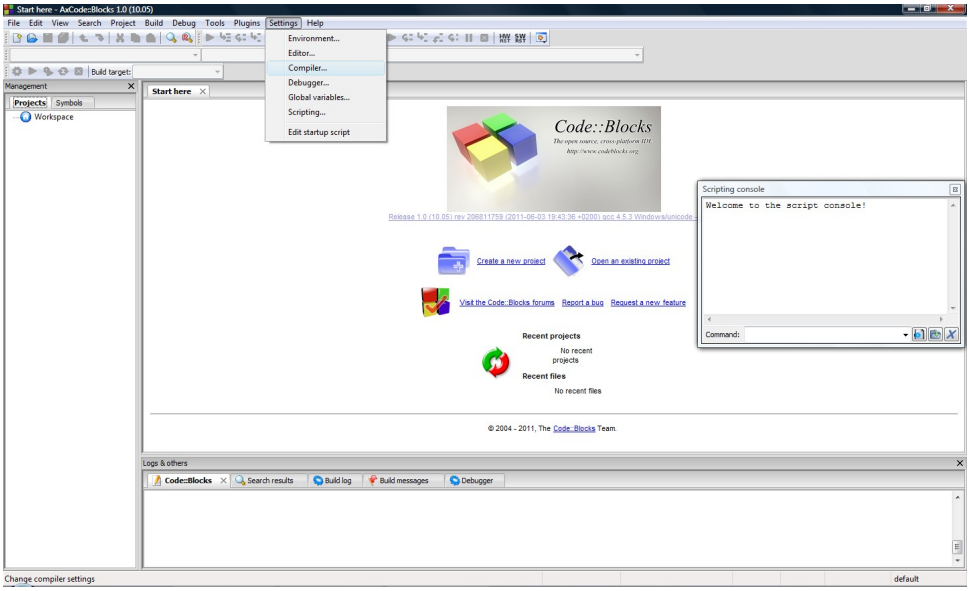


Check that AXSDB now recognizes the target.

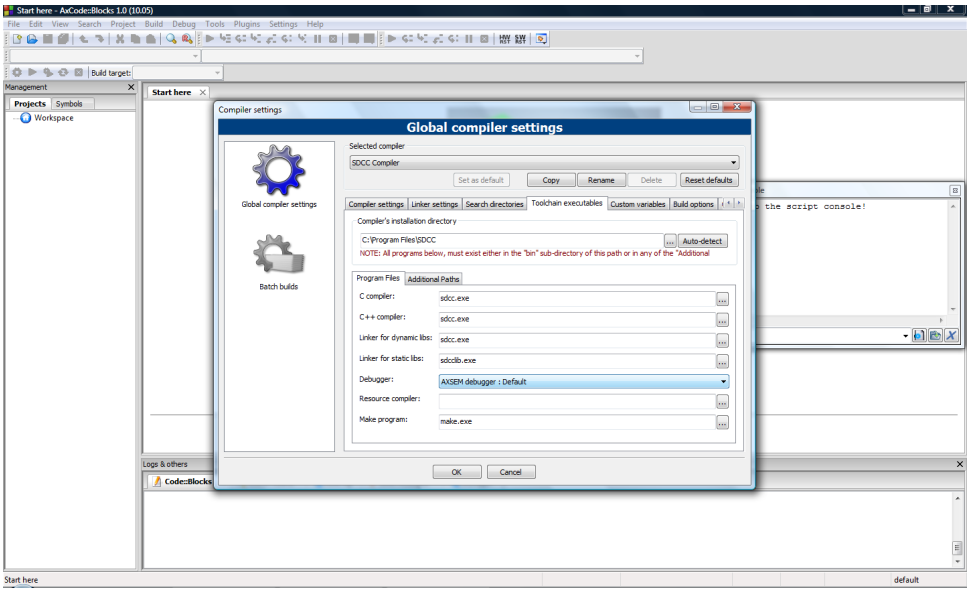
The following error means that the debugger plugin is not correctly configured. See below for how to correct this error.



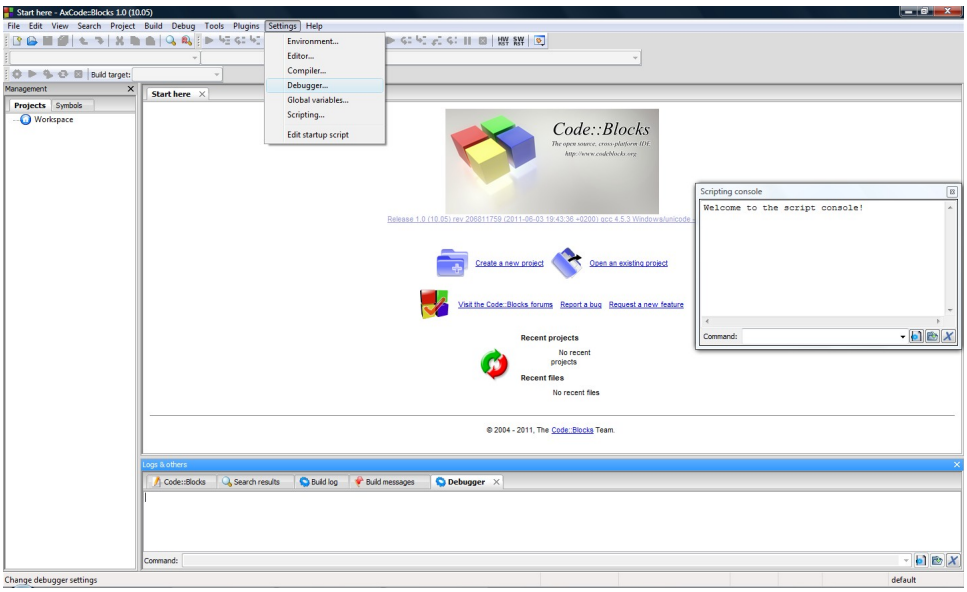
Start AxCode::Blocks. Verify that the correct debugger plugin is selected. To do this, open the compiler settings window.



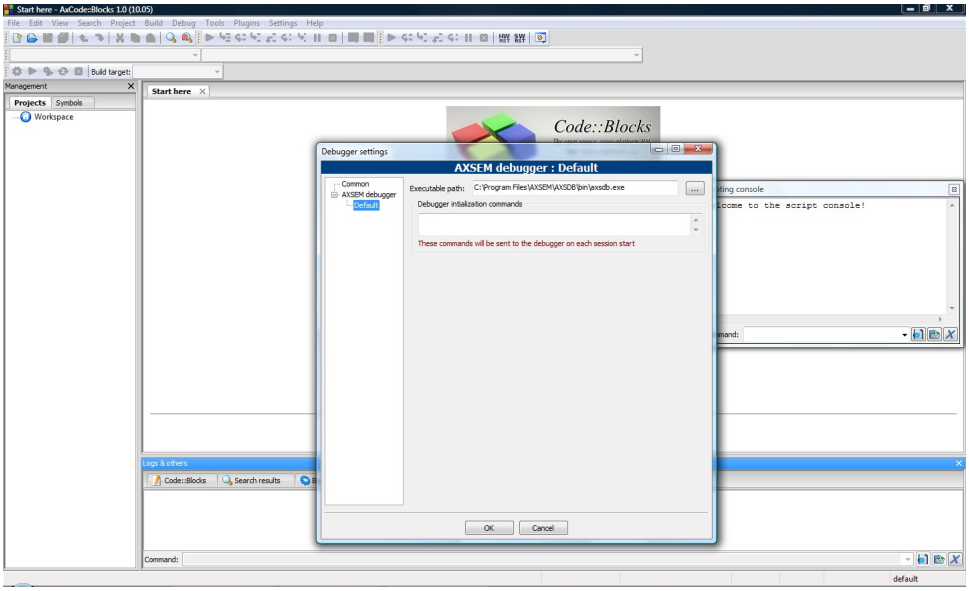
Check that the debugger plugin named “AXSEM debugger : Default” is selected.



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



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



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**