Fautronix GmbH

When complexity seems easy.

User guide

for

Getting started with FIDEx

Guide:	FX-TCN-FIDEx-UG0001
Since FIDEx revision:	2014-09.0
Revision:	0.2

Author and Copyright: Fautronix GmbH
Hegelstraße 16
72762 Reutlingen, Germany

Website: http://www.fautronix.com

Table of changes

Revision	Date	Author	Change (Page/Chapter)
0.2	2014-09-18	Christoph Fauck	Added chapter 4, Installing and uninstalling FIDEx All chapter: Changed some opticals Added chapter 7.2.4, Using multiple source files
0.1	2014-09-12	Christoph Fauck	Initial version

Table I: Table of changes

Table of audits

The following table shows an overview of all audits of the current document, both, in-house audits and audits by independent quality auditors.

Audited revision	Date	Auditor	Notes
0.2	2014-09-18	Christoph Fauck	
0.1	2014-09-13	Simone Fauck	

Table II: Table of audits

Table of content

1 Welcome to FIDEx

FIDEx is an integrated assembler development environment (IDE) for soft-core processors and is developed for Linux and Windows platforms.

It enables the development of assembler code with features that are otherwise available in high-level language toolchains only. Especially larger programs take advantage of these features and can be developed much faster.

The aim of the development was and is to develop a maximum-performance tool which is still simple and intuitive to use by beginners and experts similarly.

Note

We originally developed FIDEx for us and use it intensively for our own projects. Just the time saved through the source navigator and the simulator is quite a few weeks per year. To say nothing of the increase in quality and reduced troubleshooting due to the increased source transparency.

2 Contact and Support

If you have Questions or any suggestions or you need a special function please contact us via

- eMail fidex@fautronix.com or
- in our accordingly forums.

3 Getting Example Projects, FIDEx and further information

FIDEx can be downloaded as a functional limited edition for free. To enable the whole functionality an optional license has to be purchased from Fautronix.

The download link, errata and further license information are located on:

http://www.fautronix.com/fidex

The example project FX-TCN-FIDEx-UG0001-PRJ.zip generated during this starting guide can also be downloaded from there.

4 Installing and uninstalling FIDEx

- **Installing on Microsoft Windows**

  For installing FIDEx on Microsoft Windows please download and run the setup binary release for Microsoft Windows.

- **Installing/Uninstalling on Linux with Debian package manager**

  To install FIDEx on a Linux system which is managed by a Debian package manager like Debian or Ubuntu please download the corresponding deb files.

  After Downloading please install the downloaded files using
  “sudo dpkg -i FIDEx_xxxx.deb” command or by double clicking to the files and installing using a graphical package manager frontend of your Linux distribution.

  To uninstall FIDEx please run “sudo dpkg -e fidex” or use the graphical package manager frontend of your Linux distribution.

- **Installing/Uninstalling on Linux with Red Hat package manager**

  To install FIDEx on a Linux system which is managed by a Red Hat package manager like Red Hat, Fedora or SuSE please download the corresponding rpm files.

  After Downloading please install the downloaded files using
  “sudo rpm -ivh FIDEx-xxx.rpm” command or by double clicking to the files and installing using a graphical package manager frontend of your Linux distribution.

  To uninstall FIDEx please run “sudo rpm -e fidex” or use the graphical package manager frontend of your Linux distribution.

5 Overview

For a rapid and optimized software development FIDEx provides the following compo-nents:

- **a Project manager**

  for combining project dependent configuration settings and source file paths relative to the project location.

- **a Code editor**

  to write efficient assembler code and inserting templates.

- **a Source navigator**

  to keep the overview by developing and handling of larger programs.

- **a Multi-Pass-Assembler**

  for giving maximum freedom structuring your sources.

- **a Simulator/Debugger**

  to simulate the written assembler code step by step before simulating in a vhdl/verilog simulator or test via trial and error in the hardware.

- **a Detailed Online Manual**

  to have a fast access to the processors properties and the assembler instruction mnemonic and documentation.

- **Message viewer**

  to get efficient help on warnings and errors - the messages contains clickable links referring to the online help

6 The online help

Before we start with a new project, we have to look to the online help. FIDEx comes with a detailed online help to support you by efficient writing your code.

The online help is splitted into three manuals:

- **Application manual**

  Here you will find all about FIDEx itself and the different FIDEx modules.

- **Coding manual**

  The coding manual contains the documentation about instructions and directives as well as how to bring your code to the hardware.

- **Processor manual**

  This manual contains processor specific informations like getting started instructions, the processor resources and the mnemonic translations.

If we speak from a manual in the further document we mean the corresponding manual in the online help.

7 Starting with a new project

To explain how FIDEx works for you we will setup an example project for the Xilinx PicoBlaze™ 3 [1]) processor.

7.1 Generating a new project

After starting FIDEx the main window shows you the currently empty source navigator on the left side, the project manager and its online help page in the central of the window and further an online help navigator on the right side.

To start with a new project we select the “New” tab of the Project Manager and specify a project location using the folder button on the tab.

Since the project title will be included in the main window title and also used in the project collection we have to specify a project title on the “New” tab also.

Now we can alternatively

- generate and open the project or
- generate and add the project to the project collection (we do)

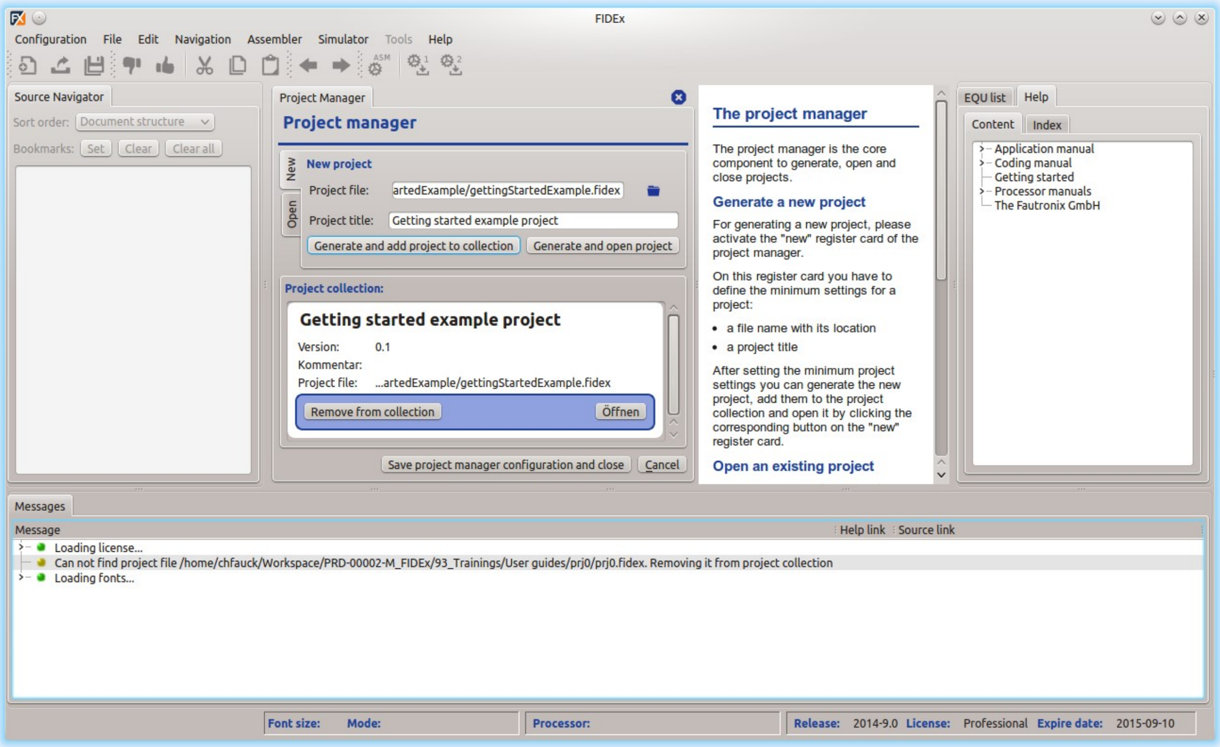
by using the appropriate buttons.

Note

To save changed content of the project collection you have to close the project manager by using the save button or by opening a project.

Figure 7.1-1 shows you an example project added to the project collection.

1 Xilinx PicoBlaze™ is a trademark of Xilinx Inc.



Project manager

New project

Project file: artedExample/gettingStartedExample.fidex

Project title: Getting started example project

Project collection:

Getting started example project

Version: 0.1
Kommentar:
Project file: ...artedExample/gettingStartedExample.fidex

The project manager

The project manager is the core component to generate, open and close projects.

Generate a new project

For generating a new project, please activate the "new" register card of the project manager.

On this register card you have to define the minimum settings for a project:

- a file name with its location
- a project title

After setting the minimum project settings you can generate the new project, add them to the project collection and open it by clicking the corresponding button on the "new" register card.

Open an existing project

```
Loading license...
Can not find project file /home/chfauck/Workspace/PRD-00002-M_FIDEx/93_Trainings/User guides/prj0/prj0.fidex. Removing it from project collection
Loading fonts...
```

Release: 2014-9.0 License: Professional Expire date: 2015-09-10

Figure 7.1-1: Project Manager with a project in the project collection

To open the new generated project now we can double click to the project collection entry or press the open button of the collection entry.

After opening the new project the project manager closes and the main window shows an empty source navigator and an empty central area.

In the messages viewer we will see that no assembler backend is registered. This means that no processor was defined for the currently loaded project.

To define a processor we call the “Configuration → Configure...” item from menu bar and go to the Processor tab in the opening configuration dialog. There we select the processor we want to develop for. In this example we select “Xilinx PicoBlaze 3”.

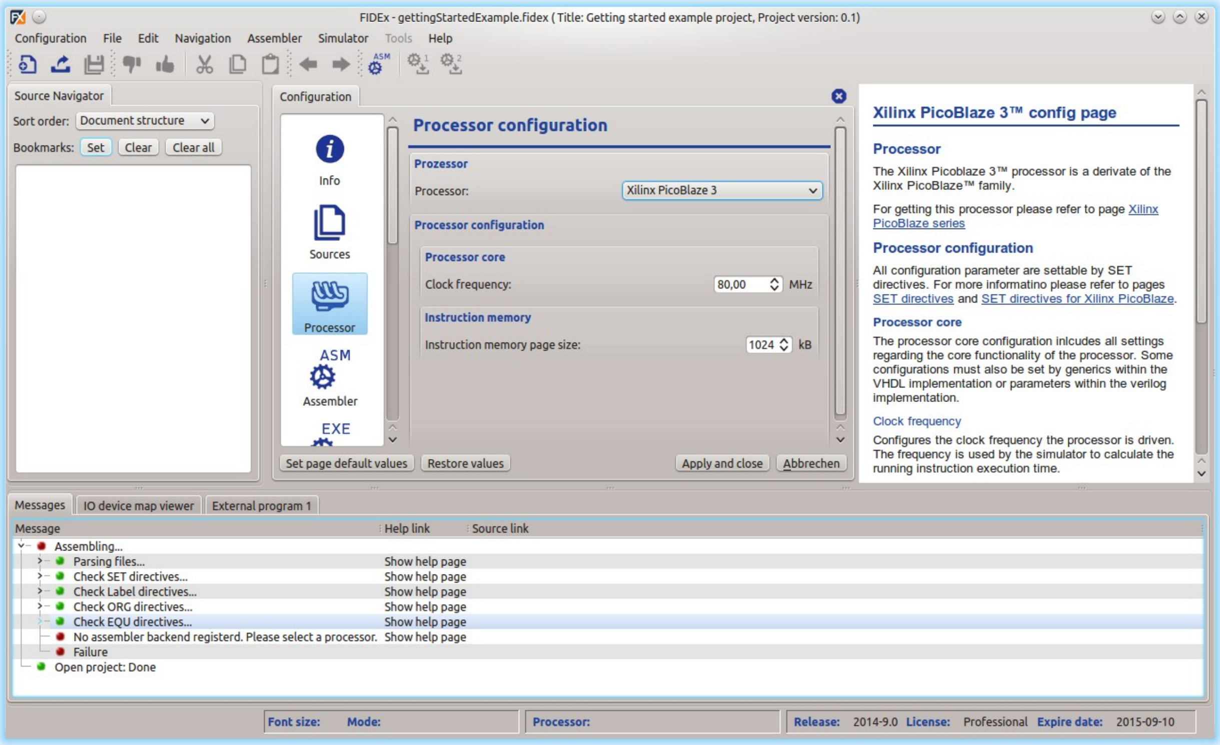
Aside to the configuration dialog page the corresponding online help page is shown (Figure 7.1-2).

After making the configuration settings we press “Apply and close” to close the configuration dialog.

Now a processor was defined and FIDEx automatically assembles after closing the configuration dialog. The selected processor is shown in the status bar.

Note

Configuration dialog settings are becoming valid after pressing the “Apply and close” button only.


FIDEx - gettingStartedExample.fidex (Title: Getting started example project, Project version: 0.1)
Configuration File Edit Navigation Assembler Simulator Tools Help
Source Navigator
Sort order: Document structure
Bookmarks: Set Clear Clear all
Configuration
Info
Sources
Processor
ASM
Assembler
EXE
Set page default values
Restore values
Apply and close
Abbrechen
Messages
IO device map viewer
External program 1

Processor configuration

Prozessor
Processor: Xilinx PicoBlaze 3

Processor configuration

Processor core
Clock frequency: 80,00 MHz

Instruction memory
Instruction memory page size: 1024 kB

Xilinx PicoBlaze 3™ config page

Processor

The Xilinx Picoblaze 3™ processor is a derivate of the Xilinx PicoBlaze™ family.

For getting this processor please refer to page Xilinx PicoBlaze series

Processor configuration

All configuration parameter are settable by SET directives. For more informatino please refer to pages SET directives and SET directives for Xilinx PicoBlaze.

Processor core

The processor core configuration inlcudes all settings regarding the core functionality of the processor. Some configurations must also be set by generics within the VHDL implementation or parameters within the verilog implementation.

Clock frequency

Configures the clock frequency the processor is driven. The frequency is used by the simulator to calculate the running instruction execution time.

Message	Help link	Source link
● Assembling...		
● Parsing files...	Show help page	
● Check SET directives...	Show help page	
● Check Label directives...	Show help page	
● Check ORG directives...	Show help page	
● Check EQU directives...	Show help page	
● No assembler backend registerd. Please select a processor.	Show help page	
● Failure		
● Open project: Done		

Font size: Mode: Processor: Release: 2014-9.0 License: Professional Expire date: 2015-09-10

Figure 7.1-2: Mai window with the opened configuration dialog

7.2 Adding sources to the project

After working through chapter 7.1, Generating a new project we have generated a new project and defined a processor.

Since there are currently no instructions to assemble in the project so we will see an appropriate error message in the message viewer.

To add new sources to the project, we have the following options:

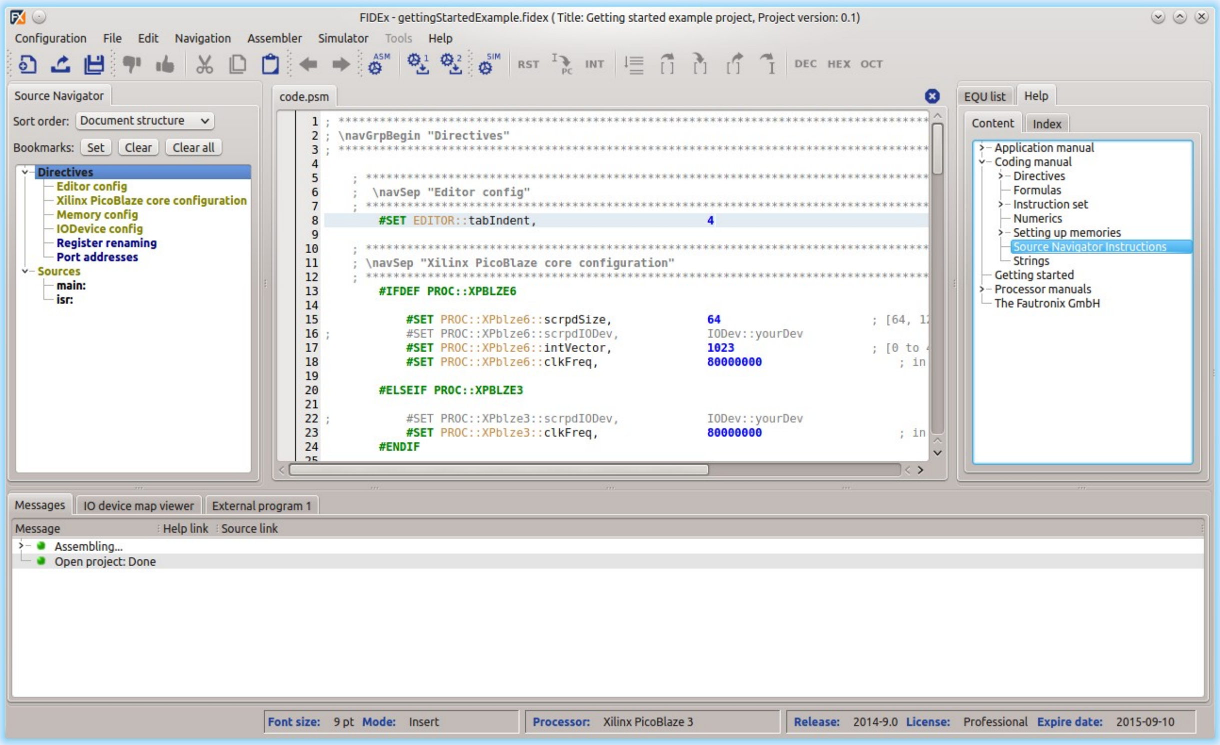
- starting from scratch using a skeleton
- importing existent sources by using the import filter
- starting from scratch without anything (for hardliner)

The following chapters describes the above mentioned options.

7.2.1 Starting from scratch using a skeleton

If you don't have already written sources or you want to see a basic example you should start using a skeleton.

FIDEx provides a template mechanism based on template files located within the installation directory. The template files are read and parsed during startup. So you can add your own template files.


FIDEx - gettingStartedExample.fidex (Title: Getting started example project, Project version: 0.1)

```
; ******************************************************************************
; \navGrpBegin "Directives"
; ******************************************************************************

    ; ***************************************************************************
    ;  \navSep "Editor config"
    ; ***************************************************************************
      #SET EDITOR::tabIndent,                    4

    ; ***************************************************************************
    ; \navSep "Xilinx PicoBlaze core configuration"
    ; ***************************************************************************
      #IFDEF PROC::XPBLZE6

          #SET PROC::XPblze6::scrpdSize,          64                     ; [64, 1
;         #SET PROC::XPblze6::scrpdIODev,         IODev::yourDev
          #SET PROC::XPblze6::intVector,          1023                   ; [0 to 
          #SET PROC::XPblze6::clkFreq,            80000000                 ; in

      #ELSEIF PROC::XPBLZE3

;         #SET PROC::XPblze3::scrpdIODev,         IODev::yourDev
          #SET PROC::XPblze3::clkFreq,            80000000                 ; in
      #ENDIF
```

Message	Help link	Source link
Assembling...		
Open project: Done		

Font size: 9 pt Mode: Insert | Processor: Xilinx PicoBlaze 3 | Release: 2014-9.0 License: Professional Expire date: 2015-09-10

Figure 7.2.1-1: Mai window with an inserted skeleton

To get started we first need to add an empty source file to the project. Therefore we call the “File → New...” item from menu bar to generate a new file.

After them we call one of the “File → Save...” items from menu bar or press the Ctrl + s short cut to save the file. We select the storage location within the project folder and press save. Before saving we will be asked to add the new file to the project and we confirm.

Note

Only files which are added to the project are assembled by FIDEx in the order in which the files are listed in the configuration dialog.

Now we have added our first empty source file to the project. The next step is to insert the skeleton for the previous defined processor.

To insert the skeleton we call the “Edit → Insert... → Templates → Skeletons → Xilinx PicoBlaze” item from menu bar. Alternatively we can reach the “Insert...” menu item from context menu shown after clicking with the right mouse button to the editor window.

After inserting the skeleton we save the file again. FIDEx assembles automatically after storing and the message viewer tells that there are no further errors.

Now we have written our first assembler program ;-) If you don't understand all code you have inserted please continue reading this guide or have a look to the Coding manual.

The source navigator on the left side shows you:

- **Labels**

  All labels positioned on the beginning of a line are shown. Indented labels are not shown in the source navigator.

- **Additional entries**

  Additional entries are defined by special navigator commands located in comments.

  These commands are used to define an optical structure of your source code within the navigator. So it is possible to generate simple entries or to generate groups within the navigator.

  For more information please refer to the chapter “Source Navigator Instructions” within the Coding manual.

By double clicking to a source navigator entry the text editor jumps to the corresponding position within your source code.

7.2.2 The skeleton

The skeleton is splitted into two sections:

- a directive section
- an instruction section

Both can be mixed because FIDEx contains a multi-pass-assembler. So the sources are parsed and then first all directives are assembled and then the processor instructions.

The directive section shows you a small subset of #set and #equ directives that are available. Most directives are self-explanatory. For further information please have a look to the Coding manual.

7.2.3 Where to find the instruction mnemonic

With FIDEx it is possible to write assembler for different processor platforms using the same assembler mnemonic.

Therefore FIDEx uses an own Assembler dialect for all processors. The chapter “Instruction set” of the Coding manual lists and describes all supported instructions and compares them over the supported processors.

If you have already experiences with the manufacturers mnemonic of a processor please have a look to the Processor manual. Each supported processor has a Mnemonic chapter there where you can find the translation from the manufacturers mnemonic to the FIDEx mnemonic.

To become familiar with FIDEx it is absolutely necessary to look over the Coding manual of the online help.

7.2.4 Using multiple source files

FIDEx supports the usage of multiple source files added to a project. The source files are assembled in the order listed within the “Sources” configuration dialog page. There you can change the order of the source files. During assembling you can see the assembling order within the message viewer.

Note

FIDEx assembles all source files added to a project in the order defined within the “Sources” configuration dialog page.

Why does FIDEx doesn't support inlcudes?

Since FIDEx manages all source files as part of a project there is no need to deal with in-cludes.

For including "inline code" like in C FIDEx will support macros in a later release.

7.2.5 Importing existent sources by using the import filter

FIDEx provides an import filter to import existing sources from 3rd party manufacturers.

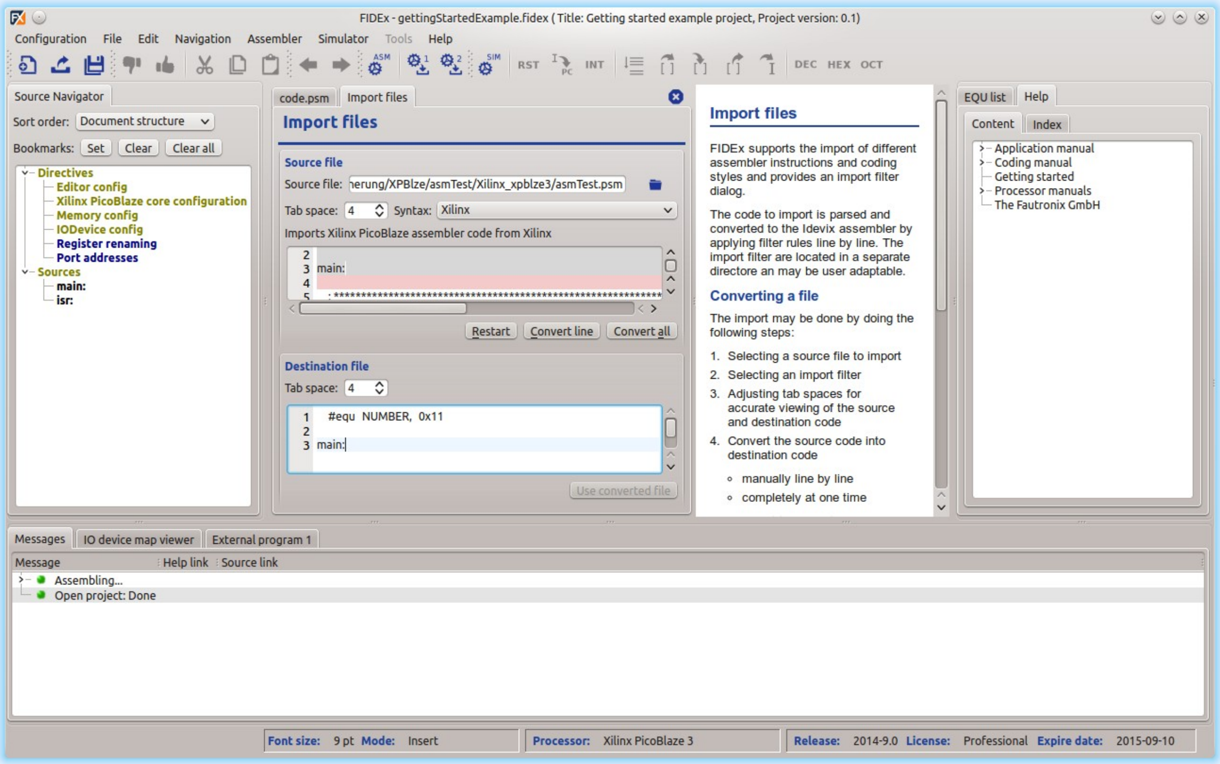
To import an existing source please call the "File → Import files..." item from menu bar to open the "Import files" dialog and its corresponding online help page.

To import your source files please read the shown online help page for further instructions.

Note

The import filter conversion stops with a message in the message window if an error occurs.

Before you use the conversion results please convince yourself that the conversion process is complete.



FIDEx - gettingStartedExample.fidex (Title: Getting started example project, Project version: 0.1)

Sort order: Document structure

Import files

Source file

Source file: ıerung/XPBlze/asmTest/Xilinx_xpblze3/asmTest.psm

Tab space: 4 Syntax: Xilinx

Imports Xilinx PicoBlaze assembler code from Xilinx

```
main:

; ****************************************************************
```

Destination file

Tab space: 4

```
  #equ  NUMBER,  0x11

main:
```

Import files

FIDEx supports the import of different assembler instructions and coding styles and provides an import filter dialog.

The code to import is parsed and converted to the Idevix assembler by applying filter rules line by line. The import filter are located in a separate directore an may be user adaptable.

Converting a file

The import may be done by doing the following steps:

1. Selecting a source file to import
2. Selecting an import filter
3. Adjusting tab spaces for accurate viewing of the source and destination code
4. Convert the source code into destination code
   - manually line by line
   - completely at one time

```
Assembling...
Open project: Done
```

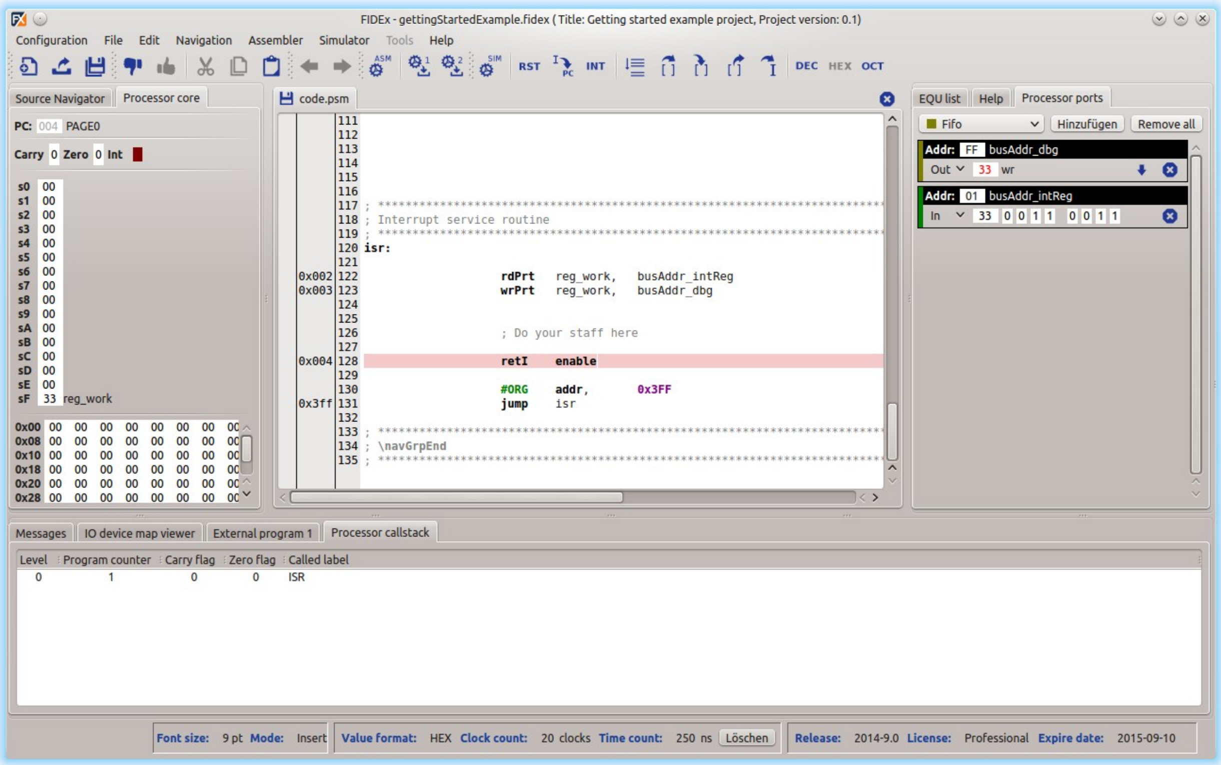
Font size: 9 pt Mode: Insert | Processor: Xilinx PicoBlaze 3 | Release: 2014-9.0 License: Professional Expire date: 2015-09-10

Figure 7.2.5-1: Main window with import filter dialog

8 Simulating/debugging a project

For debugging purposes FIDEx contains a powerful simulator for each supported processor.

To start the simulator for the defined processor please call the “Simulator → Enable/disable simulator” item from menu bar.



FIDEx - gettingStartedExample.fidex (Title: Getting started example project, Project version: 0.1)

```
      ; ****************************************************************
      ; Interrupt service routine
      ; ****************************************************************
      isr:

0x002                rdPrt   reg_work,   busAddr_intReg
0x003                wrPrt   reg_work,   busAddr_dbg


                     ; Do your staff here

0x004                retI    enable

                     #ORG    addr,       0x3FF
0x3ff                jump    isr

      ; ****************************************************************
      ; \navGrpEnd
      ; ****************************************************************
```

Level	Program counter	Carry flag	Zero flag	Called label
0	1	0	0	ISR

Font size: 9 pt **Mode:** Insert **Value format:** HEX **Clock count:** 20 clocks **Time count:** 250 ns Löschen **Release:** 2014-9.0 **License:** Professional **Expire date:** 2015-09-10

Figure 8-1: Main window with simulator

The simulator starts and shows interactive processor elements as well as the ports around the central area:

- **the processor core on the left side**

  The processor core contains the program counter (PC) of the processor, the carry and zero flags as well as the clickable interrupt flag. It contains the register banks and the internal scratch pad memory.

- **the processor call stack**

  The processor call stack shows you the return parameter for the next return instruction.

- **the processor ports**

  The most processors have a port to communicate with surrounding FPGA logic like registers, FIFOs or memories.

  In this area you can add such devices, assign addresses and show respectively manipulate port values.

8.1 Controlling the simulation

The simulation can be controlled using the menu bar or the tool bar. The following table shows the tool bar icons with its functionality.

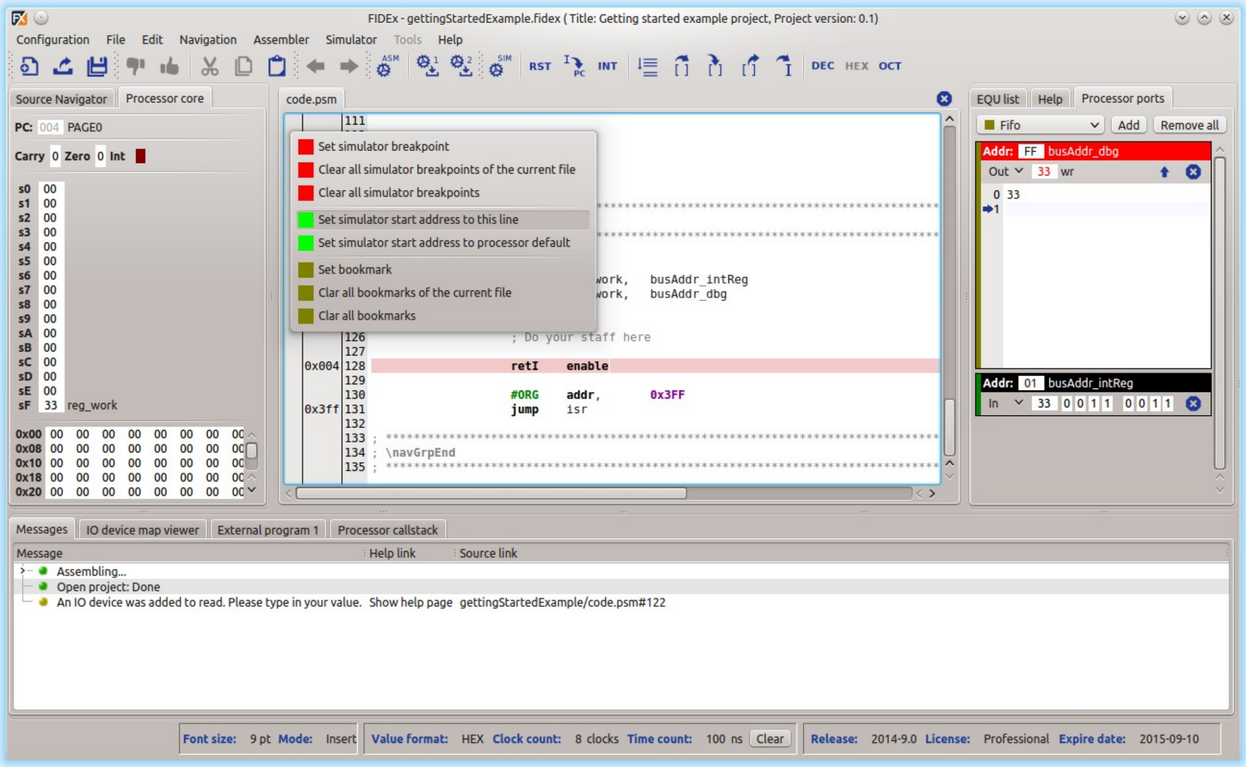
Tool bar icon	Function description
SIM ⚙	Enables and disables the simulator. Each button press toggles the simulator enable state.
RST	Resets the simulator. All simulator components are cleared and the program counter will be set to the defined start address. See chapter 8.2, The simulators start address for more information.
I ↷ PC	Sets the program counter to the address defined by the current cursor position of the text editor.
INT	Assigns an interrupt to the simulator.
↓≡	Run simulator in endless mode.
↷ [ ]	Simulates the next instruction. If the instruction is a call, the complete call will be simulated until the call stack reaches the originally level at the beginning of the simulation step.
↓ [ ]	Simulates the next instruction.
↱ [ ]	Simulates until the next return instruction.
↷ I	Simulates until the address defined by the current cursor position in the text editor.
DEC HEX OCT	Sets all simulator values to decimal, hexadecimal or octal.

Table 8.1.I: Controlling the simulation using the tool bar

8.2 The simulators start address

The simulators start address can be defined by three approaches:

- No explicit definition. The simulators start address is the processors start address.
- Set the simulators start address actively from current cursor position in the text editor described in chapter 8.1, Controlling the simulation.
- Define a new start address using the right mouse button context menu of the text editors left margin (Figure 8.2-1).
- If you change your code with enabled simulator and save your changes, the code will be assembled again. The simulator automatically gets the new assembled code and try to set the program pointer to the last position to continue the simulation. So you can make changes on-the-fly if you found a bug during simulation.


FIDEx - gettingStartedExample.fidex (Title: Getting started example project, Project version: 0.1)
Configuration
File
Edit
Navigation
Assembler
Simulator
Tools
Help
RST
INT
DEC
HEX
OCT
Source Navigator
Processor core
code.psm
EQU list
Help
Processor ports
Set simulator breakpoint
Clear all simulator breakpoints of the current file
Clear all simulator breakpoints
Set simulator start address to this line
Set simulator start address to processor default
Set bookmark
Clar all bookmarks of the current file
Clar all bookmarks
Fifo
Add
Remove all
Messages
IO device map viewer
External program 1
Processor callstack

```
PC: 004 PAGE0
Carry 0 Zero 0 Int

s0 00
s1 00
s2 00
s3 00
s4 00
s5 00
s6 00
s7 00
s8 00
s9 00
sA 00
sB 00
sC 00
sD 00
sE 00
sF 33 reg_work

0x00 00 00 00 00 00 00 00 00
0x08 00 00 00 00 00 00 00 00
0x10 00 00 00 00 00 00 00 00
0x18 00 00 00 00 00 00 00 00
0x20 00 00 00 00 00 00 00 00
```

```
                       work,    busAddr_intReg
                       work,    busAddr_dbg

                       ; Do your staff here

0x004                  retI     enable

                       #ORG     addr,      0x3FF
0x3ff                  jump     isr

      ; ************************************************************
      ; \navGrpEnd
      ; ************************************************************
```

```
Addr: FF busAddr_dbg
Out 33 wr
0 33
1

Addr: 01 busAddr_intReg
In 33 0 0 1 1 0 0 1 1
```

```
Message                                                    Help link        Source link
Assembling...
Open project: Done
An IO device was added to read. Please type in your value.  Show help page  gettingStartedExample/code.psm#122
```

Font size: 9 pt Mode: Insert | Value format: HEX Clock count: 8 clocks Time count: 100 ns Clear | Release: 2014-9.0 License: Professional Expire date: 2015-09-10

Figure 8.2-1: Context menu of the editors left margin

8.3 Breakpoints

Simulations are interrupted by the following conditions:

- **manually set breakpoints**

  Using the context menu of the editors left margin as shown in Figure 8.2-1. You can individually set breakpoints. If a breakpoint is not placed into a line containing an instruction, the breakpoint will be applied to the next following instruction.

- **implicit breakpoints**

  If the simulator tries to read from a port address that is not served by a simulator port component, the simulator stops and adds a red colored register component with the requested address to the processors port area. Within the message window you will get the hint to define a register value to read. After them you can continue the simulation manually.

8.4 Simulating with renamed registers and constant defined port addresses

One of the most popular features of FIDEx is the ability to show the names of renamed registers and constant defined port addresses by accessing a simulation component.

In Figure 8.2-1 you can see

- the register name “reg_work” by accessing the register sF in the simulators register component

- the name of the constants “busAddr_intReg” and “busAddr_dbg” used to access the ports in the simulators port components.

9 Generating output files of a project

After writing a program with successful simulation in the simulator it's time to tell how to bring the assembled program to the hardware.

To do this FIDEx supports multiple types of output files and mechanisms:

- Generation of MEM or HEX files
- Generation of VHDL or Verilog files by reading and replacing tags of a appropriate template file.

For information how to setup a template file manually please read chapter “Setting up memories” of the coding manual.

If you use a Xilinx PicoBlaze processor you will find an appropriate template file within the PicoBlaze package down loadable from Xilinx.

For this example we use our own template file. You will find it in the sources of this getting started example. It contains additional tags for VHDL/Verilog documentation using doxy-gen.

9.1 The IO device concept

Before we start to generate output files we have to talk about the FIDEx IO device con-cept. It is described in detail in the “Setting up memories” chapter of the Coding manual.

An IO device is a black box representing a part of your logic design. This can currently be

- a register,
- a FIFO or
- a Memory block.

In the future this list will be extended by script based devices or by socket devices to han-dle communications between multiple FIDEx instances to simulate multiprocessor designs.

So an IO device can be used

- to initialize a port component or the processors internal scratch pad in the simulator,
- to initialize one or more block rams via VHDL or Verilog or
- to generate a HEX or MEM file for block ram or external flash initializations.

Note

All output files that FIDEx can generate are feed from an IO device.

An IO device will be feed with its configuration and its values. These values can be an ini-tial value replaced by individual values from instruction memory and arbitrary set values, alternatively or combined. Figure 9.1-1 shows the IO device concept.

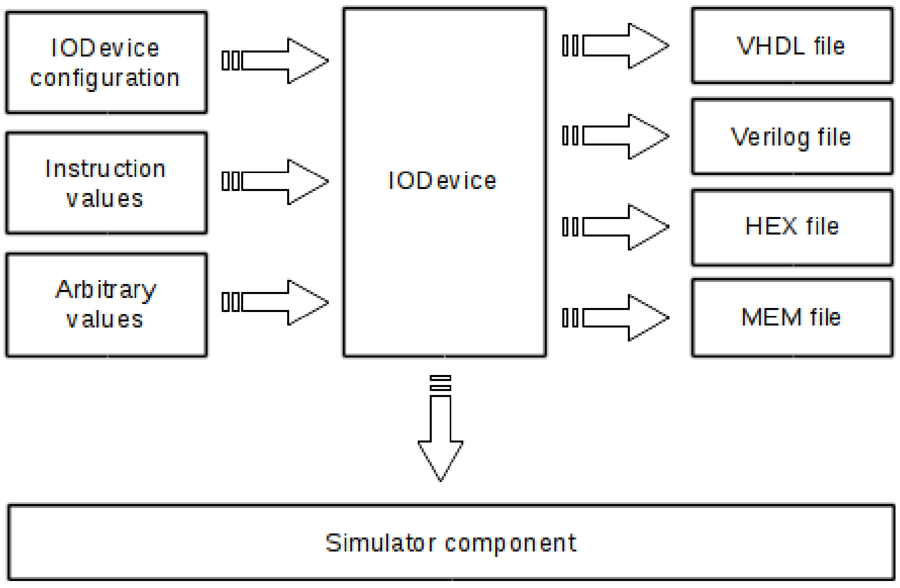


Figure 9.1-1: IO device concept

9.2 Setting up an IO device for our instruction memory

Now we have seen some basics about IO devices and we can start to generate some output files from our example project.

The IO device we need for the instruction memory can be setup

- implicitly by using the configuration dialog only or
- explicitly by using #set directives.

If you don't instantiate an IO device using #set directives FIDEx generates a default one for you. The output files connected to this default IO device are configured in the configuration dialog only.

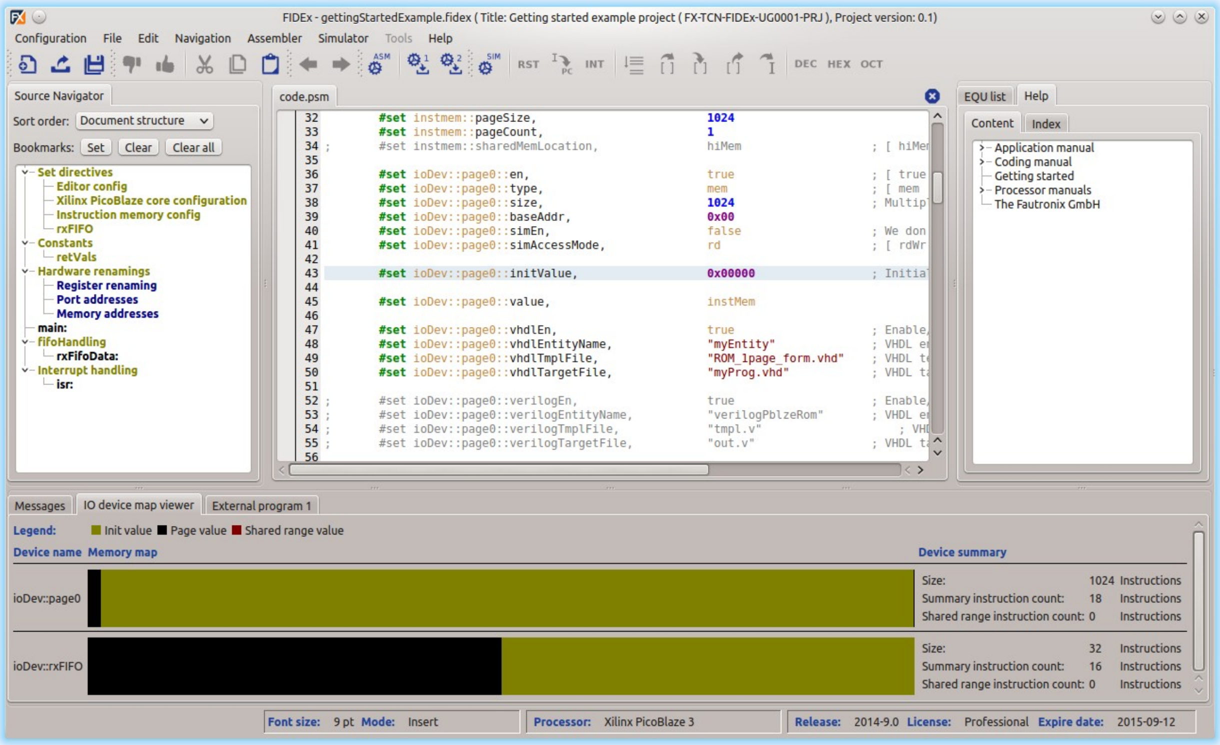
Since the skeleton already contains two IO devices, one for the instruction memory and one for simulation of a FIFO and we want to have a maximum of freedom we decide to use explicit IO devices defined by #set directives.

You can see the defined IO devices in the IO device viewer, located in the bottom area of the main window. Figure 9.2-1 shows the IO device viewer of our example project. You can see the the page0 device as well as the rxFIFO device.

By moving the mouse over the device graphs the content value of the corresponding address is shown. By double clicking to the graph, the text editor jumps to the directive which defines the double clicked device address.

Please double click to the white space representing the initial values of the page0 graph. The editor jumps to the corresponding #set directives. In Figure 9.2-1 shows the IO device definition of the instruction memory. The most directives are self-explanatory. For further information please refer to chapter “Setting up memories” of the Coding manual.

Now we comment in the code lines for the VHDL and the HEX output files. In Figure 9.2-1 you will see how to set the directives.



```
        #set instmem::pageSize,                        1024
        #set instmem::pageCount,                       1
;       #set instmem::sharedMemLocation,               hiMem               ; [ hiMe

        #set ioDev::page0::en,                         true                ; [ true
        #set ioDev::page0::type,                       mem                 ; [ mem
        #set ioDev::page0::size,                       1024                ; Multip
        #set ioDev::page0::baseAddr,                   0x00
        #set ioDev::page0::simEn,                      false               ; We don
        #set ioDev::page0::simAccessMode,              rd                  ; [ rdWr

        #set ioDev::page0::initValue,                  0x00000             ; Initia

        #set ioDev::page0::value,                      instMem

        #set ioDev::page0::vhdlEn,                     true                ; Enable
        #set ioDev::page0::vhdlEntityName,             "myEntity"          ; VHDL e
        #set ioDev::page0::vhdlTmplFile,               "ROM_1page_form.vhd" ; VHDL t
        #set ioDev::page0::vhdlTargetFile,             "myProg.vhd"        ; VHDL t

;       #set ioDev::page0::verilogEn,                  true                ; Enable
;       #set ioDev::page0::verilogEntityName,          "verilogPblzeRom"   ; VHDL e
;       #set ioDev::page0::verilogTmplFile,            "tmpl.v"              ; VHD
;       #set ioDev::page0::verilogTargetFile,          "out.v"             ; VHDL t
```

Legend: 🟩 Init value ⬛ Page value 🟥 Shared range value

Device name	Size:			Summary instruction count:			Shared range instruction count:		
ioDev::page0	Size:	1024	Instructions	Summary instruction count:	18	Instructions	Shared range instruction count:	0	Instructions
ioDev::rxFIFO	Size:	32	Instructions	Summary instruction count:	16	Instructions	Shared range instruction count:	0	Instructions

Font size: 9 pt Mode: Insert | Processor: Xilinx PicoBlaze 3 | Release: 2014-9.0 License: Professional Expire date: 2015-09-12

Figure 9.2-1: IO device viewer

9.3 Writing output file events

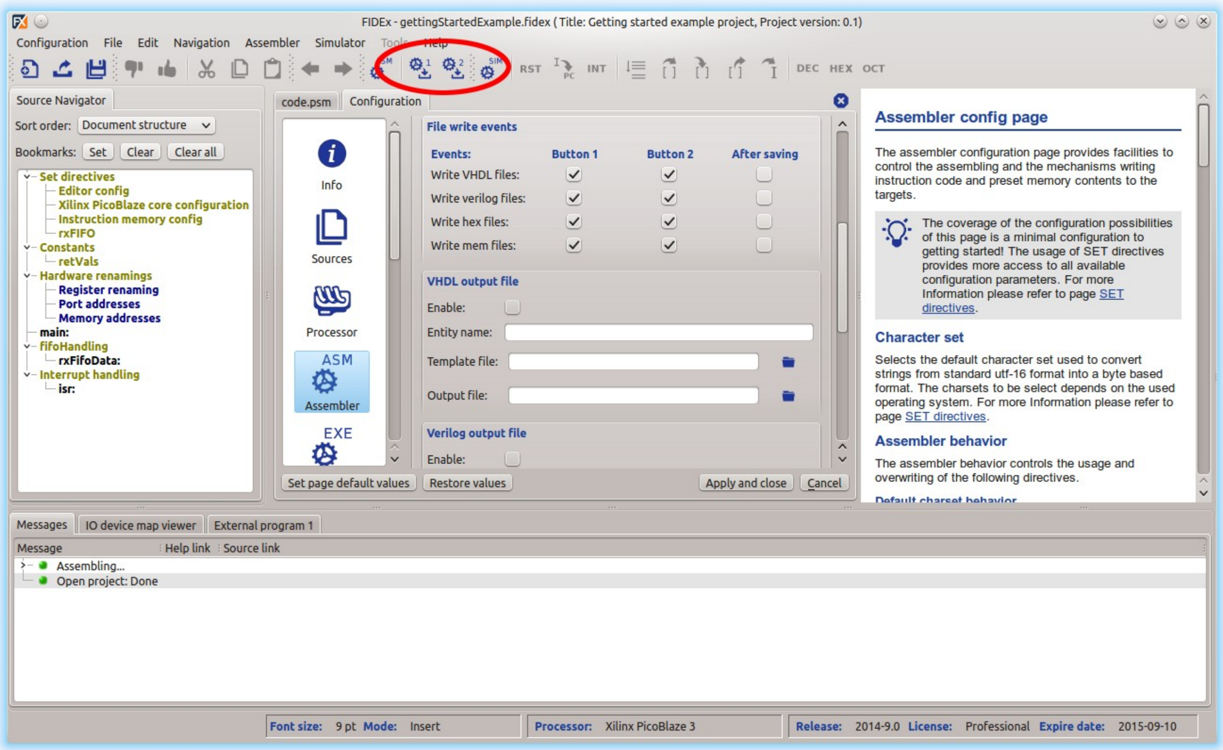
Now we have setup a project, successful simulated the project and setup IO devices for out instruction memory and a FIFO for simulation purposes.

To induce FIDEx to write the previous defined output files we have to couple the file type of each file to a file write event.

This can only be done in the configuration dialog on the Assembler configuration page. Figure 9.3-1 shows the “Output file configuration” section of the assembler configuration page.

In the upper range you can see a matrix of file write events and output file types. There you can connect the writing of an output file type to one of the two buttons (encircled red) or the saving event of a source file.

Now you are ready to use FIDEx for your own project.



File write events

Events:	Button 1	Button 2	After saving
Write VHDL files:	✓	✓	☐
Write verilog files:	✓	✓	☐
Write hex files:	✓	✓	☐
Write mem files:	✓	✓	☐

VHDL output file

Enable: ☐
Entity name:
Template file:
Output file:

Verilog output file

Enable: ☐

Assembler config page

The assembler configuration page provides facilities to control the assembling and the mechanisms writing instruction code and preset memory contents to the targets.

The coverage of the configuration possibilities of this page is a minimal configuration to getting started! The usage of SET directives provides more access to all available configuration parameters. For more Information please refer to page SET directives.

Character set

Selects the default character set used to convert strings from standard utf-16 format into a byte based format. The charsets to be select depends on the used operating system. For more Information please refer to page SET directives.

Assembler behavior

The assembler behavior controls the usage and overwriting of the following directives.

Default charset behavior

Message	Help link	Source link
Assembling...		
Open project: Done		

Font size: 9 pt Mode: Insert | Processor: Xilinx PicoBlaze 3 | Release: 2014-9.0 License: Professional Expire date: 2015-09-10

Figure 9.3-1: Assembler configuration dialog page

10 Some notes about case sensitivity

FIDEx parses your code case insensitive with the following exceptions:

- Labels are case sensitive
- The names defined in equ directives are case sensitive

11 Some notes about strings

FIDEx supports the usage of strings with default character set and string specific character set defined as attribute.

Please refer to chapter “Strings” of the coding manual.

12 Saving configuration data

FIDEx has to save some configuration data. Therefore two locations are used:

- the users home directory for storing project independent application configuration data,
- the project file for storing project dependent configuration data.

On Microsoft Windows platforms FIDEx doesn't save its configuration data into the Windows registry!

List of tables

List of figures