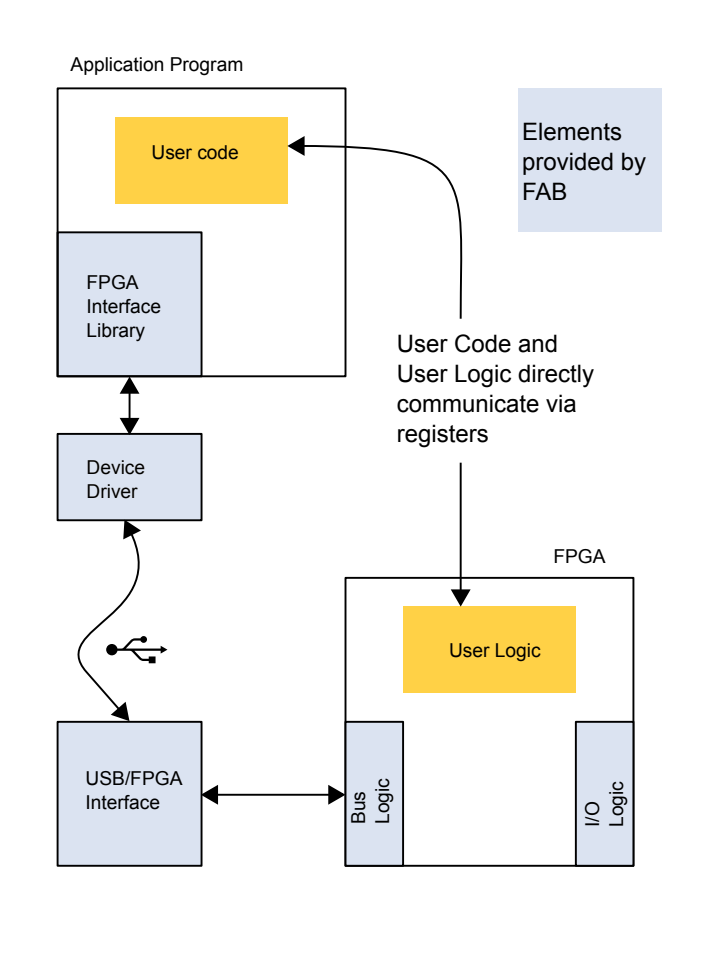
1 Overview

The FAB-3226 system connects a user defined FPGA logic design with a user defined application program running on a computer system (PC or embedded). The FAB-3226 hardware is connected to the computer via an USB connection.

In order to communicate with the application, the logic implements a set of registers. The application exchanges data with these registers by read and write requests respectively. This way the application can control the logic.

No firmware development is required as the FAB kit contains all the neccessary drivers.

The FPGA logic is **soft**, i.e. the application initializes the FPGA upon each start. The FPGA is de-initialized (reset) when the application terminates. By use of an server application the FPGA logic might persist even when a single client terminates.

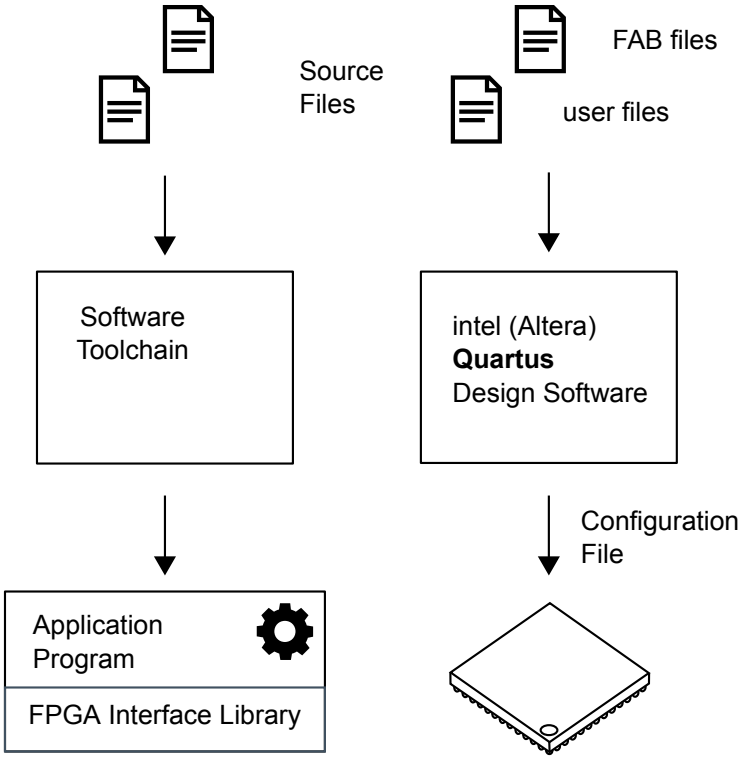


2 Design Tools

A hardware and software toolchain are required to deploy the FAB-3226: Altera/Intel Quartus for logic design, a C/C++ compiler for application software. GCC (GNU Compiler Collection) for Linux and Windows (mingw), Visual C++ for Windows. All toolchains are available as Web Editions for download free of charge.

All the neccessary design elements are provided by the FAB kit: Quartus source files, C++ include files and libraries. Ready to run example applications serve as starting point for user specific developments.

Figure 2: FAB kit Design Tools



3 Application Programming

Class Documentation

Class documentation for FAB kit is provided here: annotated class reference, file list, namespace list, functions, examples.

Class RioTaskPlex connects to the FAB-3226. It allows to initialize the FPGA (callRioInitFile()), read registers (callRioRead()), and write registers (callRioWrite()). You can control one or more FAB-3226, per unit connected there is exactly one object of this type.

The fab_basic example demonstrates how to load a .rbf programming file into the FPGA. The FPGA is held initialized during the runtime of the example program. After termination of the program the FPGA is held in reset.

The fab_regs example demonstrates (after loading a .rbf programming file into the FPGA) how to communicate with the FPGA logic via register data transfers. After termination of the program the FPGA is held in reset.

4 Logic Synthesis

The Quartus design tool is used for logic synthesis for the FAB-3226. The directories in the `quartus` branch of the installation tree contain the neccessary source files and examples for logic synthesis. Each directory contains the following files:

`fiora_fxio.sdc`	Timing constraints for simulation and synthesis.
`fiora_fxio.qsf`	Settings and pin assignments for fxio IO-ring architecture. This file is read-only and must not be changed. **Warning: Changing the settings in this file might result in hardware failure and/or damage.**
`fiora_fxio.vh`	contains the interface declaration of the Fx-Io interface. It must be included after the opening '(' character of the module declaration's port_list. It contains the port_list, closing ')' character and ports_declaration of the module declaration.
`fab.qsf`	Project-wide and entity-level assignments and settings for the current revision of the project. `fab.qsf` sources (includes) `fiora_fxio.qsf` with the specific settings for the FAB-3226 board
`fab.qpf`	Quartus project file.
`fab.v`	HDL file containing the top-level design entitity.

The `fab.v` always contain the toplevel module named `fab`:

```
  module fab(
❶ `include "fiora_fxio.vh"
    reg[27:0]  rwTimer;
    initial begin
      rwTimer <= 0;
    end
    always @ (posedge clock) begin
      rwTimer <= (rwTimer == 30000000) ? 0  : rwTimer + 1;
    end
    assign #1 owLED[0] = rwTimer[24];
  endmodule
```

❶ `fiora_fxio.vh` contains the interface declaration of the Fx-Io interface. It must be included after the opening '(' of the module declaration's port_list. It contains the port_list, closing ')' and ports_declaration of the module declaration.

Fx-Io

The Fx-Io environment (a.k.a. FGPA I/O Ring Architecture = fiora) specifies signals and pins and thus the port list of the top-level design entity (module). The port list contains the following members relevant to the user:

`input [15:0] iwDin;`	Input data from the 16 I/O cells to the FPGA.
`output [15:0] owDout;`	Output data from the FPGA to the I/O cells.
`output [15:0] owDoen;`	Output Enable from the FPGA to the I/O cells. Active low, i.e. a logic 0 enables the corresponding output. A logic 1 disables the output (High-Z). When the FPGA is unconfigured, the weak pull-up resistors at the FPGA pins disable all outputs.

`wire clock;`	30 MHz clock. Reference clock for all interface signals, except TBD ...
`wire[4:0] wwAddr;`	Address for register I/O operations.
`wire wbRdp;`	Read Pulse active one clock cycle during a register read operation.
`wire wbWrp;`	Write Pulse active one clock cycle during a register write operation.
`wire [15:0] wwRdData;`	Read data to be presented by the core during read operations.
`wire [15:0] wwWrData;`	Write data to be latched by the core during write operations.

Read-Registers

The Read register set is implemented by a multiplexer connecting the individial read register bits with the `wwRdData` bus. `wwAddr[4:0]` define the currently selected register address to be read. That's all required to read values from registers:

```
assign #1 wwRdData[15:0] = wwAddr == 0 ? wwReg0[15:0] :
  (wwAddr == 1 ? wwReg1[15:0] :
  (wwAddr == 2 ? wwReg2[15:0] :
  16'ha55a));
```

If a read acknowledge is required, `wwRdp` must be ANDed with the relevant address `wwAddr[4:0]`:

```
wire wbRead10 = (wwAddr == 10) && wbRdp;
```

The inputs of the read multiplexer may be constants, regs, inputs, of any combinatoric operation on them.

Write-Registers

Write registers are implemented by `reg` instances that are synchronously reloaded if the write pulse is active together with the respective address present.

```
reg[1:0] rwLed; /* two bit LED register */
initial begin
  rwLed <= 0; /* default value on start-up */
end
always @ (posedge clock) begin
  if (wwAddr == 1 && wbWrp) rwLed[1:0] <= wwWrData[1:0] ;
end
```

Reset

As the FAB-3226 is always initialized when the controlling application begins, a dedicated reset signal is not neccessary. Hence it is not provided by the FX interface. Register initialization is done by `initial` blocks in the HDL code. If an explicit reset is required (e.g. by existing HDL modules with reset input), it can be implemented in two ways:

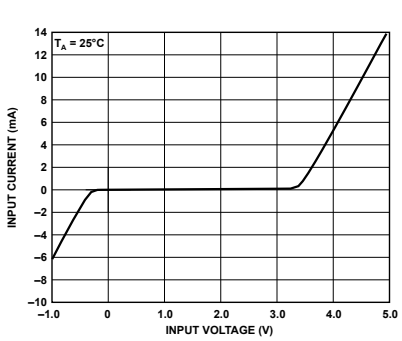
- via a synchronous reset signal. This signal can be set by a "reset command" written by the controlling application, and/or a reset register set true in an initial block then set false after the first clock cycle(s).
- via an asynchronous reset by use of an address line. This asynchronous reset becomes true by a read or write of an address with the selected address line set. It becomes false by a read or write of an address

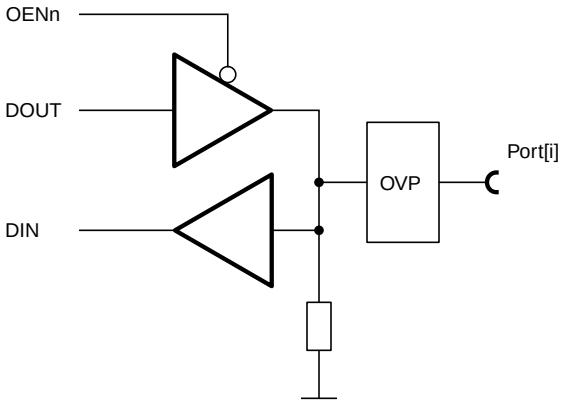
with the selected address line unset. Please note that the selected address line will be no more available for regular addressing.

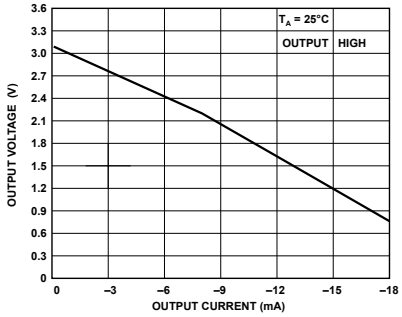
5 Connecting External Signals

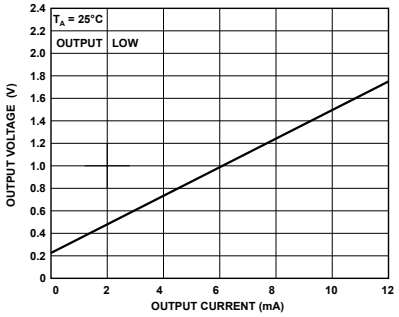
You can connect up to 16 digital I/O signals to the FAB-3226. Digital I/O lines are labeled from IO00 to IO15. GND is the ground-reference for all lines. Each signal port can be programmed to be either input or output. At power-on or reset, all IO lines are set to high-impedance inputs with weak pull-down resistors (it does not drive the line high or low). See Figure 3.

Figure 3: FAB-3226 I/O characteristics









FAB-3226 Signal by Pin

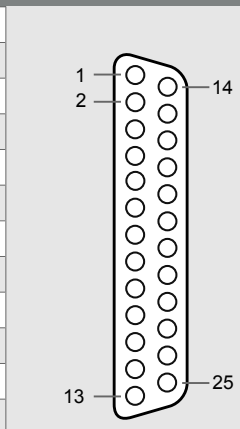
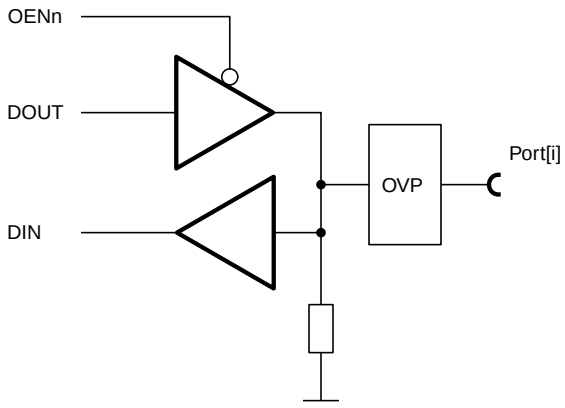
Signal	Pin	Connector View	Pin	Signal
3,3V	1		14	GND
IO15	2		15	IO14
GND	3		16	IO13
IO12	4		17	GND
IO11	5		18	IO10
GND	6		19	IO09
IO08	7		20	GND
IO07	8		21	IO06
GND	9		22	IO05
IO04	10		23	GND
IO03	11		24	IO02
GND	12		25	IO01
IO00	13			

Figure 4: Title



Important Notice

No references to the C++ API documentation are available in this standalone version of the manual. In order to access the C++ API documentation download and install the complete driver software package.

Trademarks

Product, service, or company names used in this document are for identification purposes only and may be either trademarks or registered trademarks of the relevant trademark owners. LabView, NI-488.2, LabWindows, PXI, DASYLab, DIAdem are trademarks or registered trademarks of National Instruments Corp., USA, in the United States and/or other countries. Microsoft, Windows, Windows NT, Windows CE, Windows 2000, Windows ME, Windows XP, Windows Vista, Visual Basic, Visual-C++ are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Specifications

All specifications are subject to change without prior notice.

Limited warranty and liability

Information in this document is believed to be accurate and reliable. However, Ines does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. Ines takes no responsibility for the content in this document if provided by an information source outside of Ines. In no event shall Ines be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, Ines' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of Ines.

Software

ALL SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.