EFM 01 examples

EFM-01-examples.pdf

This document was automatically created.

Please visit www.cesys.com for the latest version.

July 30, 2012, 9:58 pm

Introduction to example FPGA designs

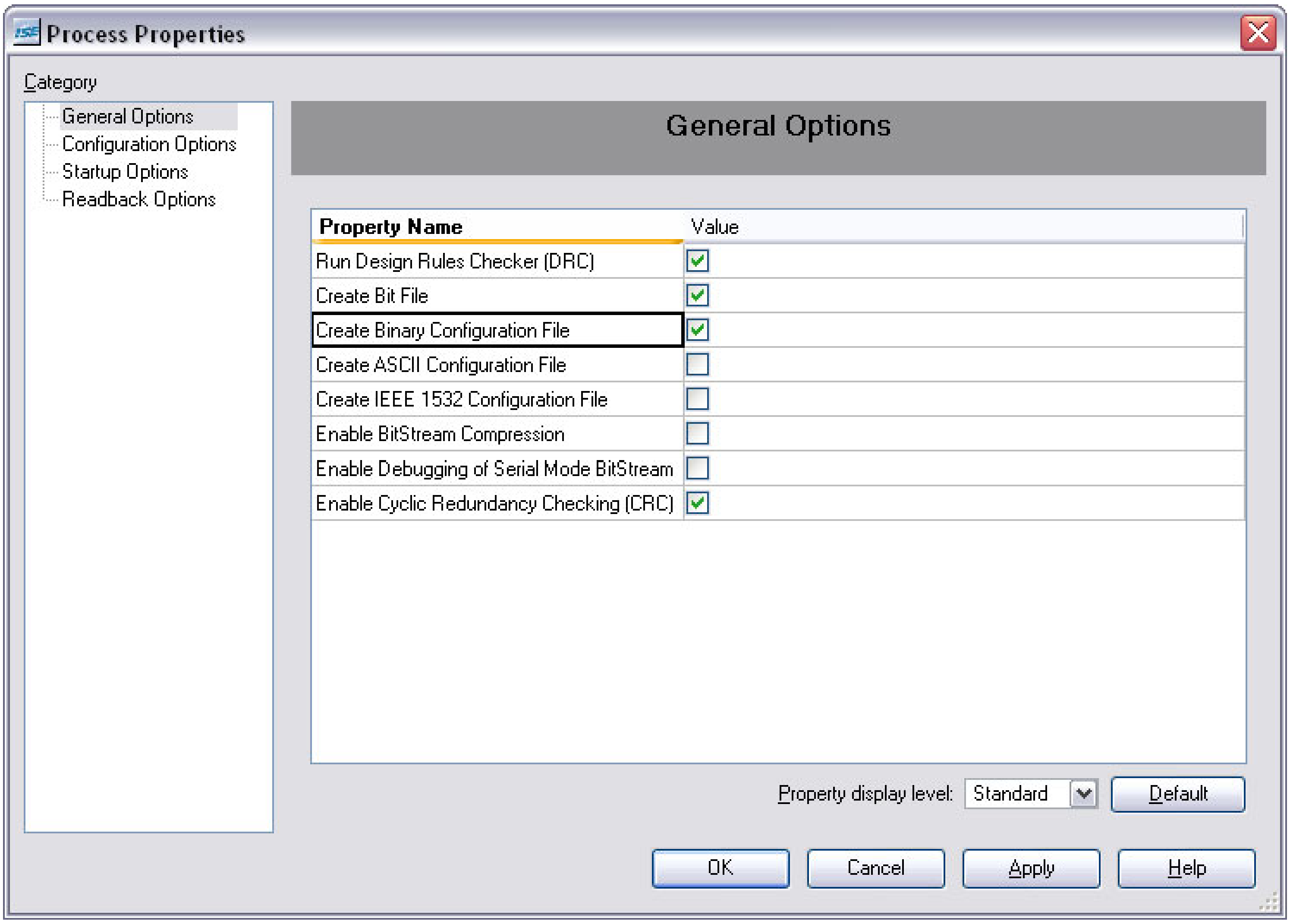
The CESYS EFM01 FPGA-module comes with some demonstration FPGA designs (part of the UDK) to provide an easy starting point for your own development projects. The whole source code is written in VHDL. Verilog and schematic entry design flows are not supported.

The design “efm01” demonstrates the implementation of a system-on-chip (SOC) with USB access to peripherals like GPIOs, Flash Memory and BlockRAM. This design requires a protocol layer over the simple USB bulk transfer (see CESYS application note “Transfer Protocol for CESYS USB products” for details), which is already provided by CESYS software API.

The design “efm01_perf” allows high speed data transfers from and to the FPGA over USB and can be used for software benchmarking purposes. This design uses 512 byte aligned USB bulk transfer without an additional protocol layer.

The Spartan-3E XC3S500E Device is supported by the free Xilinx™ ISE Webpack development software. You will have to change some options of the project properties for own applications.

A bitstream in the “*.bin”-format is needed, if you want to download your FPGA design using the CESYS software API-functions **LoadBIN()** and **ProgramFPGA()**. The generation of this file is disabled by default in the Xilinx™ ISE development environment. Check “createbinary configuration file” at right click “generate programming file”=>properties=>generaloptions:



Property Name	Value
Run Design Rules Checker (DRC)	✓
Create Bit File	✓
Create Binary Configuration File	✓
Create ASCII Configuration File	☐
Create IEEE 1532 Configuration File	☐
Enable BitStream Compression	☐
Enable Debugging of Serial Mode BitStream	☐
Enable Cyclic Redundancy Checking (CRC)	✓

After ProgramFPGA() is called and the FPGA design is completely downloaded, the pin #FPGA_RESET (note: the prefix # means, that the signal is active low) is automatically pulsed (HIGH/LOW/HIGH). This signal can be used for resetting the FPGA design. The API-function ResetFPGA() can be called to initiate a pulse on #FPGA_RESET at a user given time.

The following sections will give you a brief introduction about the data transfer from and to the FPGA over the Cypress FX-2 USB peripheral controller's slave FIFO interface, the WISHBONE interconnection architecture and the provided peripheral controllers.

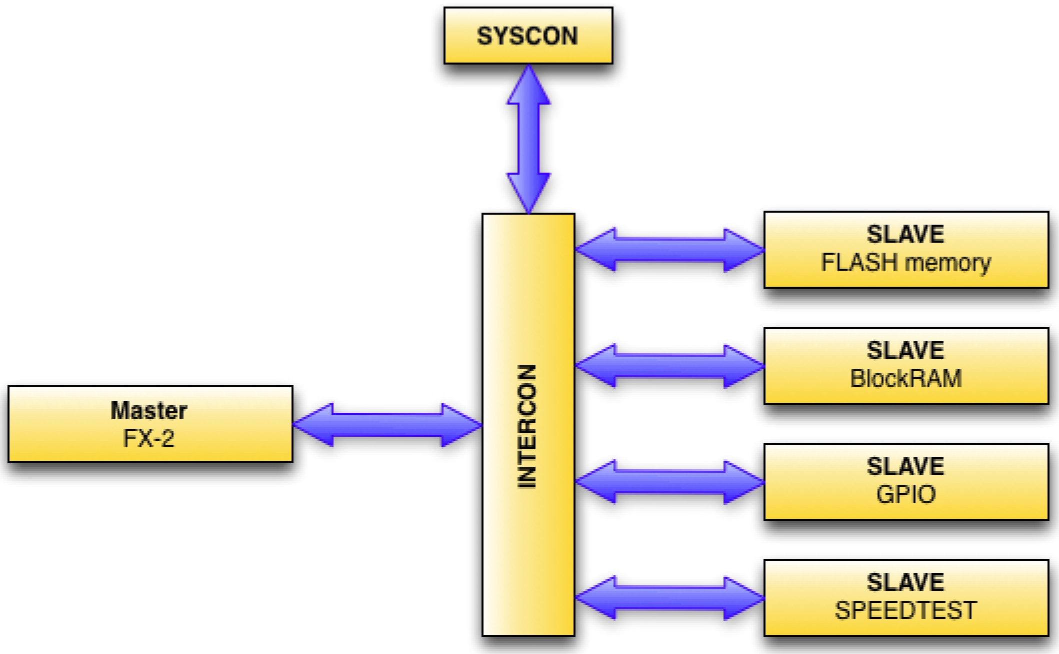
The EFM01 uses only slave FIFO mode for transferring data.

For further information about the FX-2 slave FIFO mode see Cypress FX-2 user manual (EZ-USB_TRM.pdf) and datasheet (cy7c68013a_8.pdf) and about the WISHBONE architecture see specification B.3 (wbspec_b3.pdf).

Design “efm01”

An on-chip-bus system is implemented in this design. The VHDL source code shows you, how to build a 32 Bit WISHBONE based shared bus architecture. All devices of the WISHBONE system support only SINGLE READ / WRITE Cycles. Files and modules having something to do with the WISHBONE system are labeled with the prefix “wb_”. The WISHBONE master is labeled with the additional prefix “ma_” and the slaves are labeled with “sl_”.

WISHBONE system overview



Files and modules

src/wishbone.vhd:

A package containing datatypes, constants, components, signals and information for software developers needed for the WISHBONE system. You will find C/C++-style “#define”s with important addresses and values to copy and paste into your software source code after VHDL comments (“--”).

src/efm01_top.vhd:

This is the top level entity of the design. The WISHBONE components are instantiated here.

src/wb_syscon.vhd:

This entity provides the WISHBONE system signals RST and CLK. It uses #FPGA_RESET and SYSCLK as external reset and clock source. SYSCLK is identically to FX2_IFCLK. That means FX-2 slave FIFO interface and WISHBONE system are fully synchronous.

src/wb_intercon.vhd:

All WISHBONE devices are connected to this shared bus interconnection logic. Some MSBs of the address are used to select the appropriate slave.

src/wb_ma_fx2.vhd:

This is the entity of the WISHBONE master, which converts the CESYS USB protocol into one or more 32 Bit single read/write WISHBONE cycles. The low level FX-2 slave FIFO controller (fx2_slfifo_ctrl.vhd) is used and 16/32 bit data width conversion is done by using special FIFOs (sfifo_hd_a1Kx18b0K5x36.vhd).

src/wb_sl_bram.vhd:

A internal BlockRAM is instantiated here and simply connected to the WISHBONE architecture. It can be used for testing address oriented data transactions over USB.

src/wb_sl_speedtest.vhd:

A single register with zero delay slave handshake response. It can be used for benchmarking purposes. Auto address increment must be deactivated.

src/wb_sl_gpio.vhd:

This entity controls the signals at connectors J3 and J4. 50 I/Os can be used as general purpose I/Os. Each of these I/Os can be configured as an in- or output. Additional pinout information is provided by an embedded comma separated values file after VHDL comments (“--”).

src/wb_sl_flash.vhd:

The module encapsulates the low level FLASH controller flash_ctrl.vhd. The integrated command register supports the BULK ERASE command, which erases the whole memory by programming all bits to '1'. In write cycles the bit values can only be changed from '1' to '0'. That means, that it is not allowed to have a write access to the same address twice without erasing the whole flash before. The read access is as simple as reading from any other WISHBONE device. Please see the SPI-FLASH data sheet (m25p40.pdf) for details on programming and erasing. It is used for programming FPGA configuration bitstream to SPI-FLASH.

src/fx2_slfifo_ctrl.vhd:

This controller copies data from FX-2 endpoints to internal FPGA buffers (sync_fifo16.vhd) and vice versa.

src/sync_fifo16.vhd:

This entity is a general purpose synchronous FIFO buffer with 15 data entries. It is build of FPGA distributed RAM.

src/sfifo_hd_a1Kx18b0K5x36.vhd:

This entity is a general purpose synchronous FIFO buffer with mismatched port widths. It is build of a FPGA BlockRAM.

src/flash_ctrl.vhd:

The low level FLASH controller for the 4MBit SPI FLASH memory. It supports reading and writing of four bytes of data at one time as well as erasing the whole memory.

efm01.ise:

Project file for XilinxTM ISE

efm01.ucf:

User constraint file with timing and pinout constraints

Wishbone transactions

The software API-functions ReadRegister(), WriteRegister() lead to one and ReadBlock(), WriteBlock() to several consecutive WISHBONE single cycles.
Bursting is not allowed in the WISHBONE demo application. The address can be incremented automatically in block transfers. You can find details on enabling/disabling the burst mode and address auto-increment mode in the CESYS application note “Transfer Protocol for CESYS USB products” and software API documentation.

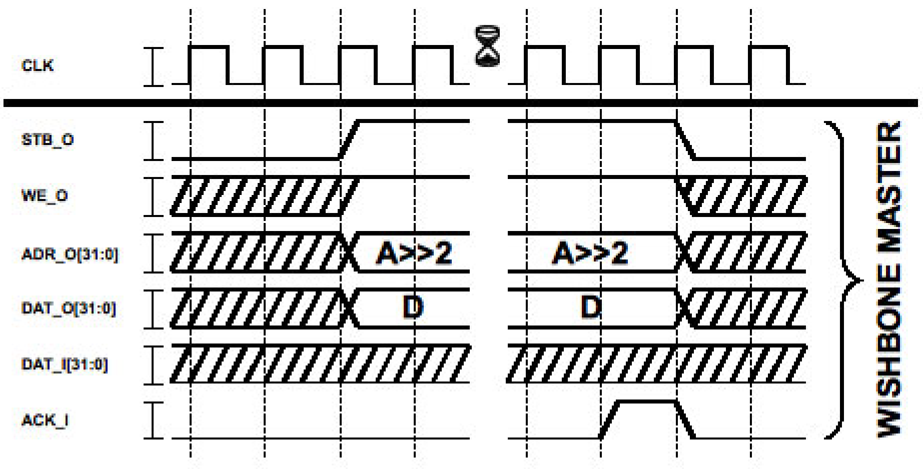
CESYS USB transfer protocol is converted into one or more WISHBONE data transaction cycles. So the FX-2 becomes a master device in the internal WISHBONE architecture. Input signals for the WISHBONE master are labeled with the postfix “_I”, output signals with “_O”.

WISHBONE signals driven by the master

- **STB_O**: strobe, qualifier for the other output signals of the master, indicates valid data and control signals
- **WE_O**: write enable, indicates, if a write or read cycle is in progress
- **ADR_O[31:0]**: 32-Bit address bus, the software uses BYTE addressing, but the WISHBONE system uses DWORD (32-Bit) addressing. The address is shifted two bits inside the WISHBONE master module
- **DAT_O[31:0]**: 32-Bit data out bus for data transportation from master to slaves

WISHBONE signals driven by slaves

- **DAT_I[31:0]**: 32-Bit data in bus for data transportation from slaves to master
- **ACK_I**: handshake signal, slave devices indicate a successful data transfer for writing and valid data on bus for reading by asserting this signal, slaves can insert wait states by delaying this signal, it is possible to assert ACK_I in first clock cycle of STB_O assertion using a combinatorial handshake to transfer data in one clock cycle (recommendation: registered feedback handshake should be used in applications, where maximum data throughput is not needed, because timing specs are easier to meet)



The WISHBONE signals in these illustrations and explanations are shown as simple bit types or bit vector types, but in the VHDL code these signals could be encapsulated in extended data types like arrays or records.

Example:

```
...
port map
(
  ...
  ACK_I => intercon.masters.slave(2).ack,
  ...
```

Port ACK_I is connected to signal ack of element 2 of array slave, of record masters, of record intercon.

Copyright information

FPGA source code copyright information

FPGA source code license

Disclaimer of warranty

Design “efm01_perf”

This design is intended to demonstrate behavior of low level slave FIFO controller entity fx2_slfifo_ctrl. It handles the FX-2 slave FIFO interface.
It can be synthesized in two modes, data loopback mode and infinite data source/sink mode with 16 bit counting data source.

Ports of fx2_slfifo_ctrl connected to FX-2 are labeled with prefix fx2_ and ports connected to user logic are labeled with prefix app_.
Sometimes the abbreviations _h2p_ (host to peripheral) and _p2h_ (peripheral to host) are used in signal names to indicate data flow direction.

Files and modules

src/efm01_perf.vhd:

This is the top level module. It instantiates the low level slave FIFO controller (fx2_slfifo_ctrl.vhd). A generic variable selects between data loopback and infinite data mode at synthesis time.

src/fx2_slfifo_ctrl.vhd:

See Wishbone example “Design efm01”

src/sync_fifo16.vhd:

See Wishbone example “Design efm01”

efm01_perf.ise:

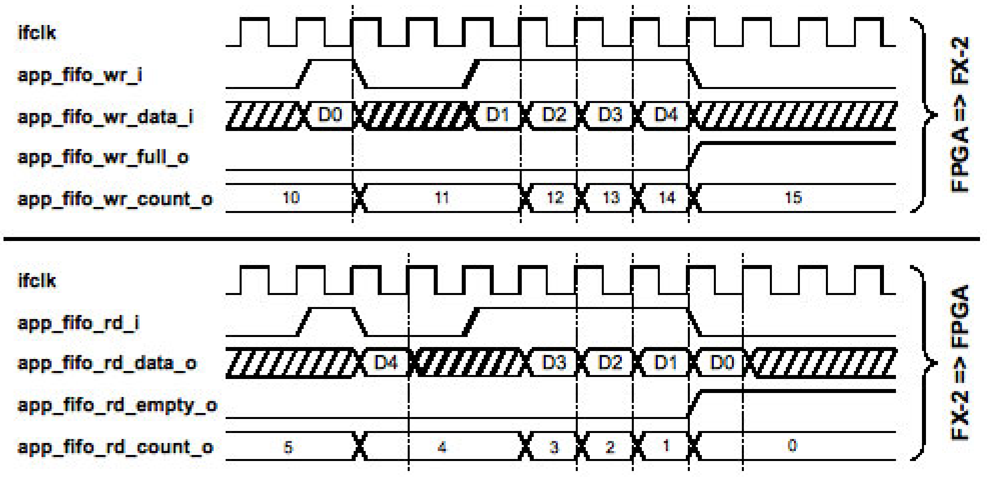
Project file for XilinxTM ISE.

efm01_perf.ucf:

User constraint file with timing and pinout constraints.

Slave FIFO transactions

The software API functions ReadBulk() and WriteBulk() lead to 512 byte aligned USB bulk transfers without CESYS USB transfer protocol. So it is possibly to achieve maximum data rates over USB. fx2_slfifo_ctrl checks FX-2 FIFO flags and copies data from FX-2 endpoint buffers to FPGA and vice versa. So the USB data link looks like any other FPGA FIFO buffer to user logic.



FIFO transactions with ReadBulk() and WriteBulk() at user logic side

The upper waveform demonstrates the behavior of app_fifo_wr_full_o and app_fifo_wr_count_o when there is no transaction on the slave FIFO controller side of the FIFO. During simultaneous FIFO-read- and FIFO-write-transactions, the signals do not change. The signal app_fifo_wr_full_o will be cleared and app_fifo_wr_count_o will decrease, if there are read-transactions at the slave FIFO controller side, but no write-transactions at the application side.

The lower waveform demonstrates the behavior of app_fifo_rd_empty_o and app_fifo_rd_count_o when there is no transaction at the slave FIFO controller side of the FIFO. During simultaneous FIFO-read- and FIFO-write-transactions, the signals do not change. The signal app_fifo_rd_empty_o will be cleared and app_fifo_rd_count_o will increase, if there are write-transactions on the slave FIFO controller side, but no read-transactions at the application side. Please note the one clock-cycle delay between app_fifo_rd_i and app_fifo_rd_data_o!

The signals app_usb_h2p_pktcount_o[7:0] and app_usb_p2h_pktcount_o[7:0] (not shown) are useful to fit the 512 byte USB bulk packet alignment. They are automatically incremented, if the appropriate read- (app_fifo_rd_i) or write-strobe (app_fifo_wr_i) is asserted. These signals count 16 bit data words, not data bytes! 512 byte alignment is turned into a 256 16 bit word alignment at this interface.