Prepared by M. Wasif Adnan at IfA

Avnet Lx9 Microboard + Lattice LM32 Softcore Processor

Configuration and Programmer's Guide

Notes:

1. Avnet Lx9 Microboard is equipped with a Xilinx Spartan 6 (Lx9) FPGA.
2. The document mostly focuses on integrating Lattice IP with Xilinx tools. Information given in referenced tutorials is not repeated.
3. Most documents referenced in the document can be found in the folder "Reading Material". For the rest, links are provided.
4. For sections marked with (**) It is assumed that the reader is familiar with development on Xilinx ISE. Software programmers do need to read those sections.

Software Required

1. Xilinx ISE Logic Edition (13.1 or better)
2. Digilent Adept System (latest)
    - Digilent plug-in for Xilinx ISE (make sure that it is valid for your version of ISE)
3. Lattice Diamond** (analogous to Xilinx ISE for Lattice FPGAs)
    - Only required for creating HDL for the LM32 based SoC. SW Programmers do not need to install this.
4. LatticeMico System for Lattice Diamond 2.1 (analogous to Xilinx EDK)
    - Used to configure CPU (LM32), and peripherals. Also used to write and compile software for the processor. Automatically adds the software libraries according to the SW configuration.

    http://www.latticesemi.com/Products/DesignSoftwareAndIP.aspx

LatticeMico32 Processor – Introduction

1. 32-bit RISC processor, with Harvard architecture, Big Endian memory interface
2. Implemented in Verilog HDL
3. Optimized for Lattice FPGAs, but performs quite well for Xilinx FPGAs (better than open-source soft-core processors designed for ASICs)
    - The document titled *'**Selection of a Soft-core Processor for Real-Time Control**'* contains a comparison of open-source soft-core processors
4. 6 stage pipelining, single cycle instruction execution

5. Uses Wishbone bus – WISHBONE B3

   http://cdn.opencores.org/downloads/wbspec_b3.pdf

Integrating a LM32 based SoC with Xilinx tools**

Notes of creating and configuring LM32 based SoC on LatticeMico System:

1. The documents below provide a guide to using LatticeMico System
   a. LatticeMico System User Guide
   b. LatticeMico32 Hardware User Guide

2. Following functionalities have not been tested, so they may not work with Xilinx tools
   a. JTAG Debugger for the CPU
   b. Interface to external SRAM or SDRAM
   c. Cache functionality

The following configuration for the LM32 based SoC has been tested:

1. 32 bit CPU – (Caches disabled, JTAG debugger disabled)
2. GPIO
3. UART
4. Data/Instruction Memory via WISHBONE Bus (Option 1)
   a. Due to arbitration issues, memory access has a larger delay. Since this approach also requires Block RAMs, Option 2 is better.
5. Inline Data/Instruction memory (Option 2)
   a. Separate Block Memory for both Data (2KB) and Instruction (16KB)
   b. No arbitration or caching in between instruction and data paths.
   c. This allows memory access of two clock cycles. Without above design, memory access was costlier.
6. Direct access to slave interface on Wishbone bus (Memory Mapped).
   a. Shared Bus Architecture (Option 1)
   b. Slave Side Arbitration (Option 2)

Steps for Integration with Xilinx Tools

1. In LatticeMico System, design the processor architecture. Insert the CPU, peripherals and connect them to the WB bus (in GUI).
2. Generate the HDL source files (verilog) from LatticeMico System.
3. In ISE, add the file <hdl_directory>/cpu/soc/cpu.v. All other required source files are added to the project automatically, according to the proper hierarchy.
4. Comment the inclusion (`include) of the file ‘pmi_def.v’ from the file ‘lm32_include_all.v’. This file contains simulation parameters that are Lattice specific. This may necessitate inclusion of `timespec directives in some v files.

5. SYNCHRONOUS RST –Enable synchronous RST, to give better optimization on Xilinx FPGAs – change the macro CFG_RESET_SENSITIVITY in the file 'lm32_include.v'
6. Identify the missing source files. These correspond to missing - Lattice FPGA specific - primitives (e.g. block RAM, arithmetic blocks, etc.).
7. Replace lattice specific primitives with Xilinx primitives (e.g. use Code Generator etc). Make sure that the functionality of the Xilinx primitives matches those of Lattice primitives (esp. for memory primitives e.g registered ports, dual/single port etc).
   a. In some cases, some basic logic may be added to mimic the functionality of Lattice primitives. (e.g. almost full/empty flags for FIFOs)
   b. The project files include a folder with all verilog files that had to be modified for integration. The details of the modifications are given at the end of the files.
8. Construct a top wrapper, instantiate the core into it. Route the outputs to the correct pins in the ucf file.
   a. For the reset signal 'sys_reset' in the top level of cpu, set attribute *(*TIG="TRUE"*)*

*In the project files, the folder titled **'Xilinx Compatibility Mods'** contains the (Verilog) source files of LM32 SoC that were modified for integration with Xilinx. Scroll to the end of those files to see the modification log.*

Using Block RAM for Code/Data memory

1. Make sure size of block RAM is sufficient for code space required. For LX9, maximum block RAM size is 64KB.
2. The implementation of Block RAM in (Code Generator) should be carefully set. Make sure that the division of the memory space among RAMB16 primitives is known. Use the option 'Fixed Primitive Algorithm' for this in COREGEN. Also make sure that RAMB8 primitives are not used. There are two possible schemes:
   a. The 32-bit data bus is divided among separate RAMB16 primitives. (typically 4-bits per primitive). All primitives get the same address
   b. The address space is divided among separate RAMB16 primitives. Every primitive has a 32-bit data bus. *This approach is preferable for separate Code and Data memories.*
3. Build code in LatticeMico System to generate elf file.
4. **Block Memory Map File (BMM)**

   BMM file specifies the location of specific memory sections on Block RAM primitives. This is used to write the contents of code memory to the corresponding block RAM. (see next section).

Create a BMM file manually by looking at the exact location of the Block RAM primitives on FPGA (use FPGA Editor or PlanAhead). The organization of memory space among block memory primitives (see point 2, above) should be known. *Keep in mind that the LM32 is a BIG_ENDIAN processor and the elf file is created accordingly. Whereas, everything in BlockRAM is arranged in Little-endian.*(For reference, look at the sample file 'blkmemMap_lm32.bmm'.

Refer to following documents to get details of creating BMM files and using them to modifying the bit files:

http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/data2mem.pdf

http://www.xilinx.com/Attachment/Xilinx_Answer_46945_Data2Mem_Usage_and_Debugging_Guide.pdf

Using Xilinx data2mem Utility to update bitfile with Software Executable

1. Put the bit file of FPGA design, executable file of software (elf), and a correctly written bmm file in one location.
2. Open *ISE Design Suite Command Prompt* and navigate to the above directory.
3. Use Xilinx tool – ***data2mem*** – to insert the elf file contents into the bitfile. The command line instruction is:

```
data2mem -bm <bmm_file_name>.bmm -bd
<elf_file_name >.elf -bt <bitfile_name> -o b
download.bit
```

4. Program the updated bit file (download.bit) in the FPGA, using IMPACT.

Configuring the FPGA via JTAG-USB port

The Avnet Lx9 Micorboard is equipped with one USB, and one Micro-USB port.

1. The Micro-USB port is connected to a UART-USB controller. Its interfacing is fairly standard. Upon connection to a PC, the necessary drivers are automatically installed.

2. The other USB port is used to configure the FPGA via the JTAG chain. It can also be used to program the FLASH.

Use the document below to learn the details of installing the Plug-in for IMPACT

LX9 Configuration Guide

http://www.em.avnet.com/Support%20And%20Downloads/Avnet_Spartan-6_LX9_MicroBoard_Configuration_Guide_v1_1.pdf

Note on Programming Flash:

The document, given above, also explains the procedure to program the flash.

However, because of the method used to configure the bit file, using the data2mem utility, the data bus width of the SPI bus can only be set to 1 (not 4, as given in the document).