An Embedded Reconfigurable Logic Module

Final Report
For
NASA Langley Research Center

NASA grant
NAG-1-01042
NLPN: 01-111

Principal Investigator:
Jerry H. Tucker
Associate Professor of Electrical Engineering
Virginia Commonwealth University

Co-Principal Investigator:
Robert H. Klenke
Associate Professor of Electrical Engineering
Virginia Commonwealth University

NASA Technical Monitor
Qamar A. Shams
Langley Research Center
Abstract

A Miniature Embedded Reconfigurable Computer and Logic (MERCAL) module has been developed and verified. MERCAL was designed to be a general-purpose, universal module that can provide significant hardware and software resources to meet the requirements of many of today's complex embedded applications. This is accomplished in the MERCAL module by combining a sub credit card size PC in a DIMM form factor with a XILINX Spartan II FPGA. The PC has the ability to download program files to the FPGA to configure it for different hardware functions and to transfer data to and from the FPGA via the PC's ISA bus during run time. The MERCAL module combines, in a compact package, the computational power of a 133 MHz PC with up to 150,000 gate equivalents of digital logic that can be reconfigured by software. The general architecture and functionality of the MERCAL hardware and system software are described.

1. Introduction

Desktop applications are now dominated by the IBM compatible PC. This is due to high performance coupled with a combination of low cost hardware and, a wide verity of inexpensive software. Because of the pervasiveness of PC's and their sophisticated development tools, PC based software development is more cost effective than other platforms. This makes the PC attractive for use in embedded systems. Particularly in few-of-a-kind systems where development cost cannot be prorated over many systems. Unfortunately the size and power requirements of PC's precluded their use in most embedded applications. The MERCAL module has been designed to address these and other issues. It provides a single consistent platform capable of satisfying the requirements of many embedded applications. The MERCAL module offers the power and flexibility of an IBM compatible personal computer in a size ideally suited for many embedded applications. It provides a single platform that is constant, flexible, and reliable. The MERCAL module contains configurable logic, in the form of an FPGA, which either entirely eliminates or drastically reduces the need for the digital interface cards required in embedded systems using conventional PC's. Using MERCAL, the only additional hardware required by an embedded system would typically be the converters and drivers specifically required by the applications. All PC interface and application control logic is contained in the FPGA internal to MERCAL and can be configured and optimized to suit the application. A standard desktop PC can be used as a development platform.

2. The MERCAL module

The block diagram of the MERCAL module is shown in Figure 1. There are two primary components to the MERCAL module. They are the DIMM-PC and a Xilinx FPGA. The only other active components are a power converter and a 32 MHz oscillator used to clock the FPGA. The passive components consist primarily of capacitors, with a
few resistors and diodes. Three connectors are used to interface to the MERCAL module. The interface to the MERCAL module provides 81 general-purpose input output (IO) pins from the FPGA. The functions of these IO pins can be determined by the needs of the application and controlled by the configuration of the FPGA. In addition to the general-purpose FPGA IO pins, signals are also available from the DIMM-PC. The DIMM-PC signals include two RS232 serial communication ports, and certain other selected signals. These signals were selected to provide sufficient flexibility and capability to support advanced applications. For example, these selected signals have been used to provide the capability of using a compact flash memory as an IDE disk drive.

Top and bottom views of the actual MERCAL module are shown in Figures 2 and 3. The schematic diagram of the MERCAL module is shown in Appendix A. The pin outs of the MERCAL module are shown in appendix B.

A standard desktop PC can be used as a development platform, and software has been developed so that the DIMM-PC can serially download a bit file to configure the FPGA. The configuration file is downloaded using the printer port interface built into the DIMM-PC. This made it possible to configure the FPGA without requiring the addition of separate logic.

![Figure 1. Block diagram of the MERCAL module.](image-url)
Figure 2. Top view of the MERCAL module showing the DIMM-PC, 32 MHz oscillator IC, and power converter IC.

Figure 3. Bottom view of the MERCAL module showing the Spartan II FPGA and the three connectors.
2.1 The DIMM-PC

The DIMM-PC is shown in Figure 2. It is a commercially available, functionally complete, extremely compact (40 X 67 X 6 mm) PC motherboard. Several versions of the DIMM-PC are available that can be used in the MERCAL module. The DIMM-PC processor can either be a 66 MHz 486SX for low-end applications or a 133 MHz AMD Elan SC586 for more demanding applications. Typically the onboard memory consists of 16 to 32 Mbytes of RAM and a 16 to 32 Mbyte Flash Disk. The DIMM-PC peripheral interface consists of two serial ports, one parallel printer port, keyboard, floppy, and IDE Hard disk controller port. In MERCAL, the peripheral interface is used primarily for development and diagnostic purposes. However, the printer port is dedicated to the task of programming the Xilinx FPGA. Since this device is a SRAM based FPGA, it is possible, with software that has been developed, for the DIMM-PC to reconfigure the FPGA to satisfy the digital logic requirements for various applications.

Detailed information including specifications and user manual of the DIMM-PC can be obtained from http://www.jumptecadastra.com/iaad014_dimm.html.

2.2 The XILINX FPGA

The FPGA used in MERCAL is the Xilinx XC2S150 Spartan-II in a PQ208 package. About half of the available I/O pins on the FPGA are used to interface to the DIMM-PC and the others are available external to the MERCAL module through connectors. The XC2S150 Spartan-II FPGA contains the equivalent of 150,000 gates with 200 MHz system performance. A complete description of the Spartan-II can be found at http://www.xilinx.com/. Various tools are available that can be used to implement the FPGA portion of a design. Typically, either schematic capture or a hardware description language such as VHDL will be used to specify the particular implementation. Several examples using schematic capture are shown in Appendix E.

2.3 DIMM-PC to FPGA interface

The interface between the DIMM-PC and the FPGA consists of two parts. Both parts of the interface are realized without the need for external logic.

The first part of the interface is required to download configuration files from the DIMM-PC to the FPGA. The configuration is accomplished by placing the FPGA in slave serial mode, and using selected pins of the DIMM-PC’s printer port to control the DIN and CCLK pins of the FPGA. Software developed by electrical engineering students at Virginia Commonwealth University is used to transfer “bit” files to the FPGA via the DIN and CCLK pins of the FPGA.

The second interface between the PC and the FPGA is used to transfer data between the two during system operation and is accomplished by connecting the necessary PC bus signals directly to I/O pins of the FPGA. The PC bus is used as the primary interface between the PC and FPGA. Typically, this interface is implemented by configuring 16-bit input and output ports within the FPGA.
2.4 The software

So far, only DOS and LINUX have been used as operating systems on MERCAL. Other operating systems could be used as long as they can operate in an embedded environment and do not require resources beyond those provided by the DIMM-PC. For the discussion to follow we will restrict the description to the DOS environment; however, the procedures for other operating systems will be similar.

Before the DIMM-PC is placed into the MERCAL module, a resident monitor program is loaded onto the flash drive of the DIMM-PC. At the same time an AUTOEXEC.BAT file that invokes the monitor program is loaded onto the flash drive of the DIMM-PC. The DIMM-PC can now be placed in the MERCAL module. When the MERCAL module is powered up or reset the AUTOEXEC.BAT file runs the monitor program which checks to determine if a host PC is connected to the serial port of the DIMM-PC. If there is not a connection to the serial port the monitor program exits. If there is a connection to the serial port, the monitor program enters a mode to allow files to be transferred to the DIMM-PC. Typically, several files will be uploaded to the DIMM-PC. These include a bit file to configure the FPGA, a program to transfer the bit file to the FPGA, the application program, and an AUTOEXEC.BAT file to invoke the various programs. To use MERCAL for a different application it is typically only necessary to upload a new bit file and application program. When the monitor program exits a program to configure the FPGA from the bit file is executed. After configuration of the FPGA the application program is executed.

The various files required by MERCAL have been written by electrical engineering students at Virginia Commonwealth University. The programs that run on the DIMM-PC have been written primarily in C++, and the bit files for the FPGA have been generated by using both VHDL and schematic capture.

Appendix C describes in detail the procedure for using DOS with the MERCAL module, and Appendix D describes the procedure for using Linux.

3 Example application

In order to test the concept and prototype implementation of the MERCAL system, an example application was developed using it. This application consisted of a dynamic spectrum analyzer display for audio frequencies. The functional block diagram is shown in Figure 4. The application uses an FFT algorithm to produce the frequency spectrum data of the sound information that has been amplified, filtered, digitized, and stored in a FIFO buffer. The spectrum output data produced by the FFT algorithm is displayed as a moving bar graph on a standard VGA display.

In this application, a prototype of the MERCAL was used that consisted of the DIMM-PC in its development board, connected to a separate board containing the Xilinx FPGA via ribbon cables. However, it should be noted that none of the interface capabilities of the DIMM-PC development board, including the VGA display adapter, was utilized in the performance of the application. All of the system functionality was contained in the DIMM-PC, the Xilinx FPGA, and a small signal pre-processing board, which contained the amplifier, filter, and Analog-to-Digital converter. Figure 5 shows the hardware block diagram of the MERCAL system in this application.
Once processed by the A-to-D converter, the sound samples are held in a digital FIFO module implemented in the FPGA. The FFT algorithm is executed on the DIMM-PC. When ready to process a new packet of samples, the DIMM-PC downloads the samples from the FIFO module on the FPGA. It then performs the FFT algorithm and transfers the spectrum data back to another hardware module on the FPGA. This module uses the spectrum data to generate the VGA display. All of the signals required by the VGA display are generated in this hardware module in the FPGA. Figure 6 is a photograph of this initial prototype, which was used to prove the MERCAL concept, in operation.

Figure 4. Prototype MERCAL application system functional block diagram.

Figure 5. Prototype MERCAL application system hardware.
4. Conclusion

The MERCAL module combines, in a very compact platform, the processing power, flexibility, and ease of programming of a PC platform, with a significant amount of high-speed digital logic for implementing interface functions to custom hardware or accelerating critical portions of an application. The hardware and software in a MERCAL module can be reprogrammed for a number of applications, even in-situ and during system operation. The MERCAL module provides researchers at NASA Langley Research Center and industry with a powerful new tool for implementing embedded systems that require processing power, flexibility, and reduced form factor coupled with ease of development.

5. Acknowledgements

The work reported here was supported by grant NAG-1-01042 from NASA Langley Research Center. Significant contributions were made to this work by several Virginia Commonwealth University electrical engineering students. They are: Austin Kim, Larry McDaniel, Matthew Sprinkle, David Staples, Andrew Gamble, Joshua Bell, Jason Blevins, Jonathan Andrews, Sean Laughter, Timothy Niemczyk, and Erick Donald.
Appendix A

MERCAL Module Schematic Diagrams
Appendix B

MERCAL module pin outs
## MERCAL pin outs

<table>
<thead>
<tr>
<th>Description</th>
<th>External Conn</th>
<th>Pin #</th>
<th>Xilinx Pin #</th>
<th>Dimm PC (J3) pin #</th>
<th>Schematic name</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS</td>
<td>J2</td>
<td>4</td>
<td>2</td>
<td></td>
<td>TMS</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>25</td>
<td>3</td>
<td></td>
<td>XIO3</td>
<td>PRTA&lt;0&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>27</td>
<td>4</td>
<td></td>
<td>XIO4</td>
<td>PRTA&lt;1&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>29</td>
<td>5</td>
<td></td>
<td>XIO5</td>
<td>PRTA&lt;2&gt;</td>
</tr>
<tr>
<td>IO, VREF</td>
<td>J2</td>
<td>31</td>
<td>6</td>
<td></td>
<td>XIO6</td>
<td>PRTA&lt;3&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>33</td>
<td>7</td>
<td></td>
<td>XIO7</td>
<td>PRTA&lt;4&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>35</td>
<td>8</td>
<td></td>
<td>XIO8</td>
<td>PRTA&lt;5&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>37</td>
<td>9</td>
<td></td>
<td>XIO9</td>
<td>PRTA&lt;6&gt;</td>
</tr>
<tr>
<td>IO, VREF</td>
<td>J2</td>
<td>39</td>
<td>10</td>
<td></td>
<td>XIO10</td>
<td>PRTA&lt;7&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>41</td>
<td>14</td>
<td></td>
<td>XIO14</td>
<td>PRTC&lt;0&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>43</td>
<td>15</td>
<td></td>
<td>XIO15</td>
<td>PRTC&lt;1&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>45</td>
<td>16</td>
<td></td>
<td>XIO16</td>
<td>PRTC&lt;2&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>47</td>
<td>17</td>
<td></td>
<td>XIO17</td>
<td>PRTC&lt;3&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>49</td>
<td>18</td>
<td></td>
<td>XIO18</td>
<td>PRTC&lt;4&gt;</td>
</tr>
<tr>
<td>IO, VREF</td>
<td>J2</td>
<td>51</td>
<td>20</td>
<td></td>
<td>XIO20</td>
<td>PRTC&lt;5&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>53</td>
<td>21</td>
<td></td>
<td>XIO21</td>
<td>PRTC&lt;6&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>55</td>
<td>22</td>
<td></td>
<td>XIO22</td>
<td>PRTC&lt;7&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>30</td>
<td>23</td>
<td></td>
<td>XIO23</td>
<td>PRTD&lt;0&gt;</td>
</tr>
<tr>
<td>IO, IRDY</td>
<td>J2</td>
<td>40</td>
<td>24</td>
<td></td>
<td>XIO24</td>
<td>PRTD&lt;1&gt;</td>
</tr>
<tr>
<td>IO, TRDY</td>
<td>J2</td>
<td>50</td>
<td>27</td>
<td></td>
<td>XIO27</td>
<td>PRTD&lt;2&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>69</td>
<td>29</td>
<td></td>
<td>XIO29</td>
<td>PRTD&lt;3&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>61</td>
<td>30</td>
<td></td>
<td>XIO30</td>
<td>PRTD&lt;4&gt;</td>
</tr>
<tr>
<td>IO, REF</td>
<td>J1</td>
<td>53</td>
<td>31</td>
<td></td>
<td>XIO31</td>
<td>PRTD&lt;5&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>33</td>
<td>33</td>
<td></td>
<td>XIO33</td>
<td>PRTD&lt;6&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>76</td>
<td>34</td>
<td></td>
<td>XIO34</td>
<td>PRTD&lt;7&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>74</td>
<td>35</td>
<td></td>
<td>XIO35</td>
<td>PRTD&lt;0&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>72</td>
<td>36</td>
<td></td>
<td>XIO36</td>
<td>PRTD&lt;1&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>70</td>
<td>37</td>
<td></td>
<td>XIO37</td>
<td>PRTD&lt;2&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>68</td>
<td>41</td>
<td></td>
<td>XIO41</td>
<td>PRTD&lt;3&gt;</td>
</tr>
<tr>
<td>IO, REF</td>
<td>J1</td>
<td>66</td>
<td>42</td>
<td></td>
<td>XIO42</td>
<td>PRTD&lt;4&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>64</td>
<td>43</td>
<td></td>
<td>XIO43</td>
<td>PRTD&lt;5&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>62</td>
<td>44</td>
<td></td>
<td>XIO44</td>
<td>PRTD&lt;6&gt;</td>
</tr>
<tr>
<td>IO, REF</td>
<td>J1</td>
<td>60</td>
<td>45</td>
<td></td>
<td>XIO45</td>
<td>PRTD&lt;7&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>58</td>
<td>46</td>
<td></td>
<td>XIO46</td>
<td>PRTF&lt;0&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>56</td>
<td>47</td>
<td></td>
<td>XIO47</td>
<td>PRTF&lt;1&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>54</td>
<td>48</td>
<td></td>
<td>XIO48</td>
<td>PRTF&lt;2&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>52</td>
<td>49</td>
<td></td>
<td>XIO49</td>
<td>PRTF&lt;3&gt;</td>
</tr>
<tr>
<td>MI</td>
<td></td>
<td>50</td>
<td></td>
<td></td>
<td>XM1</td>
<td></td>
</tr>
<tr>
<td>M0</td>
<td></td>
<td>52</td>
<td></td>
<td></td>
<td>XM0</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>46</td>
<td>57</td>
<td></td>
<td>XIO57</td>
<td>PRTF&lt;4&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>44</td>
<td>58</td>
<td></td>
<td>XIO58</td>
<td>PRTF&lt;5&gt;</td>
</tr>
<tr>
<td>IO, REF</td>
<td>J1</td>
<td>42</td>
<td>59</td>
<td></td>
<td>XIO59</td>
<td>PRTF&lt;6&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>40</td>
<td>60</td>
<td></td>
<td>XIO60</td>
<td>PRTF&lt;7&gt;</td>
</tr>
<tr>
<td>Description</td>
<td>Extern Conn</td>
<td>Pin #</td>
<td>Xilinx Pin #</td>
<td>Dimm PC (J3) pin #</td>
<td>Schematic name</td>
<td>Comments</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------</td>
<td>--------------</td>
<td>--------------------</td>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>38</td>
<td>61</td>
<td>XIO61</td>
<td></td>
<td>PRTG&lt;0&gt;</td>
</tr>
<tr>
<td>IO, REF</td>
<td>J1</td>
<td>36</td>
<td>62</td>
<td>XIO62</td>
<td></td>
<td>PRTG&lt;1&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>34</td>
<td>63</td>
<td>XIO63</td>
<td></td>
<td>PRTG&lt;2&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>32</td>
<td>67</td>
<td>XIO67</td>
<td></td>
<td>PRTG&lt;3&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>30</td>
<td>68</td>
<td>XIO68</td>
<td></td>
<td>PRTG&lt;4&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>28</td>
<td>69</td>
<td>XIO69</td>
<td></td>
<td>PRTG&lt;5&gt;</td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>70</td>
<td>77REF_N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO, REF</td>
<td>J4</td>
<td>17</td>
<td>73</td>
<td>143SA00</td>
<td></td>
<td>PRTG&lt;6&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>26</td>
<td>74</td>
<td>XIO74</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>18</td>
<td>75</td>
<td>139SA01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCK1, I</td>
<td></td>
<td>77</td>
<td></td>
<td>141OSC</td>
<td></td>
<td>14.3 MHz.</td>
</tr>
<tr>
<td>CLK 32 MHZ</td>
<td></td>
<td>80</td>
<td>XGCK0,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>19</td>
<td>81</td>
<td>137SA02</td>
<td></td>
<td>PRTG&lt;7&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>50</td>
<td>82</td>
<td>XIO82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>83</td>
<td>133SA03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO, REF</td>
<td>J4</td>
<td>24</td>
<td>84</td>
<td>131BALE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>16</td>
<td>85</td>
<td>129SD15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>87</td>
<td>127SA04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>88</td>
<td>125TC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>15</td>
<td>89</td>
<td>123SD14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>14</td>
<td>94</td>
<td>XIO94</td>
<td></td>
<td>PRTH&lt;0&gt;</td>
</tr>
<tr>
<td>IO, REF</td>
<td>J4</td>
<td>14</td>
<td>95</td>
<td>117SD13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>96</td>
<td>115SA06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>12</td>
<td>97</td>
<td>XIO97</td>
<td></td>
<td>PRTH&lt;1&gt;</td>
</tr>
<tr>
<td>IO, REF</td>
<td></td>
<td>98</td>
<td>111DRQ7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>99</td>
<td>109DACK7_N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>10</td>
<td>100</td>
<td>XIO100</td>
<td></td>
<td>PRTH&lt;2&gt;</td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>101</td>
<td>105SA07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>13</td>
<td>102</td>
<td>103SD12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DONE</td>
<td></td>
<td>104</td>
<td>126DONE&lt;&gt;PE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRGM</td>
<td></td>
<td>106</td>
<td>108PD2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO(/INIT)</td>
<td></td>
<td>107</td>
<td>102ERR_N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO, D7</td>
<td>J1</td>
<td>10</td>
<td>108</td>
<td>D7</td>
<td></td>
<td>PRTH&lt;3&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>35</td>
<td>109</td>
<td>XIO109</td>
<td></td>
<td>PRTH&lt;4&gt;</td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>110</td>
<td>99SA08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO, REF</td>
<td>J4</td>
<td>12</td>
<td>111</td>
<td>97SD11</td>
<td></td>
<td>PRTH&lt;5&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>37</td>
<td>112</td>
<td>XIO112</td>
<td></td>
<td>PRTH&lt;6&gt;</td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>113</td>
<td>93SA09</td>
<td></td>
<td></td>
<td>PRTH&lt;7&gt;</td>
</tr>
<tr>
<td>IO, D6</td>
<td>J1</td>
<td>12</td>
<td>115</td>
<td>D6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO, D5</td>
<td>J1</td>
<td>78</td>
<td>119</td>
<td>D5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Extern Conn</td>
<td>Pin #</td>
<td>Xilinx Pin #</td>
<td>Dimm PC (J3) pin #</td>
<td>Schematic name</td>
<td>Comments</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>10</td>
<td>122</td>
<td>65SD09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>9</td>
<td>127</td>
<td>79SD08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO,TRDY</td>
<td>J1</td>
<td>9</td>
<td>129</td>
<td>TRDY</td>
<td>PRTJ&lt;0&gt;</td>
<td></td>
</tr>
<tr>
<td>IO,IRDY</td>
<td>J1</td>
<td>11</td>
<td>132</td>
<td>IRDY</td>
<td>PRTJ&lt;2&gt;</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>39</td>
<td>133</td>
<td>XIO133</td>
<td>PRTJ&lt;3&gt;</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>134</td>
<td>71</td>
<td>SA13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO,D3</td>
<td>J1</td>
<td>31</td>
<td>135</td>
<td>D3</td>
<td>PRTJ&lt;4&gt;</td>
<td></td>
</tr>
<tr>
<td>IO,REF</td>
<td>J1</td>
<td>71</td>
<td>136</td>
<td>XIO136</td>
<td>PRTJ&lt;5&gt;</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>138</td>
<td>67</td>
<td>SA14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>73</td>
<td>139</td>
<td>XIO139</td>
<td>PRTJ&lt;6&gt;</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>140</td>
<td>63</td>
<td>SA15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>75</td>
<td>141</td>
<td>XIO141</td>
<td>PRTJ&lt;7&gt;</td>
<td></td>
</tr>
<tr>
<td>IO,D2</td>
<td>J1</td>
<td>77</td>
<td>142</td>
<td>D2</td>
<td>PRTK&lt;0&gt;</td>
<td></td>
</tr>
<tr>
<td>IO,D1</td>
<td>J1</td>
<td>79</td>
<td>146</td>
<td>D1</td>
<td>PRTK&lt;1&gt;</td>
<td></td>
</tr>
<tr>
<td>IO,REF</td>
<td>J2</td>
<td>8</td>
<td>147</td>
<td>XIO147</td>
<td>PRTK&lt;2&gt;</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>22</td>
<td>148</td>
<td>57IOR_N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>149</td>
<td>55</td>
<td>SA17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO,REF</td>
<td>J4</td>
<td>150</td>
<td>53</td>
<td>IRO15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>151</td>
<td>51</td>
<td>IOW_N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>152</td>
<td>49</td>
<td>SA18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>60</td>
<td>153</td>
<td>PD0</td>
<td>Programming</td>
<td></td>
</tr>
<tr>
<td>DOUT,BUSY</td>
<td>J2</td>
<td>1</td>
<td>154</td>
<td>DOUT,BUSY</td>
<td>Programming</td>
<td></td>
</tr>
<tr>
<td>CCLK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDO</td>
<td>J2</td>
<td>3</td>
<td>157</td>
<td>TD0</td>
<td>Boundry Scan</td>
<td></td>
</tr>
<tr>
<td>TDI</td>
<td>J2</td>
<td>71</td>
<td>159</td>
<td>TD1</td>
<td>Boundry Scan</td>
<td></td>
</tr>
<tr>
<td>IO/(CS)</td>
<td>J2</td>
<td>73</td>
<td>160</td>
<td>CS_N</td>
<td>PRTK&lt;3&gt;</td>
<td></td>
</tr>
<tr>
<td>IO/(WR)</td>
<td>J2</td>
<td>75</td>
<td>161</td>
<td>WR_N</td>
<td>PRTK&lt;4&gt;</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>56</td>
<td>162</td>
<td>XIO162</td>
<td>PRTK&lt;5&gt;</td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>163</td>
<td>45</td>
<td>SMEMR_N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO,REF</td>
<td>J2</td>
<td>164</td>
<td>43</td>
<td>SA19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>165</td>
<td>41</td>
<td>IRQ11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>166</td>
<td>39</td>
<td>SMEMW_N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO,REF</td>
<td>J2</td>
<td>167</td>
<td>37</td>
<td>AEN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>168</td>
<td>35</td>
<td>IRQ10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>Extern Conn</td>
<td>Pin #</td>
<td>Xilinx Pin #</td>
<td>Dimm PC (J3) pin #</td>
<td>Schematic name</td>
<td>Comments</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------</td>
<td>--------------</td>
<td>-------------------</td>
<td>---------------</td>
<td>----------</td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>27</td>
<td>172</td>
<td>33 ICHRDY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>173</td>
<td>31 SBHE N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>1</td>
<td>174</td>
<td>28 SD00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>25</td>
<td>175</td>
<td>27 OSC16 N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>176</td>
<td>250WS N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO,REF</td>
<td>J4</td>
<td>2</td>
<td>178</td>
<td>23 SD01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td></td>
<td>179</td>
<td>21 MEMCS16 N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>3</td>
<td>180</td>
<td>19 SD02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>4</td>
<td>181</td>
<td>17 SD03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCK2,1</td>
<td>J2</td>
<td>5</td>
<td>182</td>
<td>XGCK2,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCK3,1</td>
<td>J2</td>
<td>7</td>
<td>185</td>
<td>XGCK3,1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>54</td>
<td>187</td>
<td>XIO187</td>
<td></td>
<td>PRTK&lt;6&gt;</td>
</tr>
<tr>
<td>IO,REF</td>
<td>J4</td>
<td>5</td>
<td>188</td>
<td>13 SD04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>52</td>
<td>191</td>
<td>XIO191</td>
<td></td>
<td>PRTK&lt;7&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>7</td>
<td>192</td>
<td>7 SD06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>8</td>
<td>193</td>
<td>5 SD07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J4</td>
<td>23</td>
<td>194</td>
<td>3 RSTDRV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO</td>
<td>J1</td>
<td>48</td>
<td>195</td>
<td>XIO195</td>
<td></td>
<td>TEST</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>9</td>
<td>199</td>
<td>XIO199</td>
<td></td>
<td>PRTB&lt;0&gt;</td>
</tr>
<tr>
<td>IO,REF</td>
<td>J2</td>
<td>11</td>
<td>200</td>
<td>XIO200</td>
<td></td>
<td>PRTB&lt;1&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>13</td>
<td>201</td>
<td>XIO201</td>
<td></td>
<td>PRTB&lt;2&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>15</td>
<td>202</td>
<td>XIO202</td>
<td></td>
<td>PRTB&lt;3&gt;</td>
</tr>
<tr>
<td>IO,REF</td>
<td>J2</td>
<td>17</td>
<td>203</td>
<td>XIO203</td>
<td></td>
<td>PRTB&lt;4&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>19</td>
<td>204</td>
<td>XIO204</td>
<td></td>
<td>PRTB&lt;5&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>21</td>
<td>205</td>
<td>XIO205</td>
<td></td>
<td>PRTB&lt;6&gt;</td>
</tr>
<tr>
<td>IO</td>
<td>J2</td>
<td>23</td>
<td>206</td>
<td>XIO206</td>
<td></td>
<td>PRTB&lt;7&gt;</td>
</tr>
<tr>
<td>TCK</td>
<td>J2</td>
<td>2</td>
<td>207</td>
<td>TCK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IRQ14</td>
<td>J4</td>
<td>20</td>
<td>59 IRQ14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J2</td>
<td></td>
<td>72</td>
<td>135 MASTER N</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C

How to use MERCAL with DOS
1. Items Needed
2. Installing Microsoft DOS 6.22 on DIMM-PC
3. Source code for app.bat on MERCAL Install Floppy
4. Setting up DOS on DIMM-PC
5. Launching the Command Prompt from the MERCAL
6. Sending BIT Files
7. Running application programs
8. The DOS Mercal Configuration Files
9. FPGA Configure
10. Source code for autoexec.bat MERCAL Install Floppy
11. Source code for Mercal.bat MERCAL Install Floppy
12. Source code for app.bat on MERCAL Install Floppy

Preliminary: October 16th, 2002
Part 1 - Items Needed

Hardware Items Needed:
- DIMM-PC
- Floppy Drive
- JUMPtec Development Board
- Serial Cable
- MERCAL Board
- MERCAL Development Board
- Serial Cable (Straight through not crossover)

Software Items Needed:
- MERCAL Install Floppy
- Microsoft DOS 6.22 Floppy Disk Installation
- MERCAL Link Program

How to create media needed for installation:

1. Format one blank floppy diskette.
2. Copy all the files from the MERCAL install floppy directory to the floppy diskette.
Part 2 – Installing Microsoft DOS 6.22 on DIMM-PC

1. Insert the DIMM-PC into the JUMPtec development board and connect the floppy drive.
2. Set up the BIOS to boot from the floppy drive.
3. Insert disk one of Microsoft DOS 6.22 and boot up the computer.
4. Press F3 twice to exit out of the boot menu.
5. Type fdisk.
6. Select option 4 to display partition status.
7. Delete any partitions that may be on the DIMM-PC using option 3.
8. Next create a primary partition by going into option 1, Create new partition.
9. Select one again for Primary DOS partition.
10. Enter Y to confirm.
11. Allow computer to reboot back into the DOS setup.
12. Press enter to continue installation.
13. Select format drive and allow drive to format.
14. Press enter to confirm settings.
15. Press enter to confirm install path, it should be C:\DOS.
16. Follow the rest of the directions of the DOS installation.
17. When the computer restarts change the boot device from floppy disk to hard drive and reboot into DOS installation.
18. The DIMM-PC should boot up to a DOS prompt

Part 3 – Setting up DOS on DIMM-PC

1. Insert the Mercal install floppy
2. Type copy a:\*.* c:\
3. Press Yes to overwrite any files

Part 4 – Launching the Command Prompt from the MERCAL

1. Now take the DIMM-PC and put it into the MERCAL and the MERCAL into the MERCAL development board.
2. Connect a serial cable between the MERCAL development board and your computer.
3. On your computer run the MERCAL.EXE file that allows PC access into the MERCAL.
4. Power up the MERCAL and click on CONNECT on the MERCAL program.
5. You may have you press CONNECT several times if it times out before the link is established.
6. Press LINK to bring up the DOS prompt on the MERCAL.

Part 5 – Sending BIT Files

1. Follow steps 1 to 5 from part 4.
2. Press SEND FILE and send your file named XI.BIT.
Part 6 – Running application programs

1. Follow steps 1 to 5 from part 4.
2. Press SEND FILE and send over your executable file as app.exe.
3. Reboot the MERCAL without the serial cable connected to allow the program to run.

Part 8 – The DOS Mercal Configuration Files

This is the files needed to use the MERCAL in DOS.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config-l.exe</td>
<td>Configures the FPGA</td>
</tr>
<tr>
<td>merc.exe</td>
<td>Check link status to run program or wait for link</td>
</tr>
<tr>
<td>xi.bit</td>
<td>Auto configuration bit file</td>
</tr>
<tr>
<td>app.bat</td>
<td>Application that will be run</td>
</tr>
<tr>
<td>autoexec.bat</td>
<td>DOS auto start script</td>
</tr>
<tr>
<td>mercal.bat</td>
<td>Batch file to check link status to run program or wait for link</td>
</tr>
</tbody>
</table>

Part 9 – FPGA Configure

This is the program that will download the bit file for the FPGA using the parallel port.

```c
/*
   Author: Jason M. Blevins
   FileName: config-l.cpp
   Date:     August 10, 2001 (most recent revision)
   Desc:     Downloads binary file to an FPGA.
*/

#include<iostream.h>
#include<ctype.h>
#include<stdlib.h>
#include<fstream.h>
#include<stdio.h>
#include<conio.h>

void ProcessArgs(int argc, char argv1, ifstream& DataIn);
void ClearFPGA();
void TransferBits(ifstream& DataIn);
int ExtractBit(int WhichBit, int PortAddress);
```

```
void ProcessArgs(int argc, char *argv1, ifstream &DataIn);
// Processes command line arguments & attempts to open bit file

void ClearFPGA(); // Clears the FPGA
void TransferBits(ifstream &DataIn);
// reads bits from bit file and transfers them, bit by bit, to FPGA
int ExtractBit(int WhichBit, int PortAddress);
// Extracts the bit number 'WhichBit' from the 'PortAddress'
```

```c
/*
   pin 2  a0   DIN
   pin 3  a1   CCLK
   pin 4  a2   'PROG
   pin 5  a3   ---
   pin 6  a4   ---
   *(inverted)*
   pin 7  a5   ---
   pin 8  a6   ---
   *(inverted)*
   pin 9  a7   ---
   *(inverted)*
   pin15 b3   !INIT
   pin13 b4   ---
   pin12 b5   DONE
   pin11 b7   ---
   pin10 b6   ---
   pin 1   c0   ---
   pin 14 c1   ---
   pin 16 c2   ---
```
// *(inverted)*

#define PortA 0x378;  // address of PortA
#define PortB 0x379;  // address of PortB
#define TRUE 0x01    // Boolean flag
#define FALSE 0x00    // Boolean flag
#define MSB 0x07      // MSB of a byte
#define LSB 0x00      // LSB of a byte
#define PROG_LOW 0x00 // Value of PortA that pulls PROG low
#define PROG_HIGH 0x04 // Value of PortA that pulls PROG high
#define CCLK_LOW 0x00 // Value of PortA that pulls CCLK low
#define CCLK_HIGH 0x02 // Value of PortA that pulls CCLK high
#define INIT_BIT 0x03 // Bit number of INIT in PortB
#define DONE_BIT 0x05 // Bit number of DONE in PortB
#define BYTE_SIZE 0x08 // # of array slots needed to hold a byte
#define ONE_ARG 0x01  // Specifies that one argument was supplied
#define TWO_ARG 0x02  // Specifies that two arguments were supplied

void main(int argc, char *argv[]) {
    ifstream DataIn; // declares file stream variable
    ProcessArgs(argc, argv[0], DataIn); // processes users command line args
    ClearFPGA(); // clears FPGA
    TransferBits(DataIn); // transfers bit file to FPGA
}

void TransferBits(ifstream& DataIn) {
    int bits[BYTE_SIZE];
    int Init = TRUE;
    unsigned char CharByte;
    unsigned int IntByte;
    int Done;

    while ((Init == TRUE) && (!DataIn.eof())) {
        DataIn.get(CharByte); // gets a byte from bit file as type char
        IntByte = CharByte; // converts byte from type char to int
        for (int a=LSB; a <= MSB; a++) // extracts the 8 bits from IntByte and stores them in array bits[]
            bits[a] = IntByte % 2;
        IntByte = IntByte / 2;
        for (int b=MSB; b>=LSB; b--) // puts bit on DIN and then clocks it into FPGA
            outp(PortA, (PROG_HIGH + CCLK_LOW + bits[b])); // equivalent to '0000010X' where X represents bits[a] -- clock low
            outp(PortA, (PROG_HIGH + CCLK_HIGH + bits[b])); // equivalent to '0000011X' where X represents bits[a] -- clock high
        Init = ExtractBit(INIT_BIT, PortB); // extracts PIN
    }
    DataIn.close(); // detaches file stream variable from external file
    Done = ExtractBit(DONE_BIT, PortB); // extracts DONE
    if (Done == TRUE) {
        cout << "Programming successful!\n";
    } else {
        cout << "Error. Program was not successful.\n";
    }
}
void ClearFPGA()
{
    int Init;
    outp(PortA, PROG_LOW); // pulls !PROG low
    outp(PortA, PROG_HIGH); // '12' is equivalent to '00000100' -- pulls !PROG high
    Init = ExtractBit(INIT_BIT, PortB); // extracts !INIT
    while (Init == FALSE) // waits for !INIT to signal that the FPGA is clear
    {
        Init = ExtractBit(INIT_BIT, PortB); // extracts !INIT
    }
}

void ProcessArgs(int argc, char *argv1, ifstream& DataIn)
{
    if (argc == ONE_ARG) // if user doesn't specify which bit file to read from
    {
        cout << "\nError. No file name specified.\n";
        abort();
    }
    else if (argc == TWO_ARG) // if user specifies the bit file
    {
        DataIn.open(argv1, ios::nocreate | ios::binary); // opens bit file as specified by user -- if non-existant, no new file is created
        if (!DataIn.fail())
        {
            cout << "\nError. Invalid path or file name.\n";
            abort();
        }
        else // if user specifies too many arguments...
        {
            cout << "\n\nProgramming Spartan II FPGA #1...\n";
        }
    }
    else // if user specifies too many arguments...
    {
        cout << "\nError. Invalid arguments.\n";
        abort();
    }
}

int ExtractBit(int WhichBit, int PortAddress)
{
    unsigned int DataByte;
    DataByte = inp(PortAddress);
    return ((DataByte / int(pow(2,WhichBit))) & 2);
}

Part 10 - Source code for autoexec.bat MERCAL Install Floppy

This file is the startup file for DOS.

1: @ECHO OFF
2: PROMPT $p$g
3: PATH C:\DOS;C:\MERCAL
4: SET TEMP=C:\DOS
5: mercal.BAT

Part 11 - Source code for Mercal.bat MERCAL Install Floppy

1: @ECHO OFF
2: CTTY CON:
3: :START
4: MERC.EXE
5: IF ERRORLEVEL 3 GOTO ERROR
6: IF ERRORLEVEL 2 GOTO LINK
7: IF ERRORLEVEL 1 GOTO RUN
8: :LINK
9: CTTY COM1:
10: GOTO END
11: :RUN
12: APP.BAT
13: GOTO END
14: :ERROR
15: GOTO END
16: :END

Part 12 - Source code for app.bat on MERCAL Install Floppy

1: config~1.EXE C:\XI.bit
2: rem "EXE file goes here"
3: app.exe
Appendix D

How to use MERCAL with Linux
MERCAL Development Handbook
for DIMM-PCs running Whitedwarf Linux

Written by: Erik Donald

1. Items Needed
2. Installing Whitedwarf Linux on the DIMM-PC
3. Setting up Whitedwarf Linux on the DIMM-PC
4. Setting up Vmware
5. Installing RedHat Linux 7.3 on Vmware
6. Logging into Linux in VMware
7. Logging into the MERCAL via a serial connection
8. Sending files to the MERCAL
9. Receiving files from the MERCAL
10. Shutting down the MERCAL
11. Development using gcc
12. What does the MERCAL Install Floppy Do?
13. Source code for install.sh on MERCAL Install Floppy
14. Source code for FPGAConfigure on MERCAL Install Floppy
15. Source code for /etc/inittab on MERCAL Install Floppy
16. Source code for /etc/rc.d/rc.local on MERCAL Install Floppy
17. Source code for /etc/rc.d/rc.serial on MERCAL Install Floppy
18. Source code for /etc/lilo.conf on MERCAL Install Floppy

Preliminary: August 1st, 2002
Revised: October 3rd, 2002
Part 1 - Items Needed

Hardware Items Needed:
- DIMM-PC
- Floppy Drive
- CD-ROM (Not all CD-ROMs work)
- JUMPtec Development Board
- Serial Cable
- MERCAL Board
- MERCAL Development Board
- Serial Cable (Straight through not crossover)

Software Items Needed:
- RedHat Linux 7.3 (www.redhat.com)
- Whitedwarf Linux CD-ROM (www.whitedwarflinux.org)
- Whitedwarf Linux Boot Disk (www.whitedwarflinux.org)
- MERCAL Install Floppy
- RawWrite to create Installation Floppies

Optional Software Items:
- VMWare (www.vmware.com)

VMWare allows you to run Linux inside a Windows computer. It makes development easier if you need to use both Linux and Windows applications at the same time. When you run VMWare it simulates a full Linux computer inside a window in Windows. So you can either install Linux on your PC inside Windows, or dedicate your PC entirely to Linux. If you choose to dedicate your computer to Linux, you can skip steps 5 and 6 and just install Linux the normal way by putting the install Red Hat Linux CDROM in your computer and booting from the CDROM.

How to create media needed for installation:

1. To create the Whitedwarf CD-ROM, use the ISO image and burn it to a CD using the CD Burning software of your choice.
2. Launch rawwritewin.exe and select your floppy drive
3. Insert a blank floppy diskette
4. Select wdboot.img and click write. This will create the WhiteDwarf Linux Boot Disk.
5. Select Mercal Linux Install.img and click write. This will create the MERCAL install floppy to allow the DIMM-PC to run Linux while in the MERCAL module.
Part 2 – Installing Whitedwarf Linux on DIMM-PC

1. Connect a DIMM-PC into the JUMPtec development board. Connect a keyboard, monitor, floppy and a CD-ROM.
2. Go into the DIMM-PC Bios and set it to boot from the floppy drive and tell it there is a CD-ROM present by selecting auto as the slave drive.
3. Insert Whitedwarf Boot Disk Floppy and CD-ROM.
4. Boot the machine using this boot disk.
5. Press <enter> when it says, “Please insert wd 1.11 CD-ROM or root disk”.
6. Select OK when the installer comes up.
7. Select OK to the install device as /dev/hdai
8. Select CREATE
9. Select NEW
10. Select PRIMARY
11. Enter drive size (30MB)
12. Select BOOTABLE
13. Select WRITE
14. Type YES and press enter
15. Select QUIT
16. Select CLEAN
17. Select YES when it tells you formatting will erase all data
18. Select CD-ROM and wait for the base installation to occur
19. Follow the table below and only install the packages listed

<table>
<thead>
<tr>
<th></th>
<th>bin</th>
<th>diff</th>
<th>Yes</th>
<th>Ppp</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elvis</td>
<td>Yes</td>
<td>gcc_dev</td>
<td>No</td>
<td>syslogd</td>
<td>Yes</td>
</tr>
<tr>
<td>find</td>
<td>Yes</td>
<td>getty</td>
<td>Yes</td>
<td>telnet</td>
<td>No</td>
</tr>
<tr>
<td>gzip</td>
<td>Yes</td>
<td>gpm</td>
<td>No</td>
<td>vim</td>
<td>No</td>
</tr>
<tr>
<td>sh_utils</td>
<td>Yes</td>
<td>Grep</td>
<td>Yes</td>
<td>wget</td>
<td>No</td>
</tr>
<tr>
<td>tar</td>
<td>Yes</td>
<td>kbd</td>
<td>No</td>
<td>bind</td>
<td>Yes</td>
</tr>
<tr>
<td>txtutils</td>
<td>Yes</td>
<td>less</td>
<td>Yes</td>
<td>glibc2.1</td>
<td>No</td>
</tr>
<tr>
<td>util</td>
<td>Yes</td>
<td>lynx</td>
<td>No</td>
<td>glibc2.2</td>
<td>No</td>
</tr>
<tr>
<td>apache</td>
<td>No</td>
<td>minicom</td>
<td>Yes</td>
<td>kernel_source</td>
<td>No</td>
</tr>
<tr>
<td>ash_sh</td>
<td>No</td>
<td>perl</td>
<td>No</td>
<td>ncurses</td>
<td>No</td>
</tr>
<tr>
<td>bzip</td>
<td>yes</td>
<td>pkgtool</td>
<td>No</td>
<td>perl_libs</td>
<td>No</td>
</tr>
</tbody>
</table>

20. Select OK when it asks for network configuration
21. Select CANCEL when it asks for the hostname
22. Type in the root password and do not forget it!
23. Select OK
24. Remove the disk from the disk drive
25. The DIMM-PC should now restart and you should have Whitedwarf on the DIMM-PC
Part 3 - Setting up Whitedwarf Linux on DIMM-PC

1. Reboot the DIMM-PC
2. Login using the username root and your password
3. Type the following to install the MERCAL files (yes that is: period space
   install.sh)
   
   mount /dev/fd0 /mnt
   
   cd /mnt
   
   . install.sh
   
   umount /dev/fd0

4. Now take the DIMM-PC and place it in the MERCAL board and the MERCAL in
   the development board.
5. Connect the serial cable between the two. The communications protocol is
   115200,8,N,1 on the MERCAL.

Part 4 - Setting up Vmware

1. Install VMware to your computer
2. Launch Vmware
3. Click on File → New → New Virtual Machine
4. Select Typical and click on next
5. Select Linux and click on next
6. Review the directories and click on next
7. Select Use Host Only Networking and click on next
8. Click on your virtual machine in the left hand box
9. Right click on it and select settings
10. Click add
11. Click Serial Port
12. Make sure to select connect at power on
13. Select which serial port you wish to use
14. Click Finish
**Part 5 - Installing RedHat Linux 7.3 in VMWare**

1. Launch VMware
2. Insert the RedHat 7.3 Disc 1 into the CD-ROM
3. Select Linux in the left hand box and press Power On
4. After the CD gets booted up and there is a text screen in Linux to install, type `text` and press enter.
5. During the install select autopartition and install everything.
6. Reboot the machine and start up linux
7. Type in your username and password
8. Then type in these commands:
   ```
   mount -t iso9660 /dev/cdrom /mnt
   cp /mnt/vmware-linux-tools.tar.gz /tmp
   umount /dev/cdrom
   cd /tmp
   tar zxf vmware-linux-tools.tar.gz
   cd vmware-linux-tools
   ./install.pl
   ```

**Part 6 - Running RedHat Linux 7.3 in VMWare**

1. Launch VMWare
2. Press Power On
3. At the login prompt enter root and your password
4. At the command prompt type:
   ```
   startx
   ```
5. Press OK on the box warning you about logging in as root
**Part 7 - Logging into the MERCAL**

1. In RedHat Linux launch a terminal window in X (the black monitor icon on the bar at the bottom) and type:
   `xminicom &`
2. Press `<alt>+<o>` for options
3. Select Serial Port Setup
4. Press `<a>`
5. Change device to `/dev/ttyS0`
6. Press `<enter>`
7. Press `<e>`
8. Press `<e>` then `<q>`
9. The terminal should now be setup for 115200 8N1
10. Press `<enter>` twice
11. Select Save setup as defl
12. Press Enter
13. You should now see a terminal window of the MERCAL
14. Press enter to bring up a login prompt
15. At the login type
   ```
   mercal
   su
   password (or the password you have used for root)
   ```
16. You are now logged in as a super user and can use the MERCAL for development

**Part 8 - Sending files to MERCAL**

1. Launch xminicom and log into the MERCAL
2. Press `<alt>`,`<s>`
3. Select zmodem
4. Select the file to send and select Okay

**Part 9 – Receiving files from MERCAL**

1. Launch xminicom and log into the MERCAL
2. At the MERCAL command line type:
   ```
   lsz filename
   ```
   This file will be saved to /root

**Part 10 – Shutting Down the MERCAL**

1. Launch xminicom and log into the MERCAL
2. At the MERCAL command line type (to reboot replace `-h` with `-r`):
   ```
   shutdown -h now
   ```
3. Wait until the device has shutdown (approximately 30 seconds).
**Part 11 – Development using gcc**

1. To compile your C or C++ source code, first start a terminal window
2. Type:
   
   gcc source.cpp -o outputfile -static
3. The static option links the libraries with the file so that it will run correctly on the Linux installation on the DIMM-PC
4. Send your file to the MERCAL
5. To launch at run time edit your /etc/rc.d/rc.local file accordingly
6. You can use Kdevelop to develop your source code with a debugger. It resembles Visual Studio and is available at www.kdevelop.org.

**Part 12 – What does the MERCAL Install Floppy Do?**

The MERCAL Install Floppy simplifies the installation of Whitedwarf Linux on the DIMM-PC. You can look at the file install.sh to see exactly the commands it executes.

The first thing it does is create the MERCAL directory structure on the DIMM-PC. This structure looks like this:

```
/mercal/bin
/mercal/bit
/mercal/etc
/mercal/ftp
/mercal/http
```

There are currently only two files that are in this structure:

```
/mercal/bin/fpgaconfig  configures the FPGA
/mercal/bit/autoprogam.bit  the automatically programmed bit file
```

After this is created it copies over the initiation files to start up the serial port for a terminal session. After this it does the one thing that is necessary for the DIMM-PC to boot on the MERCAL board. You see, the MERCAL board is not ISA-PNP so the commands must be issued to disable the PNP ISA bus (as well as other things that could cause a problem like PCI routines). To do this /etc/lilo.conf is edited to include these statements then lilo is run to update the master boot record. Doing this allows the DIMM-PC not hang on boot up in the MERCAL board.
Part 13 – Source code for install.sh on MERCAL Install Floppy

This is the file install.sh script file that runs the MERCAL installation on the DIMM-PC.

1:  cp mercal.tar.gz /
2:  cd /
3:  gzip -d mercal.tar.gz
4:  tar -xf mercal.tar
5:  rm mercal.tar
6:  cd /mnt
7:  cp files/rc.local /etc/rc.d/rc.local
8:  cp files/rc.0 /etc/rc.d/rc.0
9:  cp files/rc.serial /etc/rc.d/rc.serial
10: cp files/inittab /etc/inittab
11: cp files/lilo.conf /etc/lilo.conf
12: cd /
13: umount /dev/fd0
14: mkdir /mnt/floppy
15: mkdir /mnt/cdrom
16: mkdir /mnt/flashdisk
17: /sbin/lilo
18: useradd mercal -p ''

Part 14 – FPGA Configure

This is the modified source code for the FPGA configuration using Linux. The only few changes made. First, a few lines are added to the header file. outb is changed to outb and inp is changed to inb. Also the order of the parameters between outb and outp are reversed. Also the ioperm command give the program permission to read a port and write to a port.

```c
/*
   Author: Jason M. Blevins
   Modified for Linux: Erik Donald
   FileName: FPGAconfigure.cpp
   Desc: Downloads binary file to an FPGA.
*/

#include<iostream.h>
#include<ctype.h>
#include<stdlib.h>
#include <sys/io.h>
#include <unistd.h>
#include<fstream.h>
#include<stdio.h>
#include<conio.h>

void ProcessArgs(int argc, char *argv1, ifstream& DataIn);  // Processes command line arguments & attempts to open bit file
void ClearFPGA();  // Clears the FPGA
void TransferBits(ifstream& DataIn);  //
int ExtractBit(int WhichBit, int PortAddress);
// Extracts the bit number 'WhichBit' from the 'PortAddress'

*/
// addresses of PortA
// addresses of PortB
#define TRUE 0x01 // Boolean flag
#define FALSE 0x00 // Boolean flag
#define MSB 0x07 // MSB of a byte
#define LSB 0x00 // LSB of a byte
#define PROG_LOW 0x00 // Value of PortA that pulls PROG low
#define PROG_HIGH 0x04 // Value of PortA that pulls PROG high
#define CCLK_LOW 0x00 // Value of PortA that pulls CCLK low
#define CCLK_HIGH 0x02 // Value of PortA that pulls CCLK high
#define INIT_BIT 0x03 // Bit number of INIT in PortB
#define DONE_BIT 0x05 // Bit Number of DONE in PortB
#define BYTE_SIZE 0x08 // # of array slots needed to hold a byte
#define ONE_ARG 0x01 // Specifies that one argument was supplied
#define TWO_ARG 0x02 // Specifies that two arguments were supplied

void main(int argc, char *argv[]) 
{
    if (ioctl(0x378, 3) == -1):
    
        perror("ioctl");
        exit(1);
    
    ifstream DataIn; // declares file stream variable

    ProcessArgs(argc, argv[1], DataIn); // processes users command line args
    ClearFPGA(); // clears FPGA
    TransferBits(DataIn); // transfers bit file to FPGA
}

void TransferBits(ifstream& DataIn) 
{
    int bits[BYTE_SIZE];
    int Init = TRUE;
    unsigned char CharByte;
    unsigned int IntByte;
    int Done;

    while ((Init == TRUE) && (!DataIn.eof()))
    {
        DataIn.get(CharByte); // gets a byte from bit file as type char
        IntByte = CharByte; // converts byte from type char to int
        for (int a=LSB; a <= MSB; a++)
        // extracts the 8 bits from IntByte and stores them in array bits[]
        {
            bits[a] = IntByte % 2;
            IntByte = IntByte / 2;
        }
FPGA

for (int b=MSB; b>=LSB; b--) // puts bit on DIN and then clocks it into
{
    outb((PROG_HIGH + CCLK_LOW + bits[b]), PortA);
    // equivalent to '0000010X' where X represents bits[a] -- clock low
    outb((PROG_HIGH + CCLK_HIGH + bits[b]), PortA);
    // equivalent to '0000011X' where X represents bits[a] -- clock high
}

Init = ExtractBit(INIT_BIT, PortB); // extracts !INIT
DataIn.close(); // detaches file stream variable from external file
Done = ExtractBit(DONE_BIT, PortB); // extracts DONE
if (Done == TRUE)
{
    cout << "Programming successful!\n";
}
else
{
    cout << "Error. Program was not successful.\n";
}

void ClearFPGA()
{
    int Init;
    outp(PROG_LOW, PortA); // pulls !PROG low
    outp(PROG_HIGH, PortA); // '1' is equivalent to '00000100' -- pulls !PROG high
    Init = ExtractBit(INIT_BIT, PortB); // extracts !INIT
    while (Init == FALSE) // waits for !INIT to signal that the FPGA is clear
    {
        Init = ExtractBit(INIT_BIT, PortB); // extracts !INIT
    }
}

void ProcessArgs(int argc, char *argv[], ifstream& DataIn)
{
    if (argc == ONE_ARG) // if user doesn't specify which bit file to read from
    {
        cout << "\nError. No file name specified.\n";
        abort();
    }
    else if (argc == TWO_ARG) // is user specifies the bit file
    {
        DataIn.open(argvL, ios::nocreate | ios::binary);
        // opens bit file as specified by user -- if non-existant, no new file is
        created
        if (DataIn.fail())
        {
            cout << "\nError. Invalid path or file name.\n";
            abort();
        }
        cout << "\nProgramming Spartan II FPGA #1...\n";
    }
    else // if user specifies too many arguments...
    {
        cout << "\n\nError. Invalid arguments.\n";
        abort();
    }
}

int ExtractBit(int WhichBit, int PortAddress)
{
    unsigned int DataByte;
    DataByte = inb(PortAddress);
    return ((DataByte / int(pow(2,WhichBit))) & 2);
Part 15 - Source code for /etc/initab on MERCAL Install Floppy

Only the changed lines are shown from the original installation to the new installation for the MERCAL. This sets up the serial transfers between the DIMM-PC and the computer.

68:  # Serial lines
69:  sl:12345:respawn:/sbin/agetty 115200 ttyS0 vt100
70:  s2:12345:respawn:/sbin/agetty 115200 ttyS1 vt100

Part 16 - Source code for /etc/rc.d/rc.local on MERCAL Install Floppy

This file is like the autoexec.bat in DOS. This runs the programs at startup. The only difference is that the ampersand lets the program run in the background.

1:  #!/bin/sh
2:  
3:  # /etc/rc.d/rc.local: Local system init. script.
4:  
5:  # Put any local setup commands in here:
6:  
7:  # Start up Serial Communications
8:  agetty std.115200 ttys0 vt100 &
9:  
10: # Automatically Program FPGA
11: /mercal/bin/fpgaconfig /mercal/bit/autoprogram.bit

Part 17 - Source code for /etc/rc.d/rc.serial on MERCAL Install Floppy

Only the changes from the original file are shown. Basically the serial ports are set from auto-configured to manually configured by commenting out lines 45 and 46 and uncomment lines 152 and 153.

45:  #${SETSERIAL} /dev/ttyS0 ${AUTO_IRQ} skip_test
     autoconfig ${STD_FLAGS}
46:  #${SETSERIAL} /dev/ttyS1 ${AUTO_IRQ} skip_test
     autoconfig ${STD_FLAGS}

152: ${SETSERIAL} /dev/ttyS0 uart 16450 port 0x3F8 irq 4
     ${STD_FLAGS}
153: ${SETSERIAL} /dev/ttyS1 uart 16450 port 0x2F8 irq 3
     ${STD_FLAGS}
The lilo.conf file needs to be changed then the command lilo must be executed to rewrite the boot sector of the hard drive. This is done because the MERCAL is not ISA compilient at boot time and therefore hangs the system. In order to correct this the ISA PnP bus is turned off. The PCI bus is turned off for safety since there is not PCI bus on the DIMM-PC.

```
1: # LILO configuration file
2: # generated by 'liloconfig'
3: #
4: # Start LILO global section
5: boot = /dev/hda
6: #compact # faster, but won't work on all systems.
7: delay = 5
8: vga = normal    # force sane state
9: # ramdisk = 0    # paranoia setting
10: # End LILO global section
11: # Linux bootable partition config begins
12: image = /vmlinuz
13:   root = /dev/hdal
14:   label = linux
15: append = "noisapnp floppy=none pci=off hdb=none"
16:   read-only
17: # Linux bootable partition config ends
```
Appendix E

Examples using schematic capture
Schematic Design Xilinx Foundation Series 3.1i

Written by: Timothy P. Niemczyk
Creating new designs can be entered either as a schematic or as a text based entry using VHDL or Verilog. Figure 1 is an example of a top-level schematic design. This design implements input & output ports, input & output pads, multiplexer, clock signal, D-Latch Flip Flops, and combinational logic.

Implementation of Ports is as follows:
Input Port A – 8 Bit enable IOR/Address 300
Input Port B – 4 Bit enable IOR/Address 302
Output LED’s – 8 Bit enable IOW/Address 302
Output Port E – 4 Bit enable IOW/Address 303

Figure 1. Xilinx Foundation Schematic Editor – implementation of 8 bit input and output ports.
This design also implemented hierarchical macros. Hierarchical macros are previously designed circuit that is isolated from the rest of the design and is stored in the library as its own symbol. Macro symbols are placed within the design and can be used in multiple instances. The actual macro schematic is connected to the top-level or upper hierarchy design via the terminals. These terminals correspond to a pin within the macro symbol.

Figure 2 demonstrates an example of an implemented hierarchical macro. This macro selects the address in which data will flow in or out.

![Figure 2. Address Multiplexer](image)
Figure 3 demonstrates a hierarchical macro that controls the data flow. By using the tri-state buffers data can be controlled reliably.

By the using hierarchical macros, designs can be broken up into parts. This will allow a faster design turnaround and a simper approach to designing complex circuits.