FPGA-based Experiment Platform for Hardware-Software Codesign and Hardware Emulation

Yajuvendra Nagaonkar

Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Electrical and Computer Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/420

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
FPGA-BASED EXPERIMENT PLATFORM FOR
HARDWARE-SOFTWARE CODESIGN AND
HARDWARE EMULATION

by

Yajuvendra Nagaonkar

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering
Brigham Young University
August 2006
This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date Mark L. Manwaring, Chair

Date James K. Archibald

Date Doran K. Wilde
As chair of the candidate’s graduate committee, I have read the thesis of Yajuendra Nagaonkar in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Mark L. Manwaring
Chair, Graduate Committee

Accepted for the Department

Michael A. Jensen
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of Engineering and Technology
ABSTRACT

FPGA-BASED EXPERIMENT PLATFORM FOR HARDWARE-SOFTWARE CODESIGN AND HARDWARE EMULATION

Yajuvendra Nagaonkar

Department of Electrical and Computer Engineering

Master of Science

An FPGA-based experiment platform for hardware-software codesign experiments was developed. The proposed platform would be used by an engineer who can be affiliated with academia, research or industry for codesign experiments or hardware emulation. The platform utilizes a combination of a microcontroller and a FPGA device to enable sufficient flexibility in exploring the design space to implement codesign experiments. The FPGA device operation is integrated with that of the microcontroller to provide an overall embedded solution for codesign experimentations. It is anticipated that the platform will be used in academia for educating the students the concepts of computer architecture and microprocessor design. Future work suggested includes development of performance metrics of hardware and software solutions, and in the partitioning stage of the codesign flow.
I would like to thank Dr. Manwaring for encouraging me and providing me with the opportunity to pursue graduate education. His guidance in my research as well as life in general have enabled me to complete this work and prepare for my career. Dr. Manwaring, it has been an honor to work with you and be mentored by you. Thank you for your patience and for your confidence in my abilities while working on this thesis. I would also like to thank Dr. Archibald and Dr. Wilde not only for willing to serve on my committee, but also for their guidance and friendship during my time at Brigham Young University (BYU). Many thanks to Dr. Michael Jensen for his support in his role as the Graduate Coordinator and also for being a great professor of one of my undergraduate courses at BYU. I would also like to thank Dr. Robert Burton in the Computer Science Department for mentoring me and encouraging me to pursue graduate studies, in my early years as an undergraduate student. I am forever indebted to many who serve in the noble profession of teaching.

I would like to express my gratitude to the staff of the Electrical and Computer Engineering department including Ann Tanner, Janalyn Mergist, Adrian Evans, Mark Ormsby, Gordon Billings and Joe Bussio. I acknowledge the funding provided by the department, including the travel grants for presenting my research work at conferences. I am grateful for the opportunity to attend BYU for both my undergraduate and graduate education. I view my experiences as a student at BYU as a time of great personal growth and of rich association with some of the finest people I have had a chance to know.

I would also like to express my gratitude to many friends who provided constant support and encouragement, and for pointing out the finer things in life apart
from my thesis. Many thanks to Don Wichern and Preston Manwaring for their friendship and patience in helping me with my initial efforts of this thesis.

Finally, I would like to thank my family (including the Sanchi, Smriti and Subhash Khare from Wisconsin) for their support and encouragement in my every step in these educational endeavors and other goals in life and believing that my success was theirs. They have established in me a desire to work hard and accomplish great things.
Contents

Acknowledgments vi

List of Tables xii

List of Figures xiv

1 Introduction 1

2 Problem Statement 3

3 A Perspective on Hardware Software Codesign 7
   3.1 Electronic System Level Design 7
   3.2 Codesign with Reconfigurable Computing 9
   3.3 Platform Overview 10
   3.4 Classification of User Groups 10

4 Platform Designer’s Approach to Codesign 13
   4.1 Hardware Architecture 14
      4.1.1 Microcontroller Module 17
      4.1.2 FPGA Module 23
   4.2 Software Architecture 31
      4.2.1 Downloading the Bitstream 32
      4.2.2 Programming the FPGA with the bitstream 33
      4.2.3 SRAM driver 41

5 Consumer’s Approach to Codesign 43
   5.1 Codesign Experiments 46
<table>
<thead>
<tr>
<th>Section</th>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.4.2</td>
<td>top.ucf</td>
<td>211</td>
</tr>
<tr>
<td>C.4.3</td>
<td>ramtester.vhd</td>
<td>212</td>
</tr>
<tr>
<td>C.4.4</td>
<td>ramtest.ucf</td>
<td>217</td>
</tr>
</tbody>
</table>

Bibliography

219
List of Tables

3.1 ESL design  ..................................................... 8
B.1 Bill of Materials for FPGA module  ................................ 63
B.2 Bill of Materials for the Microcontroller module  ............... 64
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>System level design flow chart</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>A complementary combination of the microcontroller and the FPGA creates an ideal platform for Codesign Experimentation</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>Our approach to codesign flow for our system.</td>
<td>12</td>
</tr>
<tr>
<td>4.1</td>
<td>The steps involved in using the experiment platform for hardware-software codesign.</td>
<td>14</td>
</tr>
<tr>
<td>4.2</td>
<td>The three custom design boards form the experiment platform.</td>
<td>15</td>
</tr>
<tr>
<td>4.3</td>
<td>The Communication and Power Supply Board (CommPS) with which the FPGA and Microcontroller modules are interfaced to the PC.</td>
<td>16</td>
</tr>
<tr>
<td>4.4</td>
<td>Microcontroller board with a 8-bit Atmel Atmega128 and Atmel 8Mbit Dataflash. The board also has three voltage regulators to provide the three different voltages required by the FPGA.</td>
<td>18</td>
</tr>
<tr>
<td>4.5</td>
<td>FPGA board with 400K gate Xilinx XC3S400 Spartan 3 FPGA and 256KX16 (4Mbit) Cypress Semiconductor SRAM for application program storage. JTAG programming option is provided for debugging the configuration process.</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>The platform-designer’s overview of the Experiment Platform.</td>
<td>30</td>
</tr>
<tr>
<td>4.7</td>
<td>The two major tasks which are achieved by the software developed.</td>
<td>31</td>
</tr>
<tr>
<td>4.8</td>
<td>The Atmel’s DataFlash consists of Flash Memory Array, Two Buffers and a Simple I/O Interface</td>
<td>35</td>
</tr>
<tr>
<td>4.9</td>
<td>Command sequence for AT45DB041B Flash ROM read/write operations (except status register read operation)</td>
<td>37</td>
</tr>
<tr>
<td>4.10</td>
<td>Flowchart for slave serial method of configuring Xilinx Spartan FPGAs [1]</td>
<td>40</td>
</tr>
</tbody>
</table>
4.11 Logic block diagram of the static RAM device [2] .......................... 40

5.1 The command line menu system of lubloader downloading utility.  . 44

5.2 The program memory shows how the software is organized on the microcontroller. ......................................................... 45

5.3 The consumer’s view of the experiment platform. ....................... 46

5.4 Arrangement of the hardware modules for a codesign problem. .... 48

A.1 Schematic of Microcontroller Board ................................. 55

A.2 Schematic of Microcontroller Board ................................. 56

A.3 Layout of Microcontroller Board ................................. 57

A.4 Schematic of FPGA Board ................................. 58

A.5 Schematic of FPGA Board ................................. 59

A.6 Schematic of FPGA Board ................................. 60

A.7 Layout of FPGA Board ................................. 61
Chapter 1

Introduction

The number of applications combining hardware and software continues to grow. The design of embedded systems often involves design around strict performance, area and power constraints. Often software-heavy implementations are not satisfactory, thus requiring investigation of a mixed hardware software implementation. A mixed hardware-software system is based upon an efficient partitioning of tasks to be implemented in each domain. The problems of co-design of mixed systems can be approached via concepts such as co-synthesis, co-specification and co-simulation.

The reconfigurability and increasing density of FPGA circuits have made the use of such devices popular for digital system design and prototyping. Academia has also been successful in taking advantage of this technology for developing teaching tools. However there still exists a distinction in university curriculum between hardware and software design. While software design is typically the responsibility of computer science department, hardware design is taught at the electrical engineering department. With increasing relevance of codesign, it is expected that this concept be integrated into the engineering curriculum.

To perform codesign experiments there is a need for hardware circuit boards with appropriate hardware and software resources. There are several FPGA-based platforms developed and offered by the FPGA vendors [3], but many of these platforms ignore the field of codesign and often have steep a learning curve on their user interface. The cost of these boards can often exceed the university budgets. In this thesis we propose an experiment platform for codesign experiments. One of the goals
in developing this platform was to create a low cost and student friendly board for experimentation of hardware software codesign and hardware emulation. The user interface with the modules of the platform is intuitive allowing for the user to focus on the concept implementation rather than adapting to a specific implementation environment. It is expected that the board will be used by students in any kind of hardware design course, from elementary digital circuits to codesign, covering areas of computer architecture and microprocessor designs.

This thesis describes the design of the experiment platform, developed to aid in experimenting with hardware software codesign problems, and how the modules provide the means for hardware emulation. The thesis chapters are organized as follows. Chapter 2 presents the problem statement for the need for an experiment platform for codesign experiments. Chapter 3 introduces the theory of hardware software codesign techniques and their evolution with the advent of reconfigurable technology. This chapter also defines the two groups of people - the first being the platform designer who utilizes codesign principles to develop the platform and the second group being the consumers who use the platform for their experiments. Chapter 4 provides the designer’s approach to codesign. This chapter details the development of the hardware modules and software which runs on it. Chapter 5 provides a consumer’s view to codesign. This chapter describes how the platform which was developed in previous chapter can be used. The conclusion chapter provides a summary of the ideas presented and lists the possible future improvements to our modular system. At the end of the document are appendices containing schematics and code listings for the experiment platform.
Chapter 2

Problem Statement

The complexity of today’s electronic systems and rapid advances in silicon technology has prompted a change in the traditionally sequential processor-based design methodology of electronic systems. The approach to the design of systems has been the independent development of hardware and software. In the design process the paths evolve separately and meet together only at the integration stage. In this approach designers make architectural decisions early in the process and the high recurring engineering costs seldom result in an optimal design.

The standard approach to system level design, as shown in figure 2.1, begins with specification and development of the hardware platform. The hardware platform can be selected as an ASIC or a soft core implemented on a programmable fabric. The next step in the design process creates the software system specific to the hardware which was designed in the previous step. This step can involve the development of a custom compiler according the architecture of the hardware chosen. In the traditional method the software is the only component of the design flow which is redesigned. With the advent of the reconfigurable hardware, the complete system can be evaluated and redesigned.

This has motivated the development of new design methodologies which allow for quick and reliable design of systems, while exploring the design space sufficiently to ensure more optimal solutions. The hardware-software codesign concept encompasses the techniques, tools and practices that support the integration of hardware and software components during system design and development.
In this thesis the concept of hardware software codesign is utilized by two groups. The first group is composed of the platform designer, who creates the experiment board modules to allow for the codesign experiments. The second group is classified as the consumers who consist of engineers who would utilize the experiment platform for their codesign experiments. The engineers of the consumer group could use the experiment board according to their affiliation in academia, research or industry.

Academia is in need of design platforms which allow a student to explore the design space in implementing theoretical concepts learned in class. A design of a processor and its interface functionality are a critical component of any computer engineering curriculum. Often the theory of processor design is rigidly dictated to the students because of the lack of time and resources available. The project laboratories often use software simulators for processor architecture and this prevents a truly
hands-on experience which any student would require. The instruction methodology of a processor design often follows a bottom-up approach. A student would start with a design of a the Central Processing Unit (CPU) consisting of Arithmetic Logic Units (ALUs), register files and other components. The next step would involve interfacing the CPU with memory through a datapath. This interfacing would depend on the architecture design chosen. For example, if a Harvard architecture is chosen, then it would require two separate buses for program instructions and data. In contrast a Van Neumann architecture has a single bus that is shared for both program instructions and data. In order to complete the design of a processor system, the student would be required to make several design decisions. The design process can become very complex and a software simulator will be unable to cover all aspects of the design process.

The area of systems research is also in need of a reliable experiment platform for developing new processor designs. A researcher can approach the design of a system from two perspectives. One perspective is that of focusing on the silicon design. This approach is similar to the System-On-a-Chip (SOC) concept which focuses on creating Application Specific Integrated Circuits (ASIC) components. The other approach is an embedded design or platform-based design approach where performance optimization is focused on software. Both these approaches require flexibility in exploring the design space extensively to come up with the most optimal product.

FPGA-based hardware systems are already available in the low-end category such as the Xess board [4] and the several boards from Digilent [5], and other high-end boards for specific applications such as the XtremeDSP Development Kit board from Nallatech [6]. The low-end category boards are targeted toward simpler designs and for mostly in prototyping digital systems. On the other hand the high end category boards are targeted toward complex applications such as digital signal processing (DSP) and the FPGA device forms a major component of the application platform. Many of these boards have restricted resources or components or include redundant components not applicable to design. These boards may not provide an adequately flexible solution for a researcher or a student implementing their processor-based
design targeted toward simpler designs and for use for mainly prototyping digital systems.

There is a need for a common hardware platform which will afford flexibility and accurate emulation/prototyping to any academician or researcher in their system design efforts. Students should be able to rapidly develop a processor-based design and customize any component which can be constrained by a software simulator. They would also require options of adding or removing hardware features dynamically during the design process. A researcher would need to be able to prototype an original processor design where all the design parameters can be realistically experimented with. The codesign infrastructure needs to robust enough to allow the user to explore the design space extensively to create a performance-effective system.
Chapter 3

A Perspective on Hardware Software Codesign

The standard approach to the design of a system has been to separate the hardware and software design efforts at an early stage. After independent development the two design paths are eventually combined during the integration phase for testing. The complexity of the partitioning during initial stages and the high recurring engineering costs of re-partitioning, create an inefficient system design. This problem is applicable in the popular processor-based systems.

Processor systems were originally board-level systems in which the processor functioned with external hardware on the board. Advances in silicon technology allowed for combining both the CPU and other subsystems onto one chip. The system design problem could then be separated into the design of the CPU hardware and the design of the software running on the CPU. The simultaneous design of this CPU (hardware) and the software can be termed as hardware-software codesign. The term can also refer to the design, trade-off analysis and optimization starting from the abstract function and architecture specification down to the detailed hardware and software implementation. One of the goals of hardware-software codesign in this thesis is to provide a rigorous yet flexible environment in which users can develop their own systems. However there exist various computation models, tools and design flows which produce a challenge for creating such environments [7].

3.1 Electronic System Level Design

Electronic System Level (ESL) [9] design is one of the methods used for classifying the codesign problem. ESL design methods implement the design of hardware
Table 3.1: The different stages of Electronic System Level (ESL) Design. The different stages describe the methodology of solving a system design problem [8].

<table>
<thead>
<tr>
<th>ESL Design Stage</th>
<th>Activities &amp; Models</th>
<th>Design Objectives &amp; Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional Stage</strong></td>
<td>Creating &amp; verifying a functional model of the application (with its test bench)</td>
<td>Defining the right functionality?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What inputs/outputs?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What organization? (structural components &amp; communication data flow)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What behavior for each active component? (control flow)</td>
</tr>
<tr>
<td><strong>Application-driven architectural design</strong></td>
<td>Creating a high level description of the platform Mapping the functional application on the platform Verifying the resulting architectural model Finding the optimal platform</td>
<td>Defining the right architecture for the right application that meets costs &amp; performance constraints:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>How many processors?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What processor characteristics?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What interconnection characteristics?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Are performance constraints met? (processor loads, interconnection throughputs etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What functions in HW and SW?</td>
</tr>
<tr>
<td><strong>Platform-oriented architectural design</strong></td>
<td>Creating a low level description of the platform Fine-tuning the hardware architecture</td>
<td>Defining a virtual prototype of the HW platform:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What kind of processor?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>How much memory?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What bus occupancy, transactions and contention?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>How many cache hits and misses?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>What processor utilization for software tasks?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>How to optimize power consumption?</td>
</tr>
</tbody>
</table>

and software concurrently. ESL defines different stages of hardware-software codesign as follows:

- **Functional Design**
- **Architectural Design**
  - Application-driven architectural design
  - Platform-oriented architectural design

The functional design is the first stage and it describes a system’s behavior from an internal viewpoint. This can include the structure, communications and elementary activities or behaviors. At this stage, no decision is made on choosing hardware or software for implementation. The architectural design stage focuses on describing the system’s architecture in terms of the required hardware and application software to be implemented on this platform. The architectural design can approached from two perspectives. If approached from the software perspective, it is called *application-driven* design, and it is called *platform-oriented* if approached from
the hardware perspective. The functional design and application-driven architectural design typically follows a top-down design style as they start from the application. On the other hand, platform-oriented architectural design typically follows a bottom-up approach by collecting existing components or IP blocks together [8]. The possible design problems and objectives addressed by an ESL tool are shown in table 3.1. The ESL techniques provide a higher level abstraction for a system specification, which can be applied effectively in solving a hardware-software codesign problem.

3.2 Codesign with Reconfigurable Computing

Reconfigurable computing has allowed for practical implementation of hardware-software codesign techniques. FPGA devices are being used to implement the reconfigurable computing technology for solving codesign problems. Many FPGA vendors manufacture devices which combine programmable logic fabric with one or more hard CPUs like IBM’s PowerPC® in Xilinx’s Virtex4® FPGA devices. In a hardware-software codesign problem, the designer can use the ESL methods to conduct performance studies and then make decisions on partitioning of resources and tasks. The analysis might lead to implementation of certain tasks in hardware, which can become expensive. The use of reconfigurable technology makes this process more cost-effective by avoiding the non-recurring costs of ASIC hardware. This allows for a more incremental approach to codesign.

With the increasing density of FPGAs, they are able to implement larger applications on chip. Earlier efforts in hardware-software codesign have utilized FPGAs as co-processors for acceleration of parts of the application which do not meet the required specifications [10]. The advances in silicon technology have resulted in platform FPGAs, which integrate high density configurable logic, embedded processors, interconnect, dedicated multipliers and block RAMs on a single chip. A single FPGA device can thus be used to target a wide range of applications.

Soft processor cores for FPGAs enable customization of processor cores as an application specific embedded processor [11]. Soft processor cores enable a more
accurate prototyping of the system. The cores can also be encapsulated as intellectual property (IP) and can be easily reused several times on multiple FPGA devices.

The biggest advantage of using an FPGA for hardware-software codesign is that there is no need to finalize the partitioning of the design at the beginning of the design process or to create a complex co-simulation environment to model communication between hardware and software. The reconfigurable fabric allows for accurate emulation of the hardware, and the interaction between the hardware and software can be tested easily. There are profiling tools developed which can obtain the clock cycle-accurate profile of the real time performance of a software program running on the soft-core processor instantiated on the FPGA. This assists in obtaining performance measurements and provides better feedback for partitioning[12].

3.3 Platform Overview

As part of this thesis, the experiment platform created consists of a microcontroller and an FPGA. These two devices in combination allow for abundant design space exploration in codesign experiments. The two devices complement each other in functionality as the FPGA forms a part of an embedded solution for a codesign experiment. The microcontroller can support the FPGA operation as a co-processor. It is also used for configuring the FPGA with the bitstream, avoiding any external configuration hardware. Figure 3.1 indicates the current stage of platform design which can be considered as the architecture design stage of ESL design for a codesign experiment. In a codesign experiment, the architecture design stage can be focused around the microcontroller for a platform-oriented solution. In contrast, if the architecture stage is application-driven, the codesign experiment can be implemented on the FPGA which allows for customizable hardware and software to run on it.

3.4 Classification of User Groups

This thesis utilizes the concept of hardware-software codesign from two perspectives. The platform designer utilizes codesign techniques to develop a system which can be used for experimenting with hardware-software codesign. The con-
Figure 3.1: A complementary combination of the microcontroller and the FPGA creates an ideal platform for Codesign Experimentation.

Consumers use this system to experiment with their codesign ideas. The platform designer has to make decisions on component selection, software interface and the flexibility of the design space that is required by the consumers for their experiments. The consumers, consisting of engineers, affiliated with either academia, research groups or industry, use codesign principles to implement their experiments using the hardware and software resources available on the platform developed.

The platform designer utilizes reconfigurable technology in the form of an FPGA to allow for emulation and prototyping of the hardware components of a system. The traditional design approach is modified to include hardware as a part of the redesign process. The architectural design stage of ESL was adapted to our FPGA-based platform and is shown in figure 3.2.
Figure 3.2: Our approach to codesign flow for our system.
Chapter 4

Platform Designer’s Approach to Codesign

One of the main difficulties of building and evaluating the combined hardware-software systems is the inter-dependent nature of hardware and software components. With the increasing complexity of systems, the designer has to consider multiple dependencies between hardware and software. Software simulations of such systems are often not feasible because the complexity of the resulting simulation models is so high, that the simulation run-times are no longer practical. In many cases the required level of detail is achieved only by prototyping a sufficiently large part of the system. The use of breadboard assemblies is no longer practical in prototyping current large systems. The advent of FPGAs has made possible accurate prototyping of such large systems at low cost per gate.

The experiment platform developed as a part of this thesis is used by engineers in their codesign experiments. Figure 4.1 shows the different steps in development and use of the experiment platform. This chapter elaborates on the first step of creating the platform and programming the microcontroller. The platform for codesign experiments is considered as a hardware-software codesign challenge in itself. From a platform designer’s perspective, the concept of hardware-software codesign is defined as the process of creating a platform with appropriate hardware and software components to enable the users the freedom to explore the design space for their experiments. Electronic System Level (ESL) design techniques can be used to address the challenge of designing the platform. The ESL’s functional stage defines the organization of the experiment platform. The experiment platform is built to be
The software system on the PC consists of:

- **Operating System**: Windows XP® professional edition
- **Compiler Tools**: GCC/G++ for C/C++ programming
- **Integrated Development Environments**: Programmer’s Notepad
- **Hardware Synthesis Tools**: Xilinx XST® for Xilinx FPGAs, Xilinx EDK® for soft-core synthesis for Xilinx FPGAs.

The experiment platform is designed with the above software configuration on the PC. Some of the above parameters such as the choice of operating system can be varied. Much of the software is available at no cost for personal or academic use.

### 4.1 Hardware Architecture

The hardware architecture of the experiment platform consists of three custom designed boards as shown in figure 4.2. The first board is the Communication and
Power Supply (CommPS) module. The CommPS board was developed earlier, for use in research and classroom work, and two remaining modules have been built to be compatible with it. The two boards—the Microcontroller module and the FPGA module, were custom created as part of the project upon which this thesis is based. The boards connect together with the CommPS board via a 40-pin header.

Figure 4.2: The three custom design boards form the experiment platform.
The CommPS board was created to interface the PC with any other custom boards developed in the research lab. This board fits well into the ESL design concept as an application-driven design. The CommPS has available on it different communication protocol hardware interfaces such as RS-485, RS-232 and USB as seen in figure 4.3. The board is also equipped with voltage translators so any commercial voltage supply from a 9V to 15V supply can be used to generate the 5V power supply on the board. The flexibility provided by this board allows a user to explore the design space for codesign experiments. The user is able to make trade-off decisions on selecting a certain communication protocol or using a particular voltage supply for the system. Different microprocessor and FPGA boards can be connected to this CommPS board via the 40-pin header to exploit the CommPS’s communication channels for easy programmability and communication with multiple boards.

Figure 4.3: The Communication and Power Supply Board (CommPS) with which the FPGA and Microcontroller modules are interfaced to the PC.
The experiment platform design was influenced by ESL design techniques. A combination of a microcontroller and an FPGA device were chosen to conform to the ESL’s architecture design stage of platform-oriented design and application-driven design. One of the objectives of the system was to be able to integrate the FPGA operation as a part of an embedded solution. In the initial design stage, the FPGA system was to appear on the same board as the microcontroller. The microcontroller and FPGA devices were later moved onto two separate boards. The division into two separate boards allowed the design process to be simplified. It was expected that the verification and debugging of the system consisting of two separate boards would be more feasible. The design process and component selection of the two modules is described in the remainder of this chapter.

The major components utilized in the system are:

- Xilinx Spartan-3® XC3S400 FPGA [13]
- Cypress Semiconductor® 256X16K 4Mbit Static SRAM [2]
- Atmel Dataflash® 8-megabit FLASH ROM [14]
- Atmel Atmega128® 8-bit Microcontroller [15]

4.1.1 Microcontroller Module

The architectural design stage in the development of the experiment platform involves decisions of selecting either an application-driven or a platform-oriented approach. As shown in table 3.1, the platform-oriented architectural design encourages a focus toward the use of the hardware components. A microcontroller has an on-chip CPU which can provide the functionality of a processor. From a classical processor system perspective, a memory storage device is required to complement the processor. A non-volatile storage device such as a Flash ROM is utilized for this purpose. The microcontroller board as seen in figure 4.4, also contains voltage regulators which provide the three different power supply sources for the FPGA (Xilinx Spartan 3). Two LEDs are also provided on this board which enable verification of proper operation of the microcontroller.
4.1.1.1 Microcontroller selection

The selection of the microcontroller device required taking into account any specific computational need in a co-design experiment. It was expected that the microcontroller device chosen would also function as a co-processor with a soft processor core implemented on the FPGA. The device would need to have a sufficient number of pins for interfacing with the FPGA and other peripherals such as the Flash ROM. During the design process, it was found that the task of configuring the FPGA can be integrated into the software implemented on the microcontroller. Xilinx FPGAs are volatile as they are based on SRAM technology and thus a battery backup was considered to maintain the configuration. To avoid the need for the continuous power supply to the FPGA, it was decided to use the method of power-on configuration.

In order to have a power-on configuration, the application subroutines of the microcontroller which performed this task had to run every time power was turned on to the devices. This is done by making the reset vector point to the boot loader section of a microcontroller. After the power up the microcontroller will execute the application in the boot loader section and is then free to implement other subroutines.
in the application section of the memory. The microcontroller selected should have a large enough boot loader section to support the application code for configuring the FPGA.

As an initial step in the platform-oriented (hardware centric) design perspective, a Atmel Atmega128 microcontroller is chosen. The Atmega128 is based on an 8-bit Reduced Instruction Set Computer (RISC) model, and serves as the CPU for the codesign experiments. Henceforth the term microcontroller will refer to the Atmel Atmega128 microcontroller. The microcontroller does not permit any modifications to the architecture of the on-chip CPU itself. The codesign experiments would remain constrained with the microcontroller as the platform microprocessor. The microcontroller has on-chip, 128K Bytes of on-chip programmable memory. This memory can be useful if the microcontroller is used for buffering data. The memory is also large enough to place application code for configuring the FPGA in the memory’s boot loader section. Additional reasons for selecting this microcontroller from Atmel were that it has been used in different research projects in the Brain Instrumentation Lab (BIL) at Brigham Young University, and experiments with software examples were available.

Verification

There are several settings which have to be enabled correctly for the proper functioning of the microcontroller. The first step is to ensure that the device can be detected by the programming software installed on the PC. To ensure this, the device needs to have a functioning clock. The Atmel microcontrollers have an internal oscillator which can be used as the system clock. The clock source is specified by setting the $CLK3-0$ fuse bits of the microcontroller. The low power versions (Atmega128L) are constrained to a maximum external oscillator frequency of 8Mhz. If the internal oscillator is used, the device characteristics restrict the most efficient rate of communication at 19200 baud with an error rate of 2% [15]. Therefore it is advisable to use an external crystal oscillator of 7.37Mhz, as with an external oscillator the clock signal is stable and not sensitive to temperature changes or other factors related to the
microcontroller operation. The 7.37 Mhz is easily divisible by the baud rate constant, allowing for an efficient communication at baud rate of 115200 baud with 0% error rate [15].

Once the device is detected by the programming tools and is programmable, the I/O capability of the microcontroller can be tested. For this purpose there are two LEDs available on the board connected to I/O ports of the microcontroller. A simple way to verify the I/O ports of the microcontroller is to implement a software routine which makes the LEDs blink.

The microcontroller can transfer and process data via its communications hardware. The communication hardware which was chosen was the Universal Asynchronous Receiver Transmitter (UART). Software programs such as Hyperterminal are easily available on most PCs. These programs use the UART protocol for communication. The microcontroller has two UARTs on chip in addition to other communication resources such as Serial Peripheral Interface (SPI) and Boundary Scan technology developed by Joint Test Action Group (JTAG). A simple UART test involves transmitting a character to the PC’s UART and echoing it back to the microcontroller’s UART. The UART can be further tested by implementing software routines which respond based on characters received via the UART. The UARTs were tested by implementing an echo subroutine and further extended it by incorporating the blink LED subroutine to test the I/O ports of the microcontroller. The software code used for these tests is listed in appendix C.3.10.

4.1.1.2 FLASH ROM

The Flash ROM provides a non-volatile storage space for application programs and data. The FPGA configuration data requires storage in a non-volatile location. The FPGA vendor Xilinx provides a Programmable Read-Only Memory (PROM) specifically for this purpose [1]. We can avoid the use of this PROM by making use of the existing on-board Flash ROM. The advantages of using the on-board Flash ROM, and the reasons for using it this way are:
• The FPGA serial configuration protocol can be integrated into the initialization routine of a higher level embedded solution. This allows the overall reset and initialization routine to be easier to coordinate.

• As a part of the software implemented on the microcontroller, the configuration protocol can be modified with relative ease. This is useful in prototype development and also for product upgrades.

• The FPGA configuration bitstream can be embedded into the processor’s program. Thus there are fewer code files, which become easy to manage.

The Flash ROM to be used has to have sufficient capacity for storing the FPGA bitstream and other data. The FPGA device chosen is Xilinx’s Spartan-3 XC3S400 and the size of the bitstream for this FPGA is 1,699,136 bits \[13\]. The size of the bitstream is constant for a particular FPGA and is independent of the size or complexity of the design. The Flash ROM chosen was a 8-megabit Atmel’s Dataflash\(^\text{®}\) which exceeded the requirement for storing FPGA configuration bitstream. The Atmel Dataflash will be referred to as flash ROM or flash in the remainder of this thesis. The flash ROM may also function as storage for application code of the microcontroller and as a standard non-volatile data storage for applications on the system.

**Verification**

The correct operation of the flash ROM had to be tested after mounting the component on the printed circuit board (PCB). The SPI interface between the microcontroller and Flash was implemented, which verified proper connection of the pins between them. A test for correct operation of the flash was to write a byte of data to a specific address on the flash and to read the same data back from the same location. This test was implemented with the microcontroller and was able to verify the correct mounting of the device and proper functioning of the device according to its data sheet. The software code for these tests is listed in appendix C.3.2.
4.1.1.3 FPGA power supply

The microcontroller module has on board voltage regulators which supply the different voltages required for the FPGA. Increasing gate densities and higher clock speeds in programmable logic ICs have resulted in higher current requirements while smaller device geometries are driving lower core supply voltages. The inherent flexibility of interfacing FPGAs to devices of different voltage standards leads to the requirement for multiple supply voltages and currents. While the core voltage supply requirement decreases, the device must continue to support both newer and legacy interfaces, potentially requiring an I/O supply voltage which is different from the core voltage.

The Xilinx Spartan-3 FPGA has the following supply voltage requirements:

- **Core Voltage:** It is designated as $V_{CCINT}$ and is the main power supply for FPGA’s internal logic. The voltage required is 1.2V and the general current requirement for standard designs implemented on the FPGA are about three amps. The FPGA can source up to tens of amps in the worst case depending on the number of gates being used and clock frequency.

- **I/O Voltage:** It is designated as $V_{CCO}$ and this power supply requirement varies from 1.5V, 1.8V, 2.5V or 3.3V. The I/O standards can be set independently by block in the FPGA, so more than one I/O voltage for a single device is possible. The Spartan-3 device supports 17 single-ended standards and seven differential standards [13]. The I/O voltages required by an FPGA are determined by the devices the FPGA is interfacing with and often this low voltage supply generated for the FPGA I/Os can be shared to power the I/O of the companion device and possibly other circuits. In the experiment platform developed in this thesis, the supply is set at 3.3V to enable interfacing with other peripheral devices powered at 3.3V.

- **Auxiliary Supply:** It is designated as $V_{AUX}$ and is important with the newer generation of FPGAs because it is tied into the JTAG, Digital Clock Manager...
(DCM) and other circuitry. This power supply must be sufficiently decoupled in order to avoid power supply transients coupling into the FPGA’s clock circuitry.

The power consumption of an FPGA depends on a number of factors and is specific to the design being implemented. Xilinx provides a power estimator utility on the company website [16] which determines the power system that would be required for a design. This calculator considers design resource usage, toggle rates, operating clock frequencies, I/O usage and other factors to predict the power specifications.

The voltage regulators were chosen from the LM138 series from National Semiconductor. The LM138 are adjustable 3-terminal positive voltage regulators capable of supplying in excess of 5A over a 1.2V to 32V output range. They are easy to use and require two resistors to set the output voltage [17]. All three FPGA voltage supplies were created using this regulator with a 5V supply sourced from the CommPS board. This simplified the design and reduced cost by avoiding specific regulators for each of the three voltages. There are several vendors who provide power supply solutions specifically for FPGAs which tend to be complex and would have been more difficult to incorporate into our design [18].

Verification

The voltage regulators were the first components to be mounted on the board. The output voltage is a function of ratio of two output resistors. The resistors are selected to generate the three voltages of 1.2V, 2.5V and 3.3V. The required voltage values were verified at the corresponding pins on the 40-pin header using a voltmeter.

4.1.2 FPGA Module

In order to extend flexibility to the codesign experiments, an FPGA device is utilized. The reconfigurable features of the FPGA promote an application-driven (software centric) approach to the platform design. It is possible to create Register Transfer Level (RTL) models of a CPU on an FPGA using any Hardware Description Language (HDL), which can be considered as software. The HDL (software) affords the user of the platform the opportunity to explore the processor design space, and
utilize the FPGA as a rapid prototyping device for implementing an *application-driven* processor. This processor can be a custom design or a commercial soft CPU like Xilinx’s Microblaze® or a variation of the latter.

Figure 4.5: FPGA board with 400K gate Xilinx XC3S400 Spartan 3 FPGA and 256KX16 (4Mbit) Cypress Semiconductor SRAM for application program storage. JTAG programming option is provided for debugging the configuration process.

The processor cores written in VHDL are synthesized on the FPGA to emulate the processor component in a hardware-software codesign flow. The processor core implemented on the FPGA device requires the selection of a memory device to execute the application program running on it. This memory storage is a faster volatile memory in form of a SRAM device.

In the current version of the FPGA board as seen in figure 4.5, the JTAG ports are provided to debug any problems arising from the microcontroller-FPGA interface. The microcontroller and the FPGA board are compact (2.5 X 2.5 inches) while lending maximum functionality. A set of two 40 pin headers on the FPGA board provide communication with microcontroller board for configuration data and communication with additional FPGA boards or other peripheral device boards.
4.1.2.1 FPGA device selection

A primary objective of the platform was to be able to emulate a soft processor core on the FPGA. The FPGA chosen had to have enough logic gates for practical testing of codesign problems and to support popular soft CPU cores like Microblaze®, Picoblaze® or user designs of similar complexity. A high density FPGA dispenses the need of partitioning a larger circuit across a sea of smaller FPGAs with the resultant increase in complexity.

Xilinx’s Picoblaze is a 8-bit microcontroller and uses 96 slices (4 slices = 1 Configurable Logic Block (CLB)). The more complex Microblaze soft core is a 32-bit RISC CPU and uses 525 slices. The Spartan-3 series of FPGAs from Xilinx was an ideal choice. It has been aggressively marketed for use in an academic setting. Among the various FPGAs in the Spartan-3 series, even the device with the fewest number of logic gates is big enough to support Microblaze soft core. The XC3S50 is the smallest Spartan-3 device consisting of 50K system gates and the Microblaze soft core occupies a minimum of 60% of the gates. The Picoblaze soft core uses between 13%-25% of the system gates on this device. The largest device in Spartan-3 series was the XC3S5000 consisting of five million system gates, where the Microblaze soft core occupied a minimum of 1.6% of the logic gates [19].

An FPGA device had to be chosen which would be the most efficient in terms of maximum logic for the cost and the number of I/O pins available. The FPGA device selection was further filtered down to those which were available in surface mount packages. Surface mount packages were preferable because of the prohibitive costs of commercially mounting a Ball Grid Array (BGA) package. Since all the devices are able to support the standard 8-bit and 32-bit soft CPU cores, the selection methodology became focused on getting the most amount of logic in a surface mount package. The options for the choice of the package for Spartan-3 was narrowed to:

- XC3S400 PQ208 (208 pins)
- XC3S400 TQ144 (144 pins)
The TQ 144 package was chosen because of sufficient available I/O for peripheral devices even after all the I/O had been allocated to the dedicated devices. Also the pins in this package are less dense and would be easy to mount using hand methods.

One of the objectives while using an FPGA device on the experiment platform was to avoid any FPGA vendor specific configuration hardware which would divert the focus from system design. It was preferred that the operation of the FPGA board be integrated with the existing microcontroller environment which was already familiar to the students who would have taken a microcontroller-based course and for researchers familiar with microcontroller interfaces. The functioning of the FPGA module diverges from operating other traditional devices, as it operates on 3.3V and that the tools for configuring the FPGA mostly rely on JTAG method. To integrate the FPGA module into the existing platform, the issues of voltage levels for communication and an efficient method of configuration of the FPGA board had to be addressed.

There are different ways the Spartan-3 FPGA can be configured [20] [13]. The different configuration modes are selectable via the mode pins.

- **JTAG or Boundary Scan Mode:** JTAG or Boundary Scan Mode is an industry standard (IEEE 1149.1 or 1532) serial programming mode. The JTAG programming pins are driven by the external logic from a cable, a microprocessor or any other logic device. Configuration through the boundary-scan port is always available, independent of the mode selection. Initiating JTAG configuration mode simply turns off the other modes. The FPGA has pins for exclusive use for JTAG configuration and these pins are $TD$, $TM$, $TDO$ and $TCK$.

- **Serial Mode:** In Slave Serial mode, the FPGA receives configuration data in bit-serial form from a serial PROM or other serial source of configuration data. The serial bitstream must be set up at the DIN input a short time before each rising edge of the externally generated clock. The FPGA has pins which are used exclusively for serial configuration. These pins are $CCLK$, $DONE$ and $PROGB$. 

26
The data pin or $D0$ is a dual-purpose pin which implies that after configuration this pins is available for any I/O function. The other dual-purpose pins used in this mode is $INITB$ and $BUSY/DOUT$. These pins are used for debugging and for detecting errors in configuration process. In Master Serial mode, the $CCLK$ pin is an output pin. It is similar to slave serial method except that an internal oscillator is used to generate the configuration clock ($CCLK$). The slave serial mode is selected by applying $<111>$ and master serial mode is selected by applying $<000>$ to the mode pins $M0$, $M1$, $M2$ respectively.

- **Parallel mode:** The parallel modes support the fastest configuration as they allow 8-bits of data to written to the FPGA simultaneously. The $BUSY$ flag controls the data configuration of the FPGA. An external source provides the 8-bit wide data, an active-Low Chip Select ($CSB$) signal and an active-Low Write signal ($RDWRB$). If $BUSY$ signal is asserted high by the FPGA, the data must be held until $BUSY$ goes low. If $RDWRB$ is asserted, configuration data is read out of the FPGA as part of a read back operation. In the case of slave parallel mode, the configuration clock signal $CCLK$ is provided externally while with master parallel mode the clock is generated internally by the FPGA and data is clocked in. The dual purpose pins in this mode are $DIN/D0-D7$, $DOUT/BUSY$, $INITB$, $CSB$ and $RDWRB$, which can be used as user I/Os after configuration. The Master parallel mode is selected by applying $<110>$ and slave parallel mode is selected by applying $<011>$ to the mode pins $M0$, $M1$, $M2$ respectively.

The serial configuration mode was used by taking advantage of the microcontroller device in the system. The Spartan-3 FPGA’s dedicated configuration pins are powered by 2.5V while the Atmega128 microcontroller is powered at 3.3V. The configuration signals are an input to the FPGA, and thus require current-limiting series resistor to limit the reverse current to 10mA or less [13]. This current-limiting resistor value is 56 Ohms, and is connected serially between the $CCLK$ and $PROGB$ pins. It has been demonstrated by board designers [21] that by using an existing
microcontroller in a design for FPGA configuration can reduce the cost of the overall system.

**Verification**

The FPGA module can function once the voltage regulators have been properly verified on the microcontroller board. The FPGA is sensitive to fluctuations in the voltages supply values that it requires (1.2V, 2.5V, 3.3V).

A JTAG port was added to the module to verify the correct mounting of the FPGA device and other components on the Printed Circuit Board (PCB). The JTAG port makes it possible to program the FPGA independent of the microcontroller. The software tools for JTAG configuration are readily available from the FPGA vendor. The crystal has to be correctly mounted for proper functioning of the FPGA device. There are two LEDs available on the FPGA board which can be used to test the functionality. A simple VHDL program which instantiates a hardware counter to blink this LED, can be used to test the FPGA functionality. This code is listed in the appendix C.4.

### 4.1.2.2 SRAM

A traditional processor requires external memory for storage of application data, and hence a Static Random Access Memory (SRAM) is required in combination with the processor core implemented on the FPGA. In this design only one SRAM device was chosen to keep the initial design and interface simple. From an academic perspective, the Von Neumann architecture can be emulated by dividing the single SRAM into application program storage and data storage. To emulate the Harvard architecture, it is anticipated that two SRAMs with dedicated I/O to FPGA for each of them would be designed. This would more accurately emulate the Harvard architecture, where the processor can access both the application program and data simultaneously. A dual-port SRAM memory could also be used to emulate the Harvard architecture.
The device chosen was a 4Megabit, 256X16K SRAM from Cypress Semiconductor. It was estimated that 4Megabit size would be enough for initial experimental applications on the microprocessor core. The FPGA and SRAM combination allow for emulation of processor soft-cores with different architecture features.

**Verification**

The SRAM can be tested by writing data to an address location on the SRAM and reading back from that location. This can be made elegant by implementing a UART on the FPGA through which data can be transferred from the microcontroller and back. This code is listed in the appendix C.4.3.

### 4.1.2.3 Auxiliary Components

The Spartan-3 device can be run at a maximum rate of 326 Mhz. We chose a crystal oscillator with frequency of 50 Mhz to clock the FPGA. This oscillator frequency is expected to be sufficient for codesign experiments. Higher oscillator frequencies require addressing the issues of signal bounce and noise. The FPGA has eight input clock ports and only one port needs to be connected to an external oscillator. Spartan-3 devices provide flexible, complete control over clock frequency, phase shift and skew through the use of the Digital Clock Manager (DCM) feature. The DCM employs a Delay-Locked Loop (DLL), a fully digital control system that uses feedback to maintain clock signal characteristics with a high degree of precision despite normal variation in operating temperature and voltage.

The communication resources for the system can also be designed from a codesign perspective. The standard communication protocols available from PC are RS-232 and USB. These protocols can be supported by designing the appropriate communication hardware on the microcontroller and FPGA module. The communication between components also allows for various protocols in the design space. The microcontroller has on-chip hardware resources to support RS-232 and SPI protocols among others. The *platform-driven* approach to the design is achieved by instantiating the appropriate communication hardware on the FPGA using HDLs for the
protocol that is selected. The communication between the modules and CommPS is constrained by the 40-pin header. This was a trade off in order to keep the modules compact and to maintain the flexibility of stacking on additional hardware component modules.

The combination of the microcontroller and the FPGA, as shown in figure 4.6, allows us to achieve an application-driven design approach. The combined FPGA and microcontroller interface provides for concurrency of functionality implemented on the FPGA with the external interface provided by the microcontroller. The interface between the FPGA and the microcontroller is designed in software, while the microcontroller provides the hardware for external communication. It is also possible to implement hardware on the FPGA module external communication independent of the microcontroller module.

![Diagram of Experiment Platform](image)

Figure 4.6: The platform-designer’s overview of the Experiment Platform.

The presence of a non-volatile memory in form of Flash ROM allows the system to function as a stand alone machine, as the FPGA can be configured on startup from the bitstream stored in the ROM. The system forms a unique environment, in which any custom processor-based system can be constructed, utilizing the FPGA as its
strength. The soft cores of processors or Intellectual Property (IP) cores of machines with their own unique compiler support and software applications can be implemented on the system. The connectability of the board allows for interfacing with other peripheral devices when solving the problem of hardware-software codesign.

4.2 Software Architecture

Software is developed to allow the consumers to use the experiment platform for their hardware-software codesign experiments. The software architecture is classified based on the functionality provided by the experiment platform as seen in figure 4.7. The software provides the following functionality:

1. Download the FPGA configuration bitstream to the flash memory.

2. Program the FPGA using the bitstream stored on flash memory.

The program running on the PC communicates with the software running on the microcontroller to perform the various tasks. The software is developed to run in three environments - PC, Microcontroller and the FPGA.

Figure 4.7: The two major tasks which are achieved by the software developed.
4.2.1 Downloading the Bitstream

The bitstream is a binary representation of an implemented FPGA design, which contains the location information of logic on a device. The embedded source code running on the microcontroller requires a flat binary image of this bitstream to be located on the flash memory. The bitstream for Spartan-3 FPGA is generated by the Xilinx bit generation tools (BitGen) and is denoted with the extension <.bit>. This file contains header information that should not be downloaded to the FPGA. The microcontroller removes this header before transferring it to the flash memory.

The software on the PC for downloading data to the flash memory is based on an open source project LoonBoard Unified Bootloader (LUB) [22]. The download utility consists of two software components – one implemented on the PC and other on the microcontroller. The software component on the PC is a command-line based software called lubloader. The parameters passed at the command line specify the serial port device, the file to be programmed, file format, and other debugging information. The lubloader software communicates with a software present in the boot loader section of the microcontroller. When the lubloader software is running it requires a reset of the microcontroller to make it execute the program in the boot sector. The lubloader communicates with microcontroller using CRC-8 error correction protocol for reliable transmission. The transmission packets are organized as follows:

[SYNC][Packet Number][Packet Type][Data LSB]...[Data MSB][CRC8]

The SYNC byte indicates the start of the packet. The packet number starts at 1 for the first packet and is one byte long. It rolls over to 0 after reaching 255 packets. The packet type defines how many data bytes should be expected by the receiver. The process of transmission involves use of ACK and NACK for controlling the data rate. The next packet of 264 bytes is transmitted only if it receives an ACK for the previous packet. The software running on the microcontroller implements a similar data transfer protocol. This allows the software on the PC and microcontroller to interface more easily. This software interface is available to the public under General Public License (GPL) and is listed in the appendix C.2.
4.2.2 Programming the FPGA with the bitstream

The term configuration refers to the programming of an FPGA with a bitstream. This bitstream is called a configuration bitstream, as it contains the information about the logic which is configured on the FPGA. The configuration subroutine on microcontroller programs the FPGA device via the slave serial method. In this method the software on the microcontroller is considered the Master as it is responsible for retrieving the data from the flash memory and serializing the bitstream. The FPGA device is considered the slave as it receives the data from the microcontroller.

The configuration of the FPGA can occur at two different instances. The configuration subroutine is executed from the boot loader section for automatic configuration of the FPGA at power-up. The other instance allows the consumer to control the configuration of the FPGA by implementing the subroutine within the software running on the microcontroller. To provide a control interface to the user the software running on the microcontroller has to provide the following functions:

- Driver for using the UART
- Driver for the on-board FLASH ROM
- FPGA configuration subroutine using slave serial method.

These functions were implemented successfully and they are described in detail in the succeeding sections.

UART

The microcontroller’s UART has to be initialized before any communication can take place. The microcontroller has different I/O registers for its UART. The initialization process consists of setting the baud rate, setting frame format, and enabling the transmitter or the receiver depending on the usage of the UART. This is done by writing the values to the respective I/O registers. The baud rate is determined based on the clock frequency provided to the microcontroller. The equation used for calculating the value to be written into the UART’s Baud Rate Register (UBRR) is
determined by the equation:

\[
UBRR = \frac{oscillator \, frequency}{BAUD \, (in \, bits \, per \, sec)} - 1
\]  \hspace{1cm} (4.1)

The pseudo code for the initialize subroutine is

```c
void UART_initialize(int baud) {
    Set Baud Rate by writing to the UBRR0H and UBRR0L registers
    Enable transmitter and Receiver by writing to the UCSRB register
    Set Frame format to 8 data bits and 1 stop bit by writing to UCSRC register
    Initialize the Queue for receiving data
    Initialize the Queue for transmitting data
}
```

The data transmission is initiated by loading the transmit buffer. This is done by writing to the UART data register's I/O location. A queue of length 32 is used for buffering data of the UART. The buffer allows for transmission of up to 32 bytes at a time. The subroutine for the data transmission waits for the queue to be full before transmitting the data. The UART subroutines are interrupt driven and an ISR keeps track of the data in the queue. Similarly for the receiving the data, the queue is used for buffering the incoming data. The pseudo code for the transmitting data is:

```c
char transmit(char byte) {
    Enqueue data to be sent out.
    Keep trying if the queue is full
    If queue is full, an ISR routine send the data in the queue.
}
```

Additional subroutines are provided for transmitting array of characters or strings and formatting the data in a integer and hexadecimal formats. The complete code is provided in the appendix C.2.23.
FLASH ROM driver

The software for interfacing with the flash memory is developed after analyzing the underlying architecture of the device. Figure 4.8 shows the organization of the memory on the flash device. The 8650752 bits of memory are organized as 4096 pages of 264 bytes each. In addition to main memory the flash also contains two SRAM buffers of 264 bytes each. The buffers allow receiving of data while a page in the main memory is being reprogrammed, as well as a continuous data stream. The flash uses the SPI (mode 0 and 3) serial interface to sequentially access its data. This simple serial interface facilitates hardware layout, increases system reliability, minimizes switching noise, and reduces package size and active pin count.

The flash interfaces with other devices using only seven signal leads, three of which are dedicated to the serial bus (SCK, SI and SO). The remaining signals leads

![Diagram of flash memory organization](image)

Figure 4.8: The Atmel’s DataFlash consists of Flash Memory Array, Two Buffers and a Simple I/O Interface

The flash interfaces with other devices using only seven signal leads, three of which are dedicated to the serial bus (SCK, SI and SO). The remaining signals leads
on the flash include a chip select (CS), chip reset input (RESET), a write protect input (WP), and a read/busy output (RDY/BUSY). The flash has to be initialized with specific parameters before it can be used. The initialization subroutine consists of setting up the SPI communication protocol in the desired modes. The SPI operating modes determine the clock phase and polarity for transmitting or receiving data. The SPI mode 3 is chosen, in which the rising edge of the SCK signal always clocks the data in, while the falling edge always clocks the data out. The chip select pin functions the same as that of a chip select pin on any memory device and is an important pin for controlling data flow. The pseudo code for the initialization subroutine of flash memory consists of:

```c
void flash_spi_init(void)
{
    Set CS, SCK and MOSI as outputs
    Set the CS high
    Enable SPI in master mode, mode3, with frequency oscillator/4
}
```

To begin the operation on the flash, the system must send a command to the device. The flash device consists of a Command User Interface (CUI) and a state machine that controls all internal operations. The CUI receives the user’s software commands and translates them into state machine operations, and determines the command’s validity. All operations start with an opcode followed by three address bytes that are clocked into the DataFlash (as shown in figure 4.9). The three address bytes (24 bits) are used to address the memory array or buffers for the device. The four MSB bits are Reserved bits; bits 5-15 denote a page number; bits 16-24 denote a specific byte address within the 264-byte page or buffer.

The read operation consists of a specific sequence of opcodes sent to the device. As the dataflash is used for storing larger data like the FPGA configuration data, we use a continuous read opcode. The continuous read command is utilized to sequentially read a continuous stream of data from the device by simply providing a
Figure 4.9: Command sequence for AT45DB041B Flash ROM read/write operations (except status register read operation)

clock signal and no additional addressing information or control signals need to be provided. The dataflash incorporates an internal address counter that will automatically increment on every clock cycle, allowing one continuous read operation without the need of additional address sequences. The continuous read opcode also bypasses both data buffers and leaves the contents of the buffers unchanged. The data is read a byte at a time by reading the SPI data register.

```c
char flash_spi_RW(char output)
{
    Put byte 'output' in SPI data register
    Wait for transfer to complete, poll SPIF-flag
    Read value in SPI data reg.
    Return the byte clocked in from the SPI slave.
}

char Read_flash_status(void)
{
    Toggle ChipSelect (CS) signal to reset the flash command decoder
    Send the status register read opcode
    Dummy write to get the result
}
```
The write operation involves the user of data buffers to transfer the data to the main memory. The data is written to the buffer by first sending a “write to buffer” opcode. The Dataflash’s buffers are static RAMs (SRAM) and therefore data stored within the buffers is not guaranteed if the supply voltage drops below the specified minimum operating level. The buffer’s static nature eliminates the need for refreshing and data will not change until new data is loaded into the buffers. The software routine on the microcontroller keeps track of the number of bytes. Once the counter reaches 264 which is the size of the buffer, a “buffer to page” opcode is sent to the device which transfers the contents of the buffer to main memory.

```c
void buffer_write(char buffer_num, int IntPageAdr, char data)
{
    Toggle CS to reset the flash command decoder
    Send buffer 1write op-code
    Send don’t cares
    Send upper part of the internal buffer address
    Send lower part of the internal buffer address
    Write data byte
}

void buffer_to_page(char buffer_num, int PageAddr)
{
    Toggle the CS signals
    Send opcode for buffer1 to flash
    Write the upper part of the address
    Write the lower part of the address don’t cares
    Initiate flash page programming
    Monitor the status registers, wait until busy-flag is high.
}
Slave Serial Configuration

The serial configuration of FPGA takes advantage of existing resources in the platform, thereby reducing the cost, part count, memory size and board space associated with FPGA configuration circuitry. The microcontroller writing the configuration data plays the controlling role of “Master”, where the FPGA receiving the data, serves as a “slave”. Hence the configuration method is called slave serial configuration of the FPGA. Slave serial configuration is accomplished by providing the Spartan-3 FPGA with a Serial Clock and delivering a single data bit at every rising edge of the Serial clock until the final configuration bit has been sent.

The Slave Serial mode consists of the four steps:

1. Clearing Configuration Memory
2. Initialization
3. Configuration
4. Start-Up

Figure 4.10 shows the configuration flow diagram for the slave serial method of configuration. First the processor writes to the PROGRAM pin which resets the FPGA, clearing the configuration memory. A loop is required to assert the PROGRAM pin for the minimum time period. This step is not required if the FPGA has been powered up or reset through other means. In the next step the INIT signal is checked if it is asserted or not. The subroutine enters a for loop that retrieves a byte of configuration data from the flash memory in each iteration. This byte of data is serialized into bits and presented to the DIN pin of the FPGA. The for loop iteration cycles are equal to the size of the bitstream which is predetermined based on the FPGA device used. After all the data is sent the DONE signal is checked for assertion. A few dummy data are presented to the FPGA for a few extra cycles until the DONE signal is asserted. The codes for implementing the slave serial configuration is provided in the appendix C.3.3.
Figure 4.10: Flowchart for slave serial method of configuring Xilinx Spartan FP-
GAs [1]

Figure 4.11: Logic block diagram of the static RAM device [2]
4.2.3 SRAM driver

The SRAM device is interfaced with the FPGA module. To program the SRAM, software has to be written on the FPGA. The SRAM device is a high-performance CMOS Static RAM organized as 262,144 words by 16 bits as seen in figure 4.11.

Writing to the device is accomplished by taking Chip Enable $\overline{CE}$ and Write Enable $\overline{WE}$ inputs low. If the Byte Low Enable $\overline{BLE}$ is low, then data from the I/O pins (I/O$_0$ - I/O$_7$) is written into the location specified on the address pins (A$_0$-A$_{17}$). If Byte High Enable ($\overline{BHE}$) is low, then data from I/O pins (I/O$_8$-I/O$_{15}$) is written into the location specified on the address pins (A$_0$-A$_{17}$).

Reading from the device is accomplished by taking Chip Enable ($\overline{CE}$) and Output Enable ($\overline{OE}$) low while forcing the Write Enable ($\overline{WE}$) high. If Byte Low Enable ($\overline{BLE}$) is low then data from the memory location specified by the address pins will appear on I/O$_0$ - I/O$_7$. If Byte High Enable ($\overline{BHE}$) is low, then data from memory will appear on I/O$_8$ to I/O$_{15}$. The code for the SRAM interface is provided in the appendix C.4.3.
Chapter 5

Consumer’s Approach to Codesign

The consumers, consisting of engineers, affiliated with either academia, research groups or industry, use codesign principles to implement their experiments using the hardware and software resources available on the platform developed. The design and development of the platform was described in the previous chapter. In this chapter, procedures of using the platform for hardware-software codesign experiments are presented.

Reconfigurable technology enables a consumer to utilize FPGAs that contain both memory and logic elements along with an Intellectual Property (IP) processor core to implement a computer and custom hardware for system-on-a-chip (SoC) applications. The design to be implemented on the FPGA is written with any Hardware Description Language (HDL) like VHDL or Verilog. The HDL description of the system is compiled using synthesis and place and route tools to create a resulting bitstream file which can be used to program/configure the FPGA.

The consumer will use the lubloader utility to download data onto the flash memory. This software is listed in the appendix C.2. The download utility consists of two software components – one implemented on the PC and the other on the microcontroller. The software on the PC provides a command line menu as seen in figure 5.1. The lubloader allows one to transfer files to five different locations on flash depending upon the file type. The download utility stores the FPGA configuration data by default at address location 0x1502 in flash memory. The location at which the bitstream is stored can be changed by the consumer. The command sequence used for storing the FPGA bitstream is lubloader -P com2 -f FPGA.bit -b 115200 -v.
This command sequence sets up communication to the microcontroller module on the PC’s COM2 port at baud rate of 115200 baud. The FPGA configuration bitstream is stored in the FPGA.bit file. The microcontroller is programmed with software which utilizes an error control data transfer protocol. This software is event-based and interfaces with lubloader software on the PC for downloading to be successful.

```
$ lubloader
Example Usage: lubloader -P /dev/ttyS0 -f hardware_interface.bit -a main.hex -q lub.hex -v -b 57600
lubloader -P com1 -a main.hex -b 57600
lubloader -P /dev/ttyS0 -b 57600
  Note: This last example will force the bootloader to load from DataFLASH to AVR
  the normal run-time code
Usage: lubloader [options]
Options:
  -P <port>                  Serial port device.
  -s                        Use safemode bootloader.
  -f <filename>             .bit file to download to FPGA
  -a <filename>             .hex file to download to AVR
  -q <filename>             .hex file for new safemode bootloader
  -v                        Verbose output, use -vv for more
  -b <baud-rate>            Use baudrate specified
  -e                        ERASE the DataFLASH; ONLY FOR DEBUGGING
                           WILL RESULT IN A HARD SYSTEM
  -d                        Enters debug mode in bootloader
  -h                        Display this usage.
```

LicornBoard Unified Bootloader (LUB) project: http://www.newao.com

Figure 5.1: The command line menu system of lubloader downloading utility.

There are two ways in which the FPGA can be configured. Figure 5.2 shows the program memory of the microcontroller with the location of the configuration subroutines. In the first instance the configuration subroutine is implemented in the bootloader section of the microcontroller. The reset vector of the microcontroller is programmed to point to the beginning of the bootloader section. This enables the subroutine to be executed every time on power-up and configures the FPGA using the bitstream stored in the flash memory. The other method of configuring the FPGA is by embedding the configuration subroutine within the application program running on the microcontroller. This subroutine is listed in the appendix C.3.3. A
software implemented on the PC interacts with a menu-based software implemented on the microcontroller to allow the user to control the different tasks like FPGA configuration. This PC software is listed in the appendix C.1.

Figure 5.2: The program memory shows how the software is organized on the microcontroller.

Once an FPGA is configured with the soft-core bitstream, it emulates an independent processor and utilizes the microcontroller as a co-processor. Figure 5.3 provides a perspective of how the experiment platform appears after the configuration of the FPGA with a soft-core processor. The next step is to write and compile software that will be executed on the soft core processor. The FPGA vendors provide tools for development of their soft cores. Xilinx provides tools for development of Microblaze soft core processor which includes a C/C++ compiler targeted at this processor. The consumer can then implement ‘stand-alone’ programs to run on the processor. As an option the consumer is also able to compile code for an operating system targeted for the processor core. A port of regular Linux operating system called µClinux is
available for the Microblaze soft core processor [23]. In the experiment environment, the application program can be revised several times before the final program is complete. A bootloader program can be implemented on the SRAM, which can be used for loading the application program and starting the code execution on the soft-core processor of the FPGA. This can be done “on the fly” without having to reload the FPGA’s hardware configuration.

![Figure 5.3: The consumer’s view of the experiment platform.](image)

### 5.1 Codesign Experiments

A example of a hardware-software codesign 5.4 is the problem of selecting the best soft-processor core to support application of an RF module. The RF module has a weak communication link and requires error detection and correction. The codesign problem can be approached by either moving the error detection and correction tasks to hardware or software. To begin the codesign flow, a microprocessor soft-core is selected. This core could be a readily available standard core such a Microblaze...
or a custom core built to reduce the area costs and consisting of only the required processing resources. The core is prototyped on the FPGA module with an HDL. The next step is to partition the design into hardware and software. If a function is implemented in hardware, the appropriate hardware module can be interfaced with the FPGA module. This hardware module can either be synthesized on the FPGA using an HDL or it can be an Application Specific Integrated Circuit (ASIC) module interfaced with the processor soft-core on the FPGA module. If the function is supported in software, the corresponding application is developed based on the processor soft-core chosen. Also the software has to include drivers to effectively communicate with any peripheral hardware chosen. These steps are iterated until a satisfactory solution is derived. The soft-core enables rapid prototyping of traditional, modified versions of traditional or custom processors which might be selected in the final application.

5.2 Academic

In recent years, FPGAs have lent themselves to applications within the computer engineering curriculum at universities worldwide. The re-programmable nature of the FPGA makes them ideal for educational purposes as it allows students to iterate in their design tasks [24]. The use of FPGA technology often requires tools and software environments produced by the specific FPGA vendor. The costs of these can often affect the research or teaching budget at the university. Further, there is a significant learning curve in using commercial CAD tools for FPGA boards [25]. The experiment platform developed here is targeted to minimize the use of such tools and to develop a system with a low-cost configuration environment with necessary on-board resources.

A computer engineering undergraduate curriculum usually includes an embedded systems course which familiarizes the students with introductory concepts of microprocessor applications and exposes them to the versatility of applications that can be performed by the microprocessor. It is a challenge to create a course which provides students with actual experience designing a microprocessor hardware. A
Figure 5.4: Arrangement of the hardware modules for a codesign problem.
standard computer architecture course educates the students about the hardware architecture of a microprocessor. But such courses often employ software simulators for computer architecture concepts like pipelining, cache organization etc.

The experiment platform developed here can provide an excellent means for educating students at an undergraduate level about microprocessor design. The advent of soft cores allow for emulation of a microprocessor on the FPGA. The IP processor cores or soft cores are often available from the commercial FPGA vendors like Xilinx’s Microblaze and Altera’s Nios. The soft-core designs of traditional processors like the Intel 8051 are often available through open source communities on the world wide web like www.opencores.org. The soft core processors can be very feature rich and flexible, often allowing the designer to specify the data path width, the arithmetic logic unit (ALU) functionality, number and types of peripherals and memory-address space parameters at compile time. The undergraduate student is able to incorporate concepts from other courses such as real time operating system design, computer architecture and digital logic design to implement his/her own processor design. The student with his knowledge of HDLs from other courses can design a processor similar to the LC3 [26] or PIC [27] processor on the FPGA. The experiment platform provides a unique experience in being able to provide realistic behavior of the hardware designed.

At the graduate level, the student can experiment with advanced architecture concepts such as instruction set design or custom application specific processors like signal processing processors. The experiment platform allows the student the flexibility to study different variations of processor design like a RISC or CISC architecture. The ability to add peripheral hardware modules can allow the student to experiment with processors for applications in networking or other areas.

5.3 Metrics

One of the challenges in hardware-software codesign experiments is the concept of metrics. Metrics are generally statistical data which allow the consumer to compare designs for different performance attributes. A popular metric specifies the cost of
implementing the design in hardware or software. Hardware cost metrics can be the execution time, chip area, power consumption or testability. Software cost metrics may include execution time and the amount of required program and data memory.

The experiment platform developed here allows for design space exploration where a partitioning algorithm would be able to produce different solutions in short computation time. One of the useful feature of the experiment platform is that it provides a stable interface between the software implementation on any one module and a hardware implementation on any FPGA-based module. This consistent platform enables implementation of performance studies and to generate metrics.
Chapter 6

Conclusion

This thesis presents the process of designing and developing a platform for use in hardware-software codesign experiments. It was demonstrated that the hardware-software codesign principles could be used to create a platform for implementing codesign experiments. The platform utilized a combination of a microcontroller and a FPGA to provide the consumers with a sufficient design space to explore for their hardware-software codesign experiments. Two custom designed hardware boards were built to support the microcontroller and FPGA operation. Software was developed to enable the use of the two boards with a PC. A menu driven system allows the users to implement their experiments in hardware-software codesign and hardware emulation. The operation of the experiment platform was demonstrated by implementing a Picoblaze® processor-based system.

6.1 Integrated Design

The platform developed provided an integrated system for codesign experimentation. The FPGA device operation was integrated in the main firmware implemented on the microcontroller. The software was implemented such that, the FPGA could be configured on power-up enabling the FPGA module to behave as a stand-alone system. The software also provides an option to the user to configure the FPGA within the main software implemented on the microcontroller. The incremental and the modular approach to the platform design, makes it attractive for use in an academic setting. Any student with knowledge of principles of computer architecture, digital
logic design and basic knowledge of any HDL is able to use this platform for their experiments in codesign, design of custom processors and processor-based systems.

6.2 Future Work

The next step in advancing this thesis work would be to implement different processor-based systems and analyze various performance statistics. These can be used to develop metrics to aid the partitioning stage of hardware-software codesign. Future research directions would include making the current hardware module more compact by combining the FPGA and microcontroller module into a single hardware module. Other improvements in the hardware module could include the ability to use FPGAs from different vendor on the FPGA module. This could provide another opportunity to develop metrics for performance of FPGAs from different vendors.
Appendix
Appendix A

Hardware Schematics

Figure A.1: Schematic of Microcontroller Board
Figure A.2: Schematic of Microcontroller Board
Figure A.3: Layout of Microcontroller Board
Figure A.4: Schematic of FPGA Board
Figure A.5: Schematic of FPGA Board
Figure A.6: Schematic of FPGA Board
Figure A.7: Layout of FPGA Board
# Appendix B

## Bill of Materials

Table B.1: Bill of Materials for FPGA module

<table>
<thead>
<tr>
<th>Qty</th>
<th>Value</th>
<th>Device</th>
<th>Parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50 Mhz</td>
<td>Citizen CSX532FBC</td>
<td>IC3</td>
</tr>
<tr>
<td>3</td>
<td>Red, Green, Yellow</td>
<td>LED 3MM</td>
<td>LED1, LED2, LED3</td>
</tr>
<tr>
<td>3</td>
<td>56Ω</td>
<td>Resistor SMT</td>
<td>R10, R11, R12</td>
</tr>
<tr>
<td>4</td>
<td>100Ω</td>
<td>Resistor SMT</td>
<td>R1, R2, R3, R4</td>
</tr>
<tr>
<td>3</td>
<td>270Ω</td>
<td>Resistor SMT</td>
<td>R6, R8, R9</td>
</tr>
<tr>
<td>1</td>
<td>4.7kΩ</td>
<td>Resistor SMT</td>
<td>R5</td>
</tr>
<tr>
<td>19</td>
<td>0.1µ F</td>
<td>Capacitors SMT</td>
<td>C1-C19</td>
</tr>
<tr>
<td>1</td>
<td>XC3S400</td>
<td>Spartan-3 FPGA</td>
<td>IC1</td>
</tr>
<tr>
<td>2</td>
<td>40 pin</td>
<td>Header Connector</td>
<td>SV1, SV3</td>
</tr>
<tr>
<td>1</td>
<td>6 pin</td>
<td>Male Connector</td>
<td>SV2</td>
</tr>
<tr>
<td>1</td>
<td>2X3 pin</td>
<td>Male Connector</td>
<td>SV4</td>
</tr>
</tbody>
</table>
Table B.2: Bill of Materials for the Microcontroller module

<table>
<thead>
<tr>
<th>Qty</th>
<th>Value</th>
<th>Device</th>
<th>Parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.1µF</td>
<td>C-US025-025X050 Capacitor</td>
<td>C1, C8</td>
</tr>
<tr>
<td>3</td>
<td>0.1µF</td>
<td>C-US025-030X050 Capacitor</td>
<td>C10, C12, C14</td>
</tr>
<tr>
<td>1</td>
<td>7.3728 MHz</td>
<td>TC26H Crystal Oscillator</td>
<td>Q1</td>
</tr>
<tr>
<td>1</td>
<td>20µF</td>
<td>CPOL-USE2.5-5 Capacitor</td>
<td>C9</td>
</tr>
<tr>
<td>1</td>
<td>22pF</td>
<td>C-US025-024X044 Capacitor</td>
<td>C4, C6</td>
</tr>
<tr>
<td>1</td>
<td>100µF</td>
<td>CPOL-USE2.5-5 Capacitor</td>
<td>C7, C11, C13</td>
</tr>
<tr>
<td>4</td>
<td>120Ω</td>
<td>Resistor SMT</td>
<td>R2, R9, R10, R11</td>
</tr>
<tr>
<td>1</td>
<td>200Ω</td>
<td>Resistor SMT</td>
<td>R8</td>
</tr>
<tr>
<td>2</td>
<td>390Ω</td>
<td>Resistor SMT</td>
<td>R1, R3</td>
</tr>
<tr>
<td>1</td>
<td>AT-128</td>
<td>Atmega128®Microcontroller</td>
<td>IC1</td>
</tr>
<tr>
<td>1</td>
<td>AT45DB321B</td>
<td>Dataflash®Memory</td>
<td>U$3</td>
</tr>
<tr>
<td>2</td>
<td>Red, Green</td>
<td>LEDs 3MM®</td>
<td>D1, D2</td>
</tr>
<tr>
<td>3</td>
<td>LM338</td>
<td>National®Voltage Regulator</td>
<td>IC4, IC5, IC6</td>
</tr>
<tr>
<td>2</td>
<td>40 pin</td>
<td>Header Connector</td>
<td>SV1, SV3</td>
</tr>
<tr>
<td>1</td>
<td>12 pin</td>
<td>Male Connector</td>
<td>SV2</td>
</tr>
</tbody>
</table>
Appendix C

Software

C.1 PC Menu

C.1.1 WinCommAppMain.cpp

/* *******************************************************
* WinCommAppMain.cpp Author: Yaju Nagaonkar
* Created 20 Nov 2005
* This file provides the menu interface to the
* Experiment Platform. The Code is implemented on the PC
* and communicates via a RS-232 protocol
******************************************************** */

#include <iostream.h>
#include <stdio.h>
#include "WINCOMM.h"

FILE *FileBin;
FILE *FileWrite;
const char *filename ="temp_write.bin";
const char *writefile ="temp_read.bin";
int main( void)
{
    long filesize;
    unsigned int temp4;
    unsigned char temp3, temp1;
    long i =0;
    int m;

    SerialSetup(1, 3);

    printf("Opening up the serial communication\n");
    printf("Enter 1) Program the FPGA 2) Write to flash memory \n");
    scanf("%d",&temp4);

    switch(temp4)
    {
    case 1:
        // communicate with the menu system on microcontroller code
        // to configure the FPGA
        SerialPut('2');
        printf("Configuring the FPGA using bitstream on the Flash ROM\n");
        break;
    case 2:
        printf("Opening the file\n");
        FileBin = fopen(filename, "rb");
        fseek(FileBin,0, SEEK_END);
filesize = ftell(FileBin);
printf("Number of bytes in the file to be written %ld bytes. \n", filesize);
fseek(FileBin,0, SEEK_SET);

FileWrite = fopen(writefile, "wb");
if (FileBin == NULL)
{
    printf("File not found\n");
}
else
{
    // communicate with the menu system on teh microcontroller
    // to enable transfer of data to the flash rom
    SerialPut('1');
    do
    {
        fread(&temp3,1,1,FileBin);
        SerialPut(temp3);
        temp1 = SerialGet();
        fwrite(&temp1,1,1,FileWrite);
        if(temp3 == temp1)
        {
            if (i == 10 || i == 100 || i == 1000 || i == 10000 || i == 150000 || i > 212380)
            { printf("k 	");}
        }
        if(temp3 != temp1)
        {
            printf("ERROR ERROR ERROR ERROR ERROR ERROR \\
        ERROR ERROR\n");
            printf("Char sent \%c\n",temp3);
            printf("Char received \%c \n", temp1);
        }
        if (i == 10 || i == 100 || i == 1000 || i == 10000 || i == 150000 || i > 212380)
        {
            printf("Iteration i = %d ", i);
            printf("char sent \%c\t",temp3);
            fprintf(stderr, " [\%02lx] \n", temp3);
            printf("char received \%c\t",temp1);
            fprintf(stderr, " [\%02lx] \n", temp3);
        }
        i++;
    }
    while(!feof(FileBin) && (i != 212393));
}
fclose(FileBin);
fclose(FileWrite);
printf("closing the file and value of i %d\n", i);
break;

default:
    printf("Try again\n");
    break;
}

printf("Closing the serial communication\n");
SerialShutdown();
return 0;
/* **********************************************************************
* Filename : WINCOMM.CPP
* Used for communicating between the Host Loader and Resident.
* Utilizes simple serial (RS-232) communication on the Windows Operating
* System
* Author: Mark L. Manwaring, Ph.D.
* Modified : Yaju Nagaonkar June 8, 2005.
* ********************************************************************** */

#include <conio.h>
#include <iostream.h>
#include <time.h>
#include <windows.h>
#include <stdio.h>

/* Global Variables */
HANDLE HandleSerPort;
int CommRunning,
FirstStart = true;

/* Functions ..... */

void Purge(void)
{
   int timer = 0;
   char c;
   DWORD nBytes = 0;

   for(timer = 0; timer < 10024; timer++)
   {
      ReadFile (HandleSerPort, &c, 1, &nBytes, NULL);
   }
}

char SerialGet(void)
{
   DWORD nBytes = 0;
   time_t finish;

   int success = 0;
   char c;

   finish = time(NULL) + 5;

   while ((success == 0) && (time(NULL) < finish))
   {
      ReadFile (HandleSerPort, &c, 1, &nBytes, NULL);
      success = nBytes;
   }
   if (success == 0)
   {
      MessageBox (NULL, "Timed out waiting for communication", "", MB_OK);
      CommRunning = false;
   }
}
void SerialPut (char c)
{
    DWORD nBytes;
    if (!WriteFile (HandleSerPort, &c, 1, &nBytes, NULL))
    {
        MessageBox (NULL, "Problem writing to serial port", "", MB_OK);
        CommRunning = false;
    }
}

void SerialSetup (int PortNum, int BaudRate)
{
    DCB dcbSerPort;
    COMMTIMEOUTS
        t = {MAXDWORD, 0,0,0,0};
    char *PortName = "COM2";
    int CommBaud;
    CommRunning = false;
    switch ( PortNum )
    {
        case 1 : PortName = "COM1"; break;
        case 2 : PortName = "COM2"; break;
        case 3 : PortName = "COM3"; break;
        case 4 : PortName = "COM4"; break;
        default: PortName = "COM2"; break;
    }
    switch ( BaudRate )
    {
        case 1: CommBaud = CBR_9600; break;
        case 2: CommBaud = CBR_19200; break;
        default: CommBaud = CBR_115200; break;
    }
    HandleSerPort = CreateFile (PortName, GENERIC_READ | GENERIC_WRITE,
        0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
        NULL);
    if (HandleSerPort == INVALID_HANDLE_VALUE)
    {
        MessageBox (NULL, "Cannot open Serial port", PortName, MB_OK);
        return;
    }
    GetCommState (HandleSerPort, &dcbSerPort);
    dcbSerPort.BaudRate = CommBaud;
    dcbSerPort.fBinary = TRUE;
    dcbSerPort.fParity = FALSE;
dcbSerPort.fDsrSensitivity = FALSE;
dcbSerPort.fOutxDsrFlow = FALSE;
dcbSerPort.fOutX = FALSE;
dcbSerPort.fInX = FALSE;
dcbSerPort.fNull = FALSE;
dcbSerPort.fAbortOnError = FALSE;
dcbSerPort.ByteSize = 8;
dcbSerPort.Parity = NOPARITY;
dcbSerPort.StopBits = ONESTOPBIT;

if( !SetCommState (HandleSerPort, &dcbSerPort))
{
    MessageBox (NULL, "Bad parameters for port", PortName, MB_OK);
    return;
}

SetCommTimeouts (HandleSerPort, &t);
CommRunning = true;
}*/end SerialSetup/*
****************************************************************************/

void SerialShutdown ( void )
{
    if (FirstStart)
    {
        FirstStart = false;
        return;
    }
    if ( !CloseHandle (HandleSerPort))
    { MessageBox (NULL, "Problem closing the serial port", "", MB_OK); }
    CommRunning = false;
}

C.1.3 WINCOMM.h
****************************************************************************/
 void Purge(void);
 char SerialGet ( void );
 void SerialPut ( char c);
 void SerialSetup (int PortNum, int BaudRate);
 void SerialShutdown ( void );
C.2 Lubload Download Utility

This software is released under the GNU General Public License. For more details please refer to article published in February 2006 edition of Circuit Cellar[22].

C.2.1 main.c

/*
 * lubloader - NewAE LoomBoard Unified Bootloader program
 * Copyright (C) 2005 Colin O'Flynn <coflynn@newae.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

/*
 * Note: code is based heavily on source code to avrdude, which is
 * Copyright (C) 2000-2004 Brian S. Dean <bsd@bsdhome.com>
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <fcntl.h>
#include <limits.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>

#include "fileio.h"
#include "lub.h"
#include "serial_lowlevel.h"
#include "crc8.h"
#include "report_progress.h"

void usage(void);

int get264Bytes(unsigned char * data, unsigned char * file, unsigned long
 current_byte, unsigned long size);

int sendLUBBuffer(unsigned char datatype, unsigned char * file_buffer, unsigned long
 size_of_file);

unsigned char getCRC(void);

long getTarget(unsigned char target);

const char progname[] = "lubloader";

int verbose; /* verbose output */
unsigned char max_tries; /* Maximum allowed retries */

int main (int argc, char * argv [])

70
{
  char data[270];    /* Used for packet buffer */
  char ch;          /* options flag */
  char * e;         /* for strtol() error checking */
  char * port;      /* device port (/dev/zzz) */
  char * file_fpga; /* filename of FPGA file */
  char * file_avr;  /* filename of AVR file */
  char * file_lub;  /* filename of AVR LUB file */
  char * file_safelub; /* filename of AVR LUB file */
  long baudrate;    /* baud rate */
  char bootloader;  /* Set to 1 for safemode bootloader*/
                   /* Set to 0 for normal bootloader */
                   /* Set to 3 for debug */
  char erase;       /* Set to 1 to erase DataFLASH */

  FILE * fpga_fptr; /* Pointer to FPGA file */
  FILE * avr_fptr; /* Pointer to AVR file */
  FILE * lub_fptr; /* Pointer to AVR LUB file */
  FILE * safelub_fptr; /* Pointer to AVR Safemode LUB file */

  long size_of_file;  /* Size of file read */
  long size_of_file2; /* Size of file read */
  unsigned char file_buffer_real[1000000];
  unsigned char * file_buffer; /* 1 MB file buffer */

  long avrsize;      /* Maximum AVR file size */

  unsigned char local_crc;

  char empty_string[] = {'0'};

  /* Note: we will reassign the addresses if we have command line
   * options, so there CAN all point to the same spot. */
  port = empty_string;
  file_fpga = empty_string;
  file_avr = empty_string;
  file_lub = empty_string;
  file_safelub = empty_string;

  baudrate = 115200;
  verbose = 0;
  bootloader = 0;
  max_tries = 5;

  if (argc == 1)
  {
    usage();
    exit(1);
  }

  /*
   * process command line arguments
   */
  while ((ch = getopts(argc, argv, "hdevs:w:b:P:f:a:")) != -1)
  {
    switch (ch)
    {
    case 'b':    /* override default baud rate */
      baudrate = strtol(optarg, &e, 0);
      if ((e == optarg) || (*e != 0))
      {
        fprintf(stderr, "%-s: invalid baud rate specified '%s'\n",
                 progname, optarg);
        exit(1);
      }
  
}
if (port == empty_string)
{
    fprintf(stderr, "%s: ERROR: You must specify a serial port with -P!\n", proname);
    exit(1);
}

if (verbose > 0)
{
    fprintf(stderr, "%s: Using serial port: %s at baud of %ld\n", proname, port, baudrate);
    fprintf(stderr, "%s: Using FPGA filename as: %s\n", proname, file_fpga);
    fprintf(stderr, "%s: Using AVR filename as: %s\n", proname, file_avr);
    fprintf(stderr, "%s: Using BOOTLOADER filename as: %s\n", proname, file_lub);
}
fprintf(stderr, "%s: Using SAFEMODE BOOTLOADER filename as: %s\n", progname, file_safelub);
}

/* Sanity checking. Cannot re-write safemode bootloader if we are going to use
safemode to enter the
bootloader, doing so would be suicide.
Also do not write both safemode and normal mode bootloader at once. Again,
suicide */
if ((file_lub != empty_string) && (file_safelub != empty_string))
{
fprintf(stderr, "%s: SAFTEY OVERRIDE: You are attempting to write both the
safemode and\n" "files were corrupt.\n", progname, progname, progname, progname);
exit(1);
}

if ((file_safelub != empty_string) && (bootloader == 1))
{
fprintf(stderr, "%s: SAFTEY OVERRIDE: You are attempting to write the
safemode bootloader\n" "while running from the safemode
bootloader. If the file\n" "was corrupt you may have no working
bootloaders on the\n" "target. Write the normal bootloader and
program the safemode\n" "bootloader from there.\n", progname, progname, progname, progname, progname);
exit(1);
}

/* Figure out which files we should be using, open them */
if (file_fpga != empty_string)
{
fpfpga_fptr = fopen(file_fpga, "rb");
if (fpfpga_fptr == NULL)
{
fprintf(stderr, "%s: Unable to open file %s\n", progname, file_fpga);
exit(1);
}
}

if (file_avr != empty_string)
{
avr_fptr = fopen(file_avr, "rb");
if (avr_fptr == NULL)
{
fprintf(stderr, "%s: Unable to open file %s\n", progname, file_avr);
exit(1);
}
}

if (file_lub != empty_string)
{
lub_fptr = fopen(file_lub, "rb");
if (lub_fptr == NULL)
{
fprintf(stderr, "%s: Unable to open file %s\n", progname, file_lub);
exit(1);
}
}
if (file_safelub != empty_string)
{
    safelub_fptr = fopen(file_safelub, "rb");
    if (safelub_fptr == NULL)
    {
        fprintf(stderr, "%s: Unable to open file %s\n", proname, file_safelub);
        exit(1);
    }
}

/* Open the serial port */
serial_init(port, baudrate);

/* Communicate with LUB */
fprintf(stderr, "\n%s: Attempting to communicate with Loonboard Unified Bootloader\n", proname);
fprintf(stderr, "%s: Power cycle device now to initiate LUB\n", proname);

/* The LightLUB can understand three commands: 's' for safe mode bootloader
'b' for normal bootloader
'l' for normal startup */
ife (lub_EnterLUB(bootloader) != 0)
{
    fprintf(stderr, "%s: Unable to enter Loonboard Unified Bootloader\n", proname);
    exit(1);
}

/* Send Start Of Transmission */
if (verbose > 0)
    fprintf(stderr, "%s: Sending SOT...\n", proname);
if (lub_SendPacket(LUB_SOT, NULL, max_tries) != 0)
    {
        fprintf(stderr, "%s: Comm error\n", proname);
        exit(1);
    }
if (lub_GetPacket(data, 10, max_tries) != 0)
    {
        fprintf(stderr, "%s: Comm error\n", proname);
        exit(1);
    }
if (data[2] != LUB_SIGN_ON)
    {
        fprintf(stderr, "%s: Received %x, Expecting %x\n", proname, data[2],
                LUB_SIGN_ON);
        exit(1);
    }
if (verbose > 0)
    {
        fprintf(stderr, "%s: LUB Target: %d\n", proname, data[3]);
        printf(stderr, "%s: LUB Version: %d\n", proname, data[4]);
    }

avrsiz = getTarget(data[3]);
if (erase == 1)
{

fprintf(stderr, "%%s: ERASE Of DataFLASH in progress.\n", proname);
if (lub_SendPacket(LUB_SUICIDE, NULL, max_tries) != 0)
{
    fprintf(stderr, "%%s: Comm error\n", proname);
    exit(1);
}
fprintf(stderr, "%%s: DataFLASH ERASED.\n", proname);

/*************** SEND FPGA FILE ******************/
if (file_fpga != empty_string)
{
    /* Open FPGA File, assign to buffer */
    file_buffer = file_buffer_real;
    size_of_file = fread(file_buffer, 1, 1000000, fpga_fptr);
    
    if (size_of_file > 1000000)
    {
        fprintf(stderr, "%%s: Internal limit, file for FPGA must be smaller than 1MB\n", proname);
        exit(1);
    }
    /* Okay... so we need to process the header somewhat, find the point with 0x65 0 x00 */
    while (((* file_buffer != 0x65) || (*( file_buffer + 1) != 00)) && (size_of_file != 0))
    {
        file_buffer ++;
        size_of_file --;
        continue;
    }
    if (size_of_file == 0)
    {
        fprintf(stderr, "%%s: Unable to read file. Did you specify the proper option?\n", proname);
        exit(1);
    }
    file_buffer += 2;
    /* The next three bytes will be the size of the file */
    size_of_file2 = *((file_buffer) * (65536));
    size_of_file2 += *((file_buffer+1) * (256));
    size_of_file2 += *((file_buffer+2) * (1));
    /* Check that the sizes are the same */
    size_of_file -= 5;
    if (size_of_file2 != size_of_file)
    {
        fprintf(stderr, "%%s: File size mismatch between reported and actual.\n", proname);
        fprintf(stderr, "%%s: Reported: %%ld Actual: %%ld.\n", proname, size_of_file2, size_of_file);
        exit(1);
    }
    if (verbose > 0)
    {
        fprintf(stderr, "%%s: FPGA Bitstream has size of %%ld bytes\n", proname, size_of_file);
    }
/* We send three bytes of size info first though... */
size_of_file += 3;

// Clear CRC
crc8_clear1();

fprintf(stderr, "%s: Sending FPGA bit file\n", proname);
sendLUBBuffer(LUB_SEND_FPGA, file_buffer, size_of_file);

// Save local checksum
local_crc = crc8_getcrc1();

// Check CRC
if (verbose > 0)
{
    fprintf(stderr, "%s: Local checksum: %X\n", proname, local_crc);
}
if (getCRC() == local_crc)
{
    fprintf(stderr, "%s: Program verify OK\n", proname);
}
else
{
    fprintf(stderr, "%s: Program verify FAILED\n", proname);
}

/**************************** Send LUB File ***************************/

if (file_lub != empty_string)
{
    fprintf(stderr, "%s: Sending Bootloader file\n", proname);

    /* Convert AVR ihex to bin */
    file_buffer = file_buffer_real;
    size_of_file = (long)ihex2b(file_lub, lub_fptr, file_buffer, 1000000);
    if (size_of_file < 0)
    {
        fprintf(stderr, "%s: AVR LUB hex file error\n", proname);
        exit(1);
    }
    if (size_of_file > avrsize)
    {
        fprintf(stderr, "%s: WARNING: AVR binary may be too big. You will not be told
height again.\n", proname);
    }
    if (verbose > 0)
    {
        fprintf(stderr, "%s: AVR LUB binary has size of %ld bytes\n", proname,
size_of_file);
    }

    /* We pad the file with 0xFF */
    while (size_of_file != avrsize)
    {
        file_buffer[size_of_file] = 0xFF;
        size_of_file++;
    }
// Clear CRC
crc8_clear1();

// Send data
sendLUBBuffer(LUB_SEND_BOOT, file_buffer, size_of_file);

// Save local checksum
local_crc = crc8_getcrc1();

// Check CRC
if (verbose > 0) {
    fprintf(stderr, "%s: Local checksum: %X\n", progname, local_crc);
}
if (getCRC() == local_crc) {
    fprintf(stderr, "%s: Program verify OK\n", progname);
} else {
    fprintf(stderr, "%s: Program verify FAI LE\n", progname);
}

/********************************************************* Send Safemode LUB File *******************************/
if (file_safelub != empty_string) {
    fprintf(stderr, "%s: Sending Safemode Bootloader file\n", progname);

    /* Convert AVR ihex to bin */
    file_buffer = file_buffer_real;
    size_of_file = (long)ihex2b(file_safelub, safelub_fptr, file_buffer, 1000000);
    if (size_of_file < 0) {
        fprintf(stderr, "%s: AVR Safemode LUB hex file error\n", progname);
        exit(1);
    }
    if (size_of_file > avrsize) {
        fprintf(stderr, "%s: WARNING: AVR Safemode binary may be too big. You will not be told again.\n", progname);
    }
    if (verbose > 0) {
        fprintf(stderr, "%s: AVR Safemode LUB binary has size of %ld bytes\n", progname, size_of_file);
    }
    /* We pad the file with 0xFF */
    while (size_of_file != avrsize) {
        file_buffer[size_of_file] = 0xFF;
        size_of_file++;
    }
}
/Clear CRC
crc8_clear1();

sendLUBBuffer(LUB_SEND_SAFEBOOT, file_buffer, size_of_file);

//Save local checksum
local_crc = crc8_getcrc1();

//Check CRC
if (verbose > 0)
{
    fprintf(stderr, "%s: Local checksum: %X\n", progname, local_crc);
}

if (getCRC() == local_crc)
{
    fprintf(stderr, "%s: Program verify OK\n", progname);
}
else
{
    fprintf(stderr, "%s: Program verify FAILED\n", progname);
}

/******************* Send AVR File *******************/

if (file_avr != empty_string)
{
    /* Convert AVR hex to bin */
    file_buffer = file_buffer_real;
    size_of_file = (long)ihex2b(file_avr, avr_fptr, file_buffer, 1000000);
    if (size_of_file < 0)
    {
        fprintf(stderr, "%s: AVR hex file error\n", progname);
        exit(1);
    }
    if (size_of_file > avrsize)
    {
        fprintf(stderr, "%s: WARNING: AVR binary may be too big. You will not be told
        again.\n", progname);
    }
    if (verbose > 0)
    {
        fprintf(stderr, "%s: AVR binary has size of %ld bytes\n", progname,
            size_of_file);
    }
    /* We send three bytes of size info first though... */
    /*size_of_file += 3;
    /* We pad the file with 0xFF */
    while (size_of_file != avrsize)
    {
        file_buffer[size_of_file] = 0xFF;
        size_of_file++;
    }
    fprintf(stderr, "%s: Sending AVR file\n", progname);
// Clear CRC
crc8_clear1();
sendLUBBuffer(LUB_SEND_AVF, file_buffer, size_of_file);

// Save local checksum
local_crc = crc8_getcrc1();

// Check CRC
if (verbose > 0)
  {  
    fprintf(stderr, "%s: Local checksum: %X\n", proname, local_crc);
  }

if (getCRC() == local_crc)
  {  
    fprintf(stderr, "%s: Program verify OK\n", proname);
  }
else
  {  
    fprintf(stderr, "%s: Program verify FAILED\n", proname);
  }

/* Send End of Transmission (EOT) */

if (verbose > 0)  
  {
      fprintf(stderr, "%s: Sending EOT Packet\n", proname);
  }

if (lub_SendPacket(LUB_EOT, NULL, max_tries) != 0)
  {  
        fprintf(stderr, "%s: Comm error.\n", proname);
        exit(1);
  }

/* Reset the loader to force it to load the new AVR code */

fprintf(stderr, "%s: Reseting AVR and forcing load of new code\n", proname);

// No longer true
// fprintf(stderr, "%s: FPGA code will still not be loaded until hard reset\n", proname);

while (serial_getc() != '?')
  {  
      continue;
  }

serial_putc('l');
exit(0);

/* Send a buffer */
int sendLUBBuffer(unsigned char datatype, unsigned char * file_buffer, unsigned long size_of_file)
  {  
      unsigned char data[264];
      unsigned char returnvalue;
      unsigned int progress;
          }
unsigned long current_byte;

/** We are now in the main LUB, so first send the fact we want to store a new AVR file */
if (verbose > 0) {
  fprintf(stderr, "%s: Sending request to store file.\n", progname);
}

//send request
if (lub_SendPacket(datatype, NULL, max_tries) != 0) {
  fprintf(stderr, "%s: LUB not responding\n", progname);
  return -1;
}

progress = 0;
current_byte = 0;

if (verbose < 2) {
  report_progress (progress, size_of_file / 264, "Writing");
}

/* Send a 264 byte data packet until we are done */
do {
  progress++;
  returnvalue = get264Bytes(data, file_buffer, current_byte, size_of_file);
  if (verbose < 2) {
    report_progress (progress, size_of_file / 264, NULL);
  }
  if (verbose > 1) {
    fprintf(stderr, "%s: Sending 264 byte data packet\n", progname);
  }
  //Send data
  if (lub_SendPacket(LUB_264_DATA, data, max_tries) != 0) {
    fprintf(stderr, "%s: Data comm error.\n", progname);
    return -1;
  }
  current_byte += 264;
} while (returnvalue == 0);

if (verbose > 0) {
  fprintf(stderr, "%s: File sent\n", progname);
}
return 0;

/*************************************************************************/

int get264Bytes(unsigned char * data, unsigned char * file, unsigned long current_byte, unsigned long size)
{
  unsigned int current_count = 0;
  /* Get 264 Bytes */
  return 0;
}
// Do 264 bytes
while (current_count != 264)
{
    // Note: since current_byte starts at 0, and size starts at 1,
    // every single byte will be read and there is no problem with
    // using a < here
    if (current_byte < size)
    {
        data[current_count] = file[current_byte];
    }
    else
    {
        // Fill unused with 0xFF
        data[current_count] = 0xFF;
    }
    current_count ++;
    current_byte ++;
}
if (current_byte < size)
    return 0;
else
    return -1;
}

/*
* Figure out the target information
*/
long getTarget (unsigned char target)
{
    if (target == 0x03)
    {
        if (verbose > 0)
        {
            fprintf(stderr, "%s: Target is AtMega88 with 7680 byte limit\n", proiname);
        }
        return 7680;
    }
    else if (target == 0x04)
    {
        if (verbose > 0)
        {
            fprintf(stderr, "%s: Target is AtMega168 with 15872 byte limit\n", proiname);
        }
        return 15872;
    }
    else if (target == 0x05)
    {
        if (verbose > 0)
        {
            fprintf(stderr, "%s: Target is AtMega128 with 130046 byte limit\n", proiname);
        }
        return 130046;
    }
    else
    {
        fprintf(stderr, "%s: Error: Unknown Target\n", proiname);
        exit(0);
    }
}

/*
* usage message
void usage(void) {
    fprintf(stderr, "Example Usage:
    \%s -P /dev/ttyS0 -f hardware_interface.bit -a main.hex -q lub.hex -v -b 57600\n    \%s -P com1 -a main.hex -b 57600\n    \%s -P /dev/ttyS0 -b 57600\n    Note: This last example will force the bootloader to load from DataFLASH to
    AVR\n    the normal run-time code\n    "
    "Usage: \%s [options]\n    "Options:\n    " -P <port> Serial port device\n    " -s Use safemode bootloader\n    " -f <filename> .bit file to download to FPGA\n    " -a <filename> .hex file to download to AVR\n    " -q <filename> .hex file for new bootloader\n    " -w <filename> .hex file for new safemode bootloader\n    " -v Verbose output, use -vv for more\n    " -b <baudrate> Use baudrate specified\n    " -e ERASE the DataFLASH: ONLY FOR DEBUGGING\n    " -d Enters debug mode in bootloader\n    " -h Display this usage\n    "\n    \n    LoonBoard Unified Bootloader (LUB) project: http://www.newae.com\n    ,progname, proname, proname, proname);)
}

unsigned char getCRC(void) {
    unsigned char data[10]; /* Used for packet buffer */
    /* Check file */
    fprintf(stderr, "%s: Instructing lub to calculate CRC\n", progname);
    if (lub_SendPacket(LUB_CRC1, data, max_tries) != 0) {
        fprintf(stderr, "%s: Comm error\n", progname);
        exit(1);
    }
    if (lub_GetPacket(data, 10, max_tries) != 0) {
        fprintf(stderr, "%s: Comm error\n", progname);
        exit(1);
    }
    if (verbose > 0) {
        fprintf(stderr, "%s: Remote checksum: %X %X\n", progname, data[4], data[3]);
    }
    return (data[3]);
}
C.2.2 report_progress.h

void report_progress (int completed, int total, char *hdr);

C.2.3 report_progress.c

/*@* avrdude - A Downloader/Uploader for AVR device programmers
* Copyright (C) 2000-2004 Brian S. Dean <bsd@bsdhome.com>
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA */

#include <stdio.h>
#include <time.h>
#include <sys/time.h>
#include "report_progress.h"

/* mingw doesn't support gettimeofday, make our own */
#ifdef GETTIMEOFDAY
/* These are winbase.h definitions */
typedef struct _FILETIME {
  unsigned long dwLowDateTime;
  unsigned long dwHighDateTime;
} FILETIME;

void __stdcall GetSystemTimeAsFileTime(FILETIME*);

void gettimeofday(struct timeval* p, void* tz /* IGNORED */);

void gettimeofday(struct timeval* p, void* tz /* IGNORED */)
{
  union {
    long long ns100; /* time since 1 Jan 1601 in 100ns units */
    FILETIME ft;
  } _now;

  GetSystemTimeAsFileTime( &(_now.ft) );
  p->tv_usec=(long)((_now.ns100 / 10LL) % 1000000LL );
  p->tv_sec= (long)((_now.ns100-(116444736000000000LL))/10000000LL);
  return;
}
#endif /* GETTIMEOFDAY */

static void update_progress_tty (int percent, double etime, char *hdr);
static void update_progress_no_tty (int percent, double etime, char *hdr);
typedef void (*FP_UpdateProgress)(int percent, double etime, char *hdr);

static FP_UpdateProgress update_progress;

/* Report the progress of a read or write operation from/to the device.

The first call of report_progress() should look like this (for a write op):
    report_progress(0, 1, "Writing");

Then hdr should be passed NULL on subsequent calls while the operation is
progressing. Once the operation is complete, a final call should be made as
such to ensure proper termination of the progress report:
    report_progress(1, 1, NULL);

It would be nice if we could reduce the usage to one and only one call for
each of start, during and end cases. As things stand now, that is not
possible and makes maintenance a bit more work. */

void report_progress(int completed, int total, char *hdr)
{
    static int last = 0;
    static double start_time;
    int percent = (completed * 100) / total;
    struct timeval tv;
    double t;

    if (update_progress == NULL)
        // return;
        update_progress = update_progress_tty;

    gettimeofday(&tv, NULL);
    t = tv.tv_sec + ((double)tv.tv_usec)/1000000;

    if (hdr)
        {
        last = 0;
        start_time = t;
        update_progress(percent, t - start_time, hdr);
        }

    if (percent > 100)
        percent = 100;

    if (percent > last)
        {
        last = percent;
        update_progress(percent, t - start_time, hdr);
        }

    if (percent == 100)
        last = 0;  /* Get ready for next time. */
    }

static void update_progress_tty(int percent, double etime, char *hdr)
{
    static char hashes[51];
    static char *header;
    static int last = 0;
    int i;

    hashes[50] = 0;

    memset(hashes, ' ', 50);
    for (i=0; i<percent; i+=2)
        {
        hashes[i/2] = '#';
        }

    if (hdr)
        {
fprintf (stderr, "\n");
last = 0;
header = hdr;
}

if (last == 0) {
    fprintf(stderr, "\r%s | %s | %d%% %0.2fs",
            header, hashes, percent, etime);
}

if (percent == 100) {
    last = 1;
    fprintf(stderr, "\n\n");
}

static void update_progress_no_tty (int percent, double etime, char *hdr)
{
    static int done = 0;
    static int last = 0;
    int cnt = (percent >>1)*2;

    if (hdr) {
        fprintf (stderr, "\n%s | ", hdr);
        last = 0;
        done = 0;
    } else {
        while ((cnt > last) && (done == 0)) {
            fprintf (stderr, "#");
            cnt -= 2;
        }
    }

    if ((percent == 100) && (done == 0)) {
        fprintf (stderr, " | 100%% %0.2fs\n\n", etime);
        last = 0;
        done = 1;
    } else
        last = (percent >>1)*2;  /* Make last a multiple of 2 */
}
C.2.4  ser_win32.c

/*
 * avrdude - A Downloader/Uploader for AVR device programmers
 * Copyright (C) 2003, 2004 Martin J. Thomas <mthomas@rhrk.uni-kl.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

/* $Id: ser_win32.c,v 1.8 2005/11/24 15:00:49 c_oflynn Exp $ */

/*
 * Native Win32 serial interface for avrdude.
 */

#ifdef WIN32NATIVE

#include <windows.h>
#include <stdio.h>
#include <ctype.h>  /* for isprint */
#include "serial.h"

static char * progname = "lubloader";
extern int verbose;

long serial_recv_timeout = 5000;  /* ms */
#define W32SERBUFSIZE 1024

struct baud_mapping {
  long baud;
  DWORD speed;
};

static struct baud_mapping baud_lookup_table[] = {
  { 1200, CBR_1200 },
  { 2400, CBR_2400 },
  { 4800, CBR_4800 },
  { 9600, CBR_9600 },
  { 19200, CBR_19200 },
  { 38400, CBR_38400 },
  { 57600, CBR_57600 },
  { 115200, CBR_115200 },
  { 0, 0 }  /* Terminator. */
};

static DWORD serial_baud_lookup(long baud)
{
  struct baud_mapping * map = baud_lookup_table;

  while (map->baud) {
    if (map->baud == baud) {
      return map->speed;
      map++;
    }
  }

  return 0;
}

#endif
static BOOL serial_w32SetTimeOut(HANDLE hComPort, DWORD timeout) // in ms
{
    COMMTIMEOUTS ctm;
    ZeroMemory(&ctm, sizeof(COMMTIMEOUTS));
    ctm.ReadIntervalTimeout = timeout;
    ctm.ReadTotalTimeoutMultiplier = timeout;
    ctm.ReadTotalTimeoutConstant = timeout;
    return SetCommTimeouts(hComPort, &ctm);
}

static int ser_setspeed(int fd, long baud)
{
    DCB dcb;
    HANDLE hComPort = (HANDLE)fd;

    ZeroMemory(&dcb, sizeof(DCB));
    dcb.DCBlength = sizeof(DCB);
    dcb.BaudRate = serial_baud_lookup(baud);
    dcb.fBinary = 1;
    dcb.fDtrControl = DTR_CONTROL_DISABLE;
    dcb.fRtsControl = RTS_CONTROL_DISABLE;
    dcb.ByteSize = 8;
    dcb.Parity = NOPARITY;
    dcb.StopBits = ONESTOPBIT;

    if (!SetCommState(hComPort, &dcb))
        return -1;

    return 0;
}

static int ser_open(char * port, long baud)
{
    LPVOID lpMsgBuf;
    HANDLE hComPort = INVALID_HANDLE_VALUE;

    /* if (hComPort != INVALID_HANDLE_VALUE)
       fprintf(stderr, "\%s: ser_open(): \"%s\" is already open\n",
               progname, port);
     */

    hComPort = CreateFile(port, GENERIC_READ | GENERIC_WRITE, 0, NULL,
                           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (hComPort == INVALID_HANDLE_VALUE) {
        FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
                      FORMAT_MESSAGE_FROM_SYSTEM |
                      FORMAT_MESSAGE_IGNORE_INSERTS,
                      NULL,
                      GetLastError(),
                      MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default
                      language
                      (LPTSTR) &lpMsgBuf,
                      0,
                      NULL);
        fprintf(stderr, "\%s: ser_open(): can't open device \"%s\": %s\n",
                progname, port, (char*)lpMsgBuf);
    }
LocalFree ( lpMsgBuf );
exit (1);
}

if (!SetupComm ( hComPort , W32SERBUFSIZE , W32SERBUFSIZE ))
{
    CloseHandle ( hComPort );
    fprintf ( stderr , "%s: ser_open (): can't set buffers for \"%s\"\n",
               proiname , port);
    exit (1);
}

if (ser_setspeed ((int) hComPort , baud ) != 0)
{
    CloseHandle ( hComPort );
    fprintf ( stderr , "%s: ser_open (): can't set com-state for \"%s\"\n",
               proiname , port);
    exit (1);
}

if (!serial_w32SetTimeOut ( hComPort ,0 ))
{
    CloseHandle ( hComPort );
    fprintf ( stderr , "%s: ser_open (): can't set initial timeout for \"%s\"

               \n",
               proiname , port);
    exit (1);
}

return (int) hComPort;
}

static void ser_close ( int fd )
{
    HANDLE hComPort = (HANDLE) fd;
    if ( hComPort != INVALID_HANDLE_VALUE )
        CloseHandle ( hComPort );
    hComPort = INVALID_HANDLE_VALUE;
}

static int ser_send ( int fd , char * buf , size_t buflen )
{
    size_t len = buflen;
    unsigned char c = '\0';
    DWORD written;
    char * b = buf;
    HANDLE hComPort = (HANDLE) fd;
    if ( hComPort == INVALID_HANDLE_VALUE )
        {
            fprintf ( stderr , "%s: ser_send (): port not open\n",
                       proiname);
            exit (1);
        }
    if (!len)
        return 0;
    if (verbose > 3)
        {
            fprintf ( stderr , "%s: Send: \", proiname);
            while (len )
                {
                    c = *b;
                    b++;
if (isprint(c)) {
    fprintf(stderr, "%c ", c);
} else {
    fprintf(stderr, ". ");
    b++;
    len--;
}
fprintf(stderr, "\n">
serial_w32SetTimeOut(hComPort, 500);

if (!WriteFile(hComPort, buf, buflen, &written, NULL)) {
    fprintf(stderr, "\%s: ser_send(): write error: \%s\n", 
            programe, "sorry no info avail"); // TODO 
    exit(1);
}

if (written != buflen) {
    fprintf(stderr, "\%s: ser_send(): size/send mismatch\n", 
            programe);
    exit(1);
}
return 0;
}

static int ser_recv(int fd, char * buf, size_t buflen) {
    unsigned char c;
    char * p = buf;
    size_t len = 0;
    DWORD read;
    HANDLE hComPort = (HANDLE)fd;
    if (hComPort == INVALID_HANDLE_VALUE) {
        fprintf(stderr, "\%s: ser_read(): port not open\n", 
                programe);
        exit(1);
    }
    serial_w32SetTimeOut(hComPort, serial_recv_timeout);
    if (!ReadFile(hComPort, buf, buflen, &read, NULL)) {
        LPVOID lpMsgBuf;
        FormatMessage(
            FORMAT_MESSAGE_ALLOCATE_BUFFER | 
            FORMAT_MESSAGE_FROM_SYSTEM | 
            FORMAT_MESSAGE_IGNORE_INSERTS, 
            NULL, 
            GetLastError(), 
            MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language 
            (LPTSTR) &lpMsgBuf, 
            0, 
            NULL);
        fprintf(stderr, "\%s: ser_recv(): read error: \%s\n", 
                programe, (char*)lpMsgBuf);
        LocalFree(lpMsgBuf);
        exit(1);
    }
    p = buf;
if (verbose > 3) {
    fprintf(stderr, "%s: Recv: ", progname);

    // while (len) {
    c = *p;
    if (isprint(c)) {
        fprintf(stderr, "%c ", c);
    } else {
        fprintf(stderr, ". ");
    }
    fprintf(stderr, "[%02 x] ", c);
    // p++;
    // len--;
    // }
    fprintf(stderr, "\n");
    return 0;
}

static int ser_drain(int fd, int display)
{
    // int rc;
    unsigned char buf[10];
    BOOL readres;
    DWORD read;
    HANDLE hComPort=(HANDLE)fd;

    if (hComPort == INVALID_HANDLE_VALUE) {
        fprintf(stderr, "%s: ser_drain(): port not open\n", progname);
        exit(1);
    }
    serial_w32SetTimeOut(hComPort,250);
    if (display) {
        fprintf(stderr, " drain >");
    }
    while (1) {
        readres=ReadFile(hComPort, buf, 1, &read, NULL);
        if (!readres) {
            LPVOID lpMsgBuf;
            FormatMessage(
                FORMAT_MESSAGE_ALLOCATE_BUFFER |
                FORMAT_MESSAGE_FROM_SYSTEM |
                FORMAT_MESSAGE_IGNORE_INSERTS,
                NULL,
                GetLastError(),
                MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
                (LPTSTR) &lpMsgBuf,
                0,
                NULL);
            fprintf(stderr, "%s: ser_drain(): read error: %s\n", progname, (char*)lpMsgBuf);
            LocalFree( lpMsgBuf );
            exit(1);
        }
        if (read) { // data avail!
            if (display) fprintf(stderr, "%02x ", buf[0]);
        }
} else { // no more data
    if (display) fprintf(stderr, "<drain\n");
    break;
}
} // while
return 0;
}

struct serial_device serial_serdev =
{
    .open = ser_open,
    .setspeed = ser_setspeed,
    .close = ser_close,
    .send = ser_send,
    .recv = ser_recv,
    .drain = ser_drain,
};

struct serial_device *serdev = &serial_serdev;

#endif /* WIN32NATIVE */
C.2.5  serial_lowlevel.h

#ifndef __LUB_SERIAL_LOWLEVEL_HEADER
#define __LUB_SERIAL_LOWLEVEL_HEADER

void serial_init(char * port, long baud);
void serial_putchar(char c);
unsigned char serial_getchar(void);
void serial_putchar(unsigned char * data, unsigned int number_data_bytes);
void serial_getchar(unsigned char * data, unsigned int number_data_bytes);
void serial_end(void);
#endif // __LUB_SERIAL_LOWLEVEL_HEADER

C.2.6  serial_lowlevel.c

#include <stdio.h>
#include "serial.h"
#include "serial_lowlevel.h"

unsigned int fd;

void serial_init(char * port, long baud)
{
    fd = serial_open(port, baud);
}

void serial_putchar(char c)
{
    serial_putchar(fd, &c, 1);
    return;
}

unsigned char serial_getchar(void)
{
    static unsigned char buffer;
    serial_getchar(fd, &buffer, 1);
    return buffer;
}

void serial_putchar(unsigned char * data, unsigned int number_data_bytes)
{
    if (number_data_bytes > 0)
        serial_putchar(fd, data, number_data_bytes);
    return;
}

void serial_putchar(unsigned char * data, unsigned int number_data_bytes)
{
    if (number_data_bytes > 0)
        serial_putchar(fd, data, number_data_bytes);
    return;
}

void serial_end(void)
{
    serial_close(fd);
}
C.2.7  crc8.h

/**************************************************************************
* PROJECT: LoonBoard Project
* 
* COPYRIGHT: Colin O'Flynn of NewAE, (C) 2005 <coflynn@newae.com>
* 
* LICENSE: This file is released under a license specified in the LICENSE
* file in the main project directory that this file was in. If
* the license file is missing, then this file is NOT licensed
* and all standard copyright laws apply.
* 
* FILE NAME: crc8.h
* 
* PURPOSE: Provides the CRC8 routines used as error checking for the
* LUB protocol to transmit and receive packets.
* 
* GLOBAL VARIABLES:
* 
* Variable Type Description
* -------- ---- -----------
* none
* 
* DEVELOPMENT HISTORY:
* 
* Date(DMY) Author Release Description Of Change
* ---- ------ ------- ---------------------
* 30-04-2005 Colin O'Flynn Initial Initial file written
* 
**************************************************************************/

#ifndef __LUB_CRC8_HEADER
#define __LUB_CRC8_HEADER

void crc8_clear ( void );
void crc8_addbyte ( unsigned char data );
unsigned char crc8_getcrc ( void );

void crc8_clear1 ( void );
void crc8_addbyte1 ( unsigned char data );
unsigned char crc8_getcrc1 ( void );
#endif /* __LUB_CRC8_HEADER */

C.2.8  crc8.c

#include "crc8.h"

static unsigned char crc, crc1;

void crc8_clear ( void )
{
    crc = 0;
}

void crc8_clear1 ( void )
{
    crc1 = 0;
}

void crc8_addbyte ( unsigned char data )
{
    unsigned char bit_counter;
    unsigned char feedback_bit;

    bit_counter = 8;
    do
    {
    
    
}
feedback_bit = (crc ^ data) & 0x01;
if (feedback_bit == 0x01)
{
    crc = crc ^ 0x18;  // 0X18 = X^8+X^5+X^4+X^0
}
crc = (crc >> 1) & 0x7F;
if (feedback_bit == 0x01)
{
    crc = crc | 0x80;
}
data = data >> 1;
bit_counter--;
}while (bit_counter > 0);

void crc8_addbyte1(unsigned char data)
{
    unsigned char bit_counter;
    unsigned char feedback_bit;
    bit_counter = 8;
do
    {
        feedback_bit = (crc1 ^ data) & 0x01;
        if (feedback_bit == 0x01)
        {
            crc1 = crc1 ^ 0x18;  // 0X18 = X^8+X^5+X^4+X^0
        }
        crc1 = (crc1 >> 1) & 0x7F;
        if (feedback_bit == 0x01)
        {
            crc1 = crc1 | 0x80;
        }
        data = data >> 1;
        bit_counter--;
    }while (bit_counter > 0);
}

unsigned char crc8_getcrc(void)
{
    return crc;
}

unsigned char crc8_getcrc1(void)
{
    return crc1;
}
C.2.9 fileio.h

/*
 * avrdude - A Downloader/Uploader for AVR device programmers
 * Copyright (C) 2000-2004 Brian S. Dean <bsd@bsdhome.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 */

/* $Id: fileio.h,v 1.1.1.1 2005/05/25 20:49:55 c_oflynn Exp $ */

#ifndef __fileio_h__
#define __fileio_h__

typedef enum {
    FMT_AUTO,
    FMT_SREC,
    FMT_IHEX,
    FMT_RBIN,
    FMT_IMM
} FILEFMT;

struct fioparms {
    int op;
    char * mode;
    char * iodesc;
    char * dir;
    char * rw;
};

enum {
    FIO_READ,
    FIO_WRITE
};

char * fmtstr(FILEFMT format);

int ihex2b(char * infile, FILE * inf,
            unsigned char * outbuf, int bufsize);

#endif

C.2.10 fileio.c

/*
 * avrdude - A Downloader/Uploader for AVR device programmers
 * Copyright (C) 2000-2004 Brian S. Dean <bsd@bsdhome.com>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 */
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <ctype.h>
#include "fileio.h"

#define IHEX_MAXDATA 256
#define MAX_LINE_LEN 256 /* max line length for ASCII format input files */

struct ihexrec {
    unsigned char reclen;
    unsigned int loadofs;
    unsigned char rectyp;
    unsigned char data[IHEX_MAXDATA];
    unsigned char cksum;
};

extern char * progname;
extern char progbuf[];

int b2ihex(unsigned char * inbuf, int bufsize, int recsize, int startaddr, char * outfile, FILE * outf);
int b2srec(unsigned char * inbuf, int bufsize, int recsize, int startaddr, char * outfile, FILE * outf);
int src2b(char * infile, FILE * inf, unsigned char * outbuf, int bufsize);
int ihex_readrec(struct ihexrec * ihex, char * rec);
int srec_readrec(struct ihexrec * srec, char * rec);
int fileio_rbin(struct fioparms * fio, char * filename, FILE * f, unsigned char * buf, int size);
int fileio_lhex(struct fioparms * fio, char * filename, FILE * f, unsigned char * buf, int size);
int fileio_srec(struct fioparms * fio, char * filename, FILE * f, unsigned char * buf, int size);
int fmt_autodetect(char * fname);

char * fmtstr(FILEFMT format)
{
    switch (format) {

case FMT_AUTO : return "auto-detect"; break;
case FMT_SREC : return "Motorola S-Record"; break;
case FMT_IHEX : return "Intel Hex"; break;
case FMT_RBIN : return "raw binary"; break;
default : return "invalid format"; break;
};

int b2ihex(unsigned char * inbuf, int bufsize,
            int recsize, int startaddr,
            char * outfile, FILE * outf)
{
    unsigned char * buf;
    unsigned int nextaddr;
    int n, nbytes, n_64k;
    int i;
    unsigned char cksum;

    if (recsize > 255) {
        fprintf(stderr, "%s: recsize =%d, must be < 256\n",
                progname, recsize);
        return -1;
    }

    n_64k = 0;
    nextaddr = startaddr;
    buf = inbuf;
    nbytes = 0;

    while (bufsize) {
        n = recsize;
        if (n > bufsize)
            n = bufsize;

        if ((nextaddr + n) > 0x10000)
            n = 0x10000 - nextaddr;

        if (n) {
            cksum = 0;
            fprintf(outf, " :%02 X %04 X00 ", n, nextaddr);
            cksum += n + ((nextaddr >> 8) & 0x0ff) + (nextaddr & 0x0ff);
            for (i=0; i<n; i++) {
                fprintf(outf, " %02 X", buf[i]);
                cksum += buf[i];
            }
            cksum = -cksum;
            fprintf(outf, " %02 X\n", cksum);
            nextaddr += n;
            nbytes += n;
        }

        if (nextaddr >= 0x10000) {
            int lo, hi;
            /* output an extended address record */
            n_64k ++;
            lo = n_64k & 0xff;
            hi = (n_64k >> 8) & 0xff;
            cksum = 0;
            fprintf(outf, " :02000004%02X%02X", hi, lo);
            cksum += 2 + 0 + 4 + hi + lo;
            cksum = -cksum;
            fprintf(outf, " %02 X\n", cksum);
            nextaddr = 0;
        }
    }

    return 0;
}
int ihex_readrec(struct ihexrec * ihex, char * rec)
{
int i, j;
char buf[8];
int offset, len;
char * e;
unsigned char cksum;
int rc;

len = strlen(rec);
offset = 1;
cksum = 0;
/* reclen */
if (offset + 2 > len)
    return -1;
for (i = 0; i < 2; i++)
    buf[i] = rec[offset++];
buf[2] = 0;
ihex->reclen = strtoul(buf, &e, 16);
if (e == buf || *e != 0)
    return -1;
/* load offset */
if (offset + 4 > len)
    return -1;
for (i = 0; i < 4; i++)
    buf[i] = rec[offset++];
buf[4] = 0;
ihex->loadofs = strtoul(buf, &e, 16);
if (e == buf || *e != 0)
    return -1;
/* record type */
if (offset + 2 > len)
    return -1;
for (i = 0; i < 2; i++)
    buf[i] = rec[offset++];
buf[2] = 0;
ihex->rectyp = strtoul(buf, &e, 16);
if (e == buf || *e != 0)
    return -1;
cksum = ihex->reclen + ((ihex->loadofs >> 8) & 0x0ff) + (ihex->loadofs & 0x0ff) + ihex->rectyp;
/* data */
for (j = 0; j < ihex->reclen; j++) {
/* advance to next 'reclen' bytes */
    buf[2] = n;
    bufsize -= n;
}
if ( offset + 2 > len )
    return -1;
for (i = 0; i < 2; i++)
    buf[i] = rec[offset++];
buf[i] = 0;
ihex->data[j] = strtoul(buf, &e, 16);
if (e == buf || *e != 0)
    return -1;
cksum += ihex->data[j];
}
/* cksum */
if ( offset + 2 > len )
    return -1;
for (i = 0; i < 2; i++)
    buf[i] = rec[offset++];
buf[i] = 0;
ihex->cksum = strtoul(buf, &e, 16);
if (e == buf || *e != 0)
    return -1;
rc = -cksum & 0x000000ff;
return rc;

/*
 * Intel Hex to binary buffer
 *
 * Given an open file 'inf' which contains Intel Hex formatted data,
 * parse the file and lay it out within the memory buffer pointed to
 * by outbuf. The size of outbuf, 'bufsize' is honored; if data would
 * fall outside of the memory buffer outbuf, an error is generated.
 *
 * Return the maximum memory address within 'outbuf' that was written.
 * If an error occurs, return -1.
 *
 */
int ihex2b(char * infile, FILE * inf,
unsigned char * outbuf, int bufsize)
{
    char buffer [ MAX_LINE_LEN ];
    unsigned char * buf;
    unsigned int nextaddr, baseaddr, maxaddr;
    int i;
    int lineno;
    int len;
    struct ihexrec ihex;
    int rc;

    lineno = 0;
    buf = outbuf;
    baseaddr = 0;
    maxaddr = 0;

    while (fgets((char *)buffer, MAX_LINE_LEN, inf) != NULL) {
        lineno++;
        len = strlen(buffer);
        if (buffer[len-1] == '\n')
            buffer[--len] = 0;
        if (buffer[0] != ':')
            continue;
        rc = ihex_readrec(&ihex, buffer);
        if (rc < 0) {
            fprintf(stderr, "%s: invalid record at line %d of \"%s\"\n",}
progname, lineno, infile);
return -1;
}
else if (rc != ihex.cksum) {
    fprintf(stderr, "progname, lineno, infile); %s: ERROR: checksum mismatch at line %d of \\"%s\\n", 
    proname, lineno, infile);
    fprintf(stderr, "%s: checksum=0x%02x, computed checksum=0x%02x\\n", 
    proname, ihex.cksum, rc);
    return -1;
}

switch (ihex.rectyp) {
    case 0: /* data record */
        nextaddr = ihex.loadofs + baseaddr;
        if (nextaddr + ihex.reclen > bufsize) {
            fprintf(stderr, 
                "progname, nextaddr+ihex.reclen, lineno, infile); %s: ERROR: address 0x%04x out of range at line %d of %s\\n", 
                proname, nextaddr+ihex.reclen, lineno, infile);
            return -1;
        }
        for (i=0; i<ihex.reclen; i++) {
            buf[nextaddr+i] = ihex.data[i];
        }
        if (nextaddr+ihex.reclen > maxaddr)
            maxaddr = nextaddr+ihex.reclen;
        break;
    case 1: /* end of file record */
        return maxaddr;
    break;
    case 2: /* extended segment address record */
        baseaddr = (ihex.data[0] << 8 | ihex.data[1]) << 4;
        break;
    case 3: /* start segment address record */
        /* we don’t do anything with the start address */
        break;
    case 4: /* extended linear address record */
        baseaddr = (ihex.data[0] << 8 | ihex.data[1]) << 16;
        break;
    case 5: /* start linear address record */
        /* we don’t do anything with the start address */
        break;
    default:
        fprintf(stderr, 
            "%s: don’t know how to deal with rectype=%d " 
            "at line %d of %s\\n", 
            proname, ihex.rectyp, lineno, infile);
        return -1;
        break;
}
/* while */

fprintf(stderr, 
    "%s: WARNING: no end of file record found for Intel Hex " 
    "file \\"%s\\n", 
    proname, infile);

return maxaddr;
int b2srec(unsigned char * inbuf, int bufsize,
  int recsize, int startaddr,
  char * outfile, FILE * outf)
{
  unsigned char * buf;  
  unsigned int nextaddr;
  int n, nbytes, addr_width;
  int i;
  unsigned char cksum;
  char * tmpl = 0;

  if (recsize > 255) {
    fprintf(stderr, "%s: ERROR: recsize=%d, must be < 256\n",
            progname, recsize);
    return -1;
  }

  nextaddr = startaddr;
  buf = inbuf;
  nbytes = 0;
  addr_width = 0;

  while (buysize) {

    n = recsize;

    if (n > bufsize)
      n = bufsize;

    if (n) {
      cksum = 0;
      if (nextaddr + n <= 0xffff) {
        addr_width = 2;
        tmpl = "S1 %02 X %04 X";
      }
      else if (nextaddr + n <= 0xffffff)
        addr_width = 3;
        tmpl = "S2 %02 X %06 X";
      else if (nextaddr + n <= 0xffffffff)
        addr_width = 4;
        tmpl = "S3 %02 X %08 X";
      else {
        fprintf(stderr, "%s: ERROR: address=%d, out of range\n",
                  progname, nextaddr);
        return -1;
      }

      fprintf(outf, tmpl, n + addr_width + 1, nextaddr);
      cksum += n + addr_width + 1;

      for (i = addr_width; i > 0; i--)
        cksum += (nextaddr >> (i-1) * 8) & 0xff;

      for (i = nextaddr; i < nextaddr + n; i++) {
        fprintf(outf, "%02X", buf[i]);
        cksum += buf[i];
      }

      cksum = 0xff - cksum;
      fprintf(outf, "%02X\n", cksum);

      nextaddr += n;
      nbytes += n;
  }
}
} /* advance to next 'recsize' bytes */
bufsize -= n;
}

/* -----------------------------------------------------------------
add the end of record data line
----------------------------------------------------------------- */
cksum = 0;
n = 0;
nextaddr = 0;

if (startaddr <= 0xffff) {
    addr_width = 2;
tmpl = "S9 %02 X %04 X";
}
else if (startaddr <= 0xfffff) {
    addr_width = 3;
tmpl = "S9 %02 X %06 X";
}
else if (startaddr <= 0xffffffff) {
    addr_width = 4;
tmpl = "S9 %02 X %08 X";
}

fprintf(outf, tmpl, n + addr_width + 1, nextaddr);

cksum += n + addr_width +1;
for (i= addr_width ; i >0; i --)
    cksum += (nextaddr >> (i - 1) * 8) & 0xff;
cksum = 0xff - cksum;
fprintf(outf, " %02 X\n", cksum);

return nbytes;
}

int srec_readrec(struct ihexrec * srec, char * rec) {
    int i, j;
    char buf[8];
    int offset, len, addr_width;
    char * e;
    unsigned char cksum;
    int rc;

    len = strlen(rec);
    offset = 1;
    cksum = 0;
    addr_width = 2;

    /* record type */
    if (offset + 1 > len)
        return -1;
srec->rectyp = rec[offset ++];
    if (srec->rectyp == 0x32 || srec->rectyp == 0x38)
        addr_width = 3; /* S2,S8-record */
    else if (srec->rectyp == 0x33 || srec->rectyp == 0x37)
        addr_width = 4; /* S3,S7-record */

    /* reclen */
    if (offset + 2 > len)
        return -1;
    for (i=0; i<2; i++)
        buf[i] = rec[offset ++];
    buf[i] = 0;
srec->reclen = strtol(buf, &e, 16);
cksum += srec->reclen;
srec->reclen -= (addr_width+1);
if (e == buf || *e != 0)
    return -1;

/* load offset */
if (offset + addr_width > len)
    return -1;
for (i=0; i<addr_width*2; i++)
    buf[i] = rec[offset++];
buf[i] = 0;
srec->loadofs = strtoull(buf, &e, 16);
if (e == buf || *e != 0)
    return -1;

for (i=addr_width; i>0; i--)
    cksum += (srec->loadofs >> (i - 1) * 8) & 0xff;

/* data */
for (j=0; j<srec->reclen; j++) {
    if (offset+2 > len)
        return -1;
    for (i=0; i<2; i++)
        buf[i] = rec[offset++];
    buf[i] = 0;
    srec->data[j] = strtoul(buf, &e, 16);
    if (e == buf || *e != 0)
        return -1;
    cksum += srec->data[j];
}

/* cksum */
if (offset + 2 > len)
    return -1;
for (i=0; i<2; i++)
    buf[i] = rec[offset++];
buf[i] = 0;
srec->cksum = strtoull(buf, &e, 16);
if (e == buf || *e != 0)
    return -1;
rc = 0xff - cksum;
return rc;
}

int srec2b(char *infile, FILE *inf,
        unsigned char *outbuf, int bufsize)
{
    char buffer [ MAX_LINE_LEN ];
    unsigned char *buf;
    unsigned int nextaddr, baseaddr, maxaddr;
    int i;
    int lineno;
    int len;
    struct ihexrec srec;
    int rc;
    int reccount;
    unsigned char datarec;
    char *msg = 0;

    lineno = 0;
    buf = outbuf;
    baseaddr = 0;
    maxaddr = 0;
    reccount = 0;

    while (fgets(buffer, MAX_LINE_LEN, inf)) {
        i = 0;
        while (buffer[i] != ':')
            i++;
        if (buffer[i] != ':')
            continue;

        for (i++; i < strlen(buffer); i++)
            buffer[i] = 0;

        srec = (struct ihexrec *)malloc(sizeof(struct ihexrec));
        srec->rec = buffer;
        srec->size = strlen(buffer);
        srec->line = lineno;
        srec->nextaddr = nextaddr;
        srec->baseaddr = baseaddr;
        srec->maxaddr = maxaddr;
        srec->reccount = reccount;
        srec->datarec = datarec;
        srec->cksum = 0;
        srec->loadofs = 0;
        srec->reclen = 0;

        srec2b(infile, inf, outbuf, bufsize,
        srec->reclen, srec->loadofs, srec->data);

        free(srec);
    }

    return rc;
}
while (fgets((char *)buffer,MAX_LINE_LEN,inf)!=NULL) {
    lineno++;  
    len = strlen(buffer);
    if (buffer[len-1] == '\n')  
        buffer[--len] = 0;
    if (buffer[0] != 0x53) 
        continue;
    rc = srec_readrec(&srec, buffer);

    if (rc < 0) { 
        fprintf(stderr, "%s: ERROR: invalid record at line %d of \"%s\"\n",  
                progname, lineno, infile);
        return -1;
    } else if (rc != srec.cksum) {  
        fprintf(stderr, "%s: ERROR: checksum mismatch at line %d of \"%s\"\n",  
                progname, lineno, infile);
        fprintf(stderr, "%s: checksum =0x %02x, computed checksum=0x%02x\n",  
                progname, srec.cksum, rc);
        return -1;
    }

datarec=0;
    switch (srec.rectyp) {
    case 0x30: /* S0 - header record*/  
        /* skip */  
        break;
    case 0x31: /* S1 - 16 bit address data record */
        datarec=1;
        msg="%s: ERROR: address 0x%04x out of range at line %d of %s\n";
        break;
    case 0x32: /* S2 - 24 bit address data record */
        datarec=1;
        msg="%s: ERROR: address 0x%06x out of range at line %d of %s\n";
        break;
    case 0x33: /* S3 - 32 bit address data record */
        datarec=1;
        msg="%s: ERROR: address 0x%08x out of range at line %d of %s\n";
        break;
    case 0x34: /* S4 - symbol record (LSI extension) */
        fprintf(stderr,  
               "%s: ERROR: not supported record at line %d of %s\n",  
               progname, lineno, infile);
        return -1;
    case 0x35: /* S5 - count of S1,S2 and S3 records previously tx'd */
        if (srec.loadofs != reccount){
            fprintf(stderr,  
                   "%s: ERROR: count of transmitted data records mismatch "  
                   "at line %d of \"%s\"\n",  
                   progname, lineno, infile);
            fprintf(stderr,  
                   "%s: transmitted data records= %d, expected = "  
                   "value= %d\n",  
                   progname, reccount, srec.loadofs);
            return -1;
        } 
        break;
    case 0x37: /* S7 Record - end record for 32 bit address data */
    case 0x38: /* S8 Record - end record for 24 bit address data */
    case 0x39: /* S9 Record - end record for 16 bit address data */
        return maxaddr;
    default:  
        104
fprintf(stderr, 
    "%s: ERROR: don't know how to deal with rectype S%d " 
    "at line %d of %s\n", 
    progname, srec.rectyp, lineno, infile);
    return -1;
}

if (datarec == 1) {
    nextaddr = srec.loadofs + baseaddr;
    if (nextaddr + srec.reclen > bufsize) {
        fprintf(stderr, msg, progname, nextaddr+srec.reclen, lineno, infile);
        return -1;
    }
    for (i = 0; i < srec.reclen; i++)
        buf[nextaddr+i] = srec.data[i];
    if (nextaddr+srec.reclen > maxaddr)
        maxaddr = nextaddr+srec.reclen;
    reccount++;
}

fprintf(stderr, 
    "%s: WARNING: no end of file record found for Motorola S-Records " 
    "file \"%s\"\n", 
    progname, infile);
    return maxaddr;
}

int fileio_rbin(struct fioparms * fio, 
    char * filename, FILE * f, unsigned char * buf, int size) 
{
    int rc;
    switch (fio->op) {
    case FIO_READ:
        rc = fread(buf, 1, size, f);
        break;
    case FIO_WRITE:
        rc = fwrite(buf, 1, size, f);
        break;
    default:
        fprintf(stderr, "%s: fileio: invalid operation=%d\n", 
            progname, fio->op);
        return -1;
    }
    if (rc < 0 || (fio->op == FIO_WRITE && rc < size)) {
        fprintf(stderr, 
            "%s: %s error %s %s: %s; %s %d of the expected %d bytes\n", 
            progname, fio->iodesc, fio->dir, filename, strerror(errno), 
            fio->rw, rc, size);
        return -1;
    }
    return rc;
}

int fileio_imm(struct fioparms * fio, 
    char * filename, FILE * f, unsigned char * buf, int size) 
{
    int rc = 0;
    char * e, * p;
    unsigned long b;
    int loc;
switch (fio->op) {
    case FIO_READ:
        loc = 0;
        p = strtok(filename, " ");
        while (p != NULL && loc < size) {
            b = strtoul(p, &e, 0);
            if (*e != 0) {
                fprintf(stderr, "%s: invalid byte value (%s) specified for immediate mode\n", progname, p);
                return -1;
            }
            buf[loc++] = b;
            p = strtok(NULL, " ");
        }
        rc = loc;
        break;
    default:
        fprintf(stderr, "%s: fileio: invalid operation=%d\n", progname, fio->op);
        return -1;
    }
    if (rc < 0 || (fio->op == FIO_WRITE && rc < size)) {
        fprintf(stderr, "%s: %s error %s %s: %s; %s %d of the expected %d bytes\n", progname, fio->iodesc, fio->dir, filename, strerror(errno), fio->rw, rc, size);
        return -1;
    }
    return rc;
}

int fileio_ihex(struct fioparms * fio,
    char * filename, FILE * f, unsigned char * buf, int size)
{
    int rc;
    switch (fio->op) {
    case FIO_WRITE:
        rc = b2ihex(buf, size, 32, 0, filename, f);
        if (rc < 0) {
            return -1;
        }
        break;
    case FIO_READ:
        rc = ihex2b(filename, f, buf, size);
        if (rc < 0) {
            return -1;
        }
        break;
    default:
        fprintf(stderr, "%s: invalid Intex Hex file I/O operation=%d\n", progname, fio->op);
        return -1;
        break;
    }
    return rc;
}

int fileio_srec(struct fioparms * fio,
    char * filename, FILE * f, unsigned char * buf, int size)
int rc;

switch (fio->op) {
  case FIO_WRITE:
    rc = b2srec(buf, size, 32, 0, filename, f);
    if (rc < 0) {
      return -1;
    }
    break;

  case FIO_READ:
    rc = srec2b(filename, f, buf, size);
    if (rc < 0)
      return -1;
    break;

  default:
    fprintf(stderr, "%s: ERROR: invalid Motorola S-Records file I/O 
                      operation=%d\n",
            progname, fio->op);
    return -1;
    break;
}

return rc;
}

int fmt_autodetect(char * fname)
{
  FILE * f;
  unsigned char buf[MAX_LINE_LEN];
  int i;
  int len;
  int found;

  f = fopen(fname, "r");
  if (f == NULL) {
    fprintf(stderr, "%s: error opening %s: %s\n",
            progname, fname, strerror(errno));
    return -1;
  }

  while (fgets((char *)buf, MAX_LINE_LEN, f)!= NULL) {
    buf[MAX_LINE_LEN-1] = 0;
    len = strlen((char *)buf);
    if (buf[len-1] == '\n')
      buf[len-1] = 0;

    /* check for binary data */
    found = 0;
    for (i=0; i<len; i++) {
      if (buf[i] > 127) {
        found = 1;
        break;
      }
    }
    if (found)
      return FMT_RBIN;

    /* check for lines that look like intel hex */
    if ((buf[0] == ':') && (len >= 11)) {
      found = 1;
      for (i=1; i<len; i++) {
        if (!isxdigit(buf[i])) {
          found = 0;
          break;
        }
      }
    }
  }

  return 0;
}
if (found)
    return FMT_IHEX;
}

/* check for lines that look like motorola s-record */
if ((buf[0] == 'S') && (len >= 10) && isdigit(buf[1])) {
    found = 1;
    for (i=1; i<len; i++) {
        if (!isdigit(buf[i])) {
            found = 0;
            break;
        }
    }
    if (found)
        return FMT_SREC;
}

return -1;
}
C.2.11  lub.h

/******************************************************************************
 * PROJECT: LoonBoard Project
 */

/* COPYRIGHT: Colin O’Flynn of NewAE, (C) 2005 <coflynn@newae.com>
 */

/* LICENSE: This file is released under a license specified in the LICENSE
 * file in the main project directory that this file was in. If
 * the license file is missing, then this file is NOT licensed
 * and all standard copyright laws apply.
 */

/* FILE NAME: lub.h
 */

/* PURPOSE: Provides a number of routines to use the LoonBoard Unified
 * Bootloader (LUB) protocol to transmit and receive packets.
 */

/* GLOBAL VARIABLES:
 */

/* Variable Type Description
 -------------- ---------
 none
 */

/* DEVELOPMENT HISTORY:
 */

/* Date(DMY) Author Release Description Of Change
 ---- ------ ------- ---------------------
 30-04-2005 Colin O’Flynn Initial Initial file written
 */

*******************************************************************************/

#ifndef __LUB_HEADER
#define __LUB_HEADER

int lub_GetPacketStream ( void (* char_process )( unsigned char , unsigned int ), void (* verify_data )( void ), unsigned char maxtries );
int lub_GetPacket ( char * data , int maxsize , unsigned char maxtries );
int lub_SendPacket ( unsigned char packettype , unsigned char * data , unsigned char maxtries );

int lub_EnterLUB ( unsigned char which_lub );

#define LUB_SYNC 0xA9
#define LUB_SIGN_ON 0x00
#define LUB_ACK 0x01
#define LUB_NACK 0x02
#define LUB_SOT 0x03
#define LUB_EOT 0x04
#define LUB_SEND_FPGA 0xA0
#define LUB_SEND_AVR 0xA1
#define LUB_SEND_BOOT 0xA2
#define LUB_SEND_SAFEBOOT 0xA3
#define LUB_CRC1 0xAE
#define LUB_264_DATA 0xB0
#define LUB_SUICIDE 0xC0

#endif //__LUB_HEADER

C.2.12  lub.c

/******************************************************************************
 * PROJECT: LoonBoard Project
 */
FILE NAME: lub.c

PURPOSE: Provides a number of routines to use the LoonBoard Unified Bootloader (LUB) protocol to transmit and receive packets.

FILE REFERENCES:

EXTERNAL VARIABLES:

EXTERNAL REFERENCES:

ABNORMAL TERMINATION CONDITIONS, ERROR AND WARNING MESSAGES:

ASSUMPTIONS, CONSTRAINTS, RESTRICTIONS:

NOTES:

REQUIREMENTS/FUNCTIONAL SPECIFICATIONS REFERENCES:

DEVELOPMENT HISTORY:

Date(DMY) Author Release Description Of Change
---- ------ ------- ---------------------
30-04-2005 Colin O'Flynn Initial Initial file written
14-05-2005 Colin O'Flynn 0.1 Changes

************************************************************************

/* Include Files */
#include "serial_lowlevel.h"
#include "crc8.h"
#include "lub.h"

#ifndef NULL
#define NULL 0
#endif

static unsigned char lub_packet_counter = 1;

/* Local function prototypes */
unsigned int lub_NumberBytes (unsigned char packettype);
int lub_GetPacketStream_lowlevel (void (*)(char_process)(unsigned char, unsigned int ));
FUNCTION NAME: lub_GetPacketStream

ARGUMENTS:

ARGUMENT TYPE I/O DESCRIPTION
------- ---- --- ------------------
char_process ptr to func I On every databyte (note: data part of packet only)

verify_data ptr to func I Once the packet is received this function will be called if the packet was OK, and when this function returns it will send an ACK

maxtries unsigned char I Maximum number of times to try before giving up

RETURNS: int depending on error or success:

VALUE MEANING
--- ------------------
0 All OK, Packet verified OK
-1 Ran out of tries

Others Oh crap - something VERY wrong

*************************************************************************/

int lub_GetPacketStream(void (*char_process)(unsigned char, unsigned int), void (*verify_data)(void), unsigned char maxtries)
{
    lub_packet_counter++;

    while (maxtries != 0)
    {
        lub_packet_counter--;

        if (lub_GetPacketStream_lowlevel(char_process) == 0)
        {
            //Call user routine
            verify_data();

            //Send ACK
            lub_packet_counter--;

            lub_SendPacket_lowlevel(LUB_ACK, NULL);
            return 0;
        }
    }
}
else
{
    // send NACK
    lub_packet_counter --;
    lub_SendPacket_lowlevel(LUB_NACK, NULL);
}
maxtries --;
}
return -1;
}  

/*******************************************************************************
* FUNCTION NAME: lub_GetPacket
* ARGUMENTS:
* ARGUMENT TYPE I/O DESCRIPTION
* -------- ---- --- -----------
* data ptr to char O Incomming packet stored at this location
* maxsize int I Max number of bytes to store
* maxtries unsigned char I Max number of times to try
* RETURNS: int depending on error or success:
* VALUE MEANING
* ----- -------------------
* 0 All OK, Packet verified OK
* -1 Packet ran out of tries
* Others Oh crap - something VERY wrong
*******************************************************************************/

int lub_GetPacket ( char * data , int maxsize , unsigned char maxtries )
{
    // fprintf ( stderr ,"GetPacketStream: Attempting to get packet ");
    lub_packet_counter ++;
    while ( maxtries != 0 )
    {
        // fprintf ( stderr ,"%d ...", lub_packet_counter);
        lub_packet_counter --;
        if ( lub_GetPacket_lowlevel(data, maxsize) == 0 )
        {
            // Send ACK
            lub_packet_counter --;
            lub_SendPacket_lowlevel(LUB_ACK, NULL);
            return 0;
        }
        else
        {
            // send NACK
            lub_packet_counter --;
            lub_SendPacket_lowlevel(LUB_NACK, NULL);
        }
    }
    return -1;
}
maxtries--;  
}
return -1;
}

int lub_SendPacket ( unsigned char packettype , unsigned char * data , unsigned char maxtries )
{
    unsigned char ack_buffer[10];
    lub_packet_counter++;  

    while ( maxtries != 0 )
    {
        // lub_SendPacket will increment the counter
        lub_packet_counter--;  
        lub_SendPacket_lowlevel ( packettype , data );

        // GetPacket will increment the packet counter
        lub_packet_counter--;  
        if ( lub_GetPacket_lowlevel ( ack_buffer , 10 ) == 0 )
        {
            if ( ack_buffer[2] == LUB_ACK )
            {
                return 0;  
            }
        }
        maxtries--;  
    }
    return -1;
}
int lub_GetPacketStream_lowlevel(void (*char_process)(unsigned char, unsigned int))
{
    unsigned char recv_packet_counter;
    unsigned char recv_packettype;
    unsigned int number_data_bytes;
    unsigned int bytenumber;

    unsigned char errors = 0;
    unsigned char c;

    // Clear CRC8
    crc8_clear();

    // wait for sync
    while (serial_getc() != LUB_SYNC)
    {
        continue;
    }

    // get byte count
    recv_packet_counter = serial_getc();
    crc8_addbyte(recv_packet_counter);

    // get packet type
    recv_packettype = serial_getc();
    crc8_addbyte(recv_packettype);

    /* If the command is for the start of a packet, this must be packet #1 */
    if (recv_packettype == LUB_SOT) {
    
}
lub_packet_counter = 1;
}
else
{
    lub_packet_counter++;  
}

// Check this is the packet we are expecting
if (lub_packet_counter != recv_packet_counter)
{
    errors |= 0x80;
}

// Figure out how much data we should be getting
number_data_bytes = lub_NumberBytes( recv_packettype );

// Receive as many databytes as we should, processing
// them as instructed and then CRC8ing them
bytenumber = 0;
char_process( recv_packettype, bytenumber);

while (number_data_bytes != 0)
{
    bytenumber++;
    c = serial_getc();
    crc8_addbyte(c);
    char_process(c, bytenumber);
    number_data_bytes--;  
}

// The final step... CRC8 the CRC8 received, we should get 0
crc8_addbyte( serial_getc());

if (crc8_getcrc() != 0)
{
    errors |= 0x01;
}

if (errors == 0)
{
    return 0;
}
else
{
    return -1;
}

/******************************************************************************
* FUNCTION NAME: lub_GetPacket_lowlevel
* ARGUMENTS:
* ARGUMENT TYPE I/O DESCRIPTION
* data ptr to char O Incomming packet stored at this location
* maxsize int I Max number of bytes to store
* RETURNS: int depending on error or success:
* VALUE MEANING
* ------ ---------------
int lub_GetPacket_lowlevel (char * data, int maxsize) {
    unsigned char recv_packet_counter;
    unsigned char recv_packettype;
    unsigned int number_data_bytes;
    unsigned char errors = 0;
    unsigned char c;
    // Clear CRC8
    crc8_clear();
    // wait for sync
    while (serial_getc() != LUB_SYNC)
    {
        continue;
    }
    *data = LUB_SYNC;
    data++;
    // get byte count
    recv_packet_counter = serial_getc();
    *data = recv_packet_counter;
    data++;
    crc8_addbyte(recv_packet_counter);
    // get packet type
    recv_packettype = serial_getc();
    *data = recv_packettype;
    data++;
    crc8_addbyte(recv_packettype);
    /* If the command is for the start of a packet, this must be packet #1 */
    if (recv_packettype == LUB_SOT)
    {
        lub_packet_counter = 1;
    }
    else
    {
        lub_packet_counter++;
    }
    // Check this is the packet we are expecting
    if (lub_packet_counter != recv_packet_counter)
    {
        errors |= 0x80;
    }
    // Figure out how much data we should be getting
    number_data_bytes = lub_NumberBytes(recv_packettype);
    // Check we have enough room
    if ((number_data_bytes + 4) > maxsize)
    {
        errors |= 0x40;
        data = NULL;
    }
Receive as many databytes as we should, processing them as instructed and then CRC8ing them.
serial_getcs(data, number_data_bytes);
while (number_data_bytes != 0)
{
    c = serial_getc();
    *data = c;
    crc8_addbyte(*data);
    data++;
    number_data_bytes--; 
}

The final step... CRC8 the CRC8 received, we should get 0
 crc8_addbyte(*data = serial_getc());

if (crc8_getcrc() != 0)
{
    errors |= 0x01;
}

if (errors == 0)
{
    return 0;
}
else
{
    return -1;
}

------------------------------
 FUNCTION NAME: lub_SendPacket_lowlevel
 ARGUMENTS:
 PACKETTYPE UNDEFINED I The LUB packet type specified in LUB specs

 DATA * UNDEFINED I Should point to any data that needs to be transmitted. The size of this buffer will depend on the packet type. If the sent packet doesn’t have any data, set to NULL

 RETURNS: int depending on error or success:
 VALUE MEANING
 0 Packet Sent
 Others Oh crap - something VERY wrong

------------------------------
int lub_SendPacket_lowlevel(unsigned char packettype, unsigned char * data)
{
    unsigned int number_data_bytes;
/Clear CRC8
crc8_clear();

//Send sync
serial_putchar(LUB_SYNC);

/* If the command is for the start of a packet, this must be packet #1 */
if (packettype == LUB_SOT) {
  lub_packet_counter = 1;
} else {
  lub_packet_counter++;
}

//Send byte counter
serial_putchar(lub_packet_counter);
crc8_addbyte(lub_packet_counter);

//Send type of packet
serial_putchar(packettype);
crc8_addbyte(packettype);

//Send data bytes if there should be some
number_data_bytes = lub_NumberBytes(packettype);

/* For every data byte send it and add to CRC8 */
serial_putchars(data, number_data_bytes);
while (number_data_bytes != 0) {
  //serial_putchar(*data);
crc8_addbyte(*data);
crc8_addbyte1(*data);  //Add to cumulative CRC ONLY the data
data++;
  number_data_bytes--;  }

/* Send CRC8 */
serial_putchar(crc8_getcrc());

return 0;

/* *******************************************************
 * FUNCTION NAME: lub_EnterLUB
 * DESCRIPTION: Tries to enter the LUB
 * ARGUMENTS:
 * ARGUMENT    TYPE    I/O DESCRIPTION
 * ----------- ----- --- ---------------
 * which_lub   unsigned char I   When set to 0 loads the
 *               normal LUB, when set to
 *               1 loads the emergency
 *               LUB
 * RETURNS: int depending on error or success:
 * VALUE       MEANING
 * ----------- ------------
 * 0           LUB Entered
 */
int lub_EnterLUB(unsigned char which_lub)
{
    unsigned char lubmode; /* Which lub character to send */

    /* Wait for the ? */
    while (serial_getc() != '?')
    {
        continue;
    }

    /* Send a 's' for safe mode bootloader, and a 'b' for the normal mode bootloader */
    if (which_lub == 0)
    {
        lubmode = 'b'; /* Normal bootloader */
    }
    else if (which_lub == 3)
    {
        lubmode = 'd'; /* Debug */
    }
    else
    {
        lubmode = 's'; /* Failsafe bootloader */
    }

    serial_putchar(lubmode);

    /* Sometimes lubloader has to fake it and will send a second ? */
    /* So we check it twice... */
    if (serial_getc() == 'A')
    {
        return 0;
    }

    if (serial_getc() == 'A')
    {
        return 0;
    }

    // Send data bytes if there should be some
    if (packettype == LUB_264_DATA)
    {
        /* *******************************************************
         * FUNCTION NAME: lub_NumberBytes
         * ARGUMENTS:
         * ARGUMENT TYPE I/O DESCRIPTION
         * --------- ---- --- -----------
         * packettype unsigned char I    The LUB packet type
         * that this packet is,
         * specified in LUB specs
         * RETURNS: Unsigned int corresponding to the number of data
         * bytes that that type of packet should have attached
         * *******************************************************
         * unsigned int lub_NumberBytes(unsigned char packettype)
         {
             // Send data bytes if there should be some
             if (packettype == LUB_264_DATA)
             {
                 */
return 264;
}
else if (packettype == LUB_SIGN_ON || packettype == LUB_CRC1)
{
    return 2;
}
else
{
    return 0;
}
C.2.13  Bootloader Code

This code is compiled and placed in the boot sector of the Atmega128 microcontroller.

---

**PROJECT:** LoonBoard Project

**COPYRIGHT:** Colin O’Flynn of NewAE, (C) 2005 <coflynn@newae.com>

**LICENSE:** This file is released under a license specified in the LICENSE file in the main project directory that this file was in. If the license file is missing, then this file is NOT licensed and all standard copyright laws apply.

**FILE NAME:** tinyloader.S

**PURPOSE:** This resident bootloader is always present in the AVR. It loads the actual main bootloader into the AVR space, as well as loading the FPGA.

**ABNORMAL TERMINATION CONDITIONS, ERROR AND WARNING MESSAGES:**

**ASSUMPTIONS, CONSTRAINTS, RESTRICTIONS:**

**NOTES:**

**REQUIREMENTS/FUNCTIONAL SPECIFICATIONS REFERENCES:**

* Pin Connections:

***** LoonBoard Hardware, uses mega88 or Mega168

* PORT PIN DIR USE
  ---- --- --- ---
  * PORTC 0 OUT Data Out to FPGA
  * PORTD 0 IN RX from PC (dedicated AVR hardware)
  * PORTD 1 OUT TX to PC (dedicated AVR hardware)
  * PORTD 2 OUT Clock Out to FPGA
  * PORTD 7 OUT to PROGRAM pin on FPGA

* PORTB 3 OUT MOSI to DataFLASH (dedicated AVR hardware)
* PORTB 4 IN MISO from DataFLASH (dedicated AVR hardware)
* PORTB 5 OUT SCK to DataFLASH (dedicated AVR hardware)
* PORTB 7 OUT CS to DataFLASH

* ADC 6 IN DONE from FPGA
* ADC 7 IN INIT from FPGA

***** Yaju Hardware, uses AtMega128

* PORT PIN DIR USE
  ---- --- --- ---
  * PORTA 0 OUT Data Out to FPGA
  * PORTA 2 IN INIT from FPGA

  IN RX from PC (dedicated AVR hardware)
  OUT TX to PC (dedicated AVR hardware)

* PORTD 0 IN DONE from FPGA
* PORTD 1 OUT Clock Out to FPGA
* PORTF 0 OUT to PROGRAM pin on FPGA

* PORTB 2 OUT MOSI to DataFLASH (dedicated AVR hardware)
* PORTB 3 IN MISO from DataFLASH (dedicated AVR hardware)
* PORTB 1 OUT SCK to DataFLASH (dedicated AVR hardware)
PORTB 4 OUT CS to DataFLASH

DEVELOPMENT HISTORY:

Date (DMY) Author Release Description Of Change
--- ------ ------- ---------------------
30-04-2005 Colin O'Flynn Initial Initial file written
14-05-2005 Colin O'Flynn 0.1 Bug fixes, it works
20-05-2005 Colin O'Flynn 0.2 Optimized slightly
11-10-2005 Colin O'Flynn 0.3 Changed DataFLASH locations
31-01-2006 Colin O'Flynn 0.4 Added support for new hardware

#include <avr/io.h>

; Reuse this one, it's only used for FPGA
#define setbit0 R22
#define setbit1 R26

#define temp1 R16
#define temp2 R17
#define zero_reg R20

#define DF_highpagereg R18
#define DF_lowpagereg R19
#define DF_datareg R21

; Registers used for UART
#define UART_UDR R23

; Registers used for AVR
#define AVR_pagecountlo YL
#define AVR_pagecounthi YH

#define looplo R24
#define loophi R25
#define spmcval R26

#define checkff R22

#define fpgaload R27

#define FPGA_INIT 7
#define FPGA_DONE 6

#define PORT_I0(x) _SFR_IO_ADDR(x)
#define PORT_MEM(x) _SFR_MEM_ADDR(x)

; Get pagesize in bytes
#define PAGESIZEB SPM_PAGESIZE
#endif

#define SELFPRGEN SPMEN
#endif

#define WDTCSR WDTCR
#endif

/* This adds 10 bytes onto code, save a second or two of programming time */
#define FAST1

付き合حصول** LoonBoard Hardware 付き合حصول**
/* Read OSCCAL from Page 1*/
.macro CLOCK_SETUP
  ldi DF_highpagereg , 0x00
  ldi DF_lowpagereg , 0x01
  rcall DF_Cont_Read
  rcall DF_Send_Rec
  sts PORT_MEM(OSCCAL), DF_datareg
  CS_HIGH
.endm

/* Chip Select (CS) pin to DataFLASH*/
.macro CS_HIGH
  sbi PORT_IO(PORTB), 7
.endm

.macro CS_LOW
  cbi PORT_IO(PORTB), 7
.endm

/* Program pin to FPGA*/
.macro PGM_SETUP
  out PORT_IO(PORTD), zero_reg
.endm

.macro PGM_HIGH
  cbi PORT_IO(DDRD), 7
.endm

.macro PGM_LOW
  sbi PORT_IO(DDRD), 7
.endm

/* Init read-only pins, INIT and DONE */
.macro INIT_FPGA_PIN
  ;in order to cheat out more I/O pins, the dedicated ADC channels are used as inputs
  ; ADC with AVCC as ref, Left-Adjusted, FPGA_INIT pin as first pin
  ldi temp1, (1<<REFS0) | 1<<ADLAR | FPGA_INIT
  sts PORT_MEM(ADMUX), temp1
  ldi temp1, (1<<ADEN) | (1<<ADSC) | (1<<ADATE) | (1<<ADPS2) | (1<<ADPS0)
  sts PORT_MEM(ADCSRB), zero_reg
  ;ADC enabled, start first conversion, free-running, divide clock by 32
  sts PORT_MEM(ADCSRA), temp1
.endm

.macro WAIT_INIT_GOES_HIGH
  waiting_for_init:
  lds temp1, PORT_MEM(ADCH)
  sbrs temp1, 7 ;If bit is set, INIT is high
  rjmp waiting_for_init
.endm

/* CCLK to FPGA */
.macro CCLK_LOW
  cbi PORT_IO(PORTD), 2
.endm
.macro CCLK_HIGH
    sbi PORT_IO ( PORTD ), 2
.endm

.macro CCLK_SETUP
    sbi PORT_IO ( DDRC ), 2
.endm

/*/ Data Out (DO) to FPGA */
.macro DO_SETUP
    sbi PORT_IO ( DDRC ), 0
.endm

.macro DO_HIGH
    sbi PORT_IO ( PORTC ), 0
.endm

.macro DO_LOW
    cbi PORT_IO ( PORTC ), 0
.endm

/*/ DONE from FPGA */
.macro DONE_SETUP
    ldi temp1, FPGA_DONE | 1<<REFS0 | 1<<ADLAR
    sts PORT_MEM ( ADMUX ), temp1
.endm

.macro SKIP_IF_DONE_IS_HIGH
    lds temp1, PORT_MEM ( ADCH )
    sbrs temp1, 7
.endm

/*/ General DataFLASH setup */
.macro DATAFLASH_SETUP
    ldi temp1, 1<<7 | 1<<2 | 1<<0; ; Also red LED is on
    out PORT_IO ( PORTB ), temp1
    ldi temp1, 1<<2 | 1<<7 | 1<<3 | 1<<5 | 1<<1 | 1<<0;
    out PORT_IO ( DDRB ), temp1
.endm

/*/ Setup AVR as main program is expecting pretty much */
.macro HARDWARE_READY
    ; Set CS as input, external resistors take over
    cbi PORT_IO ( DDRB ), 7
    ; Green LED On, Red Off
    sbi PORT_IO ( PORTB ), 1
    cbi PORT_IO ( PORTB ), 0
.endm

.macro READ_SPMCSR_TO_TEMP1
    in temp1, PORT_IO ( SPMCSR )
.endm

.macro WRITE_SPMCSR
    out PORT_IO ( SPMCSR ), spmcval
.endm

#if defined ( YAJU )
    ; 130 046 Bytes per AVR device
    ; 264 bytes per DataFLASH page
    ; = 500 dataflash pages
#define BOOT_PAGE 2
#define SAFE_PAGE 502
#define AVR_PAGE 1002
#endif
#define FPGA_PAGE 1502

/* #define UART_SETTING ((((F_CPU / 115200) + 8) / 16) - 1) */

#define UART_SETTING 0x03
/* None of this */

.macro CLOCK_SETUP
.endm

/* Chip Select (CS) pin to DataFLASH */
.macro CS_HIGH
    sbi PORT_IO (PORTB), 4
.endm

.macro CS_LOW
    cbi PORT_IO (PORTB), 4
.endm

/* Program pin to FPGA */
.macro PGM_SETUP
    sts PORT_MEM (PORTF), zero_reg
.endm

.macro PGM_HIGH
    sts PORT_MEM (DDRF), zero_reg
.endm

.macro PGM_LOW
    ldi setbit0, 1
    sts PORT_MEM (DDRF), setbit0
.endm

/* Init read-only pins, INIT and DONE */
.macro INIT_FPGA_PIN
    sts PORT_MEM (DDRA), zero_reg ; Init
    sts PORT_MEM (DDRD), zero_reg ; Done
.endm

.macro WAIT_INIT_GOES_HIGH
    waiting_for_init:
        lds temp1, PORT_MEM (PINA)
        sbrs temp1, 2 ; If bit is set, INIT is high
        rjmp waiting_for_init
.endm

/* CCLK to FPGA */
.macro CCLK_LOW
    sts PORT_MEM (PORTD), zero_reg
.endm

.macro CCLK_HIGH
    sts PORT_MEM (PORTD), setbit1
.endm

.macro CCLK_SETUP
    ldi setbit1, 1<<1
    sts PORT_MEM (DDRD), setbit1
.endm

/* Data Out (DO) to FPGA */
.macro DO_SETUP
    sts PORT_MEM (PORTA), setbit0
.endm

.macro DO_HIGH
    sts PORT_MEM (PORTA), setbit0
.endm
.macro DO_LOW
  sts PORT_MEM(PORTA), zero_reg
.endm

/*DONE from FPGA*/
.macros DONE_SETUP
; Already set up
.endm

.macros SKIP_IF_DONE_IS_HIGH
  in temp1, PORT_IO(PIND)
  sbrs temp1, 0
.endm

/* General DataFLASH setup */
.macros DATAFLASH_SETUP
  ldi temp1, 1<<4; ;CS high
  out PORT_IO(PORTB), temp1
  /* SS , CS , MOSI , SCK are all outputs */
  ldi temp1, 1<<0 | 1<<4 | 1<<2 | 1<<1;
  out PORT_IO(DDRB), temp1
.endm

/* Setup AVR as main program is expecting pretty much */
.macros HARDWARE_READY
; Nothing to do!
.endm

.macros READ_SPMCSR_TO_TEMP1
  lds temp1, PORT_MEM(SPMCSR)
.endm

.macros WRITE_SPMCSR
  sts PORT_MEM(SPMCSR), spmcrval
.endm

#endif

/******************** End hardware specific part, start LUB **********************/

.section .text
.global main
main:

; There is a chance some sort of WDT was on, we need to
; disable that or we are screwed

; Reset Watchdog Timer
  wdr
  ; Clear WDRF in MCUSR
  in temp1, PORT_IO(MCUSR)
  andi temp1, (0xff & (0<<WDRF))
  out PORT_IO(MCUSR), temp1
  ; Write logical one to WDCE and WDE
  ; Keep old prescaler setting to prevent unintentional time-out
  lds temp1, PORT_MEM(WDTCSR)
  orl temp1, (1<<WDCE) | (1<<WDE)
  sts PORT_MEM(WDTCSR), temp1

; Turn off WDT
  ldi temp1, (0<<WDE)
  sts PORT_MEM(WDTCSR), temp1
ldi temp1, hi8(RAMEND) ; Stack-pointer needs setup
out PORT_ID(SPH), temp1
ldi temp1, lo8(RAMEND)
out PORT_ID(SPL), temp1
clr zero_reg ; Zero-reg is always cleared

; Get rid of interrupts
cli

; Enable receiver and transmitter
ldi temp1, (1<<RXEN0) | (1<<TXEN0)
sts PORT_MEM(UCSR0B), temp1

; Get that baudrate right
; XTAL BAUD UBRR
; 6.75 MHz 57600 6
; 7.37 MHz 115200 3
; 7.37 MHz 57600 7
ldi temp1, UART_SETTING
sts PORT_MEM(UBRR0L), temp1
sts PORT_MEM(UBRR0H), zero_reg

PGM_SETUP ; Program is low behind-the-scenes

; Set SS, MOSI, SCK, and the real CS connected to our dataflash's outputs
DATAFLASH_SETUP

; SPI Interface active
ldi temp1, 1<<SPE | 1<<MSTR | 1<<CPHA | 1<<CPOL
out PORT_ID(SPCR), temp1

; Have to wait for DataFLASH to come up
rcall delay2

; Read OSCCAL, address 0 on page 1 if we use it
CLOCK_SETUP

; Make sure oscillator is stable
rcall delay2

; UART Debugging routine
; aoe:
; lds temp1, UCSR0A
; sbrs temp1, RXCO
; rjmp aoe

; lds UART_UDR, UDRO
; rcall UART_Transmit
; rjmp aoe

; Don't load FPGA by default
clr fpgaload

;*** Startup - Try communicating with computer
; Send sync byte of ?
ldi UART_UDR, '?'
rcall UART_Transmit

; Check for a return character, wait a bit first
rcall delay2

lds UART_UDR, PORT_MEM(UDRO)

;a b means load bootloader code
cpi UART_UDR, 'b'
breq STARTUP_BOOTLOADER
; a $ means safemode bootloader code
cpi UART_UDR, 's'
breq STARTUP_SAFEMODE

; a l means load the normal AVR code
cpi UART_UDR, 'l'
breq STARTUP_NORMAL

;d is for debugging
; cpi UART_UDR, 'd'
; breq STARTUP_DEBUG

; Otherwise we are starting up in normal mode, jump to FPGA loading
rjmp FPGA_Load
rjmp STARTUP_NORMAL

STARTUP_BOOTLOADER:
; Setup location of AVR bootloader File (0x002)
ldi DF_highpagereg, hi8 (BOOT_PAGE)
ldi DF_lowpagereg, lo8 (BOOT_PAGE)
rjmp AVR_Load

STARTUP_SAFEMODE:
; Setup location of AVR Safemode bootloader file (0x042)
ldi DF_highpagereg, hi8 (SAFE_PAGE)
ldi DF_lowpagereg, lo8 (SAFE_PAGE)
rjmp AVR_Load

; STARTUP_DEBUG:
; rjmp Stuck

STARTUP_NORMAL:
; Setup location of AVR File (0x082)
ldi DF_highpagereg, hi8 (AVR_PAGE)
ldi DF_lowpagereg, lo8 (AVR_PAGE)
ldi fpgaload, 0xFF

;******************************************************************** AVR Loading Section

AVR_Load:
; How many pages to write
; = 128 - Pages used for this bootloader
; 32 pages in max bootloader section
; One page = PAGESIZEB in bytes (32 words/64 bytes for Mega 88, 64 words/128 bytes for Mega168)

ldi AVR_pagecountlo, lo8 (((FLASHEND + 1) / PAGESIZEB) - ((FLASHEND - BOOTSTART + 1) / PAGESIZEB))
ldi AVR_pagecounthi, hi8 (((FLASHEND + 1) / PAGESIZEB) - ((FLASHEND - BOOTSTART + 1) / PAGESIZEB))

; Load Z Pointer with 0

#ifdef RAMPZ
sts PORT_MEM (RAMPZ), zero_reg
#endif

clr ZL
clr ZH

; Enable read from DF
rcall DF_Cont_Read

; Now until we have written every page we go
Page_Write_Loop:
; Write the current page
rcall Write_page
; Select next page, starts at bit 6 for page
#if PAGESIZEB == 64
  adiw ZL, 63
  adiw ZL, 1
#elif PAGESIZEB == 128
  subi ZL, -128
  sbci ZH, -1
#elif PAGESIZEB == 256
  subi ZL, -255
  sbci ZH, -1
  adiw ZL, 1
#else
  #error "Undefined PAGESIZEB"
#endif

/* Check for RAMPZ support */
#ifdef RAMPZ
  brne norampz
  ldi temp1, 1
  sts PORT_MEM(RAMPZ), temp1
norampz:
#endif

; Count pages read
sbiw AVR_pagecountlo, 1
BRNE Page_Write_Loop

; Disable continuous read
CS_HIGH

; If this was a routine AVR load we also do want to load the FPGA
sbrs fpgaload, 1
rjmp Done

;**********************************************************************
; FPGA Loading Section
;**********************************************************************

; Load the FPGA
FPGA_Load:

; Setup FPGA pins as inputs or whatever
INIT_FPGA_PIN

; First step of programming algorithm for FPGA - pulse /PROGRAM low
PGM_LOW
nop
nop
PGM_HIGH

; Wait for INIT to go high
WAIT_INIT_GOES_HIGH

; Clear CCLK
CCLK_LOW

; Wait about 100 uS
rcall delay

; Set D0 and CCLK as outputs
CCLK_SETUP
D0_SETUP
; Set DONE pin as being read
DONE_SETUP

; Setup location of FPGA File (0x0200)
ldi DF_highpagereg, hi8(FPGA_PAGE)
ldi DF_lowpagereg, lo8(FPGA_PAGE)

; Enable read from DF
rcall DF_Cont_Read

; First three bytes are 24-bit size of bitstream, to be stored in specified values
rcall DF_Send_Rec
rcall DF_Send_Rec
rcall DF_Send_Rec

; Now this part is easy - pretty much just grab a bit from the
; dataflash and put it into the FPGA, since everything is done serially
;
; In this case we do 8 bits at a time so we can use the hardware SPI interface. It
; makes using something
; beside DataFLASH easier. It works out as the same code size and about the same
; speed as doing it software
; grab a bit and throw it out the other pin.
;
; You could physically connect the CCLK line on the FPGA to the SCK line on the
; DataFLASH, and
; the MISO pin on the DataFLASH and the DI pin on the FPGA if you wanted, which
; would be faster and
; smaller
;
; Done pin goes high if config OK, or INIT pin goes low if init failed

waiting_for_done:
rcall DF_Send_Rec
rcall FPGA_Send_Byte

SKIP_IF_DONE_IS_HIGH
rjmp waiting_for_done

CS_HIGH ; Disable continuous read

******* Cleaning Up *******************************************************

Done:

/* Clean up hardware */

HARDWARE_READY

; Disable DF registers
out PORT_IO (SPCR), zero_reg

; Jump home to main application
Jump_home:
    clr ZL
    clr ZH
    ijmp

Stuck:
; Green LED On
    sbi PORT_IO (PORTB), 0
    rjmp Stuck

;************************ DataFLASH Subroutines *******************************

; Pulses CS Pin
DF_Pulse_CS:
    CS_HIGH
    nop
CS_LOW
reti

; Waits for transmission from SPI - from Datasheet
DF_Wait_Transmit: ; Wait for transmission complete
    in temp1, PORT_IO(SPSR)
    sb rs temp1, SPIF
rjmp DF_Wait_Transmit
reti

DF_Send_Rec:
    out PORT_IO(SPDR), DF_datareg ; put data out
    rcall DF_Wait_Transmit ; Wait for trans complete
    in DF_datareg, PORT_IO(SPDR) ; Read data in
    reti ; Return

DF_Cont_Read:
    rcall DF_Pulse_CS
    ldi DF_datareg, 0x68 ; continuous read
    rcall DF_Send_Rec

    ; Need to send address bytes: 0 0 0 0 A10 A9 A8 A7 then A6 A5 A4 A3 A2 A1 0
    mov temp1, DF_highpagereg
    mov temp2, DF_lowpagereg
    lsl temp2 ; Lower is shifted once, upper bit into carry
    ROL temp1 ; Upper is rolled once, carry goes into lower bit
    mov DF_datareg, temp1
    rcall DF_Send_Rec
    mov DF_datareg, temp2
    rcall DF_Send_Rec

    ; Send 5 blank bytes
    mov DF_datareg, zero_reg
    rcall DF_Send_Rec
    rcall DF_Send_Rec
    rcall DF_Send_Rec
    rcall DF_Send_Rec
reti

;****************************** Subroutines for AVR Loading ******************************
; Write one page from DataFLASH to AVR
; DataFLASH must have already been enabled in constant read mode at the proper address
; Z-Pointer must point to AVR FLASH location to write to

Write_page:
    ; Page Erase
    ldi spmcval, (1<<PGERS) | (1<<SELFPRGEN)
    rcall Do_spm
    ; re-enable the RWW section
    ldi spmcval, (1<<RWWSRE) | (1<<SELFPRGEN)
    rcall Do_spm
    ; transfer data from DF to Flash page buffer
    ldi looplo, lo8(PAGESIZEB); Init loop variable
    ldi loophi, hi8(PAGESIZEB); not required for PAGESIZEB<=256
#ifdef FAST1
    ldi checkff, 0xff
#endif
Wrloop:
    ; Get data from DataFLASH for program
    rcall DF_Send_Rec
mov R0, DF_datareg
rcall DF_Send_Rec
mov R1, DF_datareg

#elifdef FAST1
  ;We only want to write data if it isn't all FF's. We check this here:
  and checkff, R0
  and checkff, R1
#endif

ldi spmcrval, (1<<SELFPRGEN)
rcall Do_spm

adiw ZL, 2
sbiw looplo, 2 ;use subi for PAGESIZEB<=256
brne Wrloop

; execute Page Write
subi ZL, lo8(PAGESIZEB);restore pointer
sbc ZH, hi8(PAGESIZEB);not required for PAGESIZEB<=256

#elifdef FAST1
  ;Check for FF
  com checkff
  ;Screw writing if we are done
  breq Return
#endif

ldi spmcrval, (1<<PGWRT) | (1<<SELFPRGEN)
rcall Do_spm

Return:

READ_SPMCSR_TO_TEMP1

sbrs temp1, RWWSB ; If RWWSB is set, the RWW section is not ready yet
ret
; re-enable the RWW section
ldi spmcrval, (1<<RWWSRE) | (1<<SELFPRGEN)
rcall Do_spm
rjmp Return

Do_spm:
; check for previous SPM complete
; No EEPROM Bootloading done here so don't worry
; about the EEPROM
Wait_spm:
  READ_SPMCSR_TO_TEMP1
  sbrc temp1, SELFPRGEN
  rjmp Wait_spm
WRITE_SPMCSR
sm
ret

;************************* Subroutines for FPGA Loading ****************************

;Use the CLK and DO pins to send a databyte to FPGA
FPGA_Send_Byte:
  ldi temp1, 0x08

FPGA_Send_Byte_loop:
  CCLK_LOW
  lsl DF_datareg ;Shift out MSb to Carry Flag
  DO_LOW ;Clear data bit
brcc FPGA_Send_Byte_sendlow ; Check if carry is low

DO_HIGH ; Nope - set that bit high

FPGA_Send_Byte_sendlow:

CCLK_HIGH ; Clock is high
dec temp1
brbc 1, FPGA_Send_Byte_loop ; Check to see if this is bit 0
reti

; ****************** Subroutines for UART ************************************************

UART_Transmit:

; Wait for empty transmit buffer
lds temp1, PORT_MEM (UCSR0A)
sbrs temp1, UDRE0
rjmp UART_Transmit

; Put data (r16) into buffer, sends the data
sts PORT_MEM (UDR0), UART_UDR
ret

; ****************** Delay's *********************************************************

; Need a proper delay in FPGA routine of about 100 uS
; Delays 4 cycles per loop, does 256 loops = 1024 cycle delay
; WILL HAVE TO MODIFY THIS FOR YOUR CLOCK FREQUENCY
delay:

; subi zero_reg, 110 ; Modify this to make the delay the proper number of cycles or
; Comment out if not needed.
delay1:
	nop
dec zero_reg ; Dec zero_reg
brbc 1, delay1 ; If zero flag is clear, we branch
reti

; Just a longer delay
delay2:
clr temp1
delay_start:
rcall delay
dec temp1 ; Dec temp1
brbc SREG_Z, delay_start ; If zero flag is clear, we branch
ret
The code is compiled and implemented on the Atmega128 microcontroller.

C.2.14 main.c

#include <avr/io.h>
#include <avr/boot.h>
#include <avr/pgmspace.h>
#include <avr/wdt.h>
#include <stdlib.h>
#include "led_control.h"

#ifndef F_CPU
#error "CPU Frequency Undefined"
#endif

#define ubrr (unsigned int)(((F_CPU / BAUDRATE) + 8L) / 16L) - 1L

#ifdef LOONBOARD

134
# define BOOT_PAGE 0x0002
# define SAFE_PAGE 0x0042
# define AVR_PAGE 0x0082
# define FPGA_PAGE 0x0200

#elif defined YAJU
# define BOOT_PAGE 2
# define SAFE_PAGE 502
# define AVR_PAGE 1002
# define FPGA_PAGE 1502

// Doesn't have LEDs

/* # undef LED_RED
# undef LED_GREEN
# undef LED_INIT */

#else
#error "----------- Undefined hardware!-------------"
#endif

#include "lub.h"
#include "uart.h"
#include "serial_lowlevel.h"
#include "dataflash.h"
#include "crc8.h"
#include "avr/signal.h"
#include "avr/interrupt.h"

static unsigned int DF_buffernum = 1;
static unsigned int DF_page = 0;
static unsigned char Global_packettype;
static unsigned char Write_flash = 0;
static unsigned int CRCStartPage = 0;

void ProcessLubData(unsigned char data, unsigned int bytenum);
void FlashData(void);
void DealWithCRC(void);
void CommitSuicide(void);

int main(void)
{
    unsigned char buffer[10];
    sei();

    /* Enable LEDs */
    LED_INIT();
    LED_RED(ON);
    LED_GREEN(ON);

    // Enable SPI for read/writing from DataFLASH memory
    DF_SPI_init();
    Read_DF_status();

    /* LoonBoard has the RTC correction */
    #ifdef LOONBOARD
    /* Setup X1226 */
    twi_init();
    */

135
/* Only do this if X1226 is responding */
if ( x1226_init () == ERROR_NONE )
{
    buffer[0] = 0x02;
    buffer[1] = 0x06;

    // Enable writing to X1226 CCR Register
    x1226_random_write(X1226_CCR, 0x3F, &buffer[0], 1);
    x1226_random_write(X1226_CCR, 0x3F, &buffer[1], 1);

    // Setup 4096 KHz clock
    DDRD &= ~(1 << 3);
    PORTD |= 1 << 3;
    buffer[0] = 1 << 4;
    x1226_random_write(X1226_CCR, 0x11, &buffer[0], 1);

    // Setup clock to 7372800 Hz
    CalOsc();
}

// Store new OSCCAL value
Buffer_Write_Byte(1, 0, OSCCAL);
Buffer_Write_Byte(1, 1, ~OSCCAL); // Stores inverse
Buffer_To_Page(1, 1);
while(!(Read_DF_status() & 0x80));

#if defined (__AVR_ATmega88__)  
buffer[0] = 0x03;
#elif defined (__AVR_ATmega168__)  
buffer[0] = 0x04;
#elif defined (__AVR_ATmega128__)  
buffer[0] = 0x05;
#else
    #error "Unsupported target"
#endif

buffer[1] = 0x01;

/* Send sign-on message */
lub_SendPacket(LUB_SIGN_ON, buffer, 9);

/* Now until we get EOT start processing */
/* Start writing to DF on first page */
DF_buffernum = 1;
do {
    lub_GetPacketStream(ProcessLubData, FlashData, 9);
    //Requested CRC16
    if (Global_packettype == LUB_CRC1) {
        DealWithCRC();
    }
    if (Global_packettype == LUB_SUICIDE) {
        CommitSuicide();
    }
} while (Global_packettype != LUB_EOT);

/* AVR: I can’t deal with the pain of it anymore
   DOC: Please no, you can just jump to the bootloader, there’s no
   need to reset
   AVR: It’s not worth it... here! */
wdt_enable(WDTO_15MS);

/* DOC: Oh no, please reset your WDT, you still have too much
   to live for. */
while (1 == 1) {
    continue;
/* BROTHER: Dibs on his RAM
   DOC: <Runs off-stage right crying> */
}
return 1;

/* Thus routine processes each byte as it comes in */
void ProcessLubData(unsigned char data, unsigned int bytenum) {
    Write_flash = 0;
    if (bytenum == 0) {
        //Don’t write this to FLASH
        Write_flash = 0;
        Global_packettype = data;
    }
    if (Global_packettype == LUB_SEND_BOOT) {
        DF_page = BOOT_PAGE;
        CRCStartPage = DF_page;
    }
    if (Global_packettype == LUBSEND_SAFEBOOT) {
        DF_page = SAFE_PAGE;
        CRCStartPage = DF_page;
    }
    if (Global_packettype == LUBSEND_AVR) {
    }
DF_page = AVR_PAGE;
CRCStartPage = DF_page;
}

if (Global_packettype == LUB_SEND_FPGA)
{
    DF_page = FPGA_PAGE;
    CRCStartPage = DF_page;
}
else if ((Global_packettype == LUB_264_DATA) && (bytenum == 1))
{
    // Enable
    Buffer_Write_Enable(DF_buffernum, 0);
    // Go!
    DF_SPI_RW(data);
}
else if ((Global_packettype == LUB_264_DATA) && (bytenum == 264))
{
    DF_SPI_RW(data);
    // CS Inactive to end stream mode
    DF_CS_inactive;
    // We need to write this to FLASH
    Write_flash = 1;
}
else if (Global_packettype == LUB_264_DATA)
{
    DF_SPI_RW(data);
}

void FlashData(void)
{
    if (Write_flash == 1)
    {
        // Wait for last write to go through
        while(!(Read_DF_status() & 0x80));
        Buffer_To_Page(DF_buffernum, DF_page);
        // Select next page
        DF_page++;
        // Select other buffer that is current being written
        if (DF_buffernum == 1)
            DF_buffernum = 2;
        else
            DF_buffernum = 1;
    }
    Write_flash = 0;
}
}

void DealWithCRC(void)
{
    unsigned int intcounter;
    unsigned int pagecounter;
    unsigned char buffer[10];
    // Wait for last write to go through
    while(!(Read_DF_status() & 0x80));
    Cont_Flash_Read_Enable(CRCStartPage, 0x00);
crc8_clear();
intcounter = 0;
pagecounter = CRCStartPage;

while (pagecounter != DF_page)
{
crc8_addbyte(DF_SPI_RW(0x00));
intcounter++;
if (intcounter == 264)
{
    intcounter = 0;
    pagecounter++;
}
}

DF_CS_inactive;

/* CRC */
buffer[0] = crc8_getcrc();
buffer[1] = 0x00;

/* Send CRC message */
lub_SendPacket(LUB_CRC1, buffer, 9);

return;
}

/* Commits suicide, aka erases DataFLASH */
void CommitSuicide(void)
{
    unsigned char block = 0;
    Erase_Block(block);

    /* Kill off the DataFLASH */
do
    {
        block++;
        Erase_Block(block);
    }while(block != 0xFF);
}
C.2.15 calibrateosc.h

void CalOsc(void);

C.2.16 calibrateosc.c

#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <avr/delay.h>
#include "calibrateosc.h"

volatile static unsigned int actual_count;

void CalOsc(void)
{
    unsigned char sreg_backup;
    unsigned int target_count;

    // Save the SREG
    sreg_backup = SREG;

    // Enable int 1 on rising edge
    EIMSK |= 1<<INT1;
    EICRA |= 1<<ISC11 | 1<<ISC10;

    // Enable interrupts
    sei();

    // Enable timer from system clock
    TCNT1 = 0;
    TCCR1B = 0x01;

    // What we want as the counter value
    // This should be done at compile time
    // target_count = target / reference - 10;
    target_count = 1805;

    while(1==1)
    {
        // Let it take a few readings
        _delay_loop_2(6000);

        // Clock is running too fast
        if (actual_count > (target_count + 5))
        {
            // Bit 7 selects low or high range
            // We only need low
            OSCCAL = (OSCCAL - 1) & ~(1<<7);
        }

        // Clock is running too slow
        else if (actual_count < (target_count - 5))
        {
            // Bit 7 selects low or high range
            // We only need low
            OSCCAL = (OSCCAL + 1) & ~(1<<7);
        }

        // Clock is just right
        else
        {
            break;
        }
    }

    // OSCCAL = 0x00;
    EIMSK &= ~(1<<INT1);
TCCR1B = 0x00;
SREG = sreg_backup;
return;
}

SIGNAL(SIG_INTERRUPT1)
{
  //Stop
  TCCR1B = 0x00;

  //Record
  actual_count = TCNT1;

  //Go!
  TCNT1 = 0;
  TCCR1B = 0x01;
}
C.2.17  crc8.h

#include "crc8.h"

static unsigned char crc;

void crc8_clear(void) {
    crc = 0;
}

void crc8_addbyte(unsigned char data) {
    unsigned char bit_counter;
    unsigned char feedback_bit;
    bit_counter = 8;
    do {
        feedback_bit = (crc ^ data) & 0x01;
        if (feedback_bit == 0x01) {
            crc = crc ^ 0x18;  // 0X18 = X^8+X^5+X^4+X^0
        }
        crc = (crc >> 1) & 0x7F;
        if (feedback_bit == 0x01)
            break;
    } while (--bit_counter);
}

unsigned char crc8_getcrc(void) {
    return crc;
}

C.2.18  crc8.c
{  
crc = crc | 0x80;
}

data = data >> 1;
bit_counter--;  
}while (bit_counter > 0);

unsigned char crc8_getcrc (void)  
{
  return crc;
}
C.2.19  lub.h

#define __LUB_HEADER // __LUB_HEADER

int lub_GetPacketStream (void (* char_process )(unsigned char , unsigned int ), void (* verify_data )( void ), unsigned char maxtries );
int lub_GetPacket (char * data , int maxsize , unsigned char maxtries );
int lub_SendPacket (unsigned char packettype , unsigned char * data , unsigned char maxtries );
int lub_EnterLUB (unsigned char which_lub );

#define LUB_SYNC 0xA9
#define LUB_SIGN_ON 0x00
#define LUB_ACK 0x01
#define LUB_NACK 0x02
#define LUB_SOT 0x03
#define LUB_EOT 0x04
#define LUB_SEND_FPGA 0xA0
#define LUB_SEND_AVR 0xA1
#define LUB_SEND_BOOT 0xA2
#define LUB_SEND_SAFEBOOT 0xA3
#define LUB_CRC1 0xAE
#define LUB_264_DATA 0xB0
#define LUB_SUICIDE 0xC0

C.2.20  lub.c

#define __LUB_HEADER // __LUB_HEADER

int lub_GetPacketStream (void (* char_process )(unsigned char , unsigned int ), void (* verify_data )( void ), unsigned char maxtries );
int lub_GetPacket (char * data , int maxsize , unsigned char maxtries );
int lub_SendPacket (unsigned char packettype , unsigned char * data , unsigned char maxtries );
int lub_EnterLUB (unsigned char which_lub );

#define LUB_SYNC 0xA9
#define LUB_SIGN_ON 0x00
#define LUB_ACK 0x01
#define LUB_NACK 0x02
#define LUB_SOT 0x03
#define LUB_EOT 0x04
#define LUB_SEND_FPGA 0xA0
#define LUB_SEND_AVR 0xA1
#define LUB_SEND_BOOT 0xA2
#define LUB_SEND_SAFEBOOT 0xA3
#define LUB_CRC1 0xAE
#define LUB_264_DATA 0xB0
#define LUB_SUICIDE 0xC0

endef // __LUB_HEADER
* PURPOSE: Provides a number of routines to use the LoonBoard Unified Bootloader (LUB) protocol to transmit and receive packets.

* FILE REFERENCES:

* EXTERNAL VARIABLES:

* EXTERNAL REFERENCES:

* ABNORMAL TERMINATION CONDITIONS, ERROR AND WARNING MESSAGES:

* ASSUMPTIONS, CONSTRAINTS, RESTRICTIONS:

* NOTES:

* REQUIREMENTS/FUNCTIONAL SPECIFICATIONS REFERENCES:

* DEVELOPMENT HISTORY:

* Date(DMY) | Author | Release | Description Of Change
* --- | --- | --- | ---------------------
* 30-04-2005 | Colin O'Flynn | Initial | Initial file written
* 14-05-2005 | Colin O'Flynn | 0.1 | Changes

****************************************************************** */

#include "serial_lowlevel.h"
#include "crc8.h"
#include "lub.h"

#ifndef NULL
#define NULL 0
#endif

static unsigned char lub_packet_counter = 1;

/* Local function prototypes */
unsigned int lub_NumberBytes (unsigned char packettype);
int lub_GetPacketStream_lowlevel (void (* char_process )(unsigned char, unsigned int));

145
FUNCTION NAME: lub_GetPacketStream

ARGUMENTS:

ARGUMENT TYPE I/O DESCRIPTION
---------- ----- --- ---------------------
char_process ptr to func I On every databyte (note: data part of packet only)
* received this function will * be called with two arguments:
* the first is the current * databyte, and the second * is the databyte number.
* When the databyte number is * zero, the data held is the * packet type
* an example: * void function_name ( * unsigned char data, * unsigned int bytenum);
* verify_data ptr to func I Once the packet is received * this function will be called * if the packet was OK, and when * this function returns it will * send an ACK
* maxtries unsigned char I Maximum number of times to try * before giving up

RETURNS: int depending on error or success:

VALUE MEANING
------- ----------------------
0 All OK, Packet verified OK
-1 Ran out of tries
Others Oh crap - something VERY wrong

******************************************************************************/

int lub_GetPacketStream(void (*char_process)(unsigned char, unsigned int), void (* verify_data)(void), unsigned char maxtries)
{
    lub_packet_counter ++;
    while (maxtries != 0)
    {
        lub_packet_counter --;
        if (lub_GetPacketStream_lowlevel(char_process) == 0)
        {
            //Call user routine
            verify_data();
            //Send ACK
            lub_packet_counter --;
            lub_SendPacket_lowlevel(LUB_ACK, NULL);
            return 0;
        }
    }
}
```c
else
{
    // send NACK
    lub_packet_counter--;
    lub_SendPacket_lowlevel(LUB_NACK, NULL);
}
maxtries--;
}
return -1;
}

/*******************************************************************************/
*  FUNCTION NAME:  lub_GetPacket
*  ARGUMENTS:
*  ARGUMENT TYPE I/O DESCRIPTION
*  ------ ---- ---  -----------
*  data ptr to char  O Incomming packet stored
*  at this location
*  maxsize int  I Max number of bytes to store
*  maxtries unsigned char I Max number of times to try
*  RETURNS:  int depending on error or success:
*  VALUE MEANING
*  ----- ---------------
*  0 All OK, Packet verified OK
*  -1 Packet ran out of tries
*  Others Oh crap - something VERY wrong
*******************************************************************************/

int lub_GetPacket (char * data, int maxsize, unsigned char maxtries)
{
    // fprintf(stderr,"GetPacketStream: Attempting to get packet ");
    lub_packet_counter++;
    while (maxtries != 0)
    {
        // fprintf(stderr,"%d...", lub_packet_counter);
        lub_packet_counter--;
        if (lub_GetPacket_lowlevel(data, maxsize) == 0)
        {
            // Send ACK
            lub_packet_counter--;
            lub_SendPacket_lowlevel(LUB_ACK, NULL);
            return 0;
        }
        else
        {
            // send NACK
            lub_packet_counter--;
            lub_SendPacket_lowlevel(LUB_NACK, NULL);
        }
    }
    return -1;
}
```
maxtries--;
}

return -1;
}

int lub_SendPacket ( unsigned char packettype , unsigned char * data , unsigned char maxtries )
{
  unsigned char ack_buffer[10];
  lub_packet_counter++;

  while ( maxtries != 0 )
  {
    // lub_SendPacket will increment the counter
    lub_packet_counter--;
    lub_SendPacket_lowlevel ( packettype , data );

    // GetPacket will increment the packet counter
    lub_packet_counter--;
    if ( lub_GetPacket_lowlevel ( ack_buffer , 10 ) == 0 )
    {
      if ( ack_buffer[2] == LUB_ACK )
      {
        return 0;
      }
    }
    maxtries--;
  }

  return -1;
}
FUNCTION NAME: lub_GetPacketStream_lowlevel

ARGUMENTS:

ARGUMENT TYPE I/O DESCRIPTION
-------- ---- --- -----------
char_process ptr to func I On every databyte (note:
data part of packet only)

ARGUMENT TYPE I/O DESCRIPTION
-------- ---- --- -----------
number_data_bytes I
bytenumber I
errors = 0;
char c;

//Clear CRC8
crc8_clear();

//wait for sync
while (serial_getc() != LUB_SYNC)
{
    continue;
}

//get byte count
recv_packet_counter = serial_getc();
crc8_addbyte(recv_packet_counter);

//get packet type
recv_packettype = serial_getc();
crc8_addbyte(recv_packettype);

/* If the command is for the start of a packet, this must be packet #1 */
if (recv_packettype == LUB_SOT)
{

}
lub_packet_counter = 1;
}
else
{
    lub_packet_counter++;
}

// Check this is the packet we are expecting
if (lub_packet_counter != recv_packet_counter)
{
    errors |= 0x80;
}

// Figure out how much data we should be getting
number_data_bytes = lub_NumberBytes(recv_packettype);

// Receive as many databytes as we should, processing them as instructed and then CRC8ing them
bytenumber = 0;
char_process(recv_packettype, bytenumber);

while (number_data_bytes != 0)
{
    bytenumber++;
    c = serial_getc();
    crc8_addbyte(c);
    char_process(c, bytenumber);
    number_data_bytes--;
}

// The final step... CRC8 the CRC8 received, we should get 0
if (crc8_getcrc() != 0)
{
    errors |= 0x01;
}

if (errors == 0)
{
    return 0;
}
else
{
    return -1;
}
int lub_GetPacket_lowlevel(char * data, int maxsize)
{
    unsigned char recv_packet_counter;
    unsigned char recv_packettype;
    unsigned int number_data_bytes;
    unsigned char errors = 0;
    unsigned char c;

    // Clear CRC
    crc8_clear();

    // wait for sync
    while (serial_getc() != LUB_SYNC)
    {
        continue;
    }
    *data = LUB_SYNC;
    data++;

    //get byte count
    recv_packet_counter = serial_getc();
    *data = recv_packet_counter;
    data++;
    crc8_addbyte(recv_packet_counter);

    //get packet type
    recv_packettype = serial_getc();
    *data = recv_packettype;
    data++;
    crc8_addbyte(recv_packettype);

    /* If the command is for the start of a packet, this must be packet #1 */
    if (recv_packettype == LUB_SOT)
    {
        lub_packet_counter = 1;
    }
    else
    {
        lub_packet_counter++;
    }

    // Check this is the packet we are expecting
    if (lub_packet_counter != recv_packet_counter)
    {
        errors |= 0x80;
    }

    //Figure out how much data we should be getting
    number_data_bytes = lub_NumberBytes(recv_packettype);

    // Check we have enough room
    if ((number_data_bytes + 4) > maxsize)
    {
        errors |= 0x40;
        data = NULL;
    }
}
void serial_getcs(uint8_t *data, uint8_t number_data_bytes);

while (number_data_bytes != 0) {
    c = serial_getc();
    *data = c;
    crc8_addbyte(*data);
    data++;
    number_data_bytes--;
}

// The final step.. CRC8 the CRC8 received, we should get 0
    crc8_addbyte(*data = serial_getc());

    if (crc8_getcrc() != 0) {
        errors |= 0x01;
    }

    if (errors == 0) {
        return 0;
    } else {
        return -1;
    }

/******************************************************************************
* FUNCTION NAME: lub_SendPacket_lowlevel
* ARGUMENTS:
*        packettype unsigned char I The LUB packet type that this packet is,
*        specified in LUB specs
*        data unsigned char I Should point to any data that needs to be
*        transmitted. The size of this buffer will depend on the packet type. If
*        the sent packet doesn't have any data, set to NULL
* RETURNS: int depending on error or success:
* VALUE MEANING
* 0 Packet Sent
* Others Oh crap - something VERY wrong
*******************************************************************************/
int lub_SendPacket_lowlevel(unsigned char packettype, unsigned char *data) {
    unsigned int number_data_bytes;
// Clear CRC8
crc8_clear();

// Send sync
serial_putchar(LUB_SYNC);

/* If the command is for the start of a packet, this must be packet #1 */
if (packettype == LUB_SOT)
{
lub_packet_counter = 1;
}
else
{
lub_packet_counter++;
}

// Send byte counter
serial_putchar(lub_packet_counter);
crc8_addbyte(lub_packet_counter);

// Send type of packet
serial_putchar(packettype);
crc8_addbyte(packettype);

// Send data bytes if there should be some
number_data_bytes = lub_NumberBytes(packettype);

/* For every data byte send it and add to CRC8 */
serial_putchars(data, number_data_bytes);
while (number_data_bytes != 0)
{
    // serial_putchar(*data);
crc8_addbyte(*data);
data++;
    number_data_bytes--;
}

/* Send CRC8 */
serial_putchar(crc8_getcrc());
return 0;

/*******************************************************************************
 * FUNCTION NAME: lub_EnterLUB  
 * DESCRIPTION: Tries to enter the LUB  
 * ARGUMENTS:  
 * ARGUMENT TYPE I/O DESCRIPTION  
 * -------- ---- ---  -----------  
 * which_lub unsigned char I When set to 0 loads the  
 * normal LUB, when set to 1 loads the emergency  
 * LUB  
 *  
 * RETURNS: int depending on error or success:  
 * VALUE MEANING  
 * ----  ---  
 * 0 LUB Entered  
 * -1 LUB did not enter  
 */
int lub_EnterLUB(unsigned char which_lub)
{
    unsigned char lubmode; /* Which lub character to send */
    /* Wait for the ? */
    while (serial_getc() != '?')
    {
        continue;
    }
    /* Send a 's' for safe mode bootloader, and a 'b' for the normal mode bootloader */
    if (which_lub == 0)
        lubmode = 'b'; /* Normal bootloader */
    else if (which_lub == 3)
        lubmode = 'd'; /* Debug */
    else
        lubmode = 's'; /* Failsafe bootloader */
    serial_putc(lubmode);

    /* Sometimes lubloader has to fake it and will send a second ? */
    /* So we check it twice... */
    if (serial_getc() == 'A')
        return 0;
    if (serial_getc() == 'A')
        return 0;
    return -1;
}

unsigned int lub_NumberBytes(unsigned char packettype)
{
    // Send data bytes if there should be some
if (packettype == LUB_264_DATA)
{
    return 264;
}
else if (packettype == LUB_SIGN_ON || packettype == LUB_CRC1)
{
    return 2;
}
else
{
    return 0;
}
C.2.21 twi_comms.h

/* Copyright Colin O'Flynn (c) 2005 - coflynn@newae.com */

#ifndef __TWI_COMMS_H
#define __TWI_COMMS_H

#include "error.h"

// 50 KHz TWI Clock
#define TWCLK 50000

#define TWI_TVP5150_SLA 0xB8
#define TWI_X1226_CLOCK_SLA 0xDE
#define TWI_X1226_E2_SLA 0xAE

void twi_init(void);
void twi_stop(void);

ERROR_T twi_send(uint8_t sla, uint8_t * data, unsigned char numbytes);
ERROR_T twi_send_data(uint8_t * data, unsigned char numbytes);
ERROR_T twi_get(uint8_t sla, uint8_t * data, unsigned char numbytes);

#endif //__TWI_COMMS_H

C.2.22 twi_comms.c

/***********************/

#include <avr/io.h>
#include <compat/twi.h>
# include "twi_comms.h"
# include "error.h"

#ifndef F_CPU
#error "Error: F_CPU Undefined"
#endif

// TWI prescaler
#define TWPRESCALER 1

/* From datasheet:
TWCLK = SYSCLK/(16 + 2*(TWBR)*(TWPRESCALE))
(sysclk/twclk - 16)/(2*prescale) = twbr
So:
TWBR = (SYSCLK/TWCLK - 16) / (2*TWPRESCALE)
*/
#define TWBRval ( uint8_t ) ((( F_CPU / TWCLK ) - 16) / (2*TWPRESCALER))
#define twi_timeout_int() loop_until_bit_is_set (TWCR , TWINT)

void twi_init(void)
{
    /* Setup prescaler */
    #if TWPRESCALER == 1
        TWSR = 0;
    #elif TWPRESCALER == 4
        TWSR = 1;
    #elif TWPRESCALER == 16
        TWSR = 2;
    #elif TWPRESCALER == 64
        TWSR = 3;
    #endif //TWPRESCALER

    TWBR = TWBRval;
    // According to datasheet, if in master mode we need TWBR to be at least
    // 10 for reliable operation */
    if ( TWBR < 10 )
    {
        TWBR = 10;
    }

    TWCR = (1<<TWEN) | (1<<TWEA);
    return;
}

void twi_stop(void)
{
    TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);
    return;
}

ERROR_T twi_send(uint8_t sla , uint8_t * data , unsigned char numbytes)
{
    unsigned char currbyte;
    uint8_t status;
    //send start condition
    TWCR =(1<<TWINT) | (1<<TWSTO) | (1<<TWEN);
// Wait for TWINT Flag set. This indicates that the START condition has been transmitted
twi_timeout_int();

switch(TW_STATUS) {
    case TW_REP_START:
    case TW_START:
        break;
    default:
        return TW_STATUS + TWI_OFFSET;
}

// Load SLA_W into TWDR Register.
TWDR = sla | TW_WRITE;
TWCR = (1<<TWINT) | (1<<TWEN);
twi_timeout_int();

status = TW_STATUS;
switch(status) {
    case TW_MT_SLA_ACK:
        break;
    default:
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
        return status + TWI_OFFSET;
}

currbyte = 0;

while(currbyte < numbytes) {
    TWDR = data[currbyte];
    TWCR = (1<<TWINT) | (1<<TWEN);
twi_timeout_int();

    status = TW_STATUS;
    switch(status) {
        case TW_MT_DATA_ACK:
            break;
        default:
            TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
            return status + TWI_OFFSET;
    }

currbyte++;
}

return ERROR_NONE;
}

ERROR_T twi_send_data(uint8_t * data, unsigned char numbytes) {
    unsigned char currbyte;
    uint8_t status;
currbyte = 0;

while(currbyte < numbytes)
{
    TWDR = data[currbyte];
    TWCR = (1<<TWINT) | (1<<TWEN);
    twi_timeout_int();
    status = TW_STATUS;
    switch(status)
    {
        case TW_MT_DATA_ACK:
            break;
        default:
            TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
            return status + TWI_OFFSET;
    }
    currbyte++;
}

return ERROR_NONE;
}

ERROR_T twi_get(uint8_t sla, uint8_t * data, unsigned char numbytes)
{
    unsigned char currbyte;
    uint8_t status;

    // send start condition
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

    // Wait for TWINT Flag set. This indicates that the START condition has been transmitted
    twi_timeout_int();
    status = TW_STATUS;
    switch(status)
    {
        case TW_REP_START:
        case TW_START:
            break;
        default:
            return status + TWI_OFFSET;
    }

    // Load SLA_R into TWDR Register.
    TWDR = sla | TW_READ;
    TWCR = (1<<TWINT) | (1<<TWEN);
    twi_timeout_int();
    status = TW_STATUS;
    switch(status)
    {
        case TW_MR_SLA_ACK:
            break;
        default:
            TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
            return status + TWI_OFFSET;
    }
}
currbyte = 0;

while(currbyte < numbytes)
{
    /* If this is the last byte we NACK it, otherwise ACK it */
    if (currbyte + 1 == numbytes)
    {
        TWCR = (1<<TWINT) | (1<<TWEN);
    }
    else
    {
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    }

twi_timeout_int();

status = TW_STATUS;
switch(status)
{
    case TW_MR_DATA_ACK:
    case TW_MR_DATA_NACK:
        break;
    default:
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
        return status + TWI_OFFSET;
}

data[currbyte] = TWDR;

currbyte++;
}

return ERROR_NONE;
}
C.2.23  uart.h

#ifndef _LUB_UART_H
#define _LUB_UART_H

void uart_putc(char c);
char uart_getc(void);
void uart_puts(char *s);
void uart_putcs(char *s, unsigned int counter);
void uart_getcs(char *s, unsigned int counter);
void uart_puts_P(const char *addr);
#endif // _LUB_UART_H

C.2.24  uart.c

#include <avr/io.h>
#include <avr/pgmspace.h>
#include "uart.h"

void uart_putc(char c) {
  loop_until_bit_is_set (UCSROA, UDRE0);
  UDRO = c;
}

char uart_getc(void) {
  loop_until_bit_is_set(UCSROA, RXC0);
}
void uart_getcs(char *s, unsigned int counter)
{
    while(counter != 0)
    {
        *s = uart_getc();
        s++;
        counter--;
    }
    return;
}

void uart_putcs(char *s, unsigned int counter)
{
    while(counter != 0)
    {
        uart_putchar(*s);
        s++;
        counter--;
    }
    return;
}

void uart_puts(char *s)
{
    do
    {
        uart_putchar(*s);
        s++;
    } while(*s != '\0');
    return;
}

void uart_puts_P(const char *addr)
{  
    char c;

    do  
    {  
        c = pgm_read_byte(addr);
        uart_putchar(c);
        addr++;
    } while(c != '\0');

}
C.2.25 x1226.h

/* Copyright Colin O’Flynn (c) 2005 - coflynn@newae.com */

#ifndef __X1226_H
#define __X1226_H

#include "error.h"

enum x1226_sla_t { X1226_CCR = 0xDE, X1226_EE = 0xAE }

ERROR_T x1226_init(void);

ERROR_T x1226_random_read(enum x1226_sla_t x1226_sla, uint16_t address, uint8_t *data,
                           unsigned char numbytes);

ERROR_T x1226_random_write(enum x1226_sla_t x1226_sla, uint16_t address, uint8_t *data,
                            unsigned char numbytes);

#endif // __X1226_H

C.2.26 x1226.c

/*******************************************************************
* PROJECT: LoonBoard Project
* *
* COPYRIGHT: Colin O’Flynn of NewAE, (C) 2005 <coflynn@newae.com>
* *
* LICENSE: This file is released under a license specified in the LICENSE
* file in the main project directory that this file was in. If
* the license file is missing, then this file is NOT licensed
* and all standard copyright laws apply.
* *
* FILE NAME: x1226.c
* *
* PURPOSE: Communications for X1226
* *
* FILE REFERENCES:
* *
* Name     I/O Description
* ----     ---  -----------
* twi_comms.c IO Two Wire Interface
* *
* ABNORMAL TERMINATION CONDITIONS, ERROR AND WARNING MESSAGES:
* *
* ASSUMPTIONS, CONSTRAINTS, RESTRICTIONS:
* *
* NOTES:
* *
* REQUIREMENTS/FUNCTIONAL SPECIFICATIONS REFERENCES:
* *
* DEVELOPMENT HISTORY:
* *
* Date(DMY) Author Release Description Of Change
* ----     ------ ----------  ---------------------------------------
* 15-02-2005 Colin O’Flynn 0.1 First working release
* *
*******************************************************************/

#include <avr/io.h>

#include "x1226.h"
#include "twi_comms.h"
#include "error.h"

ERROR_T x1226_init(void) {

unsigned char buffer[1];
buffer[0] = 0x02;

//Enable write access to chip
return (x1226_random_write(X1226_CCR, 0x3F, buffer, 1));
}

ERROR_T x1226_random_read(enum x1226_sla_t x1226_sla, uint16_t address, uint8_t *data, unsigned char numbytes)
{
    uint8_t adddata[2];
    ERROR_T error;
    adddata[1] = (uint8_t) address;
    adddata[0] = (uint8_t) ((uint16_t)address >> 8);

    error = twi_send(x1226_sla, adddata, 2);
    if (error != ERROR_NONE)
    {
        return error;
    }

    error = twi_get(x1226_sla, data, numbytes);
    if (error != ERROR_NONE)
    {
        return error;
    }

twi_stop();

    return ERROR_NONE;
}

ERROR_T x1226_random_write(enum x1226_sla_t x1226_sla, uint16_t address, uint8_t *data, unsigned char numbytes)
{
    uint8_t adddata[2];
    ERROR_T error;
    unsigned char maxtries;
    adddata[1] = (uint8_t) address;
    adddata[0] = (uint8_t) ((uint16_t)address >> 8);

    /* The device returns NACK to the address if it is busy, so try a bunch of times*/
    maxtries = 0;
    do
    {
        error = twi_send(x1226_sla, adddata, 2);
        maxtries++;
    }while ((error != ERROR_NONE) && (maxtries < 30));

    if (maxtries == 30)
    return error;

    error = twi_send_data(data, numbytes);
    if (error != ERROR_NONE)
    {
        return error;
    }

twi_stop();

    return ERROR_NONE;
}

ERROR_T x1226_testee(void)
uint16_t address;
uint8_t data[1];
ERROR_T error;

address = 0x0000;

while (address < 0x01FF)
{
    data[0] = (uint8_t)address ^ 0xAA;
    error = x1226_random_write(X1226_EE, address, data, 1);
    //printf("Address %x wrote %x\n", address, data[0]);
    if (error != ERROR_NONE)
        return error;
    address++;
}

address = 0x0000;
while (address < 0x01FF)
{
    error = x1226_random_read(X1226_EE, address, data, 1);
    //printf("Address %x read %x\n", address, data[0]);
    if (error != ERROR_NONE)
        return error;
    if (data[0] != ((uint8_t)address ^ 0xAA))
        return ERROR_TEST_FAILED;
    address++;
}

return ERROR_NONE;
# define serial_getc uart_getc
# define serial_putc uart_putc
# define serial_putcs uart_putcs
# define serial_getcs uart_getcs

void uart_putc(char c);
char uart_getc(void);
void uart_getcs(char *s, unsigned int counter);
void uart_putcs(char *s, unsigned int counter);
C.2.28  error.h

#ifndef __ERROR_H
#define __ERROR_H

#define ERROR_T uint16_t

/* Offsets: */
0x0000  0x00FF: General Error codes
0x00100  0x01FF: TWI Error codes

#define TWI_OFFSET 0x0100

/* Error codes */
/* General */
#define ERROR_NONE 0
#define ERROR_OVERFLOW 1
#define ERROR_UNDERFLOW 2
#define ERROR_RANGE 3
#define ERROR_READY 4
#define ERROR_TEST_FAILED 5
#define ERROR_TIMEOUT 6
#define ERROR_GENERAL -1

#endif // __ERROR_H
C.2.29  led_control.h

#ifndef LOONBOARD_LED_CONTROL_H
#define LOONBOARD_LED_CONTROL_H

#ifndef ON
#define ON 1
#define OFF 0
#endif

#define LED_INIT() DDRD |= 7<<1 | 6<<1; LED_RED(OFF); LED_GREEN(OFF);
#define LED_RED(x) if (x == ON) { PORTD |= 7<<1;} else { PORTD &= ~(7<<1);}
#define LED_GREEN(x) if (x == ON) { PORTD |= 6<<1;} else { PORTD &= ~(6<<1);}

/*
#define LED_INIT() DDRB |= 1<<0 | 1<<1; LED_RED(OFF); LED_GREEN(OFF);
#define LED_RED(x) if (x == ON) { PORTB |= 1<<0;} else { PORTB &= ~(1<<0);}
#define LED_GREEN(x) if (x == ON) { PORTB |= 1<<1;} else { PORTB &= ~(1<<1);}
*/

#endif*/
C.2.30 Makefile

# Hey Emacs, this is a -*- makefile -*-
#===================================================================
# WinAVR Makefile Template written by Eric B. Weddington, Jg Wunsch, et al.
#
# Released to the Public Domain
#
# Additional material for this makefile was written by:
# Peter Fleury
# Tim Henigan
# Colin O’Flynn
# Reiner Patein
# Markus Pfaff
# Sander Pool
# Frederik Rouleau
#
#===================================================================

# On command line:
#
# make all = Make software.
#
# make clean = Clean out built project files.
#
# make coff = Convert ELF to AVR COFF.
#
# make extcoff = Convert ELF to AVR Extended COFF.
#
# make program = Download the hex file to the device, using avrdude.
# Please customize the avrdude settings below first!
#
# make debug = Start either simulavr or avarice as specified for debugging,
# with avr-gdb or avr-insight as the front end for debugging.
#
# make filename.s = Just compile filename.c into the assembler code only.
#
# make filename.i = Create a preprocessed source file for use in submitting
# bug reports to the GCC project.
#
# To rebuild project do "make clean" then "make all".
#===================================================================

# MCU name
MCU = atmega128

# Hardware in use: Options are LOGNBOARD and YAJU
# HARDWARE = LOGNBOARD
HARDWARE = YAJU

# Processor frequency.  
# This will define a symbol, F_CPU, in all source code files equal to the 
# processor frequency. You can then use this symbol in your source code to 
# calculate timings. Do NOT tack on a 'UL' at the end, this will be done 
# automatically to create a 32-bit value in your source code.
F_CPU = 7372800
#F_CPU = 3686000
#F_CPU = 8000000

# Baudrate to communicate at
BAUD = 115200UL
#BAUD = 19200UL

# Output format. (can be srec, ihex, binary)
FORMAT = ihex

# Target file name (without extension).
TARGET = main

SRC = $(TARGET).c lub.c crc8.c dataflash.c uart.c calibrateosc.c x1226.c twi_comms.c
# List Assembler source files here.
# Make them always end in a capital .S. Files ending in a lowercase .s
# will not be considered source files but generated files (assembler
# output from the compiler), and will be deleted upon "make clean"!
# Even though the DOS/Win* filesystem matches both .s and .S the same,
# it will preserve the spelling of the filenames, and gcc itself does
# care about how the name is spelled on its command-line.

ASRC =

# Optimization level, can be [0, 1, 2, 3, s].
# 0 = turn off optimization. s = optimize for size.
# (Note: 3 is not always the best optimization level. See avr-libc FAQ.)
OPT = s

# Debugging format.
# Native formats for AVR-GCC's -g are dwarf-2 [default] or stabs.
# AVR Studio 4.10 requires dwarf-2.
# AVR [Extended] COFF format requires stabs, plus an avr-objcopy run.
DEBUG =

# List any extra directories to look for include files here.
# Each directory must be separated by a space.
# Use forward slashes for directory separators.
# For a directory that has spaces, enclose it in quotes.
EXTRAINCDIRS =

# Compiler flag to set the C Standard level.
# c89 = "ANSI" C
#-gnu89 = c89 plus GCC extensions
# c99 = ISO C99 standard (not yet fully implemented)
#-gnu99 = c99 plus GCC extensions
CSTANDARD = -std=gnu99

# Place -D or -U options here
CDEFS = -DF_CPU=$(F_CPU)UL -DBAUDRATE=$(BAUD) -D$(HARDWARE)

# Place -I options here
CINCS =

#---------------- Compiler Options ----------------
# -g*: generate debugging information
# -O*: optimization level
# -f...: tuning, see GCC manual and avr-libc documentation
# -Wall...: warning level
# -Wa,...: tell GCC to pass this to the assembler.
# -adhlns...: create assembler listing
CFLAGS = -g$(DEBUG)
CFLAGS += $(CDEFS) $(CINCS)
CFLAGS += -O$(OPT)
CFLAGS += -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums
CFLAGS += -Wall -Wstrict-prototypes
CFLAGS += -Wa,-adhlns=$(<:.c:.lst)
CFLAGS += $(patsubst %,-I%,$(EXTRAINCDIRS))
CFLAGS += $(CSTANDARD)

#---------------- Assembler Options ----------------
# -Wa,...: tell GCC to pass this to the assembler.
# -ahlms: create listing
# -gstabs: have the assembler create line number information; note that
# for use in COFF files, additional information about filenames
# and function names needs to be present in the assembler source
# files -- see avr-libc docs [FIXME: not yet described there]
ASFLAGS = -Wa,-adhlns=$( <:.S=. lst ),-gstabs

#---------------- Library Options ----------------
# Minimalistic printf version
PRINTF_LIB_MIN = -Wl,-u,vfprintf -lprintf_min
# Floating point printf version (requires MATH_LIB = -lm below)
PRINTF_LIB_FLOAT = -Wl,-u,vfprintf -lprintf_flt
# If this is left blank, then it will use the Standard printf version.
PRINTF_LIB =
#PRINTF_LIB = $(PRINTF_LIB_MIN)
#PRINTF_LIB = $(PRINTF_LIB_FLOAT)

# Minimalistic scanf version
SCANF_LIB_MIN = -Wl,-u,vfscanf -lscanf_min
# Floating point %[ scanf version (requires MATH_LIB = -lm below)
SCANF_LIB_FLOAT = -Wl,-u,vfscanf -lscanf_flt
# If this is left blank, then it will use the Standard scanf version.
SCANF_LIB =
#SCANF_LIB = $(SCANF_LIB_MIN)
#SCANF_LIB = $(SCANF_LIB_FLOAT)

MATH_LIB = -lm

#---------------- External Memory Options ----------------
# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# used for variables (.data/.bss) and heap (malloc()).
#EXTMEMOPTS = -Wl,-Tdata=0x801100,-- defsym=\_\_heap\_end=0x80ffff
# 64 KB of external RAM, starting after internal RAM (ATmega128!),
# only used for heap (malloc()).
#EXTMEMOPTS = -Wl,-- defsym=\_\_heap\_start=0x801100,-- defsym=\_\_heap\_end=0x80ffff
EXTMEMOPTS =

#---------------- Linker Options ----------------
# -Wl,...: tell GCC to pass this to linker.
# -Map: create map file
# --cref: add cross reference to map file
LDFLAGS = -Wl,--section-start=.bootloader=0x1C00 -Wl,-Map=$(TARGET).map,--cref
LDFLAGS += $(EXTMEMOPTS)
LDFLAGS += $(PRINTF_LIB) $(SCANF_LIB) $(MATH_LIB)

#---------------- Programming Options (avrdude) ----------------
# Programming hardware: alf avr910 avrISP bascom bsd
dt006 pavr picoweb pony-stk200 spi12 stk200 stk500
# Type: avrdude -c ?

171
# to get a full listing.
# AVRDUDE_PROGRAMMER = pony-stk200

# com1 = serial port. Use lpt1 to connect to parallel port.
#AVRDUDE_PORT = /dev/ttyS0  # programmer connected to serial device
#AVRDUDE_PORT = lpt1  # programmer connected to serial device
AVRDUDE_WRITE_FLASH = -U flash:v:$(TARGET).hex
#AVRDUDE_WRITE_EEPROM = -U eeprom:v:$(TARGET).eep

# Uncomment the following if you want avrdude's erase cycle counter.
# Note that this counter needs to be initialized first using -Yn,
# see avrdude manual.
#AVRDUDE_ERASE_COUNTER = -y

# Uncomment the following if you do /not/ wish a verification to be
# performed after programming the device.
#AVRDUDE_NO_VERIFY = -V

# Increase verbosity level. Please use this when submitting bug
# reports about avrdude. See <http://savannah.nongnu.org/projects/avrdude>
# to submit bug reports.
#AVRDUDE_VERBOSE = -v -V

AVRDUDE_FLAGS = -p $(MCU) -P $(AVRDUDE_PORT) -c $(AVRDUDE_PROGRAMMER)
AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)

---------------- Debugging Options ----------------

# For simulavr only - target MCU frequency.
DEBUG_MFREQ = $(F_CPU)

# Set the DEBUG_UI to either gdb or insight.
# DEBUG_UI = gdb
DEBUG_UI = insight

# Set the debugging back-end to either avarice, simulavr.
DEBUG_BACKEND = avarice
#DEBUG_BACKEND = simulavr

# GDB Init Filename.
GDBINIT_FILE = __avr_gdbinit

# When using avarice settings for the JTAG
JTAG_DEV = /dev/com1

# Debugging port used to communicate between GDB / avarice / simulavr.
DEBUG_PORT = 4242

# Debugging host used to communicate between GDB / avarice / simulavr, normally
# just set to localhost unless doing some sort of crazy debugging when
# avarice is running on a different computer.
DEBUG_HOST = localhost

# Define programs and commands.
SHELL = sh
CC = avr-gcc
OBJCOPY = avr-objcopy
OBJDUMP = avr-objdump
SIZE = avr-size
NM = avr-nm
AVRDUDE = avrdude
REMOVE = rm -f
COPY = cp
WINSHELL = cmd

# Define Messages
# English
MSG_ERRORS_NONE = Errors: none
MSG_BEGIN = -------- begin --------
MSG_END = -------- end --------
MSG_SIZE_BEFORE = Size before:
MSG_SIZE_AFTER = Size after:
MSG_COFF = Converting to AVR COFF:
MSG_EXTENDED_COFF = Converting to AVR Extended COFF:
MSG_FLASH = Creating load file for Flash:
MSG_EEPROM = Creating load file for EEPROM:
MSG_EXTENDED_LISTING = Creating Extended Listing:
MSG_SYMBOL_TABLE = Creating Symbol Table:
MSG_LINKING = Linking:
MSG_COMPILING = Compiling:
MSG_ASSEMBLING = Assembling:
MSG_CLEANING = Cleaning project:

# Define all object files.
OBJ = $( SRC :.c=.o) $( ASRC :.S=.o)

# Define all listing files.
LST = $( SRC :.c=.lst ) $( ASRC :.S=.lst )

# Compiler flags to generate dependency files.
GENDEPFLAGS = -MD -MP -MF . dep /$(@F).d

# Combine all necessary flags and optional flags.
# Add target processor to flags.
ALL_CFLAGS = -mmcu=$( MCU ) -I. $( CFLAGS ) $( GENDEPFLAGS )
ALL_ASFLAGS = -mmcu=$( MCU ) -I. -x assembler-with-cpp $( ASFLAGS )

# Default target.
all: begin gccversion sizebefore build sizeafter end

build: elf hex eep lss sym

elf: $(TARGET).elf
hex: $(TARGET).hex
eep: $(TARGET).eep
lss: $(TARGET).lss
sym: $(TARGET).sym

# Eye candy.
# AVR Studio 3.x does not check make's exit code but relies on
# the following magic strings to be generated by the compile job.
begin:
@echo
@echo $(MSG_BEGIN)

end:
@echo $(MSG_END)
@echo

# Display size of file.
HEXSIZE = $(SIZE) --target=$(FORMAT) $(TARGET).hex
ELFSIZE = $(SIZE) -A $(TARGET).elf
AVRMEM = avr-mem.sh $(TARGET).elf $(MCU)

sizebefore:
    @if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_BEFORE); $(ELFSIZE); $(AVRMEM) 2>/dev/null; echo; fi

sizeafter:
    @if test -f $(TARGET).elf; then echo; echo $(MSG_SIZE_AFTER); $(ELFSIZE); $(AVRMEM) 2>/dev/null; echo; fi

# Display compiler version information.
gccversion:
    @$(CC) --version

# Program the device.
program: $(TARGET).hex $(TARGET).eep
    $(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH) $(AVRDUDE_WRITE_EEPROM)

# Generate avr-gdb config/init file which does the following:
# define the reset signal, load the target file, connect to target, and set
# a breakpoint at main().
gdb-config:
    @$(REMOVE) $(GDBINIT_FILE)
    @echo define reset >> $(GDBINIT_FILE)
    @echo SIGNAL SIGHUP >> $(GDBINIT_FILE)
    @echo end >> $(GDBINIT_FILE)
    @echo file $(TARGET).elf >> $(GDBINIT_FILE)
    @echo target remote $(DEBUG_HOST):$(DEBUG_PORT) >> $(GDBINIT_FILE)
    ifdef ($(DEBUG_BACKEND), simulavr)
        @echo load >> $(GDBINIT_FILE)
    endif
    @echo break main >> $(GDBINIT_FILE)

default: gdb-config $(TARGET).elf
    @$(WINSHELL) /c start avarice --jtag $(JTAG_DEV) --erase --program --file $(TARGET).elf $(DEBUG_HOST):$(DEBUG_PORT)
    @$(WINSHELL) /c pause
    @$(WINSHELL) /c start simulavr --gdbserver --device $(MCU) --clock-freq $(DEBUG_MFREQ) --port $(DEBUG_PORT)
    @$(WINSHELL) /c start avr-$(DEBUG_UI) --command=$(GDBINIT_FILE)

# Convert ELF to COFF for use in debugging / simulating in AVR Studio or VMLAB.
COFFCONVERT=$(OBJCOPY) --debugging \

--change-section-address .data-0x800000 \
--change-section-address .bss-0x800000 \
--change-section-address .noinit-0x800000 \
--change-section-address .eeprom-0x810000

coff: $(TARGET).elf
    @echo
    @echo $(MSG_COFF) $(TARGET).cof
    $(COFFCONVERT) -O coff-avr $< $(TARGET).cof

extcoff: $(TARGET).elf
    @echo
    @echo $(MSG_EXTENDED_COFF) $(TARGET).cof
    $(COFFCONVERT) -O coff-ext-avr $< $(TARGET).cof

# Create final output files (.hex, .eep) from ELF output file.
%.hex: %.elf
    @echo
    @echo $(MSG_FLASH) $@
    $(OBJCOPY) -O $(FORMAT) -R .eeprom $< $@

%.eep: %.elf
    @echo
    @echo $(MSG_EEPROM) $@
    -$(OBJCOPY) -j .eeprom --set-section-flags=.eeprom="alloc,load" \  
    --change-section-lma .eeprom=0 -O $(FORMAT) $< $@

# Create extended listing file from ELF output file.
%.iss: %.elf
    @echo
    @echo $(MSG_EXTENDED_LISTING) $@
    $(OBJDUMP) -h -S $< > $@

# Create a symbol table from ELF output file.
%.sym: %.elf
    @echo
    @echo $(MSG_SYMBOL_TABLE) $@
    $(NM) -n $< $@

# Link: create ELF output file from object files.
.SECONDARY: $(TARGET).elf
.PRECIOUS: $(OBJ)
%.elf: $(OBJ)
    @echo
    @echo $(MSG_LINKING) $@
    $(CC) $(ALL_CFLAGS) $^ --output $@ $(LDFLAGS)

# Compile: create object files from C source files.
%.o: %.c
    @echo
    @echo $(MSG_COMPILING) $<
    $(CC) -c $(ALL_CFLAGS) $< -o $@

# Compile: create assembler files from C source files.
%.s: %.c
    $(CC) -S $(ALL_CFLAGS) $< -o $@

# Assemble: create object files from assembler source files.
%.o: %.s
@echo
@echo $(MSG_ASSEMBLING) $<
$(CC) -c $(ALL_ASFLAGS) $< -o $@

# Create preprocessed source for use in sending a bug report.
%.i : %.c
  $(CC) -E -mmcu=$(MCU) -I. $(CFLAGS) $< -o $@

# Target: clean project.
clean: begin clean_list end

clean_list :
  @echo
  @echo $(MSG_CLEANING)
  $(REMOVE) $(TARGET).hex
  $(REMOVE) $(TARGET).eep
  $(REMOVE) $(TARGET).cof
  $(REMOVE) $(TARGET).elf
  $(REMOVE) $(TARGET).map
  $(REMOVE) $(TARGET).sym
  $(REMOVE) $(TARGET).lss
  $(REMOVE) $(OBJ)
  $(REMOVE) $(LST)
  $(REMOVE) $(SRC:.c=.s)
  $(REMOVE) $(SRC:.c=.d)
  $(REMOVE) .dep/*

# Include the dependency files.
include $(shell mkdir . dep 2>/dev/null) $(wildcard .dep/*)

# Listing of phony targets.
.PHONY : all begin finish end sizebefore sizeafter gccversion build elf hex eep lss sym coff extcoff clean clean_list program debug gdb-config
C.3 Microcontroller Code

C.3.1 Appmain.c

/**********************************************************
* AppMain.c Author: Yaju Nagaonkar
* 11 April 2006 5:46 PM
* This is the main executable program which runs on the
* microcontroller and provides a menu based user-interface
**********************************************************/

#include "sercom.h"
#include "avr/interrupt.h"
#include "avr/signal.h"
#include "avr/io.h"
#include "bit.h"
#include "Blink.h"
#include "dataflash.h"
#include "slaveserial.h"
#include <string.h>
#include <stdio.h>

extern unsigned int dirty_bit;
extern unsigned int rd_dirty_bit;

int main(void)
{
    sei(); //enable interrupts
    SerCom0Init(23); //19200 Baud Rate
    //Print message to screen to validate
    //successful UART initialization
    PutString0 ("UART initialised \t\r\n");
    //Initialize the pins connected to the FPGA
    //used for slave serial configuration
    init_slaveserial();
    
    /* Initialize SPI interface to DFlash */
    DF_SPI_init();
    Read_DF_status();
    
    /* Temp variables */
    unsigned char temp, temp_d;
    unsigned int i;
    long k;
    
    /* Dataflash variables */
    dirty_bit = 0;
    rd_dirty_bit = 0;
    
    int bufcnt =0;
    int pagecounter = 0;
    
    while (1)
    {
        if(Hit0())
        {
            temp = GetChar0();
            switch (temp)
            {
                case '1'://blinks the LED
                    redON();
                    redOFF();
                    break;
                case '2'://FPGA configuration
redON();
PutString0("FPGA slave serial programming\r\n");

// resets the FPGA
clearFPGA();
// reset the Flash ROM variables
// FPGA configuration data is stored at location 1502
bufcnt = 0;
pagecounter = 1502;

while(!(Read_DF_status() & 0x80))
{ Cont_Flash_Read_Enable(1502, 0x00);
  for(k = 0; k < 212393; k++)
  {
    slaveserialPC(DF_SPI_RW(0x00));
    bufcnt++;
    if(bufcnt == 264)
    {
      bufcnt = 0;
      pagecounter++;
    }
    rd_dirty_bit = 1;
  }
  redOFF();
  break;
}

default : // echoes the character back
PutChar0(temp);
break;
}

// end Hit0()
// end while(1)
return 0;
}
C.3.2 dataflash.h

/*
  Last change: RM 17 Jan 2001 11:35
*/

/*
  ****************************************************************************
  
  COPYRIGHT (c) ATMEL Norway, 1996-2001
  
  The copyright to the document(s) herein is the property of
  ATMEL Norway, Norway.
  
  The document(s) may be used and/or copied only with the written
  permission from ATMEL Norway or in accordance with the terms and
  conditions stipulated in the agreement/contract under which the
  document(s) have been supplied.
  
  ****************************************************************************
  
  File........: DATAFLASH.H
  
  Author(s)....: ATMEL Norway
  
  Target(s)....: Independent
  
  Description.: Defines and prototypes for AT45Dxxx
  
  Revisions...:
  YYYYMMDD - VER. - COMMENT - SIGN.
  20010117 - 0.10 - Generated file - RM
  20031009 port to avr-gcc/avr-libc - M. Thomas
  
  ****************************************************************************
*/

#ifndef __DATAFLASH_INCLUDED
#define __DATAFLASH_INCLUDED
#endif

#define MTEXTRAS

General macro definitions
changed to *DF for avr-libc future compatibility
#define sbiDF(port,bit) ( port |= (1<<bit))
#define cbiDF(port,bit) ( port &= ~(1<<bit))

#define SetBit(x,y) (x |= (y))
#define ClrBit(x,y) (x &= ~(y))
#define ChkBit(x,y) (x & (y))

#define BUFFER_1 1
#define BUFFER_2 2

#define TRUE 0xFF
#define FALSE 0x00

//delay values based on a 8Mhz CPU clock
#define QTR_MICROSECOND 2
#define HALF_MICROSECOND 4
#define ONE_MICROSECOND 8
#define TWO_MICROSECONDS 16
#define THREE_MICROSECONDS 24
#define FIVE_MICROSECONDS 40
#define TEN_MICROSECONDS 80

/*****************************/
# define TWENTY_MICROSECONDS 160

// Dataflash opcodes
#define FlashPageRead 0x52 // Main memory page read
#define FlashToBuf1Transfer 0x53 // Main memory page to buffer 1 transfer
#define Buf1Read 0x54 // Buffer 1 read
#define FlashToBuf2Transfer 0x55 // Main memory page to buffer 2 transfer
#define Buf2Read 0x56 // Buffer 2 read
#define StatusReg 0x57 // Status register
#define AutoPageReWrBuf1 0x58 // Auto page rewrite through buffer 1
#define AutoPageReWrBuf2 0x59 // Auto page rewrite through buffer 2
#define FlashToBuf1Compare 0x60 // Main memory page to buffer 1 compare
#define FlashToBuf2Compare 0x61 // Main memory page to buffer 2 compare
#define ContArrayRead 0x68 // Continuous Array Read (Note: Only A/B-parts supported)
#define FlashProgBuf1 0x82 // Main memory page program through buffer 1
#define Buf1ToFlashWE 0x83 // Buffer 1 to main memory page program with built-in erase
#define Buf1Write 0x84 // Buffer 1 write
#define FlashProgBuf2 0x85 // Main memory page program through buffer 2
#define Buf2ToFlashWE 0x86 // Buffer 2 to main memory page program with built-in erase
#define Buf2Write 0x87 // Buffer 2 write
#define Buf1ToFlash 0x88 // Buffer 1 to main memory page program without built-in erase
#define Buf2ToFlash 0x89 // Buffer 2 to main memory page program without built-in erase
#define PageErase 0x81 // Page erase, added by Martin Thomas

// Pin definitions for interface to the Dataflash
#define DF_CS_active cbiDF(PORTB,4)
#define DF_CS_inactive sbiDF(PORTB,4)

// Function definitions
void DF_SPI_init (void);
unsigned char DF_SPI_RW (unsigned char output);
unsigned char Read_DF_status (void);
unsigned char PBits (void);
unsigned int PSize (void);
void Page_To_Buffer (unsigned int PageAdr, unsigned char BufferNo);
unsigned char Buffer_Read_Byte (unsigned char BufferNo, unsigned int IntPageAdr);
void Buffer_Read_Str (unsigned char BufferNo, unsigned int IntPageAdr, unsigned int No_of_bytes, unsigned char * BufferPtr);
void Buffer_Write_Enable (unsigned char BufferNo, unsigned int IntPageAdr, unsigned char Data);
void Buffer_Write_Str (unsigned char BufferNo, unsigned int IntPageAdr, unsigned int No_of_bytes, unsigned char * BufferPtr);
void Buffer_To_Page (unsigned char BufferNo, unsigned int PageAdr);
void Cont_Flash_Read_Enable (unsigned int PageAdr);
#define Page_Erase (unsigned int PageAdr); // added by mthomas
unsigned char Page_Buffer_Compare(unsigned char BufferNo, unsigned int PageAdr); // added by mthomas

// added YSN 01/16/2006
void write_to_flash(unsigned char flash_data);
void read_from_flash(long size);
void flash_erase(void);
unsigned char DF2FPGA(void); // endif

// ********************************[ End Of DATAFLASH.H ]********************************

// endif

C.3.3 dataflash.c

/***********************************************************************************/
dataflash.c  Author: Yaju Nagaonkar
11 April 2006 5:49PM
This file contains subroutines to provide read/write operation on the
the Atmel's dataflash

This file is a modified version of the original software provided by
Atmel
which is COPYRIGHT (c) ATMEL Norway, 1996-2001
***********************************************************************************/

// includes
#include <avr/io.h>
#include <inttypes.h>
#include <avr/pgmspace.h>
#include "dataflash.h"
#include "sercom.h"

// constants
// Look-up table for these sizes -> 512k, 1M, 2M, 4M, 8M, 16M, 32M, 64M
const unsigned char DF_pagebits[] ={9, 9, 9, 9, 10, 10, 11};
// index of internal page address bits
const uint8_t DF_pagebits[] PROGMEM ={9, 9, 9, 9, 10, 10, 11};
// index of internal page address bits
// Look-up table for these sizes -> 512k, 1M, 2M, 4M, 8M, 16M, 32M, 64M
const unsigned int DF_pagesize[] ={264, 264, 264, 264, 528, 528, 1056}; // index of pagesizes
const uint8_t DF_pagesize[] PROGMEM ={264, 264, 264, 264, 528, 528, 1056}; // index of pagesizes

// globals
unsigned char PageBits = 0;
unsigned int PageSize = 0;
unsigned char Page_Buffer_Compare(unsigned char BufferNo, unsigned int PageAdr); // added by mthomas

// data dirty bit
unsigned int dirty_bit;
unsigned int rd_dirty_bit;

// functions

/***********************************************************************************/
* Function name : DF2FPGA
* Returns : One read byte (any value)
* Parameters : None
* Purpose : Reads one byte from the flash memory at the
* location of the FPGA configuration bitstream
* function added by Yaju Nagaonkar
***********************************************************************************/
unsigned char DF2FPGA() 
{
static unsigned int page_counter;
static unsigned int buffer_counter;
unsigned char active_buffer = 1;
// long bytes = 0;

if(rd_dirty_bit == 0)
{
    page_counter = 1502;
    buffer_counter = 0;
}

unsigned char temp;
Cont_Flash_Read_Enable(page_counter,0);
Page_To_Buffer(page_counter,active_buffer);
while(!(Read_DF_status() & 0x80)); // wait until transfer of page0 to buffer1 is finished

temp = Buffer_Read_Byte(BUFFER_1,buffer_counter);
buffer_counter++;
if(buffer_counter > 255)
{
    buffer_counter = 0;
    if(page_counter < 4096)
    {
        Page_To_Buffer(page_counter,BUFFER_1);
        page_counter++;
    }
}
return temp;

 Melissa Atientu

/****************************************************************************
* Function name : write_to_flash
* Returns : None
* Parameters : flash_data -> byte of data
* Purpose : Write a byte of data to the flash memory.
*            Counters within the function are used for tracking.
* function added by Yaju Nagaonkar
*****************************************************************************/

void write_to_flash(unsigned char flash_data)
{
    static unsigned int buffer_counter; // keeps track of byte number in the buffer
    static unsigned int page_counter; // track of page number the buffer will be written to
    if(dirty_bit == 0)
    {
      //PutString0("Dirty Bit was not set \r\n");
        buffer_counter = 0;
        page_counter = 0;
    }

    while(!(Read_DF_status() & 0x80));
    Buffer_Write_Enable(BUFFER_1,buffer_counter);
    Buffer_Write_Byte(BUFFER_1,buffer_counter,flash_data);
    //Buffer_To_Page(BUFFER_1,page_counter);
    buffer_counter++;
    if(buffer_counter > 255)
    {
        buffer_counter = 0;
        if(page_counter < 4096)
        {
void read_from_flash (long size) {
    unsigned int page_counter = 1502;
    unsigned int buffer_counter = 0;
    unsigned char active_buffer = 1;
    long bytes = 0;
    Cont_Flash_Read_Enable (1502, 0);
    Page_To_Buffer (page_counter, active_buffer);
    while (!(Read_DF_status () & 0x80)); // wait until transfer of page0 to buffer1 is finished
    while ((page_counter < 4095) && (bytes < size)) {
        while ((buffer_counter < 264)) {
            PutChar0 (Buffer_Read_Byte (BUFFER_1, buffer_counter));
            bytes ++;
            buffer_counter ++;
        }
        page_counter ++;
        Page_To_Buffer (page_counter, BUFFER_1);
        buffer_counter = 0;
    }
}

void DF_SPI_init (void) {
    PORTB |= (1<<PB0); // set SS high
    DDRB |= (1<<DDB0);

    PORTB |= (1<<PB4); // enable ChipSelect high
    DDRB |= (1<<DDB4) | (1<<DDB1) | (1<<DDB2); // Set CS, SCK AND MOSI as outputs

    // SPSR = (1<<SPI2X); //SPI double speed
    SPCR = (1<<SPE) | (1<<MSTR) | (1<<CPHA) | (1<<CPOL); // Enable SPI in Master mode, mode 3, Fosc/4
}

void DF_SPI_RW (void) {
    Byte read from SPI data register (any value)
}
Parameters: Byte to be written to SPI data register (any value)
Purpose: Read and writes one byte from/to SPI master

unsigned char DF_SPI_RW (unsigned char output)
{

    unsigned char input;

    SPDR = output; // put byte 'output' in SPI data register
    while(!(SPSR & 0x80)); // wait for transfer complete, poll SPIF-flag
    input = SPDR; // read value in SPI data reg.

    return input; // return the byte clocked in from SPI slave
}

/* ****************************************************************************
* Function name : Read_DF_status
* Returns : One status byte. Consult Dataflash datasheet for further decoding info
* Parameters : None
* Purpose : Status info concerning the Dataflash is busy or not.
*          Status info concerning compare between buffer and flash page
*          Status info concerning size of actual device
* mt: the 'if' marked with 'mt 200401' is a possible optimisation
* if only one type of Dataflash is used (like on the Butterfly).
* If the uC controls different types of dataflash keep the PageBits and PageSize decoding in this function to avoid problems.
***************************************************************************** */

unsigned char Read_DF_status (void)
{

    unsigned char result , index_copy;

    DF_CS_inactive; // make sure to toggle CS signal in order
    DF_CS_active; // to reset dataflash command decoder
    result = DF_SPI_RW(StatusReg); // send status register read op-code
    result = DF_SPI_RW(0x00); // dummy write to get result

    index_copy = ((result & 0x38) >> 3); // get the size info from status register
    result = DF_SPI_RW(0x00); // dummy write to get result

    if (!PageBits) { // mt 200401
        PageBits = DF_pagebits[index_copy]; // get number of internal page address bits from look-up table
        PageSize = DF_pagesize[index_copy]; // get the size of the page (in bytes)
        PageBits = pgm_read_byte(0xDF_pagebits[index_copy]); // get number of internal page address bits from look-up table
        PageSize = pgm_read_word(0xDF_pagesize[index_copy]); // get the size of the page (in bytes)
    }

    return result; // return the read status register value
}

unsigned char PBits(void)
{

    unsigned char result , index_copy;

}
DF_CS_inactive;  // make sure
to toggle CS signal in order
DF_CS_active;   // to reset
dataflash command decoder
result = DF_SPI_RW(StatusReg); // send status register read
code
result = DF_SPI_RW(0x00);   // dummy write to get result

index_copy = ((result & 0x38) >> 3); // get the size info from status register
PageBits = DF_pagebits[index_copy]; // get number of internal page address
bits from look-up table
PageSize = DF_pagesize[index_copy]; // get the size of the page (in bytes)
return PageBits;  // return the read status register value

unsigned int PSize(void)
{
    unsigned char result, index_copy;
    DF_CS_inactive;  // make sure
to toggle CS signal in order
    DF_CS_active;    // to reset
dataflash command decoder
    result = DF_SPI_RW(StatusReg); // send status register read
code
    result = DF_SPI_RW(0x00);   // dummy write to get result
    index_copy = ((result & 0x38) >> 3); // get the size info from status register
    PageBits = DF_pagebits[index_copy]; // get number of internal page address
    bits from look-up table
    PageSize = DF_pagesize[index_copy]; // get the size of the page (in bytes)
    return PageSize;  // return the read status register value
}

_asm
unsigned int PSize(void)
{
    unsigned char result, index_copy;
    DF_CS_inactive;  // make sure
to toggle CS signal in order
    DF_CS_active;    // to reset
dataflash command decoder
    result = DF_SPI_RW(StatusReg); // send status register read
code
    result = DF_SPI_RW(0x00);   // dummy write to get result
    index_copy = ((result & 0x38) >> 3); // get the size info from status register
    PageBits = DF_pagebits[index_copy]; // get number of internal page address
    bits from look-up table
    PageSize = DF_pagesize[index_copy]; // get the size of the page (in bytes)
    return PageSize;  // return the read status register value
}
if (2 == BufferNo) //transfer flash page to buffer 2
{
    DF_SPI_RW(FlashToBuf2Transfer); //transfer to buffer 2 op-code
    DF_SPI_RW((unsigned char)(PageAdr >> (16 - PageBits))); //upper part of page address
    DF_SPI_RW((unsigned char)(PageAdr << (PageBits - 8))); //lower part of page address
    DF_SPI_RW(0x00); //don’t cares
}
#endif
DF_CS_inactive; //initiate the transfer
DF_CS_active;
while(!(Read_DF_status() & 0x80)); //monitor the status register, wait until busy-flag is high

unsigned char Buffer_Read_Byte (unsigned char BufferNo, unsigned int IntPageAdr)
{
    unsigned char data;
    data = '0'; //mt
    DF_CS_inactive; //make sure to toggle CS signal in order
    DF_CS_active; //to reset dataflash command decoder
    if (1 == BufferNo) //read byte from buffer 1
    {
        DF_SPI_RW(Buf1Read); //buffer 1 read op-code
        DF_SPI_RW(0x00); //don’t cares
        DF_SPI_RW((unsigned char)(IntPageAdr>>8)); //upper part of internal buffer address
        DF_SPI_RW((unsigned char)(IntPageAdr)); //lower part of internal buffer address
        DF_SPI_RW(0x00); //don’t cares
        data = DF_SPI_RW(0x00); //read byte
    }
    //endif USE_BUFFER2
    else
    {
        if (2 == BufferNo) //read byte from buffer 2
        {
            DF_SPI_RW(Buf2Read); //buffer 2 read op-code
            DF_SPI_RW(0x00); //don’t cares
        }
    }
DF_SPI_RW((unsigned char)(IntPageAdr >> 8)); // upper part of internal buffer address
DF_SPI_RW((unsigned char)(IntPageAdr)); // lower part of internal buffer address
DF_SPI_RW(0x00); // don’t cares
data = DF_SPI_RW(0x00); // read byte
}
#endif
return data; // return the read data byte

/******************************************************************************
* Function name : Buffer_Read_Str
* Returns : None
* Parameters : BufferNo -> Decides usage of either buffer 1 or 2
* IntPageAdr -> Internal page address
* No_of_bytes -> Number of bytes to be read
* * BufferPtr -> address of buffer to be used for read bytes
* Purpose : Reads one or more bytes from one of the dataflash internal SRAM buffers, and puts read bytes into buffer pointed to by *BufferPtr
*******************************************************************************/
void Buffer_Read_Str (unsigned char BufferNo, unsigned int IntPageAdr, unsigned int No_of_bytes, unsigned char * BufferPtr)
{
unsigned int i;

DF_CS_inactive; // make sure to toggle CS signal in order
DF_CS_active; // to reset dataflash command decoder

if (1 == BufferNo) // read byte(s) from buffer 1
{
    DF_SPI_RW(Buf1Read); // buffer 1 read op-code
    DF_SPI_RW(0x00); // don’t cares
    DF_SPI_RW((unsigned char)(IntPageAdr >> 8)); // upper part of internal buffer address
    DF_SPI_RW((unsigned char)(IntPageAdr)); // lower part of internal buffer address
    DF_SPI_RW(0x00); // don’t cares
    for (i = 0; i < No_of_bytes; i++)
    {
        *(BufferPtr) = DF_SPI_RW(0x00); // read byte and put it in AVR buffer pointed to by *BufferPtr
        BufferPtr++; // point to next element in AVR buffer
    }
}
#else USE_BUFFER2
else
if (2 == BufferNo) // read byte(s) from buffer 2
{
    DF_SPI_RW(Buf2Read); // buffer 2 read op-code
}
DF_SPI_RW(0x00);                // don't cares
DF_SPI_RW((unsigned char)(IntPageAdr>>8)); // upper part of internal buffer address
DF_SPI_RW((unsigned char)(IntPageAdr));  // lower part of internal buffer address
DF_SPI_RW(0x00);                  // don't cares
for ( i = 0; i < No_of_bytes; i++)
{
    *(BufferPtr) = DF_SPI_RW(0x00);  // read byte and put it in AVR buffer pointed to by *BufferPtr
    BufferPtr++;                      // point to next element in AVR buffer
}
#else

/****************************************************************************
* Function name : Buffer_Write_Enable
* Returns : None
* Parameters : IntPageAdr -> Internal page address to start writing from
*             BufferAdr -> Decides usage of either buffer 1 or 2
* Purpose : Enables continuous write functionality to one of the dataflash buffers.
*           Note : User must ensure that CS goes high to terminate this mode before accessing other dataflash functionalities
*****************************************************************************

void Buffer_Write_Enable (unsigned char BufferNo, unsigned int IntPageAdr)
{
    DF_CS_inactive;                    // make sure to toggle CS signal in order
    DF_CS_active;                     // to reset dataflash command decoder
    if (1 == BufferNo)
    {
        DF_SPI_RW(Buf1Write);        // buffer 1 write op-code
        DF_SPI_RW(0x00);             // don't cares
        DF_SPI_RW((unsigned char)(IntPageAdr>>8)); // upper part of internal buffer address
        DF_SPI_RW((unsigned char)(IntPageAdr));  // lower part of internal buffer address
    }
    ifndef USE_BUFFER2
    else
    {
        if (2 == BufferNo)
        {
            DF_SPI_RW(Buf2Write);        // buffer 2 write op-code
            DF_SPI_RW(0x00);             // don't cares
            DF_SPI_RW((unsigned char)(IntPageAdr>>8)); // upper part of internal buffer address
            DF_SPI_RW((unsigned char)(IntPageAdr));  // lower part of internal buffer address
        }
    }
    endif
}
*/
void Buffer_Write_Byte (unsigned char BufferNo, unsigned int IntPageAdr, unsigned char Data)
{
    DF_CS_inactive;  // make sure to toggle CS signal in order
    DF_CS_active;    // to
        // reset dataflash command decoder
        if (1 == BufferNo)  // write byte to buffer 1
        {
            DF_SPI_RW(Buf1Write);  // buffer 1
                write op-code
            DF_SPI_RW(0x00);         // don't cares
            DF_SPI_RW((unsigned char)(IntPageAdr>>8));  // upper part of internal
                buffer address
            DF_SPI_RW((unsigned char)(IntPageAdr));     // lower part of internal
                buffer address
            DF_SPI_RW(Data);         // write data byte
        }
        // #ifdef USE_BUFFER2
        else  // write byte to buffer 2
        {
            DF_SPI_RW(Buf2Write);  // buffer 2 write op-code
                write op-code
            DF_SPI_RW(0x00);         // don't cares
            DF_SPI_RW((unsigned char)(IntPageAdr>>8));  // upper part of internal
                buffer address
            DF_SPI_RW((unsigned char)(IntPageAdr));     // lower part of internal
                buffer address
            DF_SPI_RW(Data);         // write data byte
        }
        // #endif

}
void Buffer_Write_Str ( unsigned char BufferNo , unsigned int IntPageAdr , unsigned int No_of_bytes , unsigned char * BufferPtr )
{
    unsigned int i;
    DF_CS_inactive; // make sure to toggle CS signal in order
    DF_CS_active;  // to reset dataflash command decoder

    if (1 == BufferNo) // write byte(s) to buffer 1
    {
        DF_SPI_RW (Buf1Write);    //buffer 1
        DF_SPI_RW (0x00);         //don't cares
        DF_SPI_RW ((unsigned char)(IntPageAdr >> 8));  //upper part of internal buffer address
        DF_SPI_RW ((unsigned char)(IntPageAdr));       //lower part of internal buffer address
        for ( i = 0; i < No_of_bytes; i++)
        {
            DF_SPI_RW (*(BufferPtr));    //write byte pointed at by *BufferPtr to dataflash buffer 1 location
            BufferPtr++;                //point to next element in AVR buffer
        }
    }

    else if (2 == BufferNo) // write byte(s) to buffer 2
    {
        DF_SPI_RW (Buf2Write);    //buffer 2
        DF_SPI_RW (0x00);         //don't cares
        DF_SPI_RW ((unsigned char)(IntPageAdr >> 8));  //upper part of internal buffer address
        DF_SPI_RW ((unsigned char)(IntPageAdr));       //lower part of internal buffer address
        for ( i = 0; i < No_of_bytes; i++)
        {
            DF_SPI_RW (*(BufferPtr));    //write byte pointed at by *BufferPtr to dataflash buffer 2 location
            BufferPtr++;                //point to next element in AVR buffer
        }
    }
}

void Buffer_To_Page ( unsigned char BufferNo , unsigned int PageAdr )
{
}
DF_CS_inactive;  // make sure to toggle CS signal in order
DF_CS_active;    // to reset dataflash command decoder

if (1 == BufferNo) // program flash page from buffer 1
{
    DF_SPI_RW(Buf1ToFlashWE);  // buffer 1 to flash with erase op-code
    DF_SPI_RW((unsigned char)(PageAdr >> (16 - PageBits))); // upper part of page address
    DF_SPI_RW((unsigned char)(PageAdr << (PageBits - 8))); // lower part of page address
    DF_SPI_RW(0x00);       // don't cares
}

 elseif USE_BUFFER2
else if (2 == BufferNo) // program flash page from buffer 2
{
    DF_SPI_RW(Buf2ToFlashWE);  // buffer 2 to flash with erase op-code
    DF_SPI_RW((unsigned char)(PageAdr >> (16 - PageBits))); // upper part of page address
    DF_SPI_RW((unsigned char)(PageAdr << (PageBits - 8))); // lower part of page address
    DF_SPI_RW(0x00);       // don't cares
}
#endif

DF_CS_inactive;    // initiate flash page programming
DF_CS_active;

while (!((Read_DF_status() & 0x80)));
    // monitor the status register, wait until busy-flag is high
}

/******************************************************************************
* Function name : Cont_Flash_Read_Enable
* Returns : None
* Parameters : PageAdr -> Address of flash page where cont.read starts from
*              IntPageAdr -> Internal page address where cont.read starts from
* Purpose : Initiates a continuous read from a location in the DataFlash
******************************************************************************/
void Cont_Flash_Read_Enable (unsigned int PageAdr, unsigned int IntPageAdr)
{
    DF_CS_inactive;  // make sure to toggle CS signal in order
    DF_CS_active;    // to reset dataflash command decoder

    DF_SPI_RW(ContArrayRead);        // Continuous

    Array Read op-code

}
DF_SPI_RW((unsigned char)(PageAdr >> (16 - PageBits)));  
// upper part of page address
DF_SPI_RW((unsigned char)((PageAdr << (PageBits - 8)) + (IntPageAdr >> 8)));  
// lower part of page address and MSB of int.page adr.
DF_SPI_RW((unsigned char)(IntPageAdr));  
// LSB byte of internal page address
DF_SPI_RW(0x00);  
// perform 4 dummy writes
DF_SPI_RW(0x00);  
// in order to initiate DataFlash
DF_SPI_RW(0x00);  
// address pointers
DF_SPI_RW(0x00);

}  

// #ifdef MTEXTRAS
/****************************************************************************
* Function name : Page_Buffer_Compare
* Returns : 0 match, 1 if mismatch
* Parameters : BufferAdr -> Decides usage of either buffer 1 or 2
* PageAdr -> Address of flash page to be compared with buffer
* Purpose : compare Buffer with Flash-Page
* added by Martin Thomas, Kaiserslautern, Germany. This routine was not
* included by ATMEL
******************************************************************************/
unsigned char Page_Buffer_Compare(unsigned char BufferNo, unsigned int PageAdr) {
  unsigned char stat;
  DF_CS_inactive;  
  // make sure to toggle CS signal in order
  DF_CS_active;  
  // to reset dataflash command decoder
  if (1 == BufferNo) {
    DF_SPI_RW(FlashToBuf1Compare);
    DF_SPI_RW((unsigned char)(PageAdr >> (16 - PageBits)));  
    // upper part of page address
    DF_SPI_RW((unsigned char)(PageAdr << (PageBits - 8)));  
    // lower part of page address and MSB of int.page adr.
    DF_SPI_RW(0x00);  
    // "dont cares"
  }
#else USE_BUFFER2
  else if (2 == BufferNo) {
    DF_SPI_RW(FlashToBuf2Compare);
    DF_SPI_RW((unsigned char)(PageAdr >> (16 - PageBits)));  
    // upper part of page address
    DF_SPI_RW((unsigned char)(PageAdr << (PageBits - 8)));  
    // lower part of page address
    DF_SPI_RW(0x00);  
    // "don't cares"
  }
#endif
  DF_CS_inactive;
  DF_CS_active;
  do {
    stat=Read_DF_status();
  } while (!((stat & 0x80)));  
  // monitor the status register, wait until busy-flag is high


return (stat & 0x40);
}

//****************************************************************************
* Function name : Page_Erase
* Returns : None
* Parameters : PageAdr -> Address of flash page to be erased
* Purpose : Sets all bits in the given page (all bytes are 0xff)
* function added by mthomas.
*****************************************************************************

void Page_Erase (unsigned int PageAdr)
{
    DF_CS_inactive;       //make
    sure to toggle CS signal in order
    DF_CS_active;         //to
                          //reset dataflash command decoder
    DF_SPI_RW(PageErase);  //Page erase op-code
    DF_SPI_RW((unsigned char)(PageAdr >> (16 - PageBits))); //upper part of page address
    DF_SPI_RW((unsigned char)(PageAdr << (PageBits - 8)));  //lower part of page address and MSB of int.page adr.
    DF_SPI_RW(0x00);      //"dont cares"
    DF_CS_inactive;      //initiate flash page erase
    DF_CS_active;
    while(!(Read_DF_status() & 0x80));        //monitor the status register, wait until busy-flag is high
}

void flash_erase ()
{
    DF_CS_inactive;       //make
    sure to toggle CS signal in order
    DF_CS_active;         //to
                          //reset dataflash command decoder
    unsigned int PageAdr = 0;
    int pages;
    for (pages=0; pages < 4095; pages++)
    {
        DF_SPI_RW(PageErase);  //Page erase op-code
        DF_SPI_RW((unsigned char)(PageAdr >> (16 - PageBits))); //upper part of page address
        DF_SPI_RW((unsigned char)(PageAdr << (PageBits - 8)));  //lower part of page address and MSB of int.page adr.
        DF_SPI_RW(0x00);      //"dont cares"
        DF_CS_inactive;      //initiate flash page erase
        DF_CS_active;
        while(!(Read_DF_status() & 0x80));        //monitor the status register, wait until busy-flag is high
    }
}

//#endif
// ******************************************************************************
[ End Of DATAFLASH.C ]******************************************************************************
C.3.4  slaveserial.h

/*
slaveserial.h
*/
#include "bit.h"
void CCLK_HI(void);
void CCLK_LOW(void);
void init_slaveserial(void);
void slaveserial(unsigned char flash_start_addr, unsigned char flash_end_addr);
void shiftdataout(u08 data);
unsigned int check_done_bit(void);
void clearFPGA(void);
void clearFPGA2(void);
unsigned int check_init_bit(void);
void slaveserialPC(unsigned char data);

C.3.5  slaveserial.c

/* *****************************************************************************
* Filename: slaveserial.c
*
* Author: Preston K. Manwaring & Donald Wichern
*
* Date: Tuesday, January 25, 2005 21:08:21 -0700
*
* This is the main source file that will allow the microcontroller to
* configure Xilinx Spartan-3 device via the Slaveserial Configuration mode.
* This code is adapted from Xilinx Application Note, XAPP502.
*
*******************************************************************************/
#include "slaveserial.h"
#include "avr/io.h"
#include "sercom.h"
#include "bit.h"
#include "dataflash.h"
#include "Blink.h"

extern unsigned int rd_dirty_bit;
void init_slaveserial()
{
    // enable SPI for Dataflash read and write
    DF_SPI_init();
    Read_DF_status();

    // set PD1 as output. PD1 connects to CCLK
    PORTD |= (1<<PD1);
    DDRD |= (1<<DD1);

    // set PA0 as output. PA0 connects to DIN
    PORTA |= (1<<PA0);
    DDRA |= (1<<DDA0);

    // check the done bit
    // check_done_bit();
}

void slaveserial(unsigned char flash_start_addr, unsigned char flash_end_addr)
{
    // u16 Data16;
    u08 Data8;
    long i=0;

    // toggle the PROGB bit to reset FPGA.
    PutString0("Clearing FPGA \r\n");
    clearFPGA();
}
// check INIT bit is set high indicating ready for initialization

while (!bit_is_set(PINA,2));
PutString0("INIT bit is set \
\n");

// begin slave - serial configuration
// while (!check_done_bit() || i < 212393)
while (i < 212392)
{
    // read data from Dataflash
    if(!(check_init_bit()))
    {
        PutString0("INIT low; error detected \
\n");
    }
    Data8 = DF2FPGA();
    i++;
    rd_dirty_bit = 1;
    shiftdataout(Data8);
    // shiftdataout(0xAA);
    if((i == 10) || (i == 100) || (i == 10000) || (i == 20000) || (i == 212300))
    {
        PutString0("The iteration is 	");
        PutInt0(i);
        PutString0("\r\n");
    }
    if(i>212383 && i<212392)
    {
        PutString0("Data transferred\t");
        PutChar0(Data8);
        PutString0("\r\n");
    }
    //PutInt0(i);
    CCLK_LOW();
    CCLK_HI();
    CCLK_LOW();
    CCLK_HI();
    CCLK_LOW();
    CCLK_HI();
    CCLK_LOW();
    CCLK_HI();
    CCLK_LOW();
    CCLK_HI();
    CCLK_LOW();
    CCLK_HI();
    CCLK_LOW();
    CCLK_HI();

    PutString0("FPGA config done\r\n");
    rd_dirty_bit = 0;
}

void shiftdataout(u08 data8)
{
    u08 DataOut;
    DataOut = (data8 & 0x80) ? 1 : 0;

if(DataOut)
{ bit_set_8(Porta, (1<<PA0));}
else
{ bit_clear_8(Porta, (1<<PA0));}
CCLK_LOW();
CCLK_HI();

DataOut = (data8 & 0x40) ? 1 : 0;
if(DataOut)
{ bit_set_8(Porta, (1<<PA0));}
else
{ bit_clear_8(Porta, (1<<PA0));}
CCLK_LOW();
CCLK_HI();

DataOut = (data8 & 0x20) ? 1 : 0;
if(DataOut)
{ bit_set_8(Porta, (1<<PA0));}
else
{ bit_clear_8(Porta, (1<<PA0));}
CCLK_LOW();
CCLK_HI();

DataOut = (data8 & 0x10) ? 1 : 0;
if(DataOut)
{ bit_set_8(Porta, (1<<PA0));}
else
{ bit_clear_8(Porta, (1<<PA0));}
CCLK_LOW();
CCLK_HI();

DataOut = (data8 & 0x08) ? 1 : 0;
if(DataOut)
{ bit_set_8(Porta, (1<<PA0));}
else
{ bit_clear_8(Porta, (1<<PA0));}
CCLK_LOW();
CCLK_HI();

DataOut = (data8 & 0x04) ? 1 : 0;
if(DataOut)
{ bit_set_8(Porta, (1<<PA0));}
else
{ bit_clear_8(Porta, (1<<PA0));}
CCLK_LOW();
CCLK_HI();

DataOut = (data8 & 0x02) ? 1 : 0;
if(DataOut)
{ bit_set_8(Porta, (1<<PA0));}
else
{ bit_clear_8(Porta, (1<<PA0));}
CCLK_LOW();
CCLK_HI();

DataOut = (data8 & 0x01) ? 1 : 0;
if(DataOut)
{ bit_set_8(Porta, (1<<PA0));}
else
{ bit_clear_8(Porta, (1<<PA0));}
CCLK_LOW();
CCLK_HI();

}

void slaveserialPC(unsigned char data)
{
    shiftdataout(data);
if(!(check_init_bit()))
{
    while(1)
    {
        greenLEDblink(1);
    }
}

void CCLK_HI()
{
    bit_set_8(PORTD, (1<<PD1));
}

void CCLK_LOW()
{
    bit_clear_8(PORTD, (1<<PD1));
}

void clearFPGA(void)
{
    PORTF |= (1<<PF0);
    DDRF |= (1<<DDFO);
    
    long temp;
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    bit_clear_8(PORTF, (1<<PF0));
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    for(temp=0; temp < 3000000; temp++);
    bit_set_8(PORTF, (1<<PF0));
}

/*doesn't work*/
void clearFPGA2(void)
{
    PORTF |= (1<<PF0);
    DDRF |= (1<<DDFO);
    bit_clear_8(PORTF, (1<<PF0));
    if (bit_is_set(PINA,2))
    {PutString0("INIT bit is set \t\r\n");
        PutStringHex0(PINA);}
    else
        PutString0("INIT bit is not set");
}

unsigned int check_done_bit(void)
{
    //Read the DONE signal on the uP I/O register
    if bit_is_set(PIND,0)
    {return 1;}
    else
    {return 0;}
}

unsigned int check_init_bit(void)
{
    if (bit_is_set(PINA,2))
        return 1;
    else
        return 0;
C.3.6 sercom.h

/**************************************************************/
/* Filename: sercom.h                                         */
/* Author: Preston K. Manswaring & Donald Wichern            */
/* Date: July 3, 2003                                        */
/* Description: This file contains headers for sercom.c       */
/***************************************************************/

/**************************************************************/
/* --- FUNCTION PROTOTYPES for SERCOM.C --------------------- */
/***************************************************************/
unsigned char Hit0( void );
void SerCom0Init( unsigned int baud );
char PutChar0( unsigned char byte );
unsigned char GetChar0( void );
void PutString0( unsigned char *str );
void GetString0( unsigned char *str, unsigned int counter);
void PutInt0( long numi);
void PutInt0( long numi);
void PutHex0( unsigned char num );
void PutStringHex0( unsigned char * str );

unsigned char Hit1( void );
void SerCom1Init( unsigned int baud );
char PutChar1( unsigned char byte );
unsigned char GetChar1( void );
void PutString1( unsigned char *str );

/**************************************************************/
/* These settings assume a Fclk of 14.7456 MHz                */
#define BAUD_19200 47
#define BAUD_28800 31
#define BAUD_38400 23
#define BAUD_57600 15
#define BAUD_115200 7
#define BAUD_230400 3

198
C.3.7 sercom_int.c

/* ****************************************************************************
* Filename: sercom_int.c
* Author: Preston K. Manwaring & Donald Wichern
* Date: Tuesday, January 25, 2005 21:08:21 -0700
* Description: This file contains functions to write to the UART. On the
* ATMega8. All UART functions are interrupt driven.
* **************************************************************************** */

#include "sercom.h"
#include <avr/signal.h>
#include <avr/io.h>
#include "queue.h"
#include <string.h>

#define RCVD_QUEUE_SIZE 32
#define SEND_QUEUE_SIZE 32

/* --- GLOBAL VARIABLES -------------------------------------------------------
Queue
 UART_received,
 UART_send;

/* create the data storage arrays */
char
 send_queue[SEND_QUEUE_SIZE],
 rcvd_queue[RCVD_QUEUE_SIZE];

unsigned char n2h [16] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};

/* --- unsigned char Hit0 ( void ) ---------------------------------------------
unsigned char Hit0 ( void )
{ return ( ! QEmpty ( &UART_received )); }
*/

/* --- void SerCom0Init ( unsigned int baud ) -----------------------------------
void SerCom0Init ( unsigned int baud )
{
    // set baud rate
    UBRR0H = (unsigned char)(baud>>8);
    UBRR0L = (unsigned char)(baud);
    // enable receiver and transmitter
    UCSRB = (1<<RXEN) | (1<<TXEN) | (1<<RXCIE); // enable interrupts
    // set frame format 8 data bits, 1 stop bit
    UCSRC = (1<<USBS) | (3<<UCSZ0);
    QInit ( &UART_received, RCVD_QUEUE_SIZE );
    QInit ( &UART_send, SEND_QUEUE_SIZE );
}

/* --- void PutChar0 ( unsigned char byte ) ------------------------------------
char PutChar0 ( unsigned char byte )
{
    // wait for empty transmit buffer
    // while ( !( UCSRA & (1<<UDRE) ) );
    // send data
    UDR = byte;
}*/
/* Enqueue data to be sent out. Keep trying if the queue is full */
while ( QFull ( &UART_send ));
if ( QFull ( &UART_send ))
{
    return ( True );
}
send_queue[ QEnqueue ( &UART_send )] = byte;
UCSR0B |= _BV ( UDRIE );
return ( False );

/* --- unsigned char GetChar0 ( void ) ----------------------------------------- */
unsigned char GetChar0 ( void )
{
    return ( rcvd_queue[ QDequeue ( &UART_released )]);
}

/* --- void PutString0 ( unsigned char * str ) ----------------------------------- */
void PutString0 ( unsigned char * str )
{
    unsigned char i = 0;
    while ( str[i] )
    {
        PutChar0 ( str[i] );
        i++;
    }
}

/* --- void PutStringExt0 ( unsigned char * str, unsigned int counter ) */
void PutStringExt0 ( unsigned char * str, unsigned int counter )
{
    while ( counter != 0 )
    {
        *str = GetChar0 ();
        str++;
        counter--;
    }
}

/* --- void PutInt0 ( long numi ) ---------------------------------------------- */
void PutInt0 ( long numi )
{
    unsigned char i;
    unsigned char nums[10];
    if ( numi < 0 )
    {
        PutChar0 ( '-' );
        numi = -numi;
    }
    i = 0;
    do
    {
        nums[i] = numi % 10;
        numi = numi / 10;
        i++;
    } while ( numi );
    while ( i )
    {
        i--;
        PutChar0 ( nums[i] + '0' );
    }
}
void PutHex0 (unsigned char num)
{
    PutChar0 (n2h[(num >> 4) & 0xF]);
    PutChar0 (n2h[(num) & 0xF]);
}

void PutStringHex0 (unsigned char *str)
{
    unsigned char i = 0;
    while ( str[i] )
    {
        PutHex0 (str[i]);
        i++;
    }
}

/* --- SIG data received ISR --------------------------------------------------*
 * This ISR stores the incoming data.
 *--------------------------------------------------------------------------*/
SIGNAL (SIG_UART0_RECV)
{
    rcvd_queue [QEnqueue (&UART_received)] = UDR0; // Enqueue data from the UART
}

/* --- SIG data transmit complete ISR -----------------------------------------*
 * This ISR sends data stored in a queue when it can.
 *--------------------------------------------------------------------------*/
SIGNAL (SIG_UART0_DATA)
{
    if (QEmpty (&UART_send)) // if queue is empty, make sure the ISR won’t fire
    {
        UCSRB &= ~_BV(UDRIE);
    }
    else
    {
        UDR0 = send_queue [QDequeue (&UART_send)];
    }
}
C.3.8 queue.h

/**************************************************************
* Filename: queue.h
* Author: Preston K. Manwaring
* Date: Wednesday, March 23, 2005 12:08:35 -0700
* Description: Header file for queue.c. Should be included by for interrupt
* based SPI and UART functions.
***************************************************************/

/**************************************************************
* --- DEFINITIONS -------------------------------------------*
* #define False 0
* #define True ! False
* #define NotOK False
* #define OK True
***************************************************************/

/**************************************************************
* --- STRUCTURE DEFINITIONS ---------------------------------
* struct queue
* {
*    signed char Head;
*    signed char Tail;
*    char Size;
*};
* typedef struct queue Queue;
* typedef Queue *QueuePtr;
***************************************************************/

/**************************************************************
* --- FUNCTION PROTOTYPES ------------------------------------*
* #ifndef S_SPLINT_S
* unsigned char QClear ( QueuePtr q );
* signed char QDequeue ( QueuePtr q );
* unsigned char QEmpty ( QueuePtr q );
* signed char QEnqueue ( QueuePtr q );
* signed char QSize ( QueuePtr q );
* unsigned char QFull ( QueuePtr q );
* signed char QHead ( QueuePtr q );
* unsigned char QInit ( QueuePtr q, char NumElem );
* #endif
***************************************************************/

C.3.9 queue.c

/**************************************************************
* Filename: QUEUE.CPP
*
* Functional Description: The functions within this file compose
*       operations on queues. Queues are used in several of the L0S
*       device drivers.
* **************************************************************/

/* COMMENT --
The functions within this file create the appearance of a queue that is
constructed from an array. The actual data that is stored in the array
is the data that would be enqueued and dequeued. The data type of the
array is dependent on the data that is queued. The functions below
only operate on the indices of the array to give the appearance of a queue.

Within the file that uses the queue, two items must be declared. First, the
array holding the queue data is declared. Second, a variable of the type
QueuePtr is declared, that points to a structure that contains variables to
access the queue, the Head and Tail variables. These variables are indices
to the array containing the queue data.

To use these functions, the programmer passes arguments to them consisting
of the pointer to the queue structure, and a value indicating the size of
the array holding the queue data. The different functions return values
dependent upon their purpose and the results derived.

202
As an example, consider the following declarations within a file that uses some of these queue functions. (Examine the declarations at the start of this file to fully understand this example application.)

#define NumElem 128 -- the maximum number of elements in the queue
typedef uchar QData; -- the type of data stored within the queue
Queue QSnd; -- a variable of type containing the queue indices
QueuePtr SndQueue = & QSnd; -- pointer to the variable of queue indices
QData SndData[NumElem]; -- the array containing the queued data

With these declarations within the using file, the functions contained therein can use the queue accessing functions below to generate indices to SndData to allow enqueueing, dequeueing and other operations on the array as if it were a queue.

/* INCLUDE FILES */
#include "queue.h"

/* STRUCTURE DEFINITIONS */
/*
struct queue
{
  char Head;
  char Tail;
  char Size;
};
*/
typedef struct queue Queue;
typedef Queue *QueuePtr;
/*

/* FUNCTION DEFINITIONS */
/**--- 17 May 2000 -----------------------------------------------------------*/
unsigned char QClear ( QueuePtr q )
{ /* Remove all elements from the queue. Return OK; */
  q->Head = q->Tail = -1;
  return ( OK );
} /* end of function QClear */

/**--- 17 May 2000 -----------------------------------------------------------*/
signed char QDequeue ( QueuePtr q )
{ /* Return the index of the oldest (head or front) element from the queue and update the queue pointers as if it had been removed. Return -1 if the queue is empty. */
  unsigned char temp;
}
/* In order to remove an element from the queue, it must not be empty */
if( q->Head == -1 )
    return( -1 );

temp = q->Head;

/* Update the Head and Tail indices */
if( q->Head == q->Tail )
    q->Head = q->Tail = -1; /* Queue is now empty */
else
    q->Head = ( ++(q->Head)) % q->Size;
return( temp );
} /* end of function QDequeue */

unsigned char QEmpty ( QueuePtr q )
/*
Return True if a queue is empty. Otherwise, return False.
*/
{
    if( q->Head == -1 )
        return( True );
    return( False );
} /* end of function QEmpty */

signed char QEnqueue ( QueuePtr q )
/*
Get the index where a new element is to be inserted, at the rear
(tail or back) of the queue. Return -1 if the queue is full.
*/
{
    /* In order to insert an element, the queue must not be full */
    if( ((q->Tail+1) % q->Size) == q->Head )
        return( -1 );

    q->Tail = ( ++(q->Tail) ) % q->Size;
    if( q->Head == -1 ) /* The queue was empty before this enqueue */
        q->Head = q->Tail;

    return( q->Tail );
} /* end of function QEnqueue */

unsigned char QFull ( QueuePtr q )
/*
Return True if the queue is full. Otherwise, return False.
*/
{
    if( ((q->Tail + 1) % q->Size) == q->Head )
        return( True );
    return( False );
} /* end of function QFull */

signed char QHead( QueuePtr q )
/*
Return the index of the oldest (head or front) element from the
queue. The index is not changed, as it would be if the element were
being dequeued. If the queue is empty, return -1.
*/
{ /* Remember the queue might be empty, q->Head = -1 */ return ( q->Head ); } /* end of function QHead */

/**** 17 Mar 2001 -----------------------------------------------------------**/
unsigned char QInit( QueuePtr q, char NumElem )
/* Initialize the data structure elements associated with the queue before using them in the other functions. */
{
    q->Head = q->Tail = -1;
    q->Size = NumElem;
    return ( OK );
} /* end of function QInit */

/**** 17 May 2000 -----------------------------------------------------------**/
signed char QSize( QueuePtr q )
/* Return the number of elements currently in the queue */
{
    if( q->Head == -1 )
        return ( 0 ); /* queue is empty */
    else
    {
        if( q->Tail >= q->Head )
            return ( q->Tail - q->Head + 1 );
        else
            return ( q->Size - q->Head + q->Tail + 1 );
    }
} /* end of function QSize */

/**** end of file QUEUE.CPP ==================================================*/
C.3.10 blink.h

void redON(void);
void redOFF(void);
void redLEDblink(unsigned char Cnt);
void greenON(void);
void greenOFF(void);
void greenLEDblink(unsigned char Cnt);

C.3.11 blink.c

#include <avr/io.h>
#include "bit.h"
#include "Blink.h"

void redON()
{
    bit_set_8 (DDRD, (1<<PD7));
    bit_clear_8 (PORTD, (1<<PD7));
}
void redOFF()
{
    bit_set_8 (DDRD, (1<<PD7));
    bit_set_8 (PORTD, (1<<PD7));
}
void redLEDblink(unsigned char Cnt)
{
    long i;
    unsigned char j = 0;

    // set PortD pin 2 as an output, DDRD is the direction register for port D. A
    // one sets the pin as an output, a zero sets the pin as an input.
    // sbi ( DDRD, PD2 );
    bit_set_8 (DDRD, (1<<PD7));
    //DDR = 0X80;
    // set PD2 as a low value. PORTD is the mode register for port D. If set to
    while ( j < Cnt )
    {
        for ( i=0; i<3000000; i++ );
        for ( i=0; i<3000000; i++ );
        for ( i=0; i<3000000; i++ );

        bit_clear_8 (PORTD, (1<<PD7));
        //PORTD = 0X00;

        for ( i=0; i<3000000; i++ );
        for ( i=0; i<3000000; i++ );
        for ( i=0; i<3000000; i++ );

        bit_set_8 (PORTD, (1<<PD7));
        //PORTD = 0X80;
        j++;
    }
}
void greenON()
{
    bit_set_8 (DDRD, (1<<PD6));
    bit_clear_8 (PORTD, (1<<PD6));
}
void greenOFF()
{
    bit_set_8 (DDRD, (1<<PD6));
    bit_set_8 (PORTD, (1<<PD6));
}
void greenLEDblink(unsigned char Cnt)
{
    long i;
    unsigned char j = 0;

    // set PortD pin 2 as an output, DDRD is the direction register for port D. A
    // one sets the pin as an output, a zero sets the pin as an input.
    // sbi( DDRD, PD2 );
    bit_set_8(DDRD, (1<< PD6));
    // DDRD = 0X80;
    bit_clear_8(PORTD, (1<<PD6));
    // set PD2 as a low value. PORTD is the mode register for port D. If set to
while ( j < Cnt)
{
    for ( i=0; i<3000000; i++ );
    for ( i=0; i<3000000; i++ );
    for ( i=0; i<3000000; i++ );

    //PORTD = 0X00;
    bit_clear_8(PORTD, (1<<PD6));

    for ( i=0; i<3000000; i++ );
    for ( i=0; i<3000000; i++ );
    for ( i=0; i<3000000; i++ );

    bit_set_8 (PORTD, (1<<PD6));
    //PORTD = 0X80;
    j++;
}
}
C.3.12  bit.h

#include <inttypes.h>

#define outb(addr, data) addr = (data)
#define inb(addr) (addr)

#define true -1
#define false 0

#define u08 uint8_t
#define s08 int8_t
#define u16 uint16_t
#define s16 int16_t
#define u32 uint32_t
#define s32 int32_t

// 8-bit versions
#define bit_set_8(var, mask) ((uint8_t)(var) |= (mask))
#define bit_clear_8(var, mask) ((uint8_t)(var) &= ~(mask))
#define bit_toggle_8(var, mask) ((uint8_t)(var) ^= (mask))
#define bit_read_8(var, mask) ((uint8_t)(var) & (mask))

// 16-bit version
#define bit_set_16(var, mask) ((var) |= (uint16_t)(mask))
#define bit_clear_16(var, mask) ((var) &= (uint16_t)~(mask))
#define bit_toggle_16(var, mask) ((var) ^= (uint16_t)(mask))
#define bit_read_16(var, mask) ((var) & (uint16_t)(mask))

// Shorter named versions for the common operations.
#define bit_set(var, mask) bit_set_8(var, mask)
#define bit_clear(var, mask) bit_clear_8(var, mask)
#define bit_toggle(var, mask) bit_toggle_8(var, mask)
#define bit_read(var, mask) bit_read_8(var, mask)
C.4 FPGA VHDL code

C.4.1 top.vhd

--

-- Title : top
-- Design : Blink LEDs
-- Author : Yaju Nagaonkar
-- Company : Electrical and Computer Engineering, Brigham Young University
--
--

-- File : top.vhd
-- From : interface description file
-- By : Itf2Vhdl ver. 1.20
--
--

-- Description : Blink LEDs on the FPGA module
--
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- For instantiating Xilinx primitive library components
-- library UNISIM;
-- use UNISIM.VComponents.all;

entity top is
  Port (                      
    -- main board i/o
    clk : in std_logic;       -- 50 MHz clock
    rst_in : in std_logic;    -- button on main board,
    led : out std_logic_vector(1 downto 0) -- led on main board
  );
end top;

architecture behavioral of top is

signal locked : std_logic;
-- reset signals
signal rst : std_logic;
signal rsta : std_logic_vector(3 downto 0);
-- clock divider
signal clkdiv : std_logic_vector(23 downto 0);

begin
  process (clk, rst)
  begin
    -- when no reset is given, you rely on the power-on reset of the FPGA
    if clk'event and clk='1' then
      clkdiv <= clkdiv + 1;
    end if;
  end process;
with clkdiv(23 downto 22) select led <=
"01" when "00", -- digit 0
"00" when "11",
"11" when "01", -- digit 1
"10" when others; -- disable digits 2 and 3

-- reset circuit
-- these flip-flops synchronize and extend reset
process(clk, rst_in, locked)
begin
  if rst_in='1' or locked='0' then
    rsta <= "1111";
  elsif clk'event and clk='1' then
    rsta <= '0' & rsta(3 downto 1);
  end if;
end process;

rst <= rsta(0);

end Behavioral;
C.4.2  top.ucf

#PACE: Start of Constraints generated by PACE
#PACE: Start of PACE I/O Pin Assignments

NET "clk" LOC = "P128" | IOSTANDARD = LVCMOS33 ;
NET "led<0>" LOC = "P129" | IOSTANDARD = LVCMOS33 ;
NET "led<1>" LOC = "P127" | IOSTANDARD = LVCMOS33 ;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- synopsys translate_off
library UNISIM;
use UNISIM.vcomponents.all;

-- synopsys translate_on

entity ramtest is port
(clk50_in : in std_logic;
led_out : out std_logic_vector (7 downto 0);
-- 4 push button inputs
pb_in : in std_logic_vector (3 downto 0);
-- SRAM ports
sram_ce1_out : out std_logic;
sram_ub1_out : out std_logic;
sram_lb1_out : out std_logic;
sram_we_out : out std_logic;
sram_oe_out : out std_logic;
sram_a_out : out std_logic_vector(17 downto 0);
sram_io : inout std_logic_vector(15 downto 0)
);
end ramtest;

architecture Behavioral of ramtest is
-- 50 Mhz clock (creates 20ns cycles)
signal clk50 : std_logic;
signal clk50_before_dll : std_logic;
signal clk50_int : std_logic;
signal clk50_non_g : std_logic;

-- led
signal led : std_logic_vector (7 downto 0);

-- push button signal
signal pb : std_logic_vector (3 downto 0);
signal rst : std_logic;

-- sram
signal sram_ce1 : std_logic := '1';
signal sram_we : std_logic := '1';
signal sram_oe : std_logic := '1';
signal ioBUF_t : std_logic := '0';

-- where to read data from memory
signal sram_data_read : std_logic_vector(15 downto 0);

-- where to write data in memory
signal sram_data_write : std_logic_vector(15 downto 0);

-- location to write the desired address
signal sram_addr : std_logic_vector(17 downto 0) := "000000000000000000";
signal high_value : std_logic := '1';

signal low_value : std_logic := '0';
-- a counter to generate some events.
signal ctr : std_logic_vector(2 downto 0) := "000";
-- a counter to display data.
signal ctr_display : std_logic_vector(25 downto 0) := "00000000000000000000000000";

component OBUF_LVCMOS33 port
(O : out std_logic;
 I : in std_logic);
end component;

component ibufg port
(O : out std_logic;
 I : in std_logic);
end component;

-- CMOS33 input buffer primitive
component ibuf_lvcmos33 port
(i : in std_logic;
 o : out std_logic);
end component;

component IOBUF_LVCMOS33 port
(O : out std_logic;
 IO : inout std_logic;
 I : in std_logic;
 T : in std_logic);
end component;

component CLKDLL
port
(CLK0 : out std_logic;
 CLK90 : out std_logic;
 CLK180 : out std_logic;
 CLK270 : out std_logic;
 CLK2X : out std_logic;
 CLKDV : out std_logic;
 LOCKED : out std_logic;
 CLKIN : in std_logic;
 CLKFB : in std_logic;
 RST : in std_logic);
end component;

component ibufg_lvcmos33 port
(i : in std_logic;
 o : out std_logic);
end component;

component bufg port
(i : in std_logic;
 o : out std_logic);
end component;

begin
rst <= pb(0);
high_value <= '1';
low_value <= '0';

-- this CLKDLL component
-- generates the clk100 signal from the clk50_in signal
dll0 : CLKDLL port map
(CLKIN => clk50_before_dll,
CLKFB => clk50,
RST => low_value,
CLK0 => clk50_non_g,
CLK90 => OPEN,
CLK180 => OPEN,
CLK270 => OPEN,
CLK2X => OPEN,
CLKDV => OPEN,
LOCKED => OPEN);
if rst='1' then
    led <= "00000000";
    ctr_display <= "00000000000000000000000000";
    read_addr <= "00000000";
    write_addr <= "00000000";
    sram_addr <= "00000000000000000000000000";
    sram_data_write <= "0000000000000000";
    ctr <= "000";
    sram_oe <= '1';
    sram_ce1 <= '1';
    sram_we <= '0';
else
    led(7) <= read_addr(0);
    if ctr="000" then
        -- step 1 address controlled memory read
        iobuf_t <= '1';
        sram_addr <= "00000100001" & read_addr;
        sram_oe <= '0';
        sram_ce1 <= '0';
        sram_we <= '1';
        ctr <= "001";
    elsif ctr="001" then
        -- step 2 reset stuff and read values
        iobuf_t <= '0';
        sram_addr <= "00000100001" & read_addr;
        sram_oe <= '0';
        sram_ce1 <= '1';
        sram_we <= '1';
        if ctr_display="00000000000000000000000000" then
            read_addr <= read_addr + "000001";
        end if;
        ctr <= "010";
        led(0) <= sram_data_read(0);
        led(1) <= sram_data_read(1);
        led(2) <= sram_data_read(2);
        led(3) <= sram_data_read(3);
        led(4) <= sram_data_read(4);
        led(5) <= sram_data_read(5);
    elsif ctr="010" then
        -- step 3 WE controlled write to memory
        -- with OE low during write cycle
        iobuf_t <= '0';
        sram_addr <= "00000100001" & write_addr;
        sram_oe <= '0';
        sram_ce1 <= '0';
        sram_we <= '0';
        sram_data_write <= "111111111" & write_addr;
        ctr <= "011";
    elsif ctr="011" then
        -- step 4 clear everything and disable sram
        iobuf_t <= '0';
        sram_addr <= "00000000000000000000000000";
        sram_oe <= '0';
        sram_ce1 <= '1';
        sram_we <= '1';
        sram_data_write <= "0000000000000000";
        ctr <= "000";
    else
        ctr <= "000";
    end if;
end if;

end if; -- rst='1'
end if;
end process;
end Behavioral;
C.4.4 ramtest.ucf

# lab8.ucf for sdram lab

NET "clk_in" LOC = "T9" | IOSTANDARD = LVCMOS33 ;
NET "clk50_in" LOC = "T9" | IOSTANDARD = LVCMOS33 ;

# UART junk
NET "rxd_in" LOC = "T13" | IOSTANDARD = LVCMOS33 ;
NET "txd_out" LOC = "R13" | IOSTANDARD = LVCMOS33 ;

# button0 for resetting the whole ball of wax
NET "reset" LOC = "M13" | IOSTANDARD = LVCMOS33 ;

# LED for telling when DCM is locked on
NET "led_out<0>" LOC = "K12" | IOSTANDARD = LVCMOS33 ;
NET "led_out<1>" LOC = "P14" | IOSTANDARD = LVCMOS33 ;
NET "led_out<2>" LOC = "L12" | IOSTANDARD = LVCMOS33 ;
NET "led_out<3>" LOC = "N14" | IOSTANDARD = LVCMOS33 ;
NET "led_out<4>" LOC = "P13" | IOSTANDARD = LVCMOS33 ;
NET "led_out<5>" LOC = "M12" | IOSTANDARD = LVCMOS33 ;
NET "led_out<6>" LOC = "P12" | IOSTANDARD = LVCMOS33 ;
NET "led_out<7>" LOC = "P11" | IOSTANDARD = LVCMOS33 ;

# 4 push button inputs
NET "pb_in<0>" LOC = "M13" | IOSTANDARD = LVCMOS33 ;
NET "pb_in<1>" LOC = "M14" | IOSTANDARD = LVCMOS33 ;
NET "pb_in<2>" LOC = "L13" | IOSTANDARD = LVCMOS33 ;
NET "pb_in<3>" LOC = "L14" | IOSTANDARD = LVCMOS33 ;

# SRAM ports
NET "sram_ce1_out" LOC = "N5" | IOSTANDARD = LVCMOS33 ;
NET "sram_ub1_out" LOC = "R4" | IOSTANDARD = LVCMOS33 ;
NET "sram_lb1_out" LOC = "P5" | IOSTANDARD = LVCMOS33 ;
NET "sram_we_out" LOC = "G3" | IOSTANDARD = LVCMOS33 ;
NET "sram_oe_out" LOC = "K4" | IOSTANDARD = LVCMOS33 ;

NET "sram_a_out<0>" LOC = "L3" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<1>" LOC = "K5" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<2>" LOC = "K3" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<3>" LOC = "J3" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<4>" LOC = "H4" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<5>" LOC = "H3" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<6>" LOC = "G5" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<7>" LOC = "E4" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<8>" LOC = "E3" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<9>" LOC = "F4" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<10>" LOC = "F3" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<11>" LOC = "G4" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<12>" LOC = "L4" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<13>" LOC = "M3" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<14>" LOC = "M4" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<15>" LOC = "N3" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<16>" LOC = "L5" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<17>" LOC = "P2" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<18>" LOC = "N2" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<19>" LOC = "M2" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<20>" LOC = "K1" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<21>" LOC = "J1" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<22>" LOC = "G2" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<23>" LOC = "E1" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<24>" LOC = "D1" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<25>" LOC = "D2" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<26>" LOC = "E2" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<27>" LOC = "G1" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<28>" LOC = "F5" | IOSTANDARD = LVCMOS33 ;
NET "sram_a_out<29>" LOC = "F5" | IOSTANDARD = LVCMOS33 ;
NET "sram_io<12>" LOC = "C3" | IOSTANDARD = LVCMOS33 ;
NET "sram_io<13>" LOC = "K2" | IOSTANDARD = LVCMOS33 ;
NET "sram_io<14>" LOC = "M1" | IOSTANDARD = LVCMOS33 ;
NET "sram_io<15>" LOC = "N1" | IOSTANDARD = LVCMOS33 ;
Bibliography


