WP200 (v1.1) November 14, 2003

Using Spartan-3 FPGAs As Low-Cost Controllers for Remote Digital Cameras

By: Darrell Wilburn, Dan Hafeman, Al Rogers, and Helen Yu

The introduction of Spartan-3™ devices has created multiple changes in the evolution of embedded control designs and pushed processing capabilities to the "almost-free stage." With these new FPGAs falling under $20, in volume, with over 1 million system gates, and under $5 for 100K gate-level units, any design with programmable logic has a readily available 8- or 16-bit processor costing less than 75 cents and 32-bit processor for less than $1.50.

This white paper explores the benefits, system requirements, cost, design process, software and hardware architecture, and expansion strategy, along with many details of these systems.

Benefits

Today's System Design and business requirements create tremendous challenges to produce products quickly and at lower cost than competitive products. It is now common to require that a project deliver an entry-level product in four to six months for an early adopter program. Achieving first mover advantages in a marketplace has been shown to dramatically impact product success. Marketing considerations may even render a later entry economically unfeasible, causing termination of the project. Given this aggressive design environment, the new Spartan-3 SOC (system on a chip) on a platform FPGA offers significant benefits:

- Complete hardware and software solution available in four to six months because of the advantages of parallel design flow and late-in-project hardware adaptability
- Enhanced testing and verification
- Multiple processors for peripheral acceleration easily achieved
- Simplicity to offload the OS software to avoid peripheral bottlenecks
- Incremental hardware updates enabling the solving of field timing problems in hardware
- Flash memory write circuits that allow remote configuration and program memory updates

Spartan-3 cost advantages are the direct result of higher densities and larger wafer sizes in the Spartan-3 production process. The new cost structure enables low-cost application advantages with almost free MIPS, wherever a PLD device is used next to a RISC processor – with or without DSP.

Figure 1 illustrates the application chosen for the example system design discussed in this publication.

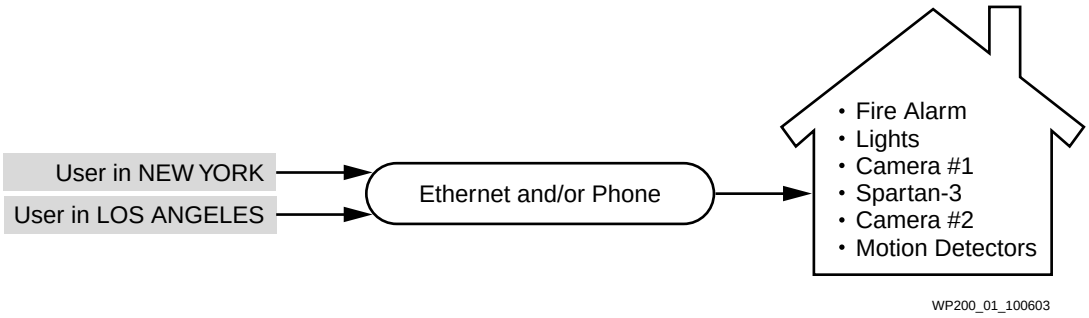

Figure 1: **Example System Design**

The application is a security camera and control system that provides access from anywhere on the Internet, using any standard browser. From a remote site, access is provided to a home, cabin, or business facility via a password-secure internet connection. Once connected, the user can snap pictures from various low-cost cameras and view them at will. Furthermore, a user may activate controls such as turning on or off lights with verification of the results via cameras or other sensors. The user may view sensors such as temperature, water flow, or electrical usage. The system is able to contact the user and others, such as police and fire departments, as a result of activity in the house from motion detectors or fire alarms. Since many remote areas do not have broadband, the application requires dial up modem access over POTS (plain old telephone systems). In the future, wireless access and more controlled elements will be added.

Given the vision and promise of the new technology, how do engineers begin to take advantage of these facilities? What new design flow is required to exploit the technology? How is the new design flow dramatically less expensive and faster to market? What are some design techniques that can be applied economically at these new levels of integration? The remaining sections address these issues through a design example.

System Requirements

Cost

The low-cost security camera and controller was derived from marketing requirements for a camera product for the coming Christmas season. The unit must be small, on the order of 4" X 6" X 2", and low cost, on the order of $40 per system at volume of 5000 per month. Current products in the market with less capability are priced from $199 to $399 and do not support home controls. The intent is to sell this product with USB web cameras at under $120 per unit, yielding a 2 or 3 to 1 price advantage, which will certainly make it a winner.

Notes: there is no need to support full-motion video for most security applications, but full-motion video will be provided as a high-end option in the future. The product cannot be implemented using a personal computer, because of reliability and cost issues. It must be compatible with low-cost USB-controlled cameras in the range of $20 to $40. The system must be robust, with automatic recovery from power failure, and must provide a watchdog timer capability to recover from any random failure in a few seconds time.

Communications

Communications interfaces will include high-speed Internet via an RJ45 connector or dial-up service via an RJ11 connector over any PPP telephone connection. This enables the product to be easily connected into a home with Ethernet already installed, but also provides access via dialup for the majority of home environments where no broadband capability exists. In the future, wireless broadband (802.11a,b,g) will be supported. Communication over any ordinary phone line requires a modem capability as an option and system capability to handle dial up protocols. The system must be capable of automatically dialing out to send emergency e-mails or instant messages to a small number (five) of locations. The system must implement an Internet server to support both broadband and dial up. A minimal TCP/IP stack capable of hosting a simple web page, sending emergency e-mails using push technology and receiving product upgrades over the Internet are required.

User Interface

A simple user interface will be provided via a standard web page usable by a relatively inexperienced web novice anywhere on the Internet. The password-protected page will be accessible via any standard web browser running on a PC. The web page guides the owner through the operations of picture taking, zoom, pan, and remote object activation. In addition to password protection, the product can also be configured to qualify accesses based on a short list of allowable URLs to provide even more security. Of course this requires the accessing machine to have a fixed IP address and a more sophisticated user capable of determining that address and programming it into the Camcon.

Controls

The product must provide control outputs for up to 256 standard X10 devices, which are controlled via power line modulation. Up to 16 buffered TTL-level outputs will be provided to add controllers and interfaces to other controllers in the future. Four of these outputs will be used for each camera with positioning features for pitch, roll, yaw and zoom features. Other possible uses might be ringing an alarm or turning off a main water valve, etc. Up to 16 buffered TTL-level inputs will be provided for inputs from motion detectors, fire alarms, and other sensor input. Analog inputs are not required.

Upgradability

Field upgradability is a major system requirement for both hardware and software. Specifications are evolving with follow-on products scheduled over the coming months

requiring both FPGA hardware and software upgrades. To minimize total product cost, the user alone should accomplish the upgrade without factory support in more than 95 percent of cases. Thus, all products must work on one of a small number of available configurable hardware platforms automatically detectable by the downloading software. In the future, new products may be sold with a single field upgrade of software and hardware configuration only. An FPGA will be used to implement the core of each configurable platform. All digital electronics, including the embedded microprocessor system, will be implemented in the FPGA. The FPGA hardware must contain upgrade circuitry to allow incremental hardware updates. As discussed later, this includes FLASH memory write circuitry to allow FPGA configuration and instruction memory update.

Future Performance and Features

Current acceptable camera performance is about six pictures per minute. This is adequate to detect motion. In future product enhancements, however, MPEG video will be supported enabling the broadband user to remotely access live video feed. DSP functions will be added enhance pictures and extract additional information. Lighting compensation, automatic motion detection, or missing object verification are all potential future features in the video arena. Annotations, time stamping, picture enhancement, and battery backed-up operation are all expected future enhancements as well.

This list below summarizes system-level requirements:

- Low cost, at $40 in quantity of 5000/month
- Small size, 4 X 6 X 2”
- Less than half sales price of current products on market
- Compatible with low-cost USB cameras
- Power failure and watchdog
- RJ45 Ethernet and RJ11 dialup
- Web server with push technology for dial out
- Simple user interface via web browser
- Password protection
- Control up to 256 X10 devices
- 16 outputs for level controls
- 16 inputs for levels from sensors
- Remote upgradability of both HW configuration and firmware
- Future DSP, faster pictures, and enhancements

X10 Controller

The X10 controller is a specialized control protocol for controlling devices in the home over its power lines. While not an official standard, it has become the de facto technology for low-cost home control. The X10 Platform Basics are summarized on the X10 protocol website. Please access www.x10pro.com to learn more.

From the remote camera controller point of view, the most important element of X10 is that it is the market-dominant technology in home controls, largely as a result of its long history and low cost. Newer technologies have failed to meet the price/performance points effectively served by X10. The remote camera/controller will receive X10 commands via a remote web page user input. CGI (common gateway interface) programs will be utilized to translate a command into zero crossing bursts of high frequency over 47 cycles of the power line. The most difficult system requirement is that the bursts must begin within 50 µsec of the zero

crossing, without fail. This constitutes a hard real-time requirement and scheduling issue for any system required to supply the interface. In like manner, the controller must report status back to the web page by monitoring other valid X10 signals on the line and report the resulting activity. Some command queue management may be required in high use and power fail recovery conditions.

X10 controller requirements are listed here:

- X10 controls appliances over power lines
- Dominant in home controls
- Controls up to 256 devices
- Two-byte commands received via CGI and translated to 47 bursts
- Hard real-time requirement within 50 µsec of zero crossing
- Demodulate X10 signals on the line
- Command queue management
- Watchdog transmit on failure

Implementation Overview

Figure 2 and Figure 3 show block diagrams of the system board and FPGA-based logic design. The design is based on a MicroBlaze embedded processor resident in the FPGA. Because the computer resides in the FPGA, it can be enhanced in the future to support additional features like MPEG video. In addition, problems can be resolved by either changing software and/or *hardware in the system.* This flexibility allows for elegant solutions to real-time embedded software timing problems. These upgrades will be available for download by the owners of the product.

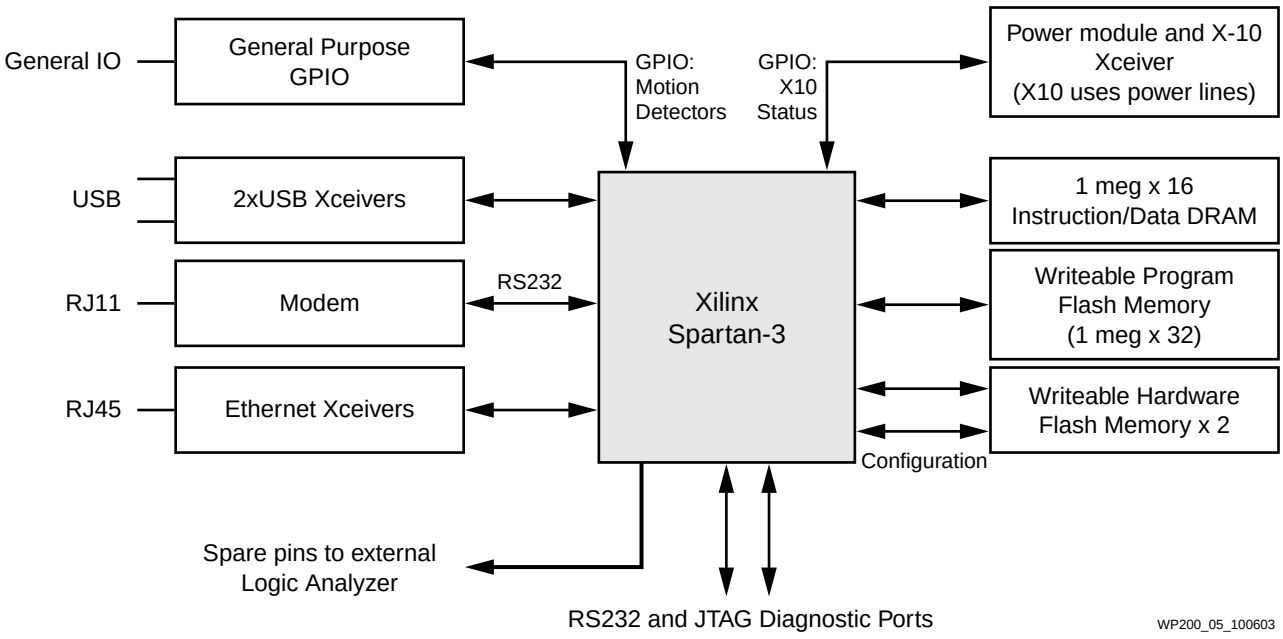

Figure 2: **Block Diagram of the System Board**

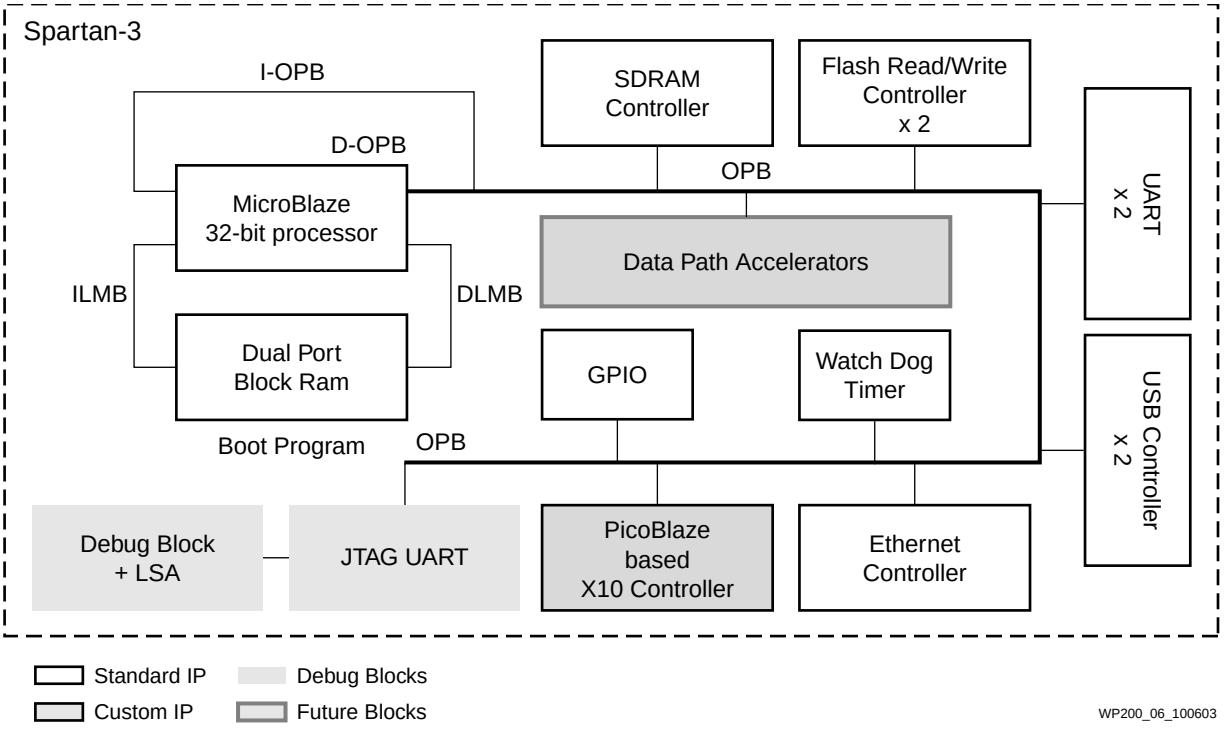

Figure 3: **Block Diagram of the FPGA-Based Logic Design**

Design Process

As can be seen from Figure 2, all digital logic, except for the modem and transceivers, reside in the Spartan FPGA. The PWB (printed wiring board) consists only of memory, connectors, transceivers, and modem. It can be developed in parallel with both the hardware and software designs. Most importantly, the hardware functionality can be changed in the field without upgrading the PWB, provided that the upgrades don't require additional IO connector resources. Because the PWB itself is simple and generic, there is low probability that a design-induced bug will cause the PWB to require any additional design spins. There may be spins resulting from manufacturing issues, but these can be implemented at convenient times and shouldn't impact the project schedule

Another benefit of implementing the logic on a configurable platform is that the hardware design itself can be changed throughout the project with no fab delays, provided that the PWB is available before the first design is ready for testing. Once a version of the design is complete, it can be in the system and ready to test in minutes. There will be no ASIC fab or PWB fab turnaround delays. Because of this, the team can implement the hardware incrementally, much like software teams do. We envision a hardware design flow as follows:

- Implement a working demonstration MicroBlaze system with only on-chip resident program and data memory
  - Run a simple program to verify that the configuration FLASH PROM is functional, the JTAG interface is functional, and the development tools are operational
- Implement the MicroBlaze processor system with minimal IO
  - This will enable the software team to do RTOS bringup and start the testing of IO-independent tasks
- Implement the networking interfaces
  - The software teams can now test the network stack and develop the web page; in addition, configuration software can be developed and tested

- Implement the USB camera interfaces
- Implement the X10 controller

As the software is ported to the platform, the project management has the option to address problems in either software, hardware, or both. For example, to relax real-time software constraints, IO devices can be given additional intelligence by use of PicoBlaze state machine processors. The decision can be made at any time during the project, unlike conventional hardware where specs are essentially frozen before software development starts. This is known as a software-driven hardware design flow.

Even though hardware turnaround times are fast and the FPGA debug facilities are quite good, this project will make extensive uses of simulation for all new logic. Many capabilities of software simulation, such as coverage analysis and boundary case testing, don't exist on the FPGA. These simulations help to produce a more robust design.

The entire Camcon application can be implemented on a standard personal computer. If it were not for cost, reliability, and physical size issues, we would have chosen a notebook computer as the Camcon platform. However, the PC presents a highly functional user-friendly emulator of the Camcon product. Our intent is to use a Linux PC equipped with an X10 interface and a second Ethernet port as the primary software development platform. Most new software programs will first be tested in this environment before being ported to the Camcon platform. The PC will also be used to evaluate cameras and implement their drivers. The software is implemented in C/C++ and will be compiled native on the PC. Thus there is no need for an instruction set accurate model of the MicroBlaze running on the PC.

Software Architecture

The system software components and inter-task communication paths are illustrated in Figure 4.

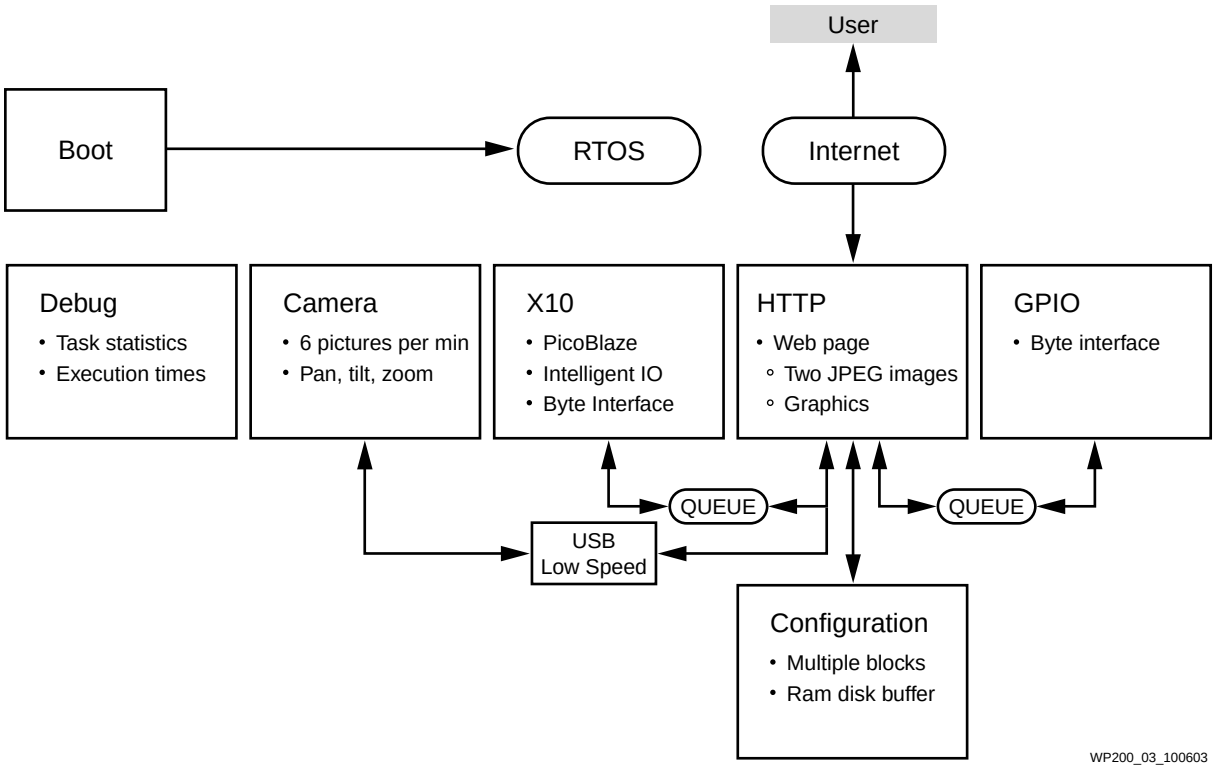

Figure 4: **Tasks and Communications**

The most important aspect of the architecture shown in Figure 4 is the simplicity resulting from independent task functionality with only minimal inter-task data flow requirements. Using a PicoBlaze processor to implement the X10 controller, the system eliminates the need for preemptive capability and greatly simplifies the software task (discussed further under X10 Controller Task).

Software components

The software components include:

- Real-Time Operating system
- HTTP Engine Task
- Camera Controller Task
- X10 Controller Task
- General Purpose IO Control Task
- Debug Task
- Configuration Task/Function
- Boot Program
- Intertask Data Flow
- Web Page
- CGI Command Programs

Real-Time Operating System

While implementation of the X10 with PicoBlaze makes OS requirements minimal, a real-time operating system was included for several reasons. To succeed on an aggressive software schedule, inclusion of the OS provided a quick project start using already debugged modules. A large set of existing software modules were found to be available with established code bases. These included TCP/IP stacks, web servers, virtual file systems, and HTML compilers. Small memory size (2KB) with minimal functionality meant the OS would not require significant memory. Most importantly, if some control requirement in the future requires preemptive capability, it can readily be provided.

After considering several operating systems and package combinations, Chron-OS, a µC/OS-based embedded operating system from InterNiche, was selected for the remote camera/controller primarily because it comes with a complete set of proven integrated packages for µC/OS, a TCP/IP stack, virtual file system, full featured web server and an included HTML to C compiler. Since it is based on µC/OS-II, real-time OS-aware debugging facilities are readily available for real-time testing. The software development time to port to MicroBlaze was minimal compared to porting and integrating other modules or compared to writing the software from the ground up.

Software System Trade-offs in Detail

Using a commercial operating system, custom operating system or no operating system at all becomes a major decision for a product such as the camera/controller. This decision quickly becomes architecturally complex. If the hard real-time requirement is left in the main MicroBlaze processor, it will require a pre-emptive capability in the RTOS to insure service is provided within the hard real-time 50 µsec window in all cases. It would be possible to ignore this requirement and accept a small statistical error rate that would be unpredictable and indefinable over various operating environments. It was decided not to accept an error rate early in the project, even if it was considered low. If the real-time task can be off loaded from the host with a hardware accelerator, the OS requirement becomes nonpre-emptive. Furthermore, given a well done partitioning in the hardware acceleration, we may eliminate the need for a commercial RTOS altogether. The RTOS becomes a simple-to-understand execution loop. Another alternative would be to purchase a simple multitasking scheduler that would sequence through the tasks using commercially well tested task switching capabilities, thus freeing this project of the task of creating and debugging that code as well.

Eliminating the RTOS would save some code space in the beginning but may result in more space utilization later in the project. The specifics of actual RTOS size and complexity normally drive this decision. However, the short time to market, risk, and cost of producing custom system software overrode other considerations. This was true even if the simple control loop is chosen for the system software, as it, too, must survive real-time interrupts and task switching in its simplest form. Once the decision to allow off-chip memory was made (as detailed below), the per-unit cost of RTOS is extremely low. The advantages of having a real RTOS up and running quickly were considered more important. Also, considering the changeable nature of the product in the field, it was considered a great risk reduction to have the rudiments of a fully preemptive RTOS in the product in case it is needed in the future. A full-blown RTOS was included in the design.

Finally, a decision must be taken on whether to write custom software for system tasks or buy as many modules as practical for integration into the product. At one extreme all needed software could be written in assembly and force fit into the on-board BRAM. At the other extreme, all but the camera controller over USB and the X10 controller might be purchased and ported on to the MicroBlaze platform. Integration of the various pieces must be considered as a major task unless a fully integrated package can be found. Since the porting task was

estimated to be three to four weeks, an easy decision was made to buy as much commercial software as possible and do the porting. All totaled, the purchased software packages saved about four man-years of effort compared to a "from-scratch" effort.

Memory Size

Spartan-3 devices have a sizable array of on-board block RAM that is sufficient for many real-time control tasks with minimal software stores. A major trade-off arises between squeezing down code space, even assembly language implementation, to use only on-board memory. In this approach, user interfaces must be kept small and economical. The above approach must be compared with using commercial software that is usually larger in size with more robust user interfaces requiring more graphics. Using the code estimates shown in Table 1, it was clear that it would be possible to use only on-board RAM. However, the software development task would be about four or five man-years. Furthermore, recoding standard items like the TCP/IP stack would introduce significant technical schedule risk. The cost of external DRAM in volume is extremely low in dollars and board space. Early in the system design, it became apparent that adding a single memory chip for $2 to $3, in volume, would eliminate the code packing problem and make possible the use of purchased software packages. It became apparent that the user interface would be skimpy and likely not acceptable to consumer users if constrained to smaller on-chip memory space. It did not make sense to put a hard limit on the user interface this early in the project. Therefore, the decision was made to include the addtional off-chip memory.

Table 1: **Code Space Estimates**

Code Block	STD "C"	Packed
TCP/IP	12KB	6KB
RTOS	2-4KB	1KB
WebServer/HTTP	11KB	5KB
VFS	5KB	2KB
CGI	1KB	1KB
CGI Applications	15KB	2KB
Other Apps/DBG	1KB	1KB
Web Pages	50-100KB	6KB
Total	**97-149KB**	**24KB**

HTTP Engine Task

The HTTP engine task is illustrated in Figure 5. The main HTTP engine function is to serve web pages to users over the Internet and parse requests for command actions that are returned via the CGI (common gateway interface). The engine must also be capable of "push" operations to send unsolicited messages to users via e-mail.

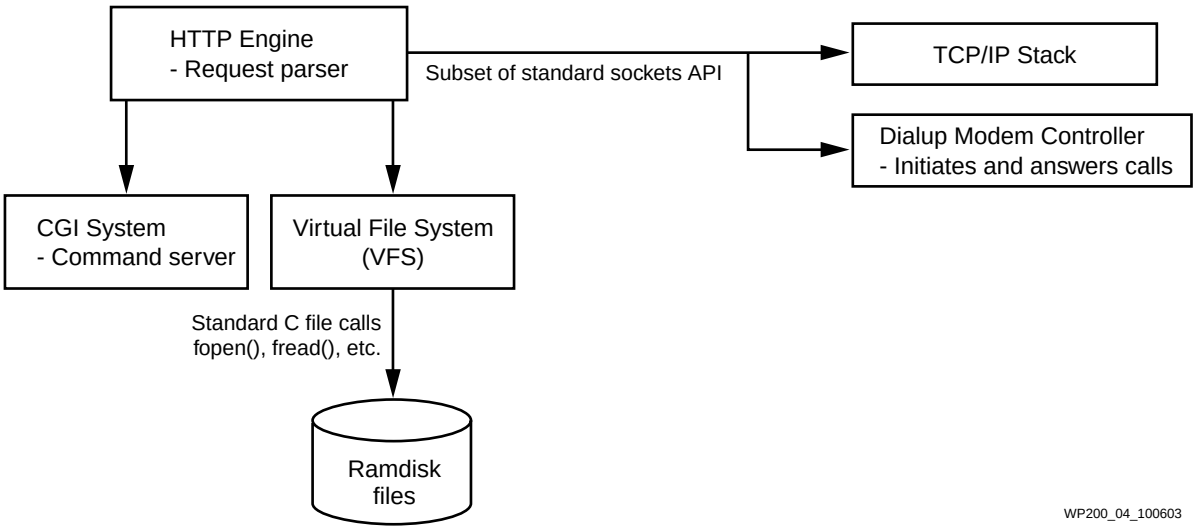

Figure 5: **HTTP Engine Task**

The Ethernet interface can be a subset of a standard sockets API, so memory can be saved by eliminating the sockets protocol. In that case, the CGI programmer must deal with a primitive send message – receive message protocol. The HTTP engine must support a dial-up modem interface to service customers who do not have broadband access. In this configuration, the web server must be capable of being an ISP on the Internet. It must listen for a PPP call and send a base page in response.

The CGI system functions to serve commands to various execution routines. When a CGI input arrives, the HTTP engine must recognize it, send it to the command server which in turn will call the appropriate execution routine to take pictures, turn on lights, or send status back to the user in a new web page.

The virtual file system (VFS) is included on a RAM disk. It allows storage and retrieval of files by standard C file calls. For web page storage, the file system allows standard HTTP anchors and references to files that will be automatically processed by the HTTP engine. As can be seen from Figure 3, the HTTP engine is the heart of the Camera/Controller. As command server, it communicates all inter-task data either from or to the HTTP engine. As described under the camera task, JPEG pictures from the camera can be stored on the RAM disk's relatively high-speed memory and subsequently can be accessed directly by file name from a web page for transmission over the Internet. The server, HTTP protocol, and virtual file system take care of all the details.

Camera Controller Task

The camera controller will send commands over two USB 1.1 connections and receive pictures in return, using standard USB camera protocols. Received pictures will be stored on RAM disc for later access by web pages. Also, the camera controller will send control signals to the camera to control motion for roll, pitch, yaw, and zoom.

In the future, the camera controller will use data path acceleration hardware to process 60 pictures per minute, save pictures in a ring buffer, perform lighting compensation and annotations. The on-board multipliers will be used for direct cosine comparisons between images to detect motion and reference patterns to detect alarms. Eventually, the camera digital electronics may be incorporated into the controller, as well.

X10 Controller Task

The X10 controller task illustrates a common system-level problem. The X10 task must respond by sending or receiving a higher frequency burst within 50 µsec of the 60-cycle power zero crossing, a hard real-time requirement. The burst must complete within 200 µsec. During that time, the controller must verify that it is a "correct" X10 burst, detect collisions, execute back-off protocol and verify that the burst is the correct number of cycles. If implemented in the main control processor, this means that other tasks running must be interrupted by a higher-priority level. Suppose, as is very common, another communication task requires 60 µsec of uninterrupted service to establish a high-speed link. The communication will happen only infrequently in this system, but eventually the X10 service will fail sporadically when the two tasks collide. Solving this problem with a classic preemptive operating system is straightforward, but it will be complex and there will always remain a finite probability of failure. Partitioning the problem with the addition of the PicoBlaze adds concurrency. The PicoBlaze can service the 60-cycle line within 50 µsec to either output or read a data bit. Communication between the PicoBlaze and MicroBlaze is reduced to 2 bytes to send or 2 bytes read from the line. Using a MicroBlaze Controlled data clock solves the issue of process master and allows the master processor (MicroBlaze) to be interrupted at any time without interference with the X10 controller. The X10 process can be run as the lowest priority on MicroBlaze and make no real-time constraints on the operating system. The interface between MicroBlaze and PicoBlaze is a 4-wire serial data interface consisting of data in, data out, data clock and reset/start. Figure 6 illustrates the interface.

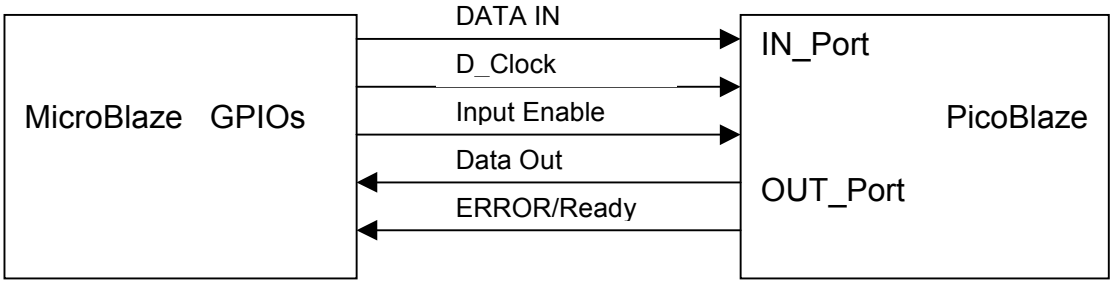

Figure 6: **MicroBlaze – PicoBlaze Bit Serial Interface**

As shown in Figure 3, the X10 task is implemented using a PicoBlaze, which only needs to pass a few bytes of data to the web page in the Engine/HTTP task. Data is passed in a standard message queue to and from the X10 task. The RTOS can now schedule the tasks as a simple round robin, time slice, or as full pre-emptive scheduler, if desired.

General-Purpose IO Control Task

As more devices become available in the future, both hardware and software upgrades will be produced. There are eight general-purpose outputs and 16 general-purpose inputs that may be used as utilities until a specific device is added into the system. Initially, the unused output lines will be treated as independent bits that may be set high or low, on command. The 16 unused

input lines will be polled about 10 times per second and displayed individually. Status will be displayed on the web page. As specific fire alarms and motion detectors are added to the system, users will be able to specify the device configuration via their web pages, where appropriate displays will be downloaded, as well.

Debug Task

As shown on Figure 4, the debug task is a task-level debugger. It provides TCP/IP stack data, task control block information, and task statistics like execution times and memory use. It is intended for only limited software development. When running, the debug task uses an RS232 port for communication with the user. The other RS232 port is reserved for modem use.

Configuration Task/Function

The configuration task is a special task capable of changing the entire hardware and software system. As such, it requires special care to verify that the downloaded configuration and new program code are correct. Extra consideration is required to provide a user error recovery mechanism by which a user in the field can sustain a communication or apply user error which kills the product. A special recover or special reset switch will be included to cause the Camera/Controller to boot up using its previous configuration and program.

Upgrades for both software and hardware will be received over the Internet and will update the appropriate FLASH PROM. The basic mechanism will be to download and store configuration and Programs in blocks of 64 KB, using the RAM disk for temporary storage. FLASH programming is a relatively slow process requiring more time than data download. Fortunately, the data download speed and buffering are automatically regulated by the RAM disk buffer manager and the TCP/IP protocol. This is another major advantage of using integrated system software.

After downloading is complete, the reconfiguration task initiates a reboot with the new hardware configuration and software. If an error had occurred in either load, the system would fail in a nonrecoverable mode. The program FLASH memory is exactly twice the size of the maximum possible program store. The result is that a catastrophic failure on download can be recovered by the user simply by rebooting via the special reset switch on the product.

Boot Program

BRAM (Block RAM) attached to the IOPB and DOPB busses contains the initial boot program. It will be placed there during the power up sequence when the FPGA is configured from FLASH. The Boot program will initialize the processor, run a set of diagnostics on the processor system, and copy the appropriate contents of program FLASH into SDRAM. Then it will jump to the start location of initialization routines for the OS in SDRAM and begin executing the RTOS. Since FLASH memory is large enough to hold both the current program configuration and also the previous one, the user can recover the previous configuration by setting the recover switch.

Intertask Data Flow

As illustrated in Figure 4, all major data paths flow between the HTTP task and some other task. The main web page and form are estimated to be 54KB to 154KB in size and reside entirely within the HTTP task on RAM disk. Access will be via the virtual file system. The main page (Index) and form are expected to be about 8KB. Two JPEG images must be passed from the camera to the HTTP task with size estimates of 10KB to 100KB. Graphics for page illustration and prompting are estimated to be from 10KB to 50KB. The data pass is shown as shared memory, possibly with a semaphore for co-ordination, although the virtual file manager may provide all the buffering needed. Finally, camera commands and status must be passed

between the HTTP and camera. These are 2- to 4-byte commands that will follow standard USB camera requirements. Also, as described earlier, four hardwired control lines extend to the camera mount to control positioning. These are not shown in the software figures.

X10 commands and status require only a small number of bytes and will be supported with a message queue from the RTOS. GPIO is from 3 to 4 bytes of data and will be handled in a standard message queue, as well.

Web Page

Web pages may be served containing HTML text files with all supported tags from HTML 2.0 and above. Pages may include forms tables, frames, graphics, Java applets, and all other features of the Web. Server side include (SSI) commands may be embedded in the HTML of the page. A maximum of two web page connections will be supported at any time. The original product may restrict usage to one connection if implementation becomes too complicated.

CGI Command Programs

Pages may invoke CGI scripts from forms, which essentially divert replies from GET and POST commands to CGI code. References may be to any file name where the files are expected to be on the virtual file system. Parameters are passed from the form appended to the filename as a string of encoded text. In normal operation, the parameter string is not expected to be more than 10-15 bytes long. Picture data will simply be served as a JPEG image with the response page to a user. In operation, all camera control, X10 and GPIO operations, will be executed from CGI programs stored as executable files.

Hardware Architecture

Figure 3 shows the block diagram of MicroBlaze processor system which resides in the FPGA. It is a conventional computer configuration based on the Xilinx OPB bus. Note that most of the blocks are standard IP devices available from Xilinx and others. Because of this, it will be possible to get a basic system running very quickly. Our plans call for providing a basic execution vehicle to the firmware team as soon as the board is available. It will include the MicroBlaze, boot ROM, SDRAM Controller, Flash Read/Write Controller, Timer, and UART. Other components will be added in later iterations. All data path accelerators, other than the PicoBlaze controller in the X10 controller, will come on board through later product design after the Camcon is shipped.

OPB Bus System

The OPB Bus forms the heart of the Camcon design. It is a standard Xilinx component that connects all peripherals to the MicroBlaze processor. The bus also serves as the Instruction/Data bus for main memory which is implemented in an external SDRAM chip. The processor D-OPB and I-OPB ports are connected directly to the OPB bus. Since our design currently does not use DMA, the only masters on the bus are the MicroBlaze interfaces.

MicroBlaze, being a Harvard processor architecture, has separate Instruction and Data bus interfaces which enables it to simultaneously access data items and instructions. In a high-performance processor application, engineers often attach data and instruction secondary caches to the two ports, respectively. The caches usually have a common bus interface to main memory. Since the Camcon computing requirements can be met without employing secondary caches, the I-OPB and D-OPB ports can be conveniently connected to the OPB bus. Primary instruction and data caches will be implemented in BRAM memory. The size of these caches will be set to use most of the remaining BRAM memory after the rest of the system has been implemented.

Some embedded designs separate data and instruction memory into separate memory components. While this again helps performance and prevents the processor from overwriting its program store in the event of a crash, it creates serious memory management problems for the software engineers. It forces software architects to estimate independent program and data memory requirements not only for current releases of the firmware, but also for future releases. It is much easier to estimate total memory requirements for the life of the product without concern for how that memory will be used. In addition, only a single external main memory chip is required, rather than two, and fewer IO pins are required with a single-memory architecture. By connecting the Instruction and Data ports directly to the OPB bus, we also avoid additional debug hardware which enables debugger software to gain write access to the instruction memory. Note that I-OPB is a read-only port. See the appropriate Xilinx application notes on this subject to gain more understanding.

Boot ROM

Dual-port block ram connected to MicroBlaze via the ILMB and DLMB buses serves as the boot ROM for the system. It is initialized at FPGA configuration time and contains a program that copies the contents from the program FLASH to the SDRAM and initializes the data area of the SDRAM at power up. Because block RAM comes as a dual-port memory, it is convenient to connect to it using both boot ports, as suggested by Xilinx.

External Memories

The Camcon board contains three memory chips: hardware flash, program flash, and SDRAM. The hardware and program flash memories must be written by the application to facilitate internet-based hardware and software upgrades. Thus, the Flash memory interfaces are bidirectional.

The hardware flash memory is implemented using a special FLASH PROM with both a serial Xilinx configuration port and a JTAG read/write access port. This type of device is available from Xilinx, Maxim, and others. At power-up, the FPGA controls the serial configuration port to load the current hardware configuration into the FPGA. During runtime, the application software can change the contents of the hardware flash memory by accessing the JTAG UART port.

The EEPROM will initially be programmed in the factory off board the Camcon controller as there is no external write access to the chip when in system, except through the JTAG system. The chip will be socketed on the board.

A tough quality control standard will be required to manage hardware upgrades. If a serious defect were to be included in an upgrade, it could cause the product to fail in a nonrecoverable manner. Only the factory repair operation of physically removing the FLASH PROM and programming it again, externally, could recover the system. Also, the application software will initiate a write to the EEPROM only *after* the entire configuration has been downloaded into ramdisk and checked for errors. This prevents a communications failure from turning into a nonrecoverable fault.

The SDRAM interface is a standard logic block available from Xilinx and others. Because the I-OPB and D-OPB are connected together on the OPB bus, there are no special requirements for the interface. The interface is 16 bits wide and connects to a single 2MB DRAM chip. As the computer system is 32 bits, two memory accesses are required to read or write a 32-bit word. This certainly slows the system, but cost constraints demand only one DRAM chip.

USB Ports

As already discussed, two on-chip USB 1.1 ports are included in the design to interface to the Web cameras. The decision to include the blocks onboard as compared to using inexpensive

USB chips off board was not an easy one. The off-chip components are inexpensive and already debugged. But they can't be upgraded via download. Also, they can be implemented in FPGA fairly inexpensively; requiring less than 2200 logic cells each. Note that at the time of this paper, we were not able to locate a USB block with an OPB connection. So a little logic will have to be added to interface the commercially available USB IP block to the OPB bus. As there is no DMA in this application, the interface should be simple and require few additional resources.

Our analysis showed that even if the USB ports were implemented off chip, a Spartan-3 S1000 would be required for the application. So, aside from the one-time cost of buying the USB IP, there is no additional cost to the product of adding USB ports. Note that by including USB on chip, we can upgrade the hardware in the future and support the higher speed 2.0 standard, if required. Note also that a hardware bug can be fixed if the core is on-chip.

Ethernet Port

We intend to use the Ethernet IP provided by Xilinx for this application. The block is complete with an OPB interface. Our application requires only the 10baseT functionality. Again, there was discussion of buying an external controller. Even with the controller on-chip, an external transceiver chip is required. And since chips are now available which combine the controller with the transceiver, it is possible to implement the controller external to the FPGA without increasing chip count. But it does increase cost and it does eliminate future field upgrades of the Ethernet hardware.

Once again, we decided to include the controller on-chip. Being directly connected to the OPB bus made the software interface simpler and we have the flexibility to make changes to the controller itself in the future, if required. With an area cost of only 3300 logic cells, the controller didn't change the size of the FPGA required, which made the decision easy.

A Note about Debug

With a processor on board, system integration and power-up can proceed on a piece-by-piece basis very easily. The process is very similar to standard board power-up for any processor-based system. This is important as the very first system loaded on the FPGA. Power-up can then proceed by checking out the JTAG debug port, the processor diagnostics, memory interfaces, memory, execution from memory via JTAG, and execution from memory in real time. Then using ChipScope or equivalent, system busses and new IP can be systematically checked out and debugged along with system software and finally applications software. A detailed discussion of the debug process will be the subject of a future white paper.

The onboard debug resources required to probe the hardware themselves take capacity from the FPGA, especially BRAM resources. There are two ways to approach providing the resources for debug in an FPGA that is almost fully utilized by the application itself: 1) use a larger FPGA to emulate the Spartan-3 XC3S1000 part or 2) build several designs, each implementing a subset of the application and thus each freeing the resources for the on-chip debugger. We hope to use a larger FPGA if it is available at debug time. But because the platform is configurable, it is possible to resort to debugging with a subset of the design.

Expansion Strategy

Upgrades and Enhancements

Throughout the system design process, expansion to add new features and even new products has been a major consideration. In the foreseeable future, expansion to digitally process images from the camera to enhance quality, increase frame rate, and extract image-based information are all expected. A frame rate of at least 60 frames per second is anticipated. Functionality for motion detection, alarm recognition by comparing reference images, annotations with time

stamps and MPEG video are all functionalities that may be added in the field that employ DSP hardware acceleration. Eventually, communications over 802.11b wireless and tight firewall security will be needed. More sophisticated security systems such as MD5 or Digest may be added or even key-based secure protocols such as IP/sec.

Finally, since the system is completely reconfigurable and will be supplied with a flexible connector scheme, entirely new products could be created in the field as new devices emerge in the home controls arena.

Field Product Fixes

In the design scenario chosen, software is allowed to drive the design, as shown in Figure 7.

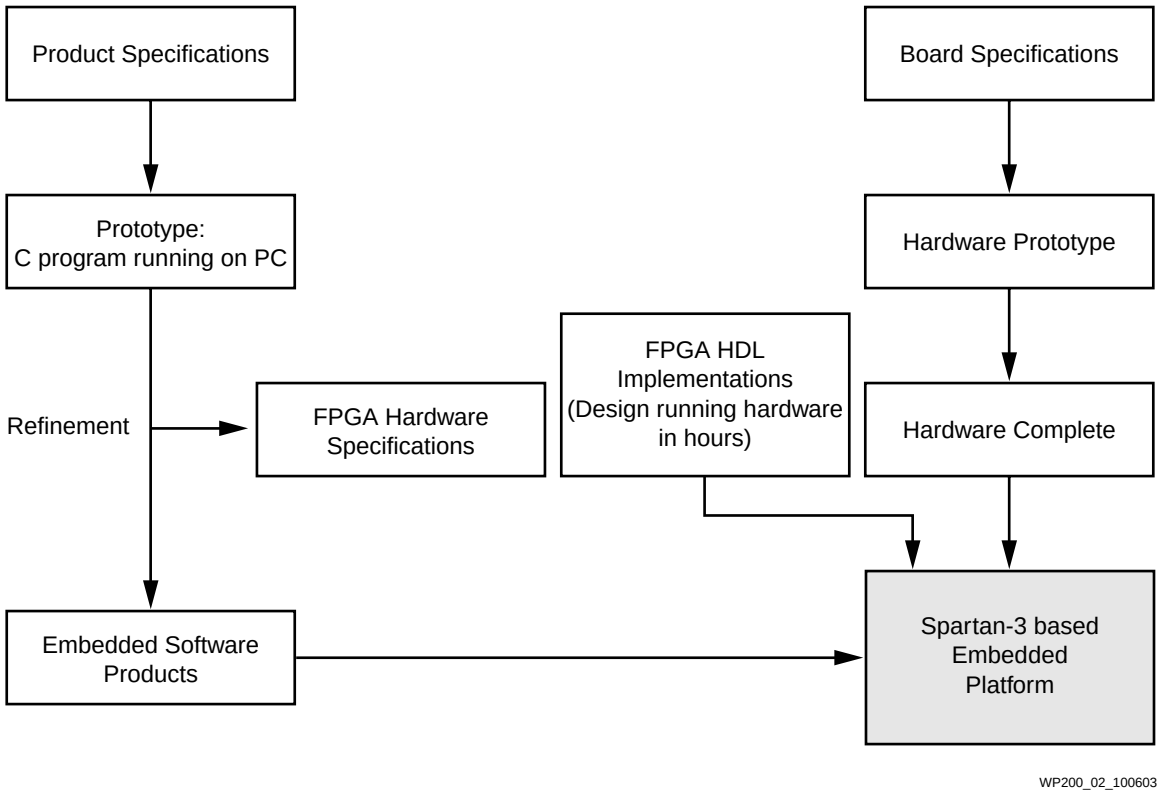

Figure 7: **Software-Driven Development**

Hardware upgrades late in the design cycle, or even in the field, will be driven by software requirements. As new controlled elements are added to the product repertoire, additional features can be optimally accelerated by logic implemented in CLBs, BRAMs, multipliers, and SRL16s. Field fixes and changes can add new data path accelerators at any time. For example, after a product ships, one of the more perplexing problems to solve is an intermittent failure at random times. Often, the problem is the result of a statistical failure from tasking software where the scheduling fails. Or, it may result from a hardware race condition or metastable problem that is discovered only when enough units are shipped to see the statistics of failure. Solving these problems with only software modifiability can be extremely difficult or impossible. Now with the ability to modify hardware and add data path accelerators at will, these problems become relatively simple to fix once they are found. The camera with X10 controller example uses a PicoBlaze processor to eliminate the tight real-time timing problem from the operating system. The same design techniques can be applied to products even after they are shipped. For the simplified design, the MicroBlaze processor is all that is needed to support the TCP/IP stack and Web page protocol.

Conclusion

The following major points summarize system design for the Camera/Controller:

- Project low cost of $41 for controller + $20 for camera
- Field configurable
  - Support new products in both hardware and software, as required
  - Fix field problems in hardware and software
- Software-driven design flow
  - Optimized hardware driven by software speeds time to market
  - Reduces technical risk during short schedule
- Future DSP functionality will use data path acceleration
- PicoBlaze used to meet real-time hardware requirement simplifies software

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/14/03	1.0	Initial Xilinx release.
11/14/03	1.1	Reformatted in standard template.