CHALMERS

Embedded Camera Remote Control

Master of Science Thesis in Integrated Electronic System Design

JOHAN RYDH

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, September 2011

Embedded Camera Remote Control

JOHAN RYDH

Supervisor: MAGNUS SJÄLANDER

Examiner: PER LARSSON EDEFORS

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, September 2011

Abstract

Modern digital single-lens reflex cameras allow remote control access via a bidirectional digital interface, as well as a unidirectional remote shutter control interface. Available products use the unidirectional interface that only allows control of autofocus and shutter, while a PC is required to access any of the camera's settings remotely.

This report describes a prototype implementation of an advanced remote control that uses bidirectional digital communication over USB to access and change common settings. The prototype is developed to support Canon DSLRs, and is tested with an EOS 5D mark II camera.

Hardware and software have been developed; the hardware is based on a template design. The prototype is implemented on an FPGA development board. A custom input device has been developed as primary input to the remote control prototype.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ACE	Advanced Configuration Environment
AF	Autofocus
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
BIOS	Basic Input/Output System
DDR	Double Data Rate
DMA	Direct Memory Access
DSLR	Digital Single-Lens Reflex
DVI	Digital Visual Interface
EEPROM	Electrically Erasable Programmable ROM
FPGA	Field-Programmable Gate Array
GPL	General Public License
GUI	Graphical User Interface
HDL	Hardware Description Language
HDR	High Dynamic Range
HPI	Host Port Interface
I2C	Inter-Integrated Circuit
IP	Intellectual Property
IR	Infrared
JTAG	Joint Test Action Group
MAC	Media Access Control
MTP	Media Transfer Protocol
PC	Personal Computer
PROM	Programmable ROM
PTP	Picture Transfer Protocol
RAM	Random Access Memory
ROM	Read-Only Memory
SDRAM	Synchronous Dynamic RAM
SRAM	Static RAM
TD	Transfer Descriptor
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VGA	Video Graphics Array
VHDL	Very high speed integrated circuit HDL

1. Introduction

Modern digital single-lens reflex cameras, DSLRs, can to a large extent be controlled from a PC using USB. While a PC can be suitable in a studio environment, a custom product may be more practical in the field. User interface, size and battery life are parameters that can be tuned to provide an easy to bring and use advanced remote control, providing access to most features available in the DSLR without risking camera shake or movement. The in-camera features are extended by a timer module to allow time-lapse and preconfigured long exposures, adjusted to the current camera settings.

Mainly exposure related settings are expected to be useful to control remotely. These include the shutter speed which controls the exposure time, the ISO speed which determines the light sensitivity of the sensor and the aperture value. The aperture value controls the light opening of the lens; a large opening allows a greater light transfer and will also narrow the depth of field, while a smaller light opening will extend the in-focus distance range. Another key setting is the white balance, which compensates for differences in color content of the light source, like the sun or a light bulb.

Compared to a classical remote shutter control, the developed remote control does also allow access to the settings described above, and will display information about other settings like the selected image quality. The timer feature is shared with a Canon remote shutter control with timer support; however, it cannot access any camera settings and its user interface is limited by a display showing only one value at a time. The manufacturer's PC software allows access to most camera settings, which has been used as a reference during the project in terms of possible features and protocol details.

Recently, other USB connected remote controls have been announced. One targets videographers by primarily allowing control of the lens' autofocus motor, which allows smoother focusing than manual adjustment of the lens. It does also allow control of a few settings, but with a limited user interface. A beta version of a coming application for Android mobile devices provides a similar set of features as the developed remote control, but may not be suitable for all scenarios or users, as discussed in Chapter 4.

A prototype has been developed, rather than an end-user product that especially would require integration of all hardware components into a user-friendly product. Also, the feature set and range of tested and supported devices can be made wider to make a more competitive product. Such requirements are beyond the scope of this project, which focus on a basic application with drivers running on an embedded processor platform. The project includes both hardware and software development, with an emphasis of low-level software and some application features.

The prototype is developed for and tested with a Canon EOS 5D mark II digital single-lens reflex camera. Analysis of the high-level protocol has mainly been done prior to the project, while some additional analysis and testing have been done using the embedded platform. Basic comparison of the protocol of a few other Canon DSLR camera models has shown minor differences compared to the tested DSLR, implying a likelihood of the prototype supporting those; however, none of them has been tested and minor adjustments may be required for full support.

This report will describe the hardware used and implemented, in Chapter 2. The software running on top, including hardware drivers, will be described in Chapter 3. A conclusion of the project will be given in Chapter 4.

2. Hardware

This chapter will describe the hardware of the Embedded Camera Remote Control, and to some extent driver considerations. The remote control prototype is implemented on an FPGA development board, based on a HDL template design. It uses external user input and output devices, and can be connected to a PC for programming and debug support.

2.1. Hardware Overview

The hardware is based on a template design of Aeroflex Gaisler's GRLIB with a LEON3 processor. The design is implemented on an FPGA development board. It has been customized to include two communication VHDL cores developed during the project, described in Chapter 2.4. A simplified overview of the remote control prototype is shown in Figure 2.1, where the custom cores are marked with dotted boxes.

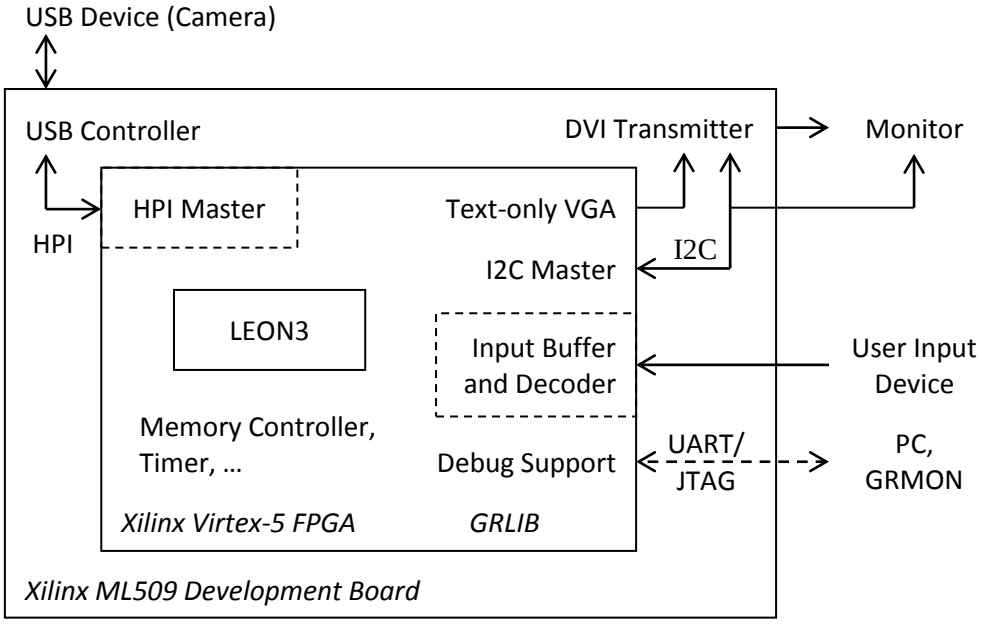

Figure 2.1: Simplified hardware overview

The DSLR camera is a Hi-Speed USB 2.0 device, which is connected to a USB host port. The port is controlled by a full speed USB 2.0 controller available on the development board. This will reduce the maximum bandwidth to 12 Mb/s, including protocol overheads. The transferred data consists of commands and responses that do not require high bandwidth, implying only a limited performance loss with the setup compared to if a Hi-Speed USB host controller would have been used. The main processor communicates with the USB controller over a HPI bus, via a bus master VHDL core developed during the project.

The user interface of the prototype is, due to simplicity, split into an input system and an output system; an external monitor can then be used. User input from a custom built prototype input device, described in Chapter 2.5, is read using a developed VHDL core that buffers and decodes the signals. The graphical output is generated by a text-only IP core, transmitted to a monitor using an onboard DVI transmitter. The prototype can also

be controlled and programmed using the debug link, either with a UART or a JTAG connection to a PC with Aeroflex Gaisler's GRMON software.

2.2. Development Board

The development board used is Xilinx ML509, featuring a Xilinx Virtex-5 series FPGA, while sharing other features with the Xilinx ML505 board [1]. The board includes additional components required for an embedded processor platform, such as clock generator, reset and FPGA configuration support using a Xilinx System ACE device; SRAM, DDR2 SDRAM and flash memories for storing FPGA configurations and other data as software programs. Both FPGA configuration and software are stored on two flash memories and loaded when the board is powered up.

The board does also have several devices and connectors for external I/O, which can be controlled by the FPGA. Of the board's features, the USB controller and a DVI graphics transmitter are especially important for the remote control prototype, which will be described in the following chapters.

2.2.1. FPGA

The FPGA available on the development board is a Virtex-5 XC5VLX110T. The FPGA is designed for high-performance logic and includes serial communication hardware blocks, such as a PCI Express endpoint and Ethernet MACs [2], which are unused by the remote control. The FPGA contains enough logic slices and user I/O pins to support the HDL design.

2.2.2. USB Controller

The onboard USB controller is a Cypress CY7C67300 device, supporting USB 2.0 at low or full speed [3]. The controller supports up to four peripheral USB ports or two host ports, using two serial interface engines. Only one host port, which is used to communicate with the camera, and one peripheral port are connected to the development board's external connectors.

The USB controller has an embedded 16-bit processor, with approximately 15 kB of user available RAM. External RAM or ROM memory can be mapped into to the memory space; however, none is available on the development board. The controller boots into one of four different modes, depending on the logic level of two pins after a reset. The mode is hardwired on the development board and is set to standalone operation, in which the controller is further configured by reading an onboard EEPROM. The other three modes are all coprocessor operation mode, but with different interfaces to the main processor. The development board ships with a preprogrammed EEPROM configuration that will enable coprocessor operation via the Host Port Interface bus, while the standalone mode remains as a possibility.

In standalone mode the controller executes a program with data that has been copied from the EEPROM during booting. Cypress provides both development tools and a software framework containing USB support routines. The limiting factor is the available RAM. Example applications using the framework tend to require close to the amount available, leaving an uncertainty whether the required functionality will be implementable [4].

In coprocessor mode, commands are sent to the controller from the main processor. The commands are handled by the controller's BIOS software, which both reduces the memory usage of the controller and the complexity of the main processor application software by handling most of the low-level details of the USB host port. Only commands and command related data needs to be transferred and stored on the controller. This is the default configuration of the development board, and the one selected for this project. The HPI bus, over which commands and data are transferred, is described in Chapter 2.4.1. Commands and data structures are described in Chapter 3.2.

2.2.3. DVI Transmitter

To output the graphical user interface to an external monitor the Chrontel CH7301C DVI Transmitter on the development board is used. The input is provided by a text-only video controller IP core included in GRLIB implemented in the FPGA. The output is connected to the DVI connector of the board, providing both digital and analog output. Output resolutions up to 1920 × 1200 are possible [5], while the video controller IP core only outputs a 640 × 480 signal with a 60 Hz refresh rate [6].

The DVI transmitter is configured using an I2C bus, which on the development board also is shared with the DVI connector, as seen in Figure 2.1, enabling communication with a connected monitor. The bus is controlled by the main processor using an I2C master IP core; it is used to set up the input clock and data format, power up and enable the transmitter. The monitor is not accessed through the I2C bus, support of the fixed resolution is assumed.

2.3. GRLIB Template Design

The HDL design is based on Aeroflex Gaisler's template design of GRLIB, version 1.1.0-b4104, for the development board used. The IP library is available under GNU GPL. The design contains a LEON3 processor connected to an AMBA bus, and supporting IP cores required for an embedded processor platform, such as clock configuration, memory controller and debug link support [7]. GRLIB does also contain a text-only VGA controller used as primary output for the prototype. The VGA controller displays an array of 80 × 37 characters, and supports hardware assisted scrolling by having a larger character buffer; when a non-displayed line is written to, the hardware will scroll one line [6].

The LEON3 processor is expected to be able to run at up to 90 MHz on the FPGA [8]. The design is synthetized with a target speed of 80 MHz for processor and APB bus. The GRLIB template configuration is mainly changed to use the text-only VGA IP core, and to disable the Xilinx System ACE bus controller IP. The template design has also been customized by adding HDL cores, described in the following chapter.

2.3.1. Customization

Two HDL cores have been added to the design, connected to the AMBA APB bus. The two cores have been assigned to two free bus indexes and the additional FPGA pins used have been added to the user constraint file. On the development board, most of the FPGA pins connected to the HPI bus controlling the USB controller are shared with the bus connecting the Xilinx System ACE controller device. The prototype application does not need to access the System ACE device, and there is no need to implement a

bus arbiter when only one bus core is implemented. The top VHDL module of the design has been modified to enable the HPI master only when the System ACE controller has been disabled by the GRLIB design configuration tool.

2.4. Custom VHDL Cores

The custom cores are connected to the AMBA APB with 32-bit registers mapped into the memory space of the LEON3 processor. The two cores are based on a basic bus peripheral core that handles both bus reads and writes, and are extended with the necessary logic to perform the intended functionality.

The design intention is to keep a balance between hardware complexity and performance, where overall simplicity of both hardware and software is preferred. This implies that an available general purpose I/O core is not optimal, since all processing is done in software, causing unnecessary performance losses. Handling of one HPI bus operation has been selected as a good balance in the case of the Host Port Interface bus master. To handle more than one operation would require queuing of operations and their results, which can be handled in software without a significant performance loss, while the timing of individual data and control signals is hardware based for a single bus operation.

2.4.1. Host Port Interface Bus Master

The Host Port Interface bus allows communication with a HPI slave device. The timing and the interface are adjusted to those of the USB controller described in Chapter 2.2.2, which is available on the development board. The HPI bus can run at up to 8 MHz, providing a data rate of up to 16 MB/s due to the 16-bit wide data bus [3]. Table 2.1 summarizes the signals that the HPI bus consists of.

Table 2.1: HPI bus signals

Name	Width	Description
Data	16	Bi-directional 16-bit data bus
Address	2	HPI port register address, see Table 2.2
$\overline{CS}$	1	Active-low chip select signal
$\overline{RD}$	1	Active-low read enable signal
$\overline{WR}$	1	Active-low write enable signal
Interrupt	1	Active-high interrupt signal from the slave

The HPI bus allows direct memory access of the slave-device's memory. A DMA operation is initiated by writing the 16-bit address to access to the HPI address register. An arbitrary long sequence of only reads or only writes can then follow, using the HPI data register. The address register is automatically incremented to the next 16-bit word after each memory access. The HPI port register addresses are given in Table 2.2.

Table 2.2: HPI port register addresses

Address	HPI port register
00b	HPI data register
01b	HPI mailbox register
10b	HPI address register, write-only
11b	HPI status register, read-only

Commands to the device's BIOS are sent to the mailbox register, preceded by a DMA transfer of any additional arguments. An available response is signaled by the interrupt signal, and can be read from the mailbox register. The mailbox read operation will automatically clear the interrupt signal. The interrupt status can also be checked by reading the HPI status register.

The HPI bus master handles one data transfer at a time and ensures correct timing, as long as it is connected to an APB bus running at 100 MHz at most. Lower bus frequencies are supported, but the transfer rate is then lowered since it is related to the APB bus clock frequency, due to implementation simplicity.

Table 2.3: HPI bus master registers

APB address offset	Register
00h	Status register (read) Control register (write)
04h	Reset register (write-only)

The HDL core is controlled by a control register and checked using a status register. The registers are listed in Table 2.3. The reset register given in Table 2.6 is used to perform a full device reset of the USB controller. Care needs to be taken to ensure correct timing since the reset signal is forwarded to the slave device without any additional logic. The HDL core can be tested by ensuring that the *reset* and *busy* flags in the status register, defined in Table 2.4, are set properly during and after the reset.

Table 2.4: HPI Status register, read operation

31	30	29	28 24	23	22 16	15 0
Busy	New data	Interrupt	*Reserved*	Reset	*Reserved*	Data

Reserved	Reads as zeros
Busy	Set while the HPI master is busy or the *reset* flag is set, cleared when a new command can be written to the control register
New data	Set when the data from a HPI read operation is valid, cleared on any write to the control register
Interrupt	Forwards the USB controller's interrupt output signal
Reset	Set while the HPI reset signal is active
Data	Last read 16-bit word from the HPI slave, valid when the *new data* flag is set

Table 2.5: HPI Control register, write operation

31 19	18	17 16	15 0
Reserved	Data direction	HPI address	Data

Reserved	Ignored, should be set to zeros
Data direction	'0': Start HPI write operation '1': Start HPI read operation
HPI address	HPI port register address, see Table 2.2
Data	16-bit word to write (ignored for HPI read operations)

The *busy* flag should be ensured to be low before any transfer is initiated by the software, otherwise the transfer will be ignored. Table 2.5 lists the fields that should be set in a single write operation to initiate a HPI transfer. Data direction and HPI port address are always required, while the 16-bit data word only is necessary for HPI bus

writes. A HPI bus write does not require any more attention, other than verification that the *busy* flag has been reset by the hardware before the next HPI transfer.

A HPI bus read will cause the *new data* flag of the status register to be set when the read word is available in the status register. Care should be taken to also check the busy flag before initiation of a new HPI transfer, since the read word is available slightly before the HPI bus transaction is complete. The *new data* flag is cleared when the control register is written.

Table 2.6: HPI Reset register, write-only

31 … 1	0
Reserved	Reset

Reserved	Ignored, should be set to zeros
Reset	Active-high slave device reset signal control bit

2.4.2. User Input Device Buffer and Decoder

The VHDL implementation of the core to buffer and decode the signals from user input device is designed to be simple. Input from keys and buttons are buffered one clock cycle to synchronize the signals with the clock, and can then be read via the AMBA bus. The user input device is described in Chapter 2.5. The two Gray coded inputs from the dial are also synchronized, but are not directly accessible via the bus; instead a toggling of one of them will cause a counter to increment or decrement, depending of the corresponding Gray code state transition.

Table 2.7: User input device buffer and decoder registers

APB address offset	Register
00h	Key status register (read-only)
04h	Dial counter register

To access the user input status from the processor, two registers can be read, with address offsets given in Table 2.7. The key status register is a read-only bitmap consisting of five active-high bits corresponding to keys. The layout of the key status register is defined in Table 2.8.

Table 2.8: Key status register, read-only

31 … 7	6	5	4	3 … 2	1	0
Reserved	Shutter	AF	Shift	*Reserved*	Return	Enter

Reserved	Reads as zeros
Shutter	Reads as one while the shutter key is pressed
AF	Reads as one while the autofocus key is pressed
Shift	Reads as one while the shift key is pressed
Return	Reads as one while the Return key is pressed
Enter	Reads as one while the Enter key is pressed

The dial counter register keeps a signed value; the register can both be read and written, as stated in Table 2.9. A bus read returns the current value of the counter, while a bus write will subtract the written value from the counter. This implementation ensures that any dial rotation will be included in the final value even if it occurs just

before or at the same time as a bus write operation, while keeping the software handling the dial state simple.

Table 2.9: Dial counter register

31	0
Dial counter (sign extended)	

Dial counter	Reads the current value, sign extended to 32 bits A written value is subtracted from the counter

The dial counter is implemented as an 8-bit wide signed counter, which is sign extended to the bus width of 32 bits, during a bus read. In software, the counter register should be handled as any 32-bit value, but with the understanding that it cannot handle large values. During normal operation, the register should be checked at a regular basis to ensure a smooth user experience, which also should be frequent enough to avoid any overflowing of the internal counter.

The intended software handling of the dial counter register is to check for an absolute value greater than or equal to a limit, a preconfigured resolution. While the test is false, the dial has not been rotated enough; no further action should be taken by the software. A rotation event is noticed by the software when the test becomes true; the sign of the read value determines a clockwise or counter clockwise rotation. Finally, the counter should be subtracted with the configured limit, but with the same sign as the read value, in order to reset the counter. This method of reading the dial will introduce hysteresis, if the division factor is at least two, avoiding repeated events with minimal movement of the dial when just in between two positions.

2.5. User Input Device

The user input device is a custom design for the prototype, designed to be similar to the user interface of the camera, which uses two dials to adjust settings. The input device has a single dial that together with a shift key enables adjustment of a secondary setting. There is also a combined autofocus and shutter button, of which the autofocus is activated when pressed halfway. Finally, two keys are used to navigate between a few operational modes of the remote control. The function of these keys, Enter and Return, is displayed in the user interface. A photo of the input device is shown in Figure 2.2.

The dial is optically read using IR light and has no endpoints. The dial is encoded using a 2-bit Gray code, which allows for detection of relative rotation, while only one bit can change between any two positions. The current encoder disc used has 32 positions, thus repeating the 2-bit code pattern eight times, providing a resolution of approximately 11°. Some variations of the effective width of the positions exist since the encoder disc used is handcrafted. The noticeable differences can however be reduced by reducing the effective resolution in software, to a more reasonable level; a factor of three is currently used. At the same time, a potential problem of repeated toggling between two positions can be avoided by introducing hysteresis in the software implementation; see also Chapter 2.4.2, describing the software interface.

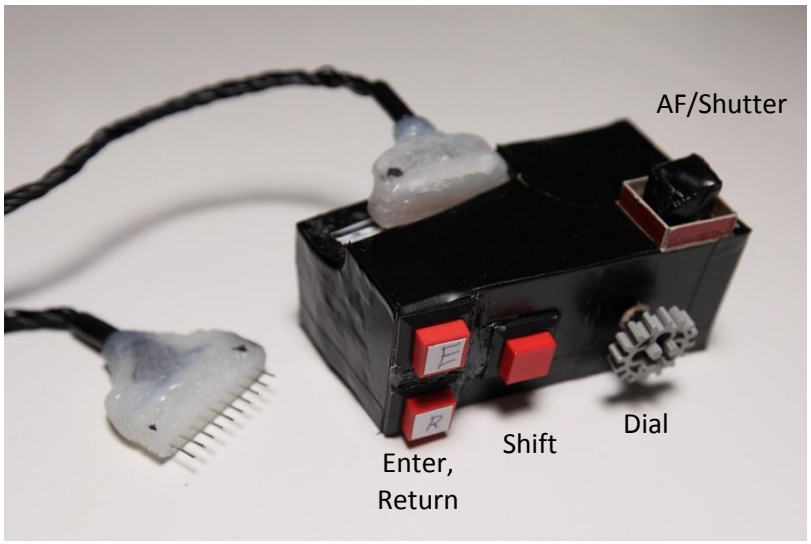

Figure 2.2: Photo of the user input device

The input device is connected to one of the expansion headers of the FPGA development board by a 9-pin parallel connector, carrying power and digital signals. All signals are connected to an open-drain type driver or equivalent on the device, to avoid voltage level problems when connected to the FPGA. The FPGA is configured to enable pull-up resistors for these pins. The schematic of the input device is shown in Figure 2.3. The three buttons is connected between signal line and ground without any transistors, while both the dial and AF/shutter button is connected by a comparator chip to output digital signals. The AF/shutter button is mechanical and custom built for the prototype, it could however be replaced by an optical, while keeping the electronics.

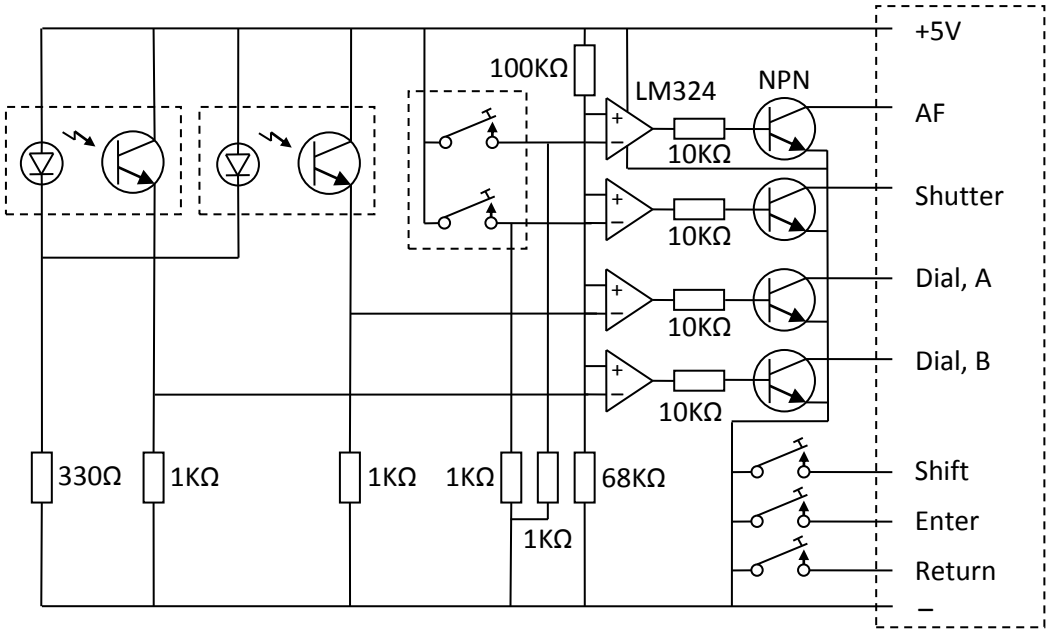

Figure 2.3: Schematic of the user input device

3. Software

This chapter will describe the software running on the main processor. The software is implemented in C as a standalone application, using standard libraries for memory management and string processing and outputting for debug purposes. The software is the limiting factor of the feature set of the Embedded Camera Remote Control, and could be extended to support additional features or a wider range of devices.

3.1. HPI Bus Master Driver

The set of functions the HPI bus master driver consists of is adjusted to the targeted USB Controller that is described in Chapter 2.2.2. The set contains simple functions to read or write one of the HPI port registers, listed in Table 2.2, such as status and DMA support registers. The functions ensure that the bus master core is ready for a new bus transfer and returns any requested word when available.

The driver does also contain functions to read or write a block of consecutive bytes using the DMA feature of the HPI bus, which is more suitable for a device driver communicating with the slave device over the HPI bus. Only even addresses are allowed as the targeted device only allows accesses aligned to its internal 16-bit word memory layout. A block of an odd number of bytes is however allowed; the driver will then perform a read-modify-write operation for the last byte in a block write operation, or ignore writing the additional byte to the local memory for a block read operation.

Control registers of the USB controller that are mapped into the memory space cannot be accessed directly using DMA. Instead, they can be accessed by BIOS commands sent using the mailbox register and arguments via DMA. The driver hides the details and provides functions with the same interface as single word read or write operations, with the exception that a control register write operation takes an extra argument to specify an ordinary write operation, a bitwise AND or bitwise OR operation.

3.2. Low-level USB Driver

The low-level USB driver handles data transfers using the USB protocol. The driver is divided in two major layers. The first layer transfers commands to the USB controller device that describes what USB request blocks it should send; this layer has to split USB communication requests into commands and data blocks that are sent and received over the HPI bus. The second layer uses the first one to setup and handle a connected USB device. It handles both the setup phase of a device and supports data transfers over USB requested by a high-level USB driver, like the MTP framework described in Chapter 3.3.

The driver support is limited to full speed capable devices and only control and bulk transfers for simplicity. Neither interrupt nor isochronous transfers are needed to communicate with the camera. Support of interrupt transfers could be added to the driver set by handling interrupt related errors and by requesting any interrupt data at the regular interval defined by the device’s endpoint descriptor. An interrupt endpoint is defined in the MTP standard as stated in Chapter 3.3, but is unused by the targeted device.

3.2.1. USB Controller Driver

The lower layer of the low-level USB driver is developed with respect to the specific USB controller used. It includes functions to reset and initiate the controller to operate as a USB host, and to power on the USB port connecting the camera. It does also contain a memory allocation function used internally, which ensures that temporary commands and data sent to and stored on the controller will not overlap, using a circular buffer strategy. The allocation function will also ensure that a data block only will be allocated to begin with an even address, to meet the requirements of the bus master of the 16-bit wide HPI bus.

Commands and USB data to send or receive are sent to the USB controller as a linked list of transfer descriptors, with a pointer to the data [9]. A TD list element contains all information the BIOS of the controller needs in order to perform a USB transaction. A transaction contains three packets in general; it begins with a token packet such as SETUP, IN or OUT, which is followed by a data packet and finally a handshake packet [10] (also simplified in [11]). The BIOS will handle handshaking and some errors, while some errors remain for the driver to handle, such as a retransmission or when an unexpected amount of bytes have been received. The driver checks the result by reading status fields of the TD elements that is updated by the BIOS after transmission.

The TD element format is given in Table 3.1; the internal data structure format is similar, but has additional pointers to main memory objects. Before sending a TD list to the controller, it is built when the upper layer driver is queuing communication requests. When the list is ready it is transferred together with related data to the USB controller, the address of the first list element is then written to an address checked by the BIOS. The BIOS will execute the operations given in the list and will finally signal completion by raising an interrupt signal. The driver will read status fields in the TD elements, transfers with retryable errors are rescheduled and any received data is finally copied from the controller to specified buffers in main memory.

Table 3.1: TD list element format

Offset	Width (bits)	Field	Description
00h	16	Base address	Pointer to data buffer
02h	16	Port/Length	USB port number and data length
04h	8	PID/EP	PID of initial packet and target endpoint
05h	8	Device address	Target device address
06h	8	Control	Control bitmap
07h	8	Status	Transaction status, set by BIOS
08h	8	Retry cnt, etc.	Retry counter, USB transfer type and active flag, updated by BIOS
09h	8	Residue	Residue after transfer, set by BIOS
0Ah	16	Next TD ptr.	Pointer to next TD list element

3.2.2. USB Support

The upper layer of the driver handles the setup phase of a USB device and allows transfers of blocks of arbitrary data to or from a specified endpoint. The setup phase includes functions to request the device descriptor that describes basic device properties like the device class and manufacturer, and the configuration descriptor describing available interfaces and especially their endpoints. The setup functions will store read

descriptor fields in a few linked data structures with a similar format as the descriptors; a simplified view is given in Figure 3.1. Most data substructures have a pointer to the main data structure, the device descriptor; this allows the driver to access necessary fields when a data substructure, such as an endpoint descriptor, is given as an argument to a function.

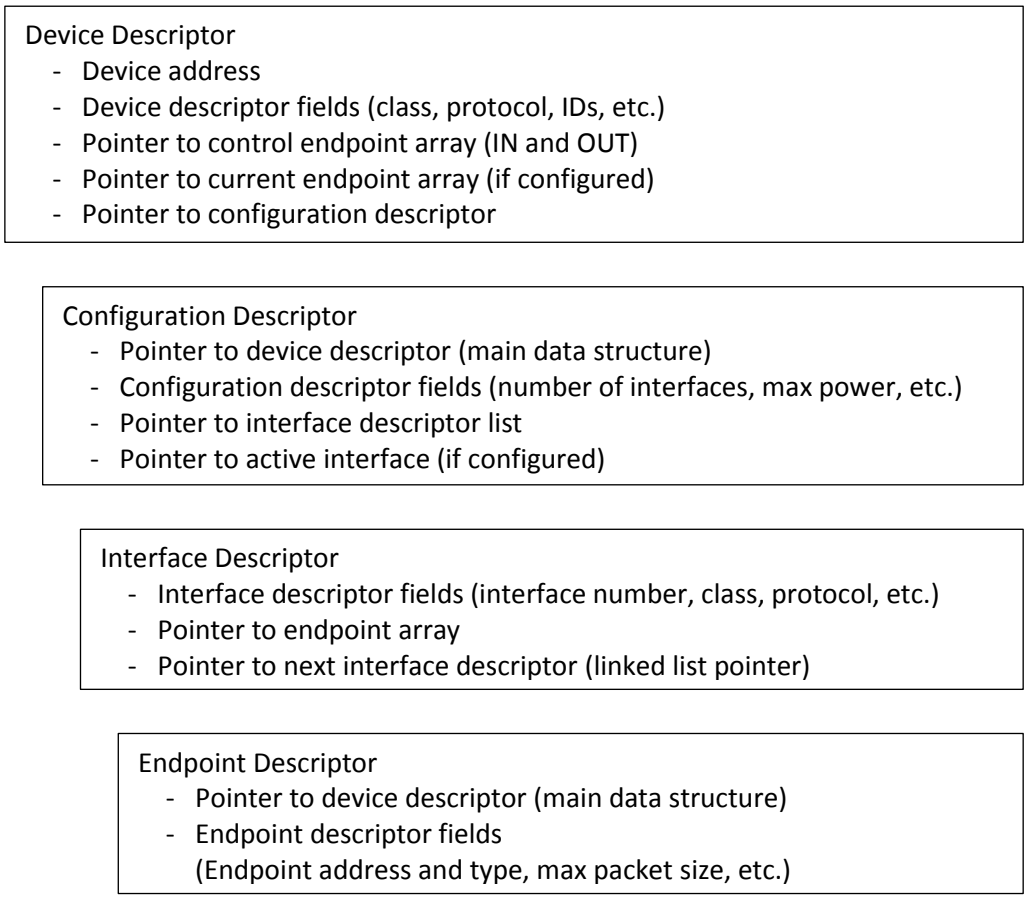

Figure 3.1: Simplified descriptor data structure format

The driver does also have a function to automatically setup a device, which will return a device descriptor data structure that can be used by a high-level communication driver. To simplify the high-level implementation, the driver provides search functions that given a data structure will return the class or protocol of the device, or an endpoint of the current interface according to specified endpoint type and data direction. The class and protocol can be used to select a proper high-level driver, while matching endpoints are used to specify the source or target in data communication controlled by the high-level driver.

3.3. MTP Framework

Media Transfer Protocol is a standard protocol for media devices; it is based on and extends the Picture Transfer Protocol, PTP, designed for communication with imaging devices such as cameras. MTP extends the defined set of commands, events and properties to allow access to features of modern media devices.

Communication using MTP is based on transactions that are initiated by the host. Each transaction consists of a request container sent by the host, an optional data container

sent or received by the host depending on the request, and a response container received from the device. Both the request and response containers may contain up to five 32-bit arguments, while the length of the data container only is limited by the current request and its data format. The containers share a common structure that is shown in Table 3.2; fields are transferred least significant byte first. [12]

Table 3.2: MTP Container format

Offset	Width (bits)	Field	Description
00h	32	Container size	Total size of container including the head
04h	16	Container type	Type of container (request, data or response)
06h	16	Code	Operation or response code
08h	32	Transaction ID	ID starting from zero, incremented each transaction
0Ch	N/A	Payload	Data payload or up to five 32-bit parameters

The device can send events using a mandatory USB interrupt endpoint defined in the standard. However, analysis of the DSLR communication has indicated that the camera never signals any event using the USB interrupt endpoint; instead a special request command is used to query the camera for events and corresponding data. Thus, support for checking the interrupt endpoint for data is not required for the targeted camera and has not been implemented.

The Media Transfer Protocol framework implemented consists of an initiation function that sets up internal variables and prepares a MTP session, based on a USB device description structure provided by the low-level USB driver. It does also contain functions to format and transfer a transaction by its containers; functions are available to send a request container, send or receive a data container and receive a response container. It is assumed that the function are called in a valid order and the caller is responsible of the informative payload data, while the framework will add and handle required MTP fields such as length and transaction number. The operation code given in the call to the request function will be reused during that transaction.

3.4. High-level Protocol

The high-level protocol used by recent Canon DSLRs is MTP, which is described in Chapter 3.3. Most necessary commands and properties are defined in the standard; however, the camera is mainly using vendor specific commands and properties. Also, events are received using an active polling command that will return a list of any changed properties or other events.

The vendor specific commands and properties are partly documented by third party sources. The documentation, together with further analysis of recorded USB communication between the camera and a PC, have yielded enough information to control most of the basic features of the camera; enough for a prototype implementation. The analysis has mainly been performed prior to the project, and a C# program has been developed to convert recorded log-files into a text file of MTP transactions with parsed commands, properties and values. Some further analysis has been performed with the help of the developed prototype platform.

The high-level protocol is centered on properties and uses a few commands to get updated properties, set a property and control AF and shutter. A main focus of the driver is to parse and store current and available property settings. Properties do generally

belong to a type that defines the data format. The most common type is the one for value properties, which are used for properties such as aperture value, shutter speed and white balance. A value is 32-bit wide, holds a signed, unsigned or enumeration value. A value property does also contain a list of currently available values, such as aperture values that depend on the lens attached, mode dial setting and camera configuration. An empty list signals that no setting can be set, which could apply when, for an example, the aperture value is set automatically by the camera when in shutter priority mode.

Other property types are string data that could hold user and lens name, and an AF information dataset describing where in the frame AF points are located and which ones are enabled or have achieved focus. There is also a type for image quality, which like the value type includes a list of available settings. The current camera date and time is given as a value type property, in seconds since a common offset date. When the time property is received it needs to be handled specially, since the local time value at reception also needs to be stored to be able to compute the current time later.

The camera sends the current value and a list of available properties separately using two different preceding event codes; they may be included in the same list of events. The split and use of different codes implies that all properties could have a list of available settings; however it is not applicable for some properties such as a name. There are also other event codes, some signaling that the camera will enter sleep mode in a specified amount of seconds, or has cancelled it, when the host for example sets a property. Currently, the remote control will abort the sleep mode by sending a property updating command.

3.5. Text-based Graphics Driver

The text-only VGA IP core has a text buffer that stores the current screen and also offers hardware assisted scrolling. The implemented driver does not explicitly support scrolling and assumes that all characters are written within the visible area. The hardware supported scrolling feature can however be used and be explicitly controlled by the application.

The driver has a limited set of features. The main features are to set a write position and to write a string of characters, optionally terminated by an added line break. The driver will place text that is split over several lines, so it starts from the column specified by the last write position setting. The driver does also support clearing of specified lines, or a specified rectangular area. It is also possible to draw a rectangle border.

The driver will immediately write characters to the hardware core without any double buffering, which could result in some flickering if an area is repeatedly cleared and written to. To avoid flicker, the text can be overwritten without clearing in between.

3.6. Application

The main application consists of two major parts, the user interface and the control logic behind it. The graphical user interface is text-based and displays common settings in a main area, while a settings area below shows the current mode of user interactivity. The user interface is described in Chapter 3.6.1. The application has a separate C-module that handles text formatting related to the high-level protocol, which is a key component when outputting text to the GUI.

The control logic relates to the user interface since a major task is to support the current settings mode and handle operations and changes requested by the user. The control logic relies on the drivers developed, especially the high-level protocol that handles all communication with the camera.

The application is implemented without any interrupt service routine; instead, an active polling approach has been chosen. Tasks that need to be performed regularly include USB communication that mainly involves querying the camera about any recent events, checking for and handle user input, handle a started timer operation and finally update the graphical user interface when required. These tasks do not have any critical time limit for response times and does generally not take very long to complete. The sequential approach will also ensure that a high-level communication operation is completed before any further operation is started.

The USB communication will typically be the limiting factor in terms of response time, since it is implemented as a blocking I/O operation in the drivers. A high-level MTP transaction contains up to three USB I/O operations, which each takes time to generate and send low-level data to the USB controller, time for the USB transfer, and time to check status including possible retransmissions. The update rate is still high enough to not be noticeable in the general case; the event querying rate may even be reduced in order to lower the idle load at the camera to possibly extend battery life. One USB operation that may cause a short lockup is the AF activation command, which the camera will not respond to until focus is achieved or has failed. However, no communication can be performed meanwhile, leaving few possibilities independent of implementation.

The application contains some code to support debugging, testing and analysis of the high-level protocol. It includes a possibility to enable printing of triggered events received from the camera and printing of their data if wanted. All debug support need a debug connection to receive character coded commands to enable or disable features and to print the debug output. By default most features are disabled and won't be noticed without a PC connection.

The application is stored in an onboard flash memory. The application is loaded and decompressed using a boot loader program. The boot loader has been generated by Aeroflex Gaisler's MKPROM2 boot PROM builder, which also can compress the application executive.

3.6.1. User Interface

The graphical user interface consists of two main areas; one area displays current values of common properties, with the most common at its top line. The layout is unchanged independent of the current mode or camera settings, but some fields may not apply and are then blanked out. The second area is a settings area displaying what settings currently can be set. The settings area reflects the current mode of operation. There is also a navigation legend located right of the main areas, displaying current functions of the Enter and Return keys, as well as the shutter button in timer mode. The GUI in default operation mode is shown in Figure 3.2, with each field and area named.

Due to limitations in the character set available, some of the camera's symbols might not be as comprehensive as desired; after some time getting used to the character combinations used, most is expected to be understandable, especially when considering

the limited number of symbols in total. For a description of the symbols, the camera's user manual should be referred to.

In the default mode, which is entered when a supported device is connected, up to two properties can be adjusted. The input dial is rotated to increase or decrease the primary value or the secondary value while the shift key is pressed. The properties available depend on the mode dial setting on the camera, and correspond to the camera's main and secondary dials. The current, previous and next values are shown, the step size is the one configured on camera, typically in increments of one third or one half of a stop. The settings area in default mode is shown in the bottom half of Figure 3.2.

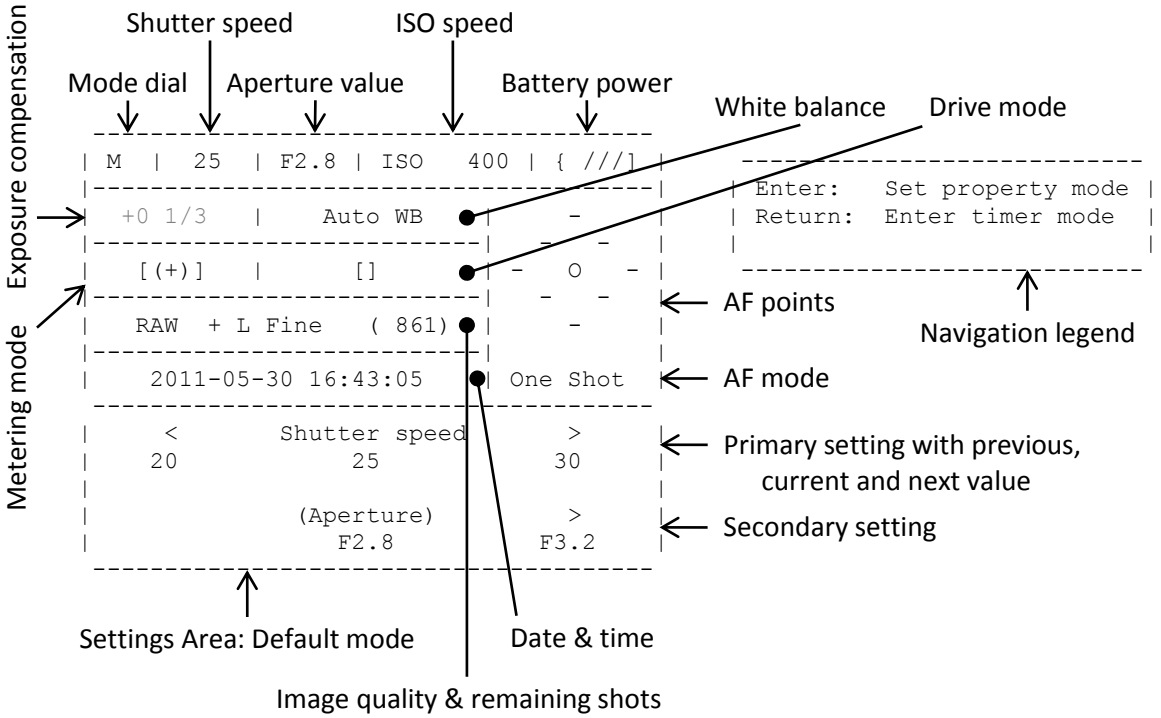

Figure 3.2: User interface in default mode

From the default mode, both timer mode and property setting mode can be selected using the Return and Enter navigation keys, correspondingly. From either of the two modes it is only possible to return to the default mode. The property setting mode is similar to the default mode; a property can be changed as the primary setting in the default mode, while the secondary property is replaced by selection of a property. An example of the settings area is shown in Figure 3.3. The set of properties available is configurable in the application by adding their codes to a list. The properties available for adjustment are ISO speed, white balance with an additional color temperature property available just when manual color temperature has been selected, drive mode and metering mode. AF mode is also available, but the camera will generally not accept its property change command; changing is still possible on camera. Some properties and also the property list is circular, while a property like ISO speed has a minimum and maximum value, as illustrated in the bottom line of Figure 3.2 for the aperture value.

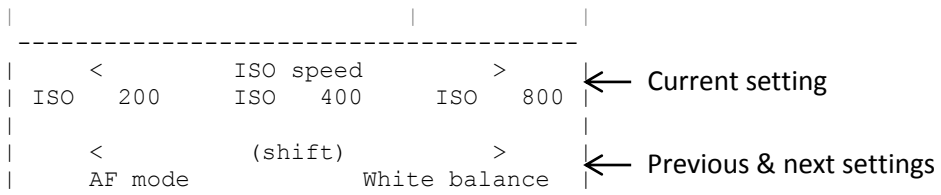

Figure 3.3: Settings area in set property mode

The timer mode has up to five settings depending on the camera settings. All possible settings are shown in Figure 3.4. A cursor that selects a value is controlled by the dial; a selected value can also be incremented or decremented by turning the dial while pressing the shift key. To change a value one can also press the Enter key; the selected value will then be changeable in three parts, typically in hours, minutes and seconds. This is shown in Figure 3.5. The selected part is underlined and the value changed by rotating the dial, while the selected part is selected while the shift key is pressed. To return to the selection of a value, press Return.

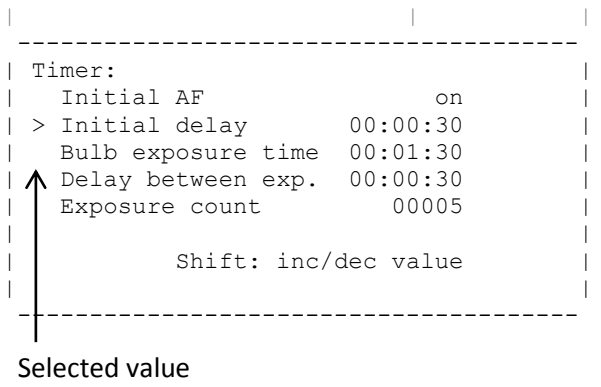

Figure 3.4: Settings area in timer mode (value selection)

Initial AF can be enabled or disabled, if AF is possible with camera and lens configuration. If disabled, no focus movement will be performed, allowing a preset focus which also will avoid any additional delay due to a long focusing time. The initial delay is counted down before the first exposure is started; if initial AF is enabled it will be started a few seconds before the first exposure. The bulb exposure time controls the exposure time in seconds when the camera is set to Bulb mode. The delay between exposures has to be long enough so the camera is ready for a new exposure. The value is especially important when the camera is set to automatically take a second dark exposure, with the shutter closed, to reduce sensor noise in the main exposure. The exposure count can be set to a huge value; however, the user should ensure that battery life and free memory card space are enough for the exposure session to complete.

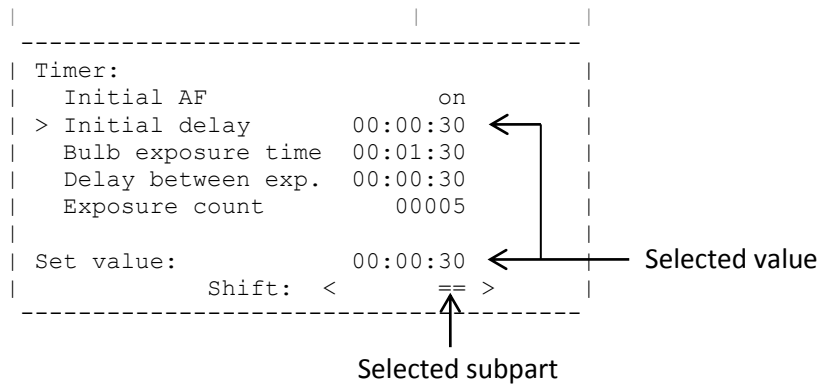

Figure 3.5: Settings area in timer mode (set value)

To start a timer operation the shutter button should be pressed completely. The values will begin to count down one at a time during the timed shooting, until all exposures are completed. It is possible to break a timer operation by pressing the Return key. The navigation legend is updated with relevant information at all times.

3.6.2. Usage Examples

A time-lapse exposure example is shown in Figure 3.6; a frame is taken every ten seconds, but only one in fifty is included in the figure. Manual exposure has been set and all frames have been exposed the same, leading to a time-lapse that fades to black over time. To capture a similar time-lapse exposure, the aperture value, exposure time and ISO speed together with any other settings should be set using the remote control in default and property setting mode, or be set on the camera, which should be in Manual or Bulb mode. The remote control should then be set to timer mode, and especially the delay between exposures and the exposure count need be configured; a delay of ten seconds have been used in the example, which consist of approximately 350 frames. The exposure count can be set with a margin since it is possible to abort a timer operation.

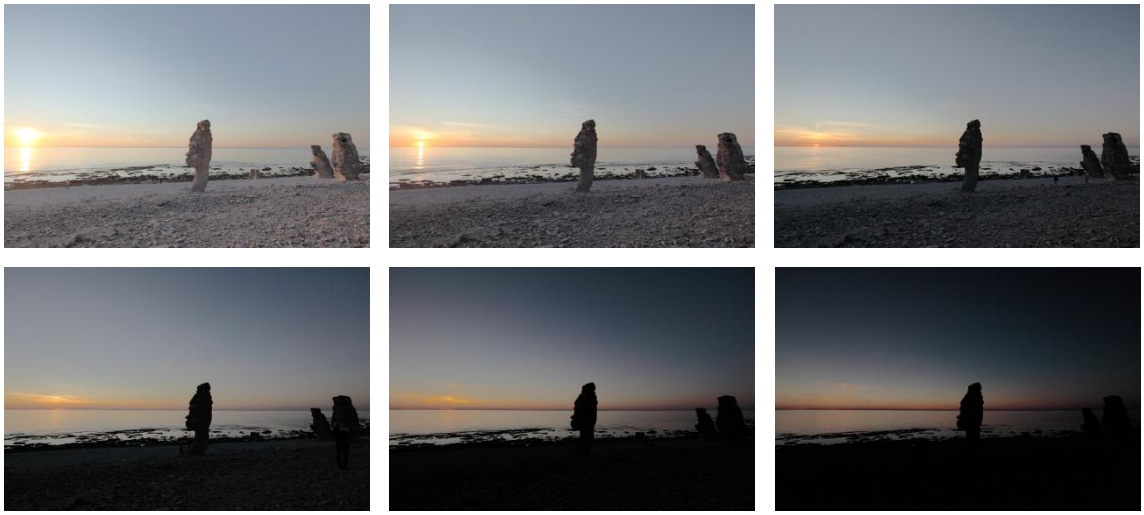

Figure 3.6: Time-lapse exposure of sunset

To take a preconfigured long exposure, the camera needs to be set to Bulb mode. The exposure level is set manually by setting the aperture value and ISO speed, and by setting the exposure time in the timer mode of the remote control. If only one exposure should be taken, the exposure count should be set to one, while the time between exposures can be ignored. An example photo is shown in Figure 3.7; an exposure time of four minutes has been chosen for some motion blur of the clouds and the sea. A smaller aperture value has been chosen to increase the depth of field, and the base ISO speed of 100 has been selected for minimum noise; both these settings require a long exposure time for correct exposure in a low light condition.

No guidance about correct exposure level is available in Bulb mode; the user should rely on light metering available in other camera modes or use an external light meter. Experimenting by taking test shots is another possibility, but will introduce some setup time to find a good exposure value.

Figure 3.7: Long exposure of a late evening sky

4. Conclusion

An embedded camera remote control prototype has been developed and is described in this report. A template hardware platform has been customized during the project and developed communication cores have been added to the design. To more resemble a remote control, and to give a better view of how a remote control product might be controlled, a custom input device has been built.

The hardware platform provides the features and performance needed for the remote control. The platform will support additional features as long as the computational and USB bandwidth demands are limited, such as for features related to properties. The prototype platform is however not suitable for field usage since it is relatively large and sensitive, and especially since an external monitor is required.

In order to make a user friendly product, the development platform needs to be replaced by a much smaller custom one. Also, the external monitor has to be replaced by a display integrated into the remote control. The text-based GUI would be necessary to replace with a pixel-based one, which could display camera symbols correctly.

The software includes both drivers customized to the hardware used, and more general software, such as the main application, and some high-level drivers. The feature set of the remote control is limited by the software, but includes most settings that can be expected to be practical to access remotely. The timer module extends the in-camera features, enabling preconfigured long exposure times and time-lapse exposures, for example.

Depending on the user, the available timer settings may not be enough; no settings can be changed during a started timer operation. Settings that would be good to be configurable to automatically change during a timer operation are the exposure settings: aperture value, shutter speed and ISO speed. If the camera is set to a mode in which it automatically will set the exposure level, it will adjust to changing light conditions; but it will result in some noticeable flicker in a final time-lapse exposure during playback, due to some noise in the light and light metering, unless compensated for in post-processing.

A more advanced feature that would be practical during long time-lapse exposures would be to low-pass filter the exposure value to support varying light conditions, like a sunset. However, the camera limits this feature since it does not send any exposure level value in manual mode, but displays it on the camera. The feature could possibly be implemented if camera support would be added in a future firmware update.

Another solution could be to configure both initial and final exposure values, which can be useful when the light changes evenly over time. This solution could also be used when taking high dynamic range, HDR, photos, where several differently exposed shots are merged together to allow high contrast scenes to be captured without any under- or overexposed areas in the final image.

The prototype does not support more advanced settings, like the camera's custom functions that for example can enable or disable mirror lockup, which reduces camera shaking when mounted on a tripod, by releasing the mirror a few seconds before the exposure is started. This implies that the remote control prototype does not handle the case when mirror lockup is enabled in any special way, during a timer operation; the combination has not been tested. The manufacturer's PC software does not allow timer

configuration when mirror lockup is enabled, in comparison. The software could be updated with an option for mirror lockup, possibly with an option for the time before the exposure, similar to the initial AF option.

The prototype remote control does only support shooting when the image could be seen through the viewfinder. The camera can also be controlled by a PC in live view mode, where the image is forwarded from the sensor to the PC. The high-level protocol related to live view mode has not been analyzed; thus, the video stream compression algorithm is unknown. Handling live view mode has been outside the scope of this project; still, common settings can be accessed and exposures be taken when the camera is manually set to live view mode. The live view image is then displayed on the camera monitor, or on an externally connected monitor.

If more advanced features should be implementable in software, like image and video stream displaying, the hardware design will have to be extended and changed. More computational power could be provided by, for example, a video decompression hardware core to accelerate the software implementation and to balance the power consumption, by reducing requirements of the main processor. A Hi-Speed USB connection would likely be required to provide enough bandwidth to transfer the live view video stream. Also, color graphics with a reasonable high resolution and quality would be required.

An advanced camera remote control with timer support in a portable format is currently missing on the market, while products making use of the USB connectivity begin to enter. Similar products will probably be available within the next few years. The portion of photographers that will benefit from an advanced remote control is probably dependent on the size and feature set, which may contradict each other in some circumstances. A classical remote shutter control will probably still be handy in many cases, while a wider feature set is better in other.

For some potential users, such as hobby photographers, the price may be an important factor. The end-user cost could be reduced by using existing hardware, such as portable devices with USB host support, on which an application could enable most features. However, a custom product may still be preferable when an uninterrupted shooting session is desired or when in worse weather conditions.

References

[1] Xilinx, Inc., "ML505/ML506/ML507 Evaluation Platform: User Guide, ver. 3.1.2," 16 June 2011. [Online]. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf. [Accessed July 2011].

[2] Xilinx, Inc., "Virtex-5 Family Overview, ver. 5.0," 6 February 2009. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf. [Accessed July 2011].

[3] Cypress Semiconductor Corporation, "CY7C67300 EZ-Host™ Programmable Embedded USB Host and Peripheral Controller with Automotive AEC Grade Support, rev. K," 5 July 2011. [Online]. Available: http://www.cypress.com/?docID=30079. [Accessed July 2011].

[4] J. Hyde, "USB Multi-Role Device Design By Example," 2003. [Online]. Available: http://www.usb-by-example.com/Multi-Role.pdf. [Accessed July 2011].

[5] Chrontel, Inc., "CH7301C DVI Transmitter Device, rev. 1.5," 17 March 2010. [Online]. Available: http://www.chrontel.com/pdf/7301ds.pdf. [Accessed July 2011].

[6] Aeroflex Gaisler, "GRLIB IP Core User's Manual, ver. 1.1.0 B4108," 28 June 2011. [Online]. Available: http://www.gaisler.com/products/grlib/grip.pdf. [Accessed July 2011].

[7] Aeroflex Gaisler, "GRLIB IP Library User's Manual, ver. 1.1.0 B4108," 28 June 2011. [Online]. Available: http://www.gaisler.com/products/grlib/grlib.pdf. [Accessed July 2011].

[8] Aeroflex Gaisler, "GRLIB/LEON3 Readme for Xilinx ML509, ver. 1.1.0 B4108," 28 June 2011. [Online]. Available: http://www.gaisler.com/products/grlib/grlib-gpl-1.1.0-b4108.zip (grlib-gpl-1.1.0-b4108\designs\leon3-xilinx-ml509\README.txt). [Accessed July 2011].

[9] Cypress, Inc., "BIOS User's Manual, ver. 1.1," 2003. [Online]. Available: http://www.cypress.com/?docID=14346. [Accessed July 2011].

[10] Compaq Computer Corporation, et al., "Universal Serial Bus Specification, rev. 2.0," 27 April 2000. [Online]. Available: http://www.usb.org/developers/docs/usb_20_071411.zip (usb_20_071411\usb_20.pdf). [Accessed July 2011].

[11] MQP Electronics Ltd., "USB Made Simple," [Online]. Available: http://www.usbmadesimple.co.uk/index.html. [Accessed July 2011].

[12] Microsoft Corporation, "Media Transfer Protocol Enhanced, rev. 0.96," 31 August 2006. [Online]. Available: http://www.microsoft.com/download/en/details.aspx?id=19678. [Accessed July 2011].