Purpose: This document describes the control and function of the Flasher Board. The firmware registers are defined, and Forth code is presented as an example for further software development. The appendix contains the full text of a Forth application which has proven to exercise the Flasher interface, with the possible exception of reading the LED current waveforms via the ATWD.

Acknowledgement: A sincere thank you to Nobuyoshi Kitamura, Xu Zhai, Mark Krasberg and Chris Wendt for the extensive work they provided in developing the design concepts, writing CPLD Firmware and testing many aspects of the Flasher board function. Each of these individuals provided exceptional support during the project development.

Basics: Communication with the Flasher is via an eight bit parallel interface. The interface is straightforward and recognizable. There are eight data bits, a read signal, write signal, and a clock. The interface is bi-directional capable, however not all registers are read as well as write ports. The logic functions are provided by a Xilinx CPLD, (XCR3256XL-10TQ144I). The JTAG pins required to program the CPLD are connected to the Flasher interface connector to allow firmware changes to occur after deployment if such a need should arise.

Power: The Flasher board is normally in a power down state when not in use. A single bit control connects or disconnects the Flasher to the Main Board 3.3volt & +/-5 volt power sources by activating solid state relays on the Flasher card itself.

- Turn on the Flasher: \texttt{Fon 2 \$500000F9 c! ;}
- Turn off the Flasher: \texttt{Foff 0 \$500000F9 c! ;}

AuxReset: Pin 35 of the Flasher interface connector is labeled AUX_RESET. Writing an address on the Main Board activates this pin. At the moment no function is assigned to this bit, however a connection exists from the AUX_RESET pin to the CPLD pin 79. A past iteration of the Flasher prototype had this bit assigned. It may be considered as a spare input to the CPLD under Main Board control.

- Turn on the AUX_RESET Bit: \texttt{pt1 \$4000000 \$90081018 !;}
- Turn off the AUX_RESET Bit: \texttt{pt0 \$000000 \$90081018 !;}

Trigger: The Flash command is executed by issuing “trigger”. Pin 30 of the Flasher interface connector is labeled TRIGP. This pin is activated by writing an address on the Main Board or through a Main Board state machine. Interactive control may be issued with the following method:

- \texttt{: t0 \$000000 \$90081018 !;}
- \texttt{: t1 \$100000 \$90081018 !;}

• `trig t0 t1 t0` \ one pulse

The Main Board FPGA has been programmed to provide a 600hz auto trigger:

• `: flashc t0` ;
• `: done t1` ;

**Reset:** Shortly after turning on the Flasher power a “reset” command should be issued. In the example Forth code a 5000us delay is placed between the Power on command and the Reset. That is much longer than required. The reset command insures that all the registers internal to the CPLD are in a known state. The CPLD state machine should be in “Normal Mode” after a reset. (Modes will be defined shortly.)

• Reset the Flasher CPLD: `: Freset 1 fl_base addr_reset + c!` ;

**CPLD ID:** An 8 bit read only register reports the CPLD firmware revision.

• Print CPLD Version: `: Fver fl_base addr_cpld_id + c@ . drop` ;

**PCB ID:** An 8 bit read only register reports the PCB revision:

<table>
<thead>
<tr>
<th>PCB Rev</th>
<th>Hex Value</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0x07</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0x00</td>
<td>10</td>
</tr>
<tr>
<td>E</td>
<td>0x01</td>
<td>60</td>
</tr>
<tr>
<td>F</td>
<td>0x02</td>
<td>400?</td>
</tr>
<tr>
<td>G</td>
<td>0x03</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>0x04</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>0x05</td>
<td></td>
</tr>
</tbody>
</table>

• Print PCB Version: `: PCBver fl_base c@ . drop` ;

**Flasher Communication Modes:** There are three types of register communications implemented in the CPLD. Each of the three modes service specific devices. The “Normal” mode, which is the reset default, allows communication with the CPLD internal registers. Normal refers to the standard 8bit parallel register configuration. The SPI Mode is used to communicate with the Maxim Digital POT that controls the pulse amplitude at the LEDs when they are requested to flash. The “One-Wire” protocol is assigned to communicate with the Dallas Serial number IC that uniquely identifies each Flasher Board.

• Normal2SPI: `: SpiCom 1 fl_base addr_mode_select + c!` ;
• Normal2OneWire: `: OneWireCom 2 fl_base addr_mode_select + c!` ;
• SPI/OneWire2Normal: `: NormCom 0 fl_base addr_mode_select + c!` ;
Serial Number: A 64 bit read only register reports the Flasher Serial Number. A state machine implemented in the CPLD provides low level timing required interacting with the OneWire interface. When properly managed with a software driver the state machine extracts 64 bits one at a time from the Serial Number circuit.

- The OneWire data and status information is read at the Flasher Base address when the Mode is “OneWire”. Bit6 is the OneWire data and Bit5 is the OneWire operation status. When Bit5 is low the data Bit6 is a valid 0 or 1, if Bit5 is high the data Bit6 is not valid. A list of commands that may be sent to the OneWire device at the Flasher Base address follows:

<table>
<thead>
<tr>
<th>Command</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset OneWire</td>
<td>0x0F</td>
</tr>
<tr>
<td>Send OneWire “1”</td>
<td>0x09</td>
</tr>
<tr>
<td>Send OneWire “0”</td>
<td>0x0A</td>
</tr>
<tr>
<td>Set OneWire state to Read Mode</td>
<td>0x0B</td>
</tr>
</tbody>
</table>

- Steps to read the 64 bits
  - Setup OneWire Mode
  - Send OneWire reset by writing 0x0F to Flasher Base Address.
  - Read OneWire status for presence at Flasher Base Address.
  - Send “readrom = 0x33” serially one bit at a time, (with 0x09 & 0x0A).
  - Read back one bit per read, 64 times.
  - Setup Normal Mode

Digital Pot LED Pulse Amplitude: The digital pot provides 128 settings, which alter the voltage applied to the LED, buffer drivers. The digital pot is a SPI device, which uses three control wires to communicate with controlling circuits. Bits 0,1 and 2 refer to the SPI devices signals Sclk, MOSI and nCS respectively. Examination of the following code will illustrate how data is sent to the Digital Pot. The details of clock data and chip select may be found in the SPI specifications for the Digital Pot.

- Write to the SPI address from stack top: `: spi fl_base addr_spi + c! ;`
- Set data low, clock low, CS on: `: ss 0 spi ;`
- Set data low, clock low, CS off: `: sd 4 spi ;`
- Send a High bit: `: hidata 2 spi 3 spi 2 spi ;`
- Send a Lo bit: `: lodata 0 spi 1 spi 0 spi ;`
- Send bit7: `: s7 dup 128 and if hidata else lodata endif ;`
- Send bit6: `: s6 dup 64 and if hidata else lodata endif ;`
- Send bit5: `: s5 dup 32 and if hidata else lodata endif ;`
- Send bit4: `: s4 dup 16 and if hidata else lodata endif ;`
- Send bit3: `: s3 dup 8 and if hidata else lodata endif ;`
- Send bit2: `: s2 dup 4 and if hidata else lodata endif ;`
- Send bit1: `: s1 dup 2 and if hidata else lodata endif ;`
• Send bit0:  \texttt{s0 dup 1 and if hidata else lodata endif} ;
• Send bits:  \texttt{sss s7 ss s6 ss s5 ss s4 ss s3 ss s2 ss s1 ss s0 ss sd} ;
• Set Pulse Voltage from integer on stack, 0 \rightarrow 127.
  \texttt{: setV SpiCom sd sss drop NormCom sd} ;

Digital Delay, LED Pulse Width: The Maxim Delay generator has been set up as a parallel device which only requires that data be presented and a latch pin signaled to strobe the data into the device. In practice the latching signal may be asserted and allowed to remain asserted since the data source is latched internally to the CPLD.

  • Assert the latch enable:  \texttt{: DelayOpen 1 fl_base addr_le_dp + c!} ;
  • De-assert:  \texttt{: DelayClose 0 fl_base addr_le_dp + c!} ;
  • Load data from stack:  \texttt{: setDelay fl_base addr_delay_adjust + c!} ;
  \begin{itemize}
    \item DelayOpen
    \item \texttt{<0:127> setDelay}
    \item DelayClose
  \end{itemize}

DC to DC Control: The LEDs are driven by up to 14.0volts. In order to provide this potential a DC to DC converter is included in the Flasher design. A control bit is provided to turn on or off the converter itself. The strategy here was to provide a lower total current demand at the time the Flasher is turned on. The control bit is logic zero at the time the Flasher is turned on, therefore the inrush current demand is reduced. Some interval of time after the Flasher is turned on the DCDC control bit needs to be raised. Doing so will activate the converter and provide a 15volt source to the Flasher circuits which then source up to 14 volts to the LED driver.

  • Start up the converter:  \texttt{: onDCDC 1 fl_base addr_dcdc_ctrl + c!} ;
  • Kill the converter:  \texttt{: offDCDC 0 fl_base addr_dcdc_ctrl + c!} ;

Selecting the LEDs: The individual LEDs may be enabled to flash or disabled by setting or resetting a single bit corresponding to each LED. Three addresses are used for this purpose each address providing access to four of the twelve LEDs.

  • D4, D3, D2, D1 each correspond to the low four bits b3, b2, b1, b0 of an integer placed on the Forth stack. (\texttt{<15:0> SelectLEDs4_1})
    \texttt{: SelectLEDs4\_1 fl_base addr_LEDEn4\_1 + c!} ;
  • D8, D7, D6, D5 each correspond to the low four bits b3, b2, b1, b0 of an integer placed on the Forth stack. (\texttt{<15:0> SelectLEDs8_5})
    \texttt{: SelectLEDs8\_5 fl_base addr_LEDEn8\_5 + c!} ;
  • D12, D11, D10, D9 each correspond to the low four bits b3, b2, b1, b0 of an integer placed on the Forth stack. (\texttt{<15:0> SelectLEDs12_9})
    \texttt{: SelectLEDs12\_9 fl_base addr_LEDEn12\_9 + c!} ;
Monitoring the LED current waveform: The Flasher Board includes circuitry intended to allow monitoring the current waveform of each LED one at a time. The monitoring is accomplished by using a set of video multiplexes to select a voltage waveform that is created by the LED current passing through a sense resister. Four MUX integrated circuits in a cascade structure provide the required switching topology. Three ICs in the first layer each with four inputs, connect to the waveform source resistors. The fourth IC collects the three output ports of the first layer and also provides one free input that is used to examine a TTL pulse representative of LED driving pulse. Three types of control bits manipulate the function of the MUX circuits. These control bits are packed into two registers. Note: Both registers must be managed correctly!

- The “sel_mux” register is used to select one of the thirteen signal sources. The lower four bits written to this register correspond to:

 MUX3, MUX2, MUX1, MUX0

<table>
<thead>
<tr>
<th>MUX3</th>
<th>MUX2</th>
<th>MUX1</th>
<th>MUX0</th>
<th>Signal Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LED D1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LED D2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>LED D7</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>LED D8</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>LED D3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>LED D4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>LED D9</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>LED D10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>LED D5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>LED D6</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>LED D11</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>LED D12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>TTL Pulse</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>TTL Pulse</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>*</td>
<td>*</td>
<td>TTL Pulse</td>
</tr>
</tbody>
</table>

- The “sel_muxen” register is used to control shut down and selection of the individual MUX integrated circuits themselves. Shutdown control is included in the design because the power dissipation of the video mux circuits is significant. Proper use of the MUX “SD” control and the MUX nEN control will save power in the operation of the Flasher. There are four MUX nEN controls, one for each of the four MUX circuits. MUX nEN selects or deselects a particular IC. Note that two MUX nEN signals must be true for any particular setting of the “sel_mux” register, and the MUX “SD” must also be true when any of the signals require examination. If the LED current waveform are not being examined the MUX “SD” signal should be turned off. The top six bits of the “sel_muxen” register are assigned as follows from the msb to the lsb: MUX“SD”, FINAL_ENn, don’t_care, CMUX_ENn, BMUX_ENn, AMUX_ENn, don’t_care, don’t_care.
The following Forth code illustrates an implementation of primitives suitable for the Video Mux selection.

```forth
: setflmux  fl_base addr_mux + c! ;
: setflmuxen fl_base addr_mux_en + c! ;

: muxoff $FC setflmuxen ;
\( n - ) 0,1,2, or 3 == mux input to select.
: muxaon $18 setflmuxen $0 + setflmux ;
: muxbon $14 setflmuxen $4 + setflmux ;
: muxcon $0C setflmuxen $8 + setflmux ;
: ttl $1C setflmuxen $C setflmux ;

: V1 $0 muxaon ;
: V2 $1 muxaon ;
: V3 $0 muxbon ;
: V4 $1 muxbon ;
: V5 $0 muxcon ;
: V6 $1 muxcon ;
: V7 $2 muxaon ;
: V8 $3 muxaon ;
: V9 $2 muxbon ;
: V10 $3 muxbon ;
: V11 $2 muxcon ;
: V12 $3 muxcon ;
```
Appendix A

Flasher Example Forth Code:

```forth
$60000000 constant fl_base
&0 constant addr_status
&1 constant addr_reset
&2 constant addr_cpld_id
&3 constant addr_mode_select
&4 constant addr_spi
&5 constant addr_dcdc_ctrl
&6 constant addr_six
&7 constant addr_seven
&8 constant addr_delay_adjust
&9 constant addr_ledenLo
&10 constant addr_ledenMid
&11 constant addr_ledenHi
&12 constant addr_mux
&13 constant addr_mux_en
&14 constant addr_le_dp
&15 constant addr_fifteen

\ Turn on the Flasher Board
: Fon 2 $500000f9 c! ;

\ Turn off the Flasher Board
: Foff 0 $500000f9 c! ; : foff disableFL ;

\ Buzz Adr Lines
: aBuzzw0 64 0 ?DO 0 fl_base i + c! LOOP ;
: aBuzzw1 64 0 ?DO $FF fl_base i + c! LOOP ;
: aBuzzr 64 0 ?DO fl_base i + c@ drop LOOP ;
: aTst 0 ?DO aBuzzw0 aBuzzw1 aBuzzr LOOP ;

\ Reset the Flasher Board
: Freset 1 fl_base addr_reset + c! ;

\ Read base address
: base. fl_base c@ . drop ;

\ Read the Flasher Board CPLD firmware version number
: Fver fl_base addr_cpld_id + c@ . drop ;

\ Turn on DCDC
: onDCDC 1 fl_base addr_dcdc_ctrl + c! ;

\ Turn off DCDC
: offDCDC 0 fl_base addr_dcdc_ctrl + c! ;

\ Write a byte to the programmable delay line
\ value setDelay
: DelayOn 1 fl_base addr_le_dp + c! ;
: setDelay fl_base addr_delay_adjust + c! ;

\ Write to L[3..0]
```
\value setConfig0
: sledenLo fl_base addr_ledenLo + c! ;
\Write to L[7..4]
\value setConfig1
: sledenMid fl_base addr_ledenMid + c! ;
\Write to L[11..8]
\value sleden12_9
: sledenHi fl_base addr_ledenHi + c! ;
\SPI Support  Bits are: CLk = 0, Data = 1, CS = 2
: spi fl_base addr_spi + c! ;
: ss 0 spi ; \selected, data low, clock low, cs on
: sd 4 spi ; \selected, data low, clock low, cs off
\SPI Mode ON ( - )
: onSPI 1 fl_base addr_mode_select + c! sd ;
: offSPI 0 fl_base addr_mode_select + c! sd ;
: hidata 2 spi 3 spi 2 spi ;
: lodata 0 spi 1 spi 0 spi ;
: s7 dup 128 and if hidata else lodata endif ;
: s6 dup 64 and if hidata else lodata endif ;
: s5 dup 32 and if hidata else lodata endif ;
: s4 dup 16 and if hidata else lodata endif ;
: s3 dup 8 and if hidata else lodata endif ;
: s2 dup 4 and if hidata else lodata endif ;
: s1 dup 2 and if hidata else lodata endif ;
: s0 dup 1 and if hidata else lodata endif ;
\setV (data -) \ 0 to 255
: sss ss s7 ss s6 ss s5 ss s4 ss s3 ss s2 ss s1 ss s0 ss sd ;
: setV onSPI sss drop offSPI ;
\Trigger stuff ( - )
: t0 $0000000 $90081018 ! ;
: t1 $1000000 $90081018 ! ;
: trig t0 t1 t0 ;
: pt0 $0000000 $90081018 ! ;
: pt1 $4000000 $90081018 ! ;
: aux_reset pt0 pt1 pt0 ;
\new FPGA implement 600Hz flash
: flashc $1000000 $90081018 ! ;
: done $0000000 $90081018 ! ;
\Write to the mux enable byte
\value setflmux
: setflmux fl_base addr_mux + c! ;
: setflmuxen fl_base addr_mux_en + c! ;
: muxoff $FC setflmuxen ;
\(n - ) 0,1,2, or 3 == mux input to select.
: muxaon $18 setflmuxen $0 + setflmux ;
: muxbon $14 setflmuxen $4 + setflmux ;
muxcon $0C setflmuxen $8 + setflmux ;
ttl $1C setflmuxen $C setflmux ;

V1 $0 muxaon ;
V2 $1 muxaon ;
V3 $0 muxbon ;
V4 $1 muxbon ;
V5 $0 muxcon ;
V6 $1 muxcon ;
V7 $2 muxaon ;
V8 $3 muxaon ;
V9 $2 muxbon ;
V10 $3 muxbon ;
V11 $2 muxcon ;
V12 $3 muxcon ;

\ Startup Sequence Flasher Board
ledson 15 sledenLo sledenMid 15 sledenHi ;
ledsoff 0 sledenLo 0 sledenMid 0 sledenHi ;
d0 ledsoff muxoff ;
d1 ledsoff 1 sledenLo muxoff ;
d2 ledsoff 2 sledenLo muxoff ;
d3 ledsoff 4 sledenLo muxoff ;
d4 ledsoff 8 sledenLo muxoff ;
d5 ledsoff 1 sledenMid muxoff ;
d6 ledsoff 2 sledenMid muxoff ;
d7 ledsoff 4 sledenMid muxoff ;
d8 ledsoff 8 sledenMid muxoff ;
d9 ledsoff 1 sledenHi muxoff ;
d10 ledsoff 2 sledenHi muxoff ;
d11 ledsoff 4 sledenHi muxoff ;
d12 ledsoff 8 sledenHi muxoff ;

\ With video mux
D0 d0 ;
D1 d1 V1 ;
D2 d2 V2 ;
D3 d3 V3 ;
D4 d4 V4 ;
D5 d5 V5 ;
D6 d6 V6 ;
D7 d7 V7 ;
D8 d8 V8 ;
D9 d9 V9 ;
D10 d10 V10 ;
D11 d11 V11 ;
D12 d12 V12 ;
pupw 5000 usleep ;
Pup Fon pupw Freset pupw aux_reset pupw onDCDC pupw ;
startFL Pup muxoff 0 setV DelayOn ;

bright 127 setV ;
see 96 setV ;
: dim 0 setV ;
: long 127 setDelay ;
: med 64 setDelay ;
: short 32 setDelay ;

: 1ms 1000 usleep ;
: 1sec 1000 0 ?DO 1ms LOOP ;

: vramp dim 1sec see 1sec bright 1sec ;
: nramp 0 ?DO vramp LOOP ;

\ trig once wait 1ms
: flash trig 1000 usleep ;
\( n -) flash n times at 1ms rate
: nflash 0 ?DO flash LOOP ;
\( n -) flash for n seconds
: sflash 1000 * nflash ;

\ Flash one at a time
: L0 D0 100 nflash ;
: L1 D1 100 nflash ;
: L2 D2 100 nflash ;
: L3 D3 100 nflash ;
: L4 D4 100 nflash ;
: L5 D5 100 nflash ;
: L6 D6 100 nflash ;
: L7 D7 100 nflash ;
: L8 D8 100 nflash ;
: L9 D9 100 nflash ;
: L10 D10 100 nflash ;
: L11 D11 100 nflash ;
: L12 D12 100 nflash ;

: Lall L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 L12 ;
: sLall 0 ?DO Lall LOOP ;

: Z0 L0 D0 nflash ;
: Z1 L1 D1 nflash ;
: Z2 L2 D2 nflash ;
: Z3 L3 D3 nflash ;
: Z4 L4 D4 nflash ;
: Z5 L5 D5 nflash ;
: Z6 L6 D6 nflash ;
: Z7 L7 D7 nflash ;
: Z8 L8 D8 nflash ;
: Z9 L9 D9 nflash ;
: Z10 L10 D10 nflash ;
: Z11 L11 D11 nflash ;
: Z12 L12 D12 nflash ;

\ one wire stuff
\ one-wire on/off mode words
: 1wire fl_base addr_mode_select + c! ;
: on1wire 2 1wire ;
: off1wire 0 1wire ;

\reset one-wire
: 1rst $0F fl_base c! 1000 usleep fl_base c@ 1 lshift 7 rshift not ;

: rstat fl_base c@ 2 lshift 7 rshift not ;

: wr_err crlf s" error write" type type ;

\send a 1 to one-wire
: w1 rstat if $09 fl_base c! else wr_err endif 1000 usleep;

\send a 0 to one-wire
: w0 rstat if $0A fl_base c! else wr_err endif 1000 usleep;

\read a bit/byte from one-wire
: r1 fl_base c@ 1 lshift 7 rshift ;
: r2 crlf s" error: read" type type ;

: rbit $0B fl_base c! 1000 usleep rstat if r1 else r2 endif ;
: rbyt 0 8 0 ?DO 1 rshift rbit if $80 + endif LOOP ;

\one-wire READROM command (0x33)
: rrom w1 w1 w0 w0 w1 w0 w0 w0 ;

\read back 64 bits serial number (print out is decimal)
: ser rbyt rbyt rbyt rbyt rbyt rbyt rbyt rbyt ;

\one-wire protocol: RESET -> COMMAND(READROM) -> read data
: NoDS crlf s" No DS2401 found!" type type ;
: SN on1wire 1rst if rrom ser 8 0 ?DO . drop LOOP else NoDS endif off1wire ;

: Pass ;