Stereo Lepton Manual
Rev. E

Jennifer Holt

Overview:

The Stereo Lepton is a development board for the FLIR Lepton® thermal imaging cameras. Its original purpose was as a stereoscopic night vision device, but it is also useful as a general prototyping platform. The board supports 2 cameras, 2 displays, an FPGA and a high-speed USB interface. In addition there is a 128Mbit SPI flash chip, 4 LEDs, 2 pushbuttons and test points on all camera and display signals. A block diagram showing the components of the board is shown in Figure 1: Block diagram of the Stereo Lepton board..

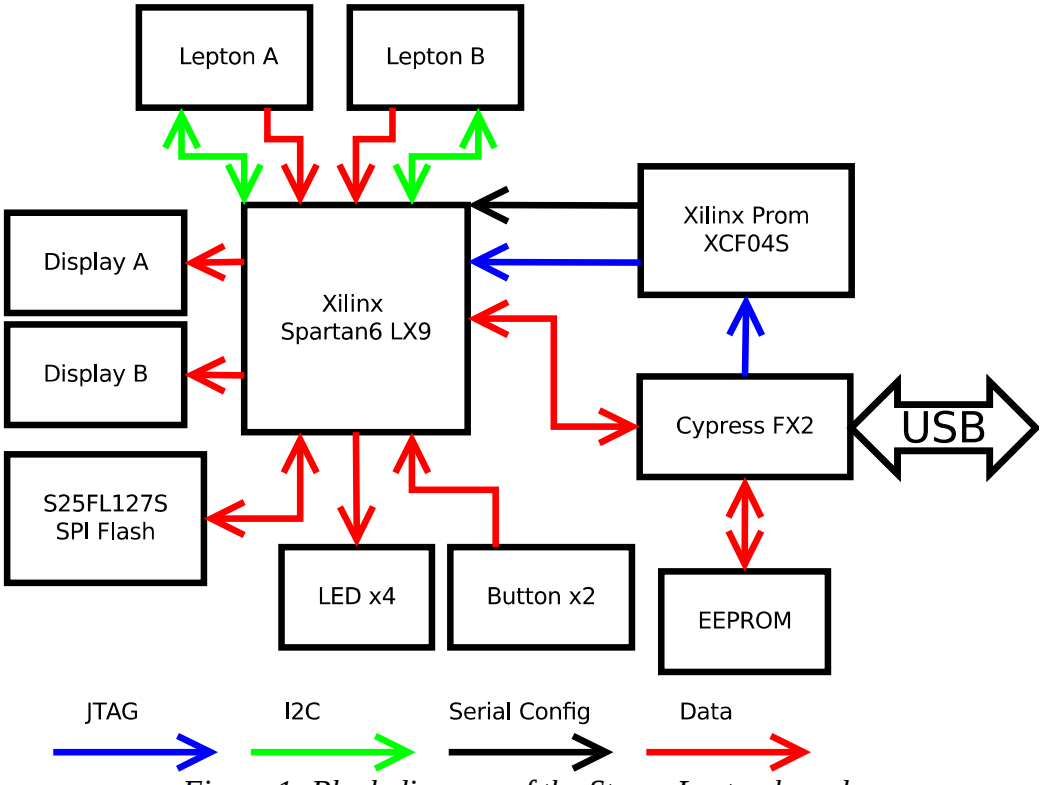

Figure 1: Block diagram of the Stereo Lepton board.

Quick Start Guide:

The USB interface of the Stereo Lepton board consists of a Cypress EZ-USB FX2LP microcontroller.

This microcontroller loads its firmware either over USB or from a serial EEPROM. On the Stereo Lepton board, the EEPROM is disconnected and un-programmed by default. There is a small solder jumper by the USB connector which will connect the EEPROM and allow for firmware to be written to non-volatile memory, but this is not necessary and is mostly useful if the board is intended to be used as a USB peripheral.

Programming of the board is accomplished over JTAG with the FX2 serving as the JTAG interface. To do this it is recommended to use the Makestuff flcli command-line utility by Chris McClelland. Chris has also written the very useful libfpgalink which is used for data communication to the FPGA. You can download his software from github at https://github.com/makestuff . Once you have the flcli utility, programming the board is done using the following command:

flcli -v 1d50:602b -i 04b4:8613 -p J:D0D1D2D3:<firmware>

The -v switch gives the VID/PID that flcli will try to communicate with. It is also the id the board will be assigned after configuration, this is a volatile setting and will be reset if the board is disconnected from USB, unless the libfpgalink firmware is flashed to the EEPROM. 1D50:602B is a VID/PID assigned to the OpenMoko project which they have given for use by libfpgalink devices http://wiki.openmoko.org/wiki/USB_Product_IDs

The -i switch gives the “initial” VID/PID which flcli will look for if the VID/PID under the -v switch is not found. 04B4:8613 are the default IDs for an un-programmed cypress FX2.

The -p switch defined which pins on the FX2 are used for jtag communication. In order they are: -p J:<TDO><TDI><TMS><TCK> These pins are detailed in Table 13: JTAG chain connections. The final part of the -p switch is the firmware file. These files are actually just a series of jtag commands which are played through the interface. These are located in the firmware folder and are named as follows:

top_e_L2.xsvf
top_e_L3.xsvf
top_e_L2_prom.xsvf
top_e_L3_prom.xsvf

The L2/L3 designation is for the Lepton version 2/3 cameras, and the _prom designation indicates that the firmware will be written to the Xilinx prom. The non-prom firmware will be lost when power is disconnected. The prom firmwares will be written to the XCF04S chip and will be loaded into the fpga on power-up, eliminating the need for programming with a computer in the future.

Principles of Operation:

There are two main subsystems on the Stereo Lepton board. The USB microcontroller, and the FPGA. The microcontroller provides a host computer interface for JTAG programming and for data communication to the design in the FPGA. The microcontroller also provides a 48MHz clock to the FPGA. This clock is used for all logic in the FPGA design, and serves as the input to a pll which provides the 25MHz master clock for the lepton cameras.

Libfpgalink (https://github.com/makestuff/libfpgalink) provides the microcontroller firmware, as well as an interface core in the FPGA. The libfpgalink interface consists of 128 “channel addresses”

to which bytes can be read or written. This can be done using various code bindings provided with the library, or with the flcli command line program. For full details, refer to the documentation provided with the libfpgalink software.

Figure2: HDL block diagram Shows the overall layout of the cores in the FPGA. The FPGA link core connects directly to the FX2LP microcontroller and handles the low level interface. This core connects to the PicoController core, which uses a Xilinx picoblaze soft processor to handle startup and configuration commands. The PicoController core can also be disabled, allowing the host system to directly interface to the other parts of the design. After the PicoController, there is a multiplexer which connects different cores to the fpgalink interface depending on which channel address is being accessed. There is one global configuration register which controls the reset status of the rest of the design, and two lepton interface cores which handle the interface to the cameras.

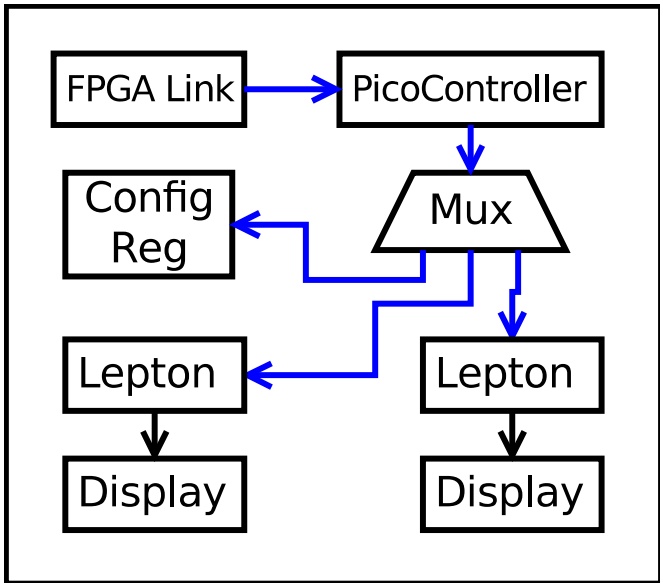

Figure2: HDL block diagram

CommFPGA_fx2wrapper.v:

This core wraps the commfpga core provided in libfpgalink into a convenient form. It provides the CommFPGA data bus to the rest of the design. The data bus consists of a 7-bit wide address, and one input and one output data channel. Each data channel has an 8-bit wide data connection and two control connections. For details of the Data/Valid/Ready system, refer to the libfpgalink documentation.

CommFPGA_PicoController.v:

This core provides a way for configuration to be automatically applied to the design without a host computer. The core is designed to sit between CommFPGA_fx2wrapper.v and the rest of the system. On power-up or reset, the picoblaze soft processor runs and has control of the output CommFPGA bus. From the host side, the PicoController has 1 8-bit register. If a 1 is written to this register, the picoblaze processor is placed in reset and the output CommFPGA bus is connected as a pass-through. This allows a host system to take control and adjust settings, read data etc. If a 0 is written to the control register, the picoblaze core comes out of reset and executes its code from the beginning.

In the Stereo Lepton board, the PicoController program handles setting the system enable bits and the I2C communication to the cameras to place them in the proper mode. In the future, this program might be expanded to include the ability to change color pallets or adjust AGC settings using

the buttons.

CommFPGA_mux.v:

This core sits between the PicoController and the other cores. Its job is to connect the CommFPGA data bus to the correct core based on the channel address. There is some simple logic external to the core which translates channel address into a mux setting. For a listing of channel addresses and which cores they connect to, see Table 1: Stereo Lepton Register Map.

Channel Address		Core	Description	Width	Direction
0x00	0	CommFPGA_PicoController.v	PicoController Enable	8	R/W
0x01	1	CommFPGA_ConfigMem.v	System Config Register	8	R/W
0x02	2	(A) Lepton.v	Lepton A Config Register	8	R/W
0x03	3	(A) Lepton.v	Lepton A Pixel Stream	8	R
0x04	4	(A) Lepton.v	Lepton A Telemetry Stream	8	R
0x05	5	(A) Lepton.v	Lepton A I2C FIFO	8	R/W
0x06	6	(A) Lepton.v	Lepton A I2C FIFO write level	8	R
0x07	7	(A) Lepton.v	Lepton A I2C FIFO read level	8	R
0x08	8	(A) Lepton.v	Lepton A I2C Status	8	R
0x09	9	(A) Lepton.v	Lepton A I2C Slave Address	16	R/W
0x0a	10	(A) Lepton.v	Lepton A I2C Bytes to Read	16	R/W
0x0b	11	(B) Lepton.v	Lepton B Config Register	8	R/W
0x0c	12	(B) Lepton.v	Lepton B Pixel Stream	8	R
0x0d	13	(B) Lepton.v	Lepton B Telemetry Stream	8	R
0x0e	14	(B) Lepton.v	Lepton B I2C FIFO	8	R/W
0x0f	15	(B) Lepton.v	Lepton B I2C FIFO write level	8	R
0x10	16	(B) Lepton.v	Lepton B I2C FIFO read level	8	R
0x11	17	(B) Lepton.v	Lepton B I2C Status	8	R
0x12	18	(B) Lepton.v	Lepton B I2C Slave Address	16	R/W
0x13	19	(B) Lepton.v	Lepton B I2C Bytes to Read	16	R/W

Table 1: Stereo Lepton Register Map

CommFPGA_configMem.v:

This core is used many times and provides a configuration memory implemented as a set of 8-bit locations. The core is configurable for the number of registers and the number of bytes in a register. Each register gets one channel address, and the register number is equal to the channel address - base address. There is internal logic which handles reading and writing multi-byte registers. Internal to the core is a byte pointer, when a read/write is initiated, the byte pointer is set to the register

number*number of bytes in a register. Each successive byte read/written increments the byte pointer by one. This means that is is possible to access all data in a configmem by referencing the first register and reading/writing bytes equal to the total size of the memory. For example, the I2C interface in the Lepton.v core has a 16-bit wide register for setting the slave address used in the I2C transaction. To write to this register, a 2-byte write must be made to channel address 0x09. It is important to do the access in a single command, since the core will reset its internal pointer after the read/write is complete. If you are using the flcli command-line interface the command would be “w09 0001” to write 0x0001 to the register. “w09 01” and then “w09 00” would write 0x01 to the first byte, and then write 0x00 to the first byte, which does not give the desired result. If 4 bytes are written to channel address 0x09, the data will continue into the next register located at channel address 0x0a.

There is a single system-wide configuration register located at channel address 0x01. The meaning of the bits is detailed in Table 2: System Config Reg.

System Config Register. ChannAddr 0x01								
Bit	7	6	5	4	3	2	1	0
	X	X	X	LB_EN	LA_EN	DB_EN	DA_EN	GB_EN
LB_EN	Lepton B enable							
LA_EN	Lepton A enable							
DB_EN	Display B enable							
DA_EN	Display A enable							
GB_EN	Global enable							

Table 2: System Config Reg

LCD_Driver2.v:

This core handles the configuration and updating of an LCD display. The displays used are surplus Nokia 6300 LCD's and are 320x240 pixels with 24-bit color. While I don't know the exact controller used in these displays, it appears to be mostly compatible with the MC2PA8201. There is also a wealth of information on these displays here: http://andybrown.me.uk/2012/06/05/nokia-lcd-for-arduino-mega-1

The core takes video timing and pixel data from the Lepton.v core and buffers the data in a dual port RAM. This buffering allows one pixel from the camera to be written to several locations on the display. This is used to up-scale the 80x60 or 160x120 pixel image from the camera to the full 320x240 resolution of the display. Due to the limited memory in the dual-port RAM, video data cannot come in faster than it can be written out to the display. In order to achieve this, the VOSIP serial clock is slower on the Lepton2 camera, to give the upscaling logic time to write 4 LCD lines for each camera video line.

Lepton.v:

This is a composite core made up of all the necessary parts to provide a complete interface to a Lepton camera. Included in this core is a CommFPGA_mux.v, a CommFPGA_configMem.v, a CommFPGA_I2CMaster.v, a Lepton_Packet.v and a Lepton_Framer.v. There is also some logic and FIFOs to allow reading the pixel byte stream and/or the telemetry byte stream directly. The Lepton_Packet, Lepton_Framer and CommFPGA_I2CMaster cores will be described in their own sections.

The Lepton.v core has one 8-bit configuration register:

Lepton Config Register. ChannAddr 0x02; 0x0b								
Bit	7	6	5	4	3	2	1	0
	X	AGC	Tmode [1:0]		Pfmt	FR_EN	PA_EN	ST_EN
AGC	AGC mode, set to 1 if AGC is enabled							
Tmode	Telemetry Mode. 00 = disabled, 01 = footer, 10 = header, 11 = reserved							
Pfmt	Pixel format. 1 = RGB888, 0=RAW14							
FR_EN	Framer enable							
PA_EN	Packet enable							
ST_EN	Startup enable							

Table 3: Lepton Configuration register

AGC, Tmode and Pfmt need to be set to match the camera configuration. The settings in the config register only affect the packet interface and the framer. If the camera settings are changed via the I2C interface, these settings must be updated to match. The enable signals are used to reset the various cores when changing modes or on startup. The recommended sequence on startup is to enable ST, PA and FR at the same time. Internal logic takes care of the sequencing. If the camera mode is changed after power-up, it is recommended to disable the packet interface and framer until all of the settings are modified, then enable both at once.

The Lepton.v core also provides a stream interface to the pixel and telemetry data. Reading from either stream will return the raw bytes from the Lepton_Packet core. Internal logic synchronizes the read to the start of a frame. All bytes corresponding to a complete frame should be read in one transaction. RAW14 pixels are stored as 16-bit integers, and RGB888 pixels are stored as 24-bit integers. So to read one complete frame from an 80x60 camera in RAW14 mode requires reading 80x60x2 = 9600 bytes.

Lepton_Packet.v:

This core provides the VOSPI interface to the Lepton camera. It syncs to the serial data, detects out of sync conditions, and provides a deserialized data stream to the framer core. The data stream consists of an 8-bit data output, 16-bit ID and CRC outputs and timing signals. A sync signal is a one system clock wide strobe that signals the start of a packet. Both the ID and CRC outputs are valid on this strobe. After the sync, there are a series of pulses on the valid signal. Each pulse is one system clock wide and signifies a valid data byte on the data output. Depending on the pixel format setting, there will be 163 or 243 data bytes. In addition to the data stream, there are 3 signals which flag each packet as normal, discard or special. These signals are used by the framer.

Lepton_Framer.v:

This core takes the deserialized packet stream from Lepton_Packet.v and formats the data into a video stream. The video output consists of a vsync and hsync signal, a pixel valid strobe and 24-bit parallel pixel data. In addition to the video output there is also a telemetry output stream. This consists of tsync and valid pulses and a 16-bit wide data word. All of the timing pulses are 1 system clock wide. The pixel/word valid pulses come as data is received. The timing is dependent on the VOSPI serial clock, and despite their names, the vsync and hsync signals are not compatible with any video

standard. Detailed timing information is available in the source code file Lepton_Framer.v

CommFPGA_I2CMaster.v:

This core provides an I2C master core with a CommFPGA bus host interface. There is an incoming and an outgoing FIFO. Both FIFOs are accessed at the base address of the core. Reads from this address take bytes from the receive FIFO and writes to this address fill the transmit FIFO. At base address + 1, there is an 8-bit wide count of the bytes in the transmit FIFO. At base address + 2, there is an 8-bit wide count of the bytes in the read FIFO. Base address + 3 is the status register, for details of this register see Table 4: I2C Status Register. Base address + 4 is a 16-bit wide register which holds the slave address, details are in Table 5: I2C Slave Address Register. Base address + 5 is a 16-bit wide register which holds a count of the number of bytes to read from a slave in a read transaction.

I2C Status Register								
Bit	7	6	5	4	3	2	1	0
	Reserved					ARB	NAK	Ready
ARB	Transcation Failed from lost arbitration							
NAK	Transaction Failed from Slave not ack							
Ready	Core is Ready							

Table 4: I2C Status Register

I2C Slave Address Register																
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	X					TB	A[9:7]			A[6:0]						
TB	1 = Use 10-bit Address, 0 = Use 7-bit Address															
A[9:7]	Top 3 bits of 10-bit Slave address															
A[6:0]	7-bit Slave Address															

Table 5: I2C Slave Address Register

To initiate a write transaction on the I2C bus, set the slave address in the slave address register, set the bytes to read register to 0, and write the bytes to send to the FIFO. It is assumed that the commFPGA bus is much faster than the I2C bus and so the FIFO will not under-run. Once all the bytes have been written to the FIFO, the status register can be polled to determine the status of the transmission. The transmission is finished when Ready is set. If the transmission failed, either NAK or ARB will be set as well.

To initiate a read transaction, first set the slave address register and the bytes to read. Next write any command bytes(such as the slave internal memory address) to the FIFO. After all the bytes in the FIFO are sent, a repeated start condition is generated and the number of bytes in the bytes to read register are read into the receive FIFO. The status register can be polled to determine the end of the transaction, and then the received bytes can be read from the FIFO.

Building the Firmware:

USB Interface:

Connections and Pinouts:

Lepton A		
Signal	Device Pin	FPGA Pin
GPIO3	2	119
VOSPI MISO	12	112
VOSPI CLK	13	111
VOSPI nCS	14	114
SCL	21	115
SDA	22	116
nPWD	23	117
nRST	24	118
MCLK	26	120

Table 6: Lepton A connections

Lepton B		
Signal	Device Pin	FPGA Pin
GPIO3	2	131
VOSPI MISO	12	142
VOSPI CLK	13	143
VOSPI nCS	14	141
SCL	21	140
SDA	22	139
nPWD	23	138
nRST	24	133
MCLK	26	132

Table 7: Lepton B connections

Display A		
Signal	Device Pin	FPGA Pin
D0	6	92
D1	18	97
D2	8	94
D3	17	100
D4	9	95
D5	15	105
D6	10	98
D7	14	101
DCX	19	93
WRX	5	88
RESX	12	99
TE	13	104
Back Light EN	N/A	85

Table 8: Display A connections

Display B		
Signal	Device Pin	FPGA Pin
D0	6	9
D1	18	6
D2	8	10
D3	17	5
D4	9	12
D5	15	8
D6	10	14
D7	14	15
DCX	19	22
WRX	5	7
RESX	12	16
TE	13	11
Back Light EN	N/A	1

Table 9: Display B connections

SPI Flash		
Signal	Device Pin	FPGA Pin
nCS	1	35
SCK	6	24
SIO0	5	23
SIO1	2	34
SIO2	3	27
SIO3	7	26
Power Enable	N/A	40

Table 10: SPI Flash connections

Buttons and LED	
Signal	FPGA Pin
LED0	47
LED1	46
LED2	45
LED3	41
Button 1	48
Button 2	38

Table 11: Buttons and LEDs connections

Cypress FX2LP		
Signal	Device Pin	FPGA Pin
FD0	25	83
FD1	26	82
FD2	27	81
FD3	28	80
FD4	29	74
FD5	30	75
FD6	31	78
FD7	32	79
FLAGB	37	67
FLAGC	38	66
SLRD	8	57
SLWR	9	58
SLOE	42	57
FIFOADR1	45	51
PKTEND	46	50
IFCLK	20	84

Table 12: FX2LP connections

JTAG			
Signal	Cypress FX2	XCF042	XC6SLX9
TMS	Pin 54 (PD2)	5	107
TCK	Pin 55 (PD3)	6	109
PROM TDI	Pin 53 (PD1)	4	N/A
PROM TDO/FPGA TDI	N/A	17	110
FPGA TDO	Pin 52 (PD0)	N/A	106

Table 13: JTAG chain connections

Test Points (see Figure 3: Test Point locations)			
Signal	Test Point	Signal	Test Point
GND	1	Display A D0	30
2.8 V	2	Display A D4	31
1.2 V	3	Display A RESX	32
GND	4	Display A TE	33
Display B WRX	5	Display A DCX	34
Display B D2	6	Display A D1	35
Display B D6	7	Display A D3	36
Display B DCX	8	Display A D5	37
Display B D1	9	GND	38
Display B D0	10	Lepton A VOSPI CLK	39
Display B D4	11	Lepton A VOSPI nCS	40
Display B RESX	12	Lepton A SDA	41
Display B D3	13	Lepton A nRST	42
Display B D5	14	Lepton A Master CLK	43
Display B TE	15	GND	44
Display B D7	16	Lepton A VOSPI MISO	45
Flash 3.3 V	17	Lepton A SCL	46
GND	18	Lepton A nPWD	47
LED3	19	Lepton A GPIO3	48
LED2	20	Lepton B VOSPI MISO	49
LED1	21	Lepton B SCL	50
LED0	22	Lepton B nPWD	51
USB 5 V	23	Lepton B Master CLK	52
GND	24	Lepton B VOSPI CLK	53
3.3 V	25	Lepton B nCS	54
Display A WRX	26	Lepton B SDA	55
Display A D2	27	Lepton B nRST	56
Display A D6	28	Lepton B GPIO3	57
Display A D7	29	GND	58

Table 14: Test Point connections

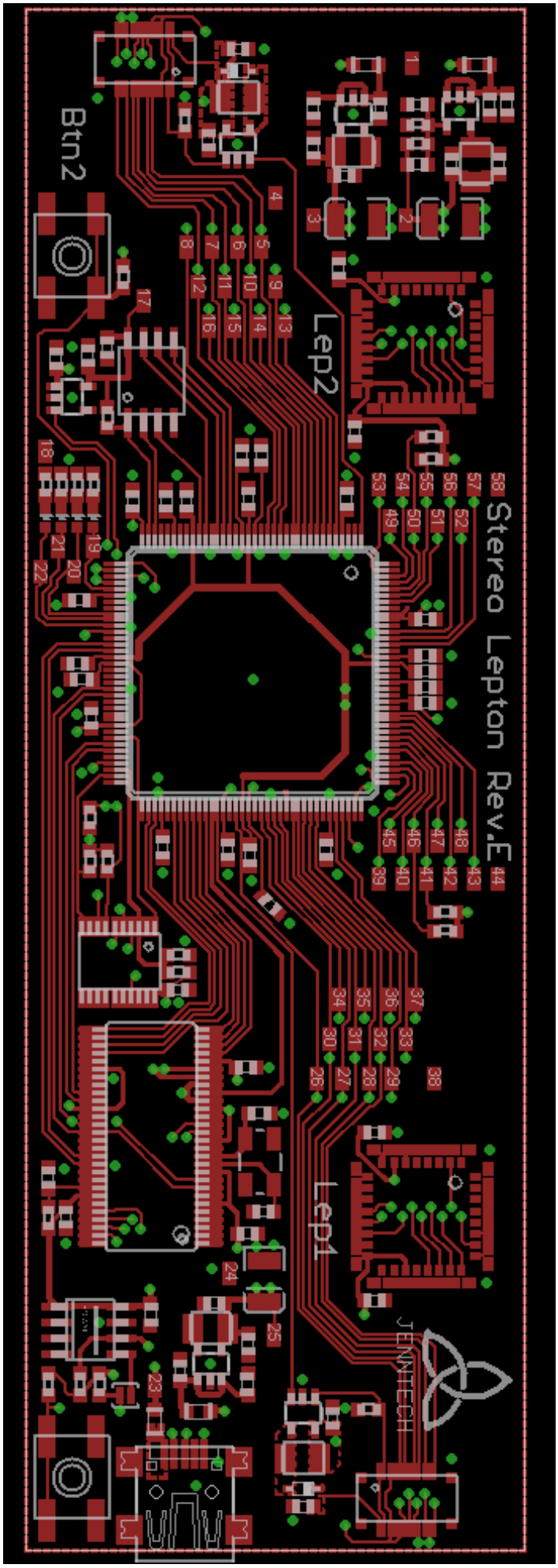

Figure 3: Test Point locations