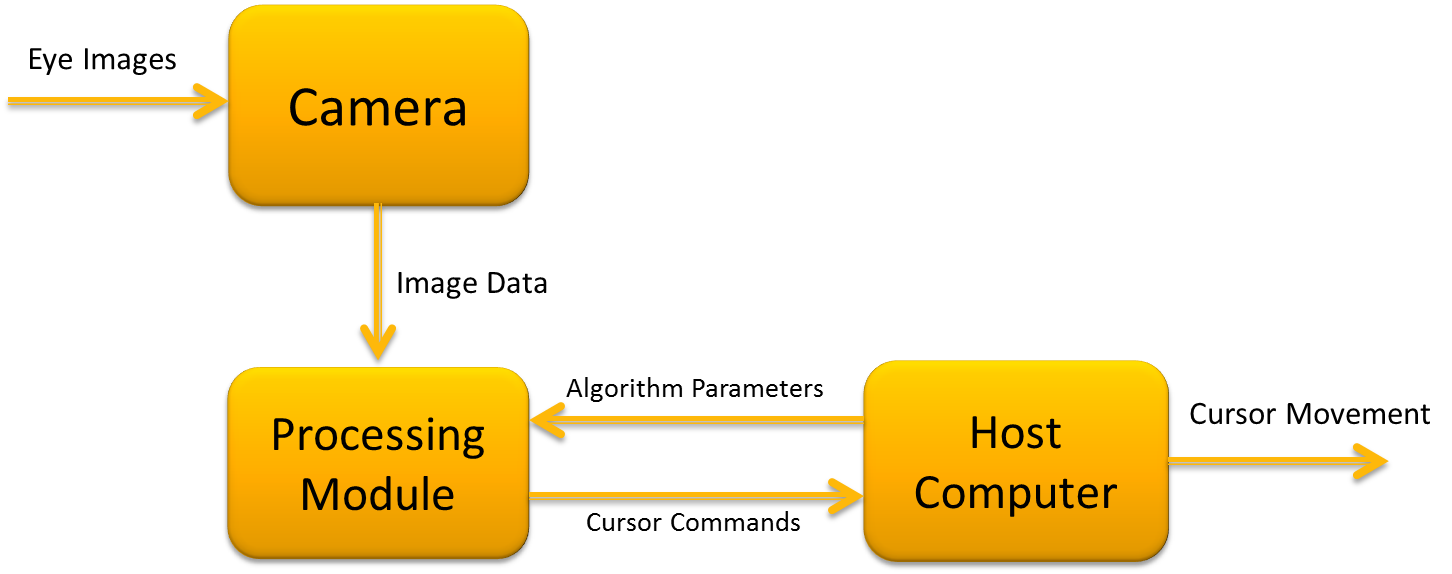
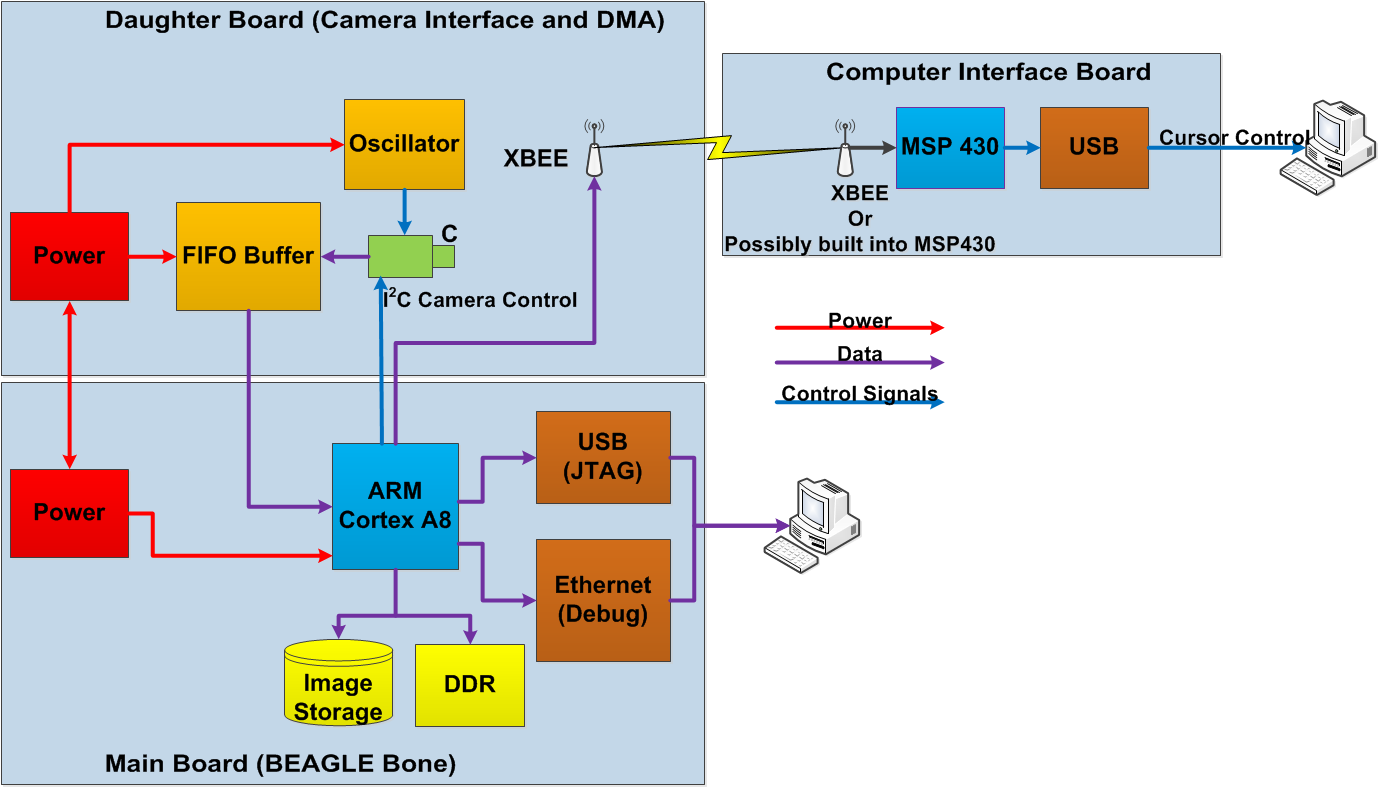
**Level 0: System Overview**

****

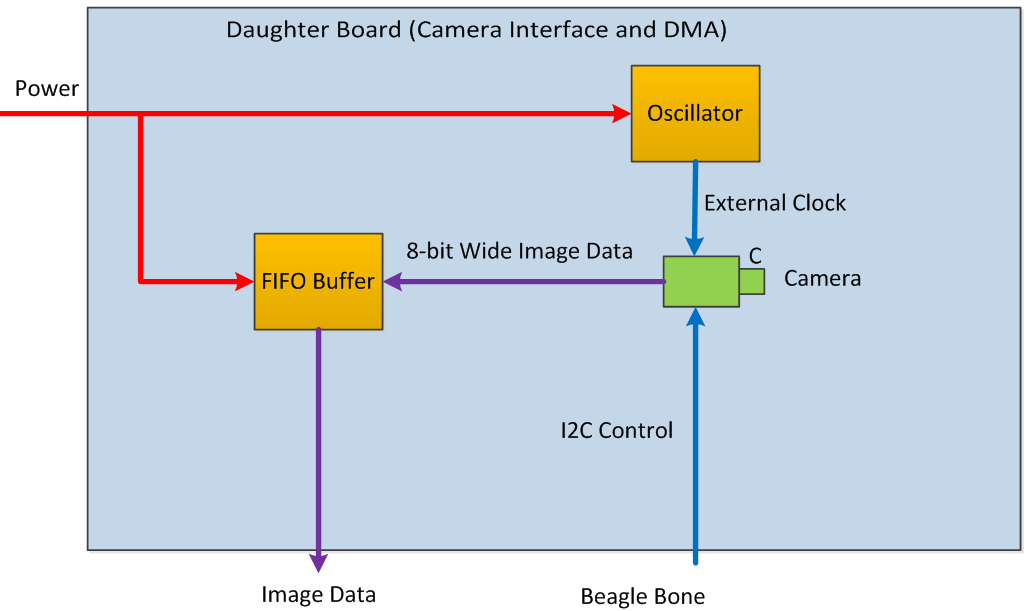
**Figure 1**

|  |  |
| --- | --- |
| **Module** | **Eye tracking system** |
| **Inputs** | Visual image of the eye |
| **Outputs** | Cursor movement |
| **Functionality** | Eye images will be taken by the camera. The image data will be processed by the processing module to determine the user’s gaze. The user’s gaze will be translated to cursor commands and will be sent to the host computer. User profiles will require initial calibration, where algorithm parameters are tailored to the specific individual. These parameters are sent from the host computer to the processing module. |
| **Test Plan** | Verify that eye movements correspond to correct cursor movements. |

**Level 1: Hardware**

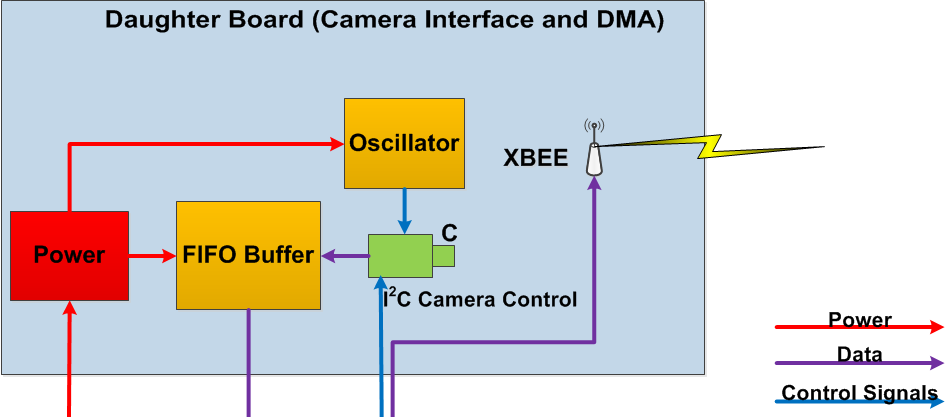


**Figure 2**



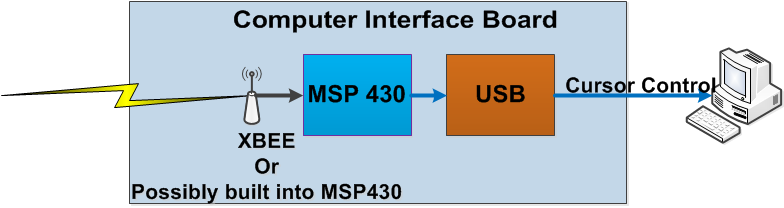
**Figure 3**

|  |  |
| --- | --- |
| **Module** | **Camera board** |
| **Inputs** | Visual data |
| **Outputs** | 8-bit data bus with image information in RGB 5:6:5.  HD, VD, DCLK |
| **Functionality** | The camera board houses the camera and supporting hardware. The camera will output data in the configuration above. Each pulse of DCLK signifies another 8-bit parallel chunk is ready to be read. HD signifies the end of a line in the 640x480 resolution of the final images while VD signifies the end of the image. |
| **Test Plan** | For early testing, observation of the DCLK will be used to see if the camera is outputting any data at all. As the project progresses, manual inspection  of the data received on the Beagle Bone and finally streaming the video data over Ethernet. |



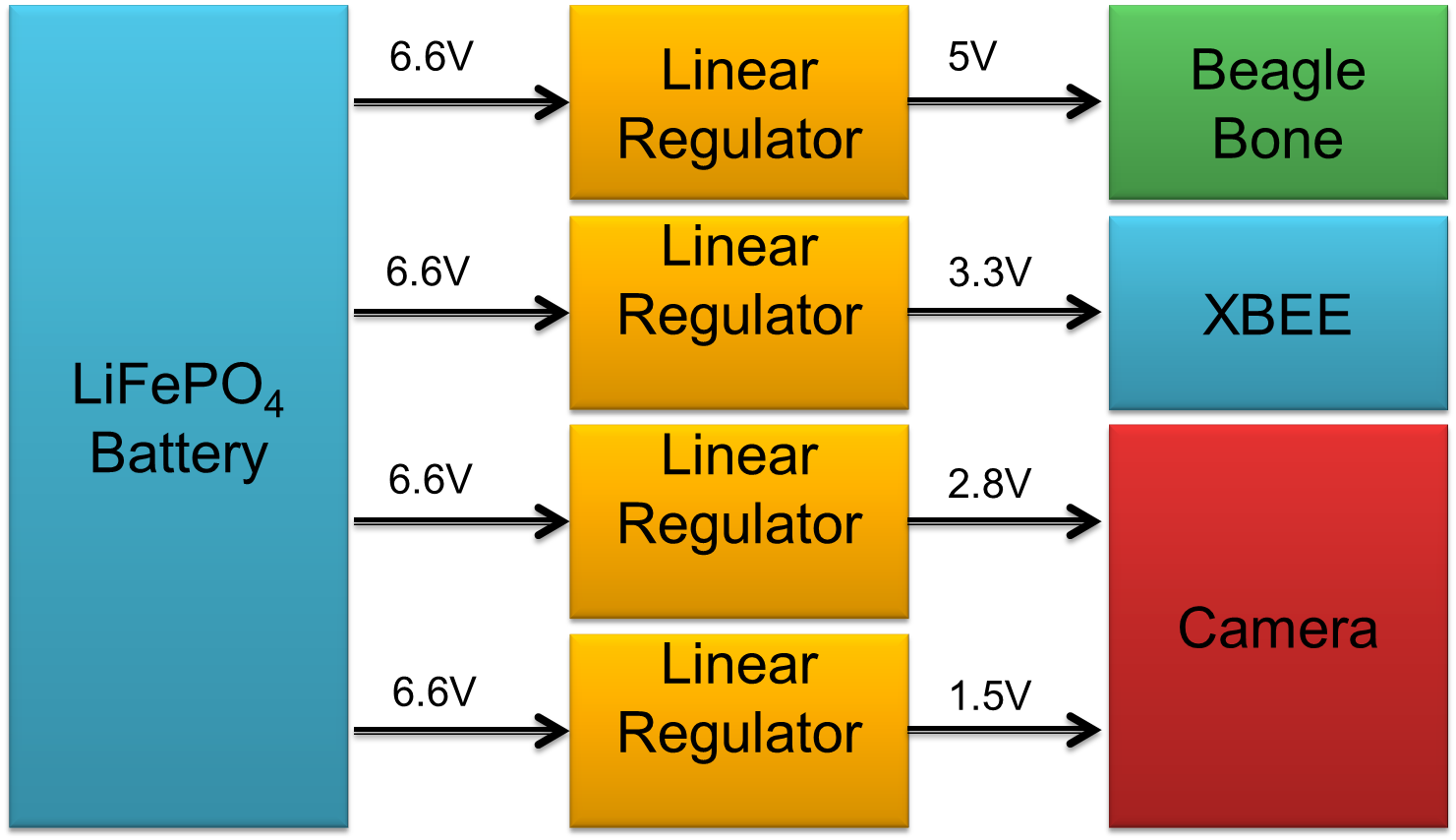
**Figure 4**

|  |  |
| --- | --- |
| **Module** | **Daughter Board** |
| **Inputs** | 8-bit parallel image data  VD, HD, DCLK  Serial data from MSP430 board |
| **Outputs** | Buffered camera data.  Serial data to MSP430 board  VD, HD |
| **Functionality** | This board will read data from the camera board, and buffer it. The buffer utilizes DCLK to read the data in. VD and HD are sent along to the Beagle Bone to be used in Image Processing algorithms. Serial data will be both sent and received using an XBee. |
| **Test Plan** | Before connecting anything, test the power rails gong to the camera are 1.5V and 2.8V with tolerance Manually probing these with a multimeter would be sufficient.  To test data transfer to the Beagle Bone, either read the data on the Beagle, or utilize a logic analyzer to see the data changing.  The same methods can be used to test the XBee functionality. |



**Figure 5**

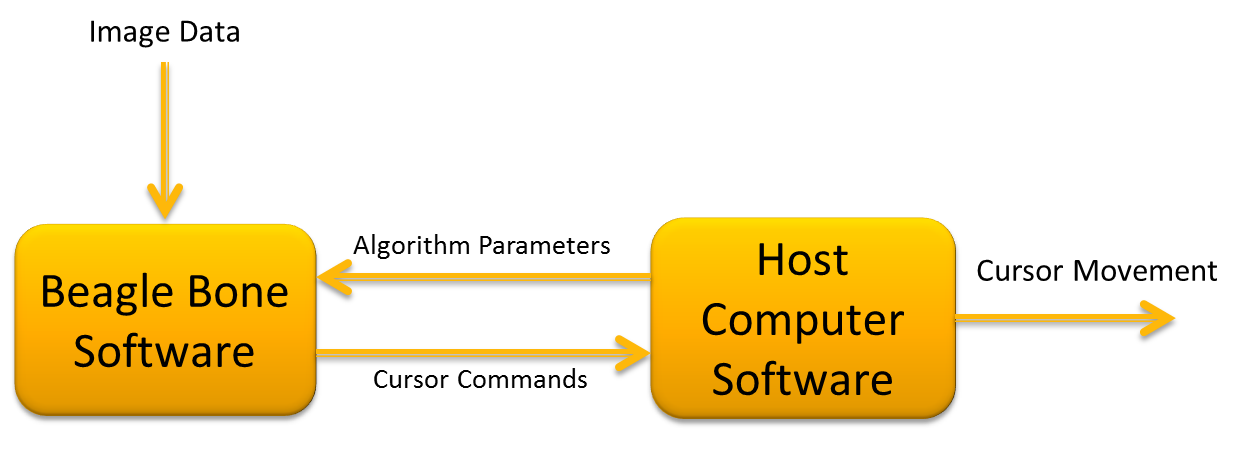
|  |  |
| --- | --- |
| **Module** | **MSP430 Board** |
| **Inputs** | Serial data from daughterboard/Beagle Bone and  Calibration data from host computer |
| **Outputs** | Cursor movement commands |
| **Functionality** | This board will be a liaison between the Beagle Bone and the host computer, effectively shuffling data from the Beagle to the host computer and back. The data will already be in the final forms. This board may be extended to have a switching mechanism to turn off/on eye tracking cursor control. |
| **Test Plan** | Initial testing will verify that the host computer can receive data packets by displaying them on the host computer. To test the XBee connections, we will send and receive packets from the Beagle Bone. The packets will be compared with what we know we are sending to verify no data was lost. To test the extended switching functionality, flip the switch to the off state and verify that eye tracking is disabled, flip it to the on state and verify that eye tracking resumes. |

****

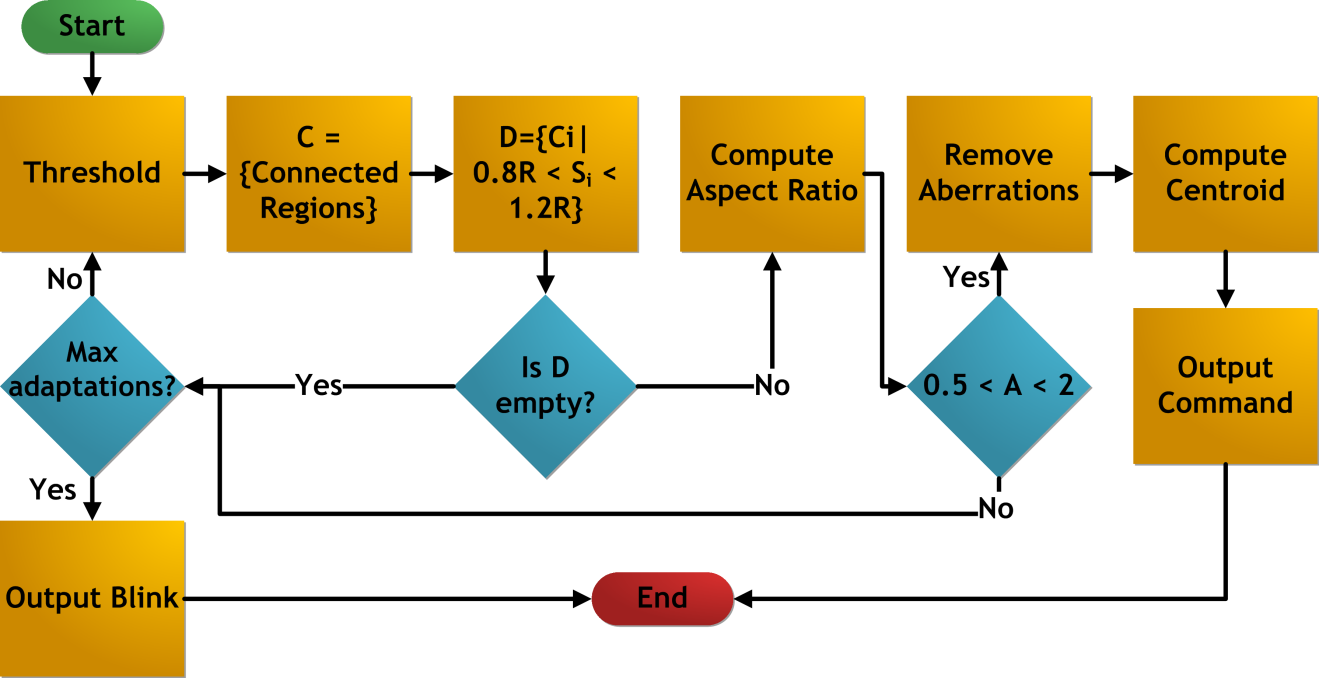
**Figure 6**

|  |  |
| --- | --- |
| **Module** | **Power Supply** |
| **Inputs** | 6.6V from LiFePO4 battery |
| **Outputs** | Linear Regulators output 5V, 3.3V, 2.8V, and 1.5V voltage rails to hardware components. |
| **Functionality** | To provide power to Beagle Bone, Beagle Bone XBEE, and Camera. |
| **Test Plan** | Using a multi-meter, all voltages will be measured to ensure that all input and output voltages will be within a specified tolerance. |

**Level 1: Software**



**Figure 7: eyeCU software overview**



**Figure 8: Beagle Bone software overview**

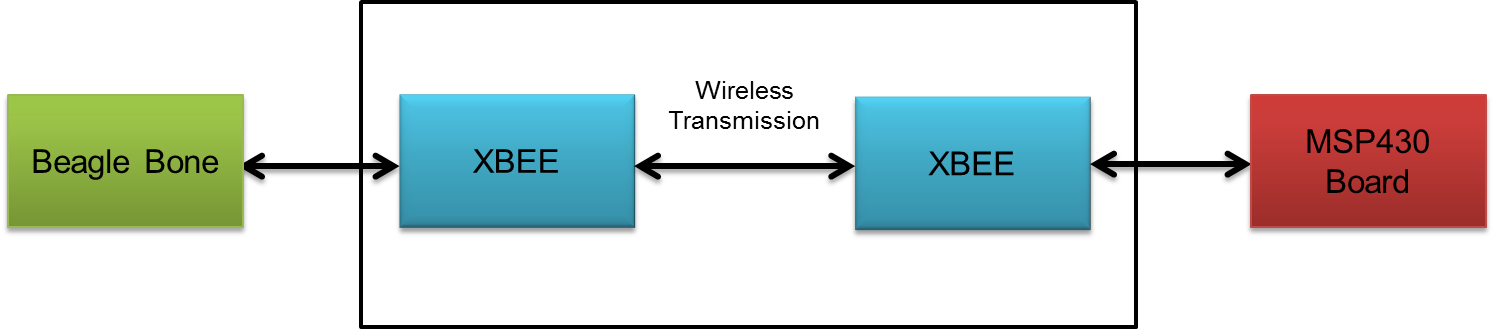
|  |  |
| --- | --- |
| **Module** | **Beagle Bone Software** |
| **Inputs** | 640x480 RGB image data |
| **Outputs** | Command indicating the direction of the user’s gaze |
| **Functionality** | Processes the image to obtain the centroid of the pupil. It then compares the pupil centroid to the reference centroid to determine the direction of gaze. |
| **Test Plan** | Before the Beagle Bone is ready, sample images will be collected via a webcam and processed using an implementation of the algorithm that runs on a computer. Once the Beagle Bone and camera are ready, real-time testing can be performed and the direction of the user’s gaze can be indicated on the LCD. |

|  |  |
| --- | --- |
| **Module** | **Host Computer Software** |
| **Inputs** | Processed calibration frame during the calibration mode, Cursor Command in normal mode. |
| **Outputs** | Cursor movement, GUI interface, User calibration values |
| **Functionality** | Initiates a GUI that allows user to interface with the software. The user would be capable of start/stopping gaze tracking software, and adjusting algorithm parameters during the calibration process. The Host computer software will also parse the cursor command from the Beagle Bone and move the cursor accordingly. |
| **Test Plan** | Verify that user can interact with the GUI to |

**Level 1: Firmware**

|  |  |
| --- | --- |
| **Module** | **Beagle Bone** |
| **Inputs** | Data from the Camera  XBee receive data |
| **Outputs** | Control flow to the camera over I2C |
| **Functionality** | The Beagle Bone firmware is in charge of running the camera, providing data to the DSP software handling communication with the XBee module and communicating with a host computer over USB and Ethernet for debugging purposes. |
| **Test Plan** | Each module will be checked for functionality and once the constituent modules are working at a satisfactory level, the system will be integrated and tested for overall functionality. |

|  |  |
| --- | --- |
| **Module** | **MSP430 FTDI** |
| **Inputs** | Serial data from the MSP430  USB data from the host computer |
| **Outputs** | Serial data to the MSP430  USB data to the host computer |
| **Functionality** | This module will facilitate communication between the host computer and the MSP430 board. The main data that will be transferred is cursor commands going to the host computer, and calibration data coming from the host computer. |
| **Test Plan** | To test the functionality of the data path packets will be sent from the MSP430 to the host computer where they will be checked against what was sent. If the same data that was sent is received, the test passes. Data will also be sent from the host computer to the MSP430 and checked on each side. |

****

**Figure 9**

|  |  |
| --- | --- |
| **Module** | **Wireless** |
| **Inputs** | Wireless data generated by Beagle Bone. UART data from the MSP430. |
| **Outputs** | UART data to the MSP430 board. Wireless data to the Beagle Bone. |
| **Functionality** | The XBEE wireless will have two-way communication. The Beagle Bone XBEE will be able to send cursor commands to the MSP430 XBEE. And the MSP430 XBEE will send algorithm parameter data the Beagle Bone XBEE. |
| **Test Plan** | For initial wireless tests, a string of characters will be sent from one XBEE to the other XBEE and vice versa to verify that there is two-way communication. For this portion of testing, one XBEE will use a USB explorer and the other XBEE will use the MSP430.  When the Beagle Bone XBEE is operational, the testing will check if the correct data packets of cursor commands or algorithm parameters for calibration are being sent correctly between the Beagle Bone and the MSP430 board. The data will be verified by inspection of data received by each board. |

**Level 2: Primary Hardware Components**

|  |  |
| --- | --- |
| **Module** | **Camera** |
| **Inputs** | I2C initialization and control data  Clock signal (24.5 MHz) |
| **Outputs** | 8-bit parallel data |
| **Functionality** | The camera is the TCM8230MD. The camera is capable of being cocked at different rates, but for 30 fps out a 24.5 MHz crystal is used. The data output and input must undergo level shifting to translate from 3.3V logic (Beagle Bone) to 2.8 V logic (camera). |
| **Test Plan** | This module will be tested at first by attempting to set the parameters for operation via I2C. Functionality is verified at this stage by probing DCLK and looking for 18.4 MHz signal coming out. The data can also be probed to look for transitions. Once the Beagle Bone is ready to accept the data, the images will be stored initially for retrieval and will be visually inspected. |

|  |  |
| --- | --- |
| **Module** | **Buffer** |
| **Inputs** | 8-bit parallel data from the camera. |
| **Outputs** | 8-bit parallel data to the Beagle Bone |
| **Functionality** | The buffer is a hardware buffer with 9 inputs and outputs. The 9th data line is ignored, as we only need 8-bits. The buffer is not large enough to store a complete image, and as such should be checked for data as often as possible. The buffer has four programmable flags to show the status of the buffer. EF – empty flag, PAE – programmable almost empty flag, PAF – programmable almost full flag, FF – full flag. |
| **Test Plan** | To test the buffer we will first verify the data is able to pass through the buffer unmolested and be read by the Beagle Bone. The image will be visually inspected for corruption of data. To test the flags, we will drive the buffer to either full or empty and visually confirm the LEDs attached to each line light up appropriately. The lines could also be directly probed for voltage level. |

|  |  |
| --- | --- |
| **Module** | **Oscillator driver** |
| **Inputs** | None |
| **Outputs** | 24.5 MHz clock signal |
| **Functionality** | The oscillator driver is a SN74LVC1404 from TI. The chip utilizes an external crystal oscillator and power to drive a clock signal for chips. We are using a 24.5 MHz crystal to clock the camera for 30 fps out of the camera. |
| **Test Plan** | As crystals are sensitive to capacitance changes, we are unable to directly test the oscillator circuit. Instead, we will observe the DCLK coming from the camera for 18.4 MHz, which would indicate that the camera is outputting data corresponding to 30 fps. |

**Level 2: Firmware Modules**

|  |  |
| --- | --- |
| **Module** | **Camera** |
| **Inputs** | Camera data |
| **Outputs** | Camera control |
| **Functionality** | This driver is in charge of running the camera. It will be responsible for receiving data from the camera and storing it appropriately for the DSP software to access. It will also be required to configure and control the camera over I2C. |
| **Test Plan** | To test this module, each of the smaller functions that comprise it will need to be tested first. When they are working at a satisfactory level, the module will be verified as a whole by inspection. If we are able to get data from the camera, then we successfully configured it and data is being received by definition. We will also verify that the data is correct by inspecting the image that is taken. |

|  |  |
| --- | --- |
| **Module** | **Serial** |
| **Inputs** | Rx |
| **Outputs** | Tx |
| **Functionality** | This module is the communications interface for the various UART serial devices we have. These include the XBee and FTDI. Rx signifies data received from the microcontrollers and Tx signifies data sent to the microcontrollers. |
| **Test Plan** | Communication between the devices will be tested by sending data between different parts of the system and verifying that the data is received with no corruption. The data will be verified by inspection and comparing it to what was sent. |

****

**Figure 10**

|  |  |
| --- | --- |
| **Module** | **Beagle Bone XBEE** |
| **Inputs** | Cursor commands via UART generated by Beagle Bone. |
| **Outputs** | Cursor commands wirelessly to the MSP430 XBEE. |
| **Functionality** | The XBEE unit will be connected to the UART pins of the Beagle Bone. The XBEE will wirelessly transmit cursor commands generated by the Beagle Bone to the MSP430 XBEE using the IEEE 802.15.4 protocol. |
| **Test Plan** | To test the functionality, verify that the Beagle Bone is sending data by probing the Rx and Tx lines going to the XBee, then inspect the data that the MSP430 receives to see if it is the same data as what was sent. |

****

**Figure 11**

|  |  |
| --- | --- |
| **Module** | **MSP430 XBEE** |
| **Inputs** | Algorithm parameters from the MSP430 Board. |
| **Outputs** | Algorithm parameters to the Beagle Bone XBEE. |
| **Functionality** | The XBEE unit will be connected to the UART pins of the MSP430. The XBEE will wirelessly transmit the algorithm parameters from the MSP430 board to the Beagle Bone XBEE using the IEEE 802.15.4 protocol. |
| **Test Plan** | To test this module, first check to make sure that the Tx lines going to the XBee from the MSP430 are carrying data by probing them. Then, check that algorithm calibration parameter packets are sent to the Beagle Bone XBEE and that the Beagle Bone receives the algorithm calibration parameter packets with no error by inspection. |

**Level 2: Calibration**

|  |  |
| --- | --- |
| **Module** | **Calibration** |
| **Inputs** | 640x480 RGB calibration frames |
| **Outputs** | Reference pupil centroid, reference pupil area, processing region |
| **Functionality** | A GUI interface on the host computer displays the images being captured by the camera with an overlay of the processing. The GUI allows the user to modify algorithm parameters. After some parameters are chosen manually, the user will be guided through a process to collect the remaining calibration values. |
| **Test Plan** | Visually verify that the modified parameters result to a modified overlay. Once this has been tested, verify that the parameters were successfully sent to the Beagle Bone by displaying them on the screen. |

**Level 2: Cursor Movement**

|  |  |
| --- | --- |
| **Module** | **Cursor Movement** |
| **Inputs** | Command code |
| **Outputs** | Computer cursor movement |
| **Functionality** | Uses Windows API to get and set cursor position. Command code is translated into a direction vector. Updated cursor position will be set to old cursor position plus speed multiplied by the direction vector. |
| **Test Plan** | Compile a list of simulated cursor commands and visually verify that the cursor moves as desired. |

**Level 2: Serial Communication**

|  |  |
| --- | --- |
| **Module** | **Serial Communication** |
| **Inputs** | Packets from MSP430 over USB |
| **Outputs** | Data extracted from packets |
| **Functionality** | Receives packets from the MSP430 at rate of 30Hz, extracts commands codes from the packet and puts them into a queue to wait to be processed by the cursor movement module. |
| **Test Plan** | Program the MSP430 board to output a set of simulated data and verify that it is correctly received by the host computer. The data is inspected to verify that is the same as the data that was sent. |

**Level 2: Beagle Bone DSP**

**Primary Software Data Structures:**

The following data structures are commonly used in the algorithm. They are presented first

|  |  |  |
| --- | --- | --- |
| **Data Structure Name:** | **Data Structure Type** | **Data Stored** |
| **imageData** | Three dimensional array of bytes | The RGB pixel values for each coordinate in a given frame. |
| **crPointList** | Two dimensional array of points | Each row contains the coordinates of a connected region. Only regions that meet the area requirement are stored. |
| **crBinary** | Three dimensional array of binary Value | Binary(i,x,y) = 1 if the coordinate (x,y) is in region i, and 0 otherwise |
| **crMap** | Three dimensional array of integers | Contains the integer index which maps a point from the matrix form of the region to an element in crPointList. |
| **crSize** | Array of integers | Elements are the sizes of the connected regions. |
| **crCount** | Integer | Number of connected regions stored. |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| **Module Name:** | **threshold()** |
| **Inputs:** | imageData, initial threshold |
| **Outputs:** | List of points that satisfy threshold criteria |
| **Functional Description:** | Scans each pixel in the region of interest in a frame and checks to see which pixels are dark enough to belong to the pupil. This process is repeated until a region (computed with getConnectedRegions()) with an area close to a reference area is found, or until a maximum number of iterations has been reached. If the maximum number of iterations is reached and no suitable regions are detected, identify the user as blinking. |
| **Test Plan:** | Color the dark pixels as red, and visually inspect the image to ensure that pixels that meet the threshold requirement have been marked. |

|  |  |
| --- | --- |
| **Module Name:** | **getConnectedRegions()** |
| **Inputs:** | List of dark points identified in threshold() |
| **Outputs:** | crPointList, crSize, crCount,crBinary |
| **Functional Description:** | Uses a stack based implementation of the flood fill algorithm to identify connected regions of dark points. |
| **Test Plan:** | Color each connected region that meets the size requirement a different color, and visually inspect the resulting image. |

|  |  |
| --- | --- |
| **Module Name:** | **getAspectRatio()** |
| **Inputs:** | crBinary, crCount |
| **Outputs:** | Aspect ratio for each connected region in CR, index of the connected region with aspect ratio nearest to one |
| **Functional Description:** | Computes the ratio of the longest horizontal and longest vertical lengths. The connected region with the aspect ratio closest to one is identified as the pupil. |
| **Test Plan:** | Print out a list of the aspect ratios computed and visually inspect an image with the connected regions in CR. |

|  |  |
| --- | --- |
| **Module Name:** | **removeAberrations()** |
| **Inputs:** | crPointList, crMap, crSize, Index indicating chosen region |
| **Outputs:** | Updated crPointList and crSize |
| **Functional Description:** | Computed the number of pixels in each row of the connected region and find the mean and standard deviation of the pixel counts. Remove rows that have pixel counts that fall out of a certain number of standard deviations away from the mean. Repeat the process in the vertical direction. |
| **Test Plan:** | Display the image with the chosen region before and after removal of aberrations and verify that aberrations have indeed been removed. |

|  |  |
| --- | --- |
| **Module Name:** | **computeCentroid()** |
| **Inputs:** | crPointList, crSize, Index indicating chosen region |
| **Outputs:** | Coordinates of the centroid |
| **Functional Description:** | Sum the coordinates of all points belonged to the pupil region and divide by the total number of points. The result is the coordinate of the centroid. |
| **Test Plan:** | Indicate the centroid with horizontal and vertical lines and verify by visual inspection that the intersection falls on the centroid of the region. |

**Level 3: Calibration**

|  |  |
| --- | --- |
| **Module** | **GUI** |
| **Inputs** | User adjusted parameters |
| **Outputs** | Visualization of the image processing |
| **Functionality** | Allows the user to control parameters in the algorithm such as initial threshold value and cursor speed. The image processing is visualized by overlaying colored regions over the original image. Also it allows user to pause eye controlled cursor movement. |
| **Test Plan** | Test to see if changes in parameters correspond to correct modification in the overlaying image. |

|  |  |
| --- | --- |
| **Module** | **Indicators** |
| **Inputs** | User starts calibration |
| **Outputs** | Visual cues to guide the user through calibration, Reference pupil centroid, reference pupil area, processing region |
| **Functionality** | Displays on-screen indicators to tell the user to look at a series of calibration points. From these points, a processing region and reference pupil size and location can be determined. |
| **Test Plan** | Save the frames used for each step in calibration, and manually verify that the parameters generated are correct. |

**Level 3: Camera Firmware**

|  |  |
| --- | --- |
| **Module** | **I2C.c** |
| **Inputs** |  |
| **Outputs** | I2C data (SDA, SCL) |
| **Functionality** | I2C drive. Sets up basic I2C protocol for communication with the camera |
| **Test Plan** | Using a logic analyzer, probe the SDA and SCL lines to inspect that the signals comply with the I2C protocol.  Once functionality of individual functions has been confirmed also probe camera output of DCLK along with input I2C signals from Beagle Bone. |

|  |  |
| --- | --- |
| **Module** | **I2CMCUInit()** |
| **Inputs** | I2C Initialization parameters for clocking and I2C Master or Slave Mode |
| **Outputs** | Beagle Bone initialized as I2C Master and clock frequency is 100kbps. |
| **Functionality** | Beagle Bone board I2C initialization. Clock speed is set directly from crystal on evaluation board. GPIOs have been enabled for I2C communication. Master/Slave have been enabled and the I2C SCL speed is set, 100kbps. |
| **Test Plan** | Use logic analyzer to confirm SCL frequency by sending an I2C command across GPIOs. Hook analyzer up to SDA0, SCL0, and GND on Port B of the Beagle Bone . |

|  |  |
| --- | --- |
| **Module** | **I2CMasterWrite()** |
| **Inputs** | Register address and Data |
| **Outputs** | Byte of data has been written to camera. |
| **Functionality** | Performs necessary protocol to write a byte of specified data to camera to at indicated register.  The overall sequence for a write to the camera is the following:  Master sends Start Command, then the 7bit Slave Address (MSB) is sent followed by a zero. The Master then waits for the Slave to generate an Acknowledge. Once the Acknowledge is received, the 8bit register address is sent and the Master waits again for an Acknowledge generated by the Slave. After the Acknowledge, the Master sends n bytes of data, each followed by an Acknowledge from the Slave. Following the nth byte of data and the associated Acknowledge from the Slave, the Master sends a Stop command to the Slave indicating the end of transmission to that particular Slave register address. |
| **Test Plan** | Use logic analyzer to confirm proper data has been sent out on I2C lines and the addresses are correct. Attach the analyzer to the SCL and SDA lines, making sure that it is properly grounded.  Use the I2CMasterRead() function to check that intended registers were properly set.  Can only be called after I2CMCUInit() has been successfully run. |

|  |  |
| --- | --- |
| **Module** | **I2CMasterRead()** |
| **Inputs** | Camera register that will be read. |
| **Outputs** | Byte of data has been read from the camera. |
| **Functionality** | Performs the necessary protocol to read n bytes of data from registers within the camera.  The overall sequence for a read to the camera is the following:  Master sends Start Command, then the 7bit Slave Address (MSB) is sent followed by a zero. The Master then waits for the Slave to generate an Acknowledge. Once the Acknowledge is received, the 8bit register address is sent and the Master waits again for an Acknowledge generated by the Slave. After the Acknowledge, the Master sends the Start command followed by the 7bit register address padded at the end with a one. The Slave then sends the first byte of data at that address and then the Slave waits for an Acknowledge generated by the Master. Once all the data at that address has been transmitted from the Slave then the Master sends the Stop command.  This module initializes the camera for operation. Parameters that need to be set include; Frame rate, Image output format, synchronizations, and output mode. This module calls I2CMasterWrite(). |
| **Test Plan** | Use logic analyzer to confirm proper data has been sent out on I2C lines and the addresses are correct. Attach the analyzer to the SCL and SDA lines, making sure that it is properly grounded.  Use the I2CMasterWrite() function to check that intended registers were properly read.  Can only be called after I2CMCUInit() has been successfully run. |

|  |  |
| --- | --- |
| **Module** | **I2CInitCamera()** |
| **Inputs** | User defined camera configuration settings (i.e. RGB, VGA, output mode of synchronization signals) |
| **Outputs** | Camera has been configured to output RGB 5:6:5, VGA, and VD/HD are set to output. |
| **Functionality** | This module initializes the camera for operation. Parameters that need to be set include; Frame rate, Image output format, synchronizations, and output mode. This module calls I2CMasterWrite(). |
| **Test Plan** | Use logic analyzer to confirm proper configuration settings have been sent out on I2C lines. Attach the analyzer to the SCL and SDA lines, making sure that it is properly grounded. |