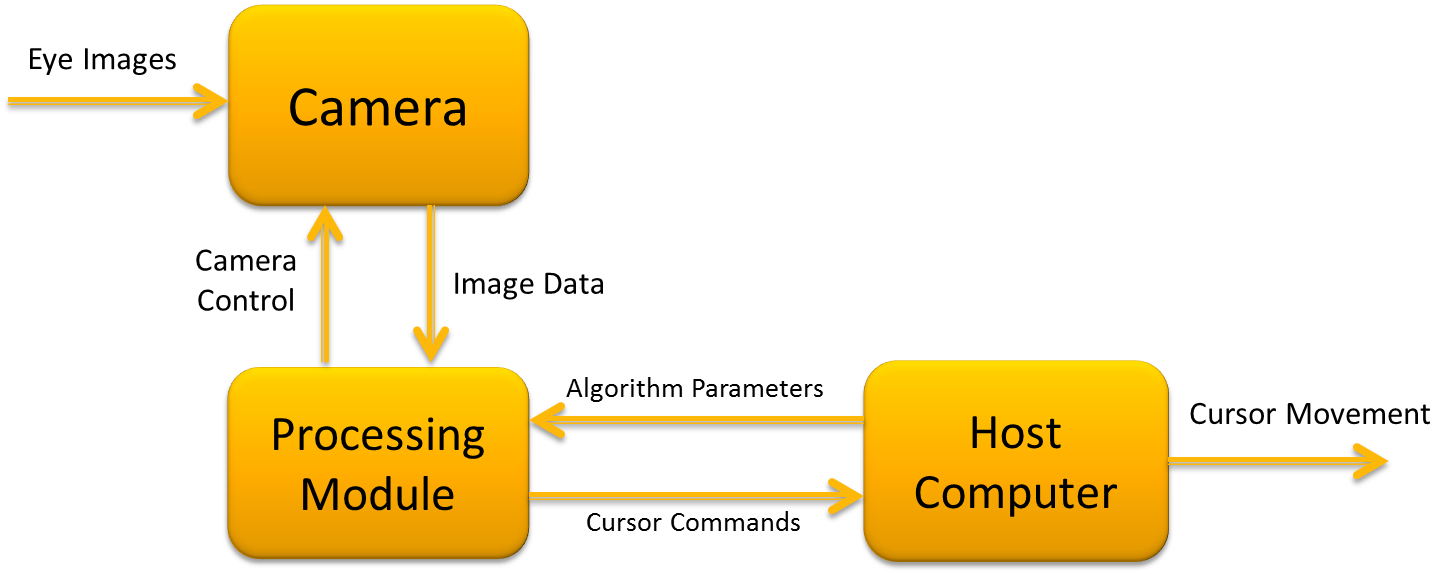
**Level 0: System Overview**

****

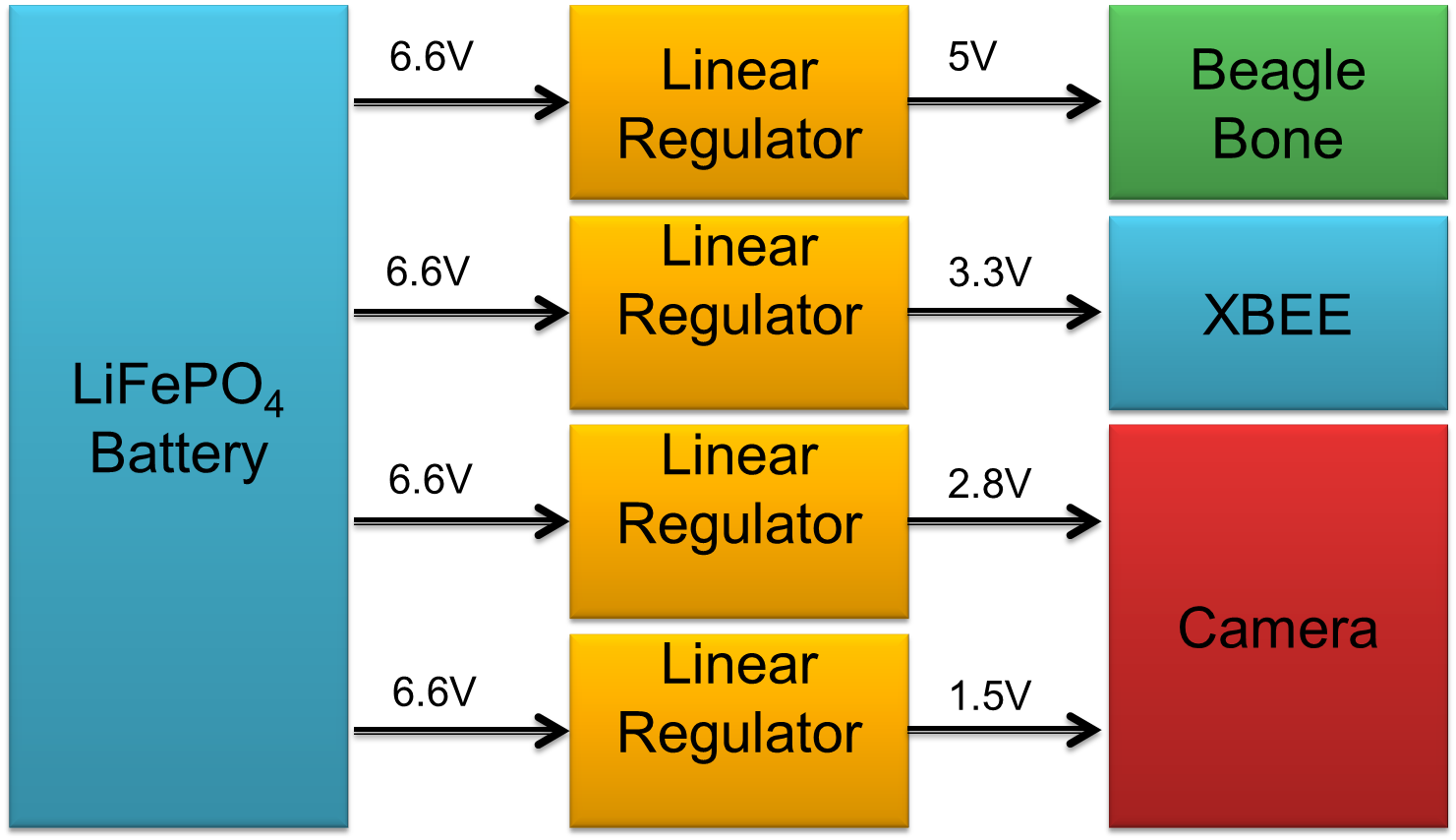
|  |  |
| --- | --- |
| **Module** | Eye tracking system |
| **Inputs** | Visual image of the eye |
| **Outputs** | Cursor movement |
| **Functionality** | Eye images will be taken by the camera. The image data will be processed by the processing module to determine the user’s gaze. The user’s gaze will 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**

|  |  |
| --- | --- |
| **Module** | Camera board |
| **Inputs** | Visual data |
| **Outputs** | 8-bit data bus with image information in 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. |

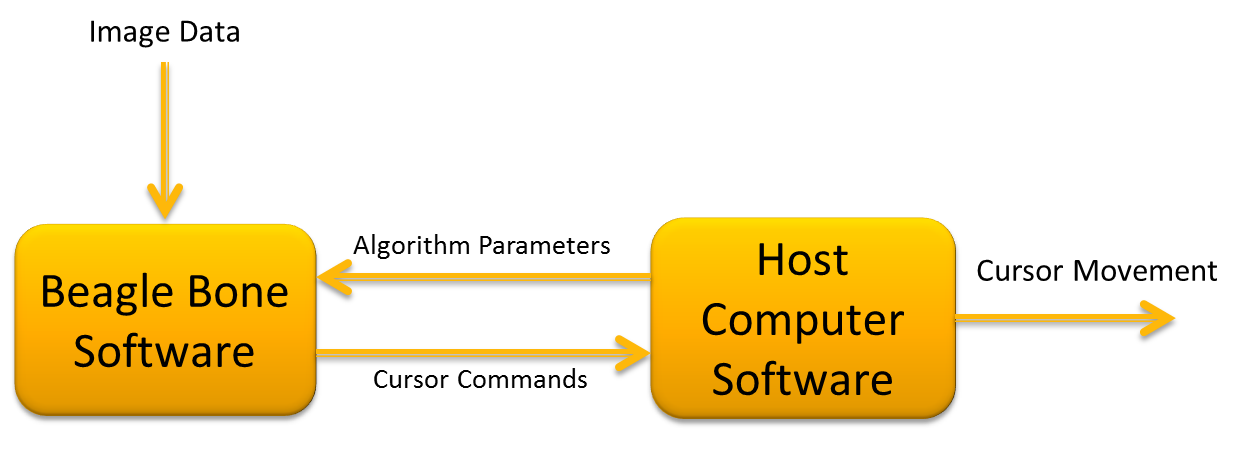
|  |  |
| --- | --- |
| **Module** | Daughter Board |
| **Inputs** | 8-bit parallel image data  VD, HD, DCLK  Serial data from MSP430 board |
| **Outputs** | Buffered camera data.  Serial data to MSP430 board  VD, HD |
| **Functionality** | This board will read data from the camera board, and buffer it. The buffer utilized DCLK to read the data in. VD and HD are sent along to the Beagle Bone to be used in image reading algorithms. Serial data will be both sent and received using an XBee. |
| **Test Plan** | Test the power output, this board is involved in powering the camera so the first step is to test the power rails going to the camera. They should be 1.5V and 2.8V. Manually probing these with a multimeter is 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. |

|  |  |
| --- | --- |
| **Module** | MSP430 Board |
| **Inputs** | Serial data from daughterboard/Beagle Bone  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 the host computer can receive data packets and trying to read it on the host computer. To test XBee transmission, the data on the Beagle Bone. 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. |

****

|  |  |
| --- | --- |
| **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**





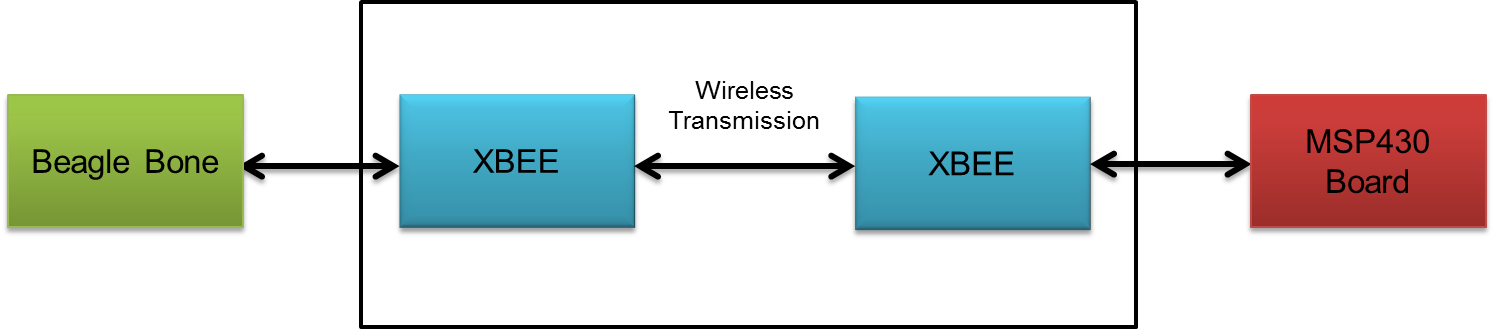
|  |  |
| --- | --- |
| **Module** | **Beagle Bone Software** |
| **Inputs** | 640x480 RGB image data |
| **Outputs** | Command indicating the direction of the user’s gaze |
| **Functionality** | Processes the image to obtain the centroid of the pupil. It then compares the pupil centroid to the reference 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** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

**Level 1: Firmware**

|  |  |
| --- | --- |
| **Module** | Beagle Bone |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

|  |  |
| --- | --- |
| **Module** | MSP430 FTDI |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

****

|  |  |
| --- | --- |
| **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. |

**Level 2: Primary Hardware Components**

|  |  |
| --- | --- |
| **Module** | Camera |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

|  |  |
| --- | --- |
| **Module** | Buffer |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

|  |  |
| --- | --- |
| **Module** | Oscillator driver? |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

|  |  |
| --- | --- |
| **Module** |  |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

|  |  |
| --- | --- |
| **Module** |  |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

**Level 2: Firmware Modules**

|  |  |
| --- | --- |
| **Module** | Camera |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

|  |  |
| --- | --- |
| **Module** | Serial |
| **Inputs** |  |
| **Outputs** |  |
| **Functionality** |  |
| **Test Plan** |  |

****

|  |  |
| --- | --- |
| **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** | Check that cursor command packets are sent to the MSP430 XBEE and that the MSP430 recognizes the cursor command packets. |

****

|  |  |
| --- | --- |
| **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** | Check that algorithm calibration parameter packets are sent to the Beagle Bone XBEE and that the Beagle Bone recognizes the algorithm calibration parameter packets. |

**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** | Verify that 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 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 placed into the queue. |

**Level 2: Beagle Bone DSP**

**Primary Software Data Structures:**

|  |  |  |
| --- | --- | --- |
| **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. |
| **CR** | Two dimensional array of points | Each row contains the coordinates of a connected region. Only regions that meet the area requirement are stored. |
| **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:** | CR, CRSize, CRCount |
| **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:** | CR, CRSize, 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:** | CR, CRSize, Index indicating chosen region |
| **Outputs:** | Updated CR 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:** | CR, CRSize |
| **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 Functions**

|  |  |
| --- | --- |
| **Module** | **Camera Drivers Project** |
| **Inputs** | None No inputs? |
| **Outputs** | None No output? |
| **Functionality** | I2C driver for Stellaris LM3S6965. Sets up basic I2C protocol for camera, including camera initialization over I2C. Uses a startup sequence on Stellaris to ensure protection of Stellaris in the event of a software error within project. |
| **Test Plan** | Use combination of CCSv4 debugger and logic analyzer to test. |

|  |  |
| --- | --- |
| **Module** | **cameraI2C.c** |
| **Inputs** | None No inputs? |
| **Outputs** | None No outputs? |
| **Functionality** | I2C driver for Stellaris LM3S6965. Sets up basic I2C protocol for camera, including camera initialization over I2C. |
| **Test Plan** | Step through the assembly generated by this file using the debugger and inspect memory in order to ensure the correct registers are written to. Use the logic analyzer I2C interpreter to confirm functionality of individual functions within module. Hook analyzer up to SDA0, SCL0, and GND on Port B of the Stellaris. You should be able to step through the C file, not assembly  Once functionality of individual functions has been confirmed also probe camera output of DCLK along with input I2C signals from Stellaris. |

|  |  |
| --- | --- |
| **Module** | **I2CMCUInit() What does this do?** |
| **Inputs** | User adjusted parameters Vague |
| **Outputs** | Visualization of the image processing Vague |
| **Functionality** | Stellaris board I2C initialization. Clock speed is set directly from crystal. 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 Stellaris. |

|  |  |
| --- | --- |
| **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.  \*\*\*\*\*\*\*\*\* WRITE MODE \*\*\*\*\*\*\*\*  \* Start  \* Slave Address (MSB 7bit)  \* 0  \* <Acknowledge>  \* Sub Address (8 bit)  \* <Acknowledge>  \* Data 1 (8 bit)  \* <Acknowledge>  \* Data n (8 bit)  \* <Acknowledge>  \* Stop  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Exlain in words, not this chart |
| **Test Plan** | Use logic analyzer to confirm proper data has been sent out on I2C lines and the addresses are correct. Hook analyzer up to SDA0, SCL0, and GND on Port B of the Stellaris.  Write a function to read camera registers to check that registers were properly set.  Can only be called after I2CMCUInit()has been successfully run. |

|  |  |
| --- | --- |
| **Module** | **I2CInitCamera()** |
| **Inputs** | None There should be something here -> even it’s hard coded stuff |
| **Outputs** | Byte of data has been written to camera. |
| **Functionality** | Camera has been initialized and configured for operation. 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. Hook analyzer up to SDA0, SCL0, and GND on Port B of the Stellaris. Why are we talking about the stellaris so much? Shouldn’ we be using Beagle. |