Hardware

Figure 1: Hardware system overview

Figure 1 shows the high level hardware overview of our eye tracking system. There are three main boards across 4 PCBs. For our main processing board we chose to use the Beagle Bone [1].

Beagle Bone

The Beagle Bone features an ARM Cortex A8 processor [2]. This processor is a good choice for our project for several reasons. Since we are doing digital signal processing we wanted a processor with plenty of speed. The A8 on the Beagle Bone runs at 500MHz when powered off USB and 720MHz when powered off of a 5V source. It also has floating point hardware acceleration. These two features make it ideal for our uses. In addition to speed, the ARM architecture is very flexible, and the many GPIO pins can be reconfigured to different settings. This allowed us to easily set up the UART and I2C interfaces we needed, along with general IO for communication with the FIFO buffer. The Beagle Bone also had Ethernet and USB interfaces built into the board.

Daughter Board

To interface hardware such as the camera and the XBEE to the Beagle Bone, we created a custom PCB we named the Daughter Board. This board is in fact two boards, one for the camera, and another for the XBEE and FIFO buffer.

Camera Board

Figure 2: Camera Board Schematic

The most significant component on the Camera Board is the camera. In addition to the camera, there is a level shifter for the data coming from the camera, some linear regulators for powering the camera, and 3 n-channel MOSFETs. The MOSFETs are there to do level shifting for the I2C.

TMC8230MD

The camera we chose was the TMC8230MD from Toshiba. This camera is a CMOS camera. The CMOS features lower power consumption over CCD and is also cheaper. This camera was idea for our project for its low cost and data output options.

There are many different output options from the camera; however the maximum resolution is 640 by 480. This would correspond to VGA resolution. The data is output on 8 data lines, and can be in two main formats, YCrCb or RGB. For our DSP algorithm, we wanted the output in RGB, since we would only be using the red channel. The YCrCb is output as an 8 bit format, while the RGB is in a 5:6:5 format. For the RGB the specification is 16 bit; 5 bits of red, 6 of gree, and 5 of blue for each pixel.

The frame output speed of the camera is mainly controlled by the speed of the clock the camera is supplied with. To achieve a 30 fps output, a 24.54MHz oscillator is used. It is also possible to set the fps using the I2C control; however for our setup it can only be set to 30 fps or 15 fps.

There are several control signals that the camera can be configured to output. The Horizontal Sync (HD), Vertical Sync (VD), and Data Clock (DCLK) signals can be used to identify the end of lines, frames, and the data output.

The HD signal shows the end of a horizontal line of pixels, while the VD signal signifies the end of a frame. Both of these signals are active low. The DCLK signal is clocked for each set of 8 bits the camera is outputting. For our configuration, the DCLK is clocked twice for each pixel. This leads to approximately an 18MHz clock signal.

Figure 3: TMC8320MD signal diagram

To control and configure the camera, there is an I2C interface. The interface must run at least 100KHz clock and at most 400KHz clock. The controls are used to set the image format, frame speed, VD HD and DCLK settings, and can be used for some image correction settings.

A pinout for the TMC8230MD camera is shown in Figure 3.

Figure 4: TMC8230MD pinout

This camera requires two voltage sources. One source is a 1.5V source, and is used to power the digital circuits and the A/D converter. The second source is 2.8V and is used for the photo diode and input output signals. Since the IO for the camera is 2.8V, we needed level shifting to convert to the 3.3V logic that the Beagle Bone is expecting.

Level Shifter – SN74LVCH16T245

To convert the data signals and synchronization signals coming from the camera we used a SN74LVCH16T245 bus transceiver. This chip comes from TI and meets all the requirements for this interface.

Figure 5: SN74LVCH16T245 diagram

The most important requirements are the number of signals that we can shift, and the speed at which we can shift them. Since the DCLK is going to be running at as much as 20MHz, we required a transition time from one port to the other of at most 1/20MHz. The transition time for this chip from A to B for 3.3V is a maximum of 7.5 ns which is well within our requirements. Once we had the chip set up correctly, we tested the signals coming out for integrity, and to verify that it met the speed requirements. Figure 6 shows an oscilloscope capture of the DCLK line after level shifting. It is important to note that the voltage is correct, and the speed is maintained.

Figure 6: DCLK capture out of level shifter

I2C Level Shifting

I2C is a bi-directional interface, so the level shifter we used for the data and synchronization signals will not work to level shift this interface as the direction is specified with the 1DIR and 2DIR signals. To level shift I2C we utilized two n-channel MOSFETs. This interface was fond in an application note [3]. Figure 7 shows the diagram for this interface.

Figure 7: I2C level shifting diagram

The principle behind this interface is that when the lower voltage commands a zero, the gate to source voltage, tgs, is met and the transistor turns on. This pulls the voltage low. When the higher voltage side commands a zero the body diode on the MOSFET conducts, pulling the lines low. For this interface, pull-up resistors are required on both sides of the MOSFET. When selecting MOSFETs for this interface, it is important to choose components that will function for the speed requirement and voltage requirement that the system has. For us, we needed a low tgs so that a 3.3V difference would easily meet the threshold of the MOSFET, and we needed a MOSFET that could handle reasonably high speed switching frequencies. This meant that the input capacitance had to be small to maintain signal integrity, and the switching times, commonly ton and toff were low. Other factors that are important are the drain current and the on resistance.

Interface/XBEE Board

The interface and XBEE board is designed to connect the FIFO signals to the exposed GPIO pins for the Beagle Bone, and set up the FIFO circuitry for the camera data.

Figure 8: Interface/XBEE board schematic

Even though the GPIO pins on the Beagle Bone are highly configurable, there are some considerations when selecting where to connect signals. Only some of the pins can be selected to be UART ports, which we needed for the XBEE.

XBEE

Figure 9: XBEE

Interfacing to the XBEE in a configuration that is purely for UART communication between boards is straightforward. There are only 4 lines that need to be hooked up.

Figure 10: XBEE pinout

First and foremost, the power lines need to be connected. There are only two other connections that need to be made for a minimalistic interface, the rx and tx lines. Additionally, we chose to connect an L.E.D. to the sleep pin to give visual feedback when the XBEE was not in sleep mode.

When laying out the XBEE, it is desireable to add a cutout in ground and or power planes below the footprint. As the XBEE is a wireless transceiver communicating at 2.4 GHz, it can impart a lot of high frequency noise onto any planes that are below it, possibly corrupting the ground and disrupting other chips on the board.

If we were to do another revision of this portion of the board, we would add L.E.D.s to the rx and tx lines to visually show when data was being received and transmitted from the module.

FIFO

The FIFO we selected for the project was a IDT72V251. This FIFO was not large enough to store an entire image, but could store a few lines. Ideally, a FIFO that could store an entire image would be used; however the chips become quite expensive as the size increases.

Figure 11: FIFO diagram

The one we chose had a size of 4096x9. This allowed us to store just over 6 lines of an image at 640x480, RGB 5:6:5 settings.

The FIFO has several features that made it a good selection for our use. First, it had enough speed to handle the signals we required, with some margin. At read and write times of just 10ns it was well within the bounds of 20MHz. It also had 4flags that it output detailing the state of the buffer. Two static flags, the full flag (FF) and the empty flag (EF) showed when the FIFO was full or empty respectively. In addition to the static flags, there are two programmable flags to show when the FIFO is almost empty (PAE) and almost full (PAF). The intention is to use the PAF flag to signal when to read the data from the FIFO to avoid loss of data.

To program the PAF and PAE flags a condition must be met on startup, show in Figure 12.

Figure 12: FIFO initialization timing diagram

Again, if we were able to revise the board, we would change the way the FIFO was setup. To program the PAF and PAE flags to different values (they default to full-7 and empty -7 each), the read and write ports are used. By default, we hooked up the write ports, but that only allows us to set the PAF flag. We would have added additional signals going to the inputs of the FIFO to allow us to program the PAE.

Software

There are two main ways to run the Beagle Bone. The first is by using Texas Instruments StarterWare [2], and the second is to run embedded Linux on the board.

StarterWare

Initially we had chosen to run the board without an operating system. Many of the features that an OS could provide were not necessary for our project, and it added extra overhead that we wanted to avoid. StarterWare is a set of libraries that can be downloaded to get started programming several of TI’s processors quickly. The package gives an abstraction layer to many of the hardware components on such complicated System on Chip (SOC) microprocessors.

SD

Uart

I2c

Gpio

Camera Drivers

FIFO Drivers

Linux

UART Communication

[1]