Pivoting Object Tracking System

[CSEE 4840 Project Design - March 2009]

Damian Ancukiewicz
Applied Physics and Applied Mathematics Department
da2260@columbia.edu

Arjun Roy
Computer Science Department
ar2409@columbia.edu

Jinglin Shen
Electrical Engineering Department
js3478@columbia.edu

Baolin Shao
Computer Science Department
bs2530@columbia.edu

Abstract

In this paper, we describe our system in detail. We first give an overview of our system, including the system's high level architecture and a description of, in general, how our system will work. We then thoroughly describe each component by providing every component's block and timing diagram. In addition, we analyze timing constrains in sections 4 and 7 to show that our system can display and recognize objects in real time.

1. Introduction

This project aims to implement an object recognition system, where a camera tracks the position of an object. Additionally, the camera is mounted on an iRobot Create two-wheeled robot and is able to command the robot to rotate left and right in order to keep the object it is tracking in its field of view.

2. System Overview

In Figure 1, we give an overview of all components in our system, which are all hooked up to the Avalon bus.

In our system, a video camera is connected to the Altera DE2 and sends NTSC analog signals to the board. An Analog Devices ADV7181 converts analog signals to digital signals in YUV format. The converter has an I^2C interface, which allows for the output format and other parameters to be configured. A block and timing diagram of our I^2C controller is given in Figure 2. The ADV7181 Decoder Controller converts each pixel from YUV to 16-bit RGB and sends it to a buffer in the SDRAM using the DMA (direct memory access) controller. The Nios II then performs processing on the buffer in SDRAM in order to find the object in question and to mark up the image. Additionally, the Nios II uses a serial interface to command the iRobot Create to turn in the appropriate direction if necessary. Subsequently, the buffer in SDRAM is sent to the VGA controller, which in turn transfers the buffer to SRAM. This buffer in SRAM is used to display the marked-up image on a VGA screen.

3. DMA

With the use of DMA, transfers from the ADV7181 decoder and SDRAM and between the SDRAM and VGA controller can be done without passing through the Nios II. The decoder to SDRAM copy needs to start at the beginning of the video frame and it needs to copy each byte sent by the decoder. The SDRAM to VGA copy needs to be done during the fairly narrow interval when no video is being drawn on the VGA screen.

The DMA controller allows for flow control. With flow control, the slave device (which in this case is the ADV7181 or VGA controller) can assert a signal that it is ready for reading and/or writing. Thus, any data transfer can occur at the speed and timing that the slave requires. A slave peripheral can drive the `readyfordata` signal high to indicate that it is ready to begin a transfer, and it can drive the `datavailable` signal high to indicate that it is ready to be written to. With this, we can let the decoder alert the DMA controller when a new video frame has begun so that the SDRAM can be written to, and we can let the VGA controller alert the when the visible part of the VGA frame has ended so that the SDRAM can be read.

In addition to having Avalon master ports to facilitate the data transfer, the DMA controller also has an Avalon slave port which is used by the Nios II to initiate the transfer and set the memory locations to be transferred and the length of the transferred data.

4. VGA Design

The VGA controller is an Avalon peripheral that displays a framebuffer located on the SRAM on a VGA screen. The controller is directly connected to the SRAM, to the Avalon bus, and to the VGA interface. It accepts the main 50 MHz

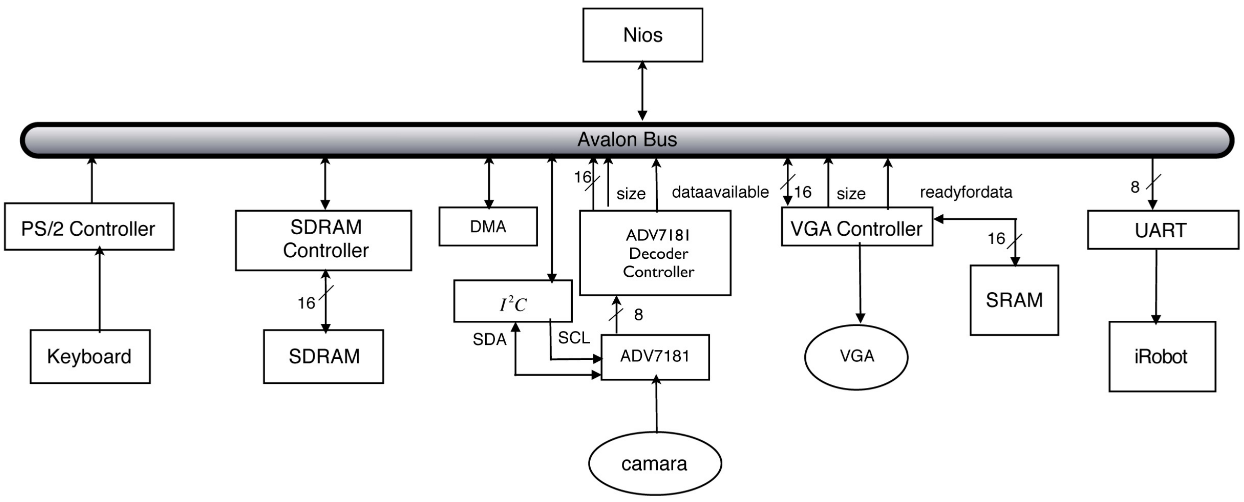

Figure 1. System High Level Design (Block Diagram)

system clock for its logic, which it then scales to 25 MHz for use as the VGA pixel clock. We plan on driving the interface at a resolution of 640×480 at 60 frames per second. Whenever the VGA controller needs to draw a frame, it accesses the SRAM to get the color of each pixel. Whenever the Nios II finishes processing a frame of video located on the SDRAM, it initiates a copy from the SDRAM to the VGA controller over the Avalon bus; the VGA controller directly relays the data on to the SRAM. This copy is facilitated by the DMA controller as mentioned in the previous section.

Use of the SRAM presents several limitations on the possible resolution of the framebuffer. Since each pixel will be stored in 16 bits, a 640×480 buffer would take up 614 kB, which is more than the 512 kB size of the SRAM on the Altera DE2. Thus, the size of the buffer will need to be 320×240 or smaller, with each pixel in the buffer mapping to four physical pixels on the VGA screen.

Additionally, because the SRAM cannot be written to and read from concurrently, the write to SRAM needs to occur during the period in which the VGA controller is not drawing any active pixels. At a resolution of 640×480, the VGA interface allows for 45 horizontal lines during which the vertical sync, front porch and back porch occur and no lines are drawn. Additionally, within each line, there are 160 pixel clock cycles in which the horizontal sync, front porch and back porch occur and no pixels are drawn. With a 320×240 buffer, this does not present enough time to do the transfer during the vertical sync/porch interval. The vertical resolution needs to be truncated in order to allow for more transfer time. A resolution of 320×230 should be sufficiently small. A total number of $(45+480-460) \times 800 = 52000$ VGA pixel clocks is available for the transfer. Since the system clock is twice as fast as the VGA clock, there are 104000 system clocks available. Since the SRAM interface is 16 bits wide, the framebuffer can be transferred at a rate of one pixel per system cycle. Thus, the transfer will require only $320 \times 230 = 73600$ system clocks, which leaves a comfortable margin of error.

In order to coordinate timing and ensure that the transfer occurs only when the VGA controller has finished reading pixels from the framebuffer, flow control will be used on the Avalon interface to the VGA controller. Once the VGA controller finishes drawing, the controller will assert the `readyfordata` signal to the Avalon bus, which will alert the DMA controller to begin transferring at the appropriate time.

5. ADV7181 Decoder Design

In Figure 3 and Figure 4, we describe the design and timing diagram of the ADV7181 Decoder Controller respectively. The controller takes the inputs `data[7:0]`, `TD_HS`, `TD_VS` and `Field` from the ADV7181 video decoder. Since NTSC is an interlaced format, the ADV7181 outputs one field of alternating lines of a frame, followed by another field of alternating lines. Since we need to decrease the resolution to ensure there is enough time for frame transmission between SDRAM and SRAM, we only use one field of the input video data. The data is output in single bytes on a 27 MHz clock, with two bytes representing each pixel. The bytes sent are in YUV color space, with luma and alternating

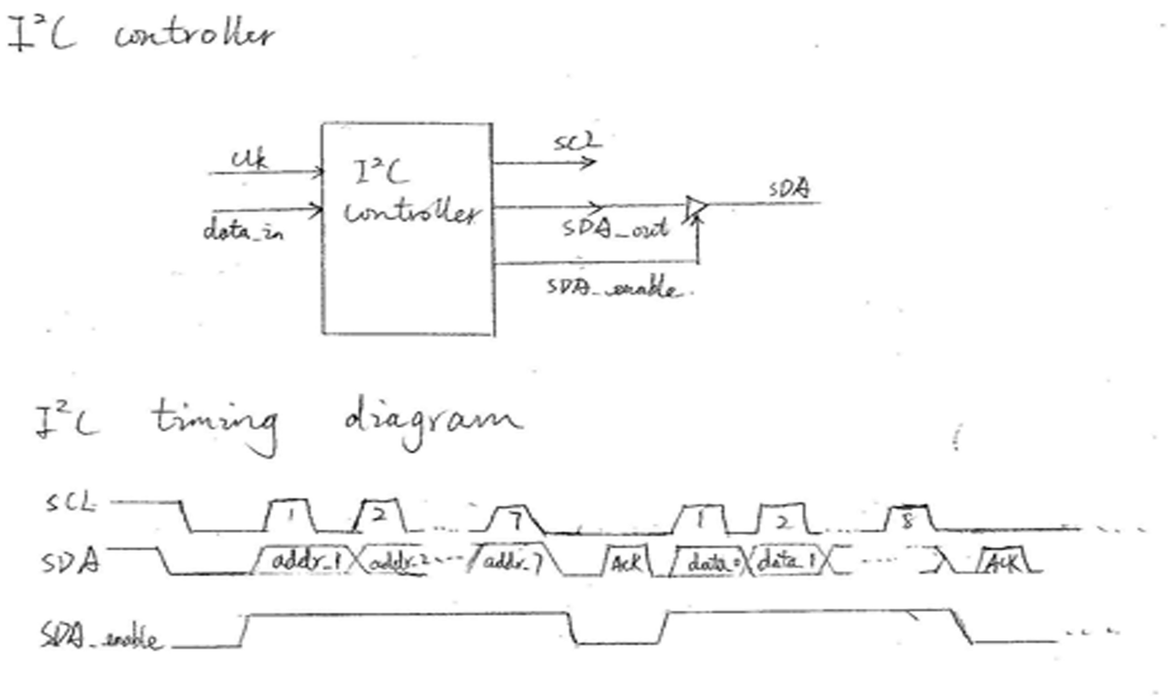

Figure 2. I^2C Block and Timing Diagram

chroma components. Thus, one pixel is Y/Pr, while the next is Y/Pb.

There are 720 active pixels in each line and 242 active lines in each field. Three signals from the ADV7181 are used to align with the data coming from each frame. Before reading the data for a frame, decoder controller waits for the `VS` signal to be asserted in order to align itself with the beginning of the video frame. The `Field` signal allows us to know which of the two interlaced fields is being read. Additionally, the `TD_HS` signal marks the beginning of each line of video. The ADV7181's I^{2}C interface allows for the timing of these signals to be offset as desired.

The data then goes through the 4:2:2 to 4:4:4 module and the YUV to RGB module. The RGB data coming out is then decimated in order to provide a 320x230 image: the first and last 40 pixels in each line are ignored, and then only every other pixel is sent. Similarly, the first and last six lines are ignored. The decimated data is then store in a FIFO buffer and waits to be transferred to the SDRAM. The FIFO buffer asserts the `dataavailable` signal to indicate whether it is empty or not to the DMA. Since each component we mentioned above is a combinatorial logic block which does not have strict timing constraints for the data transmission, and there is available source code to refer to, we do not include all the detailed design of each component here.

6. iRobot Create and Keyboard

The platform that we will use as the camera mount is the iRobot Create, a simple robot that is capable of moving in a 2-dimensional plane using differential drive. We shall only use a subset of the robot's capabilities. Specifically, we will use the ability of the robot to rotate in place to easily enable our system to pivot the camera about the vertical axis, allowing to track objects.

Commanding the robot is done using a standard RS-232 serial interface. The robot provides a set of operation codes to give basic commands, such as rotating in a given direction at a provided speed until it has rotated a certain specified angle. Since a serial controller peripheral is already readily available for the Nios II processor, we do not need to design it ourselves and use it essentially as a black box device. The opcodes themselves are simply byte values, transmitted in a certain order defined by an interface published by the robot manufacturers.

Two cases exist in which the robot moves. We can manually, through the use of an attached PS/2 keyboard, rotate the robot to a given position to manually orient the camera. To use the PS/2 keyboard, we will reuse the peripheral given to us in Lab 2 as a black box.

The second case in which the robot moves is when our image processing software, in order to track an object, in-

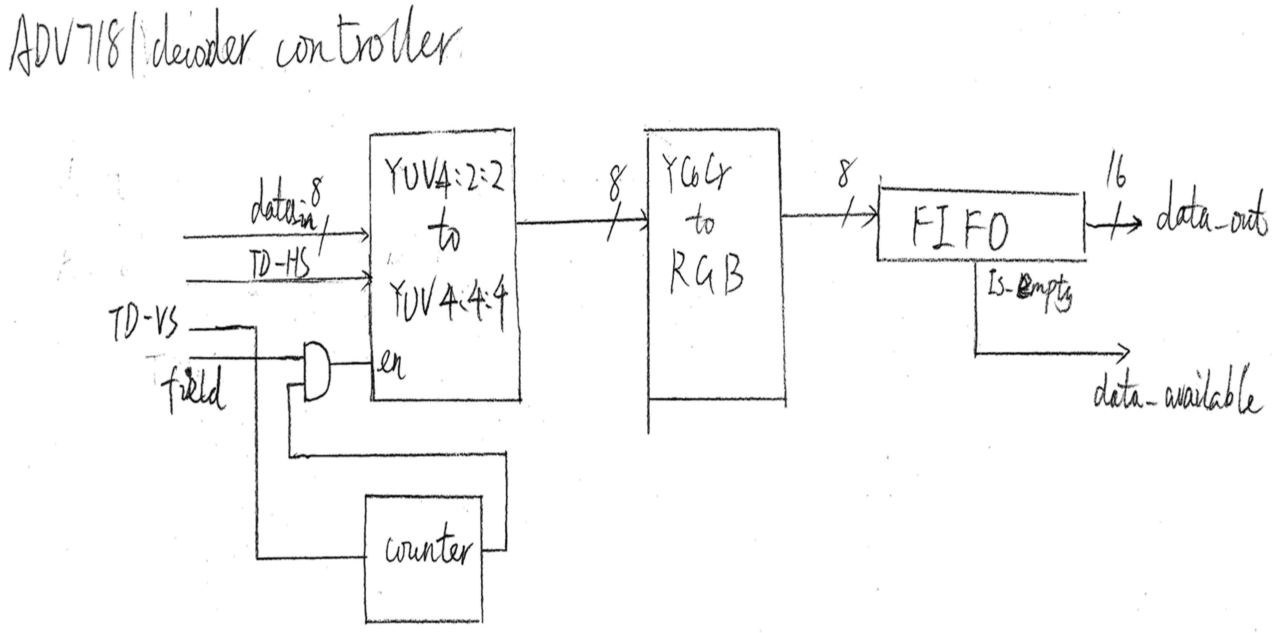

Figure 3. ADV7181 Decoder Controller Block Diagram

structs the robot to turn an angle that will bring the target to the center of the screen.

7. Nios II Software

So far, we have a system that takes in analog video input from a camera, converting it to a format suitable for display on a VGA controller while being able to control the camera mount. In addition to the base hardware, we plan on adding a software layer controlling the state of the system based on environmental inputs and user commands.

Specifically, the user will command an initial orientation for the camera mount by pivoting the robot. The system will then be tasked with tracking an object satisfying a certain criterion. Currently, we aim to track an object by its color; a possible algorithm would be to find the centroid for all pixels close to a given color, and count the number of pixels that match this criterion. If the number of pixels is above a threshold, we have found an object centered at the centroid calculated earlier, and we can command our robot to rotate until the object is in the center of the camera's field of view.

The robot would continue to autonomously track the object until interrupted by the user, who could specify searching for an object of another color or to stop searching altogether. We have decided not to decide all the features and algorithms used in our software at present. However, since the software layer is flexible, added features can be added as development progresses depending on time and desired performance characteristics.

All software would run on the general purpose Nios II processor, communicating with the SDRAM memory through the Avalon bus. Software would be responsible for image processing, responding to user input and sending commands to the robot platform.

Additionally, the software will command the DMA controller to initiate copies from the decoder controller to SDRAM and from SDRAM to the VGA controller. The software does not need to take precise hardware timing into account, as the VGA and decoder controllers will assert the `dataavailable` and `readyfordata` signals as needed. Nevertheless, based on our hardware design, there are a two main timing restrictions on the software used. In order to pull a frame from the ADV7181, the system needs to wait for the start of a frame of video before beginning the transfer. Since each NTSC field takes 1/60 sec, an entire frame takes 1/30 sec. Additionally, in order for the buffer in SDRAM be copied to the video controller, the system needs to wait for the interval in which no pixels are drawn, which only occurs with a 1/60 sec period. If the system is to have a frame rate of 15 fps, it can decode one frame and then process and send it to SRAM before the next frame is finished sending to the ADV7181, so that it can then pull the frame after that. In this scheme, 1/15 sec − 1/60 sec (for VGA) −

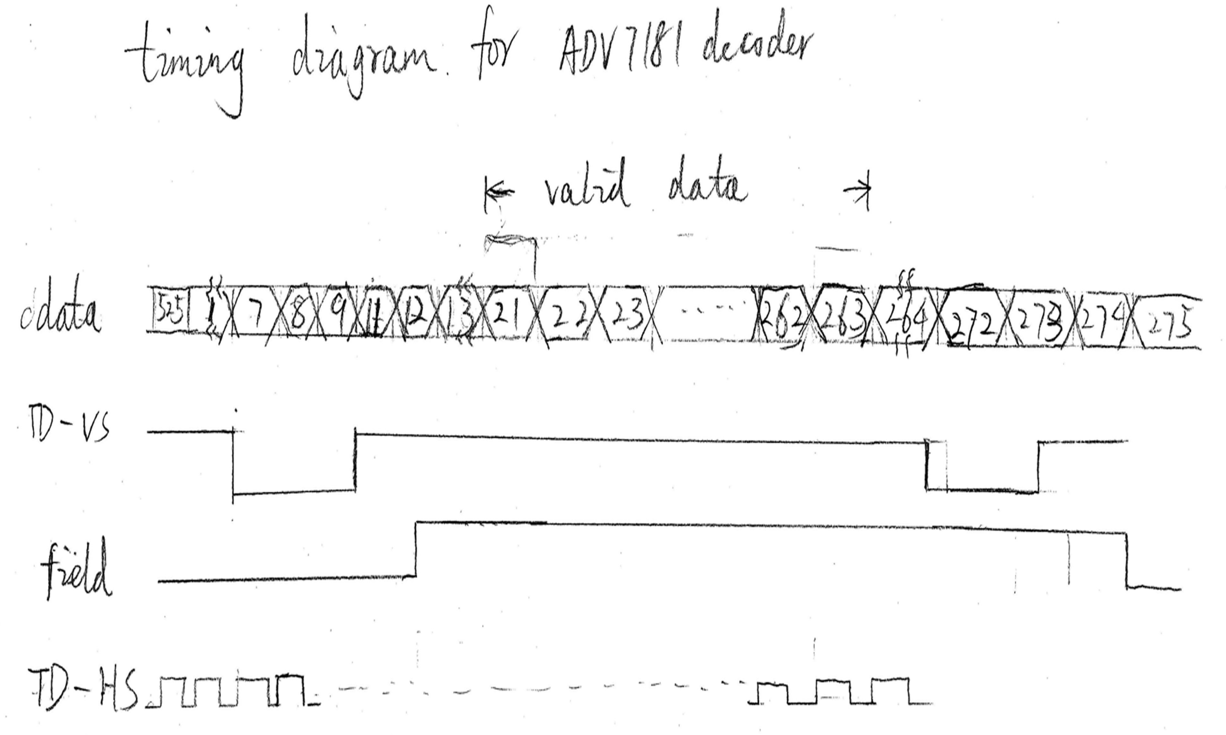

Figure 4. ADV7181 Decoder Controller Timing Diagram

1/60 sec (for acquiring one field from the camera) = 1/30 sec is avialable for processing. At a resolution of 320 × 230 and a system clock of 50 MHz, this allows for 22 clock cycles to be spent on each pixel.

8. Milestones

Milestone 1(March 30)

The first milestone will be to have the necessary components to get data from the video camera to SDRAM and to send this data to the VGA controller, not necessarily in real time.

Milestone 2 (April 13)

By this milestone, we plan on implementing real-time video and on allowing a user to move the robot left and right with a keyboard.

Milestone 3 (April 29)

In this milestone, we will implement the computer vision algorithm and tune it in order to obtain a satisfactory frame rate. Additionally, we will allow the software to steer the robot as needed to follow an object.