

Vision Tracking Device

EMBEDDED SYSTEMS III - BARNEKOW

BOESIGER, GRANT A.

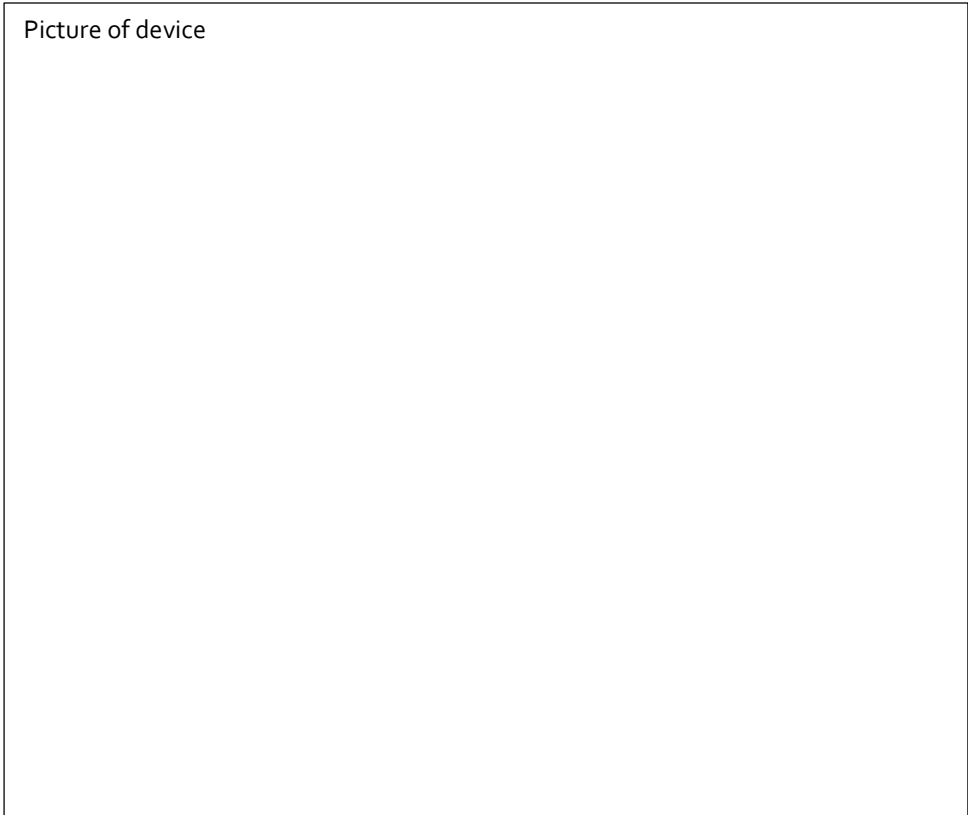
Table of Contents

➤ Introduction	2
➤ Milestones	3
• Simple Memory Read/Write	3
• Servo Motor Position	4
• Camera Read/Write	5
• Camera Image Capture	6
➤ Appendix	8
•	

Introduction

The end goal of the vision tracking device project is to have a camera mounted on two servo motors capable of tracking a black square on a white plane. This will be accomplished through an algorithm that finds the black square in the camera's pixel data and sends a signal to the servo motors so that the camera will adjust itself to be centered on the black square. Ideally, the camera will be able to track the square over relatively quick speeds. However, due to limitations in processing speed, this will not be a likely outcome. The project consists of multiple milestones over the course of two-month period. Each milestone is set to tackle a specific aspect of the project as a whole, eventually leading up to the finished product.

Picture of device



Milestones

Simple Memory Read/Write

REQUIREMENTS The first of the milestones involved creating the framework for inputting commands from the console because many of the other milestones would require this functionality. On top of this, two commands were added that allowed the user to read and write from a memory address. To read from a memory address, the user must input RD followed by two valid hexadecimal memory addresses. The program will then output all data values from the first memory address to the second. If a second address is not specified, the program will only output the data at the first address. The write operation has similar syntax, where the user enters WR followed by a hexadecimal address and a data value to store at that address.

IMPLEMENTATION The fundamental design of the program revolves around a switch statement dependent on the first word entered by the user. The values entered afterwards will then be handled accordingly. The program will infinitely loop through user commands until termination. User input is retrieved and parsed through `fgets()` and `sscanf()` commands. For each read and write command, the addresses input by the user are made into pointers so that the data can be written and read to/from the address.

TESTING Testing of the milestone involved a brute force run through of as many different iterations of a user command as could be thought of to ensure that not only would the commands operate correctly, but that invalid inputs were handled accordingly.

Servo Motor Position

REQUIREMENTS The second milestone required the functionality of the servo motors through PWM inputs. The two servos must be able to move based on inputs from the user. The user could choose to enter either PAN or TILT followed by a number within a certain range to move the servo motor to a specific position. The range was specified such that the motors could not attempt to turn past their physical restrictions and risk straining the motor. This milestone is a small step towards a vital part of the device, as the program will eventually need to be able to adjust the motor positions based on the camera feed.

DESIGN For this milestone, a new header and c file were created specifically for interfacing with the PWM. Both files were very short, as changing PWM duty cycle is as simple as writing a value to either of the two PWM registers. The two registers corresponded to the two servo motors. The values input by the user corresponded to a percent duty cycle output by the PWM. The main program, once again, is a switch statement of the user command. Before writing to the PWM, the value is forced between the hardcoded ranges such that a value that is too high will be changed to the highest allowed value. The same was done for the low end of the range.

TESTING Initial testing of the servo motors involved finding the values where the motor would be at its physical limit. Those values were then added to the code to act as a boundary range as mentioned previously. Each possible position of both motors were tested to verify correct responses.

Camera Read/Write

REQUIREMENTS The third milestone of the project is our first experience in I²C interfacing. Two new commands were to be added to read and write to the cameras sub-addresses. In order to make the code more legible, a new header and c file were added specifically for interacting with the camera registers. Therefore, nearly all of the code for these two commands were outside of main. The user could enter either CamRead followed by the hexadecimal register to be read from, or CamWrite with the register and data to be written.

DESIGN I²C requires a fairly specific algorithm of setting and reading bits within the main registers of the camera. It involves starting up a transmission with directions to read or write from a register, then waiting for that transmission to complete, and then finally verifying a successful transmission in order to move and read or write to a sub-address and repeat the loop. In this design, polling was used to wait for the transmission to complete, even though interrupts are available to use. This choice was made due to the fact that nothing else needs to be done while waiting for the transmission.

TESTING To test the CamRead command, two registers were read from where the value written to them was static and known. The command printed the correct value and, therefore, worked properly. The CamWrite command was a bit more involved. A certain register within the camera allowed for the manual change of the PCLK frequency (PCLK will be discussed in a later milestone). Conveniently, the expansion board being used in this device ported the PCLK signal straight to a pin which could be hooked up to an oscilloscope. Using the equation found in the specifications of the register, a frequency value was chosen to be written to the register and that same value was observed on the oscilloscope.

Camera Image Capture

REQUIREMENTS This milestone required a single command to start a feed of data from the camera to the VGA port of our FPGAs. Once again, the majority of code was placed in the Camera.c file so as not to clutter the main program. The camera pixel dimensions and VGA memory dimensions differed greatly, so certain adjustments needed to be made in order to correctly fit the camera data to the VGA memory and, subsequently, to a monitor.

DESIGN There are quite a few design decisions to point out before proceeding with the actual code. First off is the PCLK frequency. The PCLK is the pixel clock responsible for signaling when a new pixel is available for sampling. The default PCLK frequency is actually too fast for the program to keep up with due to the multiple instructions that need to be executed between each sample. Therefore, before anything can be done, the frequency must be changed to something more manageable. This was done using the commands from the previous milestone. Another major problem in mapping pixel data to the VGA port is the size difference of the two. The camera's pixel array is 356 X 292. However, the VGA port only allows for an array of 80 X 60 pixels. Two things must be done to counteract this. The first of which is simply changing a setting in one of the camera registers that cuts the camera pixel array down to 176 X 144. This is still more than four times the size of the VGA port. Therefore, we need to omit most of the pixels that are received. The best way to do this is to ignore the outermost pixels so that you just have the central 160 X 120 pixels. Then every other pixel is skipped in this array. This leaves a low resolution image but still gets enough of the data to output a reasonable image.

In order to implement the transmission of pixel data to the VGA port, there are three signals that must be watched to know when data is ready to be read. PCLK has already been explained. The other two are VSYNC and HREF. A VSYNC pulse signals that a new frame of data is ready. This signal is polled at the beginning of a capture request to indicate when to start reading pixel data again. HREF will be high whenever a row of pixels is being read and go low when waiting for the next row to get ready. HREF is polled at the beginning to count how many rows have been skipped, and then again after each row of pixel data is read to know when to start reading data again. PCLK is polled in the most nested loop of the function. A rising edge of the PCLK signals

the prime time to read pixel data from the pixel port and write it to the corresponding VGA pixel. It is then also polled to signify that a pixel has been skipped.

TESTING Testing of the camera feed was made extremely easy with access to a monitor with a VGA port. Simply hooking up the monitor and looking at the resulting image streaming from the camera. There were some visible syncing issues evident in the feed in the form of shifting rows moving up the screen periodically. However, it is good enough for the purposes of recognizing a black square on a white plane.

Appendix

Lab2.c

```
#include <stdint.h>
#include <system.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <io.h>
#include "Pwm.h"
#include "Camera.h"

int main(void){
    pwmMotor1(7); //Initiate motors to centered position
    pwmMotor2(7);
    camInit(); //Initialize camera settings
    char input[30];
    char cmd[30]; //Variables for user input
    int val1;
    int val2;
    while(1){
        val1 = 0;
        val2 = 0;
        fgets(input, 30, stdin);
        sscanf(input, "%s %x %x", cmd, &val1, &val2); //Correct input: cmd val1 val2
        uint8_t *adr1 = val1;
        uint8_t *adr2 = val2;

        /* RD and WR to/from memory */
        if(strcmp(cmd, "RD")==0){ //User input RD command
            if(adr1 == 0){
                printf("Invalid Command: Please enter a valid address\n"); //User failed to enter a valid address
            }else{
                if(adr2 == 0){ //Read from single address
                    printf("0x%x: 0x%x\n", adr1, *adr1);
                }else{ //Else read from block of addresses
                    if(adr2>adr1){
                        int lines = ((adr2-adr1)/16); //16 bytes per line of output
                        while(lines>0){
                            printf("0x%x: %x %x %x %x %x %x %x %x %x %x %x %x %x %x\n", adr1, *(adr1+0), *(adr1+1), *(adr1+2),
                                *(adr1+3), *(adr1+4), *(adr1+5), *(adr1+6), *(adr1+7), *(adr1+8), *(adr1+9), *(adr1+10),
                                *(adr1+11), *(adr1+12), *(adr1+13), *(adr1+14), *(adr1+15));
                            adr1+=16;
                            lines--;
                        }
                    }else{
                        printf("End address must be greater than start address\n"); //If addresses were put in wrong order
                    }
                }
            }
        }else if(strcmp(cmd, "WR")==0){ //User input WR command
            if(adr1 == 0){ //Invalid write address
                printf("Invalid Command: Please enter a valid address\n");
            }else{
                *adr1 = adr2; //Assign data of address the specified value
            }
        }

        /*Pan and Tilt motor*/
        }else if(strcmp(cmd, "PAN")==0){ //User input PAN command
            if(val1>12){ //Ensure that value falls within range
                val1 = 12;
            }else if(val1<2){
                val1=2;
            }
            pwmMotor1(val1); //Set PWM output
        }else if(strcmp(cmd, "TILT")==0){ //User input WR command
            if(val1>9){ //Ensure that value falls within range
                val1 = 9;
            }else if(val1<2){
                val1=2;
            }
            pwmMotor2(val1); //Set PWM output

        /* RD/WR to camera */
        }else if(strcmp(cmd, "CamRead")==0){
            uint8_t data = camRead(val1);
            printf("The value at Camera sub-address 0x%x is 0x%x", val1, data);
        }else if(strcmp(cmd, "CamWrite")==0){
            camWrite(val1, val2);

        /* Camera image to VGA */
        }else if(strcmp(cmd, "Capture")==0){
            while(1){
                capture();
            }
        }else{
            printf("Invalid Command: Unknown command\n"); //Otherwise, command is invalid
        }
    }
}
```

PWM.h

```
#ifndef PWM_h
#define PWM_h

#include <stdint.h>
#include <system.h>
#include <unistd.h>
#define PWM (uint16_t volatile *)0x00802028
#define PWMB (uint16_t volatile *)0x0080202a

void pwmMotor1(int duty);
void pwmMotor2(int duty);

#endif
```

PWM.c

```
#include <system.h>
#include <stdio.h>
#include <io.h>
#include "PWM.h"

float dutyConvert = 200;          //Convert Duty % to number

void pwmMotor1(int duty){
    printf("Pan duty cycle: %d Percent \n", duty);
    duty *= dutyConvert;          //Convert duty from percent to number
    IOWR_16DIRECT(PWM,0, duty);
}

void pwmMotor2(int duty){
    printf("Tilt duty cycle: %d Percent \n", duty);
    duty *= dutyConvert;          //Convert duty from percent to number
    IOWR_16DIRECT(PWMB,0, duty);
}
```

Camera.h

```
#ifndef CAMERA_H_
#define CAMERA_H_
#include <stdint.h>
#include <system.h>
#include <unistd.h>

#define VGA (uint8_t volatile *)0x00800000
#define CAMCTRL (uint8_t volatile *)0x00802000
#define PXLPORT (uint16_t volatile *)0x00802010
#define CAMERA (uint16_t volatile *)0x00802020

void camInit();
uint8_t camRead(int reg);
void camWrite(int reg, int data);
void capture();
#endif
```

Camera.c

```
#include <system.h>
#include <stdio.h>
#include <io.h>
#include "Camera.h"

#define SLA_W (uint8_t volatile) 0xC0
#define SLA_R (uint8_t volatile) 0xC1

void camInit(){
    IOWR_8DIRECT(CAMERA, 2, 0x80);           //set EN bit
    printf("pixel rate\n");
    camWrite(0x11, 0x08);                     //Reduce pixel rate to 985kHz
    printf("resolution\n");
    camWrite(0x14, 0x20);                     //Reduce resolution to 176x144
    printf("PCLK generate\n");
    camWrite(0x39, 0x40);                     //PCLK generated only when HREF=1
}

uint8_t camRead(int reg){
    IOWR_8DIRECT(CAMERA, 3, SLA_W);           //write 0xC0 to Transmit Register
    IOWR_8DIRECT(CAMERA, 4, 0x90);           //set STA and WR bit
    while((IORD_8DIRECT(CAMERA, 4)>>1)&0x01){
        //Wait for TIP to clear
    }
    if((IORD_8DIRECT(CAMERA,4)>>7)&0x01){       //read RXACK (Should be 0)
        printf("Device not found\n");         //RXACK = 1 signals that device can not be found to read from
    }else{
        IOWR_8DIRECT(CAMERA, 3, reg);         //write sub-address to Transmit Register
        IOWR_8DIRECT(CAMERA, 4, 0x50);         //set WR and STO bit
        while((IORD_8DIRECT(CAMERA, 4)>>1)&0x01){
            //Wait for TIP to clear
        }
        if((IORD_8DIRECT(CAMERA,4)>>7)&0x01){   //read RXACK (Should be 0)
            printf("Sub address not found\n"); //RXACK = 1 signals that sub address can not be found
        }else{
            IOWR_8DIRECT(CAMERA, 3, SLA_R);   //write 0xC1 to Transmit Register
            IOWR_8DIRECT(CAMERA, 4, 0x90);     //set STA and WR bit
            while((IORD_8DIRECT(CAMERA, 4)>>1)&0x01){
                //Wait for TIP to clear
            }
            if((IORD_8DIRECT(CAMERA,4)>>7)&0x01){ //read RXACK (Should be 0)
                printf("Sub address not able to be written to\n"); //RXACK = 1 signals that sub address can not receive data
            }else{
                IOWR_8DIRECT(CAMERA, 4, 0x64); //set RD, STO and ACK bit
                while((IORD_8DIRECT(CAMERA, 4)>>1)&0x01){
                    //Wait for TIP to clear
                }
                return IORD_8DIRECT(CAMERA, 3);
            }
        }
    }
}

void camWrite(int reg, int data){
    IOWR_8DIRECT(CAMERA, 3, SLA_W);           //write 0xC0 to Transmit Register
    IOWR_8DIRECT(CAMERA, 4, 0x90);           //set STA and WR bit
    while((IORD_8DIRECT(CAMERA, 4)>>1)&0x01){
        //Wait for TIP to clear
    }
    if((IORD_8DIRECT(CAMERA,4)>>7)&0x01){       //read RXACK (Should be 0)
        printf("Device not found\n");         //RXACK = 1 signals that device can not be found to read from
    }else{
        IOWR_8DIRECT(CAMERA, 3, reg);         //write sub-address to Transmit Register
        IOWR_8DIRECT(CAMERA, 4, 0x10);         //set WR bit
        while((IORD_8DIRECT(CAMERA, 4)>>1)&0x01){
            //Wait for TIP to clear
        }
        if((IORD_8DIRECT(CAMERA,4)>>7)&0x01){   //read RXACK (Should be 0)
            printf("Sub address not found\n"); //RXACK = 1 signals that sub address can not be found
        }else{
            IOWR_8DIRECT(CAMERA, 3, data);     //write data to Transmit Register
            IOWR_8DIRECT(CAMERA, 4, 0x50);     //set WR and STO bit
            while((IORD_8DIRECT(CAMERA, 4)>>1)&0x01){
                //Wait for TIP to clear
            }
            if((IORD_8DIRECT(CAMERA,4)>>7)&0x01){ //read RXACK (Should be 0)
                printf("Sub address not able to be written to\n"); //RXACK = 1 signals that sub address can not receive data
            }
        }
    }
}
```

```

void capture(){
    int i,j;
    uint8_t row,col;

    /* VSYNC PULSE*/
    while(!((IORD_8DIRECT(CAMCTRL, 0)>>2)&0x01)){
        //Wait for VSYNC to pulse high
    }
    while(!((IORD_8DIRECT(CAMCTRL, 0)>>2)&0x01)){
        //Wait for VSYNC to go low
    }

    /*SKIP FIRST 8 ROWS*/
    for(i=0; i<8; i++){
        while(!((IORD_8DIRECT(CAMCTRL, 0)>>1)&0x01)){
            //Wait for HREF to pulse high
        }
        while((IORD_8DIRECT(CAMCTRL, 0)>>1)&0x01){
            //Wait for HREF to go low
        }
    }
    for(row=0; row<60; row++){
        while(!((IORD_8DIRECT(CAMCTRL, 0)>>1)&0x01)){
            //Wait for HREF to pulse high
        }
        for(j=0; j<12; j++){
            while(!((IORD_8DIRECT(CAMCTRL, 0))&0x01)){
                //Wait for PCLK to pulse high
            }
            while((IORD_8DIRECT(CAMCTRL, 0))&0x01){
                //Wait for PCLK to go low
            }
        }
        for(col=0; col<80; col++){
            while(!((IORD_8DIRECT(CAMCTRL, 0))&0x01)){
                //Wait for PCLK to pulse high
            }
            uint16_t coords = (row<<7)+col;
            uint8_t data = IORD_8DIRECT(PXLPORT, 0);
            IOWR_8DIRECT((VGA+coords),0,data); //Write PixelPort data to correct VGA coordinates
            while((IORD_8DIRECT(CAMCTRL, 0))&0x01){
                //Wait for PCLK to go low
            }
            while(!((IORD_8DIRECT(CAMCTRL, 0))&0x01)){
                //Skip a pixel
                //Wait for PCLK to pulse high
            }
            while((IORD_8DIRECT(CAMCTRL, 0))&0x01){
                //Wait for PCLK to go low
            }
        }
        while((IORD_8DIRECT(CAMCTRL, 0)>>1)&0x01){
            //Wait for HREF to pulse low
        }
    }
}

```