Embedded Systems III Final Report

Reid Rumack
Prof. Barnekow
CE 3910
May 18, 2017

Table of Contents

The System

This system uses the DE0 board, with the MSOE 2800 expansion board, using the NiosII fast processor. The peripheral devices used include an I^2C Master device, VGA interface, PWM module with two channels, C3088 color camera module with the Omni Vision OV6620 image sensor, and two PWM controlled servo motors.

Introduction

The task given for this quarter was to design and build A turret mounted camera. The first requirement was to be able to track and follow a black dot on a white background. Keeping the object in the center of its vision using two servo motors. In addition to this the image from the camera must be viewable on a VGA monitor while the System is in operation. Third, the user should be able to interact through the systems command line before it enters tracking mode. Finally, the system must be able to operate in normal room light conditions and run off of a 12V DC supply.

Milestone I

For the first mile's stone, the task was creating a command system. This system allowed the user to interact with the system via the system command line. The system was required to provide at least two commands, Read ("RD") and Write ("WR"). These commands were to read and write from the system memory.

The Read command allows the user to enter a command in the following form:

```
RD [Address] [Count]
```

Where address is the memory address in hexadecimal. Count is the number of bytes the user wants to have returned and printed to the console. The values are return in with a max of 16 bytes per row. If a value for `Count` is not given, then only one byte is returned to the console.

The Write command must be given in the following form:

```
WR [Address] [Value]
```

The parameter `Value` is a one-byte hexadecimal number. When the command is entered, the system writes `Value` to the given hexadecimal address.

The design for the command system relied on using the system command line. The system first prints out a list of valid commands and their format.

```
Please enter a command.
RD address
RD address count(max of 16)
WR address data
```

Figure 1

After the command is entered, `scanf()` is used to extract and store the information into an appropriate struct: `readcmd`, `writecmd`. The `structs`' are the used by the `memWriter` or `memReader` where the command is executed.

The fuction of the system was varifyed by testing each command separately. First, the write command was tested by writing values to the LEDs on the DE0. First witing 0xFF(the first eight leds are on) figure 3 then 0xAA (every other led first eight are on) figure 5.

```
Please enter a command.
RD address
RD address count(max of 16)
WR address data

WR 80802030 FF

Please enter a command.
RD address
RD address count(max of 16)
WR address data

RD 80802030

          +0
0x80802030: FF
```

Figure 2: Shows the command line inputs and outputs. WR and RD tests.

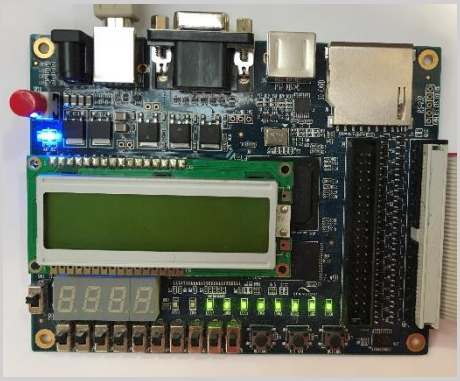


Figure 3

```
Please enter a command.
RD address
RD address count(max of 16)
WR address data

WR 80802030 AA

Please enter a command.
RD address
RD address count(max of 16)
WR address data

RD 80802030

          +0
0x80802030: AA
```

Figure 4: Shows the command line inputs and outputs for 0xAA.

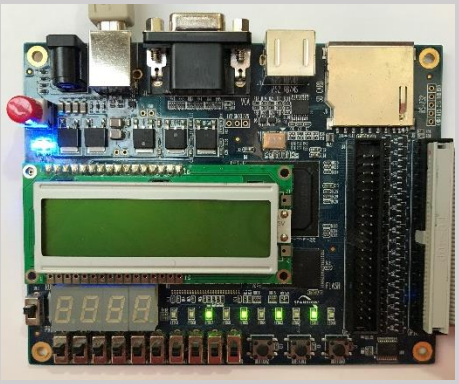


Figure 2

```
Please enter a command.
RD address
RD address count(max of 16)
WR address data

RD 80802030 36

           +0  +1  +2  +3  +4  +5  +6  +7  +8  +9  +10 +11 +12 +13 +14 +15
0x80802030: AA  00  00  00  02  07  4F  00  00  00  00  00  00  00  4F  00

           +0  +1  +2  +3  +4  +5  +6  +7  +8  +9  +10 +11 +12 +13 +14 +15
0x80802070: 00  00  00  00  00  00  00  00  00  00  00  00  FF  00  00  00

           +0  +1  +2  +3
0x808020B0: 00  00  00  00
```

Figure 3: This show the that system will only output 16 bytes per line.

Milestone II

In this milestone an API for the Servo motors was developed. This API is required to provide the functions to control and read the position of the servo motors: pan and tilt. Also, an initialization function is required. This function centers both of the servos.

The functions `pan(int col)` and `tilt(int row)` servo control function must take in a column value or row value that related to the number of a location in the cameras image sensor. These function must check and make sure that the given values are valid and correlates to a location in the image sensors array. Col must be between 0 and 175, and row must be between 0 and 145.

In order to test the operation tilt and pan commands where added to the command line interface.

`T [Row]`
`P [Column]`

In order to convert the given row or column value to a valid value for the PWM, values where calculated to adjust each of the `col` and `row` values. The PWM value for the tilt servo is called `OCRA1B`, and the PWM value for the pan servo is called `OCRA1A` The general form of the equation used to adjust the values is:

$$y = mx + b$$
$$OCRA1A = N_{pan}(col) + offset$$
$$OCRA1B = N_{tilt}(row) + offset$$
$$offset = n_{90°ccw} - n_{90°cw}$$
$$1000 = 2000 - 1000$$

$$N_{pan} = \frac{1000}{175} = 5.714$$
$$N_{tilt} = \frac{1000}{145} = 6.897$$

The final equations are:

$$OCRA1A = 5.714(col) + 1000$$
$$OCRA1B = 6.897(row) + 100$$

Milestone III

For the third milestone, the objective was to establish communication with the camera. The particular camera we used was the OV6620, which uses an I^2C like interface called SCCB. Functional SCCB is the same as I^2C which allow us to use the systems built in I^2C Master device. As with the other milestones commands for the systems command line were added.

```
camRdReg [Register Number]
camWrReg [Register Number] [Value]
```

Where `Register Number` is the address of one the cameras internal registers. When the read command is used both the register number and its value are both printed to the system console. The algorithms to for talking to the camera are as follows:

$$Cpu\ clock = 50MHz \qquad bitrate = 100\ Kbit/s$$

$$Prescale\ Value = \frac{Cpu\ clock}{5 \times bitrate} - 1 = 99 = 0x0063$$

- **`I2C_init`**`()`: must be called, but only needs to be called once.
  1. Write the Prescale values to Prescale Lo and Prescale Hi
     - `I2C_LO = 0x63`
     - `I2C_HI = 0x00`
  2. Set the enable bit in the control register
     - `I2C_CTRL = 0x80`
- **`I2C_action`**`(command)`: executes the given command.
  1. Sets the `I2C_CMD`(command register) value to `command`
  2. Polls `TIP` (transfer in progress) bit unit it is '0'
  3. Checks and returns `RxACK` bit in the `I2C_STAT` (status register)
- **`I2C_write`**`(slaveAddr, regNumber, value)`: writes the given value to given sub-address of the give slave address.
  1. Write slave address (0xC0 is the camera write address) to `I2C_TRANSMIT`
  2. Call `I2C_action(START_I2C|WRITE_I2C)`; (0x80|0x10)
  3. Write `regNumber` to `I2C_TRANSMIT`
  4. Call `I2C_action(WRITE_I2C)`; (0x80)

5. Write `value` to `I2C_TRANSMIT`
6. Call `I2C_action(WRITE_I2C|STOP_I2C)`; (0x80|0x40)
7. Return value from `I2C_action()`

- **`I2C_read`**`(slaveAddr, regNumber)`: Read the value from the given location
  1. Write camera write address (0xC0) to `I2C_TRANSMIT`
  2. Call `I2C_action(START_I2C|WRITE_I2C)`; (0x80|0x10)
  3. Write `regNumber` to `I2C_TRANSMIT`
  4. Call `I2C_action(WRITE_I2C|STOP_I2C)`; (0x80|0x40)
  5. Write slave address (0xC1 is the camera read address) to `I2C_TRANSMIT`
  6. Call `I2C_action(START_I2C|WRITE_I2C)`; (0x80|0x10)
  7. Call `I2C_action(READ_I2C|STOP_I2C|ACK_I2C)`; (0x20|0x40|0x08)
  8. Read `I2C_RECEIVE` and return the value

In order to test **`I2C_read()`** a read command was issued to read internal register 0x1C and 0x1D. These registers hole the manufactures Id numbers, they are hardwired and the same across all the OV6620 cameras. The values held at 0x1C and 0x1D are 0x7F and 0xA2 respectively.

To test **`I2C_write()`** a logic analyzer was used to monitor the value the pixel clock (PCLK). By writing to `regNumber` 0x11, the rate of PCLK can be altered by a factor of X.

Milestone IV

Now that commutation to the camera has been established an image can now be obtained. But the image is useless until it can be displayed. The system that is in uses has a black and white (B&W) VGA interface that provides 80 columns and 60 rows of pixels. All of the VGA pixels have their own address in the on chip memory. The address for the VGA pixels range from 0x00800000- 0x00801FFF because the VGA port is an IO device, 0x80800000- 0x80801FFF must be used so that the systems cache is bypassed. The pixel address format uses the first 7-bits as for the column address and the second 6-bits as the row address.

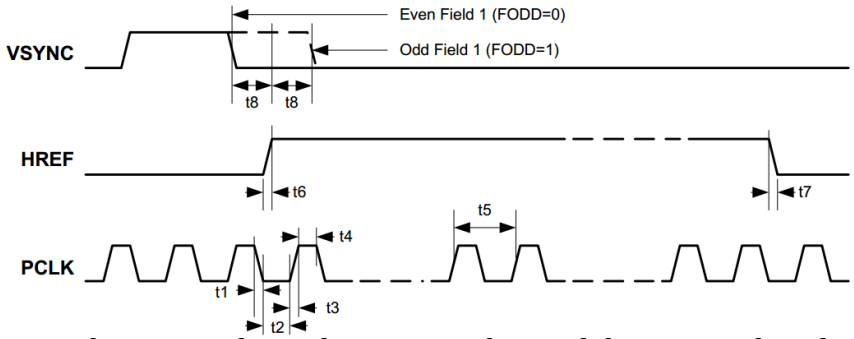
VGA address brake down (0x80800000 is the base address)									
					Row location			Column location	
1000	0000	1000	0000	000	R	RRRR	R	CCC	CCCC

This VGA device can be accessed the same way as the LEDs. The color is set using an address pointer to the desired pixel. The color of a pixel must have a value between 0xF0 (white) and 0x10 (black).

The OV6620 is a color camera with a default resolution of 356 x 292 in CIF mode. This would make displaying an image using the systems VGA interface difficult. But looking at the OV6620's datasheet the camera can be set into B&W and QCIF mode, limiting the resolution to 176 x 144. To select QCIF mode bit 5 in the camera's COMC register needs to be set to a '1'. B&W mode is activated by setting bit 6 in the camera's COMH register to a '1'. Then if an 8 column buffer on both sides and 12 rows on top and bottom are applied. The image is now 160x120. Finally, if just the even columns and rows are taken the image is now 80x60.

Now, in order to obtain a picture from the camera PCLK must be slowed to give the system time to run. This was done by writing an 0x8 to the cameras CLKRC register. This reduces PCLK's rate by a factor of 8. Now that the camera is initialized the possessor can now start getting images. Because the camera only outputs one pixel at a time, it provides three signals timing signals:

1. VSYNC: Pulsed "high" one per before the start of a frame; "low" during Frame.
2. HREF: Pulled "high" for the duration of a row. Then pulled "low" for a blanking interval between rows.
3. PCLK: Puled once per pixel; a falling edge indicates there is a new pixel, a rising edge means there is valid pixel data on the `PIXEL_PORT` and is ready to be sampled.



In order to conserve the systems limited memory. The pixel data is sent directly to the VGA port rather than storing it in a data structure. This can be done because each pixel's data is latched, allowing the system to read a pixel's value directly from its address pointer.

Milestone V

After getting all of the individual parts of the system work. It is time to develop the detection algorithm. For this, a kernel based approach was taken. For horizontal edge detection the algorithm uses an origin pixel that iterates through the VGA pixels. The algorithm looks at the values of the pixels around the origin.

1. If `pixL` (pixel column before the origin) is white, and `pixR` (pixel column after the origin) is black, then a left edge is detected. If `pixL` is black and `pixR` is white, then a right edge is detected.

    a. **left edge detected**
        i. the column of the origin is added to a running sum called `leftColSum`, and `leftColNum` (number of values added) is incremented.
        ii. If the pixel on top of `pixR` (`pixTop`) is white and the one below(`pixBott`) is black, then a top edge is detected. And the row value is added to `topRowSum` and `topRowNum` is incremented.
        iii. If `pixTop` is black and `pixBott` is white, then a bottom edge is detected. Then the row value is added to `botRowSum` and `botRowNum` is incremented.

    b. **right edge detected**
        i. the column of the origin is added to `rightColSum`, and `rightColNum` is incremented.
        ii. If the pixel on top of `pixL` (`pixTop`) is white and the one below (`pixBott`) is black, then a top edge is detected. Then the row value is added to `topRowSum` and `topRowNum` is incremented.
        iii. If `pixTop` is black and `pixBott` is white, then a bottom edge is detected. Then the row value is added to `botRowSum` and `botRowNum` is incremented.

2. All averages are calculated
    a. $leftColAvr = \frac{leftColSum}{leftColNum}$ $rightColAvr = \frac{rightColSum}{rightColNum}$
    b. $topRowAvr = \frac{topRowSum}{topRowNum}$ $botRowAvr = \frac{botRowSum}{botRowNum}$

3. Objects' center point is calculated using the midpoint formula.
    a. $XCenterPoint = \frac{leftColAvr + rightColAvr}{2.0}$
    b. $YCenterPoint = \frac{topRowAvr + botRowAvr}{2.0}$

4. Checks to see if the center points are valid.
    a. $XCenterPoint$ must be between 1 and 78, if not the objects' center X value is set to 39 (VGA's center column value).

b. $YCenterPoint$ must be between 1 and 58, if not the objects' center Y value is set to 29 (VGA's center row value)

5. The calculated coordinates the objects' center are returned.

Milestone VI

Now that the object has been detected and it center point calculated tracking can begin. By comparing the center point of the pixel array to that of the object, the distance and direction from each other is known. In order to make The physical tracking easier, the servos now take use A Cartesian like plane, where the Domain is -39 to 39, and The range is -29 to 29. Also, System keeps track of the last PWM value. This allows the system to worry only about the difference between the center of the array and the object.

In order to get the system working well the N_{pan} and N_{tilt} values needed to be tweaked: N_{pan} is now 6, and N_{tilt} is now 8. The new equations for OCRA1A and OCRA1B are now:

$$OCRA1A = currentPwmValue_{pan} - (N_{pan}(col))$$
$$OCRA1B = currentPwmValue_{tilt} - (N_{tilt}(row))$$

$$OCRA1A = currentPwmValue_{pan} - (6(col))$$
$$OCRA1B = currentPwmValue_{tilt} - (8(row))$$

Conclusion

Although, the main functionality of the system works and the system is able to track an object. Not all of the requirements were met most. The user is not able to use the command line to interact with the camera. However, they still are able to read and write to memory and control the servo motors. During the development, this project provided many technical challenges. The hardest challenge was dealing with all of the little bugs that only caused as the system was integrated. An Example of problem was that The print statements used to debug would often cause the system to crash. In the end, this project was both challenging and rewarding

Appendix

PointTracker.c

```
/*****************************************************************************
*@AUTHOR:   Reid Rumack                                                      *
*@FILE:     PointTracker.c                                                   *
*Currnent Milestone : Week 2                                                 *
*@DATE:     March 15, 2017                                                   *
*@PROVIDES: Runable file for the system. contains Main                       *
*****************************************************************************/

/****   INCLUDES  ****/
#include "PointTracker.h"
#include "niosii_uart.h"
#include "CameraCmdParser.h"
#include "niosii_pwm.h"
#include "ServoAPI.h"
#include "CameraAPI.h"
#include "niosii_I2C.h"
#include "msoeIoAdresses.h"
#include "niosii_vga.h"

#include <stdint.h>
#include <stdio.h>
#include <string.h>

/**** PROTO-TYPES ****/
int main(void);
struct Point findXmidPoint();
struct Point calcDiff(struct Point objCenter);
/**** IMPLIMENTED FUNCTIONS  ****/
typedef struct Point{
	int X_cord;
	int Y_cord;

};

int main(void) {
	uint16_t panServoVal,tiltServoVal;
	panServoVal = 1500;
	tiltServoVal = 1500;
	struct Point objCenter;
	objCenter.X_cord = 39;
	objCenter.Y_cord = 29;

	initServo();
	I2C_init();
	cameraInit();
	cameraWrite(0xF, 0x11);
	cameraCmdSystem();

	while(1){
		struct Point difference;
		cameraDisplayVGA();
		objCenter = findXmidPoint();
		difference = calcDiff(objCenter);
		panServoVal = pan( difference.X_cord,panServoVal);
```

```
                tiltServoVal = tilt( difference.Y_cord,tiltServoVal);
        }
        return 0;
}


struct Point findXmidPoint(){
        struct Point dotCenter;
        uint8_t row;

    float leftColSum,rightColSum = 0.0;
    float leftColNum,rightColNum = 0.0;
    float topRowSum, botRowSum = 0.0;
    float topRowNum, botRowNum = 0.0;

       for(row = 58; row > 1 ; --row){
         uint8_t col;
         for(col = 78; col > 1 ; --col){
                     uint8_t pixL, pixR;
                     pixL = niosii_vga_read_pixel(col-1,row);
                     pixR = niosii_vga_read_pixel(col+1,row);
                            if(pixL >= 0xB0 && pixR <= 0x50){
                                   leftColSum += col;
                                   leftColNum ++;
                                   uint8_t pixTop, pixBott;

                                   pixTop = niosii_vga_read_pixel(col+1,row-1);
                                   pixBott = niosii_vga_read_pixel(col+1,row+1);
                                   if(pixTop >= 0xB0 && pixBott <= 0x50){
                                          topRowSum += row;
                                          topRowNum++;
                                   }else if(pixTop  <= 0x50 && pixBott  >= 0xB0){
                                          botRowSum += row;
                                          botRowNum++;
                                   }
                            }else if(pixL  <= 0x50 && pixR  >= 0xB0){
                                   rightColSum += col;
                                   rightColNum ++;
                                   uint8_t pixTop, pixBott;

                                   pixTop = niosii_vga_read_pixel(col-1,row-1);
                                   pixBott = niosii_vga_read_pixel(col-1,row+1);
                                   if(pixTop >= 0xB0 && pixBott <= 0x50){
                                          topRowSum += row;
                                          topRowNum++;
                                   }else if(pixTop  <= 0x50 && pixBott  >= 0xB0){
                                          botRowSum += row;
                                          botRowNum++;
                                   }
                            }
             }
    }
       float leftColAvr, rightColAvr;
       leftColAvr  = leftColSum  / leftColNum;
       rightColAvr = rightColSum / rightColNum;
       int XCenterPoint = (int)(leftColAvr + rightColAvr)/(2.0);
       (XCenterPoint >= 1 && XCenterPoint <= 78)? ((dotCenter.X_cord)=(80 -
XCenterPoint)):((dotCenter.X_cord) = 39);
       float topRowAvr, botRowAvr = 0.0;
       topRowAvr = topRowSum/topRowNum;
       botRowAvr = botRowSum/botRowNum;

       int YCenterPoint = (int)(topRowAvr + botRowAvr)/2.0;
```

```
    (YCenterPoint >= 1 && YCenterPoint <= 58)?
((dotCenter.Y_cord)=(YCenterPoint)):((dotCenter.Y_cord)=29);

      return dotCenter;
}


struct Point calcDiff(struct Point objCenter){
      struct Point diffrence;
      diffrence.X_cord = 39 - objCenter.X_cord;
      diffrence.Y_cord = 29 - objCenter.Y_cord;
    return diffrence;
}

/*****************************************************************************
***
*@AUTHOR:    Reid Rumack
*
*@FILE:      CameraCmdParser.c
*
*Currnent Milestone : Week 2
*
*@DATE:      March 15, 2017
*
*@PROVIDES: fuctions to both read and write to memory using the system
console.*
*****************************************************************************
**/

/****   INCLUDES  ****/
#include "msoeIoAdresses.h"
#include "CameraCmdParser.h"
#include "ServoAPI.h"
#include <stdint.h>
#include <stdio.h>
#include <string.h>

#define MAX_COUNT 16
#define MIN_COUNT 1
#define READ_CMD "RD"
#define WRITE_CMD "WR"
#define PAN_CMD "P"
#define TILT_CMD "T"

#define IS_A_MATCH 0
#define FALSE 0
#define VALID 1
#define INVALID 0



typedef struct readcmd ReadCmd;
typedef struct writecmd WriteCmd;


struct readcmd{
    uint32_t *addr;
    uint32_t  count;
```

```
};

struct writecmd{
    uint32_t *addr;
    uint8_t  data;
};


/**** PROTO-TYPES ****/
void memWriter(uint32_t* address, uint8_t data);
uint8_t* memReader(uint32_t* address, uint32_t count);
uint8_t memRead(uint32_t* address);
void memPrinter(uint32_t* staringAddr, uint32_t numToRead, uint8_t* rdData);
uint8_t cameraCmdSystem(void);
ReadCmd rdPaser(char* command);
WriteCmd wrPaser(char* command);

/**** IMPLIMENTED FUNCTIONS  ****/

void memWriter(uint32_t* address, uint8_t data){
        *address = data;
}

uint8_t* memReader(uint32_t* address, uint32_t count){
    uint8_t* readData;
    uint8_t i;
        for(i= 0; i<count;i++){
            *(readData+i) = *(address+i);
        }
    return readData;
}

uint8_t memRead(uint32_t* address){
    return (uint8_t)*address;
}

void memPrinter(uint32_t* staringAddr, uint32_t numToRead, uint8_t* rdData){;
      uint32_t numbPrinted, toPrint,numbInLine;
      numbPrinted = 0;
      while((numToRead-numbPrinted)>0){
            toPrint=numToRead-numbPrinted;
          uint32_t i;

            if(toPrint>MAX_COUNT){
                  numbInLine = MAX_COUNT;
            }else{
                  numbInLine = toPrint;
            }

            printf("\n          ");

            for(i = 0; i < numbInLine; i++){
                  printf(" +%-2u", i);
            }

            printf("\n0x%08X:",staringAddr+numbPrinted);
```

```
            for(i = 0; i < numbInLine; i++){
                  printf(" %02X ",*(rdData+numbPrinted+i));
            }
            numbPrinted += i;
            printf("\n");
      }
}


uint8_t cameraCmdSystem(){
    uint8_t isValid = INVALID;
    char cmd[32];
    char* op;
    char* cmdPrompt = "\n Please enter a command.\n RD address\n"
         " RD address count(max of 16)\n WR address data\n T row\n P
column\n";
    printf(cmdPrompt);
    if(fgets(cmd, sizeof(cmd), stdin) != NULL){
        char* newLine = strchr(cmd,'\n');
        if(newLine != NULL){
            *newLine = '\n';
        }
    }
    sscanf(cmd,"%s",op);
    if (strcmp(op,READ_CMD) == IS_A_MATCH) {
      ReadCmd rdCmd = rdPaser(cmd);
        uint8_t* memData = memReader(rdCmd.addr,rdCmd.count);
        memPrinter(rdCmd.addr,rdCmd.count, memData);

    } else if (strcmp(op,WRITE_CMD) == IS_A_MATCH) {
      WriteCmd wrCmd = wrPaser(cmd);
        memWriter(wrCmd.addr,wrCmd.data);

    } else if(strcmp(op,PAN_CMD) == IS_A_MATCH) {
      int col=0;
      sscanf(cmd,"%*s %d", &col);
      pan(col,1500);
      *PWM_OCRA1A =col;
    }else if(strcmp(op,TILT_CMD) == IS_A_MATCH) {
      int row=0;
      sscanf(cmd,"%*s %d", &row);
      tilt(row,1500);
      PWM_OCRA1B=row;
    }
    return isValid;
}

ReadCmd rdPaser(char* command){
    ReadCmd toReturn;
    unsigned int count = 0;
    sscanf(command,"%*s %x %u", &toReturn.addr, &count);
    (count == 0)?( toReturn.count = MIN_COUNT):(toReturn.count = count);
    return toReturn;
}

WriteCmd wrPaser(char* command){
    WriteCmd toReturn;
```

```
    sscanf(command,"%*s %x %x", &toReturn.addr, &toReturn.data);
    return toReturn;
}
```

CameraCmdParser.c

```
/*****************************************************************************
*@AUTHOR:   Reid Rumack                                                      *
*@FILE:     CameraCmdParser.c                                               *
*Currnent Milestone : Week 2                                                *
*@DATE:     March 15, 2017                                                  *
*@PROVIDES: fuctions to both read and write to memory using the system console.*
*****************************************************************************/

/****   INCLUDES  ****/
#include "msoeIoAdresses.h"
#include "CameraCmdParser.h"
#include "ServoAPI.h"
#include <stdint.h>
#include <stdio.h>
#include <string.h>

#define MAX_COUNT 16
#define MIN_COUNT 1
#define READ_CMD "RD"
#define WRITE_CMD "WR"
#define PAN_CMD "P"
#define TILT_CMD "T"

#define IS_A_MATCH 0
#define FALSE 0
#define VALID 1
#define INVALID 0



typedef struct readcmd ReadCmd;
typedef struct writecmd WriteCmd;


struct readcmd{
    uint32_t *addr;
    uint32_t  count;
};

struct writecmd{
    uint32_t *addr;
    uint8_t  data;
};


/**** PROTO-TYPES ****/
void memWriter(uint32_t* address, uint8_t data);
uint8_t* memReader(uint32_t* address, uint32_t count);
uint8_t memRead(uint32_t* address);
void memPrinter(uint32_t* staringAddr, uint32_t numToRead, uint8_t* rdData);
uint8_t cameraCmdSystem(void);
ReadCmd rdPaser(char* command);
WriteCmd wrPaser(char* command);

/**** IMPLIMENTED FUNCTIONS  ****/

void memWriter(uint32_t* address, uint8_t data){
        *address = data;
}

uint8_t* memReader(uint32_t* address, uint32_t count){
    uint8_t* readData;
    uint8_t i;
        for(i= 0; i<count;i++){
            *(readData+i) = *(address+i);
```

```
        }
    return readData;
}

uint8_t memRead(uint32_t* address){
    return (uint8_t)*address;
}

void memPrinter(uint32_t* staringAddr, uint32_t numToRead, uint8_t* rdData){;
        uint32_t numbPrinted, toPrint,numbInLine;
        numbPrinted = 0;
        while((numToRead-numbPrinted)>0){
                toPrint=numToRead-numbPrinted;
            uint32_t i;

                if(toPrint>MAX_COUNT){
                        numbInLine = MAX_COUNT;
                }else{
                        numbInLine = toPrint;
                }

                printf("\n          ");

                for(i = 0; i < numbInLine; i++){
                        printf(" +%-2u", i);
                }

                printf("\n0x%08X:",staringAddr+numbPrinted);

                for(i = 0; i < numbInLine; i++){
                        printf(" %02X ",*(rdData+numbPrinted+i));
                }
                numbPrinted += i;
                printf("\n");
        }
}


uint8_t cameraCmdSystem(){
    uint8_t isValid = INVALID;
    char cmd[32];
    char* op;
    char* cmdPrompt = "\n Please enter a command.\n RD address\n"
         " RD address count(max of 16)\n WR address data\n T row\n P column\n";
    printf(cmdPrompt);
    if(fgets(cmd, sizeof(cmd), stdin) != NULL){
        char* newLine = strchr(cmd,'\n');
        if(newLine != NULL){
            *newLine = '\n';
        }
    }
    sscanf(cmd,"%s",op);
    if (strcmp(op,READ_CMD) == IS_A_MATCH) {
        ReadCmd rdCmd = rdPaser(cmd);
        uint8_t* memData = memReader(rdCmd.addr,rdCmd.count);
        memPrinter(rdCmd.addr,rdCmd.count, memData);

    } else if (strcmp(op,WRITE_CMD) == IS_A_MATCH) {
        WriteCmd wrCmd = wrPaser(cmd);
        memWriter(wrCmd.addr,wrCmd.data);

    } else if(strcmp(op,PAN_CMD) == IS_A_MATCH) {
        int col=0;
        sscanf(cmd,"%*s %d", &col);
        pan(col,1500);
        *PWM_OCRA1A =col;
    }else if(strcmp(op,TILT_CMD) == IS_A_MATCH) {
        int row=0;
        sscanf(cmd,"%*s %d", &row);
        tilt(row,1500);
        PWM_OCRA1B=row;
```

```
    }
    return isValid;
}

ReadCmd rdPaser(char* command){
    ReadCmd toReturn;
    unsigned int count = 0;
    sscanf(command,"%*s %x %u", &toReturn.addr, &count);
    (count == 0)?( toReturn.count = MIN_COUNT):(toReturn.count = count);
    return toReturn;
}

WriteCmd wrPaser(char* command){
    WriteCmd toReturn;
    sscanf(command,"%*s %x %x", &toReturn.addr, &toReturn.data);
    return toReturn;
}
```

ServoAPI.c

```
/*
 * ServoAPI.c
 *
 *  Created on: Mar 22, 2017
 *      Author: rumackr
 */
#include "niosii_pwm.h"
#include "msoeIoAdresses.h"

#define OFF_SET 1000
#define pwCenterValue 1500
const int OCR_INC_PAN = 6;
const int OCR_INCA_TILT = 8;
//
//struct cameraPosition{
//      uint16_t panPwmVal = 1500
//};
const int16_t panServoPos = 1500;



void initServo(void); // Set up OCRA and OCRB to move the servos to their center positions

uint16_t pan(int col,uint16_t currentPwmValue);
// Used to provide an absolute x-axis servo position. The number provided
// is to be a zznumber between 0 and 175. This corresponds to the horizontal
// resolution of the camera. These values should cause a full range of travel
// from far left to far right.
uint16_t tilt(int row,uint16_t currentPwmValue);
// Used to provide an absolute y-axis servo position. The number provided
                    // is to be a number between 0 and 144. This corresponds to the vertical
                    // resolution of the camera. These values should cause a full range of travel
                    // from top to bottom

void initServo(void){
        *PWM_OCRA1A = 1500;
        *PWM_OCRA1B = 1500;
} // Set up OCRA and OCRB6 to move the servos to their center positions

uint16_t pan(int col,uint16_t currentPwmValue){

        uint16_t newPwmValue =(uint16_t)currentPwmValue;
        if(col>-39&&col<39){
                uint16_t pwm_value =(uint16_t)(currentPwmValue-(OCR_INC_PAN*col));
                if(pwm_value>=1000&& pwm_value<=2000){
                        *PWM_OCRA1A = pwm_value;
                        newPwmValue = pwm_value;

                }
        }
        return newPwmValue;
```

```
	// Used to provide an absolute x-axis servo position. The number provided
}
// is to be a number between 0 and 175. This corresponds to the horizontal
// resolution of the camera. These values should cause a full range of travel
// from far left to far right.
uint16_t tilt(int row,uint16_t currentPwmValue){
	uint16_t newPwmValue = (uint16_t)currentPwmValue;
	if(row>-29&&row<29){
		uint16_t pwm_value =(uint16_t)(currentPwmValue-(OCR_INCA_TILT*row));
		if(pwm_value>=1000&& pwm_value<=2000){
			*PWM_OCRA1B = pwm_value;
			newPwmValue = pwm_value;
		}
	}
	return newPwmValue;// Used to provide an absolute y-axis servo position. The number
provided
}
		// is to be a number between 0 and 144. This corresponds to the vertical
		// resolution of the camera. These values should cause a full range of travel
		// from top to bottom
```

niosii_I2C.c

```
/*
 *
 *
 *  Created on: Mar 30, 2017
 *      Author: rumackr
 */

#include <stdint.h>
#include "niosii_I2C.h"
#include "msoeIoAdresses.h"

// For Command Register
#define START_I2C 0x80
#define STOP_I2C  0x40
#define WRITE_I2C 0x10
#define READ_I2C  0x20
#define ACK_I2C   0x08
// For Status Register
#define TIP    0x02
#define RxACK  0x80
#define BUSY   0x40
//
#define ACK       0
#define NACK      1
#define IS_BUSY   1
#define NOT_BUSY  0


void I2C_init();
// This function is needed by every I2C transfer operation.
// It sets the command needed by the particular I2C
//  transfer, polls TIP for end of transfer and returns the
// RxACK status (0 for success)
uint8_t I2C_action(uint8_t command);

// This function is used to read the contents of any of
// the camera's registers. It returns the 8-bit contents of the
// specified register.
uint8_t I2C_read(uint8_t slaveAddr,uint8_t regNumber);

// This function takes the value passed in
// and writes it to the specified register. It returns nothing.
uint8_t I2C_write(uint8_t slaveAddr,uint8_t regNumber, uint8_t value);

void I2C_init(){
	*I2C_LO = 0x63;//99
	*I2C_HI = 0x00;
	*I2C_CTRL = 0x80;//Enable
```

```
}

uint8_t I2C_action(uint8_t command){
        uint8_t ackOrNack  = 0;

        *I2C_CMD = command;
        //wait for not busy
        while((*I2C_STAT & TIP) == TIP);
        ackOrNack = (*I2C_STAT & RxACK);
        return ackOrNack;
}
uint8_t I2C_read(uint8_t slaveAddr,uint8_t regNumber){
        uint8_t valueRead = 0;
        *I2C_TRANSMIT = 0xC0;
        I2C_action(START_I2C|WRITE_I2C);
        *I2C_TRANSMIT = regNumber;
        I2C_action(WRITE_I2C|STOP_I2C);
        *I2C_TRANSMIT = slaveAddr;
        I2C_action(START_I2C|WRITE_I2C);
        I2C_action(READ_I2C|STOP_I2C|ACK_I2C);
        valueRead = *I2C_RECEIVE;
        return valueRead;
}
uint8_t I2C_write(uint8_t slaveAddr, uint8_t regNumber, uint8_t value){
        uint8_t writen = 0;
        *I2C_TRANSMIT = slaveAddr;
         I2C_action(START_I2C|WRITE_I2C);
        *I2C_TRANSMIT = regNumber;
         I2C_action(WRITE_I2C);
        *I2C_TRANSMIT = value;
         writen = I2C_action(WRITE_I2C|STOP_I2C);
         return writen;
}
```

niosii_vga.c

```
/*
 * niosii_vga.c
 *
 *  Created on: Apr 6, 2017
 *      Author: rumackr
 */


#include "niosii_vga.h"
#include "msoeIoAdresses.h"
void niosii_vga_write(void);
uint8_t niosii_vga_read_pixel(uint8_t x, uint8_t y);


uint8_t niosii_vga_read_pixel(uint8_t x, uint8_t y){
        uint8_t pixelVal;
        uint32_t pixelAddr;
        pixelAddr = (y << 7);
        pixelAddr += x;
        pixelVal = *(VGA + pixelAddr);
        return pixelVal;
}
```

```
void niosii_vga_write(){
        uint8_t row =  0;
        uint8_t* memPtr;

        memPtr = (uint8_t*)(VGA + (row << 7));
        for (row = 59 ; row > 0; --row){
                uint8_t col = 0;
                for(col = 79; col > 0; --col){
//                      if(col==39&&row==29){
//                              *memPtr = 0xF0;
                        }else
                        if((col>(60-4))&&(col<(60+4))&&(row>(30-4))&&(row<(30+4))){
                                *memPtr = 0x10;

                        }else{
                          *memPtr = 0xF0;

                        }
                        memPtr++;
                }
                memPtr = (uint8_t*)(VGA + (row << 7));
        }
}
```

cameraAPI.c

```
/*
 * cameraAPI.c
 *
 *  Created on: Mar 30, 2017
 *      Author: rumackr
 */
#include <stdint.h>
#include "msoeIoAdresses.h"
#include "niosii_I2C.h"
#include "CameraAPI.h"
#include "niosii_vga.h"

#define CAMERA_WR_ADDR 0xC0
#define CAMERA_RD_ADDR 0xC1
#define CAMERA_mfg_id_numb1 0x1c
#define CAMERA_mfg_id_numb2 0x1d
#define BLK_MAX 0x10
#define WHT_MAX 0xF0
#define COL_BUFF  7
#define ROW_BUFF  11
#define NUMB_VGA_COLS 80
#define NUMB_VGA_ROWS 60
#define NUMB_COLS 175
#define NUMB_ROWS 143
#define COMC_ADDR 0x14
#define COMH_ADDR 0x28
#define VSYNC_MASK 0x4
#define HREF_MASK 0x2
#define PCLK_MASK 0x1



uint8_t cameraDetect();
void cameraInit();
void cameraWrite(uint8_t value, uint8_t subaddress);
uint8_t cameraRead(uint8_t subaddress);
void cameraDisplayVGA();
uint8_t cameraDetect(){
        uint8_t val0, val1, detected = 0;
        val0 = cameraRead(CAMERA_mfg_id_numb1);
```

```
        val1 = cameraRead(CAMERA_mfg_id_numb2);

        ((val0 == 0x7F)&&(val1 == 0xA2 ))?  (detected = 1): (detected = 0);
        (detected==1)?  (printf("\n Camera Detected \n")): (printf("\n Camera NOT Detected \n"));
        return detected;
}

void cameraInit(){
        if (cameraDetect()){
                //initializes the camera into QCIF mode by
                cameraWrite(0x20,COMC_ADDR);
                cameraWrite(0x41,COMH_ADDR);
        }
}

uint8_t cameraRead(uint8_t subaddress){
        uint8_t readValue = 0;
        readValue = I2C_read(CAMERA_RD_ADDR, subaddress);
        return readValue;
}

void cameraWrite(uint8_t value,uint8_t subaddress){
        I2C_write(CAMERA_WR_ADDR, subaddress, value);
}


void cameraDisplayVGA(){
        int rowCamera,rowVga;
        uint8_t * currentPixel;
        rowCamera=0;
        while(!((*CAM_CTRL)&VSYNC_MASK));
        while((*CAM_CTRL)&VSYNC_MASK);
        for(rowVga = NUMB_VGA_ROWS; rowVga >= 0 ; rowVga--){
                while(!((*CAM_CTRL)&HREF_MASK));
         if(rowCamera >= ROW_BUFF && rowCamera <= 132 && rowCamera%2 == 0){
             int colCamera, colVga;
             colCamera = 0;
             for(colVga = NUMB_VGA_COLS; colVga >= 0 ; colVga--){

                while(!((*CAM_CTRL)&PCLK_MASK));

                 if(colCamera >= COL_BUFF && colCamera <= 169 && colCamera%2 == 0){
                      *currentPixel=*PIXEL_PORT;
                      currentPixel++;
                 } else {
                        colVga ++;
                 }
                 while(((*CAM_CTRL)&PCLK_MASK));
                 colCamera++;
             }
                    currentPixel = (uint8_t *)(VGA + (rowVga << 7));
        } else {
             rowVga++;
        }
        while(((*CAM_CTRL)&HREF_MASK));
        rowCamera++;
 }

}
```

msoeIoAdresses.c

```
/*****************************************************************************
 * FILENAME:     msoeIoAdresses.c                                            *
 * DATE:         11 Dec 2015                                                 *
 * AUTHOR:       rumackr@msoe.edu <Reid Rumack>                              *
 * PROVIDES:     defines register base addresses for the CE2800 DE0          *
 * To Use:                                                                   *
 *    Incluede the header and source file in the project.                    *
```

```
 ***************************************************************************/
#include <inttypes.h>
volatile uint32_t* LEDs                   = (uint32_t*) 0x80802030;

volatile uint16_t* PWM_OCRA1A             = (uint16_t*) 0x80802028;
volatile uint16_t* PWM_OCRA1B             = (uint16_t*) 0x8080202A;
volatile uint16_t* PWM_TCNT               = (uint16_t*) 0x8080202E;

volatile uint8_t*  VGA                    = (uint8_t* ) 0x80800000;

volatile uint8_t* I2C_LO                 = (uint8_t*) 0x80802020;
volatile uint8_t* I2C_HI                 = (uint8_t*) 0x80802021;
volatile uint8_t* I2C_CTRL               = (uint8_t*) 0x80802022;
volatile uint8_t* I2C_TRANSMIT           = (uint8_t*) 0x80802023;
volatile uint8_t* I2C_RECEIVE            = (uint8_t*) 0x80802023;
volatile uint8_t* I2C_CMD                = (uint8_t*) 0x80802024;
volatile uint8_t* I2C_STAT               = (uint8_t*) 0x80802024;

volatile uint8_t* CAM_CTRL               = (uint8_t*) 0x80802000;
volatile uint8_t* PIXEL_PORT             = (uint8_t*) 0x80802010;


volatile uint32_t* ADC_CH0_UPDATE         = (uint32_t*) 0x90002060;
volatile uint32_t* ADC_CH1_AUTO_UPDATE    = (uint32_t*) 0x90002064;
volatile uint32_t* ADC_CH2_SCLK_COUNTER   = (uint32_t*) 0x90002068;
volatile uint32_t* ADC_CH3                = (uint32_t*) 0x9000206C;
volatile uint32_t* ADC_CH4                = (uint32_t*) 0x90002070;
volatile uint32_t* ADC_CH5                = (uint32_t*) 0x90002074;
volatile uint32_t* ADC_CH6                = (uint32_t*) 0x90002078;
volatile uint32_t* ADC_CH7                = (uint32_t*) 0x9000207C;

volatile uint8_t*  UART_BASE              = (uint8_t* ) 0x80800020;
volatile uint8_t*  UART_RECV              = (uint8_t* ) 0x80800020;
volatile uint8_t*  UART_SEND              = (uint8_t* ) 0x80800020;
volatile uint8_t*  UART_STAT              = (uint8_t* ) 0x80800021;
volatile uint8_t*  UART_CTRL              = (uint8_t* ) 0x80800022;
```