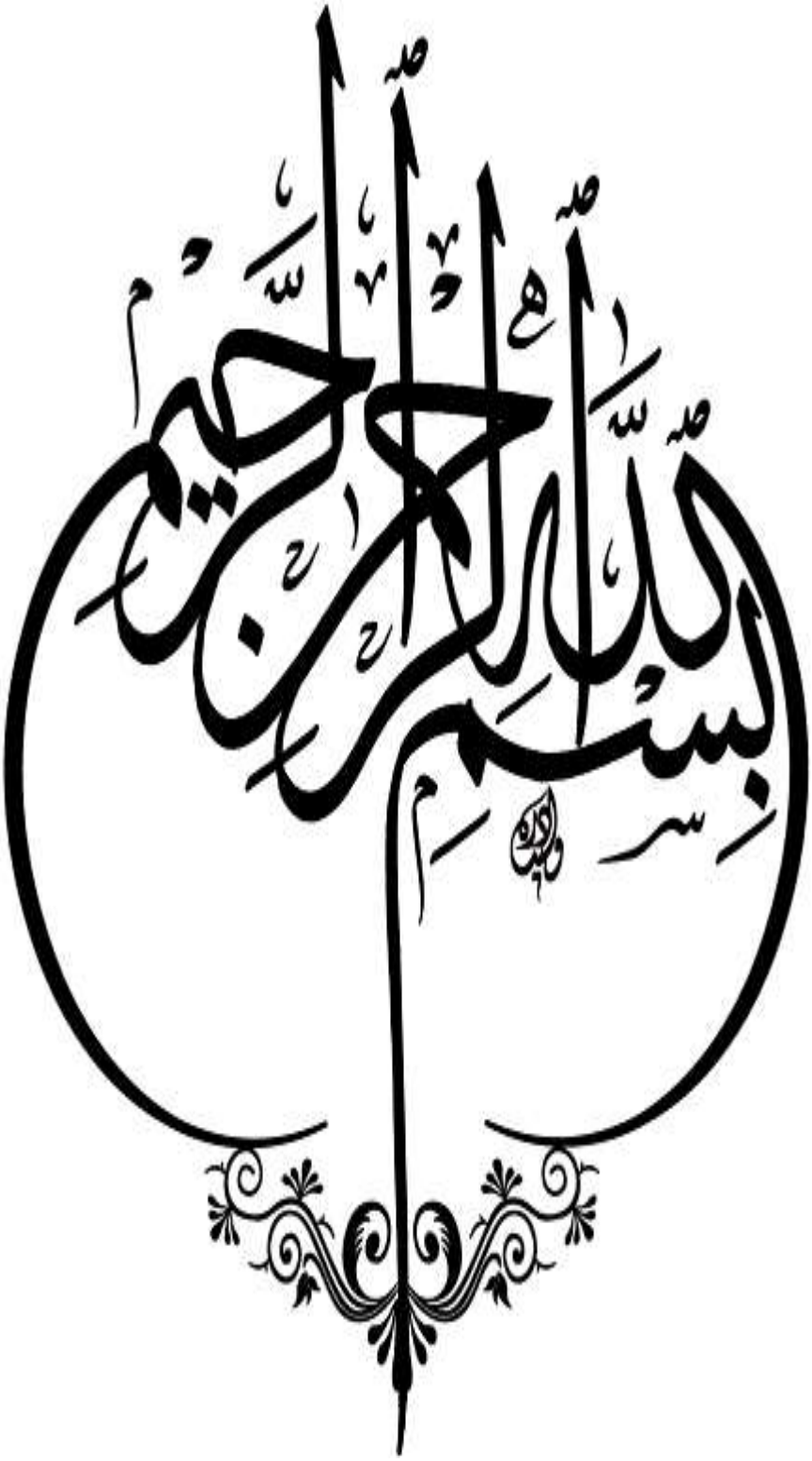

بسم الله الرحمن الرحيم

دانشکده فنی و مهندسی

بخش مهندسی کامپیوتر

استاد مربوطه: **دکتر جمشیدی**

نام پروژه:

ربات فوتبالیست

زمستان 97

ترم تحصیلی: 3971

اعضای گروه:

زهره معین

زهرا سالاری

صادق مهدیان

مجید آرمان نسب

آلماسادات پورحسینی

علیرضا عسکری خیرآبادی

فهرست مطالب

عنوان **صفحه**

مقدمه:

پروژه درس معماری کامپیوتر در ترم حاضر، ساخت ربات فوتبالیست است.
چارچوب کلی این پروژه استفاده از یک میکروکنترلر به عنوان پردازنده و یا مغز ربات است که این پردازنده باید توانایی تشخیص محیط اطراف خود، و سپس تصمیم گیری با توجه به شرایطی که در آن قرار دارد، باشد. قسمت تشخیص محیط اطراف به وسیله عمل پردازش تصویر صورت می گیرد و نهایتا با استفاده از نتایج بدست آمده از پردازش تصویر ربات، ربات تصمیم به انجام کاری(از قبیل حرکت به سمت توپ، گرفتن توپ و...) می گیرد.
به دلیل گروهی بودن این پروژه در ابتدای روند اجرا، گروه ما به سه زیر گروه 3، 2 و 1 نفره تبدیل شد[1] و همچنین پروژه را نیز به سه قسمت تبدیل کردیم که این تقسیم بندی مطابق زیر صورت گرفت:

زیر گروه 3 نفره بر روی قسمت سخت افزاری و پروگرام کردن تراشه

زیر گروه 2 نفره بر روی قسمت پردازش تصویر و کدزنی با کتابخانه OpenCV

زیر گروه 1 نفره بر روی قسمت بدنه و شاسی

[1]: که تقریبا پس از 30 روز بعد از شروع ترم، در کلاس های مکمل و توجیهی، مشخص شد که تقسیم بندی و گروه بندی گروه ما کاملا درست بوده است.

فصل دوم: قسمت سخت افزاری و پروگرام کردن میکروکنترلر

2.1: وسایل مورد نیاز برای انجام این پروژه:

هدربورد آماده (بورد آموزشی) LPC1768
نمایشگر TFT LCD SPI (2.8 Inch)
دوربین OV7670
منبع تغذیه Switching 12V/ 2A

2.2: ابزار مورد استفاده در روند اجرای پروژه:

مالتی متر[۲]
برد بورد[۳]
مقاومت (330 اهم، 100 اهم)
دیود نوری[۴]
سیم جامپر برد بوردی

2.3: نکات کلیدی مربوط به قسمت سخت افزاری:

1. منظور از هدر بورد آماده یا آموزشی، بوردی است که پورت ها و پایه های خاصی به صورت آماده بر روی آن تعبیه شده و علاوه بر این پورت ها و پایه ها، یک تراشه یا میکروکنترلر بر روی آن قرار گرفته است. در واقع این بورد آماده، واسط بین تراشه، پورت ها و پایه هاست. در این گزارش هدربورد مورد استفاده گروه ما، هدر بورد آماده دارای تراشه NXP LPC1768 است. این تراشه از نوع EEPROM [۵] است.
2. نمایشگر پیشنهادی برای استفاده در این پروژه، TFT LCD N96 بود که ما به دلیل تحریم های حاکم بر بازار امروزی کشور، از نمایشگر دیگری تحت عنوان TFT LCD SPI استفاده کردیم. نکته کلیدی در رابطه با انتخاب و خرید نمایشگر این است که نمایشگر بر روی بورد تعبیه شده باشد و به صورت Flat دار نباشد.
3. دوربین ذکر شده در ابتدای گزارش، در دو نوع موجود است. یک نوع بافردار و نوع دیگر بدون بافر.

[2] Multi Meter
[3] Bread Board
[4] LED

[5] قابلیت چندبار پروگرام کردن تراشه

4. منظور از بافر، رجیستر یا حافظه موقتی است که بر روی بورد خود دوربین تعبیه شده است. در روند اجرای این پروژه، ما ابتدا دوربین بدون بافر تهیه کردیم که نهایتا در امر پردازش تصویر به مشکل برخوردیم و مجبور شدیم دوربین بافردار نیز تهیه کنیم. به دلیل بالا بودن سرعت ارسال فریم های دوربین و پایین بودن سرعت پردازش میکروکنترلر مشکل مذکور به وجود آمد.
5. منبع تغذیه ذکر شده در قسمت ابتدای گزارش 12 ولت است که به دلیل این که ولتاژ کاری هدربورد ما 5 ولت بود، منبع تغذیه را نیز به 5 ولت تغییر دادیم.

2.4: نرم افزارهای مورد نیاز مربوط به قسمت سخت افزار:

کامپایلر Keil.MDK-ARM.v5.25 [۶]
پروگرامر Flash Magic

2.5: مرحله اول: تست هدربورد و تراشه

برای تست کردن و حصول اطمینان از سالم بودن هدربورد و تراشه، هدربورد را به وسیله کابل USB to Serial را به لپ تاپ متصل کردیم. پس از روشن کردن هدربورد، در قسمت Control Panel ویندوز، قسمت Device Manager یک پورت COM شناخته می شود. شماره پورت مربوطه نیز مشخص شده است که این به معنی شناخته شدن تراشه و هدربورد توسط لپ تاپ است.

2.6: مرحله دوم: روند کلی پروگرام کردن تراشه

1. ابتدا کد برنامه مورد نظر را در محیط کامپایلر Keil می نویسیم و سپس کامپایل کرده تا در نهایت فایل HEX تولید شود.
2. LPC را به لپ تاپ متصل می کنیم.
3. دو پایه موجود بر روی هدر بورد با عنوان Program که روبروی یکدیگر قرار دارند را به وسیله سیم دو سر ماده به هم متصل میکنیم.
4. وارد نرم افزار Flash Magic می شویم در قسمت COM شماره پورت شناخته شده را وارد کرده و در قسمت Oscillator مقدار عددی کریستال موجود بر روی بورد را وارد می کنیم(کریستال موجود بر روی بورد ما 12 مگاهرتز بود).

[6] لینک دانلود این نسخه در قسمت ضمائم پروژه موجود است

5. فایل HEX تولید شده در Keil را Add میکنیم و سپس بر روی Start کلیک میکنیم. در صورت ملاحظه پیغام Finish Programming عمل پروگرام کردن به پایان رسیده است.
6. در نهایت سیم دو سر ماده را از دو پایه Program جدا می کنیم.

2.6.1: مشکلات مربوط به کامپایل برنامه:

1. در ابتدا، که سعی می کردیم با استفاده از کامپایلر Keil MDK ARM نسخه 5.25 از کتابخانه مربوط به LPC1768 استفاده کنیم این کتابخانه برای کامپایلر شناخته شده نبود و این مشکل باعث پدیدار شدن پیغام Target Can't Create می شد. برای حل این مشکل پکیج مربوط به تراشه های Cortex-M را دانلود[۷] و نصب کردیم.
2. دومین مشکلی که برای کامپایلر وجود داشت این بود که بعد از انتخاب Device مورد نظر، صفحه ای با عنوان Manage Run Time Environment باز می شد که محتوایی در این صفحه وجود نداشت. با حذف پسوند cs از انتهای نام اولیه پروژه ایجاد شده در کامپایلر، محتوای این صفحه نمایش داده شد. سپس از قسمت CMSIS گزینه Core و از قسمت Device گزینه Startup را انتخاب کردیم که این عمل موجب اضافه شدن سه فایل مورد نیاز برای کامپایل برنامه های مربوط به تراشه LPC1768 شد.

2.7: مرحله سوم: بررسی پروگرام پذیر بودن تراشه

برای بررسی و تست پروگرامینگ تراشه، از تست دیود نوری چشمک زن[۸] استفاده کردیم. کد این برنامه به صورت زیر است:

[7] لینک دانلود این پکیج در ضمائم پروژه موجود است.

[8] Blinky LED

```
#include <lpc17xx.h>
void delay_ms(unsigned int ms)
{
  unsigned int i,j;
  for(i=0;i<ms;i++)
    for(j=0;j<20000;j++);
}
 */start the main program */
int main(){
   SystemInit();                  //Clock and PLL configuration
   LPC_PINCON->PINSEL4 = 0x000000;   //Configure the PORT2 Pins as GPIO;
   LPC_GPIO2->FIODIR = 0xffffffff;  //Configure the PORT2 pins as OUTPUT;
  while(1)
   {
     LPC_GPIO2->FIOSET = 0xffffffff;     // Make all the Port pins as high
     delay_ms(100);
     LPC_GPIO2->FIOCLR = 0xffffffff;     // Make all the Port pins as low
     delay_ms(100);
   }
}
```

توضیحات مربوط به کد:

پایه بلند LED را به مقاوت و سپس از مقاوت به پایه P2.13 هدربورد متصل کردیم. و پایه کوتاه را به GND هدربورد متصل کردیم.

2.8: **نمایشگر**

پس از حصول اطمینان از پروگرام پذیر بودن تراشه به سراغ اتصال نمایشگر و نمایش تصویر بر روی نمایشگر می رویم.

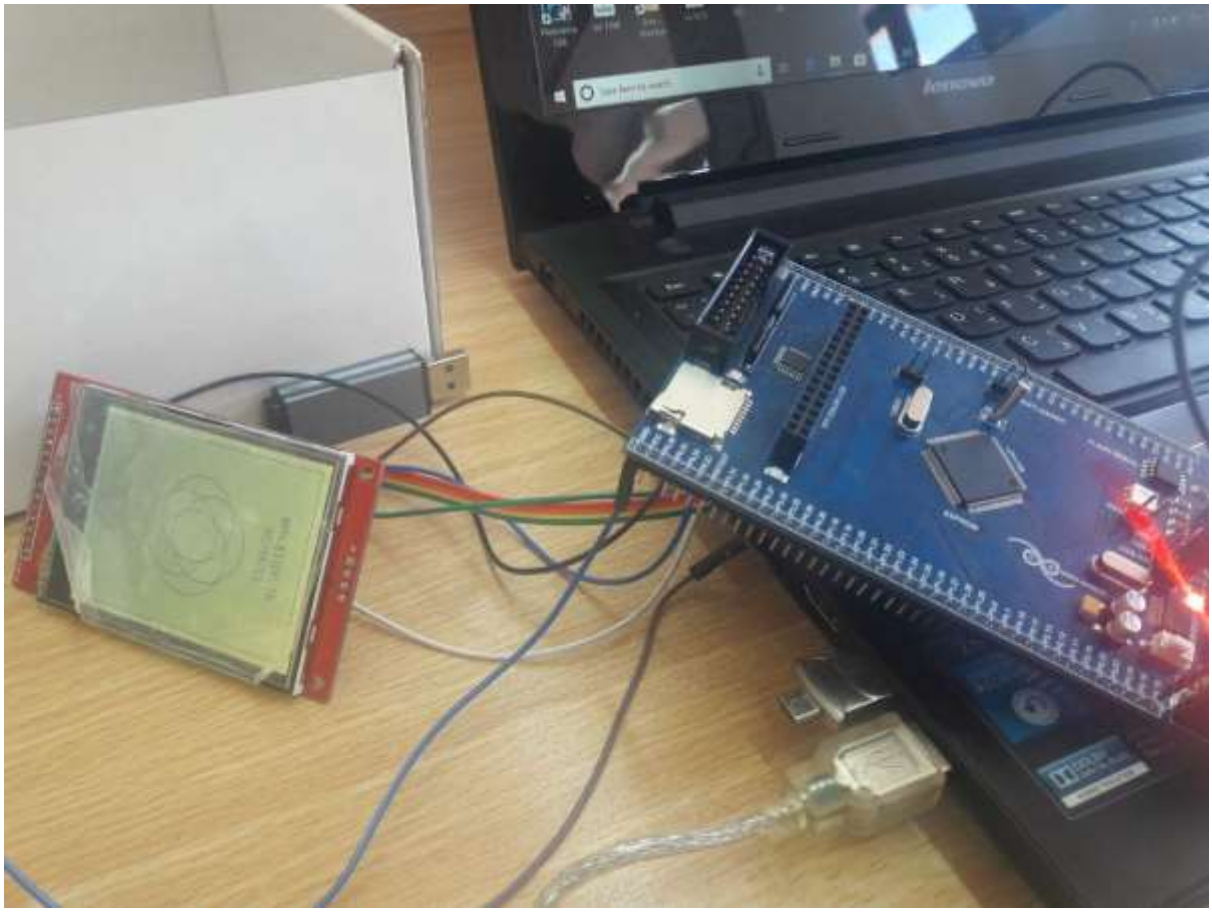
کد مربوط به نمایش تصویر بر روی نمایشگر

```
#include <lpc17xx.h>
#include "lpc17xx_gpio.h"
#include "lpc17xx_libcfg.h"
#include "lpc17xx_pinsel.h"
#include "lcd.h"
int main(void)
{
LCD_Init();
LCD_Clear(GRED);
LCD_ShowString(70,35,200,16,16");    //Says By AraghiPour *.*
LCD_ShowString(70,55,200,16,16,"  97/09/18");
```

```
LCD_DrawRectangle(10, 10, 230, 310);
Draw_Circle((240/2)-20,320/2,50);
Draw_Circle((240/2)+20,320/2,50);
Draw_Circle(240/2,(320/2)+20,50);
Draw_Circle(240/2,(320/2)-20,50);

while(1)
{
}
}
```

هدف از نوشتن این کد نمایش یک تصویر حاوی یک تاریخ و چند دایره تو در تو(به شکل گل) است.



پایه های نمایشگر:

-CS = P0.6(GPIO pin)
-RS = GND
-WR/SCK = P0.7(SCK1)
-RD = GND
-SDO = P0.8(MISO1)
-SDI = P0.9(MOSI1)

2.9: دوربین:

این دوربین دارای یک ارتباط از نوع سریال I2C می باشد.
پروتکل I2C در واقع ادغام شده از بهترین ویژگی های SPI و UART می باشد. توسط این پروتکل امکان اتصال چند Slave به یک Master (مانند SPI) و یا استفاده از چند Master برای کنترل یک یا چند Slave وجود دارد. این ویژگی زمانی که شما می خواهید از چند میکروکنترلر برای ارسال داده به یک کارت حافظه و یا نمایش بر روی LCD استفاده کنید، بسیار مناسب می باشد.
این نوع ارتباط باعث پیکربندی دوربین از جمله سایز تصویر خروجی، تنظیم کنتراست، تنظیم مقدار نرخ رنگ، تنظیم فرکانس و ... می شود.
جدا از مد پیکربندی[9] دارای 8 پین خروجی نیز می باشد که D0 تا D7 نام دارد. این پایه ها نیز وظیفه خروجی رنگ پیکسل ها را دارا می باشند.
پین های ریست کردن دوربین: RCLK و PCLK
پین PCLK برای بعضی از بردها بی استفاده است و این پین همان پین فرکانس ورودی به دوربین است که یان رویداد داخل برد انجام می شود.
پین RCLK نیز به حافظه دوربین متصل هست که با هر کلاک یک خروجی 8 بیتی به ما می دهد.
پین VSYNC که مربوط به Frame Clock است و هر بار که بافر پر شد تصویر در اندازه 320*240 آماده است و ارسال می شود.
با استفاده از پایه های CS, WR, OE,WRST مشخص می کنیم از داخل رجیستر می خواهیم بخوانیم و یا بنویسیم.
زمانی که با استفاده از پین های بالا ورودی و خروجی را مشخص کردیم با استفاده از SIOC و SIOD اطلاعات را منتقل می کنیم.

[9] Configure

2.10: کد تلفیقی اتصال دوربین و نمایشگر

```
#include <LPC17xx.H>
#include <stdio.h>
#include <math.h>
#include "lcd.h"
#include "string.h"
#include "sys.h"
#include "hardware_conf.h"
#include "ov7670.h"

u8 buf1[32];
u16 Camera_Data = 0,i=0;
extern u8 VsyncCnt;
*/
*************************************************************************
*************************************
 *Description:       Delay function
 *Arguments  :       The time of ms
 *Returns    :       None
*************************************************************************
*************************************
/*
static __inline void Delay1us(unsigned int delay1msdata)
{
unsigned int i,j,k;
for(i=0;i<delay1msdata;i++)
for(j=0;j<1;j++)
for(k=0;k<10;k++)
```

```
;
}
static __inline void Camera_RESET(void)
{
FIFO_WRST =0;
DelayNS(1);
FIFO_WRST =1;
DelayNS(1);
FIFO_RRST =0;
DelayNS(1);
FIFO_RCLK =0;
DelayNS(1);
FIFO_RCLK =1;
DelayNS(1);
FIFO_RCLK =0;
DelayNS(1);
FIFO_RCLK =1;
DelayNS(200);
FIFO_RRST =1;
}
static __inline void Camera_Display(void)
{
int x=0,y=0;
for(y=0;y<321;y++)
{
for(x = 0; x <240;) //320*240
{
```

```
FIFO_RCLK =0;
DelayNS(1);
FIFO_RCLK =1;
DelayNS(1);
Camera_Data = (FIFO_DATA_PIN<<8)&0xff00 ;
FIFO_RCLK =0;
DelayNS(1);
FIFO_RCLK =1;
DelayNS(1);
Camera_Data |= (FIFO_DATA_PIN)&0x00ff ;
LCD_SetCursor(x,y);
LCD_WriteRAM_Prepare();
LCD_WR_DATA(Camera_Data);
x=x+1;
}
}
__nop();
FIFO_RRST =0;

__nop();
for(i=0;i<128;i++)
{
FIFO_RCLK =0;
__nop();
FIFO_RCLK =1;
__nop();
}
FIFO_RRST =1;
```

```
__nop();
FIFO_WRST =0;
__nop();
DelayNS(200);
FIFO_WRST =1;
__nop();
VsyncCnt = 0;
__nop();
{
*/
************************************************************************
****************************************
 *Description:       Main function
 *Arguments  :       None
 *Returns    :       None
************************************************************************
****************************************
/*
int main (void)
{
SystemInit();
LCD_Init();
LCD_Clear(GRED);
LCD_ShowString(70,55,200,16,16,"    1397     ");
showimage(95 ,100);
Camera_Configuration();
Camera_RESET();
while (1)
```

```
{
if(OV7660_init()) break;
}
LCD_ShowString(40,75,200,16,16,"OV7670 Init.....OK");
Delay1us(10000000);
OV7660_config_window(272,0,480,640);
while (1)
{
if(VsyncCnt == 2)
{
Camera_Display();
}
}
}
```

توضیحات مربوط به پایه های استفاده شده در کد:

پایه های دوربین:

3V3------->VCC

SIOC------->P0.17

VSYNC------->P2.12

D7------->P2.7

D5------->P2.5

D3------->P2.3

D1------->P2.1

WR------->P0.4

GND------->GND

SIOD------->P0.16

D6------->P2.6

D4------->P2.4

D2------->P2.2

D0------->P2.0

PWDN------->GND

RCK------->P0.3

OE------->P0.5

RRST------->P0.1

WRST------->P0.2

HREF------>P2.8

پایه های LCD:

RST------->P1.14

D/C------->P1.15

CS------->P1.16

WR/SCK------->P0.7

SDO------->P0.8(MISO1)

SDI------->P0.9(MOSI1)

LED------->VCC

نتایج بدست آمده از آزمون و خطا بر روی پایه های دوربین:

1- اگر پایه Reset را متصل کنیم صفحه بدون برفک و رنگی نمایش داده می شود.
2- اگر پایه PWDN را به VCC متصل کنیم LCD بر روی عکس ابتدایی باقی می ماند و تصویر ارسالی توسط دوربین را نمایش نمی دهد.
3- اگر پایه PWDN را اصلا وصل نکنیم تصویر به صورت برفکی و رنگی نمایش داده می شود.

فصل سوم: قسمت پردازش تصویر

زیرگروه ما ابتدا شروع به یادگیری توابع و نحوه کارکردن با کتابخانه OpenCV کردند. پس از یادگیری نسبی، به جستجوی کدهای آماده در اینترنت پرداختیم. پس از تست کردن کدهای متعدد، یکی از آنها که به هدف این پروژه که تشخیص و دنبال کردن توپ است نزدیکتر بود انتخاب کردیم.

نحوه کار کد فعلی(که هنوز بر روی تراشه پروگرام نشده است و به زبان C++ نوشته شده است) به این صورت است که:

به دلیل این که نیاز به تشخیص عددی رنگ های مورد نیاز ربات و اعمال آنها در کد داشتیم، کد را قابل انعطاف کردیم، تا هم نتیجه پردازش تصویر با وب کم لپتاپ را بتوانیم مشاهده کنیم و هم بتوانیم با استفاده از ابزار TrackBar به روش آزمون و خطا هم کد رنگ های مورد نیاز ربات را بدست بیاوریم و هم بفهمیم که پس از تشخیص رنگ و اعمال فیلتر، تصویر دایره در تصویر ارسالی به درستی تشخیص داده می شود و بهترین بازه پارامترهای موثر بر تشخیص دایره در تصویر را بدست بیاوریم. لازم به ذکر است که اعداد مورد نیاز در تست های متعدد این کد، زمانی که بهترین پاسخ را در شرایط محیطی به ما دادند به صورت کاملا دقیق یادداشت کرده و کد خالص پردازش تصویر که قرار است بر روی تراشه پروگرام شود نیز در حال آماده شدن است.

3.1: کد پردازش تصویر ربات:

```
#include <sstream>
#include <string>
#include <iostream>
#include <vector>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv\cv.h>
#include <opencv\highgui.h>
#include <stdlib.h>
#include <stdio.h>

using namespace std;
using namespace cv;


int main(int argc, char** argv) {
	VideoCapture cap(0);

	if (!cap.isOpened())
	{
		cout << "Cannot open the webcam" << endl;
		return 1;
```

```
}

//*****************************************************************************************
//Create Trackbar for color range
//*****************************************************************************************
namedWindow("ColorControl", CV_WINDOW_AUTOSIZE);

int iLowH = 0;
int iHighH = 255;

int iLowS = 0;
int iHighS = 255;

int iLowV = 0;
int iHighV = 255;

cvCreateTrackbar("LowH", "ColorControl", &iLowH, 255);
cvCreateTrackbar("HighH", "ColorControl", &iHighH, 255);

cvCreateTrackbar("LowS", "ColorControl", &iLowS, 255);
cvCreateTrackbar("HighS", "ColorControl", &iHighS, 255);

cvCreateTrackbar("LowV", "ColorControl", &iLowV, 255);
cvCreateTrackbar("HighV", "ColorControl", &iHighV, 255);


//*****************************************************************************************
//Create Trackbar for finding circle parameters
//*****************************************************************************************
namedWindow("CircleControl", CV_WINDOW_AUTOSIZE);

int iSliderValue1 = 50;
createTrackbar("Brightness", "CircleControl", &iSliderValue1, 100);

int iSliderValue2 = 50;
createTrackbar("Contrast", "CircleControl", &iSliderValue2, 100);

int param1 = 100;
createTrackbar("param1", "CircleControl", &param1, 300);

int param2 = 200;
createTrackbar("param2", "CircleControl", &param2, 300);

int minR = 0;
createTrackbar("minRadius", "CircleControl", &minR, 300);

int maxR = 0;
createTrackbar("maxRadius", "CircleControl", &maxR, 300);

//*****************************************************************************************
//Get camera frame always
//*****************************************************************************************
while (true)
{
	Mat imgOriginal;


	bool bSuccess = cap.read(imgOriginal);
	flip(imgOriginal,imgOriginal,2);
	int iBrightness = iSliderValue1 - 50;
	double dContrast = iSliderValue2 / 50.0;
	imgOriginal.convertTo(imgOriginal, -1, dContrast, iBrightness);
```

```
		if (!bSuccess)
		{
			cout << "Cannot read a frame from video stream" << endl;
			break;
		}

		//Change color type from BGR to HSV
		Mat imgHSV;

		cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV);

		//Create mask
		Mat imgThresholded;

		inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV),
imgThresholded);

		dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH_RECT, Size(3, 3)));

		//Make mask blur
		GaussianBlur(imgThresholded, imgThresholded, Size(9, 9), 2, 2);

		//Edge detection
		Mat imgCanny;

		Canny(imgThresholded, imgCanny, 50, 200, 3);

		//Find circles
		vector<Vec3f> circles;
		double dparam1 = param1 / 1.0;
		double dparam2 = param2 / 1.0;

		HoughCircles(imgCanny, circles, CV_HOUGH_GRADIENT,
			2,				// accumulator resolution (size of the image / 2)
			imgCanny.rows / 4,		// minimum distance between two circles
			dparam1,			// Canny high threshold
			dparam2,			// minimum number of votes
			minR, maxR);			// min and max radius

		cout << circles.size() << endl <<
"****************************************************************************************" <<
endl;

		//Show circles
		for (size_t i = 0; i < circles.size(); i++)
		{
			Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
			int radius = cvRound(circles[i][2]);
			circle(imgOriginal, center, 3, Scalar(0, 255, 0), -1, 8, 0);
			// circle outline
			circle(imgOriginal, center, radius, Scalar(0, 0, 255), 3, 8, 0);
			putText(imgOriginal, "Circle Found", Point(+0, 50), 1, 2, Scalar(0, 255, 0), 2);
		}

		imshow("Original", imgOriginal);
		imshow("Mask", imgThresholded);
		imshow("Canny", imgCanny);


		if (waitKey(30) == 27)
		{
			cout << "esc key is pressed by user" << endl;
			break;
```

```
                }
        }
        return 0;
}
```

پس از بدست آوردن مقادیر(فاکتور های) رنگی دقیق توپ، کد را از نحوه همه منظوره(داشتن TrackBar و انعطاف زیاد) به کد اختصاصی ربات خودمان تغییر دادیم.

داده های رنگی مربوط به توپ مورد استفاده ما در فرمت HSV به صورت بازه ای از قرار زیر است:

H: 0.0516 <---- 0.0488

S: 0.9130 <---- 0.8039

V: 1.0000 <---- 0.9020

3.2. توضیحات مربوط به کد:

در کتابخانه Open CV با استفاده از تابع HoughCircles می توان دایره های موجود در تصویر را پیدا کرد که مشخصات این دایره ها شامل مختصات X و Y مرکز دایره و شعاع آن در آرایه ای از بردار های ۳ متغیره ذخیره می شود.

برای پیدا کردن فاصله مد نظر که توپ از دوربین باید داشته باشد با استفاده از کتابخانه Open CV می توان بعد از پیدا کردن توپ و شعاع آن با نزدیک کردن توپ به دوربین هر بار شعاع آن را چاپ کنیم و هنگامی که به فاصله مد نظر رسید شعاع آن را یادداشت کرد.

هرگاه شعاع توپ بزرگتر مساوی شعاعی باشد که در مرحله قبل پیدا کردیم یعنی توپ در محدوده مورد نظرمان است.

****

**

تابع IsDetect تابعی برای پیدا کردن توپ با رنگ و شعاع مد نظر است که یک شیء از کلاس Mat گرفته و درصورت پیدا کردن توپ در آن True برمیگرداند.

این تابع ابتدا با استفاده از تابع cvtColor(pic, imgHSV, COLOR_BGR2HSV) فرمت رنگی تصویر را به HSV تبدیل میکند و سپس با استفاده از تابع

inRange(imgHSV, Scalar(0, 45, 80), Scalar(20, 255, 255), imgThresholded);

و همچنین

dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH_RECT, Size(3, 3)))

ماسک مورد نظر و بازه های رنگی را اعمال میکنیم.

تابع

GaussianBlur(imgThresholded, imgThresholded, Size(9, 9), 2, 2)

افکت تصویری مناسبی را روی تصویر اعمال می کند.

با استفاده از تابع

Canny(imgThresholded, imgCanny, 50, 200, 3)

در ماسک مورد نظر گوشه یابی می کنیم.

با تابع HoughCircles دایره های موجود در تصویر را پیدا میکنیم و در آرایه ای از بردار های ۳ متغیره ذخیره میکنیم.

اگر دایره ای پیدا شد و شعاع آن در محدوده شعاع مورد نظر بود True برمیگردانیم.

کد کامل پیدا کردن توپ به وسیله شعاع و رنگ مورد نظر به شرح زیر است:

```
bool Distance(int Radius)
{
	if (Radius>=195)
	{
		return true;
	}
	else
	{
		return false;
	}
}

bool isDetect(Mat pic)
{
	//Change color type from BGR to HSV
	Mat imgHSV;

	cvtColor(pic, imgHSV, COLOR_BGR2HSV);

	//Create mask
	Mat imgThresholded;

	inRange(imgHSV, Scalar(0, 45, 80), Scalar(20, 255, 255), imgThresholded);

	dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH_RECT, Size(3, 3)));

	//Make mask blur
	GaussianBlur(imgThresholded, imgThresholded, Size(9, 9), 2, 2);

	//Edge detection
	Mat imgCanny;
```

```
	Canny(imgThresholded, imgCanny, 50, 200, 3);

	//Find circles
	vector<Vec3f> circles;
	double dparam1 = 150 / 1.0;
	double dparam2 = 100 / 1.0;

	HoughCircles(imgCanny, circles, CV_HOUGH_GRADIENT,
		2,                                                    // accumulator resolution (size of the
image / 2)
		imgCanny.rows / 4,                    // minimum distance between two circles
		dparam1,                              // Canny high threshold
		dparam2,                              // minimum number of votes
		0, 300);                      // min and max radius

	int Radius = -1;

	for (size_t i = 0; i < circles.size(); i++)
	{
		if (Radius>circles[i][2])
		{
			Radius = circles[i][2];
		}
	}
	if (Distance(Radius))
	{
		if (circles.size() > 0)
		{
			return true;
		}
		else
		{
			return false;
		}
	}
}
```

3.3: پردازش تصویر بر روی تراشه

زمانی که گروه ما متوجه این موضوع شد که به دلیل حجم پایین حافظه فلش تراشه، امکان اضافه کردن کتابخانه OpenCV به بقیه کدها نیست، این موضوع که توانایی استفاده از این کتابخانه را نداریم برایمان روشن شد. پس به این فکر افتادیم که پردازش تصویر را به صورت کاملا دستی[10] در داخل خود کدهای ربات پیاده سازی کنیم که این کار تقریباً بزرگترین چالشی بود که در طول انجام کل پروژه با آن روبرو شدیم.

بنابر نحوه ارسال داده از دوربین که به صورت هر دو کلاک اطلاعات یک پیکسل است، تصمیم گرفتیم تا پردازش پیکسل به پیکسلی بر روی داده های ارسالی توسط دوربین انجام دهیم به این شکل که ابتدا تصمیم داشتیم تا آرایه ای از پیکسل ها، که البته هر پیکسل شامل سه مشخصه R و G و B که هر کدام از نوع integer هستند، است بسازیم تا بتوانیم پس از ارسال هر فریم، بر روی پیکسل های آن فریم پردازش خود را انجام دهیم. در اینجا با چالشی

[10] Manual

جدید روبرو شدیم که علت آن باز هم حجم پایین حافظه فلش تراشه بود. چرا که زمانی که ما این کار انجام دادیم حجم فایل تولیدی ما از 1 مگابایت بیشتر شد و این در حالی بود که ما فقط 512 کیلوبایت حافظه در اختیار داشتیم.

پس در نهایت متوجه شدیم که نمی توانیم فریم به فریم بر روی تصویر ها پردازش انجام دهیم. فلذا تصمیم بر آن شد که پیکسل به پیکسل پردازش را انجام دهیم و اما چطور؟

سیاست پیاده سازی این تصمیم، به این نحو بود که در صورتی که در هر فریم ارسالی که شامل 320*240 پیکسل است یک شمارنده قرار دهیم و هر زمان که تعداد بیشتر از 50 پیکسل در یک لاین از یک فریم، مطابق با رنگ مورد نظرمان بود نتیجه حاصل می شود که توپ در میدان دید ربات قرار دارد و لازم است تا برای گرفتن توپ به جلو حرکت کند.

شاید حرف زدن راجع به این کار آسان باشد، اما پیاده سازی آن بسیار بیشتر از حرف زدن راجع به آن وقت گیر و طاقت فرساست.

نیاز بود که اطلاعات هر پیکسل را دریافت کنیم اما چطور؟ اگر به کد تلفیقی اتصال دوربین و ال سی دی نگاهی بیندازید میبینید که یک متغیر از نوع u16 و به نام Camera_Data وجود دارد که این متغیر در دو حلقه for تو در تو که شمارنده های این حلقه ها نشانگر پیمایش فریم به فریم تصویر بودند، به تابع LCD_WR_DATA ارسال می شود. ما احتمال دادیم این متغیر شامل همان اطلاعاتی است که ما به دنبال آن هستیم. یعنی اطلاعات مربوط به هر پیکسل.

پس دست به کار شدیم تا بتوانیم از عددی که در این integer ذخیره شده بود اطلاعات رنگی یا RGB پیکسل مورد نظر را استخراج کنیم.

ابتدا یک آرایه بولی 16 خانه ای ساختیم و سپس این عدد صحیح را به نحوه زیر به 16 بیت صفر و یک تبدیل کردیم تا بتوانیم اطلاعات RGB دوربین که با فرمت 565 ارسال می شدند را به دست آوریم. در زیر کد مربوطه را مشاهده می نمایید:

```
void Dec2Binary(int Camera_DA)
{
	while (n>0)
	{
		if (n % 2 == 1)
			RGB_ARRAY[i] = true;
		else
			RGB_ARRAY[i] = false;
		n = n / 2;
		i++;
	}
}
```

البته لازم به ذکر است که با این کار در آرایه فرمت BGR ذخیره میشد که برای ما مهم نبود.
پس از این کار دیگر اطلاعات رنگی پیسکل ها را میتوانستیم بدست بیاوریم با تابع زیر:

```
Bool ImageProcessing()
{
	int Red = 0, Green = 0, Blue = 0;
	short i;
	for (i = 0; i<5; i++)
	{
		Blue += ((short)RGB_ARRAY[i])*Power(2, i);
	}
	for (i = 5; i<11; i++)
	{
		Green += ((short)RGB_ARRAY[i])*Power(2, i - 5);
	}
	for (i = 11; i<16; i++)
	{
		Red += ((short)RGB_ARRAY[i])*Power(2, i - 11);
	}
}
```

حال که به مقصود مورد نظر رسیدیم فقط کافی بود تا با یک دستور if، اطلاعات هر پیکسل را با اطلاعات رنگی توپ مد نظرمان مقایسه کنیم و در صورت تطبیق True و در غیر این صورت False را برگردانیم.

به این دلیل که کیفیت دوربین ما با کیفیت دوربین گوشی و همچنین وب کم لپ تاپ قابل قیاس نبود و البته داده های ارسالی در هنگام ارسال دچار نویز می شدند، مجبور به انجام آزمون و خطا در رابطه با دریافتن RGB توپ مورد استفاده. پس از تقریباً 150 بار تست کد با بازه های مختلف RGB و شرایط نوری و محیطی مختلف به این نتیجه دست پیدا کردیم که اگر محیط و در اصل پشت زمینه تصویری که ربات دریافت می کند به رنگ سیاه، و مولفه های پیکسلی که ما چک میکنیم به حالت زیر باشند بهترین عملکرد را بدست می آوریم:

```
if ((Red>190) && ((Green>20)) && ((Blue>85)) && (Green<150))
{
	return TRUE;
}
else
{
	return FALSE;
}
```

البته متوجه شدیم که فرمت نرمال RGB به صورت 888 است تا هر مولفه بین 0 و 255 قرار بگیرد. این تبدیل را نیز انجام دادیم که با مراجعه به کد میتوانید متوجه این امر شوید.

فصل چهارم: قسمت شاسی و بدنه ربات

4.1: نحوه حرکت ربات :

**در ابتدا که کد های اولیه حرکت برای ربات زده شد به شکل زیر بود.

- توضیحات:

فرم حرکت ربات به این شکل است که مبتنی بر مرکز تصویر گرفته شده کار نمیکند و بر اساس دید لحظه ای حرکت می کند و همواره با پردازش تصویر در تعامل است و از آن دستور می گیرد.

به شکلی که ابتدا به صورت پیش فرض به سمت راست میچرخد تا زمانی که توپ را مشاهده کند. سپس به سمت آن حرکت میکند و اگر در بین حرکت نیز زمانی که توپ از میدان دید آن خارج شود می ایستد و دوباره آنقدر به سمت راست می چرخد تا توپ را ببیند و این عمل آن قدر انجام می شود تا به فاصله کافی از توپ رسیده باشد؛ در این مرحله کچر فعال شده و توپ را میگیرد و سپس به سمت چپ میچرخد تا دروازه را ببیند و به سمت آن حرکت کرده و وقتی به آن نزدیک شد ایستاده و شوتر فعال می شود و توپ را گل می کند.

کد های حرکت دارای توابع زیر است:

int IsDetect(int x) : این تابع باگرفتن تصویر و یک ورودی int، تصویر را برای یافتن توپ جست و جو می کند

int IsDistant(int x) : این تابع در کنار پردازش تصویر فاصله ربات را از توپ یا دروازه به شکل تقریبی محاسبه میکند

int forward1() : تابع جلو برنده ربات برای توپ

int forward2() : تابع جلو برنده ربات برای دروازه

void backward() : تابع عقب برنده ربات

int left() : تابع چپ گرد درجا

int right() : تابع راست گرد درجا

int cach() : تابع گیرنده توپ

void shoot() : تابع شوت کننده توپ

**اما الان توابع حرکت از ریشه تغییر کرده و بر اساس مرکز گرایی کار می کنند.

- توضیحات:

به شکلی که ابتدا ربات برای یافتن توپ به سمت راست می چرخد تا توپ را پیدا کند و به محض دیدن آن می ایستد و از تابع پردازش تصویر برای تعیین جهت حرکت خود استفاده می کند، یعنی اگر توپ در سمت چپ تصویر بود گردش به راست کند و اگر در وسط تصویر بود مستقیم به سمت آن حرکت کنند ؛ در کنار پردازش تصویر از تابع تعیین فاصله برای مشخص کردن فاصله خود از توپ استفاده کند.

بعد از نزدیک شدن به توپ کچر فعال شده و توپ را میگیرد.

حال ربات باید به دنبال دروازه بگردد، سپس به سمت چپ می چرخد تا دروازه را پیدا و مثل مراحل قبل رفتار میکند و بعد از نزدیک شدن به دروازه شوتر فعال شده و توپ را گل می کند.

کد های حرکت دارای توابع زیر است.

char IsDetect(char i) : این تابع با گرفتن بک کاراکتر مشخص می کند که آیا توپ مشاهده شده است یا خیر.

char IsDistant(char j) : این تابع در کنار پردازش تصویر فاصله ربات را از توپ یا دروازه به شکل تقریبی محاسبه میکند

void Isball() : این تابع متد های جستجو و تعیین حرکت ربات در آن ها فراخوانی میشود برای یافتن توپ

void Isgate() : این تابع متد های جستجو و تعیین حرکت ربات در آن ها فراخوانی میشود برای یافتن توپ

void forward() : تابع جلو برنده ربات برای توپ و دروازه

void backward() : تابع جلو برنده ربات برای توپ و دروازه

void left() : تابع چپ برنده ربات برای توپ و دروازه

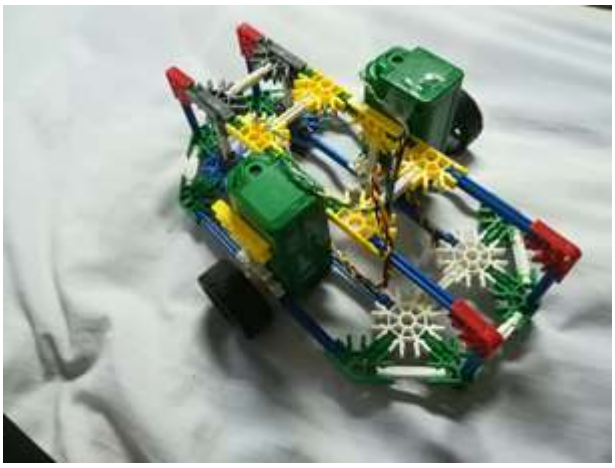
void right() : تابع راست برنده ربات برای توپ و دروازه

void cach() : تابع گیرنده توپ

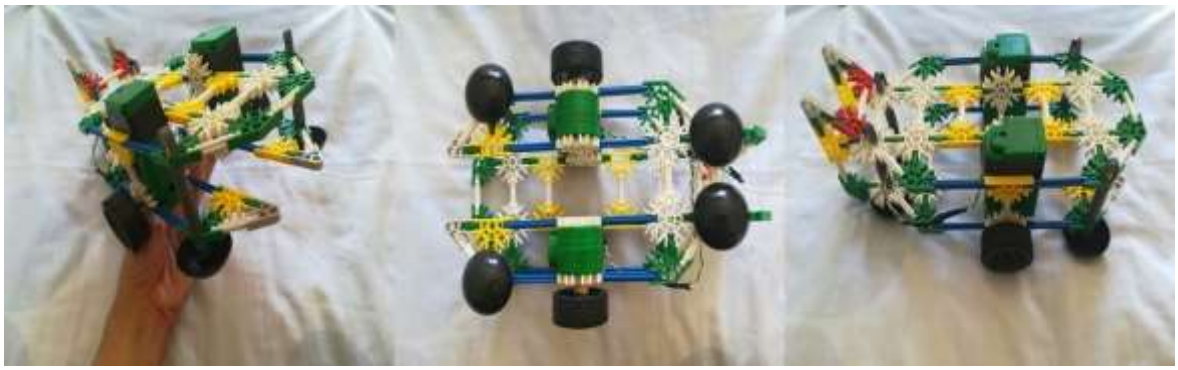
void shoot() : تابع شوت کننده توپ

4.2: شاسی ربات :

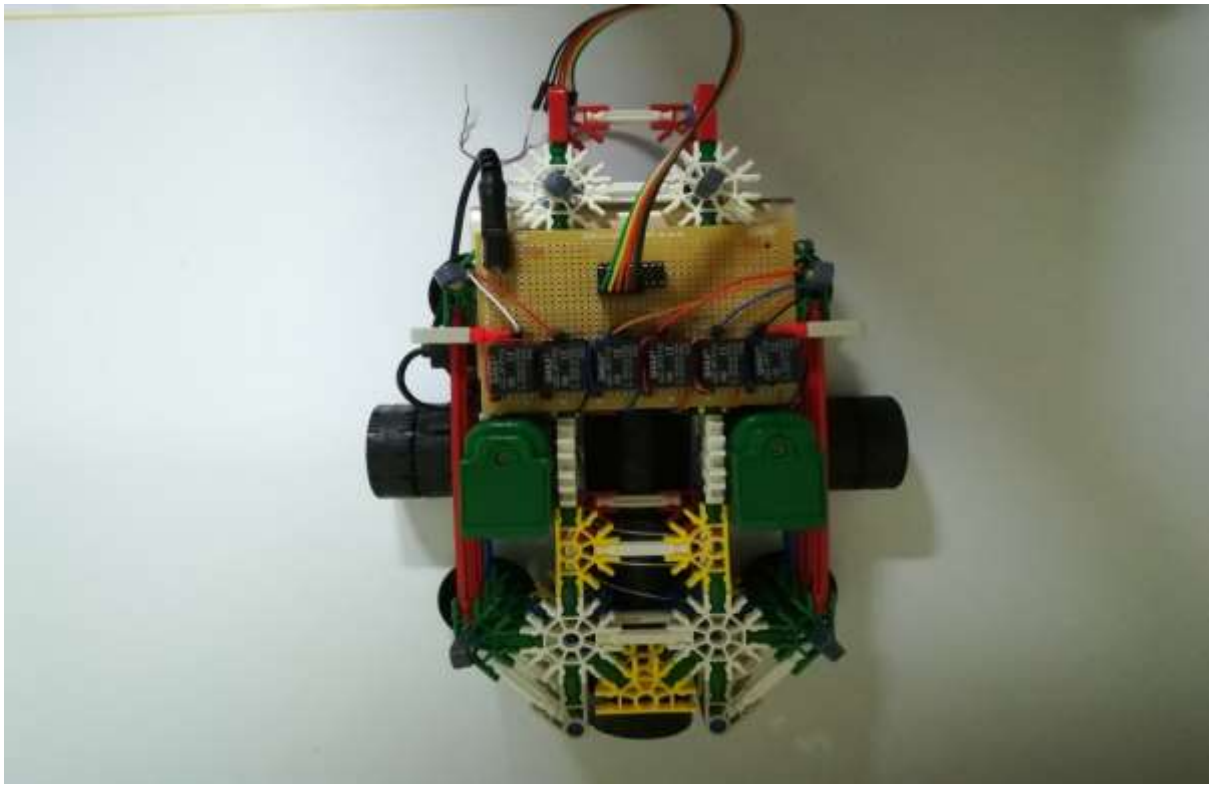
برای طراحی شاسی ربات از ساده ترین شکل ممکن شروع کرده و آن را گسترش دادیم. لازم به ذکر است که ما برای ساخت شاسی از وسایل آمده مخصوص ربات سازی استفاده کردیم.



در وهله بعد به شکل زیر تبدیل شد:



و شکل نهایی (بدون میکرو ، ال سی دی و دوربین) :

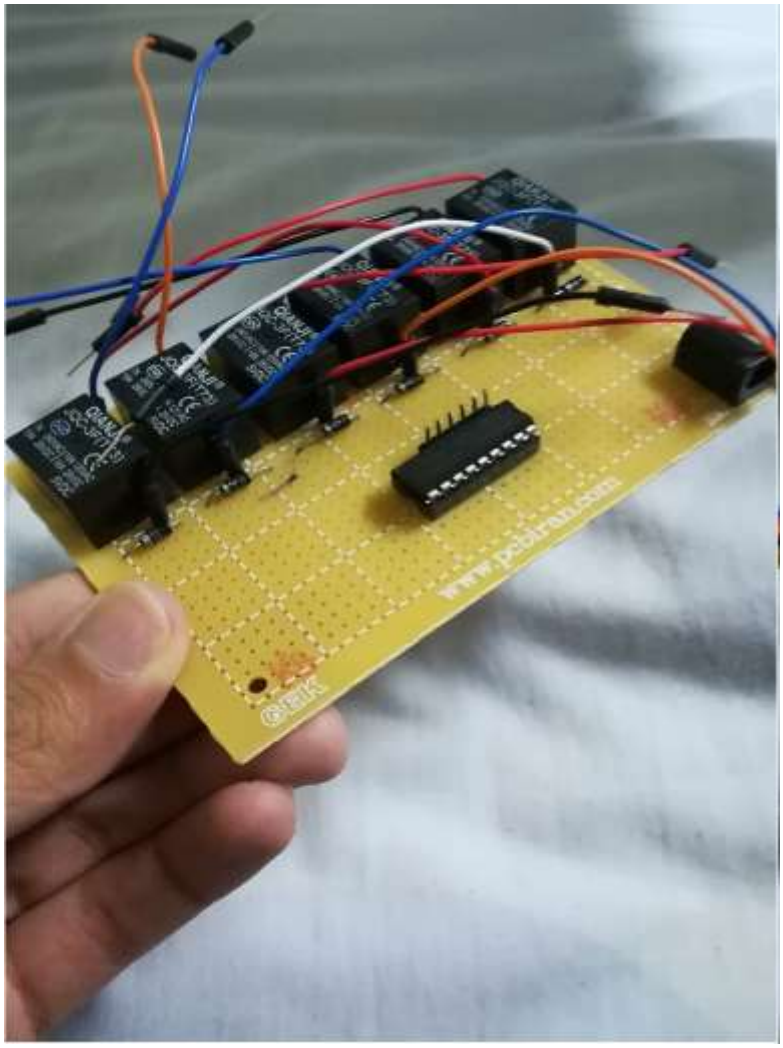


اجزای شاسی:

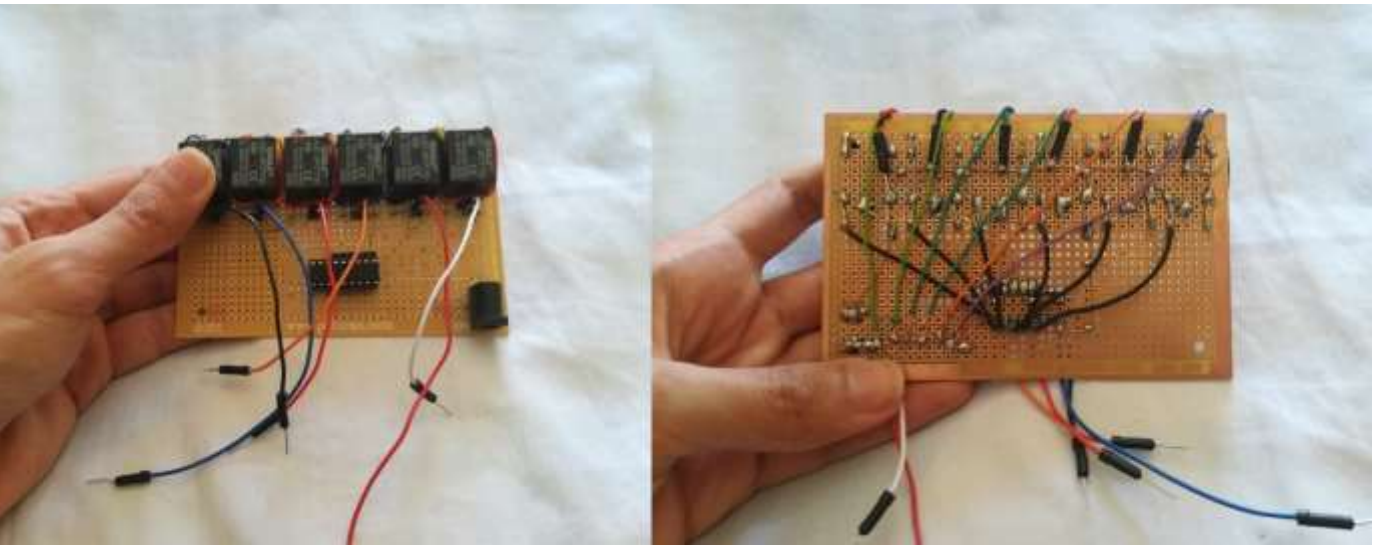
2 موتور گیرباکسی DC با ولتاژ ورودی 3 تا 5 ولت و 500mA تا 1 آمپر با 2 چرخ و چند پایه ایستا کننده.

4.3: درایور موتور:

یک برد دارای 6 رله و یک تراشه ULN2803 که قرار است 3 موتور را راه اندازی کند. البته در ابتدا آماده نبود و اتصالات اجزای آن به شکل کامل انجام نشده بود .



بعد از لحیم کاری و اتصال چند سیم درایور موتور آماده است و حتی می تواند چپ گرد و راست گرد بودن موتور هارا تعیین کند.



نحوه اتصال درایور به قطعات:

6 پین هدر تراشه را به ترتیب به پایه های میکرو که قبلا انتخاب شده اند را با سیم های 2 سر ماده به تراشه متصل می کنیم.
حال سیم های پایه کام 2 رله کنار هم را به ترتیب به پایه 1 موتور متصل می کنیم در واقع 2 رله برای 1 موتور. و این کار را برای 2 موتور دیگر تکرار می کنیم.
هر رله دارای 2 پایه Nc , No است که پایه های Nc همه رله هارا منفی میدهیم و پایه ای No همه رله هارا مثبت می دهیم.

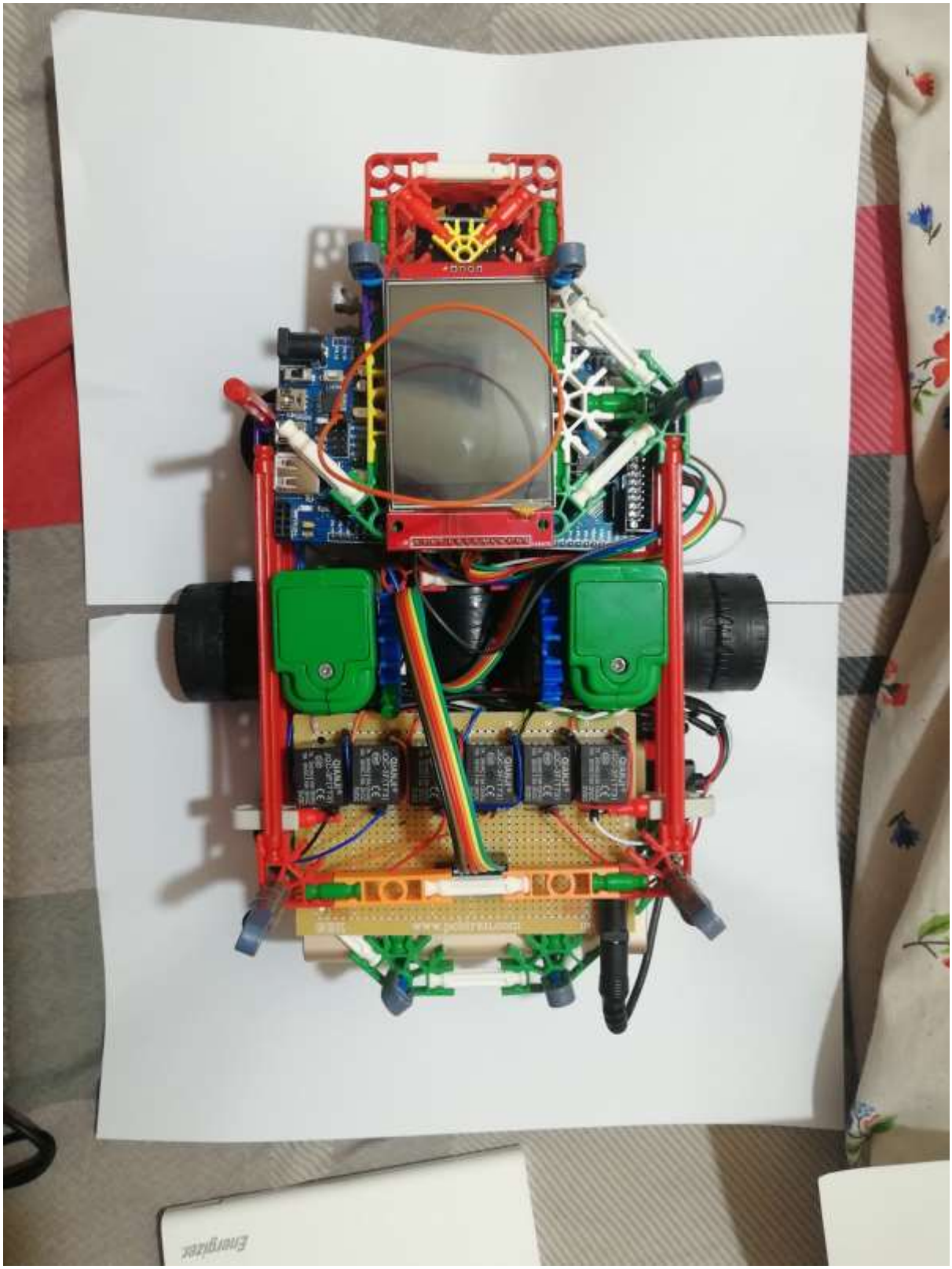
اکنون با فعال کردن 1 رله از جفت رله 1 موتور ، می توان راست گرد و چپ گرد بودن آن را مشخص کرد.

4.4: کچر / شوتر :

یک خلاقیت جالب از ترکیب کردن 2 وظیفه گرفتن توپ و شوت کردن توپ با مکنده.

این مکنده دارای یک موتور کورلس کواد کپتر است و ولتاژ ورودی 3.7 تا5 ولت با 300mA جریان که توپ پینگ پنگ را از فاصله 3 سانتی متری می کشد.

4.5: تصویر نهایی ربات



فصل پنجم

نکاتی چند در رابطه با انجام این پروژه:

در این قسمت قصد داریم تا برخی از اطلاعات مفید و مهم که برداشت مستقیم از دیتاشیت سخت افزارهای مورد استفاده ما هستند و همچنین ما حصل جستجوهایی چند در منابع مختلف است را بیان کنیم.

تراشه LPC 1768:

این تراشه یا میکروکنترلر از ریزپردازنده Arm Cortex M3 استفاده می کند. نکته جالب توجه در مورد ریزپردازنده های سری Cortex M3 از شرکت ARM این است که بسیار کم مصرف هستند و به نسبت مصرف کم، توان پردازشی بالایی دارند.. به طور مثال سرعت برخی از پردازنده های اینتل مانند 80286 به طور تقریبی با سرعت Cortex M3 برابر می باشد. این در حالی است که مصرف پردازنده اینتل 50 برابر Cortex M3 می باشد. از این رو اگر پردازنده اینتل یک باتری را در مدت زمان یک ساعت خالی کند، Cortex M3 همان باتری را در مدت زمان 50 ساعت خالی خواهد کرد. این ریزپردازنده در دسته ریزپردازنده های همه منظوره قرار می گیرد

و اما مشخصاتی چند درباره این ریزپردازنده:

- فرکانس کاری بیش از 100 مگاهرتز
- بر پایه معماری Harvard با قابلیت پیش بینی دستورهای جامپ و دارای پایپ لاین[11] با 3 استیج[12]
- دارای 512 کیلوبایت حافظه فلش بر روی تراشه[13]
- دارای 64 کیلوبایت حافظه داده
- دارای 70 پین I/O همه منظوره

[11] Pipeline
[12] Stage
[13] On Chip

- 32/16 کیلوبایت حافظه SRAM بر روی CPU
- کریستال اوسیلاتور با محدوده کاری 1 تا 25 مگاهرتز

تا به اینجا مشخصات کلی این ریزپردازنده گفته شد. حال می خواهیم به مشخصات جزئی تر این ریزپردازنده بپردازیم.

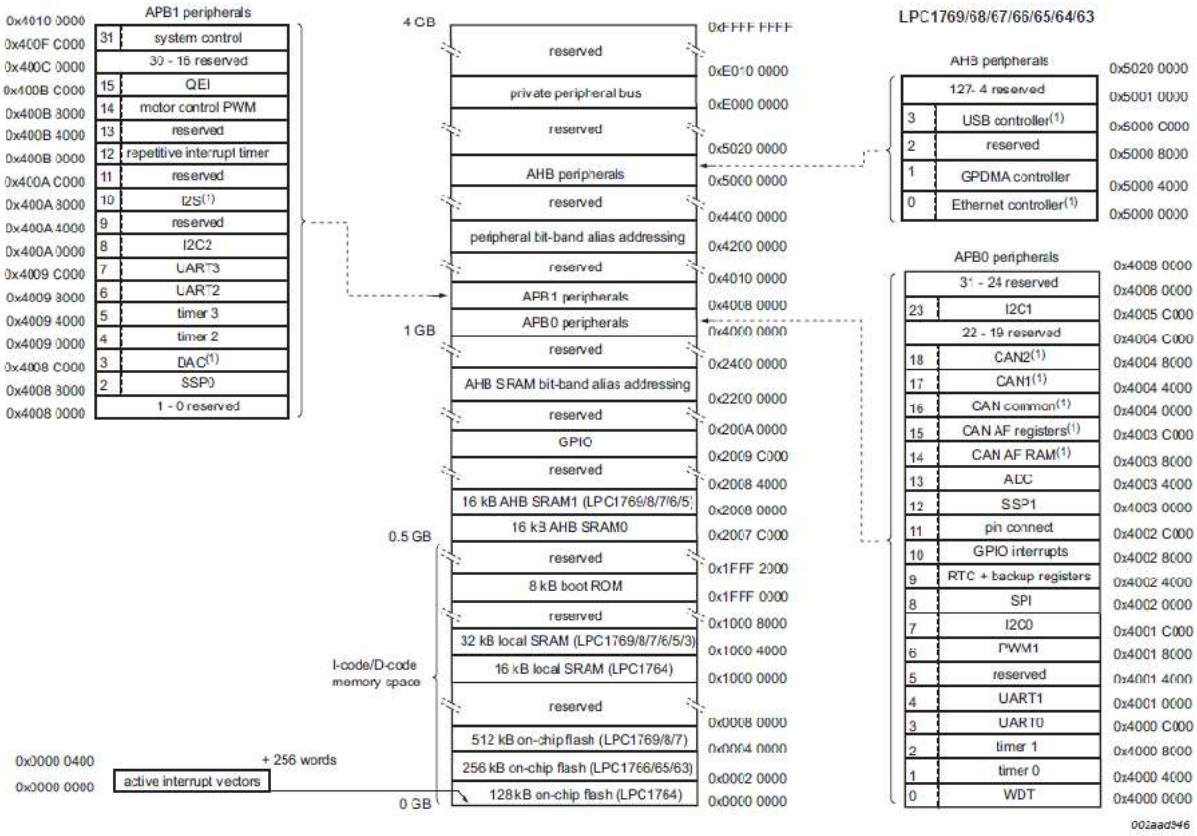
ریزپردازنده Cortex M3 یک ریزپردازنده 32 بیتی است. در این ریزپردازنده، سه گذرگاه وجود دارد به نام های:

1. System Bus
2. I-Code Bus
3. D-Code Bus

که گذرگاه I-Code برای واکشی دستورات[۱۴] و گذرگاه D-Code برای دسترسی به داده ها استفاده می شود. این دو گذرگاه که در دیتاشیت از آنها با نام "گذرگاه های هسته ای[15] یاد شده است، از گذرگاه سیستم سرعت بیشتری دارند. استفاده از دو گذرگاه هسته ای، این اجازه را می دهد تا اگر دو عمل به طور همزمان نیازمند به اطلاعات دو دستگاه مختلف بودند، قادر باشند عملکردی همزمان را به اجزا بگذارند.

میکروکنترلر LPC1768 از یک ماتریس AHB[۱۶] (گذرگاه پیشرفته پربازده) چند لایه برای اتصال به گذرگاه های ARM Cortex M3 استفاده می کند. این ماتریس، گذرگاه های داده و دستور واحد پردازنده مرکزی ARM Cortex M3 را به حافظه فلش، حافظه اصلی SRAM (32 KB) و حافظه بوت رام متصل می کند.

[14] Fetching Instructions
[15] "Core Buses"
[16] Advanced High-performance Bus



تصویر الف) Memory Map of LPC1768

این میکرو کنترلر دارای واسط USB[17] نیز هست.

واسط USB یک گذرگاه 4 سیم است که ارتباط میان یک میزبان[18] و یک یا بیشتر(تا 127) وسایل جانبی[19] را پشتیبانی می کند. کنترلر میزبان، پهنای باند را از طریق یک پروتکل که بر پایه Token بنا شده است به وسایل جانبی اختصاص می دهد. امکان تبادل اطلاعات بین یک میزبان و یک دستگاه که از طریق USB به هم متصل شده اند با سرعت 12 مگابیت بر ثانیه وجود دارد.

هر میکروکنترلر LPC که نام آن با شماره 17 شروع شود، دارای 4 میکروچیپ UART[20] است.

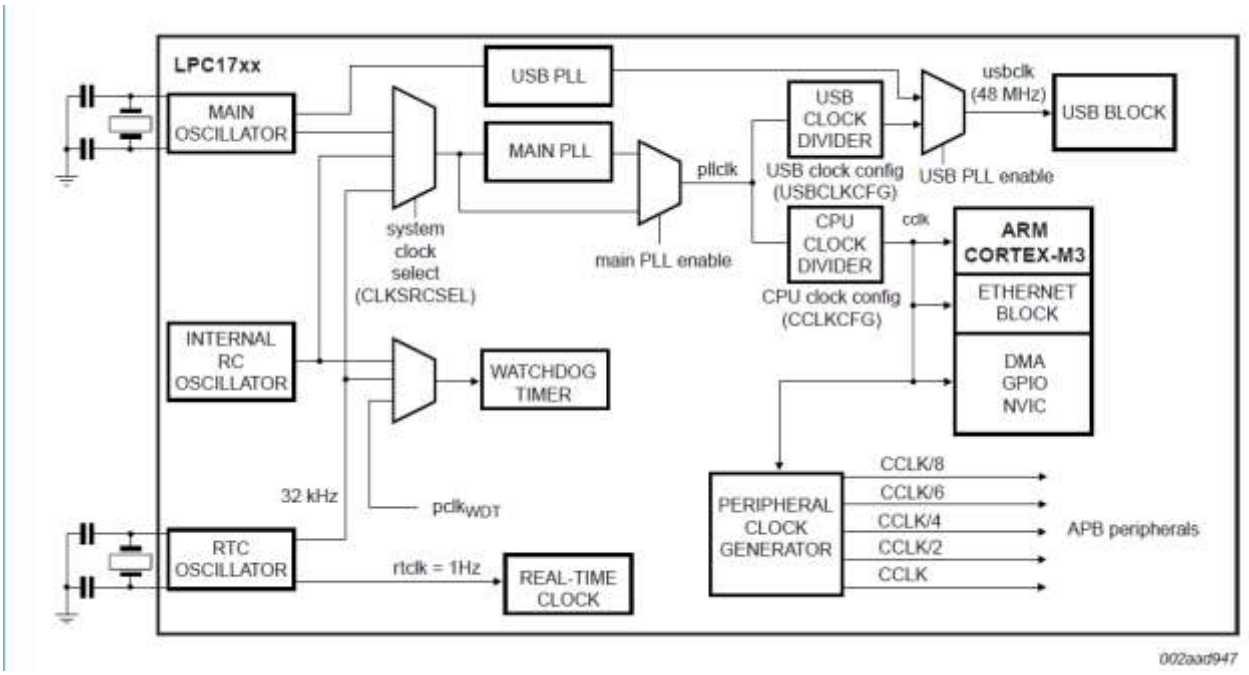
UART ها دارای یک مولد کوچک Baud Rate هستند که به وسیله آن، Baud Rate استانداردی مانند 115200 Bd با هر کریستالی با فرکانس بالاتر از 2 مگاهرتز قابل دسترسی است. بالاترین نرخ بیت داده در UART، 6.25 مگابیت بر ثانیه است.

[17] Universal Serial Bus
[18] Host
[19] Peripheral
[20] Universal Asynchronous Receiver / Transmitter

هر میکروکنترلر LPC که نام آن با شماره 17 شروع شود دارای اوسیلاتور مستقل هست به نام های:

- Main Oscillator
- IRC Oscillator
- RTC Oscillator

که اوسیلاتور RTC می تواند به عنوان منبع کلاک برای وسایل جانبی خارجی متصل به تراشه نیز استفاده شود.



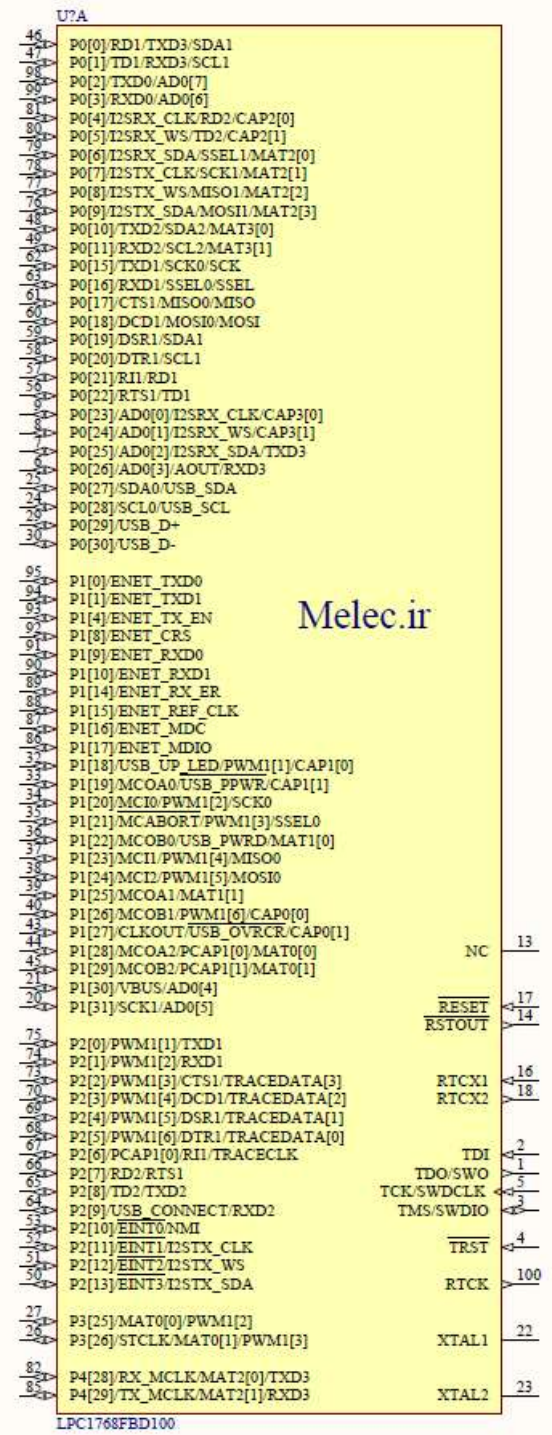
تصویر ب) مدار تولید کننده کلاک LPC17XX

تا به اینجا از برخی امکانات موجود بر روی تراشه مذکور مطلع گشتید.

در اینجا کمی در مورد کار با این تراشه صحبت میکنیم و نحوه برنامه نویسی برای این تراشه را کمی توضیح خواهیم داد.

ممکن است شما هم پس از خرید هدربورد شامل این تراشه، نحوه نام گذاری پایه ها برایتان سوال باشد.
لازم است بدانید که میکروکنترلر LPC1768 حداکثر دارای 5 پورت و این 5 پورت سرجمع دارای 70 پین ورودی و خروجی(GPIO) می باشند. نام گذاری این پورت ها به صورت P0,P1,P2,P3,P4 می باشد.

به طور مثال اگر کنار یک پین نوشته P0.1 را مشاهده کردید بیانگر پین یک از پورت صفر است.



تصویر ج) نقشه پایه های میکروکنترلر LPC1768

حال بهتر است تا نگاهی کوتاه به رجیسترهای GPIO این میکروکنترلر بیندازیم:

رجیستر FIOxDIR

این رجیستر برای تعیین جهت پورت ها (ورودی یا خروجی) استفاده می شود.

رجیستر FIOxSET

این رجیستر جهت 1 کردن پین هایی که بصورت خروجی تعریف شده اند به کار می رود.

رجیستر FIOxCLR

این رجیستر جهت 0 کردن پین هایی که بصورت خروجی تعیین شده اند به کار می رود.

رجیستر FIOxPIN

این رجیستر دسترسی مستقیم به پورت های ورودی و خروجی را فراهم می کند. بطور مستقیم میتوانیم روی پین بنویسیم و بخوانیم.

رجیستر FIOxMASK

برای اینکه در پورت بعضی بیت ها را از عملیات نوشتن/خواندن یا تغییر حالت ورودی/ خروجی محافظت کنیم از این رجیستر استفاده میکنیم.

رجیسترهای PINSEL 0-10

جهت انتخاب نحوه عملکرد پین ها مورد استفاده قرار میگیرد. بعضی پین ها عملکرد های مختلفی مانند GPIO,UART,SPI,... دارند.

رجیسترهای PINMODE 0-9

جهت تنظیم مقاومت های PULL-UP و PULL-DOWN داخلی پورت ها

رجیستر های PINMODE_OP 0-4

برای انتخاب حالت Open Drain یا درین باز پین ها مورد استفاده قرار میگیرند.

حال به بررسی توابع کتابخانه LPC17XX_gpio.h

تابع GPIO_ReadValue

```
Uint32_t  GPIO_ReadValue(uint8_t portNum);
```

برای خواندن مقدار یک پورت استفاده می شود.

ورودی تابع port_Num شماره پورتی که میخواهید بخوانید.

خروجی تابع uint32_t وضعیت پورت

تابع GPIO_SetDir

void GPIO_SetDir(uint8_t portNum, uint32_t bitValue, uint8_t dir);

برای تعیین ورودی/خروجی پین های پورت مورد استفاده قرار میگیرد.

ورودی : portNum شماره پورت مورد نظر

ورودی :bitValue شماره پین های مورد نظر

ورودی : dir اگر 0 باشد ورودی و اگر 1 باشد خروجی

تابع GPIO_SetValue

void GPIO_SetValue(uint8_t portNum, uint32_t bitValue);

برای یک کردن پین هایی که خروجی تعیین شده اند.

ورودی : portNum شماره پورت مورد نظر

ورودی : bitValue شماره پین های مورد نظر

تابع GPIO_ClearValue

void GPIO_ClearValue(uint8_t portNum, uint32_t bitValue);

برای صفر نمودن پین های یک پورت استفاده میشود.

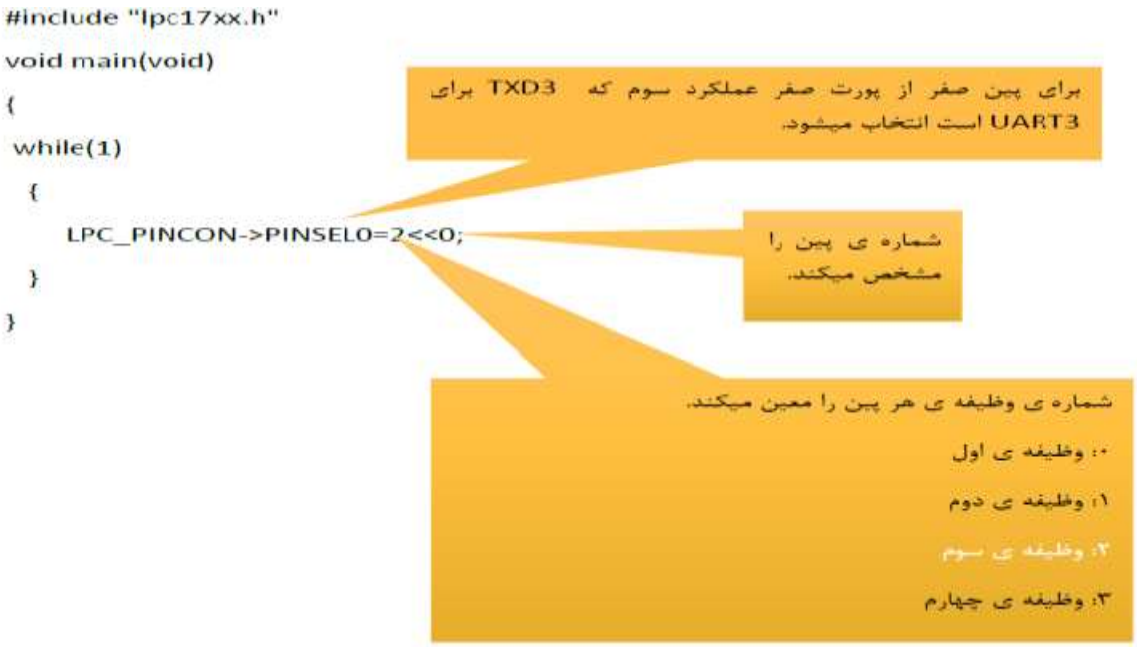
ورودی :portNum شماره پورت مورد نظر

ورودی :bitValue شماره پین های مورد نظر

جالب است بدانید که هر یک از پایه های میکروکنترلر، چند وظیفه ای هستند که با انتخاب یکی از آنها، بقیه غیرفعال خواهند شد. به عنوان مثال بر روی پین صفر از پورت صفر، 4 کارکرد با عناوین زیر مالتیپلکس شده اند که شما هر کاربرد که مورد نیازتان است را میتوانید فعال کنید.

- PORT 0.0 (General Purpose I/O)
- RD1 (CAN1 Receiver input)
- TXD3 (Transmitter output for UART3)
- SDA1 (I2C1 data I/O. Open-drain output)

روی هر پین حداکثر 4 وظیفه موجود است. پس برای تعیین وظیفه هر پین میکروکنترلر، به دو بیت نیاز داریم.



تصویر د) راهنمای کد فعالسازی وظیفه یک پین میکروکنترلر

اولین کدی که برای شروع این پروژه نیاز هست بنویسید کد دیود چشمک زن است. اما چگونه می توان این کار را انجام داد؟

به نحوی الگوریتم این کار به صورت زیر خواهد بود:

1. دیود روشن
2. یک ثانیه تاخیر
3. دیود خاموش
4. یک ثانیه تاخیر

5. برو به 1

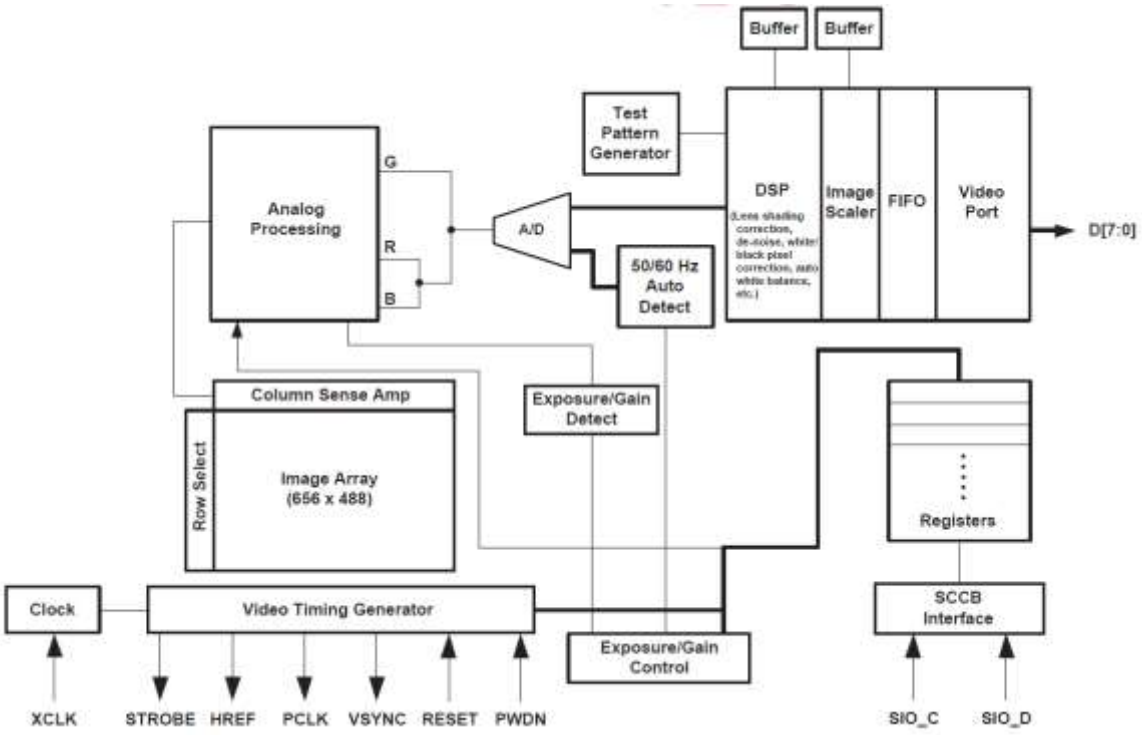
می دانیم برای روشن شدن دیود نیاز است که به پایه بلند دیود 1 منطقی و به پایه کوتاه دیود 0 منطقی وصل کنیم. خب به لطف آشنایی با توابع GPIO به راحتی با استفاده از تابع FIOxDIR پایه مورد نظرمان را تعریف و سپس به وسیله FIOxSET مقدار پایه را یک منطقی و با تابع FIOxCLR مقدار پایه را صفر منطقی می کنیم. اما این اتفاق اگر در یک حلقه While(1) صورت بگیرد به دلیل سرعت بالای اجرای پردازنده، چشم ما قادر به تششخیص چشمک زدن دیود نخواهد بود. پس لازم است تا بین یک و صفر کردن پایه تاخیری ایجاد کنیم. برای ایجاد تاخیر میتوانیم از هدرفایل delay.h و تابع delay_ms(miliSeconds) به صورت زیر استفاده کنیم.

```
while(1){
PORTA.0=1;
delay_ms(1000);
PORTA.0=0;
delay_ms(1000);
}
```

دوربین OV7670:

این دوربین یک دوربین استاندارد و کارآمد در زمینه رباتیک به شمار می رود که در اینجا قصد داریم در مورد آن توضیحاتی را مطرح کنیم.

این دوربین یک عکس با یک آرایه 488*656 از پیکسلها دارد که 480*640 از پیکسل ها فعال هستند. بالاترین سرعت ارسال فریم در این دوربین 30 فریم بر ثانیه است. لازم به ذکر است که این دوربین دارای 24 پایه است و بر پایه VGA طراحی شده است که دیاگرام بلاک مربوط به آن را در زیر مشاهده می کنیم:



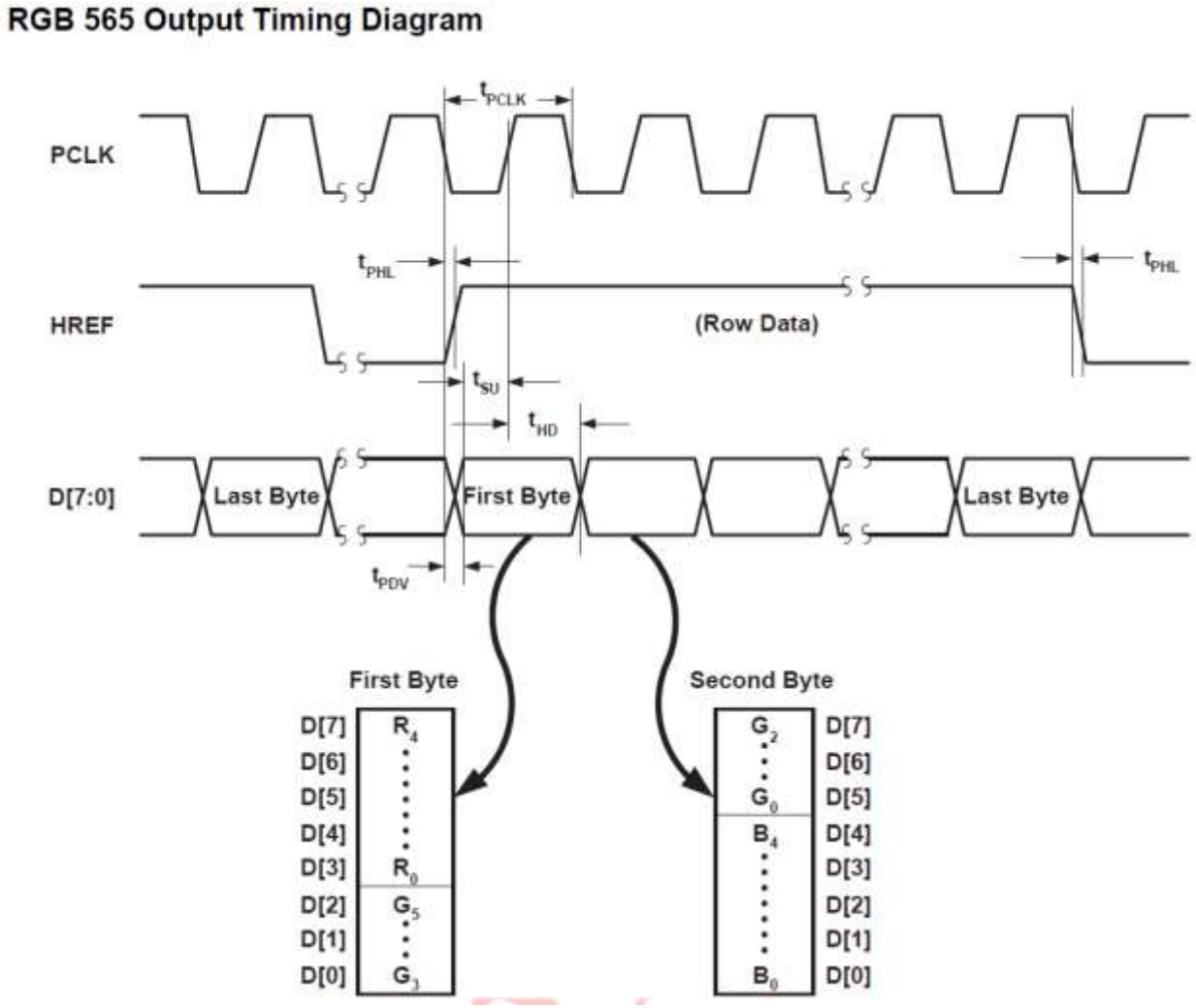
تصویر ه) دیاگرام بلاک OV7670

جالب است بدانید که این دوربین از چندین فرمت خروجی پشتیبانی می کند که در زیر به آنها اشاره شده است: (مجموعا 8 بیت برای ارسال خروجی)

- YUV/YCbCr 4:2:2
- RGB 565/555
- GRB 4:2:2
- Raw RGB Data

پایه هایی که مربوط به خروجی دوربین هستند پایه های D0 تا D7 هستند که D0 بیت کم ارزش اطلاعات و D7 بیت پر ارزش اطلاعات را تشکیل می دهد.

زمانی که این دوربین خروجی را در مود RGB به صورت 5-6-5 ارسال می کند، داده به این نحو ارسال می شوند که در اولین کلاک، 8 بیت در پایه های D0 تا D7، شامل 5 بیت رنگ قرمز، و سه بیت اول رنگ سبز است. در کلاک بعدی 8 بیت ظاهر شده در پایه های D0 تا D7 شامل 3 بیت باقی مانده از بیت های تشخیص رنگ سبز و 5 بیت رنگ آبی هستند.



تصویر و) خروجی پایه های دوربین OV7670

نتیجه این است که با هر دو کلاک خوردن، در مود RGB اطلاعات یک پیکسل به وسیله 16 بیت را در اختیار خواهیم داشت.

این ماژول دوربین همچنین یک پایه با نام XCLK دارد که به عنوان ورودی کلاک از سیستم استفاده می شود(هر زمان که نیاز بود دوربین یک Slave باشد و از یک Master کلاک دریافت کند می توان از این پایه جهت دادن کلاک استفاده نمود).

منابع

Enee.ir

ECA.ir

Melec.ir

binaryupdates.com

microlearning.ir

roboeq.ir

kavirelectronic.ir

arm-education.ir

t.me/armeducation

ضمائم

1: لینک دانلود کامپایلر Keil.v25

2: لینک دانلود پکیج مربوط به تراشه های Cortex_M3

3: لینک دانلود نرم افزار Flash_Magic