Formal Report

SVBot:
A Stereoscopic Robot

Greg Brown

University of Florida
EEL 5566 Intelligent Machine Design Lab
Instructors:
Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

Teaching Assistants:
Mike Pridgen
Ryan Stevens
Thomas Vermeer
Tim Martin
Devin Hughes

Table of Contents

Abstract

The SVBot is a targeting robot that uses stereoscopic imaging to detect the location of a target and calculate the relative distance of the target from the robot. To accomplish this, the robot reads data from two fixed cameras mounted on the front of the robot, implements stereoscopic calculations on the processed images, and computes the estimated position of the target. If the target is not in view, it will move around with obstacle avoidance until the target is found. To demonstrate that the robot has completed its distance calculation, it arcs a projectile of known parabolic trajectory into the target container.

Executive Summary

SVBot's purpose is to use stereoscopic calculations to determine the distance and angle a target's position relative to itself. It accomplishes this through the use of two Omnivision CMOS camera sensors (OV7620). This means that SVBot is also responsible for processing all raw image data coming from the sensor, as well as tracking the specific color of the target, all of which is implemented on-board (nothing outsourced to a nearby computer). The two cameras establish a depth perception, meaning a distance can be calculated.

SVBot has two IR sensors and one sonar sensor for obstacle avoidance, which it engages if the target LED is not found right away. The behavior of the target tracking relies heavily on averages and simple statistical analysis in order to ensure it has calculated an accurate distance. SVBot uses this information to orient itself in front of the target a set distance away in order for a "trap" to come down and capture the target. This target distance is arbitrarily set based on the length of the arm carrying the trap, and can actually be reprogrammed to be accurate anywhere within 1 meter.

The speed that SVBot can calculate the centroid of a color is comparable to a CMUCam 2. Although the frame rate is much less than its CMUCam counterpart, the centroid refresh rate is about the same: 2-3 per second. The distance algorithm is computationally trivial, so this is also the distance refresh rate. This results in robot that can calculate 5 distances and average them together in under 2 seconds, preventing any sluggish behavior during its target acquisition mode.

Introduction

Many target tracking robots have been built that use a camera to identify colors and identify a target. I wanted to bring this process a step further by building a robot that could not only identify a target with a camera, but also calculate its relative location from the target using stereoscopic imagery. This requires two cameras mounted adjacent to one another in a very precise manner, since the position of the cameras factors into the subsequent calculations.

The basis of stereoscopic imagery calculations lies in the disparity field. Disparity is the difference of the location of an object in each camera. When entire image pairs are processed into one stereoscopic image, a disparity matrix must be calculated. In this implementation, however, the robot is only interested in one target, so only one disparity is required. This significantly reduces the complexity of the stereoscopic algorithm. is represented by a LED globe on the ground. The robot looks for the target, and upon locating it, orients itself so that the target can be seen by both cameras. The Propeller board calculates the centroids of the two cameras, and the main board calculates the distance and angle from the robot to the target.

The scope of the project is to have a robot that can avoid obstacles, identify a target using a stereo pair of images, calculate the necessary distance equations, and perform some function that proves the robot has successfully calculated the distance.

Integrated System

The main controller for the SVBot is an AVR32 board designed by Devron Lee with basic robotics in mind. This board controls the servos via PWM, receives and interprets information from the proximity sensor array, and acts as a master to the Propeller board.

The Propeller board controls the two OV7620 cameras modules. It sends commands, configures, and receives pixel data from the cameras. It's actual function is to detect the centroid of the target color, much like the CMUCam, from each of the cameras. It then sends this information to the AVR32 board, where the distance calculations will be processed. The complete integrated system can be seen in the following block diagram (Figure 1).

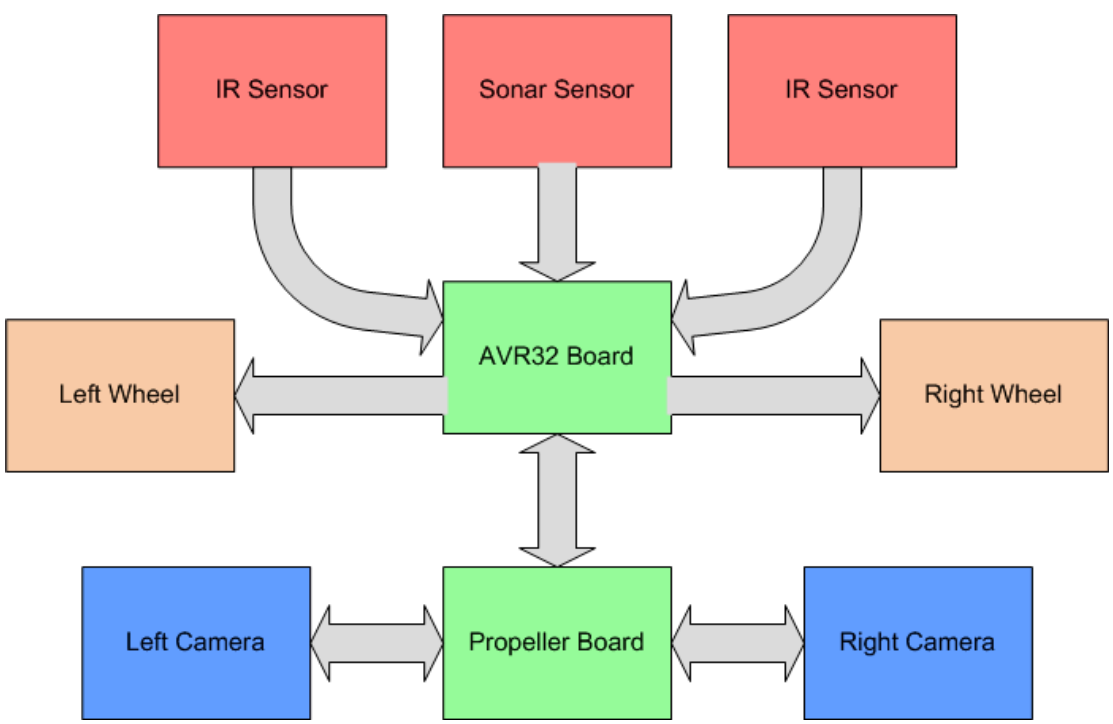


Figure 1 – System diagram

Mobile Platform

Since the robot's primary function is finding a stationary target, the platform is circular with two levels. The wheels are directly attached to continuously rotated servos on each side. For balance, there is a ball bearing in the back of the platform. The main controller board is placed on the bottom tier, while the Propeller board and cameras are mounted on the top tier. All proximity sensors are mounted to the underside of the top tier so as not to interfere with the camera.

Actuation

Two servos are controlling the wheels for movement. There is also a servo the swivels a thin arm with a "trap" at the end, which is placed over the target after completing its distance calculations.

Sensors

The robot is equipped with obstacle avoidance sensors that include two IR proximity sensors and one sonar proximity sensor. The IR sensors are mounted in the front corners of the robot and the sonar is mounted on the center of the front. This sensor array provides sufficient redundancy for obstacle avoidance. The IR sensors are angled such that a wall or other obstacle approaching from the side can be seen, and the sonar sensor has good coverage of any obstacle in front of the robot. There will also be a bump sensor in the front for extra redundancy.

The IR sensor and Sonar sensor are operated in the same way. They are both powered by 5V, and both output an analog voltage. The AVR32 board relies on an external ADC chip (AD7908) that is powered by 3.3V and a reference voltage of 2.5V. Therefore, the sensor voltage output must be scaled down to prevent possible damage to the ADC chip. To do this, a basic voltage divider circuit is placed between the sensor and ADC input. The circuit is comprised of two resistors in series: 300Ω and 150Ω. The resulting output is 2/3 that of the input, which drops 5V down to 3.3V. Although this weakens the output voltage of the sensors, it prevents electrical damage to the chip.

The special sensor is the OV7620 camera module. It is operated in 8-bit, RGB, progressive scan mode. The camera operates at 27MHz, which is equivalent to the pixel clock in 8-bit mode. In order to properly process the data, the pixel clock is divided by 16. The centroid algorithm has been programmed to find the color blue.

Behaviors

Overall

The SVBot has two stages in its operation: obstacle avoidance and target acquisition. During obstacle avoidance, the sensor array provides the stimulus for when an obstacle is seen. The AVR32 provides the reaction based on any obstacles detected, which is carried out by the servo wheels. More specifically, if an object is detected in the left IR sensor, the robot turns right, and if an object is detected in the right IR sensor, the robot turns left. If an object is detected by the Sonar sensor, the robot turns in a random direction.

When SVBot is first powered on, it enters into target acquisition mode. In this mode, it rotates in place attempting to find the blue LED target. If the target is not found after several rotations, SVBot enters into obstacle avoidance mode for a set amount of time before going back into target mode.

Once the target has been acquired, between 4 and 5 distance calculations are processed. The angle of the target from the robot is calculated as well. If all angles do not deviate past a certain threshold, SVBot assumes it has found the target. If the target is too far (greater than 1 meter away) it will approach the target by about 50cm. If the average distance is less than 1 meter away, the calculation is most likely accurate. SVBot will orient itself to face the LED and approach the target to within 50 cm. After a few more averaging distance, once SVBot is sure it is within range to drop the trap, it does so.

AVR32 Board

The AVR32 board is responsible for calculating all distances and angles, as well as managing the movement of the robot. It also does the averaging and deviation algorithms described above. The program running on this processor was written in C, and can be found in the appendix section of this report.

Propeller Board

The Propeller board is running the centroid algorithm. This algorithm is continuously running, and will only send centroid data when requested by the main board. One important aspect of the Propeller is that it processes the centroid on both cameras concurrently. This doubles the speed of a processor that would not have had parallel processing. The most interesting part of the program written for the Propeller is the centroid algorithm that was written completely in Spin Assembly. This assembly code processes the pixel data while collecting centroid data simultaneously, such that the image frame rate is equal to the centroid refresh rate.

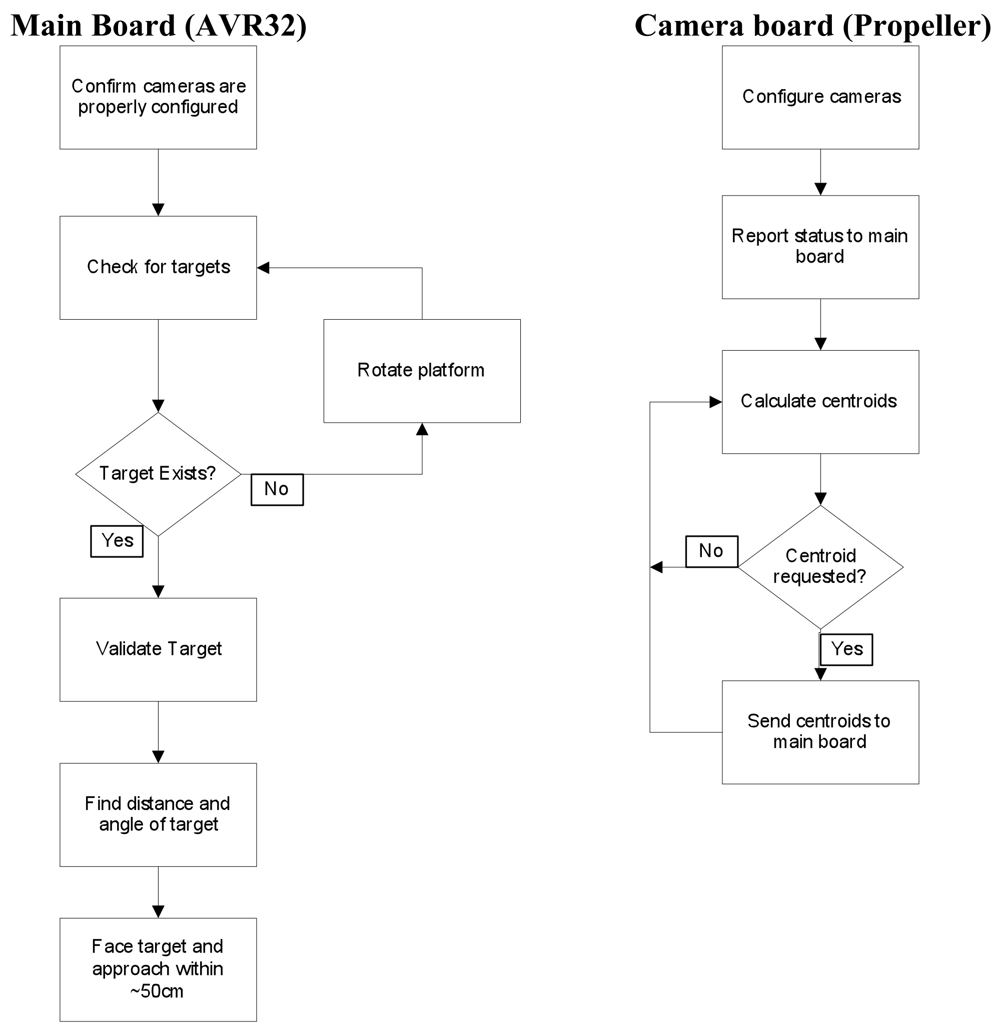
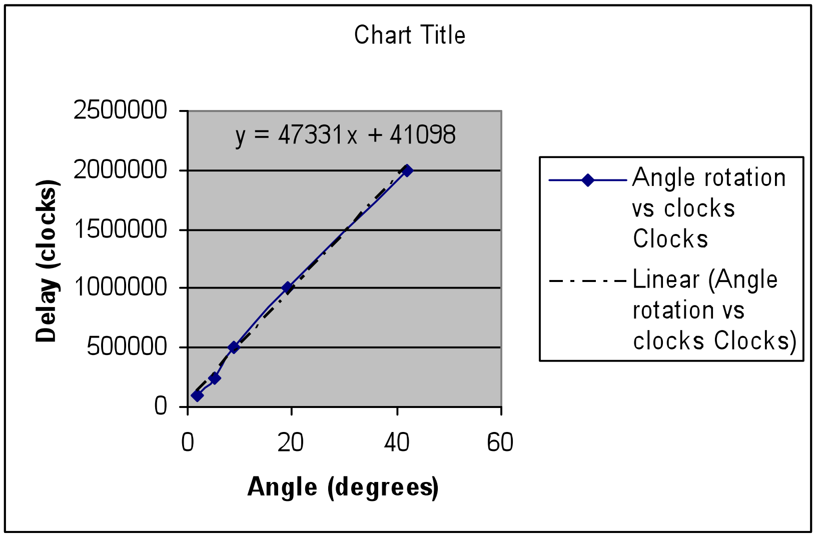


Figure 3 – Processor board flow charts

Experimental Layout and Results

The following experiments were run in order to better characterize certain aspects of the robot that were not immediately apparent. Figure 4 corresponds to data collected to figure out how long to rotate in order to rotate for a certain number of degrees, and how to move forward or backward to move a certain number of mm. This experiment was important since the robot had to orient itself toward the LED target, and then place itself at a certain distance away. These two experiments were key in converting numerical data to physical movement.



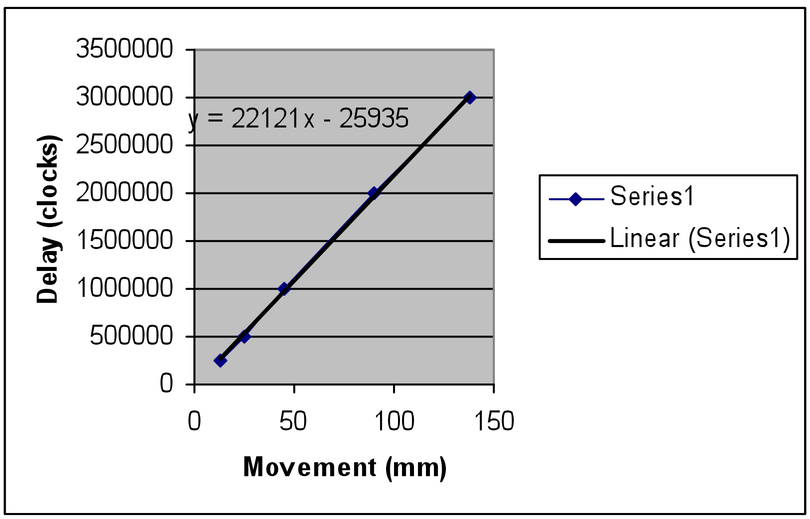


Figure 4 – Rotation and Movement Charts

Figure 5 (below) corresponds to the experiment that was carried out in order to determine the angular position of the target given the pixel column location of the centroid. Typically this would not have been necessary to acquire experimentally, but the active pixels on the CMOS camera sensor is not actually centered beneath the lens. This causes a severe angular offset in the image, which is severely undocumented in the datasheet. Therefore, an equation had to be experimentally calculated in order to calculate angular position based on the centroid data. The steep curve upward at the end of the pixel columns is due to the fish eye lens effect, since this point is the most offset point from the center of the lens.

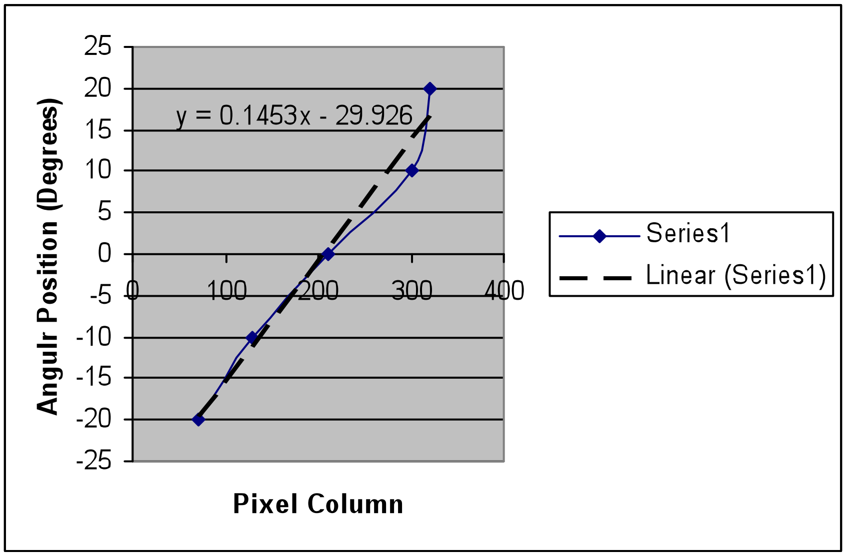


Figure 5 – Angular Position Chart

The image below is a result of requesting a frame from the camera sensor and transmitting it to the PC to verify its functionality. This posed an interesting problem, since there was no external memory. The Propeller can only store 30 rows of RGB pixels. So the image below is actually 16 picture superimposed onto each other 30 row at a time, then run through a rough demosaicing algorithm that I wrote myself. The reddish tint is most likely due to my demosaicing algorithm.

Conclusion and Lessons Learned

The performance of the robot is good considering how low-level the stereoscopic calculations are, and also considering that the system is communicating with two camera sensors, not a CMUCam. The distances are accurate to within centimeters (within the threshold distance), and the speed of the robot tracking the target isn't sluggish at all. As mentioned in the executive summary, it takes less than 2 seconds for the robot to calculate and average 5 distances and angles, allowing it to pretty continuously rotate until it finds the target.

It is unfortunate that the distance calculation becomes inaccurate past a meter. The original goal was to launch a projectile into a container based on how far away it was, but at such close distances this function was not viable. The trap that comes down was a nice, simple way to prove that the distance calculation was performed successfully. The accuracy of the trap depends entirely on how accurately the centroid was calculated. Any discrepancies in the camera feed can cause errors in the distance, which is why so much statistical analysis is employed.

There were quite a few lessons learned from this project. One was that Omnivision datasheets have horrible organization, and lack very pertinent information. I also learned that time management in key in building a robot where the main function system is undetermined. My decision to officially use the Omnivision sensors came only after I successfully received an image from one of them, which happened pretty late in the semester.

I am proud that I was able to get the centroid data I needed without the help of a third party board, but I regret the consequences of this. Once the system was to be embedded, I knew that I would be sacrificing a lot of cool stereoscopic functionality that would only be possible through the use of a computer. Had I used a wireless camera, the projectile launching might have actually been possible. In the future, I plan on working more with stereoscopic behaviors rather than embedded stereoscopic applications. If I started the project over however, I would still have used the Omnivision sensors. I believe that experience to be invaluable to my education with embedded platforms.

Documentation

[1] Tjandranegara, Edwin . "Distance Estimation Algorithm for Stereo Pair Images" ECE Technical Reports. Paper 64. http://docs.lib.purdue.edu/ecetr/64

[2] Mrovlje, Vrancic. “Distance measuring based on stereoscopic pictures”. 9th International PhD Workshop on Systems and Control: Young Generation Viewpoint. 2008. http://dsc.ijs.si/files/papers/S101%20Mrovlje.pdf

Appendix

Section 1: Propeller

Main Spin Code

```
{{ StereoProp.spin }}

CON
  'General Constants
  _CLKMODE = XTAL1 + PLL16X
  _XINFREQ = 5_000_000


  CAM1_Y0 = 0
  CAM1_Y1 = 1
  CAM1_Y2 = 2
  CAM1_Y3 = 3
  CAM1_Y4 = 4
  CAM1_Y5 = 5
  CAM1_Y6 = 6
  CAM1_Y7 = 7

  'PWDN = 8
  CAM1_RST = 8
  CAM1_HREF = 9
  CAM1_SDA = 10
  CAM1_SCL = 11
  'FODD = 12
  CAM1_VSYNC = 12
  CAM1_PIX_CLK = 13

  CAM2_Y0 = 16
  CAM2_Y1 = 17
  CAM2_Y2 = 18
  CAM2_Y3 = 19
  CAM2_Y4 = 20
  CAM2_Y5 = 21
  CAM2_Y6 = 22
  CAM2_Y7 = 23

  CAM2_RST = 24
  CAM2_HREF = 25
  CAM2_SDA = 26
  CAM2_SCL = 27
  CAM2_VSYNC = 28
  CAM2_PIX_CLK = 14

  PROP_TX = 30
  PROP_RX = 31

  RAWDATA = %00101100
  PCLK_DIVIDE = %00010001
  MODE_8BIT = %00100001
  PROGRESSIVE = %00100000
  QVGA = %00100100

  HORIZONTAL_END = %01111111
  VERTICAL_END = %00111000

  PROGRESSIVE_ONELINE = %10100000
  MANUAL_BRIGHTNESS = %10000001
  BRIGHTNESS = %10001000
  NO_AUTO_ADJUST = %00000000
  RAW_NO_GAMMA_WHITE_AUTO = %00001000
  RAW_NO_GAMMA_AUTO = %00001100
  RAW_NO_WHITE_AUTO = %00101000
```

```
OBJ
  UART : "FullDuplexSerial"
  AVR32 : "FullDuplexSerial"
  NUM : "Numbers"
  CAMERA_COMM : "Basic_I2C_Driver"
  'CAMERA2_COMM : "Basic_I2C_Driver"
  'CAMERA1_COMM : "pasm_i2c_driver"
  CAM1_TrackColor : "CAM1_Driver"
  CAM2_TrackColor : "CAM2_Driver"

VAR
  'CAM Variables
  byte  write_ack
  byte  verify
  'byte  ack1
  'byte  ack2
  byte  UARTcomm
  byte  AVR32comm
  byte  start

  long  CAM1_start
  long  CAM1_pix_stat
  long  CAM1_calc_total
  long  CAM1_Mx_total
  long  CAM1_all_count

  long  CAM2_start
  long  CAM2_pix_stat
  long  CAM2_calc_total
  long  CAM2_Mx_total
  long  CAM2_all_count

  byte  comm_start
  byte  rbyte
  long  CAM1_centroid
  long  CAM2_centroid

  'long  test_long[2]

PUB Main
  dira[CAM1_Y0..CAM1_Y7] := %00000000
  'dira[CAM1_PWDN]~~
  dira[CAM1_RST]~~
  'dira[CAM1_FODD]~
  dira[CAM1_HREF]~
  dira[CAM1_VSYNC]~
  dira[CAM1_PIX_CLK]~

  dira[CAM2_Y0..CAM2_Y7] := %00000000
  'dira[CAM2_PWDN]~~
  dira[CAM2_RST]~~
  'dira[CAM2_FODD]~
  dira[CAM2_HREF]~
  dira[CAM2_VSYNC]~
  dira[CAM2_PIX_CLK]~

  waitcnt(cnt + 10_000_000)
  UARTcomm := UART.start(PROP_RX, PROP_TX, 0, 115200)
  if UARTcomm <> 0
    'UART.str(STRING("StereoProp Ready"))
    'UART.tx(13)

    verify := verifyComm
      if verify == 1
        'UART.str(STRING("Cam1 Comm Error!"))
        'UART.tx(13)
        UART.tx($81)
        repeat
```

```
  elseif verify == 2
  if verify == 2
    'UART.str(STRING("Cam2 Comm Error!"))
    'UART.tx(13)
    UART.tx($82)
    repeat
  elseif verify == 3
    'UART.str(STRING("Ultimate Comm Error!"))
    'UART.tx(13)
    UART.tx($83)
    repeat
if CamConfig < 1
  'UART.str(STRING("Configuration Failed, StereoProp Stopped"))
  'UART.tx(13)
  UART.tx($85)
  repeat
UART.tx($80)
'UART.str(STRING("Camera Comm Established"))
'UART.tx(13)
'UART.tx(13)
waitcnt(300_000_000 + cnt)
start := 0

{AVR32comm := AVR32.start(PROP_RX, PROP_TX, 0, 115200)
  if AVR32comm <> 0
    AVR32.str(STRING("AVR32 Comm Established"))
    AVR32.tx(13)}
'repeat
start := 1
repeat while start == 0
  rbyte := AVR32.rx
  if rbyte == 1
    AVR32.tx(2)
    start := 1

'UART.str(STRING("Starting Cogs"))
'UART.tx(13)
CAM1_TrackColor.Start(@CAM1_start)
CAM2_TrackColor.Start(@CAM2_start)
'UART.str(STRING("Cogs Started"))
'UART.tx(13)

repeat
  'CAM1_start := 1
  'CAM2_start := 1

  repeat while (CAM1_pix_stat == 0) and (CAM2_pix_stat == 0)
  CAM1_centroid := CAM1_Mx_total/CAM1_calc_total
  CAM2_centroid := CAM2_Mx_total/CAM2_calc_total
  rbyte := UART.rx
  if rbyte == $02
    sendLong(CAM1_centroid)
    sendLong(CAM2_centroid)
    sendLong(CAM1_calc_total)
    sendLong(CAM2_calc_total)

  {UART.str(STRING("                Camera 1     Camera 2"))
  UART.tx(13)
  UART.str(STRING("      Mx Value:     "))
  UART.dec(CAM1_MX_total)
  UART.str(STRING("       "))
  UART.dec(CAM2_MX_total)
  UART.tx(13)
  UART.str(STRING("    Calc Value:     "))
  UART.dec(CAM1_calc_total)
  UART.str(STRING("       "))
  UART.dec(CAM2_calc_total)
  UART.tx(13)
  UART.str(STRING("Centroid Value:     "))
  UART.dec(CAM1_centroid)
```

```
    UART.str(STRING("       "))
    UART.dec(CAM2_centroid)
    UART.tx(13)
    UART.str(STRING("   Total Count:     "))
    UART.dec(CAM1_all_count)
    UART.str(STRING("       "))
    UART.dec(CAM2_all_count)
    UART.tx(13)  }
    CAM1_pix_stat := 0
    CAM2_pix_stat := 0


PUB CamConfig | status

  status := 1
  CAMERA_COMM.writeRegister(CAM1_SCL, $12, RAWDATA)         'raw data
  'CAMERA1_COMM.writeRegister(CAM1_SCL, $12, RAW_NO_WHITE_AUTO)         'raw data
  CAMERA_COMM.writeRegister(CAM1_SCL, $11, PCLK_DIVIDE)     'Pclk divide by 16
  'CAMERA1_COMM.writeRegister(CAM1_SCL, $13, MODE_8BIT)       '8 bit
  'CAMERA1_COMM.writeRegister(CAM1_SCL, $13, NO_AUTO_ADJUST)       '8 bit
  'CAMERA1_COMM.writeRegister(CAM1_SCL, $28, PROGRESSIVE)     'Progressive
  CAMERA_COMM.writeRegister(CAM1_SCL, $28, PROGRESSIVE_ONELINE)     'Progressive
  'CAMERA1_COMM.writeRegister(CAM1_SCL, $14, QVGA)            'QVGA
  'CAMERA1_COMM.writeRegister(CAM1_SCL, $18, HORIZONTAL_END)  'Horizontal window end
  'CAMERA1_COMM.writeRegister(CAM1_SCL, $1A, VERTICAL_END)    'Vertical window end
  CAMERA_COMM.writeRegister(CAM1_SCL, $2D, MANUAL_BRIGHTNESS)     'Manual brightness
  CAMERA_COMM.writeRegister(CAM1_SCL, $06, BRIGHTNESS)     'Change brightness
  'CAMERA1_COMM.writeRegister(CAM1_SCL, $16, %00000000)   'single frame (field drop OFF)


  write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $12)   'raw data
  if write_ack <> RAWDATA
    'UART.str(STRING("CAM1 Raw Data Register Error!"))
    'UART.tx(13)
    status := 0

  write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $11)   '
  if write_ack <> PCLK_DIVIDE
    'UART.str(STRING("CAM1 PCLK Register Error!"))
    'UART.tx(13)
    status := 0

  {write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $13)
  if write_ack <> MODE_8BIT
    UART.str(STRING("8-bit Mode Register Error!"))
    UART.tx(13)
    status := 0}

  write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $28)
  if write_ack <> PROGRESSIVE_ONELINE
    'UART.str(STRING("CAM1 Progressive Register Error!"))
    'UART.tx(13)
    status := 0

  write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $2D)
  if write_ack <> MANUAL_BRIGHTNESS
    'UART.str(STRING("CAM1 Manual Brightness Register Error!"))
    'UART.tx(13)
    status := 0

  write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $06)
  if write_ack <> BRIGHTNESS
    'UART.str(STRING("CAM1 Brightness Value Register Error!"))
    'UART.tx(13)
    status := 0

  {write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $13)
  if write_ack <> NO_AUTO_ADJUST
    UART.str(STRING("Auto Adjust Mode Mode Register Error!"))
```

```
   UART.tx(13)
   status := 0}
  {write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $14)
  if write_ack <> QVGA
   UART.str(STRING("QVGA Register Error!"))
   UART.tx(13)
   status := 0

  'write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $18)
  'if write_ack <> HORIZONTAL_END
  '  UART.str(STRING("Horizontal End Register Error!"))
  '  UART.tx(13)
  '  status := 0

  write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $1A)
  if write_ack <> VERTICAL_END
   UART.str(STRING("Vert End Register Error!"))
   UART.tx(13)
   status := 0
                                  }
  'write_ack := CAMERA_COMM.readRegister(CAM1_SCL, $16)
  'if write_ack <> $00
  '  UART.str(STRING("Single Frame Register Error!"))
  '  status := 0

  CAMERA_COMM.writeRegister(CAM2_SCL, $12, RAWDATA)         'raw data
  CAMERA_COMM.writeRegister(CAM2_SCL, $11, PCLK_DIVIDE)     'Pclk divide by 16
  CAMERA_COMM.writeRegister(CAM2_SCL, $28, PROGRESSIVE_ONELINE)     'Progressive
  CAMERA_COMM.writeRegister(CAM2_SCL, $2D, MANUAL_BRIGHTNESS)     'Manual brightness
  CAMERA_COMM.writeRegister(CAM2_SCL, $06, BRIGHTNESS)     'Change brightness

  write_ack := CAMERA_COMM.readRegister(CAM2_SCL, $12)   'raw data
  if write_ack <> RAWDATA
   'UART.str(STRING("CAM2 Raw Data Register Error!"))
   'UART.tx(13)
   status := 0

  write_ack := CAMERA_COMM.readRegister(CAM2_SCL, $11)   '
  if write_ack <> PCLK_DIVIDE
   'UART.str(STRING("CAM2 PCLK Register Error!"))
   'UART.tx(13)
   status := 0

  write_ack := CAMERA_COMM.readRegister(CAM2_SCL, $28)
  if write_ack <> PROGRESSIVE_ONELINE
   'UART.str(STRING("CAM2 Progressive Register Error!"))
   'UART.tx(13)
   status := 0

  write_ack := CAMERA_COMM.readRegister(CAM2_SCL, $2D)
  if write_ack <> MANUAL_BRIGHTNESS
   'UART.str(STRING("CAM2 Manual Brightness Register Error!"))
   'UART.tx(13)
   status := 0

  write_ack := CAMERA_COMM.readRegister(CAM2_SCL, $06)
  if write_ack <> BRIGHTNESS
   'UART.str(STRING("CAM2 Brightness Value Register Error!"))
   'UART.tx(13)
   status := 0


  return status
  'UART.hex(write_ack, 2)
  'UART.str(STRING("   (21)"))
 ' UART.tx(13)
  'waitcnt(2_000 + cnt)

' write_ack := CAMERA_COMM.writeRegister(CAM1_SCL, $13, %00100011)   '8 bit, single frame
```

```
PRI verifyComm | ack1, ack2, ack
 ack1 := CAMERA_COMM.readRegister(CAM1_SCL, $1D)
 ack2 := CAMERA_COMM.readRegister(CAM2_SCL, $1D)
 ack := 0
 if ack1 <> $A2
   ack += 1
 if  ack2 <> $A2
   ack += 2
 'waitcnt(2_000 + cnt)
 'verify := CAMERA1_COMM.write7(SCL, $43)
 'CAMERA1_COMM.start(SCL)
 'ack1 := CAMERA1_COMM.write(SCL, $42)
 return ack


PRI sendFrame(pixel_address) | count, current_pixel
 count := 1
 repeat while count < 4801
   current_pixel := long[pixel_address][count]
   'current_pixel<-=8
   'UART.tx((current_pixel<-=8) & $FF)
   'UART.tx((current_pixel<-=8) & $FF)
   'UART.tx((current_pixel<-=8) & $FF)
   UART.tx((current_pixel) & $FF)
   UART.tx((current_pixel>>8) & $FF)
   UART.tx((current_pixel>>16) & $FF)
   UART.tx((current_pixel>>24) & $FF)
   count += 1

PRI testSend(test_address) | count,data
 count := 1
 repeat while count < 3
   data := long[test_address][count]
   repeat 4
     UART.tx((data<-=8) & $00FF)
   count += 1

PRI sendLong(long_value)
 UART.tx(long_value & $00FF)
 repeat 3
   UART.tx((long_value->=8) & $00FF)
```

Camera driver and centroid algorithm (assembly)

```
'' OV7620 Camera Driver
'' Written by Gregory Brown

'' This is the assembly code for tracking a color using the OV7620 camera
'' module.  This code is specific to One-line, RGB mode

PUB Start(addr_start) | ack

  ack := cognew(@CAM_FUNC, addr_start)
  return ack

DAT

                org     0
CAM_FUNC                mov     START_ADDR, par
                mov     STAT_ADDR, START_ADDR
                add     STAT_ADDR, #4
                mov     CALCS_ADDR, STAT_ADDR
                add     CALCS_ADDR, #4
                mov     MX_ADDR, CALCS_ADDR
                add     MX_ADDR, #4
                mov     TOTAL_CNT_ADDR, MX_ADDR
                add     TOTAL_CNT_ADDR, #4


START_loop              'rdlong  FUNC_START, START_ADDR
               ' cmp     FUNC_START, #1  wz
      'if_nz        jmp     #START_loop
                mov     PIXEL_CNT, #1
                mov     ALGO_LOC, #1
                mov     PIXEL_LOC_ADDR, #PIXEL_LOC_START
                movd    PIXEL_LOC_WRITE, PIXEL_LOC_ADDR
                'nop
                movs    PIXEL_LOC_READ, PIXEL_LOC_ADDR
                'nop
                movd    PIXEL_LOC_ZERO, PIXEL_LOC_ADDR
ZERO_loop               cmp     ZERO_CNT, #10  wz
      if_z        jmp     #end_ZERO_loop
PIXEL_LOC_ZERO          mov     0-0, #0
                add     PIXEL_LOC_ADDR, #1
                movd    PIXEL_LOC_ZERO, PIXEL_LOC_ADDR
                add     ZERO_CNT, #1
                jmp     #ZERO_loop

end_ZERO_loop           mov     PIXEL_LOC_ADDR, #PIXEL_LOC_START
                mov     ZERO_CNT, #0

VSYNC_loop              test    VSYNC_PIN, ina  wz       'wait VSYNC
      if_z        jmp     #VSYNC_loop

                'mov     VSYNC_VAL, #1
                'jmp     #PIX_start

PIX_start               test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go high
      if_z        jmp     #PIX_start
                'mov     PIX_CLK_VAL, #1
                test    HREF_PIN, ina  wz       'make sure HREF still high
      if_z        jmp     #PIX_start
                jmp     #PIX_loop1


                 '****FIRST PIXEL***
PIX_loop1_high          test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go high
      if_z        jmp     #PIX_loop1_high
```

```
PIX_loop1           add     TOTAL_CNT, #1
                mov     PIXEL_TEMP, ina             ''read pixel, shift
                and     PIXEL_TEMP, PIX_DATA_PINS
                cmp     ROW, #0  wz
        if_nz       jmp     #odd_row1
                cmp     PIXEL_TEMP, BLUE_MIN  wc
        if_c        jmp     #not_blue
                mov     BLUE_TRUE, #1
not_blue            add     CLK_CNT, #1
                jmp     #loop1

odd_row1            add     CLK_CNT, #1

loop1               test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go low
        if_nz       jmp     #loop1
                'jmp    #PIX_loop

                '****SECOND PIXEL*****
PIX_loop2           test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go high
        if_z        jmp     #PIX_loop2

                mov     PIXEL_TEMP, ina             ''read pixel, shift
                and     PIXEL_TEMP, PIX_DATA_PINS

                add     TOTAL_CNT, #1
                cmp     ROW, #0  wz
        if_nz       jmp     #odd_row2

                cmp     BLUE_TRUE, #1  wz
        if_nz       jmp     #not_ok
                cmp     PIXEL_TEMP, GREEN_MAX  wc
        if_nc       jmp     #not_ok

PIXEL_LOC_WRITE     or      0-0, ALGO_LOC           'pixel algorithm
                'add    PIXEL_CNT, #1
not_ok              shl     ALGO_LOC, #1
                add     SHIFT_CNT, #1
                add     CLK_CNT, #1
                mov     BLUE_TRUE, #0
                cmp     SHIFT_CNT, #32  wz
        if_nz       jmp     #loop2
                mov     SHIFT_CNT, #0
                mov     ALGO_LOC, #1
                add     PIXEL_LOC_ADDR, #1
                movd    PIXEL_LOC_WRITE, PIXEL_LOC_ADDR
                jmp     #chk_end_row


odd_row2            add     CLK_CNT, #1

PIXEL_LOC_READ      test    ALGO_LOC, 0-0  wz
        if_z        jmp     #do_nothing

                cmp     PIXEL_TEMP, RED_MAX  wc
        if_nc       jmp     #do_nothing

                add     MX, PIXEL_CNT
                add     CALCS, #1
do_nothing          shl     ALGO_LOC, #1
                add     SHIFT_CNT, #1
                add     PIXEL_CNT, #1               'pixel algorithm
                'add    CLK_CNT, #1
                cmp     SHIFT_CNT, #32  wz
        if_nz       jmp     #loop2
                mov     SHIFT_CNT, #0
                mov     ALGO_LOC, #1
                add     PIXEL_LOC_ADDR, #1
                movs    PIXEL_LOC_READ, PIXEL_LOC_ADDR
```

```
chk_end_row            cmp     CLK_CNT, CLK_MAX  wz
       if_z          jmp     #HREF_ZERO

loop2                  test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go low
       if_nz         jmp     #loop2

                    jmp     #PIX_loop1_high

       '               ****END OF HREF*****
'PIX_zero
HREF_zero              test    HREF_PIN, ina  wz       'wait HREF low
       if_nz         jmp     #HREF_zero
                    'mov     PIXEL_CNT, #1
                    mov     CLK_CNT, #0
                    mov     PIXEL_CNT, #1
                    mov     SHIFT_CNT, #0
                    mov     ALGO_LOC, #1
                    mov     PIXEL_LOC_ADDR, #PIXEL_LOC_START
                    movd    PIXEL_LOC_WRITE, PIXEL_LOC_ADDR
                    nop
                    movs    PIXEL_LOC_READ, PIXEL_LOC_ADDR
                    cmp     ROW, #0  wz
            if_z     jmp     #row_to_odd
                    mov     ROW, #0
                    jmp     #ahead1
row_to_odd             mov     ROW, #1

ahead1                 add     HORIZ_CNT, #1
                    cmp     HORIZ_CNT, #480  wz
       if_z          jmp     #END_loop


                    jmp     #PIX_start

'END_loop              wrlong  CLK_CNT, CLK_CNT_ADDR
END_loop               mov     STAT, #1
                    mov     HORIZ_CNT, #0
                    wrlong  START_CLR, START_ADDR
                    wrlong  MX, MX_ADDR
                    wrlong  CALCS, CALCS_ADDR
                    wrlong  TOTAL_CNT, TOTAL_CNT_ADDR
                    wrlong  STAT, STAT_ADDR
                    mov     CALCS, #0
                    mov     TOTAL_CNT, #0
                    mov     PIXEL_CNT, #0
                    mov     MX, #0
                    mov     ROW, #0
                    mov     FUNC_START, #0
                    jmp     #START_loop


HREF_PIN               long |< 9
VSYNC_PIN              long |< 12
PIX_CLK_PIN            long |< 13
PIX_DATA_PINS          long $FF
CLK_MAX                long 640
BLUE_MIN               long    200
RED_MAX                long    90
GREEN_MAX              long    90
'x200                  long  $200
ZERO_CNT               long    0
CLK_CNT                long    0
HORIZ_CNT              long    0
START_CLR              long    0
STAT                   long    0
SHIFT_CNT              long    0
'PIXEL_RDY              long    0
'PIXEL                  long    0
CALCS                  long    0
PIXEL_TEMP             long    0
```

```
PIXEL_CNT               long    1
FUNC_START              long    0
ALGO_LOC                long    0
ROW                     long    0
BLUE_TRUE               long    0
MX                      long    0
TOTAL_CNT               long    0

START_ADDR              res     1
STAT_ADDR               res     1
CALCS_ADDR              res     1
MX_ADDR                 res     1
TOTAL_CNT_ADDR          res     1
PIXEL_LOC_ADDR          res     1
PIXEL_LOC_START         res     10
```

Picture Send code (assembly)

```
PRI sendFrame(pixel_address) | count, current_pixel
  count := 1
  repeat while count < 4801
    current_pixel := long[pixel_address][count]
    'current_pixel<-=8
    'UART.tx((current_pixel<-=8) & $FF)
    'UART.tx((current_pixel<-=8) & $FF)
    'UART.tx((current_pixel<-=8) & $FF)
    UART.tx((current_pixel) & $FF)
    UART.tx((current_pixel>>8) & $FF)
    UART.tx((current_pixel>>16) & $FF)
    UART.tx((current_pixel>>24) & $FF)
    count += 1

PRI testSend(test_address) | count,data
  count := 1
  repeat while count < 3
    data := long[test_address][count]
    repeat 4
      UART.tx((data<-=8) & $00FF)
    count += 1

DAT

              org     0
CAM_FUNC              mov     START_ADDR, par
              mov     STAT_ADDR, START_ADDR
              add     STAT_ADDR, #4
              mov     HREF_START_ADDR, STAT_ADDR
              add     HREF_START_ADDR, #4
              mov     PIXEL_ADDR, HREF_START_ADDR
              add     PIXEL_ADDR, #4


START_loop            rdlong  FUNC_START, START_ADDR
              cmp     FUNC_START, #1  wz
      if_nz       jmp     #START_loop
              rdlong  HREF_START, HREF_START_ADDR

VSYNC_loop            test    VSYNC_PIN, ina  wz       'wait VSYNC
      if_z        jmp     #VSYNC_loop

              mov     VSYNC_VAL, #1
              'jmp    #PIX_start




{VSYNC_loop1          test    VSYNC_PIN, ina  wz
      if_z        jmp     #VSYNC_zero
              cmp     VSYNC_VAL, #0  wz
      if_z        jmp     #END_loop
              jmp     #HREF1
VSYNC_zero            mov     VSYNC_VAL, #0
HREF1                 test    HREF_PIN, ina  wz       'wait HREF high
      if_z        jmp     #HREF1
              add     CLK_CNT, #1
      '       cmp     CLK_CNT, #10  wz
     ' if_z         mov     VSYNC_VAL, #0

HREF2                 test    HREF_PIN, ina  wz       'wait HREF low
      if_nz       jmp     #HREF2
              jmp     #VSYNC_loop1 }

HREF_skip             cmp     HREF_START, #0  wz
```

```
        if_z          jmp     #PIX_start
HREF_high             test    HREF_PIN, ina  wz       'make sure HREF still high
        if_z          jmp     #HREF_high
HREF_low              test    HREF_PIN, ina  wz       'wait HREF low
        if_nz         jmp     #HREF_low
                      sub     HREF_START, #1
                      jmp     #HREF_skip


PIX_start             test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go high
        if_z          jmp     #PIX_start
                      'mov    PIX_CLK_VAL, #1
                      test    HREF_PIN, ina  wz       'make sure HREF still high
        if_z          jmp     #PIX_start
                      jmp     #PIX_loop1


                      '****FIRST PIXEL***
PIX_loop1_high        test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go high
        if_z          jmp     #PIX_loop1_high


PIX_loop1             'add    PIXEL_SKIP, #1             'pixel algorithm
                      'cmp    PIXEL_SKIP, #2  wz
        'if_nz        jmp     #ahead1
                      mov     PIXEL_TEMP, ina            ''read pixel, shift
                      and     PIXEL_TEMP, PIX_DATA_PINS
                      or      PIXEL, PIXEL_TEMP

                      'cmp    PIXEL_RDY, #1  wz       ''check to see if pixel is ready,
        'if_nz        jmp     #loop1
                      'wrlong  PIXEL,PIXEL_ADDR       ''then write to main memory
ahead1                add     CLK_CNT, #1
loop1                 test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go low
        if_nz         jmp     #loop1
                      'jmp    #PIX_loop

                      '****SECOND PIXEL*****
PIX_loop2             test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go high
        if_z          jmp     #PIX_loop2


                      'cmp    PIXEL_SKIP, #2  wz
        'if_nz        jmp     #ahead2
                      mov     PIXEL_TEMP, ina            ''read pixel, shift
                      and     PIXEL_TEMP, PIX_DATA_PINS
                      shl     PIXEL_TEMP, #8
                      or      PIXEL, PIXEL_TEMP
                      'add    SHIFT_CNT, #8
                      'add    PIXEL_CNT, #1

                      'mov    PIX_CLK_VAL, #1
ahead2                add     CLK_CNT, #1            'pixel algorithm
                      'cmp    PIXEL_RDY, #1  wz
        'if_nz        jmp     #ahead1
                      'add    PIXEL_ADDR, #4
                      'mov    PIXEL_RDY, #0
                      'mov    PIXEL, #0


                      'shl    PIXEL_TEMP, SHIFT_CNT
                      'or     PIXEL, PIXEL_TEMP
                      ' add   SHIFT_CNT, #8
:loop                 test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go low
        if_nz         jmp     #:loop

        '             ****THIRD PIXEL*****
PIX_loop3             test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go high
        if_z          jmp     #PIX_loop3
```

```
                'cmp     PIXEL_SKIP, #2  wz
    'if_nz          jmp     #ahead3
                mov     PIXEL_TEMP, ina             ''read pixel, shift
                and     PIXEL_TEMP, PIX_DATA_PINS
                shl     PIXEL_TEMP, #16
                or      PIXEL, PIXEL_TEMP

                'mov     PIX_CLK_VAL, #1
{                add     CLK_CNT, #1            'pixel algorithm
                cmp     PIXEL_CNT, #4  wz
    if_nz          jmp     #loop3
                wrlong  PIXEL, PIXEL_ADDR
                add     PIXEL_ADDR, #4
                mov     PIXEL_CNT, #0      }

loop3               test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go low
    if_nz          jmp     #loop3

    '               ****FOURTH PIXEL*****
PIX_loop4           test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go high
    if_z           jmp     #PIX_loop4
                'mov     PIX_CLK_VAL, #1
                mov     PIXEL_TEMP, ina             ''read pixel, shift
                and     PIXEL_TEMP, PIX_DATA_PINS
                shl     PIXEL_TEMP, #24
                or      PIXEL, PIXEL_TEMP
                wrlong  PIXEL, PIXEL_ADDR
                add     PIXEL_ADDR, #4
                add     PIXEL_CNT, #1
                mov     PIXEL, #0
                'add     CLK_CNT, #1
                cmp     PIXEL_CNT, #160  wz
    if_z           jmp     #HREF_zero
                'add     CLK_CNT, #1            'pixel algorithm
                'cmp     SHIFT_CNT, #32  wc      ''check if 4th pixel, set RDY flag
    'if_c           jmp     #ahead2
                'mov     PIXEL_RDY, #1
                'mov     SHIFT_CNT, #0

                'cmp     CLK_CNT, CLK_MAX  wz       ''check if end of row
    'if_z           jmp     #PIX_zero
:loop               test    PIX_CLK_PIN, ina  wz    'wait pixel clock to go low
    if_nz          jmp     #:loop
                jmp     #PIX_loop1_high

'PIX_zero
HREF_zero           test    HREF_PIN, ina  wz       'wait HREF low
    if_nz          jmp     #HREF_zero
                mov     PIXEL_CNT, #0

{HREF_skip1           test    HREF_PIN, ina  wz       'wait HREF low
    if_nz          jmp     #HREF_skip1
HREF_skip2           test    HREF_PIN, ina  wz       'make sure HREF still high
    if_z           jmp     #HREF_skip2
HREF_skip3           test    HREF_PIN, ina  wz       'wait HREF low
    if_nz          jmp     #HREF_skip3
HREF_skip4           test    HREF_PIN, ina  wz       'make sure HREF still high
    if_z           jmp     #HREF_skip4
HREF_skip5           test    HREF_PIN, ina  wz       'wait HREF low
    if_nz          jmp     #HREF_skip5
HREF_skip6           test    HREF_PIN, ina  wz       'make sure HREF still high
    if_z           jmp     #HREF_skip6
HREF_skip7           test    HREF_PIN, ina  wz       'wait HREF low
    if_nz          jmp     #HREF_skip7
HREF_skip8           test    HREF_PIN, ina  wz       'make sure HREF still high
    if_z           jmp     #HREF_skip8
HREF_skip9           test    HREF_PIN, ina  wz       'wait HREF low
    if_nz          jmp     #HREF_skip9    }

                add     HORIZ_CNT, #1
```

```
                cmp     HORIZ_CNT, #30  wz
        if_z     jmp     #END_loop


                jmp     #PIX_start

'END_loop        wrlong  CLK_CNT, CLK_CNT_ADDR
END_loop         mov     STAT, #1
                mov     HORIZ_CNT, #0
                mov     PIXEL_ADDR, HREF_START_ADDR
                add     PIXEL_ADDR, #4
                mov     PIXEL_CNT, #0
                mov     CLK_CNT, #0
                wrlong  START_CLR, START_ADDR
                wrlong  STAT, STAT_ADDR

                jmp     #START_loop


HREF_PIN            long |< 9
VSYNC_PIN           long |< 12
PIX_CLK_PIN         long |< 13
PIX_DATA_PINS       long $FF
CLK_MAX             long 640
CLK_CNT             long    0
HORIZ_CNT           long    0
START_CLR           long    0
STAT                long    0
HREF_START          long    0
HREF_MAX            long    0
SHIFT_CNT           long    0
PIXEL_RDY           long    0
PIXEL               long    0
PIXEL_TEMP          long    0
PIXEL_CNT           long    0
FUNC_START          long    0
PIXEL_SKIP          long    0

START_ADDR          res     1
STAT_ADDR           res     1
CLK_CNT_ADDR        res     1
PIXEL_ADDR          res     1
VSYNC_VAL           res     1
PIX_CLK_VAL         res     1
PIX_DATA_VAL        res     1
HREF_START_ADDR     res     1
```

AVR32 Code

Spctr.c

```
#include "spctr.h"
/*My Sensors*/
#include "spctr.propeller.h"

/* UTILS */
static void software_delay(void)
{
	volatile int i;
	for(i=0; i<750000; i++){}
}

void custom_delay(int clocks)
{
	int i;
	for(i=0; i<clocks; i++){}
}
/* END UTILS */

//left = 21
//right = 20

void soft_left(void){
	int i;
	spctr_usartWriteLine("Soft left.. \r");
	spctr_pwmUpdateDuty(2,45); //turn left
	spctr_pwmUpdateDuty(3,45);
	for(i=0; i<3; i++){
		software_delay();
	}
	spctr_pwmUpdateDuty(2,45); //straight
	spctr_pwmUpdateDuty(3,100);
}

void soft_right(void){
	int i;
	spctr_usartWriteLine("Soft right.. \r");
	spctr_pwmUpdateDuty(2,100); //turn right
	spctr_pwmUpdateDuty(3,100);
	for(i=0; i<3; i++){
		software_delay();
	}
	spctr_pwmUpdateDuty(2,45); //straight
	spctr_pwmUpdateDuty(3,100);
}

void hard_left(void){
	int i;
	spctr_usartWriteLine("Hard left.. \r");
	spctr_pwmUpdateDuty(2,45); //turn left
	spctr_pwmUpdateDuty(3,45);
	for(i=0; i<6; i++){
		software_delay();
	}
	spctr_pwmUpdateDuty(2,45); //straight
	spctr_pwmUpdateDuty(3,100);
}

void hard_right(void){
	int i;
	spctr_usartWriteLine("Hard right.. \r");
	spctr_pwmUpdateDuty(2,100); //turn right
	spctr_pwmUpdateDuty(3,100);
	for(i=0; i<6; i++){
		software_delay();
```

```
	}
	spctr_pwmUpdateDuty(2,45); //straight
	spctr_pwmUpdateDuty(3,100);
}

void turn_left(void){
	int i;
	spctr_usartWriteLine("Soft left.. \r");
	spctr_pwmUpdateDuty(2,45); //turn left
	spctr_pwmUpdateDuty(3,45);
	for(i=0; i<3; i++){
		software_delay();
	}
	spctr_pwmUpdateDuty(2,75); //stand still
	spctr_pwmUpdateDuty(3,75);
}

void turn_right(void){
	int i;
	spctr_usartWriteLine("Soft right.. \r");
	spctr_pwmUpdateDuty(2,100); //turn right
	spctr_pwmUpdateDuty(3,100);
	for(i=0; i<3; i++){
		software_delay();
	}
	spctr_pwmUpdateDuty(2,75); //stand still
	spctr_pwmUpdateDuty(3,75);
}

void move_forward(void){
	spctr_pwmUpdateDuty(2,45); //straight
	spctr_pwmUpdateDuty(3,100);
}

void move_backward(void){
	spctr_pwmUpdateDuty(2,100); //backward
	spctr_pwmUpdateDuty(3,45);
}

void move_right(void){
	spctr_pwmUpdateDuty(2,100); //turn right
	spctr_pwmUpdateDuty(3,100);
}

void move_left(void){
	spctr_pwmUpdateDuty(2,45); //turn left
	spctr_pwmUpdateDuty(3,45);
}

void brake(void){
	spctr_pwmUpdateDuty(2,75); //still
	spctr_pwmUpdateDuty(3,75);
}

int distance_calc(centroid1, centroid2){
	int total_dist;
	//total_pixels = 320
	//angle = 2*arctan(4.86/(2*6)) = 0.77
	//half_angle = angle/2 = 0.385
	//cam_dist = 100
	//2*tan(half_angle) = 0.81
	total_dist = (100*320)/(0.81*abs(centroid1-centroid2));
	return total_dist;
}

void capture_target(float dist, float angle) {
	int rotate_cnt;
	int travel_cnt;
	char temp[50];
	if (angle < 0) {
```

```
			rotate_cnt = (long)(abs(angle) * 50000 + 40000);
			sprintf(temp, "rotate count: %d", rotate_cnt);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			move_left();
			custom_delay(rotate_cnt);
			brake();
			software_delay();
		}
		else if(angle > 0) {
			rotate_cnt = (long)(angle * 50000 + 40000);
			sprintf(temp, "rotate count: %d", rotate_cnt);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			move_right();
			custom_delay(rotate_cnt);
			brake();
			software_delay();
		}
		if( dist > 1000 ) {
			travel_cnt = (long) ((500) * 22500);
			sprintf(temp, "travel count: %d", travel_cnt);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			move_forward();
			custom_delay(travel_cnt);
			brake();
		}
		else if( (dist-480) > 0){
			travel_cnt = (long) ((dist - 480) * 22500);
			sprintf(temp, "travel count: %d", travel_cnt);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			move_forward();
			custom_delay(travel_cnt);
			brake();
		}
		else if( (dist-480) < 0){
			travel_cnt = (long) ( (480-dist) * 22500);
			sprintf(temp, "travel count: %d", travel_cnt);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			move_backward();
			custom_delay(travel_cnt);
			brake();
		}

}
/* MAIN */
int main(void)
{
		char spctrIDString[8];
		volatile int i = 0;
		volatile int j = 0;
		//volatile int rep = 0;
		unsigned short buffer[9];
		unsigned short IR_Left;
		unsigned short IR_Right;
		unsigned short sonar;
		int channel;
		unsigned short test;
		//int distance;
		float avg_centroid;
		float angle;
		char temp[50];
		char prop_reply = -1;
		int centroid_1;
		int centroid_2;
		int calc_total_1;
		int calc_total_2;
```

```
int exit_loop = 0;
int distances[4];
float angles[4];
int avg_dist;
//int dist;
int avg_centroid_1;
int avg_centroid_2;
float avg_angle;
int dist_diff;
float angle_diff;
float temp_angle;
int LED_search;
int dist_try;
int too_far;
int prev_dist;
int repeat;
int turns;

// Configure SPCTR board
spctr_boardInit();

// Assign SPCTR ID
spctrID = 12;

// Setup PWM channels
spctr_pwmConfCtrl();

for(i=0; i<5; i++)
{
		if(i == 0){
				spctr_pwmInit(i, 95);
		}
		else {
				spctr_pwmInit(i, 75); // 7.5% Duty cycle, center
		}
}
for(i=0; i<9; i++)
{
		buffer[i] = 0;
}

gpio_set_gpio_pin(SPCTR_GPIO5);
gpio_clr_gpio_pin(SPCTR_GPIO6);
gpio_set_gpio_pin(SPCTR_GPIO7);
gpio_clr_gpio_pin(SPCTR_GPIO8);

spctr_usartInit();
spctr_PropellerInit();

sprintf(spctrIDString, "%d", spctrID);
spctr_usartHR();
spctr_usartWriteLine("SPCTR ");
spctr_usartWriteLine(spctrIDString);
spctr_usartWriteLine(" online.\r");
spctr_usartHR();
software_delay();



//spctr_pwmUpdateDuty(0,50);

while (exit_loop == 0) {
		prop_reply = spctr_Propeller_readChar();
		if (prop_reply == USART_FAILURE) {
				spctr_usartWriteLine("USART Failure");
				//while(1){}
		}
		else if (prop_reply == 0x81) {
				spctr_usartWriteLine("Camera 1 Fail");
				while(1){}
```

```
		}
		else if (prop_reply == 0x82) {
			spctr_usartWriteLine("Camera 2 Fail");
			while(1){}
		}
		else if (prop_reply == 0x83) {
			spctr_usartWriteLine("Both Cameras Fail");
			while(1){}
		}
		else if (prop_reply == 0x85) {
			spctr_usartWriteLine("Configuration Fail");
			while(1){}
		}
		else if (prop_reply == 0x80) {
			spctr_usartWriteLine("Camera Success \r");
			exit_loop = 1;
		}
		else {
			//spctr_usartWriteLine("ACK Error");
			software_delay();
			software_delay();
			//while(1){}
		}
	}

	/*while(1){
		spctr_Propeller_sendByte(0x02);
		centroid_1 = spctr_Propeller_readInt();
		centroid_2 = spctr_Propeller_readInt();
		calc_total_1 = spctr_Propeller_readInt();
		calc_total_2 = spctr_Propeller_readInt();
		distance = distance_calc(centroid_1, centroid_2);
		sprintf(temp, "Centroid 1: %d	Calc total 1: %d", centroid_1, calc_total_1);
		spctr_usartWriteLine(temp);
		spctr_usartWriteLine("\r");
		sprintf(temp, "Centroid 2: %d	Calc total 2: %d", centroid_2, calc_total_2);
		spctr_usartWriteLine(temp);
		spctr_usartWriteLine("\r");
		sprintf(temp, "Distance: %d", dist);
		spctr_usartWriteLine(temp);
		spctr_usartWriteLine("\r");
		spctr_usartWriteLine("\r");
	}*/

	spctr_spiInit();

/********* My Code ************/
	//unsigned short channel_info = 0x0002;
	//test = ad7908Config(AD7908_SEQUENCE, AD7908_PWR_NORMAL, channel_info);
	unsigned short channel_info = 0xC800;
	test = ad7908Config(AD7908_SHADOW, AD7908_PWR_NORMAL, channel_info);



	// ******************* ADC Sensor code *****************
 //Debug Mode
	//i = 0;
	/*while(1){
		i++;
		buffer[0] = 0;
		channel = ad7908Read(buffer);
		spctr_usartWriteLine("Results from ADC: \r");
		sprintf(temp, "Channel: %d	Value: %d", channel, buffer[0]);
		spctr_usartWriteLine(temp);
		spctr_usartWriteLine("\r");
		//sprintf(temp, "Return Byte: %X", channel);
		for(rep=0; rep<3;rep++){
			software_delay();
		}
	}*/
```

```
/*for(i=0; i<6; i++){
		software_delay();
}*/
//spctr_pwmUpdateDuty(2,100); //straight
//spctr_pwmUpdateDuty(3,45);
//j = 0;
/*for(j=0; j<25; j++){
		for(i=0; i<3; i++) {
				channel = ad7908Read(buffer);
				if(channel == 0) {
						IR_Left = buffer[0];
				}
				else if(channel == 1) {
						IR_Right = buffer[0];
				}
				else if(channel == 4) {
						sonar = buffer[0];
						//sprintf(temp, "Sonar Value: %d", buffer[0]);
						//spctr_usartWriteLine(temp);
						//spctr_usartWriteLine("\r");
				}
				else {
						spctr_usartWriteLine("ADC channel read error \r");
				}
				buffer[0] = 0;
				custom_delay(200000);
		}
		if((IR_Left > 100)){
				//soft_right();
		}
		else if((IR_Right > 100)){
				//soft_left();
		}
		else if(sonar < 3){
				j++;
				if(j == 4){
						j = 0;
						//hard_left(); //make random
				}
		}


		sprintf(temp, "IR Left: %d     IR Right: %d     Sonar: %d", IR_Left, IR_Right, sonar);
		spctr_usartWriteLine(temp);
		spctr_usartWriteLine("\r");
		//for(rep=0; rep<1;rep++){
		custom_delay(200000);
		//}
}*/
//****************************************************




// *************** Main LED searching program ***********************

LED_search = 0;
dist_try = 0;
too_far = 0;
repeat = 0;
turns = 0;
while(1) {
		//spctr_usartWriteLine("Sending command \r");
		if(turns > 10) {
				// Obstacle avoid for a period of time
				turns = 0;
				move_forward();
				for(j=0; j<15; j++){
						for(i=0; i<3; i++) {
								channel = ad7908Read(buffer);
```

```
			if(channel == 0) {
				IR_Left = buffer[0];
			}
			else if(channel == 1) {
				IR_Right = buffer[0];
			}
			else if(channel == 4) {
				sonar = buffer[0];
				//sprintf(temp, "Sonar Value: %d", buffer[0]);
				//spctr_usartWriteLine(temp);
				//spctr_usartWriteLine("\r");
			}
			else {
				spctr_usartWriteLine("ADC channel read error \r");
			}
			buffer[0] = 0;
			custom_delay(200000);
		}
		if((IR_Left > 100)){
			soft_right();
		}
		else if((IR_Right > 100)){
			soft_left();
		}
		else if(sonar < 3){
			i++;
			if(i == 3){
				i = 0;
				hard_left(); //make random
			}
		}


		sprintf(temp, "IR Left: %d     IR Right: %d     Sonar: %d", IR_Left, IR_Right, sonar);
		spctr_usartWriteLine(temp);
		spctr_usartWriteLine("\r");
		//for(rep=0; rep<1;rep++){
		custom_delay(200000);
		//}
	}
}
else {
	//spctr_usartWriteLine("Command Sent \r");
	//Begin Loop
	j = 0;
	exit_loop = 0;
	avg_dist = 0;
	avg_centroid_1 = 0;
	avg_centroid_2 = 0;
	avg_angle = 0;
	spctr_usartWriteLine("Entering snapshot loop \r");
	while((j < 5) & (exit_loop < 3)) {
		// Get 5 centroids, wait for all centroids or 3 invalid images with not enough points
		//spctr_usartWriteLine("Receiving Info \r");
		spctr_Propeller_sendByte(0x02);
		centroid_1 = spctr_Propeller_readInt();
		centroid_2 = spctr_Propeller_readInt();
		calc_total_1 = spctr_Propeller_readInt();
		calc_total_2 = spctr_Propeller_readInt();
		if(j>0){


			distances[j-1] = distance_calc(centroid_1, centroid_2);
			avg_dist += distances[j-1];
			avg_centroid_1 += centroid_1;
			avg_centroid_2 += centroid_2;
			angles[j-1] = .15*((centroid_1+centroid_2)/2) - 29.93;
			avg_angle += angles[j-1];
			if ((calc_total_1 < 5) | (calc_total_2 < 5)) {
				exit_loop++;
```

```
					//custom_delay(10000);
					//continue;
				}
				else {
					j++;
				}

			}
			else {
				j++;
			}
		}
		if (exit_loop == 3) {
			// If there weren't enough points...
			spctr_usartWriteLine("Not enough points. \r");
			turn_left();
			turns++;
			for(i=0; i<3; i++){
				software_delay();
			}
			dist_try = 0;
			continue;
		}
		avg_dist = avg_dist/4;
		avg_centroid_1 = avg_centroid_1/4;
		avg_centroid_2 = avg_centroid_2/4;
		avg_angle = avg_angle/4;
		exit_loop = 0;
		spctr_usartWriteLine("Calculating Average \r");
		/*for (j = 0; j < 4; j++) {
			dist_diff = abs(avg_dist-distances[j]);
			if (dist_diff > (.2*avg_dist)) { // Standard deviation within .1 of average
				exit_loop++;
			}
		}*/
		for (j = 0; j < 4; j++) {
			angle_diff = abs(avg_angle-angles[j]);
			if (angle_diff > 1) { // Standard deviation within .1 of average
				exit_loop++;
			}
		}
		if(exit_loop < 2) {
			// If standard deviation was acceptable..
			turns = 0;
			avg_centroid = (avg_centroid_1 + avg_centroid_2)/2;
			temp_angle = .15*avg_centroid  -29.93;
			sprintf(temp, "Centroid 1: %d", avg_centroid_1);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			sprintf(temp, "Centroid 2: %d", avg_centroid_2);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			sprintf(temp, "Distance: %d", avg_dist);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			sprintf(temp, "Average Centroid: %4.2f", avg_centroid);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			sprintf(temp, "Angle: %4.2f degrees", avg_angle);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			sprintf(temp, "Centroid Angle: %4.2f degrees", temp_angle);
			spctr_usartWriteLine(temp);
			spctr_usartWriteLine("\r");
			spctr_usartWriteLine("\r");
			if (avg_dist > 1000) {
				too_far++;
				if(too_far == 1) {
					//prev_dist = avg_dist;
				}
```

```
				else if(too_far == 2){
					// Got a far distance twice, move closer
					too_far = 0;
					capture_target(avg_dist,avg_angle);
					//custom_delay(22500*500);
					//brake();
				}
			}
			else {
				repeat++;
				dist_try = 0;
				if(repeat == 1){
					prev_dist = avg_dist;
				}
				else if(repeat == 2){
					repeat = 0;
					if(abs(prev_dist-avg_dist) < (0.2*avg_dist)){
						capture_target((avg_dist+prev_dist)/2, avg_angle);
						avg_dist = 0;
						avg_centroid_1 = 0;
						avg_centroid_2 = 0;
						avg_angle = 0;
						j = 0;
						while((j < 5)) {

							spctr_Propeller_sendByte(0x02);
							centroid_1 = spctr_Propeller_readInt();
							centroid_2 = spctr_Propeller_readInt();
							calc_total_1 = spctr_Propeller_readInt();
							calc_total_2 = spctr_Propeller_readInt();

							if(j>0){

								distances[j-1] =
distance_calc(centroid_1, centroid_2);
								avg_dist += distances[j-1];
								avg_centroid_1 += centroid_1;
								avg_centroid_2 += centroid_2;
								angles[j-1] =
.15*((centroid_1+centroid_2)/2) - 29.93;
								avg_angle += angles[j-1];
								j++;

							}
							else {
								j++;
							}
						}
						avg_dist = avg_dist/4;
						avg_centroid_1 = avg_centroid_1/4;
						avg_centroid_2 = avg_centroid_2/4;
						avg_angle = avg_angle/4;
						capture_target(avg_dist,avg_angle);
						for (i=95; i > 54; i--){
							spctr_pwmUpdateDuty(0,i);
							custom_delay(400);
						}
						for(i=0; i < 10; i++) {
							software_delay();
						}
						for (i=55; i < 96; i++){
							spctr_pwmUpdateDuty(0,i);
							custom_delay(400);
						}
						while(1){}
					}
					else {

					}
				}
```

```
				}
				//while(1) {}

			}
			else {
				spctr_usartWriteLine("Bad standard deviation. \r");
				dist_try++;
				exit_loop = 0;
				if (dist_try > 2) {
					spctr_usartWriteLine("Undetermined target (deviation error) Rotating.... \r");
					dist_try = 0;
					turn_left();
					for(i=0; i<3; i++){
						software_delay();
					}
				}

			}
		}


}

// ************************** End LED code ********************************
```

spctr.ad7908.c

```
/*
 * spctr.ad7908.c
 *
 *  Created on: Sep 24, 2010
 *      Author: Greg
 */

#include "spctr.h"

/* UTILS */
static void custom_delay(int clocks)
{
	while(clocks > 0) {
		clocks--;
	}
}
/* END UTILS */


void ad7908Write(unsigned short data)
{
	spctr_spiWrite(SPCTR_SPI, data, AD7908_NPCS, 1);
	custom_delay(5);
}

int ad7908Read(unsigned short *data)
{
	unsigned short zeros = 0x0000;
	spctr_spiWrite(SPCTR_SPI, zeros, AD7908_NPCS, 1);
	unsigned short temp_data;
	char string[50];
	int channel;
	spctr_spiRead(SPCTR_SPI, &temp_data);
	channel = (temp_data >> 12);
	(*data) = (temp_data >> 4) & 0x00FF;
	//sprintf(string, "Read: %X\r", temp_data);
	//spctr_usartWriteLine(string);
	return channel;

}

short ad7908Config(char sequence_mode, char power_mode, unsigned short channel_info)
/* sequence_mode and power_mode constants can be found in the header file.
 * channel_info depends on sequence_mode.  If sequence mode is selected (not shadow), then
 * channel_info is [0 ... ADD2 ADD1 ADD0], where the ADD bits correspond to max channel.
 * If shadow mode is selected, channel_info is a sequence of bits that correspond to a channel.
 * The MSB is Vin0, and LSB is Vin7. A 1 activates the channel.
 */
{
	unsigned short SEQ;
	//char SHDW;
	unsigned short PM;
	unsigned short ADD;
	unsigned short AD7908_write = 0x8013;
	unsigned short SHADOW_write = 0x0000;
	unsigned short highs = 0xFFFF;

	if(power_mode == 1) {
		PM = 0x0100;			//Auto-Shutdown
	}
	else {
		PM = 0x0300;			//Normal
	}
	if(sequence_mode == 1) {		//SEQUENCE mode
		SEQ = 0x4080;
		ADD = (channel_info << 10) & 0x1C00;
```

```
		AD7908_write = AD7908_write | SEQ | ADD | PM;

		ad7908Write(highs);
		ad7908Write(highs);
		ad7908Write(AD7908_write);
	}
	else {					//SHADOW mode
		SEQ = 0x0080;
		ADD = 0x0000;

		AD7908_write = AD7908_write | SEQ | ADD | PM;
		SHADOW_write = channel_info;

		ad7908Write(highs);
		ad7908Write(highs);
		ad7908Write(AD7908_write);
		ad7908Write(SHADOW_write);

	}
	return AD7908_write;
}
```

Spctr.propeller.c

```
/*
 * spctr.propeller.c
 *
 *  Created on: Nov 12, 2010
 *      Author: Greg
 */

#include "spctr.h"
#include "spctr.propeller.h"

void spctr_PropellerInit(void)
{
	static const gpio_map_t PROPELLER_GPIO_MAP =
	{
		{COM_PROPELLER_RX_PIN, COM_PROPELLER_RX_FNC},
		{COM_PROPELLER_TX_PIN, COM_PROPELLER_TX_FNC}
	};

	// USART options.
	static const usart_options_t PROPELLER_OPTIONS =
	{
		.baudrate     = 115200,
		.charlength   = 8,
		.paritytype   = USART_NO_PARITY,
		.stopbits     = USART_1_STOPBIT,
		.channelmode  = USART_NORMAL_CHMODE
	};
	// Assign GPIO to USART.
	gpio_enable_module(PROPELLER_GPIO_MAP, sizeof(PROPELLER_GPIO_MAP) /
sizeof(PROPELLER_GPIO_MAP[0]));
	usart_init_rs232(COM_PROPELLER, &PROPELLER_OPTIONS, PBACLK_FREQ);
}

void spctr_Propeller_sendByte(int c) {
	usart_write_char(COM_PROPELLER, c);
}

char spctr_Propeller_readChar(void) {
	char read_val;
	read_val = usart_getchar(COM_PROPELLER);
	return read_val;
}

int spctr_Propeller_readInt(void){
	int long_val = 0;
```

```
        int char_val;
        int i;
        for(i = 0; i < 4; i++) {
                char_val = spctr_Propeller_readChar();
                char_val = char_val & 0x000000FF;
                long_val += char_val << (i*8);
        }
        return long_val;
}
```