

```

1 #include "nios.h"
2
3 // Universal Definitions
4 #define SUCCESS 1
5 #define FAILURE 0
6 #define YES 1
7 #define NO 0
8 #define TRUE 1
9 #define FALSE 0
10
11 /*-----SPEEDS-----*/
12 #define BRAKE 0
13 #define STOP 0
14 #define FULL_FORWARD 7
15 #define HALF_FORWARD 6
16 #define QUARTER_FORWARD 5
17 #define EIGHTH_FORWARD 4
18 #define NEUTRAL 3
19 #define QUARTER_REVERSE 2
20 #define EIGHTH_REVERSE 1
21
22
23 /*-----DIRECTIONS-----*/
24 #define STOP 0 //??
25 #define LEFT_10 1
26 #define LEFT_20 2
27 #define LEFT_45 3
28 #define LEFT_0 4
29 #define STRAIGHT 4
30 #define RIGHT_0 4
31 #define RIGHT_10 5
32 #define RIGHT_20 6
33 #define RIGHT_45 7
34
35
36 //Interrupt Context (what caused it)
37 typedef struct{ // Struct to hold pointer to uart struct
38     np_uart *uart; // Pointer to uart struct see nios.h
39     char rxChar; // Last byte sent through UART see nios.h
40 } UARTISRContext; // Named such to .... context of interrupt
41
42
43
44 typedef struct{
45     int rmin;
46     int rmax;
47     int gmin;
48     int gmax;
49     int bmin;
50     int bmax;
51 }Color;
52
53 static Color PINK = {210, 240, 0, 60, 10, 30};
54 static Color YELLOW = {225, 240, 225, 240, 30, 80};
55 static Color GREEN = {65, 120, 130, 240, 30, 100};
56
57 //Type C packet
58 typedef struct{
59     int x1; //The left most corner x value
60     int y1; //The left most corners y value
61     int x2; //The right most corners x value
62     int y2; //The right most corners y value
63     int pixels; //# of Pixels in the tracked region,
64     int confidence; //The (# of pixels / area)*256 of the
65 // bounded rectangle and capped at 255
66 }CdataPacket;
67

```

```

68
69 //Type M packet
70 typedef struct{
71     int mx;           //The middle of mass x value
72     int my;           //The middle of mass y value
73     int x1;           //The left most corners x value
74     int y1;           //The left most corners y value
75     int x2;           //The right most corners x value
76     int y2;           //The right most corners y value
77     int pixels;       //# of Pixels in the tracked region
78     int confidence;   //The (# of pixels / area)*256 of the
79                       // bounded rectangle and capped at 255
80 }MdataPacket;
81
82
83 //Type N packet
84 typedef struct{
85     int spos;         //The current servo position
86     int mx;           //The middle of mass x value
87     int my;           //The middle of mass y value
88     int x1;           //The left most corners x value
89     int y1;           //The left most corners y value
90     int x2;           //The right most corners x value
91     int y2;           //The right most corners y value
92     int pixels;       //# of Pixels in the tracked region
93     int confidence;   //The (# of pixels / area)*256 of the
94                       // bounded rectangle and capped at 255
95 }NdataPacket;
96
97
98 //Type S packet
99 typedef struct{
100     int rmean ;       //the mean Red or Cr (approximates r-g)
101     int gmean ;       //the mean Green or Y (approximates intensity)
102     int bmean ;       //the mean Blue or Cb (approximates b-g)
103     int rdev  ;       //the *deviation of red or Cr
104     int gdev  ;       //the *deviation of green or Y
105     int bdev  ;       //the *deviation of blue or Cb
106 }SdataPacket;
107
108
109 /*-----FUNCTIONS PROTOTYPES-----*/
110
111 // Vicon_Camera.c
112 int  SendCommand(char *s);
113 int  DisplayResponse(void);
114 int  EmptyArray(void);
115 void ViconMagic(char *s);
116 void ViconMagic2(char *s);
117 int  ViconTest(void);
118 int  PacketTokenizer(void);
119 int  UpdatePackets(void);
120 int  SetCameraIdle(void);
121 int  SetContrast (int contrastI);
122 int  GetContrast(void);
123 int  SetBrightness (int brightnessI);
124 int  GetBrightness(void);
125 int  SetColorMode (int colormode);
126 int  GetColorMode(void);
127 int  SetClockSpeed(int clockspeedI);
128 int  GetClockSpeed(void);
129 int  SetExposure(int exposureI);
130 int  GetExposure(void);
131 int  ToggleExposure(void);
132 int  DumpFrame(void);
133 int  SetDelay(int delayI);
134 int  GetDelay(void);

```

```
135 int GetMeanColor(void);
136 int GetVersion ( void );
137 int SetHalfHorizontalResolutionMode(int active);
138 int GetHalfHorizontalResolutionMode(void);
139 int GetServoInput(void);
140 int SetTrackingLight(int trackingI);
141 int SetLineMode(void);
142 int SetMiddleMass(int middlemassI);
143 int SetNoiseFilter(int noisefilterI);
144 int ToggleNoiseFilter(void);
145 int GetNoiseFilter (void);
146 int SetPollMode(int pollmodeI);
147 int GetPollMode(void);
148 int SetRawMode(int rawmodeI);
149 int GetRawMode (void);
150 int ResetCamera(void);
151 int SetServoPosition(int positionI);
152 int SetSwitchingMode(int switchingmodeI);
153 int GetSwitchingMode(void);
154 int SetWindowSize(int x, int y, int x1, int y1);
155 int TrackWindow(void);
156 int TrackColor(Color c);
157 void InterruptServiceRoutine(int context);
158 void DisableUartInterrupt(void);
159 void EnableUartInterrupt(void);
160 void InterruptHandler (int context);
161
162 // Vicon_Servo.c
163 int GetSpeed(void);
164 int GetDirection(void);
165 void SetSpeed(int spd);
166 void SetDirection(int drx);
167
168 // Vicon.c
169 int main2(void);
170 int FindColor(void);
171 int CameraStart(void);
172 int TrackState(void);
173 int DataControl(int con, int m1, int m2, int pix);
174 //int SendServo(ServoValue);
175 int Reset(void);
176 void MyPIO_ISR(int context);
177 void InitializeCamera(void);
178
179 // Vicon_pacet_test.c
180 int CheckPackets(void);
181
182
```

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 use IEEE.STD_LOGIC_ARITH.all;
4 use IEEE.STD_LOGIC_UNSIGNED.all;
5
6 ENTITY clk_div IS
7
8     PORT
9     (
10         clock_33Mhz          : IN    STD_LOGIC;
11         clock_1MHz           : OUT   STD_LOGIC;
12         clock_100KHz        : OUT   STD_LOGIC;
13         clock_10KHz         : OUT   STD_LOGIC;
14         clock_1KHz          : OUT   STD_LOGIC;
15         clock_100Hz         : OUT   STD_LOGIC;
16         clock_10Hz          : OUT   STD_LOGIC;
17         clock_1Hz           : OUT   STD_LOGIC);
18
19 END clk_div;
20
21 ARCHITECTURE a OF clk_div IS
22
23     SIGNAL count_1Mhz: STD_LOGIC_VECTOR(4 DOWNTO 0);
24     SIGNAL count_100KHz, count_10KHz, count_1KHz : STD_LOGIC_VECTOR(2 DOWNTO 0);
25     SIGNAL count_100Hz, count_10Hz, count_1Hz : STD_LOGIC_VECTOR(2 DOWNTO 0);
26     SIGNAL clock_1Mhz_int, clock_100KHz_int, clock_10KHz_int, clock_1KHz_int:
27         STD_LOGIC;
28     SIGNAL clock_100Hz_int, clock_10Hz_int, clock_1Hz_int : STD_LOGIC;
29 BEGIN
30     PROCESS
31     BEGIN
32         -- Divide by 25
33         WAIT UNTIL clock_33Mhz'EVENT and clock_33Mhz = '1';
34         IF count_1Mhz < 32 THEN
35             count_1Mhz <= count_1Mhz + 1;
36         ELSE
37             count_1Mhz <= "00000";
38         END IF;
39         IF count_1Mhz < 16 THEN
40             clock_1Mhz_int <= '0';
41         ELSE
42             clock_1Mhz_int <= '1';
43         END IF;
44         -- Ripple clocks are used in this code to save prescaler hardware
45         -- Sync all clock prescaler outputs back to master clock signal
46         clock_1Mhz <= clock_1Mhz_int;
47         clock_100KHz <= clock_100KHz_int;
48         clock_10KHz <= clock_10KHz_int;
49         clock_1KHz <= clock_1KHz_int;
50         clock_100Hz <= clock_100Hz_int;
51         clock_10Hz <= clock_10Hz_int;
52         clock_1Hz <= clock_1Hz_int;
53     END PROCESS;
54
55     -- Divide by 10
56     PROCESS
57     BEGIN
58         WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
59         IF count_100KHz /= 4 THEN
60             count_100KHz <= count_100KHz + 1;
61         ELSE
62             count_100KHz <= "000";
63             clock_100KHz_int <= NOT clock_100KHz_int;
64         END IF;
65     END PROCESS;
66

```

```

67 -- Divide by 10
68 PROCESS
69 BEGIN
70     WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
71     IF count_10Khz /= 4 THEN
72         count_10Khz <= count_10Khz + 1;
73     ELSE
74         count_10khz <= "000";
75         clock_10Khz_int <= NOT clock_10Khz_int;
76     END IF;
77 END PROCESS;
78
79 -- Divide by 10
80 PROCESS
81 BEGIN
82     WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
83     IF count_1Khz /= 4 THEN
84         count_1Khz <= count_1Khz + 1;
85     ELSE
86         count_1khz <= "000";
87         clock_1Khz_int <= NOT clock_1Khz_int;
88     END IF;
89 END PROCESS;
90
91 -- Divide by 10
92 PROCESS
93 BEGIN
94     WAIT UNTIL clock_1Khz_int'EVENT and clock_1Khz_int = '1';
95     IF count_100hz /= 4 THEN
96         count_100hz <= count_100hz + 1;
97     ELSE
98         count_100hz <= "000";
99         clock_100hz_int <= NOT clock_100hz_int;
100    END IF;
101 END PROCESS;
102
103 -- Divide by 10
104 PROCESS
105 BEGIN
106     WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
107     IF count_10hz /= 4 THEN
108         count_10hz <= count_10hz + 1;
109     ELSE
110         count_10hz <= "000";
111         clock_10hz_int <= NOT clock_10hz_int;
112     END IF;
113 END PROCESS;
114
115 -- Divide by 10
116 PROCESS
117 BEGIN
118     WAIT UNTIL clock_10hz_int'EVENT and clock_10hz_int = '1';
119     IF count_1hz /= 4 THEN
120         count_1hz <= count_1hz + 1;
121     ELSE
122         count_1hz <= "000";
123         clock_1hz_int <= NOT clock_1hz_int;
124     END IF;
125 END PROCESS;
126
127 END a;
128
129

```

```

1 #include "nios.h"
2 #include "vicon.h"
3
4 /*-----*/
5 * Author(s): Kwabena Asare Bosompem
6 *           Andre Moore
7 *           Jeff Vickers
8 *           Kevin Walker
9 * Created:   May 30th, 2002.
10 * Modified:  July 15th, 2002.
11 * Description: Functions for Servo Control
12 * Comments:
13 *-----*/
14
15
16
17 /*-----*
18 * File Wide Variables *
19 *-----*/
20 int viconspeed;
21 int vicondirection;
22 extern np_pio *direction;
23 extern np_pio *speed ;
24
25
26 /*-----*
27 * Accessors *
28 *-----*/
29 /**
30 * Gets the servo's current speed and returns it as an int
31 * Created: June 6th, 2002
32 * Last Modified: June 6th, 2002
33 * Comments: Implementation to be discussed
34 *
35 */
36 int GetSpeed(void)
37 {
38     return viconspeed;
39 }
40 //end of GetSpeed
41
42
43 /**
44 * Gets the servo's current direction and returns it as a short
45 * or preferably a binary number
46 * Created: June 6th, 2002
47 * Last Modified: June 6th, 2002
48 * Comments: Implementation to be discussed
49 *
50 */
51 int GetDirection(void)
52 {
53     return vicondirection;
54 }
55 //end of GetDirection
56
57
58 /*-----*
59 * Modifiers *
60 *-----*/
61 /**
62 * Sets the servo's speed based on integer passed in
63 * Created: June 6th, 2002
64 * Last Modified: July 15th, 2002
65 * Comments: Acceptable range 0-7
66 */
67 void SetSpeed(int spd)

```

```
68 {
69     if(spd >7)
70         spd = 7;
71     else if(spd < 0)
72         spd = 0;
73
74     viconspeed = spd;
75     speed = (np_pio *)0x430;
76     speed->np_piodata = viconspeed;
77
78 }//end of SetSpeed
79
80
81
82 /**
83  * Sets the servo's direction based on the integer passed in
84  * Created:         June 6th, 2002
85  * Last Modified:  July 15th, 2002
86  * Comments:       Acceptable range 0-7
87  *
88  */
89 void SetDirection(int drx)
90 {
91     if(drx >7)
92         drx = 7;
93     else if(drx < 0)
94         drx = 0;
95
96     vicondirection = drx ;
97     direction = (np_pio *)0x490;
98     direction->np_piodata = vicondirection;
99
100 }//end of SetDirection
101
```

```

1 #include "vicon.h"
2 #include "nios.h"
3 #include "pio_lcd16207.h"
4 #include <stdio.h>
5 #include <string.h>
6
7 /*-----*/
8 * Author(s):   Kwabena Asare Bosompem
9 *              Andre Moore
10 *              Jeff Vickers
11 *              Kevin Walker
12 * Created:     May 30th, 2002.
13 * Modified:    June 15th, 2002.
14 * Description: Functions for Camera Control
15 * Version:     0.1a
16 * Comments:    Every function that returns a data packet assumes
17 *              the poll mode is set to 1. Default camera setting
18 *              is 0. Also no raw mode. Not necessary for Nios.
19 *-----*/
20
21
22
23 /*-----*
24 * Definitions & Global Variables
25 *-----*/
26 char lastresponse[100];
27 const char VER[18] = "ACK CMUcam v1.12 :";
28 const char RVER[19] = "ACK  CMUcam v1.12 :";
29 const char ACK[5] = "ACK :";
30 const char NCK[5] = "NCK :";
31 const char COL[2] = ": ";
32 const char VER_RM[14] = "CMUcam v1.12 :";
33 const char RVER_RM[15] = " CMUcam v1.12 :";
34 const char ACK_RM[1] = ":";
35
36
37 const int AUTOGAIN_ON   = 33;           //DEFAULT
38 const int AUTOGAIN_OFF = 32;
39
40 const int YCC_ON       = 36;
41 const int YCC_OFF      = 32;
42 const int RGB_ON       = 44;
43 const int RGB_OFF      = 40;
44
45 const int TRACKING_ON  = 1;
46 const int TRACKING_OFF = 0;
47 const int TRACKING_AUTO = 2;
48
49 int crindex = 0;           // Keep track of camera response array
50 UARTISRContext gC = {0,-1}; // Context and last byte preset.
51 int k=0;                 // Index of array to be tokenized for
52                          // packets;
53 int crtemp=0;            // Temporary int holds
54 int temp=0;              // Temporary int holds
55
56 SdataPacket spacket = {0};
57 MdataPacket mpacket = {0};
58 NdataPacket npacket = {0};
59 CdataPacket cpacket = {0};
60
61
62 int pollmode   = 0;           // Initialize poll mode
63 int rawmode    = 2;           // Intialize raw mode
64 int mmassmode  = 1;           // Initialize middle mass mode
65 int uartstatus = 0;           // Status of serial port
66 int clockspeed = 17;         // Default values
67 int contrast   = 127;

```

```

68 int brightness = 127;
69 int exposure = 1;
70 int delay = 1;
71 int colormode = 40;
72 int active = 0;
73 int tracking = 2;
74 int middlemass = 1;
75 int noisefilter = 1;
76 int position = 0;
77 int switchingmode = 0;
78
79 /*-----*
80 *                F U N C T I O N S
81 *-----*/
82
83 /**
84 * S E N D      C O M M A N D      ( S T R I N G )
85 * Description: Sends through the uart(serial port) a preformatted
86 *              command(string) to the CMUcam.
87 * Created:    June 6th, 2002.
88 * Paramaters: Pointer to a string(s)
89 * Returns:    Int indicating success or failure
90 * Comments:   See nios_peripheral_reference_manual.pdf for
91 *              details on nr_uart_txstring
92 * Modified:   June 14th, 2002.
93 */
94 int SendCommand(char *s)
95 {
96     uartstatus = nr_uart_txstring(s);
97     if (uartstatus > 0)
98         return SUCCESS;
99     else
100         return FAILURE;
101
102 } //end of SendCommand(char*)
103
104
105
106 /**
107 * D I S P L A Y   R E S P O N S E ( )
108 * Description: Displays to the LCD the last string recieved from
109 *              the CMUcam.
110 * Created:    June 6th, 2002.
111 * Paramaters: Void
112 * Returns:    int to indicate successful print to lcd
113 * Comments:   Took delay out. Delays need to be added in main.
114 * Modified:   June 15th, 2002.
115 */
116 int DisplayResponse(void)
117 {
118     nr_pio_lcdwritescreen(lastresponse);
119     return SUCCESS;
120
121 } //end of Display Response
122
123
124
125 /**
126 * E M P T Y     A R R A Y ( )
127 * Description: Empties the array holding a string of the last
128 *              camera response
129 * Created:    June 6th, 2002.
130 * Paramaters: Void
131 * Returns:    int
132 * Comments:   Currently looking for quicker way to achieve this
133 *              with pointers
134 * Modified:   June 14th, 2002.

```

```

135 */
136 int EmptyArray()
137 {
138     int i;
139     for (i=0; i<100;i++)
140     {
141         lastresponse[i]= 0;        // 0 is ascii for null
142     }
143     return SUCCESS;
144
145 }// end of EmptyArray()
146
147
148
149 /**
150 * V I C O N     M A G I C     ( S T R I N G )
151 * Description: Go Figure. It calls 4 functions
152 * Created:      June 15th, 2002.
153 * Paramaters:  s. a character array of command to send camera
154 * Returns:      Nothing
155 * Comments:     None
156 * Modified:     June 15th, 2002.
157 */
158 void ViconMagic(char *s)
159 {
160     crindex = 0;
161     k = 0;
162     EmptyArray();
163     EnableUartInterrupt();
164     SendCommand(s);
165     nr_delay(100);
166     DisableUartInterrupt();
167
168 }//end of ViconMagic
169
170 void ViconMagic2(char *s)
171 //The packets need extra time in a delay since it is more characters returned.
172 //Without the 1/2 ms delay the packets would be clipped.
173 {
174     crindex = 0;
175     k = 0;
176     EmptyArray();
177     EnableUartInterrupt();
178     SendCommand(s);
179     nr_delay(500);
180     DisableUartInterrupt();
181
182 }//end of ViconMagic
183
184 /**
185 * V I C O N     T E S T     ( V O I D )
186 * Description: Go Figure.
187 * Created:      June 15th, 2002.
188 * Paramaters:  void
189 * Returns:      int SUCCESS or FAILURE
190 * Comments:     None
191 * Modified:     June 15th, 2002.
192 */
193 int ViconTest(void)
194 {
195     int rmode;
196     rmode = GetRawMode();
197     rmode = rmode & 2;
198     if (rmode == 2)
199         if (strcmp(ACK,lastresponse)!=0)
200             return FAILURE;
201     else

```

```

202         return SUCCESS;
203     else
204         if (strcmp(ACK_RM,lastresponse)!=0)
205             return FAILURE;
206         else
207             return SUCCESS;
208
209 }//end of ViconTest
210
211
212
213 /**
214 * P A C K E T   T O K E N I Z E R
215 * Description: Dumps bitmap of current still image
216 * Created: June 13th, 2002.
217 * Paramaters: Void
218 * Returns: Int
219 * Comments: Splits into consistuent numbers
220 * Modified: June 7th, 2002.
221 */
222 int PacketTokenizer(void){
223     int xi;
224     while(lastresponse[++k]== 32);
225     crtemp =lastresponse[k];
226     while((crtemp<58) && (crtemp>47)){
227         temp = temp*10 + crtemp - 48;
228         k++;
229         crtemp =lastresponse[k];
230     }//end of
231     xi = temp;
232     temp = 0;
233     return xi;
234
235 }//end of PacketTokenizer()
236
237
238
239 /**
240 * U P D A T E       P A C K E T S ( V O I D )
241 * Description: Dumps bitmap of current still image
242 * Created: June 13th, 2002.
243 * Paramaters: Void
244 * Returns: Int
245 * Comments: Splits into consistuent numbers
246 * Modified: June 7th, 2002.
247 */
248 int UpdatePackets(void){
249
250     while((k<100) || (lastresponse[k] == 0)){
251         if(lastresponse[k] == 65 )           // 65 is ascii A
252         {
253             k++;
254             k++;
255         }
256         if(lastresponse[k] == 83 )           // 83 is ascii S
257         {
258             spacket.rmean = PacketTokenizer();
259             spacket.gmean = PacketTokenizer();
260             spacket.bmean = PacketTokenizer();
261             spacket.rdev = PacketTokenizer();
262             spacket.gdev = PacketTokenizer();
263             spacket.bdev = PacketTokenizer();
264         }
265         if(lastresponse[k] == 67 )           // 67 is ascii C
266         {
267             cpacket.x1 = PacketTokenizer();
268             cpacket.y1 = PacketTokenizer();

```

```

269         cpacket.x2           = PacketTokenizer();
270         cpacket.y2           = PacketTokenizer();
271         cpacket.pixels       = PacketTokenizer();
272         cpacket.confidence    = PacketTokenizer();
273     }
274     if(lastresponse[k] == 77 )           // 77 is ascii M
275     {
276         mpacket.mx           = PacketTokenizer();
277         mpacket.my           = PacketTokenizer();
278         mpacket.x1           = PacketTokenizer();
279         mpacket.y1           = PacketTokenizer();
280         mpacket.x2           = PacketTokenizer();
281         mpacket.y2           = PacketTokenizer();
282         mpacket.pixels       = PacketTokenizer();
283         mpacket.confidence    = PacketTokenizer();
284     }
285     if(lastresponse[k] == 78 )           // 78 is ascii N
286     {
287         npacket.mx           = PacketTokenizer();
288         npacket.mx           = PacketTokenizer();
289         npacket.my           = PacketTokenizer();
290         npacket.x1           = PacketTokenizer();
291         npacket.y1           = PacketTokenizer();
292         npacket.x2           = PacketTokenizer();
293         npacket.y2           = PacketTokenizer();
294         npacket.pixels       = PacketTokenizer();
295         npacket.confidence    = PacketTokenizer();
296     }
297     k++;
298 }
299 return SUCCESS;
300
301 //end of UpdatePackets
302
303
304
305 /*****
306 /*****[ pg.13 ]*****/
307 /*****/
308
309 /**
310 * S E T   C A M E R A   I D L E ( )
311 * Description: Sets Camera Idle. Stops streaming of packets in
312 *               when camera is in tracking mode.
313 * Created:     June 7th, 2002.
314 * Paramaters:  Void
315 * Returns:     Int SUCCESS or FAILURE
316 * Comments:
317 * Modified:    June 15th, 2002
318 */
319 int SetCameraIdle(void)
320 {
321     char cmdtosend [4] = "\r";
322     ViconMagic(cmdtosend);
323     return ViconTest();
324 }
325 //end of SetCameraIdle()
326
327
328
329 /**
330 * S E T   C O N T R A S T   ( I N T )
331 * Description: Set's the camera's contrast
332 * Created:     June 15th, 2002.
333 * Paramaters:  contrastI. an int from 0-255
334 * Returns:     int (SUCCESS or FAILURE)
335 * Comments:

```

```

336 * Modified:    June 15th, 2002.
337 */
338 int SetContrast (int contrastI)
339 {
340     char cmdtosend [6];
341     contrast = contrastI;
342     if((contrast >255) || (contrast < 0))
343         return FAILURE ;
344     sprintf(cmdtosend, "CR 5 %d\r", contrast);
345     ViconMagic(cmdtosend);
346     return ViconTest();
347
348 }//end of SetContrast(int)
349 int GetContrast(void)
350 {
351     return contrast;
352
353 }//end of GetExposure()
354
355
356
357 /**
358 * S E T    B R I G H T N E S S    ( I N T )
359 * Description: Set's the camera's brightness
360 * Created:    June 15th, 2002.
361 * Paramaters: brightnessI. an int from 0-255
362 * Returns:    int (SUCCESS or FAILURE)
363 * Comments:
364 * Modified:    June 15th, 2002.
365 */
366 int SetBrightness (int brightnessI)
367 {
368     char cmdtosend [6];
369     brightness = brightnessI;
370     if((brightness >255) || (brightness < 0))
371         return FAILURE ;
372     sprintf(cmdtosend, "CR 6 %d\r", brightness);
373     ViconMagic(cmdtosend);
374     return ViconTest();
375
376 }//end of SetBrightness(int)
377 int GetBrightness(void)
378 {
379     return brightness;
380
381 }//end of GetExposure()
382
383
384
385 /**
386 * S E T    C O L O R M O D E    ( I N T )
387 * Description: Set's the camera's color mode
388 * Created:    June 15th, 2002.
389 * Paramaters: colormode. see vicon.h
390 * Returns:    int (SUCCESS or FAILURE)
391 * Comments:    See CMUcam manual for command
392 *              CR
393 * Modified:    June 15th, 2002.
394 */
395 int SetColorMode (int colormode)
396 {
397     char cmdtosend [6];
398     if((colormode >44) || (colormode < 32))    //inadequate check
399         return FAILURE ;
400     sprintf(cmdtosend, "CR 18 %d\r", colormode);
401     ViconMagic(cmdtosend);
402     return ViconTest();

```

```

403
404 }//end of SetContrast(int)
405 int GetColorMode(void)
406 {
407     return colormode;
408 }
409 }//end of GetExposure()
410
411
412
413 /**
414 * S E T      C L O C K S P E E D      ( I N T )
415 * Description: Set's camera's speed in fps
416 * Created:    June 15th, 2002.
417 * Paramaters: clockspeedI The number of frames/s
418 * Returns:    int (SUCCESS or FAILURE)
419 * Comments:
420 * Modified:   June 15th, 2002.
421 */
422 int SetClockSpeed(int clockspeedI)
423 {
424     char cmdtosend [6];
425     if((clockspeedI>12) || (clockspeedI< 2)) //inadequate check
426         return FAILURE ;
427     clockspeed = clockspeedI;
428     sprintf(cmdtosend, "CR 17 %d\r", colormode);
429     ViconMagic(cmdtosend);
430     return ViconTest();
431 }
432 }//end of SetClockSpeed(int)
433 int GetClockSpeed(void)
434 {
435     return clockspeed;
436 }
437 }//end of GetExposure()
438
439
440
441 /**
442 * S E T      E X P O S U R E ( )
443 * Description: Don't quite understand what it does for the camera
444 * Created:    June 15th, 2002.
445 * Paramaters: int AUTOGAIN_ON or AUTOGAIN_OFF
446 * Returns:    int (SUCCESS or FAILURE)
447 * Comments:
448 * Modified:   June 15th, 2002.
449 */
450 int SetExposure(int exposureI)
451 {
452     char cmdtosend [6];
453
454     if(exposureI != AUTOGAIN_ON || exposureI != AUTOGAIN_OFF)
455         return FAILURE ;
456
457     exposure = exposureI;
458     sprintf(cmdtosend, "CR 19 %d\r", exposure);
459
460     ViconMagic(cmdtosend);
461     return ViconTest();
462 }
463 }//end of SetExposure
464 int GetExposure(void)
465 {
466     return exposure;
467 }
468 }//end of GetExposure()
469 int ToggleExposure(void)

```

```

470 {
471     int status;
472     if(exposure == AUTOGAIN_ON)
473         status = SetExposure(AUTOGAIN_OFF);
474     else
475         status = SetExposure(AUTOGAIN_ON);
476     return status;
477 } //end of ToggleExposure
478
479
480
481 /*****
482 /*****[ pg.14 ]*****/
483 /*****
484
485 /**
486 * D U M P     F R A M E ( )
487 * Description: Dumps bitmap of current still image
488 * Created:     June 7th, 2002.
489 * Paramaters:  Void
490 * Returns:     Int
491 * Comments:    Not to be used. Not beneficial to ViCoN-Bot
492 *              Transmission time for 80*143
493 *              Memory required to hold 80*143 extra bytes
494 *              Processing power and extra memory required to manipulate it
495 * Modified:    June 15th, 2002.
496 */
497 int DumpFrame(void)
498 {
499     char command [5] = "DF\r";
500     //Not to be used
501     //ViconMagic(command);
502     nr_pio_lcdwritescreen("Gee! what were you thinking");
503     nr_delay(100000); //10 Sec wait. Same time it takes to dump frame
504     return SUCCESS;
505 }
506 } //end of DumpFrame
507
508
509
510 /**
511 * S E T     D E L A Y   ( I N T )
512 * Description: Sets delay before camera transmits packets
513 * Created:     June 15th, 2002.
514 * Paramaters:  delayI 0-255 1 unit = baud rate
515 * Returns:     int (SUCCESS or FAILURE )
516 * Comments:    see CMUcam Manual for details
517 *              We used firmware v1.12
518 *
519 * Modified:    June 15th, 2002.
520 */
521 int SetDelay(int delayI)
522 {
523     char cmdtosend [6];
524     if((delayI>255) || (delayI< 0)) //inadequate check
525         return FAILURE ;
526     delay = delayI;
527     sprintf(cmdtosend, "DM %d\r", delay);
528     ViconMagic(cmdtosend);
529     return ViconTest();
530 }
531 } //end of SetDelay
532 int GetDelay(void)
533 {
534     return delay;
535 }
536 } //end of GetDelay

```

```

537
538
539
540 /**
541 * G E T   M E A N   C O L O R   ( )
542 * Description: Get mean color
543 * Created:    June 15th, 2002.
544 * Paramaters:
545 * Returns:    spacket with most current measurements
546 * Comments:   In streaming mode it will return the last spacket and
547 *             fill unknown values with zero if it is truncated.
548 * Modified:   June 15th, 2002.
549 */
550 int GetMeanColor(void)
551 {
552     char cmdtosend [4] = "GM\r";
553     ViconMagic(cmdtosend);
554     UpdatePackets();
555     return SUCCESS;
556 }
557 }//end of GetMeanColor
558
559
560
561 /*****
562 /*****[ pg.15 ]*****/
563 /*****
564 /**
565 * G E T   V E R S I O N   ( )
566 * Description: Gets current Camera Version
567 * Created:    June 7th, 2002.
568 * Paramaters: Void
569 * Returns:    Int (SUCCESS or FAILURE)
570 * Comments:
571 * Modified:   June 15th, 2002.
572 */
573 int GetVersion ( void )
574 {
575     char command [5] = "GV\r";
576     int rmode;
577
578     ViconMagic(command);
579     rmode = GetRawMode();
580     rmode = rmode & 2;
581     if (rmode = 2)
582         if (strcmp(VER,lastresponse)!=0)
583             return FAILURE;
584     else
585         return SUCCESS;
586     else
587         if (strcmp(VER_RM,lastresponse)!=0)
588             return FAILURE;
589     else
590         return SUCCESS;
591 }
592 }//end of GetVersion()
593
594
595
596 /**
597 * S E T   H A L F   H O R I Z O N T A L   R E S O L U T I O N   ( I N T )
598 * Description: Puts the camera into half-horizontal resolution mode
599 *             for DumpFrame and SetLineMode
600 * Created:    June 15th, 2002.
601 * Paramaters:
602 * Returns:    int (SUCCESS or FAILURE )
603 * Comments:   Not to be implemented. Irrelevant in ViCON-Bot operation

```

```

604 *
605 *
606 * Modified:    June 15th, 2002.
607 */
608 int SetHalfHorizontalResolutionMode(int active)
609 {
610     //active = 1 => every odd column processed
611     //active = 0 => disable this mode
612     nr_pio_lcdwritescreen("HM ? Why oh why");
613     nr_delay(100000);        //Wait 10 secs
614     return SUCCESS;
615 }
616 //end of SetHalfHorizontalResolution()
617 int GetHalfHorizontalResolutionMode(void)
618 {
619     return active;
620 }
621 //end of SetHalfHorizontalResolution()
622
623
624
625 /**
626 * GET SERVO INPUT
627 * Description: Gets the voltage on the servo line.
628 *             0 = line low. 1 = line high
629 * Created:    June 15th, 2002.
630 * Paramaters: None
631 * Returns:    int -1 failure (0 low voltage or 1 high voltage)
632 * Comments:
633 * Modified:    June 15th, 2002.
634 */
635 int GetServoInput(void)
636 {
637     char cmdtosend [4] = "I1\r";
638     int pos;
639     int rmode;
640
641     ViconMagic(cmdtosend);
642
643     rmode = GetRawMode();
644     rmode = rmode & 2;
645
646     if (rmode = 2)
647         pos = 0;
648     else
649         pos = 4;
650
651     if(lastresponse[pos] == 48)
652         return 0;
653     else if(lastresponse[pos] == 49)
654         return 1;
655     else
656         return -1;
657 }
658 //end of GetServoInput
659
660
661
662 /**
663 * S E T    T R A C K I N G    L I G H T ( I N T )
664 * Description: Controls the tracking light.
665 *             Acceptable values 0,1,2(default)
666 * Created:    June 15th, 2002.
667 * Paramaters: trackingI
668 * Returns:    int (SUCCESS or FAILURE )
669 * Comments:
670 * Modified:    June 15th, 2002.

```

```

671 */
672 int SetTrackingLight(int trackingI)
673 {
674     char cmdtosend [6];
675
676     if((trackingI >2) || (trackingI < 0))
677         return FAILURE ;
678
679     tracking = trackingI;
680     sprintf(cmdtosend, "L1 %d\r", tracking);
681
682     ViconMagic(cmdtosend);
683     return ViconTest();
684 }
685 //end of SetTrackingLight
686
687
688
689 /*****
690 /*****[ pg.16 ]*****/
691 /*****
692
693 /**
694 * S E T   L I N E   M O D E ( I N T )
695 * Description: Turns on line mode
696 * Created:     June 15th, 2002.
697 * Paramaters:
698 * Returns:     int (SUCCESS or FAILURE )
699 * Comments:    Will not be implemented
700 *              Over complicates the code for the
701 *              ViCoN-Bot
702 * Modified:    June 15th, 2002.
703 */
704 int SetLineMode(void)
705 {
706     return SUCCESS;
707 }
708 //end of SetLineMode();
709
710
711
712 /*****
713 /*****[ pg.17 ]*****/
714 /*****
715 /**
716 * S E T   M I D D L E M A S S ( I N T )
717 * Description: Set middle mass
718 *
719 * Created:     June 15th, 2002.
720 * Paramaters:
721 * Returns:     int (SUCCESS or FAILURE )
722 * Comments:
723 *
724 *
725 * Modified:    June 15th, 2002.
726 */
727 int SetMiddleMass(int middlemassI)
728 {
729     char cmdtosend [6];
730
731     //if((middlemassI >10) || (middlemassI < 0))
732     // return FAILURE ;
733
734     middlemass = middlemassI;
735     sprintf(cmdtosend, "MM %d\r", middlemass);
736     ViconMagic(cmdtosend);
737     return ViconTest();

```

```

738
739 }//end of SetMiddleMass(int)
740
741
742
743 /**
744 * S E T   N O I S E   F I L T E R   ( I N T )
745 * Description: Controls the Noise filter setting on the camera
746 *             1- more selective tracking (default)
747 *             0- less selective tracking
748 * Created:    June 15th, 2002.
749 * Paramaters:
750 * Returns:    int (SUCCESS or FAILURE )
751 * Comments:
752 *
753 *
754 * Modified:   June 15th, 2002.
755 */
756 int SetNoiseFilter(int noisefilterI)
757 {
758     char cmdtosend [6];
759
760     if((noisefilterI >2) || (noisefilterI < 0))
761         return FAILURE ;
762
763     noisefilter = noisefilterI;
764     sprintf(cmdtosend, "NF %d\r", noisefilter);
765     ViconMagic(cmdtosend);
766     return ViconTest();
767 }//end of SetNoiseFilter
768 int ToggleNoiseFilter(void)
769 {
770     if(noisefilter == 0)
771         return SetNoiseFilter(1);
772     else
773         return SetNoiseFilter(0);
774
775 }//end of ToggleNoiseFilter
776 int GetNoiseFilter (void)
777 {
778     return noisefilter;
779 }//end of GetNoiseFilter
780
781
782
783 /**
784 * S E T   P O L L   M O D E   ( I N T )
785 * Description: Switchs packet dumping from streaming to a line
786 *             at a time
787 * Created:    June 7th, 2002.
788 * Paramaters: Int modeI (see CMUcam Manual)
789 * Returns:    Int
790 * Comments:   Very Functional
791 *
792 * Modified:   June 7th, 2002.
793 * Length:    24
794 */
795 int SetPollMode(int pollmodeI)
796 {
797     char cmdtosend [6];
798     //if((pollmodeI == 0) || (pollmodeI == 1))
799         // return FAILURE;
800     pollmode = pollmodeI;
801     sprintf(cmdtosend, "PM %d\r", pollmode);
802     ViconMagic(cmdtosend);
803     DisplayResponse();
804     nr_delay(800);

```

```

805     return ViconTest();
806
807 } //end of SetPollMode(int)
808 int GetPollMode(void)
809 {
810     return pollmode;
811
812 } //end of SetPollMode(int)
813
814
815
816 /*****
817 /*****[ pg.18 ]*****/
818 /*****
819
820 /**
821 * S E T   R A W   M O D E
822 * Description: Sets the input to or output from the camera
823 *
824 *
825 * Created:      June 11th, 2002.
826 * Paramaters:
827 * Returns:
828 * Comments:     Very Functional
829 * Modified:     June 15th, 2002.
830 */
831 int SetRawMode(int rawmodeI)
832 {
833     char cmdtosend [5];
834     //if((rawmodeI == 0) || (rawmodeI == 2))    //avoids other modes
835     // return FAILURE;
836     rawmode = rawmodeI;
837     sprintf(cmdtosend, "RM %d\r", rawmode);
838     ViconMagic(cmdtosend);
839     return ViconTest();
840
841 } //end of SetPollMode(int)
842 int GetRawMode (void)
843 {
844     return rawmode;
845
846 } //end of GetRawMode
847
848
849
850 /**
851 * R E S E T   C A M E R A ( )
852 * Description: Resets camera
853 * Created:      June 15th, 2002.
854 * Paramaters:   None
855 * Returns:      int (SUCCESS or FAILURE )
856 * Comments:
857 *
858 * Modified:     June 15th, 2002.
859 */
860 int ResetCamera(void)
861 {
862     char command [5] = "RS\r";
863     int rmode;
864
865     ViconMagic(command);
866     rmode = GetRawMode();
867     rmode = rmode & 2;
868     if (rmode = 2)
869         if (strcmp(RVER, lastresponse) != 0)
870             return FAILURE;
871     else

```

```

872         return SUCCESS;
873     else
874         if (strcmp(RVER_RM,lastresponse)!=0)
875             return FAILURE;
876         else
877             return SUCCESS;
878
879 }//end of ResetCamera
880
881
882
883 /**
884 * S E T   S E R V O   P O S I T I O N   ( I N T )
885 * Description: Set's the servo position
886 * Created:     June 15th, 2002.
887 * Paramaters: position 0-255
888 * Returns:     int (SUCCESS or FAILURE )
889 * Comments:
890 *
891 *
892 * Modified:    June 15th, 2002.
893 */
894 int SetServoPosition(int positionI)
895 {
896     char cmdtosend [5];
897     //if((positionI > 255) || (positionI < 0)) //avoids other modes
898     // return FAILURE;
899     position = positionI;
900     sprintf(cmdtosend,"S1 %d\r",position);
901     ViconMagic(cmdtosend);
902     return ViconTest();
903
904 }//end of SetServoPosition()
905
906
907
908 /*****
909 /*****[ pg.19 ]*****/
910 /*****/
911
912 /**
913 * S E T   S W I T C H I N G   M O D E   ( I N T )
914 * Description: Switch between color and statistic packets
915 *              in tracking modes. 0- off 1-on
916 * Created:     June 15th, 2002.
917 * Paramaters: switchingmodeI
918 * Returns:     int (SUCCESS or FAILURE )
919 * Comments:
920 *
921 * Modified:    June 15th, 2002.
922 */
923 int SetSwitchingMode(int switchingmodeI)
924 {
925     char cmdtosend [5];
926     //if(switchingmodeI != 0 || switchingmodeI != 1) //avoids other modes
927     // return FAILURE;
928     switchingmode = switchingmodeI;
929     sprintf(cmdtosend,"SM %d\r",switchingmode);
930     ViconMagic(cmdtosend);
931     return ViconTest();
932
933 }//end of SetSwitchingMode
934 int GetSwitchingMode(void)
935 {
936     return switchingmode;
937
938 }//end of GetSwitchingMode

```

```

939
940
941
942 /**
943 * SET WINDOW SIZE (INT INT INT INT)
944 * Description: Simply sets the window size of the camera
945 * Created: June 15th, 2002.
946 * Paramaters: 2 coordinates top-left bottom-right
947 * Returns: int (SUCCESS or FAILURE )
948 * Comments:
949 *
950 *
951 * Modified: June 15th, 2002.
952 */
953 int SetWindowSize(int x, int y, int x1, int y1)
954 {
955     char cmdtosend [20];
956     if((x>x1 || y>y1) || ((x<1 || y<1) || (y1>143 || x1>80)))
957         return FAILURE;
958
959     sprintf(cmdtosend, "SW %d %d %d %d\r", x, y, x1, y1);
960
961     ViconMagic(cmdtosend);
962     return ViconTest();
963
964 } //end of SetWindowSize
965
966
967
968 /*****
969 /*****[ pg.20 ]*****/
970 /*****
971
972 /**
973 * T R A C K W I N D O W ( )
974 * Description: Tracks prominent color in window
975 * Created: June 5th, 2002.
976 * Paramaters: Void
977 * Returns: int
978 * Comments: Not to be used. Not beneficial for ViCoN-Bot
979 * Modified: June 15th, 2002.
980 */
981 int TrackWindow(void)
982 {
983     char cmdtosend [3] = "TW\r";
984     ViconMagic(cmdtosend);
985     return UpdatePackets();
986
987 } //end of TrackWindow()
988
989
990
991 /**
992 * T R A C K C O L O R ( I N T )
993 * Description: Tracks a given color
994 * Created: June 5th, 2002.
995 * Paramaters:
996 * Returns:
997 * Comments:
998 * Modified: June 15th, 2002.
999 */
1000 int TrackColor(Color c) //array of color values
1001 {
1002     char cmdtosend [20];
1003     //need to check array size and valid values
1004     sprintf(cmdtosend, "TC %d %d %d %d %d %d \r", c.rmin, c.rmax, c.gmin, c.gmax, c
        .bmin, c.bmax);

```

```

1005     ViconMagic2(cmdtosend);
1006     UpdatePackets();
1007     //return UpdatePackets();
1008
1009 }//end of TrackColor
1010
1011
1012
1013 /*****
1014 /*****[ THE END ]*****/
1015 /*****
1016 /**
1017  * INTERRUPT SERVICE ROUTINE()
1018  * Description: Handles all interrupt service routines
1019  * Created:     June 5th, 2002.
1020  * Paramaters:
1021  * Returns:     int
1022  * Comments:    Not to be used. Not beneficial for ViCoN-Bot
1023  * Modified:    June 11th, 2002.
1024  */
1025 void InterruptServiceRoutine(int context)
1026 {
1027     UARTISRContext *c = (UARTISRContext *)context;
1028     int status;
1029     int rxChar;
1030     //char joy;
1031     status = c->uart->np_uartstatus;
1032     rxChar = c->uart->np_uartrxdata;
1033     c->uart->np_uartstatus = 0; // clear the interrupt condition
1034
1035     if(status & np_uartstatus_rrdy_mask ){
1036         if(rxChar == 13)
1037             rxChar = 32;
1038         if(crindex>=0)
1039             lastresponse[crindex]= (char)rxChar;
1040         //joy = (char)rxChar;
1041         //lastresponse[crindex]= joy;
1042         crindex++;
1043     }
1044     c->rxChar = rxChar; // save the character for later use.
1045
1046 }//end of InterruptServiceRoutine
1047
1048
1049
1050 /**
1051  * ENABLE UART INTERRUPT()
1052  * Description: Enables Uart as an interrupt
1053  * Created:     June 5th, 2002.
1054  * Paramaters: int
1055  * Returns:     void
1056  * Comments:
1057  * Modified:    June 7th, 2002.
1058  */
1059 void EnableUartInterrupt(void)
1060 {
1061     gC.uart = (np_uart *)na_uart1;
1062     nr_installuserisr(na_uart1_irq,InterruptServiceRoutine,(long)&gC);
1063     {
1064         np_uart *uart;
1065         uart = na_uart1;
1066         uart->np_uartcontrol = np_uartcontrol_irrdy_mask;
1067
1068     }//end of thread. don't understand this
1069
1070 }//end of EnableUartInterrupt()
1071

```

```
1072
1073
1074 /**
1075  * DISABLE UART INTERRUPT()
1076  * Description: Disables Uart as an interrupt
1077  * Created:     June 5th, 2002.
1078  * Paramaters: int
1079  * Returns:     void
1080  * Comments:
1081  * Modified:    June 7th, 2002.
1082  */
1083 void DisableUartInterrupt(void)
1084 {
1085     np_uart *uart;
1086     uart = na_uart1;
1087     uart->np_uartcontrol = 0;
1088     nr_installuserisr(na_uart1_irq,0,0);
1089
1090 }//end of DisableUartInterrupt()
1091
1092
1093
```

```

1 #include "nios.h"
2 #include "vicon.h"
3 #include "pio_lcd16207.h"
4 /*-----
5 *
6 * Author(s):    Kwabena Asare Bosompem
7 *              Andre Moore
8 *              Jeff Vickers
9 *              Kevin Walker
10 * Created:     June 10th, 2002
11 * Modified:    July 18th, 2002
12 * Description: Controls the different states needed to use the
13 *              camera as tracking utility for the robot
14 *-----*/
15
16
17 char state;
18 int mass = 8;          // Value to be used to set the Middle Mass
19 int count = 0;
20 int idle_count = 0; // Global used in camera idle
21 Color color_k;
22 Color color_checkup;
23 int packets = 2;      // Number of packets used in Tracking()
24 np_pio *direction    = na_direction_pio;
25 np_pio *speed        = na_speed_pio;
26 int TALK = 0;        // 1 is for DEBUG mode
27
28 int direction0 = 0; //Stop or Brake?
29 int direction1 = 1; //-45
30 int direction2 = 2; //-20
31 int direction3 = 3; //-10
32 int direction4 = 4; //0
33 int direction5 = 5; //10
34 int direction6 = 6; //20
35 int direction7 = 7; //45
36
37 int speed0 = 0; //Reverse 1/8
38 int speed1 = 1; //Reverse 1/4
39 int speed2 = 2; //Stop
40 int speed3 = 3; //Neutral
41 int speed4 = 4; //Foreward 1/8
42 int speed5 = 5; //Foreward 1/4
43 int speed6 = 6; //Foreward 1/2
44 int speed7 = 7; //Foreward 1
45
46 extern int pollmode;
47 extern SdataPacket spacket;
48 extern CdataPacket cpacket;
49 extern MdataPacket mpacket;
50 extern NdataPacket npacket;
51
52
53
54 //Color PINK = {220, 240, 10, 60, 10, 40};
55 //Color YELLOW = {235, 240, 235, 240, 40, 80};
56 //Color GREEN = {75, 120, 140, 240, 40, 100};
57
58
59 typedef struct {
60     int TurnDirection; // number that will hold the direction of the servo
61     int DriveSpeed;    // number that will hold the speed of the servo
62 }ServoValue;
63 ServoValue myServoValue;
64
65
66 /**
67 * M A I N ( V O I D )

```

```

68 * Description: Main function for the state machine.  Initializes the LCD and
    starts the
69 *           first state
70 * Created:   June 15th, 2002.
71 * Paramaters:
72 * Returns:
73 * Comments:
74 * Modified:
75 */
76 int main(void){
77
78     np_pio *pio = na_button_pio;    //choose the sw buttons as input
79
80     //set up interrupt handler. this responds to the pushbuttons
81     nr_installuserisr(na_button_pio_irq,MyPIO_ISR,(int)pio);
82
83     nr_pio_lcdinit(na_lcd_pio);    // Initialize LCD
84
85
86     pio->np_pioedgecapture = 0;    // clear any existing IRQ
87     pio->np_piodirection = 0;    // all input
88     pio->np_piointerruptmask = 0xff; // they all generate irq's
89
90     nr_pio_lcdwritescreen("GaTech ViCoN-Bot"); //Welcome Message
91     nr_delay(2000);
92     CameraStart();
93
94 }
95
96
97 /**
98 * C A M E R A S T A R T ( V O I D )
99 * Description: Set's the cameras initialization (MM and Poll)
100 * Created:    June 15th, 2002.
101 * Paramaters:
102 * Returns:
103 * Comments:
104 * Modified:
105 */
106 int CameraStart(void){
107
108     int ans1;
109     int ans2;
110     int ans3;
111     char tempc[32];
112
113     nr_pio_lcdwritescreen("Camera Initialized");
114     nr_delay(1000);
115
116     do{
117     ResetCamera();
118     ans2 = SetPollMode(1);
119     ans1 = SetMiddleMass(mass);
120     ans3 = SetColorMode(44);
121     }while (ans1 == FAILURE);
122
123     sprintf(tempc, "%d %d %d", ans1, ans2, ans3);
124     nr_pio_lcdwritescreen(tempc);
125     nr_delay(1000);
126     SetSpeed(NEUTRAL);    // Set the motor to neutral
127     SetDirection(STRAIGHT); // Set the wheels straight
128     color_checkup = PINK;
129     FindColor();
130 }
131
132
133 /**

```

```

134 * F I N D C O L O R ( V O I D )
135 * Description: Set's the camera in a loop to find an initial color to track
136 * Created:     June 15th, 2002.
137 * Paramaters:
138 * Returns:
139 * Comments:
140 * Modified:
141 */
142 int FindColor(void){
143
144     char nicon[32];
145     char kevin[32];
146     int count2 = 0;
147
148     int conf[3] = {0, 0, 0};
149     int conf_value;
150
151     if (count != 2)
152     {
153         count++;
154     }
155     else
156     {
157         count = 0;
158     }
159     SetSpeed(NEUTRAL);
160     SetDirection(STRAIGHT);
161
162     // Add an additional case for another color
163     switch (count)
164     {
165         case(1):
166             //color_k = PINK;
167             color_k = GREEN;
168
169             if (TALK == 1) {
170                 nr_pio_lcdwritescreen("Color is now     GREEN");
171                 nr_delay(1100);}
172
173             break;
174         case(2):
175             //color_k = YELLOW;
176             color_k = PINK;
177
178             if (TALK == 1) {
179                 nr_pio_lcdwritescreen("Color is now     PINK");
180                 nr_delay(1100);}
181
182             break;
183         case(0):
184             color_k = PINK;
185
186
187             if (TALK == 1){
188                 nr_pio_lcdwritescreen("Color is now     YELLOW");
189                 nr_delay(1100);}
190
191             break;
192         default:
193             break;
194     }
195
196 // This do loop controls how many packets are need to find if a color is present
197 // Testing has shown that only one packet is needed and exhibits the best
198 // functionality
199 //do{

```

```

200
201 TrackColor(color_k);
202
203 //      conf[count2] = npacket.confidence;   Use if more than one packet is used
204
205
206 if (TALK == 1) {
207     sprintf(kevin, "Confidence: %d   Pixels:%d", npacket.confidence, npacket.
        pixels);
208     nr_pio_lcdwritescree(kevin);
209     nr_delay(1000);}
210
211 // Use if tracking more than one packet
212 //     if (count2 == 2)
213 //     {
214 //         conf_value = ((conf[0] + conf[1] + conf[2]) / 3);
215 //
216 //         if (TALK == 1) {
217 //             sprintf(kevin, "%d %d %d", conf[0], conf[1], conf[2]);
218 //             nr_pio_lcdwritescree(kevin);
219 //             nr_delay(800);}
220 //
221 //             count2 = count2 + 1;
222 //         }
223 //     else
224 //     {
225 //         count2 = count2 + 1;
226 //     }
227 // }
228 //
229 // }while(count2 < 3);
230
231
232 count2=0;
233 conf_value=npacket.confidence;
234
235 if (TALK == 1) {
236     sprintf(kevin, "average confidence: %d", conf_value);
237     nr_pio_lcdwritescree(kevin);
238     nr_delay(1000);}
239
240 if (conf_value > 20)
241 {
242     if (TALK == 1) {
243         nr_pio_lcdwritescree("Tracking State");
244         nr_delay(500);}
245     TrackState();
246
247 }
248 else
249 {
250     if (TALK == 1) {
251         nr_pio_lcdwritescree("Finding Color");
252         nr_delay(500);}
253     FindColor();
254
255 }
256 }
257
258 /**
259 * TRACKING STATE (INT)
260 * Description: Tracks a color
261 * Created:     June 15th, 2002.
262 * Paramaters:
263 * Returns:
264 * Comments:
265 * Modified:

```

```

266 */
267 int TrackState(void){
268
269     char kevin[32];
270
271     int countk      = 0;
272     int checkup     = 0;
273
274     int state_conf  = 0;
275     int state_mx    = 0;
276     int state_my    = 0;
277     int state_x1    = 0;
278     int state_y1    = 0;
279     int state_x2    = 0;
280     int state_y2    = 0;
281     int state_pixel = 0;
282     int state_confidence = 0;
283
284
285     if (TALK == 1) {
286         nr_pio_lcdwritescreen("Tracking");
287         nr_delay(500);
288     }
289
290     if (CompareColors(color_k, color_checkup) == SUCCESS){
291         TrackColor(color_checkup);
292         checkup = npacket.confidence;
293         SetDirection(STRAIGHT);
294         //SetSpeed(EIGHTH_FORWARD);
295         if (checkup <= 20) {FindColor();};
296     }
297
298     do{
299         TrackColor(color_k);
300         state_conf      += npacket.confidence;
301         state_mx        += npacket.mx;
302         state_my        += npacket.my;
303         state_x1        += npacket.x1;
304         state_x2        += npacket.x2;
305         state_y1        += npacket.y1;
306         state_y2        += npacket.y2;
307         state_pixel     += npacket.pixels;
308         countk++;
309
310     }while(countk < packets);
311     state_conf  = (state_conf/packets);
312     state_mx    = (state_mx/packets);
313     state_my    = (state_my/packets);
314     state_pixel = (state_pixel/packets);
315
316     if (TALK == 1) {
317         sprintf(kevin, "T::: con:%d  pix:%d", state_conf, state_pixel);
318         nr_pio_lcdwritescreen(kevin);
319         nr_delay(800);}
320
321     if (state_conf > 20) {DataControl(state_conf, state_mx, state_my, state_pixel
322         );} else {FindColor();}
323
324
325
326 /**
327  * C O M P A R E C O L O R S ( S T R U C T , S T R U C T )
328  * Description: Compares two colors
329  * Created:     June 15th, 2002.
330  * Paramaters: Two structures (Color)
331  * Returns:    int (SUCCESS or FAILURE)

```

```

332 * Comments:
333 * Modified:
334 */
335 int CompareColors(Color c1, Color c2)
336 {
337     if(c1.rmin != c2.rmin)
338         return FAILURE;
339     else if (c1.bmin != c2.bmin)
340         return FAILURE;
341     else if (c1.gmin != c2.gmin)
342         return FAILURE;
343     else if (c1.rmax != c2.rmax)
344         return FAILURE;
345     else if (c1.gmax != c2.gmax)
346         return FAILURE;
347     else if (c1.bmax != c2.bmax)
348         return FAILURE;
349     else
350         return SUCCESS;
351 }
352 //end of CompareColors
353
354
355
356
357 /**
358 * DATA CONTROL ( INT , INT , INT , INT )
359 * Description: DataControl will serve as a function to calculate the distance,
360 *             direction, and speed that the ViConBot will need to control the
361 *             servos.
362 *             Created: June 15th, 2002.
363 * Paramaters:
364 * Returns:
365 * Comments:
366 * Modified:
367 */
368 int DataControl(int con, int m1, int m2, int pix){
369     char kevinw[32];
370
371     if (CompareColors(color_k, PINK)== SUCCESS){
372
373         myServoValue.TurnDirection = SetViconDirection(m1);
374
375         if (pix >= 180)
376             myServoValue.DriveSpeed = EIGHTH_FORWARD;
377         else if ((pix < 180) && (pix >= 25))
378             myServoValue.DriveSpeed = EIGHTH_FORWARD;
379         else
380             myServoValue.DriveSpeed = EIGHTH_FORWARD;
381
382
383
384     }else if (CompareColors(color_k, GREEN)== SUCCESS){
385
386         if (pix >= 200){
387             myServoValue.DriveSpeed = EIGHTH_REVERSE;
388             myServoValue.TurnDirection = SetViconDirection();}
389         else if ((pix < 200) && (pix >= 125)){
390             myServoValue.DriveSpeed = EIGHTH_FORWARD;
391             myServoValue.TurnDirection = LEFT_45;}
392         else if ((pix < 125) && (pix >= 25)){
393             myServoValue.DriveSpeed = EIGHTH_FORWARD;
394             myServoValue.TurnDirection = LEFT_20;}
395         else {
396             myServoValue.DriveSpeed = EIGHTH_FORWARD;
397             myServoValue.TurnDirection = SetViconDirection();}

```

```

398
399 }else{
400
401     myServoValue.TurnDirection = SetViconDirection(m1);
402
403     if (pix >= 80)
404         myServoValue.DriveSpeed = NEUTRAL;
405     else if ((pix < 80) && (pix >= 40))
406         myServoValue.DriveSpeed = EIGHTH_FORWARD;
407     else
408         myServoValue.DriveSpeed = EIGHTH_FORWARD;
409 }
410
411
412 if (TALK == 1) {
413     sprintf(kevinw, "Direction:%d .:. Speed:%d", myServoValue.TurnDirection,
414             myServoValue.DriveSpeed);
415     nr_pio_lcdwritescreen(kevinw);
416     nr_delay(500);
417 }
418 SetDirection(myServoValue.TurnDirection);
419 SetSpeed(myServoValue.DriveSpeed);
420 TrackState();
421 }
422
423
424
425 int SetViconDirection(int direc_value){
426
427
428     if (direc_value >= 75)
429
430         return RIGHT_20;
431
432     else if ((direc_value < 75) && (direc_value >= 65))
433
434         return RIGHT_20;
435
436     else if ((direc_value < 70) && (direc_value >= 60))
437
438         return RIGHT_10;
439
440     else if ((direc_value < 60) && (direc_value >= 40))
441
442         return STRAIGHT;
443
444     else if ((direc_value < 40) && (direc_value >= 30))
445
446         return LEFT_10;
447
448     else if ((direc_value < 30) && (direc_value >= 20))
449
450         return LEFT_20;
451
452     else
453
454         return LEFT_45;
455
456 }//End of SetViconDirection
457
458
459
460 /**
461  * R E S E T   ( V O I D )
462  * Description: Reset will reset the camera and send it back to the initial state
463                of CameraStart

```

```

463 * Created:      June 15th, 2002.
464 * Paramaters:
465 * Returns:
466 * Comments:
467 * Modified:
468 */
469 int Reset(void){
470     int startover;
471     do{
472         startover = ResetCamera();
473     }while(startover == FAILURE);
474     CameraStart();
475 }
476
477 /**
478 * I D L E R O B O T  ( I N T )
479 * Description: Sets the robot in an idle state until it receives a second second
480 *               input from the nios
481 *               pushbutton
482 * Created:      July 17, 2002
483 * Paramaters:
484 * Returns:
485 * Comments:
486 * Modified:
487 */
487 int IdleRobot(int value){
488
489     idle_count += value;
490
491     if (idle_count == 2)
492     {
493         idle_count = 0;
494         FindColor();
495     }else
496     {
497         SetDirection(STRAIGHT);
498         SetSpeed(NEUTRAL);
499     };
500
501 }
502
503
504
505 void MyPIO_ISR(int context)
506 {
507     np_pio *pio = (np_pio *)context;
508     char s[32];
509     int buttons;
510     pio->np_pioedgecapture = 0; //reset interrupt
511     buttons = pio->np_piodata; //get button pressed
512     sprintf(s, "Button Pressed:    %1x", buttons & 0xf);
513     if(s[19]!='f')
514     {
515         nr_pio_lcdwritescreen(s);
516         nr_delay(500);
517
518         if (s[19]=='b'){
519             nr_pio_lcdwritescreen("Return to CameraStart");
520             nr_delay(1500);
521             CameraStart();
522         }
523         else if(s[19]=='7'){
524             nr_pio_lcdwritescreen("Start Travelen Man!");
525             nr_delay(1500);
526             TravelFind();
527         }
528         else if(s[19]=='d'){

```

```
529         nr_pio_lcdwritescreen("Set Robot Idle");
530         nr_delay(1500);
531         IdleRobot(1);
532     }
533     else if(s[19]=='e'){
534         nr_pio_lcdwritescreen("Reset");
535         nr_delay(1500);
536         //if(currentmode == 1)
537         // currentmode = 0;
538         //else
539         // currentmode = 1;
540         //SetPollMode(currentmode);
541         Reset();
542     }
543 }
544 }
545 }
546 }
547 }
548 }
```

```

1 -----
2 -----   PWM for Speed Control   -----
3 -----
4
5 LIBRARY IEEE;
6 USE IEEE.STD_LOGIC_1164.ALL;
7 USE IEEE.STD_LOGIC_ARITH.ALL;
8 USE IEEE.STD_LOGIC_UNSIGNED.ALL;
9
10 ENTITY speed_control IS
11     PORT      (clock_100kHz      : IN  STD_LOGIC;
12               speed              : IN  STD_LOGIC_VECTOR( 2 DOWNTO 0 );
13               speed_pwm          : OUT STD_LOGIC);
14
15 END speed_control;
16
17 ARCHITECTURE servo_controller OF speed_control IS
18     SIGNAL count_motor: STD_LOGIC_VECTOR( 10 DOWNTO 0 );
19 BEGIN
20     PROCESS
21     BEGIN
22
23 -- Count_motor is a 20ms timer => [ 0 , 2000 ] based on a true 100kHz clock --
24     WAIT UNTIL clock_100kHz 'EVENT AND clock_100kHz = '1';
25     IF count_motor /= 2000 THEN
26         count_motor <= count_motor + 1;
27     ELSE
28         count_motor <= "0000000000";
29     END IF;
30
31 -- Speed Setting #1 of 8 => BRAKE or OFF = 000 --
32     IF count_motor >= 1700 AND count_motor < 1838 THEN
33         IF speed = "000" THEN
34             speed_pwm <= '0';
35         ELSE
36             speed_pwm <= '1';
37         END IF;
38
39 -- At this point the pulse is already 1 ms long --
40 -- Decide how long to allow pulse to remain high. [ 1ms , 2ms ] --
41
42 -- Speed Setting #2 of 8 => Quarter Reverse (CW) --
43     ELSIF count_motor >= 1838 AND count_motor < 1844 THEN
44         IF speed /= "000" THEN
45             CASE speed IS
46             WHEN "010" =>
47                 speed_pwm <= '0';
48             WHEN "001" =>
49                 speed_pwm <= '1';
50             WHEN "011" =>
51                 speed_pwm <= '1';
52             WHEN "100" =>
53                 speed_pwm <= '1';
54             WHEN "101" =>
55                 speed_pwm <= '1';
56             WHEN "110" =>
57                 speed_pwm <= '1';
58             WHEN "111" =>
59                 speed_pwm <= '1';
60             WHEN OTHERS => NULL;
61             END CASE;
62         ELSE
63             speed_pwm <= '0';
64         END IF;
65
66 -- Speed Setting #3 of 8 => Eighth Reverse (CW) --
67     ELSIF count_motor >= 1844 AND count_motor < 1850 THEN

```

```

68         IF speed /="000" THEN
69             CASE speed IS
70                 WHEN "010" =>
71                     speed_pwm <= '0';
72                 WHEN "001" =>
73                     speed_pwm <= '0';
74                 WHEN "011" =>
75                     speed_pwm <= '1';
76                 WHEN "100" =>
77                     speed_pwm <= '1';
78                 WHEN "101" =>
79                     speed_pwm <= '1';
80                 WHEN "110" =>
81                     speed_pwm <= '1';
82                 WHEN "111" =>
83                     speed_pwm <= '1';
84                 WHEN OTHERS => NULL;
85             END CASE;
86         ELSE
87             speed_pwm <= '0';
88         END IF;
89
90 -- Speed Setting #4 of 8 => Neutral (1.5 ms pulse) --
91     ELSIF count_motor >= 1850 AND count_motor < 1856 THEN
92         IF speed /="000" THEN
93             CASE speed IS
94                 WHEN "010" =>
95                     speed_pwm <= '0';
96                 WHEN "001" =>
97                     speed_pwm <= '0';
98                 WHEN "011" =>
99                     speed_pwm <= '0';
100                WHEN "100" =>
101                    speed_pwm <= '1';
102                WHEN "101" =>
103                    speed_pwm <= '1';
104                WHEN "110" =>
105                    speed_pwm <= '1';
106                WHEN "111" =>
107                    speed_pwm <= '1';
108                WHEN OTHERS => NULL;
109            END CASE;
110        ELSE
111            speed_pwm <= '0';
112        END IF;
113
114 -- Speed Setting #5 of 8 => Eighth Forward (CCW) --
115     ELSIF count_motor >= 1856 AND count_motor < 1863 THEN
116         IF speed /="000" THEN
117             CASE speed IS
118                 WHEN "010" =>
119                     speed_pwm <= '0';
120                 WHEN "001" =>
121                     speed_pwm <= '0';
122                 WHEN "011" =>
123                     speed_pwm <= '0';
124                 WHEN "100" =>
125                     speed_pwm <= '0';
126                 WHEN "101" =>
127                     speed_pwm <= '1';
128                 WHEN "110" =>
129                     speed_pwm <= '1';
130                 WHEN "111" =>
131                     speed_pwm <= '1';
132                 WHEN OTHERS => NULL;
133             END CASE;
134         ELSE

```

```

135         speed_pwm <= '0';
136     END IF;
137
138 -- Speed Setting #6 of 8 => Quarter Forward (CCW) --
139     ELSIF count_motor >= 1863 AND count_motor < 1875 THEN
140         IF speed /= "000" THEN
141             CASE speed IS
142                 WHEN "010" =>
143                     speed_pwm <= '0';
144                 WHEN "001" =>
145                     speed_pwm <= '0';
146                 WHEN "011" =>
147                     speed_pwm <= '0';
148                 WHEN "100" =>
149                     speed_pwm <= '0';
150                 WHEN "101" =>
151                     speed_pwm <= '0';
152                 WHEN "110" =>
153                     speed_pwm <= '1';
154                 WHEN "111" =>
155                     speed_pwm <= '1';
156                 WHEN OTHERS => NULL;
157             END CASE;
158         ELSE
159             speed_pwm <= '0';
160         END IF;
161
162 -- Speed Setting #7 of 8 => Half Forward (CCW) --
163     ELSIF count_motor >= 1875 AND count_motor < 1900 THEN
164         IF speed /= "000" THEN
165             CASE speed IS
166                 WHEN "010" =>
167                     speed_pwm <= '0';
168                 WHEN "001" =>
169                     speed_pwm <= '0';
170                 WHEN "011" =>
171                     speed_pwm <= '0';
172                 WHEN "100" =>
173                     speed_pwm <= '0';
174                 WHEN "101" =>
175                     speed_pwm <= '0';
176                 WHEN "110" =>
177                     speed_pwm <= '0';
178                 WHEN "111" =>
179                     speed_pwm <= '1';
180                 WHEN OTHERS => NULL;
181             END CASE;
182
183 -- Speed Setting #8 of 8 => Full Forward (CCW) --
184         ELSE
185             speed_pwm <= '0';
186         END IF;
187
188     ELSE
189         speed_pwm <= '0';
190     END IF;
191
192     END PROCESS;
193 END servo_controller;
194
195 -----
196 --All of the case statements with <=1 and reoccurring <=0 could be deleted.--
197 -----

```

```

1 -----
2 -----   PWM for Servo Direction Control   -----
3 -----
4
5 LIBRARY IEEE;
6 USE IEEE.STD_LOGIC_1164.ALL;
7 USE IEEE.STD_LOGIC_ARITH.ALL;
8 USE IEEE.STD_LOGIC_UNSIGNED.ALL;
9
10 ENTITY direction_control IS
11     PORT      (clock_100kHz      : IN  STD_LOGIC;
12               position          : IN  STD_LOGIC_VECTOR( 2 DOWNTO 0 );
13               direction_pwm     : OUT STD_LOGIC);
14
15     END direction_control;
16
17 ARCHITECTURE servo_controller OF direction_control IS
18     SIGNAL count_motor: STD_LOGIC_VECTOR( 10 DOWNTO 0 );
19 BEGIN
20     PROCESS
21     BEGIN
22 -- Count_motor is a 20ms timer => [ 0 , 2000 ] --
23     WAIT UNTIL clock_100kHz 'EVENT AND clock_100kHz = '1';
24     IF count_motor /= 2000 THEN
25         count_motor <= count_motor + 1;
26     ELSE
27         count_motor <= "00000000000";
28     END IF;
29
30 -- Direction Setting #1 of 8 => SIGNAL OFF = 000 --
31     IF count_motor >= 1700 AND count_motor < 1818 THEN
32         IF position = "000" THEN
33             direction_pwm <= '0';
34         ELSE
35             direction_pwm <= '1';
36         END IF;
37
38 -- At this point the pulse is already 1 ms long --
39 -- Decide how long to allow pulse to remain high. [ 1ms , 2ms ] --
40
41 -- Direction Setting #2 of 8 => 45 degrees LEFT or CW from center (1830) --
42 -- After trial and error the usable value was determined to be 1818 --
43     ELSIF count_motor >= 1818 AND count_motor < 1830 THEN
44         IF position /= "000" THEN
45             CASE position IS
46             WHEN "011" =>
47                 direction_pwm <= '0';
48             WHEN "010" =>
49                 direction_pwm <= '1';
50             WHEN "001" =>
51                 direction_pwm <= '1';
52             WHEN "100" =>
53                 direction_pwm <= '1';
54             WHEN "101" =>
55                 direction_pwm <= '1';
56             WHEN "110" =>
57                 direction_pwm <= '1';
58             WHEN "111" =>
59                 direction_pwm <= '1';
60             WHEN OTHERS => NULL;
61             END CASE;
62         ELSE
63             direction_pwm <= '0';
64         END IF;
65
66 -- Direction Setting #3 of 8 => 20 degrees LEFT or CW from center (1845) --

```

```

67 -- After trial and error the usable value was determined to be 1830 --
68     ELSIF count_motor >= 1830 AND count_motor < 1838 THEN
69         IF position /= "000" THEN
70             CASE position IS
71                 WHEN "011" =>
72                     direction_pwm <= '0';
73                 WHEN "010" =>
74                     direction_pwm <= '0';
75                 WHEN "001" =>
76                     direction_pwm <= '1';
77                 WHEN "100" =>
78                     direction_pwm <= '1';
79                 WHEN "101" =>
80                     direction_pwm <= '1';
81                 WHEN "110" =>
82                     direction_pwm <= '1';
83                 WHEN "111" =>
84                     direction_pwm <= '1';
85                 WHEN OTHERS => NULL;
86             END CASE;
87         ELSE
88             direction_pwm <= '0';
89         END IF;
90
91 -- Direction Setting #4 of 8 => 10 degrees LEFT or CW from center (1850) --
92 -- After trial and error the usable value was determined to be 1838 --
93     ELSIF count_motor >= 1838 AND count_motor < 1845 THEN
94         IF position /= "000" THEN
95             CASE position IS
96                 WHEN "011" =>
97                     direction_pwm <= '0';
98                 WHEN "010" =>
99                     direction_pwm <= '0';
100                WHEN "001" =>
101                    direction_pwm <= '0';
102                WHEN "100" =>
103                    direction_pwm <= '1';
104                WHEN "101" =>
105                    direction_pwm <= '1';
106                WHEN "110" =>
107                    direction_pwm <= '1';
108                WHEN "111" =>
109                    direction_pwm <= '1';
110                WHEN OTHERS => NULL;
111            END CASE;
112        ELSE
113            direction_pwm <= '0';
114        END IF;
115
116 -- Direction Setting #5 of 8 => 0 degrees or center (1856) --
117 -- After trial and error the usable value was determined to be 1845 --
118     ELSIF count_motor >= 1845 AND count_motor < 1862 THEN
119         IF position /= "000" THEN
120             CASE position IS
121                 WHEN "011" =>
122                     direction_pwm <= '0';
123                 WHEN "010" =>
124                     direction_pwm <= '0';
125                 WHEN "001" =>
126                     direction_pwm <= '0';
127                 WHEN "100" =>
128                     direction_pwm <= '0';
129                 WHEN "101" =>
130                     direction_pwm <= '1';
131                 WHEN "110" =>
132                     direction_pwm <= '1';

```

```

133         WHEN "111" =>
134             direction_pwm <= '1';
135         WHEN OTHERS => NULL;
136         END CASE;
137     ELSE
138         direction_pwm <= '0';
139     END IF;
140
141 -- Direction Setting #6 of 8 => 10 degrees RIGHT or CCW from center (1862) --
142     ELSIF count_motor >= 1862 AND count_motor < 1868 THEN
143         IF position /="000" THEN
144             CASE position IS
145                 WHEN "011" =>
146                     direction_pwm <= '0';
147                 WHEN "010" =>
148                     direction_pwm <= '0';
149                 WHEN "001" =>
150                     direction_pwm <= '0';
151                 WHEN "100" =>
152                     direction_pwm <= '0';
153                 WHEN "101" =>
154                     direction_pwm <= '0';
155                 WHEN "110" =>
156                     direction_pwm <= '1';
157                 WHEN "111" =>
158                     direction_pwm <= '1';
159                 WHEN OTHERS => NULL;
160             END CASE;
161         ELSE
162             direction_pwm <= '0';
163         END IF;
164
165 -- Direction Setting #7 of 8 => 20 degrees CCW from center (1868) --
166     ELSIF count_motor >= 1868 AND count_motor < 1882 THEN
167         IF position /="000" THEN
168             CASE position IS
169                 WHEN "011" =>
170                     direction_pwm <= '0';
171                 WHEN "010" =>
172                     direction_pwm <= '0';
173                 WHEN "001" =>
174                     direction_pwm <= '0';
175                 WHEN "100" =>
176                     direction_pwm <= '0';
177                 WHEN "101" =>
178                     direction_pwm <= '0';
179                 WHEN "110" =>
180                     direction_pwm <= '0';
181                 WHEN "111" =>
182                     direction_pwm <= '1';
183                 WHEN OTHERS => NULL;
184             END CASE;
185         ELSE
186             direction_pwm <= '0';
187         END IF;
188
189 -- Direction Setting #8 of 8 => 45 degrees CCW from center (1882) --
190     ELSE
191         direction_pwm <= '0';
192     END IF;
193
194     END PROCESS;
195 END servo_controller;
196 -----
197 --All of the case statements with <=1 and reoccurring <=0 could be deleted.--
198 -----

```