APPLICATION NOTE

Charmed Labs

Xport 2.0 RC Servo Configuration

Version 1.0, June 12, 2003

Rich LeGrand (rich@charmedlabs.com)

Summary

This application note explains how to use the RC Servo configuration to control up to 62 RC servos with the Xport.

Introduction

"RC servos" were originally designed to control radio-controlled models such as airplanes or cars. Because of their low cost and wide availability, they are also commonly used as microprocessor-controlled actuators. The RC servo has a high-torque output shaft that can be positioned accurately by supplying a pulse width modulated (PWM) signal. The width of the pulse determines the commanded position of the servo. The RC servo configuration for the Xport synthesizes the PWM signals required for simultaneous control of up to 62 servos.

Usage

Each PWM signal is commanded by an 8-bit value. For convenience, two 8-bit PWM values are combined into 16-bit registers. Each 8-bit PWM value determines the pulse width of the corresponding PWM signal and hence the commanded servo position. **Table 1** below details these registers and their mapping.

Since each RC-servo tends to require a slightly different pulse-width for the same output shaft position when compared to another RC-servo of a different manufacturer, the supplied pulse width can range from 2.32ms (corresponding to a commanded PWM value of 0) to 0.37ms (corresponding to a commanded PWM value of 255). These pulse widths typically correspond to positions that lie outside the possible range of movement for most servos. Thus, it is recommended that the PWM value be limited in software to correspond to the actual or desired limits of servo travel.

Table 1: RC-Servo Register Mapping

		Register contents (individual bytes shown)	
Name	Address	Most significant byte (D15→D8)	Least significant byte (D7→D0)
RCS0	0x9ffc400	PA1	PA0
RCS1	0x9ffc402	PA3	PA2
RCS2	0x9ffc404	PA5	PA4
RCS3	0x9ffc406	PA7	PA6
RCS4	0x9ffc408	PA9	PA8
RCS5	0x9ffc40a	PA11	PA10
RCS6	0x9ffc40c	PA13	PA12
RCS7	0x9ffc40e	PA15	PA14
RCS8	0x9ffc410	PA17	PA16
RCS9	0x9ffc412	PA19	PA18
RCS10	0x9ffc414	PA21	PA20
RCS11	0x9ffc416	PA23	PA22
RCS12	0x9ffc418	PA25	PA24
RCS13	0x9ffc41a	PA27	PA26

RCS14	0x9ffc41c	PA29	PA28
RCS15	0x9ffc41e	PB0	PA30
RCS16	0x9ffc420	PB2	PB1
RCS17	0x9ffc422	PB4	PB3
RCS18	0x9ffc424	PB6	PB5
RCS19	0x9ffc426	PB8	PB7
RCS20	0x9ffc428	PB10	PB9
RCS21	0x9ffc42a	PB12	PB11
RCS22	0x9ffc42c	PB14	PB13
RCS23	0x9ffc42e	PB16	PB15
RCS24	0x9ffc430	PB18	PB17
RCS25	0x9ffc432	PB20	PB19
RCS26	0x9ffc434	PB22	PB21
RCS27	0x9ffc436	PB24	PB23
RCS28	0x9ffc438	PB26	PB25
RCS29	0x9ffc43a	PB28	PB27
RCS30	0x9ffc43c	PB30	PB29

Where PAn = the nth I/O signal for PA, PBn = nth I/O signal for PB – see the *Connector Pinouts* section in the Xport 2.0 User's Manual.

For example, to set the PWM channel corresponding to I/O signal P0 to the centermost position, set RCS0 as follows:

```
*((volatile unsigned short *)0x9ffc400) = 0x0080;
```

To save FPGA logic, reading the registers has been disabled and will result in an undefined value when read.

Example Circuitry

Figure 1 below shows a recommended connection diagram. It requires a separate 4.5 to 6V power supply for the servos. The power supply, Xport and servos should share the same ground connection.

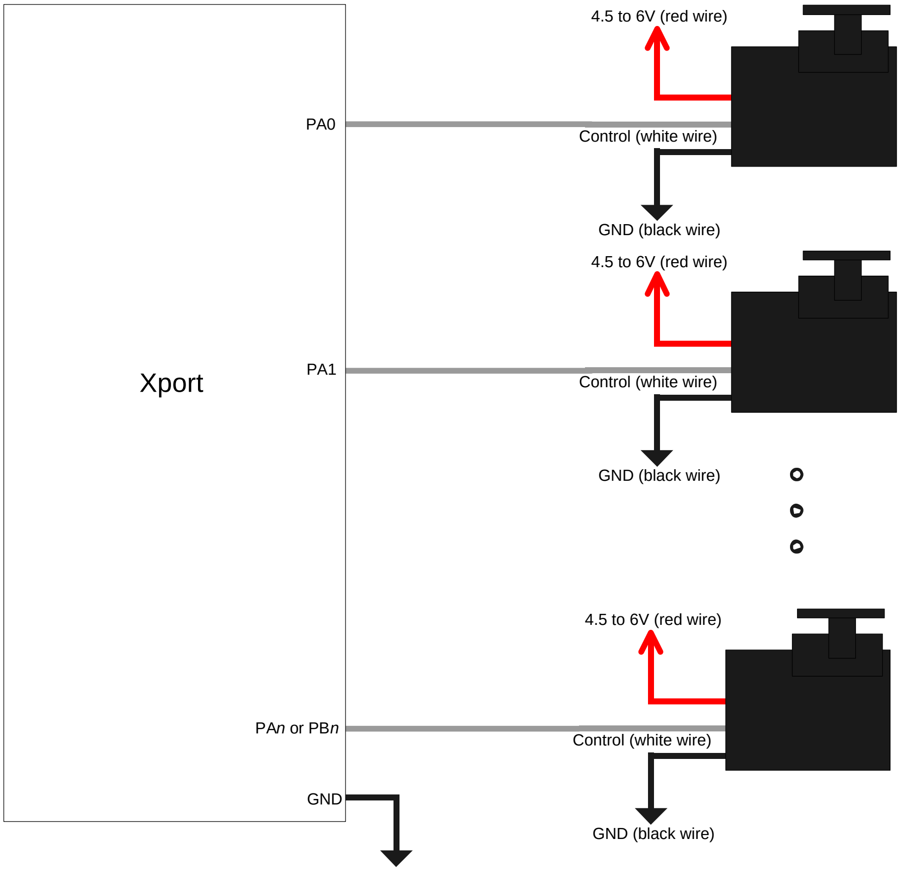

Figure 1: RC Servo Connection Diagram

Example Software

The example software implements the CRCServo class which takes care of setting the limits of travel for the individual servos and simplifies writing and reading the command values. **Figure 2** below describes the member functions of CRCServo.

Figure 2: CRCServo Members.

CRCServo(unsigned char num, unsigned short *addr, bool enable=true);

Constructor for CRCServo class.

num Number of channels. This should be 62 to control all channels in the RC servo configuration.

addr Location of beginning of RC servo register block. This should be 0x9ffc400.

enable Setting this to false will defer the enabling of the channels until later. Setting to true (default) will enable the PWM clock and thus enable the servos.

void Disable();

Disable PWM clock, thus disabling servos.

void Enable();

Enable PWM clock, thus enabling servos.

unsigned char GetPosition(unsigned char index);

Get the previously commanded position.

index PWM channel index counting from 0.

void SetPosition(unsigned char index, unsigned char pos);

Set servo position.

index PWM channel index counting from 0.

pos Desired position. This value can range from 0 to 255 with 0 corresponding to the most counterclockwise position and 255 corresponding the most clockwise position. SetPosition will take into account the “bounds” set in SetBounds, however, the range is always 0 to 255.

void SetBounds(unsigned char index, unsigned char lower, unsigned char upper);

Set the limits of servo travel.

index PWM channel index counting from 0. Note, each servo has its own bounds parameters.

lower Lower position bound. Can range from 0 to 255, but must not exceed upper bound.

upper Upper position bound. Can range from 0 to 255 but must be greater than lower bound.

Example Usage

```
#include "../../include/xport.h"
#include "../../include/textdisp.h"
#include "rcservo.h"
```

```
extern "C"
      {
      int Main(void);
      }

CTextDisp td;

#define RCSERVO_NUM     62
#define RCSERVO_ADDR    0x9ffc400

int Main(void)
      {
      // Check to make sure we are using the correct logic configuration
      if (XP_REG_IDENTIFIER!=0x8016)
            {
            td.Printf("Incorrect logic configuration.\n");
            while(1);
            }

      volatile unsigned long d;
      CRCServo servo((unsigned char)RCSERVO_NUM, (unsigned short *)RCSERVO_ADDR);

      // set bounds -- this varies from servo to servo
      servo.SetBounds(0, 64, 196);

      td.Printf("Servo demo\n");

      while(1)
            {
            // move maximum counter-clockwise
            servo.SetPosition(0, 0x00);
            td.Printf("Pos: 0x%x\n", servo.GetPosition(0));
            for (d=0; d<1000000; d++);

            // move middle
            servo.SetPosition(0, 0x80);
            td.Printf("Pos: 0x%x\n", servo.GetPosition(0));
            for (d=0; d<1000000; d++);

            // move maximum clockwise
            servo.SetPosition(0, 0xff);
            td.Printf("Pos: 0x%x\n", servo.GetPosition(0));
            for (d=0; d<1000000; d++);
            }
      }
```