Xport 2.0 RC Servo/GPIO Configuration

Version 2.0, June 12, 2003

Rich LeGrand (rich@charmedlabs.com)

Summary

This application note explains how to use the RC Servo/GPIO configuration to control up to 30 RC servos and 32 general purpose I/O signals with the Xport.

Introduction

"RC servos" were originally designed to control RC 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/GPIO configuration for the Xport synthesizes the PWM signals required for simultaneous control of up to 30 servos and 32 general purpose I/O (GPIO) signals.

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 with the actual or desired limits of servo travel.

Table 1: RC-Servo Register Block 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	PB0	PA14
RCS8	0x9ffc410	PB2	PB1
RCS9	0x9ffc412	PB4	PB3
RCS10	0x9ffc414	PB6	PB5
RCS11	0x9ffc416	PB8	PB7

RCS12	0x9ffc418	PB10	PB9
RCS13	0x9ffc41a	PB12	PB11
RCS14	0x9ffc41c	PB14	PB13

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 PA0 to the centermost position, set RCS0 as follows:

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

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

General Purpose I/O (GPIO)

Each GPIO signal has two control bits: a direction bit and a data bit. Setting the direction bit to 1 configures the corresponding I/O signal as an output. Setting the direction bit to 0 configures the I/O signal as an input. The data bit reflects the logic state of the corresponding I/O signal regardless of whether it is configured as an input or output. For convenience, these bits are grouped into 16-bit registers. The direction bits collectively form the "data direction registers" (DDRs) and the data bits form the "data registers" (DRs). **Table 2** below details these registers and their I/O signal mapping.

Table 2: GPIO Register Block Mapping

		Register contents (individual bits shown)															
Name	Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DDR0	0x9ffc600	PA30	PA29	PA28	PA27	PA26	PA25	PA24	PA23	PA22	PA21	PA20	PA19	PA18	PA17	PA16	PA15
DDR1	0x9ffc602	PB30	PB29	PB28	PB27	PB26	PB25	PB24	PB23	PB22	PB21	PB20	PB19	PB18	PB17	PB16	PB15
DR0	0x9ffc604	PA30	PA29	PA28	PA27	PA26	PA25	PA24	PA23	PA22	PA21	PA20	PA19	PA18	PA17	PA16	PA15
DR1	0x9ffc606	PB30	PB29	PB28	PB27	PB26	PB25	PB24	PB23	PB22	PB21	PB20	PB19	PB18	PB17	PB16	PB15

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 I/O signals PA15 through PA22 as output and PA23 through PA30 as input, set DDR0 as follows:

```
*((volatile unsigned short *)0x9ffc600) = 0x00ff;
```

Setting PA15 to PA18 as logic high and PA19 through PA22 as logic low, set DR0 as follows:

```
*((volatile unsigned short *)0x9ffc604) = 0x000f;
```

To read the state of PA23 through PA30, read DR0 as follows:

```
unsigned short val = *((volatile unsigned short *)0x9ffc604);   // read
val >>= 8;   // shift down PA23 through PA30 for convenience
```

Reading data bits that are configured as output should return the previously assigned value, as illustrated below:

```
*((volatile unsigned short *)0x9ffc600) = 0xffff;   // set PA15 through PA30 as output
*((volatile unsigned short *)0x9ffc604) = 0xabcd;
if (*((volatile unsigned short *)0x9ffc604)!=0xabcd)
   printf("Error: this should not happen\n");
```

Circuitry

Figure 2 below shows a recommended connection diagram for RC servos. It requires a separate 4.5 to 6V power supply for the servos. The power supply, Xport and servos should share the same ground.

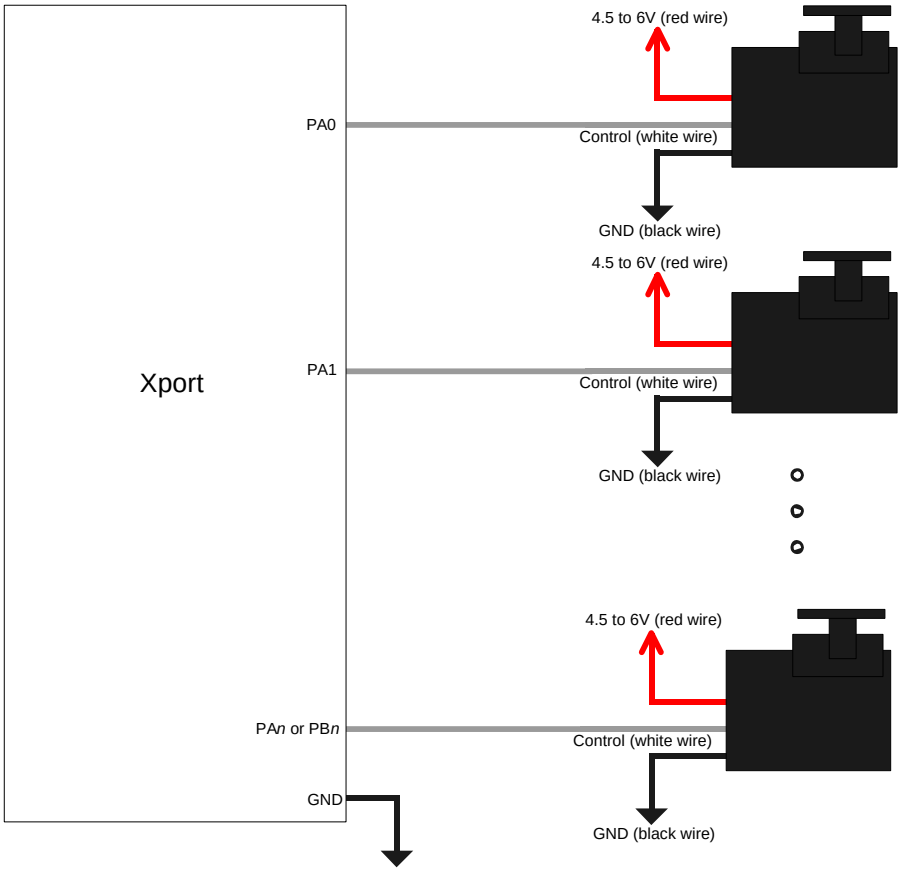

Figure 2: RC Servo Connection Diagram

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 1** below describes the member functions of CRCServo.

Figure 1: CRCServo Members.

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

Constructor for CRCServo class.

num	Number of channels. This should be 30 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 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 – each servo has its own bounds.

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 not be less than lower bound.

Example

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

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

CTextDisp td;

#define RCSERVO_NUM     30
#define RCSERVO_ADDR    0x9ffc400

#define GPIO_NUM        32
#define GPIO_ADDR       0x9ffc600

#define GPIO_REG_NUM    (GPIO_NUM+15)/16
```

```
#define GPIO_REG(i)           *((volatile unsigned short *)GPIO_ADDR+i)
#define GPIO_DDR(i)           GPIO_REG(i)
#define GPIO_DATA(i)          GPIO_REG(i+GPIO_REG_NUM)

int Main(void)
      {
      // Check to make sure we are using the correct logic configuration
      if (XP_REG_IDENTIFIER!=0x8015)
            {
            td.Printf("Incorrect logic configuration.\n");
            while(1);
            }

#if 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++);
            }
#else
      unsigned short write = 0, read;

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

      // set first 16 GPIO bits to output
      GPIO_DDR(0) = 0xffff;
      write = 0;
      while(1)
            {
            GPIO_DATA(0) = write;

            // read-back what we just wrote
            read = GPIO_DATA(0);

            if (read!=write)
                  {
                  td.Printf("ERROR 0x%x 0x%x\n", read, write);
```

```
                  while(1);
                  }
            if (write==0)
                  td.Printf(".");
            write++;
            }
#endif
      }
```