Bonus Exercises

Introduction

If there is time left after completing the 5 exercises, there are some other interesting devices available. This document contains some very brief information about these devices.

There is one assembly available for each of the following device types. They will need to be shared.

Motor – PWM Type 1

A “PWM Type 1” motor output requires 1 PWM output for speed control and 1 GPIO output for direction control.

The package **`Motor.PWM`** (**`libsimpleio/ada/objects/motor-pwm.ads`**) defines a concrete motor output subclass, **`Motor.PWM.OutputSubclass1`**. The associated constructor function **`Create`** requires one PWM output instance of type **`PWM.Output`** and one GPIO output pin instance of type **`GPIO.Pin`** and returns a motor output object instance of type **`Motor.Output`**.

The following is the boiler plate code for creating a motor output, given that the “PWM Type 1” motor assembly is plugged into **`J4`**. The rather large motor assembly includes a circuit board that provides electrical isolation between the LPC1114 I/O Processor and the motor power domains as well as some logic to encode the direction and speed signals for the particular motor driver.

```
with Ada.Text_IO; use Ada.Text_IO;

with GPIO.RemoteIO;
with Motor.PWM;
with PWM.RemoteIO;
with RemoteIO.Client.hidapi;
with RemoteIO.LPC1114;

use TYPE Motor.Velocity;

procedure test_motor_pwm1 is

  PWMFreq : CONSTANT Natural := 50;

  remdev : RemoteIO.Client.Device;
  dirout : GPIO.Pin;
  pwmout : PWM.Output;
  mot    : Motor.Output;

begin
  New_Line;
  Put_Line("Remote I/O Motor Test (PWM Type 1)");
  New_Line;

  remdev := RemoteIO.Client.hidapi.Create;
  dirout := GPIO.RemoteIO.Create(remdev, RemoteIO.LPC1114.GPIO2,
    GPIO.Output);
  pwmout := PWM.RemoteIO.Create(remdev, RemoteIO.LPC1114.PWM3, PWMFreq);
  mot    := Motor.PWM.Create(pwmout, dirout);
end test_motor_pwm1;
```

Try adding code to control the motor speed with the potentiometer assembly (plugged into **`J3`**).

Try changing the PWM pulse frequency. (The LPC1114 I/O processor allows a PWM frequency from 50 to 50000 Hz.)

Motor – PWM Type 2

A “PWM Type 2” motor requires two PWM outputs, one for nominal clockwise rotation and the other for nominal counterclockwise rotation.

The package **`Motor.PWM`** (**`libsimpleio/ada/objects/motor-pwm.ads`**) defines a concrete motor output subclass, **`Motor.PWM.OutputSubclass2`**. The associated constructor function **`Create`** requires two PWM output instances of type **`PWM.Output`** and returns a motor output object instance of type **`Motor.Output`**.

The following is the boiler plate code for creating a motor output, given that the “PWM Type 2” motor assembly is plugged into **`J4`**.

```
with Ada.Text_IO; use Ada.Text_IO;

with Motor.PWM;
with PWM.RemoteIO;
with RemoteIO.Client.hidapi;

use TYPE Motor.Velocity;

procedure test_motor_pwm2 is

  PWMFreq : CONSTANT Natural := 50;

  remdev : RemoteIO.Client.Device;
  cwout  : PWM.Output;
  ccwout : PWM.Output;
  mot    : Motor.Output;

begin
  New_Line;
  Put_Line("Remote I/O Motor Test (PWM Type 2)");
  New_Line;

  remdev := RemoteIO.Client.hidapi.Create;
  cwout  := PWM.RemoteIO.Create(remdev, RemoteIO.LPC1114.PWM2, PWMFreq);
  ccwout := PWM.RemoteIO.Create(remdev, RemoteIO.LPC1114.PWM3, PWMFreq);
  mot    := Motor.PWM.Create(cwout, ccwout);
end test_motor_pwm2;
```

Try adding code to control the motor speed with the potentiometer assembly (plugged into **`J3`**).

Try changing the PWM pulse frequency. (The LPC1114 I/O processor allows a PWM frequency from 50 to 50000 Hz.)

if you have done any of the other motor exercises already, you can reuse the code from it for this exercise.

LEGO Power Functions Remote Control Motor

The LPC1114 I/O Processor is capable of configuring any of the 8 GPIO pins as an IRED (Infrared Emitting Diode) driver for the LEGO Power Functions Remote Control Protocol. An IRED assembly and a motor test stand with the Power Functions Remote Control receiver will be passed around. Plug the IRED assembly into **`J4`**.

The package **`LEGORC`** (**`libsimpleio/interfaces/legorc.ads`**) defines an interface for all LEGO Power Functions Remote Control IRED driver outputs.

The package **`RemoteIO.LPC1114.LEGORC`** (**`libsimpleio/ada/remoteio/client/remoteio-lpc1114-legorc`**) defines a concrete class implementing **`LEGORC.OutputInterface`** called **`RemoteIO.LPC1114.LEGORC.ObjectClass`**. The constructor function **`Create`** requires a Remote I/O Abstract Device instance of type **`RemoteIO.Abstract_Device.Device`** and an LPC1114 I/O processor GPIO pin index and returns IRED driver output instance of type **`LEGORC.Output`**.

The package **`Motor.LEGORC`** (**`libsimpleio/ada/objects/motor-legorc.ads`**) defines a concrete motor output subclass called **`Motor.LEGORC.OutputSubclass`**. The associated constructor function **`Create`** requires three parameters (an IRED driver output instance of type **`LEGORC.output`**, a channel identifier of type **`LEGORC.Channel`**, a motor identifier of type **`Motor.LEGORC.MotorID`**) and returns a motor output object instance of type **`Motor.Output`**.

The following boiler plate code configures **`GPIO2`** as an IRED driver. The LPC1114 I/O processor generates the rather complicated command messages for the remote control protocol using bit-banging.

```
with Ada.Text_IO; use Ada.Text_IO;

with LEGORC;
with Motor.LEGORC;
with RemoteIO.Client.hidapi;
with RemoteIO.LPC1114.LEGORC;

use type Motor.LEGORC.Speed;

procedure test_motor_legorc is

  remdev : RemoteIO.Client.Device;
  absdev : RemoteIO.LPC1114.Abstract_Device.Device;
  ired   : LEGORC.Output;
  mot    : Motor.Output;

begin
  New_Line;
  Put_Line("Remote I/O LPC1114 LEGO Power Functions RC Motor Test");
  New_Line;
  remdev := RemoteIO.Client.hidapi.Create;
  absdev := RemoteIO.LPC1114.Abstract_Device.Create(remdev, 0);
  ired   := RemoteIO.LPC1114.LEGORC.Create(absdev,
    RemoteIO.LPC1114.LPC1114_GPIO2);
  mot    := Motor.LEGORC.Create(ired, 1, Motor.LEGORC.MotorA);
end test_motor_legorc;
```

Try adding code to control the motor speed with the potentiometer assembly (plugged into `J3`).

if you have done any of the other motor exercises already, you can reuse the code from it for this exercise.

Tone Generation (Advanced)

The package **`RemoteIO.LPC1114.Timers`** (**`libsimpleio/ada/remoteio/client/remoteio-lpc1114-timers.ads`**) provides services for using the two 32-bit counter/timers inside the LPC1114 microcontroller.

The counter/timers are capable of many different operating modes and the required services are complex. Here is some distilled code for generating a square wave output at one of the timer match outputs:

```
with Ada.Text_IO; use Ada.Text_IO;
with Interfaces;

with RemoteIO.Client.hidapi;
with RemoteIO.LPC1114.Timers;

use type Interfaces.Unsigned_32;

procedure test_squarewave is

  remdev  : RemoteIO.Client.Device;
  absdev  : RemoteIO.LPC1114.Abstract_Device.Device;
  CT32B1  : RemoteIO.LPC1114.Timers.Timer;
  freq    : Interfaces.Unsigned_32 := 1000;

begin
  New_Line;
  Put_Line("Remote I/O LPC1114 Timer Square Wave Output Test");
  New_Line;

  remdev := RemoteIO.Client.hidapi.Create;
  absdev := RemoteIO.LPC1114.Abstract_Device.Create(remdev, 0);
  CT32B1 := RemoteIO.LPC1114.Timers.Create(absdev,
    RemoteIO.LPC1114.LPC1114_CT32B1);

  -- Configure CT32B1 for square wave output

  CT32B1.Reset;

  CT32B1.Configure_Prescaler(1);

  CT32B1.Configure_Match_Action(RemoteIO.LPC1114.LPC1114_TIMER_MATCH1,
    RemoteIO.LPC1114.LPC1114_TIMER_MATCH_OUTPUT_TOGGLE, True, False);

  CT32B1.Configure_Match_Value(RemoteIO.LPC1114.LPC1114_TIMER_MATCH1,
    RemoteIO.LPC1114.LPC1114_PCLK/freq/2);

  CT32B1.Configure_Mode(RemoteIO.LPC1114.LPC1114_TIMER_MODE_PCLK);
end test_squarewave;
```

If you plug the speaker assembly into J4, you will hear a 1000 Hz tone. (It won't be very loud, to hopefully avoid annoying your neighbors).

Try adding code to play a sequence of musical notes. (The A above middle C (A_4) is 440 Hz by modern convention. With the "Equal Temper" scale, the frequencies of every pair of notes are separated by a factor of the 12th root of 2, or about 1.05946. (A_4 is 440 Hz, B_4 is 466.16 Hz, etc.)