LSE-2015 Project

`Servo Control`

Cholbi Alenda, Pablo (*Dev, Doc, Test*)
p.cholbi@alumnos.upm.es

5^{th} of May, 2015

Date	Version	Issue	Author
2015-04-19	1.0	Initial release.	*PCA*
2015-04-20	1.1	Errata in section **Test**. Added GPIO and pin number.	*PCA*
2015-04-28	1.2	Errata in GPIO numbering.	*PCA*
2015-04-29	1.3	GPIOs changed.	*PCA*
2015-05-05	1.4	Updated section **Application**.	*PCA*

Contents

1 Description

The **servoctrl** application provides the functionality of controlling up to two servo motors from a Raspberry Pi embedded system.

2 Application

The application receives input on network socket `2301`. The input commands have a fixed size of one byte in the following format:

2.0.1 `Input`

Bit	Name	Description
6-0	`pos`	Offset source
7	`sel`	Servo selector

Table 1: `Input`

The *sel* field determines which servo motor is being addressed and the *pos* field contains the position to be set; where bX0000000 is the minimum angular position of the servo and bX1111111 is the maximum angular position of the servo.

This application makes use of the following GPIOs:

2.0.2 `Output`

BCM GPIO	Header Pin	Function	Description
18	**12**	`servo0`	Servo 0 control signal
13	**33**	`servo1`	Servo 1 control signal

Table 2: `Output`

The application catches `SIGTERM` and `SIGINT` signals to terminate gracefully after a clean-up. Please note that the provided service script (Section 5) handles this appropriately.

In the case that an error occurs and **servoctrl** exits ungracefully, it is possible that application will (temporarily) not start again because the OS may not be aware that socket `2301` is no longer in use by the previous instance of **servoctrl**. In this situation, it is possible to run `sudo fuser -k 2301/tcp` to kill all processes listening on socket `2301`.

The application does not write to `STDOUT` or `STDERR`. It instead writes to **Syslog** (`/var/log/syslog`).

3 Compatibility

This code is compatible with Raspberry Pi model A+ and B+ onward. At the time of writing, this includes models A+, B+ and 2B.

The reason that this code is not compatible with previous models is because models A and B only had 1 hardware PWM pin accessible on the P1 pin header.

4 Setup

A set up script is provided at `../../scripts/setup/servoctrl_setup.sh` to ease the building and installation.

If the setup script finished successfully; the binary should be at `/usr/local/bin/servoctrl` and a UNIX System V init script should be at `/etc/init.d/servoctrl`.

Aside form the setup script; the source code and makefiles are also provided.

5 Service

To start **servoctrl** as a daemon; execute `/etc/init.d/servoctrl` as root. This daemon can take as argument **start**, **stop**, **restart** or **force-reload**.

6 Testing

A test script is provided at `../../scripts/test/servoctrl_test.sh` to test the application. The test script execute a series of test cases and then asks for user input to determine if the test executed correctly.

If the test was successful, the scripts returns 0, if the test failed a value different from 0 is returned.

The test sequence currently implemented is the following:

1. servo0 and servo1 are both set to their minimum position.
2. servo1 progresses from its minimum position to its maximum position.
3. servo0 progresses from its minimum position to its maximum position.
4. servo1 progresses from its maximum position to its minimum position.
5. servo0 progresses from its maximum position to its minimum position.
6. servo0 and servo1 both progresses together from their minimum position to their maximum position.
7. servo0 progresses from its maximum position to its minimum position.
8. servo0 progrsses from its minimum position to its maximum position while servo1 progresses from its maximum position to its minimum position.

As an example, in the specific case where the servos control the pan/tilt of an instrument the test should make the instrument follow a square route and then inscribe a cross within that square route.