Raspberry Pi LPC1114 I/O Processor Expansion Board User Guide

Revision 40
12 November 2019

by Philip Munts
Munts Technologies

http://tech.munts.com

Contents

Introduction

The Raspberry Pi LPC1114 I/O Processor Expansion Board is an expansion board for the Raspberry Pi, containing an LPC1114 ARM Cortex-M0 microcontroller. The LPC1114 microcontroller connects to the SPI, I2C, and UART ports on the Raspberry Pi expansion header and to 8 GPIO signals on the terminal block.

The LPC1114 microcontroller is capable of very fast interrupt and GPIO signal manipulation, making it suitable for a variety of applications, such as infrared remote control protocols. Such protocols typically have stringent timing requirements and would otherwise be impossible to implement with the Raspberry Pi alone.

The LPC1114 microcontroller also offers a great deal of flexibility. The GPIO signals brought out from the LPC1114 to the terminal block may be configured under software control for a variety of functions, such as A/D (Analog to Digital) inputs, PWM (Pulse Width Modulation) outputs, and GPIO (General Purpose Input Output) signals.

Around the LPC1114 I/O Processor Expansion Board

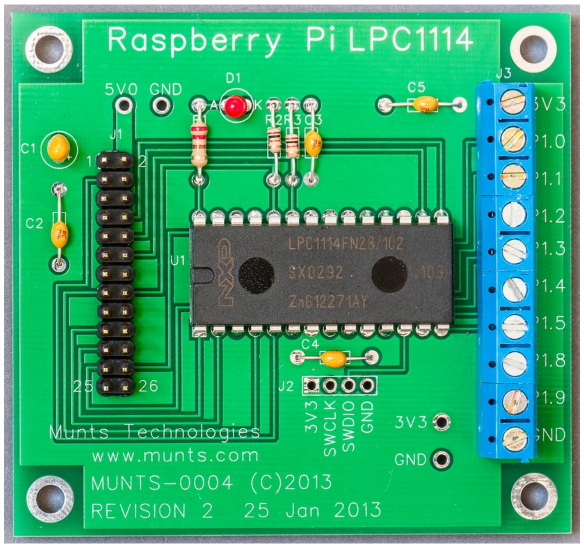


Figure 1 -- Raspberry Pi LPC1114 I/O Processor Expansion Board

From left to right, **J1** is a 26-pin header. Connect the supplied 26-conductor ribbon cable from **J1** to the Raspberry Pi **P1** expansion header. The LPC1114 I/O Processor Expansion Board takes +3.3V power from the Raspberry Pi via the cable to **J1**.

Above **J1** are test point holes for **+5V power** and **ground**. These are connected to the **+5V** power bus on the Raspberry Pi. You can connect a +5V power supply here to power *both* the Raspberry Pi and the LPC1114 I/O Processor Expansion Board if desired, eliminating the need to provide power to the Raspberry Pi via the micro-USB connector.

Note: The 5V0 test hole is connected directly to the Raspberry Pi +5V power bus, bypassing the fuse on the Raspberry Pi micro-USB connector. This means there is ***no*** *overcurrent or short circuit protection except that which may be implemented within the external power supply! This may be an advantage for embedded systems, as you can probably use a USB WiFi network interface without a powered USB hub.*

Continuing, **D1** is an LED (Light Emitting Diode) controlled by the LPC1114 P0.7 GPIO pin. The LED can be turned on and off from LPC1114 firmware for debugging or signaling purposes. One example of LED usage would be to blink the LED whenever the Raspberry Pi sends a command message to the LPC1114.

J2 is a 4 pin header (not populated) for SWD (Serial Wire Debug). You can solder header pins here and connect a SWD debug pod such as the SEGGER J-Link to **J2** to perform source level debugging with ***gdb***.

Adjacent to J2 are test point holes for **+3.3V power** and **ground**. The +3.3V power comes from the Raspberry Pi's voltage regulator. Do not attempt to take more than a few milliamperes from it or you will overload the Raspberry Pi.

J3, the 10-position terminal block, is on the right edge of the board. +3.3V power, ground and 8 LPC1114 GPIO pins are brought out to **J3**. The +3.3V power supply comes from the Raspberry Pi's voltage regulator. Do not attempt to take more than a few milliamperes from it or you will overload the Raspberry Pi.

Each of the LPC1114 GPIO pins brought out to **J3** may be configured for a variety of purposes. See Table 1. See the LPC1114 datasheet and user manual documents from NXP for more information about GPIO pin functions.

LPC1114_GPIO0	aka P1.0	General Purpose Input/Output 0, Analog Input 1, 32-bit Timer 1 Capture Input 0
LPC1114_GPIO1	P1.1	General Purpose Input/Output 1, Analog Input 2, Pulse Width Modulated Output 1, 32-bit Timer 1 Match Output 0
LPC1114_GPIO2	P1.2	General Purpose Input/Output 2, Analog Input 3, Pulse Width Modulated Output 2, 32-Bit Timer 1 Match Output 1
LPC1114_GPIO3	P1.3	General Purpose Input/Output 3, Analog Input 4, Pulse Width Modulated Output 3, 32-Bit Timer 1 Match Output 2
LPC1114_GPIO4	P1.4	General Purpose Input/Output 4, Analog Input 5, 32-Bit Timer 1 Match Output 3
LPC1114_GPIO5	P1.5	General Purpose Input/Output 5, 32-Bit Timer 0 Capture Input 0
LPC1114_GPIO6	P1.8	General Purpose Input/Output 6, 16-Bit Timer 1 Capture Input 0
LPC1114_GPIO7	P1.9	General Purpose Input/Output 7, Pulse Width Modulated Output 4, 16-Bit Timer 1 Match Output 0

Table 1 – LPC1114 GPIO Pin Functions

*Note: The LPC1114 GPIO pins are 5V tolerant **except** when configured as analog inputs. The input voltage to pins configured for analog input must **never** exceed +3.3V.*

Connecting the I/O Processor Expansion Board

It is recommended to mount your Raspberry Pi in some sturdy fashion before attaching the LPC1114 I/O Processor Expansion Board to it. Some suggestions are:

- Mount both the Raspberry Pi and the I/O Processor Expansion Board to a piece of wood, using screws and standoffs.
- Mount the Raspberry Pi inside a case such as the Pibow which is in turn mounted to a piece of wood. Mount the I/O Processor Expansion Board with screws and standoffs.
- Mount both the Raspberry Pi and the I/O Processor Expansion Board inside a project case such as the Radio Shack 270-1807 or 270-1809.

Halt and power off your Raspberry Pi before connecting the LPC1114 I/O Processor Expansion Board to it.

Connect the supplied 26-pin ribbon cable from the **J1** header on the LPC1114 I/O Processor Expansion Board to the **P1** expansion header on the Raspberry Pi. The colored stripe on the ribbon cable should oriented to pin 1 on both the Raspberry Pi and the I/O Processor Expansion Board. See figure 2 below.

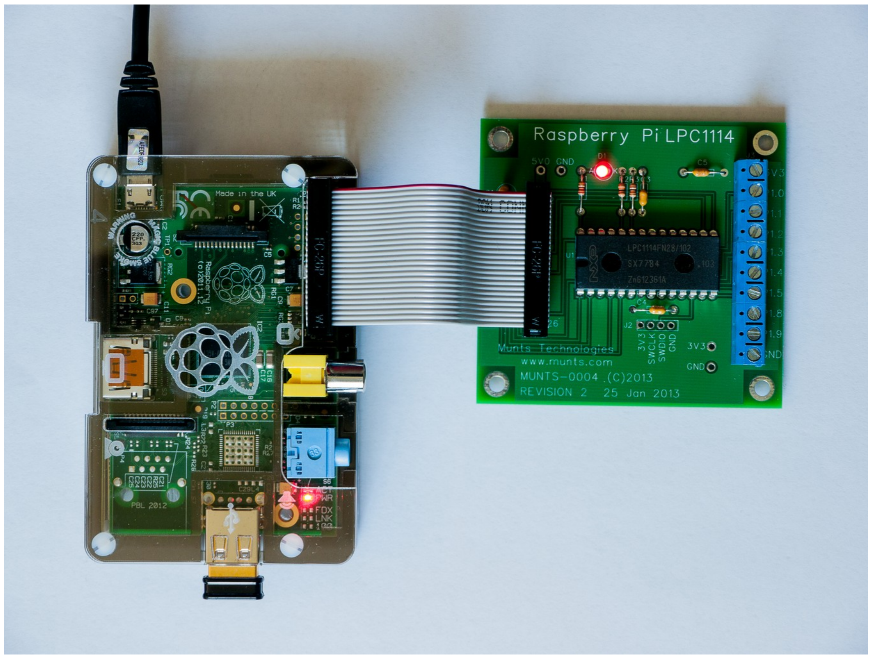


Figure 2 – Connecting to the Raspberry Pi

Software Support

To develop software for the LPC1114 I/O Processor expansion board, you will need to install several items on your Raspberry Pi:

- Certain Linux packages, especially ***munts-expansion-lpc1114.***
- Source code, including headers and sample programs.
- An optional *cross-compiler toolchain* package (C compiler, assembler, linker, librarian, debugger), if you wish to build firmware for the LPC1114 microcontroller.
- An optional *microcontroller firmware framework* source code (Run time startup, linker scripts, I/O libraries, etc.), if you wish to build firmware for the LPC1114 microcontroller.

All of these items will be installed by the following procedure. It assumes you are running the latest Raspbian distribution downloaded from www.raspberrypi.org.

Your Raspberry Pi will need to be connected to the Internet in order to download and install the software components for LPC1114 I/O Processor Expansion Board development.

System Preparation

Before you can install the software support for the LPC1114 I/O Processor Expansion Board, you must enable the SPI and serial port interfaces from the ***raspi-config*** program.

For the Raspberry Pi 3, you must also disable the internal Bluetooth interface and route the serial port ***/dev/ttyAMA0*** to the expansion header with the following command, typed in a Terminal Window:

```
sudo sh -c "echo dtoverlay=pi3-disable-bt >>/boot/config.txt"
```

You should also install the latest Raspbian updates with the following commands:

```
sudo apt-get update
sudo apt-get upgrade
```

After all of the above preparation, reboot your Raspberry Pi to apply any pending changes. Now you are ready to install software support for the LPC1114 I/O Processor Expansion board.

Software Installation

Download the setup script:

http://git.munts.com/rpi-mcu/expansion/LPC1114/src/scripts/expansion_lpc1114_setup

using the web browser and save it to your home directory. Then run it with the following command, typed in a Terminal Window:

```
sh < expansion_lpc1114_setup
```

The setup script adds http://repo.munts.com to the list of Debian package repositories for ***apt*** and installs some packages for the LPC1114 I/O Processor Expansion Board. It installs what is needed to build and run C or Python applications that run on the Raspberry Pi and communicate with the preprogrammed SPI Agent Firmware. To compile Ada, C#, Java or Pascal applications you will also need to install the necessary toolchain for each language.

If you wish to write custom firmware for the LPC1114 microcontroller, you will need to run two more commands to install the cross-compiler tool chain and ARM microcontroller framework (see http://git.munts.com/arm-mcu):

```
sudo apt-get install arm-mcu-tools
git clone http://git.munts.com/arm-mcu.git
```

Configuration File

The configuration file ***/usr/local/etc/expansion_lpc1114.config*** contains Raspberry Pi device and GPIO pin assignments for the LPC1114 I/O Processor Expansion Board. Under certain circumstances you may need to edit this file (to select I^{2}C instead of SPI, for example).

Software Updates

From time to time, additions, corrections, and bug fixes will be made to ***arm-mcu*** and ***rpi-mcu***. To import the latest LPC1114 I/O processor support software changes to your Raspberry Pi, run the following command sequence:

```
cd $HOME/arm-mcu ; git pull
cd $HOME/rpi-mcu ; git pull
sudo apt-get upgrade
```

Preprogrammed SPI Agent Firmware

The LPC1114 single chip microcomputer comes preprogrammed with a program known as the *SPI Agent*. The SPI Agent program configures the LPC1114 SPI (Serial Peripheral Interconnect) as an SPI slave and listens for commands from the Raspberry Pi.

All of the example programs in the Raspberry Pi Microcontroller Framework source directory ***rpi-mcu/expansion/LPC1114/src*** rely on the SPI Agent firmware.

LED Test

You can run the following commands at the Raspberry Pi command line to try the SPI Agent:

```
cd $HOME/rpi-mcu/expansion/LPC1114/src/c/tests
make test_led
./test_led
```

These commands compile and run the C program ***test_led***, which merely flashes the LPC1114 I/O Processor Expansion Board LED. Once a second ***test_led***, running on the Raspberry Pi, sends an SPI command message to the SPI Agent running on the LPC1114 I/O Processor Expansion Board, telling it to toggle the LED.

SPI Agent Test

Run ***spi_agent_test*** to execute the SPI Agent verification test, which exercises the SPI interconnect between the Raspberry PI and the LPC1114 I/O Processor Expansion Board:

```
spi_agent_test localhost 1000000
```

This executes over a million SPI transactions to ensure error free performance. The output from ***spi_agent_test*** will resemble the following (when run on a Raspberry Pi 3 Model B):

```
Raspberry Pi LPC1114 I/O Processor Expansion Board SPI Agent Firmware
Test

Issuing some SPI transactions...

Response: command:0    pin:0    data:12193 error:0
Response: command:1    pin:2    data:3     error:0
Response: command:7    pin:99   data:0     error:19
Response: command:99   pin:2    data:0     error:22

The LPC1114 firmware version is 12193
The LPC1114 device ID is        1A40902B
The expansion board LED is      OFF

Starting 1000000 SPI agent loopback test transactions...

Performed 1000000 loopback tests in 138 seconds
  7246.4 iterations per second
  138.0 microseconds per iteration
```

This indicates that each command to the SPI Agent firmware completes in 138 microseconds, for an average rate of about 7246 commands per second.

SPI Agent RPC Servers

A further abstraction of the LPC1114 SPI Agent Firmware can be accomplished by "wrapping" it in an RPC (Remote Procedure Call) server. There are two RPC servers programs available for the LPC1114 SPI Agent, one for the ONC/RPC (formerly known as Sun RPC) protocol and one for the XML-RPC protocol. Either of these servers can run on the Raspberry Pi to provide RPC services to client programs running on the Raspberry Pi itself *or* other computers connecting via the network.

ONC/RPC Server

The ONC/RPC server program ***`spi_agent_oncrpc_server`*** provides highly efficient RPC services to client programs running on either the Raspberry Pi itself or on remote computers connecting via the network.

Many commonly used *compiled* programming langages, such as C/C++, Java, C#, etc. can issue ONC/RPC calls, with the proper development libraries. Most *interpreted* programming languages, such as Python, Ruby, Lua, etc. *cannot* make ONC/RPC calls without a great deal of effort.

The advantage of ONC/RPC is that it is very efficient; ***`spi_agent_oncrpc_server`*** can handle over 2200 commands per second from a client program running on the same Raspberry Pi.

XML-RPC Server

The XML-RPC server program ***`spi_agent_xmlrpc_server`*** provides highly portable RPC services to client programs running on either the Raspberry Pi itself or on remote computers connecting via the network.

The advantage of XML-RPC is that it is very widely implemented; most current programming languages (including Python) have support for it available.

The disadvantage of XML-RPC is that it is *far* less efficient than ONC/RPC; ***`spi_agent_xmlrpc_server`*** can only handle about 220 commands per second from a compiled C client program running on the same Raspberry Pi. For interpreted language clients, the protocol inefficiency of XML-RPC matters less; a Python 3 client running on the Raspberry Pi can only issue about 69 commands per second, about a third of what the server can handle.

The Python language comes with XML-RPC support built-in, using the ***xmlrpc*** module. The simplest Python client for SPI Agent XML-RPC, that executes a single loopback test command, would be like this:

```
#!/usr/bin/python3

import xmlrpc.client

s = xmlrpc.client.ServerProxy('http://localhost:8080/RPC2')
print(s.system.listMethods())
print(s.spi.agent.transaction(1, 2, 3))
```

A somewhat more elaborate Python client for SPI Agent XML-RPC, which includes the ability to select the server network address, is available in the directory ***$HOME/rpi-mcu/expansion/LPC1114/src/spi-agent-xmlrpc/clients/python***.

HTTP Server

There is also an HTTP server, that although not strictly an RPC server, is used exactly in the same manner. Some programming languages may not have support for either ONC/RPC or XML-RPC but do have support for HTTP. (Microsoft Small Basic is one example. Free Pascal for Windows is another.)

The HTTP server syntax is simple: Append a query string composed of an optional ***cmd=***, and 3 integers separated by commas (command, pin, data) to the URL. The following example illustrates how to turn on the LED on a Raspberry Pi with the domain name ***zoar.munts.net***:

```
http://zoar.munts.net:8081/SPIAGENT?8,7,1
```

The HTTP server returns a minimal web page containing a response string, which is composed of 5 integers separated by commas (***ioctl()*** error, command, pin, data, error) and followed by a semicolon:

```
0,8,7,0,0;
```

Your client program should be able to handle gracefully any arbitrary response message, such as ***ERROR 404*** or other messages inserted by proxy servers.

The `libspiagent.so` Shared Library

As an even further abstraction, the dynamic shared library ***libspiagent.so*** (***libspiagent.dll*** on Windows) encapsulates several different transport mechanisms between an application program and the SPI Agent Firmware. These mechanisms include the ONC-RPC and XML-RPC protocols described above, along with HTTP and the Linux ***ioctl()*** interface.

The Ada, C/C++, Free Pascal, Java, and Python3 programming languages on the Raspberry Pi can use ***libspiagent.so***. Each of these languages has a thin binding for ***libspiagent.so*** available. For remote client computers, ***libspiagent.so*** may or may not be usable, depending on the programming language and the operating system. See ***README.txt*** in the ***spi-agent-lib*** directory for more detailed information.

SPI Agent Firmware API Reference

The authoritative source for information about the SPI Agent Firmware API (Application Programming Interface) is the C header file ***include/spi-agent.h*** and its subordinates.

Each SPI bus transaction between the Raspberry Pi and the LPC1114 consists of two message packets, one to the LPC1114 (the Command Message) and the other from the LPC1114 (the Response Message). Pointers to these message structures are handed to the Linux SPI ***ioctl()*** operation. The underlying C library functions, system calls, and kernel services pass the command message to the LPC1114 and retrieve the response message from it, all in a single ***ioctl()*** system call.

Command Message

The ***Command Message*** is a sequence of three 32-bit unsigned integers. It is defined as a structure of type ***SPIAGENT_COMMAND_MSG_t*** in ***spi-agent.h*** for the C language. Other language bindings may implement the command message in their respective record or structure equivalents, or as a simple array of 32-bit integers.

command	***command:***	command code, defined below
pin	***pin***:	LPC1114 GPIO pin identifier, defined below
data	***data***:	Depends upon the particular command

Response Message

The ***Response Message*** is a sequence of four 32-bit unsigned integers. It is defined as a structure of type ***SPIAGENT_RESPONSE_MSG_t*** in ***spi-agent.h*** for the C language. Other language bindings may implement the command message in their respective record or structure equivalents, or as a simple array or list of 32-bit integers.

command	***command***:	Command code, echoed
pin	***pin***:	LPC1114 GPIO pin identifier, echoed
data	***data***:	Depends upon the particular command
error	***error***:	Error number (from ***errno.h***)

Commands

The following commands are defined. New commands will be added from time to time, but backward compatibility will be maintained.

All response messages will echo the ***command*** and ***pin*** fields from the command message. Unless specified otherwise, the response message ***data*** field will always be set to zero. The response message ***error*** field will also be set to zero upon success, or to a value from ***errno.h*** (typically ***ENODEV*** or ***EINVAL***) upon failure.

0 SPIAGENT_CMD_NOP

The response message ***data*** field will contain the SPI Agent Firmware version number (if available).

1 SPIAGENT_CMD_LOOPBACK

The command message ***command***, ***pin***, and ***data*** fields are all echoed in the response message.

2 SPIAGENT_CMD_CONFIGURE_ANALOG_INPUT

The LPC1114 GPIO pin indicated by the command message ***pin*** field is configured as an analog input. The command message ***data*** field is ignored.

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid analog input pin. Valid pins for this service are ***LPC1114_ADC1-5***.

3 SPIAGENT_CMD_CONFIGURE_GPIO_INPUT

The LPC1114 GPIO pin indicated by the command message ***pin*** field will be configured as a digital input. If ***data*** is zero, it will have an internal pull-down resistor enabled. If ***data*** is one, it will have an internal pull-up resistor enabled. (If you need a high impedance input, with neither pull-up nor pull down, use the ***SPIAGENT_CMD_CONFIGURE_GPIO*** command described below.)

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid GPIO input pin, or ***EINVAL*** if the data field is invalid. Valid pins for this service are ***LPC1114_GPIO0-7***.

4 SPIAGENT_CMD_CONFIGURE_GPIO_OUTPUT

The LPC1114 GPIO pin indicated by the command message ***pin*** field will be configured as a digital push-pull output. The ***data*** field indicates the initial state for the GPIO output pin. (If you need an open-drain output instead of push-pull, use the ***SPIAGENT_CMD_CONFIGURE_GPIO*** command described below.)

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid GPIO output pin, or ***EINVAL*** if the data field is invalid. Valid pins for this service are ***LPC1114_GPIO0***-7, ***LPC1114_INT***, and ***LPC1114_LED***.

5 SPIAGENT_CMD_CONFIGURE_PWM_OUTPUT

The LPC1114 GPIO pin indicated by the command message ***pin*** field will be configured as a Pulse Width Modulated push-pull output. The ***data*** field sets the pulse frequency and is limited to the range 50 through 50,000 Hz inclusive.

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not configurable as a PWM output, or ***EINVAL*** if the pulse frequency is out of range. Valid pins for this service are ***LPC1114_PWM1-4***.

Note: ***LPC1114_PWM1-3*** *share the same clock generator, so setting the pulse frequency on any one of the channels always affects all three channels.* ***LPC1114_PWM4*** *can be configured for a different pulse frequency.*

6 SPIAGENT_CMD_GET_ANALOG

The response message ***data*** field will be set to the unsigned 10-bit analog value, 3.22 millivolts per step, corresponding to the analog input voltage at the LPC1114 analog input pin indicated by the command message ***pin*** field. The command message ***data*** field is ignored.

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid analog input pin. Valid pins for this service are ***LPC1114_ADC1-5***.

7 SPIAGENT_CMD_GET_GPIO

The response message ***data*** field will be set to the state of the LPC1114 GPIO pin indicated by the command message ***pin*** field. The command message ***data*** field is ignored.

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid GPIO pin. Valid pins for this service are ***LPC1114_GPIO0***-7, ***LPC1114_INT***, and ***LPC1114_LED***.

8 SPIAGENT_CMD_PUT_GPIO

The LPC1114 GPIO output pin indicated by the command message ***pin*** field is set to the state indicated by the command message ***data*** field.

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid GPIO pin, or to ***EINVAL*** if the data value is out of range (not 0 or 1). Valid pins for this service are ***LPC1114_GPIO0-7***, ***LPC1114_INT***, and ***LPC1114_LED***.

9 SPIAGENT_CMD_PUT_PWM

The LPC1114 PWM output pin indicated by the command message ***pin*** field will be set to the 16-bit duty cycle indicated by the command message ***data*** field. A level of 0 indicates minimum duty cycle at the PWM output and a value of 65535 indicates maximum duty cycle at the PWM output.

Note: The actual resolution of the duty cycle varies, depending on the pulse frequency, from a full 16-bits at low frequencies to as little as 6-bits at higher frequencies.

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a PWM output, or to ***EINVAL*** if the data value is out of range. Valid pins for this service are ***LPC1114_PWM1-4***.

10 SPIAGENT_CMD_CONFIGURE_GPIO_INTERRUPT

The LPC1114 GPIO input pin indicated by the command message ***pin*** field is configured as an interrupt input, causing an interrupt pulse to the Raspberry Pi out ***LPC1114_INT***, according to configuration in the command message ***data*** field. The following interrupt configurations are available:

0 LPC1114_GPIO_INTERRUPT_DISABLED	Interrupts are disabled (default)
1 LPC1114_GPIO_INTERRUPT_FALLING	Falling edge causes interrupt
2 LPC1114_GPIO_INTERRUPT_RISING	Rising edge cause interrupt
3 LPC1114_GPIO_INTERRUPT_BOTH	Both edges cause interrupts

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid GPIO input pin, or to ***EINVAL*** if the command message data field is invalid. Valid pins for this service are ***LPC1114_GPIO0-7***.

11 SPIAGENT_CMD_CONFIGURE_GPIO

The LPC1114 GPIO pin indicated by the command message ***pin*** field will be configured according to the configuration indicated by the command message ***data*** field. The following GPIO configurations are available:

0 LPC1114_GPIO_MODE_INPUT	High impedance input
1 LPC1114_GPIO_MODE_INPUT_PULLDOWN	Input with weak pull-down resistor
2 LPC1114_GPIO_MODE_INPUT_PULLUP	Input with weak pull-up resistor
3 LPC1114_GPIO_MODE_OUTPUT	Push-pull (sink or source) output
4 LPC1114_GPIO_MODE_OUTPUT_OPENDRAIN	Open drain (sink only) output

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid GPIO pin, or to ***EINVAL*** if the command message data field is invalid. Valid pins for this service are ***LPC1114_GPIO0-7, LPC1114_INT***, and ***LPC1114_LED.*** The interrupt and LED pins can only be configured as push-pull outputs.

12 SPIAGENT_CMD_PUT_LEGORC

A LEGO® Power Functions Remote Control command will be emitted from the LPC1114 GPIO output pin indicated by the command message ***pin*** field. The message parameters are encoded in the command message ***data*** field as follows:

Bits 0-7	Speed:	Motor A or Motor B: ***0-7***. Combo Direct: ***0-15***. Combo PWM: ***0-255***.
Bits 8-15	Direction:	Motor A or Motor B: ***0***=Reverse ***1***=Forward. All other motor values: ***0***=Not used
Bits 16-23	Motor:	***0***=All stop, ***1***=Motor A, ***2***=Motor B, ***3***=Combo Direct, ***4***=Combo PWM
Bits 24-31	Channel:	***1-4***

The response message ***error*** field will be set to ***ENODEV*** if the selected pin is not a valid GPIO output pin, or to ***EINVAL*** if the command message data field is invalid. Valid pins for this service are ***LPC1114_GPIO0-7***.

Note: The LEGO® Power Functions RC Protocol specifies that each command be transmitted 5 times, with timing between transmissions dependent upon the channel number. This service only sends the command ***one*** *time.*

Note: This service requires 20 milliseconds to execute, during which both the **ioctl()** *system call and calling program are blocked.*

Note: See the LEGO® Power Functions RC specification for more information, especially about the Combo Direct and Combo PWM commands.

13 SPIAGENT_CMD_GET_SFR

The contents of the LPC1114 peripheral register indicated by the command message ***pin*** field will be returned in the response message ***data*** field.

The response message ***error*** field will set to ENODEV if the register address passed in the command message is outside the ranges ***0x40000000*** through ***0x4007FFFF*** and ***0x50000000*** through ***0x501FFFFF***.

14 SPIAGENT_CMD_PUT_SFR

The contents of the command message ***data*** field will be written to the LPC1114 peripheral register indicated by the command message ***pin*** field.

The response message ***error*** field will set to ENODEV if the register address passed in the command message is outside the ranges ***0x40000000*** through ***0x4007FFFF*** and ***0x50000000*** through ***0x501FFFFF***.

15 SPIAGENT_CMD_CONFIGURE_TIMER_MODE

The LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command message ***pin*** field will be configured to the mode and counting source indicated by the command message ***data*** field.

The following counter/timer identifiers are available:

0 LPC1114_CT32B0
1 LPC1114_CT32B1

The following counter/timer modes are available:

0 LPC1114_TIMER_MODE_DISABLED
1 LPC1114_TIMER_MODE_RESET
2 LPC1114_TIMER_MODE_PCLK
3 LPC1114_TIMER_MODE_CAP0_RISING
4 LPC1114_TIMER_MODE_CAP0_FALLING
5 LPC1114_TIMER_MODE_CAP0_BOTH

The response message ***error*** field will be set to ***ENODEV*** if the timer identifier is invalid, or to ***EINVAL*** if the mode is invalid.

*Note: **PCLK** is the LPC1114's 48 MHz system clock. **CAP0** is the capture input (**GPIO4** for **CT32B0** and (**GPIO0** for **CT32B1**).*

16 SPIAGENT_CMD_CONFIGURE_TIMER_PRESCALER

The command message ***data*** field, minus 1, will be written to the prescaler divisor register for the LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command message ***pin*** field.

The response message ***error*** field will be set to ***ENODEV*** if the timer identifier is invalid, or to ***EINVAL*** if the divisor is zero.

17 SPIAGENT_CMD_CONFIGURE_TIMER_CAPTURE

The capture subsystem for the LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command message ***pin*** field will be configured according to the settings indicated by the command message ***data*** field. The settings are encoded as follows:

Bits 0-3	Capture edge
Bit 4	Interrupt the Raspberry Pi on each capture event
Bits 5-31	Not used

The following capture edge settings are available:

0 LPC1114_TIMER_CAPTURE_EDGE_DISABLED
1 LPC1114_TIMER_CAPTURE_EDGE_CAP0_RISING
2 LPC1114_TIMER_CAPTURE_EDGE_CAP0_FALLING
3 LPC1114_TIMER_CAPTURE_EDGE_CAP0_BOTH

The response message ***error*** field will be set to ***ENODEV*** if the timer identifier is invalid, or to ***EINVAL*** if the mode is invalid.

18 SPIAGENT_CMD_CONFIGURE_TIMER_MATCH0
19 SPIAGENT_CMD_CONFIGURE_TIMER_MATCH1
20 SPIAGENT_CMD_CONFIGURE_TIMER_MATCH2
21 SPIAGENT_CMD_CONFIGURE_TIMER_MATCH3

The corresponding match control register (***MCR0-3***) and external match register (***EMR0-3***) for the LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command message ***pin*** field will be configured according to the settings in the command message ***data*** field.

The settings are encoded as follows:

Bits 0-3	Match output action
Bit 4	Interrupt the Raspberry Pi on match
Bit 5	Reset the counter on match
Bit 6	Stop the counter on match
Bits 7-31	Not used

The following match output actions are available:

0 LPC1114_TIMER_MATCH_OUTPUT_DISABLED
1 LPC1114_TIMER_MATCH_OUTPUT_CLEAR
2 LPC1114_TIMER_MATCH_OUTPUT_SET
3 LPC1114_TIMER_MATCH_OUTPUT_TOGGLE

The response message ***error*** field will be set to ***ENODEV*** if the timer identifier is invalid, or to ***EINVAL*** if any of the settings are invalid.

*Note: The match output pins for **CT32B0** are not available, so the only allowed match output action for it is **LPC1114_TIMER_MATCH_OUTPUT_DISABLED**.*

22 SPIAGENT_CMD_CONFIGURE_TIMER_MATCH0_VALUE
23 SPIAGENT_CMD_CONFIGURE_TIMER_MATCH1_VALUE
24 SPIAGENT_CMD_CONFIGURE_TIMER_MATCH2_VALUE
25 SPIAGENT_CMD_CONFIGURE_TIMER_MATCH3_VALUE

The command message ***data*** field will be written to the corresponding match register (***MR0-3***) for the LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command message ***pin*** field.

The response message ***error*** field will be set to ***ENODEV*** if the timer identifier is invalid.

26 SPIAGENT_CMD_GET_TIMER_VALUE

The current value in the counter register (***TC***) for the LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command register ***pin*** field will be returned in the response message ***data*** field.

The response message ***error*** field will be set to ***ENODEV*** if the timer identifier is invalid.

27 SPIAGENT_CMD_GET_TIMER_CAPTURE

The current value in the capture register (***CR0***) for the LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command register ***pin*** field will be returned in the response message ***data*** field.

The response message ***error*** field will be set to ***ENODEV*** if the timer identifier is invalid.

28 SPIAGENT_CMD_GET_TIMER_CAPTURE_DELTA

The difference between the last two capture values for the LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command register ***pin*** field will be returned in the response message ***data*** field.

The response message ***error*** field will be set to ***ENODEV*** if the timer identifier is invalid.

*Note: This service is useful for taking rudimentary frequency measurements, by calculating the **frequency** from the **time** between pulse edges using:*

f=1/t

29 SPIAGENT_CMD_INIT_TIMER

The LPC1114 counter/timer (***CT32B0*** or ***CT32B1***) indicated by the command message ***pin*** field will be powered on and then initialized by writing zeros to all of the timer registers. This service should be called in timer object constructors to guarantee proper initial state of the timer hardware.

C Language API Reference

The following C service functions are included in ***`libspiagent.so`*** to make C application development for the LPC1114 I/O Processor Expansion Board much easier. All of these functions return an ***`errno`*** value in the ***`error`*** parameter. A zero value returned in ***`error`*** indicates success; all other values indicate failures unless noted below.

SPI Agent Transport Type Definitions

```
// SPI Agent Firmware commands

typedef enum
{
  SPIAGENT_CMD_NOP,
  SPIAGENT_CMD_LOOPBACK,
  SPIAGENT_CMD_CONFIGURE_ANALOG_INPUT,
  SPIAGENT_CMD_CONFIGURE_GPIO_INPUT,
  SPIAGENT_CMD_CONFIGURE_GPIO_OUTPUT,
  SPIAGENT_CMD_CONFIGURE_PWM_OUTPUT,
  SPIAGENT_CMD_GET_ANALOG,
  SPIAGENT_CMD_GET_GPIO,
  SPIAGENT_CMD_PUT_GPIO,
  SPIAGENT_CMD_PUT_PWM,
  SPIAGENT_CMD_CONFIGURE_GPIO_INTERRUPT,
  SPIAGENT_CMD_CONFIGURE_GPIO,
  SPIAGENT_CMD_PUT_LEGORC,
  SPIAGENT_CMD_GET_SFR,
  SPIAGENT_CMD_PUT_SFR,
  SPIAGENT_CMD_CONFIGURE_TIMER_MODE,
  SPIAGENT_CMD_CONFIGURE_TIMER_PRESCALER,
  SPIAGENT_CMD_CONFIGURE_TIMER_CAPTURE,
  SPIAGENT_CMD_CONFIGURE_TIMER_MATCH0,
  SPIAGENT_CMD_CONFIGURE_TIMER_MATCH1,
  SPIAGENT_CMD_CONFIGURE_TIMER_MATCH2,
  SPIAGENT_CMD_CONFIGURE_TIMER_MATCH3,
  SPIAGENT_CMD_CONFIGURE_TIMER_MATCH0_VALUE,
  SPIAGENT_CMD_CONFIGURE_TIMER_MATCH1_VALUE,
  SPIAGENT_CMD_CONFIGURE_TIMER_MATCH2_VALUE,
  SPIAGENT_CMD_CONFIGURE_TIMER_MATCH3_VALUE,
  SPIAGENT_CMD_GET_TIMER_VALUE,
  SPIAGENT_CMD_GET_TIMER_CAPTURE,
  SPIAGENT_CMD_GET_TIMER_CAPTURE_DELTA,
  SPIAGENT_CMD_INIT_TIMER,
  SPIAGENT_CMD_SENTINEL
} SPIAGENT_COMMAND_t;
```

```
// SPI Agent Firmware commmand message structure

typedef struct
{
  uint32_t command;
  uint32_t pin;
  uint32_t data;
} SPIAGENT_COMMAND_MSG_t;

// SPI Agent Firmware response message structure

typedef struct
{
  uint32_t command;
  uint32_t pin;
  uint32_t data;
  uint32_t error;
} SPIAGENT_RESPONSE_MSG_t;
```

SPI Agent Transport Service Functions

spiagent_open()

```
void spiagent_open(char *servername, int32_t *error);
```

This service opens a connection to the specified SPI Agent Firmware server.

The ***servername*** parameter indicates the Raspberry PI SPI Agent Firmware server to use. It has the following syntax:

```
[http://|ioctl://|oncrpc://|xmlrpc://]hostname
```

If the method (***http://*** et al) is omitted, a default method that is platform dependent will be used. The default method for the Raspberry Pi is ***ioctl://***. For other platforms, the default method will depend on how ***libspiagent.so*** was built, but will typically be ***http://***.

spiagent_command()

```
void spiagent_command(SPIAGENT_COMMAND_MSG_t *cmd,
  SPIAGENT_RESPONSE_MSG_t *resp, int32_t *error);
```

This service issues a command message to the SPI Agent Firmware and retrieves a response message.

The ***cmd*** parameter is a pointer to a ***SPIAGENT_COMMAND_MSG_t*** command message structure.

The ***resp*** parameter is a pointer to a ***SPIAGENT_RESPONSE_MSG_t*** response message structure.

spiagent_close()

```
void spiagent_close(int32_t *error);
```

This service closes the connection to the SPI Agent Firmware server.

GPIO Type Definitions

```
// SPI Agent Firmware GPIO data directions

typedef enum
{
  LPC1114_GPIO_INPUT,
  LPC1114_GPIO_OUTPUT,
  LPC1114_GPIO_DIRECTION_SENTINEL
} LPC1114_GPIO_DIRECTION_t;

// SPI Agent Firmware GPIO resistor directions

typedef enum
{
  LPC1114_GPIO_PULLDOWN,
  LPC1114_GPIO_PULLUP,
  LPC1114_GPIO_RESISTOR_SENTINEL
} LPC1114_GPIO_RESISTOR_t;
```

```
// SPI Agent Firmware GPIO configuration modes

typedef enum
{
  LPC1114_GPIO_MODE_INPUT,         // High impedance input
  LPC1114_GPIO_MODE_INPUT_PULLDOWN,
  LPC1114_GPIO_MODE_INPUT_PULLUP,
  LPC1114_GPIO_MODE_OUTPUT,        // Push-pull output
  LPC1114_GPIO_MODE_OUTPUT_OPENDRAIN,
  LPC1114_GPIO_MODE_SENTINEL
} LPC1114_GPIO_MODE_t;

// SPI Agent Firmware GPIO input interrupt configurations

typedef enum
{
  LPC1114_GPIO_INTERRUPT_DISABLED,
  LPC1114_GPIO_INTERRUPT_FALLING,
  LPC1114_GPIO_INTERRUPT_RISING,
  LPC1114_GPIO_INTERRUPT_BOTH,
  LPC1114_GPIO_INTERRUPT_SENTINEL
} LPC1114_GPIO_INTERRUPT_CONFIG_t;
```

GPIO Services

spiagent_gpio_configure()

```
void spiagent_gpio_configure(uint32_t pin, uint32_t direction,
  uint32_t state, int32_t *error);
```

This service configures an LPC1114 GPIO pin, setting the direction and state in a single operation.

The ***pin*** parameter indicates the pin to configure.

The ***direction*** parameter indicates whether to configure the pin as input or output. Use values from ***LPC1114_GPIO_DIRECTION_t***.

The ***state*** parameter sets the initial state for output pins (*0* or *1*) and the resistor configuration for input pins (*0*=pull-down, *1*=pull-up).

spiagent_gpio_configure_mode()

```
void spiagent_gpio_configure_mode(uint32_t pin, uint32_t mode,
  int32_t *error);
```

This service also configures an LPC1114 GPIO pin. It allows configuring an input as high impedance or an output as open drain. It does not allow setting the initial state of an output pin.

The ***pin*** parameter indicates the pin to configure.

The ***mode*** parameter indicates how to configure the pin. Use values from ***LPC1114_GPIO_MODE_t***.

spiagent_gpio_configure_interrupt()

```
void spiagent_gpio_configure_interrupt(uint32_t pin,
  uint32_t intconfig, int32_t *error);
```

This service configures an LPC1114 GPIO input pin interrupt.

The ***pin*** parameter indicates the pin to configure.

The ***intconfig*** parameter indicates how to configure the interrupt. Use values from ***LPC1114_GPIO_INTERRUPT_CONFIG_t***.

spiagent_gpio_get()

```
void spiagent_gpio_get(uint32_t pin, uint32_t *state,
  int32_t *error);
```

This service reads the state of a GPIO input *or* output pin.

The ***pin*** parameter indicates the pin to read from.

The ***state*** parameter will be set to the current state of the GPIO pin.

spiagent_gpio_put()

```
void spiagent_gpio_put(uint32_t pin, uint32_t state, int32_t *error);
```

This service writes to a GPIO output pin.

The ***pin*** parameter indicates the pin to write to.

The ***state*** parameter (***0*** or ***1***) indicates the level to write to the pin.

Analog Input Services

spiagent_analog_configure()

```
void spiagent_analog_configure(uint32_t pin, int32_t *error);
```

This service configures an LPC1114 GPIO pin as an analog input.

The ***pin*** parameter indicates the pin to configure. Allowed values are ***LPC1114_AD1*** though ***LPC1114AD5***.

spiagent_analog_get()

```
void spiagent_analog_get(uint32_t pin, float *voltage,
  int32_t *error);
```

This service samples an analog input pin.

The ***pin*** parameter indicates the pin to read from. Allowed values are ***LPC1114_AD1*** though ***LPC1114AD5***.

The ***voltage*** parameter well be set to the voltage sampled at the analog input, with a range from 0.0 to 3.3V.

Pulse Width Modulator Output Services

spiagent_pwm_set_frequency()

```
void spiagent_pwm_set_frequency(uint32_t freq, int32_t *error);
```

This service sets the PWM pulse frequency for all four PWM channels.

The ***freq*** parameter sets the PWM pulse frequency in Hz. The allowed frequency range is 50-50000 Hz.

spiagent_pwm_configure()

```
void spiagent_pwm_configure(uint32_t pin, int32_t *error);
```

This service configures an LPC1114 GPIO pin as a PWM output.

The ***pin*** parameter indicates the pin to configure. Allowed values are ***LPC1114_PWM1*** through ***LPC1114_PWM4***.

spiagent_pwm_put()

```
void spiagent_pwm_put(uint32_t pin, float dutycycle, int32_t *error);
```

This service sets the output duty cycle of a PWM output pin.

The ***pin*** parameter indicates the pin to set. Allowed values are ***LPC1114_PWM1*** through ***LPC1114_PWM4***.

The ***dutycycle*** parameter sets the output pulse train duty cycle. Allowed values are ***0*** through ***100.0*** percent, ***0*** indicating mostly off/low output and ***100.0*** percent indicating mostly on/high output.

Note: The actual resolution of the duty cycle varies, depending on the pulse frequency, from a full 16-bits at low frequencies to as little as 6-bits at higher frequencies.

spiagent_servo_configure()

```
void spiagent_servo_configure(uint32_t pin, int32_t *error);
```

This service configures one of the PWM outputs for driving a servo. The PWM frequency must be 50-400Hz.

The ***pin*** parameter indicates the pin to configure. Allowed values are ***LPC1114_PWM1*** through ***LPC1114_PWM4***.

spiagent_servo_put()

```
void spiagent_servo_put(uint32_t pin, float position,
  int32_t *error);
```

This service sets the position of an RC servo motor connected to a PWM output pin.

The ***pin*** parameter indicates the pin to set. Allowed values are ***LPC1114_PWM1*** through ***LPC1114_PWM4***.

The ***position*** parameter sets the position of the RC servo motor. Allowed values are ***-1.0*** through ***+1.0***, relative deflection from the null position. A value of ***0.0*** selects the null position.

spiagent_motor_configure()

```
void spiagent_motor_configure(uint32_t pwmpin, uint32_t dirpin,
  int32_t *error);
```

This service configures a pair of GPIO pins (PWM output and direction output) for driving an H-bridge DC motor driver.

The ***pwmpin*** parameter indicates the pin to configure as the PWM output. Allowed values are ***LPC1114_PWM1*** through ***LPC1114_PWM4***.

The ***dirpin*** parameter indicates the pin to configure as the direction output. Allowed values are ***LPC1114_GPIO0*** through ***LPC1114_GPIO7***.

pwmpin and ***dirpin*** cannot be the same pin.

spiagent_motor_put()

```
void spiagent_motor_put(uint32_t pwmpin, uint32_t dirpin,
  float speed, int32_t *error);
```

This service sets the speed for a H-bridge driven DC motor.

The ***pwmpin*** parameter indicates the pin configured as the PWM output. Allowed values are ***LPC1114_PWM1*** through ***LPC1114_PWM4***.

The ***dirpin*** parameter indicates the pin configured as the direction output. Allowed values are ***LPC1114_GPIO0*** through ***LPC1114_GPIO7***.

pwmpin and ***dirpin*** cannot be the same pin.

The ***speed*** parameter sets the normalized speed of the motor. Allowed values are ***-1.0*** (full speed reverse) through ***+1.0*** (full speed forward). A value of ***0.0*** stops the motor.

Timer Type Definitions

```
typedef enum
{
  LPC1114_CT32B0,
  LPC1114_CT32B1,
  LPC1114_TIMER_ID_SENTINEL
} spiagent_timer_id_t;

typedef enum
{
  LPC1114_TIMER_MODE_DISABLED,
  LPC1114_TIMER_MODE_RESET,
  LPC1114_TIMER_MODE_PCLK,
  LPC1114_TIMER_MODE_CAP0_RISING,
  LPC1114_TIMER_MODE_CAP0_FALLING,
  LPC1114_TIMER_MODE_CAP0_BOTH,
  LPC1114_TIMER_MODE_SENTINEL
} spiagent_timer_mode_t;

typedef enum
{
  LPC1114_TIMER_CAPTURE_EDGE_DISABLED,
  LPC1114_TIMER_CAPTURE_EDGE_CAP0_RISING,
  LPC1114_TIMER_CAPTURE_EDGE_CAP0_FALLING,
  LPC1114_TIMER_CAPTURE_EDGE_CAP0_BOTH,
  LPC1114_TIMER_CAPTURE_EDGE_SENTINEL
} spiagent_timer_capture_edge_t;

typedef enum
{
  LPC1114_TIMER_MATCH_OUTPUT_DISABLED,
  LPC1114_TIMER_MATCH_OUTPUT_CLEAR,
  LPC1114_TIMER_MATCH_OUTPUT_SET,
  LPC1114_TIMER_MATCH_OUTPUT_TOGGLE,
  LPC1114_TIMER_MATCH_OUTPUT_SENTINEL
} spiagent_timer_match_output_action_t;
```

Timer Services

Note: the internal clock of the LPC1114 microcontroller is only accurate to ±1% over the temperature range of -40° C to +85° C.

These services are complicated, in order to support all of the features of the LPC1114 timers. You will need to carefully study the ***LPC11xx User Guide*** in order to fully understand how the timers work. Experimenting with ***test_timer.c*** in the ***timer/*** directory will also be helpful.

spiagent_timer_init()

```
void spiagent_timer_init(uint32_t timer, int32_t *error);
```

This service powers on and initializes an LPC1114 timer.

The ***timer*** parameter selects which timer to configure. Use values from ***spiagent_timer_id_t***.

spiagent_timer_configure_mode()

```
void spiagent_timer_configure_mode(uint32_t timer,
  uint32_t mode, int32_t *error);
```

This service configures an LPC1114 timer's operating mode and clock source.

The ***timer*** parameter selects which timer to configure. Use values from ***spiagent_timer_id_t***.

The ***mode*** parameter selects the whether the timer is disabled, held in reset, or clocked from the LPC1114 internal system peripheral clock signal ***PCLK*** or from the ***CAP0*** capture input pin. Use values from ***spiagent_timer_mode_t***.

spiagent_timer_configure_prescaler()

```
void spiagent_timer_configure_prescaler(uint32_t timer,
  uint32_t divisor, int32_t *error);
```

This service configures an LPC1114 timer's prescale counter.

The ***timer*** parameter selects which timer to configure. Use values from ***spiagent_timer_id_t***.

The ***divisor*** parameter sets the prescale counter divisor. Allowed values are ***1*** through ***2^{32}-1*** (***0x00000001*** through ***0xFFFFFFFF***).

spiagent_timer_configure_capture()

```
void spiagent_timer_configure_capture(uint32_t timer,
  uint32_t edge, uint32_t intr, int32_t *error);
```

This service configures an LPC1114 timer's capture feature. If a counter capture event is triggered by a transition at the capture input pin ***CAP0***, the value of the counter register ***TC*** at that instant is copied to the capture register ***CR0***.

The ***timer*** parameter selects which timer to configure. Use values from ***spiagent_timer_id_t***.

The ***edge*** parameter selects which ***CAP0*** input edge or edges trigger a counter capture event. Use values from ***spiagent_timer_capture_edge_t***.

The ***intr*** parameter selects whether an interrupt to the Raspberry Pi is generated upon a capture event.

spiagent_timer_configure_match()

```
void spiagent_timer_configure_match(uint32_t timer,
  uint32_t match, uint32_t value, uint32_t action,
  uint32_t intr, uint32_t reset, uint32_t stop, int32_t *error);
```

This service configures an LPC1114 timer match register. A match will occur each time the counter register equals the match register.

The ***timer*** parameter selects which timer to configure. Use values from ***spiagent_timer_id_t***.

The ***match*** parameter selects which match register to configure. Legal values are ***0*** to ***3*** inclusive.

The ***value*** parameter will be written to the match register.

The ***action*** selects what will happen at the match output pin (***MAT0*** through ***MAT3***) if a match occurs. Not timers have all match outputs available. Use values from ***spiagent_timer_match_output_action_t***.

The ***intr*** parameter selects whether an interrupt to the Raspberry Pi is generated upon a match.

The ***reset*** parameter selects whether the timer is reset to zero upon a match.

The ***stop*** parameter selects whether the timer is stopped upon a match.

spiagent_timer_get()

```
void spiagent_timer_get(uint32_t timer, uint32_t *count,
  int32_t *error);
```

This service retrieves the current value of an LPC1114 timer's counter register.

The ***timer*** parameter selects which timer the counter value will be retrieved from. Use values from ***spiagent_timer_id_t***.

The ***count*** parameter will be set to the current value of the counter register.

spiagent_timer_get_capture()

```
void spiagent_timer_get_capture(uint32_t timer, uint32_t *count,
  int32_t *error);
```

This service retrieves the current value of an LPC1114 timer's capture register (*i.e.* the last value captured).

The ***timer*** parameter selects which timer the capture value will be retrieved from. Use values from ***spiagent_timer_id_t***.

The ***count*** parameter will be set to the current value of the capture register.

spiagent_timer_get_capture_delta()

```
void spiagent_timer_get_capture_delta(uint32_t timer,
  uint32_t *count, int32_t *error);
```

This service retrieves the difference between the last two values captured in an LPC1114 timer's capture register. It is useful for frequency measurement, by using the equation ***f=1/t*** to derive the *frequency* of a pulse train from the time between successive edges.

The ***timer*** parameter selects which timer the capture value will be retrieved from. Use values from ***spiagent_timer_id_t***.

The ***count*** parameter will be set to the difference between the last two captured values.

LEGO® Power Functions Remote Control Type Definitions

```
typedef enum
{
  LEGORC_ALLSTOP,
  LEGORC_MOTORA,
  LEGORC_MOTORB,
  LEGORC_COMBODIRECT,
  LEGORC_COMBOPWM,
  LEGORC_MOTOR_SENTINEL
} legorc_motor_id_t;

typedef enum
{
  LEGORC_REVERSE,
  LEGORC_FORWARD,
  LEGORC_DIRECTION_SENTINEL
} legorc_direction_id_t;
```

LEGO® Power Functions Remote Control Services

spiagent_legorc_put()

```
void spiagent_legorc_put(uint32_t pin, uint32_t channel,
  uint32_t motor, uint32_t direction, uint32_t speed,
  int32_t *error);
```

This service transmits a LEGO® Power Functions RC command out the specified GPIO output pin driving an infrared emitter, which must have been previously configured as an output with ***spiagent_gpio_configure()***.

The ***pin*** parameter must be ***LPC1114_GPIO0*** through ***LPC1114_GPIO7***.

The ***channel*** parameter must be 1 through 4.

The ***motor*** parameter must be 0 through 4 (All Stop, Motor A, Motor B, Combo Direct, Combo PWM).

The ***direction*** parameter must be 0 for reverse or 1 for forward for Motor A or Motor B, and 0 for all other motor values.

The ***speed*** parameter must be 0 through 7 for Motor A or Motor B, 0-15 for Combo Direct, and 0-255 for Combo PWM.

Special Function Register Services

Special Function Registers are 32-bit wide peripheral registers in the LPC1114 address space. These services allow reading from and writing arbitrary data to LPC1114 peripheral registers. Only addresses in the ranges 40000000-4007FFF and 50000000-501FFFFF (LPC1114 peripheral registers) are accessible. Reference the ***LPC111x/LPC11Cxx User manual UM10398*** for register details and exercise caution.

spiagent_sfr_get()

```
void spiagent_sfr_get(uint32_t address, uint32_t *data,
  int32_t *error);
```

This service reads from an LPC1114 special function register.

spiagent_sfr_put()

```
void spiagent_sfr_put(uint32_t address, uint32_t data,
  int32_t *error);
```

This service writes to an LPC1114 special function register.

Interrupt Services

The interrupt services are ***only*** available to a program running directly on the Raspberry Pi with an LPC1114 I/O Processor Expansion Board attached. They are ***not*** available to remote client programs.

spiagent_interrupt_enable()

```
void spiagent_interrupt_enable(int32_t *error);
```

This service enables interrupt monitoring.

spiagent_interrupt_disable()

```
void spiagent_interrupt_disable(int32_t *error);
```

This service disables interrupt monitoring.

spiagent_interrupt_wait()

```
void spiagent_interrupt_wait(uint32_t timeout, int32_t *error);
```

This service waits for an interrupt from the LPC1114 microcontroller. The calling program will be blocked and will consume no CPU time while it is waiting.

The ***timeout*** parameter indicates how long in milliseconds to wait for an interrupt from the LPC1114 microcontroller. A value of *-1* means wait forever.

An ***error*** value of ***ENODATA*** indicates the timeout expired without receiving an interrupt from the LPC1114 microcontroller.

LPC1114 GPIO Pin Assignments

The following LPC1114 I/O pins are defined in ***include/spi-agent.h***: or one of its subordinate header files:

```
// LPC1114 I/O Processor GPIO pin assignments

#define LPC1114_INT      3    // PIO0_3  (Raspberry Pi GPIO4)
#define LPC1114_READY    11   // PIO0_11 (Raspberry Pi GPIO22)
#define LPC1114_LED      7    // PIO0_7  (Expansion board LED)
#define LPC1114_GPIO0    12   // PIO1_0  (Terminal block pos 2)
#define LPC1114_GPIO1    13   // PIO1_1  (Terminal block pos 3)
#define LPC1114_GPIO2    14   // PIO1_2  (Terminal block pos 4)
#define LPC1114_GPIO3    15   // PIO1_3  (Terminal block pos 5)
#define LPC1114_GPIO4    16   // PIO1_4  (Terminal block pos 6)
#define LPC1114_GPIO5    17   // PIO1_5  (Terminal block pos 7)
#define LPC1114_GPIO6    20   // PIO1_8  (Terminal block pos 8)
#define LPC1114_GPIO7    21   // PIO1_9  (Terminal block pos 9)

// LPC1114 I/O Processor GPIO special function pin aliases

#define LPC1114_AD1              LPC1114_GPIO0
#define LPC1114_AD2              LPC1114_GPIO1
#define LPC1114_AD3              LPC1114_GPIO2
#define LPC1114_AD4              LPC1114_GPIO3
#define LPC1114_AD5              LPC1114_GPIO4

#define LPC1114_PWM1             LPC1114_GPIO1
#define LPC1114_PWM2             LPC1114_GPIO2
#define LPC1114_PWM3             LPC1114_GPIO3
#define LPC1114_PWM4             LPC1114_GPIO7

#define LPC1114_CT32B1_CAP0      LPC1114_GPIO0
#define LPC1114_CT32B1_MAT0      LPC1114_GPIO1
#define LPC1114_CT32B1_MAT1      LPC1114_GPIO2
#define LPC1114_CT32B1_MAT2      LPC1114_GPIO3
#define LPC1114_CT32B1_MAT3      LPC1114_GPIO4
#define LPC1114_CT32B0_CAP0      LPC1114_GPIO5
```

Additional Online Documentation

SPI Agent Firmware C# Language API Reference (ONC-RPC):

http://git.munts.com/rpi-mcu/expansion/LPC1114/src/csharp/SPIAgent/Lib-ONC-RPC/API.html

SPI Agent Firmware C# Language API Reference (XML-RPC):

http://git.munts.com/rpi-mcu/expansion/LPC1114/src/csharp/SPIAgent/Lib-XML-RPC/API.html

SPI Agent Firmware Python3 Language API Reference

http://git.munts.com/rpi-mcu/expansion/LPC1114/python/API.html

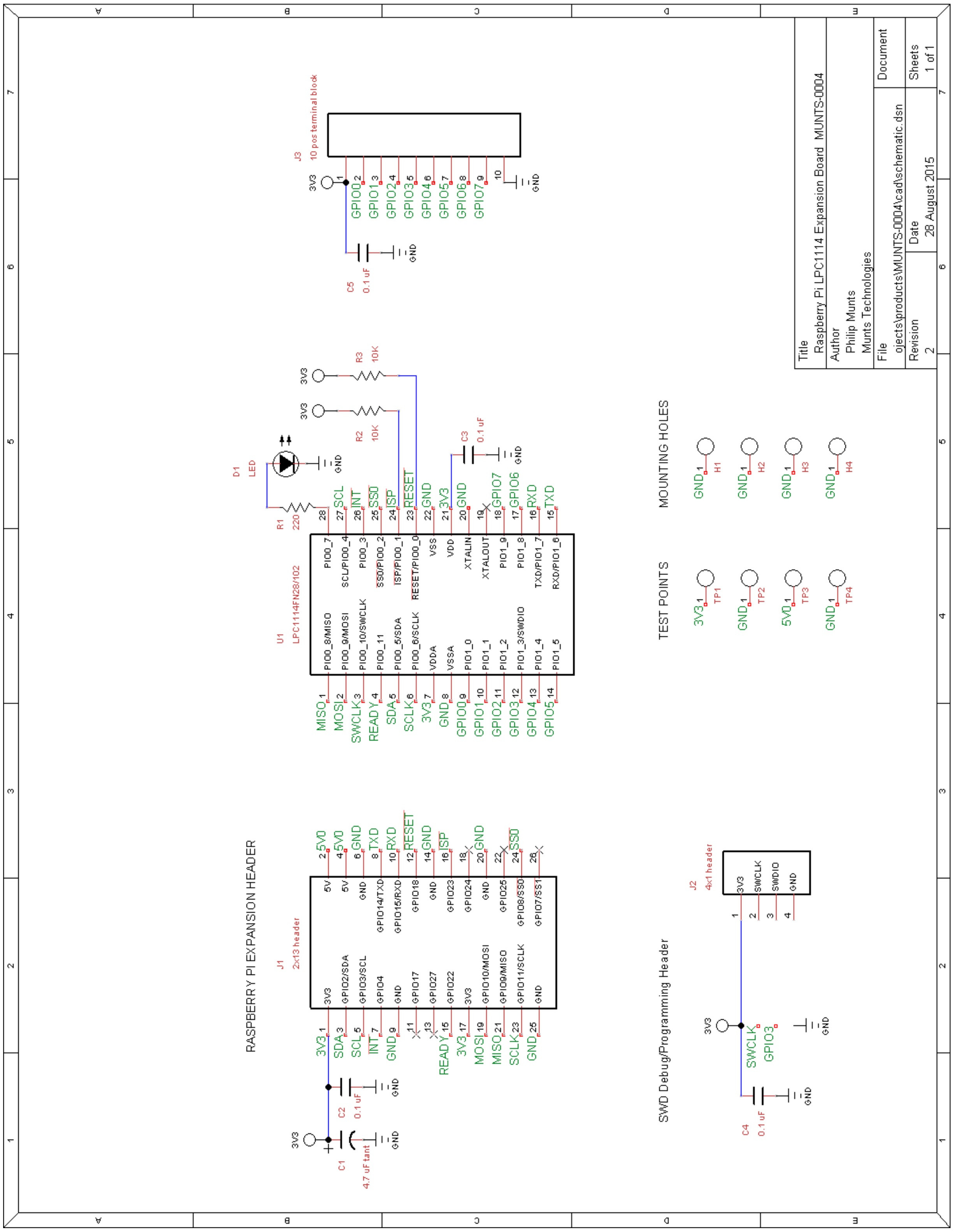
LPC1114 Data Sheet:

http://www.nxp.com/documents/data_sheet/LPC111X.pdf

LPC1114 User Manual:

http://www.nxp.com/documents/user_manual/UM10398.pdf

Schematic Diagram



Revision History

Board

Revision 1	17 December 2012 – First prototype.
Revision 2	25 January 2013 – Reduced the size of the board.

User Guide

Revision 1.0	29 June 2013 – Published for Revision 2 board.
Revision 1.1	13 July 2013 – Changed "expansion" to "I/O Processor" throughout. Added note that no extra hardware is required for programming the LPC1114 flash.
Revision 1.2	15 July 2013 – Added Quick Setup with shortened URL. Added more explanatory notes here and there.
Revision 1.3	27 July 2013 – Added instructions for updating the LPC1114 SPI Agent factory firmware.
Revision 1.4	23 August 2013 – The RPC servers are now enabled by default, so the text describing how to enable them was removed. Removed ***liblockdev1-dev*** from the list of required packages. Renamed the "Developing Firmware" section to "Software Support".
Revision 1.5	9 October 2013 – Changed "I/O Processor Board" to "I/O Processor Expansion Board" throughout. Added section describing the ***libspiagent.so*** shared library. Other minor edits.
Revision 1.6	25 October 2013 – Added Java to the list of languages that can use ***libspiagent.so***. Other minor edits.
Revision 1.7	16 December 2013 – Added SPI Agent Firmware API Reference section.
Revision 1.8	31 January 2014 – Expanded the API reference section to include all of the C functions in ***libspiagent.so***. Added timer services. Added section about the HTTP server.
Revision 1.9	26 March 2014 – Renamed ***led_test*** to ***test_led***. Added SFR and LEGO® RC services for C to ***libspiagent.so***.

Revision 1.10	8 April 2014 – Added optional ***cmd=*** prefix to the HTTP server query string. Added trailing semicolon to the HTTP server response string.
Revision 1.11	3 June 2014 – Added missing interrupt enable bit to the specification for ***SPIAGENT_CMD_CONFIGURE_TIMER_CAPTURE***. Added ***SPIAGENT_CMD_INIT_TIMER*** and the corresponding C library function ***spiagent_timer_init()***.
Revision 12	6 June 2014 – Renamed the C library functions ***spiagent_pwm_init()*** to ***spiagent_pwm_set_frequency()*** and ***spiagent_timer_get_counter()*** to ***spiagent_timer_get()***.
Revision 13	11 June 2014 – Reference ***spi_agent_test*** in ***spi-agent-lib/clients/c/*** instead of in ***spi-agent/***.
Revision 14	25 June 2014 – Added support for LEGO® Power Functions Remote Control Combo Direct and Combo PWM commands.
Revision 15	7 October 2014 – Changed the C library function ***spiagent_analog_get()*** to return a ***float*** voltage measurement, with a range of 0.0 to 3.3 volts. Changed the C library function ***spiagent_pwm_put()*** to take a ***float*** duty cycle value, with a range of 0.0 to 100.0 percent. Mention PWM outputs in Table 1. Added section about ***MuntsOS***.
Revision 16	13 October 2014 – Clarified the timer service definitions.
Revision 17	21 October 2014 – Added ***spiagent_servo_set_postion()***, for controlling RC servo motors.
Revision 18	24 October 2014 – Removed the ***runtest*** target from ***testprogram.mk***. Modified ***testprogram.mk*** to allow multiple test programs per directory.
Revision 19	6 November 2014 – Allow configuring the LED and interrupt output pins with ***SPIAGENT_CMD_CONFIGURE_GPIO***.
Revision 20	7 November 2014 – Use ***expansion_lpc1114_flash*** instead of ***make install_factory_firmware***.
Revision 21	6 February 2015 – Renamed ***muntsos-lpc1114.zip*** to ***muntsos-lpc1114-RaspberryPi.zip***. Added links to additional online documentation.

Revision 22	6 March 2015 – Added ***spiagent_servo_configure()***. Renamed ***spiagent_servo_set_position()*** to ***spiagent_servo_put()***. Added ***spiagent_motor_configure()*** and ***spiagent_motor_put()***. Also lifted the 50 Hz PWM pulse rate requirement for servos; servos can now be configured with pulse rates of 50-400 Hz. (Standard RC servos still need 50 Hz; some newer digital servos can go as high as 400 Hz.)
Revision 23	23 March 2015 – Moved the ARM microcontroller toolchain installation directory from ***/opt/arm-mcu-tools*** to ***/usr/local/arm-mcu-tools***.
Revision 24	11 June 2015 – Moved the source code repositories to: ***http://git.munts.com***
Revision 25	28 August 2015 – Refined ***LPC1114_INT1*** as ***LPC1114_INT*** and ***LPC1114_INT2*** as ***LPC1114_READY***.
Revision 26	2 October 2015 – Renamed ***LPC1114_PWM4/5/6*** to ***LPC1114_PWM1/2/3***. Added support for fourth PWM outout, ***LPC1114_PWM4*** on ***GPIO7***.
Revision 27	24 December 2015 – Renamed the script file ***00-wlan-RaspberryPi*** to ***00-wlan-RTL8192CU***. Restored the proper schematic for the Revision 2 PCB.
Revision 28	6 June 2016 – The software for the LPC1114 I/O Processor Expansion Board is now delivered as Debian package files. This greatly simplifies the setup procedure.
Revision 29	8 June 2016 – Added ***System Preparation*** section, about settting up the Raspberry Pi before installing software support for the LPC1114 I/O Processor Expansion Board.
Revision 30	19 October 2016 – Changed URLs from ***tech.munts.com*** to either ***git.munts.com*** or ***repo.munts.com***.
Revision 31	22 January 2017 – Reference the MuntsOS Thin Servers repository instead of a specific file.
Revision 32	28 August 2017 – Updated the source directory path for the ***test_led*** test program. Use the latest the output from ***spi_agent_test***.

Revision 33	7 September 2017 – Changed all ***unsigned*** and ***bool*** to ***uint32_t***. Changed all ***int*** to ***int32_t***.
Revision 34	18 September 2017 -- Renamed ***expansion_lpc1114_setup*** to ***expansion_lpc1114_setup.RaspberryPi***.
Revision 35	18 May 2018 – Switched from the old deprecated GPIO ***sysfs*** interface to the new GPIO descriptor interface. Raspberry Pi device and GPIO pins are now assigned in the configuration file ***/usr/local/etc/expansion_lpc1114.config***.
Revision 36	12 June 2018 – Removed the ***pin*** parameter from ***spiagent_interrupt_wait()***.
Revision 37	4 February 2018 – Renamed the repository for the ***MuntsOS Embedded Linux Framework*** from http://get.munts.com/arm-linux-mcu to http://git.munts.com/muntsos.
Revision 38	6 February 2018 – Renamed the setup script ***expansion_lpc1114_setup.RaspberryPi*** back to ***expansion_lpc1114_setup***.
Revision 39	17 May 2019 – Renamed ***SPIAGENT_GPIO_**** to ***LPC1114_GPIO_****, ***SPIAGENT_TIMER_**** to ***LPC1114_TIMER_**** and ***SPIAGENT_CT32B**** to ***LPC1114_CT32B****.
Revision 40	12 November 2019 – Removed reference to the ***MuntsOS Embedded Linux Framework*** and the LPC1114 I/O Processor Thin Server. The LPC1114 I/O Processor extension package is still available for MuntsOS.