How to control the IO on the SBC boards

From BananaPro/Pi

Contents

Description

Most LeMaker SBCs boards (Banana Pi/Pro, LeMaker Guitar Base Board Rev.B) has some IO ports on the boards, including GPIO, PWM, SPI, I2C, UART, etc.. In fact, the 40 Pins on Banana Pro (BananaPro/Pi:Pin definition) and LeMaker Guitar Base Board Rev.B (LeMaker_Guitar:Pin_Definition_on_Base_Board) are fully compatible with almost the same Pin definitions. Via the basic way, we can control the digital IO ports input or output from the sys user space. But if we want to use more complex interface such as SPI or I2C, it is not that easy to achieve. Thus we have provided two GPIO libraries to use: WiringLMK and LMK.GPIO, offering various API to easily control the IO on the SBC boards. Currently, both the WiringLMK C library and the LMK.GPIO Python library support Banana Pro and LeMaker Guitar. In the following, we will introduce you how to use these libraries with different numbering scheme. Note, WiringLMK and LMK.GPIO has its own different numbering scheme for the Pin definitions (#Appendix: 40 Pins GPIO Mapping Table for Banana Pro and LeMaker Guitar).

Before everything, when checking up the table, we'd be clear that the **Physical Numbering Scheme** means the physical pins on the board for both Banana Pro and LeMaker Guitar, that is, when we use function `wiringPiSetupPhys()` in WiringLMK library, or the definition `LMK.GPIO.BOARD` in LMK.GPIO library, for example, define `PIN = 8` will return the **physical pin 8** on the board. The definition **sys** means the sys user space ways for each board. When we use function `wiringPiSetupGpio()` and `wiringPiSetupSys()` in WiringLMK library, or the definition `LMK.GPIO.BCM` in LMK.GPIO library, it will return the **BCM Chip Serial Number** Scheme, for example, define `PIN = 8` will return the **physical pin 24** on the board. WiringLMK also has its own definition, when we use function `wiringPiSetup()` in WiringLMK library, for example, define `PIN = 8` will return the **physical pin 3** on the board.

Installation

For WiringLMK, please refer to: https://github.com/LeMaker/WiringLMK .
For LMK.GPIO, please refer to: https://github.com/LeMaker/LMK.GPIO .

Compilation

LEMAKER HOMEDOWNLOADFORUMBLOGPRODUCT

Before using the WiringLMK GPIO library, you need to include its header file in your programs: `#include <wiringPi.h>`. You may also need additional `#include` lines, depending on the modules you are using. To compile an example using WiringLMK, you may have to add

```
-I/usr/local/include -L/usr/local/lib -lwiringPi
```

to the compile line depending on the running environment, the important parameters you might need are listed: `-lwiringPi`, `-lwiringPiDev`, or `-lpthread`.

LMK.GPIO Python library

Python language doesn't require to compile, but you have to import it as `import LMK.GPIO` in the very beginning of the program.

Digital Input/Output

Use sysfs user space

The GPIO pins can be accessed from user space using sysfs. By default, the mapping gpio_operation from the physical GPIO to the sys file system is enabled in the kernels. Before exporting or unexporting, you have to obtain the correct numbering index from the pin name, the calculation is as below for LeMaker Guitar Base Board Rev.B:

```
GPIO A: GPIOA*NUMBER* = 0 + *NUMBER*, for example, GPIOA27 = 0+27 = 27
GPIO B: GPIOB*NUMBER* = 32 + *NUMBER*, for example, GPIOB31 = 32+31 = 63
GPIO C: GPIOC*NUMBER* = 64 + *NUMBER*, for example, GPIOC0 = 64+0 = 64
GPIO D: GPIOD*NUMBER* = 96 + *NUMBER*, for example, GPIOD3 = 96+3 = 99
GPIO E: GPIOE*NUMBER* = 128 + *NUMBER*, for example, GPIOE22 = 128+22 = 150
```

or, you can simply check the #Appendix: 40 Pins GPIO Mapping Table for Banana Pro and LeMaker Guitar to obtain the correct sys Pin number. To access a GPIO pin such as **GPIOB19** (32+19 = 51) you first need to **export** it with

```
$ echo XX > /sys/class/gpio/export
```

with **XX** being the numbering index of the desired pin, here **XX** = 51. If succeed, you would find the sys file system node

```
$ ls
#! /sys/class/gpio/gpioXX is generated
```

To set a GPIO pin as output you have to change the input/output **direction** with

```
$ echo out > /sys/class/gpio/gpioXX/direction
```

To set a GPIO pin as input you have to change the input/output **direction** with

```
$ echo in > /sys/class/gpio/gpioXX/direction
```

You can read a GPIO pin with `/sys/class/gpio/gpioXX/value` using

```
$ cat /sys/class/gpio/gpioXX/value
```

When the **direction** is set to out, you can write **1** or **0** to a GPIO pin with `/sys/class/gpio/gpioXX/value` using

```
$ echo 1 > /sys/class/gpio/gpioXX/value
```

Use WiringLMK C language

The first step should always be the initializations of wiringLMK and assumes that the calling program is going to use the wiringLMK Pin numbering scheme. Here the physical Pin mode is specified by using the function

```
int wiringPiSetupPhys (void) ;
```

which means the calling program use the Pin on boards as its physical numbering scheme. To set a GPIO pin as input/output you have to change the Pin mode input/output direction using the function

```
void pinMode (int pin, int mode) ;
```

where `int mode` can be set to `INPUT`, `OUTPUT`, `PWM_OUTPUT` or `GPIO_CLOCK`. You can read a GPIO Pin value with its physical numbering index `int pin` using the function

When the direction is set to out, you can modify the GPIO Pin value by assigning `int value` with `HIGH` or `LOW` (`1` or `0`) using the function

```
void digitalWrite (int pin, int value) ;
```

When the direction is set to in, you can setup its pull-up or pull-down resistor mode on the given pin by

```
void pullUpDnControl (int pin, int pud) ;
```

assigning `PUD_OFF`(no pull up/down), `PUD_UP`(pull to VCC 3.3V) or `PUD_DOWN`(pull to GND) to `int pud` depending on what you need in the program.

Here is the "Hello World" example showing how to make an LED blinks:

```
#include <wiringPi.h>
int main (void)
{
  wiringPiSetupPhys () ;
  pinMode (11, OUTPUT) ;
  for (;;)
  {
    digitalWrite (11, HIGH) ; delay (500) ;
    digitalWrite (11,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

Use LMK.GPIO Python language

Similarly, you always have to step up the Pin mode of SBC IO library; and for the LMK.GPIO, here the physical Pin numbering scheme is defined by using the function

```
GPIO.setmode(GPIO.BOARD)
```

which means the calling program use the Pin on boards as its physical numbering scheme. To set a GPIO pin as input/output you have to change the Pin mode input/output direction using the function

```
GPIO.setup(int pin, int mode)
```

where `int mode` can be set to `GPIO.IN`, `GPIO.OUT`. You can read a GPIO Pin value with its physical numbering index `int pin` using the function

```
GPIO.input(int pin)
```

When the direction is set to out, you can modify the GPIO Pin value by assigning `int value` with `GPIO.HIGH` or `GPIO.LOW` (`1` or `0`, `True` or `False`) using the function

```
GPIO.output(int pin, int value)
```

Also you can set the pull-up or pull-down resistor mode on the given pin by specifying `int pud` the third parameter

```
GPIO.setup(int pin, GPIO.IN, int pud)
```

with `GPIO.PUD_OFF`(no pull up/down), `GPIO.PUD_UP`(pull to VCC 3.3V), or `GPIO.PUD_DOWN`(pull to GND). Here is the "Hello World" example showing how to make an LED blinks:

```
#!/usr/bin/env python
import LMK.GPIO as GPIO
import time
#LED Mode BOARD
PIN_NUM = 7
GPIO.setmode(GPIO.BOARD)
while True:
        try:
                GPIO.setup(PIN_NUM, GPIO.OUT)
        except:
                print("Failed to setup GPIO %d", PIN_NUM)
GPIO.output(PIN_NUM, True)
        time.sleep(0.5)
        GPIO.output(PIN_NUM, False)
        time.sleep(0.5)
```

Interrupts

LEMAKER HOMEDOWNLOADFORUMBLOGPRODUCT

Before setup an interrupt pin, you normally need to set the pull-up or pull-down resistor mode first.

```
void pullUpDnControl (int pin, int pud) ;
```

The interrupt function is specified by using the

```
int wiringPiISR (int pin, int edge, void (*function)(void)) ;
```

where the `int edge` can be set to `INT_EDGE_RISING`, `INT_EDGE_FALLING`, or `INT_EDGE_BOTH` for both edges. The third parameter should be the interrupt handler function. In the below, it is an example of how the interrupt works.

```
#include <stdio.h>
#include <wiringPi.h>
void myInterrupt (void) { printf("The Physical Pin 16 is detected...\n"); }
int main(int argc, char **argv)
{
        wiringPiSetupPhys();
        pullUpDnControl(16, 1);
        wiringPiISR (16, INT_EDGE_FALLING, &myInterrupt) ;

        while(1)
        {
                delay (100) ;           //mS
        }
}
```

Use LMK.GPIO Python language

The interrupts are regarded as events that happened as `GPIO.RISING`, `GPIO.FALLING` or `GPIO.BOTH` for both edges in the second parameter of function

```
GPIO.add_event_detect(unsigned int pin, unsigned int edge,callback = void (*func_my_callback), unsigned int bouncetime)
```

The third parameter should be the interrupt event callback function. In the below, it is an example of how the interrupt works.

```
#!/usr/bin/env python
import LMK.GPIO as GPIO
import time
PIN_NUM = 24
GPIO.setmode(GPIO.BOARD)
GPIO.setup(PIN_NUM,GPIO.IN,GPIO.PUD_UP)
def my_callback(channel):
        print "The value of Pin %d is %d" %(PIN_NUM,GPIO.input(PIN_NUM))
        print "Callback trigger %d" %channel
        print "Now value of the Pin is %d" %(GPIO.input(PIN_NUM))
GPIO.add_event_detect(PIN_NUM,GPIO.RISING,callback = my_callback,bouncetime = 200)
try:
    while True:
        time.sleep(0.1)
except KeyboardInterrupt:
    GPIO.remove_event_detect(PIN_NUM)
GPIO.cleanup()
```

PWM

Use WiringLMK C language

WiringLMK includes a software-driven PWM handler capable of generating a PWM signal on any of the GPIO Pins. The minimum pulse width is not tested for each board depending on the hardware differences. However, the control of a light/LED or a motor is very achievable. Remember to include the file `#include <softPwm.h>` as the additional header and `-lwiringPi -lpthread` for compilations. To creates a software controlled PWM pin, it's suggested to set the second parameter `int pwmRange` to 100, where 0 means off and 100 means fully on for the given Pin, with function

```
int softPwmCreate (int pin, int initialValue, int pwmRange) ;
```

The initialized soft PWM pin can then updates its PWM value on the given pin (`int value` should be in-range) by

```
void softPwmWrite (int pin, int value) ;
```

Here is a short example of how to work with software PWM.

```
#include <stdio.h>
#include <wiringPi.h>
#include <softPwm.h>
#define RANGE           100
#define PIN             3
int main ()
{
        wiringPiSetupPhys()  ;
softPwmCreate (PIN, 0, RANGE) ;
```

Use LMK.GPIO Python language

LMK.GPIO also supports a software-driven PWM function and it's quite simple to setup for a given Pin. To creates a software controlled PWM pin, it's also suggested to set the second parameter `int frequency` to 100 (in theory, `frequency > 0` is sufficient) with function

```
p = GPIO.PWM(int pin, int frequency)
```

The initial value of this soft PWM is specified by `.start(int initialValue)`, and can be stopped by function `.stop()`. The duty cycle can be updated by `.ChangeDutyCycle(int dutyCycle)` where `int dutyCycle` should also be in-range. Here is the example of using LMK.GPIO software PWM.

```
#/usr/bin/env python
import LMK.GPIO as GPIO
import time
PIN_NUM = 11
frequency = 50
GPIO.setmode(GPIO.BOARD)
GPIO.setup(PIN_NUM,GPIO.OUT)
p = GPIO.PWM(PIN_NUM,frequency)
p.start(0)
raw_input("Press Enter to stop:")
p.stop()
GPIO.cleanup()
```

I2C Bus

Use WiringLMK C language

WiringLMK includes the functions which offer easier access to the on-board I2C interface. By default, the I2C driver is loaded to the kernel for both boards Banana Pro and LeMaker Guitar. To use the I2C functions, you need to:

```
#include <wiringPiI2C.h>
```

in the program and when compiling to link with `-lwiringPi`. There is also a standard system commands to check the I2C devices, for example, the `i2cdetect` program that using

```
i2cdetect -y 2 # Banana Pro and LeMaker Guitar
```

to detect the correct I2C address in the bus. Simple device read or write that do not require any register transactions while others will need to setup its own registers. The function

```
int wiringPiI2CSetup (int devID) ;
```

initialize the device on the I2C bus and the following functions operate on the device by

```
int wiringPiI2CRead (int fd) ;
int wiringPiI2CReadReg8 (int fd, int reg) ;
int wiringPiI2CReadReg16 (int fd, int reg) ;
int wiringPiI2CWrite (int fd, int data) ;
int wiringPiI2CWriteReg8 (int fd, int reg, int data) ;
int wiringPiI2CWriteReg16 (int fd, int reg, int data) ;
```

Example 1: Control MCP23017

The expansion board with MCP23017 also has its own interface that makes it more convenient to use. The additional header file `#include <mcp23017.h>` should be included. The function to setup the chip MCP23017

```
int mcp23017Setup (int pinBase, int i2cAddress) ;
```

The `int pinBase` can be any number above 64 and the i2cAddress is the address of the device in the I2C bus, this function can be called for multiple times with different `int pinBase` for each MCP23017 device. Remember to use the `i2cdetect` command to probe the I2C bus to return the correct address for devices. Here is the demo

```
#include <stdio.h>
#include <wiringPi.h>
#include <mcp23017.h>
#define LED                1
#define pinBASE 100
int main (void)
{
  wiringPiSetup () ; // Enable the on-goard GPIO
  mcp23017Setup (pinBASE, 0x24) ; // Initialize mcp23017 on the board
pinMode (LED, OUTPUT) ;
```

LEMAKER HOMEDOWNLOADFORUMBLOGPRODUCT

```
    digitalWrite (pinBASE + 0, HIGH) ;
    delay (500) ;
    digitalWrite (LED,          LOW) ;
    digitalWrite (pinBASE + 0, LOW) ;
    delay (500) ;
  }
  return 0 ;
}
```

Example 2: Control PCF8591(Analog)

Another supported expansion module is the PCF8591 Analog IO expander IC. The PCF8591 has a 4-channel, 8-bit analog input port and a single channel analog output port. PCF8591 has a 3-bit address select port, which allows to connect at most 8 PCF8591 to SBC. The additional header file `#include <pcf8591.h>` should be included. The function to setup the chip PCF8591

```
int pcf8591Setup (int pinBase, int i2cAddress) ;
```

The `int pinBase` can also any number above 64 and the i2cAddress is the address of the device in the I2C bus, and can be called for multiple times with different `int pinBase` for each PCF8591 device. Please also use the `i2cdetect` command to detect the correct address for devices. Here is the demo

```
#include <stdio.h>
#include <wiringPi.h>
#include <pcf8591.h>
#define LED                1
#define pinBASE       120
int main (void)
{
  int value, pin ;
  wiringPiSetup () ; // Enable the on-goard GPIO
  pinMode (LED, OUTPUT) ; // On-board LED
  pcf8591Setup (pinBASE, 0x48) ; // Initialize pcf8591 on the board
for (;;)
  {
    for (pin = 0 ; pin < 4 ; ++pin)
    {
      value = analogRead  (pinBASE + pin) ;
      printf ("  %5.2f  ", (double)value * 3.3 / 255.0) ;
    }
    printf ("\r") ; fflush (stdout) ;
delay (100) ;
    digitalWrite (LED, !digitalRead (LED)) ; // Flicker the LED
  }
  return 0 ;
}
```

Use Python language

LMK.GPIO does not have functions for SPI or I2C interfaces, thus here another Python library `smbus` is installed by

```
$ sudo apt-get install smbus
```

and imported in the beginning of the program with its reading and writing as

```
import smbus
Bus = smbus.SMBus(2) # smbus 2 for Banana Pro and LeMaker Guitar
Bus.write_byte_data(Address, Reg, Value) # Address on I2C bus
Value = Bus.read_byte_data(Address, Reg) # Register operator Reg
```

Example: Control MCP23017

For more detailed register configurations and usage, please download the MCP23017 datasheet. Here is only a simple demo of configuring an expansion board called LN_USB_HUB

```
import smbus
Bus = smbus.SMBus(2) # smbus 2 for Banana Pro and LeMaker Guitar
try:
        Bus.write_byte_data(0x24, 0x00, 0xFF)
        Bus.write_byte_data(0x24, 0x01, 0xFF)
        Bus.write_byte_data(0x24, 0x12, 0x55)
        Bus.write_byte_data(0x24, 0x12, 0x00)
Bus.write_byte_data(0x26, 0x00, 0xFF)
        Bus.write_byte_data(0x26, 0x01, 0xFF)
        Bus.write_byte_data(0x26, 0x12, 0x55)
        Bus.write_byte_data(0x26, 0x12, 0x00)
except:
        print("I2C error")
Address = [0x24, 0x26]
for Addr in Address:
        try:
            Bus.write_byte_data(Addr, 0x00, 0x00)
            Bus.write_byte_data(Addr, 0x01, 0x00)
            Bus.write_byte_data(Addr, 0x0A, 0x00)
        except:
            print (str(Addr) + " not availible.")
Addr = "24"
Bank = "A"
Binary = "01010101"
Binary = int(Binary, 2)
if Bank.upper() == "A":
        Bus.write_byte_data(int("0x"+Addr, 16), 0x12, Binary)
```

```
else:
        print("Incorrect Bank.")
```

SPI Bus

Use WiringLMK C language

WiringLMK includes the functions which offer easier access to the on-board SPI interface, too. By default, the SPI driver is not loaded to the kernel for Banana Pro but it is enabled for LeMaker Guitar permanently. To enable the SPI functions on Banana Pro, you need to add `spi-sun7i` and `spidev` in the modules

```
$ sudo nano /etc/modules
```

and also remember the additional header file for SPI interface should be included

```
#include <mcp23s17.h>
```

To open an SPI device and and setup its channels, the function

```
int wiringPiSPISetup (int channel, int speed) ;
```

initialize the device on the SPI bus and the following one

```
int wiringPiSPIDataRW (int channel, unsigned char *data, int len) ;
```

write and Read a block of data over the SPI bus, which works as a full-duplex operation.

Example 1: Control MCP23s17

The expansion board with MCP23s17 has its own interface as well in WiringLMK library. The additional header file `#include <mcp23s17.h>` should be included. The function to setup the chip MCP23s17

```
int mcp23s17Setup (int pinBase, int spiPort, int devID) ;
```

Similarly, the `int pinBase` can be any number above 64 and the `int spiPort` should be `0` or `1` depending on which SPI port is going to be open. Note that on Banana Pro there're two ports while on LeMaker Guitar it only has one such that `int spiPort = 0`. Here is the demo

```
#include <stdio.h>
#include <wiringPi.h>
#include <mcp23s17.h>
#define BASE    150
int main (void)
{
  int i;
wiringPiSetup () ;
  mcp23s17Setup (BASE, 0, 0) ;
for (i = 0 ; i < 8 ; ++i)
    pinMode (BASE + i, OUTPUT) ;
for (;;)
  {
     for (i = 0 ; i < 8; ++i)
     {
       digitalWrite (BASE + i, HIGH) ;
       delay (500) ;
       digitalWrite (BASE + i, LOW) ;
       delay (500) ;
     }
  }
  return 0 ;
}
```

Example 2: Control AT45DBXX DataFlash

The AT45DB041B is an SPI compatible serial interface Flash memory. The device is enabled through the chip select pin (CS) and accessed via a threewire interface consisting of the Serial Input (SI), Serial Output (SO), and the Serial Clock (SCK). There is no functions specialized for AT45DBXX DataFlash but to operate it by WiringLMK is achievable. For more completed read and write operations, please refer to the datasheet

```
#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
void at45BufWrite(unsigned char val)
{
        int ret;
        unsigned char wBuf[5] = {0x84, 0xff, 0x00, 0x02,0xff}; // write data to buffer


wBuf[4] = val;
```



```
        if(ret < 0)
        {
                printf("Write data to the AT45DB041D failed!\n");
                return;
        }
}

unsigned char at45BufRead(void)
{
        int ret;
        unsigned char rBuf[6] = {0xD4, 0xff, 0x00, 0x02, 0xff, 0xff}; // read data from buffer
ret = wiringPiSPIDataRW(0, rBuf, sizeof(rBuf));
        if(ret < 0)
        {
                printf("Read data from the AT45DB041D failed!\n");
                return;
        }
        return rBuf[5];
}

void SpiTest()
{
        int fd;
        unsigned char wData = 0x65; // write value
        unsigned char retv;


fd = wiringPiSPISetup(0, 5000000); // channel:0  5M
if(fd < 0)
        {
                printf("Open the SPI device failed!\n");
                return;
        }
        at45BufWrite(wData);
        delay(100); // delay 100ms
retv = at45BufRead();
        printf("SPI: read  data: 0x%02x\n", retv);
if(wData == retv)
        {
                printf("SPI: the spi interface is working!\n");
        }
        else
        {
                printf("SPI: the spi interface is not working! Please check out!\n");
        }
        close(fd);
}

int main(int argc, char **argv)
{
        wiringPiSetup();
        while(1)
        {
            SpiTest();
        }
        return 0;
}
```

Use Python language

LMK.GPIO does not have functions for SPI or I2C interfaces, thus here a simulated SPI operations are introduced by using LMK.GPIO `GPIO.input` and `GPIO.output` operations, function `SPIsend(opcode, addr, data)` to send data, function `SPIread(opcode, addr)` to read data, through SPI bus.

```
def SPIsendValue(value):
    GPIO.setup(SPI_MOSI, GPIO.OUT)
    GPIO.setup(SPI_SCLK, GPIO.OUT)
    for i in range(8):
        if (value & 0x80):
            GPIO.output(SPI_MOSI, GPIO.HIGH)
        else:
            GPIO.output(SPI_MOSI, GPIO.LOW)
        GPIO.output(SPI_SCLK, GPIO.HIGH)
        GPIO.output(SPI_SCLK, GPIO.LOW)
        value <<= 1
def SPIsend(opcode, addr, data):
    GPIO.setup(SPI_CS0, GPIO.OUT)
    GPIO.output(SPI_CS0, GPIO.LOW)
    SPIsendValue(opcode | SPI_SLAVE_WRITE)
    SPIsendValue(addr)
    SPIsendValue(data)
    GPIO.output(SPI_CS0, GPIO.HIGH)
def SPIread(opcode, addr):
    GPIO.setup(SPI_MISO, GPIO.IN)
    GPIO.output(SPI_CS0, GPIO.LOW)
    SPIsendValue(opcode | SPI_SLAVE_READ)
    SPIsendValue(addr)
    value = 0
    for i in range(8):
        value <<= 1
        if(GPIO.input(SPI_MISO)):
            value |= 0x01
        GPIO.output(SPI_SCLK, GPIO.HIGH)
        GPIO.output(SPI_SCLK, GPIO.LOW)
    GPIO.output(SPI_CS0, GPIO.HIGH)
    return value
```

Example: Control MCP23s17

For more detailed register configurations and usage, please download the MCP23s17 datasheet. Here is only a simple demo of configuring an expansion board called LN_Digital

```
# MCP23S17
SPI_SLAVE_ADDR = 0x40
SPI_IODIRA = 0x00
SPI_IODIRB = 0x01
SPI_GPIOA = 0x12
SPI_GPIOB = 0x13
SPI_IOCONA =0x0A
SPI_IOCONB =0x0B
SPI_OLATA =0x14
SPI_OLATB =0x15
SPI_INTENA = 0x04
SPI_INTENB = 0x05
SPI_DEFVALA = 0x06
SPI_DEFVALB = 0x07
SPI_INTCONA = 0x08
SPI_INTCONB = 0x09
SPI_GPPUA = 0x0C
SPI_GPPUB = 0x0D
SPI_SLAVE_WRITE = 0x00
SPI_SLAVE_READ = 0x01
# MCP23S17-Pins
SPI_SCLK = 11 # Serial-Clock
SPI_MOSI = 10 # Master-Out-Slave-In
SPI_MISO = 9  # Master-In-Slave-Out
SPI_CS0 = 8   # Chip-Select
SPIsend(SPI_SLAVE_ADDR, SPI_IOCONA, 0x00)
SPIsend(SPI_SLAVE_ADDR, SPI_IOCONB, 0x00)
SPIsend(SPI_SLAVE_ADDR, SPI_GPIOA, 0x00)
def LNdigital():
    try:
        GPIO.setup(SPI_SCLK, GPIO.OUT)
        GPIO.setup(SPI_MOSI, GPIO.OUT)
        GPIO.setup(SPI_MISO, GPIO.IN)
        GPIO.setup(SPI_CS0, GPIO.OUT)
GPIO.output(SPI_CS0, GPIO.HIGH)
        GPIO.output(SPI_SCLK, GPIO.LOW)
SPIsend(SPI_SLAVE_ADDR, SPI_IODIRA, 0x00)#outputs
        SPIsend(SPI_SLAVE_ADDR, SPI_IODIRB, 0xFF)#inputs
        SPIsend(SPI_SLAVE_ADDR, SPI_GPIOA, 0x00)
        SPIsend(SPI_SLAVE_ADDR, SPI_GPIOB, 0xFF)
        Test = SPIread(SPI_SLAVE_ADDR, SPI_GPIOA)
        if Test == 0x00:
            LNdigital = True
        else:
            LNdigital = False
    except:
        LNdigital = False
if __name__ == '__main__':
LNdigital()
    SPIsend(SPI_SLAVE_ADDR, SPI_IOCONA, 0x0A)
    SPIsend(SPI_SLAVE_ADDR, SPI_IOCONB, 0x0A)
    SPIsend(SPI_SLAVE_ADDR, SPI_IODIRA, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_IODIRB, 0xFF)
    SPIsend(SPI_SLAVE_ADDR, SPI_GPPUA, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_GPPUB, 0xFF)
    SPIsend(SPI_SLAVE_ADDR, SPI_INTCONA, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_INTCONB, 0xFF)
    SPIsend(SPI_SLAVE_ADDR, SPI_DEFVALA, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_DEFVALB, 0xFF)
    SPIsend(SPI_SLAVE_ADDR, SPI_GPIOA, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_GPIOB, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_INTENA, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_INTENB, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_OLATA, 0x00)
    SPIsend(SPI_SLAVE_ADDR, SPI_OLATB, 0xFF)
Chip = 0
    Bank = "A"
    Bin = "01010101"
Addr = "0x" + str((Chip * 2) + 40)
    if Chip == 5:
        Addr = "0x4A"
    elif Chip == 6:
        Addr = "0x4C"
    elif Chip == 7:
        Addr = "0x4E"
if Bank.upper() == "A":
        Bank = 0x12
    elif Bank.upper() == "B":
        Bank = 0x13
    else:
        Bank = 0x00
Bin = "0b" + Bin
SPIsend(int(Addr, 16), Bank, int(Bin, 2))
```

See Also

- LeScratch User Guide
- WiringLMK
- LMK.GPIO

Appendix: 40 Pins GPIO Mapping Table for Banana Pro and LeMaker Guitar

LEMAKER HOMEDOWNLOADFORUMBLOGPRODUCT

Physical Numbering Scheme	Banana Pro		(Base Board Rev.B)		WiringLMK Setup Functions			LMK.GPIO	Raspberry Pi
wiringPiSetupPhys() LMK.GPIO.BOARD	A20 Chip Serial Number	sys	S500 Chip Serial Number	sys	wiringPiSetup()	wiringPiSetupGpio()	wiringPiSetupSys()	LMK.GPIO.BCM	BCM Chip Serial Number
1	3.3V	--	3.3V	--	--	--	--	--	3.3V
2	5V	--	5V	--	--	--	--	--	5V
3	53/PB21	2	131/GPIOE03	131	8	2	2	2	2/GPIO02
4	5V	--	5V	--	--	--	--	--	5V
5	52/PB20	3	130/GPIOE02	130	9	3	3	3	3GPIO03
6	GND	--	GND	--	--	--	--	--	GND
7	226/PH02	4	50/GPIOB18	50	7	4	4	4	4/GPIO04
8	228/PH04	14	91/GPIOC27	91	15	14	14	14	14/GPIO14
9	GND	--	GND	--	--	--	--	--	GND
10	229/PH05	15	90/GPIOC26	90	16	15	15	15	15/GPIO15
11	275/PI19	17	64/GPIOC00	64	0	17	17	17	17/GPIO17
12	259/PI03	18	40/GPIOB08	40	1	18	18	18	18/GPIO18
13	274/PI18	27	65/GPIOC01	65	2	27	27	27	27/GPIO27
14	GND	--	GND	--	--	--	--	--	GND
15	273/PI17	22	68/GPIOC04	68	3	22	22	22	22/GPIO22
16	244/PH20	23	25/GPIOA25	25	4	23	23	23	23/GPIO23
17	3.3V	--	3.3V	--	--	--	--	--	3.3V
18	245/PH21	24	70/GPIOC06	70	5	24	24	24	24/GPIO24
19	268/PI12	10	89/GPIOC25	89	12	10	10	10	10/GPIO10
20	GND	--	GND	--	--	--	--	--	GND
21	269/PI13	9	88/GPIOC24	88	13	9	9	9	9/GPIO09
22	272/PI16	25	69/GPIOC05	69	6	25	25	25	25/GPIO25
23	267/PI11	11	86/GPIOC22	86	14	11	11	11	11/GPIO11
24	266/PI10	8	87/GPIOC23	87	10	8	8	8	8/GPIO8
25	GND	--	GND	--	--	--	--	--	GND
26	270/PI14	7	51/GPIOB19	51	11	7	7	7	7/GPIO07
27	257/PI01	0	48/GPIOB16	48	30	0	0	0	0/GPIO00
28	256/PI00	1	46/GPIOB14	46	31	1	1	1	1/GPIO01
29	35/PB03	5	47/GPIOB15	47	21	5	5	5	5/GPIO05
30	GND	--	GND	--	--	--	--	--	GND
31	277/PI21	6	42/GPIOB10	42	22	6	6	6	6/GPIO06
32	276/PI20	12	45/GPIOB13	45	26	12	12	12	12/GPIO12
33	45/PB13	13	32/GPIOB00	32	23	13	13	13	13/GPIO13
34	GND	--	GND	--	--	--	--	--	GND
35	39/PB07	19	33/GPIOB01	33	24	19	19	19	19/GPIO19
36	38/PB06	16	28/GPIOA28	28	27	16	16	16	16/GPIO16
37	37/PB05	26	34/GPIOB02	34	25	26	26	26	26/GPIO26
38	44/PB12	20	31/GPIOA31	31	28	20	20	20	20/GPIO20
39	GND	--	GND	--	--	--	--	--	GND
40	40/PB08	21	27/GPIOA27	27	29	21	21	21	21/GPIO21