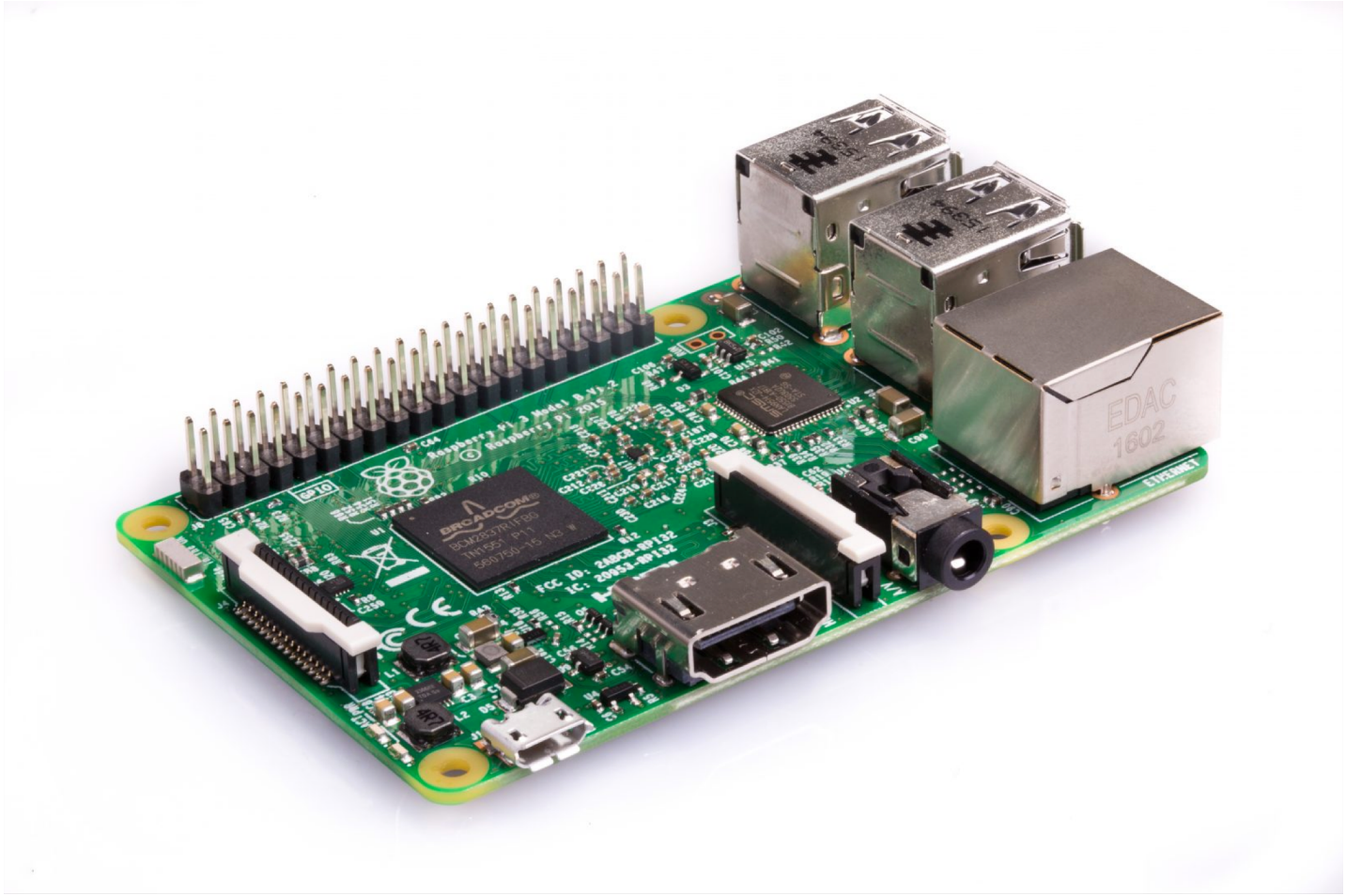
Distributed Real-Time Control Systems

Module 15

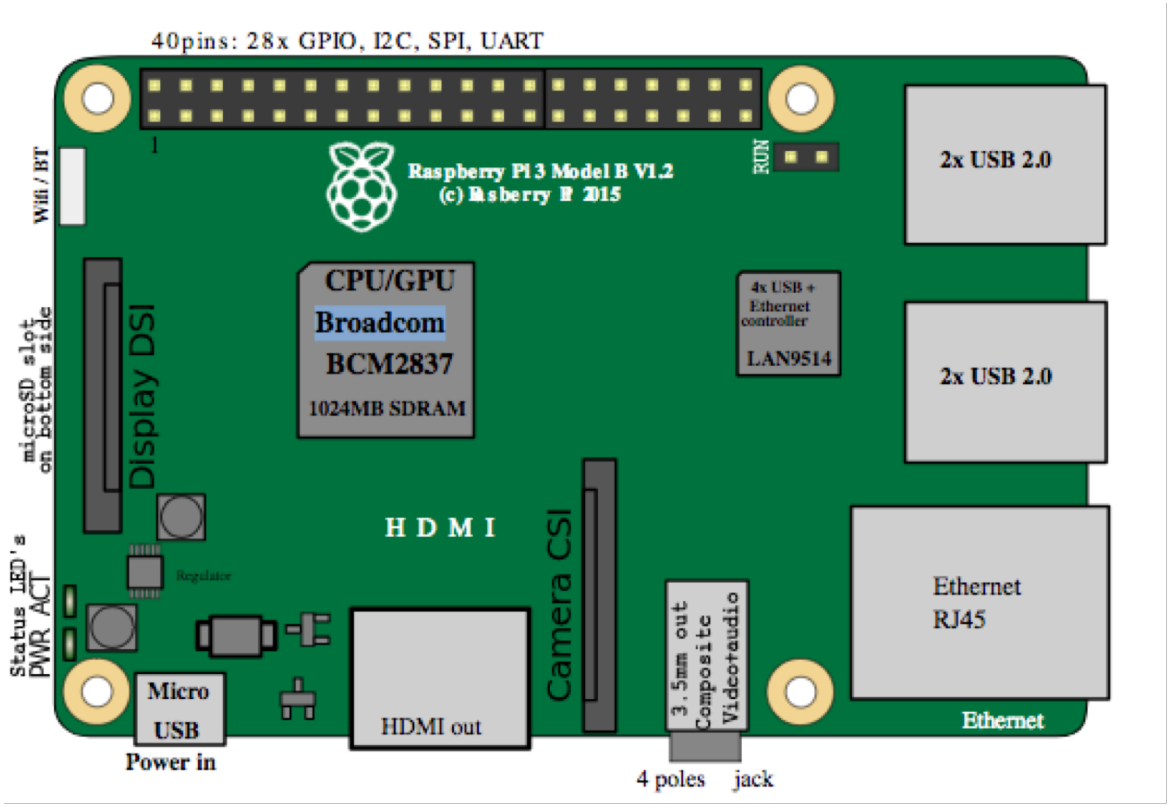
The RPI and GPIO

RPI 3



Specifications

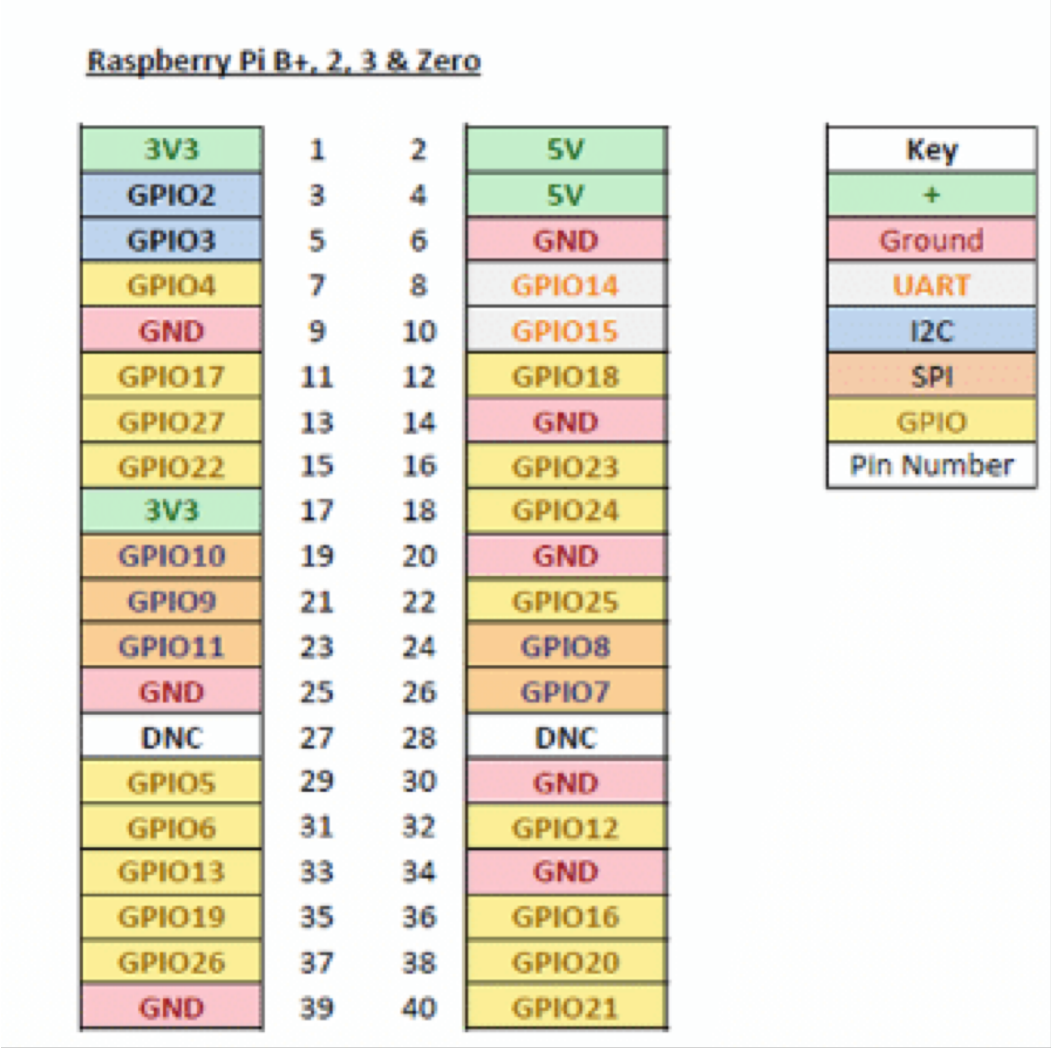
- ARM Cortex – A53
- 1.2GHz, 64-bit
- Quad-core CPU + GPU
- 1 Gb RAM
- Ethernet
- WIFI
- Bluetooth
- uSDHC
- HDMI
- 4x USB 2
- 40-Pin GPIO
- CSI camera port
- DSI Display
- Stereo
- Composite Video
- Power Source up to 2.5A



RPI Peripherals

BCM2835 ARM Peripherals

- Timers
- Interrupt Controller
- GPIO
- USB
- PCM / I2S (audio)
- DMA controller
- I2C master
- I2C / SPI Slave
- SPI0 / SPI1 / SPI2
- PWM
- UART



GPIO Modes

- GPIO Pins can be set in the following modes:

PI_INPUT
PI_OUTPUT
PI_ALT0
PI_ALT1
PI_ALT2
PI_ALT3
PI_ALT4
PI_ALT5

- Each pin has a internal pull-up or pull-down.

BROADCOM. BCM2835 ARM Peripherals

6.2 Alternative Function Assignments

Every GPIO pin can carry an alternate function. Up to 6 alternate function are available but not every pin has that many alternate functions. The table below gives a quick over view.

	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
GPIO0	High	SDA0	SA5	<reserved>			
GPIO1	High	SCL0	SA4	<reserved>			
GPIO2	High	SDA1	SA3	<reserved>			
GPIO3	High	SCL1	SA2	<reserved>			
GPIO4	High	GPCLK0	SA1	<reserved>			ARM_TDI
GPIO5	High	GPCLK1	SA0	<reserved>			ARM_TDO
GPIO6	High	GPCLK2	SOE_N / SE	<reserved>			ARM_RTCK
GPIO7	High	SPI0_CE1_N	SWE_N / SRW_N	<reserved>			
GPIO8	High	SPI0_CE0_N	SD0	<reserved>			
GPIO9	Low	SPI0_MISO	SD1	<reserved>			
GPIO10	Low	SPI0_MOSI	SD2	<reserved>			
GPIO11	Low	SPI0_SCLK	SD3	<reserved>			
GPIO12	Low	PWM0	SD4	<reserved>			ARM_TMS
GPIO13	Low	PWM1	SD5	<reserved>			ARM_TCK
GPIO14	Low	TXD0	SD6	<reserved>			TXD1
GPIO15	Low	RXD0	SD7	<reserved>			RXD1
GPIO16	Low	<reserved>	SD8	<reserved>	CTS0	SPI1_CE2_N	CTS1
GPIO17	Low	<reserved>	SD9	<reserved>	RTS0	SPI1_CE1_N	RTS1
GPIO18	Low	PCM_CLK	SD10	<reserved>	BSCSL SDA / MOSI	SPI1_CE0_N	PWM0
GPIO19	Low	PCM_FS	SD11	<reserved>	BSCSL SCL / SCLK	SPI1_MISO	PWM1
GPIO20	Low	PCM_DIN	SD12	<reserved>	BSCSL / MISO	SPI1_MOSI	GPCLK0
GPIO21	Low	PCM_DOUT	SD13	<reserved>	BSCSL / CE_N	SPI1_SCLK	GPCLK1
GPIO22	Low	<reserved>	SD14	<reserved>	SD1_CLK	ARM_TRST	
GPIO23	Low	<reserved>	SD15	<reserved>	SD1_CMD	ARM_RTCK	
GPIO24	Low	<reserved>	SD16	<reserved>	SD1_DAT0	ARM_TDO	
GPIO25	Low	<reserved>	SD17	<reserved>	SD1_DAT1	ARM_TCK	
GPIO26	Low	<reserved>	<reserved>	<reserved>	SD1_DAT2	ARM_TDI	
GPIO27	Low	<reserved>	<reserved>	<reserved>	SD1_DAT3	ARM_TMS	
GPIO28	-	SDA0	SA5	PCM_CLK	<reserved>		
GPIO29	-	SCL0	SA4	PCM_FS	<reserved>		
GPIO30	Low	<reserved>	SA3	PCM_DIN	CTS0		CTS1
GPIO31	Low	<reserved>	SA2	PCM_DOUT	RTS0		RTS1
GPIO32	Low	GPCLK0	SA1	<reserved>	TXD0		TXD1
GPIO33	Low	<reserved>	SA0	<reserved>	RXD0		RXD1
GPIO34	High	GPCLK0	SOE_N / SE	<reserved>	<reserved>		
GPIO35	High	SPI0_CE1_N	SWE_N / SRW_N		<reserved>		
GPIO36	High	SPI0_CE0_N	SD0	TXD0	<reserved>		
GPIO37	Low	SPI0_MISO	SD1	RXD0	<reserved>		
GPIO38	Low	SPI0_MOSI	SD2	RTS0	<reserved>		
GPIO39	Low	SPI0_SCLK	SD3	CTS0	<reserved>		
GPIO40	Low	PWM0	SD4		<reserved>	SPI2_MISO	TXD1
	Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5

GPIO Diagram

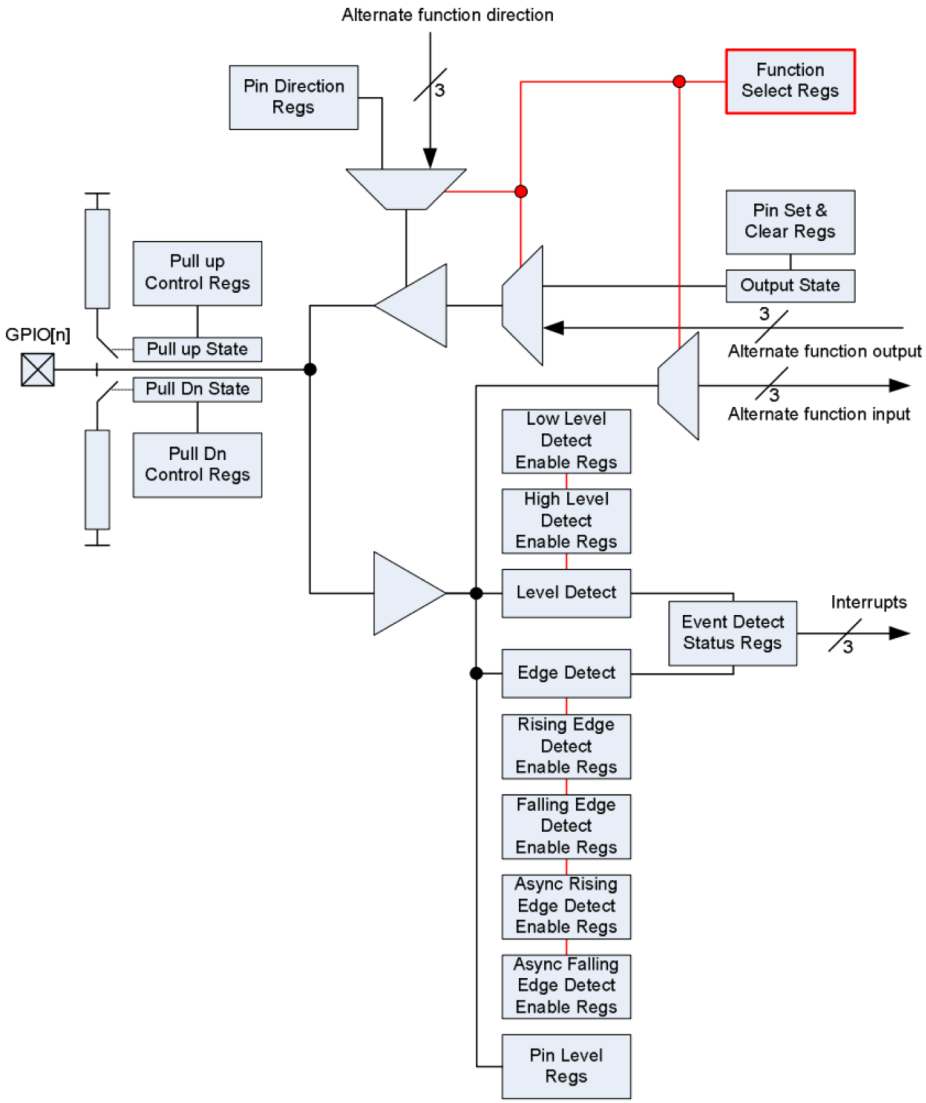
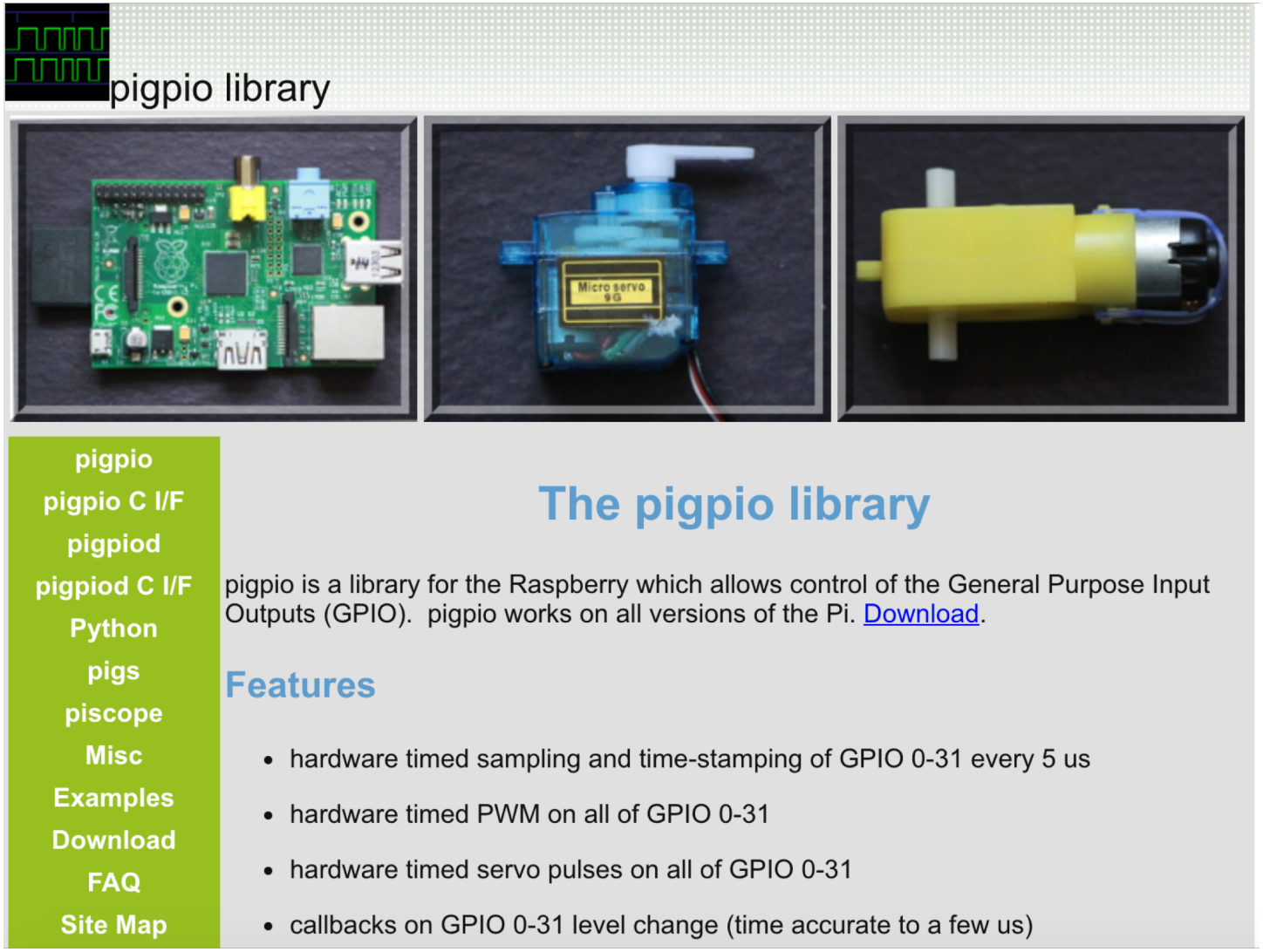


Figure 6-1 GPIO Block Diagram

Programming GPIO

- pigpio C library: http://abyz.me.uk/rpi/pigpio/



Installing PIGPIO on RASPBIAN

http://abyz.me.uk/rpi/pigpio/download.html

EASY WAY – OLD VERSION:

```
sudo apt-get update
sudo apt-get install pigpio
```

NOT SO EASY WAY – NEW VERSIONS:

```
rm pigpio.zip
sudo rm -rf PIGPIO
wget abyz.me.uk/rpi/pigpio/pigpio.zip
unzip pigpio.zip
cd PIGPIO
make
sudo make install
```

Programming GPIO

Basic Functions

```
gpioInitialize          //returns negative if fail
gpioTerminate
gpioSetMode             //input/output
gpioSetPullUpDown
gpioRead
gpioWrite
gpioPWM
gpioGetPWMdutycycle
gpioServo
gpioGetServoPulsewidth
gpioDelay.              //microseconds
gpioSetAlertFunc        //level change callback
gpioSetTimerFunc        //regular timed callback
```

Other functions for:

I2C Master
I2C Slave
SPI
Serial
Interrupts
Waveform Generation
Bulk read/writes
Bit banging serial reads, I2C, SPI

Programming GPIO

hello_pi.cpp

```
#include <iostream>
#include <pigpio.h>
using namespace std;

int main (int argc, char *argv[])
{
   if ( gpioInitialise() < 0 )
      cout << “Error Initialising pigpio” << endl;
      return 1;  //failed
   }
   auto hw = gpioHardwareRevision();
   auto sw = gpioVersion();
   cout << “Hello from  pigpio V” << sw << “on a PI rev. ”  << hw << endl;
   gpioTerminate();
}
```

COMPILE WITH:
g++ -std=c++11 –lpigpio –lpthread –lrt –o hello_pi hello_pi.cpp

RUN WITH:
sudo ./hello_pi

A more interesting example

pigpio sample: LDR.c

Measure how long a capacitor takes to charge through a LDR.

```
#include <iostream>
#include <pigpio.h>
using namespace std;
#define LDR 18
//tick (call time) is passed to alert by the callback
void alert(int pin, int level, uint32_t tick) {
  static uint32_t inited = 0, lastTick, firstTick;
  if (inited) {
    diffTick = tick - lastTick;
    lastTick = tick;
    if (level == 1)
      cout << tick-firstTick << " " << diffTick;
  }
  else {
    inited = 1;
    firstTick = tick;
    lastTick = firstTick;
  }
}
```

```
int main (int argc, char *argv[])
{
  if (gpioInitialise()<0)
    return 1;
  // call alert when LDR changes state
  gpioSetAlertFunc(LDR, alert);
  while (1) //loop {
    gpioSetMode(LDR, PI_OUTPUT); // drain cap
    gpioWrite(LDR, PI_OFF);
    gpioDelay(200); // micros
    gpioSetMode(LDR, PI_INPUT); // recharge cap
    gpioDelay(10000); //micros
  }
  gpioTerminate();
}
```

I2C Peripherals

I2C Master

BROADCOM. BCM2835 ARM Peripherals

3 BSC

3.1 Introduction

The Broadcom Serial Controller (BSC) controller is a master, fast-mode (400Kb/s) BSC controller. The Broadcom Serial Control bus is a proprietary bus compliant with the Philips® I2C bus/interface version 2.1 January 2000.

- I^2C single master only operation (supports clock stretching wait states)
- Both 7-bit and 10-bit addressing is supported.
  - Timing completely software controllable via registers

IMPORTANT NOTE: Single Master Only Operation! Does not support multi master.

I2C Slave

BROADCOM. BCM2835 ARM Peripherals

11 SPI/BSC SLAVE

11.1 Introduction

The BSC interface can be used as either a Broadcom Serial Controller (BSC) or a Serial Peripheral Interface (SPI) controller. The BSC bus is a proprietary bus compliant with the Philips® I2C bus/interface version 2.1 January 2000. Both BSC and SPI controllers work in the slave mode. The BSC slave controller has specially built in the Host Control and Software Registers for a Chip booting. The BCS controller supports fast-mode (400Kb/s) and it is compliant to the I^2C bus specification version 2.1 January 2000 with the restrictions:

- I^2C slave only operation
- clock stretching is not supported
- 7-bit addressing only
-

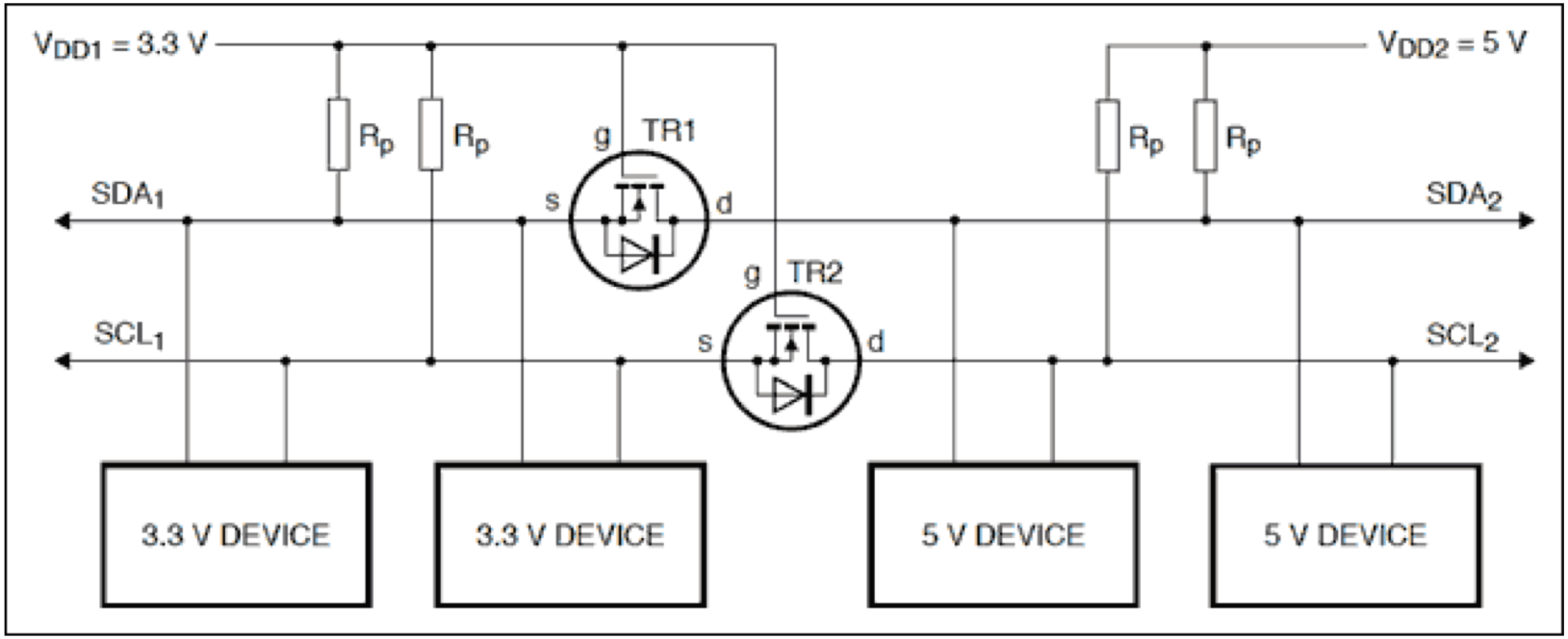
There is only one BSC/SPI slave. The registers base addresses is 0x7E21_4000.

IMPORTANT NOTE: Slave Only Operation!

I2C BUS Connection (READ/WRITE)

- RPI uses 3.3V GPIO. ARDUINO uses 5V GPIO. It is **dangerous** to connect directly IO ports on the RPI to the ARDUINO.

- If the RPI is to write on the I2C BUS, then level shifters should be used.



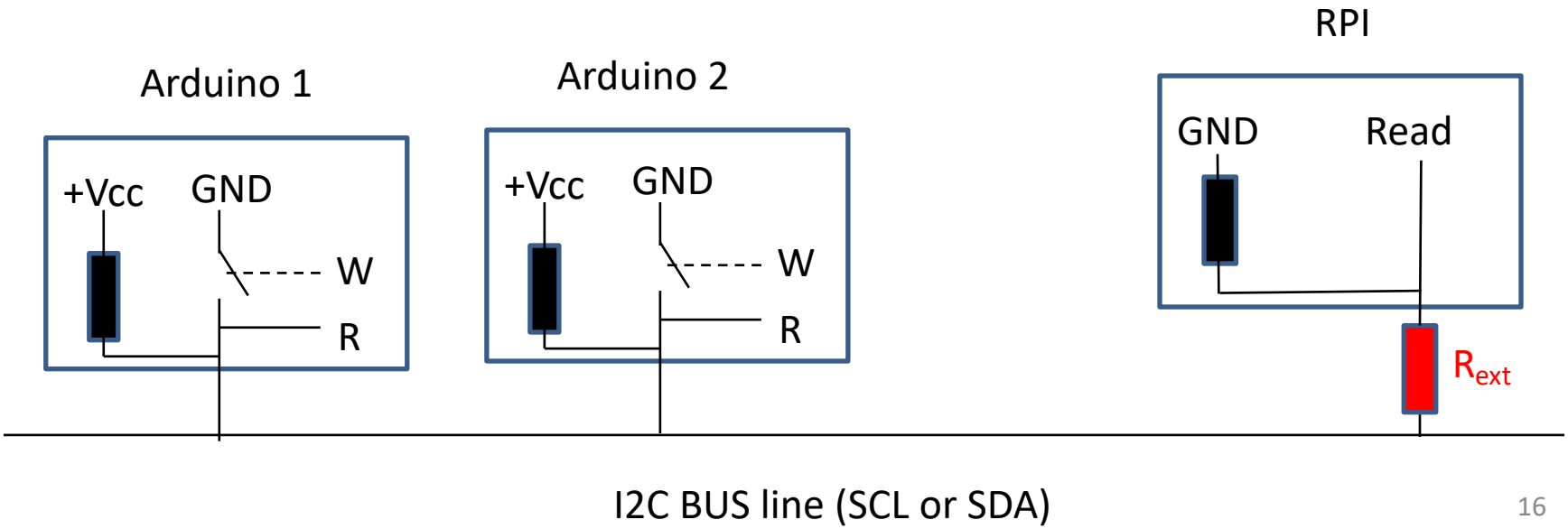
I2C BUS Connection (READ ONLY)

- If the RPI is to read only on the BUS, a low cost solution can be implemented.
- Use external resistors so that the RPI gets 3.3V HIGH and 0V LOW.

- **Important:** Take into account the internal pull-ups and pull-downs.

- **Note:** On the RPI, I2CSL pins have internal pull-downs enabled by default (20k - 50k Ohm).

- What should be the value of R_{ext} below ?



NOTE

- Different Hardware Versions may have different values of internal pull-ups and pull-downs.
- Use the multimeter to measure the voltage levels in your line and infer the values of the internal pull up/downs.
- If needed, request in the lab resistors to ensure proper voltage levels in the BUS lines.

Examples

```
#include <stdio.h>
#include <pigpio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#include <memory.h>
#define DESTINATION_ADDR 0x48
int main(int argc, char *argv[]) {
  int key  = 0;
  int handle;
  int length = 12; //11 chars + \0
  char message[] = "Hello World";
  if (gpioInitialise() < 0) {printf("Erro 1\n"); return 1;}
  handle = i2cOpen(1, DESTINATION_ADDR, 0); /* Initialize Master*/
  while(key != 'q')  {
    i2cWriteDevice(handle, message, length); /* Master Transmit */
    gpioDelay(20000);
    printf("Press q to quit. Any other ckey to continue.\n");
    key = getchar();
  }
  i2cClose(handle); /* close master */
  gpioTerminate();
}
```

Master

Slave

```
#include <stdio.h>
#include <pigpio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/i2c-dev.h>
#define SLAVE_ADDR 0x48
int main(int argc, char *argv[]) {
  int status, j, key  = 0;
  if (gpioInitialise() < 0) {
    printf("Erro 1\n"); return 1;}
  bsc_xfer_t xfer;
  status = init_slave(xfer, SLAVE_ADDR);
  while(key != 'q') {
    xfer.txCnt = 0;
    status = bscXfer(&xfer);
    printf("Received %d bytes\n", xfer.rxCnt);
    for(j=0;j<xfer.rxCnt;j++)
      printf("%c",xfer.rxBuf[j]);
    printf("\n Press q to quit.\n");
    key = getchar();
  }
  status = close_slave(xfer);
  gpioTerminate();
}
```

```
int init_slave(bsc_xfer_t &xfer, int addr) {
  gpioSetMode(18, PI_ALT3);
  gpioSetMode(19, PI_ALT3);
  xfer.control = (addr<<16) | /* Slave address */
  (0x00<<13) | /* invert transmit status flags */
  (0x00<<12) | /* enable host control */
  (0x00<<11) | /* enable test fifo */
  (0x00<<10) | /* invert receive status flags */
  (0x01<<9) | /* enable receive */
  (0x01<<8) | /* enable transmit */
  (0x00<<7) | /* abort  and clear FIFOs */
  (0x00<<6) | /* send control reg as 1st I2C byte */
  (0x00<<5) | /* send status regr as 1st I2C byte */
  (0x00<<4) | /* set SPI polarity high */
  (0x00<<3) | /* set SPI phase high */
  (0x01<<2) | /* enable I2C mode */
  (0x00<<1) | /* enable SPI mode */
   0x01 ;     /* enable BSC peripheral */
   return bscXfer(&xfer);
}
int close_slave(bsc_xfer_t &xfer) {
  xfer.control = 0;
  return bscXfer(&xfer);
}
```

See the documentation of function **bscXfer** for I2S slave operation:

int bscXfer(bsc_xfer_t *bsc_xfer)

Note

The previous master and slave codes cannot run in the same machine (gpio not reentrant).

Homework

Write a program to test the direct connection between the I2C master and slave in a RPI.

Hint: Interleave the codes of the master and the slave in a single program.