Digital Design & Computer Architecture

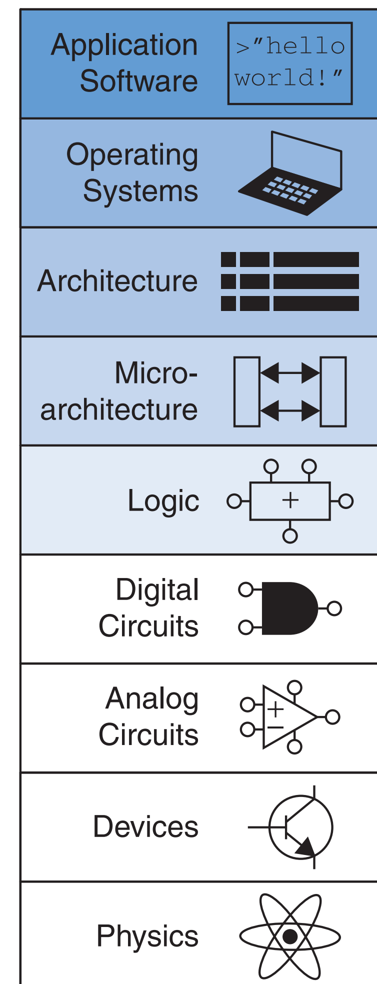
Sarah Harris & David Harris

with Joshua Brake

Chapter 9: Embedded Systems

Chapter 9 :: Topics

- Microcontrollers
- RISC-V Microcontrollers
- Memory-Mapped I/O
- General-Purpose I/O
- Device Drivers
- Delays & Timers
- Example: Morse Code
- Interfacing
- Serial Peripheral Interface
- Example: SPI Accelerometer



ELSEVIER

Microcontrollers

- **Microprocessors** are **computers on a chip**
- **Microcontrollers** are microprocessors with **flexible input/output (I/O) peripherals**
- **Peripheral examples:**
  - General-purpose I/O (GPIO): turn pins ON and OFF
  - Serial ports
  - Timers
  - Analog/Digital and Digital/Analog Converters (ADC/DAC)
  - Pulse Width Modulation (PWM)
  - Universal Serial Bus
  - Ethernet

ELSEVIER

Embedded Systems

- Microcontrollers are commonly used in embedded systems
- An embedded system is a system whose user may be unaware there is a computer inside
- **Examples:**
  - Microwave oven
  - Clock/radio
  - Electronic fuel injection
  - Annoying toys with batteries for toddlers
  - Implantable glucose monitor

ELSEVIER

Microcontroller Economics

- In 2019, about **26 billion microcontrollers sold**
  - Average price: **60 cents**
  - About **70 microcontrollers in an average new car**
- **Biggest markets:**
  - Automotive (~70 microcontrollers in an average new car)
  - Consumer electronics, industrial, medical, military
- **Highly cost-sensitive market**
  - Often < $0.40; pennies matter
  - Cheaper than using a cable or a pushrod in a system
  - A microcontroller integrated on a larger chip can have a manufacturing cost of < $0.001
- **Memory tends to dominate the cost**
  - Select one with no more memory than necessary
- **Classified as 8, 16, 32-bit based on internal bus**
  - 1970s vintage 8-bit processors can still be adequate
  - 16-bit have greatest revenue in 2019
  - 32-bit processors are faster and more flexible but use more code memory

ELSEVIER

Microprocessor Architectures

- The architecture is the native machine language a microprocessor understands.
- **Commercial Examples:**
  - RISC-V
  - ARM
  - PIC
  - 8051

ELSEVIER

Embedded Systems

RISC-V

Microcontrollers

RISC-V Microcontrollers

- RISC-V is an **open-standard** architecture
  - Developed at Berkeley starting in 2010
  - No licensing fees
  - Increasingly popular, especially for system-on-chip
- **SiFive Freedom E310-G002**
  - Second generation RISC-V microcontroller (2019)
  - Found on several low-cost boards

ELSEVIER

SiFive Freedom E310 Microprocessor

- E31 microprocessor with **5-stage pipeline**
  - Similar to the one we will discuss in class
  - RV32IMAC architecture
    - Baseline **32**-bit **R**ISC-**V** **I**nteger instructions
    - Plus **M**ultiply/Divide, **A**tomic Memory, **C**ompressed operations
  - 2.73 Coremark/MHz
- **Electrical Specs**
  - Up to 320 MHz
  - Built in antiquated 180 nm process
  - 1.8V core and 3.3V I/O

ELSEVIER

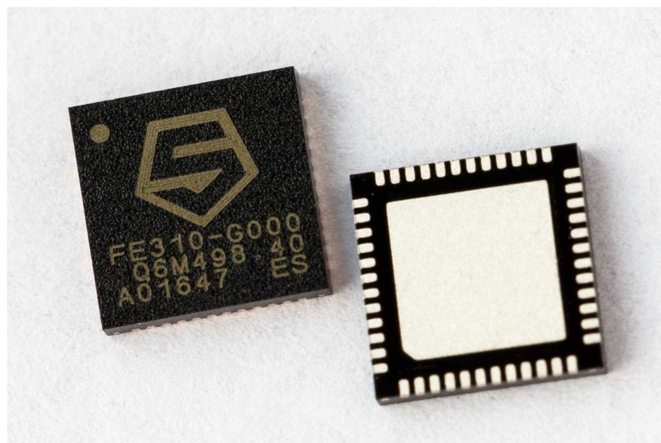
FE310 Memory & I/O

- **Onboard Memory**
  - 16 KB Data SRAM
  - 16 KB Instruction Cache
  - 8 KB Mask ROM & 8 KB OTP (one-time-programmable) Program Boot Memory
  - Most instructions located on external Flash
- **I/O Peripherals**
  - 19 GPIO Pins
  - Serial Ports: SPI x2, I2C, UART x2
  - PWM x3
  - Timer
  - JTAG Debugger Interface

ELSEVIER

FE310 Pinout

- **48-pin QFN** (quad flat pack, no leads)
  - Hard to hand solder
- **Pins:**
  - 12 power (GND pad on back)
  - 19 GPIO
  - 4 JTAG programming
  - 6 QSPI Flash code link
  - 2 clock crystals
  - 6 control



designnews.com

SiFive
FE310-G002
6x6 48-lead QFN
(0.40mm Pad Pitch)

Pin	Name
1	QSPI_DQ_3
2	QSPI_DQ_2
3	QSPI_DQ_2
4	QSPI_DQ_1
5	QSPI_CS
6	VDD
7	PLL_AVDD
8	PLL_AVSS
9	XTAL_XI
10	XTAL_XO
11	IVDD
12	OTP_AIVDD
13	JTAG_TCK
14	JTAG_TDO
15	JTAG_TMS
16	JTAG_TDI
17	AON_PMU_OUT_1
18	AON_PMU_DWAKE
19	AON_IVDD
20	AON_PSD_LFALTCI
21	AON_PSD_LFCLKS
22	AON_PMU_OUT_0
23	AON_IVDD
24	AON_ERST_N
25	GPIO_0 PWM0_0
26	GPIO_1 PWM0_1
27	GPIO_2 SPI1_CS0 PWM0_2
28	GPIO_3 SPI1_MOSI PWM0_3
29	GPIO_4 SPI1_MISO
30	VDD
31	GPIO_5 SPI1_SCK
32	IVDD
33	GPIO_9 SPI1_CS2
34	GPIO_10 SPI1_CS3 PWM2_0
35	GPIO_11 PWM2_1
36	GPIO_12 PWM2_2 I2C_SDA
37	GPIO_13 PWM2_3 I2C_SCL
38	GPIO_16 UART0_RX
39	GPIO_17 UART0_TX
40	GPIO_18 UART1_TX
41	GPIO_19 PWM1_1
42	GPIO_20 PWM1_0
43	GPIO_21 PWM1_2
44	GPIO_22 PWM1_3
45	GPIO_23 UART1_RX
46	VDD
47	IVDD
48	QSPI_SCK

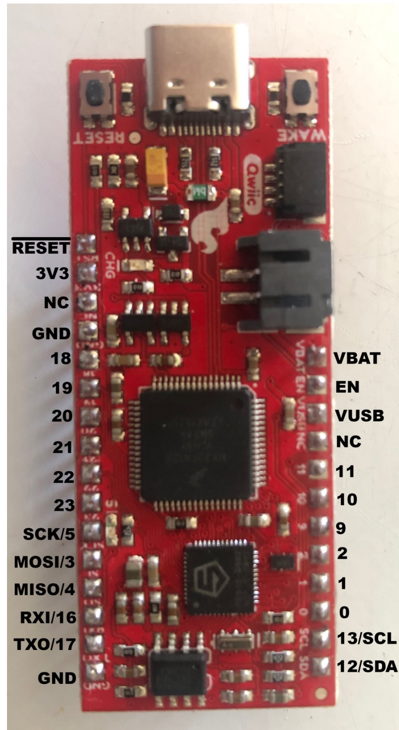
[49. (Exposed Paddle) GND

FE310 Manual

ELSEVIER

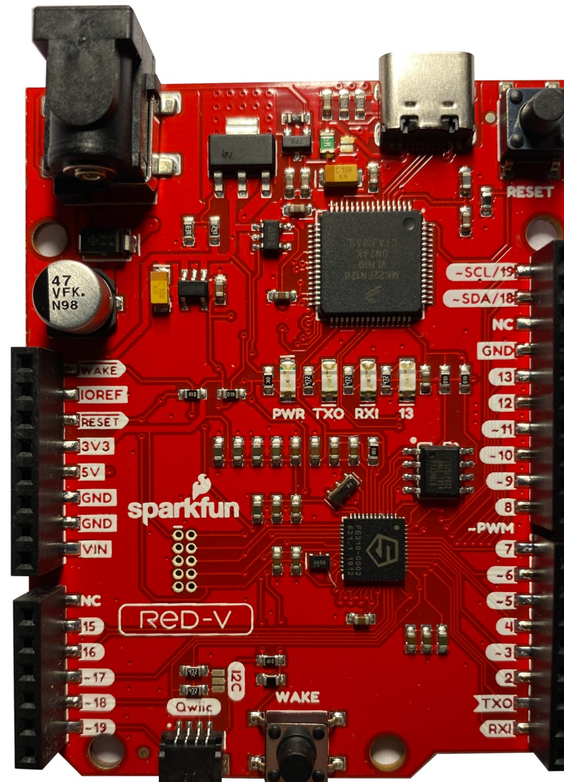
FE310 Boards

SparkFun RED-V Thing Plus



$30
Fits breadboard (2.3x0.9")
Requires soldering
header pins

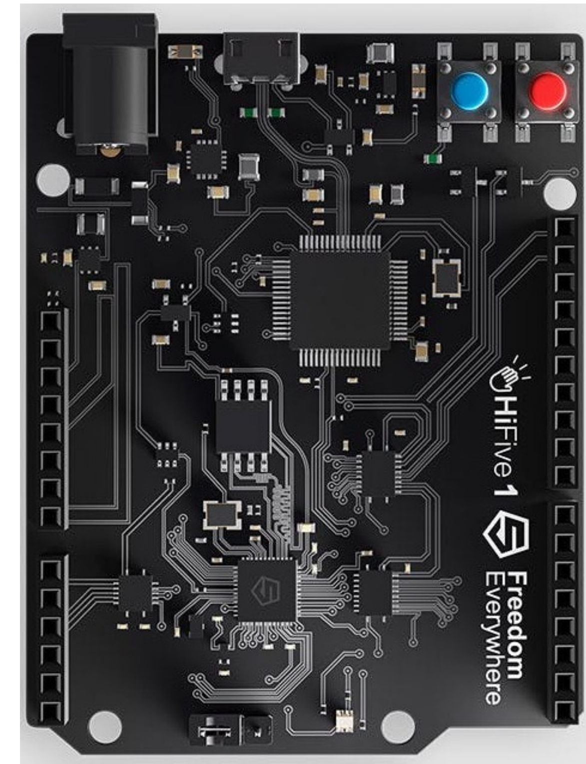
SparkFun RED-V RedBoard



sparkfun.com

$40
Headers soldered
Nonstandard pin labels

HiFive 1



sifive.com

$60
WiFi & Bluetooth

ELSEVIER

Comparison of FE310 Boards

- **Boards are functionally similar**
  - Can operate off USB connection
    - Power, programming, debugging
  - External power connector
  - All I/Os tapped out to pins
  - Software compatible
- This course uses **RED-V Thing Plus** for labs
  - Lowest cost
  - Breadboard-compatible

ELSEVIER

FE310 Datasheets

- **FE310-G002 Datasheet**
  - Pinout
  - Electrical specifications
  - Information for board designer
- **FE310-G002 Manual**
  - Microprocessor core
  - Memory map
  - Peripherals
  - Information for firmware engineer / programmer

ELSEVIER

Embedded Systems

Memory-Mapped I/O

Memory-Mapped I/O

- Control peripherals by **reading or writing memory locations** called ***registers***
  - Not really the same as a bank of flip-flops
- Just like accessing any other variable, but these registers cause **physical things** to happen.
- Portion of the address space is **reserved for I/O** registers rather than program or data.
- In C, use **pointers** to specify addresses to read or write.

ELSEVIER

Example: FE310 Memory Map

- **Boot ROM:**
  - 0x00010000-1FFFF
- **Code Flash:**
  - 0x20000000-3FFFFFFF
- **Data SRAM:**
  - 0x80000000-80003FFF
- **Peripherals:**
  - 0x02000000-0x1FFFFFFF

Base	Top	Attr.	Description	Notes
0x0000_0000	0x0000_0FFF	RWX A	Debug	Debug Address Space
0x0000_1000	0x0000_1FFF	R XC	Mode Select	On-Chip Non Volatile Memory
0x0000_2000	0x0000_2FFF		Reserved	
0x0000_3000	0x0000_3FFF	RWX A	Error Device	
0x0000_4000	0x0000_FFFF		Reserved	
0x0001_0000	0x0001_1FFF	R XC	Mask ROM (8 KiB)	
0x0001_2000	0x0001_FFFF		Reserved	
0x0002_0000	0x0002_1FFF	R XC	OTP Memory Region	
0x0002_2000	0x001F_FFFF		Reserved	
0x0200_0000	0x0200_FFFF	RW A	CLINT	On-Chip Peripherals
0x0201_0000	0x07FF_FFFF		Reserved	
0x0800_0000	0x0800_1FFF	RWX A	E31 ITIM (8 KiB)	
0x0800_2000	0x0BFF_FFFF		Reserved	
0x0C00_0000	0x0FFF_FFFF	RW A	PLIC	
0x1000_0000	0x1000_0FFF	RW A	AON	
0x1000_1000	0x1000_7FFF		Reserved	
0x1000_8000	0x1000_8FFF	RW A	PRCI	
0x1000_9000	0x1000_FFFF		Reserved	
0x1001_0000	0x1001_0FFF	RW A	OTP Control	
0x1001_1000	0x1001_1FFF		Reserved	
0x1001_2000	0x1001_2FFF	RW A	GPIO	
0x1001_3000	0x1001_3FFF	RW A	UART 0	
0x1001_4000	0x1001_4FFF	RW A	QSPI 0	
0x1001_5000	0x1001_5FFF	RW A	PWM 0	
0x1001_6000	0x1001_6FFF	RW A	I2C 0	
0x1001_7000	0x1002_2FFF		Reserved	
0x1002_3000	0x1002_3FFF	RW A	UART 1	
0x1002_4000	0x1002_4FFF	RW A	SPI 1	
0x1002_5000	0x1002_5FFF	RW A	PWM 1	
0x1002_6000	0x1003_3FFF		Reserved	
0x1003_4000	0x1003_4FFF	RW A	SPI 2	
0x1003_5000	0x1003_5FFF	RW A	PWM 2	
0x1003_6000	0x1FFF_FFFF		Reserved	
0x2000_0000	0x3FFF_FFFF	R XC	QSPI 0 Flash (512 MiB)	Off-Chip Non-Volatile Memory
0x4000_0000	0x7FFF_FFFF		Reserved	
0x8000_0000	0x8000_3FFF	RWX A	E31 DTIM (16 KiB)	On-Chip Volatile Memory
0x8000_4000	0xFFFF_FFFF		Reserved	

From FE310 Manual

ELSEVIER

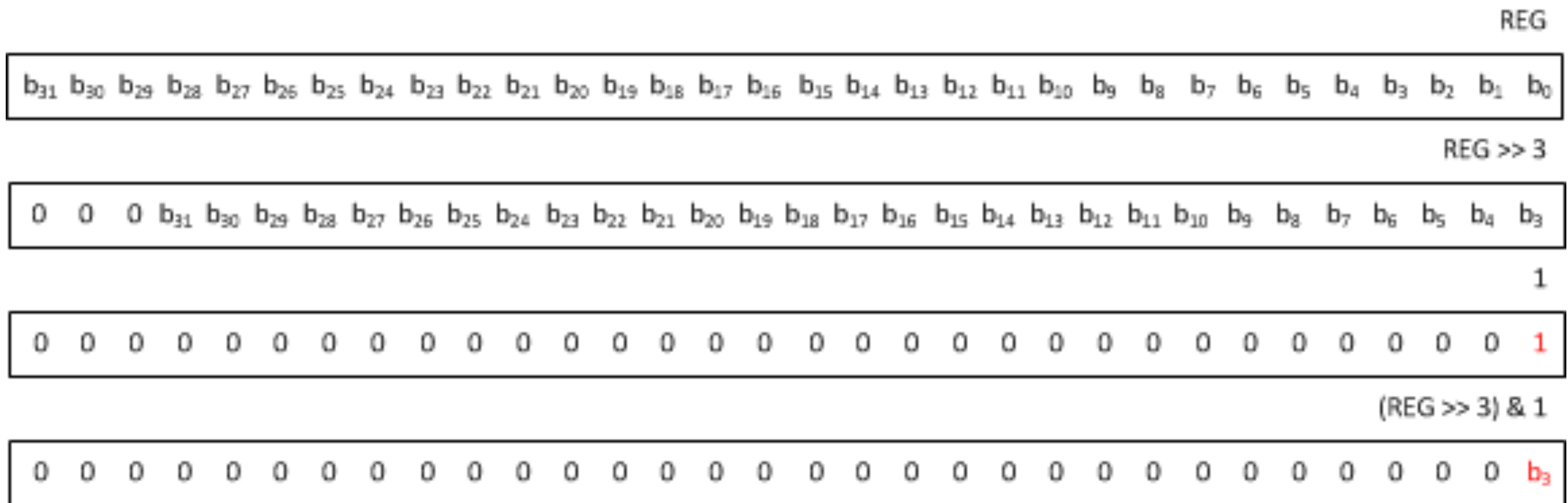
Memory Mapped I/O in C

```
#include <stdint.h>
// Pointers to memory-mapped I/O registers
volatile uint32_t *GPIO_INPUT_VAL  = (uint32_t*)0x10012000;
volatile uint32_t *GPIO_OUTPUT_VAL = (uint32_t*)0x1001200C;

// read all the GPIO inputs
uint32_t allInputs = *GPIO_INPUT_VAL;

// read GPIO bit 19
int gpio19 = (*GPIO_INPUT_VAL >> 19) & 0b1;

// Wait for bit 19 to be 0
while ((*GPIO_INPUT_VAL >> 19) & 0b1);

// Wait for bit 19 to be 1
while (!((*GPIO_INPUT_VAL >> 19) & 0b1));

// Write 1 to GPIO bit 5
*GPIO_OUTPUT_VAL |= (1<<5);

// Write 0 to GPIO bit 5
*GPIO_OUTPUT_VAL &= ~(1<<5);
```


ELSEVIER

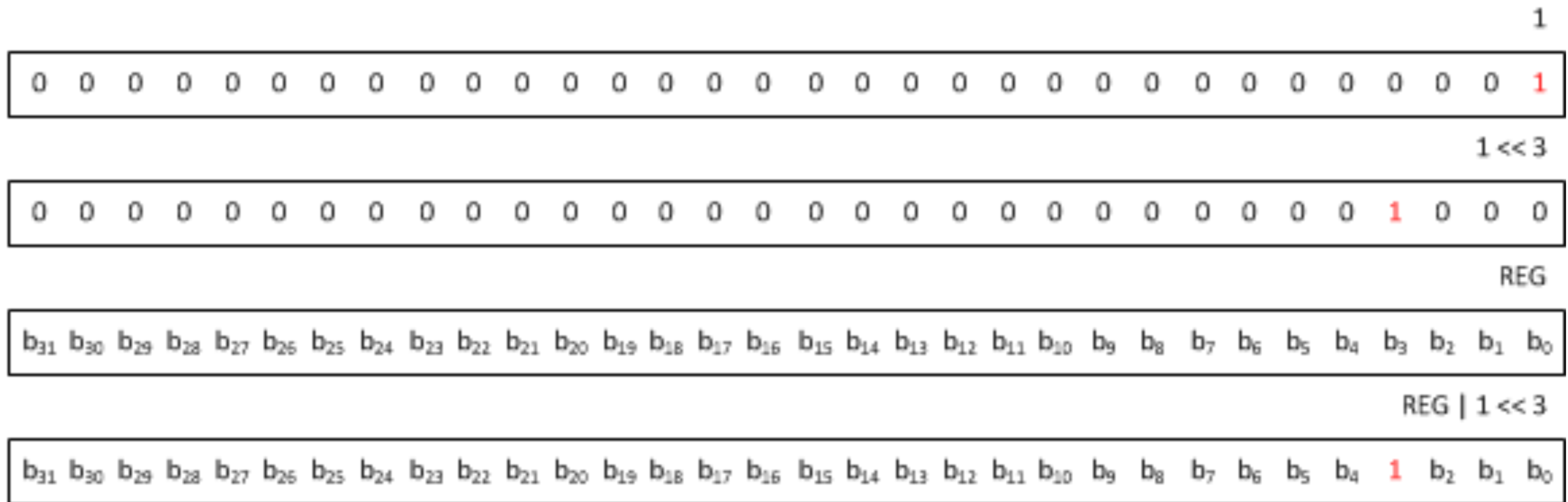
C Idioms: Read value of bit

```
int bit = (*REG >> 3) & 1;    // get value of bit 3
```



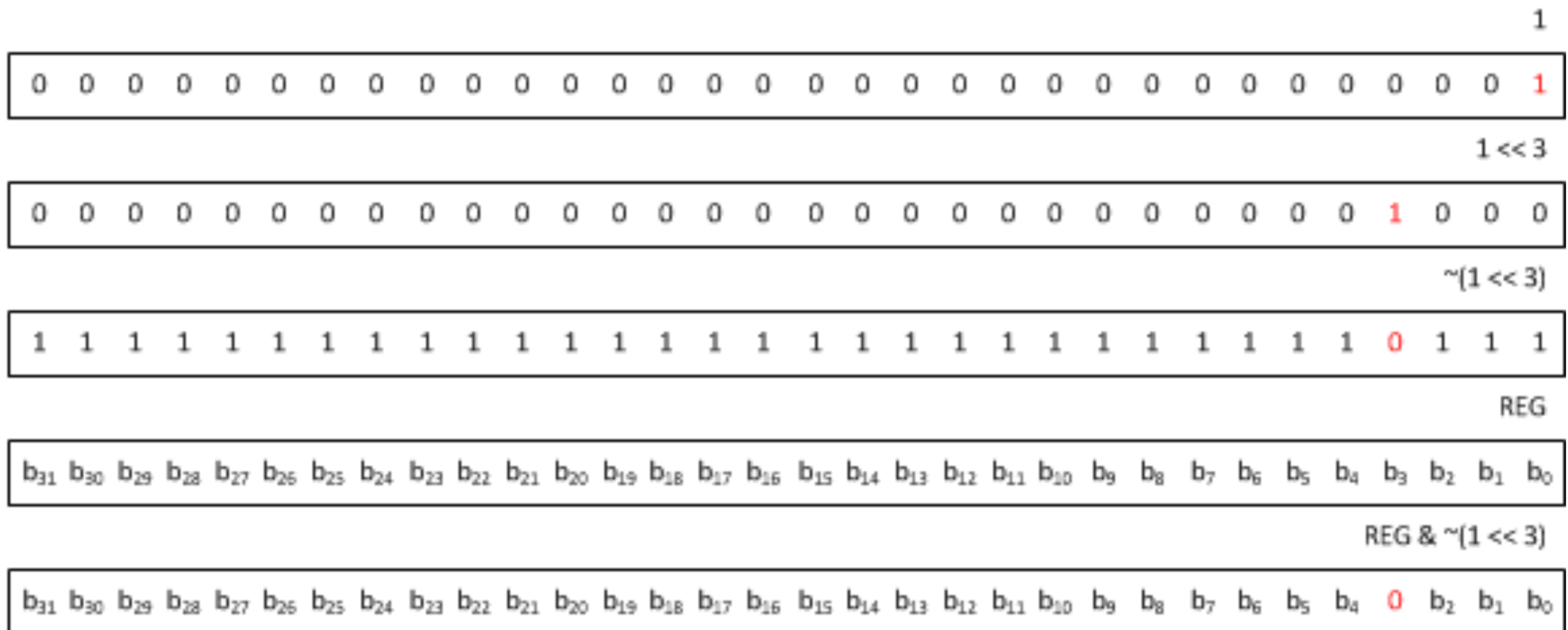
C Idioms: Write Bit to 1

```
*REG |= (1 << 3);                    // turn on bit 3
```



C Idioms: Write Bit to 0

```
*REG &= ~(1 << 3);            // turn off bit 3
```



ELSEVIER

Embedded Systems

General Purpose I/O

(GPIO)

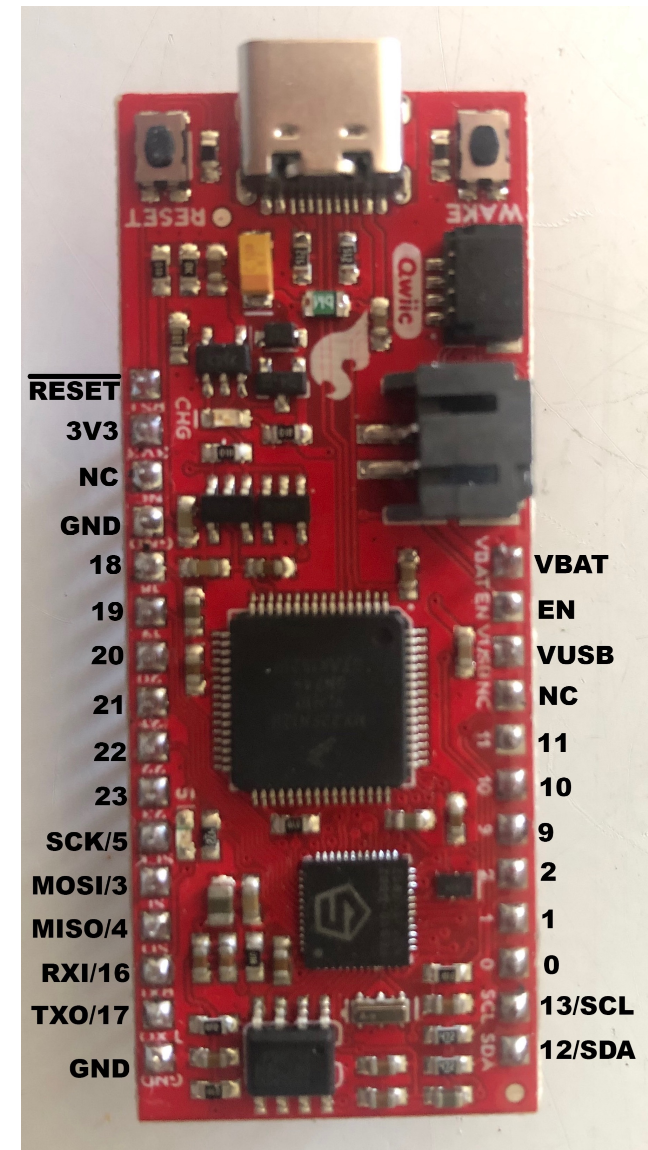
General-Purpose I/O (GPIO)

- GPIOs can be written (driven to 0 or 1) or read.
- **Examples:** LEDs, switches, connections to other digital logic
- **In general:**
  - Look up how many pins your microcontroller has, how they are named.
  - Each pin needs to be configured as an input, output, or special function.
  - Then read or write the pin.

ELSEVIER

FE310 GPIOs

- **19 GPIOs connected to external pins**
  - Not enough pins for all 32 GPIOs
- **Some GPIOs have multiple functions**
- **Blue LED on RED-V board**
  - Connected to GPIO5



ELSEVIER

FE310 GPIO Registers

- Base Address
  - 0x10012000
- 32 bits/reg
  - For 32 GPIOs
- * means enable resets to 0
- Turn on enable to use GPIO

From FE310 Manual

Offset	Name	Description
0x00	input_val	Pin value
0x04	input_en	Pin input enable*
0x08	output_en	Pin output enable*
0x0C	output_val	Output value
0x10	pue	Internal pull-up enable*
0x14	ds	Pin drive strength
0x18	rise_ie	Rise interrupt enable
0x1C	rise_ip	Rise interrupt pending
0x20	fall_ie	Fall interrupt enable
0x24	fall_ip	Fall interrupt pending
0x28	high_ie	High interrupt enable
0x2C	high_ip	High interrupt pending
0x30	low_ie	Low interrupt enable
0x34	low_ip	Low interrupt pending
0x40	out_xor	Output XOR (invert)

```
0x38    iof_en     Enable I/O function
0x3C    iof_sel    Select I/O function
```


ELSEVIER

FE310 GPIO With Enables

```
#include <stdint.h>
volatile uint32_t *GPIO_INPUT_VAL  = (uint32_t*)0x10012000;
volatile uint32_t *GPIO_INPUT_EN   = (uint32_t*)0x10012004;
volatile uint32_t *GPIO_OUTPUT_EN  = (uint32_t*)0x10012008;
volatile uint32_t *GPIO_OUTPUT_VAL = (uint32_t*)0x1001200C;

// Enable input 19 and output 5
*GPIO_INPUT_EN |= (1 << 19);
*GPIO_OUTPUT_EN |= (1 << 5);

// read GPIO bit 19
int gpio19 = (*GPIO_INPUT_VAL >> 19) & 0b1;

// Wait for bit 19 to be 1
while (!((*GPIO_INPUT_VAL >> 19) & 0b1));

// Write 1 to GPIO bit 5
*GPIO_OUTPUT_VAL |= (1<<5);

// Write 0 to GPIO bit 5
*GPIO_OUTPUT_VAL &= ~(1<<5);
```


ELSEVIER

RED-V RedBoard Pin Names

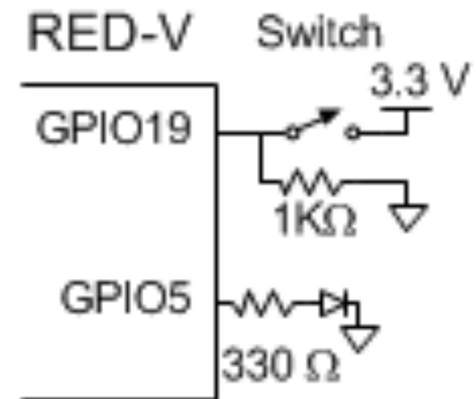
- RED-V RedBoard has Arduino-style pin names
  - Translate these to **GPIO numbers**

Red-V Pin Name	GPIO Number
D0	16
D1	17
D2	18
D3	19
D4	20
D5	21
D6	22
D7	23
D8	0
D9	1
D10	2
D11	3
D12	4
D13	5
D15	9
D16	10
D17	11
D18	12
D19	13

ELSEVIER

Example: Switches & LEDs

- Read a switch and drive an LED
  - (Could have done this with a wire!)




```
#include <stdint.h>
void main(void) {
    volatile uint32_t *GPIO_INPUT_VAL  = (uint32_t*)0x10012000;
    volatile uint32_t *GPIO_INPUT_EN   = (uint32_t*)0x10012004;
    volatile uint32_t *GPIO_OUTPUT_EN  = (uint32_t*)0x10012008;
    volatile uint32_t *GPIO_OUTPUT_VAL = (uint32_t*)0x1001200C;
    int val;

    *GPIO_INPUT_EN  |= (1 << 19);    // Set pin 19 to input
    *GPIO_OUTPUT_EN |= (1 << 5);    // set pin 5 to output
    while (1) {
         val =   (*GPIO_INPUT_VAL >> 19) & 1;    // Read value on pin 19
         if (val) *GPIO_OUTPUT_VAL |=  (1 << 5); // Turn ON pin 5
         else     *GPIO_OUTPUT_VAL &= ~(1 << 5); // Turn OFF pin 5
    }
}
```


ELSEVIER

Other I/O Functions

- Most GPIO pins also have **other special functions**
  - Serial Ports: SPI, I2C, UART
  - Pulse Width Modulation
- Activate with GPIO registers
  - iof_en enables special function
  - iof_sel selects the function

GPIO Number	IOF0	IOF1
0		PWM0_PWM0
1		PWM0_PWM1
2	SPI1_CS0	PWM0_PWM2
3	SPI1_DQ0	PWM0_PWM3
4	SPI1_DQ1	
5	SPI1_SCK	
6	SPI1_DQ2	
7	SPI1_DQ3	
8	SPI1_CS1	
9	SPI1_CS2	
10	SPI1_CS3	PWM2_PWM0
11		PWM2_PWM1
12	I2C0_SDA	PWM2_PWM2
13	I2C0_SCL	PWM2_PWM3
14		
15		
16	UART0_RX	
17	UART0_TX	
18	UART1_TX	
19		PWM1_PWM1
20		PWM1_PWM0
21		PWM1_PWM2
22		PWM1_PWM3
23	UART1_RX	
24		
25		
26	SPI2_CS0	
27	SPI2_DQ0	
28	SPI2_DQ1	
29	SPI2_SCK	
30	SPI2_DQ2	
31	SPI2_DQ3	

From FE310 Manual

Embedded Systems

EasyREDVIO
Device Driver
Library

Device Drivers

- Controlling memory mapped I/O with pointers is **error-prone**
  - Hard to see the big picture
  - Easy to mistype address of GPIO
  - Pointer programming can confuse beginners
- Write **device driver** functions to hide the details
- We will develop **EasyREDVIO** device driver library
  - Arduino-style interface
  - Start with GPIO

ELSEVIER

Memory-Mapped I/O with Structures

- Use **structures** to define memory-mapped I/O
  - Less error-prone than pointers
  - Cleaner C code

ELSEVIER

EasyREDVIO Library

```
// EasyREDVIO.h
// Joshua Brake and David Harris 7 October 2020

#define INPUT 0
#define OUTPUT 1

// Declare a GPIO structure defining the GPIO registers in the order they appear in memory mapped I/O
typedef struct
{
    volatile uint32_t   input_val;      // (GPIO offset 0x00) Pin value
    volatile uint32_t   input_en;       // (GPIO offset 0x04) Pin input enable*
    volatile uint32_t   output_en;      // (GPIO offset 0x08) Pin output enable*
    volatile uint32_t   output_val;     // (GPIO offset 0x0C) Output value
    volatile uint32_t   pue;            // (GPIO offset 0x10) Internal pull-up enable*
    volatile uint32_t   ds;             // (GPIO offset 0x14) Pin drive strength
    volatile uint32_t   rise_ie;        // (GPIO offset 0x18) Rise interrupt enable
    volatile uint32_t   rise_ip;        // (GPIO offset 0x1C) Rise interrupt pending
    volatile uint32_t   fall_ie;        // (GPIO offset 0x20) Fall interrupt enable
    volatile uint32_t   fall_ip;        // (GPIO offset 0x24) Fall interrupt pending
    volatile uint32_t   high_ie;        // (GPIO offset 0x28) High interrupt enable
    volatile uint32_t   high_ip;        // (GPIO offset 0x2C) High interrupt pending
    volatile uint32_t   low_ie;         // (GPIO offset 0x30) Low interrupt enable
    volatile uint32_t   low_ip;         // (GPIO offset 0x34) Low interrupt pending
    volatile uint32_t   iof_en;         // (GPIO offset 0x38) HW-Driven functions enable
    volatile uint32_t   iof_sel;        // (GPIO offset 0x3C) HW-Driven functions selection
    volatile uint32_t   out_xor;        // (GPIO offset 0x40) Output XOR (invert)
} GPIO;
// Define the base address of the GPIO registers and a pointer to this structure
// The 0x…U notation in 0x10012000U indicates an unsigned hexadecimal number
#define GPIO0_BASE   (0x10012000U)
#define GPIO0        ((GPIO*) GPIO0_BASE)
```

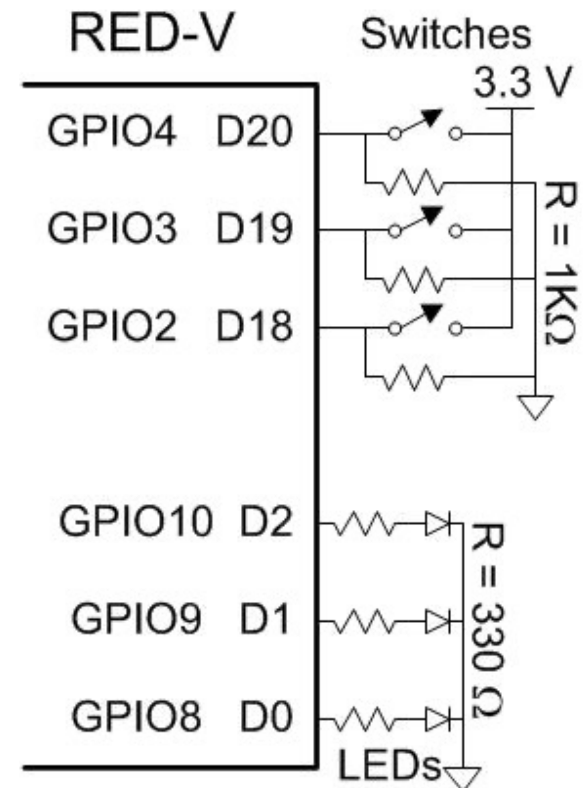
EasyREDVIO Library

```
void pinMode(int gpio_pin, int function) {
    switch(function) {
        case INPUT:
            GPIO0->input_en     |=  (1 << gpio_pin);  // Sets a pin as an input
            GPIO0->output_en    &= ~(1 << gpio_pin);  // Disable output
            GPIO0->iof_en       &= ~(1 << gpio_pin);  // Disable IOF
            break;
        case OUTPUT:
            GPIO0->output_en    |=  (1 << gpio_pin);  // Set pin as an output
            GPIO0->input_en     &= ~(1 << gpio_pin);  // Disable input
            GPIO0->iof_en       &= ~(1 << gpio_pin);  // Disable IOF
            break;
    }
}

void digitalWrite(int gpio_pin, int val) {
    if (val) GPIO0->output_val |= (1 << gpio_pin);
    else     GPIO0->output_val &= ~(1 << gpio_pin);
}

int digitalRead(int gpio_pin) {
    return (GPIO0->input_val >> gpio_pin) & 0x1;
}
```


ELSEVIER

Example: Switches & LEDs

- Reprised with EasyREDVIO
  - Control all three LEDs




```
#include "EasyREDVIO.h"
int main(void) {
  // Set GPIO 4:2 as inputs
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  pinMode(4, INPUT);
  // Set GPIO 10:8 as outputs
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  while (1) { // Read each switch and write corresponding LED
    digitalWrite(8,  digitalRead(2));
    digitalWrite(9,  digitalRead(3));
    digitalWrite(10, digitalRead(4));
  }
}
```


ELSEVIER

Embedded Systems

Delays & Timers

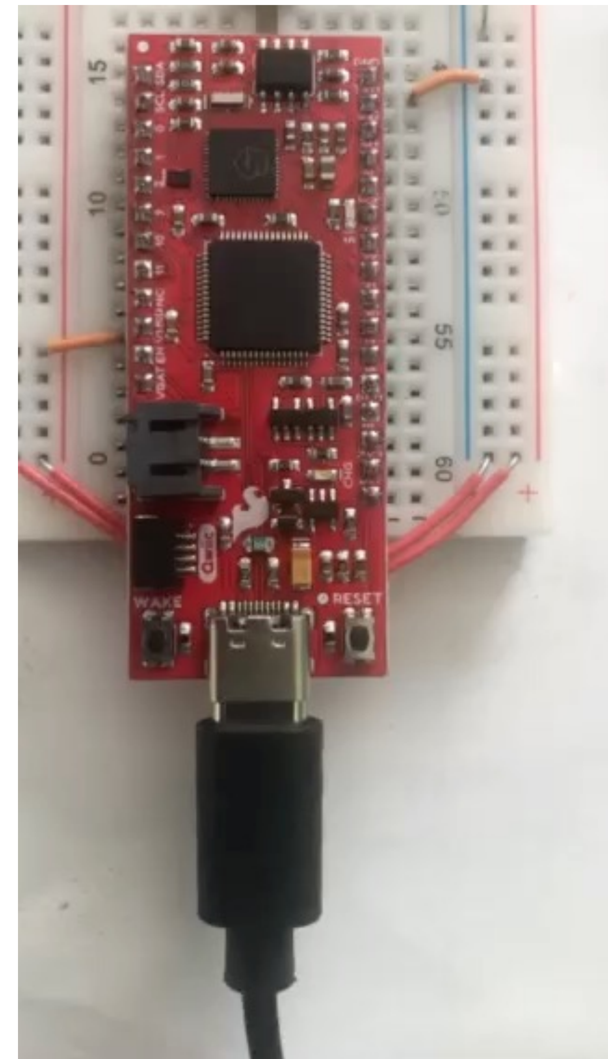
Generating Delays

```
#define COUNTS_PER_MS 1600

void delayLoop(int ms) {
    // declare loop counter volatile so it isn't optimized
    away
    // COUNTS_PER_MS empirically determined such that
    // delayLoop(1000) waits 1 sec
    volatile int i = COUNTS_PER_MS * ms;
    while (i--); // count down time
}
```


ELSEVIER

Blink Light

```
void flash(void) {
   pinMode(5, OUTPUT);

   while (1) {
      digitalWrite(5, 1);
      delayLoop(500);
      digitalWrite(5, 0);
      delayLoop(500);
   }
}
```



Timers

- **delayLoop** requires calibrating **COUNTS_PER_MS**
  – Tedious to calibrate by hand
  – Inexact
  – Could break if compiler optimizes better
- **Timers** are peripherals for time measurement
- FE310 has one **64-bit timer**
  – Counts ticks of an external 32.768 KHz oscillator

ELSEVIER

Timer Register

- Timers accessed through memory-mapped I/O
- FE310 timer in Core-Local Interruptor (CLINT)
  – 64-bit **mtime** register at **0x0200BFF8**

Address	Width	Attr.	Description	Notes
0x2000000	4B	RW	msip for hart 0	MSIP Registers (1 bit wide)
0x2004008 … 0x200bff7			Reserved	
0x2004000	8B	RW	mtimecmp for hart 0	MTIMECMP Registers
0x2004008 … 0x200bff7			Reserved	
0x200bff8	8B	RW	mtime	Timer Register
0x200c000			Reserved	

Table 24: CLINT Register Map

From FE310 Manual

ELSEVIER

Delay with Timer

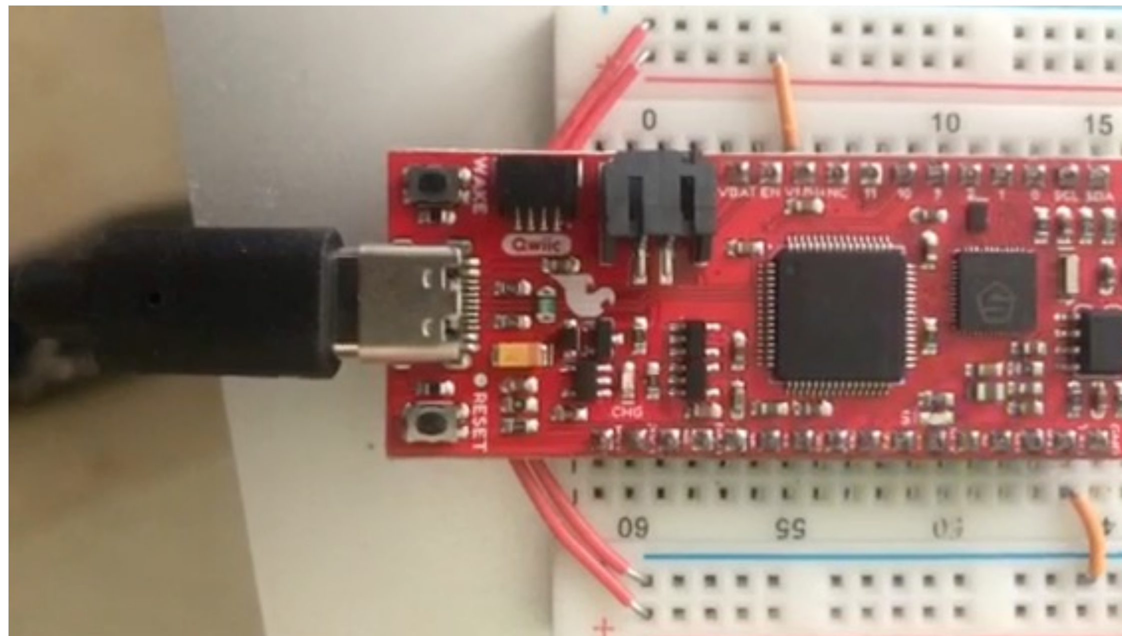
```
void delay(int ms) {
    volatile uint64_t *mtime = (uint64_t*)0x0200bff8;
    uint64_t doneTime = *mtime + (ms*32768)/1000;

    while (*mtime < doneTime); // wait until time is reached
}
```


ELSEVIER

Example: Morse Code

Morse Code Example

- Play a message in Morse code by blinking LEDs.
- Demonstrate EasyREDVIO
- Similarities to Lab 7



ELSEVIER

Morse Code Example

```
// morse.c
// David_Harris@hmc.edu

// Arduino-like library for IO functions
#include "EasyREDVIO.h"

#define DUR 100
```


ELSEVIER

Morse Code Example

```
// Define Morse code for letters
char codes[26][5] = {
 ".-",    // A
 "-...",  // B
 "-.-.",  // C
 "-..",   // D
 ".",     // E
 "..-.",  // F
 "--.",   // G
 "....",  // H
 "..",    // I
 ".---",  // J
 "-.-",   // K
 ".-..",  // L
 "--",    // M
 "-.",    // N
 "---",   // O
 ".--.",  // P
 "--.-",  // Q
 ".-.",   // R
 "...",   // S
 "-",     // T
 "..-",   // U
 "...-",  // V
 ".--",   // W
 "-..-",  // X
 "-.--",  // Y
 "--.."   // Z
};
```


ELSEVIER

Morse Code Example

```
void playChar(char c) {
  int i=0;

  while (codes[c-'A'][i]) {
    digitalWrite(5, 1); // turn on LED
    if (codes[c-'A'][i] == '.') delay(DUR);   // dot
    else                        delay(3*DUR); // dash
    digitalWrite(5, 0); // turn off LED
    delay(DUR); // pause between elements
    i++;
  }
  delay(DUR*2); // extra pause between characters
}
```


ELSEVIER

Morse Code Example

```
void playStr(char msg[]) {
  int i=0;

  while(msg[i]) playChar(msg[i++]);
}
```


ELSEVIER

Morse Code Example

```
int main(void) {
  pinMode(5, OUTPUT); // Blue LED
  while (1) {
    playStr("SOS");
    // pause between words
    delay(DUR*4);
  }
}
```


ELSEVIER

Embedded Systems

Interfacing

Interfacing

- **Interfacing:** connecting external devices to a microcontroller
  - Sensors
  - Actuators
  - Other Processors
- **Interfacing Methods**
  - **Parallel**
  - **Serial**
    - **SPI:** Serial Peripheral Interface
      - 1 clock, 1 data out, 1 data in pin
    - **UART:** Universal Asynchronous Receiver/Transmitter
      - no clock, 1 data out, 1 data in pin, agree in software about intended data rate
    - **I2C:** Inter-Integrated Circuit
      - 1 clock, 1 bidirectional data pin
  - **Analog**
    - **ADC:** Analog/Digital Converter
    - **DAC:** Digital/Analog Converter
    - **PWM:** Pulse Width Modulation

ELSEVIER

Parallel Interfacing

- Connect **1 wire / bit** of information
  - **Example:** 8-bit parallel interface to send a byte at a time
- Also send **clock** or **REQ/ACK** (request/acknowledge) to indicate when data is ready
- Parallel busses are **expensive** and **cumbersome** because of the large number of wires
- Mostly used for **high-performance applications** such as DRAM interfaces

ELSEVIER

Serial Interfacing

- Serial interface sends **one bit at a time**
  - Use many clock cycles to send a large block of information
  - Also send timing information about when the bits are valid
- **Common Serial Interfaces:**
  - Serial Peripheral Interface (**SPI**)
    - Serial clock, 2 unidirectional data wires (MOSI, MISO)
    - Very common, easy to use
  - Inter-Integrated Circuit (**I2C**) “I squared C”
    - 1 clock, 1 bidirectional data wire
    - Fewer wires, more complex internal hardware
  - Universal Asynchronous Receiver/Transmitter (**UART**) “you-art”
    - 2 unidirectional data wires (Tx, Rx)
    - Asynchronous (no clock); configure (agreed upon) speed at each end

Embedded Systems

SPI

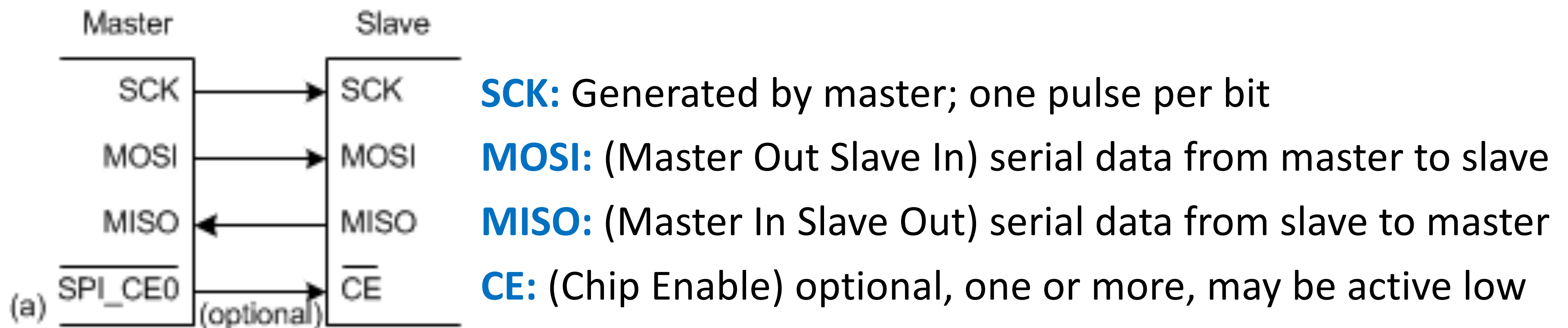
Serial Peripheral Interface (SPI)

- SPI is an easy way to connect devices
- **Master device** communicates with one or more **slave devices**
  - **Slave select signals** indicate which slave is chosen
  - **Master sends clock** and **data out**. **Slave** sends back **data in**.
- **Signals:**
  - **SCK/SCLK:** (Serial Clock) Generated by master; one pulse per bit
  - **MOSI:** (Master Out Slave In) serial data from master to slave
  - **MISO:** (Master In Slave Out) serial data from slave to master
  - **SS/CE/CS:** (Slave select/Chip Enable/Chip Select) optional, one or more, may be active low

ELSEVIER

SPI Waveforms

- SPI Slave hardware can be just a **shift register**

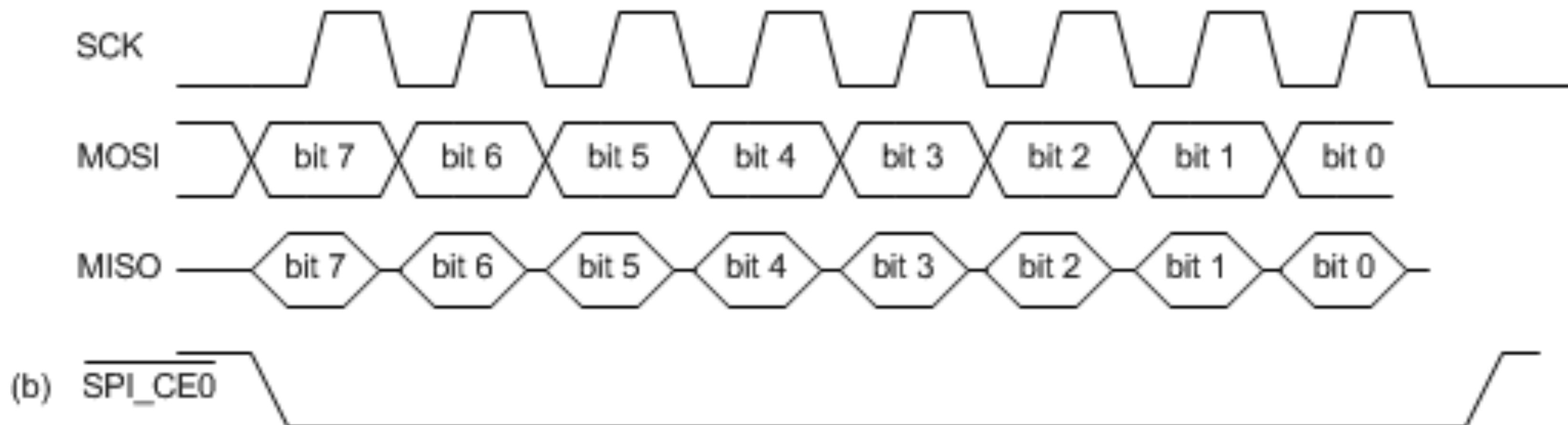


SCK: Generated by master; one pulse per bit

MOSI: (Master Out Slave In) serial data from master to slave

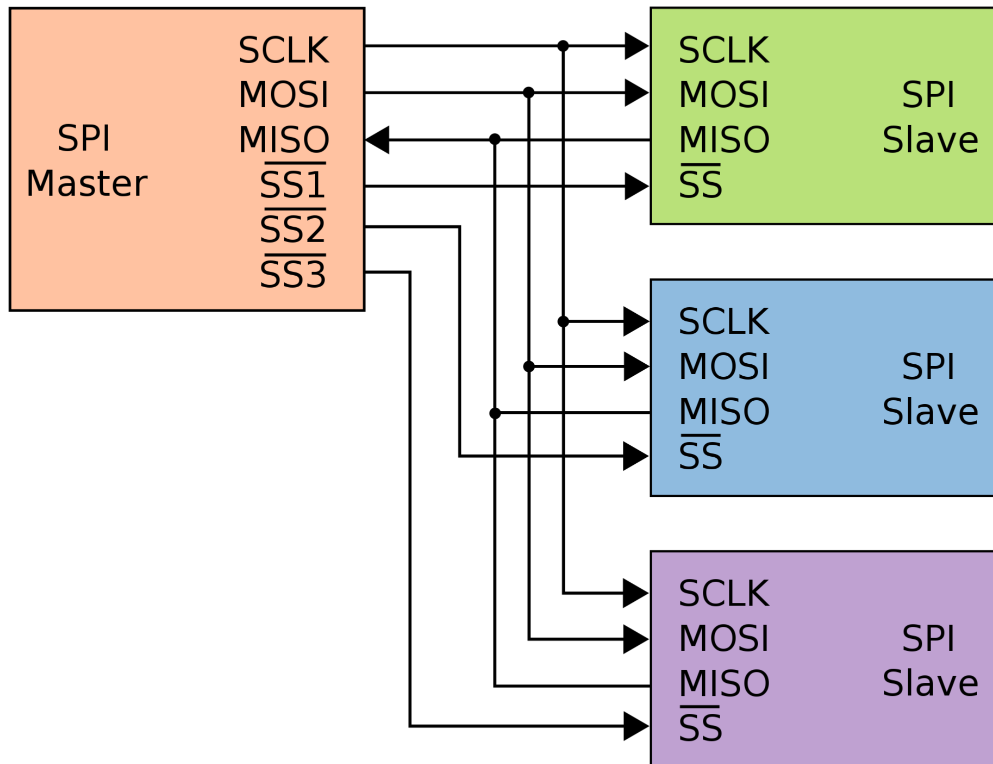
MISO: (Master In Slave Out) serial data from slave to master

CE: (Chip Enable) optional, one or more, may be active low

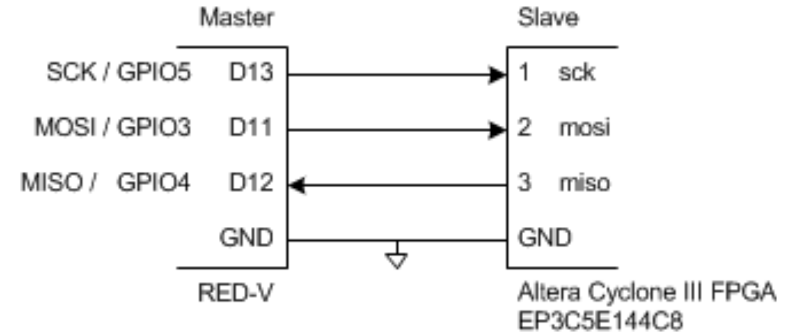


SPI Connection

Generic Master to Several Slaves



RED-V Master to One Slave



en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#/media/File:SPI_three_slaves.svg

ELSEVIER

Slave Select

- Slave Select is **active low**
  - Variously named NSS or $\overline{\text{SS}}$ or $\overline{\text{CE}}$ or $\overline{\text{CS}}$
  - Turns ON device when 0, OFF when 1
- Turning **OFF** the slave device may **save power**
- **Options:**
  - **1 slave device:** tie slave select active
  - **1 or more slave devices:** use GPIO pins to turn desired device ON before SPI transaction, turn device back OFF afterward
  - **1 or more slave devices:** use SPI controller to automatically pulse the desired SS pin during a transaction

ELSEVIER

SPI Communication

- **Configure**
  - **Choose port**
    - FE310 has SPI1 and SPI2. Let's use SPI1.
  - **Set SPI SCK, MISO, MOSI, CS0 pins to IOF0 mode**
  - **Set Baud Rate** (1 MHz or less prudent on a breadboard)
  - **Clock Polarity**
    - CPOL = 0 (default): clock idles at 0
    - CPOL = 1: clock idles at 1
  - **Clock Phase**
    - CPHA = 0: sample MOSI on the first clock transition
    - CPHA = 1: sample MOSI on the second clock transition
  - **Set other control registers as needed**
    - See EasyREDVIO code

ELSEVIER

SPI Communication Continued

- **Transmit data**
  - **Wait** for the **txdata full flag** to be 0 indicating SPI is ready for new data.
  - **Write byte** to **txdata** data field.
  - **SPI** will generate 8 clocks and send the 8 bits of data, while receiving 8 bits back.
- **Receive data**
  - **Read rxdata register.**
  - If **rxdata empty** flag is 1, there is no data yet. Repeat read.
  - When empty flag is 0, there is valid data in the **rxdata** field.

ELSEVIER

SPI Registers

- **Base Address**
  - SPI1 0x10024000
  - SPI2 0x10034000
- **Baud Rate**
  - sckdiv bits 11:0 indicate div
  - $F_{sck} = F_{in} / 2(div+1)$
  - F_{in} is 16 MHz coreclk
- **Clock Mode**
  - sckmode bit 0: pha (Clock Phase)
  - sckmode bit 1: pol (Clock Polarity
- **Transmit Data Register**
  - txdata bits 7:0: data (write to transmit)
  - txdata bit 31: full (1 indicates FIFO can accept data)
- **Receive Data Register**
  - rxdata bits 7:0: data (read received data)
  - rxdata bit 31: empty (1 indicates no data ready)

EasyREDVIO SPI Library

```
////////////////////////////////////////////////////////////////////////////////
// SPI Registers
////////////////////////////////////////////////////////////////////////////////

typedef struct
{
    volatile uint32_t   div         :   12; // Clock divisor
    volatile uint32_t               :   20;
} sckdiv_bits;

typedef struct
{
    volatile uint32_t   pha         :   1; // Serial clock phase
    volatile uint32_t   pol         :   1; // Serial clock polarity
    volatile uint32_t               :   30;
} sckmode_bits;

…
typedef struct
{
    volatile uint32_t   data        :   8; // Transmit data
    volatile uint32_t               :   23;
    volatile uint32_t   full        :   1; // FIFO full flag
} txdata_bits;

typedef struct
{
    volatile uint32_t   data        :   8; // Received data
    volatile uint32_t               :   23;
    volatile uint32_t   empty       :   1; // FIFO empty flag
} rxdata_bits;
…
```

Serial Clock Divisor Register (sckdiv)				
Register Offset	0x0			
Bits	Field Name	Attr.	Rst.	Description
[11:0]	div	RW	0x3	Divisor for serial clock. div_width bits wide.
[31:12]	Reserved			

Serial Clock Mode Register (sckmode)				
Register Offset	0x4			
Bits	Field Name	Attr.	Rst.	Description
0	pha	RW	0x0	Serial clock phase
1	pol	RW	0x0	Serial clock polarity
[31:2]	Reserved			

Transmit Data Register (txdata)				
Register Offset	0x48			
Bits	Field Name	Attr.	Rst.	Description
[7:0]	data	RW	0x0	Transmit data
[30:8]	Reserved			
31	full	RO	X	FIFO full flag

Receive Data Register (rxdata)				
Register Offset	0x4C			
Bits	Field Name	Attr.	Rst.	Description
[7:0]	data	RO	X	Received data
[30:8]	Reserved			
31	empty	RW	X	FIFO empty flag

$log_2(cs_width)$

From FE310 Manual

$log_2(cs_width)$

EasyREDVIO SPI Library

```
typedef struct
{
    volatile sckdiv_bits    sckdiv;      // (SPI offset 0x00) Serial clock divisor
    volatile sckmode_bits   sckmode;     // (SPI offset 0x04) Serial clock mode
    volatile uint32_t       Reserved1[2];
    volatile csid_bits      csid;        // (SPI offset 0x10) Chip select ID
    volatile csdef_bits     csdef;       // (SPI offset 0x14) Chip select default
    volatile csmode_bits    csmode;      // (SPI offset 0x18) Chip select mode
    volatile uint32_t       Reserved2[3];
    volatile delay0_bits    delay0;      // (SPI offset 0x28) Delay control 0
    volatile delay1_bits    delay1;      // (SPI offset 0x2C) Delay control 1
    volatile uint32_t       Reserved3[4];
    volatile fmt_bits       fmt;         // (SPI offset 0x40) Frame format
    volatile uint32_t       Reserved4[1];
    volatile txdata_bits    txdata;      // (SPI offset 0x48) Tx FIFO data
    volatile rxdata_bits    rxdata;      // (SPI offset 0x4C) Rx FIFO data
    volatile txmark_bits    txmark;      // (SPI offset 0x50) Tx FIFO watermark
    volatile rxmark_bits    rxmark;      // (SPI offset 0x54) Rx FIFO watermark
    volatile uint32_t       Reserved5[2];
    volatile fctrl_bits     fctrl;       // (SPI offset 0x60) SPI flash control*
    volatile ffmt_bits      ffmt;        // (SPI offset 0x64) SPI flash instr format*
    volatile uint32_t       Reserved6[2];
    volatile ie_bits        ie;          // (SPI offset 0x70) SPI interrupt enable
    volatile ip_bits        ip;          // (SPI offset 0x74) SPI interrupt pending
    // Registers marked * are only present on controllers with the direct-map flash ifc.

} SPI;
```

Offset	Name	Description
0x00	sckdiv	Serial clock divisor
0x04	sckmode	Serial clock mode
0x08	Reserved	
0x0C	Reserved	
0x10	csid	Chip select ID
0x14	csdef	Chip select default
0x18	csmode	Chip select mode
0x1C	Reserved	
0x20	Reserved	
0x24	Reserved	
0x28	delay0	Delay control 0
0x2C	delay1	Delay control 1
0x30	Reserved	
0x34	Reserved	
0x38	Reserved	
0x3C	Reserved	
0x40	fmt	Frame format
0x44	Reserved	
0x48	txdata	Tx FIFO Data
0x4C	rxdata	Rx FIFO data
0x50	txmark	Tx FIFO watermark
0x54	rxmark	Rx FIFO watermark
0x58	Reserved	
0x5C	Reserved	
0x60	fctrl	SPI flash interface control*
0x64	ffmt	SPI flash instruction format*
0x68	Reserved	
0x6C	Reserved	
0x70	ie	SPI interrupt enable
0x74	ip	SPI interrupt pending

From FE310 Manual

ELSEVIER

EasyREDVIO SPI Library

```
/////////////////////////////////////////////////////////////////////////////////
// SPI
/////////////////////////////////////////////////////////////////////////////////

#define QSPI0_BASE  (0x10014000U)   // QSPI0 memory-mapped base address
#define SPI1_BASE   (0x10024000U)   // SPI1 memory-mapped base address
#define SPI2_BASE   (0x10034000U)   // SPI2 memory-mapped base address

// Set up pointer to SPI structures at the base addresses
#define QSPI0 ((SPI*) QSPI0_BASE)
#define SPI1  ((SPI*) SPI1_BASE)
#define SPI2  ((SPI*) SPI2_BASE)
```


ELSEVIER

EasyREDVIO SPI Library

```
void spiInit(uint32_t clkdivide, uint32_t cpol,
             uint32_t cpha) {
    pinMode(2, GPIO_IOF0); // CS0
    pinMode(3, GPIO_IOF0); // MOSI
    pinMode(4, GPIO_IOF0); // MISO
    pinMode(5, GPIO_IOF0); // SCK

    SPI1->sckdiv.div = clkdivide; // Set clock divide
    SPI1->sckmode.pol = cpol;     // Set polarity
    SPI1->sckmode.pha = cpha;     // Set phase
}
```


ELSEVIER

EasyREDVIO SPI Library

```
uint8_t spiSendReceive(uint8_t send) {
  // Wait until transmit FIFO is ready for new data
  while(SPI1->txdata.full);

  // Transmit the character over SPI
  SPI1->txdata.data = send;

  rxdata_bits rxdata;
  while (1) {
    // Read the rxdata register EXACTLY once
    rxdata = SPI1->rxdata;
    // If the empty bit was not set, return the data
    if (!rxdata.empty) {
      return (uint8_t)rxdata.data;
    }
  }
}
```


ELSEVIER

Generalized pinMode

```
#define INPUT 0
#define OUTPUT 1
#define GPIO_IOF0 2
#define GPIO_IOF1 3
void pinMode(int gpio_pin, int function)
{
    switch(function) {
        case INPUT:
            GPIO0->input_en     |= (1 << gpio_pin);   // Sets a pin as an input
            GPIO0->iof_en       &= ~(1 << gpio_pin);  // Disable IOF
            break;
        case OUTPUT:
            GPIO0->output_en    |= (1 << gpio_pin);   // Set pin as an output
            GPIO0->iof_en       &= ~(1 << gpio_pin);  // Disable IOF
            break;
         case GPIO_IOF0:
            GPIO0->iof_en       |=  (1 << gpio_pin);  // Enable IOF
            GPIO0->iof_sel      &= ~(1 << gpio_pin);  // IO Function 0
            break;
         case GPIO_IOF1:
            GPIO0->iof_en       |=  (1 << gpio_pin);  // Enable IOF
            GPIO0->iof_sel      |=  (1 << gpio_pin);  // IO Function 1
            break;
    }
}
```


ELSEVIER

SPI Communication

```
#include <stdint.h>
#include "EasyREDVIO.h"

int main(void) {
  uint8_t sample;
  spiInit(15, 0, 0);  // Initialize the SPI to 500 KHz
  while(1) {
    spiSendReceive('0x60');
    sample = spiSendReceive('0x00’);
    printf("Read %d\n", sample);
    delay(200);
  }
}
```

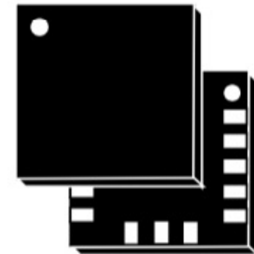

ELSEVIER

Example:

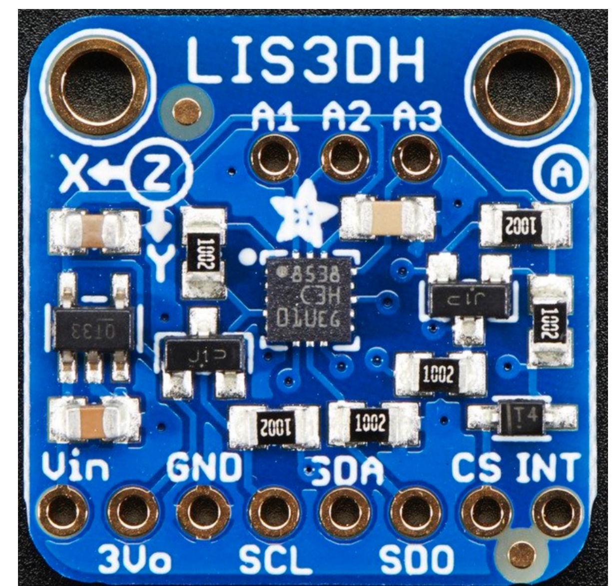
SPI Accelerometer

Example: SPI Interface to Accelerometer

- **LIS3DH 3-axis accelerometer**
  - Measures X, Y, Z acceleration
  - 2-8 G Full scale (configurable)
  - SPI interface with 16-bit transfers
  - ~1 mg sensitivity
  - 3x3 mm land grid array package
    - Hard to assemble without tooling
  - Available on a breakout board
    - $4.95 from Adafruit



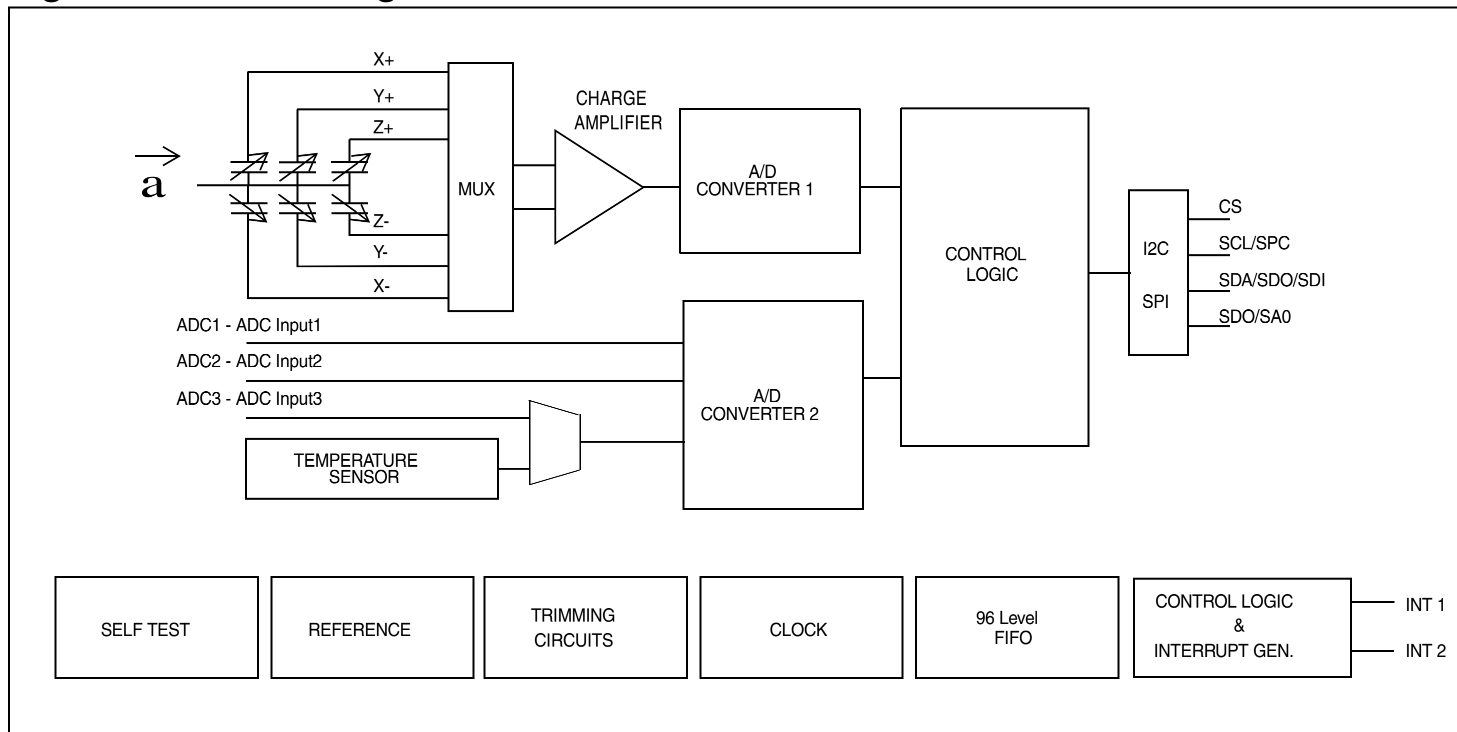
LGA-16 (3x3x1 mm)



ELSEVIER

LIS3DH Internal Operation

- MEMS cantilevers move based on acceleration
- Changes capacitance, sensed by ADC
- Temperature compensation for good accuracy

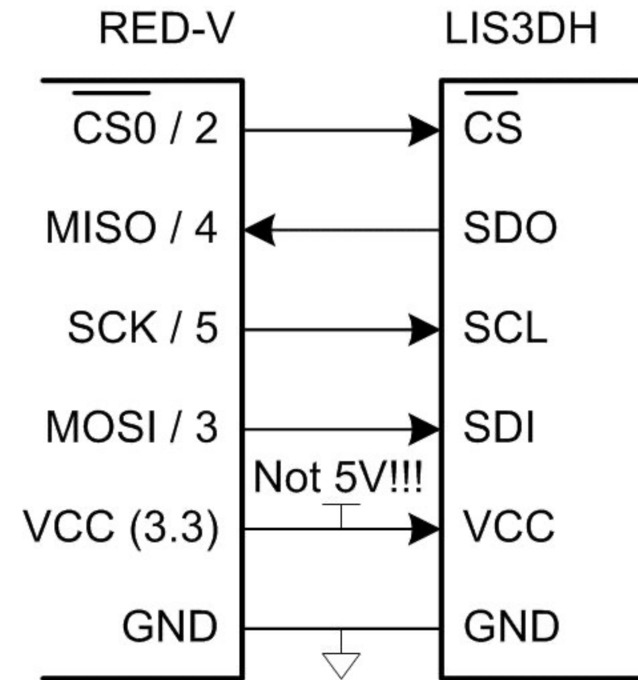


st.com LIS3DH Data Sheet

Microcontroller → Accel Interface

- 3.3V, GND
  - Connecting 5V may cook LIS3DH
- SPI interface

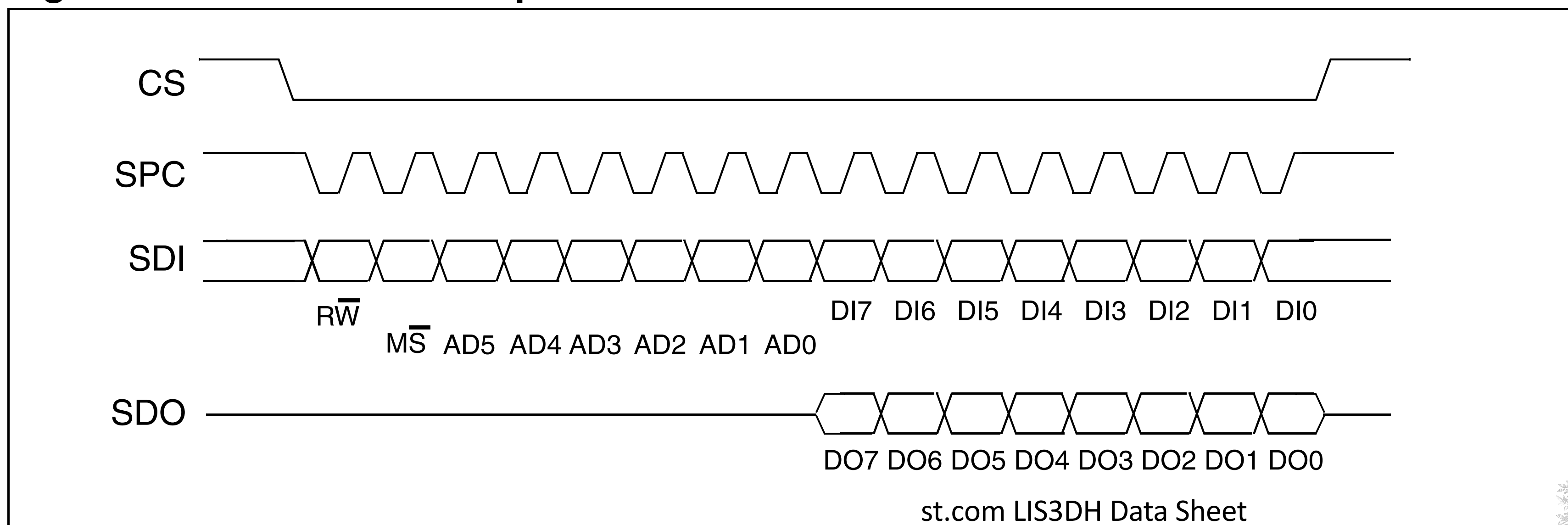
- LIS3DH also has I2C interface
  - Don't connect these pins



SPI Data Packets

- Communicate with SPI through internal registers
- Each register has 6-bit address, 8 bits of data
- Use a 16-bit SPI transaction
  - RWbar = 1 to read, 0 to write. MSbar = 0 for single register accesses
  - Keep CS low for entire 16 cycles

Figure 6. Read and write protocol



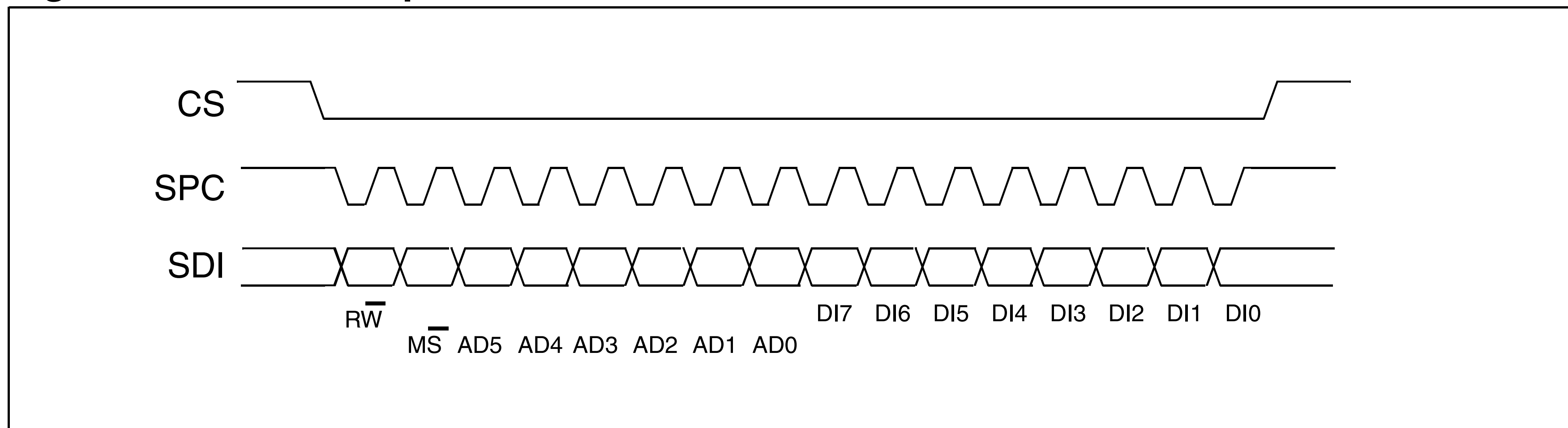
st.com LIS3DH Data Sheet

ELSEVIER

SPI Write

- RWbar = 0. MSbar = 0
- Send 6-bit address of register to write
- Send 8 bit data value to write to register
- Receive 8 bits of dummy data back (discard)

Figure 9. SPI write protocol

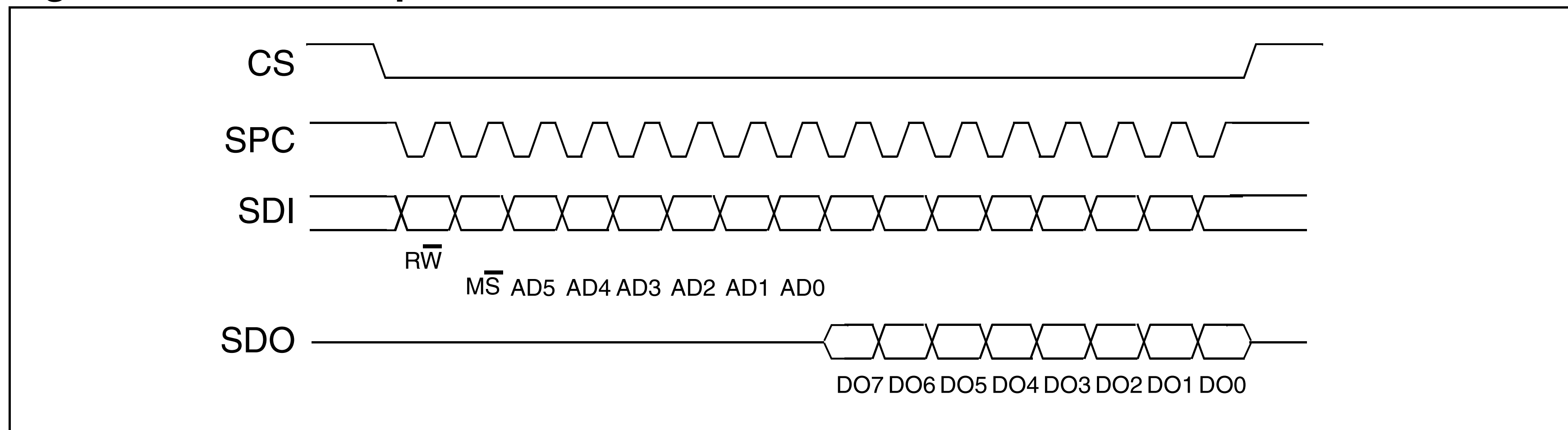


st.com LIS3DH Data Sheet

SPI Read

- RWbar = 1. MSbar = 0
- Send 6-bit address of register to read
- Send 8 bit dummy data
- Receive 8 bits of read data

Figure 7. SPI read protocol



st.com LIS3DH Data Sheet

LISD3H Registers

- Read WHO_AM_I to check SPI interface working
- Write CTRL_REG to configure accel
- Read OUT_X/Y/Z registers to measure acceleration
  - 16-bit 2's complement values
  - Stored in 8-bit high and low regs

Table 17. Register address map

Name	Type	Register address		Default	Comment
		Hex	Binary		
Reserved (do not modify)		00 - 06			Reserved
STATUS_REG_AUX	r	07	000 0111		
OUT_ADC1_L	r	08	000 1000	output	
OUT_ADC1_H	r	09	000 1001	output	
OUT_ADC2_L	r	0A	000 1010	output	
OUT_ADC2_H	r	0B	000 1011	output	
OUT_ADC3_L	r	0C	000 1100	output	
OUT_ADC3_H	r	0D	000 1101	output	
INT_COUNTER_REG	r	0E	000 1110		
WHO_AM_I	r	0F	000 1111	00110011	Dummy register
Reserved (do not modify)		10 - 1E			Reserved
TEMP_CFG_REG	rw	1F	001 1111		
CTRL_REG1	rw	20	010 0000	00000111	
CTRL_REG2	rw	21	010 0001	00000000	
CTRL_REG3	rw	22	010 0010	00000000	
CTRL_REG4	rw	23	010 0011	00000000	
CTRL_REG5	rw	24	010 0100	00000000	
CTRL_REG6	rw	25	010 0101	00000000	
REFERENCE	rw	26	010 0110	00000000	
STATUS_REG2	r	27	010 0111	00000000	
OUT_X_L	r	28	010 1000	output	
OUT_X_H	r	29	010 1001	output	
OUT_Y_L	r	2A	010 1010	output	
OUT_Y_H	r	2B	010 1011	output	
OUT_Z_L	r	2C	010 1100	output	
OUT_Z_H	r	2D	010 1101	output	

st.com LIS3DH Data Sheet

LISD3H WHO_AM_I

- WHO_AM_I register (0x0F) always reads 0x33 = 51
  - Easy way to check your SPI wiring is correct

WHO_AM_I (0Fh)

Table 21. WHO_AM_I register

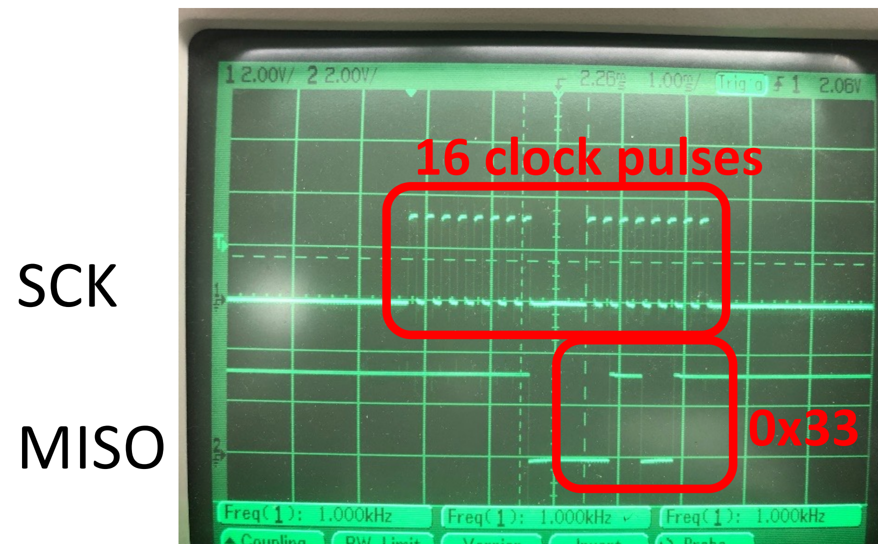
0	0	1	1	0	0	1	1

Device identification register.

st.com LIS3DH Data Sheet

SCK

MISO



LISD3H Control Registers

- CTRL_REG1 (0x20)
  - Write 0x77
  - Enable X, Y, Z axis measurements
  - Normal mode 400 Hz sampling

CTRL_REG1 (20h)

Table 24. CTRL_REG1 register

ODR3	ODR2	ODR1	ODR0	LPen	Zen	Yen	Xen

Table 25. CTRL_REG1 description

ODR3-0	Data rate selection. Default value: 00 (0000:50 Hz; Others: Refer to *Table 25*, "Data rate configuration")
LPen	Low power mode enable. Default value: 0 (0: normal mode, 1: low power mode)
Zen	Z axis enable. Default value: 1 (0: Z axis disabled; 1: Z axis enabled)
Yen	Y axis enable. Default value: 1 (0: Y axis disabled; 1: Y axis enabled)
Xen	X axis enable. Default value: 1 (0: X axis disabled; 1: X axis enabled)

ODR<3:0> is used to set power mode and ODR selection. In the following table are reported all frequency resulting in combination of ODR<3:0>

Table 26. Data rate configuration

ODR3	ODR2	ODR1	ODR0	Power mode selection
0	0	0	0	Power down mode
0	0	0	1	Normal / low power mode (1 Hz)
0	0	1	0	Normal / low power mode (10 Hz)
0	0	1	1	Normal / low power mode (25 Hz)
0	1	0	0	Normal / low power mode (50 Hz)
0	1	0	1	Normal / low power mode (100 Hz)
0	1	1	0	Normal / low power mode (200 Hz)
0	1	1	1	Normal / low power mode (400 Hz)
1	0	0	0	Low power mode (1.6 KHz)
1	0	0	1	Normal (1.25 kHz) / low power mode (5 KHz)

st.com LIS3DH Data Sheet

LISD3H Control Registers

- CTRL_REG4 (0x23)
  - Write 0x88
  - Update output after each reading
  - High resolution (16-bit) mode

CTRL_REG4 (23h)

Table 32. CTRL_REG4 register

BDU	BLE	FS1	FS0	HR	ST1	ST0	SIM

Table 33. CTRL_REG4 description

BDU	Block data update. Default value: 0 (0: continuos update; 1: output registers not updated until MSB and LSB reading)
BLE	Big/little endian data selection. Default value 0. (0: Data LSB @ lower address; 1: Data MSB @ lower address)
FS1-FS0	Full scale selection. default value: 00 (00: +/- 2G; 01: +/- 4G; 10: +/- 8G; 11: +/- 16G)

Table 33. CTRL_REG4 description (continued)

HR	High resolution output mode: Default value: 0 (0: High resolution disable; 1: High resolution Enable)
ST1-ST0	Self test enable. Default value: 00 (00: Self test disabled; Other: See *Table 34*)
SIM	SPI serial interface mode selection. Default value: 0 (0: 4-wire interface; 1: 3-wire interface).

Table 34. Self test mode configuration

ST1	ST0	Self test mode
0	0	Normal mode
0	1	Self test 0
1	0	Self test 1
1	1	--

st.com LIS3DH Data Sheet

ELSEVIER

LISD3H Output Registers

- Read X, Y, Z acceleration as 2 bytes each
- Combine into 16-bit 2's complement numbers
- Accel = (Reading – Offset) * Scale
  - Must calibrate offset and scale
  - Measure reading at level, rotated +/- 90 degrees on each axis
  - Different offset and scale for each axis

ELSEVIER

Accelerometer Starter Code

```
// E85 Lab 8: Digital Level
// David Harris & Josh Brake

#include <stdint.h>
#include "EasyREDVIO.h"
```


ELSEVIER

Accelerometer Starter Code

```
void spiWrite(uint8_t address, uint8_t value) {
  uint8_t hi, lo;
  digitalWrite(2, 0);           // pulse chip select
  hi = spiSendReceive(address);
  lo = spiSendReceive(value);
  digitalWrite(2, 1);           // release chip select
  // discard returned values on a write transaction
}

uint8_t spiRead(uint8_t address) {
  uint8_t hi, lo;
  digitalWrite(2, 0);           // pulse chip select
  hi = spiSendReceive(address | 1 << 7); // set msb for reads
  lo = spiSendReceive(0x00);    // send dummy payload
  digitalWrite(2, 1);           // release chip select
  return lo;  // return low byte from a read transaction
}
```


ELSEVIER

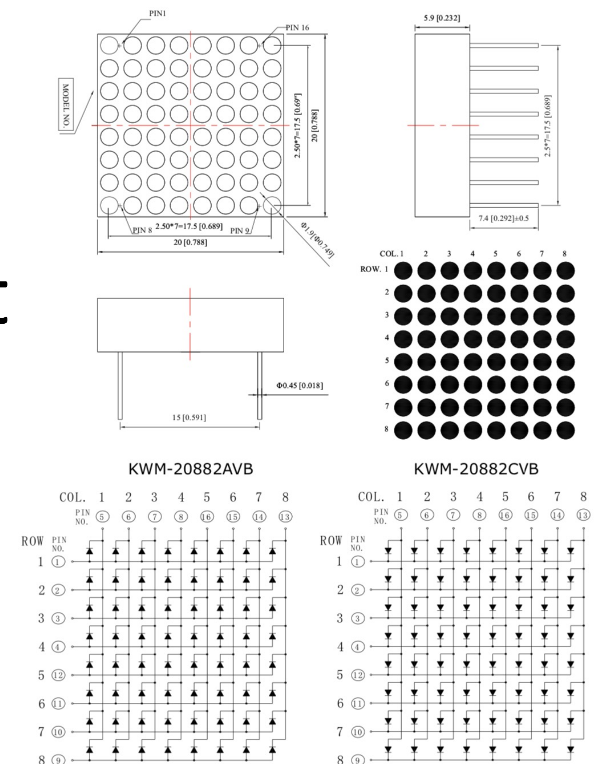
Accelerometer Starter Code

```
int main(void) {
    uint8_t debug;
    int16_t x,y;

    spiInit(15, 0, 0); // Initialize SPI pins and clocks
    digitalWrite(2, 1);
    pinMode(2, OUTPUT);

    // Check WHO_AM_I register. should return 0x33 = 51
    debug = spiRead(0x0F);

    //Setup the LIS3DH for use
    spiWrite(0x20, 0x77); // highest conversion rate, all axis on
    spiWrite(0x23, 0x88); // block update, and high resolution

    while(1) {
        // Collect the X and Y values from the LIS3DH
        x = spiRead(0x28) | (spiRead(0x29) << 8);
        y = spiRead(0x2A) | (spiRead(0x2B) << 8);

        // Small delay to reduce flicker
        delay(100);
    }
}
```


ELSEVIER

Lab 8: Digital Level

- Goal of Lab 8 is to build a **digital level**
- Accelerometer code on previous pages is provided
- You move a dot around on an LED matrix to indicate tilt
- Use digitalWrite from EasyREDVIO.h



cdn-shop.adafruit.com/datasheets/454datasheet.pdf

About these Notes

Digital Design and Computer Architecture Lecture Notes

© 2021 Sarah Harris and David Harris

These notes may be used and modified for educational and/or non-commercial purposes so long as the source is attributed.

ELSEVIER