E85 Digital Design & Computer Engineering

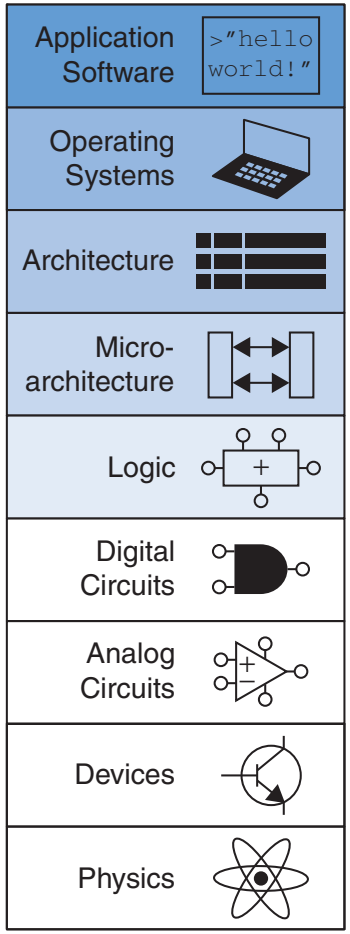
Lecture 16:

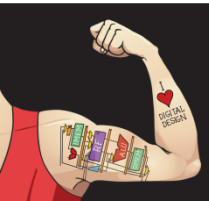
Microcontrollers
Libraries & Examples

HARVEY MUDD COLLEGE

Lecture 16

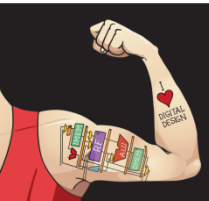
- Cortex Microcontroller Software Interface Standard (CMSIS)
- EasyNucleoIO Library
- Morse Code GPIO Example
- Accelerometer SPI Example





CMSIS

- Gives names to all the I/O registers
  - Organized as structures
  - So you don't have to look up and type in addresses
- Please don't use this in Lab 7
  - I want you to get comfortable looking up addresses once
- But take advantage of it for Lab 8



Without and With CMSIS

Turn on Nucleo Board LED (PB3)

Without CMSIS

```
volatile unsigned long *RCC_AHBENR = (unsigned long*)0x40021014;
volatile unsigned long *GPIOB_MODER = (unsigned long*)0x48000400;
volatile unsigned long *GPIOB_ODR = (unsigned long*)0x48000414;

*RCC_AHBENR |= (1 << 18);
*GPIOB_MODER |= (1 << 6);  // set PB3 mode to output
*GPIOB_ODR |= (1 << 3);    // turn on PB3
```

With CMSIS

```
#include <stm32f042x6.h>
RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
GPIOB->MODER |= 1 << 6;
GPIOB->ODR |= 1 << 3;
```

Peeking Inside CMSIS: Structures

```
typedef struct
{
  __IO uint32_t MODER;        /*!< GPIO port mode register,                Address offset: 0x00 */
  __IO uint32_t OTYPER;       /*!< GPIO port output type register,         Address offset: 0x04 */
  __IO uint32_t OSPEEDR;      /*!< GPIO port output speed register,        Address offset: 0x08 */
  __IO uint32_t PUPDR;        /*!< GPIO port pull-up/pull-down register,   Address offset: 0x0C */
  __IO uint32_t IDR;          /*!< GPIO port input data register,          Address offset: 0x10 */
  __IO uint32_t ODR;          /*!< GPIO port output data register,         Address offset: 0x14 */
  __IO uint32_t BSRR;         /*!< GPIO port bit set/reset register,       Address offset: 0x1A */
  __IO uint32_t LCKR;         /*!< GPIO port configuration lock register, Address offset: 0x1C */
  __IO uint32_t AFR[2];       /*!< GPIO alternate function low register,   Address offset: 0x20-0x24 */
  __IO uint32_t BRR;          /*!< GPIO bit reset register,                Address offset: 0x28 */
}GPIO_TypeDef;
```

Notes:

- Uint32_t is a 32-bit unsigned integer data type (equivalent to unsigned long)
- __IO is equivalent to volatile
- Each element of the struct is 4 bytes past the previous

Peeking Inside CMSIS: Addresses

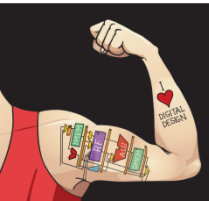
```
#define PERIPH_BASE           ((uint32_t)0x40000000)
#define AHB2PERIPH_BASE       (PERIPH_BASE + 0x08000000)
#define GPIOA_BASE            (AHB2PERIPH_BASE + 0x00000000)
#define GPIOB_BASE            (AHB2PERIPH_BASE + 0x00000400)
#define GPIOF_BASE            (AHB2PERIPH_BASE + 0x00001400)

#define GPIOA                ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB                ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOF                ((GPIO_TypeDef *) GPIOF_BASE)

#define  RCC_AHBENR_GPIOAEN  ((uint32_t)0x00020000)      /*!< GPIOA clock enable */
#define  RCC_AHBENR_GPIOBEN  ((uint32_t)0x00040000)      /*!< GPIOB clock enable */
#define  RCC_AHBENR_GPIOFEN  ((uint32_t)0x00400000)      /*!< GPIOF clock enable */
```

Notes:

- RCC_AHBENR_GPIOBEN = 0x00040000 = (1 << 18)
- GPIOB_BASE is 0x40000000+0x08000000+0x00000400 = 0x48000400
- GPIOA->ODR is at GPIOB_BASE + 0x14 = 0x48000414

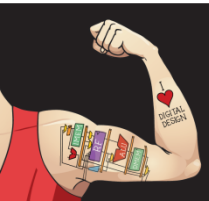
EasyNucleoIO Library

- We have developed a library giving Arduino-style access to the Nucleo I/O.
  - Built on top of CMSIS
  - Tricky part is pin numbering
- Library Functions
  - `void EasyNucleoIOInit(void);  // call before others`
  - `void pinMode(int pin, int function); // 0 = input, 1 = output, 2 = ALT`
  - `void digitalWrite(int pin, int val); // Set pin to val`
  - `int  digitalRead(int pin);  // return value of pin`
  - `void delayLoop(int ms);  // delay for given number of ms`
- Please do not use in Lab 7 but use in Lab 8



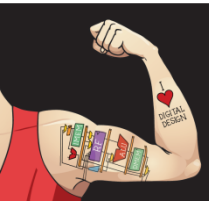
EasyNucleoIO Pin Numbering

```
//  Arduino Pin      STM32 Pin
//       D0          PA10
//       D1          PA9
//       D2          PA12          DO NOT USE
//       D3          PB0
//       D4          PB7           DO NOT USE
//       D5          PB6           DO NOT USE
//       D6          PB1
//       D7          PF0
//       D8          PF1
//       D9          PA8
//       D10         PA11
//       D11         PB5
//       D12         PB4
//       D13         PB3 (on-board green LED)
//       A0 (D14)    PA0
//       A1 (D15)    PA1
//       A2 (D16)    PA3
//       A3 (D17)    PA4
//       A4 (D18)    PA5
//       A5 (D19)    PA6
//       A6 (D20)    PA7
//       A7 (D21)    PA2
```



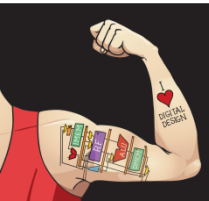
Internal Pin Numbering

- Let us create a code for pins
  - Easily identify port and pin within the port
  - Not all codes are used (16 for Port A, 8 for Port B, 2 for Port F)
- 16 codes for each port
  - Port A: 0-15 (all used for PA15:0)
  - Port B: 16-31 (only 23-16 for PB7:0)
  - Port F: 32-47 (only 33-32 for PF1:0)
  - PF0 = 32
  - PF1 = 33
- Find port based on code / 16
- Find pin based on code % 16



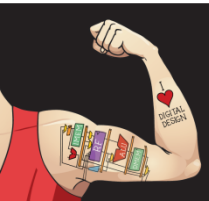
EasyNucleoIO Pin Numbering

// Arduino Pin	STM32 Pin		Code
// D0	PA10		10
// D1	PA9		9
// D2	PA12	DO NOT USE	12
// D3	PB0		16
// D4	PB7	DO NOT USE	23
// D5	PB6	DO NOT USE	22
// D6	PB1		17
// D7	PF0		32
// D8	PF1		33
// D9	PA8		8
// D10	PA11		11
// D11	PB5		21
// D12	PB4		20
// D13	PB3 (on-board green LED)		19
// A0 (D14)	PA0		0
// A1 (D15)	PA1		1
// A2 (D16)	PA3		3
// A3 (D17)	PA4		4
// A4 (D18)	PA5		5
// A5 (D19)	PA6		6
// A6 (D20)	PA7		7
// A7 (D21)	PA2		2



EasyNucleoIOInit

```
void EasyNucleoIOInit(void) {
     //Clocks:
     //enable clock for GPIOA, B, F
     RCC->AHBENR |= RCC_AHBENR_GPIOAEN |
                    RCC_AHBENR_GPIOBEN |
                    RCC_AHBENR_GPIOFEN;

     //enable clock to SPI1
     RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
}
```

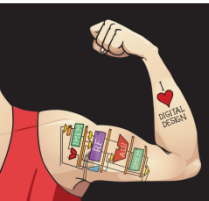


Pin Mapping Helper Functions

```
int digitalPinMapping[22] = {10, 9, 12, 16, 23, 22, 17,
                             32, 33, 8, 11, 21, 20, 19,
                             0, 1, 3, 4, 5, 6, 7, 2};

GPIO_TypeDef* pinToReg(int pin) {
      int p = digitalPinMapping[pin];
      if (p < 16) return GPIOA;
      else if (p < 32) return GPIOB;
      else return GPIOF;
}

int pinToOffset(int pin) {
      int p = digitalPinMapping[pin];
      return p % 16;
}
```

digitalRead

```
int digitalRead(int pin) {
      GPIO_TypeDef *gpio = pinToReg(pin);
      int offset = pinToOffset(pin);

      return (gpio->IDR >> offset) & 0x1;
}
```



digitalWrite

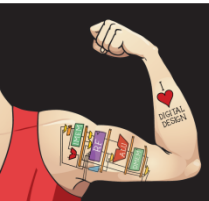
```
void digitalWrite(int pin, int val) {
     GPIO_TypeDef *gpio = pinToReg(pin);
     int offset = pinToOffset(pin);

     if (val) gpio->ODR |=  (1 << offset);
     else     gpio->ODR &= ~(1 << offset);
}
```

pinMode

```
void pinMode(int pin, int function) {
      GPIO_TypeDef *gpio = pinToReg(pin);
      int offset = pinToOffset(pin)*2;

      gpio->MODER &= ~((3 & ~function) << offset);
      gpio->MODER |=  ((3 &  function) << offset);
}
```

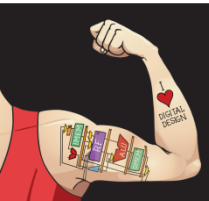
Morse Code Example

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



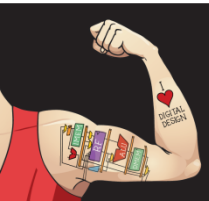
Morse Code Example

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


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


#define DUR 100
```



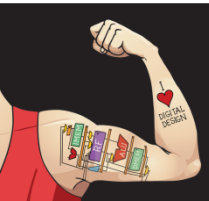
Morse Code Example

```
// 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
};
```



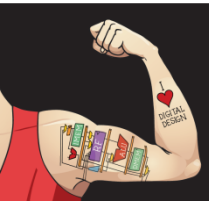
Morse Code Example

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

  while (codes[c-'A'][i]) {
    digitalWrite(13, 1); // turn on LED
    if (codes[c-'A'][i] == '.') delayLoop(DUR);   // dot
    else                        delayLoop(3*DUR); // dash
    digitalWrite(13, 0); // turn off LED
    delayLoop(DUR); // pause between elements
    i++;
  }
  delayLoop(DUR*2); // extra pause between characters
}
```



Morse Code Example

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

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

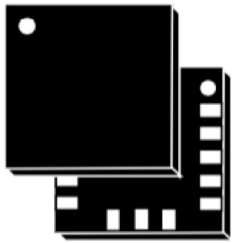


Morse Code Example

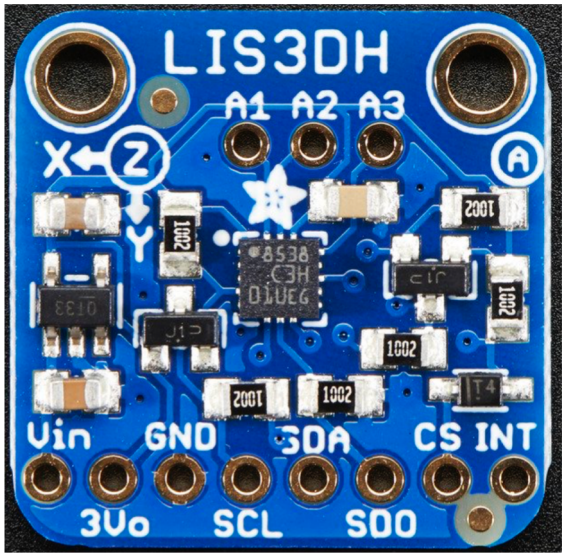
```
int main(void) {
  EasyNucleoIOInit();
  pinMode(13, OUTPUT); // Green LED
  while (1) {
    playStr("SOS");
    // pause between words
    delayLoop(DUR*4);
  }
}
```

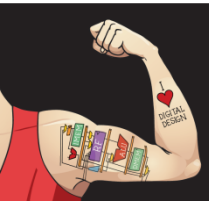
Example: SPI Interface to Accelerometer

- 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)

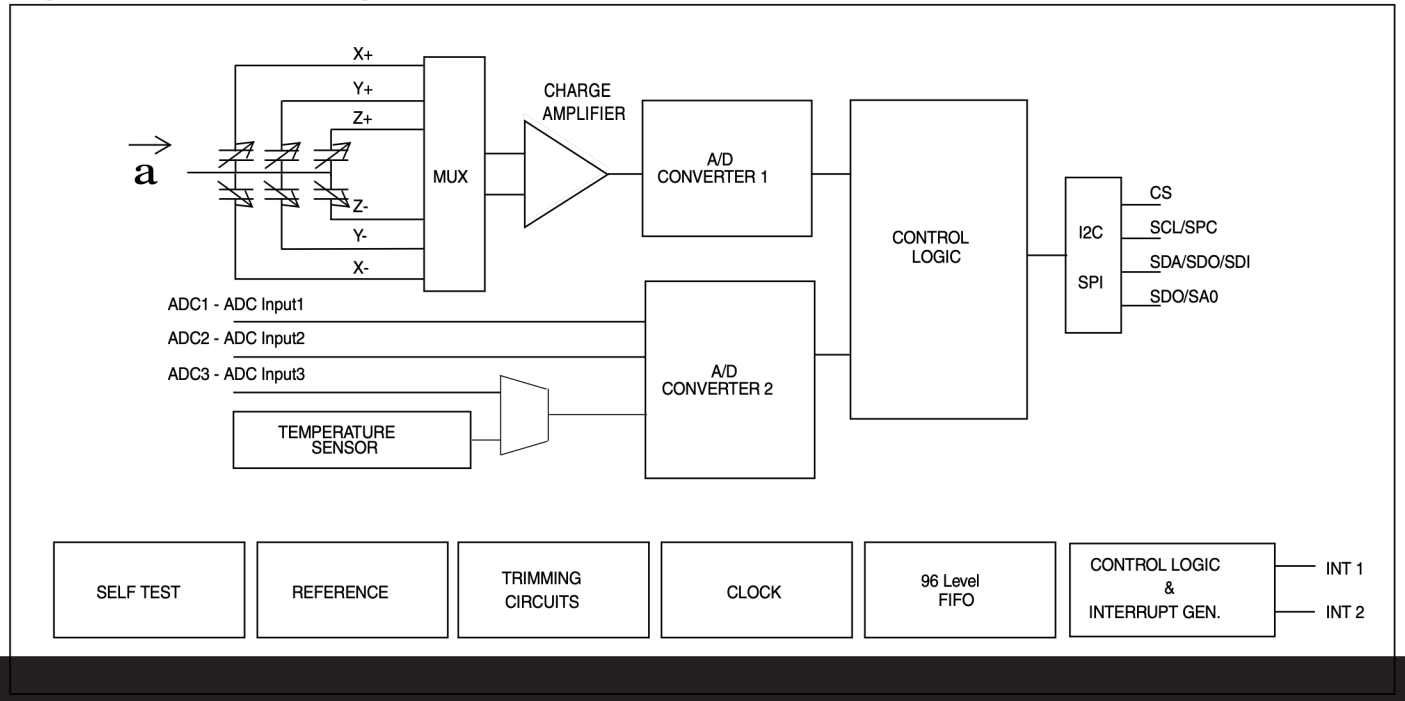


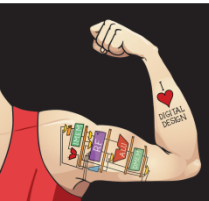


LIS3DH Internal Operation

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

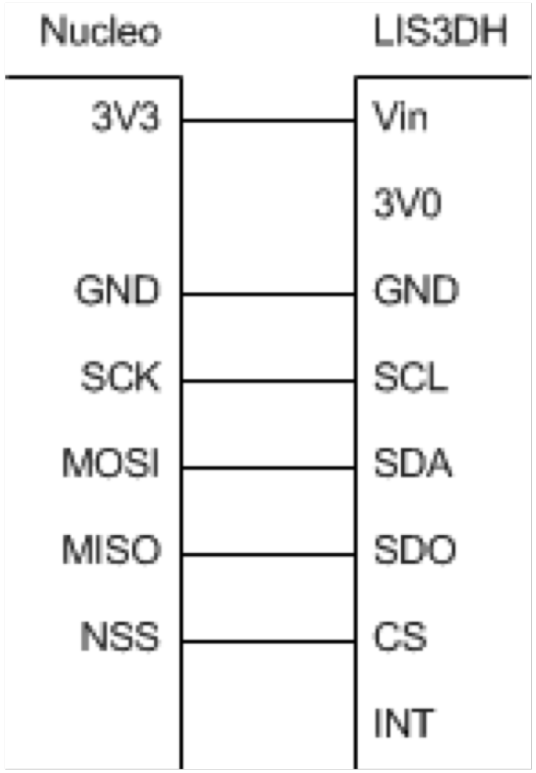
Figure 1. Block diagram

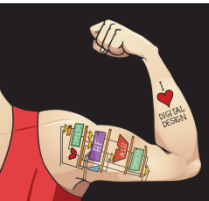




Microcontroller -> Accel Interface

- 3.3V, GND
- SPI interface
- Optional Interrupt
  - Indicates new sample ready

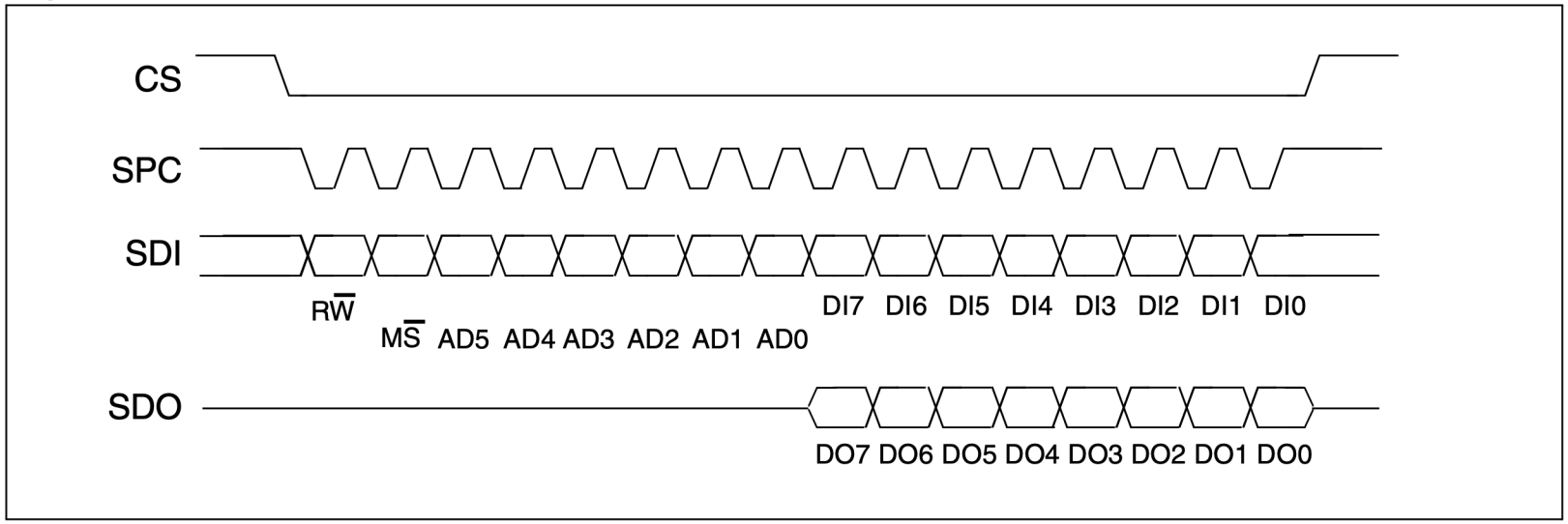


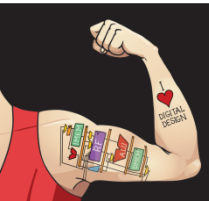


SPI Data Packets

- 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

Figure 6. Read and write protocol

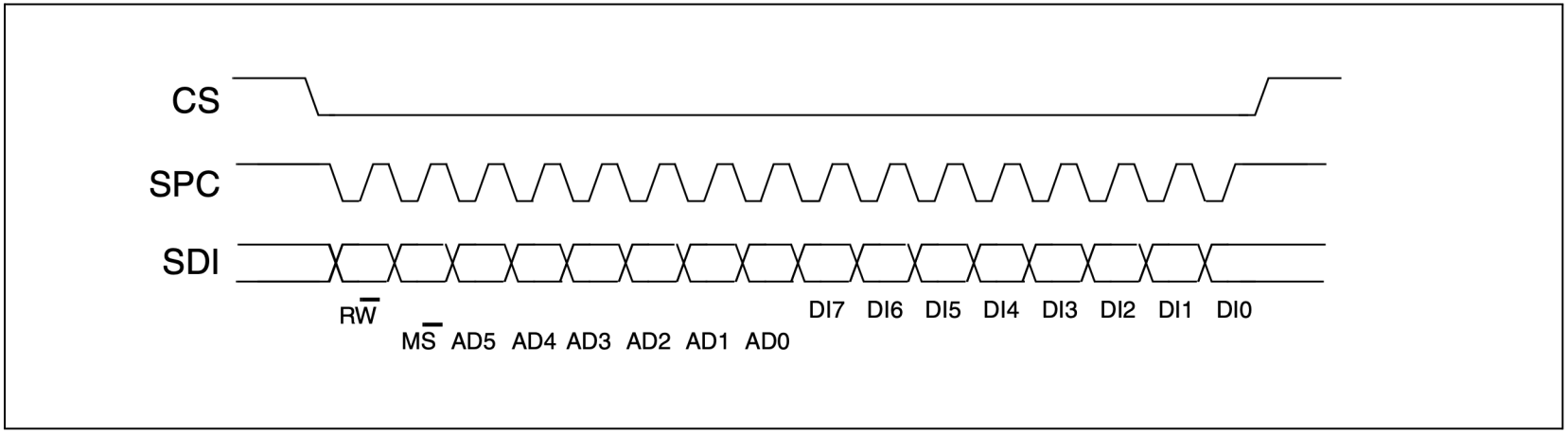


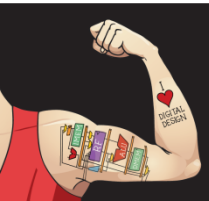


SPI Write

- 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

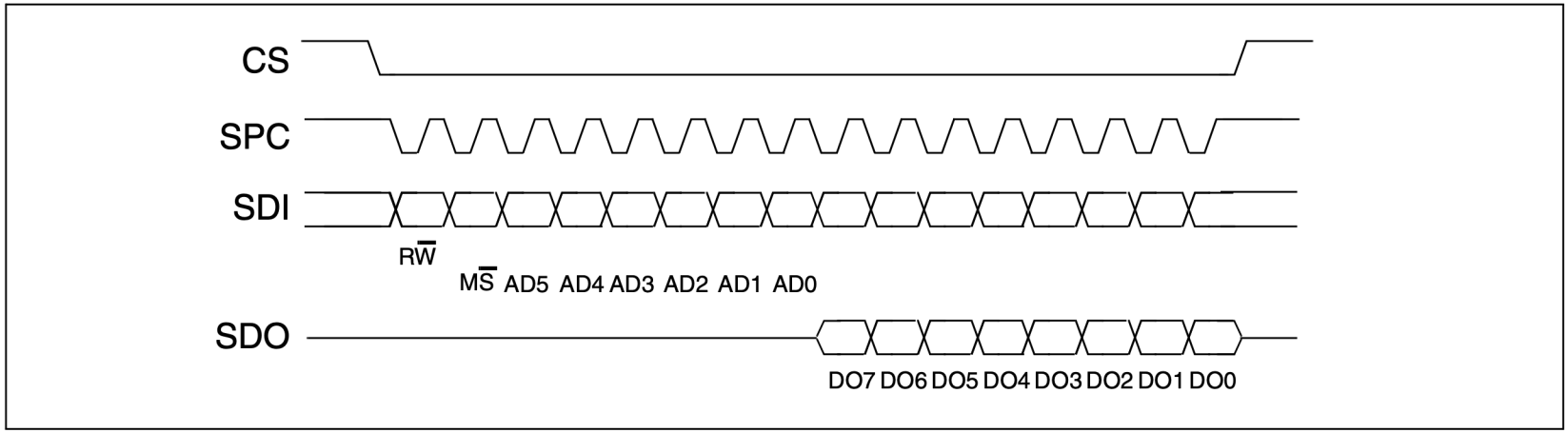




SPI Read

- 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

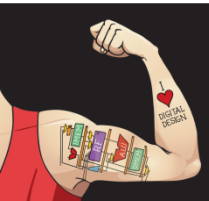


LISD3H Registers

- 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	



LISD3H WHO_AM_I

- WHO_AM_I register (0x0F) always reads 33
  - 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.

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

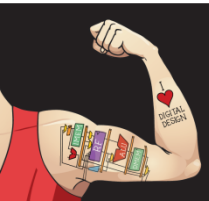
Table 25. CTRL_REG1 description

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)



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

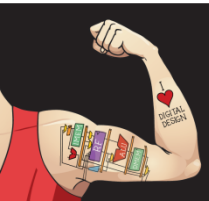
BDU	Block data update. Default value: 0 (0: continuos update; 1: output registers not updated until MSB and LSB reading)
BLE	Big/little endian data selection. Default value 0. (0: Data LSB @ lower address; 1: Data MSB @ lower address)
FS1-FS0	Full scale selection. default value: 00 (00: +/- 2G; 01: +/-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

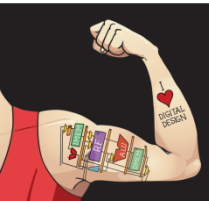
Table 34. Self test mode configuration

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



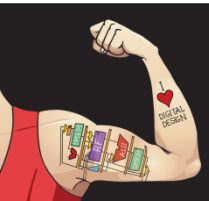
LISD3H Output Registers

- 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



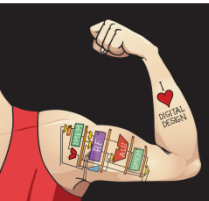
Accelerometer Starter Code

```
// E85 Lab 8: Digital Level
// Cherie Ho and David Money Harris
// Based on Bryce's SPI + GPIO STM32F0 CMSIS example

#include "stm32f0xx.h"
#include "EasyNucleoIO.h"

////////////////////////////////////////////////////////////////
////////
// SPI Functions
////////////////////////////////////////////////////////////////
////////

#define ALT 2
```



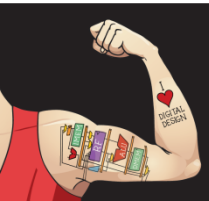
Accelerometer Starter Code

```
void spiInit() {
      //set ALT function for SPI pins
      pinMode(17, ALT); // PA4/SPI1_NSS (A3)
      pinMode(18, ALT); // PA5/SPI1_SCK (A4)
      pinMode(19, ALT); // PA6/SPI1_MISO (A5)
      pinMode(20, ALT); // PA7/SPI1_MOSI (A6)

      //configure SPI:
      //BR = fclk/8, master mode
      SPI1->CR1 |= SPI_CR1_BR_1 | SPI_CR1_MSTR;
      // Enable NSS select signal,
      //pulse NSS between transfers, 16-bit data
      SPI1->CR2 |= SPI_CR2_SSOE | SPI_CR2_NSSP | (0xF) << 8;
      //enable SPI
      SPI1->CR1 |= SPI_CR1_SPE;
}
```



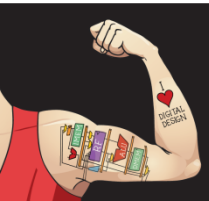
Accelerometer Starter Code

```
//send a 16-bits (short) on SPI1
unsigned short SPI1SendReceive16(unsigned short outDat) {
  SPI1->DR = outDat;
  while(!(SPI1->SR & SPI_SR_RXNE)); // wait for response
  return SPI1->DR;
}

void spiWrite(unsigned char address, unsigned char value) {
  SPI1SendReceive16(address << 8 | value);
}

unsigned char spiRead(unsigned char address) {
  return SPI1SendReceive16(address << 8 | (1 << 15));
}
```



Accelerometer Starter Code

```
int main(void) {
        volatile unsigned char debug;
        volatile short x,y;

        //setup clocks and hardware
        EasyNucleoIOInit();
        spiInit(); // Initialize SPI pins and clocks

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

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

        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
                delayLoop(100);
        }
}
```



Lab 8

- 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

  EasyNucleoIO