E85 Digital Design & Computer Engineering

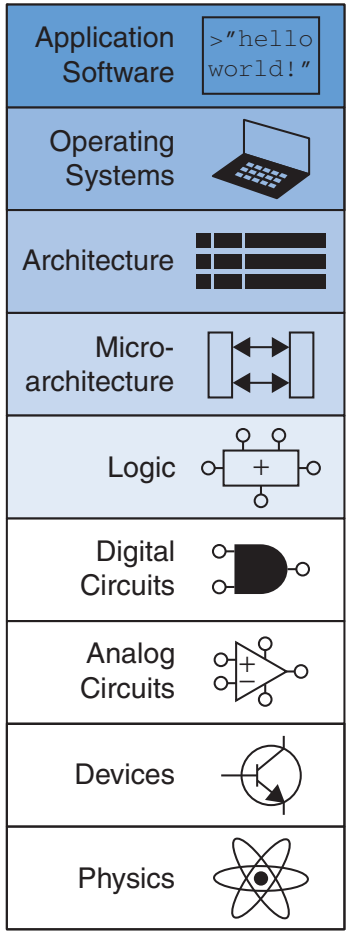
Lecture 15:

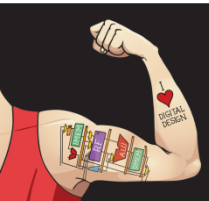
Microcontrollers
Interfacing

HARVEY MUDD COLLEGE

Lecture 15

- Interfacing
- Parallel
- Serial
  - SPI
- Analog



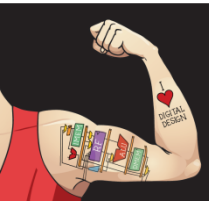


Interfacing

- Interfacing: connecting external devices to a microcontroller
  - Sensors
  - Actuators
  - Other Processors
- Interacing 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

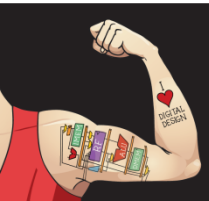
Parallel Interfacing

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



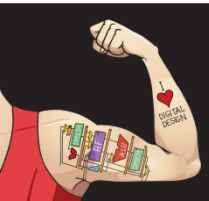
Parallel Interfacing

- *** show example



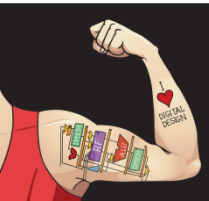
Serial Interfacing

- 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



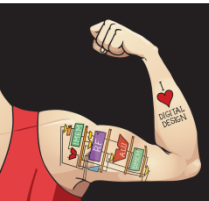
Serial Peripheral Interface (SPI)

- 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
  - SCLK: 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: Slave select (optional, one or more, may be active low)

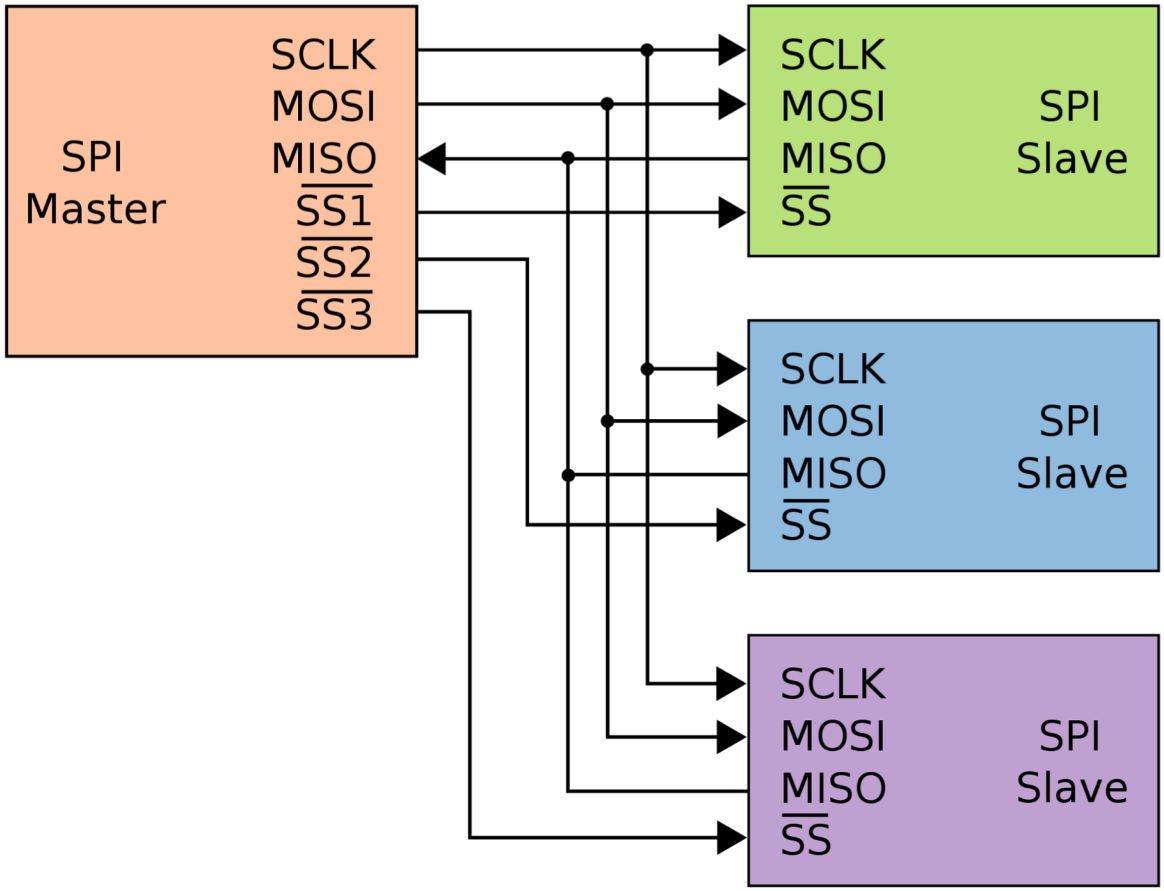


SPI Waveforms

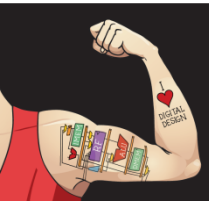
- *** show waveforms
- Show 8 clocks and 8 bits of data, value transmitted
- Another slide about using a shift register to receive.
  - Step by step



SPI Connection



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



STM32 SPI Pinout

Table 13. STM32F042x pin definitions (continued)

Pin numbers						Pin name (function upon reset)	Pin type	I/O structure	Notes	Pin functions	
LQFP48/UFQFPN48	WLCSP36	LQFP32	UFQFPN32	UFQFPN28	TSSPOP20					Alternate function	Additional functions
14	C3	10	10	10	10	PA4	I/O	TTa	-	SPI1_NSS, I2S1_WS, TIM14_CH1, TSC_G2_IO1, USART2_CK USB_NOE	ADC_IN4
15	D3	11	11	11	11	PA5	I/O	TTa	-	SPI1_SCK, I2S1_CK, CEC, TIM2_CH1_ETR, TSC_G2_IO2	ADC_IN5
16	E3	12	12	12	12	PA6	I/O	TTa	-	SPI1_MISO, I2S1_MCK, TIM3_CH1, TIM1_BKIN, TIM16_CH1, TSC_G2_IO3, EVENTOUT	ADC_IN6
17	F4	13	13	13	13	PA7	I/O	TTa	-	SPI1_MOSI, I2S1_SD, TIM3_CH2, TIM14_CH1, TIM1_CH1N, TIM17_CH1, TSC_G2_IO4, EVENTOUT	ADC_IN7

Table 13 on Page 33 of STM32F042×4 Datasheet

NSS: PA4 (A3 on Nucleo board)

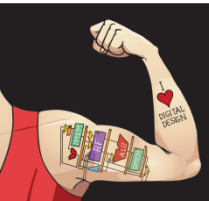
SCK: PA5 (A4 on Nucleo board)

MISO: PA6 (A5 on Nucleo board)

MOSI: PA7 (A6 on Nucleo board)

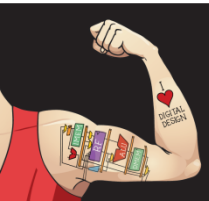
SPI Communication

- Configure
  - Enable SPI Clock Tree
  - Set SPI SCK, MISO, MOSI pins to ALT 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



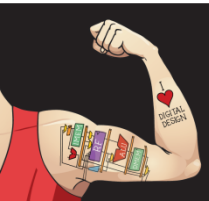
SPI Communication Continued

- Write byte to SPI data register to transmit
  - Data shifted out one bit at a time
- Wait for Receive Not Empty (SPI1_SR_RXNE bit)
- Read byte from SPI data register



SPI Slave Select

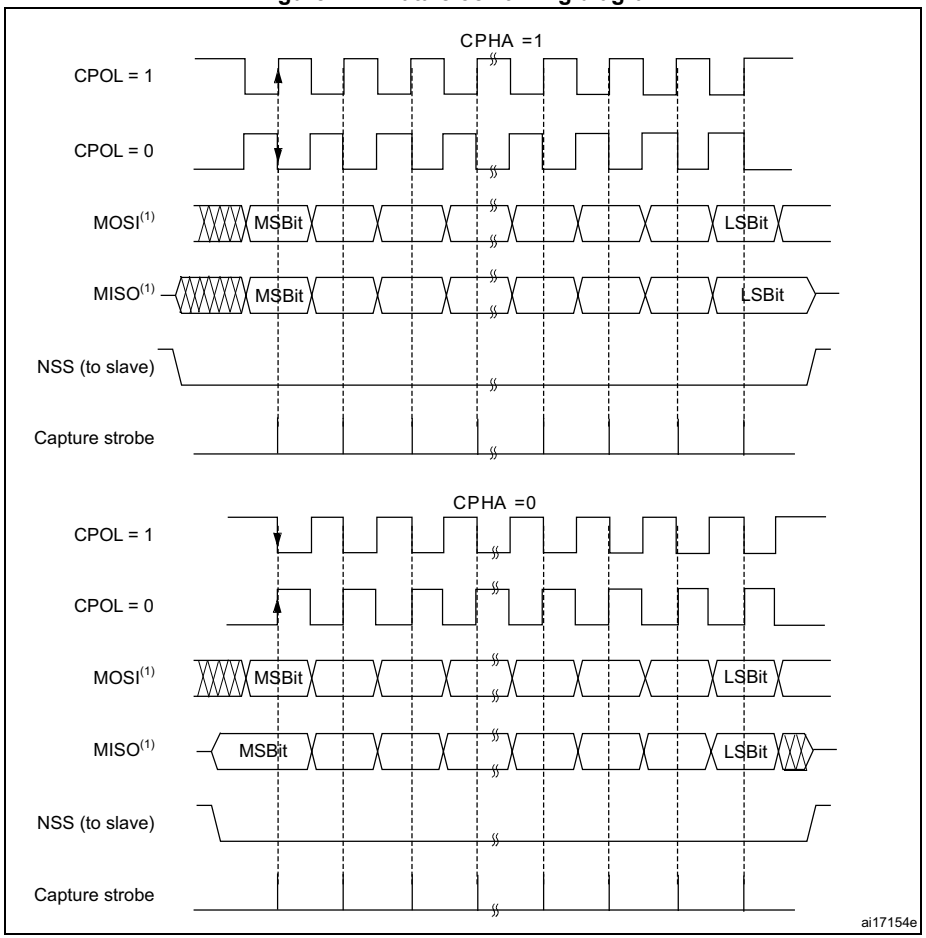
- Slave select is usually active low (labeled NSS or $\overline{SS}$)
  - Turns on device when 0, off when 1
- Turning off the slave device may save power
- Options
  - 1 slave devices: tie slave select active
  - 1 or more slave devices: use GPIO pins to turn desired device ON before SPI transaction, turn device back OFF afterward
  - 1 slave device: use SPI controller to automatically pulse the SS pin during a transaction

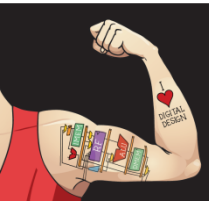


SPI Timing Diagram

- CPOL and CPHA flexible
- Easy to talk to devices

Figure 277. Data clock timing diagram

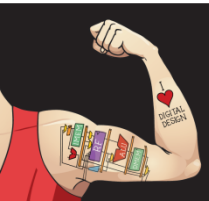




ELSEVIER

SPI Communication on STM32F0xx

- Enable SPI Clock with RCC APB2ENR bit 12: SPI1EN
  - This clock enables the peripheral, isn't the SCLK
- Assume slave select signal for device is tied active
- Set PA4, PA5, PA6 to ALT mode (10)
- Let CPOL and CPHA default to 0 with SPI1_CR1 bits 1:0
- Enable Master mode with SPI_CR1 bit 2
- Set baud rate with SPI1_CR1 bits 5:3 e.g. 010 for fclk/8
- Enable SPI port with SPI1_CR1 bit 6



Register Base Addresses

Table 1. STM32F0xx peripheral register boundary addresses

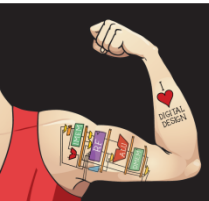
Bus	Boundary address	Size	Peripheral	Peripheral register map
	0xE000 0000 - 0xE00F FFFF	1MB	Cortex®-M0 internal peripherals	
	0x4800 1800 - 0x5FFF FFFF	~384 MB	Reserved	
AHB2	0x4800 1400 - 0x4800 17FF	1KB	GPIOF	*Section 8.4.12 on page 163*
	0x4800 1000 - 0x4800 13FF	1KB	GPIOE	*Section 8.4.12 on page 163*
	0x4800 0C00 - 0x4800 0FFF	1KB	GPIOD	*Section 8.4.12 on page 163*
	0x4800 0800 - 0x4800 0BFF	1KB	GPIOC	*Section 8.4.12 on page 163*
	0x4800 0400 - 0x4800 07FF	1KB	GPIOB	*Section 8.4.12 on page 163*
	0x4800 0000 - 0x4800 03FF	1KB	GPIOA	*Section 8.4.12 on page 163*
	0x4002 4400 - 0x47FF FFFF	~128 MB	Reserved	
AHB1	0x4002 4000 - 0x4002 43FF	1 KB	TSC	*Section 16.6.11 on page 318*
	0x4002 3400 - 0x4002 3FFF	3 KB	Reserved	
	0x4002 3000 - 0x4002 33FF	1 KB	CRC	*Section 12.5.6 on page 226*
	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved	
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH interface	*Section 3.5.9 on page 73*
	0x4002 1400 - 0x4002 1FFF	3 KB	Reserved	
	0x4002 1000 - 0x4002 13FF	1 KB	RCC	*Section 6.4.15 on page 135*
APB	0x4001 3000 - 0x4001 33FF	1 KB	SPI1/I2S1	*Section 28.9.10 on page 813*
	0x4001 2C00 - 0x4001 2FFF	1 KB	TIM1	*Section 17.4.21 on page 391*

- Base Addresses

RCC: 0x40021000

GPIOA: 0x48000000

SPI1: 0x40013000



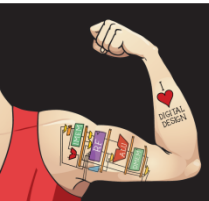
Enable SPI Clock in RCC

Table 19. RCC register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	RCC_CR	Res.	Res.	Res.	Res.	Res.	Res.	PLL RDY	PLL ON	Res.	Res.	Res.	Res.	CSSON	HSEBYP	HSERDY	HSEON	HSICAL[7:0]								HSITRIM[4:0]					Res.	HSIRDY	HSION
	Reset value							0	0					0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		1	1
0x04	RCC_CFGR	PLL NODIV	MCOPRE [2:0]			MCO [3:0]				Res.	Res.	PLLMUL[3:0]				PLLXTPRE	PLLSRC[1]	PLLSRC[0]	ADC PRE	Res.	Res.	Res.	PPRE [2:0]			HPRE[3:0]				SWS [1:0]		SW [1:0]	
	Reset value	0	0	0	0	0	0	0	0			0	0	0	0	0	0	0	0				0	0	0	0	0	0	0	0	0	0	0
0x08	RCC_CIR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CSSC	HSI48RDYC	HSI14 RDYC	PLLRDYC	HSERDYC	HSIRDYC	LSERDYC	LSIRDYC	Res.	HSI48RDYIE	HSI14 RDYIE	PLLRDYIE	HSERDYIE	HSIRDYIE	LSERDYIE	LSIRDYIE	CSSF	HSI48RDYF	HSI14 RDYF	PLLRDYF	HSERDYF	HSIRDYF	LSERDYF	LSIRDYF
	Reset value									0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	RCC_APB2RSTR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBGMCURST	Res.	Res.	Res.	TIM17RST	TIM16RST	TIM15RST	Res.	USART1RST	Res.	SPI1RST	TIM1RST	Res.	ADCRST	Res.	USART8RST	USART7RST	USART6RST	Res.	Res.	Res.	Res.	SYSCFGRST
	Reset value										0				0	0	0		0		0	0		0		0	0	0					0
0x010	RCC_APB1RSTR	Res.	CECRST	DACRST	PWRRST	CRSRST	Res.	CANRST	Res.	USBRST	I2C2RST	I2C1RST	USART5RST	USART4RST	USART3RST	USART2RST	Res.	Res.	SPI2RST	Res.	Res.	WWDGRST	Res.	Res.	TIM14RST	Res.	Res.	TM7RST	TM6RST	Res.	Res.	TIM3RST	Res.	TIM2RST
	Reset value		0	0	0	0		0		0	0	0	0	0	0	0			0			0			0			0	0			0	0
0x14	RCC_AHBENR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSCEN	Res.	IOPFEN	IOPEEN	USART4RST	IOPCEN	IOPBEN	IOPAEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CRCEN	Res.	FLITFEN	Res.	SRAMEN	DAM2EN	DMAEN
	Reset value								0		0	0	0	0	0	0											0		1		1	0	0
0x18	RCC_APB2ENR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBGMCUEN	Res.	Res.	Res.	TIM17 EN	TIM16 EN	TIM15 EN	Res.	USART1EN	Res.	SPI1EN	TIM1EN	Res.	ADCEN	Res.	USART8EN	USART7EN	USART6EN	Res.	Res.	Res.	Res.	SYSCFGCOMPEN
	Reset value										0				0	0	0		0		0	0		0		0	0	0					0

Bit 12 **SPI1EN:** SPI1 clock enable
Set and cleared by software.
0: SPI1 clock disabled
1: SPI1 clock enabled

- RCC_APB2ENR adr = 0x40021000 base + 0x18 offset
- SPI1EN is bit 12

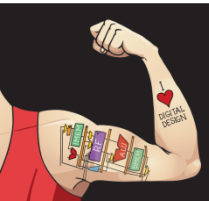


Set PA4, 5, 6 to ALT Mode

Table 24. GPIO register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**GPIOA_MODER**	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
	Reset value	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x00	**GPIOx_MODER** (where x = **B..F**)	MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]		MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- GPIOA_MODER adr = 0x48000000 base + 0x00 offset
- PA6, 5, 4 MODE bits are 13:12, 11:10, and 9:8



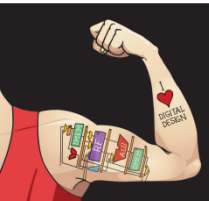
SPI Baud Rate, Polarity, Phase, Master

Table 115. SPI register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	SPIx_CR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BIDIMODE	BIDIOE	CRCEN	CRCNEXT	CRCL	RXONLY	SSM	SSI	LSBFIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	SPIx_CR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.		LDMA_TX	LDMA_RX	FRXTH	DS[3:0]				TXEIE	RXNEIE	ERRIE	FRF	NSSP	SSOE	TXDMAEN	RXDMAEN
	Reset value																		0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0x08	SPIx_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FTLVL[1:0]		FRLVL[1:0]		FRE	BSY	OVR	MODF	CRCERR	UDR	CHSIDE	TXE	RXNE
	Reset value																				0	0	0	0	0	0	0	0	0	0	1	0
0x0C	SPIx_DR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

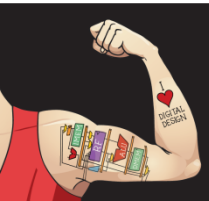
- Leave CR2 with default DS for 8-bit transfer, NSSP and SSOE for no automatic SS pulses

- SPI1_CR1 adr = 0x40013000 base + 0x00 offset
- SPI1_CR2 adr = 0x40013000 base + 0x04 offset
- SPI1_SR adr = 0x40013000 base + 0x08 offset
- SPI1_DR adr = 0x40013000 base + 0x0C offset



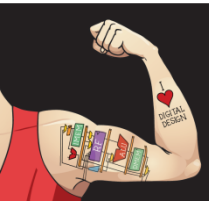
SPI Sample Code

```
void spiInit() {
        // 2.2.2 lists memory map for IO registers
        volatile unsigned long *GPIOA_MODER = (unsigned long*)0x48021000; // 8.4.1
        volatile unsigned long *SPI1_CR1 = (unsigned long*)0x40013000; // 28.9.1
        volatile unsigned long *RCC_APB2ENR = (unsigned long*)0x40021018 // 6.4.7

        *RCC_APB2ENR |= (1<<12);                    // set SPI1EN bit to activate SPI1
        *GPIOA_MODER |= (1<<9 | 1 << 11 | 1<<13); // ALT for PA4, 5, 6
        *SPI1_CR1 |= (1 << 4 | 1 << 2);             // Baud Rate = fclk/8; master mode
        *SPI1_CR1 |= 1 << 6;                        // turn on SPI
}

unsigned char SPI1SendReceive8(unsigned char outDat) {
        volatile unsigned long *SPI1_DR = (unsigned long*)0x4001300C; // 28.9.4
        volatile unsigned long *SPI1_SR = (unsigned long*)0x40013008; // 28.9.3

        *SPI1_DR = outDat;                          // Send a byte to the slave device
        while (!(*SPI1_SR & (1<<0)));               // wait for RXNE flag to clear
        return *SPI1_DR;                            // Get the byte back from slave device
}
```



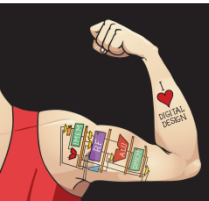
Analog/Digital Converter

- 12-bit A/D Converter
  - Reads analog input on scale of 0 – 3.3 V
  - Converts to digital value between 0 and 2^{12}-1
  - Hence the resolution is $3.3/2^{12}$ = 0.81 mV per step
- 16 Input channels
  - Can read from any of the 16 PA inputs
  - Also can read internal temperature and voltage sensors



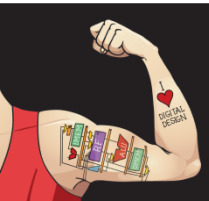
ADC Example

- 12-bit A/D Converter
  - Reads analog input on scale of 0 – 3.3 V
  - Converts to digital value between 0 and 2^{12}-1
  - Hence the resolution is $3.3/2^{12}$ = 0.81 mV per step
- 16 Input channels
  - Can read from any of the 16 PA inputs
  - Also can read internal temperature and voltage sensors



Using the ADC

- Initialize ADC
  - Enable ADC clock using the Reset/Clock Control register
  - Turn on ADC by setting the ADEN bit
  - Wait for ADRDY flag to set to indicate ADC ready to go
  - Write 111 to the SMP bits to run the sampling clock slowly
- To Convert Voltage on Channel n
  - Write a 1 to the CHSELn bit and 0s to the other CHSLE bits to choose channel n for conversion
  - Start conversion by setting the ADSTART bit
  - Wait for ADSTART bit to go low to indicate conversion complete
  - Read answer from the ADC_DR data register



Register Base Addresses

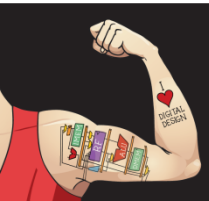
Table 1. STM32F0xx peripheral register boundary addresses (continued)

Bus	Boundary address	Size	Peripheral	Peripheral register map
APB	0x4001 5C00 - 0x4001 7FFF	9 KB	Reserved	
	0x4001 5800 - 0x4001 5BFF	1 KB	DBGMCU	*Section 32.9.6 on page 924*
	0x4001 4C00 - 0x4001 57FF	3 KB	Reserved	
	0x4001 4800 - 0x4001 4BFF	1 KB	TIM17	*Section 20.6.16 on page 545*
	0x4001 4400 - 0x4001 47FF	1 KB	TIM16	*Section 20.6.16 on page 545*
	0x4001 4000 - 0x4001 43FF	1 KB	TIM15	*Section 20.5.18 on page 528*
	0x4001 3C00 - 0x4001 3FFF	1 KB	Reserved	
	0x4001 3800 - 0x4001 3BFF	1 KB	USART1	*Section 27.8.12 on page 753*
	0x4001 3400 - 0x4001 37FF	1 KB	Reserved	
	0x4001 3000 - 0x4001 33FF	1 KB	SPI1/I2S1	*Section 28.9.10 on page 813*
	0x4001 2C00 - 0x4001 2FFF	1 KB	TIM1	*Section 17.4.21 on page 391*
	0x4001 2800 - 0x4001 2BFF	1 KB	Reserved	
	0x4001 2400 - 0x4001 27FF	1 KB	ADC	*Section 13.12.11 on page 267*
	0x4001 2000 - 0x4001 23FF	1 KB	Reserved	
	0x4001 1C00 -0x4001 1FFF	1 KB	USART8	*Section 27.8.12 on page 753*
	0x4001 1800 - 0x4001 1BFF	1 KB	USART7	*Section 27.8.12 on page 753*
	0x4001 1400 - 0x4001 17FF	1 KB	USART6	*Section 27.8.12 on page 753*
	0x4001 0800 - 0x4001 13FF	3 KB	Reserved	
	0x4001 0400 - 0x4001 07FF	1 KB	EXTI	*Section 11.3.7 on page 219*
	0x4001 0000 - 0x4001 03FF	1 KB	SYSCFG COMP	*Section 9.1.38 on page 185* *Section 15.5.2 on page 300*
	0x4000 8000 - 0x4000 FFFF	32 KB	Reserved	

ADC Register Map

Table 50. ADC register map and reset values

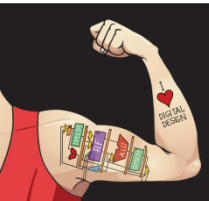
Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	ADC_ISR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	AWD	Res.	Res.	OVR	EOSEQ	EOC	EOSMP	ADRDY
	Reset value																									0			0	0	0	0	0
0x04	ADC_IER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	AWDIE	Res.	Res.	OVRIE	EOSEQIE	EOCIE	EOSMPIE	ADRDYIE
	Reset value																									0			0	0	0	0	0
0x08	ADC_CR	ADCAL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADSTP	Res.	ADSTART	ADDIS	ADEN
	Reset value	0																											0		0	0	0
0x0C	ADC_CFGR1	Res.	AWDCH[4:0]						Res.	AWDEN	AWDSGL						DISCEN	AUTOFF	WAIT	CONT	OVRMOD	EXTEN[1:0]		Res.	EXTSEL [2:0]			ALIGN	RES [1:0]		SCANDIR	DMACFG	DMAEN
	Reset value		0	0	0	0	0			0	0						0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0
0x10	ADC_CFGR2	CKMODE[1:0]		Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value	0	0																														
0x14	ADC_SMPR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SMP [2:0]		
	Reset value																														0	0	0
0x18	Reserved	Reserved																															
0x1C	Reserved	Reserved																															
0x20	ADC_TR	Res.	Res.	Res.	Res.	HT[11:0]												Res.	Res.	Res.	Res.	LT[11:0]												
	Reset value					1	1	1	1	1	1	1	1	1	1	1	1					0	0	0	0	0	0	0	0	0	0	0	0
0x24	Reserved	Reserved																															
0x28	ADC_CHSELR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CHSEL18	CHSEL17	CHSEL16	CHSEL15	CHSEL14	CHSEL13	CHSEL12	CHSEL11	CHSEL10	CHSEL9	CHSEL8	CHSEL7	CHSEL6	CHSEL5	CHSEL4	CHSEL3	CHSEL2	CHSEL1	CHSEL0
	Reset value														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C 0x30 0x34 0x38 0x3C	Reserved	Reserved																															
0x40	ADC_DR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DATA[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x44 – 0x304	Reserved	Reserved																															
0x308	ADC_CCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	VBATEN	TSEN	VREFEN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	Reset value								0	0	0																						



ELSEVIER

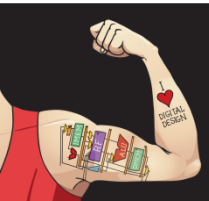
Using the ADC

- Initialize ADC
  - Enable ADC clock using the Reset/Clock Control register
  - Turn on ADC by setting the ADEN bit
  - Wait for ADRDY flag to set to indicate ADC ready to go
  - Write 111 to the SMP bits to run the sampling clock slowly
- To Convert Voltage on Channel n
  - Write a 1 to the CHSELn bit and 0s to the other CHSLE bits to choose channel n for conversion
  - Start conversion by setting the ADSTART bit
  - Wait for ADSTART bit to go low to indicate conversion complete
  - Read answer from the ADC_DR data register



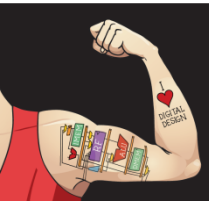
Initialize ADC

```
void adcInit(void) {
        // declare registers
        volatile unsigned long *RCC_APB2ENR = (unsigned long*)0x40021018;




        // turn on ADC clock by writing 1 to ADCEN bit 9 of RCC_APB2ENR
        *RCC_APB2ENR |= 1 << 9;

        // your code here
        // turn on ADC with ADEN bit


        // wait for ADRDY bit for ADC ready



        // Write 111 to SMP bits to set speed

}
```



Initialize ADC: Solution

```
void adcInit(void) {
        volatile unsigned long *RCC_APB2ENR = (unsigned long*)0x40021018;
        volatile unsigned long *ADC_CR =      (unsigned long*)0x40012408;
        volatile unsigned long *ADC_ISR =     (unsigned long*)0x40012400;
        volatile unsigned long *ADC_SMPR =    (unsigned long*)0x40012414;

        // Enable ADC clock by writing 1 to ADCEN bit of RCC_APB2ENR
        *RCC_APB2ENR |= 1<<9;
        *ADC_CR |= 1<<0;          // enable ADC with ADEN bit
        while (!(ADC_ISR & 1)); // wait for ADRDY saying ADC is ready
        *ADC_SMPR |= 0b111;       // set slow samping rate
}
```



Read ADC

```
int analogRead(int channel) {
        // Define register addresses




        // Select which channel to convert


        // Start conversion with ADSTART bit

        // Wait for ADSTART to become 0 to indicate conversion complete

        // Read answer from ADC_DR


}
```

Read ADC: Solution

```
int analogRead(int channel) {
        volatile unsigned long *ADC_CHSELR = (unsigned long*)0x40012428;
        volatile unsigned long *ADC_CR =     (unsigned long*)0x40012408;
        volatile unsigned long *ADC_DR =     (unsigned long*)0x40012440;

        *ADC_CHSELR = 1<<channel; // write appropriate channel select bit
        *ADC_CR |= 1<<2;          // write ADSTART bit to begin conversion
        while((*ADC_CR & (1<<2)); // wait until ADSTART bit falls
        return *ADC_DR;
}
```