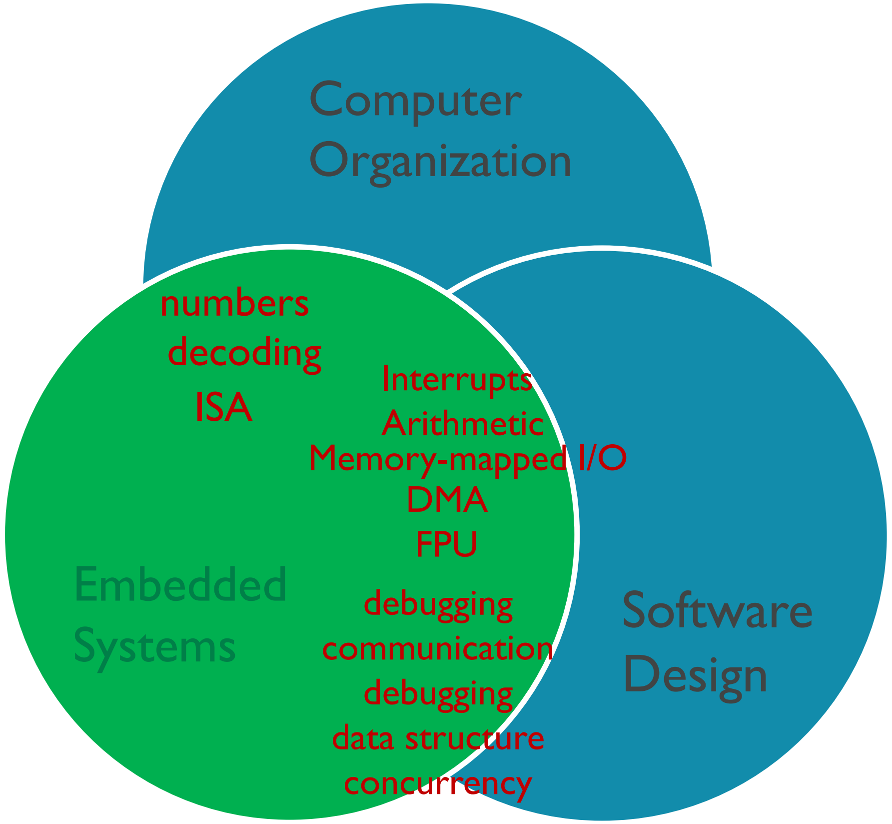
Education of Embedded Systems Programming in C and Assembly Based on ARM's Cortex-M Microprocessors

Yifeng Zhu, Libby Professor
University of Maine

Webinar Series
October 2018

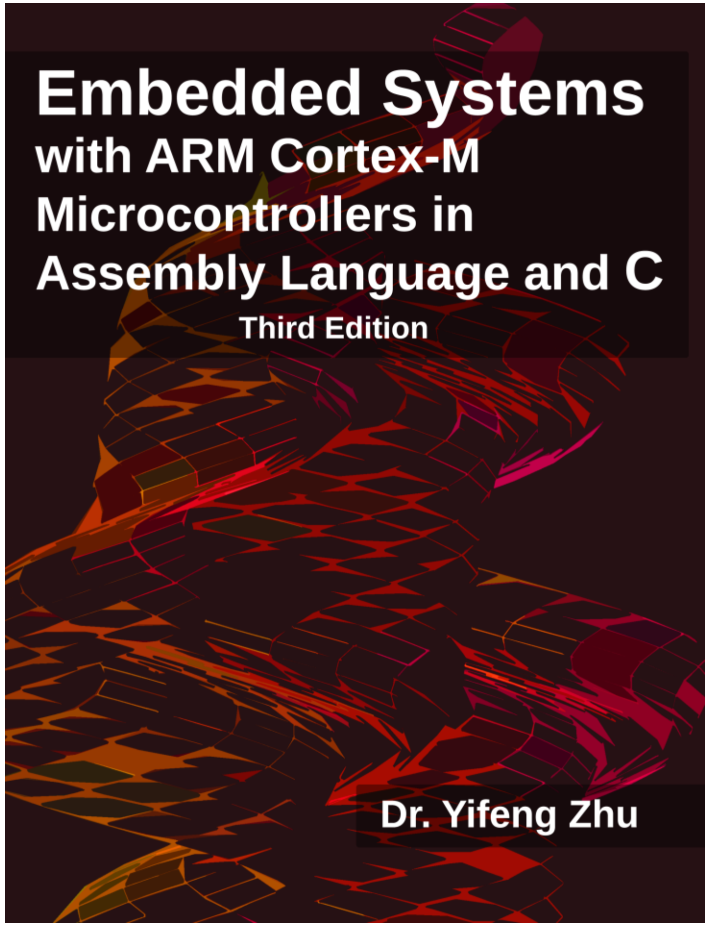
Role of Embedded Systems: Lays foundation



- Laying foundation in curriculum:
  - Computer organization & architecture
  - Operating systems
  - Software design & algorithms
  - Senior project design

- Body of Knowledge (IEEE/ACM Computer Engineering Curricula 2016)
  - Number systems and data encoding
  - Instruction set architecture
  - Relevant tools, standards and/or engineering constraints
  - Input/output interfacing and communication
  - Interrupts, timers, waveform generation
  - Implementation strategies for complex embedded systems
  - Computing platforms for embedded systems

Textbook



738 pages, $69.50

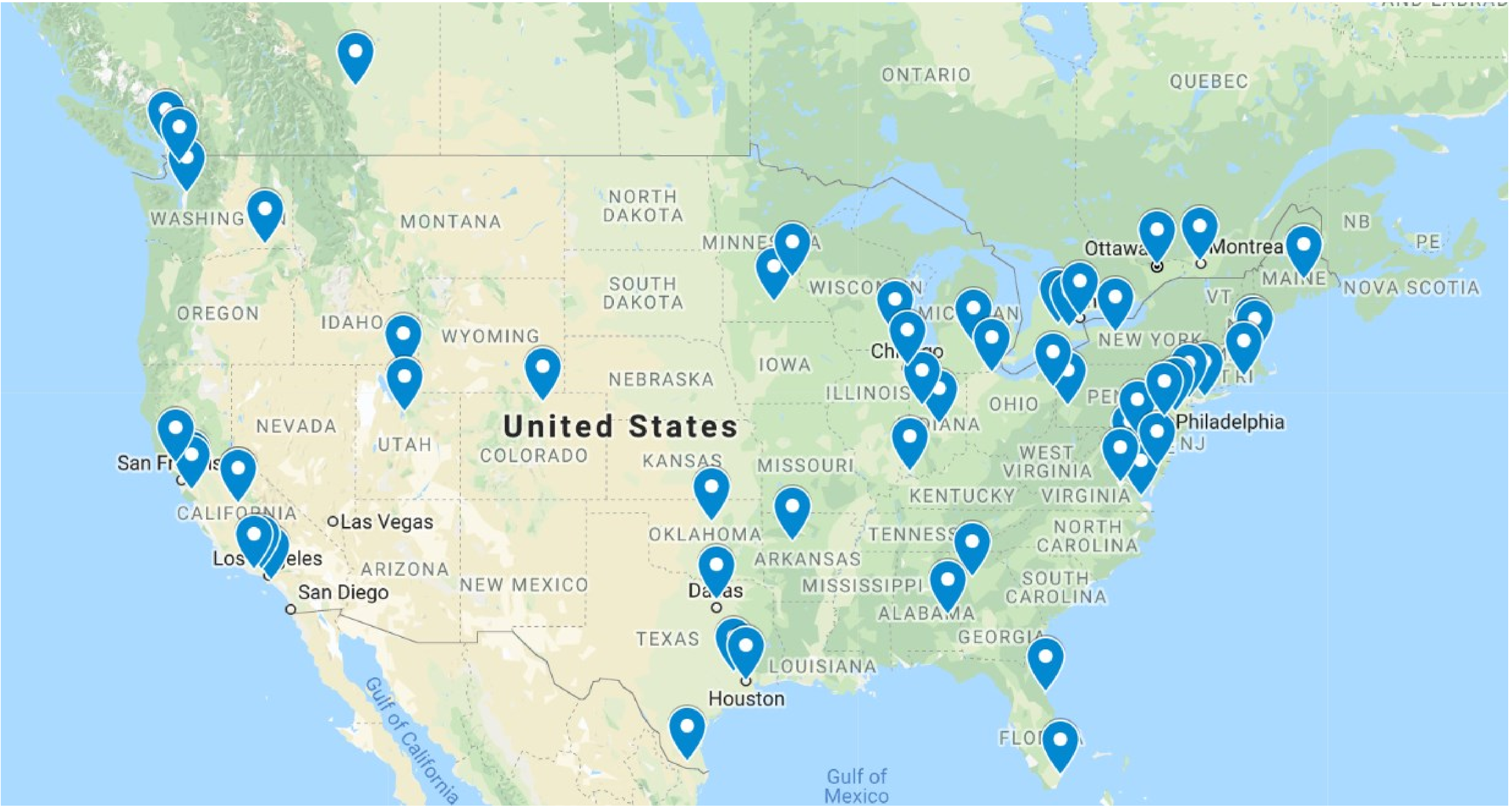
#1 Best Seller in Assembly Language Programming

#1 Best Seller in Computer Hardware Embedded Systems

1. See a program running
2. Data representation
3. ARM instruction set architecture
4. Arithmetic and logic
5. Load and store
6. Branch and conditional execution
7. Structured programming
8. Subroutines
9. 64-bit data processing
10. Mixing C and assembly
11. Interrupt
12. Fixed-point & floating-point arithmetic
13. Instruction encoding and decoding
14. General-purpose I/O
15. General-purpose timers
16. Stepper motor control
17. Liquid-crystal display (LCD)
18. Real-time clock (RTC)
19. Direct memory access (DMA)
20. Analog-to-digital converter (ADC)
21. Digital-to-analog converter (DAC)
22. Serial communication protocols
23. Multitasking
24. Digital signal processing

- Complete instructor's resource:
  - Lecture slides, quizzes and exams, tutorials, lab handouts and solutions (pre-lab, in-lab, and post-lab), solutions to end-of-chapter exercises
- Bare-metal programming at the register level without using any API libraries
- Line-by-line translation from C to ARM assembly
- Strike the balance between theoretical foundations and technical practices
- Using flowcharts as a reading guide for processor datasheets
- Online YouTube tutorials (received over 866,000 minutes of watch time)
- Adopted by over 80 universities

Adopted by universities in US & Canada



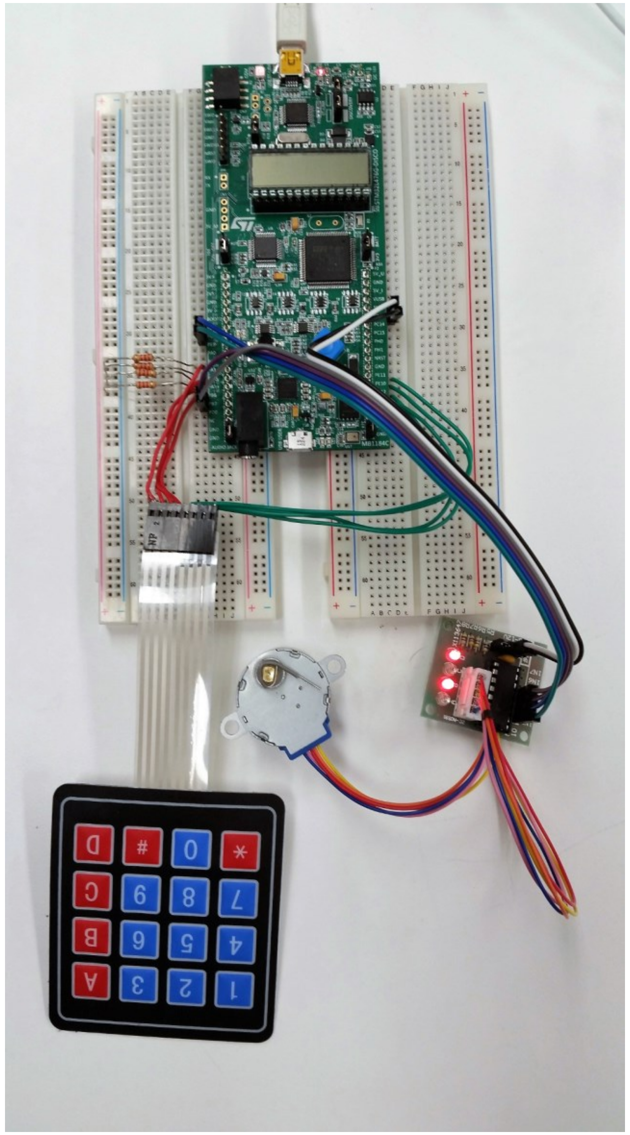
My approach of teaching

1. Using modern platforms and tools
2. Bare-metal programming
3. Structured programming in Assembly
4. Lab-centered learning
5. Online tutorials

My approach of teaching

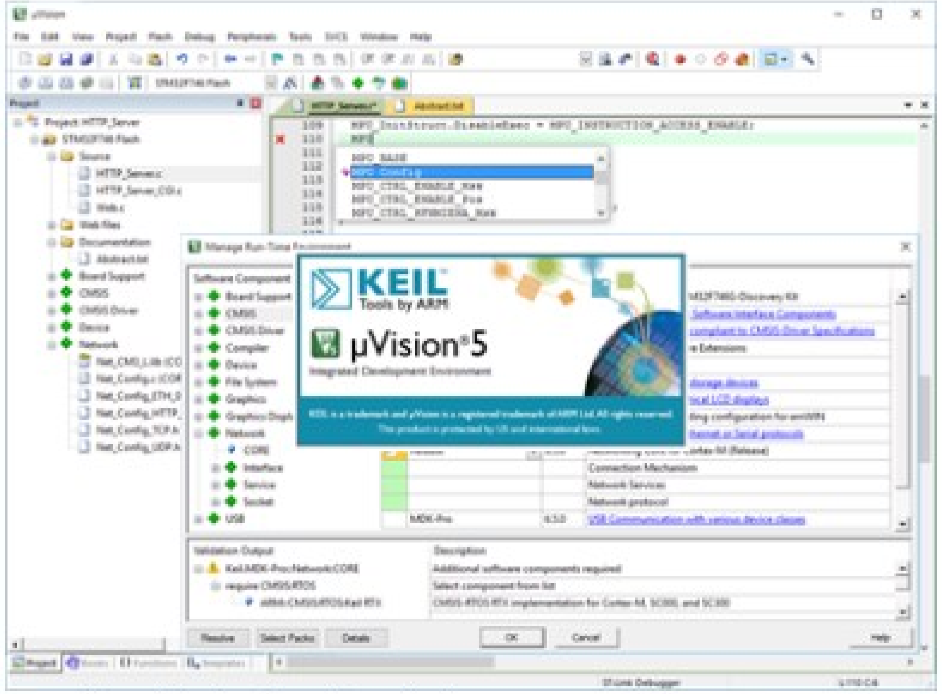
1. Using modern platforms and tools
2. Bare-metal programming
3. Structured programming in Assembly
4. Lab-centered learning
5. Online tutorials

Cheap and engaging platform and tools



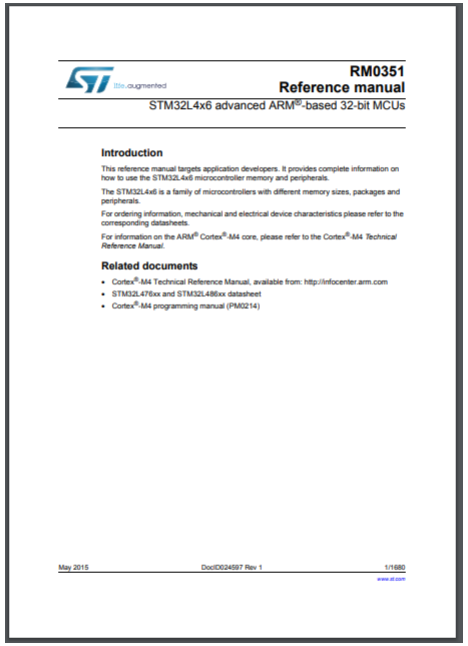
Lab in a box, $25

Friendly & robust IDE



free

Reference manual & datasheet



RM0351
Reference manual
STM32L4x6 advanced ARM®-based 32-bit MCUs

Introduction

This reference manual targets application developers. It provides complete information on how to use the STM32L4x6 microcontroller memory and peripherals.

The STM32L4x6 is a family of microcontrollers with different memory sizes, packages and peripherals.

For ordering information, mechanical and electrical device characteristics please refer to the corresponding datasheets.

For information on the ARM® Cortex®-M4 core, please refer to the Cortex®-M4 *Technical Reference Manual*.

Related documents

- Cortex®-M4 Technical Reference Manual, available from: http://infocenter.arm.com
- STM32L476xx and STM32L486xx datasheet
- Cortex®-M4 programming manual (PM0214)

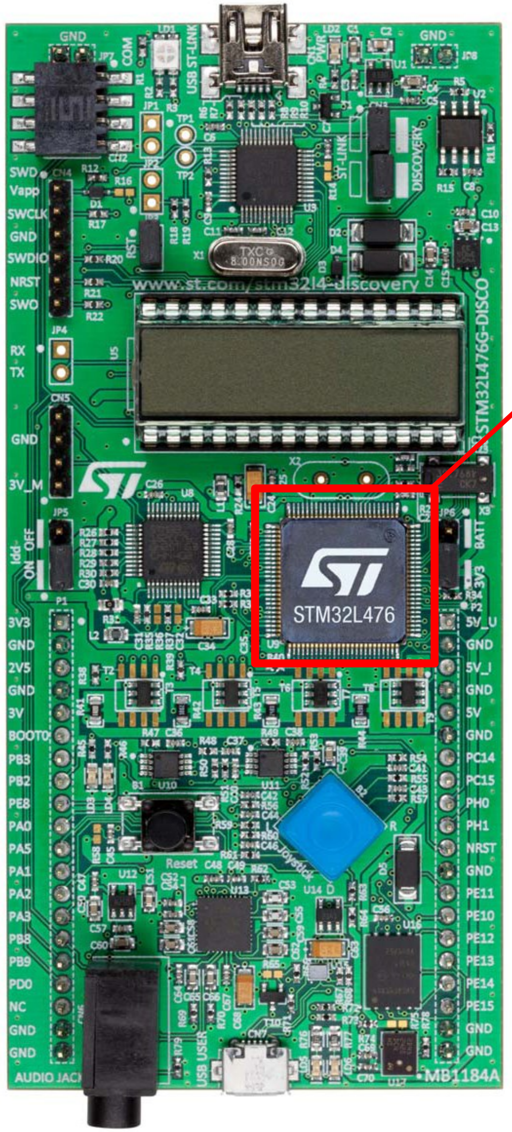
May 2015 DocID024597 Rev 1 1/1680

www.st.com

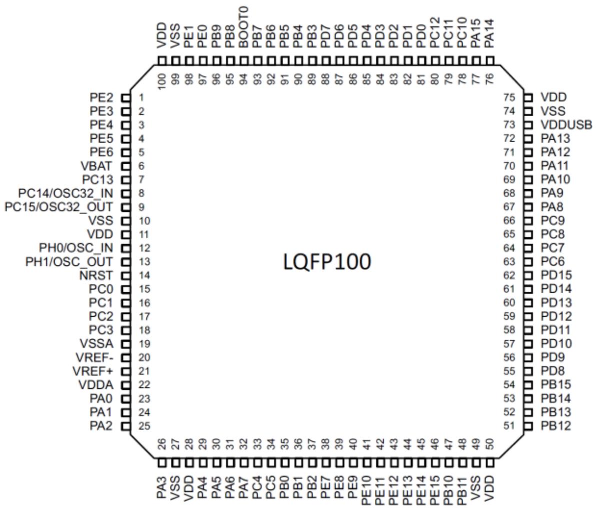
free

Selecting a Platform: Hardware Component

- Low cost
  - ~$25 each
- Hands-on experiences
  - develop and test real systems
- Rewarding and engaging
  - immediately enjoy the fruit of labor
- Convenient
  - mobile lab without time and location constrains
- Versatile
  - pins are extended for easy access



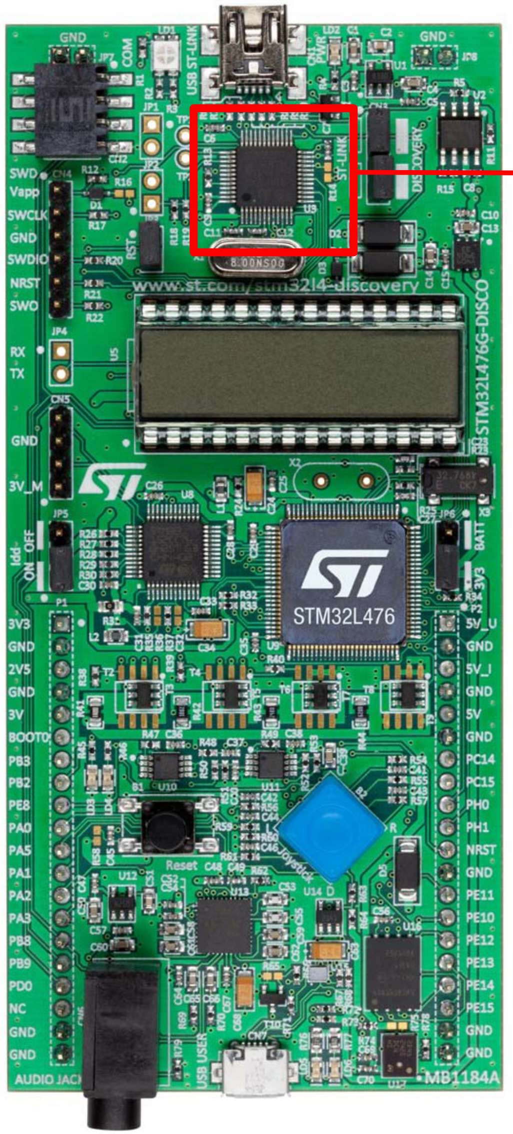
STM32L476G
ARM Cortex-M4F



STM32L4 Discovery Kit @STMicroelectronics

Selecting a Platform: Hardware Component

- Low cost
  - ~$25 each
- Hands-on experiences
  - develop and test real systems
- Rewarding and engaging
  - immediately enjoy the fruit of labor
- Convenient
  - mobile lab without time and location constrains
- Versatile
  - pins are extended for easy access



Integrated ST-Link/V2 programming and debugging tool

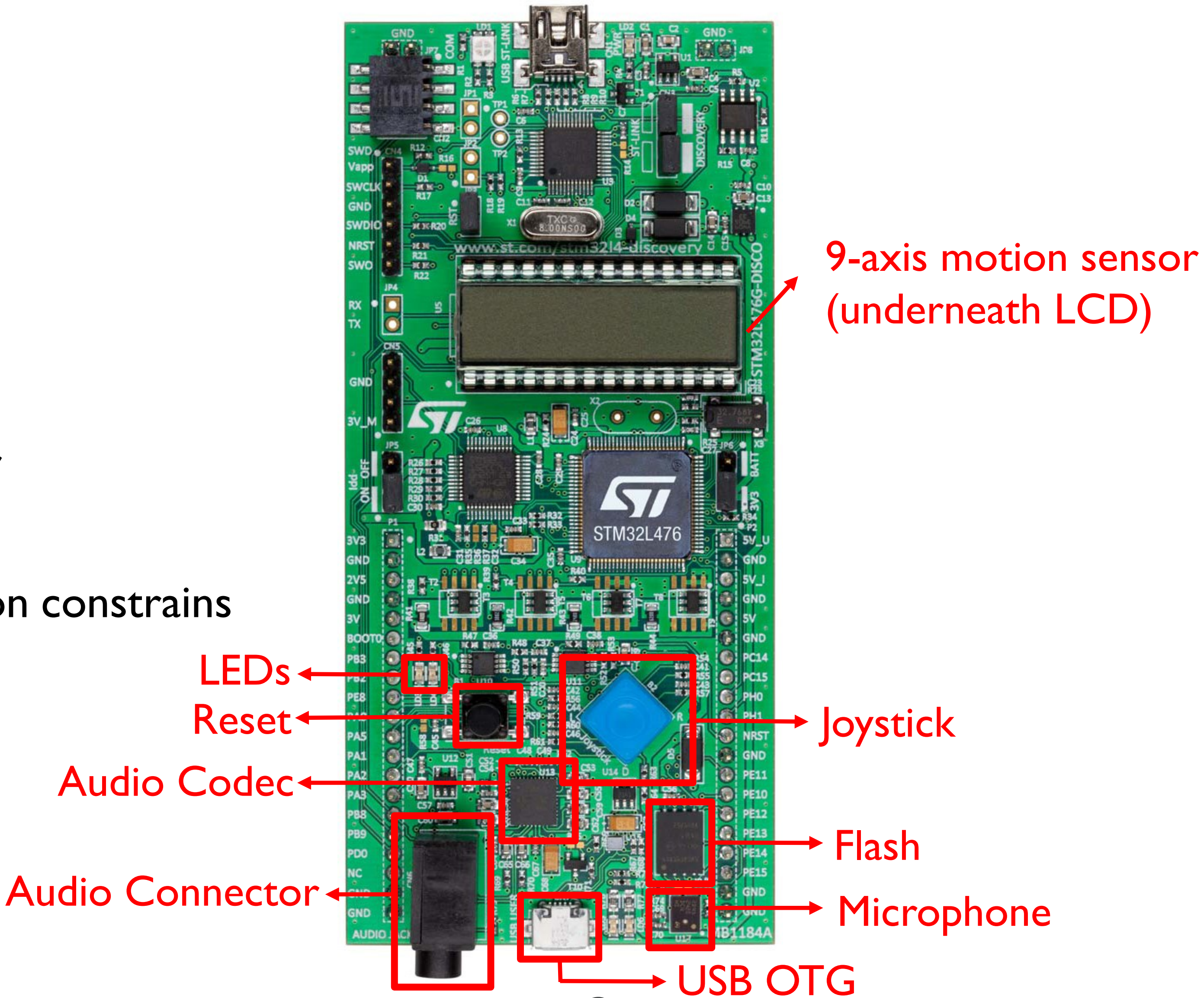
Type A

Mini B

STM32L4 Discovery Kit @STMicroelectronics

Selecting a Platform: Hardware Component

- Low cost
  - ~$25 each
- Hands-on experiences
  - develop and test real systems
- Rewarding and engaging
  - immediately enjoy the fruit of labor
- Convenient
  - mobile lab without time and location constrains
- Versatile
  - pins are extended for easy access



STM32L4 Discovery Kit @STMicroelectronics

DISCO-L476VG

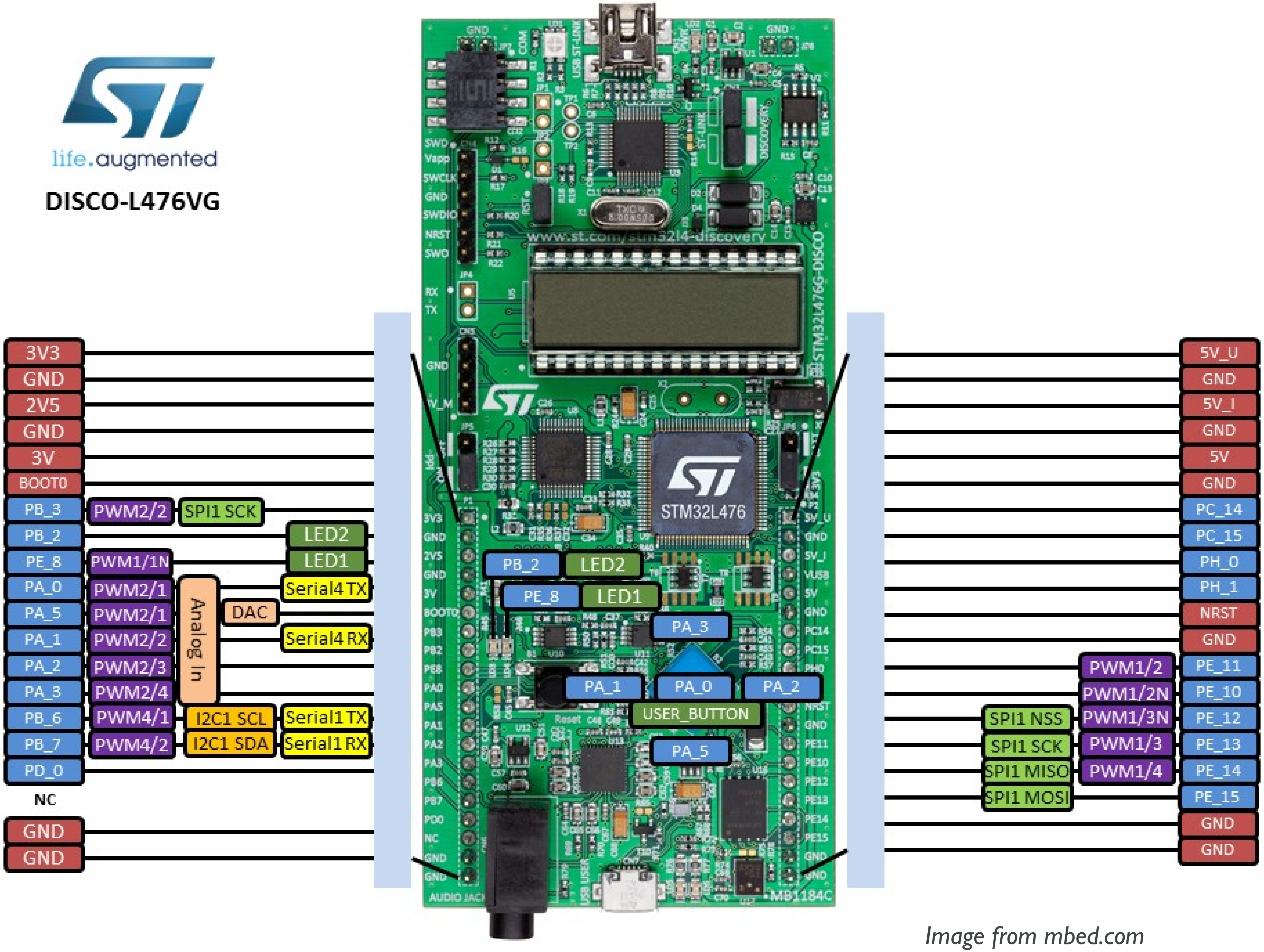
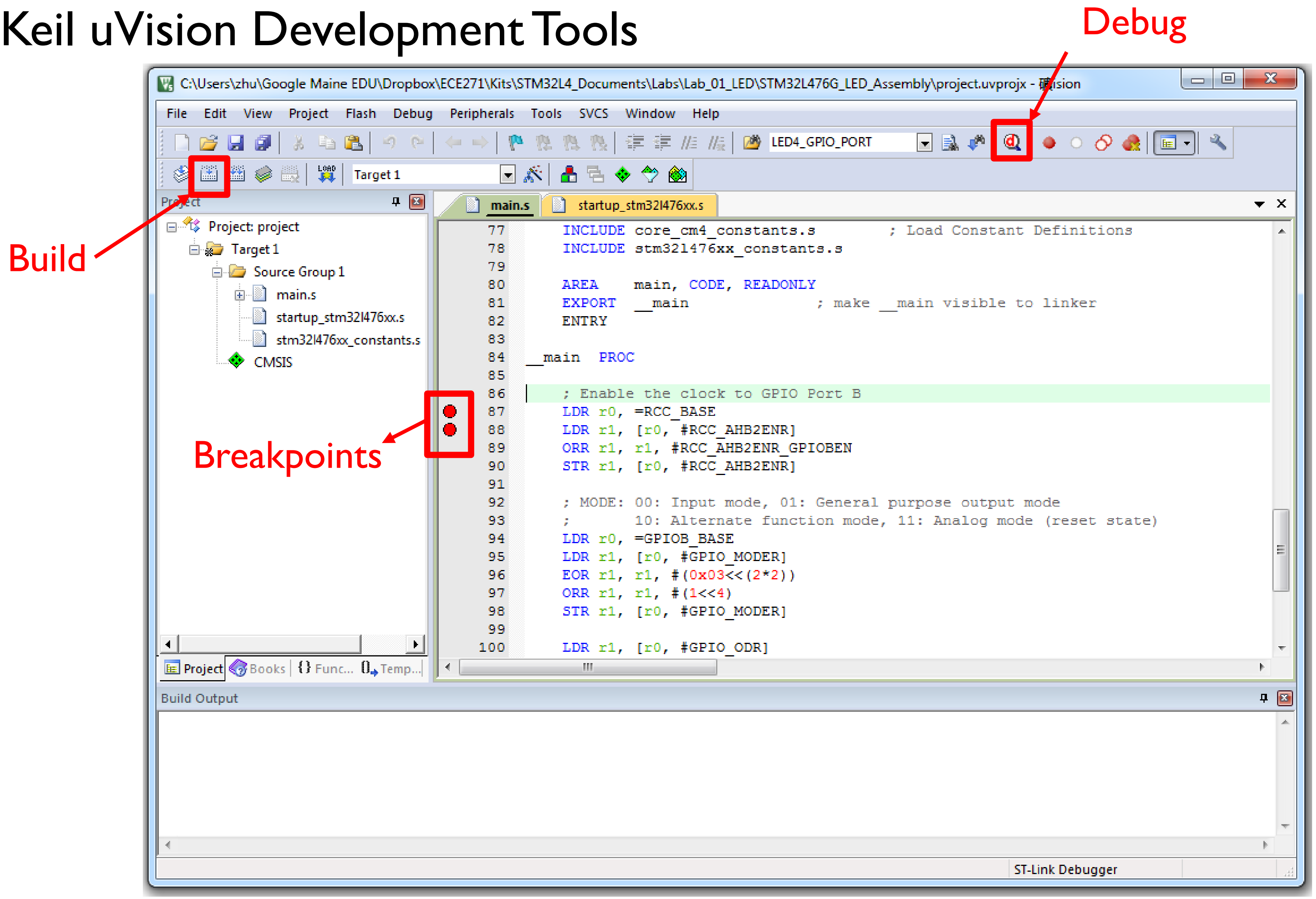
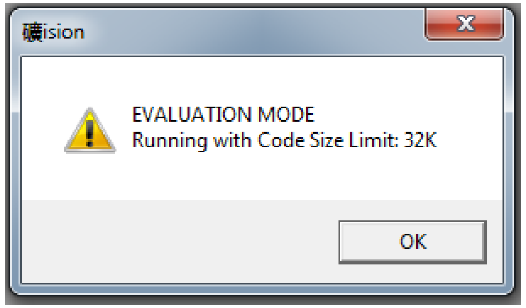


Image from mbed.com

Selecting a Platform: Software Component

- Keil uVision Development Tools

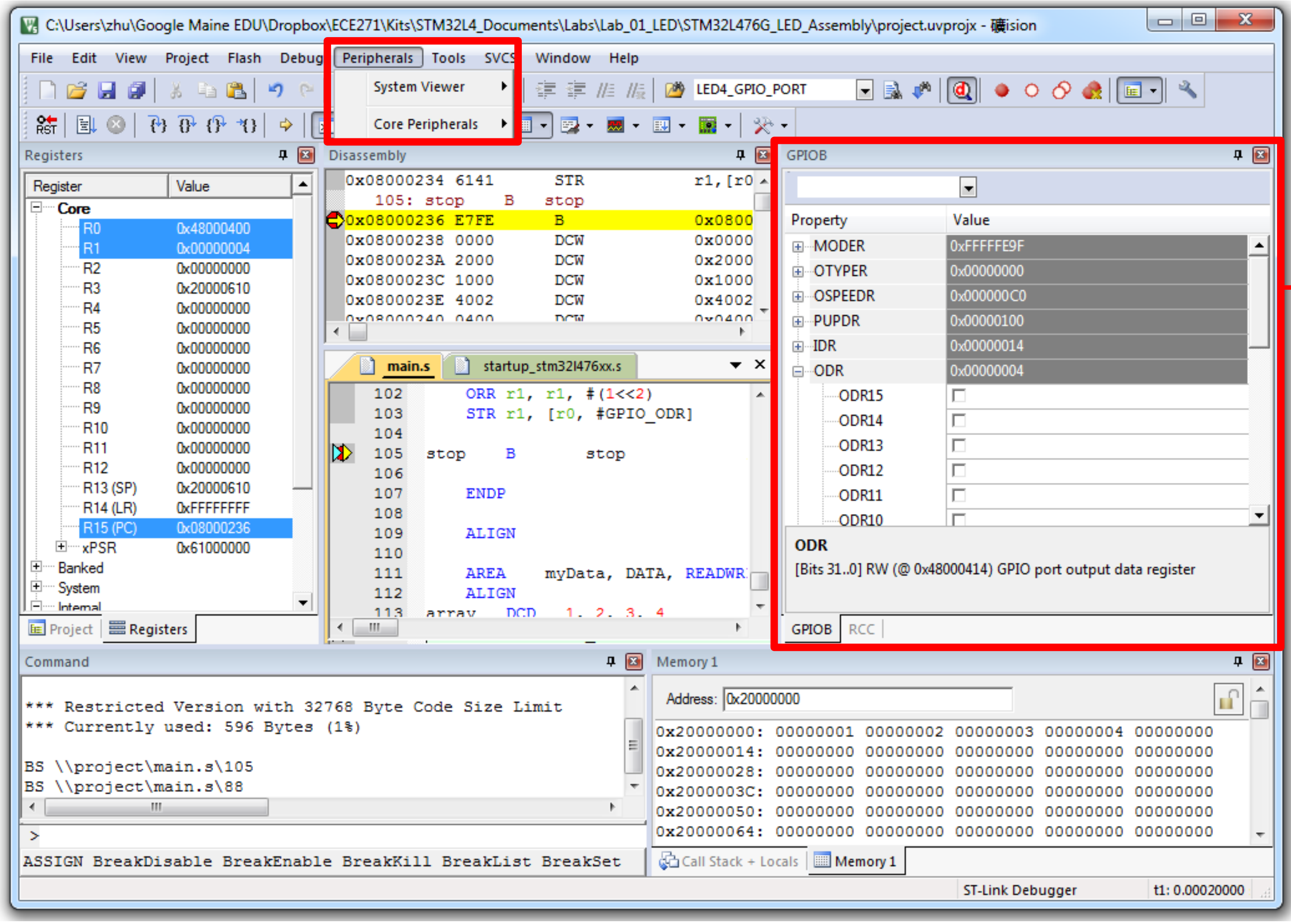




But this has not been a problem.

Selecting a Platform: Software Component

- Keil uVision Development Tools

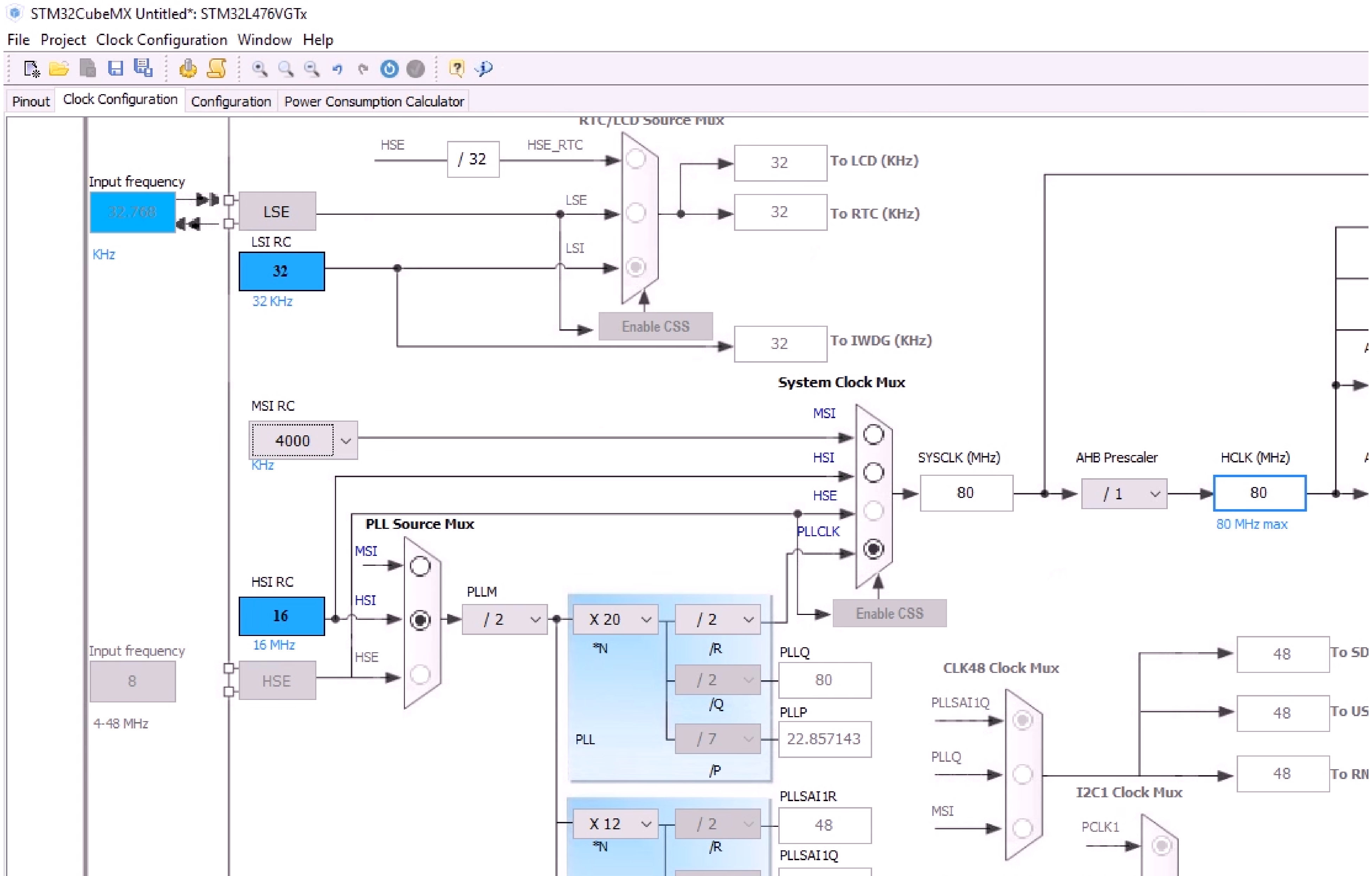


Monitor or modify peripheral registers

Students found this very helpful!

Free version limited the code size to 32 KB. But this has not been a problem.

STM32Cube



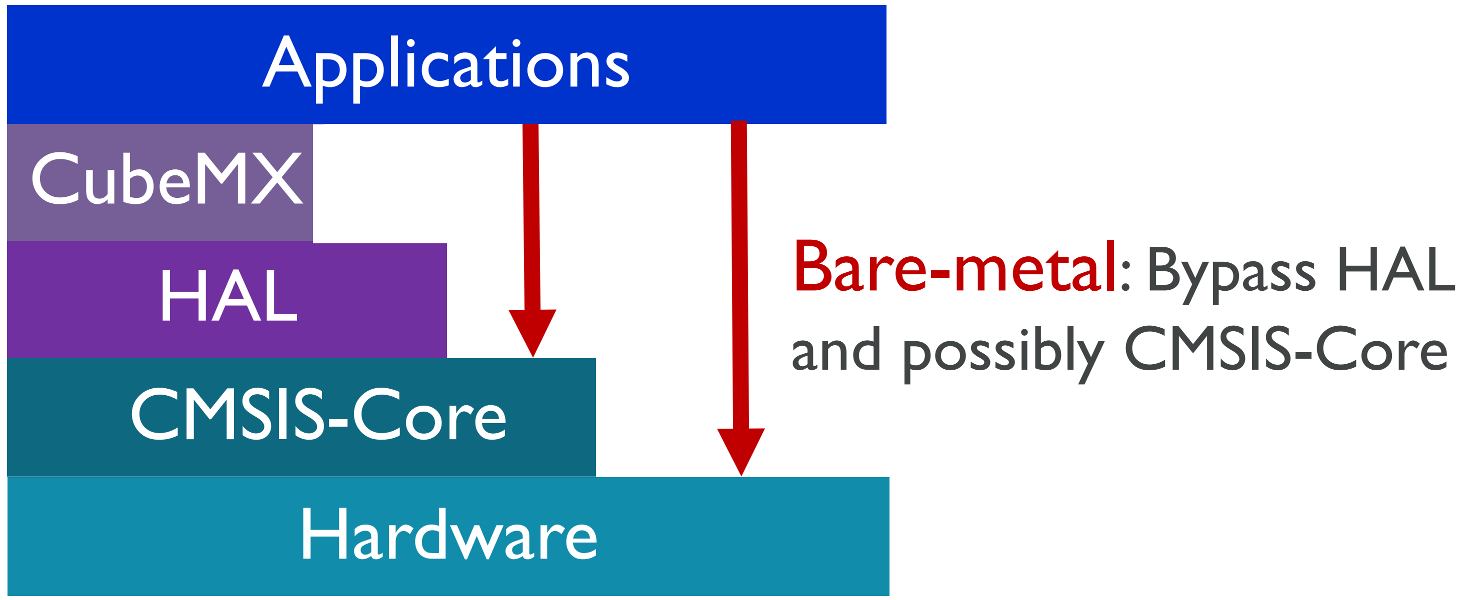
Nice clock tree visualization

My approach of teaching

1. Using modern platforms and tools
2. Bare-metal programming
3. Structured programming in Assembly
4. Lab-centered learning
5. Online tutorials

Teach at which level?

- Visual wizard tools (such as STMCubeMX)
- HAL (**H**ardware **A**bstraction **L**ayer) libraries
- Bare-metal



HAL Level

```
; Initialize the Red LED pin (PB.2)
static GPIO_InitTypeDef  GPIO_InitStruct;
GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull  = GPIO_PULLUP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Pin   = GPIO_PIN_2;

HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

HAL_GPIO_TogglePin(LED4_GPIO_PORT, LED4_PIN);
```

- Pros
  - Simplify implementation
  - Better portability
  - Many examples
- Cons
  - Very complex to understand
  - Cannot meet students' curiosity

```
void HAL_GPIO_Init(GPIO_TypeDef  *GPIOx, GPIO_InitTypeDef *GPIO_Init){
  uint32_t position = 0x00;
  uint32_t iocurrent = 0x00;
  uint32_t temp = 0x00;
  ...
}
```

130 lines

Bare-Metal Level in C

```
#define LED_PIN 2

// GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
GPIOB->MODER &= ~(3<<(2*LED_PIN));    // Clear by using mask
GPIOB->MODER |=   1<<(2*LED_PIN);     // Set as Output

// GPIO Speed: Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
GPIOB->OSPEEDR &= ~(3<<(2*LED_PIN)); // Clear by using mask
GPIOB->OSPEEDR |=   2<<(2*LED_PIN);  // Fast speed

// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
GPIOB->OTYPER &= ~(1<<LED_PIN);       // Push-pull

// GPIO Push-Pull: No pull-up pull-down (00), Pull-up (01), Pull-down (10),
Reserved (11)
GPIOB->PUPDR &= ~(3<<(2*LED_PIN));  // No pull-up, no pull-down

// Toggle up the LED
GPIOB->ODR ^= 1 << LED_PIN;
```

- Only 6 lines of code
- Focus on directly interfacing with hardware.
- Do not use any libraries!

Bare-Metal Level in Assembly

Bare-metal level programming helps learning assembly programming

Set Pin B.2 as GPIO output

```
#define LED_PIN 2

// GPIO Mode: Input(00), Output(01), AlterFunc(10), Analog(11, reset)
GPIOB->MODER &= ~(3<<(2*LED_PIN));
GPIOB->MODER |=   1<<(2*LED_PIN);     // Output(01)
```

C implementation

Translate naturally

```
LED_PIN EQU 2
    LDR r0, =GPIOB_BASE
    LDR r1, [r0, #GPIO_MODER]
    EOR r1, r1, #(0x03<<(2*LED_PIN))
    ORR r1, r1, #(1<<LED_PIN)
    STR r1, [r0, #GPIO_MODER]
```

Assembly implementation

My approach of teaching

1. Using modern platforms and tools
2. Bare-metal programming
3. Structured programming in Assembly
4. Lab-centered learning
5. Online tutorials

A Structured Approach in Assembly Programming

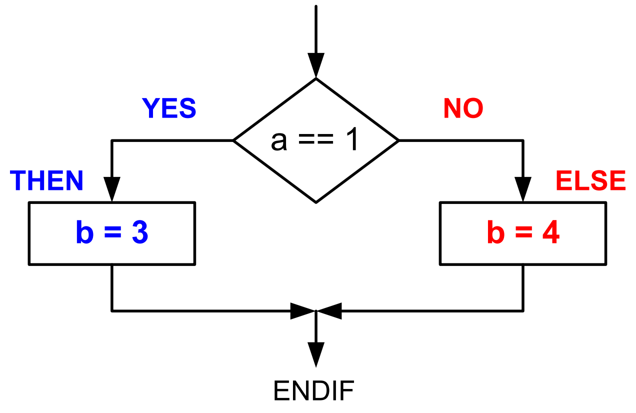
- Assembly is not a structured programming language
  - No high-level control constructs to avoid GOTOs (unconditional branches)
  - Difficulty to learn and program
  - Prone to create spaghetti codes

- My approaches
  - Using flowcharts
  - Leveraging C programs

A Structured Approach in Assembly Programming

Methods of teaching structured programming in assembly

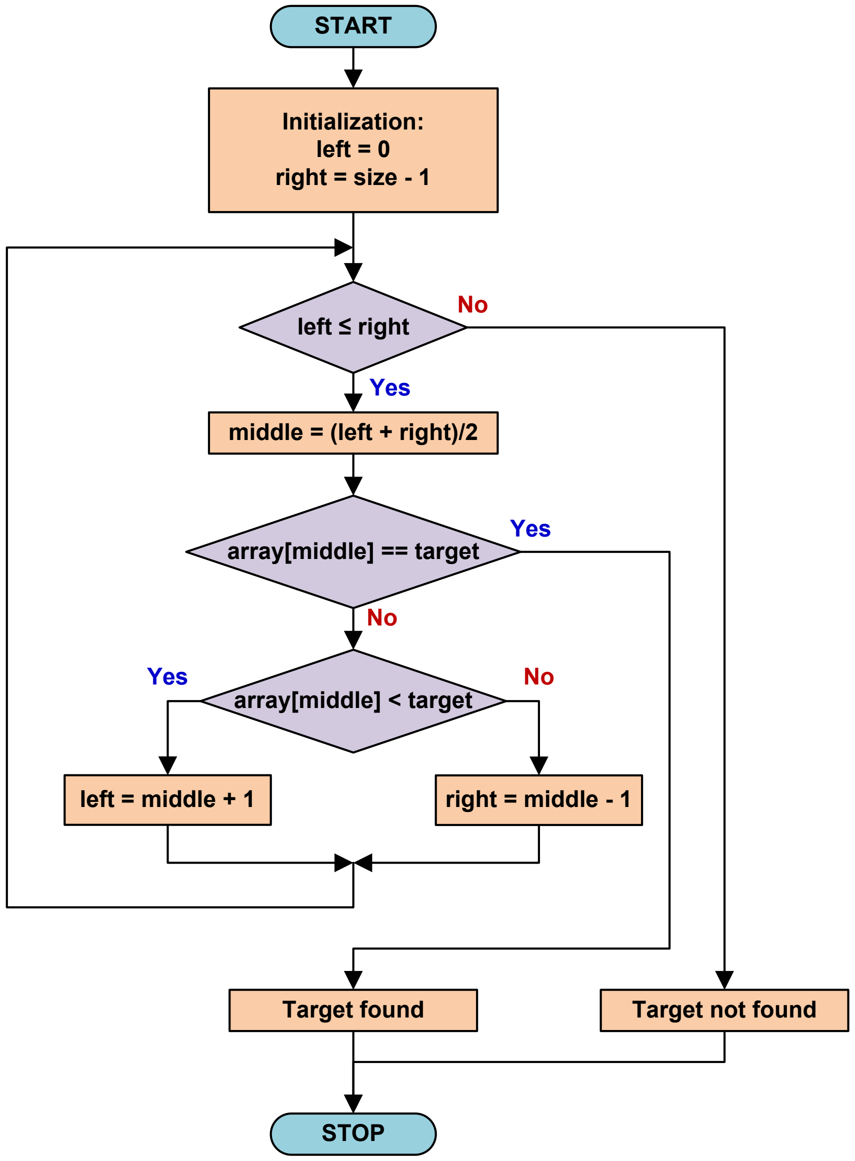
- Using flowcharts
  - Separate program structuring from code writing



A Structured Approach in Assembly Programming

Methods of teaching structured programming in assembly

- Using flowcharts
  - Separate program structuring from code writing

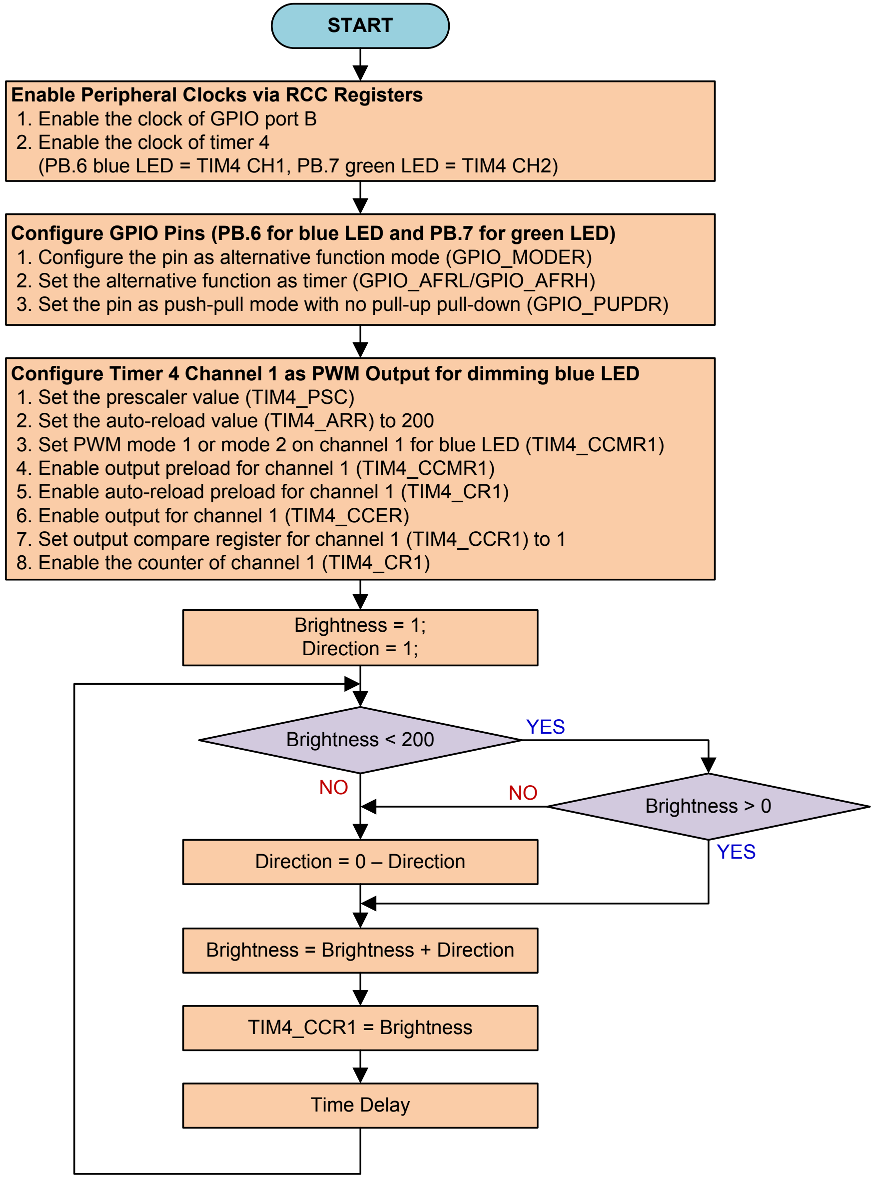


Flowchart of Binary Search

A Structured Approach in Assembly Programming

Methods of teaching structured programming in assembly

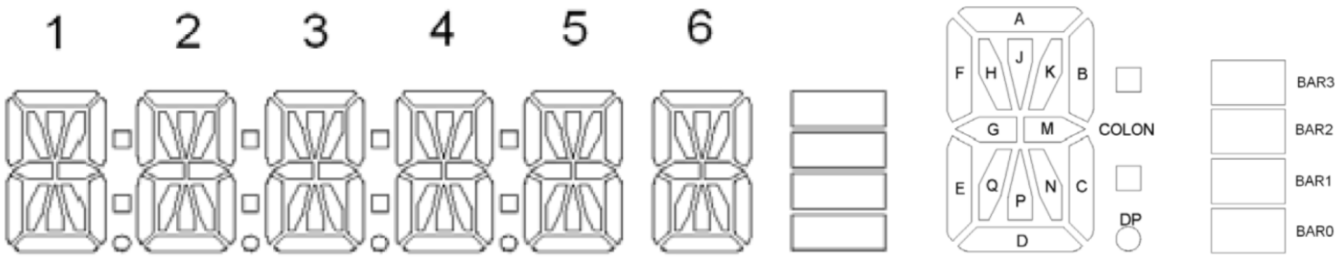
- Using flowcharts
  - Separate program structuring from code writing



Diming LED by using timer PWM output

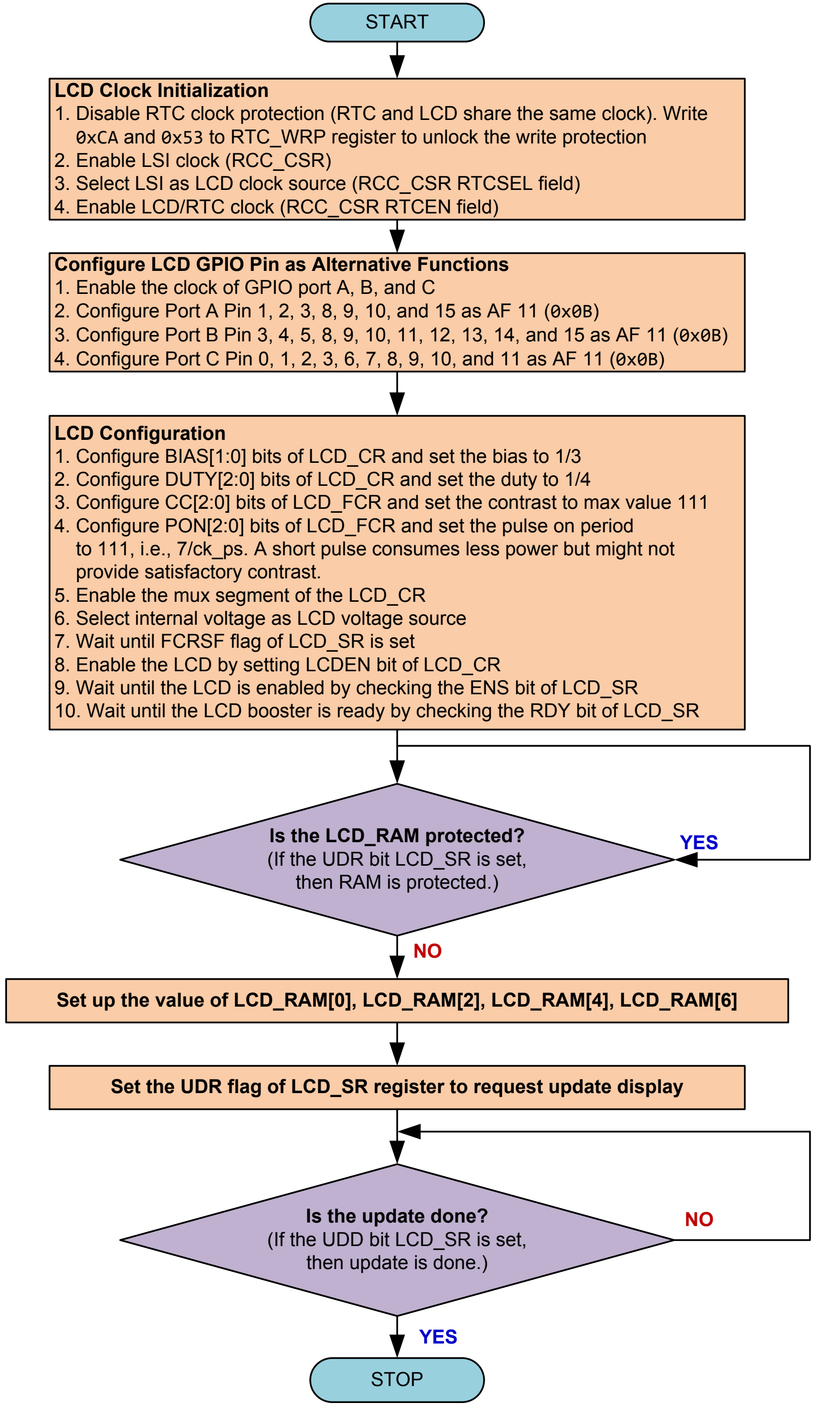
Using flowcharts in all labs

Write down your last name, and complete the following table.



Your Last Name: ________________________ (First Six Characters)

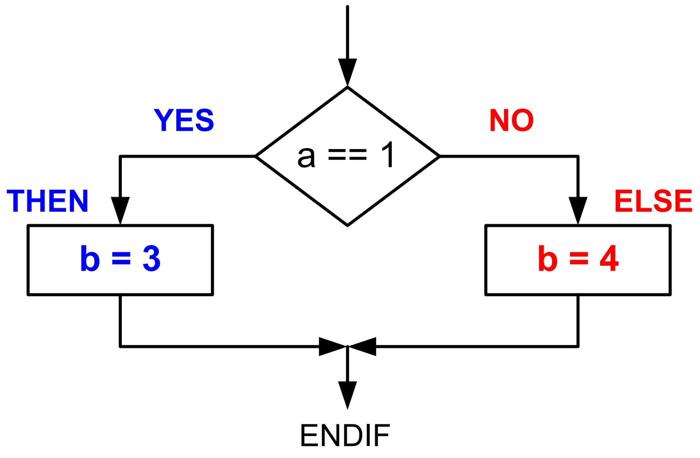
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCD_RAM[0]	4E	4G	3M	3B		6G	5M	5B	1M	1B					6E		3E	3G	2M	2B			6B	6M		2E	2G	1E	1G			
LCD_RAM[1]																													5E	5G	4M	4B
LCD_RAM[2]	4D	4F	3C	3A		6F	5C	5A	1C	1A					6D		3D	3F	2C	2A			6A	6C		2D	2F	1D	1F			
LCD_RAM[3]																													5D	5F	4C	4A
LCD_RAM[4]	4P	4Q	3 Col	3K		6Q	3 Bar	5K	1 Col	1K					6P		3P	3Q	2 Col	2K			6K	1 Bar		2P	2Q	1P	1Q			
LCD_RAM[5]																													5P	5Q	4 Col	4K
LCD_RAM[6]	4N	4H	3 DP	3J		6H	2 Bar	5J	1 DP	1J					6N		3N	3H	2 DP	2J			6J	0 Bar		2N	2H	1N	1H			
LCD_RAM[7]																													5N	5H	4 DP	4J



A Structured Approach in Assembly Programming

Methods of teaching structured programming in assembly

- Using flowcharts
  - Separate program structuring from code writing
- Leveraging C programs
  - Relate an unstructured to a structured
  - C *vs.* Assembly line-by-line comparison



C Program	Assembly Program

`if (a == 1)`
`  b = 3`
`else`
`  b = 4;`
	`; r1 = a, r2 = b`
	`     CMP r1, #1`
	`     BNE else`
	`then MOV r2, #3`
	`     B endif`
	`else MOV r2, #4`
	`endif`

A Structured Approach in Assembly Programming

Methods of teaching structured programming in assembly

- Using flowcharts
  - Separate program structuring from code writing
- Leveraging C programs
  - Relate an unstructured to a structured
  - C *vs.* Assembly line-by-line comparison
  - Mixing C and assembly

C calling assembly functions

```
int main(void) {
   ...
   s = sum(1,2,3,4);
   ...
}
```

```
sum PROC
    ADDS r0,r0,r1
    ...
    BX LR
    ENDP
```

Assembly calling C functions

```
main PROC
     ...
     BL sum
     ...
     ENDP
```

```
int sum(...) {
  return a+b+c+d;
}
```

Inline assembly

```
int sum(...) {
  __asm {
    ADD t, a, b;
    ...
  }
   ...
}
```

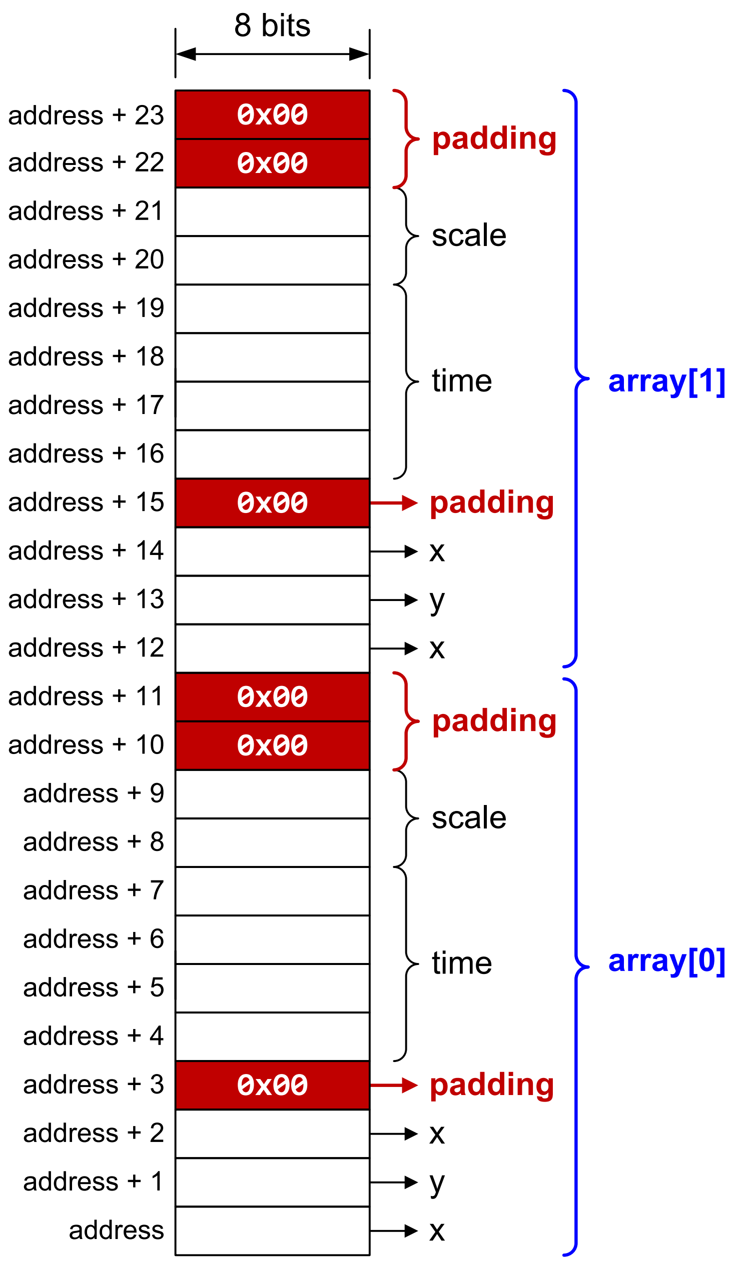
Extra benefits: Assembly helps to some difficult C concepts

- Structure padding

```
struct Position {
  char x;
  char y;
  char x;
  int time;
  short scale;
} array[10];
```

`address of array[0].time = array + offset`

When assembly access a variable in a C structure, the address offset has to take padding into consideration



Extra benefits: Assembly helps to some difficult C concepts

- *static* variables

C Program	Assembly Program

`int foo();`	`AREA myData, DATA`
	`ALIGN`
	`// Reserve space for x`
	`x    DCD   5`
	
	`AREA static_demo, CODE`
	`EXPORT __main`
	`ALIGN`
	`ENTRY`

`int main(void) {`
`  int y;`
`  y = foo();  // y = 6`
`  y = foo();  // y = 7`
`  y = foo();  // y = 8`
`  while(1);`
`}`
	`__main  PROC`
	`BL foo   ; r0 = 6`
	`BL foo   ; r0 = 7`
	`BL foo   ; r0 = 8`
	`stop    B   stop`
	`ENDP`

`int foo() {`
 	
`  // local static variable`
`  // x is initialized only once`
`  static int x = 5;`
 	
`  x = x + 1;`
`  return(x)`
`}`
	`foo     PROC`
	`; load address of x`
	`LDR r1, =x`
	`; load value of x`
	`LDR r0, [r1]`
	`ADD r0, r0, #1`
	`; save value of x`
	`STR r0, [r1]`
	
	`BX  lr`
	`ENDP`
	`END`

Extra benefits: Assembly helps to some difficult C concepts

- *volatile* variables

<table>
<tr><th>Main Program (main.c)</th><th>Interrupt Service Routine (isr.s)</th></tr>
<tr><td>

```c
volatile unsigned int counter;
extern void task();
extern void SysTick_Init();

int main(void) {
   counter = 10;
   SysTick_Init();
   while(counter != 0);  // Delay
   // continue the task
   ...
   while(1);
}
```

</td><td>

```
      AREA ISR, CODE, READONLY
      IMPORT counter
      ENTRY

SysTick_Handler PROC
      EXPORT SysTick_Handler
      LDR r1, =counter
      LDR r0, [r1]     ; load counter
      SUB r0, r0, #1   ; counter--
      STR r0, [r1]     ; save counter
      BX  LR           ; exit
      ENDP
      END
```

</td></tr>
</table>

My approach of teaching

1. Using modern platforms and tools
2. Bare-metal programming
3. Structured programming in Assembly
4. Lab-centered learning
5. Online tutorials

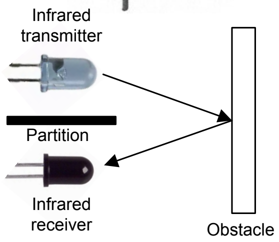
Lab modules

Covering both fundamental and advanced topics

- Lower level courses
  1. Push button and light up LEDs
  2. LCD display driver
  3. Interfacing with keypad
  4. Stepper motor control
  5. SysTick
  6. RTC
  7. PWM (diming LED, servo motors)
  8. Timer input capture (Ultra sonic distance sensor)
  9. ADC (potentiometer, infrared distance sensing)
  10. DAC (music synthesizing)



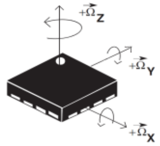




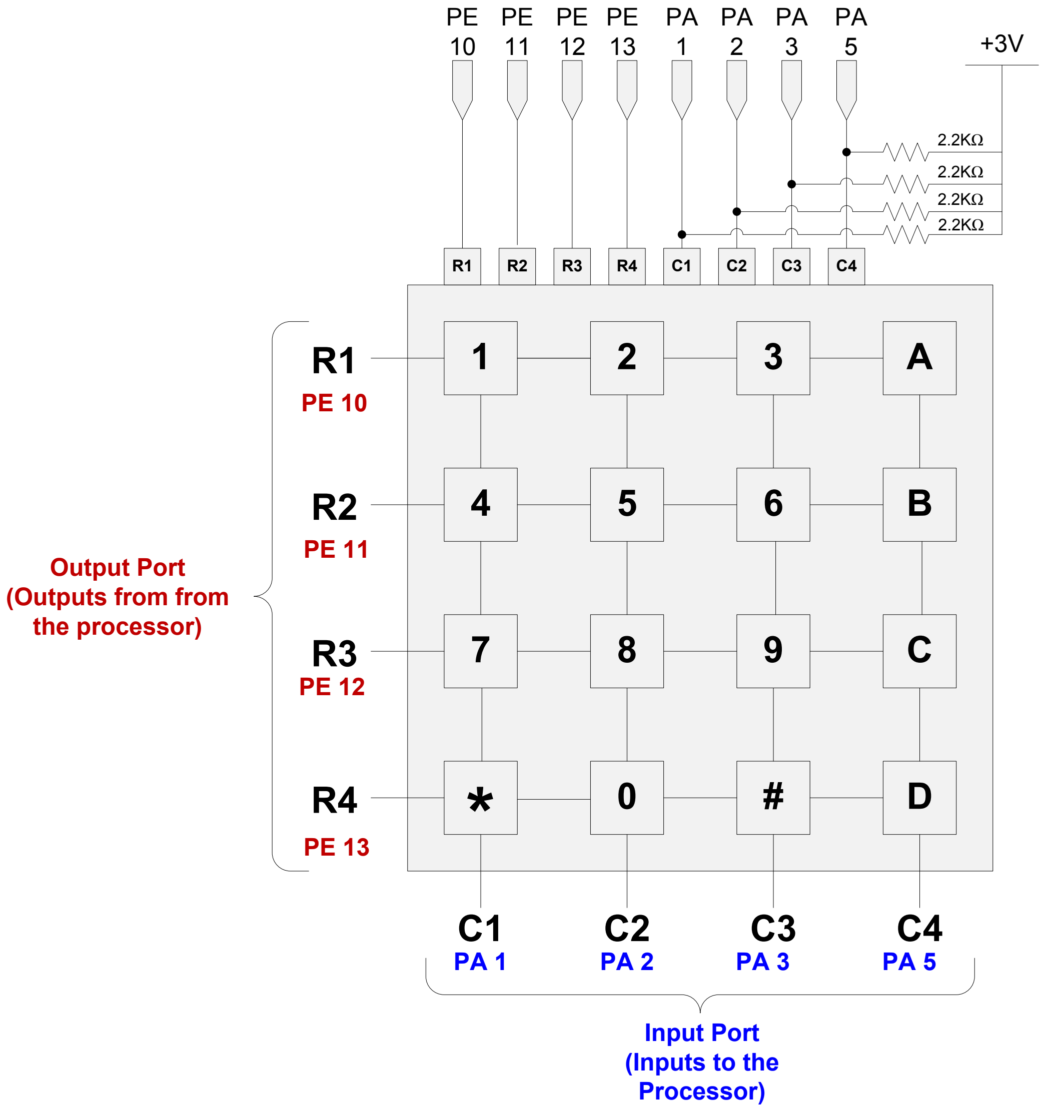
- Higher level courses
  1. External Interrupts
  2. UART (Bluetooth hc-05, ESP8266)
  3. I2C (temperature sensor, OLED display)
  4. SPI (gyro, accelerometer, nRF24L01)
  5. RGB LED strip (WS2812)
  6. ADC
  7. CODEC and Mic
  8. CRC

Polling, Interrupt, DMA

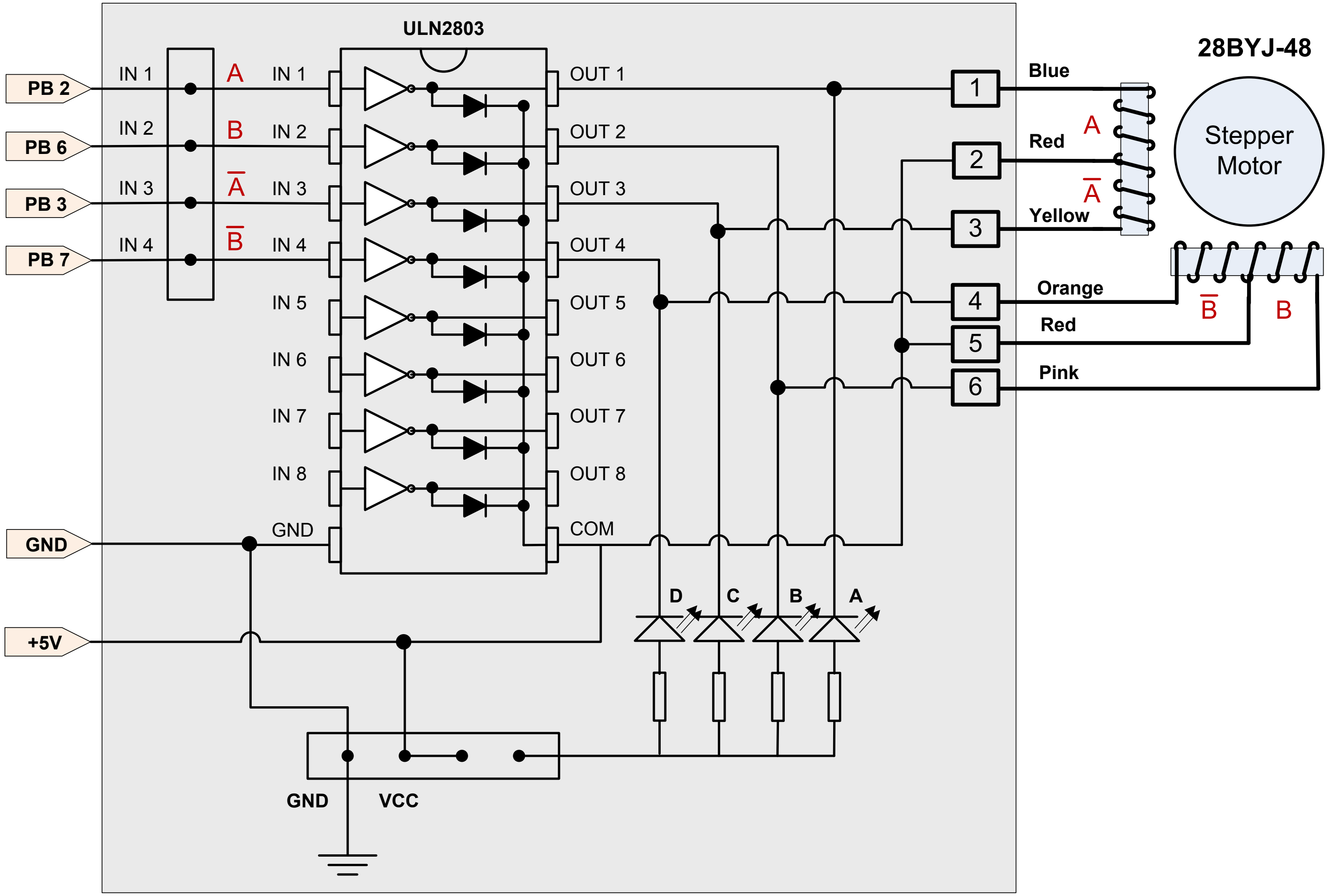


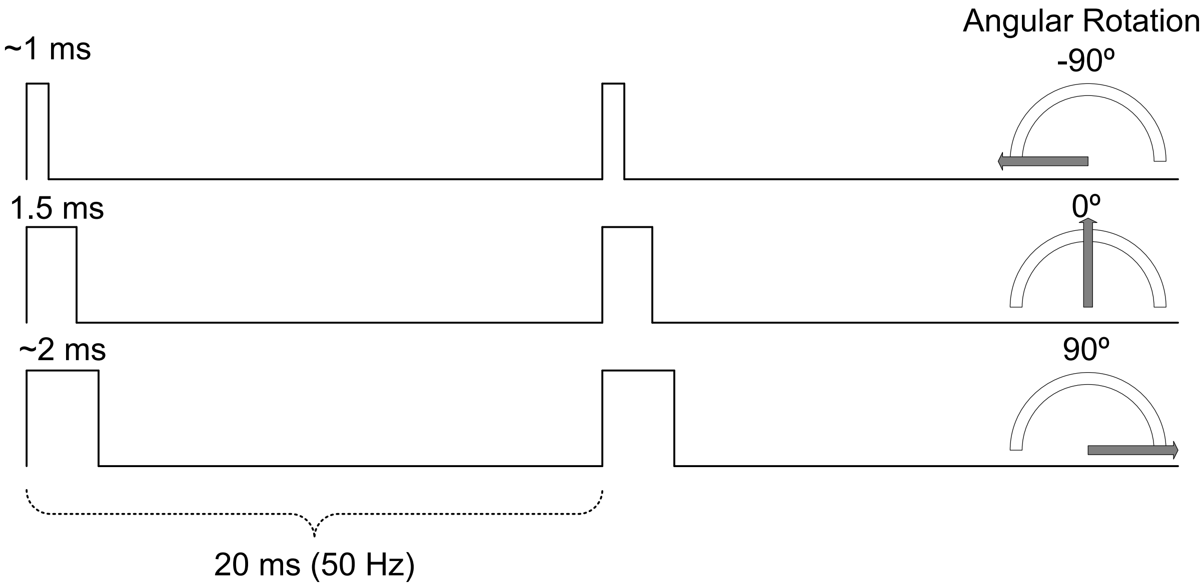

Example Lab: Digital Inputs



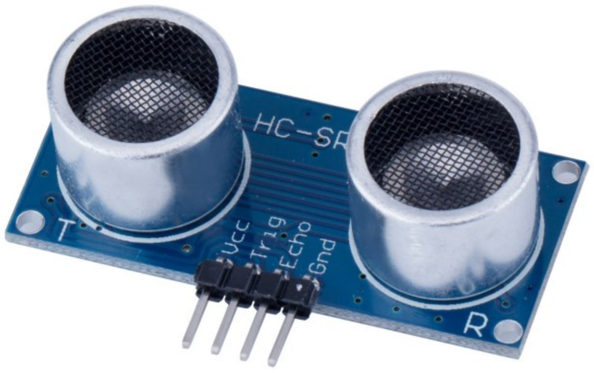
Example Lab: Digital Outputs

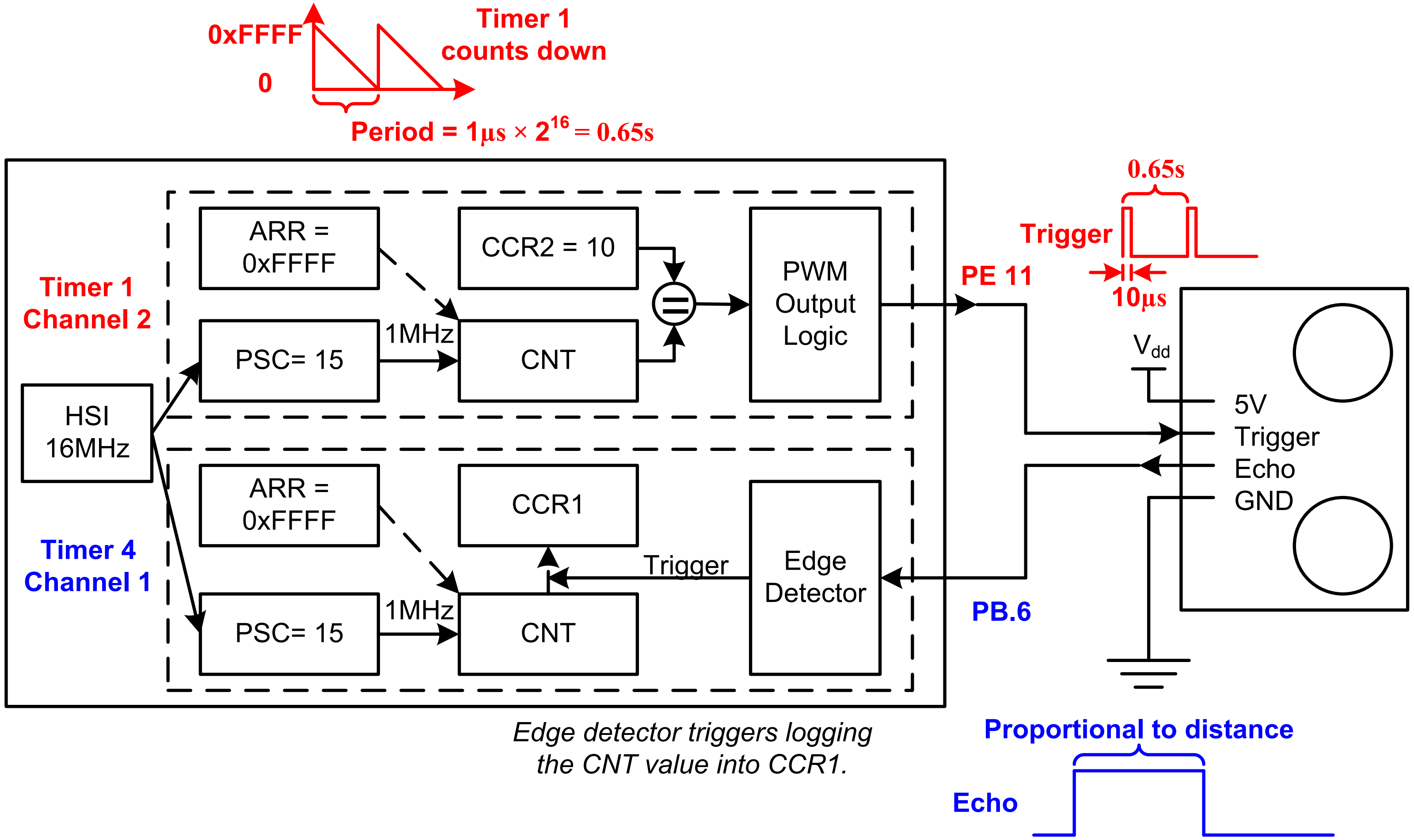


Example Lab: Timer PWM output

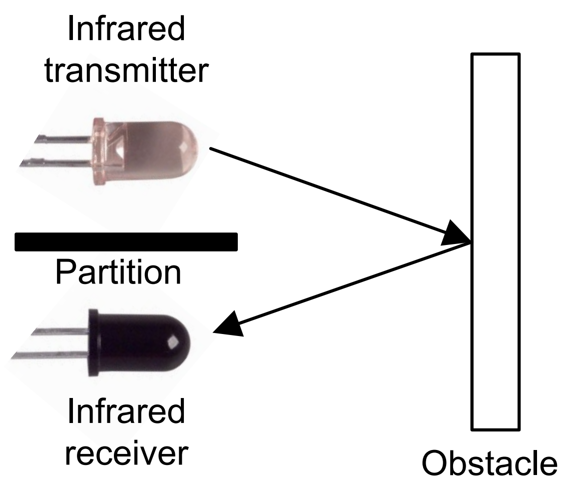


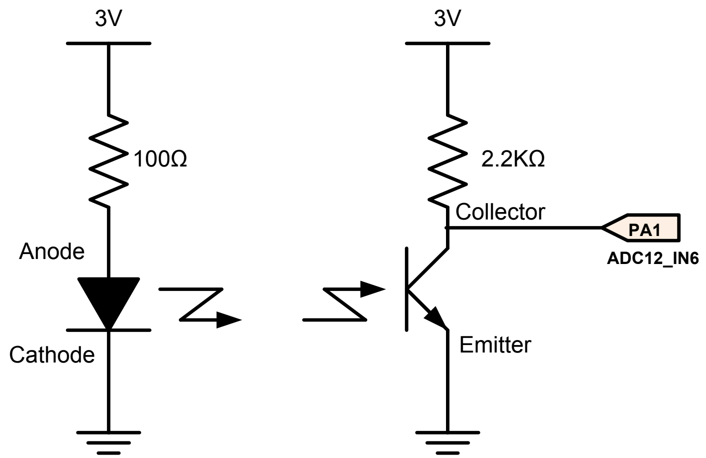
Example Lab:
Ultrasonic Distance Measurement





Example Lab: ADC





Start

1. Turn on HSI (RCC_CR_HSION)
2. Wait for it is ready (RCC_CR_HSIRDY).

Configure GPIO PB.6 as output with push-pull for blue LED

Configure GPIO PC.0 as Analog Input

Note: PC.0 is connected the ADC Channel 10 (PC.0 = ADC_IN10)

1. Enable the clock of GPIO C
2. Set PC.0 as Analog Input (GPIO_MODER)

Analog to Digital Converter 1 (ADC1) Setup

Note: HSI (16MHz) is always used for ADC on STM32L.

1. Turn on the ADC clock (RCC_APB2ENR_ADC1EN)
2. Turn off the ADC conversion (ADC1->CR2)
3. Set the length of the regular channel sequence to 1 since we only perform ADC in Channel 10. (L[4:0] bits of register ADC1->SQR1)
4. Set Channel 10 as the 1st conversion in regular sequence (SQ1[4:0] bits of register ADC1->SQR5)
5. Configure the sample time register for channel 10 (SMP10[2:0] bits of register ADC1->SMPR2)
6. Enable End-Of-Conversion interrupt (EOCIE bit of register ADC1->CR1)
7. Enable continuous conversion mode (CONT bit of register ADC1->CR2)
8. Configure delay selection as delayed until the converted data have been read (DELS[2:0] bits in register ADC1->CR2)
9. Enable the interrupt of ADC1_IRQn in NVIC
10. Configure the interrupt priority of ADC1_IRQn
11. Turn on the ADC conversion (ADON bit of register ADC1->CR2)
    Note: Make sure that we should write to CR2 register before the next step since SWSTART cannot be updated if ADC is off.
12. Start the conversion of the regular channel (ADC_CR2_SWSTART)
    Note: If SWSTART only performs one conversion, then it is very likely that your code did not set up the delay correctly in step 8.

Dead Loop

Lab Components

Pre-Lab Assignment (10%)

- Check at the beginning of the lab session
- Graded on completion, not correctness

In-lab Assignment

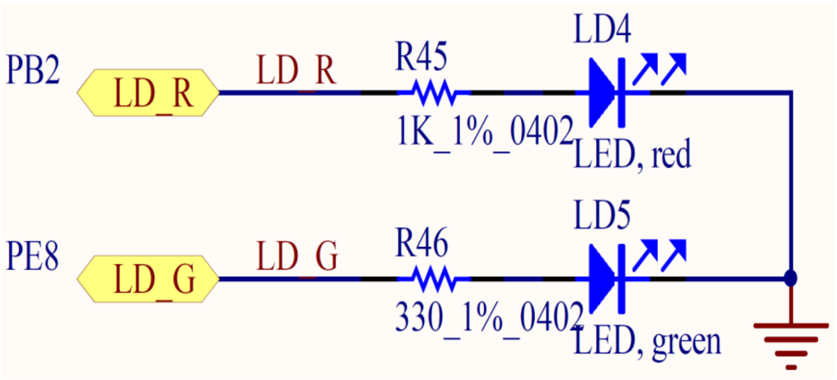
Post-lab Assignment (5%)

- Graded at beginning of next lab session
- Graded based on
  - Documentation
  - Correctness
  - Something cool (6%)

Points	Requirements	Poor	Fair	Good
2	**Completion of Pre-lab Assignments**	0	1	2
	Poor: absent from lab or does not complete pre-lab assignment			
	Fair: complete pre-lab assignment but lacks some details			
	Good: complete pre-lab assignment with details and minimal 90% correct answers			
5	**Documentation & Maintainability**	Poor	Fair	Good
	Proper indentations, whitespaces, and blank lines, ample and non-redundant comments	0	0.5	1
	Completion of readme.txt write-up (status, description of something cool, feedbacks)	0	0.5	1
	Header description (author, program objectives, pin usage, clock frequency)	0	0.5	1
	Frequent and correct commits with comments in Gitlab	0	0.5	1
	Program uses constant symbols defined whenever possible	0	0.5	1
				Total
5	**Functionality & Correctness**	Poor	Fair	Good
	No compilation errors or warnings (except warning L6314W)	0	0.5	1
	Exhibits all required functionality	0	1	2
	Concise code (Codes that are unnecessary should be deleted)	0	0.5	1
	Efficient and robust code	0	0.5	1
				Total
5	**Lab Time and Demonstration**	Poor	Fair	Good
	Make good use of lab time (Poor: leave before lab is done; Fair: accomplish a few objectives; Good: completes all objectives)	0	0.5	1
	Demo as specified by the lab assignment	0	1	2
	Answer TA's questions clearly and demonstrate thorough understanding	0	0.5	1
	Complete post-lab assignments	0	0.5	1
				Total
3	**Something Cool**	Poor	Fair	Good
	Note: Flashing LED is NOT considered as something cool except Lab 1.	0	1.5	3
Total 20 points				Number of late days

Hands-on Lab #1

Light up an LED in 100% assembly



Pre-Lab Assignment

1. **Enable the clock of GPIO Port A (for joy stick), Port B (for Red LED) and Port E (for Green LED)**

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
AHB2ENR														RNGEN		AESEN			ADCEN	OTGFSEN					GPIOPHEN	GPIOPGEN	GPIOPFEN	GPIOPEEN	GPIOPDEN	GPIOPCEN	GPIOPBEN	GPIOPAEN
Mask																																
Value																																

a. **Configure PB 2 as Output**

GPIO Mode: Input (00), Output (01), Alternative Function (10), Analog (11, default)

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
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]	
Mask																																
Value																																

b. **Configure PB 2 Output Type as Push-Pull**

Push-Pull (0, reset), Open-Drain (1)

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
OTYPER	Reserved																OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
Mask																																
Value																																

My approach of teaching

1. Using modern platforms and tools
2. Bare-metal programming
3. Structured programming in Assembly
4. Lab-centered learning
5. Online tutorials

YouTube Lectures & Tutorials

- Tutorials
  1. Create a project in Keil v5
  2. Debugging in Keil v5
  3. Clock configuration of STM32L4 processors
  4. Printing messages via UART through ST-Link V2.1
  5. How to fix common errors?

- Short Lectures
  1. Why do we use Two's Complement?
  2. Carry and Borrow Flag
  3. Overflow Flag
  4. Pointer
  5. Memory Mapped I/O
  6. GPIO Output: Lighting up a LED
  7. GPIO Input: Interfacing a joystick
  8. Timer: PWM output
  9. Interrupt Enable and Interrupt Priority
  10. Interrupts
  11. External Interrupts (EXTI)
  12. System Timer (SysTick)
  13. Booting process
  14. LCD
  15. Race Conditions

One open challenge:
How to get more female students?

- Out of 60K subscribers

Summary

1. Using modern platforms and tools
2. Bare-metal programming
3. Structured programming in Assembly
4. Lab-centered learning
5. Online tutorials

For more information

- Send email to Yifeng.Zhu@maine.edu for
  - An exam copy of my book
  - Complete instructor resources: slides, exams, quizzes, solutions, lab handouts & solutions
- My book website: http://web.eece.maine.edu/~zhu/book/
  - Sample labs, lab kit, FAQ
- My YouTube Channel: https://www.youtube.com/channel/UCY0sQ9hpSR6yZobt1qOv6DA

Thank STMicroelectronics for organizing this workshop!