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

Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C
Third Edition

Dr. Yifeng Zhu

738 pages, $69.50

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

Reference manual & datasheet

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
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

Integrated ST-Link/V2 programming and debugging tool
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

**Features:**
- 9-axis motion sensor (underneath LCD)
- LED indicators
- Joystick
- Audio Connector
- Microphone
- USB OTG
- Flash memory
- Audio Codec
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.
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 (Hardware Abstraction Layer) 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);

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

C implementation

```c
#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)
```

Assembly implementation

```assembly
LED_PIN EQU 2

LD
r0, =GPIOB_BASE
LD
r1, [r0, #GPIO_MODER]
EOR r1, r1, #(0x03<<(2*LED_PIN))
ORR r1, r1, #(1<<LED_PIN)
STR r1, [r0, #GPIO_MODER]
```
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
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
A Structured Approach in Assembly Programming

Methods of teaching structured programming in assembly

- Using flowcharts
- Separate program structuring from code writing
Using flowcharts in all labs

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

Your Last Name: ____________________________ (First Six Characters)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LCD Clock Initialization
1. Disable RTC clock protection (RTC and LCD share the same clock). Write 0xCA and 0x53 to RTC_WRP register to unlock the write protection
2. Enable LSI clock (RCC_CSR)
3. Select LSI as LCD clock source (RCC_CSR RTCSEL field)
4. Enable LCD/RTC clock (RCC_CSR RTCEO field)

Configure LCD GPIO Pin as Alternative Functions
1. Enable the clock of GPIO port A, B, and C
2. Configure Port A Pin 1, 2, 3, 8, 9, 10, and 15 as AF
3. Configure Port B Pin 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, and 15 as AF
4. Configure Port C Pin 0, 1, 2, 3, 6, 7, 8, 9, 10, and 11 as AF

LCD Configuration
1. Configure BIAS[1:0] bits of LCD_CR and set the bias to 1/3
2. Configure DUTY[2:0] bits of LCD_CR and set the duty to 1/4
3. Configure CC[2:0] bits of LCD_FCR and set the contrast to max value 111
4. Configure PON[2:0] bits of LCD_FCR and set the pulse on period to 111, i.e., 770 ps. A short pulse consumes less power but might not provide satisfactory contrast.
5. Enable the mux segment of the LCD_CR
6. Select internal voltage as LCD voltage source
7. Wait until FCRSF flag of LCD_SR is set
8. Enable the LCD by setting LCDEN bit of LCD_CR
9. Wait until the LCD is enabled by checking the ENS bit of LCD_SR
10. Wait until the LCD booster is ready by checking the RDY bit of LCD_SR

Is the LCD_RAM protected?
(If the UDR bit LCD_SR is set, then RAM is protected.)

Set the value of LCD_RAM[0], LCD_RAM[2], LCD_RAM[4], LCD_RAM[6]

Is the update done?
(If the UDD bit LCD_SR is set, then update is done.)

Set the UDR flag of LCD_SR register to request update display

Yes
No
No
Stop

Yes
Stop
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
if (a == 1)  
  b = 3  
else        
  b = 4;

Assembly Program
; r1 = a, r2 = b
CMP r1, #1
BNE else
then MOV r2, #3
B endif
else MOV r2, #4
endif
```
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

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

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

Assembly calling C functions

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

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

Inline assembly

```c
int sum(...) {
    __asm {
        ADD t, a, b;
        ...
    }
    ...
}
```
Extra benefits: Assembly helps to some difficult C concepts

- Structure padding

```c
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

<table>
<thead>
<tr>
<th>C Program</th>
<th>Assembly Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>int foo();</td>
<td>AREA myData, DATA ALIGN // Reserve space for x DCD 5</td>
</tr>
<tr>
<td></td>
<td>AREA static_demo, CODE EXPORT __main ALIGN ENTRY</td>
</tr>
<tr>
<td>int main(void) {</td>
<td>__main PROC</td>
</tr>
<tr>
<td>int y;</td>
<td>BL foo ; r0 = 6</td>
</tr>
<tr>
<td>y = foo(); // y = 6</td>
<td>BL foo ; r0 = 7</td>
</tr>
<tr>
<td>y = foo(); // y = 7</td>
<td>BL foo ; r0 = 8</td>
</tr>
<tr>
<td>y = foo(); // y = 8</td>
<td>stop B stop</td>
</tr>
<tr>
<td>while(1);</td>
<td>ENDP</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
<tr>
<td>int foo() {</td>
<td>foo PROC</td>
</tr>
<tr>
<td>// local static variable // x is initialized only once</td>
<td></td>
</tr>
<tr>
<td>static int x = 5;</td>
<td>; load address of x LDR r1, =x</td>
</tr>
<tr>
<td>x = x + 1;</td>
<td>; load value of x LDR r0, [r1]</td>
</tr>
<tr>
<td>return(x)</td>
<td>ADD r0, r0, #1</td>
</tr>
<tr>
<td>}</td>
<td>; save value of x STR r0, [r1]</td>
</tr>
<tr>
<td></td>
<td>BX lr</td>
</tr>
<tr>
<td>}</td>
<td>ENDP</td>
</tr>
<tr>
<td>}</td>
<td>END</td>
</tr>
</tbody>
</table>

```c
int foo();

// local static variable // x is initialized only once
static int x = 5;

x = x + 1;
return(x)
```
Extra benefits: Assembly helps to some difficult C concepts

- **volatile** variables

<table>
<thead>
<tr>
<th>Main Program (main.c)</th>
<th>Interrupt Service Routine (ISR.s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>volatile</strong> unsigned int <strong>counter</strong>;</td>
<td>AREA ISR, CODE, READONLY</td>
</tr>
<tr>
<td>extern void task();</td>
<td>IMPORT <strong>counter</strong></td>
</tr>
<tr>
<td>extern void SysTick_Init();</td>
<td>ENTRY</td>
</tr>
<tr>
<td>int main(void) {</td>
<td><strong>SysTick_Handler</strong> PROC</td>
</tr>
<tr>
<td>counter = 10;</td>
<td>EXPORT SysTick_Handler</td>
</tr>
<tr>
<td>SysTick_Init();</td>
<td>LDR r1, =counter</td>
</tr>
<tr>
<td>while(counter != 0); // Delay</td>
<td>LDR r0, [r1] ; load counter</td>
</tr>
<tr>
<td>// continue the task</td>
<td>SUB r0, r0, #1 ; <strong>counter</strong>--</td>
</tr>
<tr>
<td>...</td>
<td>STR r0, [r1] ; save counter</td>
</tr>
<tr>
<td>while(1);</td>
<td>BX LR ; exit</td>
</tr>
<tr>
<td>}</td>
<td>ENDP</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
</tbody>
</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
Example Lab: Digital Inputs

Output Port (Outputs from the processor)

Input Port (Inputs to the Processor)
Example Lab: Digital Outputs
Example Lab: Timer PWM output

20 ms (50 Hz)
~1 ms
1.5 ms
~2 ms

Angular Rotation
-90º
0º
90º

20 ms (50 Hz)
Example Lab: Ultrasonic Distance Measurement

- **Trigger**
- **Echo**
- **Vdd**
- **GND**
- **Timer 4 Channel 2**
- **HSI 16MHz**
- **Timer 1 Channel 2**
- **PWM Output Logic**

- **ARR = 0xFFFF**
- **CCR2 = 10**
- **PSC = 15**
- **1MHz**
- **CNT**

- **Edge Detector**
- **0.65s**

- **Proportional to distance**

**Logic**

- **$PSC = 15$**
- **$ARR = 0xFFFF$**
- **$CNT$**
- **$CCR1 = 10$**
- **$PWM$**

**Period**

$1\mu s \times 2^{16} = 0.65s$

**Diagram**

- Edge detector triggers logging the CNT value into CCR1.
Example Lab: ADC

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 STM32.
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 (SQ[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**
  - Graded at beginning of next lab session
  - Graded based on
    - Documentation
    - Correctness
    - Something cool (6%)

- **Post-lab Assignment (5%)**

**Points Requirements**

<table>
<thead>
<tr>
<th>Points</th>
<th>Requirements</th>
<th>Poor</th>
<th>Fair</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Completion of Pre-lab Assignments</strong></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Poor: absent from lab or does not complete pre-lab assignment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fair: complete pre-lab assignment but lacks some details</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Good: complete pre-lab assignment with details and minimal 90% correct answers</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Points</th>
<th>Requirements</th>
<th>Poor</th>
<th>Fair</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td><strong>Documentation &amp; Maintainability</strong></td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Proper indentations, whitespaces, and blank lines, ample and non-redundant comments</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Completion of readme.txt write-up (status, description of something cool, feedbacks)</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Header description (author, program objectives, pin usage, clock frequency)</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Frequent and correct commits with comments in Gitlab</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Program uses constant symbols defined whenever possible</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Points</th>
<th>Requirements</th>
<th>Poor</th>
<th>Fair</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td><strong>Functionality &amp; Correctness</strong></td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No compilation errors or warnings (except warning L6314W)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exhibits all required functionality</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Concise code (Codes that are unnecessary should be deleted)</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Efficient and robust code</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Points</th>
<th>Requirements</th>
<th>Poor</th>
<th>Fair</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td><strong>Lab Time and Demonstration</strong></td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Make good use of lab time (Poor: leave before lab is done; Fair: accomplish a few objectives; Good: completes all objectives)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Demo as specified by the lab assignment</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Answer TA’s questions clearly and demonstrate thorough understanding</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Complete post-lab assignments</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Points</th>
<th>Requirements</th>
<th>Poor</th>
<th>Fair</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><strong>Something Cool</strong></td>
<td>0</td>
<td>1.5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Note: Flashing LED is NOT considered as something cool except Lab 1.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Points</th>
<th>Requirements</th>
<th>Poor</th>
<th>Fair</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Total 20 points</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Number of late days</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hands-on Lab #1

Light up an LED in 100% assembly

Pre-Lab Assignment

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

<table>
<thead>
<tr>
<th>Register</th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHB2ENR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mask</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 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   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 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

- **Tutorials**
  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

  - Sample labs, lab kit, FAQ

- My YouTube Channel: https://www.youtube.com/channel/UCY0sQ9hpSR6yZobt1qOv6DA

Thank STMicroelectronics for organizing this workshop!