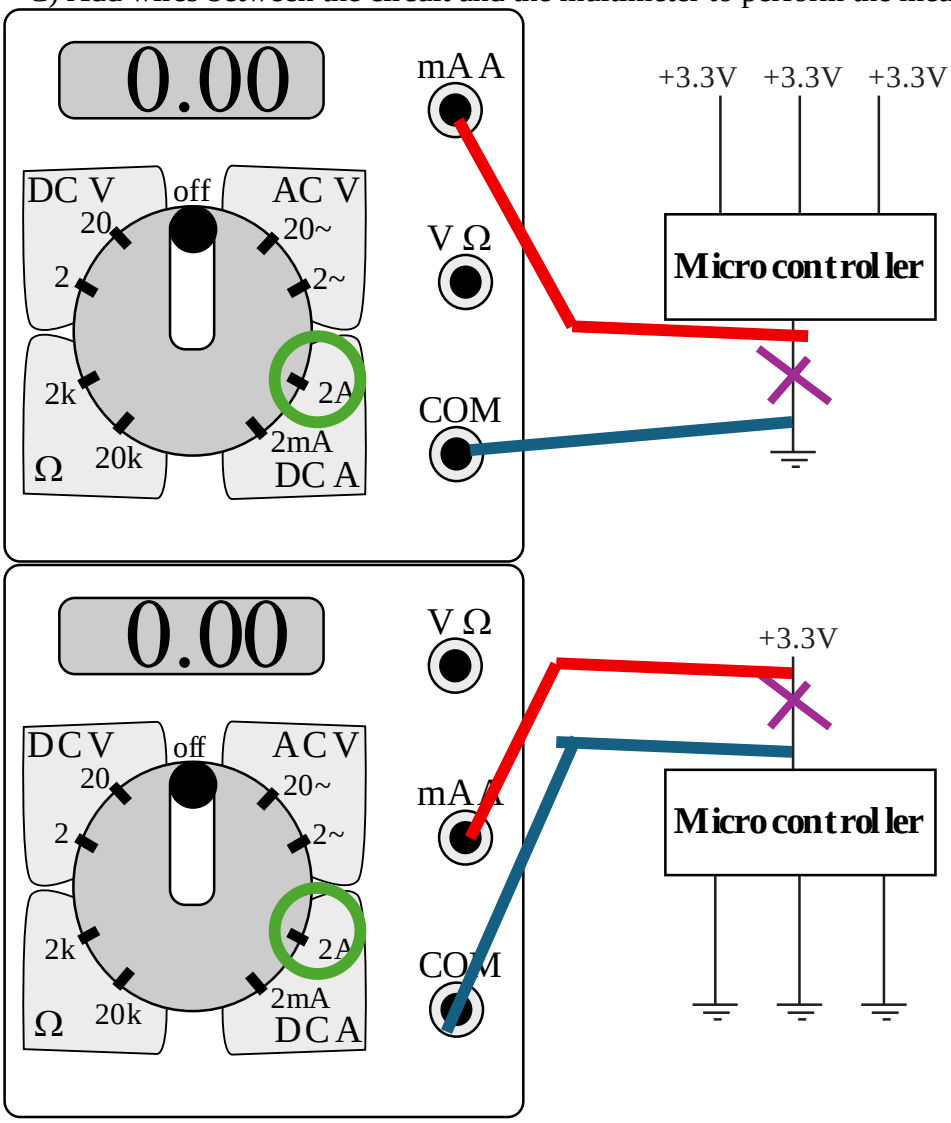
Jonathan W. Valvano Name:_____

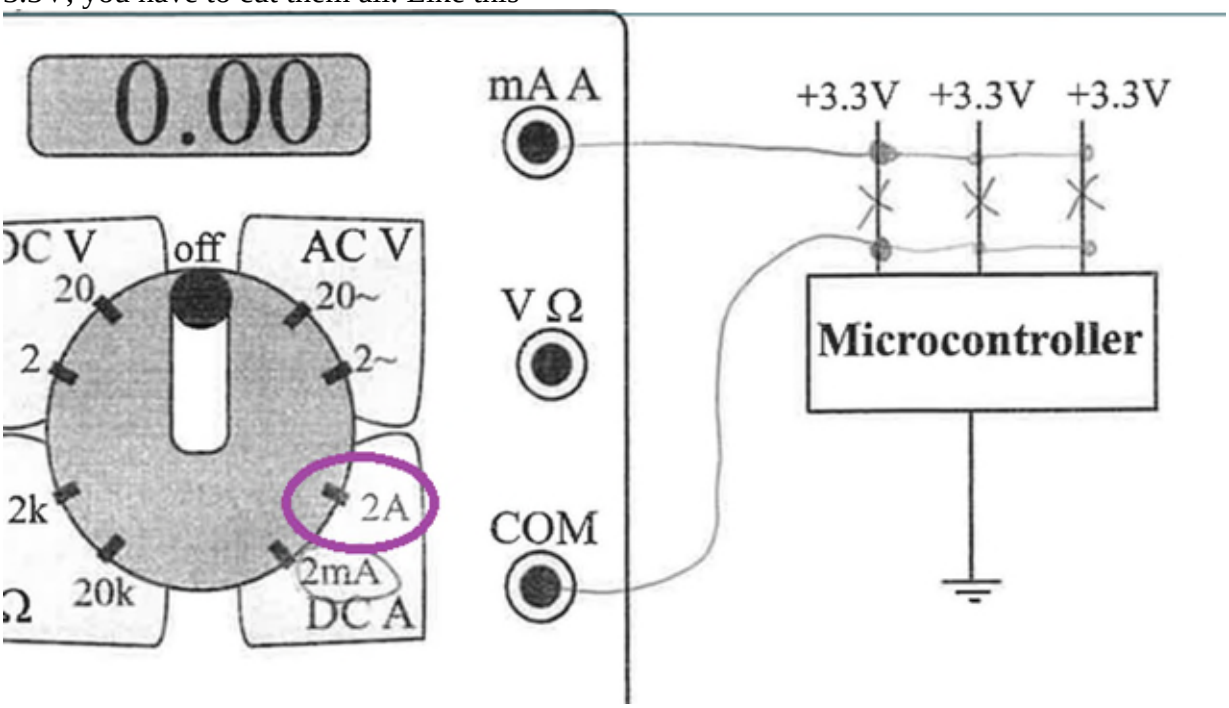
Oct 1, 2025, 3:30-4:45p. This is a closed book exam. You have 75 minutes, so please allocate your time accordingly. No calculators allowed. ***Please read the entire quiz before starting***.

**(10) Question 1.** Use a DVM to measure total current through this microcontroller, which has three sources of 3.3V supply.

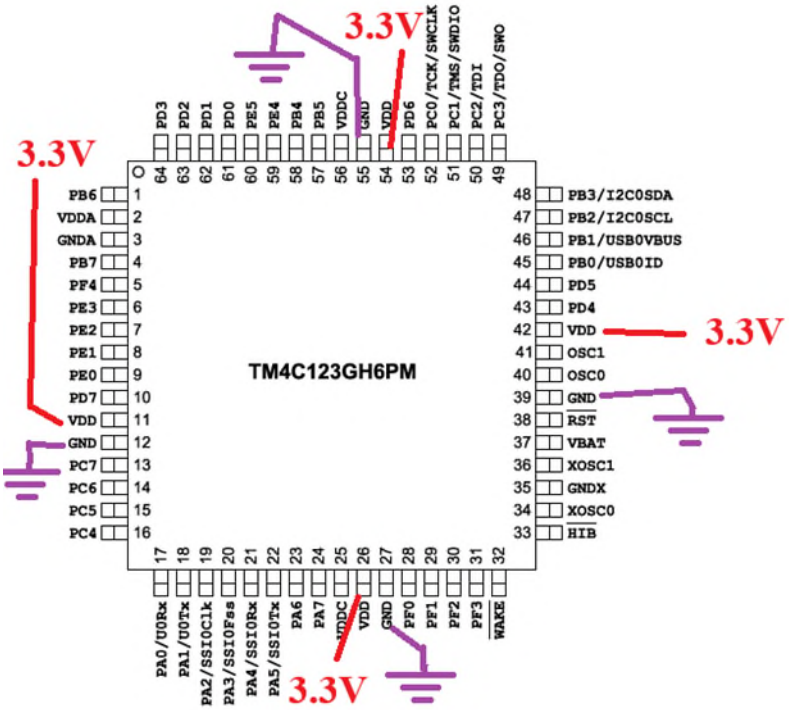
A) Circle one of the numbers on the dial specifying the correct mode to use.
B) Cut wire(s) with an **X** if needed.
C) Add wires between the circuit and the multimeter to perform the measurement.



If a microcontroller has multiple power connections into 3.3V then to measure current at the 3.3V, you have to cut them all. Like this




If a microcontroller has multiple power connections into ground then to measure current at the ground, you have to cut them all.



**(10) Question 2.** Consider two ISRs that are linked via one global variable, **Voltage**. TIMG0 runs at a higher priority than TIMG8. The 64-bit local variables **x y** exist in two registers R0,R1.

```
double Voltage; // 64 bits
void TIMG0_IRQHandler(void){
  double x;
// sets x to new double float
  Voltage = x;
}

void TIMG8_IRQHandler(void){
  double y;
  y = Voltage;
// operates on y
}
```

```
Voltage: .space 8
TIMG0_IRQHandler:
// sets R0,R1 to double float
   LDR R2,=Voltage
   STR R0,[R2]
   STR R1,[R2,#4]
   BX  LR
TIMG8_IRQHandler:
   LDR R2,=Voltage
   LDR R0,[R2]
   LDR R1,[R2,#4]
// operates on R0,R1
   BX  LR
```

Is there a critical section? If yes, circle the places in the assembly code where if an interrupt were to happen, an error would occur. If no, prove it has no critical section

Yes, critical section is between the two arrows, shown above

```
double Voltage; // 64 bits
void TIMG0_IRQHandler(void){
  double x;
// sets x to new double float
  Voltage = x;
}

void TIMG8_IRQHandler(void){
  double y;
  y = Voltage;
// operates on y
}
```

```
Voltage: .space 8
TIMG0_IRQHandler:
// sets R0,R1 to double float
   LDR R2,=Voltage
   STM R2!,{R0,R1}
   BX  LR

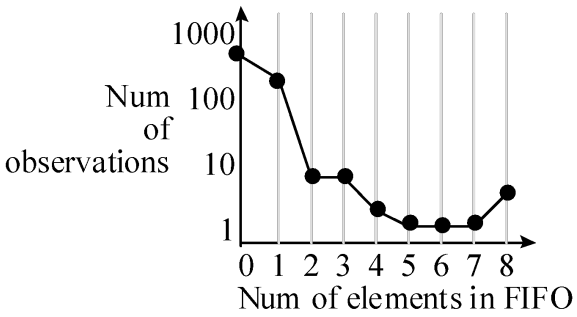
TIMG8_IRQHandler:
   LDR R2,=Voltage
   LDM R2!,{R0,R1}
// operates on R0,R1
   BX  LR
```

Is there a critical section? If yes, circle the places in the assembly code where if an interrupt were to happen, an error would occur. If no, prove it has no critical section

No, LDM is atomic so 64 bit read will not be interrupted

In an ARM Cortex-M0 processor, if an interrupt occurs during an LDM (Load Multiple) instruction, the processor abandons the operation and restarts the instruction from the beginning after finishing the Interrupt Service Routine (ISR)

**(10) Question 3.** Consider an output device that uses a FIFO to stream data from the main program (calls **OutString**, which puts into the FIFO) to the UART ISR program (which gets from the FIFO and writes to the **TXDATA** register). The FIFO can hold up to 8 elements. If it has 0 elements the FIFO is empty. If it has 8 elements the FIFO is full. The size of the FIFO was measured periodically resulting in this histogram.



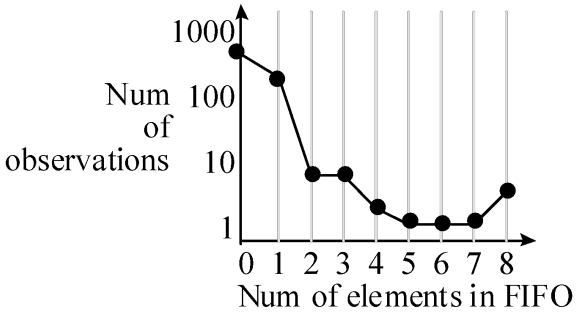

**(5) Part a)** Is this I/O bound or CPU bound (put A B C in the box)
A) *I/O bound; increasing just output hardware speed will increase bandwidth*
B) *CPU bound; increasing just software speed will increase bandwidth*
C) *Neither; improving just the I/O or CPU will not improve bandwidth*

B

**(5) Part b)** Should you increase the size of the FIFO? If yes, explain why the FIFO size should be increased. If no, explain why this FIFO size is ok.

Yes, Increase FIFO size because sometimes the FIFO is full, and if FIFO is full the system stalls or loses data

**(10) Question 3b.** Consider an input device that uses a FIFO to stream data from the UART ISR (reads **RXDATA** register and puts into the FIFO) to the main program (which calls **InString**, which gets from the FIFO). The FIFO can hold up to 8 elements. If it has 0 elements the FIFO is empty. If it has 8 elements the FIFO is full. The size of the FIFO was measured periodically resulting in the following histogram.




**(5) Part a)** Is this I/O bound or CPU bound (circle your answer)
A) *I/O bound; doubling just output hardware speed will double performance*
B) *CPU bound; doubling just software speed will double performance*
C) *Neither; improving just I/O or CPU will not improve performance*

*A*

**(5) Part b)** Should you increase the size of the FIFO? If yes, explain why the FIFO size should be increased. If no, explain why this FIFO size is ok.

Yes, Increase FIFO size because sometimes the FIFO is full, and if FIFO is full the system stalls or loses data

**(5) Question 4.** On average a FIFO has 19 bytes stored, while it is processing one byte of data. The average arrival rate of data is 1 byte per msec. What is the average response time of the system (arrival of data to completion of service)? Justify your answer.

Little's Thm $N = \lambda R$
$N$ = 19+1 = 20 bytes
$\lambda$= 1 byte/ms
$R$ = 20ms

**(5) Question 4b.** On average a FIFO has 39 bytes stored. The average arrival rate of data is 1 byte per msec. What is the average response time of the system (arrival of data to completion of service)? Justify your answer.

Little's Thm $N = \lambda R$
$N$ = 39+1 = 40 bytes
$\lambda$= 1 byte/ms
$R$ = 40ms

**(5) Question 5.** Give two advantages of **WebSockets** as compared to HTTP.

- **Low latency** — messages are delivered in milliseconds over an open connection. Since the connection is persistent, there's no need to establish a new connection for each message, significantly reducing latency.
- **Reduced overhead** — no HTTP headers are resent with every message.
- **Bidirectional flow** — both peers can initiate messages without waiting.
- **Efficient resource use** — one connection replaces hundreds of polling requests. WebSockets are more efficient than traditional polling techniques, as they don't require the client to continuously ask the server for updates.
- **Real-time Updates**: WebSockets enable instant data transmission, making them perfect for applications that require real-time updates, like live chat, gaming, or financial trading platforms.
- **Lower Overhead**: After the initial handshake, WebSocket frames have a small header (as little as 2 bytes), reducing the amount of data transferred.

**(10) Question 6.** Classify the following debugging tasks. Put N M or H in the boxes below
N) nonintrusive,
M) minimally intrusive, or
H) highly intrusive

**Part a)** Outputting one ASCII character using **printf** every 1 second, with a baud rate of 10000 bits/sec (1000 characters/sec). *M*

**Part b)** Connecting a scope and/or logic analyzer to an input pin that already exists as part of the system. *N*

**Part c)** Initializing a GPIO output pin just for debugging and toggling it three times in the ISR and observing the pin with a logic analyzer. *M*

**Part d)** Interfacing an LED to a GPIO output pin just for debugging, toggling it in the ISR and observing the LED with your eyes. *M*

**Part e)** Dumping data to the flash memory (black box flight recorder) and observing the data in the flash after a software crash. 100 times a second writing data to flash that takes 1ms to complete. *H*

**(10) Question 6.** Classify the following debugging tasks. Put N M or H in the boxes below
N) nonintrusive, M) minimally intrusive, or H) highly intrusive

**Part a)** Connecting a scope and/or logic analyzer to an input pin that already exists as part of the system. *N*

**Part b)** Initializing a GPIO output pin just for debugging and toggling it three times in the ISR and observing the pin with a logic analyzer. *M*

**Part c)** Dumping data to the flash memory (black box flight recorder) and observing the data in the flash after a software crash. 100 times a second writing data to flash that takes 1ms to complete. *H*

**Part d)** Outputting one ASCII character using **printf** every 1 second, with a baud rate of 10000 bits/sec (1000 characters/sec). *M*

**Part e)** Interfacing an LED to a GPIO output pin just for debugging, toggling it in the ISR, and observing the LED with your eyes. *M*

**(15) Question 7.** TIMG8 is configured so reading **TIMG8->COUNTERREGS.CTR** returns a 16-bit value that counts down at 1 MHz. Use this TIMG8 to measure the execution time of **TouchTask**. You may assume the execution time is less than 65ms. You must add global(s).

```
uint16_t Time; // usec
void GROUP1_IRQHandler(void){

uint16_t start; // usec


    GPIOA->CPU_INT.ICLR = 1<<8; // PA8

start = TIMG8->COUNTERREGS.CTR;

    TouchTask();

Time = start - TIMG8->COUNTERREGS.CTR;


}
```

**(10) Question 8.** The port configuration register, **GPIOA->CRL**, for pins PA7 to PA0 on STM32F103 microcontroller is given below.

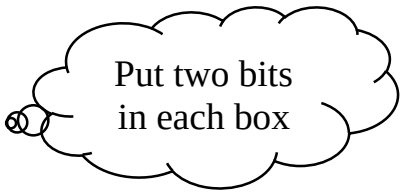
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CNF7[1:0] | | MODE7[1:0] | | CNF6[1:0] | | MODE6[1:0] | | CNF5[1:0] | | MODE5[1:0] | | CNF4[1:0] | | MODE4[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CNF3[1:0] | | MODE3[1:0] | | CNF2[1:0] | | MODE2[1:0] | | CNF1[1:0] | | MODE1[1:0] | | CNF0[1:0] | | MODE0[1:0] | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
**In input mode (MODE[1:0]=00)**:
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
**In output mode (MODE[1:0] > 00)**:
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

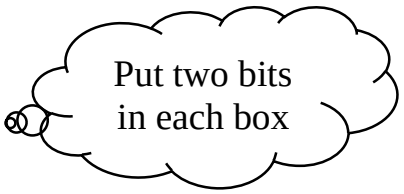
**(5) Part a)** Make PA1 on the STM32F103 microcontroller a regular GPIO output that drives high to 3.3V and low to 0V. The pin will toggle at 1 kHz. Specify these bits in the GPIOA->CRL

| | |
|---|---|
| CNF1[1:0] | 00 |
| MODE1[1:0] | 10 |

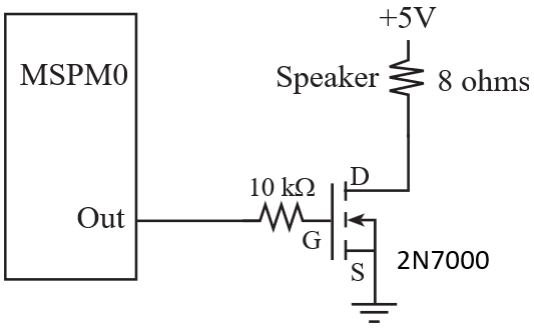



**(5) Part b)** Make PA0 on the STM32F103 microcontroller a GPIO input. Specify these bits in the GPIOA->CRL

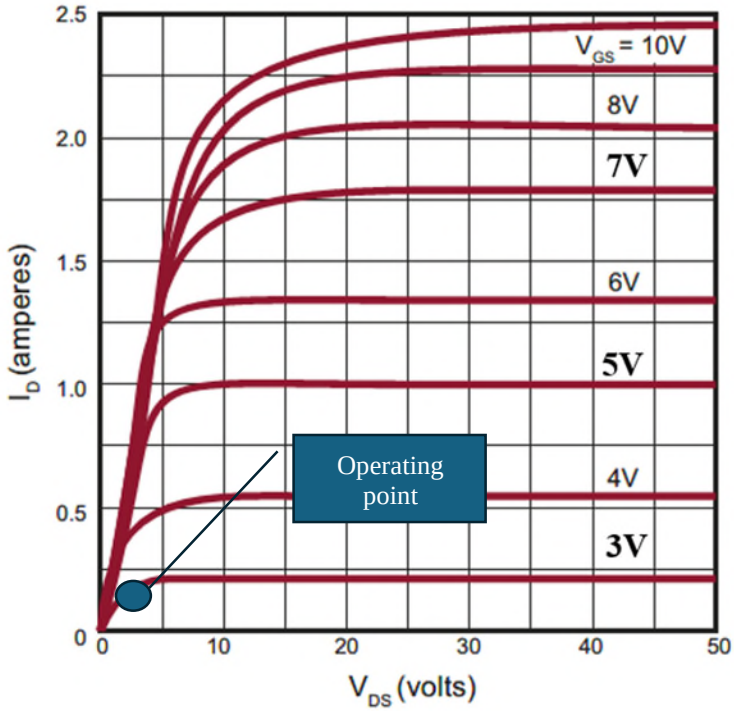
| | |
|---|---|
| CNF0[1:0] | 10 |
| MODE0[1:0] | 00 |



**(5) Question 9.** A 2N7000 N channel MOSFET was tried in Lab 3 to activate the speaker. Explain why this circuit did not generate sound, even though Out was a 1 kHz squarewave.




$V_{GS}$, which comes from the microcontroller, is too low AND +5V to top of speaker must be much larger (at 0.6A, $V_{DS}$ will be 2 to 3V even if $V_{GS}$ >5V)



**(20) Question 10.** Write the SysTick ISR that runs this line-following synchronous FSM in the background. There is a 4-bit output on PA3-PA0 and a 2-bit input on PA7-PA6. Assume PA3-PA0 are configured as GPIO outputs, and PA7-PA6 are configured as inputs. Assume SysTick is configured for 1 kHz periodic interrupts, and the CPU interrupts are enabled (I=0). ). Each interrupt invocation executes one sequence: input, next, and then output. Be friendly.

```
struct State{
  uint32_t Out;      // 8-bit output
  const struct State *Next[4];
};
typedef const struct State State_t;
#define Center &FSM[0]
#define Left &FSM[1]
#define Right &FSM[2]
State_t fsm[3] = {
  {0x0F, { Right, Right,  Left,   Center }},  // Center of line
  {0x0A, { Left,  Right,  Center, Center }},  // Left of line
  {0x05, { Right, Center, Left,   Center }}   // Right of line
};
State_t *S=Center;  // pointer to the current state
void SysTick_Handler(void){
```

```
uint32_t input, output;          // state I/O
  input = (GPIOA->IDR&0xC0)>>6; // read sensors
  S = S->Next[input];    // next depends on input and state
  output = S->Out;       // set output from FSM
  GPIOA->ODR = (GPIOA->ODR&~0x0F)|output; // do output
//THIS IS VERY BAD CODE
  GPIOA->ODR &= ~0x0F;  // clear
  GPIOA->ODR |= S->Out; // do output
// if you write to VAL, then SysTick no longer runs at 1ms
```

```
}
```

**GPIOA->IDR** Read from gets the current values of the input pins of STM32F103 port A
**GPIOA->ODR** Write to set output pins of STM32F103 port A; read from gets the last value written

| Address | 31-24 | 23-17 | 16 | 15-3 | 2 | 1 | 0 | Name |
|---|---|---|---|---|---|---|---|---|
| 0xE000E010 | 0 | 0 | COUNT | 0 | CLK_SRC | INTEN | ENABLE | SysTick->CTRL |
| 0xE000E014 | 0 | 24-bit LOAD value | | | | | | SysTick->LOAD |
| 0xE000E018 | 0 | 24-bit VAL value of SysTick counter | | | | | | SysTick->VAL |

Write anything to SysTick->VAL to clear COUNT and reload LOAD into VAL
Write 0 to CTRL for off, write 5 to CTRL for on but disarmed, write 7 to CTRL for on and armed.
The COUNT flag is set when the SysTick->VAL counts down from 1 to 0.