

ECCS-3351

Embedded Realtime Applications (ERA)

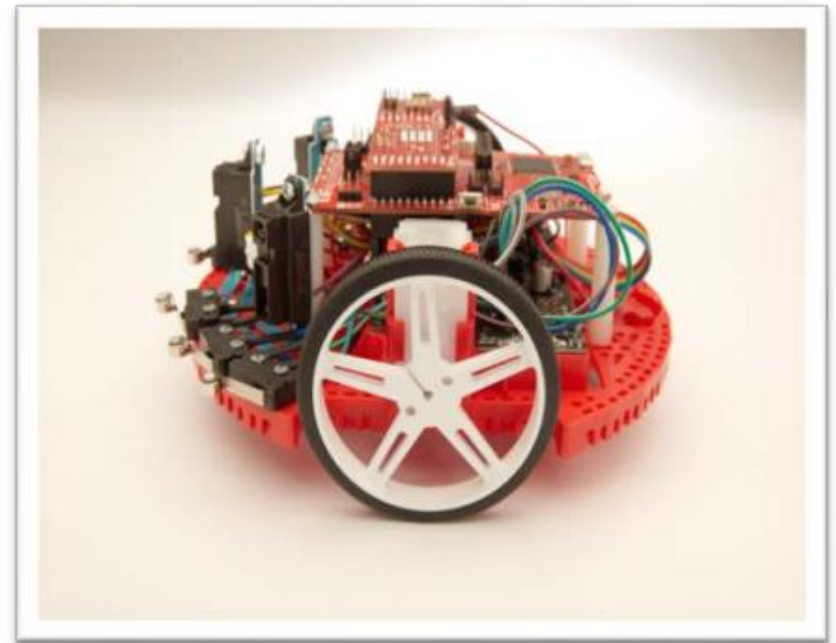
Timer Concepts

JONATHAN W. VALVANO
Modified by Drs. Kropp, Oun, and Youssfi



Objectives

- Fundamentals of SysTick Timer
- Measure elapsed time
 - Precision
 - Range
 - Resolution
- Software delay



Reintroduction to interrupts

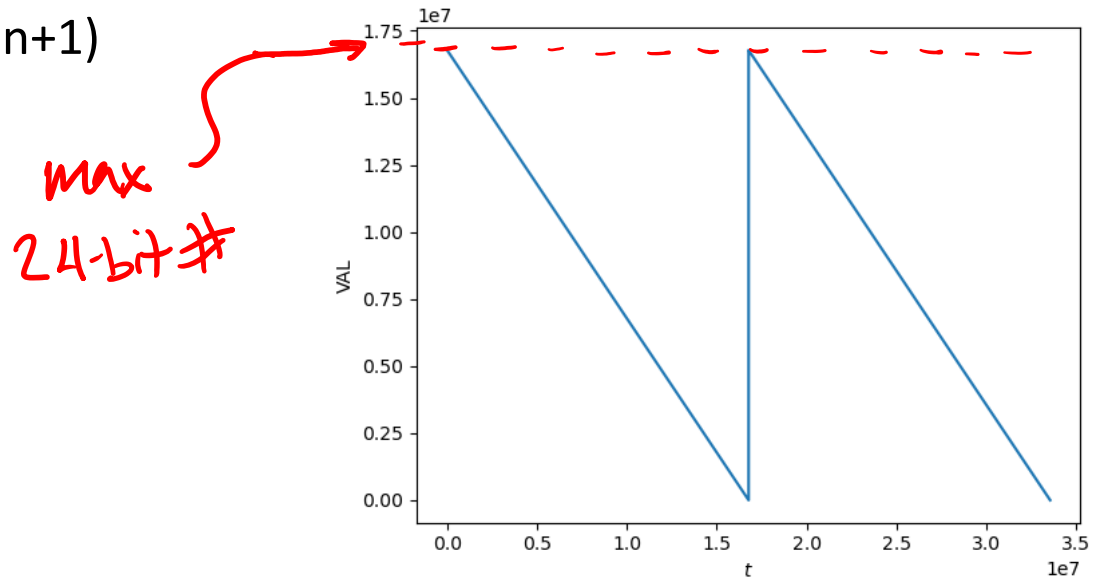
- Interrupts are mechanism by which other modules (e.g., a button, a sensor, software) may interrupt the normal sequencing of the processor
- Why would it be helpful to periodically interrupt an MCU?

Timer Fundamentals

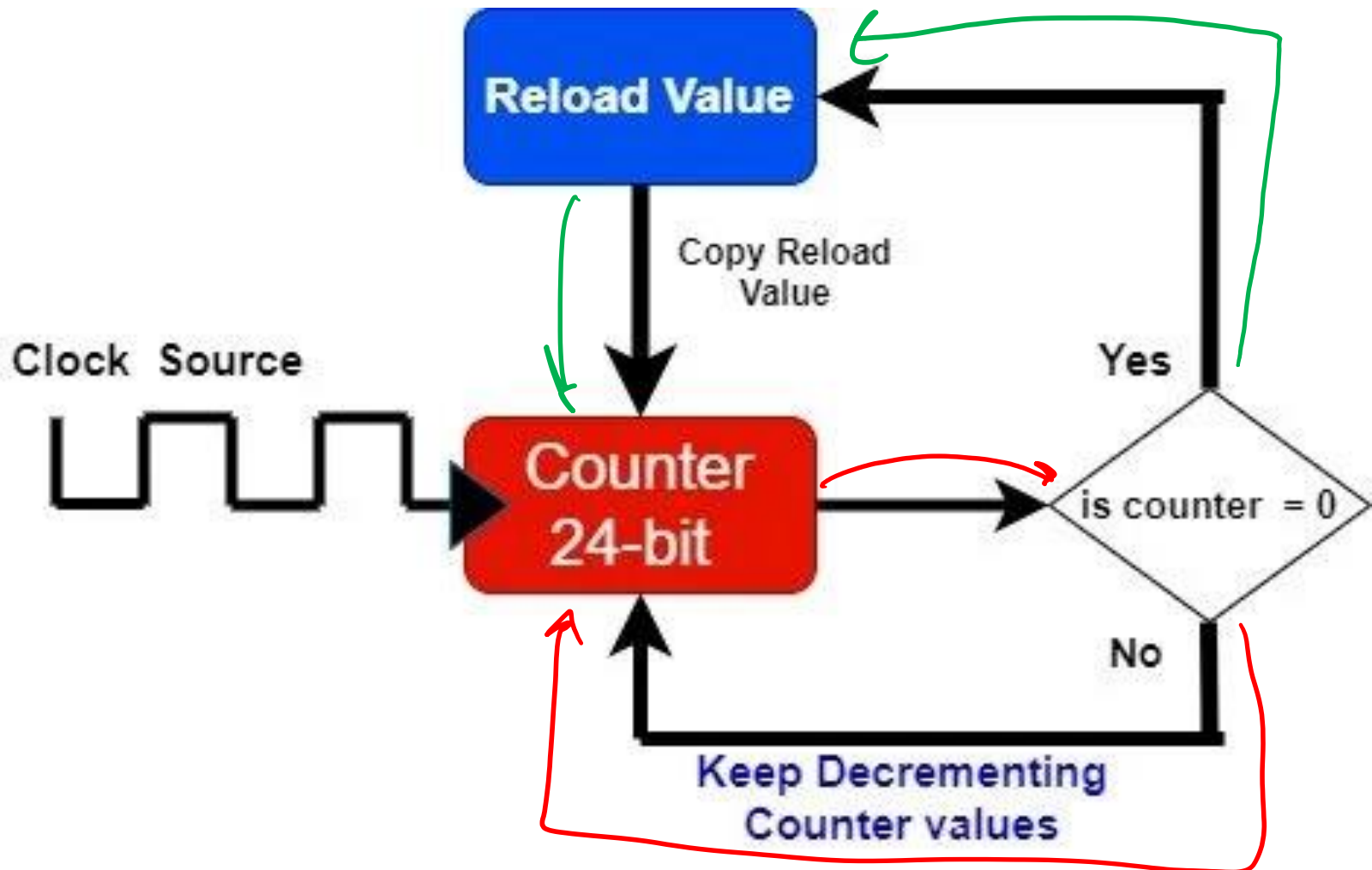
- SysTick is a simple counter on all Cortex-M that we can use to **create** time **delays** and generate **periodic interrupts**.
- Use of SysTick = your system will easily port to other Cortex-M microcontrollers.

Timer Fundamentals

- 24-bit down counter decrements at bus clock frequency
 - With a 48 MHz bus clock, decrements every 20.83 ns
- The counter, known as VAL, goes from $n \rightarrow 0$
 - i.e., $n, n - 1, n - 2, n - 3 \dots 2, 1, 0, n, n - 1 \dots$
- $VAL = (VAL - 1) \bmod (n+1)$

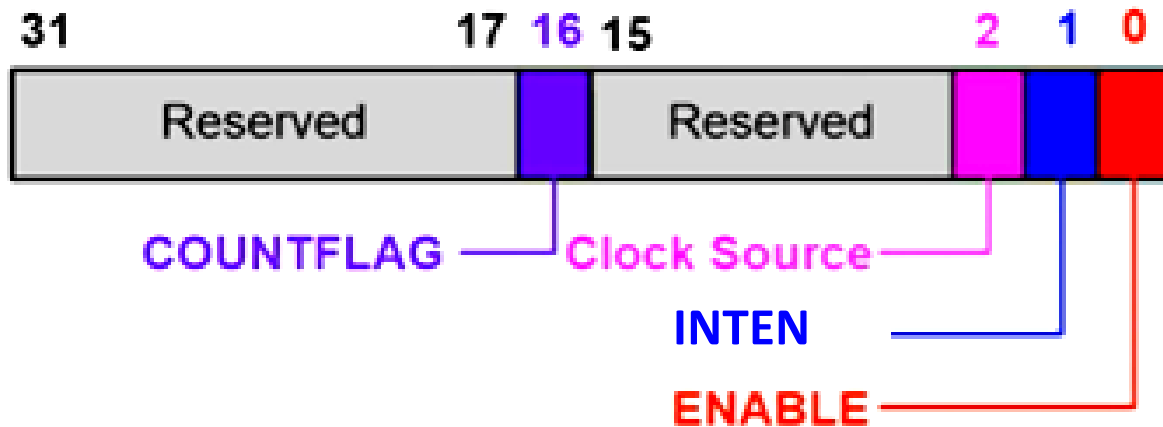


How counting occurs



Relevant counting registers: SysTick_CTRL

SysTick control and status register (SysTick_CTRL)



Register	Role
ENABLE	Turns SysTick on (1) or off (0)
Clock source	What core clock will be used (1 for system clock)
INTEN	Turns interrupts on (1) or off (0)
COUNTFLAG	Goes to 1 when VAL goes to 0

Relevant counting registers: SysTick_LOAD

SysTick reload value register (SysTick_LOAD)



Register	Role
RELOAD	What is the VAL reset to when we hit zero?

Relevant counting registers: SysTick_VAL

SysTick current value register (SysTick_VAL)



Register	Role
CURRENT	The current VAL

Initializing SysTick Timer Example

Step 1: Clear **ENABLE** bit in SysTick_CTRL to turn off SysTick during initialization.

Step 2: Specify the 24-bit **LOAD** value in SysTick_LOAD register.

Step 3: Clear the counter via SysTick_VAL (write any value to it)

Step 4: Set SysTick_CTRL values as follows:

CLK_SRC = 1 (bus clock is the only option)

INTEN = 0 for no interrupts

ENABLE = 1 to enable the counter

(Start the counter)

101

31-24	23-17	16	15-3	2	1	0
0	0	COUNT	0	CLK_SRC	INTEN	ENABLE
0	24-bit RELOAD value					
0	24-bit CURRENT value of SysTick counter					

Name
SysTick_CTRL
SysTick_LOAD
SysTick_VAL



SysTick Timer Initialization in assembly

```
SysTick_Init
; disable SysTick during setup
    LDR R1, =SysTick_CTRL
    MOV R0, #0                      ; Clear ENABLE bit (Step 1)
    STR R0, [R1]

; set reload to maximum reload value
    LDR R1, = SysTick_LOAD
    LDR R0, =0x00FFFFFF;           ; Specify RELOAD value (Step 2)
    STR R0, [R1]                   ; reload at maximum value

; writing any value to CURRENT clears it
    LDR R1, = SysTick_VAL
    MOV R0, #0
    STR R0, [R1]                   ; clear counter (Step 3)

; enable SysTick with core clock but no interrupts (for now!)
    LDR R1, = SysTick_CTRL         ; (Step 4)
    MOV R0, #0x0005                ; Enable but no interrupts
    STR R0, [R1]                   ; ENABLE and CLK_SRC bits set
    BX    LR
```



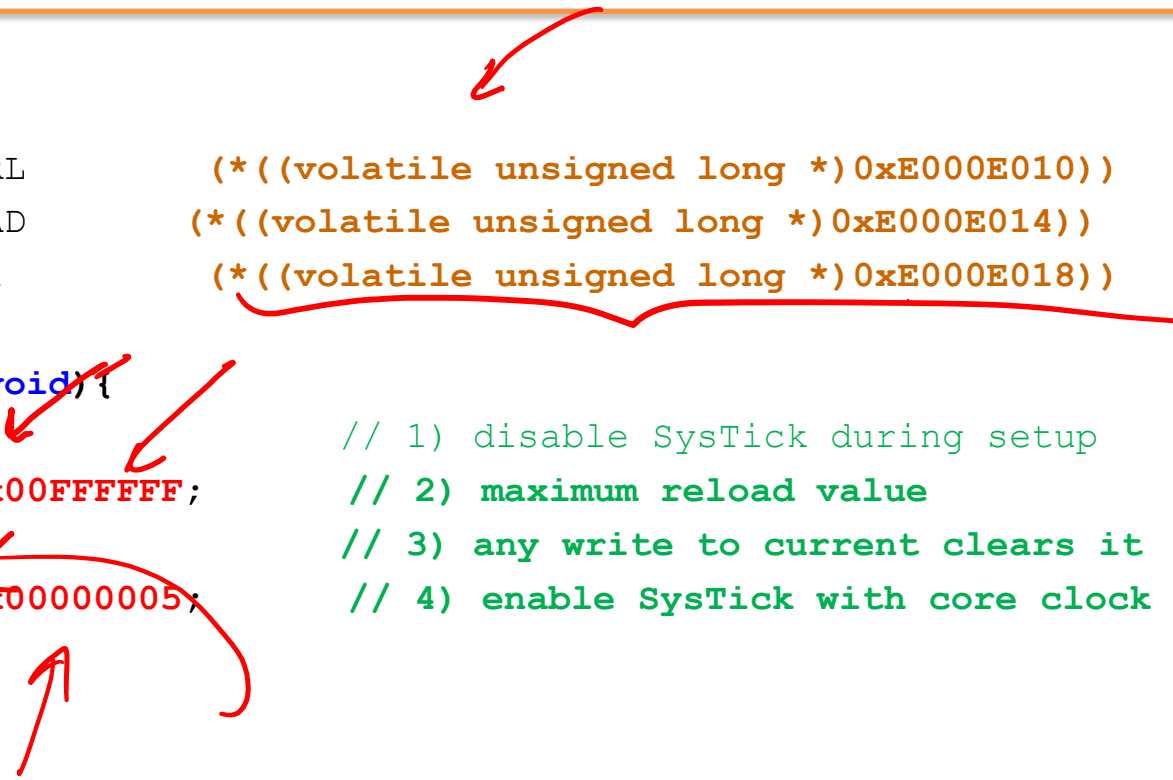
0101

SysTick Timer Initialization in C

```
#define SysTick_CTRL      (*((volatile unsigned long *)0xE000E010))
#define SysTick_LOAD      (*((volatile unsigned long *)0xE000E014))
#define SysTick_VAL       (*((volatile unsigned long *)0xE000E018))

void SysTick_Init(void){
    SysTick_CTRL = 0;
    SysTick_LOAD = 0x00FFFFFF;
    SysTick_VAL = 0;
    SysTick_CTRL = 0x00000005;
}
```

// 1) disable SysTick during setup
// 2) maximum reload value
// 3) any write to current clears it
// 4) enable SysTick with core clock



Measure Elapsed Time

```
Start = SysTick->VAL;  
SystemUnderTest();  
Stop = SysTick->VAL;  
Delta = 0x00FFFFFF & (Start-Stop);
```

Time to
execute?

What's a limitation with this approach?

At 48 MHz

- 24-bit precision ----- # of distinct measurements
- 20.83ns resolution -----Smallest change
- 349ms range -----Largest possible

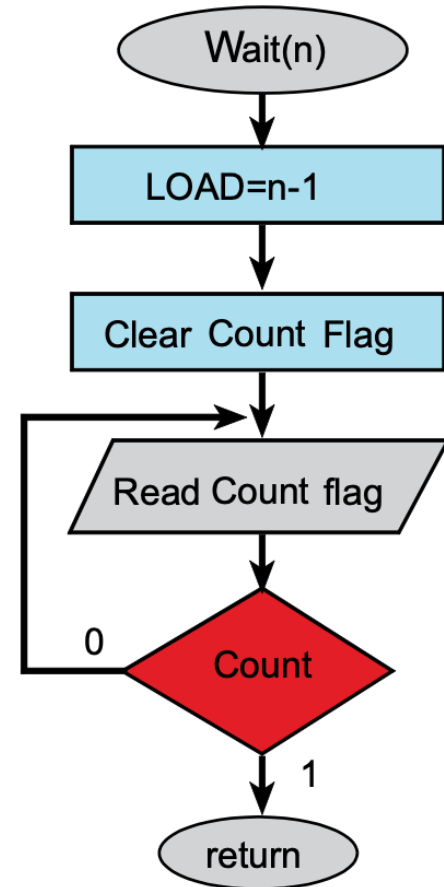
31-24	23-17	16	15-3	2	1	0
0	0	COUNT	0	CLK_SRC	INTEN	ENABLE
0	24-bit RELOAD value					
0	24-bit CURRENT value of SysTick counter					

Name
SysTick_CTRL
SysTick_LOAD
SysTick_VAL

SysTick Timer Wait

```
void SysTick_Wait(uint32_t n){  
    SysTick->LOAD = n-1;  
    SysTick->VAL = 0;    // clear Count  
    while((SysTick->CTRL&0x00010000)== 0){};  
}
```

At 48 MHz, it works up to 349ms
Doesn't work for n=0 or n=1



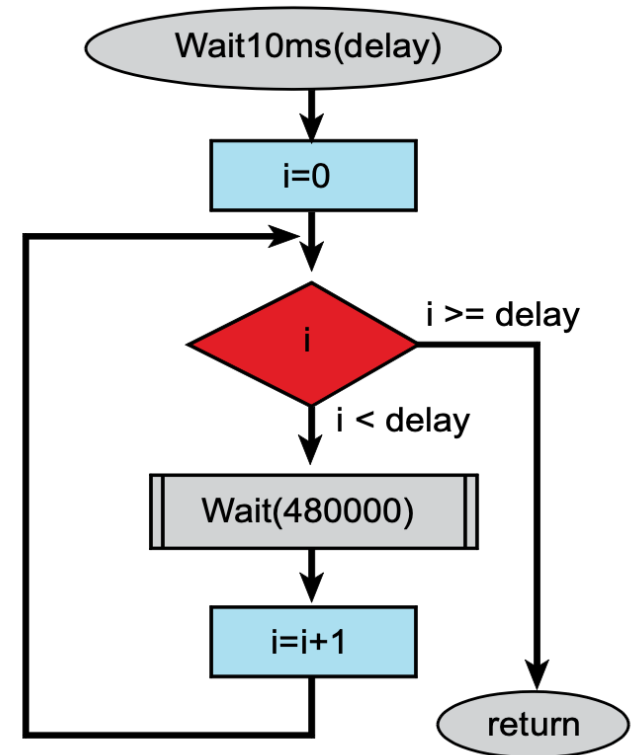
31-24	23-17	16	15-3	2	1	0
0	0	COUNT	0	CLK_SRC	INTEN	ENABLE
0	24-bit RELOAD value					
0	24-bit CURRENT value of SysTick counter					

Name
SysTick_CTRL
SysTick_LOAD
SysTick_VAL

Long Timer Wait

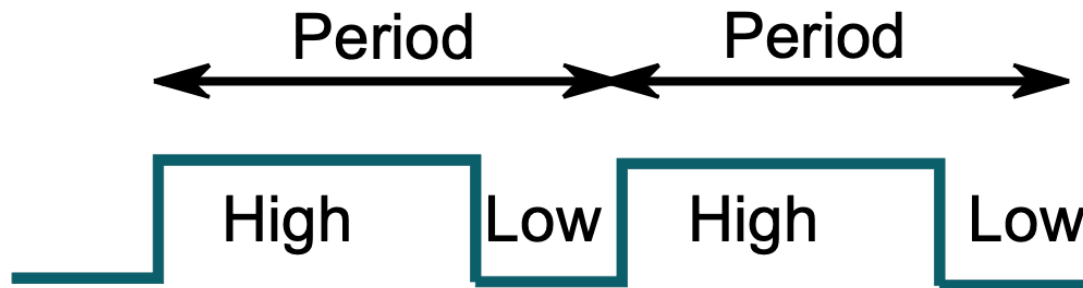
```
void SysTick_Wait10ms(uint32_t delay){  
    for(uint32_t i=0; i<delay; i++){  
        SysTick_Wait(480000);  
    }  
}
```

- 48 cycles is 1us
- 48,000 cycles is 1ms
- 480,000 cycles is 10ms



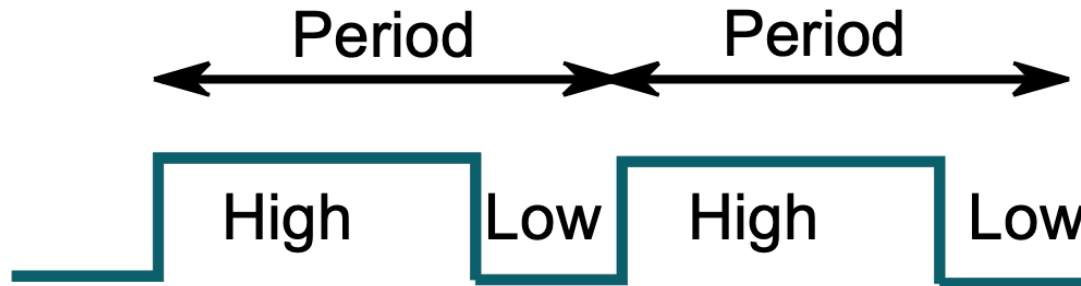
Pulse Width Modulation Concepts

- Pulse Width Modulation (PWM) is a method of representing an analog symbol as a series of regular waves
- Application:
 - Controlling motors, LEDs
 - Can be use for digital to analog conversion (DAC)
- Use SysTick to generate a waveform to modulate power delivery



Effects of duty cycle

- The ratio of high to low voltages in a PWM device
- $Duty\ cycle = \frac{High\ period}{High\ period + Low\ period} = \frac{High\ period}{Cycle\ period}$
- 60% duty cycle: signal is on 60% of the time

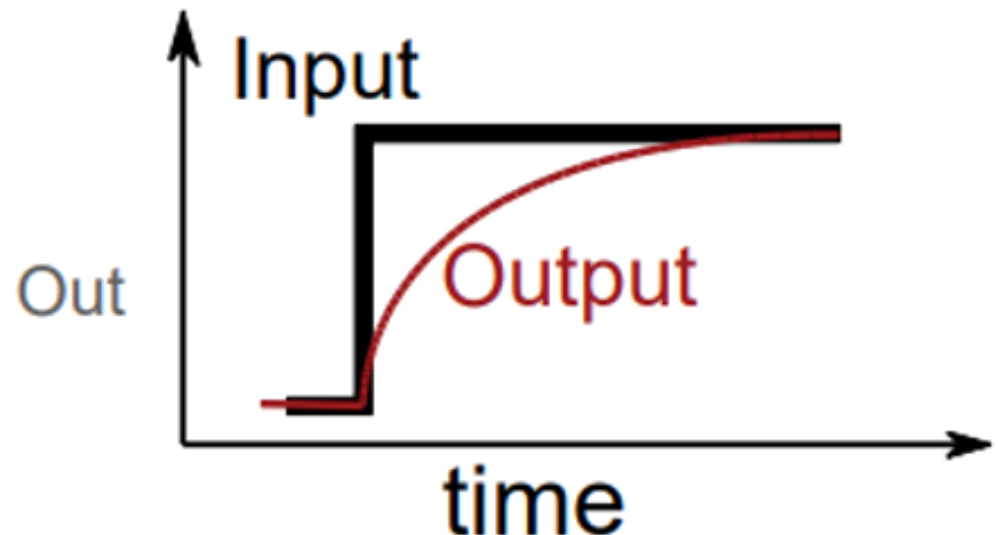


Input and output voltage of a duty cycle

- Changes in the input voltage does not translate into exact voltage output
- The output voltage depends on the following equation

$$Out(t) = A + Be^{-t/\tau}$$

- Each load device has a τ
 - τ is the time to reach 63.2% of final voltage
 - Example: HLMP-4700 LED: $\tau = 90ns$
 - Example: DC motor: $\tau = 100ms$



Coding PWM

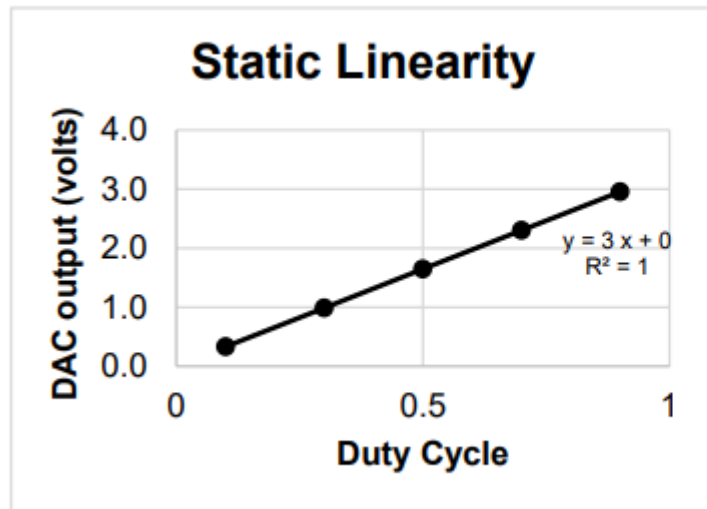
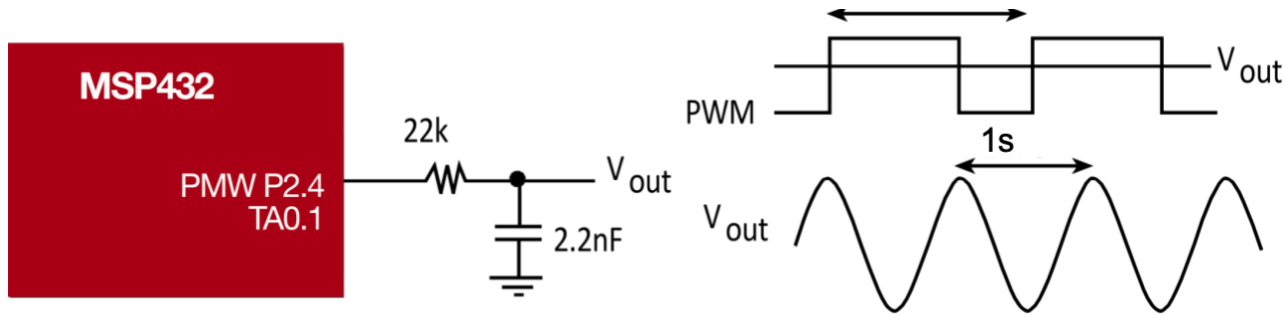
- What is the duty cycle

```
while(1) {  
    P1->OUT |= 0x01;    // red LED on  
    SysTick_Wait(High);  
    P1->OUT &= ~0x01;   // red LED off  
    SysTick_Wait(Low);  
}
```

- High + Low is a constant
- If fast enough, the device responds to the average

DAC with PWM

PWM + low pass filter



LPF cutoff:

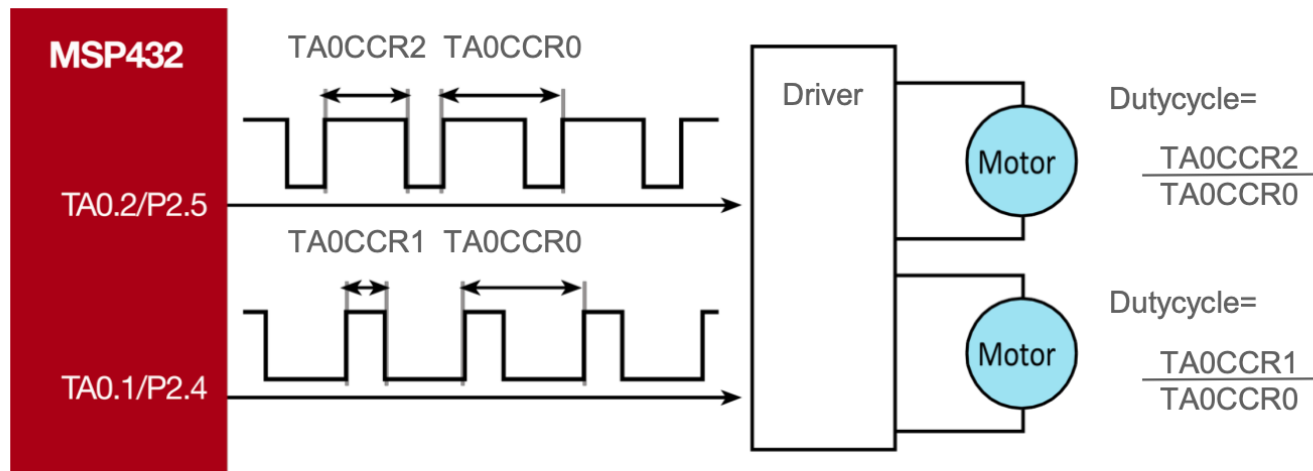
$$f_c = 1/(2\pi RC)$$

$f_c > \text{analog wave}$

$f_c < \text{digital frequency}$

Applications of PWM

- Control brightness of LEDs
- Control home appliances (120V @60Hz)
- Use it to make a DAC
- Control DC motor speed



Interface TI Launchpad with Motor driver TI DRV8838 using PWM