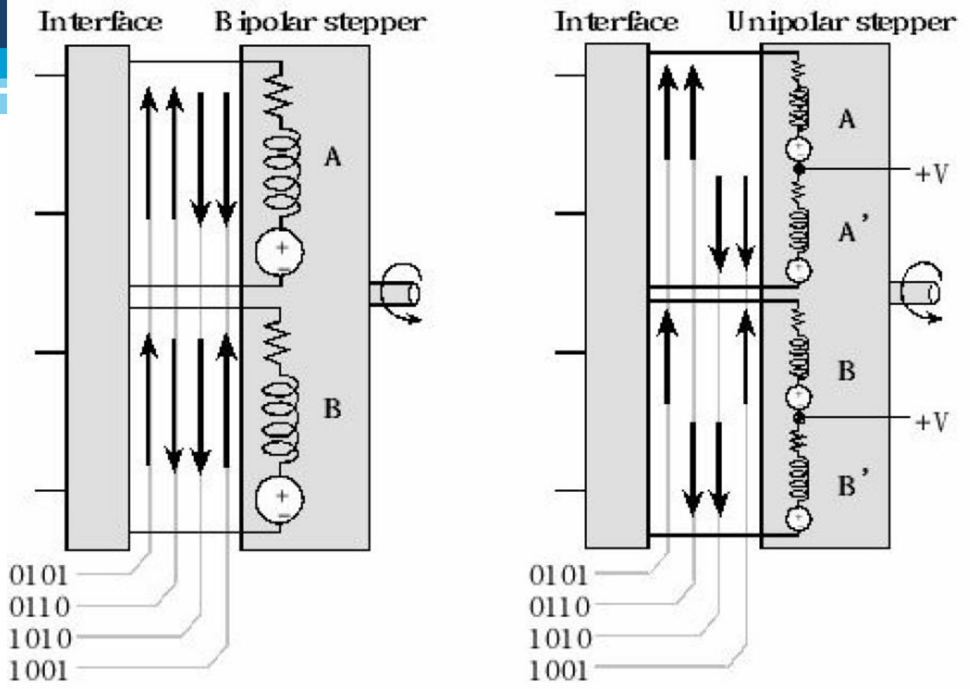
CSE 211: Introduction to Embedded Systems

Section 9

Stepper Motor

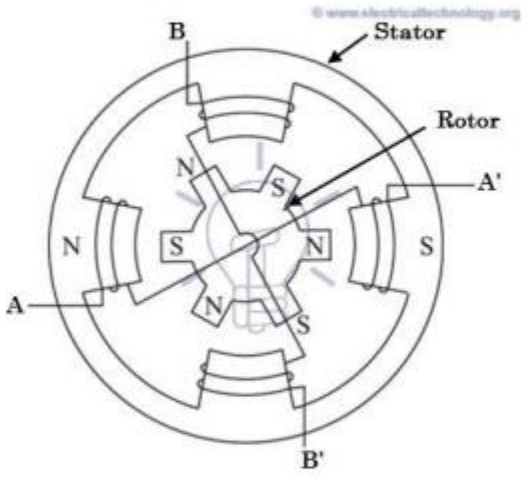
- They are used in printers to move paper and print heads.
- Stepper motors are used in applications where precise positioning is more important than high RPM, high torque, or high efficiency.

Stepper Motor



- To move a bipolar stepper, we reverse the direction of current through one (not both) of the coils
- To move it again, we reverse the direction of current in the other coil.
- Let the direction of the current be signified by up and down.
- To make the current go up, the microcontroller outputs a binary 01 to the interface.
- To make the current go down, it outputs a binary 10.
- Since there are 2 coils, four outputs will be required (e.g., 0101 means up/up).
- To spin the motor, we output the sequence 0101, 0110, 1010, 1001… over and over. Each output causes the motor to rotate a fixed angle.
- To rotate the other direction, we reverse the sequence (0101, 1001, 1010, 0110…).

Stepper Motor



$$Number\ of\ Steps\ per\ Rotation = \frac{360}{Step\ Angle}$$

$$RPM = 1000 * 60 * \frac{1}{Number\ of\ Steps\ per\ Rotation} * \frac{1}{Delay\ Between\ Steps}$$

RPM: Rotations Per Minute
Step Time in millisecond

Sheet 7

- If a stepper motor rotates with 200 steps / rotation, what is the required delay between steps to achieve 24 RPM?

Answer

- Step Time = 60 / (200*24) = 0.0125 sec = 12.5 ms

Sheet 7

- If a stepper motor rotates with 200 steps / rotation and the delay between steps is 20 milliseconds, what is the speed of the motor in RPM?

Answer

- 1 step takes 20 ms then, Steps per min = 60*1000/20 = 3000 steps
- Speed=3000/200=15 RPM

Sheet 7

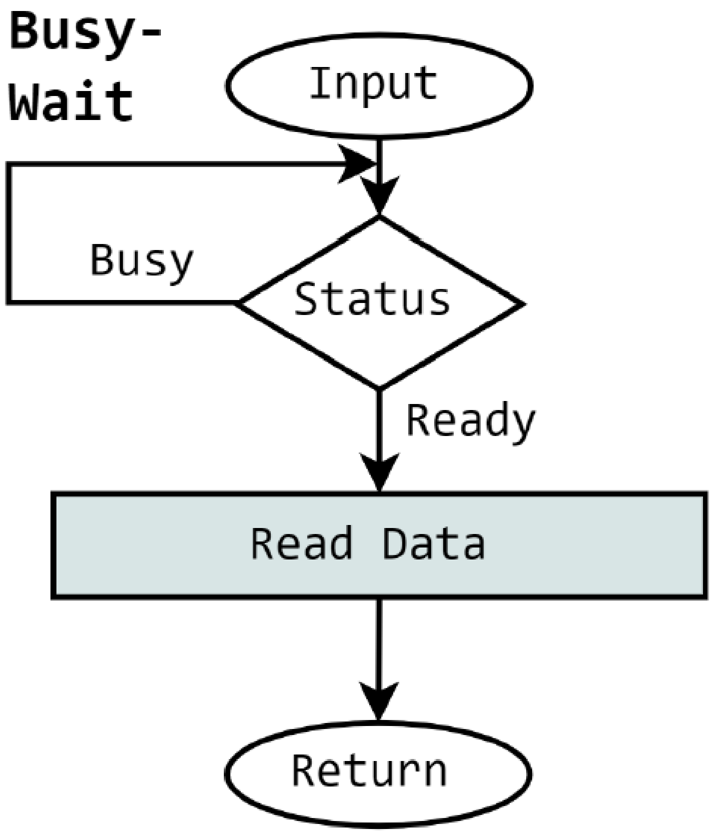
- For a motor with 200 steps / rotation. Write a C program to implement a stepper motor controller that spins this motor at 6 RPM. Assume that you have an already implemented “systick_wait_10ms ()” function that provides a delay each 10ms.

Answer

```c
void Delay(uint32_t delay){
uint32_t i;

for (i=0;i<delay;i++)
systick_wait_10ms();
}

int main(void){

SYSCTL_RCGCGPIO_R |= 0x08;
GPIO_PORTD_AMSEL_R &= ~0x0F;
GPIO_PORTD_PCTL_R &= ~0x0000FFFF;
GPIO_PORTD_DIR_R |= 0x0F; // 5) make PD3-0 out
GPIO_PORTD_AFSEL_R &= ~0x0F;// 6) disable alt func on PD3-0
GPIO_PORTD_DEN_R |= 0x0F; // 7) enable digital I/O on PD3-0

while(1){
GPIO_PORTD_DATA_R = 5;
Delay (5);
GPIO_PORTD_DATA_R = 6;
Delay (5);
GPIO_PORTD_DATA_R = 10;
Delay (5);
GPIO_PORTD_DATA_R = 9;
Delay (5);
}
 }
```

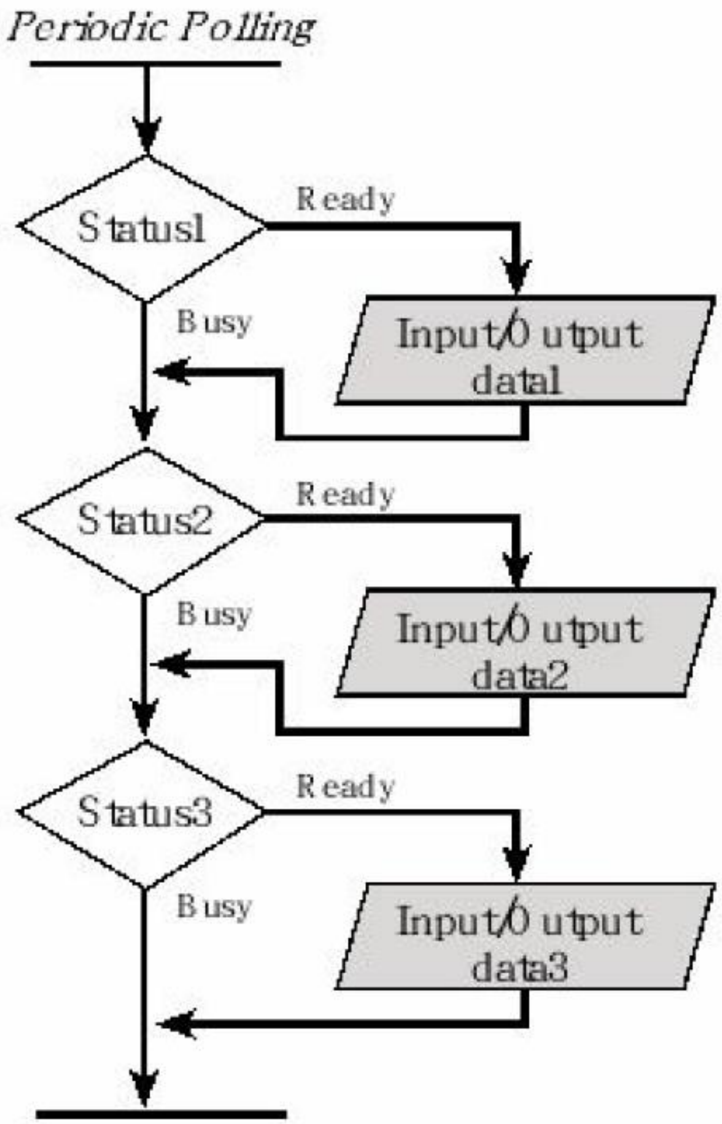
Busy-Wait

- In busy-wait synchronization, the main program polls the I/O devices continuously.



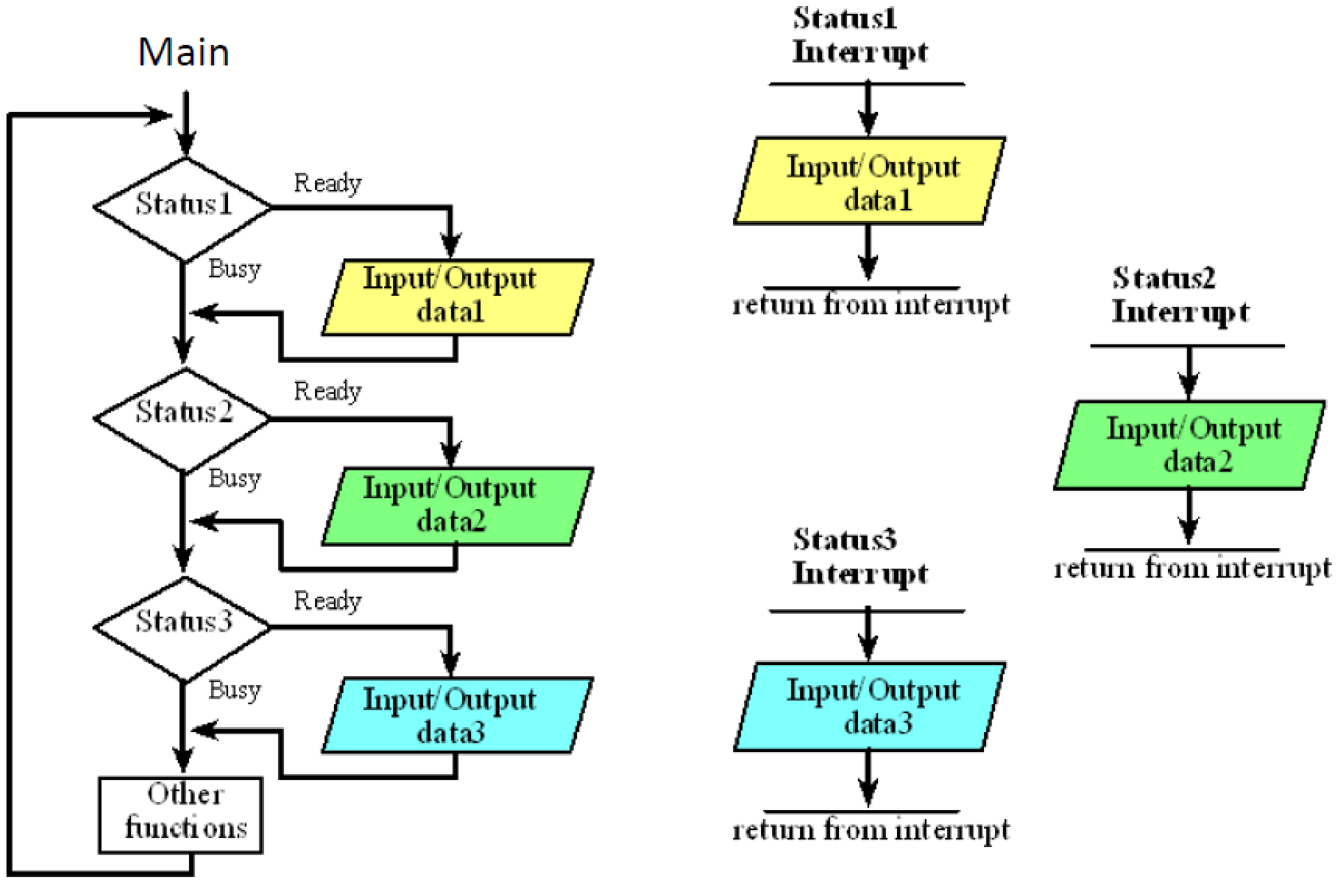
Periodic Polling

- With periodic polling, the I/O devices are polled on a regular basis.



Interrupts

- An interrupt is
the automatic transfer
of software execution in
response to a hardware event
that is asynchronous with
the current software execution.



Interrupts

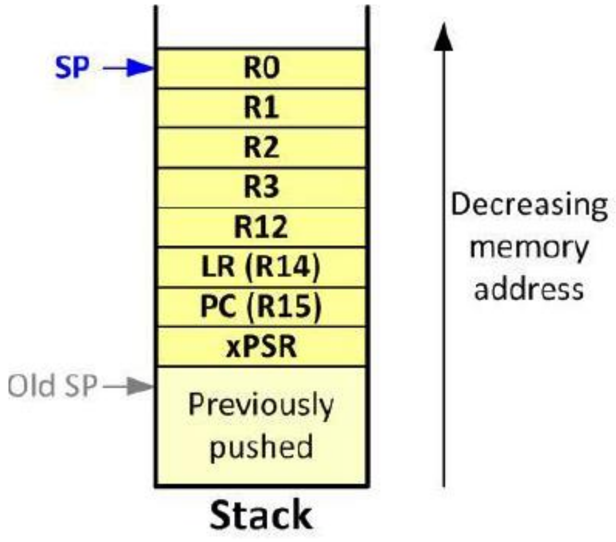
- An interrupt is a hardware/software triggered action and can be:
  - Periodic interrupt triggered by a hardware timer
  - I/O events (new data for IN, idle state for OUT)
  - Software interrupt

Exceptions

- In ARM, exceptions (including interrupts, resets, fault handlers,… etc.) have a 32-bit vector that points to the location of the ISR. Vector Table is stored in ROM starting from address 0x0000.0004.
- ROM location 0x0000.0000 has the initial stack pointer, and location 0x0000.0004 contains the initial program counter, which is called the reset vector. It points to a function called the reset handler, which is the first thing executed following reset.
- Startup code initializes the whole vector table by placing dummy ISRs.

Exception Handling

- Exceptions are managed by NVIC
- CPU state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the ISR
  - Registers: R0-R3, R12, LR, PC, PSR



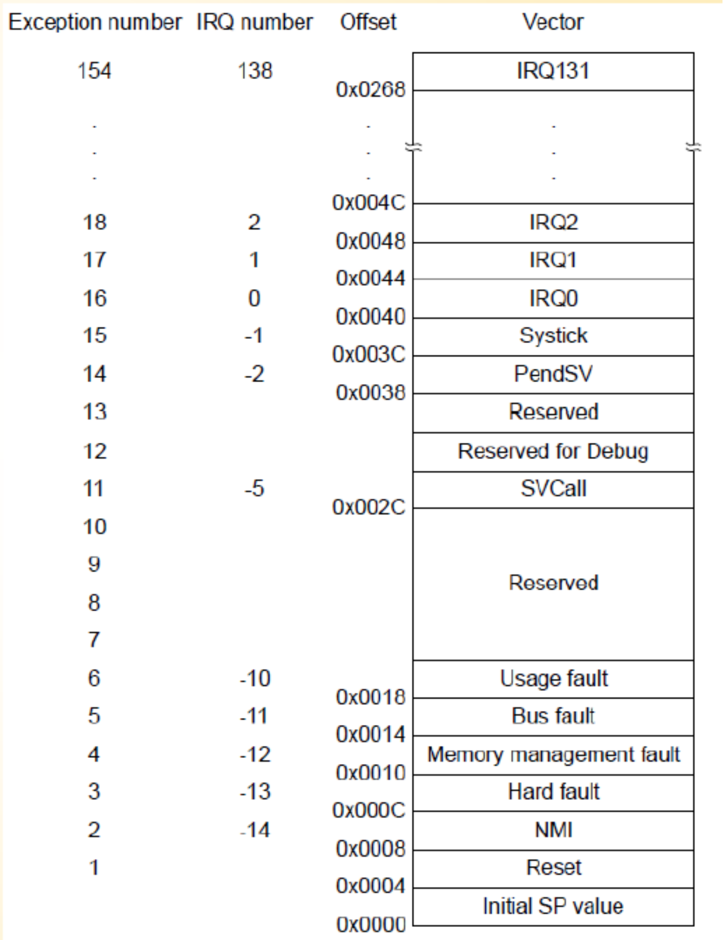
Context Switch

Actions taking to switch from one thread to another

- Hardware dependent
- In TM4C123:
  - Current instruction is finished
  - Eight registers are pushed on the stack (R0 R1 R2 R3 R12 LR PC , and PSR with the R0 on top)
  - LR is set to 0xFFFFFFF9
  - IPSR is set to the interrupt number
  - PC is loaded with the interrupt vector

Vector Table

- 32-bit vector(handler address) loaded into PC, while saving CPU context.
- Reset vector includes initial SP
- Peripherals use positive IRQ #s
- CPU exceptions use negative IRQ #s
- IRQ priorities user programmable
- NMI & Hard Fault priorities fixed



Nested Vectored Interrupt Controller (NVIC)

- Allows a programmable priority level of 0-7 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- There are 139 Interrupt Vectors.
- Registers:
- NVICENn where n is from 0 to 4
- NVICPRIn where n is from 0 to 34
- Systick is considered as a system exception not as an interrupt and it's priority is set using SYSPRI3

PRIn Register Bit Field
Bits 31:29
Bits 23:21
Bits 15:13
Bits 7:5

GPIO Interrupts

- GPIO related interrupts are configured through3 steps:
  - Individual pins
  - The whole Port (NVIC interrupt control registers).
  - Global interrupt configurations and controls for all peripherals by the processor (PRIMASK and BASEPRI registers).
  - Additionally, one also needs to configure the priority levels and enable the related interrupts for the selected GPIO Ports.
    - This can be handled by configuring NVIC Interrupt Control Registers.
      - Interrupt Priority Level Registers
      - Interrupt Set Enable Registers
    - These two register groups are used to set the priority levels for all peripherals and enable selected peripherals.

Control Registers for GPIO Pins

Register	Each Bit Value (Lowest 8-Bit) and Each Pin Function
GPIOIM	0: Interrupt is masked (disabled), 1: Interrupt is unmasked (enabled).
GPIOIS	Sensitivity 0: Edge, 1: Level.
GPIOIBE	0: Interrupt is controlled by GPIOIEV, 1: Both edges
GPIOIEV	0: A falling edge or a LOW level, 1: A rising edge or a HIGH level triggers an interrupt
GPIORIS	0: No interrupt occurred on the pin, 1: An interrupt is occurred on the pin. (RAW interrupt)
GPIOMIS	0: No interrupt occurred or the pin has been masked, 1: An interrupt has been occurred.
GPIOICR	0: No action, 1: Clear the corresponding edge-triggered interrupt

Interrupt Priority Level

- There are 35 priority level register groups
  - NVIC_PRI0_R ~ NVIC_PRI34_R
- Each group sets the interrupt priority for 4 peripherals where only upper 3 bits from each segment are used.
- Example: group 0 (NVIC_PRI0_R) concerns PORT A-D.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
		INTD				reserved				INTC				reserved		
Type	RW	RW	RW	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		INTB				reserved				INTA				reserved		
Type	RW	RW	RW	RO	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Interrupt Priority Level

- The priority group number n and the segment s are determined from the interrupt number IRQ as
  - n = IRQ /4 s = IRQ % 4
- For example, priority of PORT F interrupt (IRQ: 30) is configured in NVIC_PRI 7 _R, segment 2 (i.e., bits 21-23)

NVIC Interrupt Set Enable

- NVIC provides 5 Interrupt Set Enable Registers
  - NVIC_EN0_R ~ NVIC_EN4_R

- Each bit should be set to enable the interrupt for a specific peripheral

- Thus, the register n and bit b are determined by IRQ number as follows:
  - n = IRQ / 32 b = IRQ % 32 (IRQ – n*32)

- For example, PORT F (IRQ 30) is controlled by bit 30 in NVIC_EN 0 _R

Interrupt Setup

- Arm device (a device means to enable the source of interrupts)
- Set interrupt priority in NVIC (device specific)
- Enable interrupt in NVIC (device specific)
- Enable global interrupt (I = 0) in PRIMASK register Trigger the interrupt

```
EnableInterrupts    CPSIE  I     ;set I=0
                    BX     LR

DisableInterrupts   CPSID  I     ;set I=1
                    BX     LR
```

Sheet 8

- Write using C, a function to initialize port F pin 4 as digital input with negative edge triggered Interrupt with priority 2 and write the ISR which changes the color of RGB in a cyclic way from 000 to 111 then to 000 again.

```c
#include "tm4c123gh6pm.h"
#include <stdint.h>
#include "PLL.h"
void EnableInterrupts(void);
uint8_t counter=0;
uint32_t RGB_color [8] ={0x00,0x02,0x04,0x06,0x08,0x0A,0x0C,0x0E}
void PortF_init (void)
{
//port f pin4 configuration as digital input
SYSCTL_RCGCGPIO_R |= 0x20;
GPIO_PORTF_DIR_R &= ~0x10;
GPIO_PORTF_AFSEL_R &= ~0x10;
GPIO_PORTF_DEN_R |= 0x10;
GPIO_PORTF_PCTL_R &= ~0x000F0000;
GPIO_PORTF_AMSEL_R &= ~0x10;
GPIO_PORTF_PUR_R |= 0x10;
// negative edge triggered (falling ) configuration
GPIO_PORTF_IS_R &= ~0x10;
GPIO_PORTF_IBE_R &= ~0x10;
GPIO_PORTF_IEV_R &= ~0x10;
// configuration to enable interrupts
//arm interrupt portf pin4 (switch)
GPIO_PORTF_IM_R |= 0x10;
//enable IRQ from port f ( IRQ 30 > EN0 bit 30)
NVIC_EN0_R |= (1 << 30); // EN0 bit 30
//enable interrupts
EnableInterrupts();
//Setting portf priority (use any method of the following lines )
NVIC_PRI7_R = (NVIC_PRI7_R&0xFF00FFFF)|0x00400000 ;
NVIC_PRI7_R = (NVIC_PRI7_R & 0xFF00FFFF) | (2 << 21);
NVIC_PRI7_R = (NVIC_PRI7_R & 0xFF00FFFF) | (1 << 22)
}
```

```c
void GPIOF_Handler(void)
{
GPIO_PORTF_ICR_R |=0x10;
GPIO_PORTF_DATA_R =RGB_color [counter];
Counter++;
If (counter ==8) counter=0;
}
```

Sheet 8

- Write using C, a function to initialize SysTick periodic interrupt each 10 ms with priority 1 (assume system clock is 80 MHz) and write an ISR which increments a global variable “cnt10ms” by 1.

Answer

```
# define Period 800000 //counts=freq*delay =(80 *10^6) *(10*10^-3)
Uint32_t cnt10ms=0;
Void systick_interrupt_init (void)
{
NVIC_ST_CTRL_R = 0; // disable SysTick during setup
NVIC_ST_RELOAD_R = period - 1; // reload value
NVIC_ST_CURRENT_R = 0; // any write to current clears it
NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x20000000; //priority 1 bits 31-29
NVIC_ST_CTRL_R = 0x00000007; // enable with core clock and interrupts
EnableInterrupts();
}
void SysTick_Handler(void)
{
cnt10ms = cnt10ms + 1;
}
```

Sheet 8

Write using C, a main function that calls 3 functions which are task1(), task2(), and task3().

Task1 should run every 10 ms, task 2 should run every 20 ms, and task 3 should run every 30 ms. Assume there is a global variable “cnt10ms” that is initialized by 0 and is incremented by 1 every 10 ms.

```c
#include "tm4c123gh6pm.h"
#include "PLL.h"
void EnableInterrupts(void);
void SysTick_Init(void);
void SysTick_Handler(void);
uint32_t cnt10ms = 0;
uint32_t dummy1 = 0; //dummy variables for tasks
uint32_t dummy2 = 0;
uint32_t dummy3 = 0;
bool run_flag = true;
void SysTick_Init(void)
{
NVIC_ST_CTRL_R = 0;
NVIC_ST_RELOAD_R = 800000 - 1;
NVIC_ST_CURRENT_R = 0;
NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x20000000; // priority 1
NVIC_ST_CTRL_R = 0x0007;
}
void SysTick_Handler(void)
{ cnt10ms++;
run_flag = true;
}
void task1(void)
{ dummy1++; //dummy instruction }
void task2(void)
{ dummy2++; }
void task3(void)
{ dummy3++; }
```

```c
int main(void){
PLL_Init();
SysTick_Init();
while(1){
if(run_flag){
task1();
if((cnt10ms % 2) == 0){
task2();
}
if((cnt10ms % 3) == 0){
task3();
}
run_flag = false;
}
}
}
```

Thank You