EIE3105: ARM Programming – Timer/Counter, Interrupt and Serial Port Communication

Dr. Lawrence Cheung

Semester 1, 2021/22

Topics

- Timer/Counter
- Interrupt
- Serial Port Communication

Timer/Counter

- Timer/Counter (7 timers)

Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary outputs
TIM1	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	Yes
TIM2, TIM3, TIM4	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No

Timer/Counter

- General-purpose timers (TIM2, TIM3 and TIM4)
  - 16-bit auto-reload up/down counter
  - 16-bit prescaler
  - 4 independent channels
    - Input capture
    - Output compare
    - PWM generation
    - One-pulse mode output

Timer/Counter

- Advanced-control timer (TIM1)
  - Same as general-purpose timers except it can general complementary PWM outputs with programmable inserted dead-times

Timer/Counter

- Watchdog timers (Independent and Window)
  - Detect and recover from computer malfunctions.
  - During normal operation, the computer regularly resets the watchdog timer to prevent “time out”.
  - When the computer fails to reset the watchdog (e.g., hardware failure, program error), the timer will elapse and generate a timeout signal.
  - The timeout signal is used to initiate corrective action(s).
  - Independent: reset a device or an application
  - Window: reset a device

Timer/Counter

- SysTick timer
  - This timer is dedicated for OS.
  - Can be used as a standard down-counter.
  - 24-bit
  - Auto-reload capability
  - Maskable system interrupt generation when the counter reaches zero
  - Programmable clock source

C Programming for ARM

- Example 1: Timer (Counter up)
  - Flash the on-board LED (PA5) in every second.

```
#include "stm32f10x.h"                    // Device header

int main(void) {

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

   //GPIO set up for PA5 (on board LED)
   GPIO_InitTypeDef GPIO_InitStructure;
   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_5;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
```

C Programming for ARM

```
//Timer 2 set up
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

TIM_TimeBaseInitTypeDef timerInitStructure;
timerInitStructure.TIM_Prescaler = 18000-1;  //1/(72Mhz/18000)=0.25ms
timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
timerInitStructure.TIM_Period = 4000-1;  //0.25ms*4000 = 1s
timerInitStructure.TIM_ClockDivision = 0; //TIM_CKD_DIV1;
timerInitStructure.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM2, &timerInitStructure);
TIM_Cmd(TIM2, ENABLE);

//Enable update event for Timer2
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
```

C Programming for ARM

```
    char state = 0;
    while(1) {
        if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
            if(state == 0)  {
                GPIO_ResetBits(GPIOA, GPIO_Pin_5);
                state = 1;
            } else {
                GPIO_SetBits(GPIOA, GPIO_Pin_5);
                state = 0;
            }
            TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        }
    }
}
```

Timer/Counter

- General-purpose timers: Initialization

```
typedef struct
{
   uint16_t TIM_Prescaler;
   uint16_t TIM_CounterMode;
   uint16_t TIM_Period;
   uint16_t TIM_ClockDivision;
   uint8_t TIM_RepetitionCounter;
} TIM_TimeBaseInitTypeDef;
```

Timer/Counter

- TIM_Prescaler = clock cycle period
  - Example: 18000 – 1 => 1 / (72MHz / 18000) = 0.25 ms
  - Range: 0x0000 to 0xFFFF.
- TIM_Period = the period value to be loaded into the active Auto-Reload Register.
  - Example: 4000 – 1 => 0.25 ms * 4000 = 1 s
- TIM_ClockDivision = the division ratio between the time clock and the dead-time and sampling clock
  - Nothing related to prescaler
  - Default: TIM_CKD_DIV1 (0x00)

Timer/Counter

- TIM_CounterMode = mode
  - TIM_CounterMode_Up: count from 1 to TIM_Period
  - TIM_CounterMode_Down: count from TIM_Period to 1
  - TIM_CounterMode_CenterAligned1: use in PWM, explain later
  - TIM_CounterMode_CenterAligned2 : use in PWM, explain later
  - TIM_CounterMode_CenterAligned3 : use in PWM, explain later
- TIM_RepetitionCounter = repetition counter value
  - Use in PWM, explain later
  - Default: 0

Timer/Counter

- General-purpose timers: Operations

```
void TIM_TimeBaseStructInit (TIM_TimeBaseInitTypeDef
*TIM_TimeBaseInitStruct)
```

  - Fill each TIM_TimeBaseInitStruct member with its default value.

```
void TIM_Cmd(TIM_TypeDef * TIMx,
      FunctionalState NewState)
```

  - Enable or disable the specified TIM peripheral.
  - NewState: new state of the TIM interrupt
    - ENABLE or DISABLE

Timer/Counter

```
void TIM_ITConfig(TIM_TypeDef * TIMx,
                  uint16_t TIM_IT,
                  FunctionalState NewState)
```

- Enable or disable the specified TIM interrupt (also the timer flag).
- TIMx: select a timer.

Timer/Counter

- TIM_IT: specify the TIM interrupt source to be enabled or disabled
  - TIM_IT_Update: TIM update Interrupt source
  - TIM_IT_CC1: TIM Capture Compare 1 Interrupt source
  - TIM_IT_CC2: TIM Capture Compare 2 Interrupt source
  - TIM_IT_CC3: TIM Capture Compare 3 Interrupt source
  - TIM_IT_CC4: TIM Capture Compare 4 Interrupt source
  - TIM_IT_COM: TIM Commutation Interrupt source
  - TIM_IT_Trigger: TIM Trigger Interrupt source
  - TIM_IT_Break: TIM Break Interrupt source

Timer/Counter

```
ITStatus TIM_GetITStatus(TIM_TypeDef * TIMx,
                         uint16_t TIM_IT)
```

- Check whether the TIM interrupt (flag) has occurred or not.
- ITStatus: SET or RESET

```
void TIM_ClearITPendingBit(TIM_TypeDef * TIMx,
                           uint16_t TIM_IT)
```

- Clears the TIMx's interrupt pending bits
- You need to clear the interrupt (flag) by yourself.

Timer/Counter

```
void GPIO_ResetBits(GPIO_TypeDef * GPIOx,
                                uint16_t GPIO_Pin)
```

- Clear the selected data port bits.

```
void GPIO_SetBits(GPIO_TypeDef * GPIOx,
                                uint16_t GPIO_Pin)
```

- Set the selected data port bits.

C Programming for ARM

- Example 2: Timer (Counter down)
  - Flash the on-board LED (PA5) in every second.

```
#include "stm32f10x.h"                    // Device header

int main(void) {

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

   //GPIO set up for PA5 (on board LED)
   GPIO_InitTypeDef GPIO_InitStructure;
   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_5;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
```

C Programming for ARM

```
//Timer 2 set up
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

TIM_TimeBaseInitTypeDef timerInitStructure;
timerInitStructure.TIM_Prescaler = 18000-1;  //1/(72Mhz/18000)=0.25ms
timerInitStructure.TIM_CounterMode = TIM_CounterMode_Down;
timerInitStructure.TIM_Period = 4000-1;  //0.25ms*4000 = 1s
timerInitStructure.TIM_ClockDivision = 0; //TIM_CKD_DIV1;
timerInitStructure.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM2, &timerInitStructure);
TIM_Cmd(TIM2, ENABLE);

//Enable update event for Timer2
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
```

C Programming for ARM

```
    char state = 0;
    while(1) {
        if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
            if(state == 0)  {
                GPIO_ResetBits(GPIOA, GPIO_Pin_5);
                state = 1;
            } else {
                GPIO_SetBits(GPIOA, GPIO_Pin_5);
                state = 0;
            }
            TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        }
    }
}
```

C Programming for ARM

- Example 3: Counter
  - Press the button in PA1 six times to toggle the red LED in PB8 and the green LED in PB9.

```
#include "stm32f10x.h"    // Device header
#include "stdbool.h"      // For Boolean data type

//TI2 PA1 Tim2 Ch2
#define BUTTON_RCC_GPIO   RCC_APB2Periph_GPIOA
#define BUTTON_GPIO       GPIOA
#define BUTTON_GPIO_PIN   GPIO_Pin_1

#define L3_RCC_GPIO  RCC_APB2Periph_GPIOB
#define L3_GPIO      GPIOB
#define L3_R_PIN     GPIO_Pin_8
#define L3_G_PIN     GPIO_Pin_9

bool wait = true;
```

C Programming for ARM

```
int main(void) {

    // GPIO clock for I/O (PA1)
    RCC_APB2PeriphClockCmd(BUTTON_RCC_GPIO, ENABLE);
    RCC_APB2PeriphClockCmd(L3_RCC_GPIO, ENABLE);

    // Configure I/O for L3 (PB8 and PB9)
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.GPIO_Pin = L3_R_PIN | L3_G_PIN;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(L3_GPIO, &GPIO_InitStruct);
```

C Programming for ARM

```
    //Timer 2 set up
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    TIM_TimeBaseInitTypeDef timerInitStructure;
    timerInitStructure.TIM_Prescaler = 0;
    timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    timerInitStructure.TIM_Period = 5;
    timerInitStructure.TIM_ClockDivision = 0;
    timerInitStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM2, &timerInitStructure);
    TIM_Cmd(TIM2, ENABLE);

    TIM_TIxExternalClockConfig(TIM2, TIM_TIxExternalCLK1Source_TI2,
TIM_ICPolarity_Rising, 0);

    //Enable update event for Timer2
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
```

C Programming for ARM

```
    bool state = 0;
    while(1) {
        if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
          if(state) {
                            GPIO_SetBits(L3_GPIO, L3_R_PIN);
                            GPIO_ResetBits(L3_GPIO, L3_G_PIN);
                            state = 0;
          } else {
                            GPIO_ResetBits(L3_GPIO, L3_R_PIN);
                            GPIO_SetBits(L3_GPIO, L3_G_PIN);
                            state = 1;
          }

          TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        }
    }
}
```

Timer/Counter

- General-purpose Counters: Operations

```
void TIM_TIxExternalClockConfig(TIM_TypeDef * TIMx,
      uint16_t TIM_TIxExternalCLKSource,
      uint16_t TIM_ICPolarity,
      uint16_t ICFilter)
```

  - Configure the TIMx Trigger as External Clock.
    - TIM_TIxExternalCLKSource: Trigger source
      - TIM_TIxExternalCLK1Source_TI1ED: TI1 Edge Detector
        - » Both falling or rising edges are triggered.
      - TIM_TIxExternalCLK1Source_TI1: Filtered Timer Input 1
      - TIM_TIxExternalCLK1Source_TI2: Filtered Timer Input 2
        - » Either the falling or rising edge is triggered (TI1 and TI2).

Timer/Counter

- Pin allocations
  - CH1/ETR/PA0, CH2/PA1, CH3/PA2, CH4/PA3
  - ETR = Edge Triggered Timer Input
- For other timers/counters
  - TIM1: ETR/PA12, CH1/PA8, CH2/PA9, CH3/PA10, CH4/PA11, BKIN/PB12
  - TIM3: CH1N/PB13, CH2N/PB14, CH3N/PB15,CH1/PA6, CH2/PA7, CH3/PB0, CH4/PB1
  - TIM4: TIM4_CH1/PB6, TIM4_CH2/PB7, TIM4_CH3/PB8, TIM4_CH4/PB9

Timer/Counter

- TIM_ICPolarity = Specify the TIMx Polarity.
  - Trigger either the rising or falling edge.
  - TIM_ICPolarity_Rising
  - TIM_ICPolarity_Falling
- TIM_ICFilter = Specify the filter value.
  - Use in Input Capture
  - Value: 0x0 to 0xF
  - Default = 0

Interrupt

- In previous chapter, we have been using interrupts to implement the delay function.
  - We configure SysTick to trigger an interrupt every milli-second.
  - SysTick_Handler is an interrupt handler.
    - It is executed when the SysTick interrupt occurs.
    - It decrements the variable msTicks and then returns control to the application program.

Interrupt

- Vector Names

```
Reset_Handler                   DMA1_Channel6_IRQHandler
NMI_Handler                     DMA1_Channel7_IRQHandler
HardFault_Handler               ADC1_IRQHandler
MemManage_Handler               EXTI9_5_IRQHandler
BusFault_Handler                TIM1_BRK_TIM15_IRQHandler
UsageFault_Handler              TIM1_UP_TIM16_IRQHandler
SVC_Handler                     TIM1_TRG_COM_TIM17_IRQHandler
DebugMon_Handler                TIM1_CC_IRQHandler
PendSV_Handler                  TIM2_IRQHandler
SysTick_Handler                 TIM3_IRQHandler
WWDG_IRQHandler                 TIM4_IRQHandler
PVD_IRQHandler                  I2C1_EV_IRQHandler
TAMPER_IRQHandler               I2C1_ER_IRQHandler
RTC_IRQHandler                  I2C2_EV_IRQHandler
FLASH_IRQHandler                I2C2_ER_IRQHandler
RCC_IRQHandler                  SPI1_IRQHandler
EXTI0_IRQHandler                SPI2_IRQHandler
EXTI1_IRQHandler                USART1_IRQHandler
EXTI2_IRQHandler                USART2_IRQHandler
EXTI3_IRQHandler                USART3_IRQHandler
EXTI4_IRQHandler                EXTI15_10_IRQHandler
DMA1_Channel1_IRQHandler        RTCAlarm_IRQHandler
DMA1_Channel2_IRQHandler        CEC_IRQHandler
DMA1_Channel3_IRQHandler        TIM6_DAC_IRQHandler
DMA1_Channel4_IRQHandler        TIM7_IRQHandler
DMA1_Channel5_IRQHandler
```

Interrupt

- Interrupt Request (IRQ) Handler (example: USART1)

```
void USART1_IRQHandler(void) {
    // Check interrupt cause
...
    // Clear interrupt cause
}
```

Interrupt

- NVIC (Nested Vectored Interrupt Controller)
  - Priority and sub-priority configuration
  - Enable (disable) interrupt.
  - Set/Clear interrupt pending bit.

- The simplest way to setup an interrupt:

```
void NVIC_EnableIRQ(IRQn_Type IRQn)
```

  - Enable the specified device specific interrupt IRQn
    - IRQn = Interrupt number

Interrupt

- List of all IRQn

WWDG_IRQn	= 0,	DMA1_Channel5_IRQn	= 15,	TIM3_IRQn	= 29,
PVD_IRQn	= 1,	DMA1_Channel6_IRQn	= 16,	TIM4_IRQn	= 30,
TAMPER_IRQn	= 2,	DMA1_Channel7_IRQn	= 17,	I2C1_EV_IRQn	= 31,
RTC_IRQn	= 3,	ADC1_2_IRQn	= 18,	I2C1_ER_IRQn	= 32,
FLASH_IRQn	= 4,	USB_HP_CAN1_TX_IRQn	= 19,	I2C2_EV_IRQn	= 33,
RCC_IRQn	= 5,	USB_LP_CAN1_RX0_IRQn	= 20,	I2C2_ER_IRQn	= 34,
EXTI0_IRQn	= 6,	CAN1_RX1_IRQn	= 21,	SPI1_IRQn	= 35,
EXTI1_IRQn	= 7,	CAN1_SCE_IRQn	= 22,	SPI2_IRQn	= 36,
EXTI2_IRQn	= 8,	EXTI9_5_IRQn	= 23,	USART1_IRQn	= 37,
EXTI3_IRQn	= 9,	TIM1_BRK_IRQn	= 24,	USART2_IRQn	= 38,
EXTI4_IRQn	= 10,	TIM1_UP_IRQn	= 25,	USART3_IRQn	= 39,
DMA1_Channel1_IRQn	= 11,	TIM1_TRG_COM_IRQn	= 26,	EXTI15_10_IRQn	= 40,
DMA1_Channel2_IRQn	= 12,	TIM1_CC_IRQn	= 27,	RTCAlarm_IRQn	= 41,
DMA1_Channel3_IRQn	= 13,	TIM2_IRQn ,	= 28,	USBWakeUp_IRQn	= 42
DMA1_Channel4_IRQn	= 14,				

- IRQ handler: replace ‘n’ by “Handler”
  - Example: The name of the IRQ handler of TIM2_IRQn is TIM2_IRQHandler

Interrupt

- Example 4: Timer interrupt
  - To toggle the on-board LED (PA5) every second by using TIM2 interrupt

```
#include "stm32f10x.h" // Device header

int main(void) {

   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

   //GPIO set up for PA5 (on board LED)
   GPIO_InitTypeDef GPIO_InitStructure;
   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_5;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_Out_PP;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Interrupt

```
    //Timer 2 set up
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    TIM_TimeBaseInitTypeDef timerInitStructure;
    timerInitStructure.TIM_Prescaler = 18000 - 1;  //1/(72Mhz/18000)=0.25ms
    timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    timerInitStructure.TIM_Period = 4000 - 1;  //0.25ms*4000 = 1s
    timerInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    timerInitStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM2, &timerInitStructure);
    TIM_Cmd(TIM2, ENABLE);

    //Enable update event for Timer2
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
    NVIC_EnableIRQ(TIM2_IRQn);

    while(1);
}
```

Interrupt

```
char state = 0;

//Interrupt Subroutine
void TIM2_IRQHandler(void) {
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
      if(state == 0)  {
          GPIO_ResetBits(GPIOA, GPIO_Pin_5);
          state = 1;
      } else {
          GPIO_SetBits(GPIOA, GPIO_Pin_5);
          state = 0;
      }
      TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}
```

Interrupt

- Even in the interrupt service routine (interrupt handler), you need to check the status of the interrupt because some interrupt handlers may be shared by more than one interrupt.
- Unlike AVR, you need to clear the interrupt flag by yourself.

Interrupt

- Example 5: Counter interrupt
  - Press the button in PA1 six times to toggle the red LED in PB8 and the green LED in PB9 by using counter interrupt.

```
#include "stm32f10x.h"                    // Device header
#include "stdbool.h"

//TI2 PA1 Tim2 Ch2
#define BUTTON_RCC_GPIO   RCC_APB2Periph_GPIOA
#define BUTTON_GPIO       GPIOA
#define BUTTON_GPIO_PIN   GPIO_Pin_1

#define L3_RCC_GPIO  RCC_APB2Periph_GPIOB
#define L3_GPIO      GPIOB
#define L3_R_PIN     GPIO_Pin_8
#define L3_G_PIN     GPIO_Pin_9
```

Interrupt

```
bool state = true;

int main(void) {

    GPIO_InitTypeDef GPIO_InitStruct;

    // GPIO clock for I/O
    RCC_APB2PeriphClockCmd(BUTTON_RCC_GPIO, ENABLE);
         RCC_APB2PeriphClockCmd(L3_RCC_GPIO, ENABLE);

    // Configure I/O for L3
    GPIO_InitStruct.GPIO_Pin = L3_R_PIN | L3_G_PIN;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(L3_GPIO, &GPIO_InitStruct);

    // Configure I/O for BUTTON
    GPIO_InitStruct.GPIO_Pin = BUTTON_GPIO_PIN;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_Init(BUTTON_GPIO, &GPIO_InitStruct);
```

Interrupt

```
    //Timer 2 set up
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    TIM_TimeBaseInitTypeDef timerInitStructure;
    timerInitStructure.TIM_Prescaler = 0;
    timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    timerInitStructure.TIM_Period = 5;
    timerInitStructure.TIM_ClockDivision = 0;
    timerInitStructure.TIM_RepetitionCounter = 0;
    TIM_TimeBaseInit(TIM2, &timerInitStructure);
    TIM_Cmd(TIM2, ENABLE);

    TIM_TIxExternalClockConfig(TIM2, TIM_TIxExternalCLK1Source_TI2,
TIM_ICPolarity_Rising, 0);

    //Enable update event for Timer2
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
    NVIC_EnableIRQ(TIM2_IRQn);

    while(1) {}
}
```

Interrupt

```
void TIM2_IRQHandler(void) {
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
      if(state) {
          GPIO_SetBits(L3_GPIO, L3_R_PIN);
          GPIO_ResetBits(L3_GPIO, L3_G_PIN);
          state = 0;
      } else {
          GPIO_ResetBits(L3_GPIO, L3_R_PIN);
          GPIO_SetBits(L3_GPIO, L3_G_PIN);
          state = 1;
      }
      TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}
```

Interrupt

- The standard way: NVIC initialization

```
typedef struct
{
   uint8_t NVIC_IRQChannel;
   uint8_t NVIC_IRQChannelPreemptionPriority;
   uint8_t NVIC_IRQChannelSubPriority;
   FunctionalState NVIC_IRQChannelCmd;
} NVIC_InitTypeDef;
```

Interrupt

- NVIC_IRQChannel = specify the IRQ channel to be enabled or disabled.
  - Example: TIM2_IRQn
- NVIC_IRQChannelPreemptionPriority = specify the pre-emption priority for the IRQ channel specified in NVIC_IRQChannel.
- NVIC_IRQChannelSubPriority = specify the subpriority for the IRQ channel specified in NVIC_IRQChannel.
- NVIC_IRQChannelCmd = specify whether the IRQ channel will be enabled or disabled.
  - Value: ENABLE, DISABLE

Interrupt

- Priority Group Table
  - NVIC_IRQChannelPreemptionPriority: interrupts with different priorities can preempt each other.
  - NVIC_IRQChannelSubPriority: affect the choice of interrupt taken when two or more are pending.

The table below gives the allowed values of the pre-emption priority and subpriority according to the Priority Grouping configuration performed by NVIC_PriorityGroupConfig function

NVIC_PriorityGroup	NVIC_IRQChannelPreemptionPriority	NVIC_IRQChannelSubPriority	Description
NVIC_PriorityGroup_0	0	0-15	0 bits for pre-emption priority 4 bits for subpriority
NVIC_PriorityGroup_1	0-1	0-7	1 bits for pre-emption priority 3 bits for subpriority
NVIC_PriorityGroup_2	0-3	0-3	2 bits for pre-emption priority 2 bits for subpriority
NVIC_PriorityGroup_3	0-7	0-1	3 bits for pre-emption priority 1 bits for subpriority
NVIC_PriorityGroup_4	0-15	0	4 bits for pre-emption priority 0 bits for subpriority

Interrupt

– Example

```
#include <misc.h>

/*
 * NVIC_PriorityGroup_0  0 bits priority, 4 bits subgroup
 * NVIC_PriorityGroup_1  1 bits priority, 3 bits subgroup
 * NVIC_PriorityGroup_2  2 bits priority, 2 bits subgroup
 * NVIC_PriorityGroup_3  3 bits priority, 1 bits subgroup
 * NVIC_PriorityGroup_4  4 bits priority, 0 bits subgroup
*/

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);
```

```
NVIC_InitTypeDef NVIC_InitStructure;

// No  StructInit call in API

NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Interrupt

- External hardware interrupt

Event	Source	Vector
EXT0	PA0-PG0	EXT0_IRQHandler
EXT1	PA1-PG1	EXT1_IRQHandler
EXT2	PA2-PG2	EXT2_IRQHandler
EXT3	PA3-PG3	EXT3_IRQHandler
EXT4	PA4-PG4	EXT4_IRQHandler
EXT5	PA5-PG5	EXT9_5_IRQHandler
...	...	...
EXT15	PA15-PG15	EXT15_10_IRQHandler
EXT16	PVD	PVD_IRQHandler
EXT17	RTC Alarm	RTC_WKUP
EXT18	USB Wakeup	not on STM32 F100
EXT19	Ethernet Wakeup	not on STM32 F100

Interrupt

- Note that only one of PAx – PGx, where x is one of 1 to 15, can be configured for as an EXTI source at any moment.
- Multiple EXTI sources share a handler, pending interrupts can be determined from reading the pending register EXTI_PR.
- Any EXTI source can be triggered through software by setting the appropriate bit in the “software interrupt even register” EXTI_SWIER.

Interrupt

- The standard way: EXTI initialization

```
typedef struct
{
   uint32_t EXTI_Line;
   EXTIMode_TypeDef EXTI_Mode;
   EXTITrigger_TypeDef EXTI_Trigger;
   FunctionalState EXTI_LineCmd;
} EXTI_InitTypeDef;
```

Interrupt

- EXTI_Line = specify the EXTI lines to be enabled or disabled.
  - Value: EXTI_Line0 to EXTI_Line19
  - Value: IS_EXTI_LINE(LINE) = check the range is correct or not.
  - Value: IS_GET_EXTI_LINE(LINE) = check it is one of 19 parameters or not.
- EXTI_Mode = specify the mode for the EXTI lines.
  - Value: EXTI_Mode_Interrupt, EXTI_Mode_Event
  - Usually EXTI_Mode_Interrupt is used.
  - EXTI_Mode_Event: for wakeup the core without interrupt generation

Interrupt

- EXTI_Trigger = specify the trigger signal active edge for the EXTI lines.
  - Value: EXTI_Trigger_Rising, EXTI_Trigger_Falling, EXTI_Trigger_Rising_Falling (both rising and falling)
- EXTI_LineCmd = specify the new state of the selected EXTI lines.
  - Value: ENABLE, DISABLE

Interrupt

- Example 6: External hardware interrupt
  - Press the on-board blue button in PC13 once (external hardware interrupt) to toggle the red LED in PB8 and the green LED in PB9 by using external hardware interrupt.

```
#include "stm32f10x.h"                    // Device header
#include "stdbool.h"

//PC13
#define BUTTON_RCC_GPIO           RCC_APB2Periph_GPIOC
#define BUTTON_GPIO               GPIOC
#define BUTTON_GPIO_PIN           GPIO_Pin_13
#define BUTTON_EXTI_LINE          EXTI_Line13
#define BUTTON_GPIO_PORTSOURCE    GPIO_PortSourceGPIOC
#define BUTTON_GPIO_PINSOURCE     GPIO_PinSource13
```

Interrupt

```
#define L3_RCC_GPIO  RCC_APB2Periph_GPIOB
#define L3_GPIO      GPIOB
#define L3_R_PIN     GPIO_Pin_8
#define L3_G_PIN     GPIO_Pin_9

bool state = false;
bool state_changed = false;

int main(void) {

   GPIO_InitTypeDef GPIO_InitStruct;

   // GPIO clock for I/O
   RCC_APB2PeriphClockCmd(BUTTON_RCC_GPIO, ENABLE); // GPIOC
   RCC_APB2PeriphClockCmd(L3_RCC_GPIO, ENABLE); //GPIOB
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE); //AFIO
```

Interrupt

```
// Configure I/O for L3
GPIO_InitStruct.GPIO_Pin = L3_R_PIN | L3_G_PIN;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(L3_GPIO, &GPIO_InitStruct);

// Configure I/O for EXTI13
GPIO_InitStruct.GPIO_Pin = BUTTON_GPIO_PIN;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IPU;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(BUTTON_GPIO, &GPIO_InitStruct);

// EXTI Configuration, GPIOC, Pin 13
GPIO_EXTILineConfig(BUTTON_GPIO_PORTSOURCE, BUTTON_GPIO_PINSOURCE);
EXTI_InitTypeDef EXTI_InitStruct;
EXTI_InitStruct.EXTI_Line = BUTTON_EXTI_LINE;
EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_InitStruct.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStruct);
```

Interrupt

```
    // Enable Interrupt
    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStruct.NVIC_IRQChannel = EXTI15_10_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x02;
    NVIC_Init(&NVIC_InitStruct);

    while(1) {
      if(state_changed) {
          if(state) {
                  GPIO_SetBits(L3_GPIO, L3_R_PIN);
                  GPIO_ResetBits(L3_GPIO, L3_G_PIN);

          } else {
                  GPIO_ResetBits(L3_GPIO, L3_R_PIN);
                  GPIO_SetBits(L3_GPIO, L3_G_PIN);
          }
          state_changed = false;
      }
    }
}
```

Interrupt

```
void EXTI15_10_IRQHandler(void) {
    if (EXTI_GetITStatus(EXTI_Line13) != RESET) {
     state = !state;
     state_changed = true;
     EXTI_ClearITPendingBit(EXTI_Line13);
    }
}
```

Interrupt

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
```

- Enable the clock for Alternative function I/O
  - Originally all pins are designed for GPIO.
  - AFIO is for other peripherals like timers, interrupts, serial port communication …

```
void GPIO_EXTILineConfig(uint8_t GPIO_PortSource,
uint8_t GPIO_PinSource);
```

- Select the GPIO pin used as EXTI line.
  - GPIO_PortSource: GPIO_PortSourceGPIOx, x = A to G
  - GPIO_PinSource: GPIO_PinSourcex, x = 0 to 15

Interrupt

`void EXTI15_10_IRQHandler(void)`

- IRQ handler for EXTI 10 to 15
  - Originally all pins are designed for GPIO.
  - AFIO is for other peripherals like timers, interrupts, serial port communication …

`ITStatus EXTI_GetITStatus(unit32_t EXTI_Line)`

- Check whether the specified EXTI line is asserted or not.
  - EXTI_Line: External interrupt line x, x = 0 to 19
  - ITStatus: RESET, SET

Interrupt

`void EXTI_ClearITPendingBit(uint32_t EXTI_Line)`

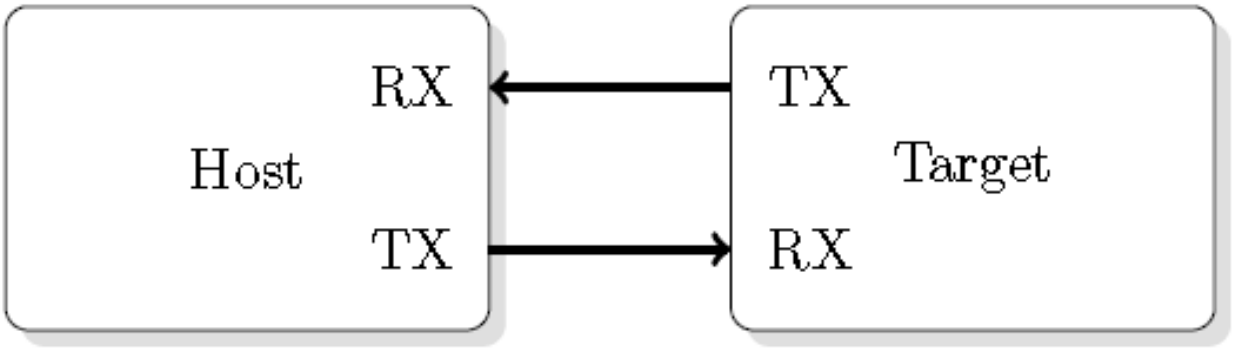
- Clear the EXTI line pending bits
  - EXTI_Line: External interrupt line x, x = 0 to 19

Communication

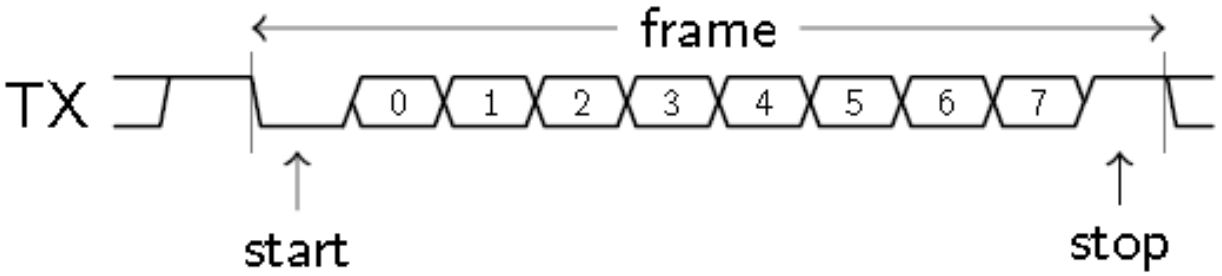
- SPI **2** (up to 18 Mbits/s)
- I2C **2** (support SM Bus 2.0/PM Bus)
- USART **3** (one up to 4.5 Mbit/s, other two up to 2.25 Mbit/s)
- USB **1** (compatible with the USB Full-speed 12 Mbs)
- CAN **1** (up to 1 Mbit/s)

Serial Port Communication

- Basic serial communication technology



- Serial communication protocol



Serial Port Communication

- USART Pins
  - USART2: debugger, default

Function	Pin x=1	Pin x=2	Pin x=3
USARTx_TX	PA9	PA2	PB10
USARTx_RX	PA10	PA3	PB11
USARTx_CK	PA8	PA4	PB12
USARTx_RTS	PA12	PA1	PB14
USARTx_CTS	PA11	PA0	PB13

Serial Port Communication

- USART Pin Configuration

USART pinout	Configuration	GPIO Configuration
USARTx_TX	Full Duplex	Alternate function push-pull
	Half duplex Synchronous mode	Alternate function push-pull
USARTx_RX	Full Duplex	Input floating/Input Pull-up
	Half duplex Synchronous mode	Not used. Can be used as General IO
USARTx_CK	Synchronous mode	Alternate function push-pull
USARTx_RTS	Hardware flow control	Alternate function push-pull
USARTx_CTS	Hardware flow control	Input floating/Input pull-up

Serial Port Communication

- Example 7: Echo characters in serial port (polling)

```
#include "stm32f10x.h"                    // Device header

int main(void) {
   //USART2 TX RX
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);

   // Tx pin
   GPIO_InitTypeDef GPIO_InitStructure;
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_Init(GPIOA, &GPIO_InitStructure);

   // Rx pin
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
   GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Serial Port Communication

```
//USART2 ST-LINK USB
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);

USART_InitTypeDef USART_InitStructure;
USART_InitStructure.USART_BaudRate = 9600;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl =
     USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USART2, &USART_InitStructure);
USART_Cmd(USART2, ENABLE);
```

Serial Port Communication

```
    uint8_t character;

    while(1) {
      while(USART_GetFlagStatus(USART2, USART_FLAG_RXNE) == RESET);
      character = USART_ReceiveData(USART2) & 0xFF;
      //flag clears automatically
      //USART_ClearITPendingBit(USART2, USART_IT_RXNE);
      while(USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET);
      USART_SendData(USART2, character);
    }
}
```

Serial Port Communication

- USART initialization

```
typedef struct
{
   uint32_t USART_BaudRate;
   uint16_t USART_WordLength;
   uint16_t USART_StopBits;
   uint16_t USART_Parity;
   uint16_t USART_Mode;
   uint16_t USART_HardwareFlowControl;
} USART_InitTypeDef;
```

Serial Port Communication

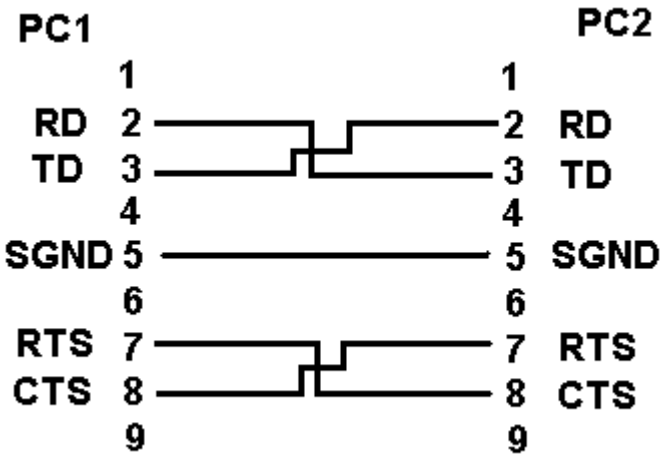
- USART_BaudRate = configure the USART communication baud rate.
  - Unit: bps
- USART_WordLength = specify the number of data bits transmitted or received in a frame.
  - Value: USART_WordLength_8b, USART_WordLength_9b
  - That means 8 bits and 9 bits.
- USART_StopBits = Specifies the number of stop bits transmitted.
  - Value: USART_StopBits_1, USART_StopBits_0_5, USART_StopBits_1_5 and USART_StopBits_2
  - The number of bits is to tell the duration.

Serial Port Communication

- USART_Parity = specify the parity mode.
  - Value: USART_Parity_No, USART_Parity_Even and USART_Parity_Odd
- USART_Mode = specify whether the Receive or Transmit mode is enabled or disabled.
  - Value: USART_Mode_Tx, USART_Mode_Rx
- USART_HardwareFlowControl = specify whether the hardware flow control mode is enabled or disabled.
  - Value: USART_HardwareFlowControl_None, USART_HardwareFlowControl_RTS, USART_HardwareFlowControl_CTS, USART_HardwareFlowControl_RTS_CTS

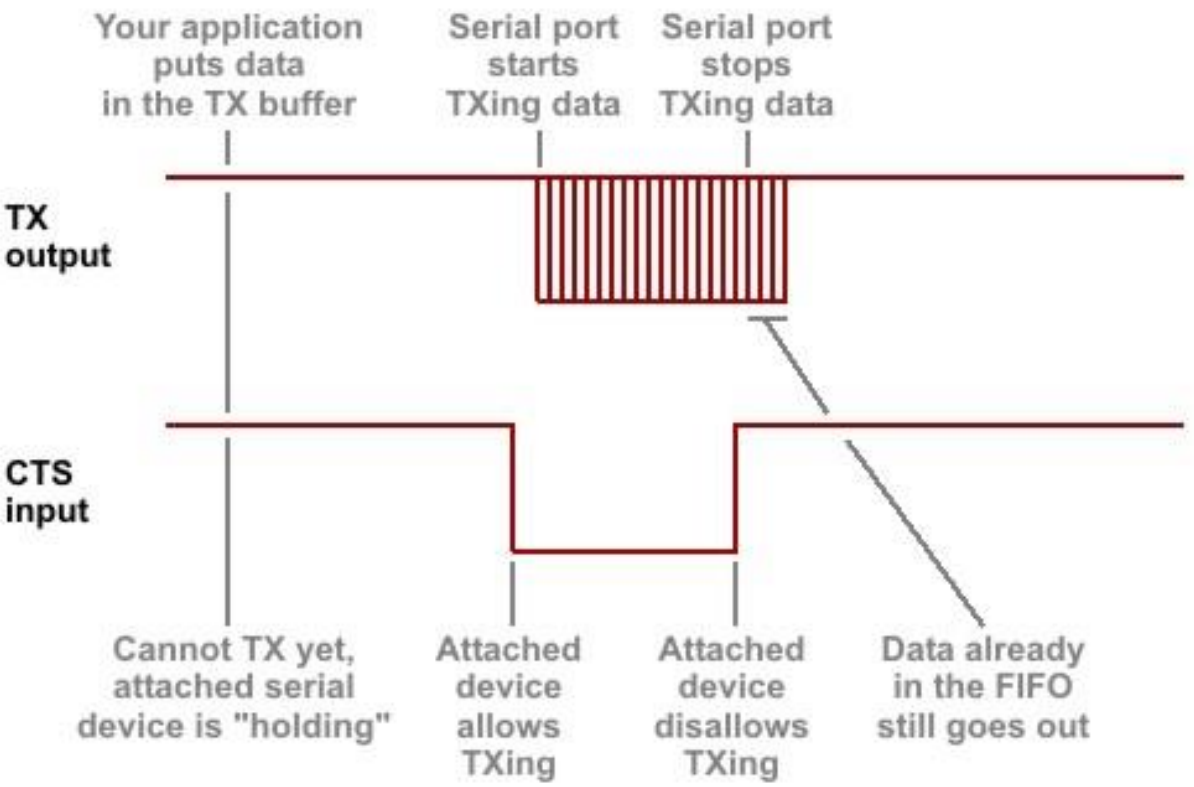
Serial Port Communication

- Hardware flow control
  - RTS/CTS flow control (RTS/CTS handshaking)
  - In common RS-232 there are pairs of control lines which are usually referred to as hardware flow control.
  - RTS (Request To Send) and CTS (Clear To Send)



Serial Port Communication

- Operation



Serial Port Communication

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);
```

- Enable the clock for USART2.

```
USART_Init(USART2, &USART_InitStructure);
```

- Initialize the USARTx peripheral according to the specific parameters in the USART_InitStructure.

```
USART_Cmd(USART2, ENABLE);
```

- Enable or disable the specified USART peripheral.

Serial Port Communication

```
FlagStatus  USART_GetFlagStatus (USART_TypeDef
*USARTx, uint16_t USART_FLAG)
```

- Check whether the specified USART flag is set or not.
  - USART_FLAG_CTS: CTS Change flag (not available for UART4 and UART5)
  - USART_FLAG_LBD: LIN Break detection flag
  - USART_FLAG_TXE: Transmit data register empty flag
  - USART_FLAG_TC: Transmission Complete flag
  - USART_FLAG_RXNE: Receive data register not empty flag
  - USART_FLAG_IDLE: Idle Line detection flag
  - USART_FLAG_ORE: OverRun Error flag
  - USART_FLAG_NE: Noise Error flag
  - USART_FLAG_FE: Framing Error flag
  - USART_FLAG_PE: Parity Error flag

Serial Port Communication

```
void  USART_SendData (USART_TypeDef *USARTx, uint16_t Data)
```

- Transmit single data through the USARTx peripheral.

```
uint16_t USART_ReceiveData (USART_TypeDef *USARTx)
```

- Returns the most recent received data by the USARTx peripheral.

```
character = USART_ReceiveData(USART2) & 0xFF;
```

- The “and” operation is to extract the low byte of the returned value.

Serial Port Communication

- Example 8: Print a message and echo characters in serial port (interrupt)

```
#include "stm32f10x.h"                    // Device header
#include "string.h"

char msg[] = "We are ready!";
//char bye[] = "Bye!";
uint8_t msg_i=0;
unsigned char character;

int main(void) {

   msg_i=0;

   //USART2 TX RX
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);
```

Serial Port Communication

```
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA, &GPIO_InitStructure);

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA, &GPIO_InitStructure);

//USART2 ST-LINK USB
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);

USART_InitTypeDef USART_InitStructure;
USART_InitStructure.USART_BaudRate = 9600;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl =
     USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
```

Serial Port Communication

```
USART_Init(USART2, &USART_InitStructure);
USART_Cmd(USART2, ENABLE);

NVIC_InitTypeDef NVIC_InitStructure;
// Enable the USART2 TX Interrupt
USART_ITConfig(USART2, USART_IT_TC, ENABLE );
NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
// Enable the USART2 RX Interrupt
USART_ITConfig(USART2, USART_IT_RXNE, ENABLE );
NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Serial Port Communication

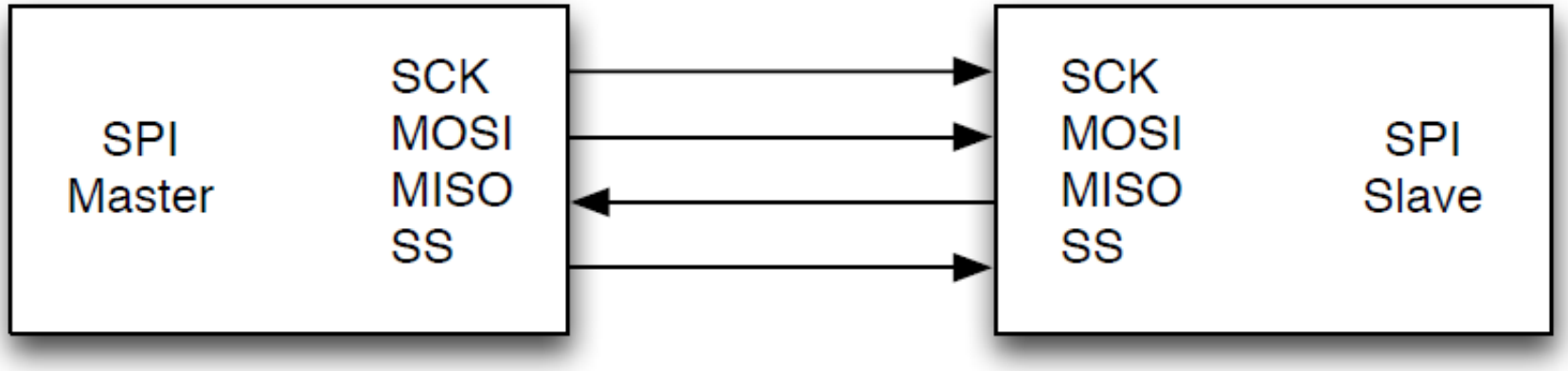
```
    while(1) {
    }
}

void USART2_IRQHandler() {
    if(USART_GetITStatus(USART2, USART_IT_TC) != RESET) {
         if(msg_i < strlen(msg)) {
                  USART_SendData(USART2, msg[msg_i++]);
         }
         //USART_ClearITPendingBit(USART2, USART_IT_TC);

    }
    if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET) {
      character = (unsigned char) USART_ReceiveData(USART2);
      USART_SendData(USART2, character);
    }
}
```

SPI

- Serial Peripheral Interface Bus
  - Synchronous serial communication
  - Full duplex, master-slave architecture
  - The master device originates the frame for reading and writing.
  - Multiple slave devices are supported through selection with individual salve select (SS) lines.

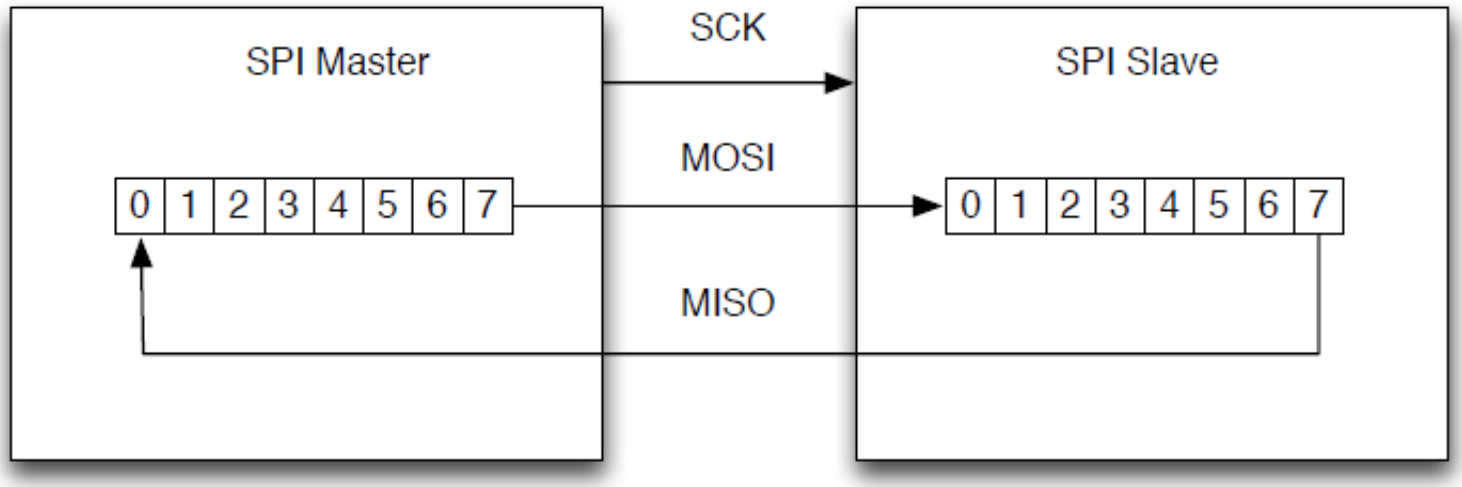


SPI

- Operations
  - SS must be replicated for every slave connected to the bus.
  - All communication is controlled by the master.
  - The master selects the slave it wishes to communicate with by lowering the appropriate SS line.
  - Then it transfer a single word (commonly one byte) serially to the slave over MOSI (Master Out Slave In).
  - At the same time, it accepts a single byte from the slave over the MISO (Master In Slave Out).

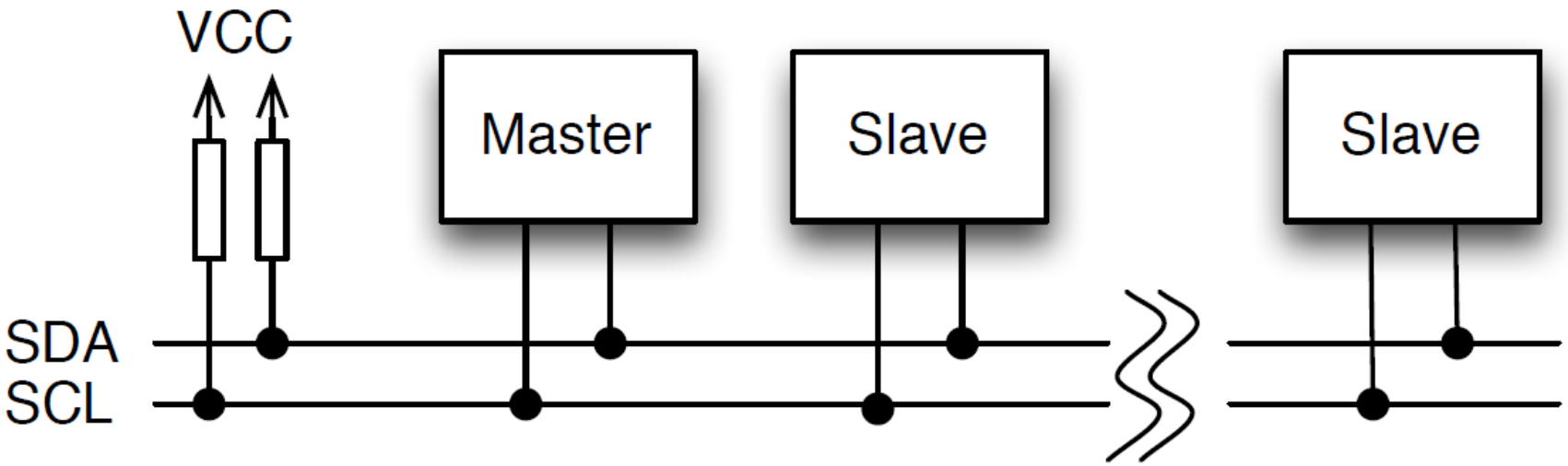
SPI

– This transfer is realized by generating 8 clock pulses on the signal SCK (serial clock).



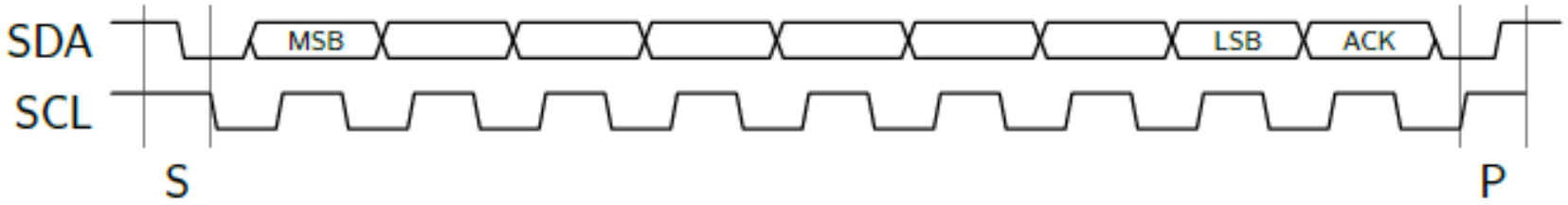
I2C

- Inter-Integrated Circuit Communication Protocol
  - Like SPI, multiple slaves can connect to a single master.
    - Can be more than one master.
  - Like serial port communication, I2C only uses two wires to transmit data between devices.



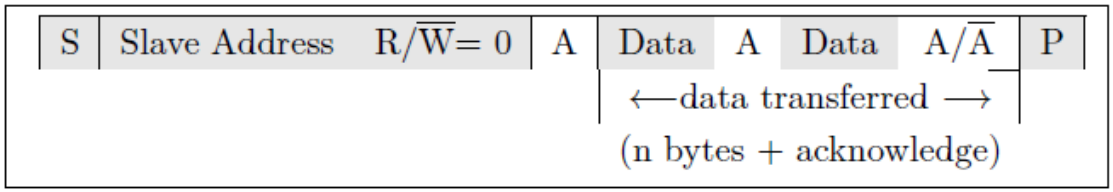
I2C

- Operations
  - SCL: serial clock line
  - SDA: serial data/address
  - A master drives a clock signal on SCL.
  - A slave drives SDA.
  - The transfer bit rate is determined by the master.

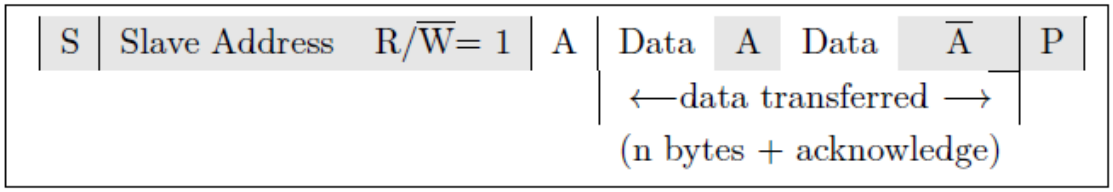


I2C

- Operations



Write Transaction



Read Transaction

From master to slave

From slave to master

A = Acknowledge

$\overline{A}$ = Not Acknowledge

S = Start condition

P = Stop Condition

Reference Readings

- http://www.longlandclan.yi.org/~stuartl/stm32f10x_stdperiph_lib_um
- Chapter 5, 6, 9 and 11 – *Discovering the STM32 Microcontroller*, Geoffrey Brown, 2012
- RM0008 Reference Manual (STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 320bit MCUs)
- Datasheet – STM32F103x8, STM32F103xB

End