EIE3105: ARM Programming 2 – PWM and Input Capture

Dr. Lawrence Cheung

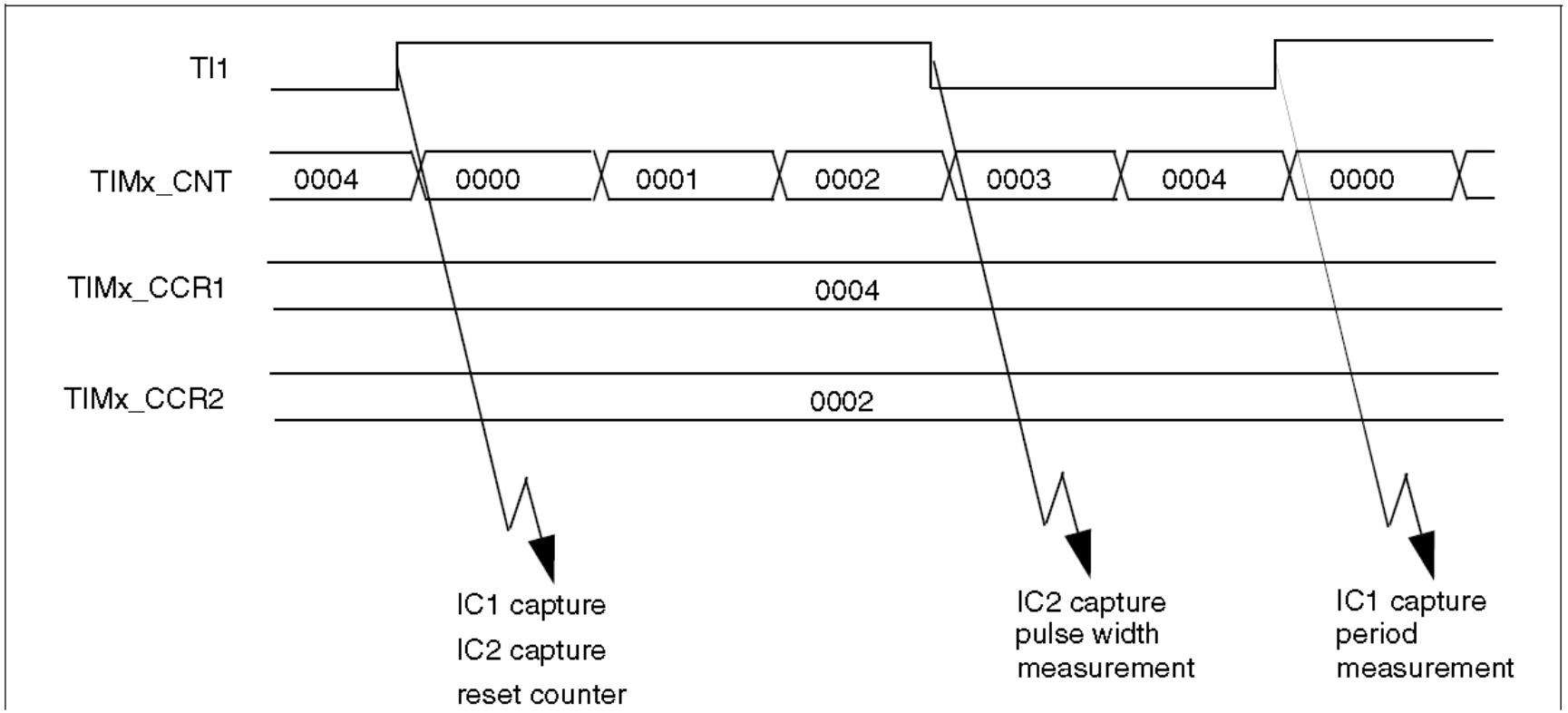
Semester 1, 2020/21

Topics

- Input Capture
- Output Compare
- PWM generation
- One-pulse mode output
- Programming: PWM and Input Capture

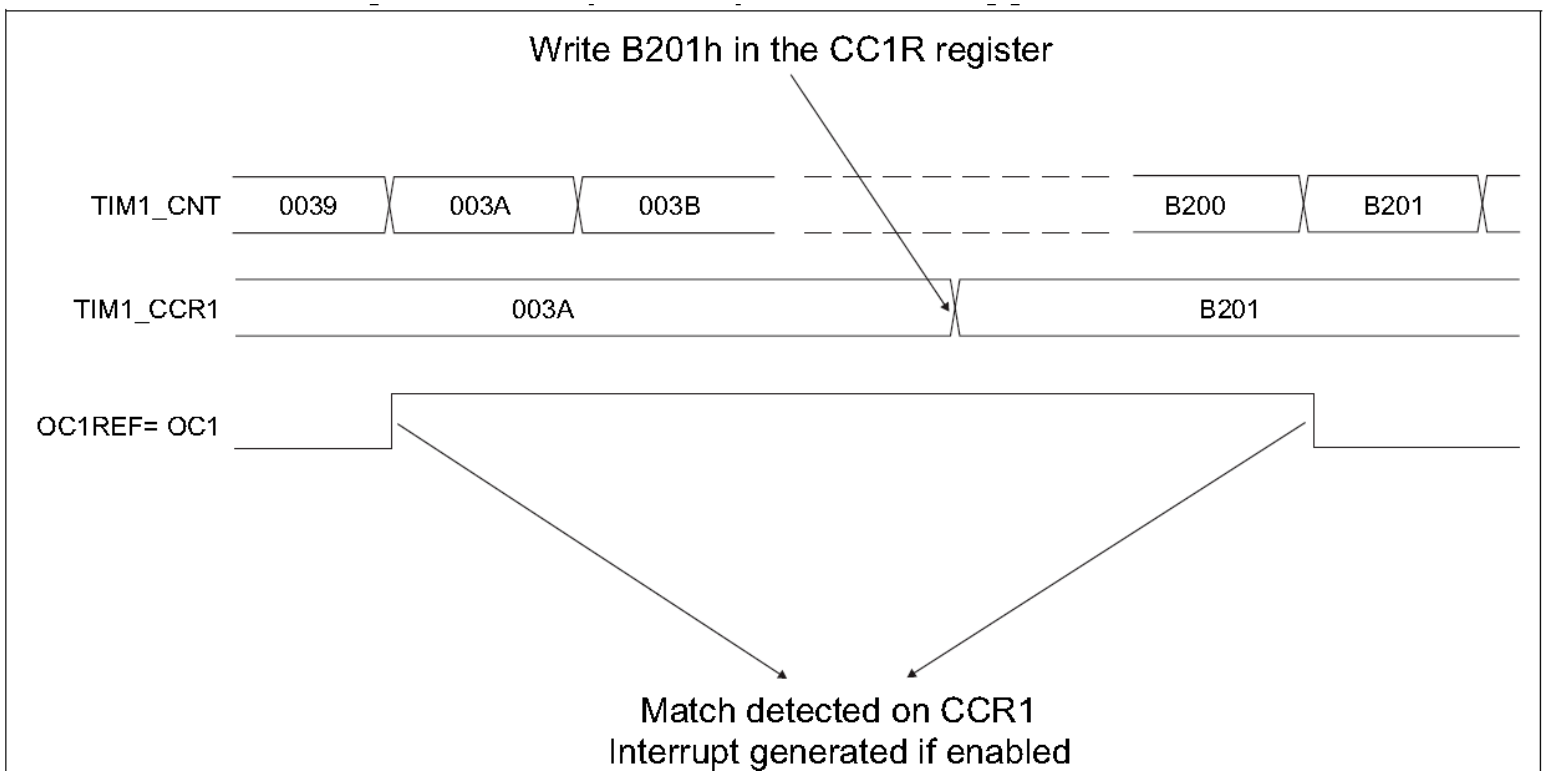
Input Capture

- A capture event causes the counter value to be transferred into the capture register and triggers an interrupt or a DMA request.



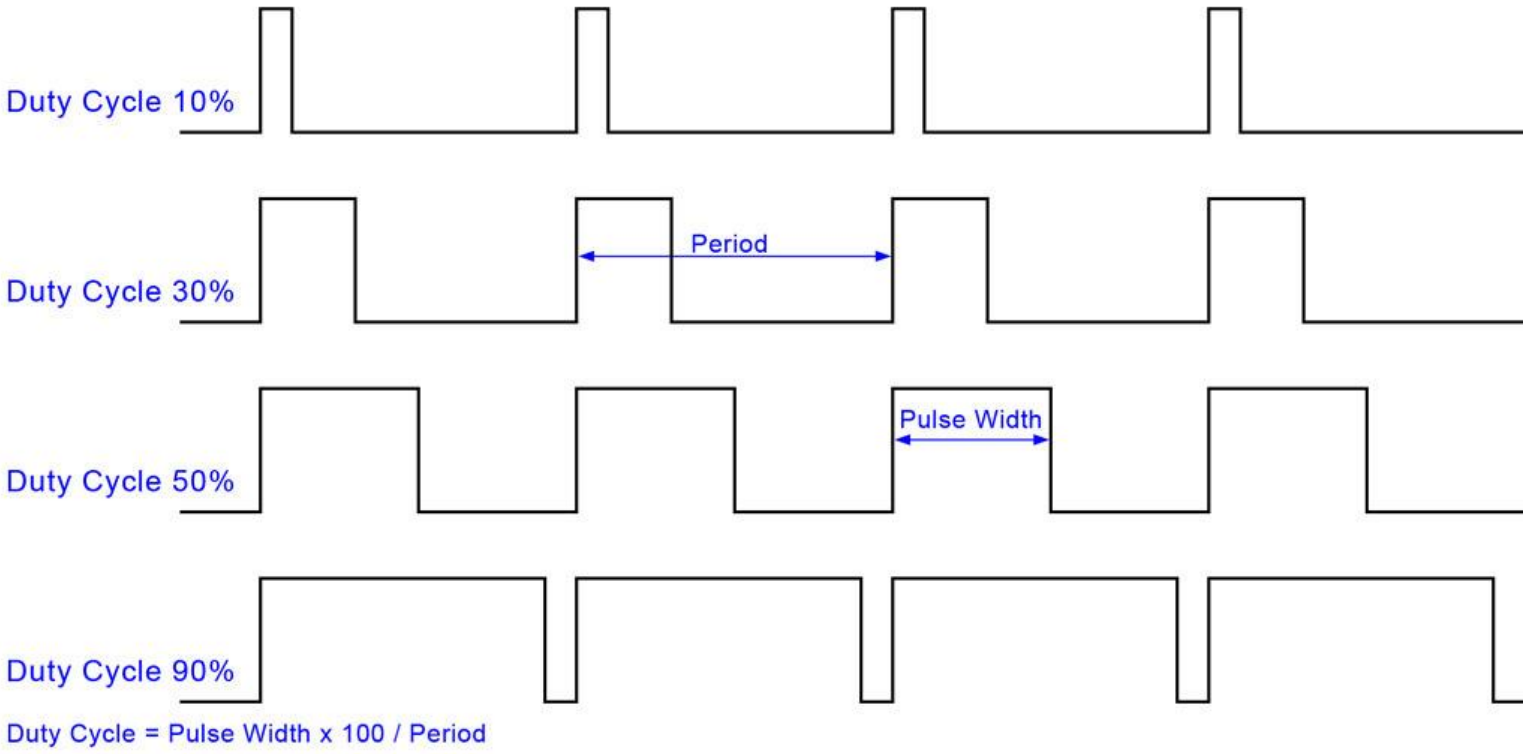
Output Compare

- This function is used to control an output waveform or indicating when a period of time has elapsed.



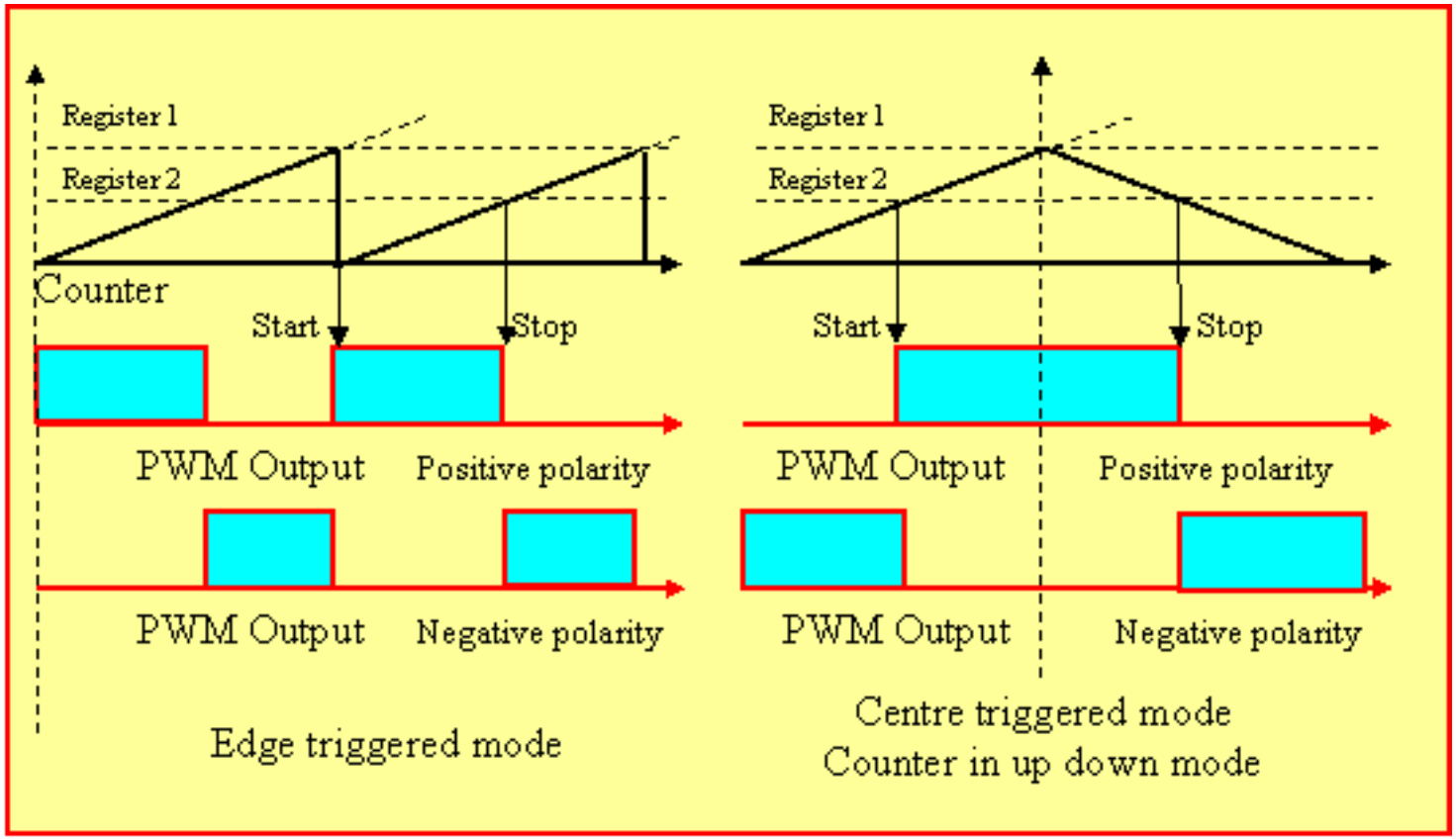
PWM Generation

- Generate waveforms with specified duty cycles and frequencies.



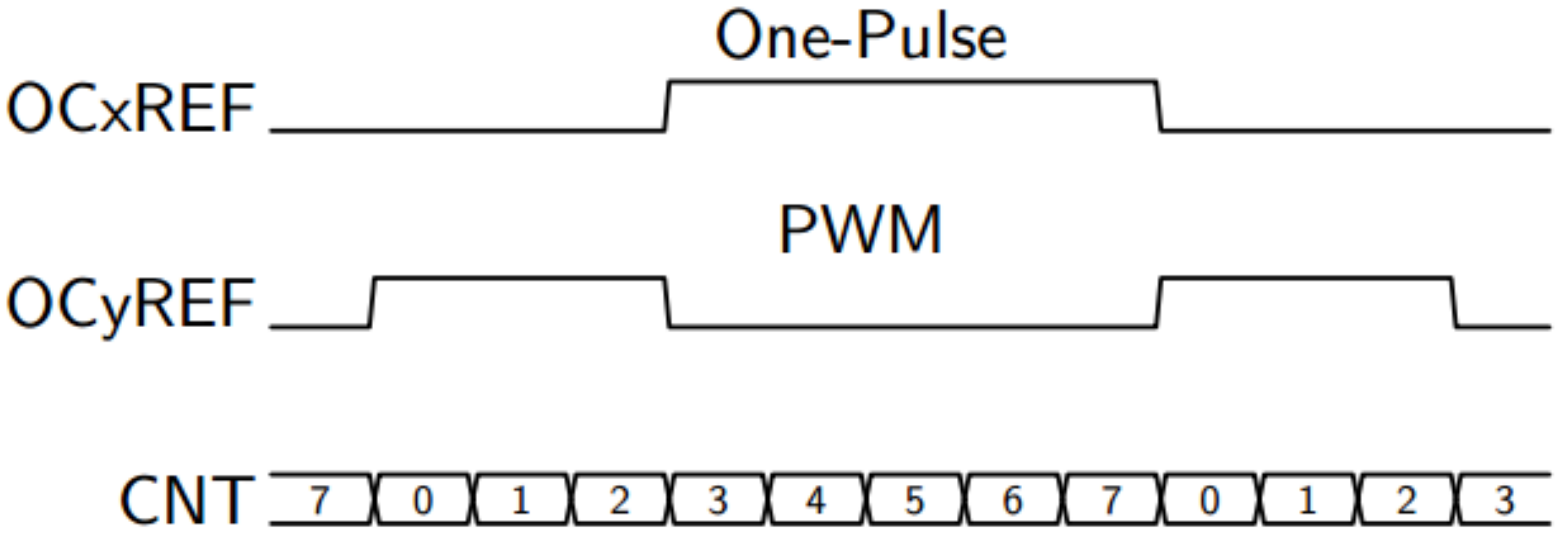
PWM Generation

- Two popular modes: Edge-triggered mode and center-triggered mode



One-Pulse Mode Output

- Allow the counter to be started in response to a stimulus and to generate a pulse.



PWM and Input Capture

- Example 1:
  - Generate a PWM signal at 20Hz with 24.7% of duty cycle from PA6.
  - Capture the signal from PB6 (PA6 and PB6 are connected).
  - Show the pulse width (the ON period) through the serial port.
- Program Files
  - PinMap.h: initialize pins and functions.
  - init.c: initialize IC1, PWM and USART2.
  - main.c: main program

PWM and Input Capture

- PinMap.h

```
// Pin Usage
// Function      **  Pin Name ** Board Pin Out
// TIM4 CH1 IC1  **  PB6      ** D10
// TIM3 CH1 PWM  **  PA6      ** D12

// TIM4 CH1 IC1  **  PB6      ** D10
#define TIM4_CH1_IT1_RCC_GPIO  RCC_APB2Periph_GPIOB
#define TIM4_CH1_IT1_GPIO      GPIOB
#define TIM4_CH1_IT1_PIN       GPIO_Pin_6

// TIM3 CH1 PWM  **  PA6      ** D12
#define TIM3_CH1_PWM_RCC_GPIO   RCC_APB2Periph_GPIOA
#define TIM3_CH1_PWM_GPIO       GPIOA
#define TIM3_CH1_PWM_PIN        GPIO_Pin_6
```

PWM and Input Capture

```
//Function prototypes
void TIM4_CH1_IC1_init(void);
void TIM3_CH1_PWM_init(void);
void USART2_init(void);
void USARTSend(char *pucBuffer, unsigned long ulCount);
```

PWM and Input Capture

- init.c

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

void TIM4_CH1_IC1_init(void) {
    GPIO_InitTypeDef GPIO_InitStruct;
    // Configure I/O for IC1
    GPIO_InitStruct.GPIO_Pin = TIM4_CH1_IT1_PIN;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_Init(TIM4_CH1_IT1_GPIO, &GPIO_InitStruct);

    //Tim4 set up
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
```

PWM and Input Capture

```
TIM_TimeBaseInitTypeDef timerInitStructure;
timerInitStructure.TIM_Prescaler = 720-1;  //1/(72Mhz/720)=0.01ms
timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
timerInitStructure.TIM_Period = 50000-1;  //0.01ms*50000 = 500ms
timerInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
timerInitStructure.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM4, &timerInitStructure);
TIM_Cmd(TIM4, ENABLE);

//Enable Tim4 Ch1 Input Capture
TIM_ICInitTypeDef InputCaptureInitStructure;
InputCaptureInitStructure.TIM_Channel = TIM_Channel_1;  //Seclect IC1
InputCaptureInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
//Capture rising
InputCaptureInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
//Map to TI1
InputCaptureInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
//Configure input frequency
InputCaptureInitStructure.TIM_ICFilter = 0; //no filter
TIM_ICInit(TIM4, &InputCaptureInitStructure);
```

PWM and Input Capture

```
    //Enable Input Capture Interrupt
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;    //TIM4 interrupt
    //Preemptive priority level 2
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
    //From the priority level 0
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    //The IRQ channel is enabled
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
    //Allow updates to interrupt, allows the CC1IE to capture interrupt
    TIM_ITConfig(TIM4,TIM_IT_Update|TIM_IT_CC1,ENABLE);

}

void TIM3_CH1_PWM_init(void) {
    RCC_APB2PeriphClockCmd(TIM3_CH1_PWM_RCC_GPIO, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
```

PWM and Input Capture

```
GPIO_InitTypeDef GPIO_InitStructure;
// Configure I/O for Tim3 Ch1 PWM pin
GPIO_InitStructure.GPIO_Pin = TIM3_CH1_PWM_PIN;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(TIM3_CH1_PWM_GPIO, &GPIO_InitStructure);

//Tim3 set up
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

TIM_TimeBaseInitTypeDef timerInitStructure;
timerInitStructure.TIM_Prescaler = 720-1;  //1/(72Mhz/720)=0.01ms
timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
timerInitStructure.TIM_Period = 5000-1;
timerInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
timerInitStructure.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM3, &timerInitStructure);
TIM_Cmd(TIM3, ENABLE);
```

PWM and Input Capture

```
    //Enable Tim3 Ch1 PWM
    TIM_OCInitTypeDef outputChannelInit;
    outputChannelInit.TIM_OCMode = TIM_OCMode_PWM1;
    outputChannelInit.TIM_Pulse = 1235-1;
    outputChannelInit.TIM_OutputState = TIM_OutputState_Enable;
    outputChannelInit.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OC1Init(TIM3, &outputChannelInit);
    TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable);
}

void USART2_init(void) {
    //USART2 TX RX
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO,
ENABLE);

    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);
```

PWM and Input Capture

```
    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;

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

PWM and Input Capture

```
void USARTSend(char *pucBuffer, unsigned long ulCount)
{
    //
    // Loop while there are more characters to send.
    //
    while(ulCount--)
    {
        USART_SendData(USART2, *pucBuffer++);
        /* Loop until the end of transmission */
        while(USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET)
        {
        }
    }
}
```

PWM and Input Capture

- Input capture initialization

```
typedef struct
{
   uint16_t TIM_Channel;
   uint16_t TIM_ICPolarity;
   uint16_t TIM_ICSelection;
   uint16_t TIM_ICPrescaler;
   uint16_t TIM_ICFilter;
} TIM_ICInitTypeDef;
```

PWM and Input Capture

- TIM_Channel = specify the TIM channel.
  - TIM_Channel_1
  - TIM_Channel_2
  - TIM_Channel_3
  - TIM_Channel_4
  - In this example, Timer_Channel_1 is selected.
- TIM_ICPolarity = specify the active edge of the input signal.
  - TIM_ICPolarity_Rising
  - TIM_ICPolarity_Falling
  - TIM_ICPolarity_BothEdge
  - In this example, a rising edge is triggered.

PWM and Input Capture

- TIM_ICSelection = specify the input.
  - TIM_ICSelection_DirectTI: TIM Input 1, 2, 3, or 4 is selected to be connected to IC1, IC2, IC3 or IC4 respectively.
    - TIM4_CH1/PB6, TIM4_CH2/PB7, TIM4_CH3/PB8, TIM4_CH4/PB9
  - TIM_ICSelection_IndirectTI: TIM Input 1, 2, 3, or 4 is selected to be connected to IC2, IC1, IC4 or IC3 respectively.
  - TIM_ICSelection_TRC: TIM Input 1, 2, 3, or 4 is selected to be connected to TRC.
  - In this example, TIM_ICSelection_DirectTI is selected.

PWM and Input Capture

- TIM1 alternate function remapping

Channel Number	Channel Name	Pin
Channel 1	TIM1_CH1	PA8
Channel 1	TIM1_CH1N	PB13
Channel 2	TIM1_CH2	PA9
Channel 2	TIM1_CH2N	PB14
Channel 3	TIM1_CH3	PA10
Channel 3	TIM1_CH3N	PB15
Channel 4	TIM1_CH4	PA11
	TIM1_BKIN	PB12
	TIM1_ETR	PA12

PWM and Input Capture

- TIM1 alternate function remapping

Channel Number	Channel Name	Pin
Channel 1	TIM2_CH1_ETR	PA0
Channel 2	TIM2_CH2	PA1
Channel 3	TIM2_CH3	PA2
Channel 4	TIM2_CH4	PA3
Channel 1	TIM3_CH1	PA6
Channel 2	TIM3_CH2	PA7
Channel 3	TIM3_CH3	PB0
Channel 4	TIM3_CH4	PB1

PWM and Input Capture

- TIM1 alternate function remapping

Channel Number	Channel Name	Pin
Channel 1	TIM4_CH1	PB6
Channel 2	TIM4_CH2	PB7
Channel 3	TIM4_CH3	PB8
Channel 4	TIM4_CH4	PB9
	TIM4_ETR	PE0

PWM and Input Capture

- TIM_ICPrescaler = specify the input capture prescaler.
  - TIM_ICPSC_DIV1: Capture performed each time an edge is detected on the capture input.
  - TIM_ICPSC_DIV2: Capture performed once every 2 events.
  - TIM_ICPSC_DIV4: Capture performed once every 4 events.
  - TIM_ICPSC_DIV8: Capture performed once every 8 events.
  - Default = TIM_ICPSC_DIV1

PWM and Input Capture

- TIM_ICFilter = specify the input capture filter.
  - Value: 0x0 to 0xF
  - Default = 0
  - It tells the number of successive waveforms such that their measured pulse widths are the same.

PWM and Input Capture

```
void TIM_ICInit(TIM_TypeDef* TIMx, TIM_ICInitTypeDef*
TIM_ICInitStruct)
```

- Initialize the TIM peripheral according to the specified parameters in the TIM_ICInitStruct.

PWM and Input Capture

- Output channel initialization (PWM)
  - You still need to initialize the timer first.

```
typedef struct {
   uint16_t TIM_OCMode;
   uint16_t TIM_OutputState;
   uint16_t TIM_OutputNState;
   uint16_t TIM_Pulse;
   uint16_t TIM_OCPolarity;
   uint16_t TIM_OCNPolarity;
   uint16_t TIM_OCIdleState;
   uint16_t TIM_OCNIdleState;
} TIM_OCInitTypeDef;
```

PWM and Input Capture

- TIM_OCMode = specify the TIM mode.
  - TIM_OCMode_Timing: The current pin output does not change even if it matches the output comparison
  - TIM_OCMode_Active: The timer output becomes active when the counter value matches
  - TIM_OCMode_Inactive: The timer output active when the counter value does not match (it becomes inactive when it matches)
  - TIM_OCMode_Toggle: When the counter value matches, the timer output is inverted

PWM and Input Capture

- TIM_OCMode_PWM1: Active when TIMx_CNT < TIMx_CCRy; otherwise inactive
- TIM_OCMode_PWM2: Inactive when TIMx_CNT < TIMx_CCRy; otherwise active
  - TIMx_CNT = TIMx's counter register
  - TIMx_CCRy = TIMx's capture compare register

PWM and Input Capture

- TIM_OutputState = specify the TIM Output Compare State.
  - TIM_OutputState_Enable: Enable timer output.
  - TIM_OutputState_Disable: Disable timer output.

- TIM_OutputNState = specify the TIM complementary Output Compare State.
  - TIM_OutputNState_Enable: Enable timer complementary output.
  - TIM_OutputNState_Disable: Disable timer complementary output.

PWM and Input Capture

- TIM_Pulse = specify the pulse value to be loaded into the Capture Compare Register (TIMx_CCRy). It is the ON period.
  - Value: 0 to 0xFFFF

- TIM_OCPolarity = specify the output polarity.
  - TIM_OCPolarity_High: Set the output polarity at active level to high level.
  - TIM_OCPolarity_Low: Set the output polarity at inactive level to high level.

PWM and Input Capture

- TIM_OCNPolarity = specify the complementary output polarity.
  - TIM_OCNPolarity_High: Set the complementary output polarity at active level to high level.
  - TIM_OCNPolarity_Low: Set the complementary output polarity at inactive level to low level.

PWM and Input Capture

- TIM_OCIdleState = specify the TIM Output Compare pin state during Idle state.
  - TIM_OCIdleState_Enable: Enable the output.
  - TIM_OCIdleState_Disable: Disable the output.

- TIM_OCNIdleState = specify the TIM complementary Output Compare State.
  - TIM_OCNIdleState_Enable: Enable the complementary output.
  - TIM_OCNIdleState_Disable: Disable the complementary output.

PWM and Input Capture

```
void TIM_OC1Init(TIM_TypeDef* TIMx,
TIM_OCInitTypeDef* TIM_OCInitStruct)
```

- Initialize the TIMx Channel1 according to the specified parameters in the TIM_OCInitStruct.

```
void TIM_OC1PreloadConfig(TIM_TypeDef* TIMx, uint16_t
TIM_OCPreload)
```

- Enable or disable the TIMx peripheral Preload register on CCR1.
  - TIM_OCPreload_Enable
  - TIM_OCPreload_Disable

PWM and Input Capture

- Auto-reload register is preloaded.
- Writing to or reading from the auto-reload register accesses the preload register.

PWM and Input Capture

- main.c

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

char buffer[50] = {'\0'};
bool pulseHigh = false;
u32 pulseWidth = 0;

int main(void) {

    TIM4_CH1_IC1_init();
    TIM3_CH1_PWM_init();
    USART2_init();
    while(1) {
    }
}
```

PWM and Input Capture

```
void TIM4_IRQHandler(void) {
    if(TIM_GetITStatus(TIM4,TIM_IT_CC1) !=RESET) {
      if(!pulseHigh) {
          pulseHigh=true;         //pulse starts
          TIM_SetCounter(TIM4,0);
          //change to detect falling
          TIM_OC1PolarityConfig(TIM4,TIM_ICPolarity_Falling);
      } else {
          pulseWidth += TIM_GetCounter(TIM4);
          //change to detect raising
          TIM_OC1PolarityConfig(TIM4,TIM_ICPolarity_Rising);
          sprintf(buffer, "%d\r\n", pulseWidth);
          USARTSend(buffer, sizeof(buffer));
          pulseHigh= false;
          pulseWidth=0;
      }
    }
    //Clear interrupt flag
    TIM_ClearITPendingBit(TIM4,TIM_IT_Update|TIM_IT_CC1);
}
```

PWM and Input Capture

```
void TIM_SetCounter(TIM_TypeDef* TIMx, uin16_t
Counter)
```

- Set the TIMx Counter value.

```
uint16_t TIM_GetCounter(TIM_TypeDef* TIMx)
```

- Get the TIMx Counter value.

```
int sprintf(char* str, const char* format)
```

- Compose a string with the same text that would be printed if the format was used on printf.
  - str: pointer to a buffer
  - format: C string that contains a format string
  - Return value: the total number of characters written

PWM and Input Capture

- Example:

```
#include <stdio.h>

int main ()
{
  char buffer [50];
  int n, a = 5, b = 3;
  n = sprintf (buffer, "%d plus %d is %d", a, b, a + b);
  printf ("[%s] is a string %d chars long\n", buffer, n);
  return 0;
}
```

- Output:

```
[5 plus 3 is 8] is a string 13 chars long
```

Reference Readings

- http://www.longlandclan.yi.org/~stuartl/stm32f10x_stdperiph_lib_um
- Chapter 10, 12 and 14 – *Discovering the STM32 Microcontroller*, Geoffrey Brown, 2012
- RM0008 Reference Manual (STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 320bit MCUs)
- AN3116 Application note (STM32 ADC modes and their applications)
- Datasheet – STM32F103x8, STM32F103xB

End