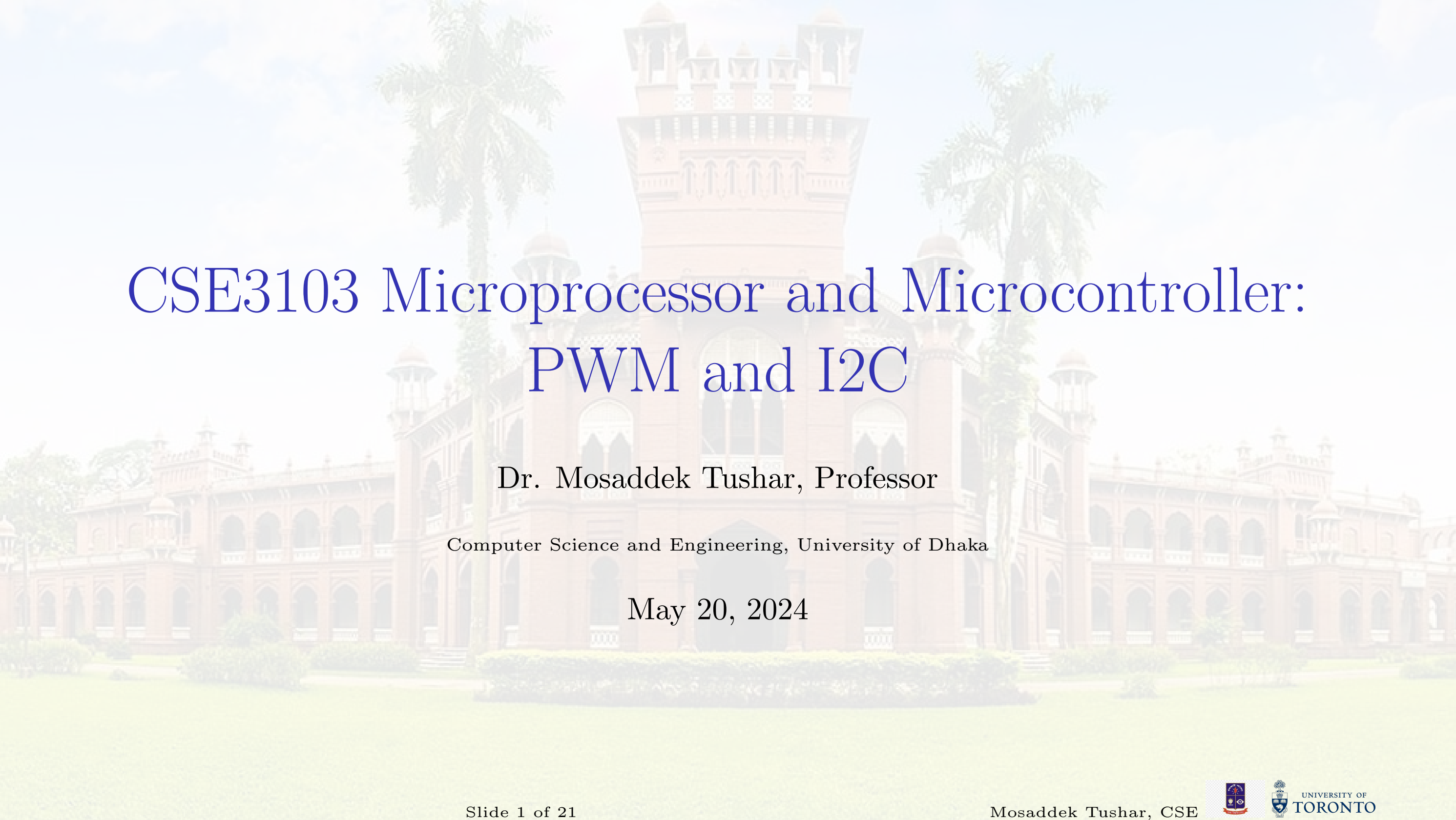


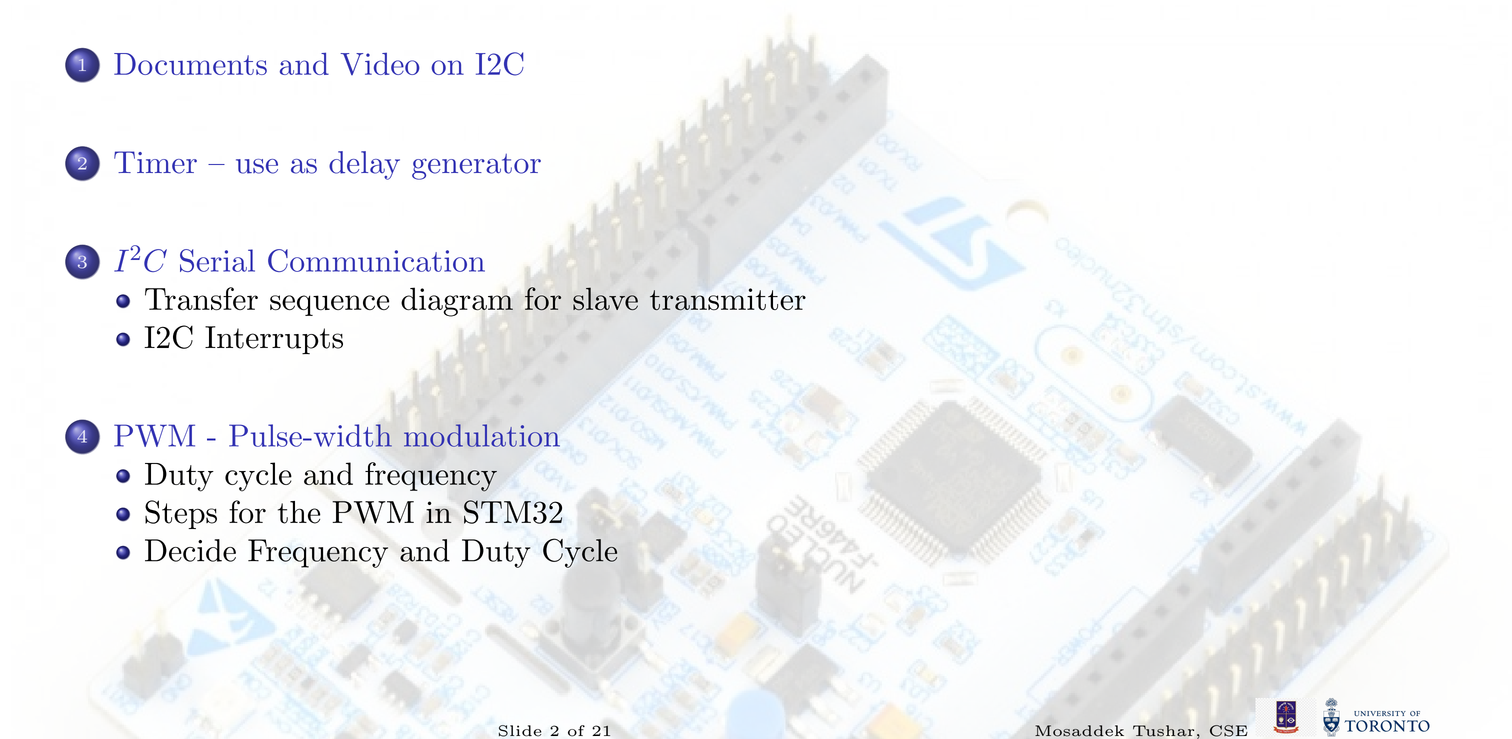
CSE3103 Microprocessor and Microcontroller: PWM and I2C

Dr. Mosaddek Tushar, Professor

Computer Science and Engineering, University of Dhaka

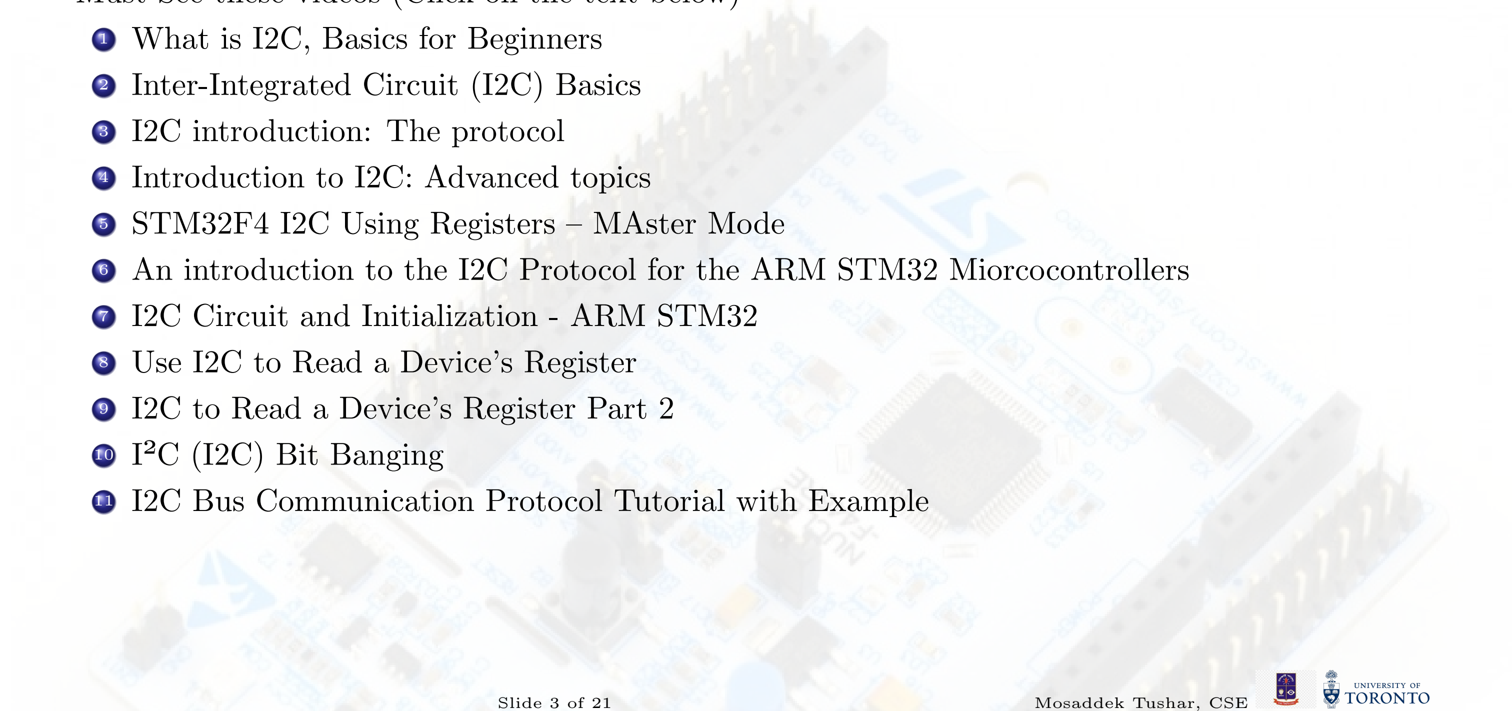
May 20, 2024

Contents



1. Documents and Video on I2C
2. Timer – use as delay generator
3. I^2C Serial Communication
   - Transfer sequence diagram for slave transmitter
   - I2C Interrupts
4. PWM - Pulse-width modulation
   - Duty cycle and frequency
   - Steps for the PWM in STM32
   - Decide Frequency and Duty Cycle

Documents and Video on I2C



Must See these videos (Click on the text below)

1. What is I2C, Basics for Beginners
2. Inter-Integrated Circuit (I2C) Basics
3. I2C introduction: The protocol
4. Introduction to I2C: Advanced topics
5. STM32F4 I2C Using Registers – MAster Mode
6. An introduction to the I2C Protocol for the ARM STM32 Miorcocontrollers
7. I2C Circuit and Initialization - ARM STM32
8. Use I2C to Read a Device's Register
9. I2C to Read a Device's Register Part 2
10. I^{2}C (I2C) Bit Banging
11. I2C Bus Communication Protocol Tutorial with Example

Timer as Delay Function: Example

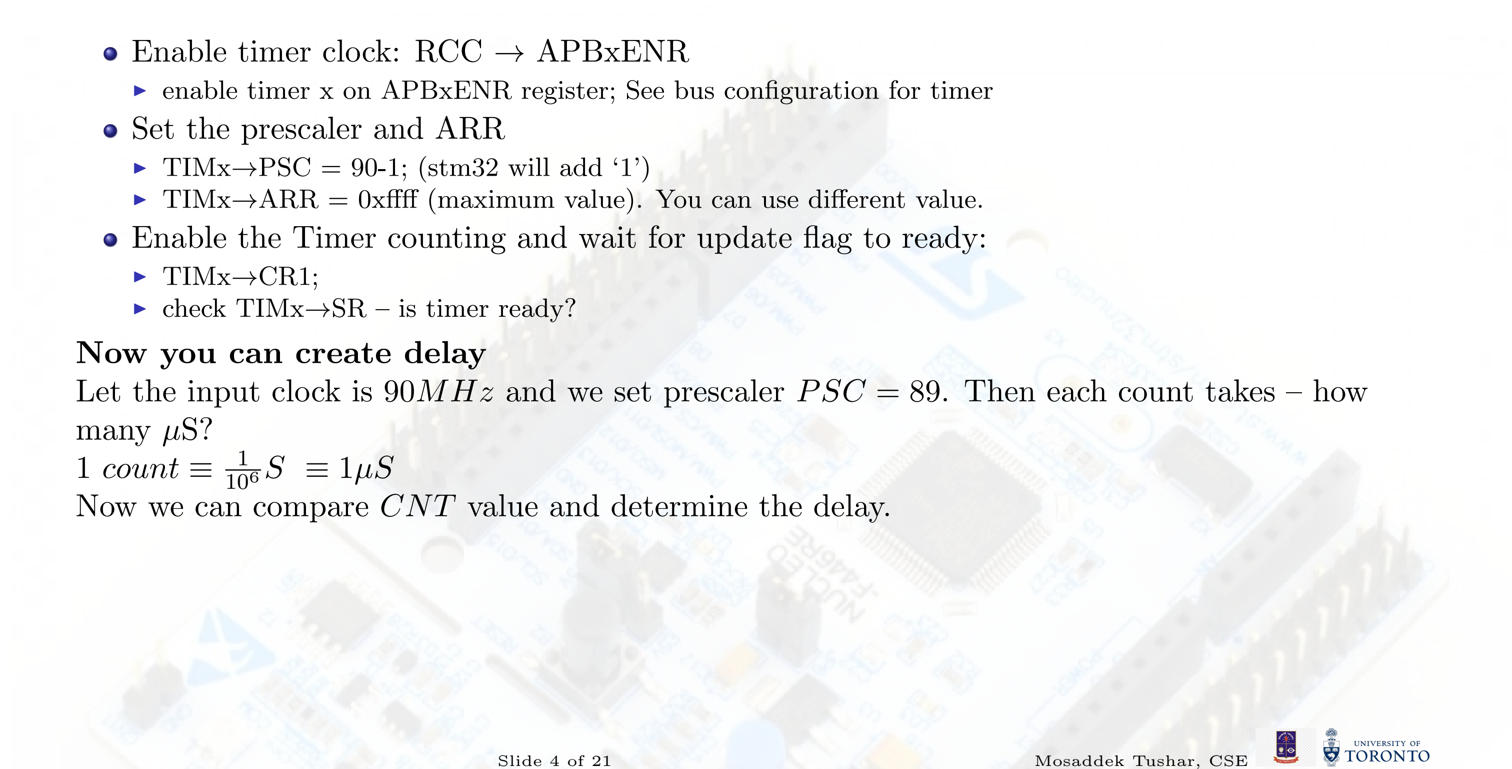
- Enable timer clock: RCC → APBxENR
  - enable timer x on APBxENR register; See bus configuration for timer
- Set the prescaler and ARR
  - TIMx→PSC = 90-1; (stm32 will add ‘1’)
  - TIMx→ARR = 0xffff (maximum value). You can use different value.
- Enable the Timer counting and wait for update flag to ready:
  - TIMx→CR1;
  - check TIMx→SR – is timer ready?

Now you can create delay

Let the input clock is $90MHz$ and we set prescaler $PSC = 89$. Then each count takes – how many μS?

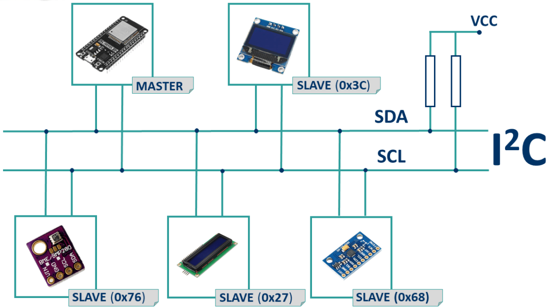
$1\ count \equiv \frac{1}{10^6}S \ \equiv 1\mu S$

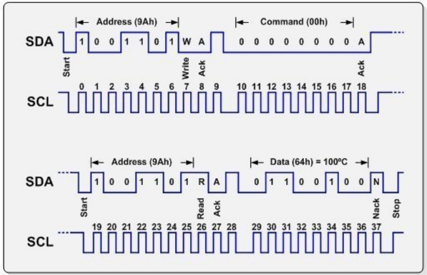
Now we can compare CNT value and determine the delay.



I^2C Serial Communication

- Also known as Inter-Integrated Circuit (IIC) Std. 100MHz, fast upto 400 MHz
- It can connect multiple slaves to a single master (Master can send to multiple slaves)
- Multiple masters to a single slave (Masters read data from a single slave)
- SDA – Serial Data Line (Bi-directional), used to transfer address, ACK and data or receive data/ACK
- SCL – Serial Clock Line (bi-directional); Send Sender (master) clock
- Transmission Mode
  - Slave transmitter
  - Slave receiver
  - Master transmitter
  - Master receiver





Transfer sequence diagram for slave transmitter

- Who is master? – Master send the START and STOP condition
- Master send 7/10 bit address plus LSB set to ‘1’ for reading
- Receiver send the ACK – Here it is master
- A high to low transition of SDA at SCL high is the start condition
- A low to high transition of SDA at SCL high defines STOP condition

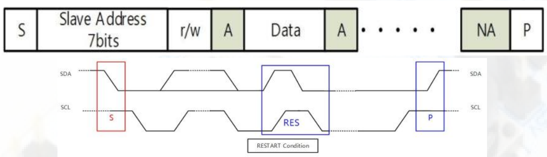
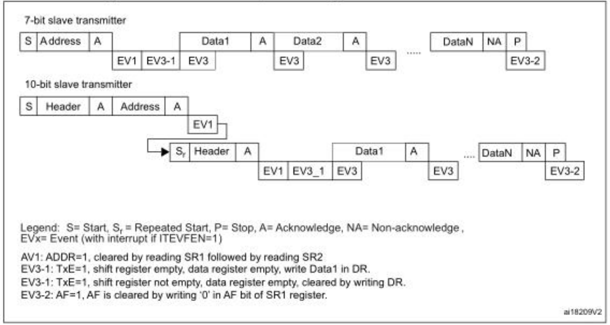


Figure 1: I2C message & Restart

Master Always initiate the transmission and reception



Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge, NA= Non-acknowledge, EVx= Event (with interrupt if ITEVFEN=1)

AV1: ADDR=1, cleared by reading SR1 followed by reading SR2
EV3-1: TxE=1, shift register empty, data register empty, write Data1 in DR.
EV3-1: TxE=1, shift register not empty, data register empty, cleared by writing DR.
EV3-2: AF=1, AF is cleared by writing ‘0’ in AF bit of SR1 register.

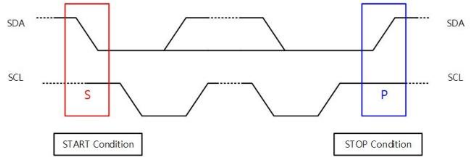


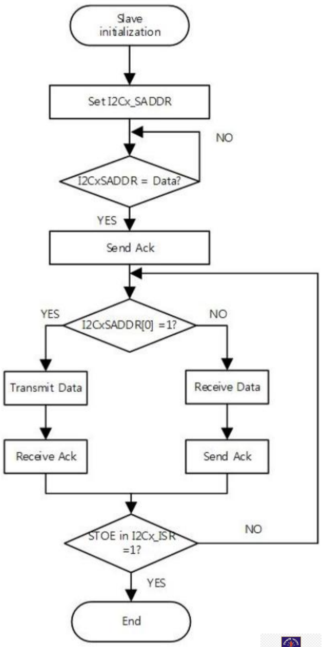
Figure 2: Start and Stop Signal I2C

Slave Transmitter Contd.

Slave ...

- Set input clock in I2C_CR2
- As soon as the start condition is detected
- Receive address from the SDA line to the shift register
- Compare the address of interface (ORA1) and ORA2 (for dual)
- Slave wait for another start if no address match found
- If address matched then send ACK to master
- Clear (wait for) ADDR flag and send data to master. Slave stretches SCL until ADDR flag is not clear
- The slave copy the data to DR register to send
- A end of byte transfer is detected by BTF in SR1 register

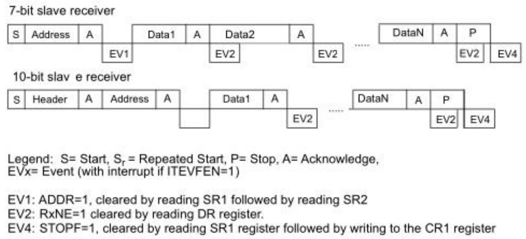
Slave Receiving and Transmitting



Slave Receiver

Slave Receiver

- Master send address (LSB of 8-bit or 10-bit is set to ‘0’)
- Slave send ACK if ACK bit in I2C_CR1 is set
- After receiving the address mactched and ADDR bit cleared slave receives byte at DR register from SDA
- An acknowledge ACK is sent if ACK in CR1 is set
- RXNE bit is set if data available in Data Register (interrupt?)
- BTF bit is set before the end of the next byte reception and until DR register is not read and RXNE is set
- BTF is cleared by a read from the I2C_DR register, stretching SCL low



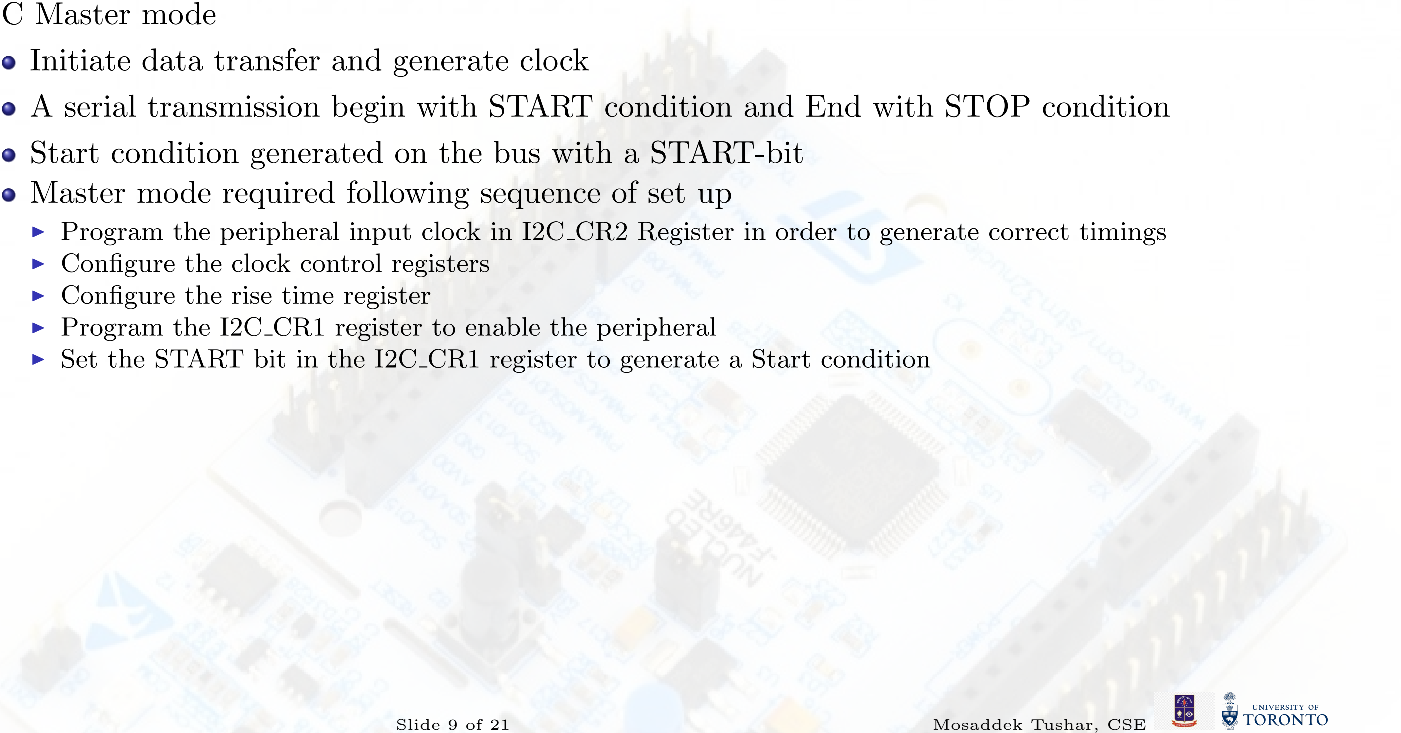
Legend: S= Start, S_r = Repeated Start, P= Stop, A= Acknowledge,
EVx= Event (with interrupt if ITEVFEN=1)

EV1: ADDR=1, cleared by reading SR1 followed by reading SR2
EV2: RxNE=1 cleared by reading DR register.
EV4: STOPF=1, cleared by reading SR1 register followed by writing to the CR1 register

Figure 3: Slave Receiver

Closing Connection/End a Session: After send all data master send a STOP Condition (generated); STOPF bit is set and cleared by reading SR1 register for the next transmission

I2C Master Mode



I2C Master mode

- Initiate data transfer and generate clock
- A serial transmission begin with START condition and End with STOP condition
- Start condition generated on the bus with a START-bit
- Master mode required following sequence of set up
  - Program the peripheral input clock in I2C_CR2 Register in order to generate correct timings
  - Configure the clock control registers
  - Configure the rise time register
  - Program the I2C_CR1 register to enable the peripheral
  - Set the START bit in the I2C_CR1 register to generate a Start condition

Configuration I2C – master mode

- Enable Clock for I2C and GPIO port
  - I2C1 and GPIOB (pin 8& 9) – SDA and Clock: Set APB1ENR bit-21 for I2C1; AHB1ENR bit-1 for GPIOB
- GPIOB pin setting
  - Set GPIOB MODER bit-16 & 18 to alternate function for pin-8&9
  - OTYPER bit-8&9 set to '1' for open drain
  - Set OSPEEDR bit-16&18 to 3 for high speed
  - Set pull-up resistors for pin-8&9 [PUPDR] bit-16&18
  - Set alternate function (Datasheet: table-11 & reference manual) AFR[1] bit-0&4 to 4
- I2C1 Setting
  - Enable and Reset I2C1: Set '1' to I2C1_CR1 bit 15 (enable) and '0' to bit-15 (reset)
  - Configure Input clock I2C1_CR2 bit-0 set 8 for 8 MHz clock (max 50 MHz) – APB1 max clock 45 MHz
  - Set I2C1 clock control register CCR [11:0](See Datasheet: table 61)

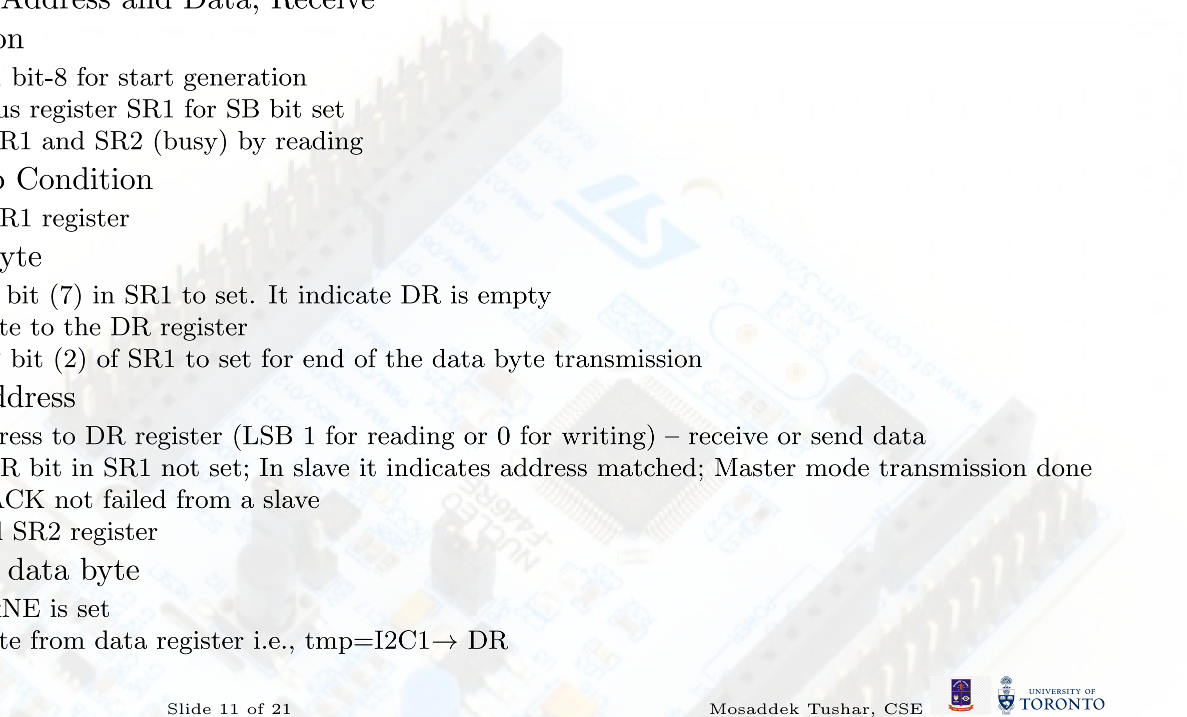
$$CCR = \frac{T_w(SCLH) + T_r(SCL)}{T_{pclk1}} \equiv \frac{4000 + 1000}{\frac{1}{45}} = 225 \tag{1}$$

  - Set I2C1 TRISE[5:0] Register – clock rising time

$$T_{rise} = \frac{T_r(SCL)}{T_{pclk1}} + 1 = \frac{1000}{\frac{1}{45}} + 1 = 46 \tag{2}$$

  - Enable the peripheral I2C1_CR1, set bit-0

Start, Stop, Send Address and Data, Receive



Start, Stop, Send Address and Data, Receive

- Start Condition
  - Set I2C1 CR1 bit-8 for start generation
  - Wait for Status register SR1 for SB bit set
  - Clear SB of SR1 and SR2 (busy) by reading
- Generate Stop Condition
  - Set bit-9 of CR1 register
- Send a data byte
  - Wait for TxE bit (7) in SR1 to set. It indicate DR is empty
  - Copy data byte to the DR register
  - Wait for BTF bit (2) of SR1 to set for end of the data byte transmission
- Send Slave Address
  - Copy the address to DR register (LSB 1 for reading or 0 for writing) – receive or send data
  - wait for ADDR bit in SR1 not set; In slave it indicates address matched; Master mode transmission done
  - It will set if ACK not failed from a slave
  - clear SR1 and SR2 register
- Receive/Read data byte
  - Wait until RxNE is set
  - Copy data byte from data register i.e., tmp=I2C1→ DR

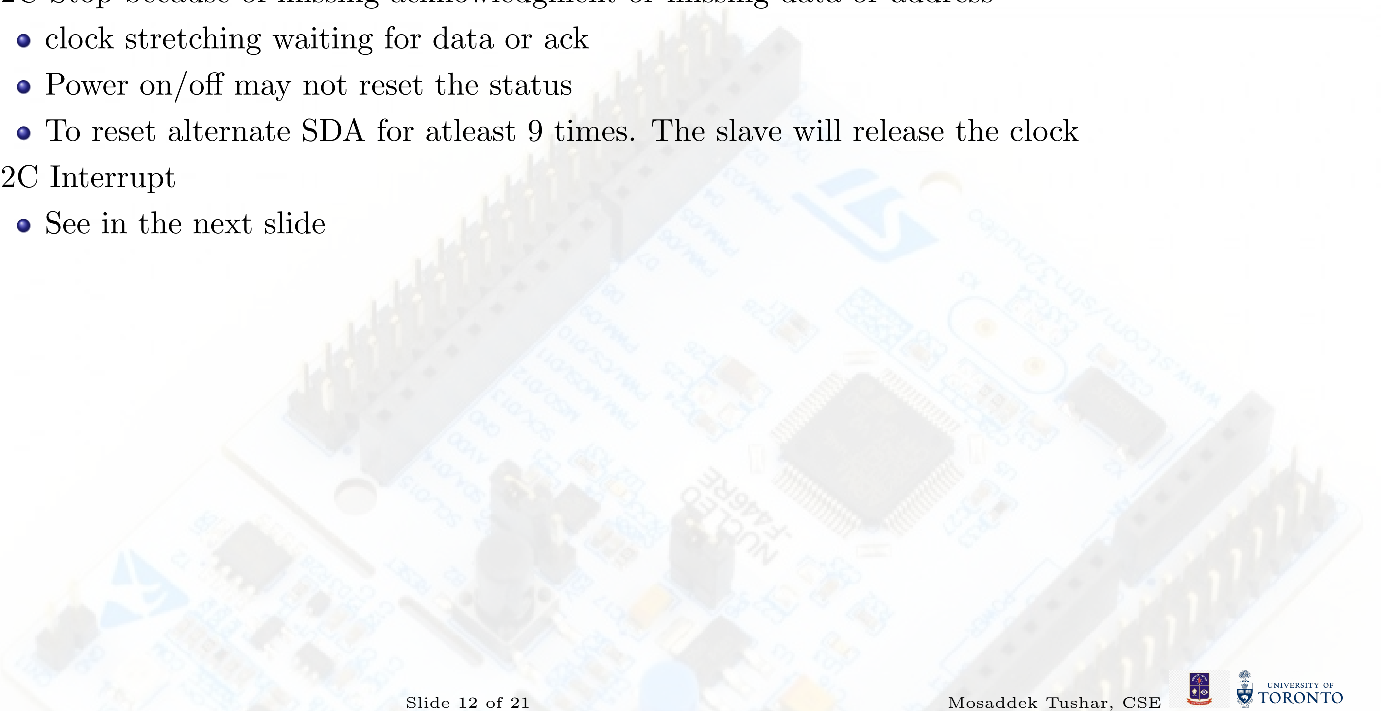
I2C Stop forever!! –Communicating and Interrupt

I2C Stop because of missing acknowledgment or missing data or address

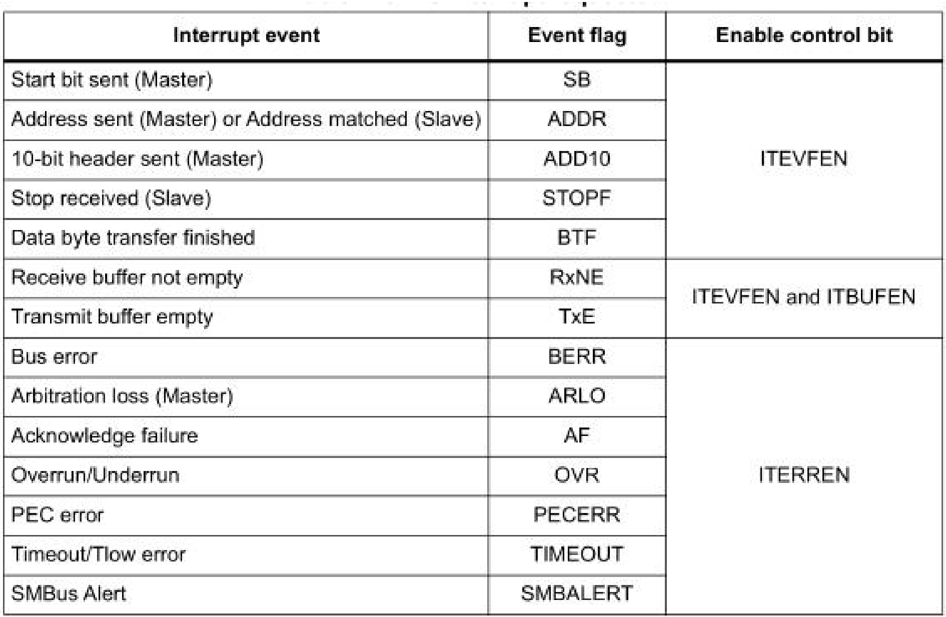
- clock stretching waiting for data or ack
- Power on/off may not reset the status
- To reset alternate SDA for atleast 9 times. The slave will release the clock

I2C Interrupt

- See in the next slide



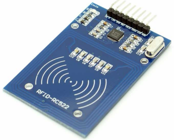
I2C Interrupts

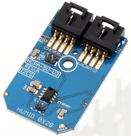


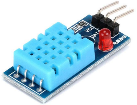
Interrupt event	Event flag	Enable control bit
Start bit sent (Master)	SB	ITEVFEN
Address sent (Master) or Address matched (Slave)	ADDR	
10-bit header sent (Master)	ADD10	
Stop received (Slave)	STOPF	
Data byte transfer finished	BTF	
Receive buffer not empty	RxNE	ITEVFEN and ITBUFEN
Transmit buffer empty	TxE	
Bus error	BERR	ITERREN
Arbitration loss (Master)	ARLO	
Acknowledge failure	AF	
Overrun/Underrun	OVR	
PEC error	PECERR	
Timeout/Tlow error	TIMEOUT	
SMBus Alert	SMBALERT	

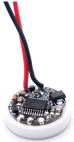
Figure 4: I2C interrupt for successfull and error communication

Use of I2C









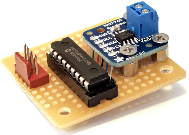
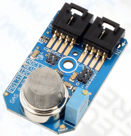


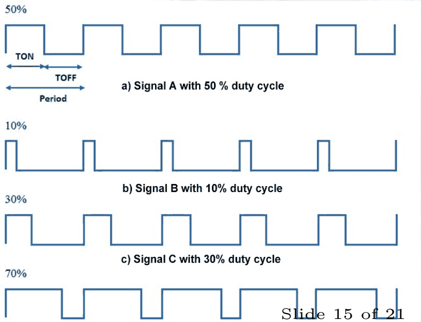
Photo by ElectroPeak



PWM - Pulse-Width Modulation

PWM - Pulse-Width Modulation

- PWM is a way to control the analog device using digital output
- Generated by comparing triangular reference wave to a input value
- PWM consists two components
  - Duty cycle: How much power it dissipates
  - frequency: define how fast it complete a cycle

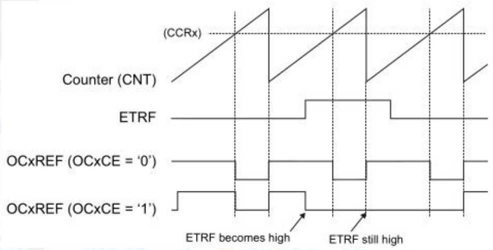


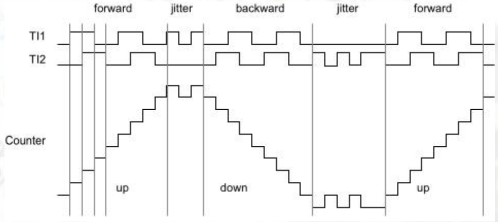
a) Signal A with 50 % duty cycle

b) Signal B with 10% duty cycle

c) Signal C with 30% duty cycle

PWM and Triangular wave





Duty cycle and frequency

Duty cycle D and frequency f of a signal are defined as,

$$D = \frac{t_{on}}{t_{on} + t_{off}} \quad \text{and} \quad f = \frac{1}{t_{on} + t_{off}} \tag{3}$$

Let energy contained of every pulse is E is a constant, $T = T_{on} + T_{off}$, $\Delta t = T_{on}$; then the rate of energy flow in every pulse is,

Peak power:

$$P_{peak} = \frac{E}{\Delta t} \tag{4}$$

Average power

$$P_{avg} = \frac{E}{T} \tag{5}$$

Therefore,

$$P_{peak}\Delta t = P_{avg}T \tag{6}$$

Hence, the duty cycle can also be defined re-arranging (6),

$$Duty\ Cycle = \frac{\Delta t}{T} = \frac{P_{avg}}{P_{peak}} \tag{7}$$

Steps for the PWM in STM32

- Decide the frequency and duty cycle
- Select the timer (Here TIM1)
- Choose the number of PWM output (channels) and Related GPIO pins (Data sheet Table 10)
  - TIM1_CH1:PA8, TIM1_CH2:PA9, TIM1_CH3:PA10, TIM1_CH4:PA11
- Set the prescaler PSC register
- Configure GPIO port for alternate function (AF1) – data sheet Table 11
- Configure TIM1 (CR1, CCMRx, CCER, BDTR, CCRx)
- ARR Register
- for Advanced configuration use DMA – really fun!!

STM32 Timer and PWM

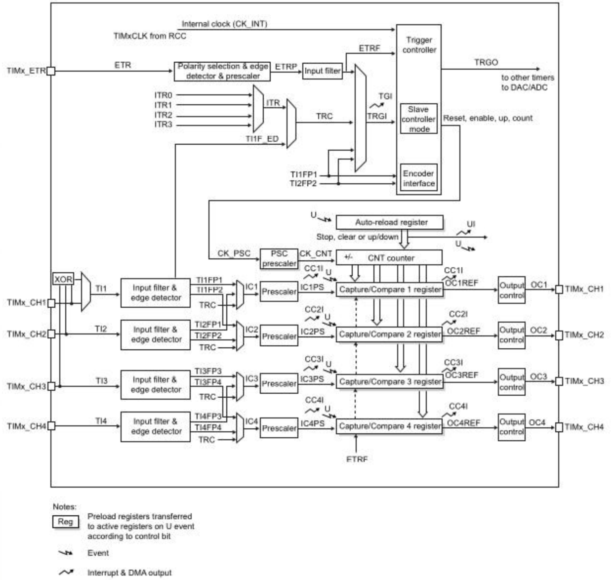
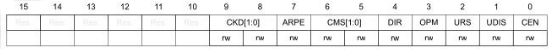


Figure 5: Timer CKT

Register Set (Exact Match TIM1&TIM8)



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 6: Control Register TIM1/8 CR1

- CEN: Enable Counter, UDIS: Update Event Disable (overflow, underflow), URS: update request source (0: DMA, CNT overflow underflow; 1:)
- CMS:'00': Edge aligned mode (up or down count); '01': center aligned mode 1 (interrupt (IE) on counting down), '10': Center-aligned mode 2 (IE counting up), '11': Center-aligned mode 3 (IE counting up or down)
- DIR: up (0) or down (1) counter
- ARPE: Auto reload preload enable (not buffered or buffered)
- OPM: One pulse mode – generate a single pulse and not stopped at update event (0), stopped at next update event

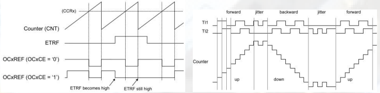


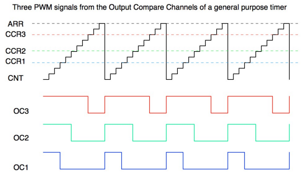
Figure 7: Edge and Center Aligned

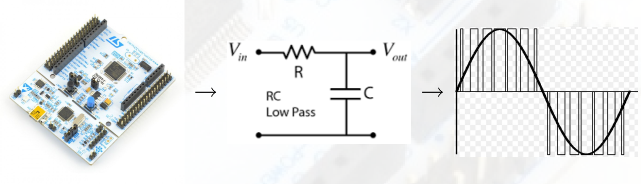
Decide Frequency and Duty Cycle

Decide Frequency and Duty Cycle

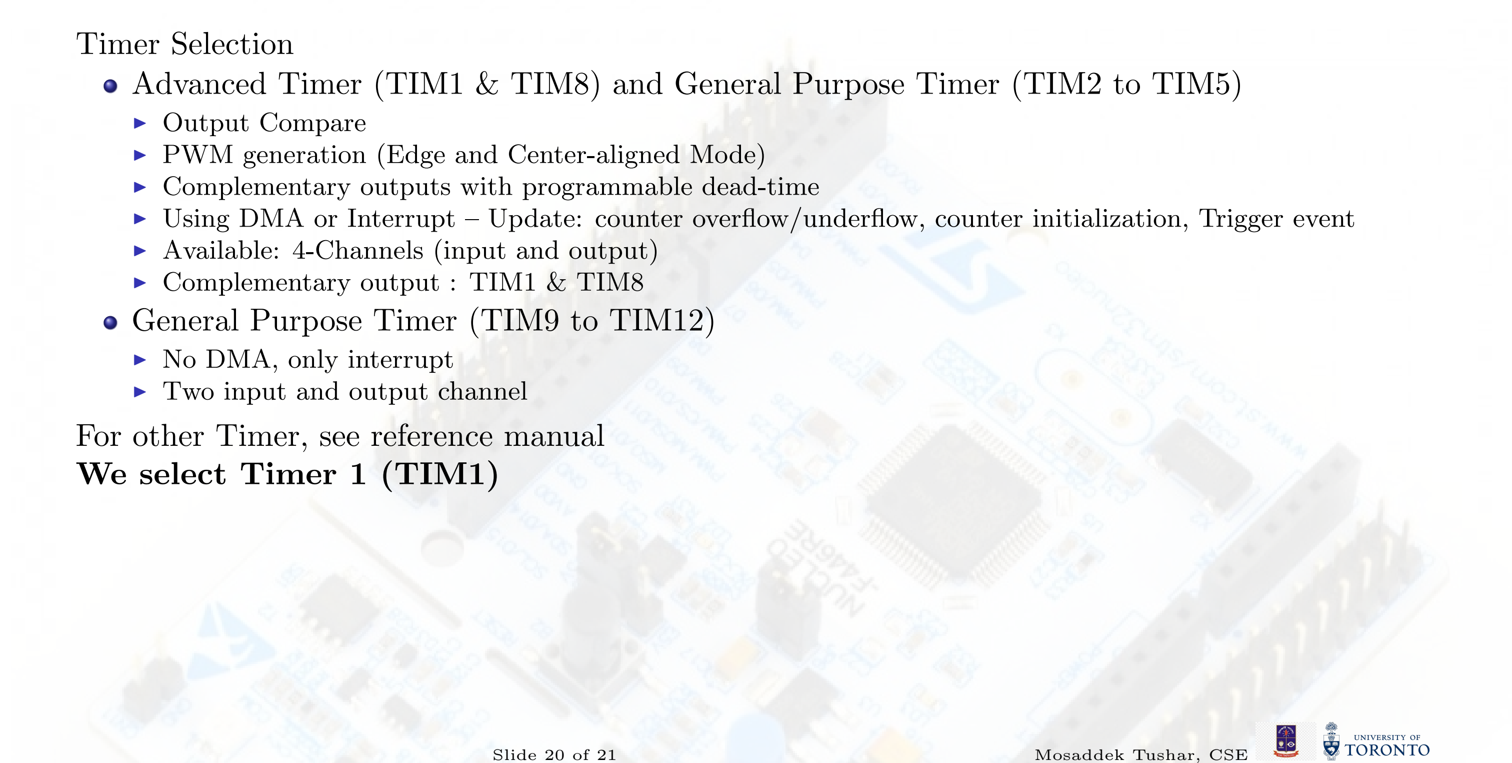
- Frequency – determine the carrier of the signal for PWM or analog signal for control
- Frequency Requirement coming from use – such as rpm to run a train, car, or lift (let us assume): 10 kHz
- Duty Cycle: How much output power is needed – torque (moment of force). Let us presume 30% of available
- Timer Counting UP, Down or both
- Select or Dynamically change the output comparator (CCRx)

- Generate the triggering point for the full analog cycle





Timer Selection



Timer Selection

- Advanced Timer (TIM1 & TIM8) and General Purpose Timer (TIM2 to TIM5)
  - Output Compare
  - PWM generation (Edge and Center-aligned Mode)
  - Complementary outputs with programmable dead-time
  - Using DMA or Interrupt – Update: counter overflow/underflow, counter initialization, Trigger event
  - Available: 4-Channels (input and output)
  - Complementary output : TIM1 & TIM8
- General Purpose Timer (TIM9 to TIM12)
  - No DMA, only interrupt
  - Two input and output channel

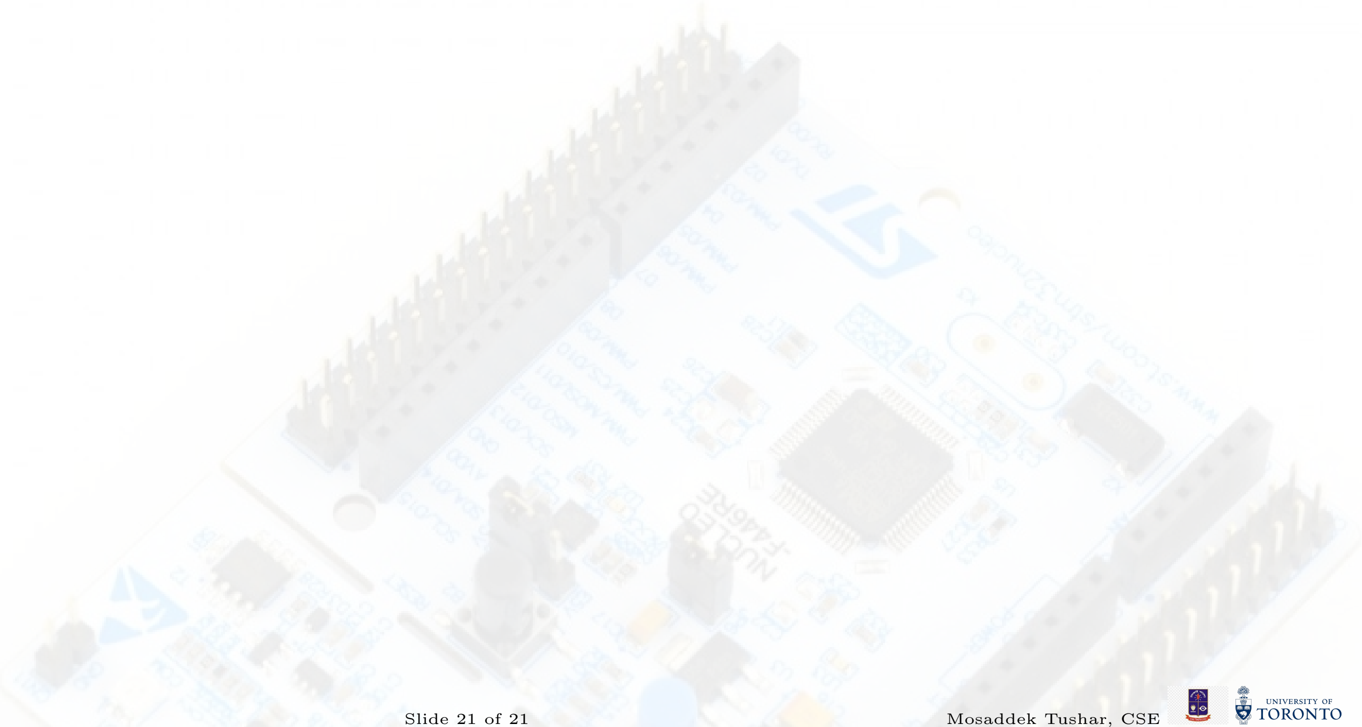
For other Timer, see reference manual

We select Timer 1 (TIM1)

Example 3-Phase PWM

The following information should be noted:

- Use GPIOA pins PA8, PA9, and PA10 for three-phase control.
- GPIOB pins PB13, PB14, and PB15 are complementary to the above pins.
- The CCR1, CCR2, and CCR3 values are crucial in the signal generation. These values are compared with the CNT (counter register) and are used to toggle the output, providing precise control over the signal generation process.
- You can adjust the CCRx values to change the duty cycles at each update event, allowing you to customize the signal output according to your specific requirements.
- To generate a signal output on PA8 (and complement PB13) with a 50% duty cycle, the ARR register should contain 999 (+1), and the CCR1 register value should be 500. Similarly, for a 75% duty cycle, the CCR1 value would be 750. These values are then loaded to CCR1 at the next update event, demonstrating the process of generating different duty cycles.
- The updates on CCRx are stored in the shadow register until the next update event occurs.
- By using PWM, you can create various signal patterns.
- The duty cycle regulates the power delivery to the output.


NUCLEO
F446RE
www.st.com/stm32nucleo
RESET

Example

3-Phase PWM example

```
void Config_TIM1(void){
RCC->APB2ENR |= 1<<0; //Enable timer 1

TIM1->CR1 &= ~(1U<<0); // reset CEN and disable counter
/*
TIM1->CR1 &= ~(3U<<5); //clear CMS
TIM1->CR1 &= ~(1U<<4); //enable upcounter
*/
/**
* Load counter range,
* input clock frequency to the counter, and
* configure enable auto reload register
**/
TIM1->ARR = 999; // counter maximum value 1000 -- expect to get 1KHz
TIM1->PSC = 179; // for 1 MHz clock input
TIM1->CR1 |= 1<<7; //ARPE --  auto preload and reload eneable
TIM1->CR1 |= 1<<5; //enable center alignment
TIM1->CR2 |= 1<<3; //Capture/Compare DMA Selection
```

Example

```
/***
* Set PWM channel
**/
/*
TIM1->CCMR1 &= ~(7U<<4); // CH1 clear
TIM1->CCMR1 &= ~(7U<<12); //CH2 clear
TIM1->CCMR2 &= ~(7U<<4); //CH3 clear
*/
//set the output channel for the PWM mode 1
TIM1->CCMR1 |= 0x6<<4;
TIM1->CCMR1 |= 0x6<<12;
TIM1->CCMR2 |= 0x6<<4;

// OCxPE for allow value of CCRx is load to the active register
// when an update event occur
TIM1->CCMR1 |= 1<<3;
TIM1->CCMR1 |= 1<<11;
TIM1->CCMR2 |= 1<<3;
```

Example

```
// Configur channels 3+3 (complementary) for output mode; Enable output
TIM1->CCER |= (1<<0) | (1<<2) | (1<<4) | (1<<6) | (1<<8) | (1<<10);
// Set active polarity for the output -- active low
TIM1->CCER |= (1<<1) | (1<<3) | (1<<5) | (1<<7) | (1<<9) | (1<<11);
//OSSR -- Off state selction for run mode low
TIM1->BDTR |= 1<<11;
TIM1->BDTR |= 0x4D<<0; // please take care of dead time for state change
\\between normal complementary output
TIM1->BDTR |= 1<<15; //main output enable

//set timer autoload preload enable
TIM1->CR1 |= 1<<7;

//timer 1 update DMA request enable
//TIM1->DIER |= 1<<8; //This should be uncomment to the USE of DMA

//channel 1/2/3 counter initialized (ARR 1000!!)
TIM1->CCR1 |= 500; //50% duty cycle
TIM1->CCR2 |= 300; //30% duty cycle
TIM1->CCR3 |= 800; //80% duty cycle
```

Example

```
//register CCR1 offset 0x34 for DMA
//Uncommet for DMA
//uint8_t offset=(&(TIM1->CCR1)-&(TIM1->CR1));
//offset &= 0x0F;
//TIM1->DCR |= (uint8_t)(offset<<0);
//TIM1->DCR |= 13<<0;

//DMA Burst size
//uncomment for DMA
//TIM1->DCR |= 3<<8; //4-DMA transfer

// uncomment enable interrupt
TIM1->DIER |= 7<<1; //enable matching interrupt

//Finally enable counting
TIM1->CR1 |= 1<<0;
}
```

Example

3-Phase PWM example GPIO Setting

```
void Config_GPIO_TIM(void){
/* GPIO Port for Channel 1, 2 & 3, 1N, 2N, 3N
* port: GPIOA and GPIOB ->  PA8, PA9, PA10, PA7/PB13, PB0/PB14, PB1/PB15
*/
RCC->AHB1ENR |= (1<<0) | (1<<1); // Enable GPIOA and GPIOB
//set Moder Register; reset and set -- optional
GPIOA->MODER &= ~(3U<<16);
GPIOA->MODER &= ~(3U<<18);
GPIOA->MODER &= ~(3U<<20);

\\ Optional
GPIOB->MODER &= ~(3U<<26);
GPIOB->MODER &= ~(3U<<0);
GPIOB->MODER &= ~(3U<<2);

//set the values for Alternate function for PWM
GPIOA->MODER |= (2<<16) | (2<<18) | (2<<20); //pin PA8, PA9,PA10 for output
GPIOB->MODER |= (2<<26) | (2<<0) | (2<<2); //pin PB13, PB0, PB1 for
\\complementary output
```

Example

```
//Ouput speed
GPIOA->OSPEEDR |= (3<<16) | (3<<18) | (3<<20);
GPIOB->OSPEEDR |= (3<<26) | (3<<0) | (3<<2);
//Set the alternate function for the PWM output
//clear the alternate function and set (optional)
GPIOA->AFR[1] &= ~(15U<<0);
GPIOA->AFR[1] &= ~(15U<<4);
GPIOA->AFR[1] &= ~(15U<<8);
GPIOB->AFR[0] &= ~(15U<<0);
GPIOB->AFR[0] &= ~(15U<<8);
GPIOB->AFR[1] &= ~(15U<<20);
//Set the alternate funtion for 3-phase PWM and complements
GPIOA->AFR[1] |= (1<<0) | (1<<4) | (1<<8);
GPIOB->AFR[0] |= (1<<0) | (1<<8);
GPIOB->AFR[0] |= (1<<20);
}
```