ARDUINO PWM PROGRAMMING

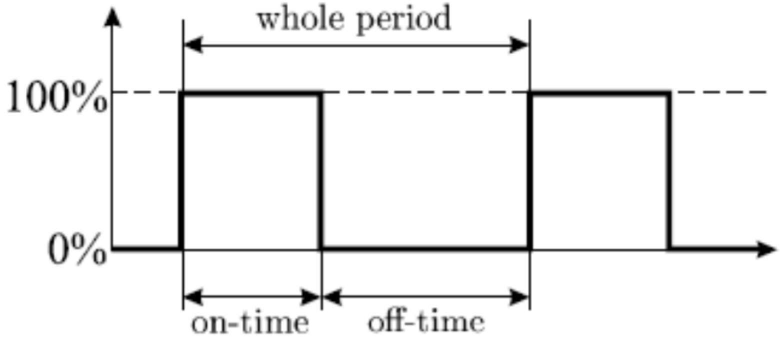
RAVI SUPPIAH

LEARNING OBJECTIVES

- By the end of this lecture, you will be able to:
  - Understand how to configure the Microcontroller Registers to generate the required PWM signals

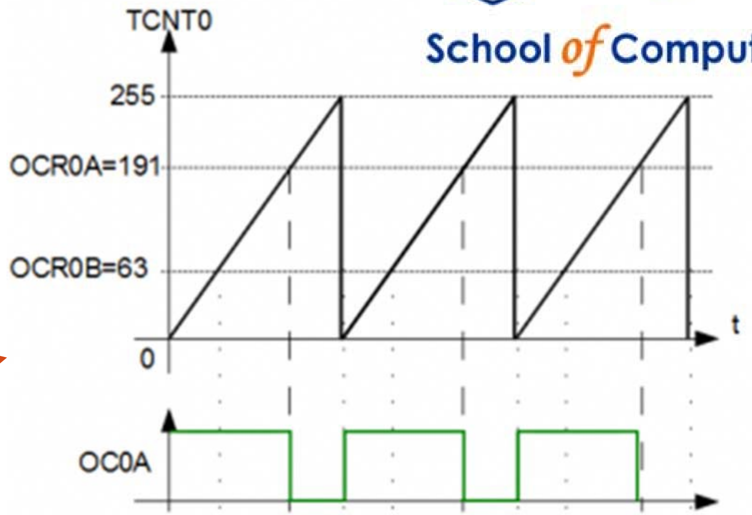
PWM CONCEPTS

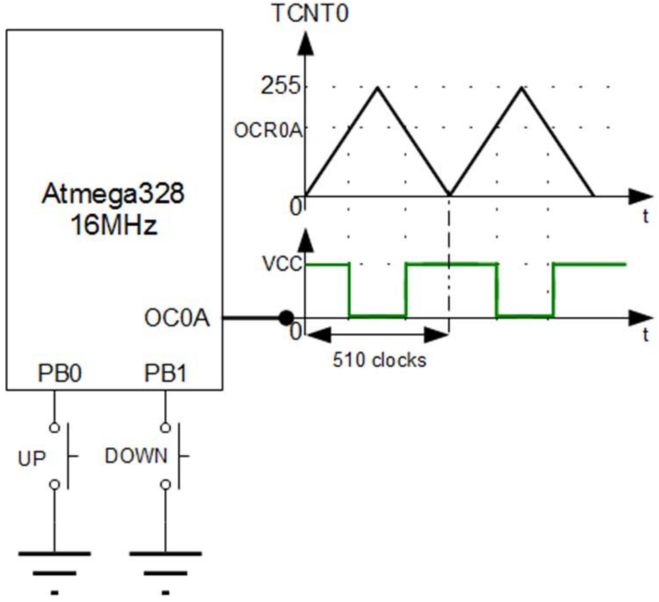
- Digital bits are measured or encoded over a fixed period of time.
- Analog values are encoded by the proportion of "1" ("on") time to "0" ("off") time within this fixed time. This proportion is called a "Duty Cycle", usually denoted D.
- E.g. if a PWM signal consists of 8-bits, and the analog signals are from 0 to 5v, then 10000000_2 corresponds to 2.5v. This is a 50% duty cycle.



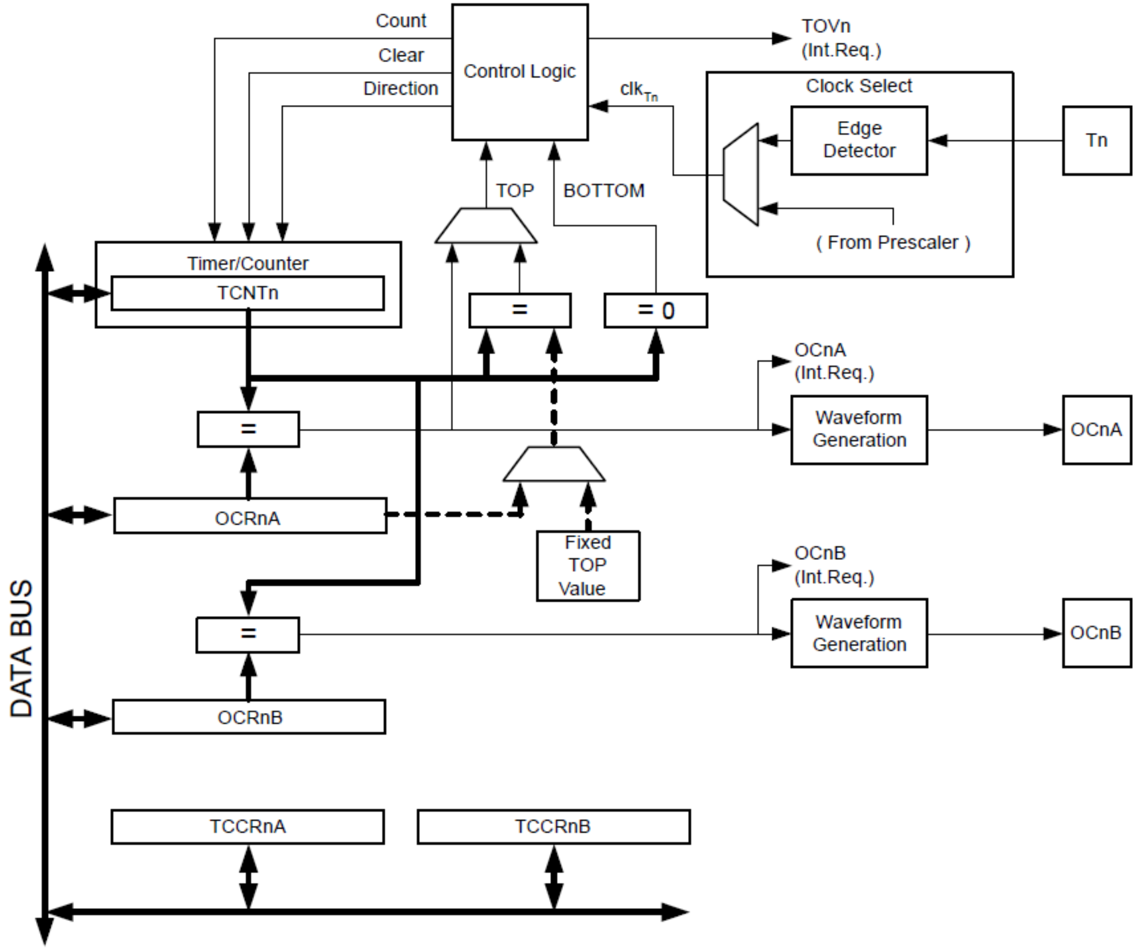
GENERATING ANALOG OUTPUT PWM ON THE ATMEGA328

- PWM is generated using the timers on the Atmega.
- There are two PWM modes on the
- Atmega:
  - Fast PWM, allowing higher frequencies.
  - Phase Correct PWM, allowing symmetric wave-forms, better resolution at expense of maximum frequency.
  - More suited for motors.
- We will focus only on phase-correct PWM.





USING TC0 TO GENERATE PWM



8-bit Timer/Counter Block Diagram

SELECT THE CLK SOURCE

- The Timer/Counter can be clocked by an internal or external clock source. The clock source is selected by writing to the Clock Select (CS0[2:0]) bits in the Timer/Counter Register (TCCR0B).

Bit	7	6	5	4	3	2	1	0
	FOC0A	FOC0B			WGM02	CS0[2:0]		
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

CA02	CA01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

WHAT'S YOUR FREQUENCY

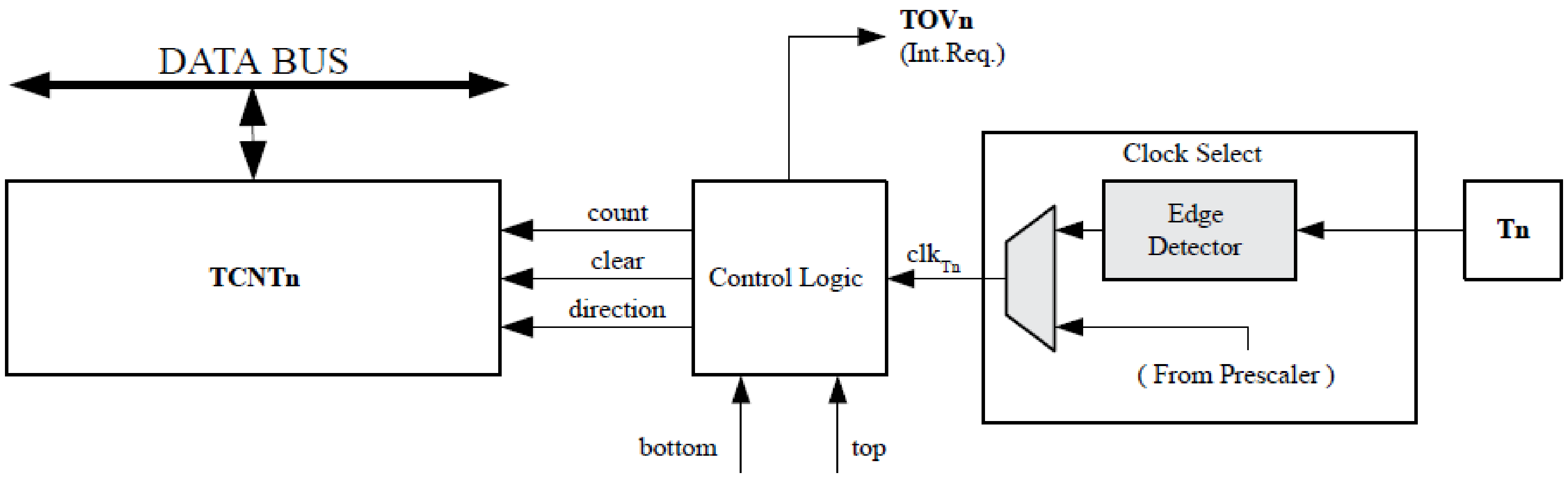
- The clk source that you select is linked with the desired PWM frequency.
- Lets set a desired frequency of 500Hz.
- We now need to select a Prescaler based on the following formula:

$$f_{\mathrm{OCnxPCPWM}} = \frac{f_{\mathrm{clk\_I/O}}}{N \cdot 510}$$

- N represents the prescaler factor (1,8,64,256, or 1024).
- Assuming that f_{clk} is 16Mhz (Uno Board), substituting f_{PWM} as 500, we solve for P as
  N = 62.745
- The closest N 64, giving us f_{PWM} = 490Hz.

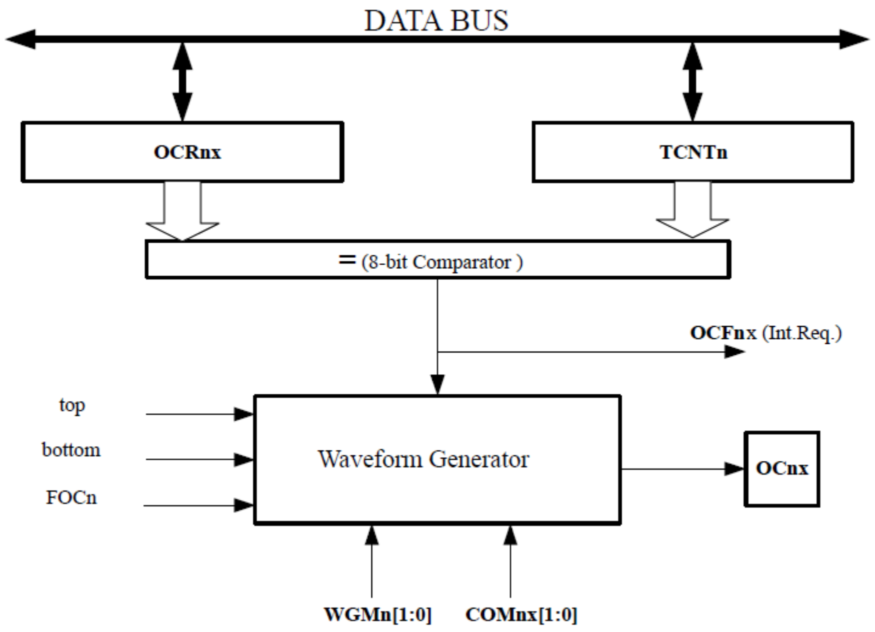
COUNTER UNIT

- Once the clk is set, the TCNTn register will increment based on the desired frequency.



OUTPUT COMPARE UNIT

- This unit continuously compares TCNT0 with the Output Compare Registers (OCR0A and OCR0B).
- Whenever TCNT0 equals OCR0A or OCR0B, the comparator signals a match.
- A match will set the Output Compare Flag (OCF0A or OCF0B) at the next timer clock cycle.
- If the corresponding interrupt is enabled, the Output Compare Flag generates an Output Compare interrupt.



SETTING DUTY CYCLE

- The duty cycle is determined by the value stored in the OCR0A register, based on the following formula:
  - $D = \frac{OCR0A}{255} \text{ x } 100$
- For a Duty Cycle of 50%, the desired voltage is 2.5V.
- The value that we want to write to OCR0A is 255/2 = 127.5 The closest we can get is 128 (or 127).
- If we choose 128, then the actual Duty Cycle will be 50.196, giving a voltage of 2.51V.

SETTING DUTY CYCLE

- OCR0A Register

Bit	7	6	5	4	3	2	1	0
	OCR0A[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – OCR0A[7:0]: Output Compare 0 A
The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT0). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC0A pin.

- Example values for different PWM Duty Cycle

Desired D (OCR0A)	Desired V (Vcc=5V)	OCR0A	Actual D	Actual V (Vcc=5V)
0 (0)	0v	0	0	0v
25 (63.75)	1.25v	64	25.098	1.255v
50 (127.5)	2.5v	128	50.196	2.510v
75 (191.25)	3.75v	191	74.902	3.745v
100 (255)	5.00v	255	100	5v

INTERRUPTS

- The timer can be configured to generate Interrupts whenever there is a Output Compare or when TCNT0 rolls over.
- This is useful when setting a new PWM duty cycle at the end of the current PWM signal.

Name: TIMSK0
Offset: 0x6E
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
						OCIEB	OCIEA	TOIE
Access						R/W	R/W	R/W
Reset						0	0	0

```
TIMSK0|=0b10;    // Enables OCIE0A IRQ
```

INITIAL AND RELOAD VALUES

- To generate the phase correct PWM, we need to
  - Load 0 into the initial count register TCNT0.
    - TCNT = 0;
  - Load the desired value into the OCR0A based on the require PWM Duty Cycle.. Fo 50 % DC, we will load it with a value of 128.
    - OCR0A = 128;

SELECTING THE DESIRED PWM

- We need to configure the WGM [2:0] bits in TCCR0A/B register to configure the desired type of PWM.

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCR0x at	TOV Flag Set on[(1)(2)]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

SELECTING THE DESIRED PWM

- There are two Phase Correct PWM modes. We will be using mode 1.
- We also need to set COM0A1:0 to 0b10. This will ensure that when TCNT0 counts up from 0 to OCR0A it clears the OC0A, and sets it when TCNT0 counts down from 255 to OCR0A.

Name: TCCR0A
Offset: 0x44
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x24

Bit	7	6	5	4	3	2	1	0
	COM0A1	COM0A0	COM0B1	COM0B0			WGM01	WGM00
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match when up-counting. Set OC0A on Compare Match when down-counting.
1	1	Set OC0A on Compare Match when up-counting. Clear OC0A on Compare Match when down-counting.

- We will use

TCCR0A=0b10000001;

FINALLY, LETS SEE IT IN ACTION!

- You are now ready for your Studio!
- You will first get to observe the PWM in action and take some measurements.
- You will then apply the PWM signal to an LED and see its effect.
- Finally, you will get to control your motors using the PWM signal and achieve complete motor control for your robot.

THE END!