University of Nottingham
UK | CHINA | MALAYSIA

Lecture 3

Timers and Counters

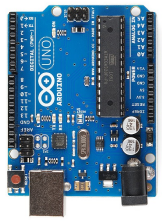
Mechatronics
MMME3085

Module Convenor – Abdelkhalick Mohammad

Objectives of lecture

- To introduce the concept of timer/counters and their uses in the Arduino
- To illustrate the use of register-level programming of hardware
- To introduce the concept of serial data communication
- To gain experience of using datasheet (a complex but complete collection of data on a device.)

A typical Mechatronics System



Computer or micro-processor

2,3,10

1,4,10

Program e.g., in C

Buses in computer

Digital output inter-face; timer

Digital input inter-face; counters

Digital signal

DAC

Analogue signal

Digital signal

5,6

Digital signal

ADC

Analogue signal

Digital signal

7,8,9

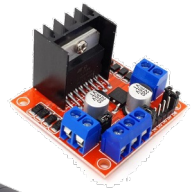
Electronic hardware

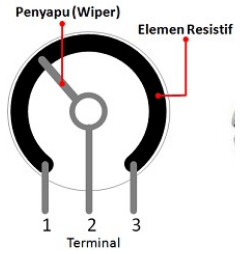
Actuators

Mechanical system

Sensors

Electronic hardware






University of
Nottingham
UK | CHINA | MALAYSIA
Recap

Digital input and output – so far…

- Last time we looked at
  - Different forms of signals: analogue, digital, *trains of pulses*
  - It was more about detail on *digital signals*
  - *Computer architecture* and how we need to make an interface to the computer
  - Digital input and output including `digitalRead()`, `digitalWrite()` and the use of direct port access via *registers*

Digital input and output – so far…

- Digital input:
  - *The Arduino way*

    ```
    bool pinState = digitalRead(12);
    ```

  - *Using registers*

    Pin 12 is bit no 6 of Port B!

    ```
    bool pinState = PINB & (1 << 6);
    ```

- Digital output:
  - *The Arduino way*

    ```
    digitalWrite(13, pinState);
    ```

  - *Using registers*

    Pin 13 is bit no 7 of Port B

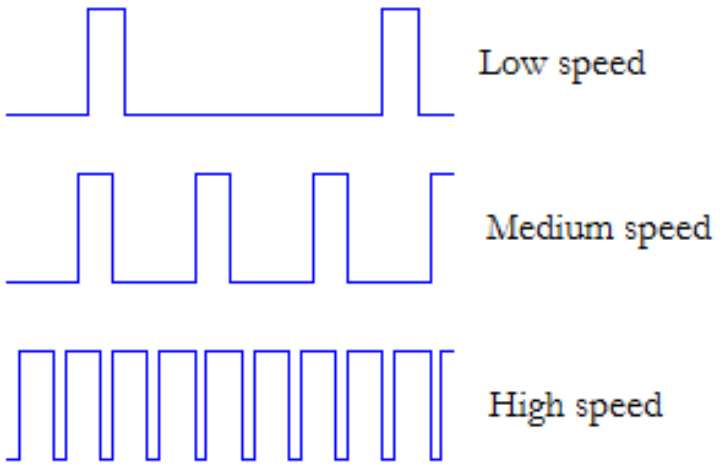
    ```
    PORTB |= (1 << 7);  // sets bit 7
    ```

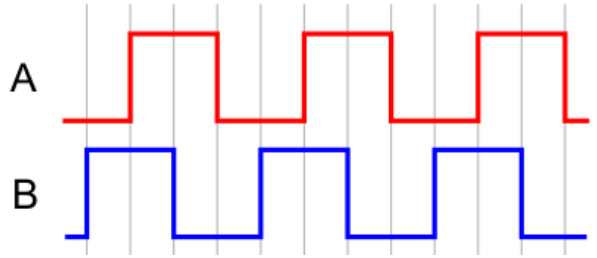
Note: **PINB** is a register, and it lives at address 0x23


University of
Nottingham
UK | CHINA | MALAYSIA
Trains of pulses

Dealing with trains of pulses

- Usually regarded as a form of digital signal:
  - Voltage alternates between *two levels* (square wave) e.g., using TTL standard.
  - Freq. of the pulse train or the no. of pulses.
  - If we treat this as a digital signal, we are interested in “rate” rather than “state”.
  - To have an up/down count (e.g., encoder) we need two sets of pulses 1/4 phase out of step (quadrature pulses) and a suitable decoder.





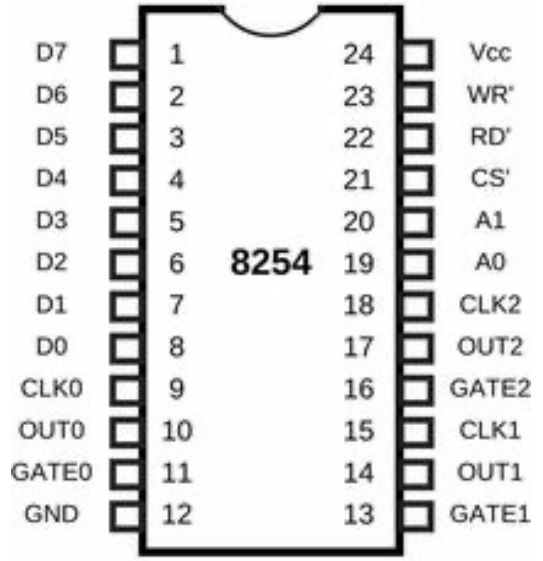
Dealing with trains of pulses

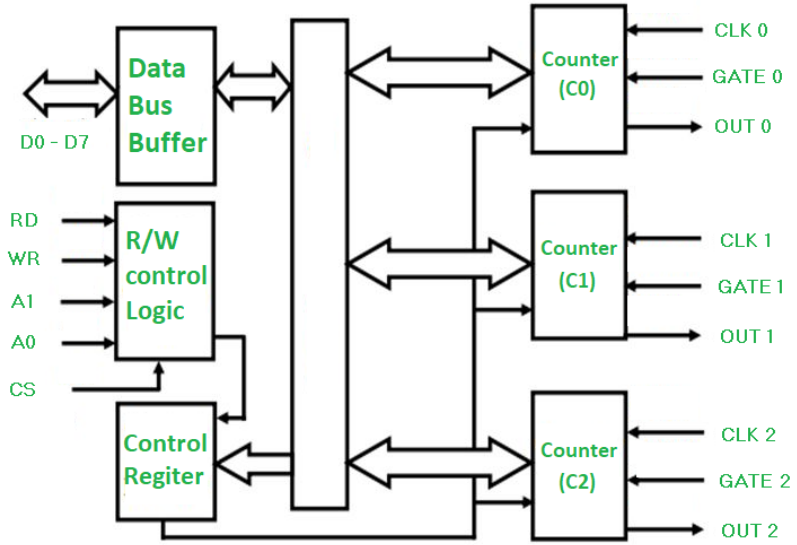
- In principle:
  - Can use repeated read of inputs to monitor for input events (i.e., "*polling*").
  - Can create pulses using software – *write high output*, *wait*, write low output, repeat!
- Not advisable with general-purpose operating systems e.g., Windows
- For example, stepper motor pulses slowed on mouse moves or antivirus activation!
- Even in Arduino, may not be fast enough!

What is the solution?!

Hardware Timers or Counters (T/C)

- A piece of hardware that is *dedicated* for counting pulses or pulse generation!
- Useful for *real time* generation of pulses and frequency counting on PC and other computers.
- As the 8254 T/C, available on boards that simply slot into back of PC.
- They are *built into* the Arduino Uno, Mega etc. but are slightly different from those on 8254.

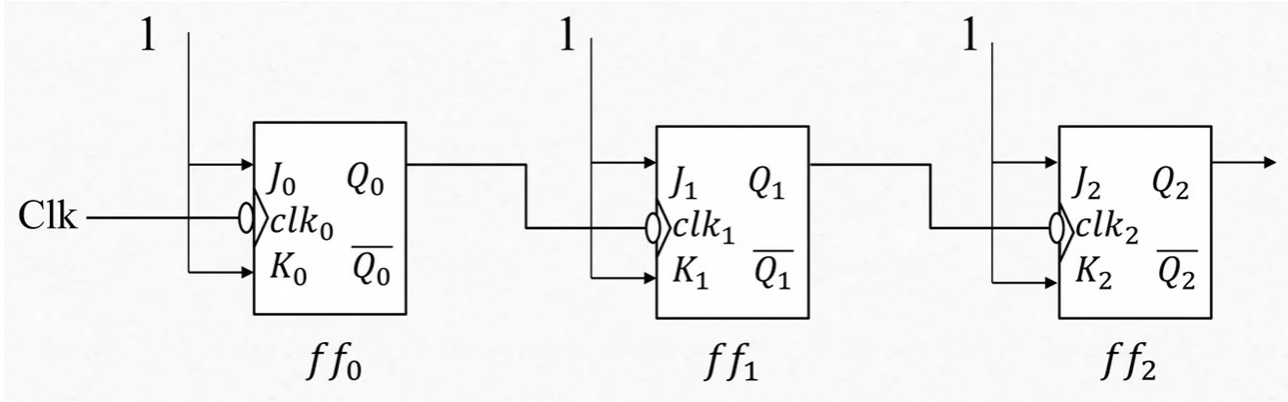


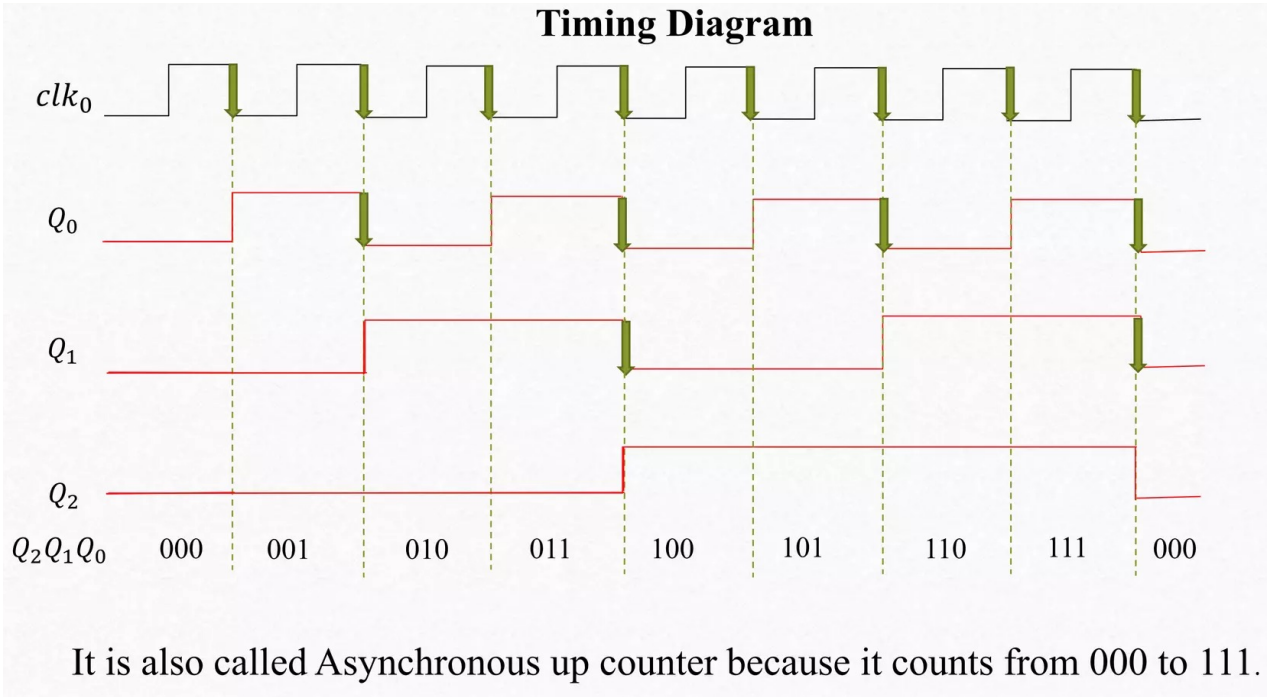



University of
Nottingham
UK | CHINA | MALAYSIA
What is a Counter?!

What is a Counter?!

A counter is a device which stores the number of times a particular event has occurred (*a clock*). The values on the (*output*) lines represent a number in the binary. Each pulse applied to the clock input increments or decrements the number on the counter.

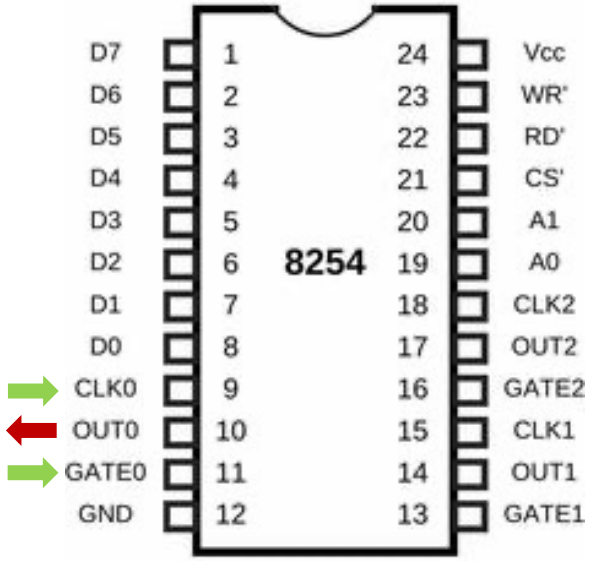




It is also called Asynchronous up counter because it counts from 000 to 111.

What is a Counter?!

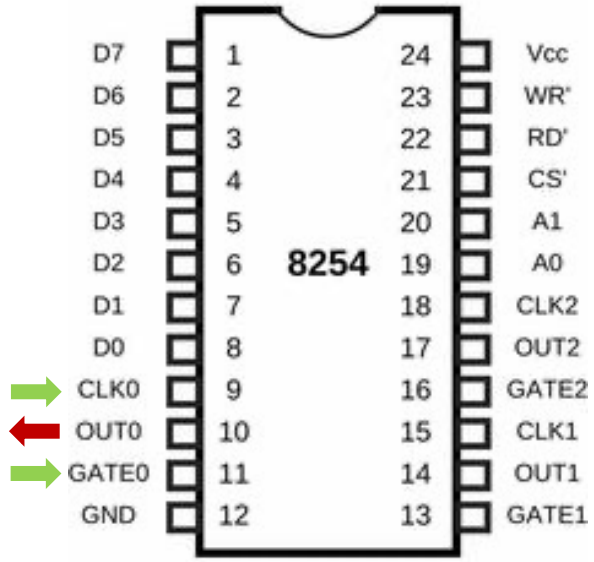
- **Counter Register**
  - Stores the current count
- **Clock** (also known as source)
  - Input signal that changes the current count
  - Active edge (*rising or falling*) of input signal changes the count
  - Choose if count *increments* or *decrements* on an active edge
- **Gate** (not directly available on Arduino counters)
  - Input signal that controls *when counting occurs*
  - Counting can occur when gate is high, low, or between various combinations of rising and falling edges
- **Out**
  - Output signal used to generate pulses



What is a Timer?!

Same hardware as a counter but:

- The clock is connected to *a known frequency* source.
- The frequency source can be external or internal source.
- The counter counts to a particular value and then it can *flip/toggle* its output (e.g., from 0 to 1 or from 1 to 0).
- In this way we can generate a *train of pulses*
- We will visit this in more details later!

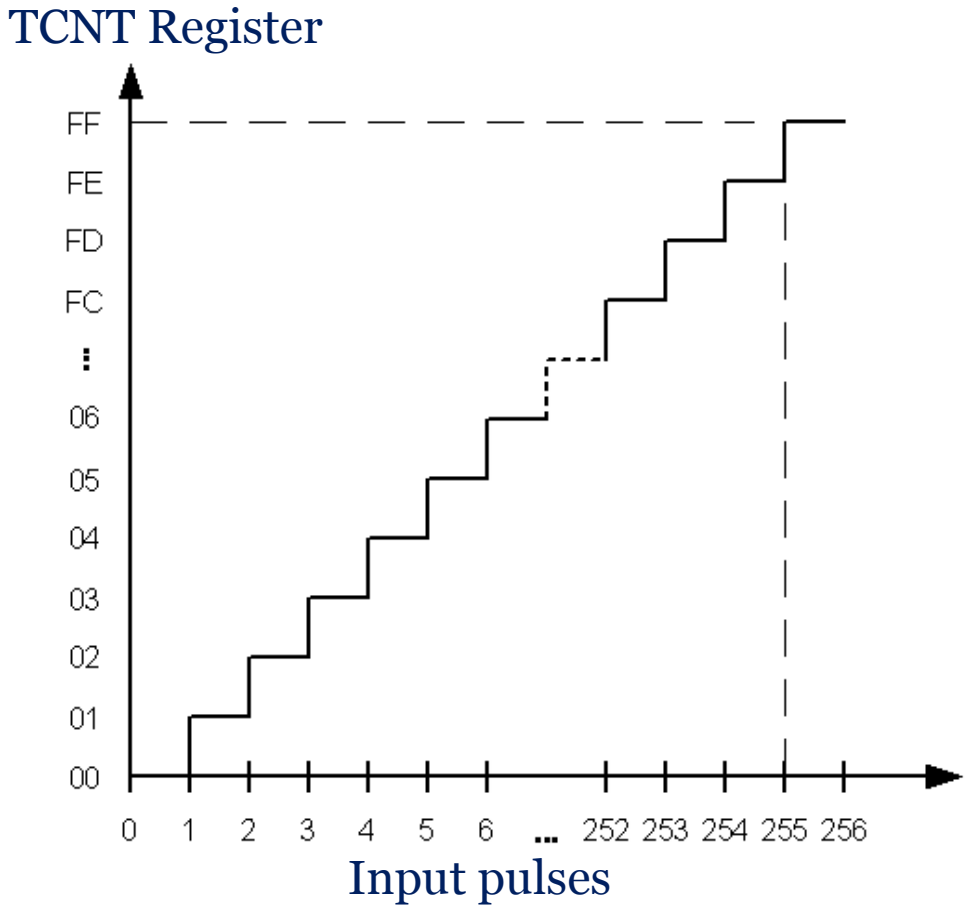


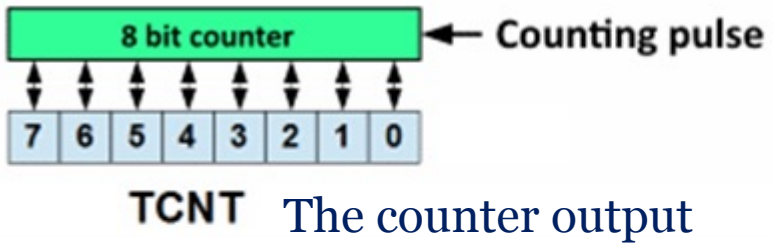
University of Nottingham
UK | CHINA | MALAYSIA

Timers and Counters on the Arduino

T/C in the AVRs

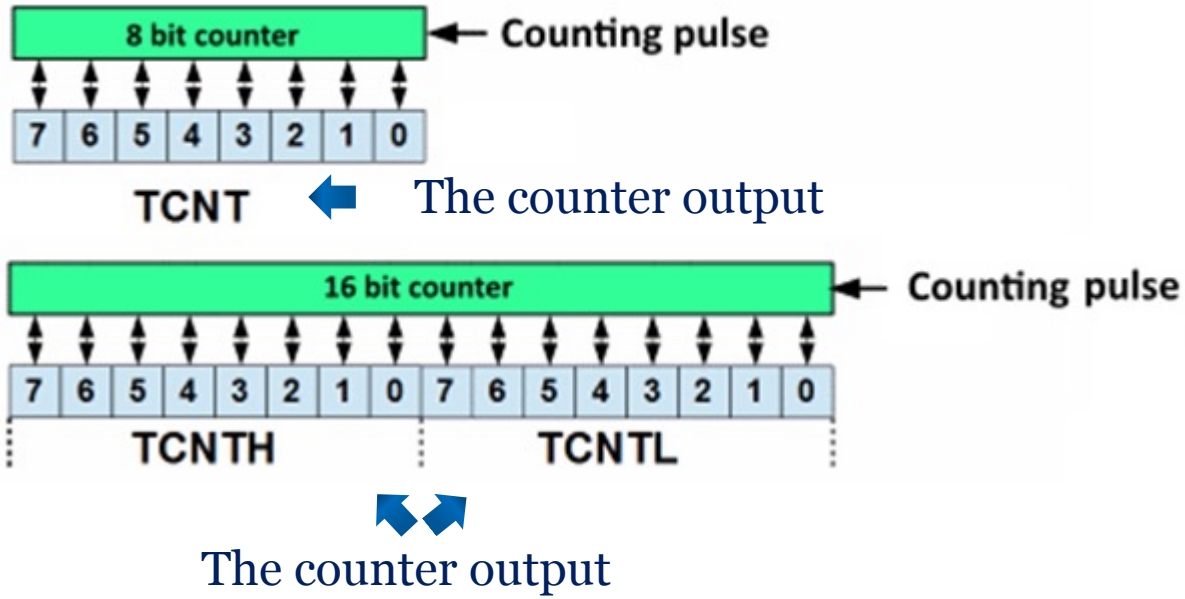
- T/C in the AVRs are either ***8*** or ***16*** bit wide.
- They count ***upward*** or ***downwards***.
- With ***8 bits*** they count from ***0 to 255***, with ***16 bits*** from 0 to ***65,535***.
- If they reach and exceed their upper limit, they ***restart again at zero (or decreases until zero)***.
- Their actual count state is available by reading the port ***TCNT (8 bits)*** or the ports ***TCNTH*** and ***TCNTL*** (16 bits)
- Those can be written (i.e., assign a value into the register), too, and the timer ***counts from*** this changed state.





T/C in the AVRs

- T/C in the AVRs are either ***8*** or ***16*** bit wide.
- They count ***upward*** or ***downwards***.
- With ***8 bits*** they count from ***0 to 255***, with ***16 bits*** from 0 to ***65,535***.
- If they reach and exceed their upper limit, they ***restart again at zero (or decreases until zero)***.
- Their actual count state is available by reading the port ***TCNT (8 bits)*** or the ports ***TCNTH*** and ***TCNTL*** (16 bits)
- Those can be written (i.e., assign a value into the register), too, and the timer ***counts from*** this changed state.



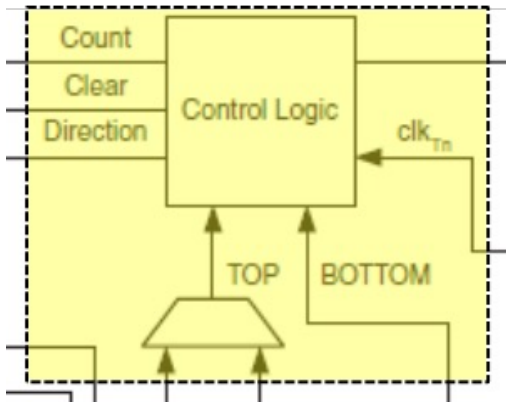
T/C on Arduino Uno and Mega

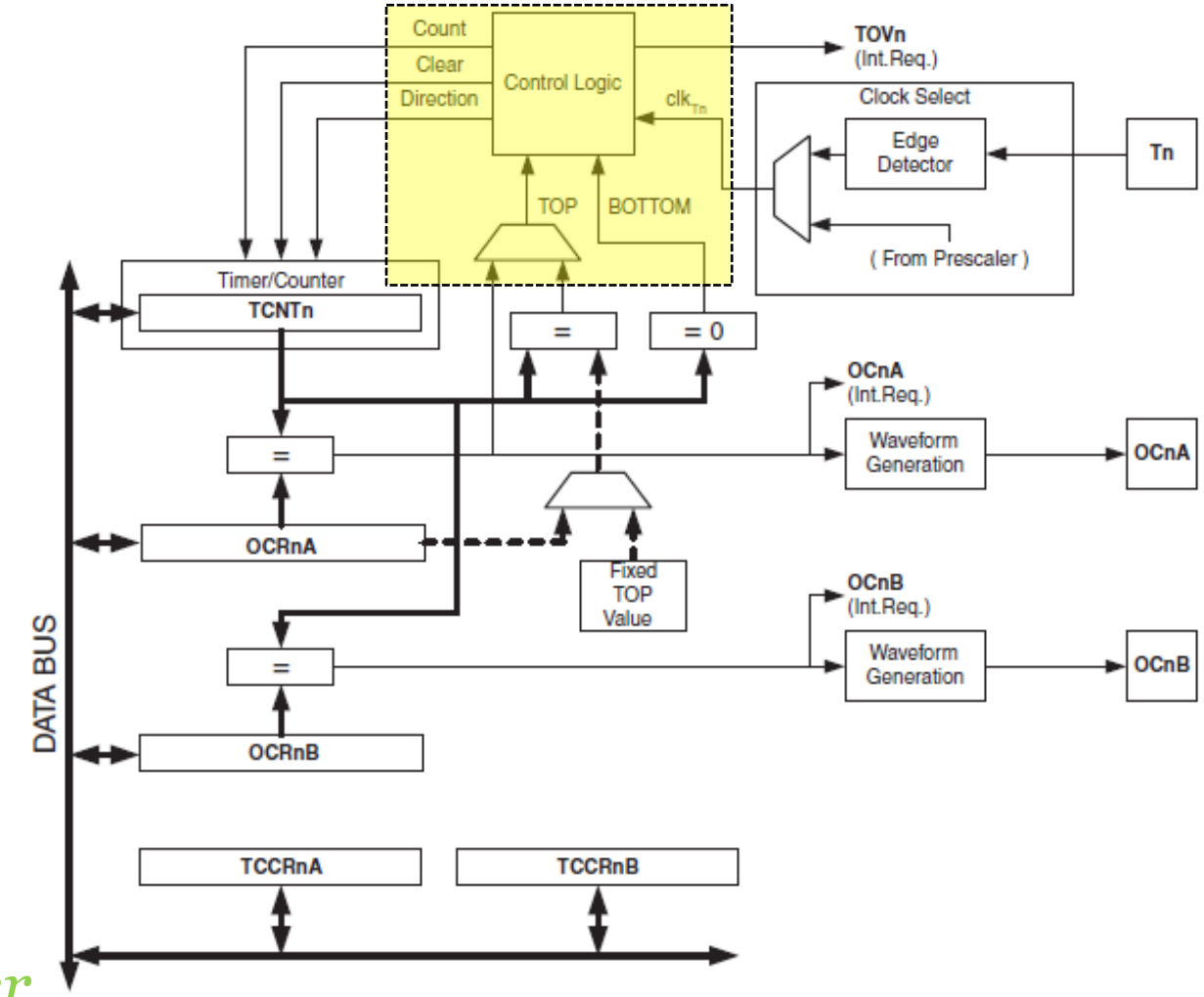
- On the Arduino **Uno** there are *three* Timer/Counters
  T/C0, T/C 1, and T/C2
- On Arduino **Mega** there are *six* Timers/Counters (0-5)
  T/C0, T/C1, T/C2, T/C3, T/C4, and T/C5
- T/C0 is 8-bit: Normally been used for the timer functions, like *delay()*, *millis()*, and *micros()*.
- T/C1 is 16-bit: Normally been used for *Servo Library* (this is done by T/C5 on Mega).
- T/C2 is 8-bit like T/C0 and the *tone()* function uses T/C2.
- T/C3, T/C4, and T/C5 are only available on Arduino Mega boards. These T/Cs are all 16-bit

T/C on Arduino Uno and Mega

- You can see the details of a typical timer on an Arduino in the figure on the right-hand side!

Let us have a detailed look!



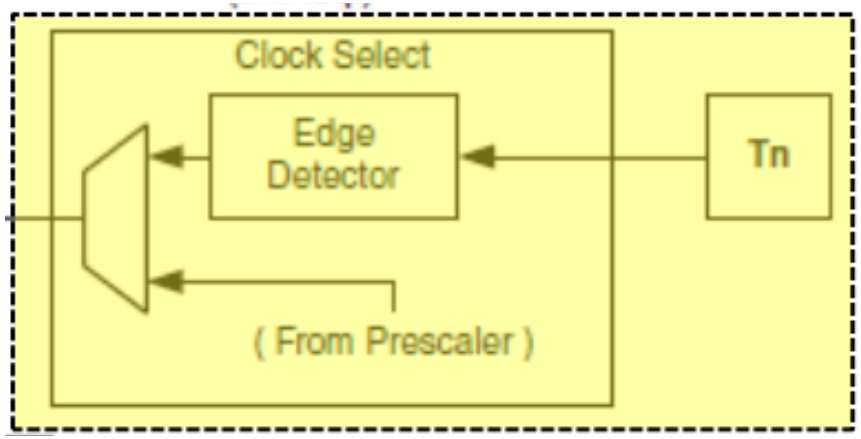


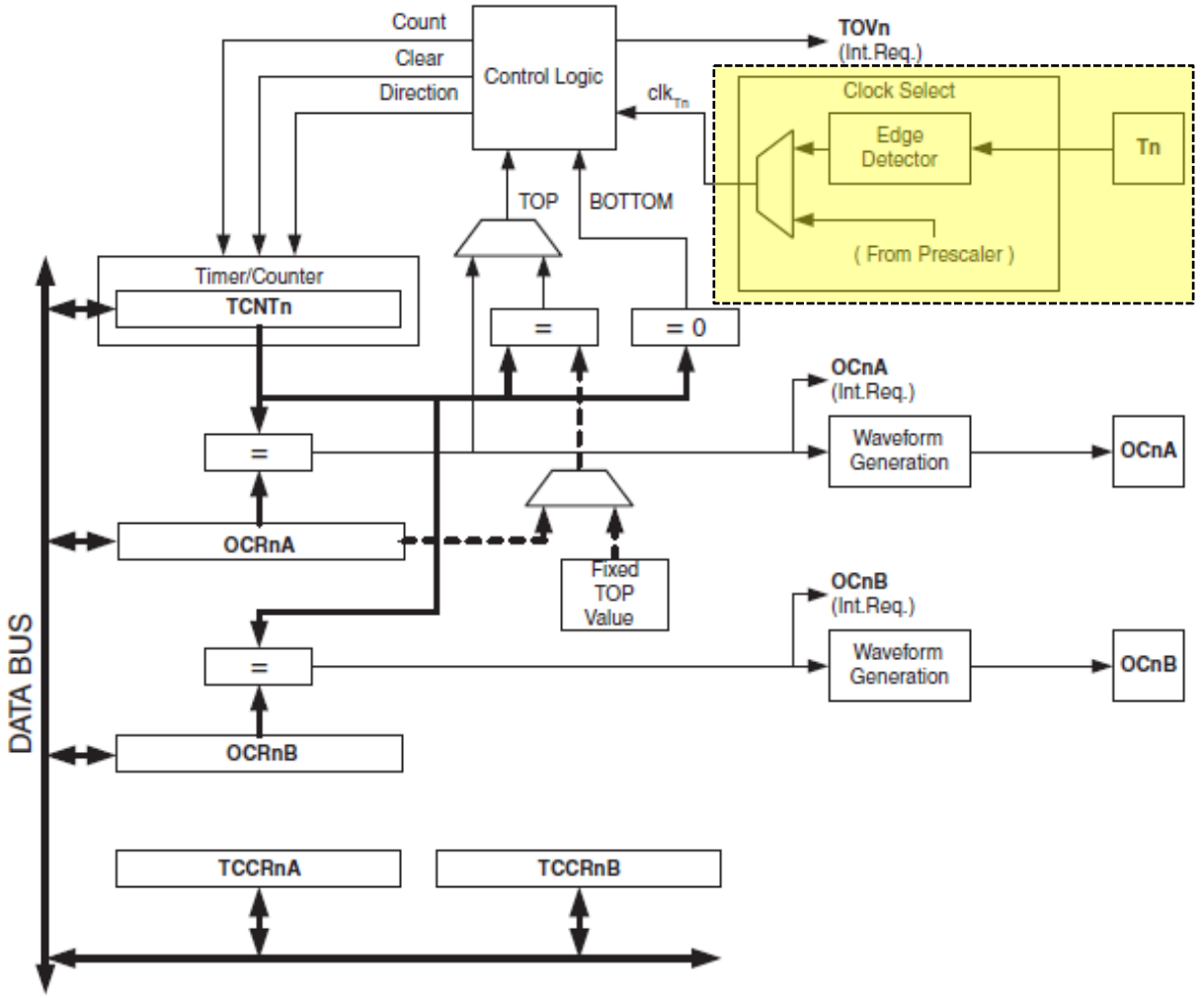
This is the T/C control logics (i.e., the combination of the *flip-flops*). Top and Bottom determine *the upper* and *the lower* limits of the counting process, respectively

T/C on Arduino Uno and Mega

- You can see the details of a typical timer on an Arduino in the figure on the right-hand side!

Let us have a detailed look!



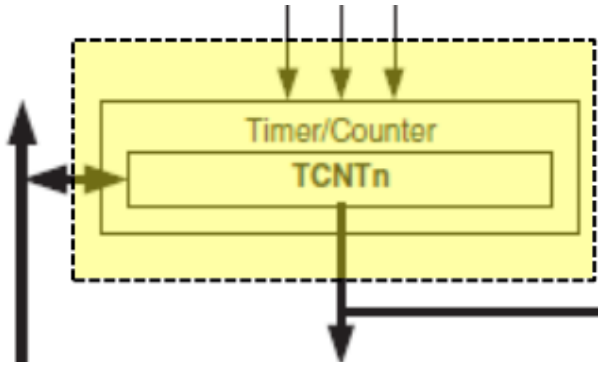


This part is to select the *clock source*: (1) from *external* source (e.g., encoder) or (2) *internal* source (prescaler)

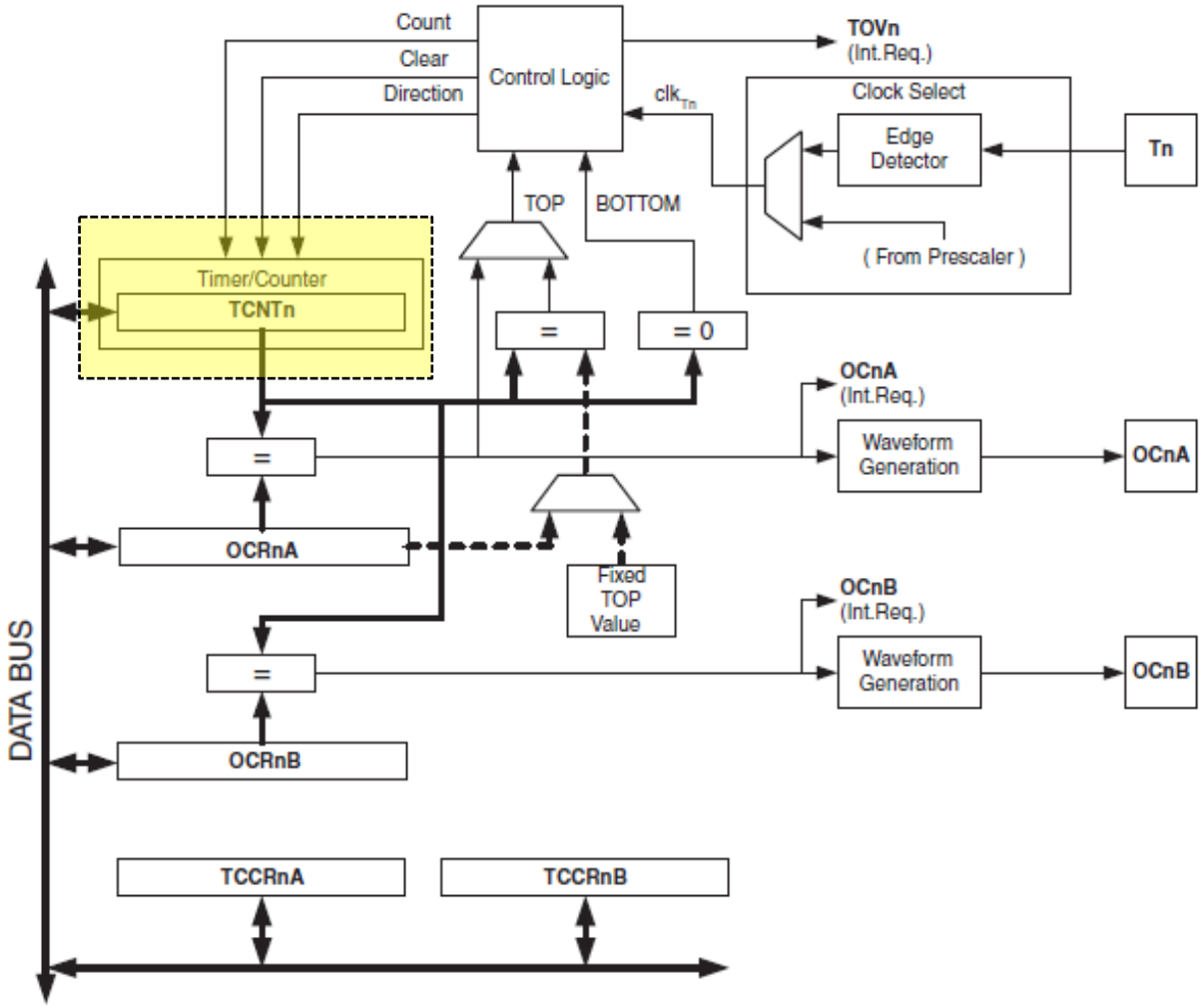
T/C on Arduino Uno and Mega

- You can see the details of a typical timer on an Arduino in the figure on the right-hand side!

Let us have a detailed look!



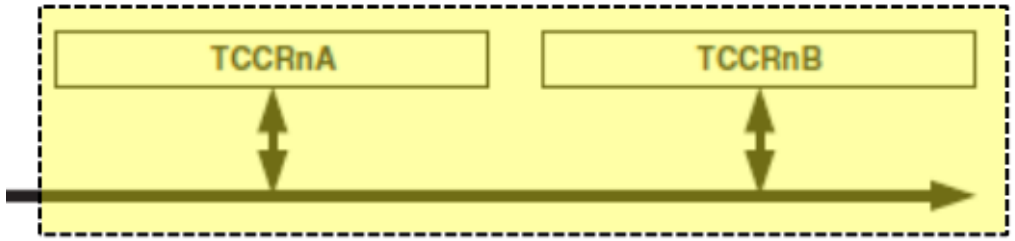
TCNTn: *T/C's Register*. The *actual* timer value is stored here, *n* refers to T/C number (e.g., TCNT2 stores the counts from T/C2)



T/C on Arduino Uno and Mega

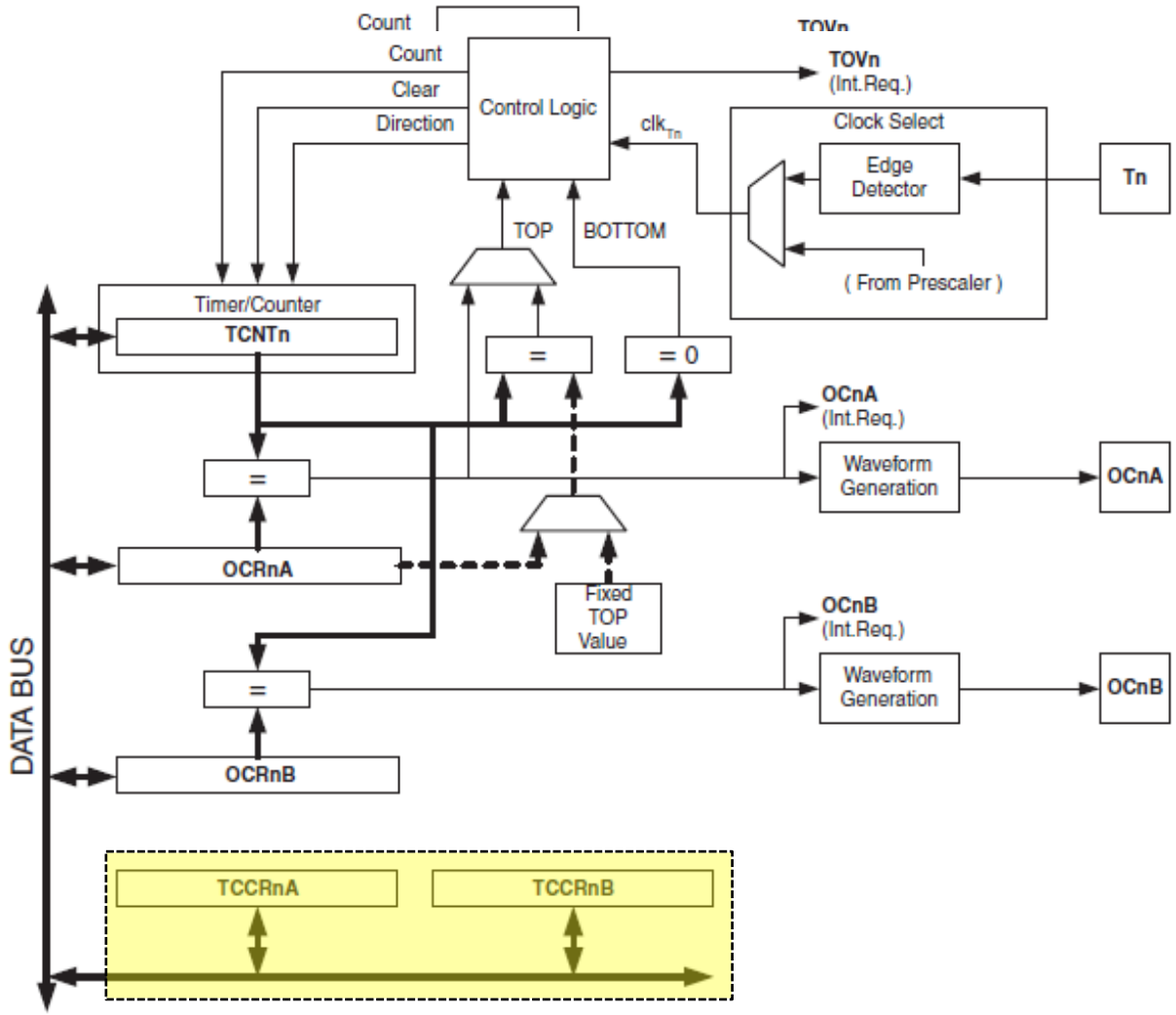
- You can see the details of a typical timer on an Arduino in the figure on the right-hand side!

Let us have a detailed look!



TCCRnA: Control Register A of T/C number n
TCCRnB: Control Register B of T/C number n
You can change the *T/C 's behaviour* via the T/C control register.

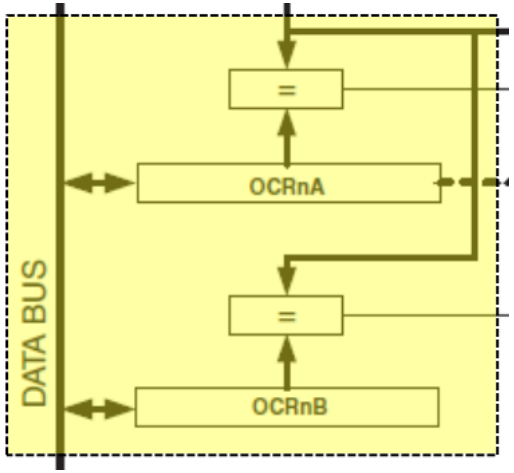
We will learn how to do this later ☺

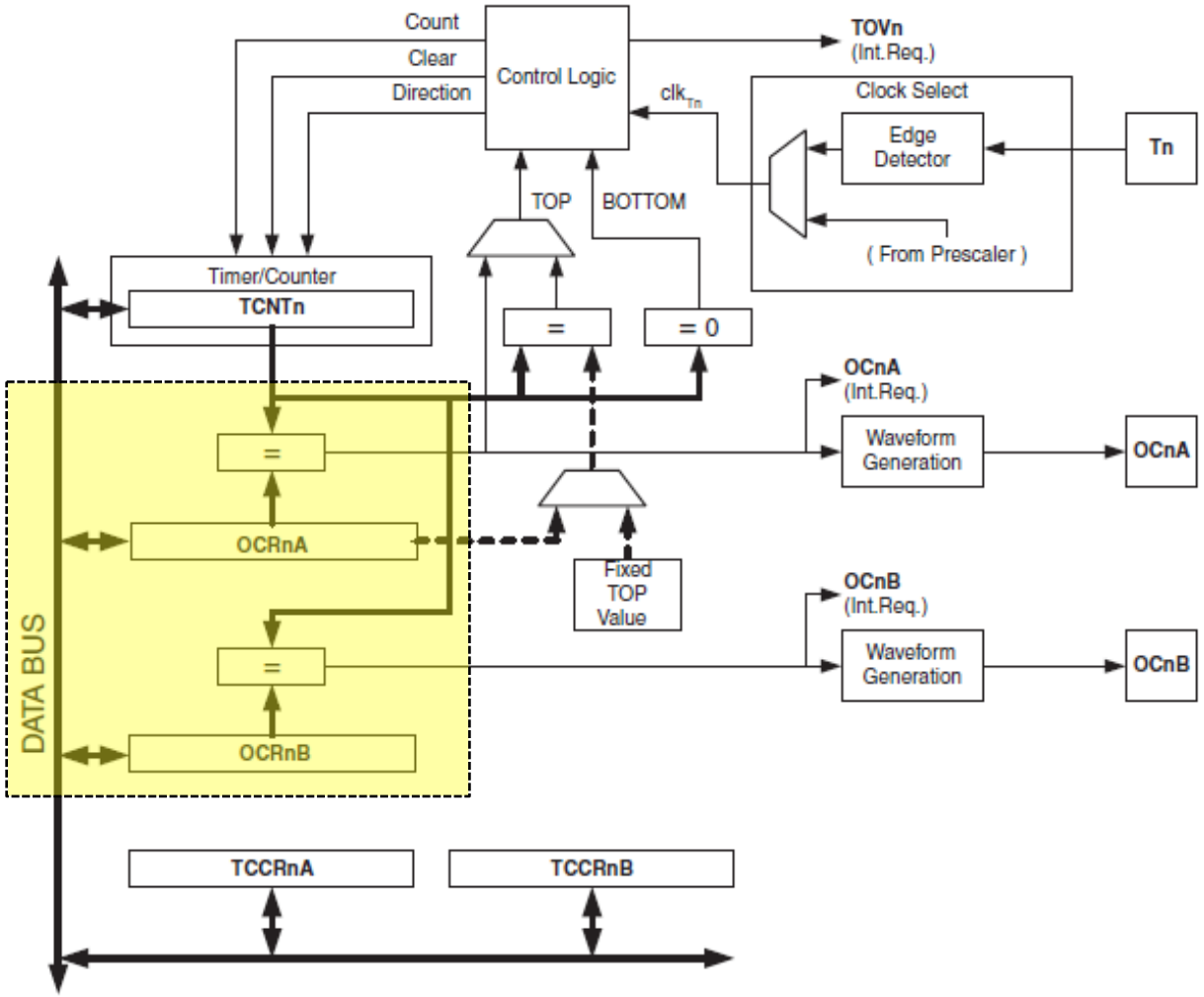


T/C on Arduino Uno and Mega

- You can see the details of a typical timer on an Arduino in the figure on the right-hand side!

Let us have a detailed look!



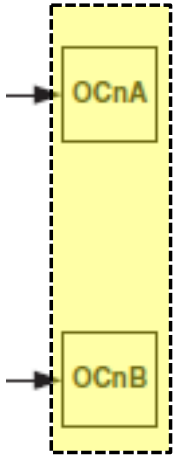


This part includes *Output Compare Registers* A and B of T/C number *n*. We can set some values in these registers to control the T/C's behaviour.

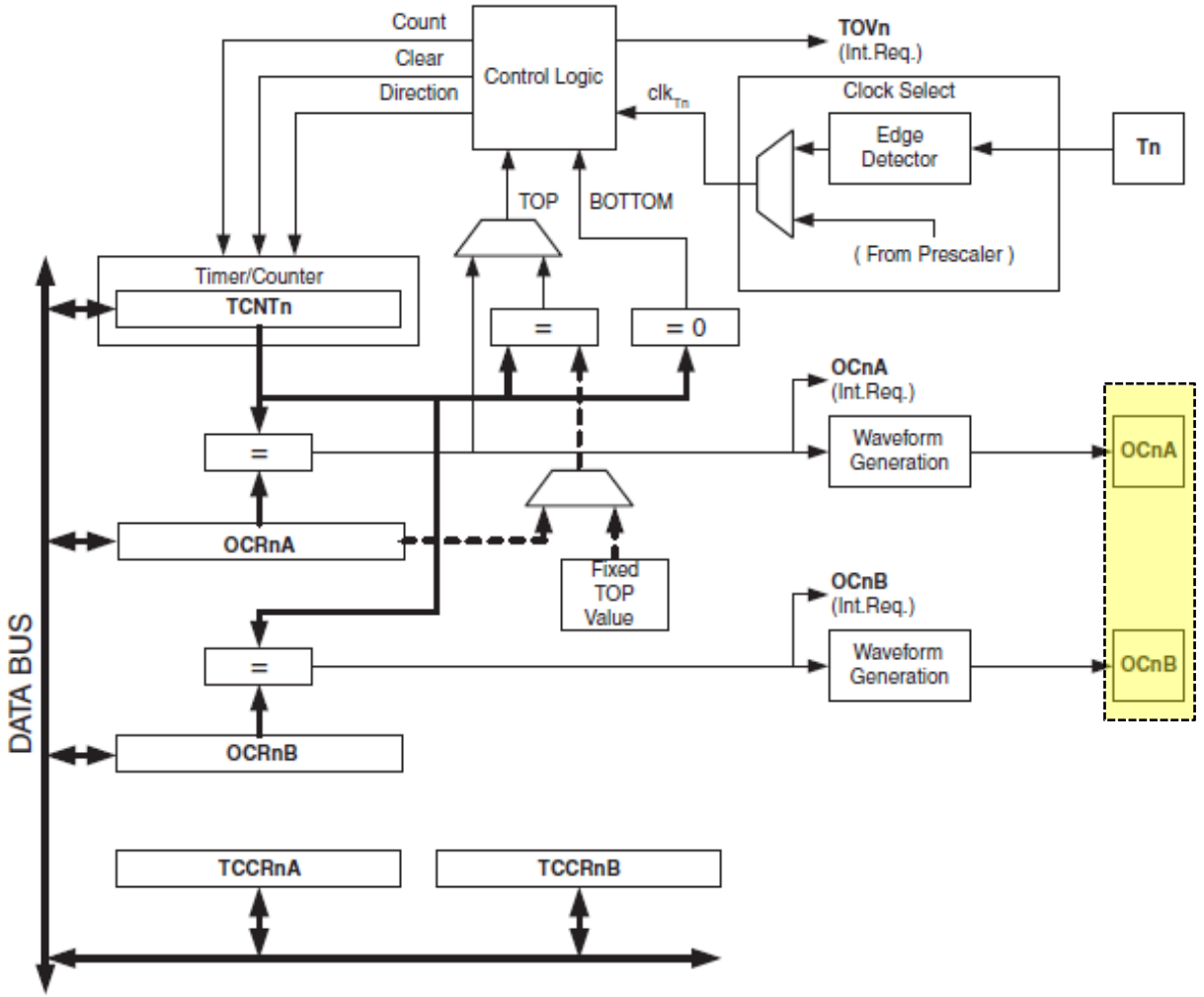
T/C on Arduino Uno and Mega

- You can see the details of a typical timer on an Arduino in the figure on the right-hand side!

Let us have a detailed look!



The status of these pins can be changed when the value in counting register *match* the value in OCRnA/B. This can be used to generate frequency.

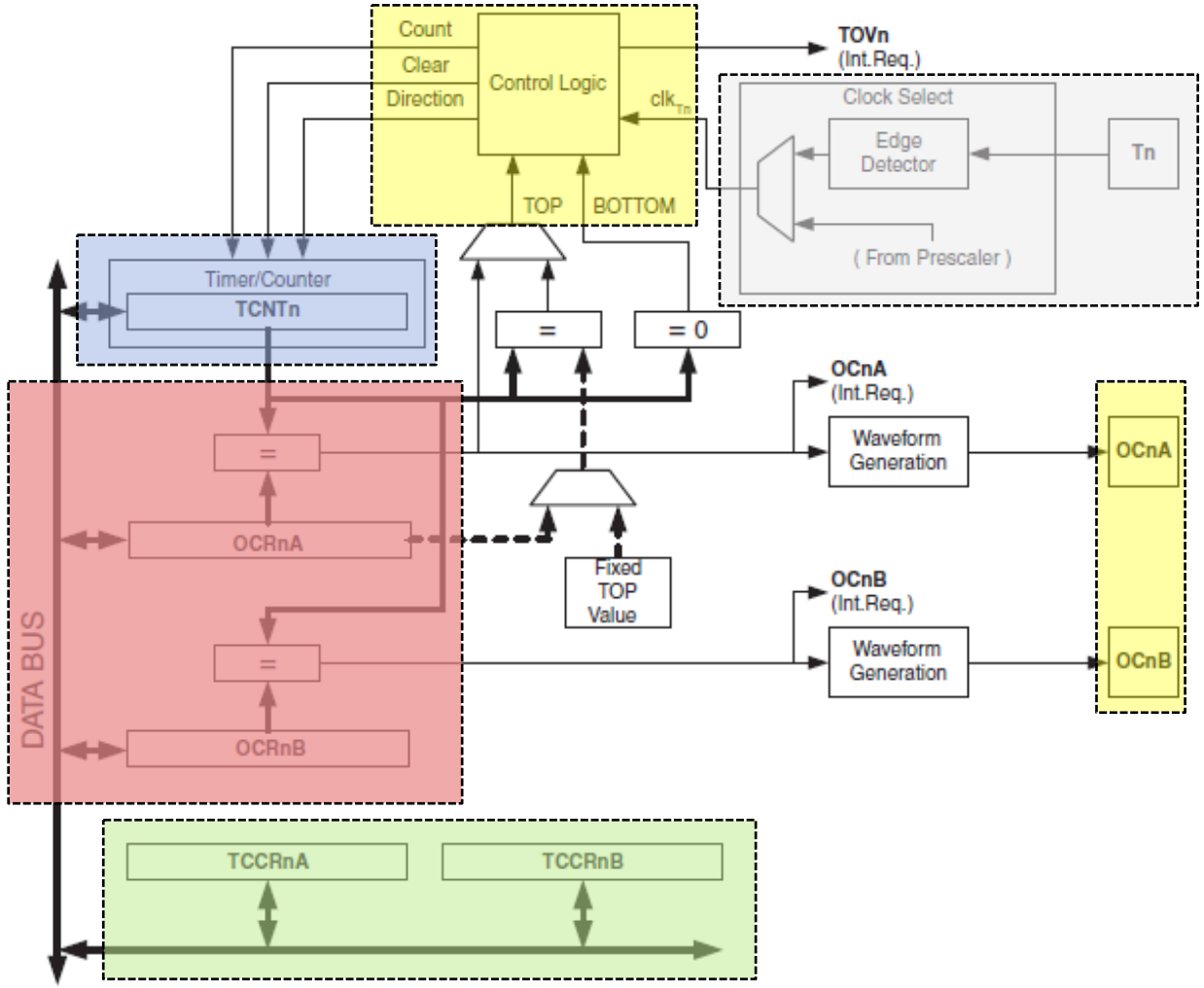


8-bit T/C on Arduino

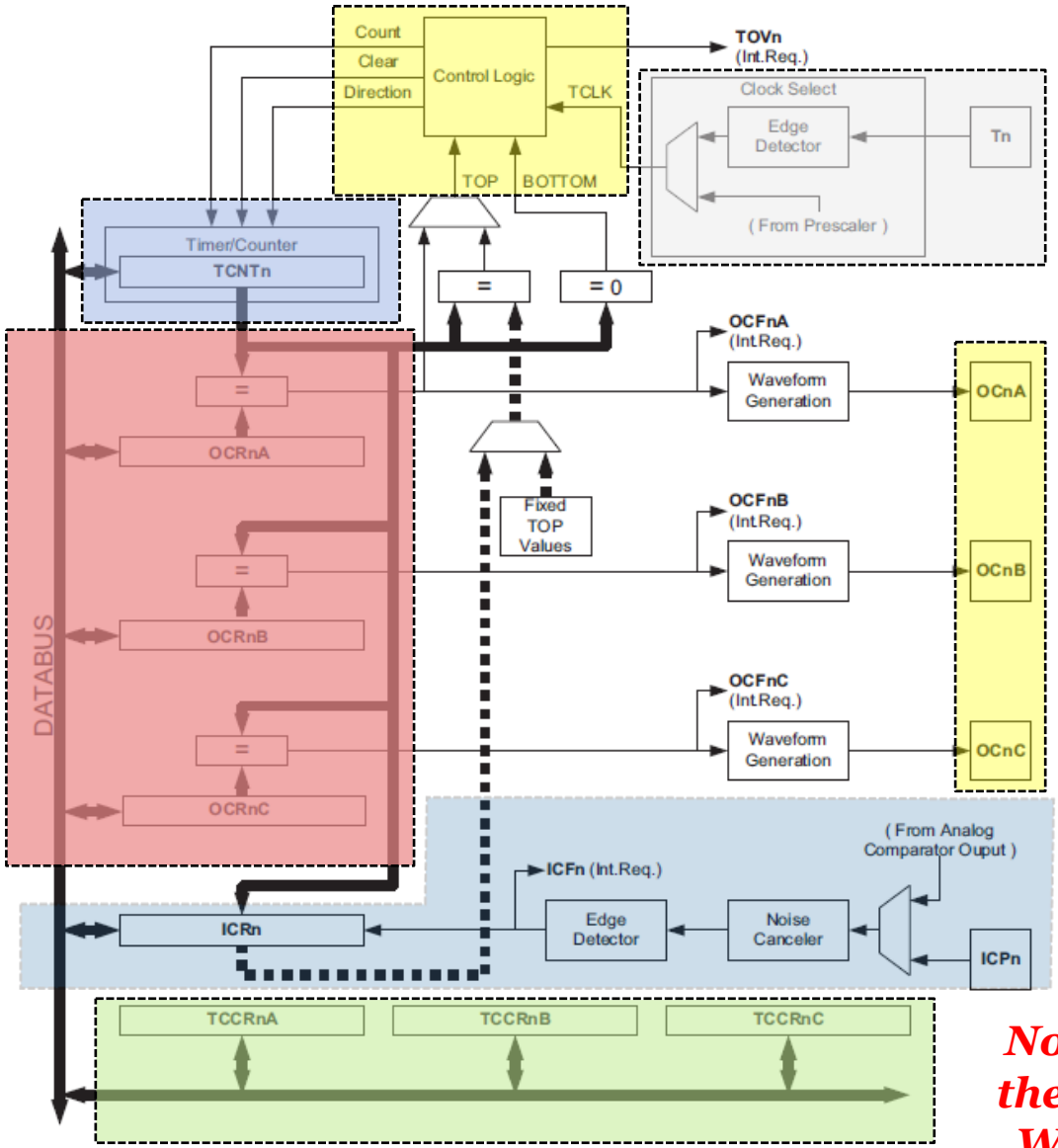
These are the main components of any 8-bit T/C on an Arduino!

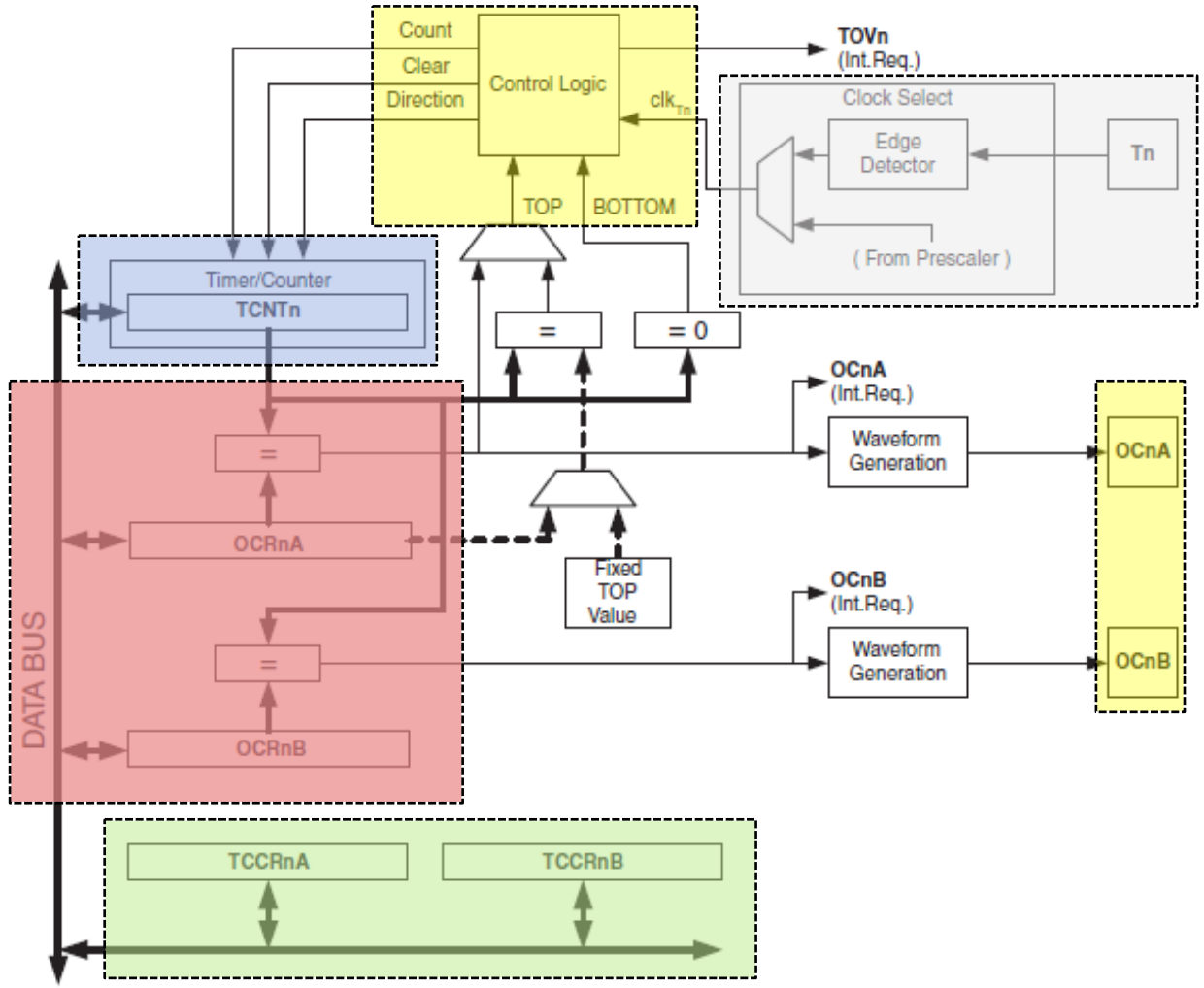
Now the question is:

What are the differences between 8-bit T/C and 16-bit T/C?!



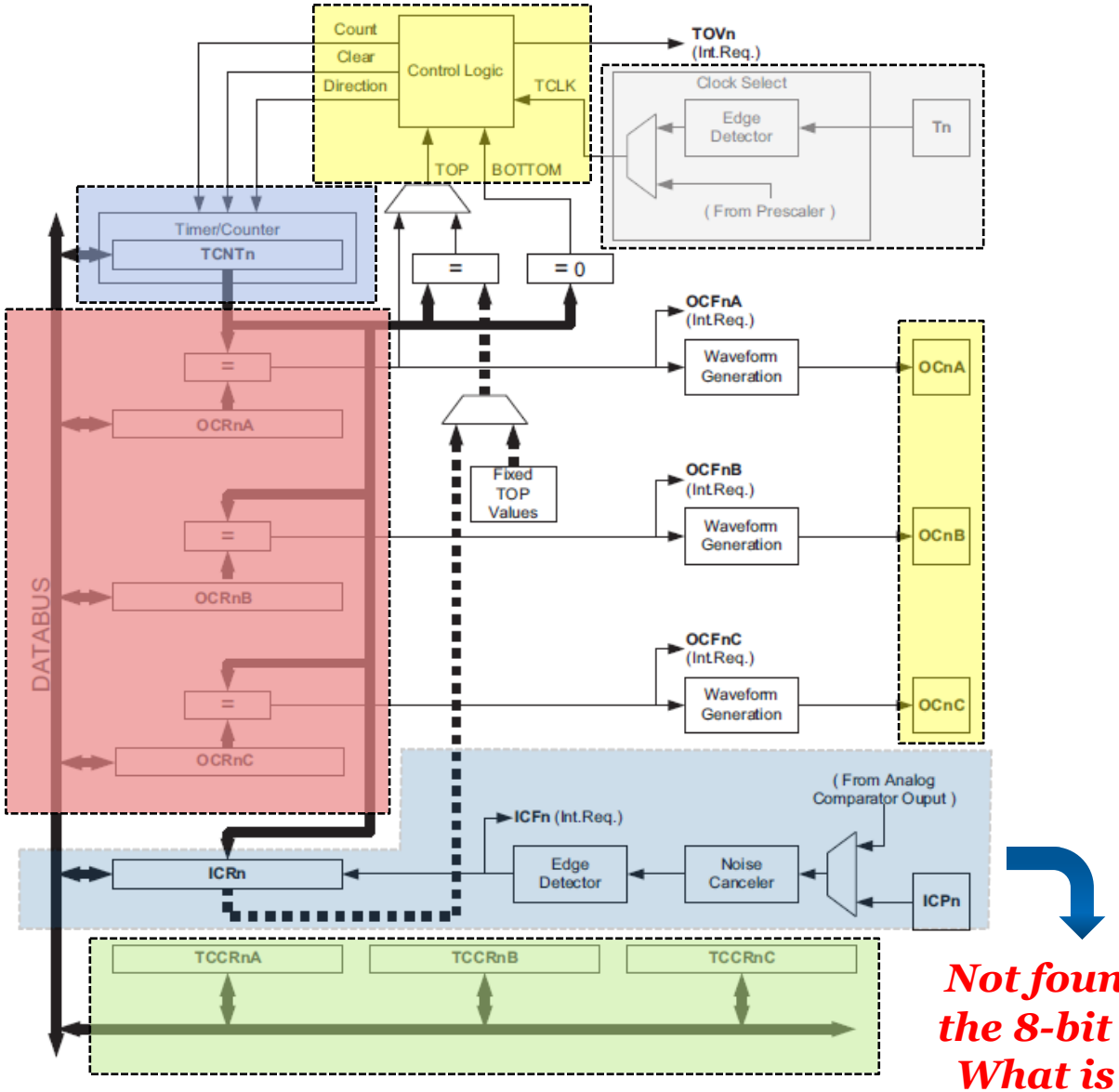
8-bit vs 16-bit T/C





Not found in the 8-bit T/C! What is it?!

8-bit vs 16-bit T/C



The 16-bit T/C incorporates an *input capture* unit that can capture external events and give them a time-stamp indicating time of occurrence.

ICPn: Input Capture *Pin* of T/C number n
ICRn: Input Capture *Register* of T/C number n

Not found in the 8-bit T/C! What is it?!

University of Nottingham
UK | CHINA | MALAYSIA

T/C Modes of Operation

T/C Modes of operations

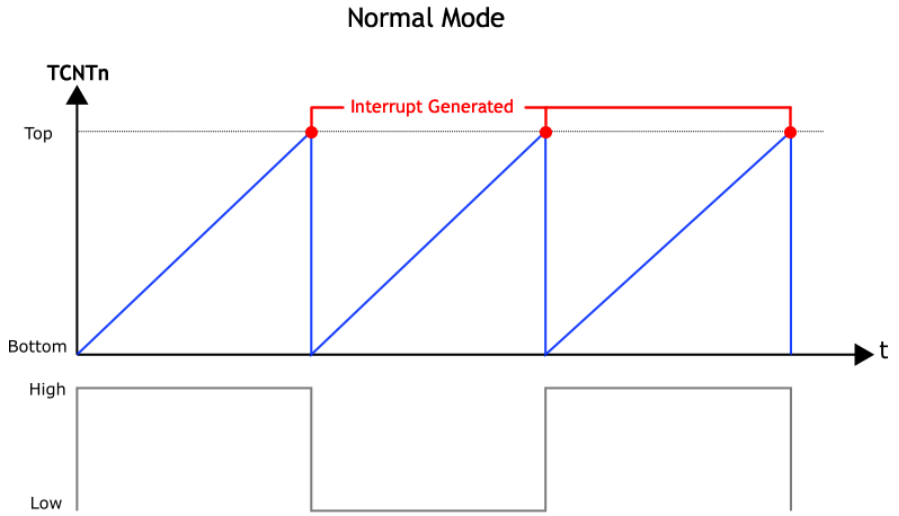
The T/C can be operated in different modes according to the *control registers settings*. The common modes of operations are as follows:

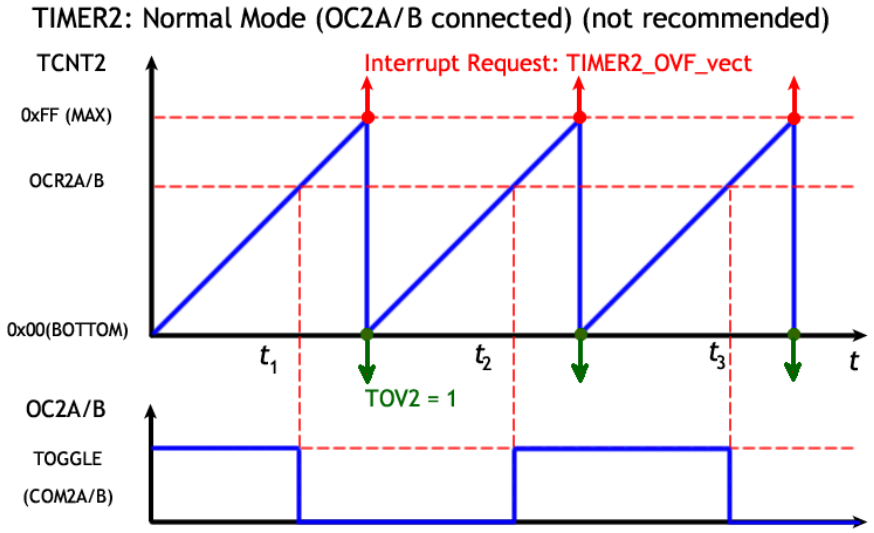
- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode
- Phase and Frequency Correct PWM Mode

In this module, we will consider only the first *three* modes of operation!

1. Normal Mode

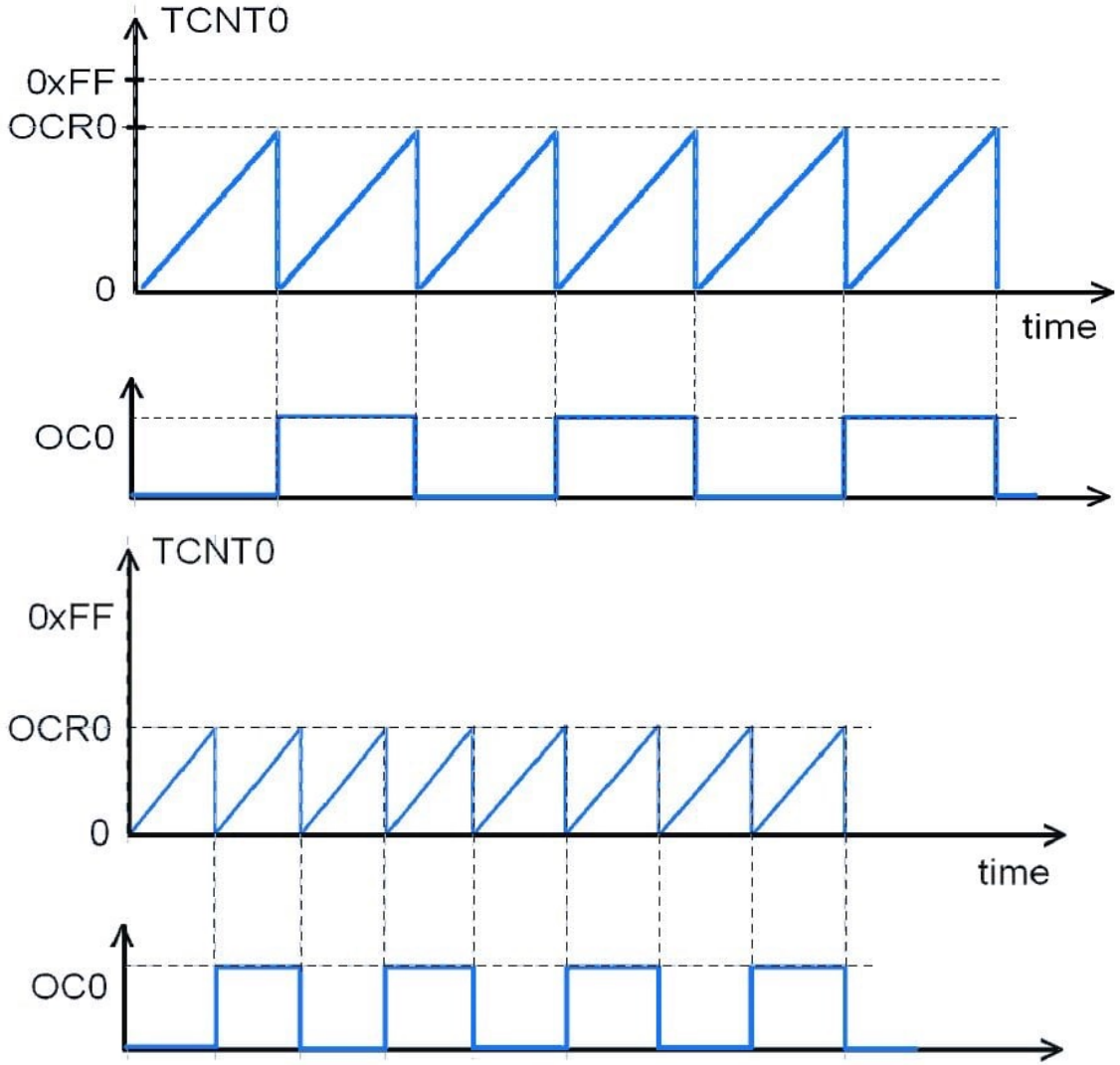
- In this mode the counting direction is always up (*incrementing*), and no counter clear is performed.
- The counter simply overruns when it passes its maximum 8/16-bit value (MAX = 0xFF/0xFFFF) and then restarts from the BOTTOM (0x00/0x0000).
- The Output Compare units can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode *is not recommended*, since this will occupy too much of the CPU time.
- The Input Capture unit is easy to use in Normal mode.





2. Clear Timer on Compare Match (CTC) Mode

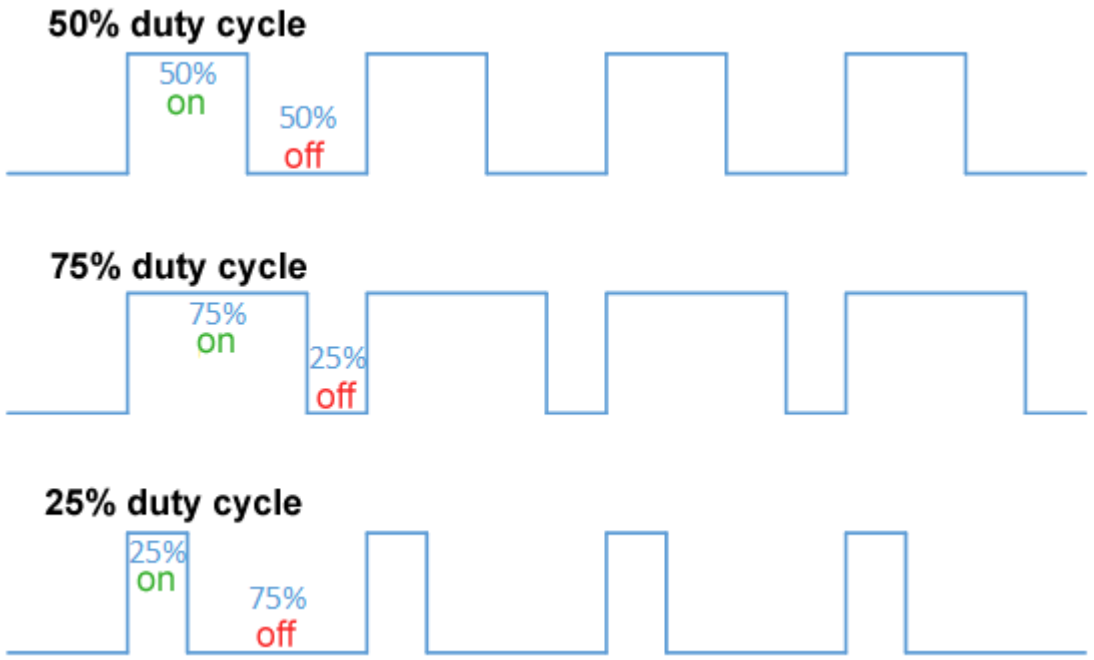
- In CTC mode the counter *is cleared to zero* when the counter value (TCNTn) matches either the OCRnA or the ICRn (*later we see this*).
- The OCRnA or ICRn define the top value for the counter, hence also its resolution.
- This mode allows greater control of the compare match output frequency.
- It also simplifies the operation of counting external events.

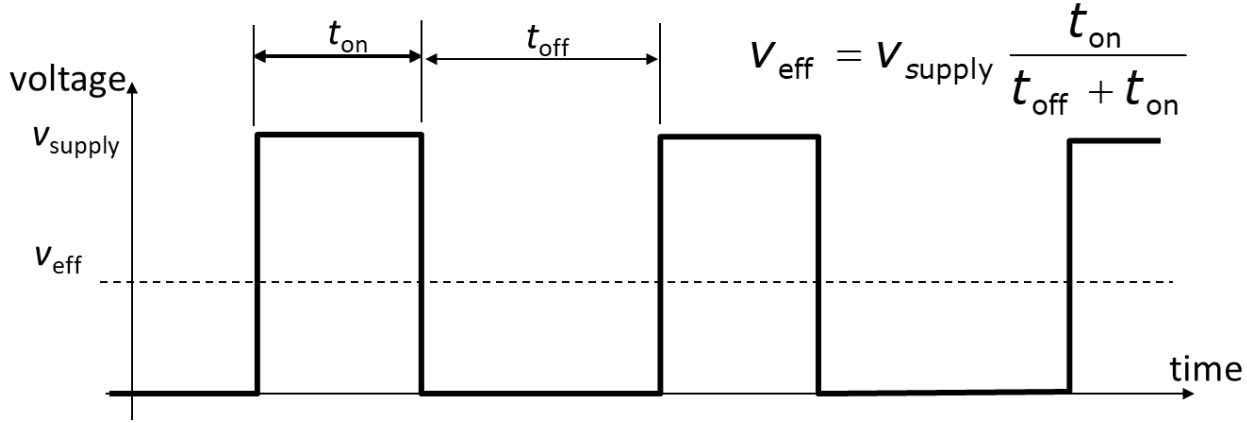


3. Fast Pulse Width Modulation (Fast PWM) Mode

Before we learn about the Fast PWM mode, what is a PWM?!

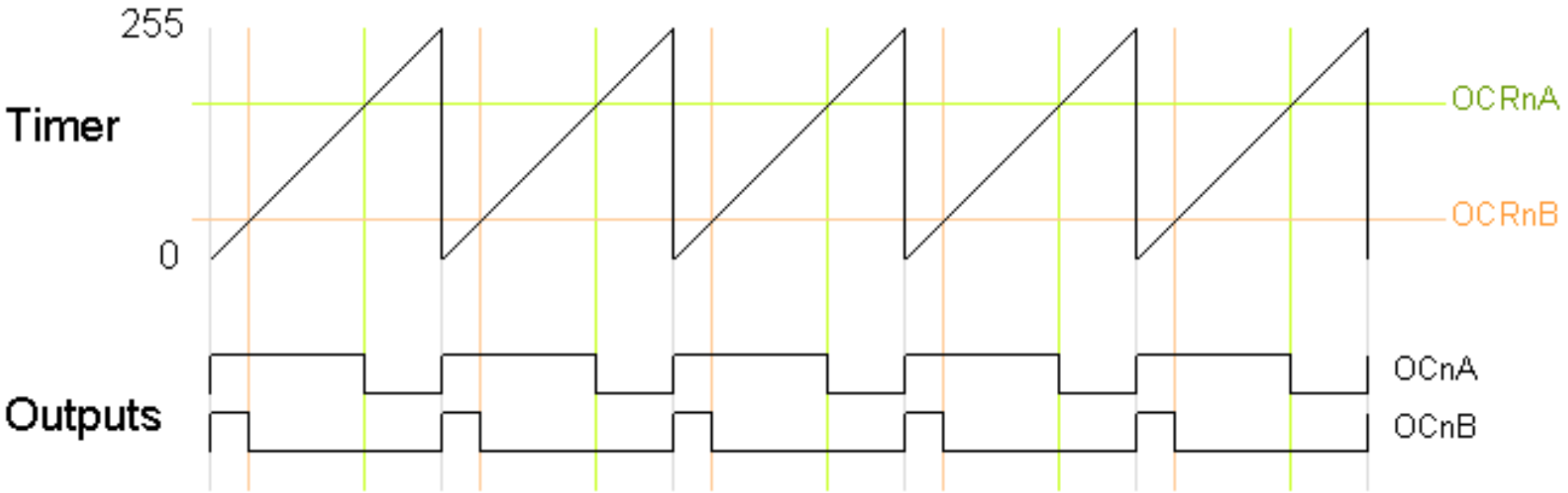
- **Pulse-width modulation** (**PWM**) is a method of controlling the average power delivered by an electrical signal.
- The average value of voltage (and current) fed to the load is controlled by *switching* the supply between 0 and 100% at a rate *faster* than it takes the load to change significantly.
- The term ***duty cycle*** describes the proportion of 'on' time to the regular interval or 'period' of time
- Duty cycle is expressed *in percent*, 100% being fully on.
- When a digital signal is on half of the time and off the other half of the time, the digital signal has a duty cycle of 50% and resembles a "square" wave.
- When a digital signal spends more time in the on state than the off state, it has a duty cycle of >50%.
- When a digital signal spends more time in the off state than the on state, it has a duty cycle of <50%.





3. Fast Pulse Width Modulation (Fast PWM) Mode

- In the simplest PWM mode, the timer repeatedly counts from 0 to its maximum capacity (e.g., 255 for 8-bit T/C).
- The output turns on when the timer is at 0 and turns off when the timer matches the output compare register (OCRnA or OCRnB).
- The higher the value in the output compare register, the higher the duty cycle. This mode is known as Fast PWM Mode.
- The following diagram shows the outputs for two values of OCRnA (high value) and OCRnB (low value). Note that both outputs have the same frequency, matching the frequency of a complete timer cycle.



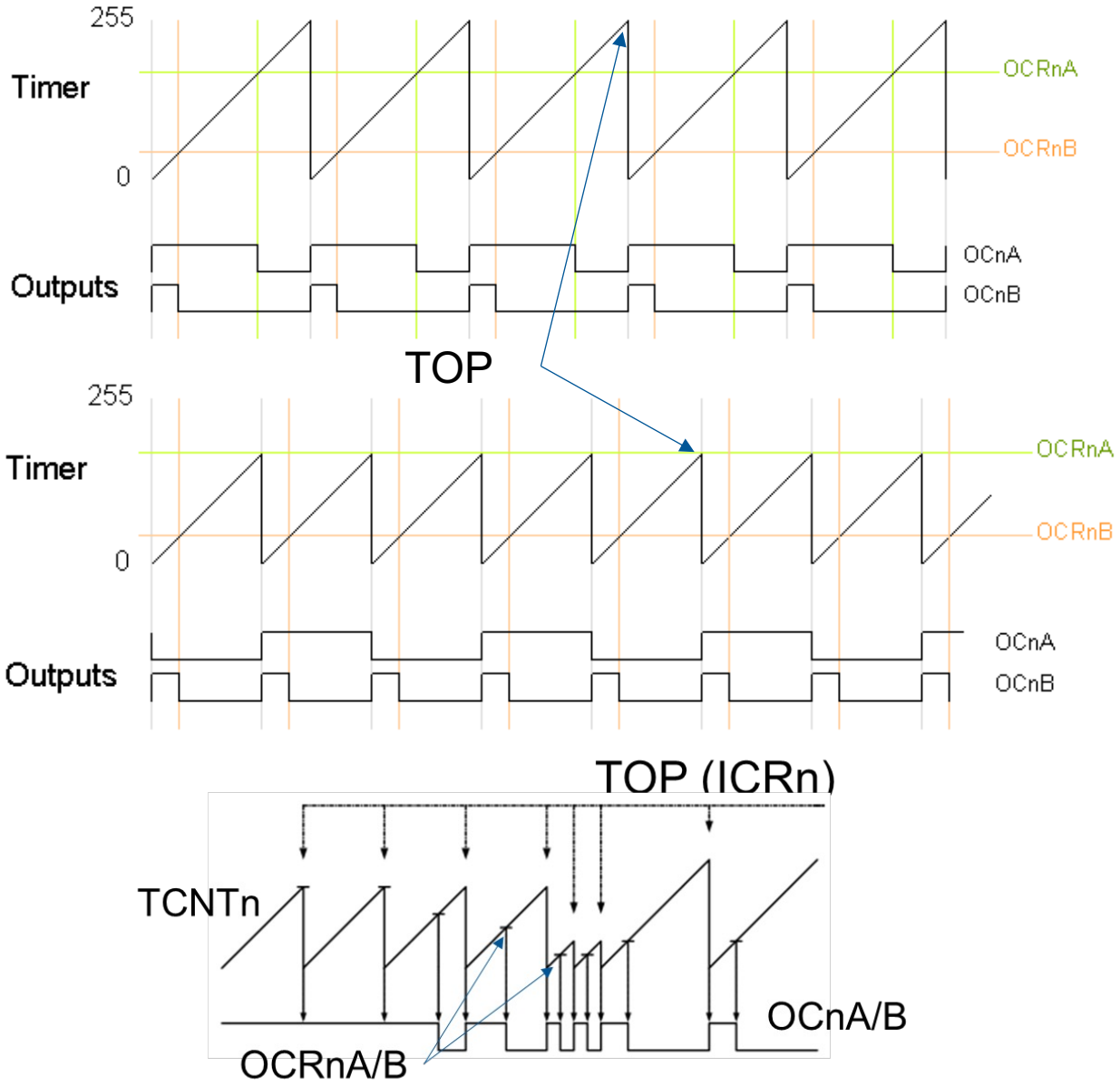
3. Fast Pulse Width Modulation (Fast PWM) Mode

Can we make it even faster?!
Yes! By varying the timer top limit: fast PWM

- In this mode, the timer counts from 0 to OCRnA (the value of output compare register A), rather than from 0 to its maximum capacity (e.g., 255).
- This gives much more control over the output frequency than the previous modes.
- Note that in this mode, only output B can be used for PWM; OCRnA cannot be used both as the **TOP** value and the PWM compare value.

Can we make it even faster and use OCnA output as well?!
Yes! By varying the TOP value using ICRn

- TCNTn counts (up) until matches value in ICRn ("TOP"), resets to 0, restarts.
- This sets output high at each cycle start.



University of Nottingham
UK | CHINA | MALAYSIA

T/C Modes of Operation Settings

T/C Control Registers Settings - TCCRnA

Bit	7	6	5	4	3	2	1	0
(0x120)	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

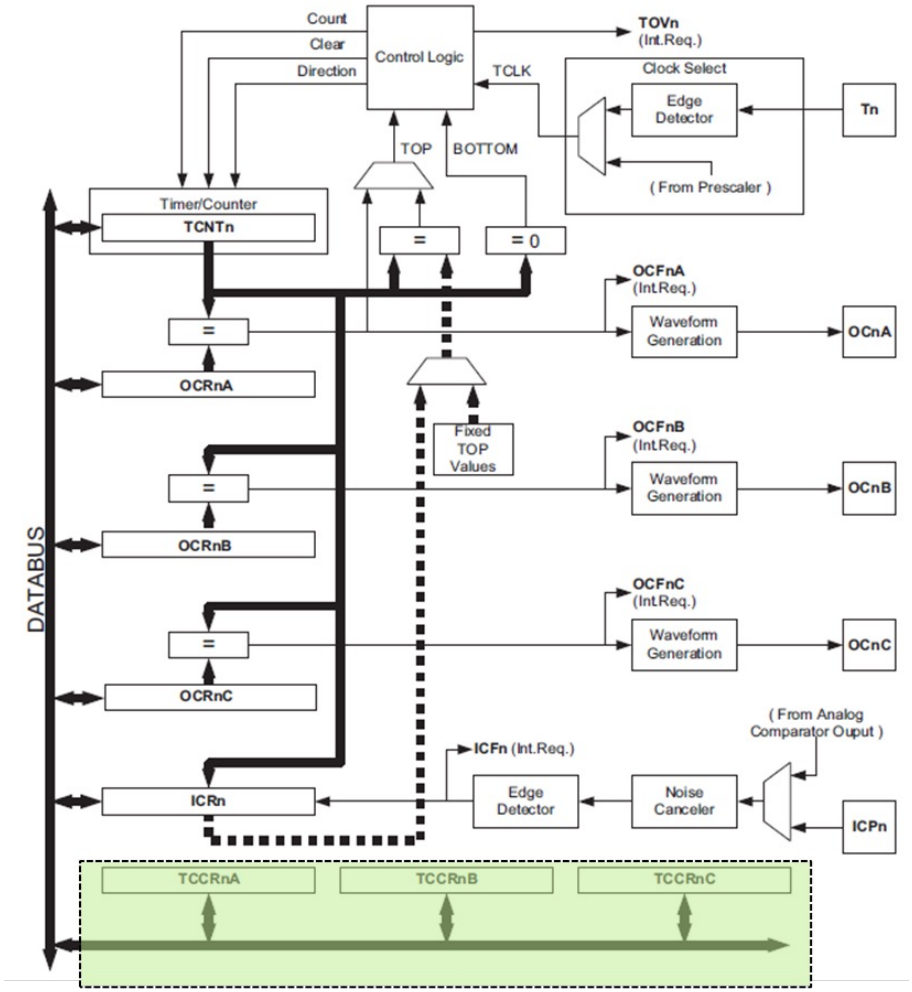
Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

Table 17-4. Compare Output Mode, Fast PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode)

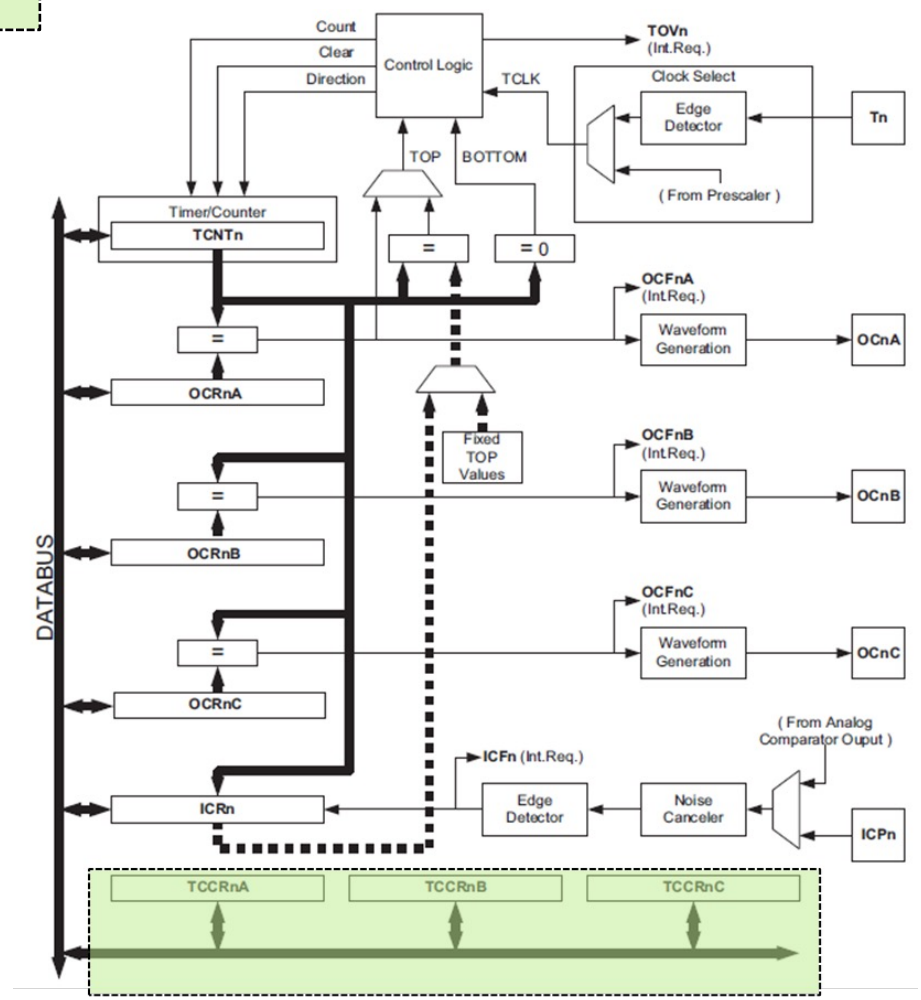
Example: *n* = 5



T/C Control Registers Settings - TCCRnA

Bit	7	6	5	4	3	2	1	0
(0x120)	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Example: *n* = 5



- **WGMn1:0** This is a **Waveform Generation Mode** bits.
- Combined with the *WGMn3:2* bits found in the TCCRnB, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used.

T/C Control Registers Settings – TCCRnA and TCCRnB

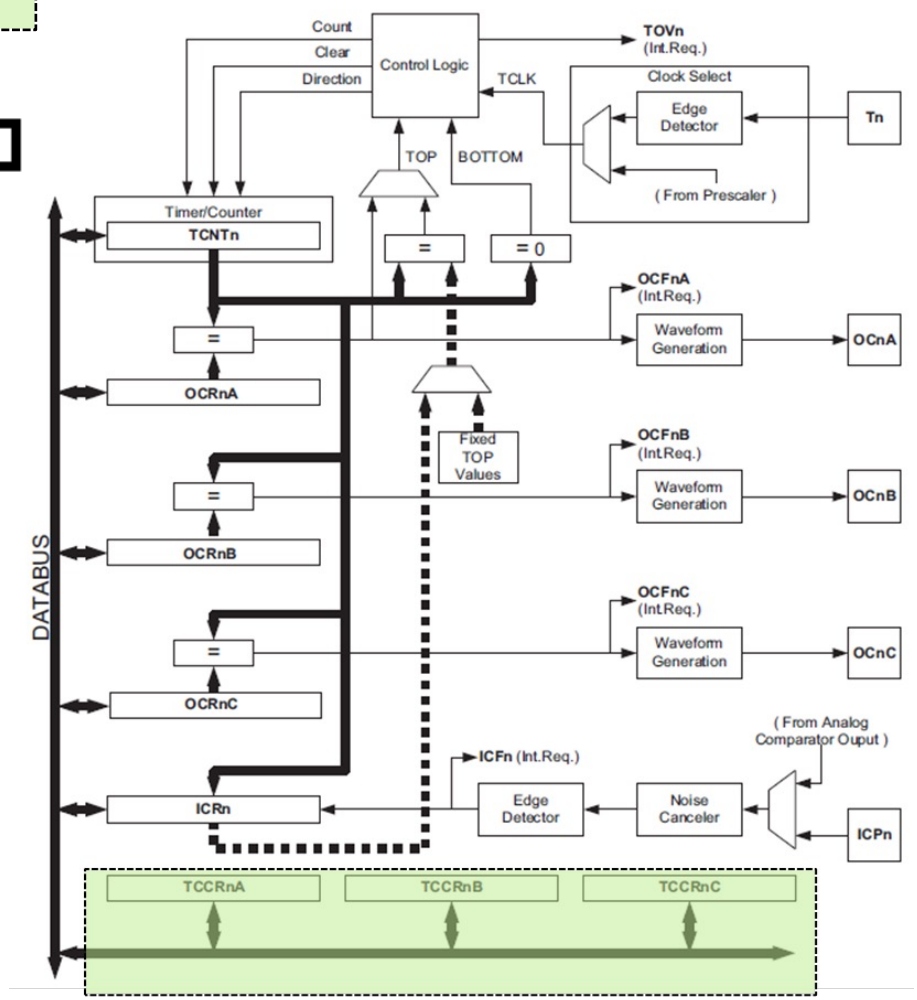
Bit	7	6	5	4	3	2	1	0
(0x120)	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Table 17-2. Waveform Generation Mode Bit Description

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnX at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Example: *n* = 5



T/C Control Registers Settings - TCCRnB

Bit	7	6	5	4	3	2	1	0
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Example: *n* = 5

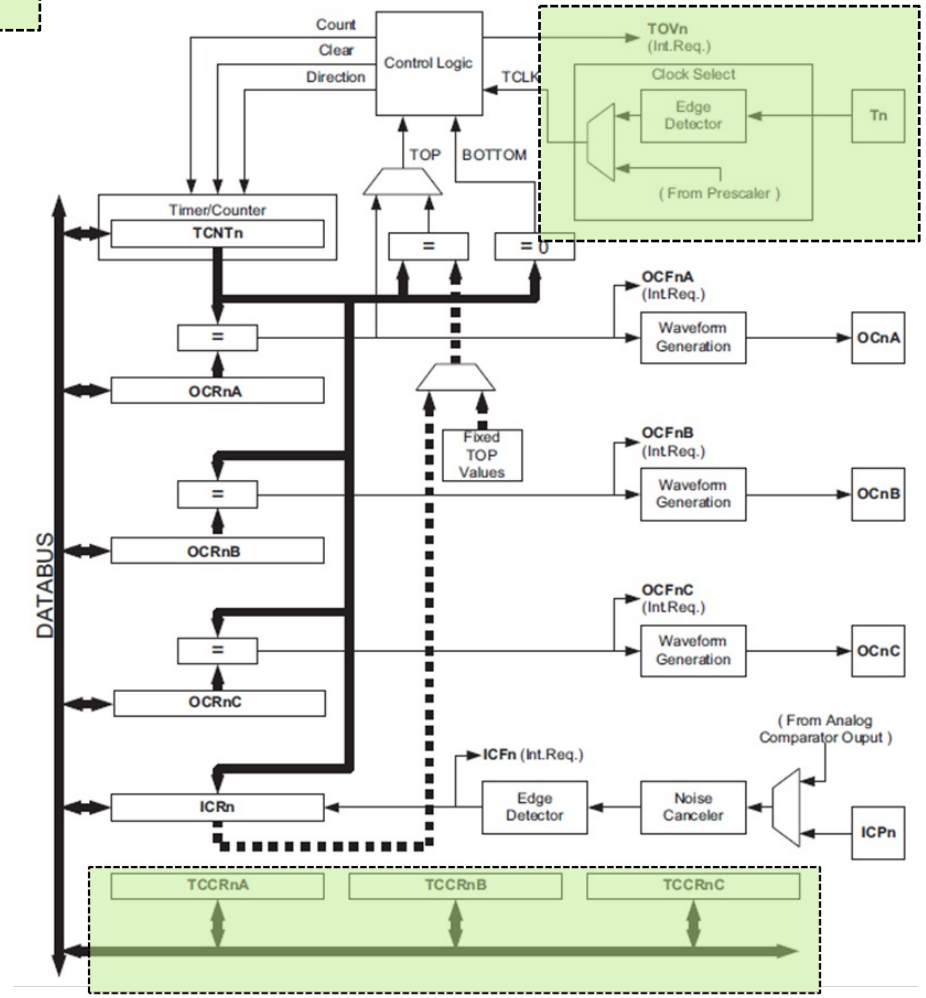


Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	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	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

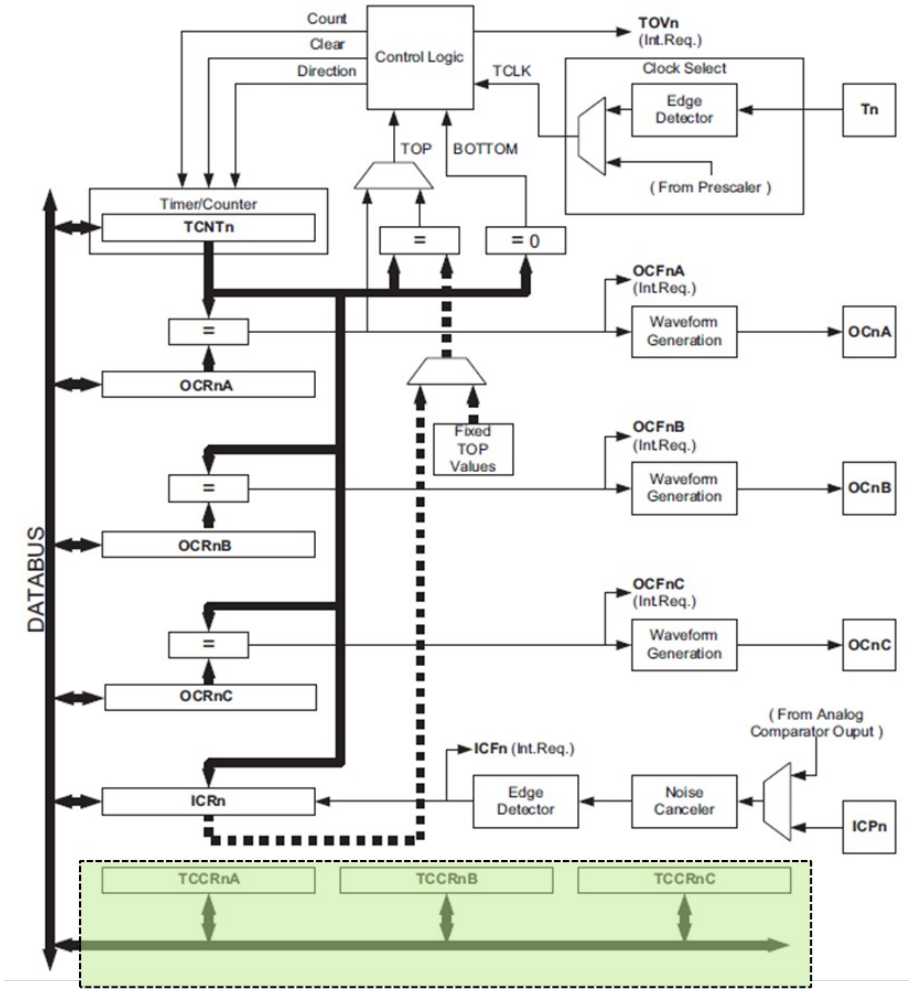
T/C Control Registers Settings - TCCRnB

Bit	7	6	5	4	3	2	1	0
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

ICNCn: Input Capture Noise Canceler
ICESn: Input Capture Edge Select
- : Reserved

We will not use these three bits in the module!

Example: *n* = 5



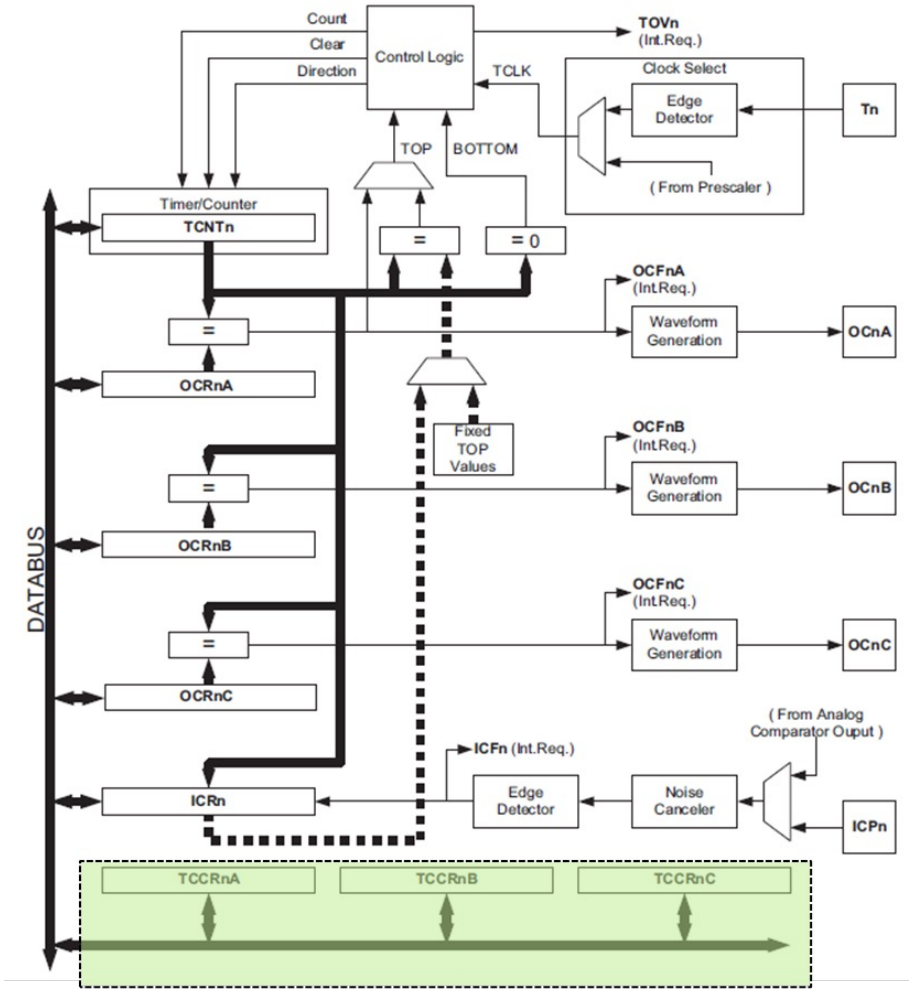
T/C Control Registers Settings - TCCRnC

Bit	7	6	5	4	3	2	1	0
(0x122)	FOC5A	FOC5B	FOC3C	–	–	–	–	–
Read/Write	W	W	W	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

FOCnA: Force Output Compare for Channel A
FOCnB: Force Output Compare for Channel B
FOCnC: Force Output Compare for Channel C
- : Reserved

We will not use these three bits in the module!

Example: *n* = 5

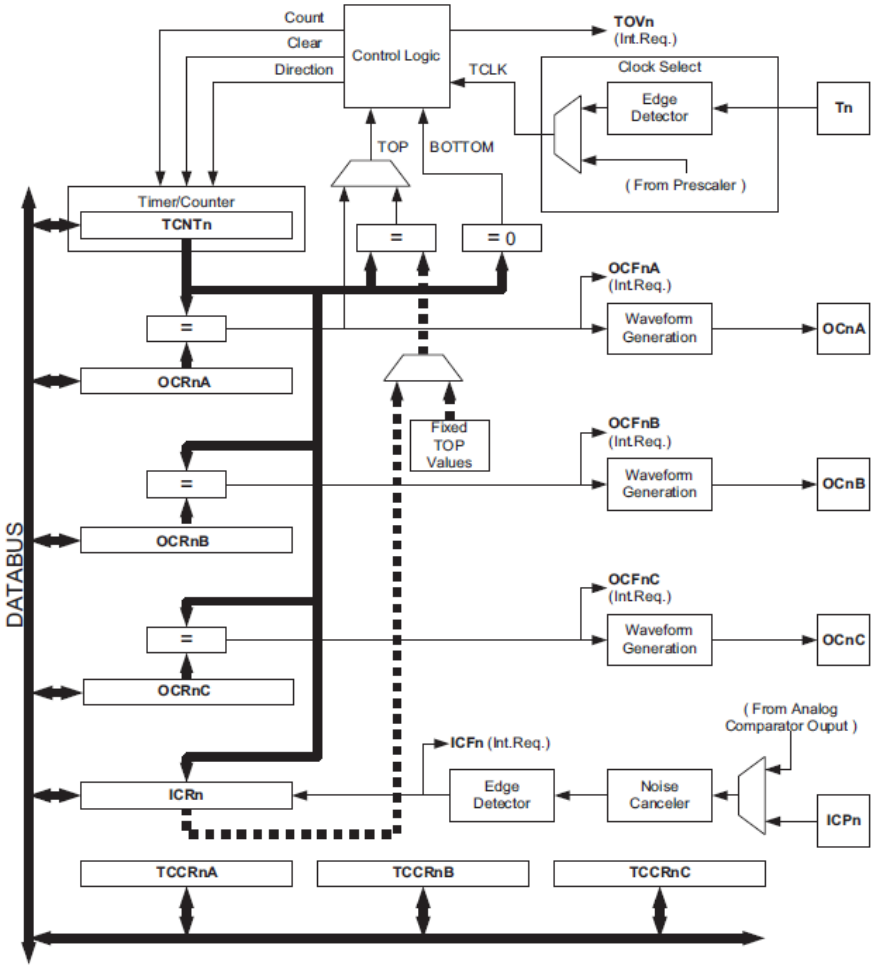


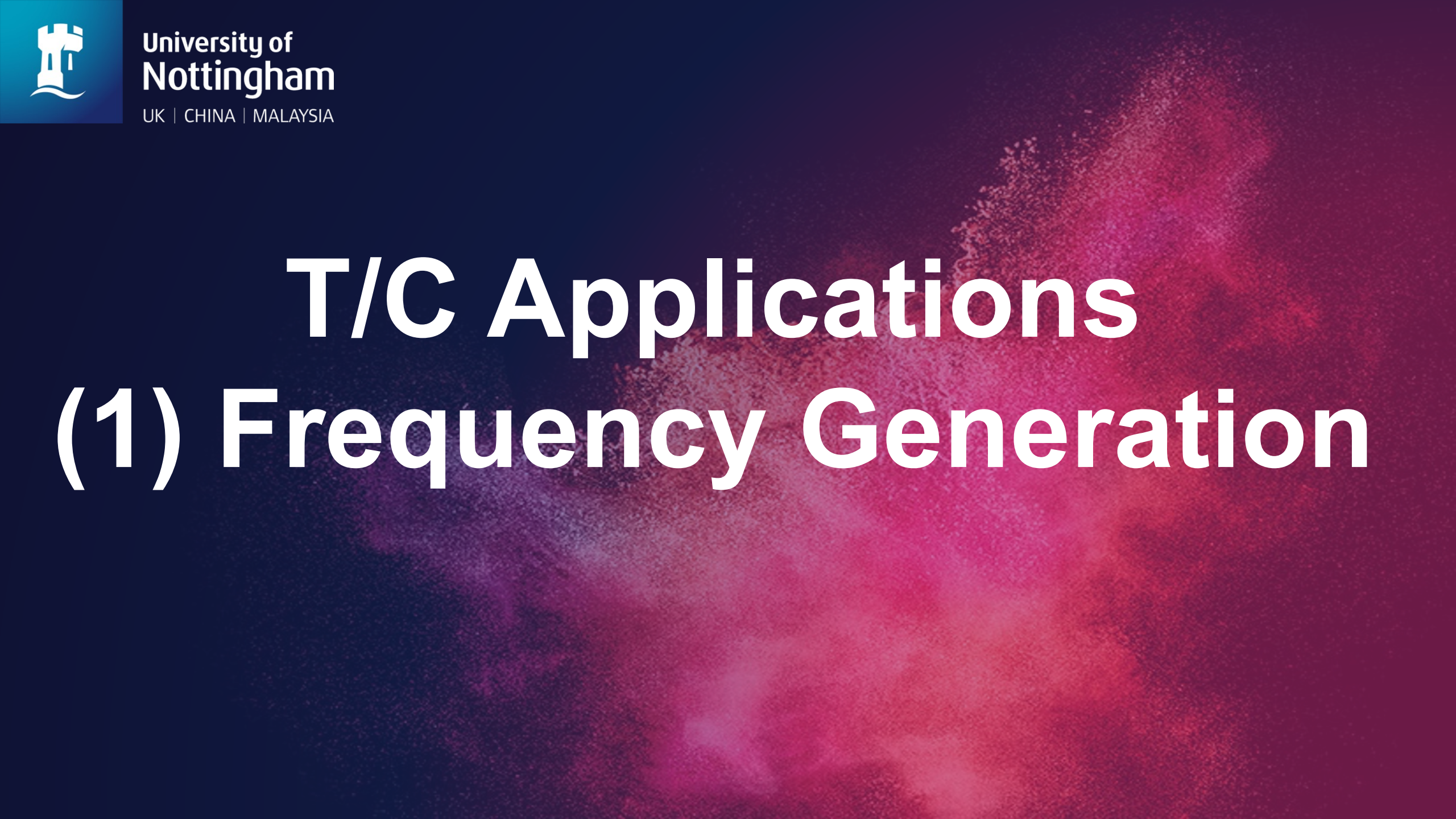
T/C Control Registers Settings - Interrupt Mask Register (TIMSKn)

Bit	7	6	5	4	3	2	1	0
(0x73)	–	–	ICIE5	–	OCIE5C	OCIE5B	OCIE5A	TOIE5
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Example: *n* = 5

- **Bit 3/2/1 – OCIEnC/B/A: T/Cn, Output Compare C/B/A Match Interrupt Enable:** When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the *T/Cn Output Compare C/B/A Match interrupt* is enabled.
- **Bit 0 – TOIEn: T/C, Overflow Interrupt Enable:** When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the *T/Cn Overflow interrupt* is enabled.
- **Bit 5 – ICIEn: T/Cn, Input Capture Interrupt Enable:** When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the *T/Cn Input Capture interrupt* is enabled.





T/C Applications
(1) Frequency Generation

Our old "Friend" Blink!

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

Arduino IDE: Easy in programming but Slow

```
void setup()
{
  DDRB = 0x80;
}
void loop()
{
  PORTB=0x80;
  delay(500);
  PORTB=0x00;
  delay(500);
}
```

Register Level: Harder in programming but run faster!

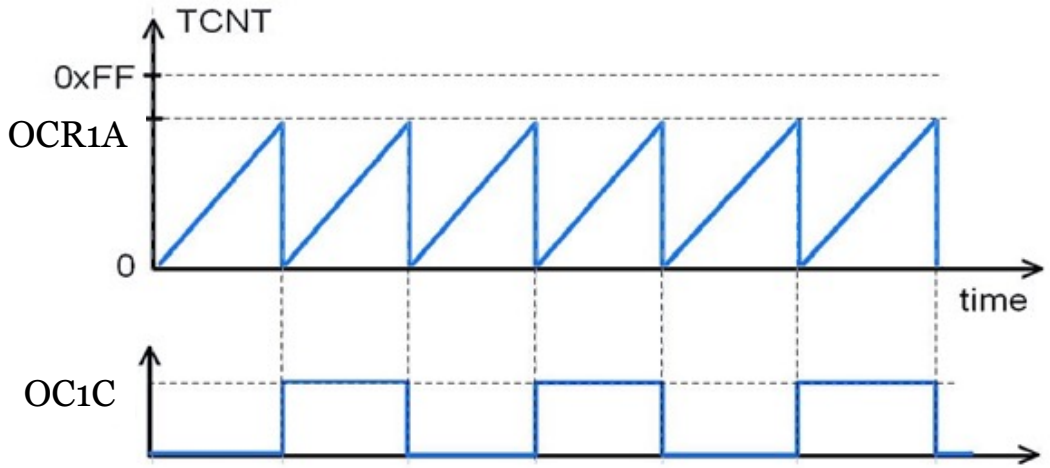
Can we make it even FASTER?!

Yes, the answer is T/C ☺

At "delay" the microprocessor is just "waiting"!

Our old "Friend" Blink!

- The output C of T/C1 on Arduino Mega is connected to pin 13 (LED).
- We aim to turn it on and off with a frequency of 0.5Hz. (i.e., 1 sec on and 1 sec off)
- We will use few lines of code to configure T/C1 to generate a frequency on its output C (i.e., pin 13).



How it works:

- Uses "clear timer on compare match" (CTC) mode.
- The value in the counter register (TCNT1) increases until it reaches the value in output compare register (OCR1C).
- At this point, the output pin OC1C swaps state, resets TCNT1 to 0.
- Big OCR1A=low frequency, small OCR1A=high frequency.
- The waveform frequency is defined by

$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

- The N variable represents the prescale factor (1, 8, 32, 64, 128, 256, or 1024).
- If the output frequency is 0.5Hz, f_clk_i/o=16,000,000 Hz, we can select N=1024, then the value in the OCR1A will be 15624.
- If we select N=256, then the value in the OCR1A will be 62499.
- Note: the value in OCR1A cannot exceed its maximum capacity (i.e., 16-bit 65535)!

Our old “Friend” Blink!

```
void setup()
{
  // Configure registers of Timer 1 to output 0.5 Hz to LED
  DDRB = (1 << 7);          // Set LED as output (pin 13, port B bit 7)
  TCCR1A = (1 << COM1C0 ); // Enable timer 1 Compare to toggle channel
  TCCR1B = (1 << WGM12 ); // Configure timer 1 for CTC mode
  TCCR1B |= ((1 << CS10 ) |(1 << CS12 )); // Start timer at Fcpu/1024
  OCR1A = 15624; // Set CTC compare value to 1Hz at 16 MHz AVR clock,
                 // with a prescaler of 1024
}

void loop()
{
  // It's empty!
}
```

T/C Control Registers Settings - TCCRnB

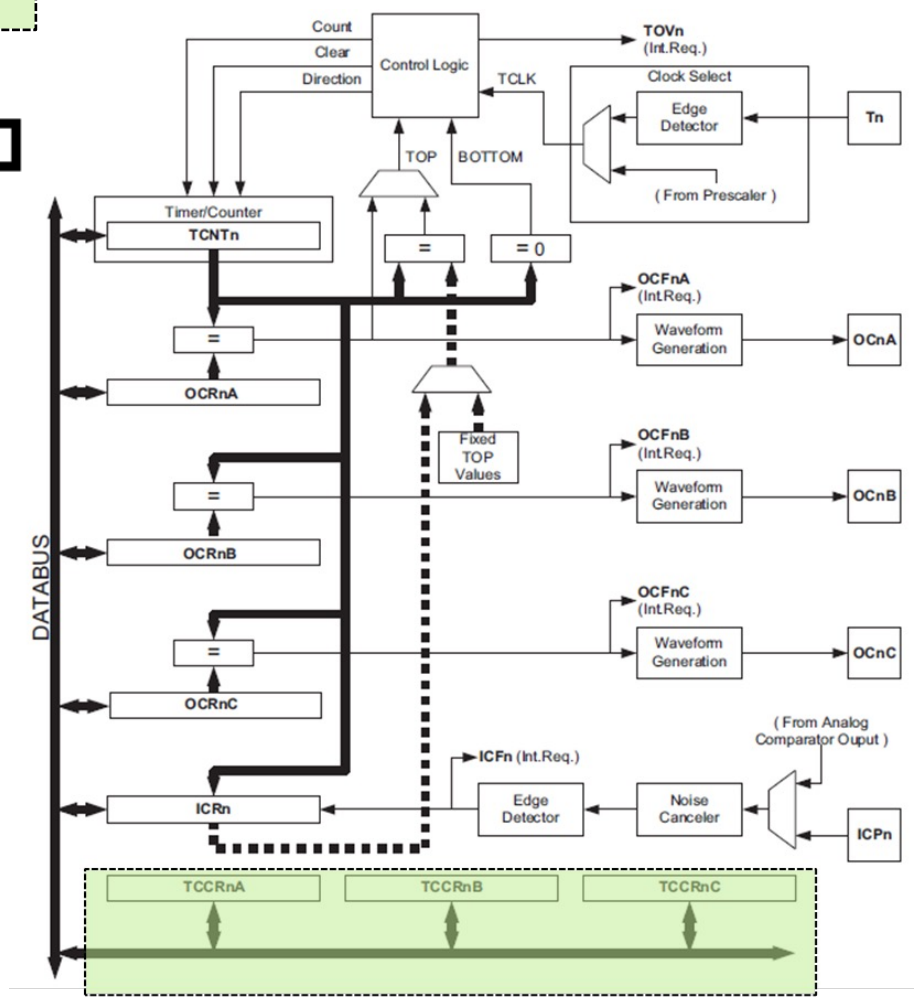
Bit	7	6	5	4	3	2	1	0
(0x120)	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Table 17-2. Waveform Generation Mode Bit Description

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnX at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Example: *n* = 5



Our old “Friend” Blink!

```
void setup()
{
  // Configure registers of Timer 1 to output 0.5 Hz to LED
✓ DDRB = (1 << 7);          // Set LED as output (pin 13, port B bit 7)
? TCCR1A = (1 << COM1C0 ); // Enable timer 1 Compare to toggle channel
✓ TCCR1B = (1 << WGM12 ); // Configure timer 1 for CTC mode
  TCCR1B |= ((1 << CS10 ) |(1 << CS12 )); // Start timer at Fcpu/1024
  OCR1A = 15624; // Set CTC compare value to 1Hz at 16 MHz AVR clock,
                 // with a prescaler of 1024
}

void loop()
{
  // It's empty!
}
```

T/C Control Registers Settings - TCCRnA

Bit	7	6	5	4	3	2	1	0
(0x120)	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

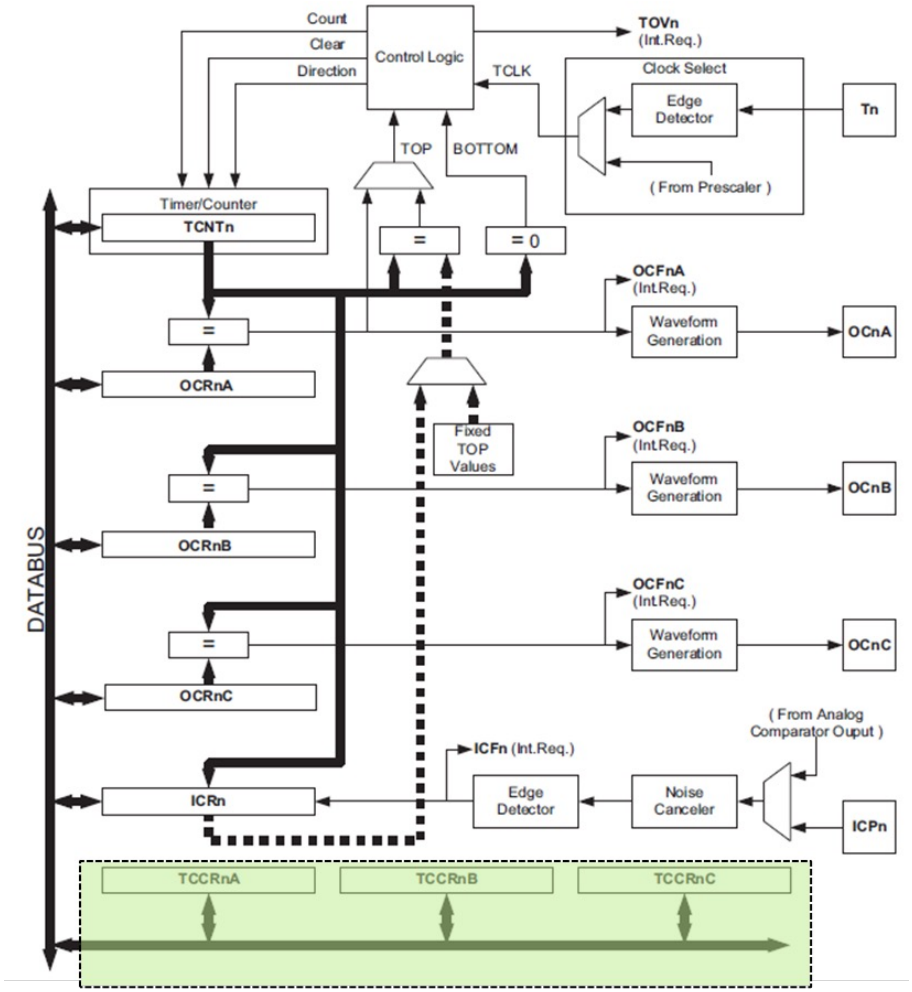
Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

Table 17-4. Compare Output Mode, Fast PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode)

Example: *n* = 5



Our old “Friend” Blink!

```
void setup()
{
  // Configure registers of Timer 1 to output 0.5 Hz to LED
✓ DDRB = (1 << 7);          // Set LED as output (pin 13, port B bit 7)
✓ TCCR1A = (1 << COM1C0 ); // Enable timer 1 Compare to toggle channel
✓ TCCR1B = (1 << WGM12 ); // Configure timer 1 for CTC mode
? TCCR1B |= ((1 << CS10 ) |(1 << CS12 )); // Start timer at Fcpu/1024
  OCR1A = 15624; // Set CTC compare value to 1Hz at 16 MHz AVR clock,
                 // with a prescaler of 1024
}

void loop()
{
  // It's empty!
}
```

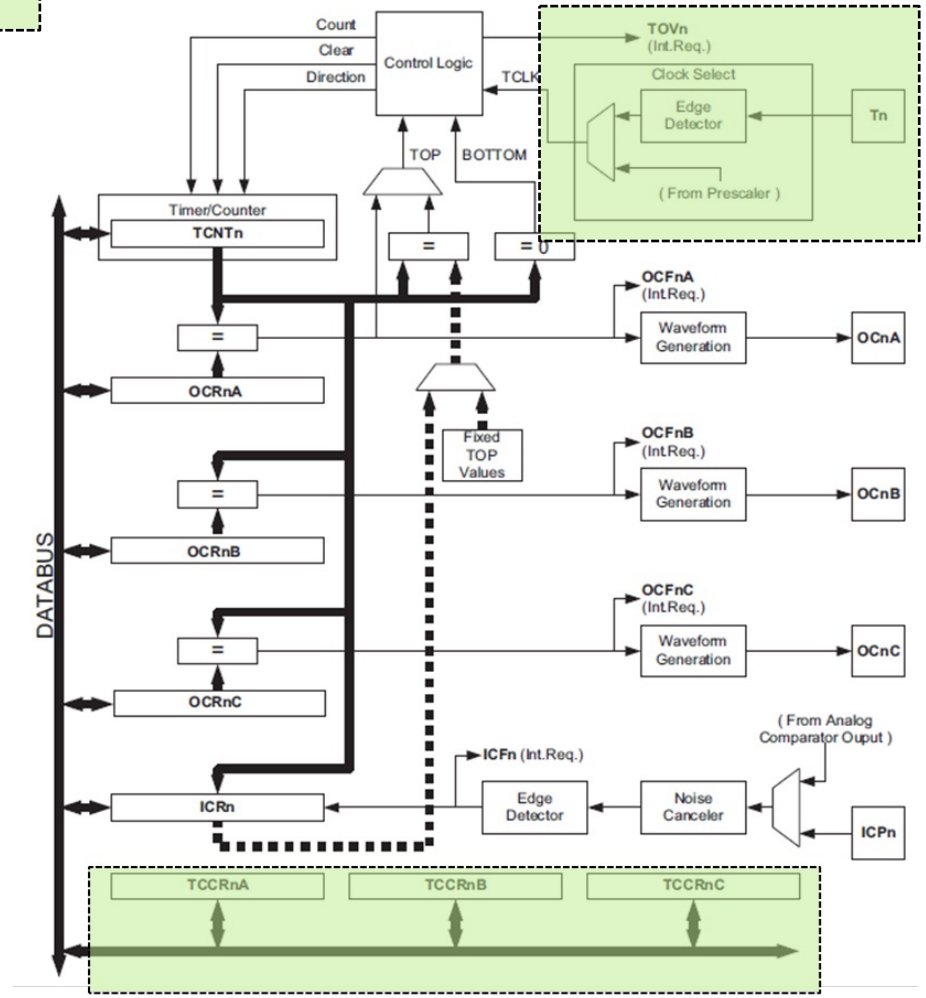
T/C Control Registers Settings - TCCRnB

Bit	7	6	5	4	3	2	1	0
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Example: *n* = 5

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	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	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge



Our old "Friend" Blink!

```
void setup()
{
  // Configure registers of Timer 1 to output 0.5 Hz to LED
✓ DDRB = (1 << 7);          // Set LED as output (pin 13, port B bit 7)
✓ TCCR1A = (1 << COM1C0 ); // Enable timer 1 Compare to toggle channel
✓ TCCR1B = (1 << WGM12 ); // Configure timer 1 for CTC mode
✓ TCCR1B |= ((1 << CS10 ) |(1 << CS12 )); // Start timer at Fcpu/1024
✓ OCR1A = 15624; // Set CTC compare value to 1Hz at 16 MHz AVR clock,
                 // with a prescaler of 1024
}

void loop()
{
  // It's empty!
}
```

$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

Understand concepts in comments, don't learn details!

Our old "Friend" Blink!

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
```

Arduino IDE: Easy in programming but slow

```
void setup()
{
  DDRB = 0x80;
}
void loop()
{
  PORTB=0x80;
  delay(500);
  PORTB=0x00;
  delay(500);
}
```

Register Level: Harder in programming but run faster!

```
void setup()
{
  DDRB = (1 << 7);
  TCCR1A = (1 << COM1C0 );
  TCCR1B = (1 << WGM12 );
  TCCR1B |= ((1 << CS10 ) |(1 << CS12 ));
  OCR1A = 15624;
}
void loop()
{
  // It's empty!
}
```

T/C: The job is purely don by hardware!
The processor can do other jobs!

At "delay" the microprocessor is just "waiting"!

University of Nottingham
UK | CHINA | MALAYSIA

T/C Applications (2) Counting High Frequency Pulses

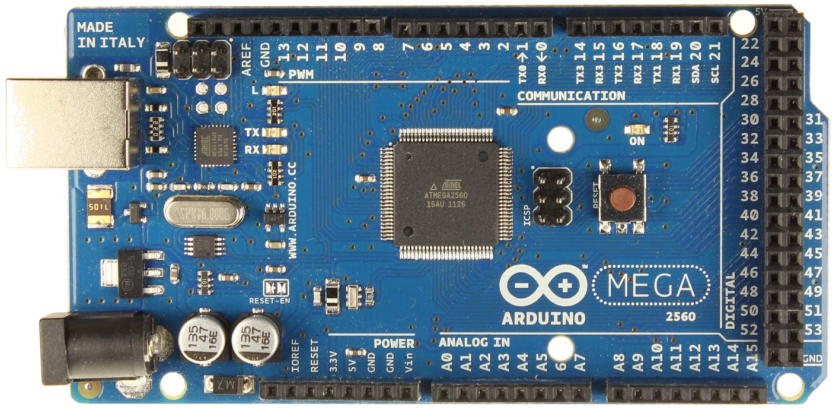
Counting High Frequency Pulses

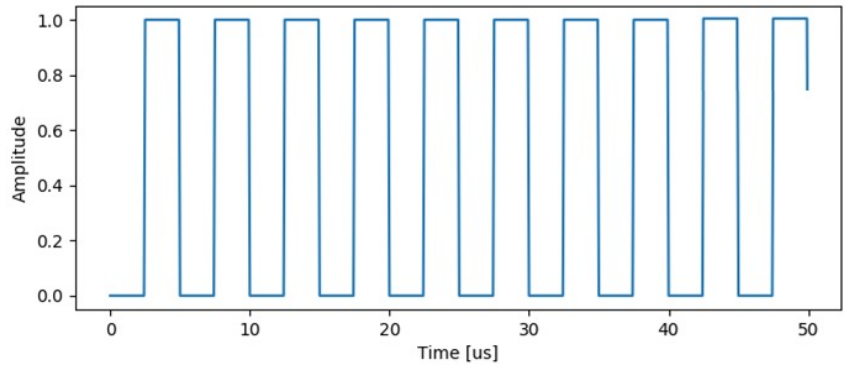
Task

- Let us consider an external pulse source is connected to Arduino Mega (pin 47). Note: one pin (only) on Uno or Mega can be used as the input to a counter.
- We aim to count how many pulses we receive on this pin and show the number of the monitor.

How it works:

- Pin 47 is connected to the input of T/C 5.
- Configure T/C 5 to have input from external pulse source (not internal clock).
- Every pulse increments counter register TCNT5 by 1.
- Configure the T/C to run in the “normal” mode, it resets to zero (“overflows”) every 65536 (its maximum capacity!) pulses.
- Read the value in the counter register and calculate total pulses.





Counting High Frequency Pulses

```
unsigned long bigLaps;

void setup()
{
  Serial.begin(9600);

  TCCR5A = 0; // No waveform generation needed.
  TCCR5B = (1<<CS50) | (1<<CS51) | (1<<CS52); // Normal mode, clk from pin T5 (47) rising edge.
  TCCR5C = 0; // No force output compare.
  TCNT5 = 0;  // Initialise counter register to zero.
  TIMSK5 = (1<<TOIE5);  // Enable overflow interrupt
  bigLaps = 0; // Initialise number of times counter overflowed
}

ISR(TIMER5_OVF_vect)
{
  //when this runs, you had 65365 pulses counted.
  bigLaps++;
}

void loop()
{
  Serial.print("total count including wraparounds count ");
  Serial.println(TCNT5 + bigLaps*65536);
  delay(1000);
}
```

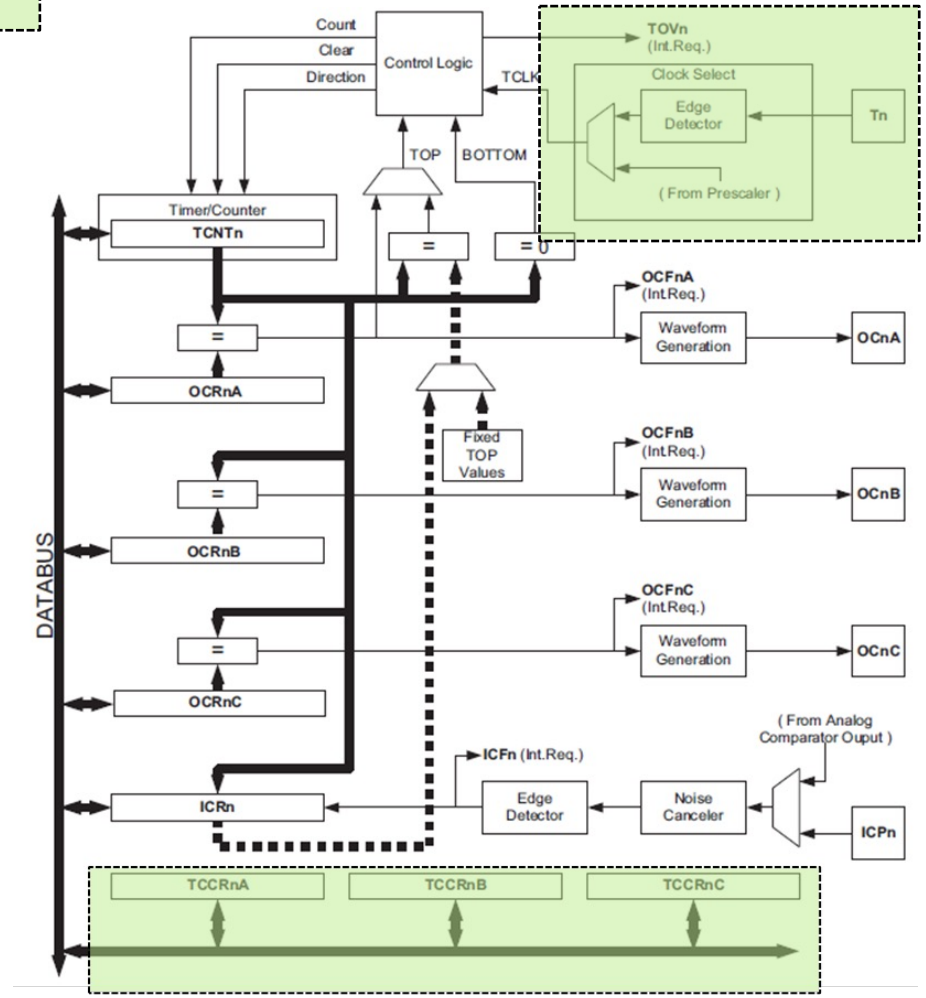
T/C Control Registers Settings - TCCRnB

Bit	7	6	5	4	3	2	1	0
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Example: *n* = 5

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	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	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge



Counting High Frequency Pulses

```
unsigned long bigLaps;

void setup()
{
  Serial.begin(9600);

  TCCR5A = 0; // No waveform generation needed.
  TCCR5B = (1<<CS50) | (1<<CS51) | (1<<CS52); // Normal mode, clk from pin T5 (47) rising edge.
  TCCR5C = 0; // No force output compare.
  TCNT5 = 0;  // Initialise counter register to zero.
  TIMSK5 = (1<<TOIE5);  // Enable overflow interrupt
  bigLaps = 0; // Initialise number of times counter overflowed
}

ISR(TIMER5_OVF_vect)
{
  //when this runs, you had 65365 pulses counted.
  bigLaps++;
}

void loop()
{
  Serial.print("total count including wraparounds count ");
  Serial.println(TCNT5 + bigLaps*65536);
  delay(1000);
}
```

T/C Control Registers Settings - TCCRnA

Bit	7	6	5	4	3	2	1	0
(0x120)	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- **Bit 7:6 – COMnA1:0: Compare Output Mode for Channel A**
- **Bit 5:4 – COMnB1:0: Compare Output Mode for Channel B**
- **Bit 3:2 – COMnC1:0: Compare Output Mode for Channel C**

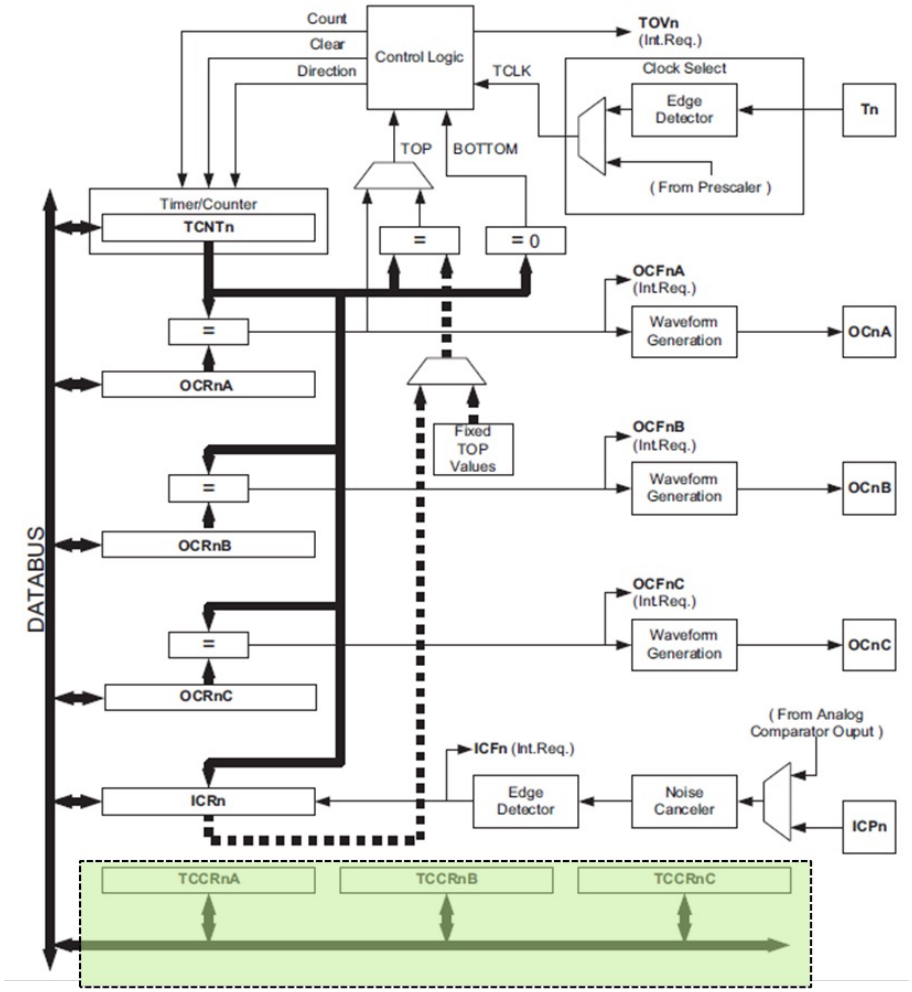
Table 17-3. Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	Toggle OCnA/OCnB/OCnC on compare match
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level)
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level)

Table 17-4. Compare Output Mode, Fast PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode)
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode)

Example: *n* = 5



T/C Control Registers Settings - TCCRnB

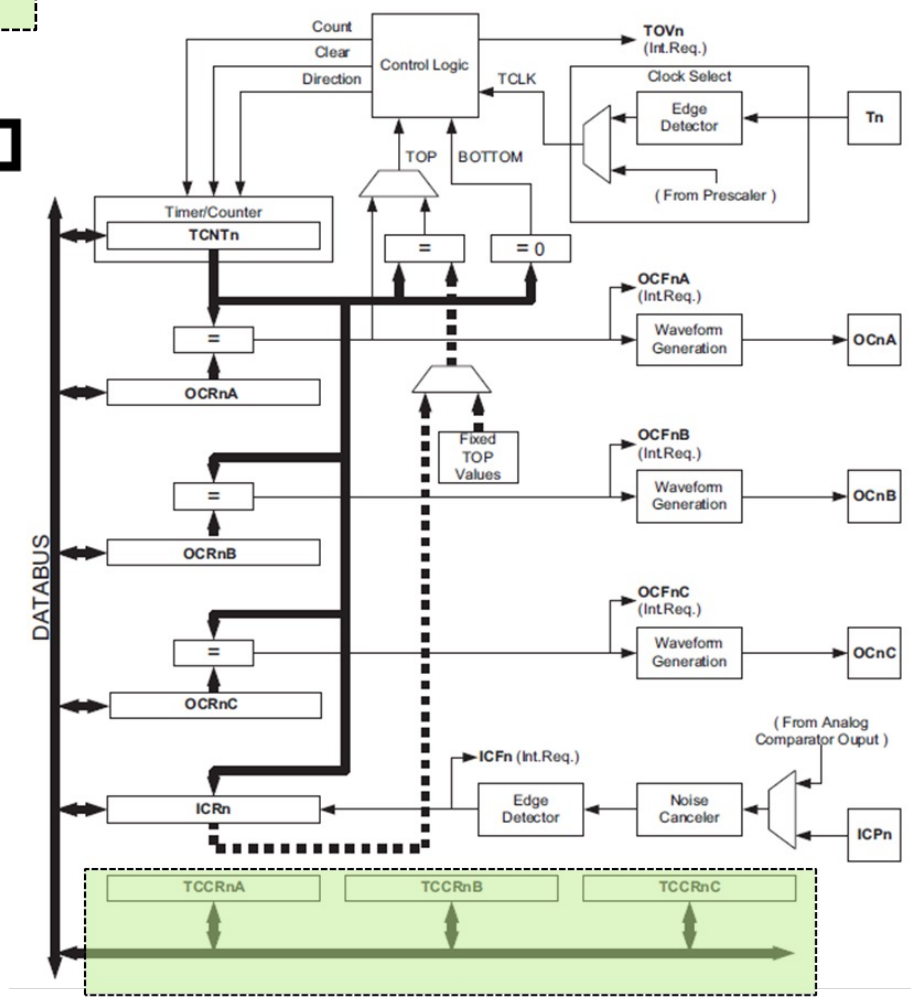
Bit	7	6	5	4	3	2	1	0
(0x120)	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
(0x121)	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Table 17-2. Waveform Generation Mode Bit Description

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnX at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Example: $n = 5$



Counting High Frequency Pulses

```
unsigned long bigLaps;

void setup()
{
  Serial.begin(9600);

  TCCR5A = 0; // No waveform generation needed.
  TCCR5B = (1<<CS50) | (1<<CS51) | (1<<CS52); // Normal mode, clk from pin T5 (47) rising edge.
  TCCR5C = 0; // No force output compare.
  TCNT5 = 0;  // Initialise counter register to zero.
  TIMSK5 = (1<<TOIE5);  // Enable overflow interrupt
  bigLaps = 0; // Initialise number of times counter overflowed
}

ISR(TIMER5_OVF_vect)
{
  //when this runs, you had 65365 pulses counted.
  bigLaps++;
}

void loop()
{
  Serial.print("total count including wraparounds count ");
  Serial.println(TCNT5 + bigLaps*65536);
  delay(1000);
}
```

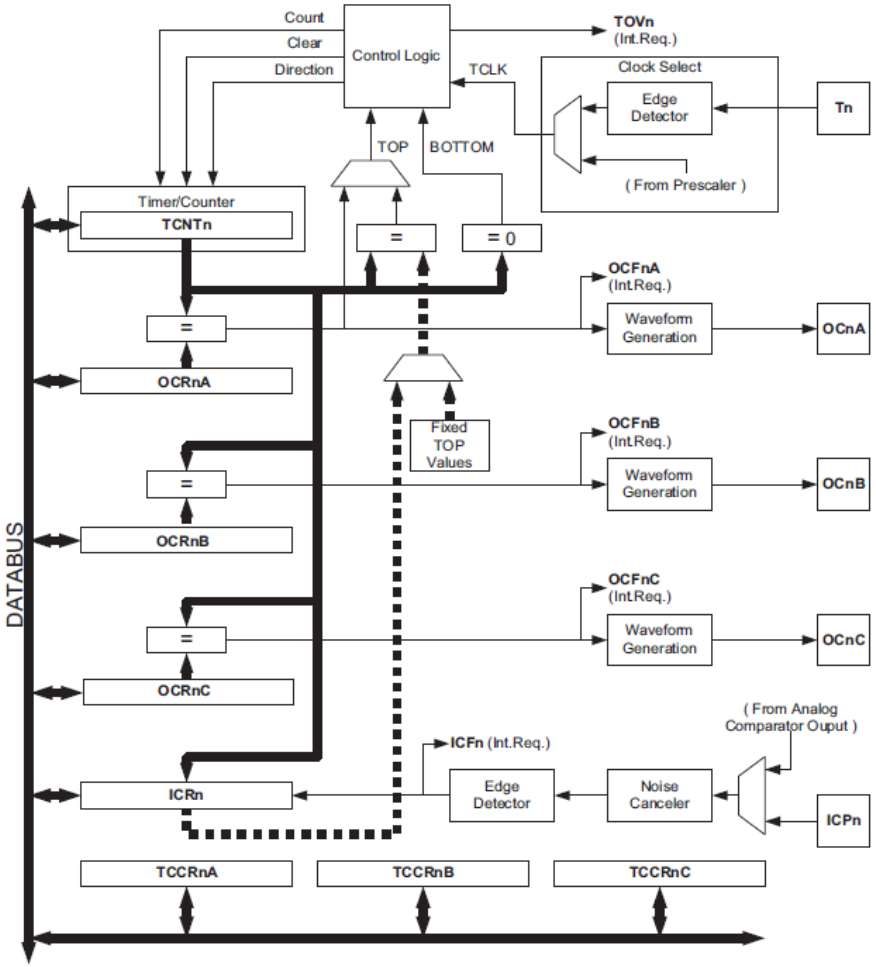
What will happen if the TCNT5 register is full?!

We will trigger an interrupt when the TCNT5 register is full (i.e., overflows), so we can keep track of number of overflows (we will call it biglaps) and hence total count.

T/C Control Registers Settings - Interrupt Mask Register (TIMSKn)

Bit	7	6	5	4	3	2	1	0
(0x73)	–	–	ICIE5	–	OCIE5C	OCIE5B	OCIE5A	TOIE5
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Example: *n* = 5



- **Bit 3/2/1 – OCIEnC/B/A: T/Cn, Output Compare C/B/A Match Interrupt Enable:** When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the *T/Cn Output Compare C/B/A Match interrupt* is enabled.
- **Bit 0 – TOIEn: T/C, Overflow Interrupt Enable:** When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the *T/Cn Overflow interrupt* is enabled.
- **Bit 5 – ICIEn: T/Cn, Input Capture Interrupt Enable:** When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the *T/Cn Input Capture interrupt* is enabled.

Counting High Frequency Pulses

```
unsigned long bigLaps;

void setup()
{
  Serial.begin(9600);

  TCCR5A = 0; // No waveform generation needed.
  TCCR5B = (1<<CS50) | (1<<CS51) | (1<<CS52); // Normal mode, clk from pin T5 (47) rising edge.
  TCCR5C = 0; // No force output compare.
  TCNT5 = 0;  // Initialise counter register to zero.
  TIMSK5 = (1<<TOIE5);  // Enable overflow interrupt
  bigLaps = 0; // Initialise number of times counter overflowed
}

ISR(TIMER5_OVF_vect)
{
  //when this runs, you had 65365 pulses counted.
  bigLaps++;
}

void loop()
{
  Serial.print("total count including wraparounds count ");
  Serial.println(TCNT5 + bigLaps*65536);
  delay(1000);
}
```

What will happen if the TCNT5 register is full?!

If overflow happens, this *routine* will be executed! This will increase *biglaps* with 1.

Counting High Frequency Pulses

```
unsigned long bigLaps;

void setup()
{
  Serial.begin(9600);

  TCCR5A = 0; // No waveform generation needed.
  TCCR5B = (1<<CS50) | (1<<CS51) | (1<<CS52); // Normal mode, clk from pin T5 (47) rising edge.
  TCCR5C = 0; // No force output compare.
  TCNT5 = 0;  // Initialise counter register to zero.
  TIMSK5 = (1<<TOIE5);  // Enable overflow interrupt
  bigLaps = 0; // Initialise number of times counter overflowed
}

ISR(TIMER5_OVF_vect)
{
  //when this runs, you had 65365 pulses counted.
  bigLaps++;
}

void loop()
{
  Serial.print("total count including wraparounds count ");
  Serial.println(TCNT5 + bigLaps*65536);
  delay(1000);
}
```

Understand concepts in comments, don't learn details!

Your turn now… Finish as Homework ☺

- You need to configure T/C 1 to output PWM ("fast PWM") on pin 13 at 20 kHz.
- Several possible solutions but suggest:
  - T/C 1 fed with 16 MHz clock signal directly (divisor 1, i.e., no prescaling)
  - Fast PWM: TCNT1 counts up until matches value in ICR1 ("TOP"), resets to 0, restarts
  - This sets output high at each cycle start
  - Output C goes low (reset) when counter register TCNT1 matches value in OCR1C (i.e., "clears on compare match")

Your turn now… Finish as Homework ☺

Where to start:

- Look for CS0…2 combination which gives 16 MHz clock source directly
- Look for WGM0..3 combination (mode) which gives fast PWM, TOP set by ICR1
- Look for COM1C0..1 settings which give “output compare match” which resets output C to 0 when TCNT1 matches OCR1C
- Build up TCCR1A, TCCR1B from these
- Assume TCCR1C is 0, no need for TIMSK1.

Your turn now… Finish as Homework ☺

Some practical points:

- Use the skeleton program provided
- Pin 13 must be declared as an output (either use `pinMode()` or set the appropriate bit of the appropriate port direction register)
- Set the value required for ICR5 to get 20 kHz according to the formula: $f_{PWM} = f_{clock}/(prescaler \times (1+ICR5))$
- For a given duty cycle (which you can enter via the serial monitor), calculate the value to be written to OCR5C then write it there!

Summary

- Explained why simple interface is not good for counting events or measuring frequencies
- Introduced need for hardware timer-counters
- Examined timer-counters configuration using registers
- Applications of T/C such as frequency generation, counting high frequency pulses and generating PWM.
- Gain some experience of using datasheet.