ECCS-3351
Embedded Realtime Applications (ERA)

Timers - Periodic Interrupts With *Timer A* modules

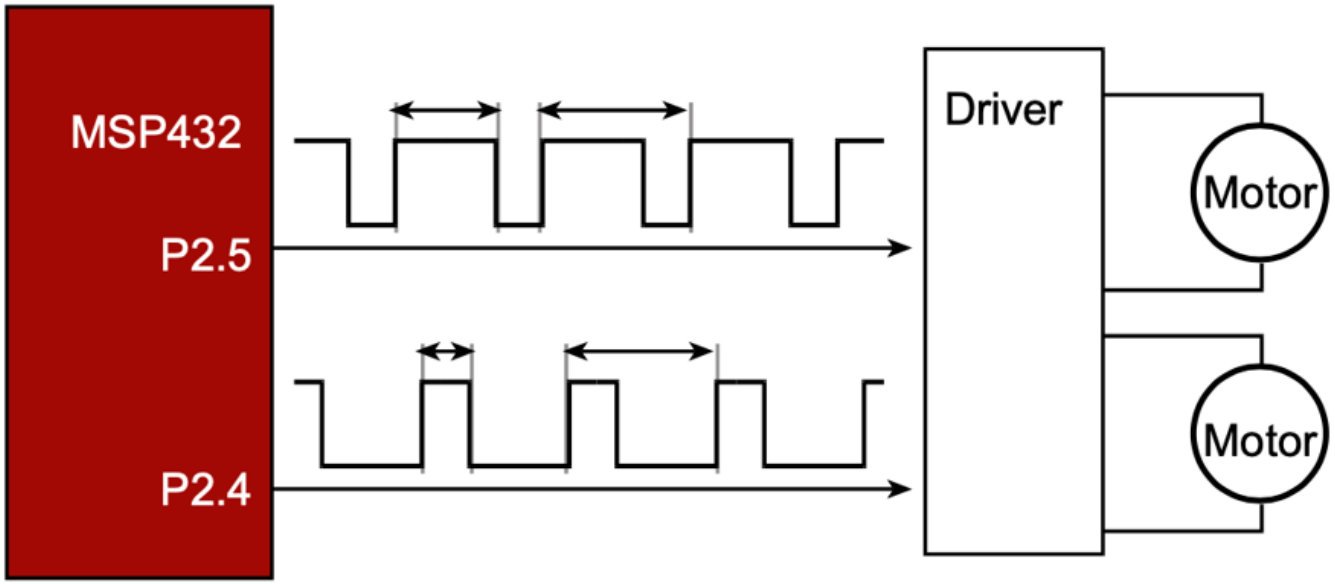
JONATHAN W. VALVANO
Modified by Drs. Kropp, Oun, and Youssfi

Objectives

- Use *Timer A* modules to:
  - Generate periodic interrupts
    - Run multiple threads (tasks)
    - Set their priority
  - Generate parallel PWM signals
    - With different duty cycle using different timers independently with different duty cycles.

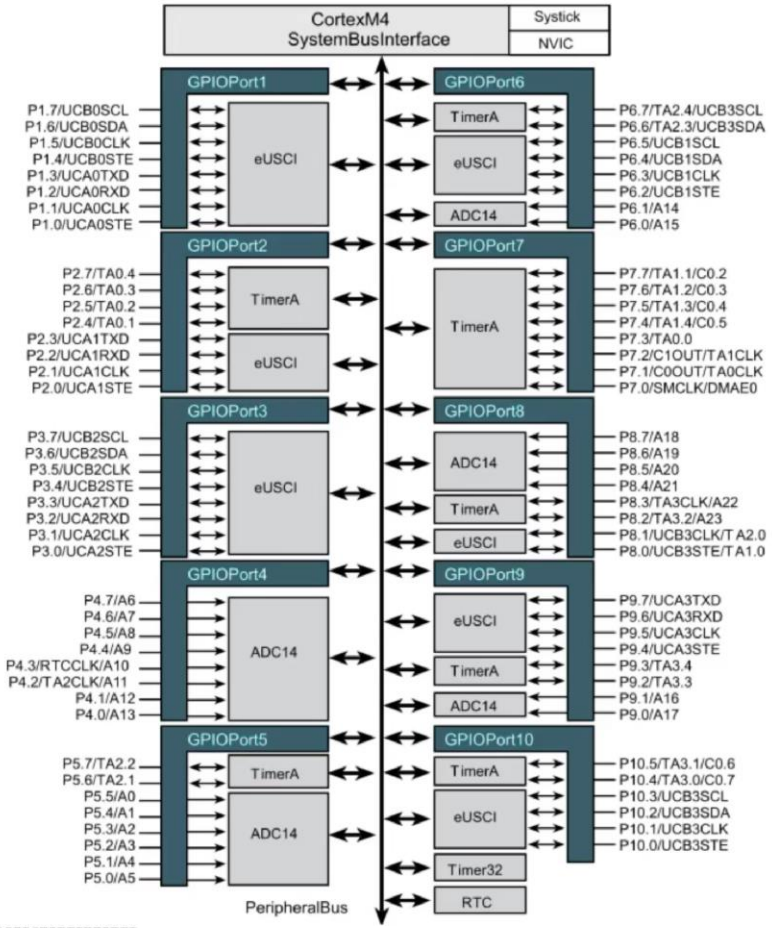
Why not SysTick?

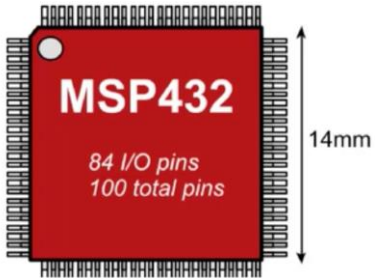
How would you program this with SysTick?



Programming two PWM signals with SysTick would be a mess…

What are the *Timer A* modules?

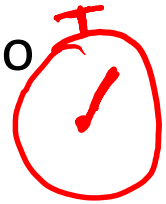




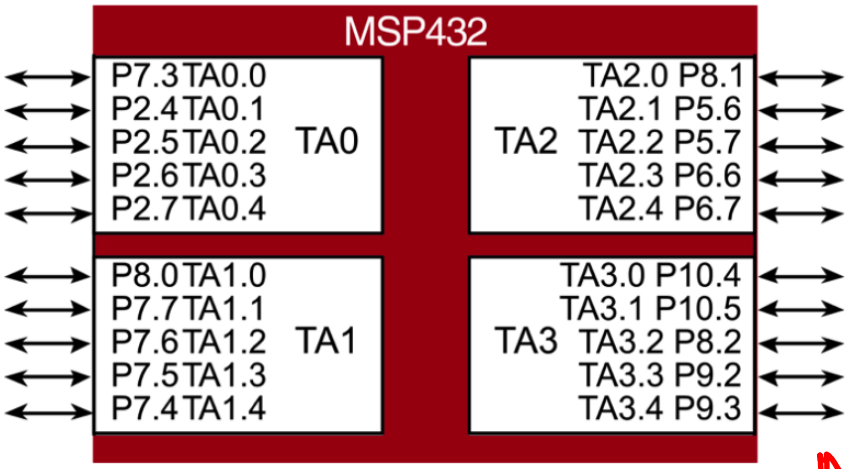
- You can configure 4 independent *Timer A* modules:
  - TA0.x
  - TA1.x
  - TA2.x
  - TA3.x
- Each timer has four sub-modules, e.g.,
  - TA1.0
  - TA1.1
  - TA1.2
  - TA1.2
  - TA1.3
  - TA1.4

Capture/compare

- These timers have two modes: capture and compare
- *Capture mode* will measure the time it takes for an event to happen on a pin (Like a stopwatch)
  - Timing how long for an input to go high
  - Frequency measurement
- *Compare mode* will trigger an event after a pre-defined period of time (like a timer)
  - PWM generation



What are the *Timer A* modules?



- These timers can work similarly to SysTick
- For example:
  1. Load TAxR with value TAxCCR0
  2. Count down till you hit 0
  3. Reset TAxR with TAxCCR0
  4. Go back to step 1
- Each module has one 16-bit timer and seven associated capture/compare registers.

Refresh value

Timer A

TAxCCR0

Place holder for which timer (TA1, TA2, ...)

capture/compare

Submodule

Other *Timer A* configuration options

- Has four possible clock sources:
  - SMCLK: Sub master clock (CPU speed)
  - ACLK: Auxiliary clock (Slower than CPU speed)
  - TAxCLK: External clock on TAxCLK
  - INCLK: Global external clock
- Has a two stage clock divider (ID=1,2,4,8) plus (TAIDEX=1,2,3,4,5,6,7,8)
- Counts up or up/down depending on the mode
- Can be set or cleared by software

Timer A registers

A single module, & all its submodule

15-10	9-8	7-6	5-4	3	2	1	0	Name
	TASSEL	ID	MC		TACLR	TAIE	TAIFG	TA0CTL

control regs

global control registers

15-14	13-12	11	10	9	8	7-5	4	3	2	1	0	
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL0
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL1
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL2
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL3
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL4
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL5
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL6

count registers

15–0	
16-bit counter	TA0R
16-bit Capture/Compare 0 Register	TA0CCR0 → TA0.0
16-bit Capture/Compare 1 Register	TA0CCR1
16-bit Capture/Compare 2 Register	TA0CCR2
16-bit Capture/Compare 3 Register	TA0CCR3
16-bit Capture/Compare 4 Register	TA0CCR4 → TA0.4
16-bit Capture/Compare 5 Register	TA0CCR5
16-bit Capture/Compare 6 Register	TA0CCR6

Compare out

Always the reset value

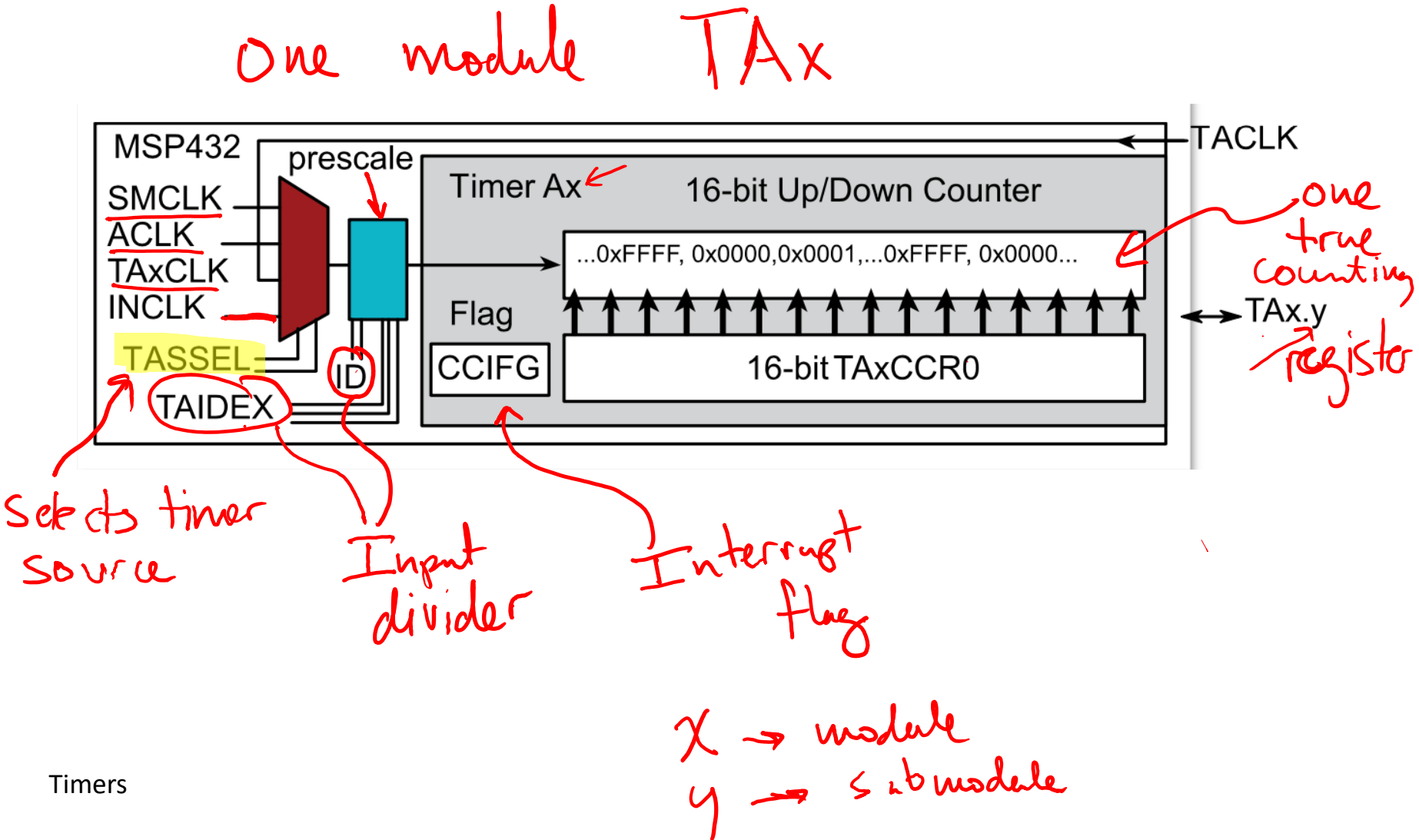
15-3	2-0	
	TAIDEX	TA0EX0

clock dividers

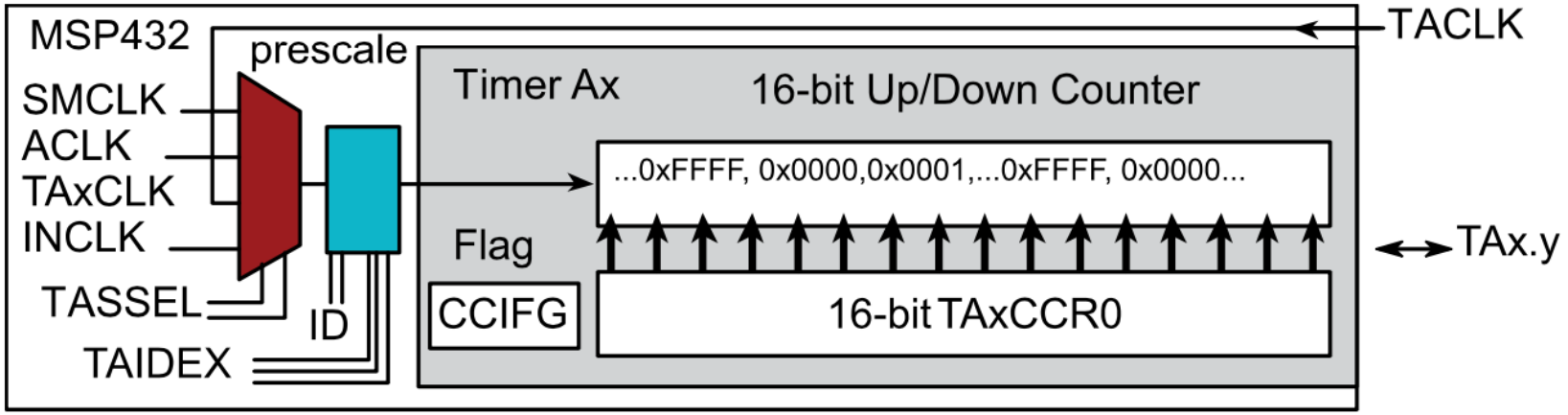
Interrupts

15-0	
TAIV	TA0IV

Timer A Configuration



Clock & Prescale



TASSEL	Selected Clock
00	TAxCLK
01	ACLK
10	SMCLK
11	INCLK

ID	Prescale
00	/1
01	/2
10	/4
11	/8

$$T_{new} = T_{old} \times 2^{\mathrm{ID}} \times (\mathrm{TAIDEX} + 1)$$

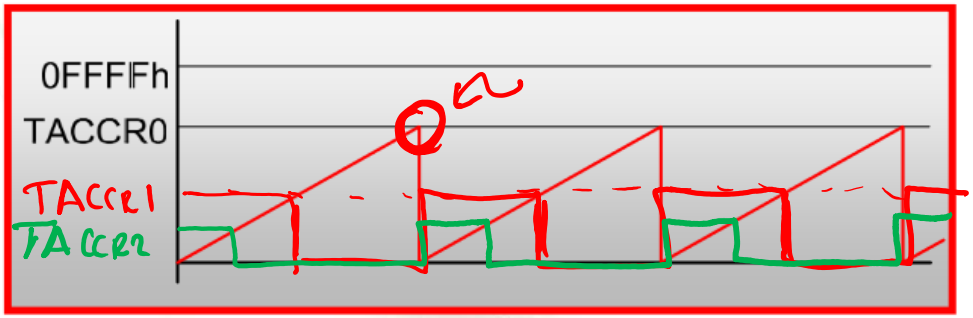
ID=01
ID=10

48MHz → 24MHz
48MHz → 12MHz

Timers

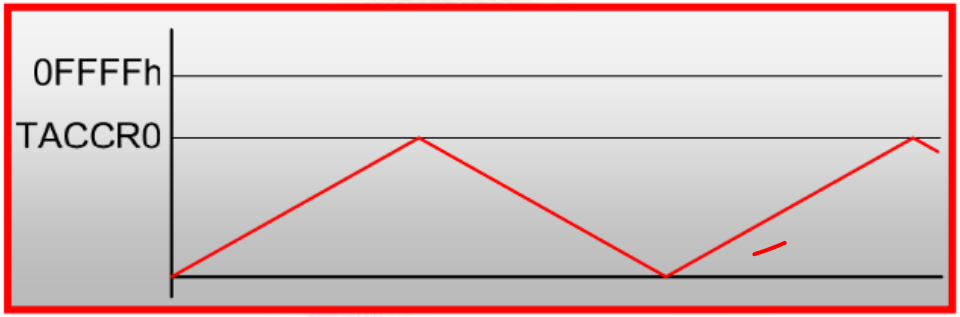
Timer A: Count modes

Up



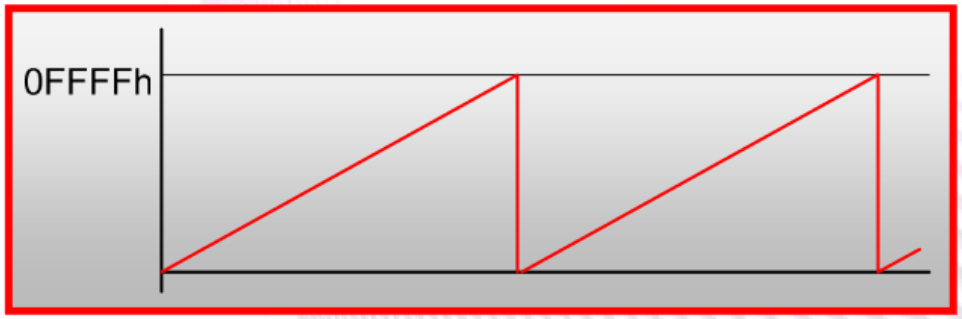
Up to FFFF, rolls over to 0000, back up to FFFF, etc.

Up/ Down



Up to value in CCR0, count down to 0000, back up to value in CCR0, etc.

Continuous

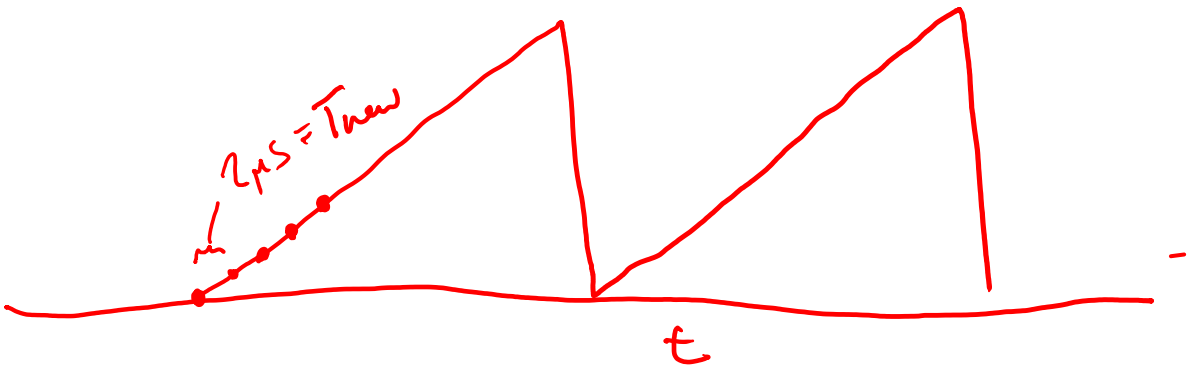


Up to value specified by CCR0, rolls over to 0000, back up to CCR0 value, etc.

https://www.ti.com/lit/ml/slap113/slap113.pdf

ID and TAIDEX

- $T_{new} = T_{old} \times 2^{\mathrm{ID}} \times (\mathrm{TAIDEX} + 1)$
- Two registers to modify the default clock:
  - ID: Can divide the timer source by 2, 4, 8
  - TAIDEX can *further* divide the timer source by 2, 3, 4, 5, 6, 7, or 8
- Example: How to make a $2\mu S$ clock period from a 12MHz signal?
- Write this out in an Excel Spreadsheet and solve for ID and TAIDEX
  - There are more than one correct answer



Timer range

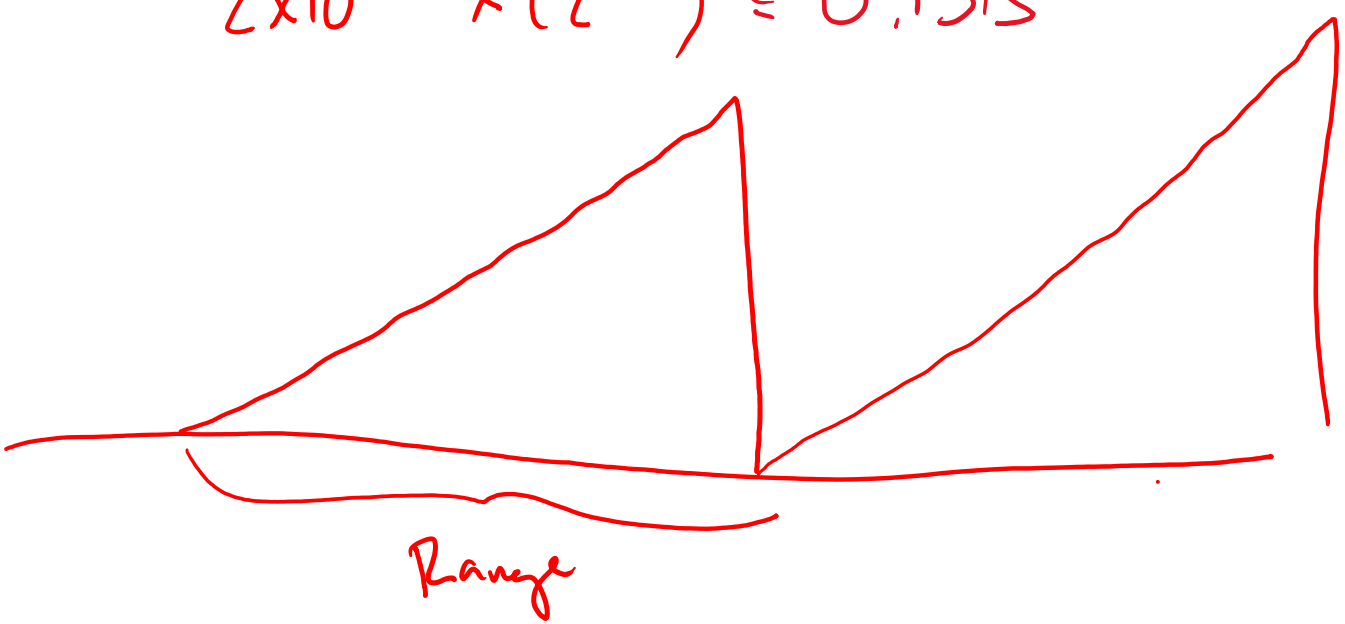
- Timer range is the length of each full cycle
- $range = T \times resolution$
  - Resolution is the bit resolution of the counter
- What's the range of a clock with a $2\mu S$ period on a 16 bit TAxR?

Period

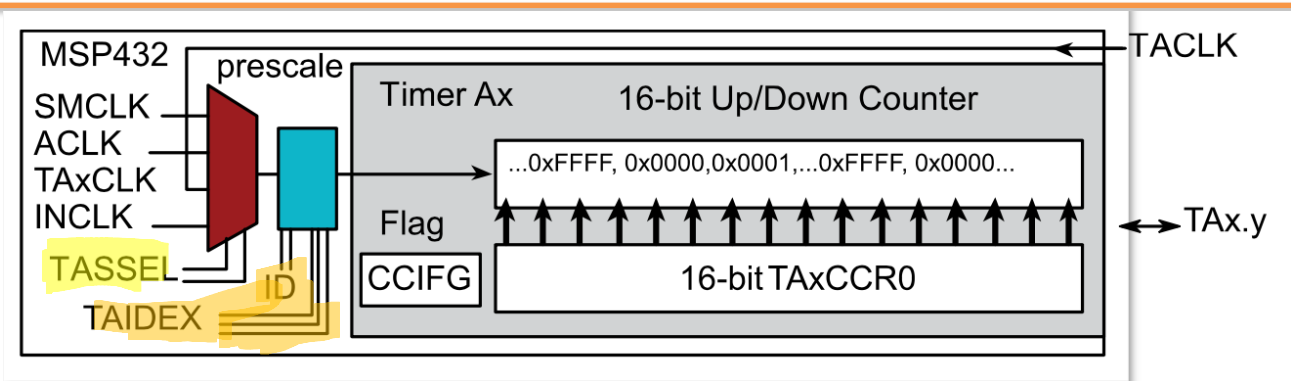
bit resolution

clock

$2 \times 10^{-6} \times (2^{16}) \approx 0.1315$



Periodic Interrupts Initialization Example



If we want to create a periodic interrupt on Module x, submodule y

1. Halt the timer (MC=00)
2. Set the timer clock and prescale
3. Set submodule 0 to compare, arm interrupt
4. Set the TAxCCR0 to the interrupt period minus 1
5. Set the interrupt priority in the correct NVIC priority register
6. Enable the interrupt in the NVIC Interrupt Enable register
7. Reset the timer and start it in up mode
8. Enable interrupts (in the main program after all devices initialized)

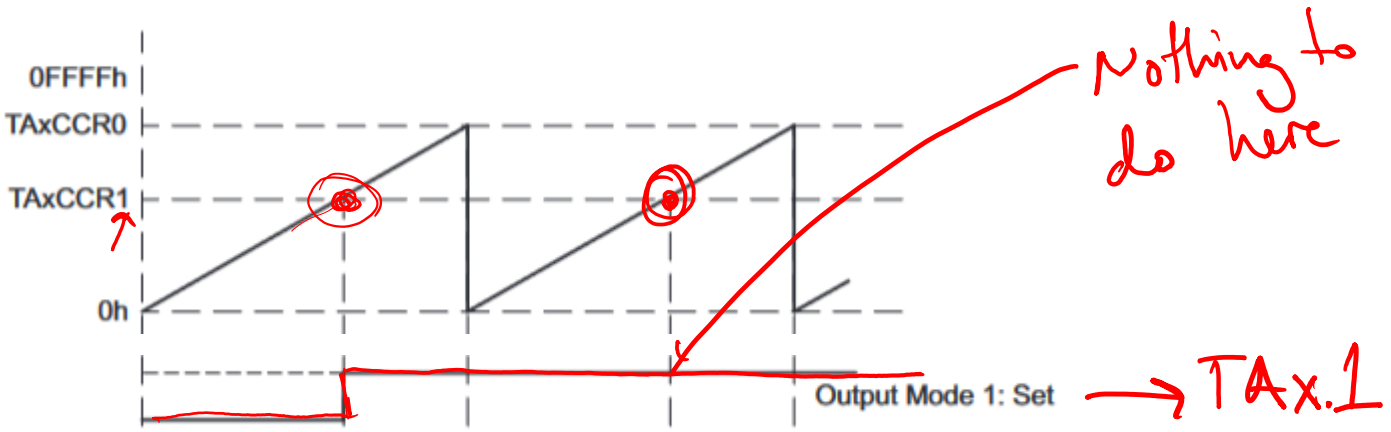
Timer A Modes

MC	Mode control
00	Stop
01	Up mode: Timer counts up to TAxCCR0
10	Continuous mode: Timer counts up to 0xFFFF
11	Up/down mode: Timer counts up to TAxCCR0 then down to 0x0000

OUTMOD	On match to TAxCCRy	On match to TAxCCR0
000	OUT bit value	
001	Set	
010	Toggle	Reset
011	Set	Reset
100	Toggle	
101	Reset	
110	Toggle	Set
111	Reset	Set

Different configurations of wave forms & duty cycles.

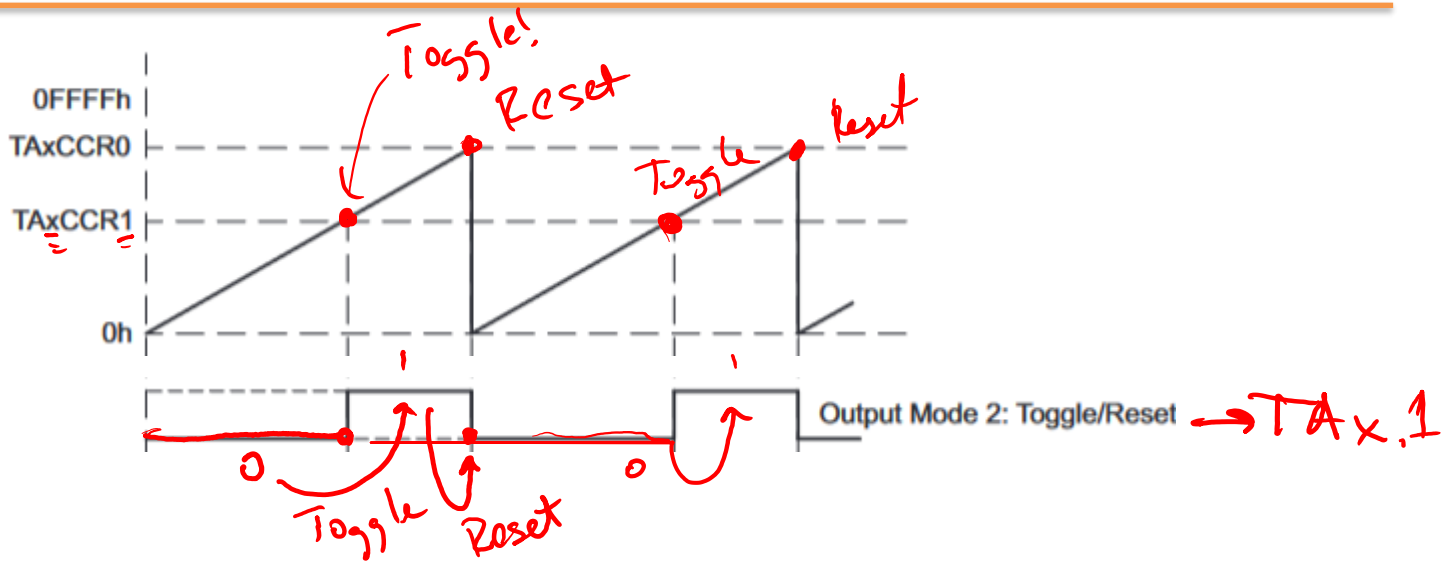
How modes work (up mode example)



Set => pin goes high
reset => pin goes low
toggle => pin is reversed

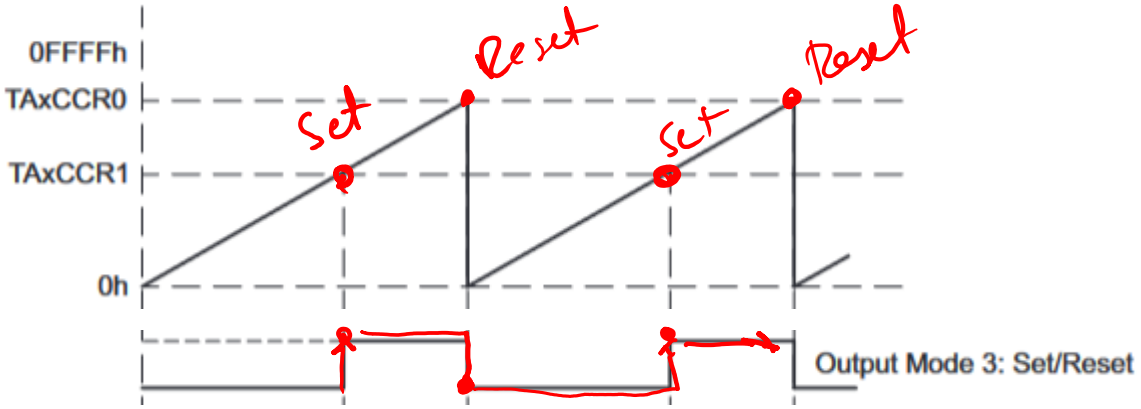
OUTMOD	On match to TAxCCRy	On match to TAxCCR0
000	OUT bit value	
001	Set	
010	Toggle	Reset
011	Set	Reset
100	Toggle	
101	Reset	
110	Toggle	Set
111	Reset	Set

How modes work (up mode example)



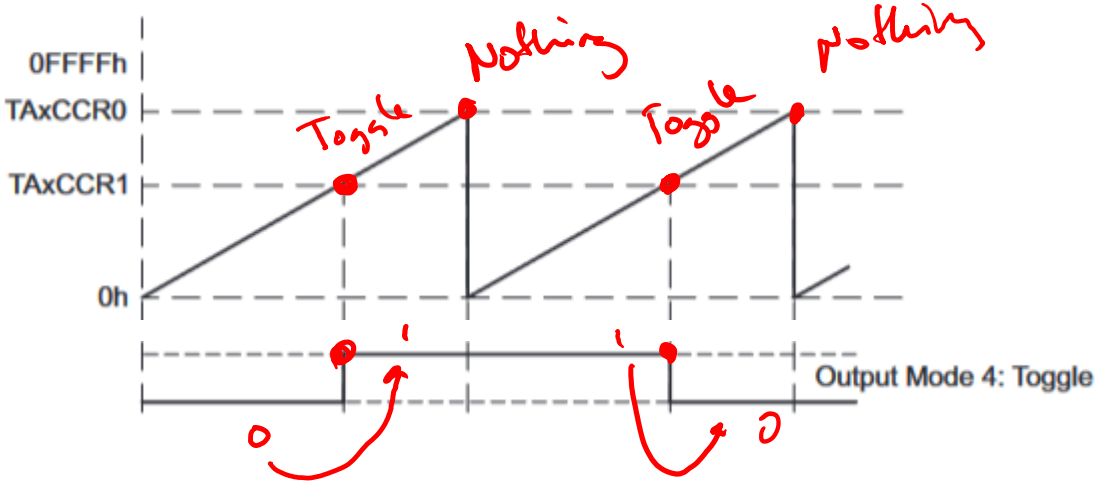
OUTMOD	On match to TAxCCRy	On match to TAxCCR0
000	OUT bit value	
001	Set	
010	Toggle	Reset
011	Set	Reset
100	Toggle	
101	Reset	
110	Toggle	Set
111	Reset	Set

How modes work (up mode example)



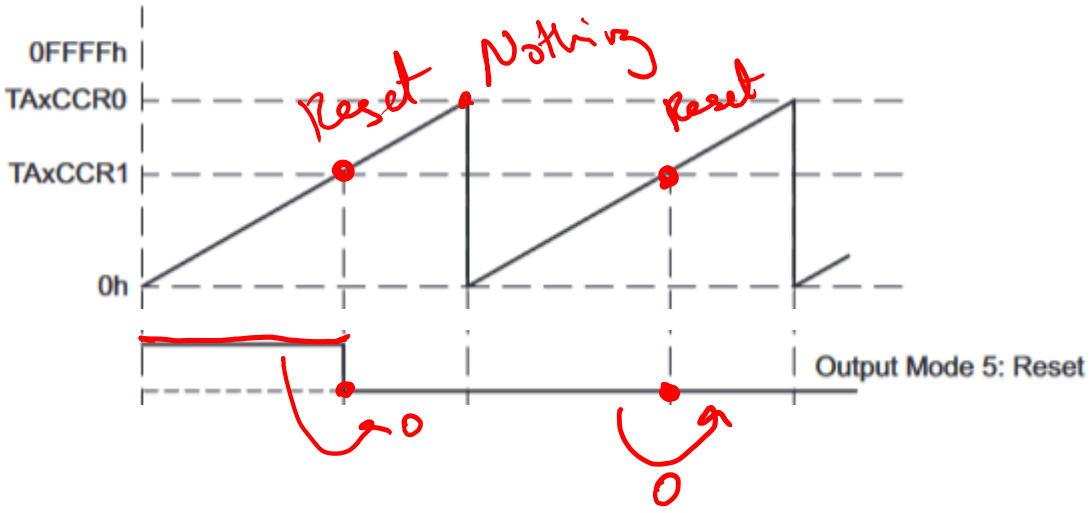
OUTMOD	On match to TAxCCRy	On match to TAxCCR0
000	OUT bit value	
001	Set	
010	Toggle	Reset
011	Set	Reset
100	Toggle	
101	Reset	
110	Toggle	Set
111	Reset	Set

How modes work (up mode example)



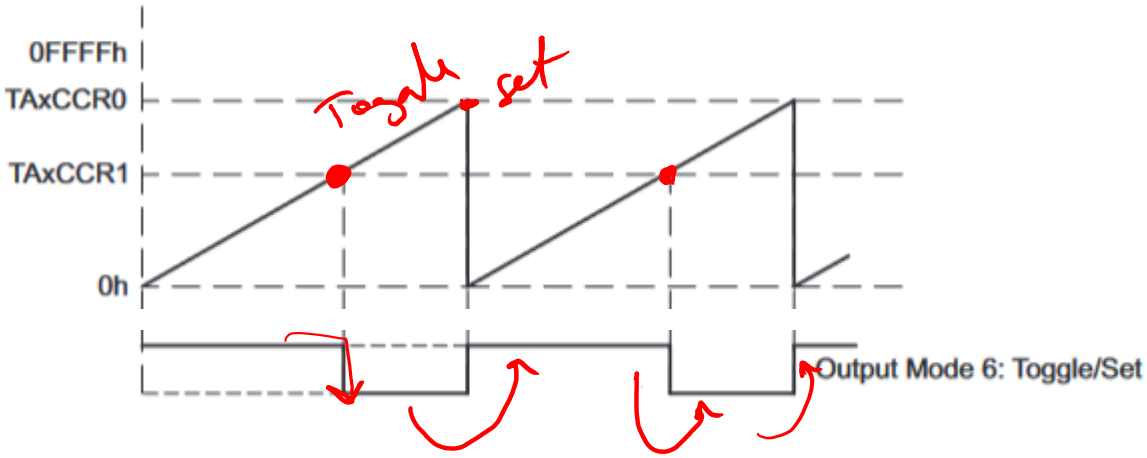
OUTMOD	On match to TAxCCRy	On match to TAxCCR0
000	OUT bit value	
001	Set	
010	Toggle	Reset
011	Set	Reset
100	Toggle	
101	Reset	
110	Toggle	Set
111	Reset	Set

How modes work (up mode example)



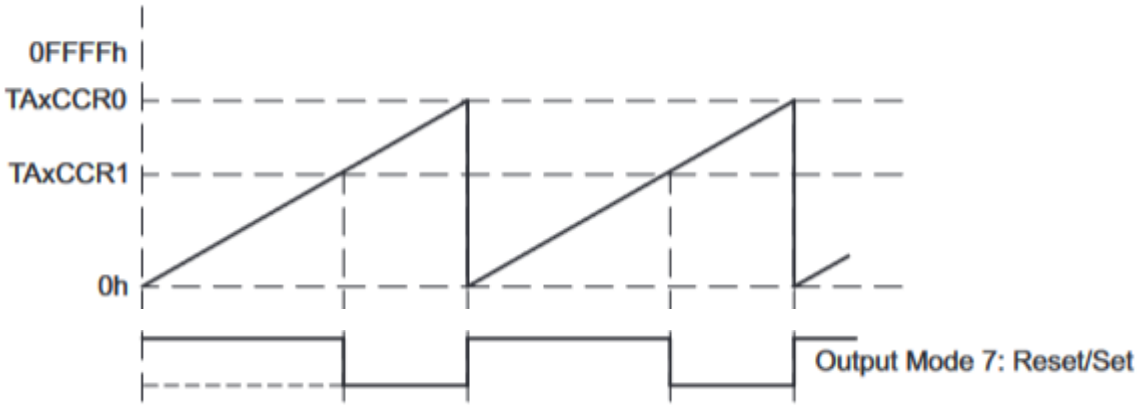
OUTMOD	On match to TAxCCRy	On match to TAxCCR0
000	OUT bit value	
001	Set	
010	Toggle	Reset
011	Set	Reset
100	Toggle	
101	Reset	
110	Toggle	Set
111	Reset	Set

How modes work (up mode example)



OUTMOD	On match to TAxCCRy	On match to TAxCCR0
000	OUT bit value	
001	Set	
010	Toggle	Reset
011	Set	Reset
100	Toggle	
101	Reset	
110	Toggle	Set
111	Reset	Set

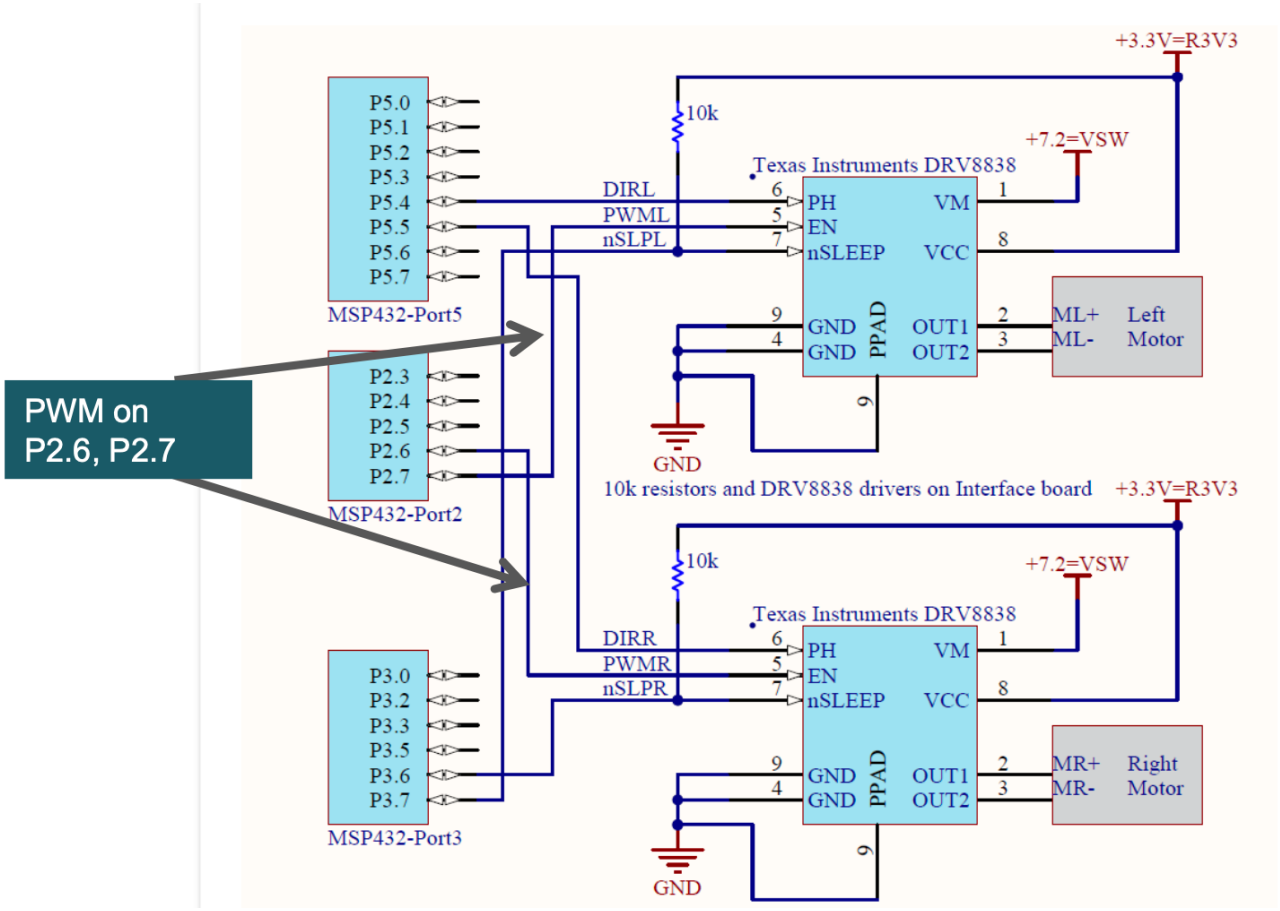
How modes work (up mode example)



OUTMOD	On match to TAxCCRy	On match to TAxCCR0
000	OUT bit value	
001	Set	
010	Toggle	Reset
011	Set	Reset
100	Toggle	
101	Reset	
110	Toggle	Set
111	Reset	Set

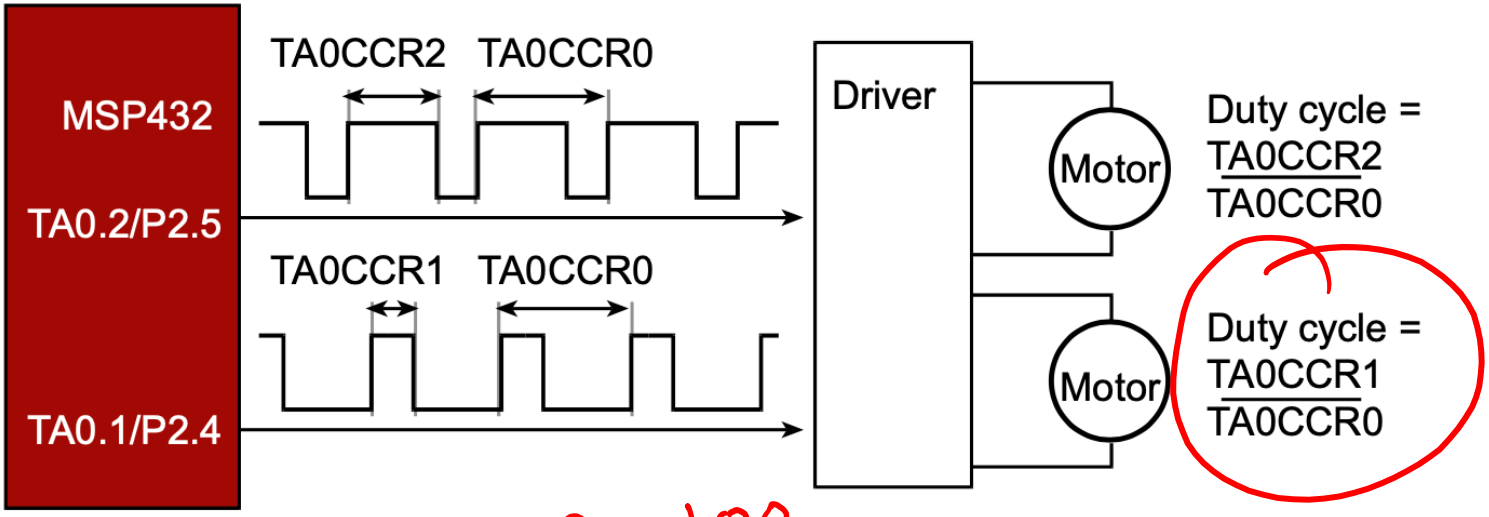
Generate Independent PWM with Timers

With different duty cycle using different timers independently (two DC motors for robot R/L wheels) with different duty cycles.



Using Timer A for Two PWM

You will convert it to
TA0.4/P2.7
TA0.3/P2.6



CCR0 = 100

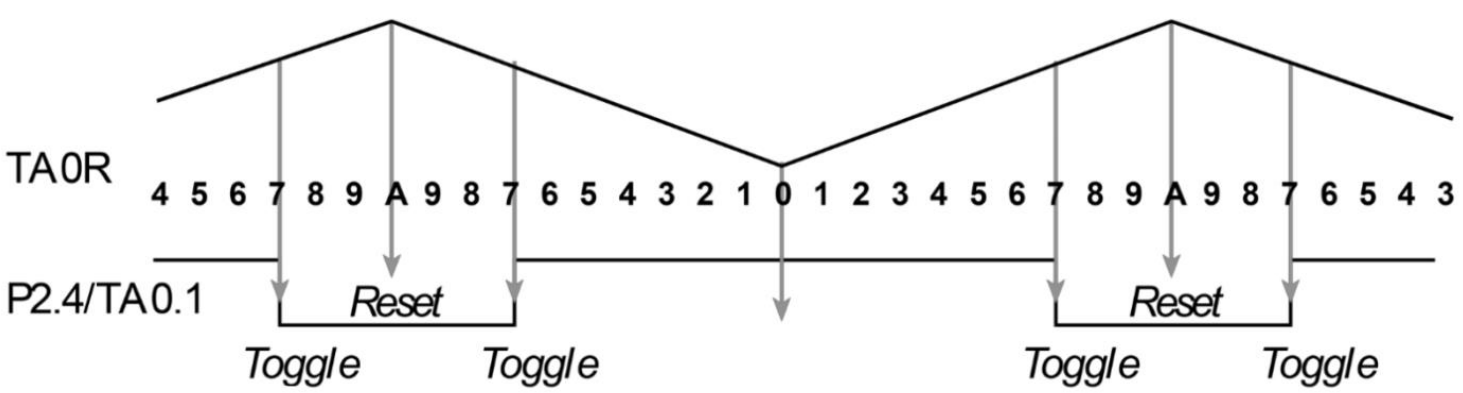
CCR 1 = 20

$\frac{20}{100} = 20\%$

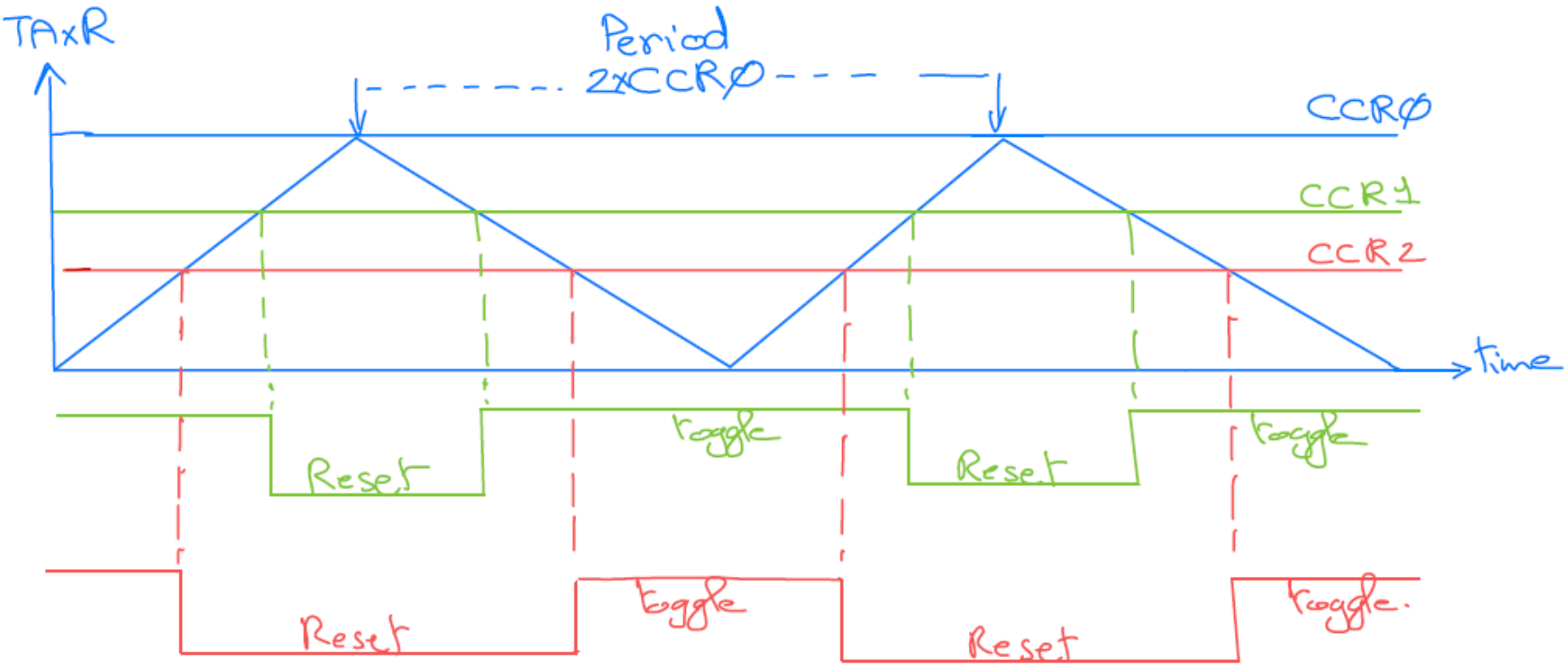
How the modes work (up/down mode example)

- **Goal:** Make a PWM with a certain duty cycle
- The main clock will go:
  - Up to CCR0
  - Down 0
  - Back up to CCR0
- P2.4 = 1 when timer equals TA0CCR1 on way down
- P2.4 = 0 when timer equals TA0CCR1 on way up
- E.g.: CCR0 = 10, CCR1 = 7 → duty cycle 7/10

T = Prescale/12MHz
Period = 2*T*CCR0
DutyCycle = CCR1/CCR0



Example of varying duty cycles



$$\text{Duty cycle 1} = \frac{CCR1}{CCR0}$$

$$\text{Duty cycle 2} = \frac{CCR2}{CCR0}$$

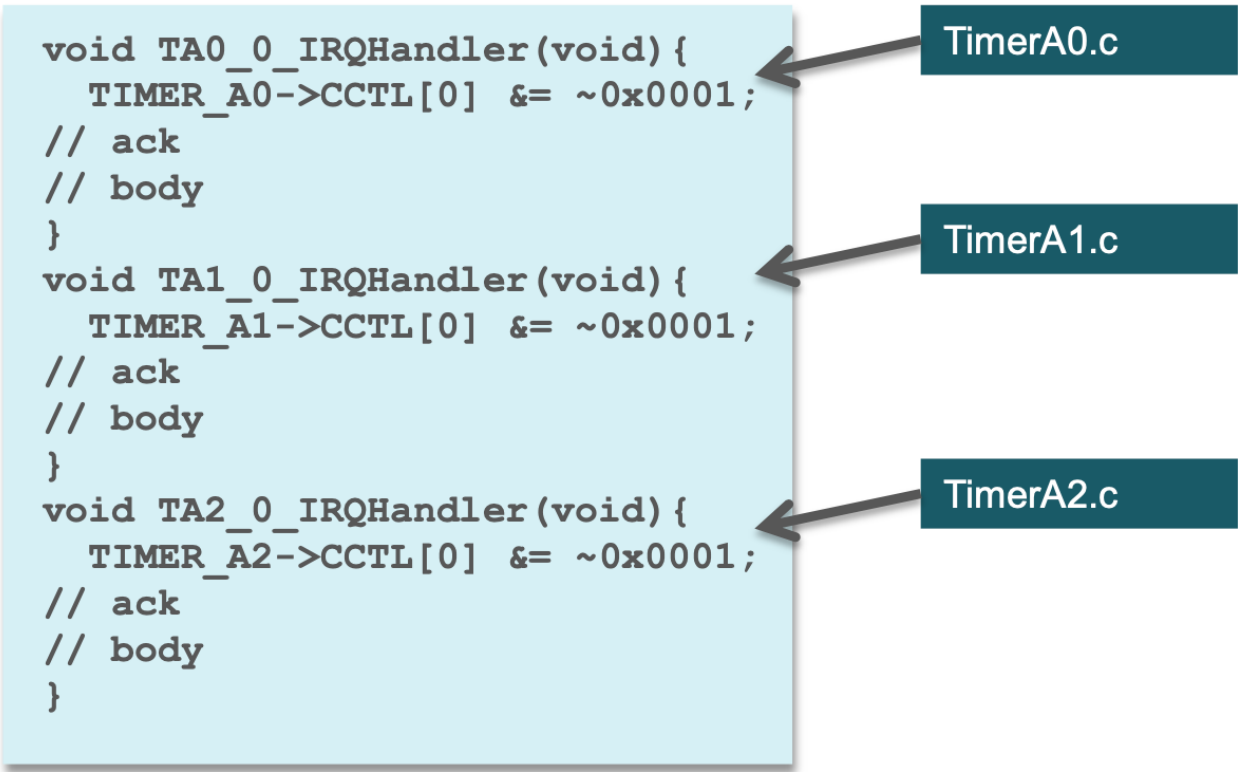
Nested vector interrupt control (NVIC)

Vector	Number	IRQ	ISR name	NVIC priority	Priority
0x0000002C	11	-5	SVC_Handler	SCB_SHPR2	31 – 29
0x00000038	14	-2	PendSV_Handler	SCB_SHPR3	23 – 21
0x0000003C	15	-1	SysTick_Handler	SCB_SHPR3	31 – 29
0x00000060	24	8	TA0_0_IRQHandler	NVIC_IPR2	7 – 5
0x00000064	25	9	TA0_N_IRQHandler	NVIC_IPR2	15 – 13
0x00000068	26	10	TA1_0_IRQHandler	NVIC_IPR2	23 – 21
0x0000006C	27	11	TA1_N_IRQHandler	NVIC_IPR2	31 – 29
0x00000070	28	12	TA2_0_IRQHandler	NVIC_IPR3	7 – 5
0x00000074	29	13	TA2_N_IRQHandler	NVIC_IPR3	
0x00000078	30	14	TA3_0_IRQHandler	NVIC_IPR3	
0x0000007C	31	15	TA3_N_IRQHandler	NVIC_IPR3	
0x00000080	32	16	EUSCIA0_IRQHandler	NVIC_IPR4	
0x00000084	33	17	EUSCIA1_IRQHandler	NVIC_IPR4	
0x00000088	34	18	EUSCIA2_IRQHandler	NVIC_IPR4	
0x0000008C	35	19	EUSCIA3_IRQHandler	NVIC_IPR4	
0x00000090	36	20	EUSCIB0_IRQHandler	NVIC_IPR5	
0x00000094	37	21	EUSCIB1_IRQHandler	NVIC_IPR5	
0x00000098	38	22	EUSCIB2_IRQHandler	NVIC_IPR5	23 – 21
0x0000009C	39	23	EUSCIB3_IRQHandler	NVIC_IPR5	31 – 29
0x000000CC	51	35	PORT1_IRQHandler	NVIC_IPR8	31 – 29
0x000000D0	52	36	PORT2_IRQHandler	NVIC_IPR9	7 – 5
0x000000D4	53	37	PORT3_IRQHandler	NVIC_IPR9	15 – 13
0x000000D8	54	38	PORT4_IRQHandler	NVIC_IPR9	23 – 21
0x000000DC	55	39	PORT5_IRQHandler	NVIC_IPR9	31 – 29
0x000000E0	56	40	PORT6_IRQHandler	NVIC_IPR10	7 – 5

```
void TA2_0_IRQHandler(void){
  TIMER_A2->CCTL[0] &= ~0x0001; // ack
// body
}
```

Timer Interrupts with Tasks

```
void TA0_0_IRQHandler(void){
  TIMER_A0->CCTL[0] &= ~0x0001;
// ack
// body
}
void TA1_0_IRQHandler(void){
  TIMER_A1->CCTL[0] &= ~0x0001;
// ack
// body
}
void TA2_0_IRQHandler(void){
  TIMER_A2->CCTL[0] &= ~0x0001;
// ack
// body
}
```



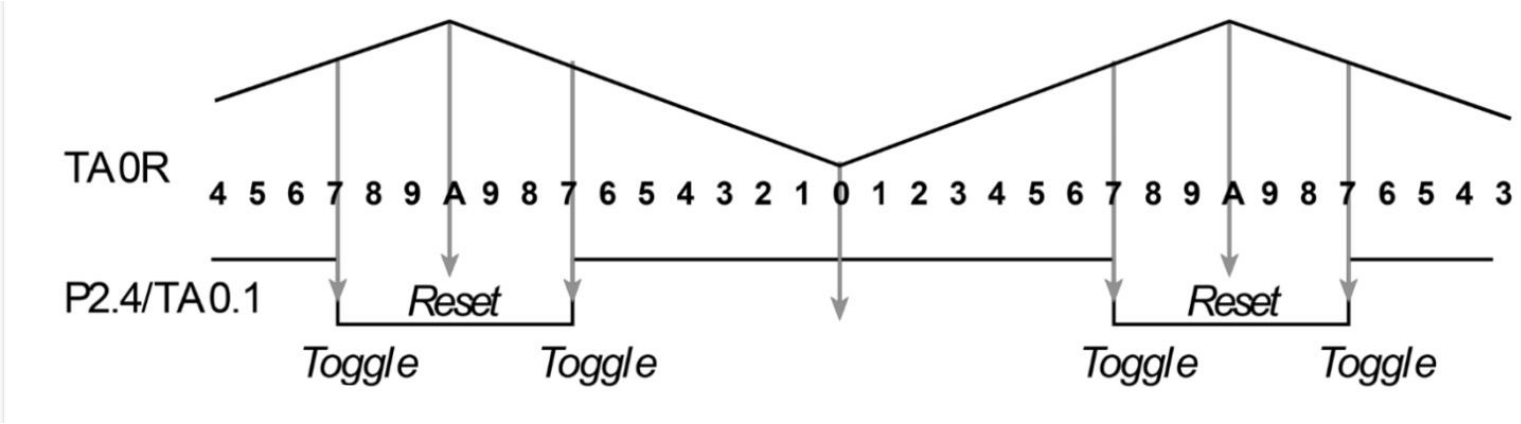
Periodic Interrupt Initialization Example

Using Timer A for Two PWM Outputs

- **PWM Mode:**
  - Up to CCR0
  - Down 0

T = Prescale/12MHz
Period = 2*T*CCR0
DutyCycle = CCR1/CCR0

- P2.4 = 1 when timer equals TA0CCR1 on way down
- P2.4 = 0 when timer equals TA0CCR1 on way up
- E.g.: CCR0 = 10, CCR1 = 7 → duty cycle 7/10



Set Up the PWMs

To set up the PWMs, we do the following:

1. Configure Timer_A in the up-down mode, so it counts up to TAxCCR0 and back to zero (using MC bits).

2. Store a value in TAxCCR0 so that the PWM period is twice as much: PWM period = 2 x TAxCCR0.

3. Configure the clock dividers and source clock SMCLK for TAx.

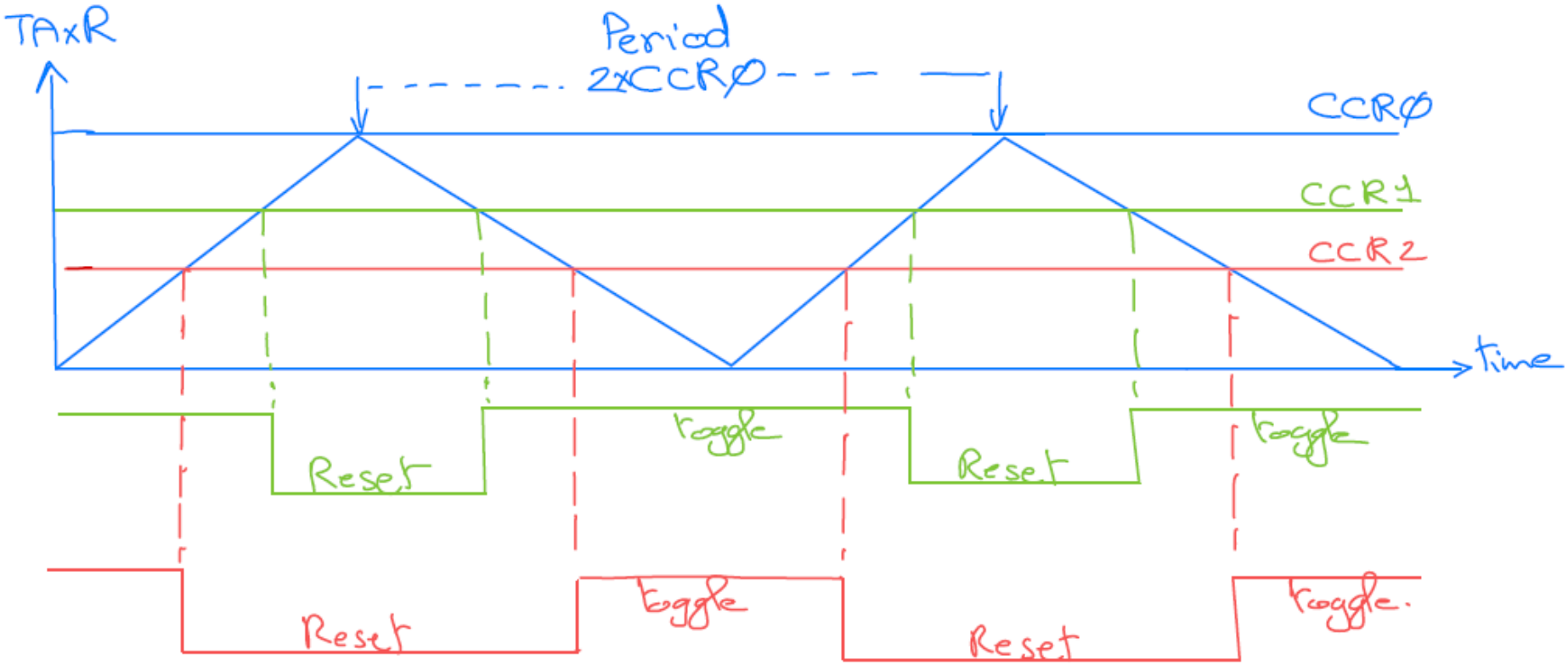
4. Configure TAx_CCR1CTL register in toggle/reset mode so that when TAxR gets up to CCR1, it resets. When it gets down to CCR1, it toggles.

5. Store a value in TAxCCR1 that gives us Duty Cycle 1 = TAxCCR1/TAxCCR0.

6. Configure TAxCCR2CTL register in toggle/reset mode so that when TAxR gets up to CCR2, it resets. When it gets down to CCR2, it toggles.

7. Store a value in TAxCCR2 that gives us Duty Cycle 2 = TAxCCR2/TAxCCR0

Note that Timer_A automatically generates the PWMs to the pins connected to TAxCCR1 and TAxCCR2 as output, i.e., we don't need interrupts to generate two PWMs!



$$\text{Duty Cycle 1} = \frac{CCR1}{CCR0}$$

$$\text{Duty Cycle 2} = \frac{CCR2}{CCR0}$$

PWM Outputs on P2.4 and P2.5

```
// SMCLK = 48MHz/4 = 12 MHz, 83.33ns
// Counter counts up to TA0CCR0 and back down
// Let Timerclock period T = 8/12MHz = 666.7ns
// Period of P2.4 is period*1.333us, duty cycle is duty1/period
// Period of P2.5 is period*1.333us, duty cycle is duty2/period
void PWM_Init12(uint16_t period, uint16_t duty1, uint16_t duty2){
  P2->DIR |= 0x30;          // P2.4, P2.5 output
  P2->SEL0 |= 0x30;        // P2.4, P2.5 Timer0A functions
  P2->SEL1 &= ~0x30;    // P2.4, P2.5 Timer0A functions
  TIMER_A0->CCTL[0] = 0x0080;    // CCI0 toggle
  TIMER_A0->CCR[0] = period;       // Period is 2*period*8*83.33ns is 1.333*period
  TIMER_A0->EX0 = 0x0000;          //    divide by 1
  TIMER_A0->CCTL[1] = 0x0040;    // CCR1 toggle/reset
  TIMER_A0->CCR[1] = duty1;        // CCR1 duty cycle is duty1/period
  TIMER_A0->CCTL[2] = 0x0040;   // CCR2 toggle/reset
  TIMER_A0->CCR[2] = duty2;       // CCR2 duty cycle is duty2/period
  TIMER_A0->CTL = 0x02F0;         // SMCLK=12MHz, divide by 8, up-down mode
// bit           mode
// 9-8  10   TASSEL, SMCLK=12MHz
// 7-6  11   ID, divide by 8
// 5-4  11   MC, up-down mode
// 2    0     TACLR, no clear
// 1    0     TAIE, no interrupt
// 0          TAIFG
}
```

PWM Outputs on P2.4 and P2.5

You will convert it to
TA0.4/P2.7
TA0.3/P2.6

```
//*****************************PWM_Duty1*********************************
// change duty cycle of PWM output on P2.4
// Inputs:  duty1
// Outputs: none
// period of P2.4 is 2*period*666.7ns, duty cycle is duty1/period
void PWM_Duty1(uint16_t duty1){
  TIMER_A0->CCR[1] = duty1;          // CCR1 duty cycle is duty1/period
}

//*****************************PWM_Duty2*********************************
// change duty cycle of PWM output on P2.5
// Inputs:  duty2
// Outputs: none// period of P2.5 is 2*period*666.7ns, duty cycle is duty2/period
void PWM_Duty2(uint16_t duty2){
  TIMER_A0->CCR[2] = duty2;          // CCR2 duty cycle is duty2/period
}
```

DutyCycle
- Precision 14999 alternatives (0 to 14998)
- Range 0 to 99.99%
- Resolution 0.0067%

15-10	9-8	7-6	5-4	3	2	1	0	Name
	TASSEL	ID	MC		TACLR	TAIE	TAIFG	TA0CTL

15-14	13-12	11	10	9	8	7-5	4	3	2	1	0	
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL0
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL1
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL2
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL3
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL4

15–0	
16-bit counter	TA0R
16-bit Capture/Compare 0 Register	TA0CCR0
16-bit Capture/Compare 1 Register	TA0CCR1
16-bit Capture/Compare 2 Register	TA0CCR2
16-bit Capture/Compare 3 Register	TA0CCR3
16-bit Capture/Compare 4 Register	TA0CCR4

15-3	2-0	
	TAIDEX	TA0EX0

15-0	
TAIV	TA0IV

Periodic Interrupt Initialization Example

```
void (*TimerA2Task)(void);   // user function
void TimerA2_Init(void(*task)(void), uint16_t period){
  TimerA2Task = task;             // user function
  // bits9-8=10,       clock source to SMCLK
  // bits7-6=10,       input clock divider /4
  // bits5-4=00,       stop mode
  // bit2=0,           set this bit to clear
  // bit1=0,           no interrupt on timer
  TIMER_A2->CTL = 0x0280;
  // bits15-14=00,     no capture mode
  // bit8=0,           compare mode
  // bit4=1,           enable capture/compare interrupt on CCIFG
  // bit2=0,           output this value in output mode 0
  // bit0=0,           clear capture/compare interrupt pending
  TIMER_A2->CCTL[0] = 0x0010;
  TIMER_A2->CCR[0] = (period - 1);    // compare match value
  TIMER_A2->EX0 = 0x0005;     // configure for input clock divider /6
  NVIC->IP[3] = (NVIC->IP[3]&0xFFFFFF00)|0x00000040; // priority 2
  NVIC->ISER[0] = 0x00001000;   // enable interrupt 12 in NVIC
  TIMER_A2->CTL |= 0x0014;      // reset and start timer in up mode
}
```

Interrupts enabled in the main program after all devices initialized

$T * 2^{ID} * (\mathbf{TAIDEX}+1)(\mathbf{TAxCCR0}+1) = 2\mu s * period$

```
void TA2_0_IRQHandler(void){
  TIMER_A2->CCTL[0] &= ~0x0001;
  *TimerA2Task();
}
```

CORRECTIONS

	15-10	9-8	7-6	5-4	3	2	1	0	Name
0.0000		TASSEL	ID	MC		TACLR	TAIE	TAIFG	TA0CTL

	15-14	13-12	11	10	9	8	7-5	4	3	2	1	0	
0.0002	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL0
0.0004	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL1
0.0006	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL2
0.0008	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL3
0.000A	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL4
0.000C	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL5
0.000E	CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL6

	15–0	
0.0010	16-bit counter	TA0R
0.0012	16-bit Capture/Compare 0 Register	TA0CCR0
0.0014	16-bit Capture/Compare 1 Register	TA0CCR1
0.0016	16-bit Capture/Compare 2 Register	TA0CCR2
0.0018	16-bit Capture/Compare 3 Register	TA0CCR3
0.001A	16-bit Capture/Compare 4 Register	TA0CCR4
0.001C	16-bit Capture/Compare 5 Register	TA0CCR5
0.001E	16-bit Capture/Compare 6 Register	TA0CCR6

	15-3	2-0	
0.0020		TAIDEX	TA0EX0

	15-0	
0.002E	TAIV	TA0IV

Correct

15-10	9-8	7-6	5-4	3	2	1	0	Name
	TASSEL	ID	MC		TACLR	TAIE	TAIFG	TA0CTL

15-14	13-12	11	10	9	8	7-5	4	3	2	1	0	
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL0
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL1
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL2
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL3
CM	CCIS	SCS	SCCI		CAP	OUTMOD	CCIE	CCI	OUT	COV	CCIFG	TA0CCTL4

15–0	
16-bit counter	TA0R
16-bit Capture/Compare 0 Register	TA0CCR0
16-bit Capture/Compare 1 Register	TA0CCR1
16-bit Capture/Compare 2 Register	TA0CCR2
16-bit Capture/Compare 3 Register	TA0CCR3
16-bit Capture/Compare 4 Register	TA0CCR4

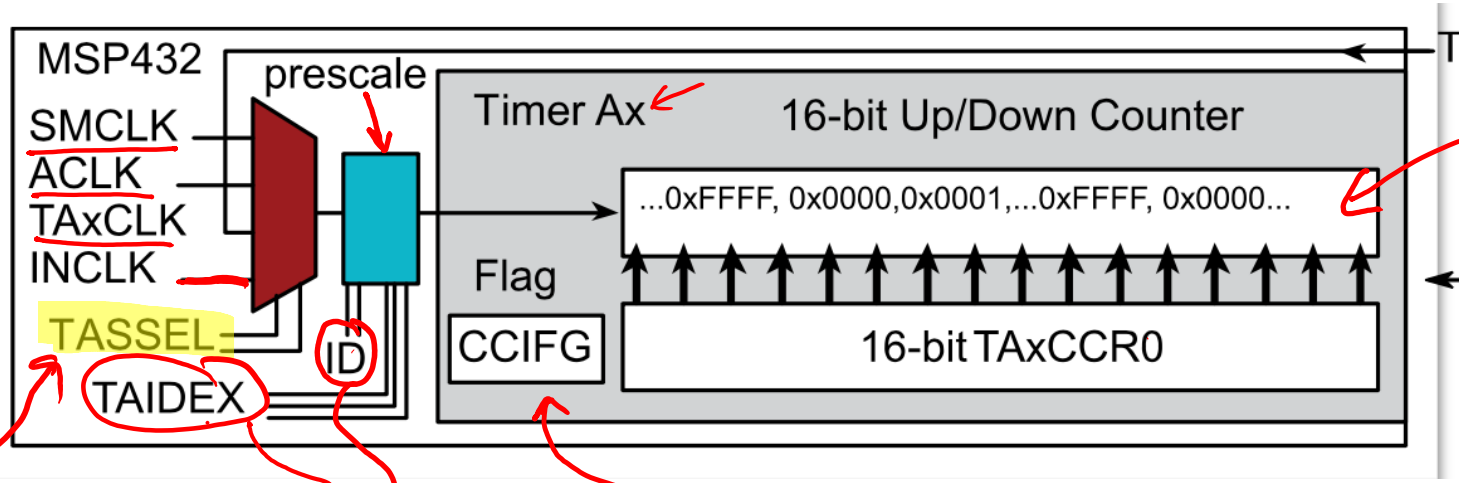
15-3	2-0	
	TAIDEX	TA0EX0

15-0	
TAIV	TA0IV

Incorrect

Timers FMS

Timer A Configuration



One module TAx

one true counting register

Selects timer source

Input divider expansion?

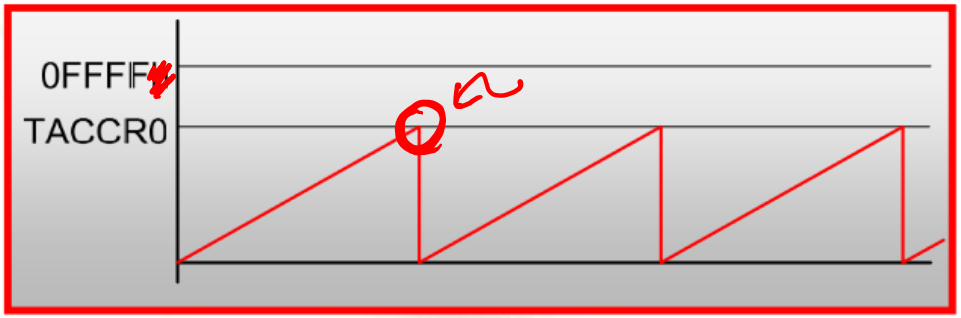
Interrupt flag

x → module

y → submodule

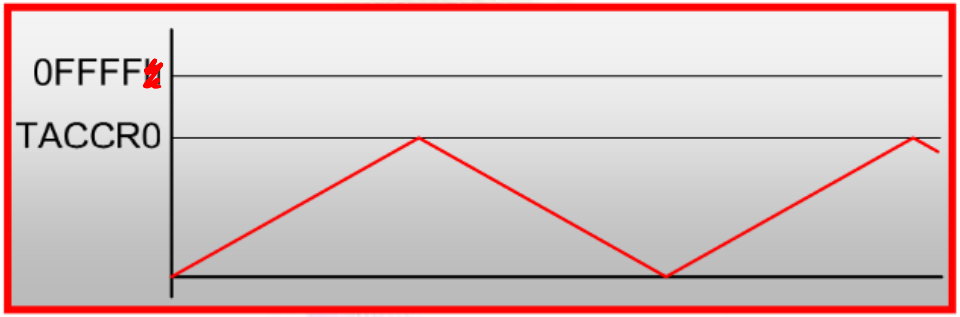
Timer A: Count modes

Up



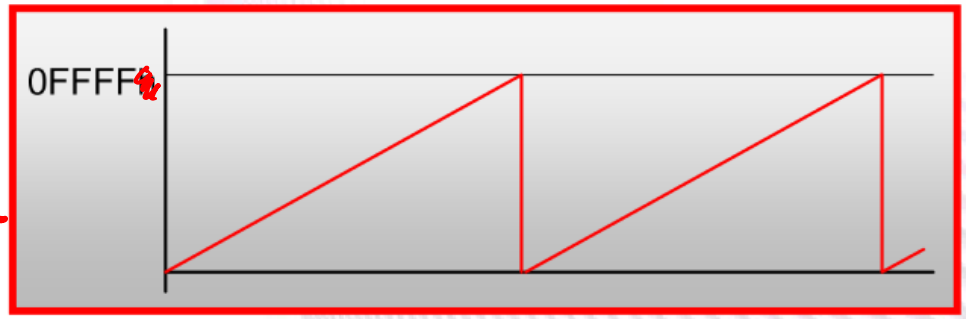
Up to FFFF, rolls over to 0000, back up to FFFF, etc.

Up/ Down



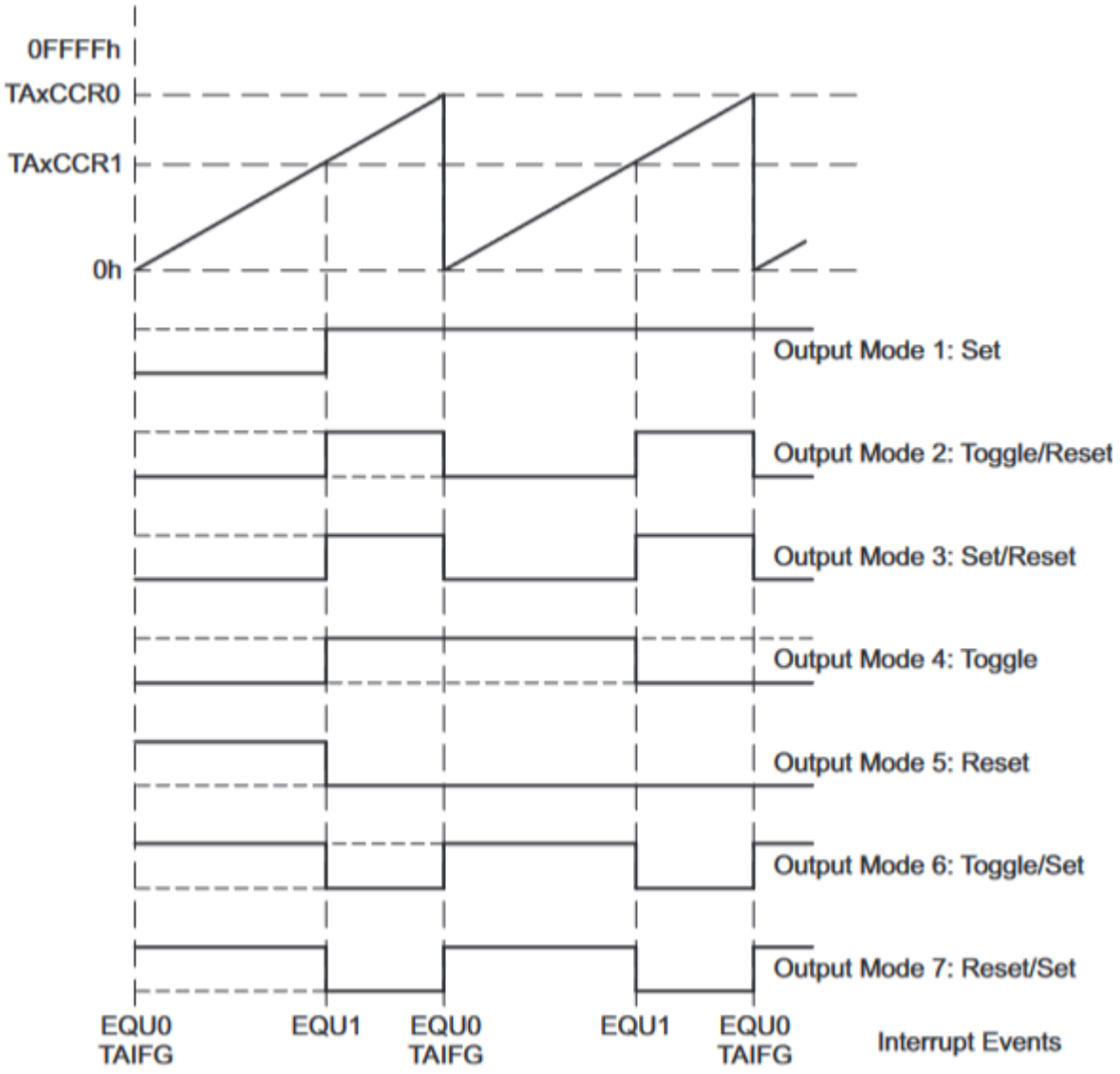
Up to value in CCR0, count down to 0000, back up to value in CCR0, etc.

Continuous



Up to value specified by CCR0, rolls over to 0000, back up to CCR0 value, etc.

https://www.ti.com/lit/ml/slap113/slap113.pdf


0FFFFh
TAxCCR0
TAxCCR1
0h
Output Mode 1: Set
Output Mode 2: Toggle/Reset
Output Mode 3: Set/Reset
Output Mode 4: Toggle
Output Mode 5: Reset
Output Mode 6: Toggle/Set
Output Mode 7: Reset/Set
EQU0 TAIFG
EQU1
EQU0 TAIFG
EQU1
EQU0 TAIFG
Interrupt Events