EEL 4742 – Embedded Systems
Module 10 – Advanced Timers

Hadi M Kamali

Department of Electrical and Computer Engineering (**ECE**)
University of Central Florida

Office Location/phone: HEC435 – (407) 823-0764
webpage: https://www.ece.ucf.edu/~kamali/
e-mail: kamali@ucf.edu

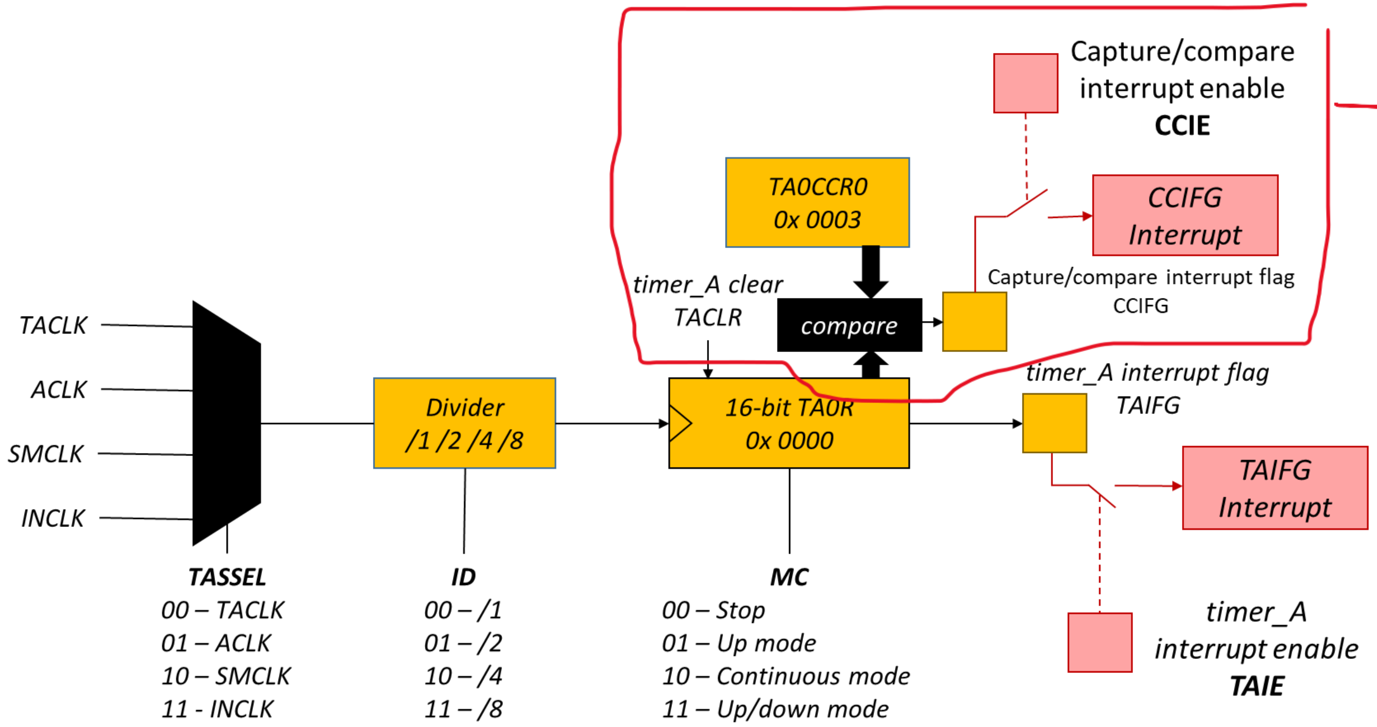
HAVEN Research Group
https://haven.ece.ucf.edu/

ARDWARE ERIFICATION ENGINEERING SSURANCE

UNIVERSITY OF CENTRAL FLORIDA

Multiple Channels of Timer_A

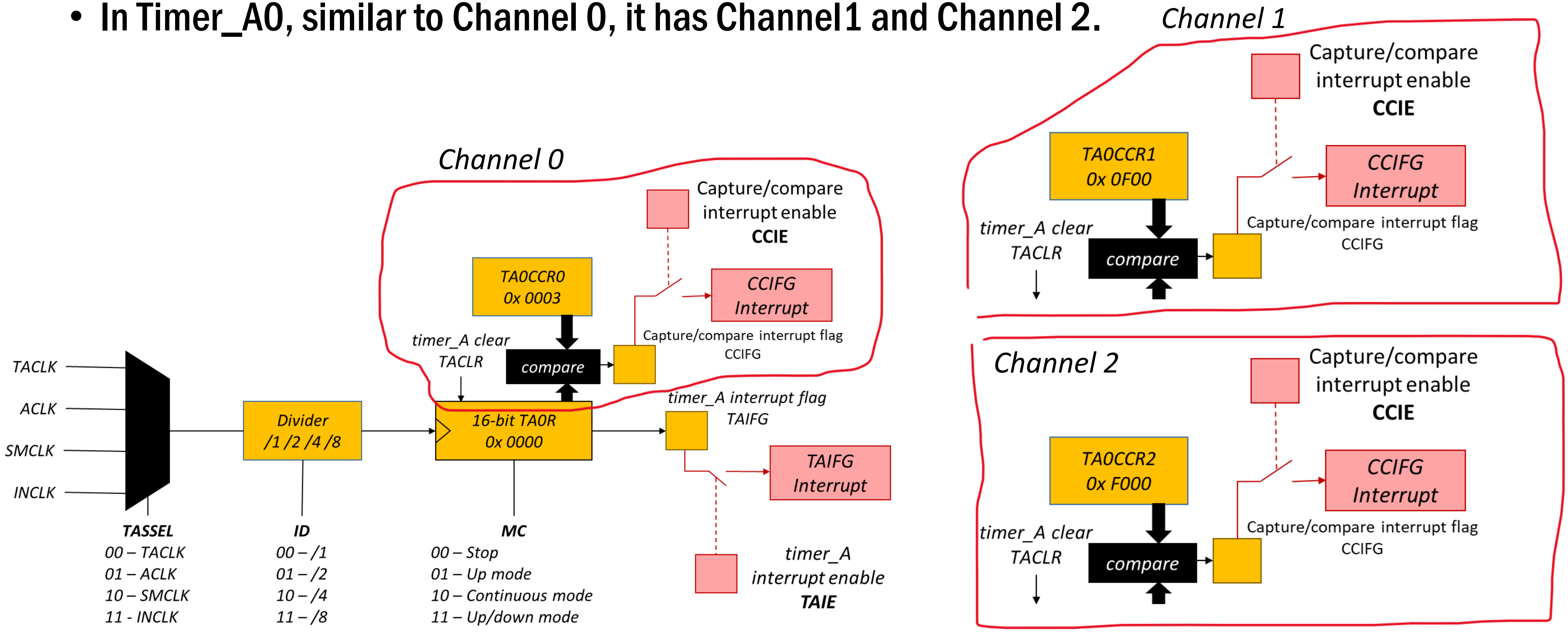
- For Timer_A
  - In Timer_A0, similar to Channel 0, it has Channel1 and Channel 2.



For the first channel of CCR0.

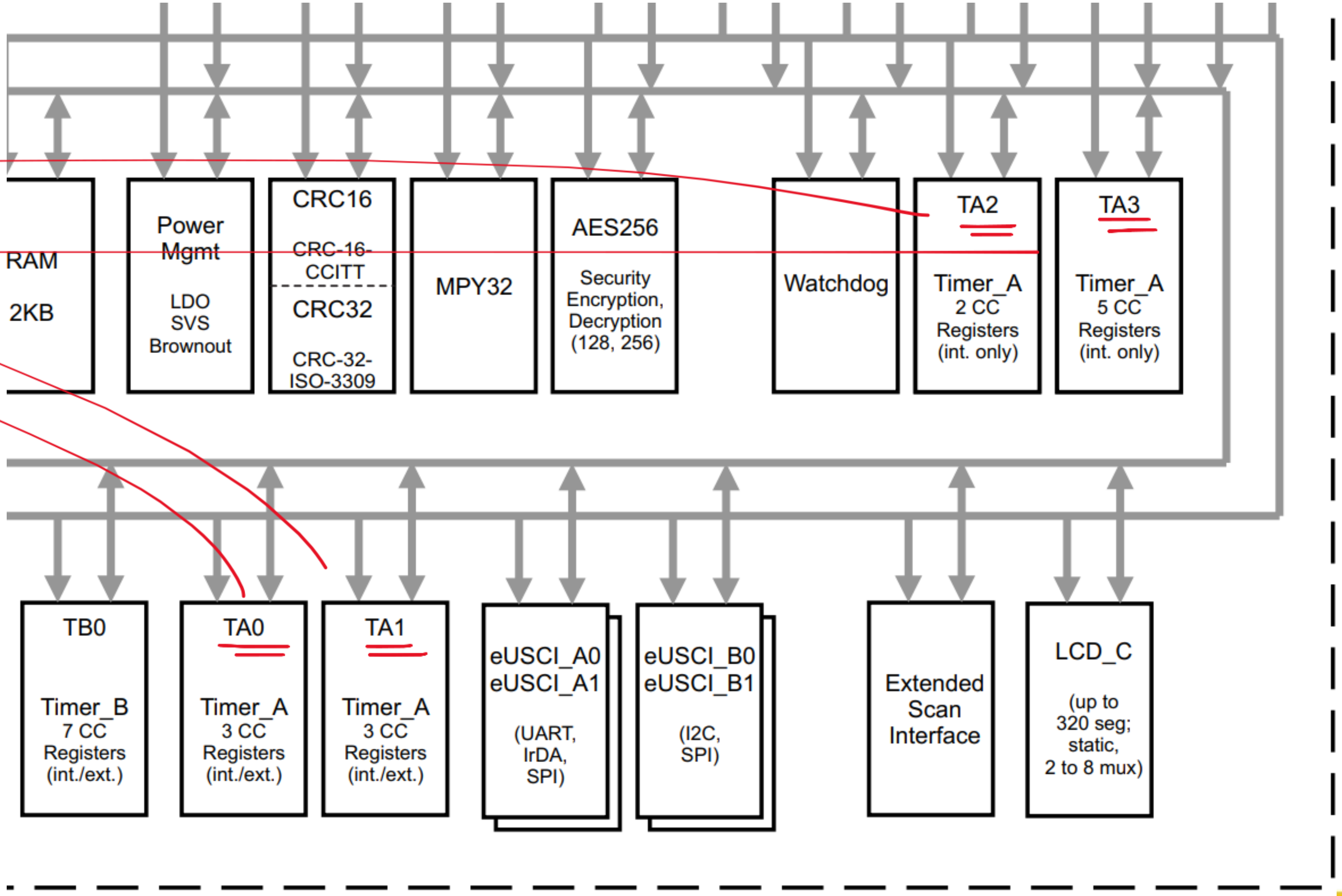
Multiple Channels of Timer_A

- For Timer_A
  - In Timer_A0, similar to Channel 0, it has Channel1 and Channel 2.



Multiple Channels of Timer_A

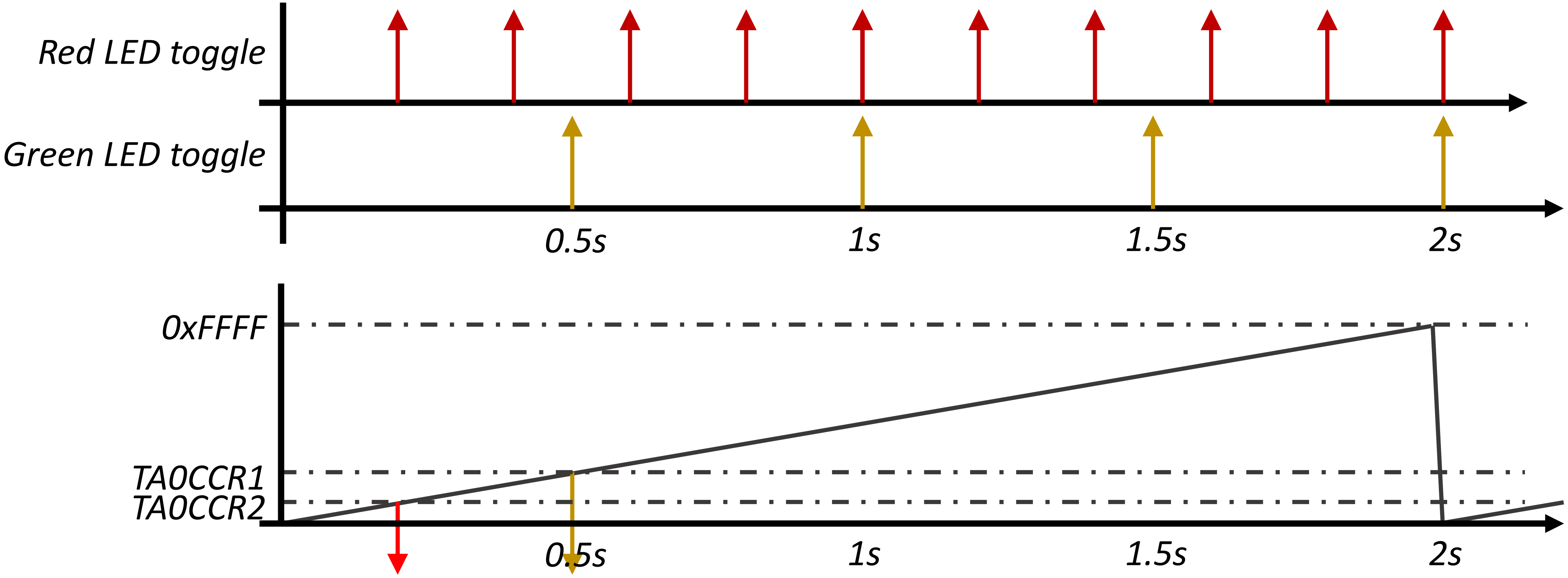
- **Four different timer modules in MSP430**
  - *TA0 (3 Channels)*
  - *TA1 (3 Channels)*
  - *TA2 (2 Channels)*
  - *TA3 (5 Channels)*



Multiple Channels of Timer_A

- Per each module → using multiple channels "simultaneously"

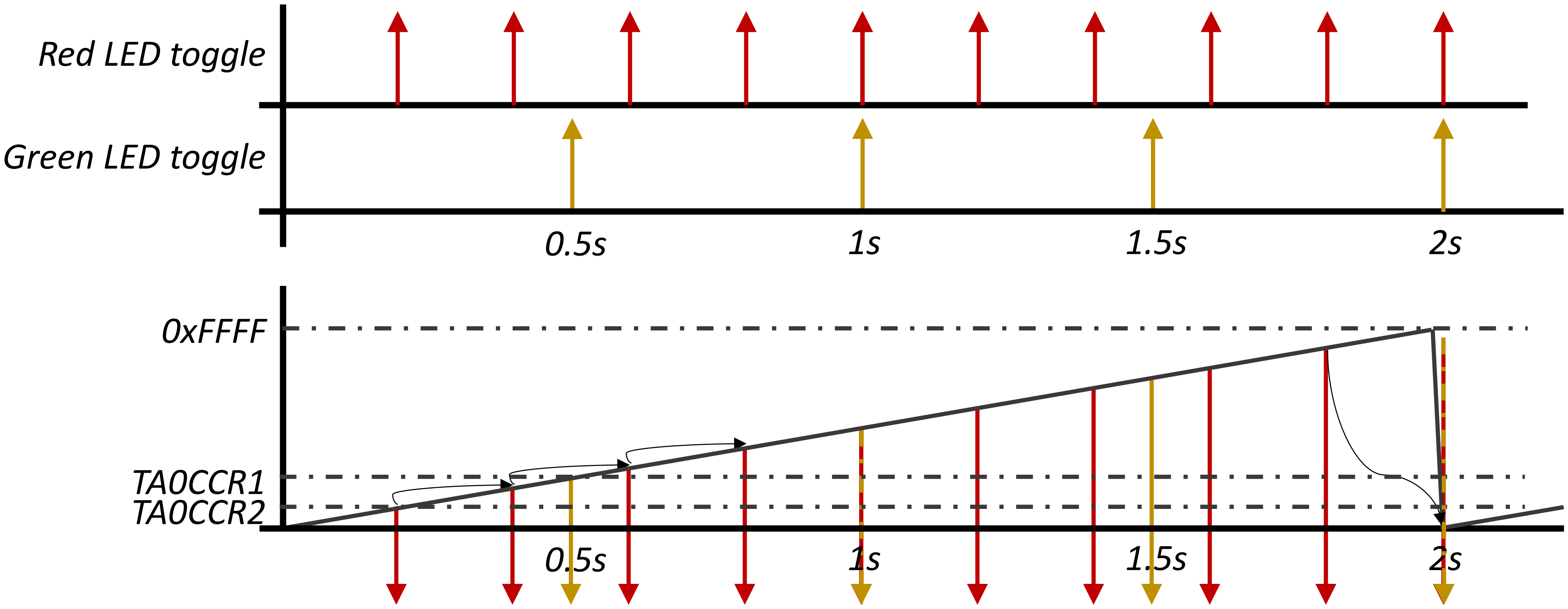
Application: *Toggle green LED every 0.5 second and*
Toggle red LED every 0.2 second



Multiple Channels of Timer_A

- Per each module → using multiple channels "simultaneously"

Application: *Toggle <u>green LED every 0.5 second</u> and*
Toggle <u>red LED every 0.2 second</u>



Multiple Channels of Timer_A

- Counting mode is <u>**continuous**</u>

Application: *Toggle <u>green LED every 0.5 second</u> and*
Toggle <u>red LED every 0.2 second</u>

To time simultaneous intervals with multiple channels, the timer module is operated in the ***continuous mode****, in which TAR counts from 0 up to 65,535 (64K).*

The channels schedule their interrupts by looking ahead from the current value of TAR

2 seconds = 65536
0.2 seconds = 65536/2 * 0.2 = 6553.6 ≈ 6554

```
main(){
    // Configure LED pins
    // Configure timer_A0

    TA0CCR1 = 6554 – 1;

    // Low power mode 3
    for(;;);
}
```

```
Channel_1_ISR(){
    // Clear flag
    // Toggle red LED
    TA0CCR1 = TA0R + (6554 – 1);
}
```

After the first overflow, the CCR is defined w.r.t. current TA0R.

rolls back to zero and counts up to desired value

Multiple Channels of Timer_A

- Counting mode is **<u>continuous</u>**

TA0R	TA0CCR1
0	*6553*
6553	*6553 + 6553 = 13106*
13106	*13106 + 6553 = 19659*
19659	*19659 + 6553 = 26212*
…	…
52424	*52424 + 6553 = 58977*
58977	*58977 + 6553 = 65530*
65530	*65530 + 6553 = 72083*
…	…

Application: *Toggle green LED every 0.5 second and*
Toggle red LED every 0.2 second

ith multiple channels, the timer module is operated in the
counts from 0 up to 65,535 (64K).

rrupts by looking ahead from the current value of TAR

= 65536
= 65536/2 * 0.2 = 6553.6 ≈ 6554

```
main(){
    // Configure LED pins
    // Configure timer_A0

    TA0CCR1 = 6554 – 1;

    // Low power mode 3
    for(;;);
}
```

```
Channel_1_ISR(){
    // Clear flag
    // Toggle red LED
    TA0CCR1 = TA0R + (6554 – 1);
}
```

After the first overflow, the CCR is defined w.r.t. current TA0R.

rolls back to zero and counts up to desired value

Multiple Channels of Timer_A

- Counting mode is <u>continuous</u>

TA0R	TA0CCR1
0	*6553*
6553	*6553 + 6553 = 13106*
13106	*13106 + 6553 = 19659*
19659	*19659 + 6553 = 26212*
…	…
52424	*52424 + 6553 = 58977*
58977	*58977 + 6553 = 65530*
65530	*65530 + 6553* *= (72083 % 65536) = 6547*
6547	*6547 + 6553 = 13100*
13100	*13100 + 6553 = 19653*
…	…

rolls back to zero and counts up to desired value

Application: *Toggle <u>green LED every 0.5 second</u> and*
Toggle <u>red LED every 0.2 second</u>

ith multiple channels, the timer module is operated in the
counts from 0 up to 65,535 (64K).

rrupts by looking ahead from the current value of TAR

= 65536
= 65536/2 * 0.2 = 6553.6 ≈ 6554

```
main(){
    // Configure LED pins
    // Configure timer_A0

    TA0CCR1 = 6554 – 1;

    // Low power mode 3
    for(;;);
}
```

```
Channel_1_ISR(){
    // Clear flag
    // Toggle red LED
    TA0CCR1 = TA0R + (6554 – 1);
}
```

After the first overflow, the CCR is defined w.r.t. current TA0R.

Multiple Channels of Timer_A

- Counting mode is <u>**continuous**</u>

Application: *Toggle green LED every 0.5 second and Toggle red LED every 0.2 second*

To time simultaneous intervals with multiple channels, the timer module is operated in the ***continuous mode****, in which TAR counts from 0 up to 65,535 (64K).*

The channels schedule their interrupts by looking ahead from the current value of TAR

2 seconds	***= 65536***
0.2 seconds	***= 65536/2 * 0.2 = 6553.6 ≈ 6554***
0.5 seconds	***= 65536/2 * 0.5 = 16384***

```
main(){
    // Configure LED pins
    // Configure timer_A0

    TA0CCR1 = 6554 – 1;
    TA0CCR2 = 16384 – 1;
    // Low power mode 3
    for(;;);
}
```

```
Channel_2_ISR(){
        // Clear flag
        // Toggle red LED
        TA0CCR2 = TA0R + (16384 – 1);
}
```

rolls back to zero and counts up to desired value

```
Channel_1_ISR(){
        // Clear flag
        // Toggle red LED
        TA0CCR1 = TA0R + (6554 – 1);
}
```

After the first overflow, the CCR is defined w.r.t. current TA0R.

Multiple Channels of Timer_A

- **Interrupts per channel**

Event	Interrupt enable	Interrupt flag	Interrupt vector
TA0R equals **TA0CCR0**	CCIE in **TA0CCTL0**	CCIFG in **TA0CCTL0**	TIMER0_A0_VECTOR
TA0R register rolls back to 0x0000	TAIE in **TA0CTL**	TAIFG in **TA0CTL**	TIMER0_A1_VECTOR

For the #pragma used for defining ISR(). There are TWO different handler for each interrupt flag (One for TAIFG and one for CCIFG)!

Multiple Channels of Timer_A

- Interrupts per channel

Event	Interrupt enable	Interrupt flag	Interrupt vector
TA0R equals TA0CCR0	CCIE in TA0CCTL0	CCIFG in TA0CCTL0	TIMER0_A0_VECTOR
TA0R register rolls back to 0x0000	TAIE in **TA0CTL**	TAIFG in **TA0CTL**	**TIMER0_A1_VECTOR**

Event	Interrupt enable	Interrupt flag	Interrupt vector
TA0R equals TA0CCR1	CCIE in TA0CCTL1	CCIFG in TA0CCTL1	**TIMER0_A1_VECTOR**
TA0R register can roll back to 0x0000	TAIE in **TA0CTL**	TAIFG in **TA0CTL**	**TIMER0_A1_VECTOR**

Event	Interrupt enable	Interrupt flag	Interrupt vector
TA0R equals TA0CCR2	CCIE in TA0CCTL2	CCIFG in TA0CCTL2	**TIMER0_A1_VECTOR**
TA0R register can roll back to 0x0000	TAIE in **TA0CTL**	TAIFG in **TA0CTL**	**TIMER0_A1_VECTOR**

Shared ISR (one function for all)

Multiple Channels of Timer_A

- Interrupts per channel

Event	Interrupt enable	Interrupt flag	Interrupt vector
TA0R equals **TA0CCR0**	CCIE in **TA0CCTL0**	CCIFG in **TA0CCTL0**	TIMER0_A0_VECTOR
TA0R register rolls back to 0x0000	TAIE in **TA0CTL**	TAIFG in **TA0CTL**	**TIMER0_A1_VECTOR**

Event	Interrupt enable	Interrupt flag	Interrupt vector
TA0R equals **TA0CCR1**	CCIE in **TA0CCTL1**	CCIFG in **TA0CCTL1**	**TIMER0_A1_VECTOR**
TA0R register can roll back to 0x0000	TAIE in **TA0CTL**	TAIFG in **TA0CTL**	**TIMER0_A1_VECTOR**

Event	Interrupt enable	Interrupt flag	Interrupt vector
TA0R equals **TA0CCR2**	CCIE in **TA0CCTL2**	CCIFG in **TA0CCTL2**	**TIMER0_A1_VECTOR**
TA0R register can roll back to 0x0000	TAIE in **TA0CTL**	TAIFG in **TA0CTL**	**TIMER0_A1_VECTOR**

The flag and enabler are separated. So, each can be used separately (but with the same ISR).

Shared ISR (one function for all)

Multiple Channels of Timer_A

- Example:
  - one LED (blinking) is based on channel 0 with periodic interrupt every 0.1 seconds
  - another LED (blinking) is based on channel 1 with periodic interrupt every 0.5 seconds.

```
#include <msp430fr6989.h>
#define redLED BIT0 // Red at P1.0
#define greenLED BIT7 // Green at P9.7
void main(void) {
        WDTCTL = WDTPW | WDTHOLD; // Stop WDT
        PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
        P1DIR |= redLED;
        P9DIR |= greenLED;
        P1OUT &= ~redLED;
        P9OUT &= ~greenLED;

        // Configure Channel 0
        TA0CCR0 = 3277
        TA0CCTL0 |= CCIE;
        TA0CCTL0 &= ~CCIFG;

        // Configure Channel 1
        TA0CCR1 = 16384
        TA0CCTL1 |= CCIE;
        TA0CCTL1 &= ~CCIFG;

        // Start the timer (any divider?)
        TA0CTL = TASSEL_1 | ID_0 | MC_2;
        // Enable the interrupts
        _enable_interrupts();
        return;
}
```

```
// ISR of Channel 0 (A0 vector)
#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR() {
        P1OUT ^= redLED; // Toggle the red LED
        TA0CCR0 += 3277; // Schedule the next interrupt
        // Hardware clears Channel 0 flag (CCIFG in TA0CCTL0)
}

// ISR of Channel 1 (A1 vector)
#pragma vector = TIMER0_A1_VECTOR
__interrupt void T0A1_ISR() {
        P9OUT ^= greenLED; // Toggle the red LED
        TA0CCR1 += 16384; // Schedule the next interrupt
}
```

Multiple Channels of Timer_A

- **What is this code for?**

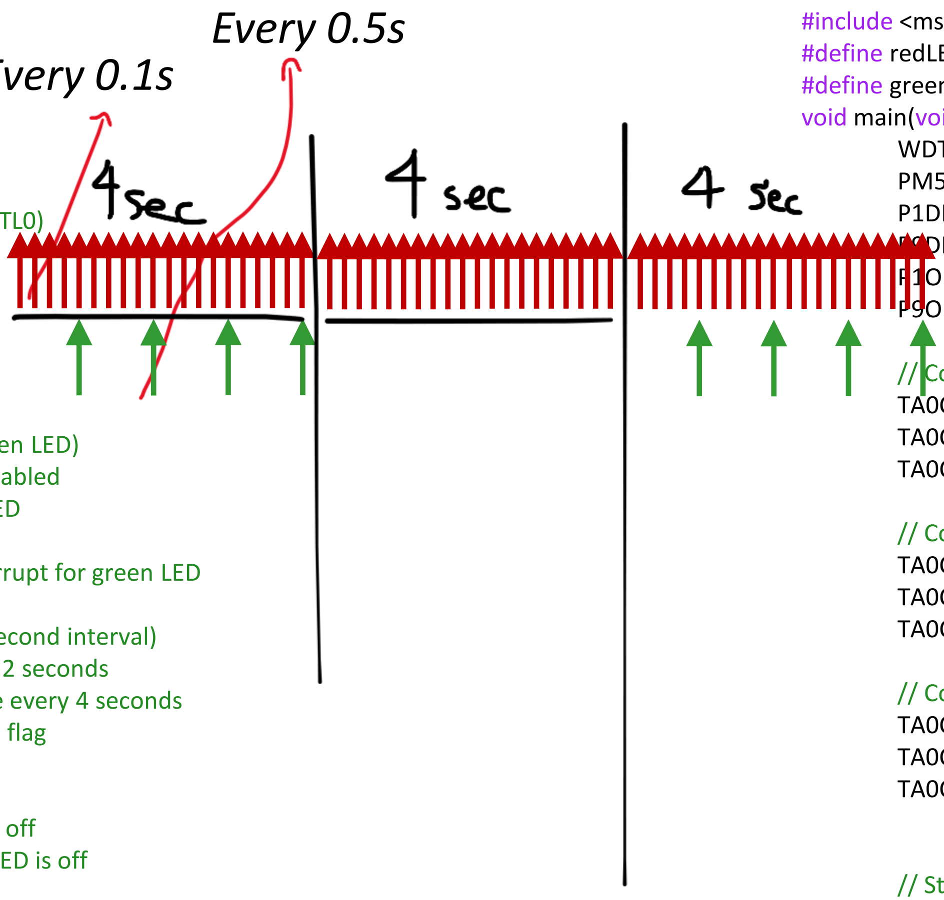
```
// ISR of Channel 0 (A0 vector)
#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR() {
            P1OUT ˆ= redLED; // Toggle the red LED
            TA0CCR0 += 3277; // Schedule the next interrupt
            // Hardware clears Channel 0 flag (CCIFG in TA0CCTL0)
}

// ISR of Channel 1 (A1 vector)
#pragma vector = TIMER0_A1_VECTOR
__interrupt void T0A1_ISR() {
          switch (TA0IV) {
               case TA0IV_TACCR1: // Channel 1 interrupt (green LED)
                  if (flashEnable) { // Only toggle if flashing is enabled
                      P9OUT ^= greenLED; // Toggle the green LED
                  }
                  TA0CCR1 += 16384; // Schedule the next interrupt for green LED
                  break;
               case TA0IV_TACCR2: // Channel 2 interrupt (2-second interval)
                  toggleCounter++; // Increment counter every 2 seconds
                  if (toggleCounter >= 2) { // Toggle flash enable every 4 seconds
                      flashEnable ^= 1; // Toggle the flash enable flag
                      toggleCounter = 0; // Reset counter
                      if (!flashEnable) { // If flash is disabled
                          P1OUT &= ~redLED; // Ensure red LED is off
                          P9OUT &= ~greenLED; // Ensure green LED is off
                      }
                  }
                  TA0CCR2 += 65536; // Schedule the next 2-second interrupt
                  break;
          }
}
```

```
#include <msp430fr6989.h>
#define redLED BIT0 // Red at P1.0
#define greenLED BIT7 // Green at P9.7

volatile unsigned int flashEnable = 1; // Flag to control LED flashing volatile
unsigned int toggleCounter = 0; // Counter for 4-second timing

void main(void) {
            WDTCTL = WDTPW | WDTHOLD; // Stop WDT
            PM5CTL0 &= ˜LOCKLPM5; // Enable GPIO pins
            P1DIR |= redLED;
            P9DIR |= greenLED;
            P1OUT &= ˜redLED;
            P9OUT &= ˜greenLED;

            // Configure Channel 0
            TA0CCR0 = 3277;
            TA0CCTL0 |= CCIE;
            TA0CCTL0 &= ˜CCIFG;
            // Configure Channel 1
            TA0CCR1 = 16384;
            TA0CCTL1 |= CCIE;
            TA0CCTL1 &= ˜CCIFG;
            // Configure Channel 2
            TA0CCR2 = 65536;
            TA0CCTL2 |= CCIE;
            TA0CCTL2 &= ˜CCIFG;

            // Start the timer (any divider)
            TA0CTL = TASSEL_1 | ID_0 | MC_2;
            // Enbale the interrupts
            _enable_interrupts();
            return;
}
```

Multiple Channels of Timer_A

- **What is this code for?**

```c
// ISR of Channel 0 (A0 vector)
#pragma vector = TIMER0_A0_VECTOR
__interrupt void T0A0_ISR() {
        P1OUT ^= redLED; // Toggle the red LED
        TA0CCR0 += 3277; // Schedule the next interrupt
        // Hardware clears Channel 0 flag (CCIFG in TA0CCTL0)
}

// ISR of Channel 1 (A1 vector)
#pragma vector = TIMER0_A1_VECTOR
__interrupt void T0A1_ISR() {
      switch (TA0IV) {
         case TA0IV_TACCR1: // Channel 1 interrupt (green LED)
           if (flashEnable) { // Only toggle if flashing is enabled
               P9OUT ^= greenLED; // Toggle the green LED
           }
           TA0CCR1 += 16384; // Schedule the next interrupt for green LED
           break;
         case TA0IV_TACCR2: // Channel 2 interrupt (2-second interval)
           toggleCounter++; // Increment counter every 2 seconds
           if (toggleCounter >= 2) { // Toggle flash enable every 4 seconds
               flashEnable ^= 1; // Toggle the flash enable flag
               toggleCounter = 0; // Reset counter
               if (!flashEnable) { // If flash is disabled
                   P1OUT &= ~redLED; // Ensure red LED is off
                   P9OUT &= ~greenLED; // Ensure green LED is off
               }
           }
           TA0CCR2 += 65536; // Schedule the next 2-second interrupt
           break;
      }
}
```




```c
#include <msp430fr6989.h>
#define redLED BIT0 // Red at P1.0
#define greenLED BIT7 // Green at P9.7
void main(void) {
        WDTCTL = WDTPW | WDTHOLD; // Stop WDT
        PM5CTL0 &= ~LOCKLPM5; // Enable GPIO pins
        P1DIR |= redLED;
        P9DIR |= greenLED;
        P1OUT &= ~redLED;
        P9OUT &= ~greenLED;

        // Configure Channel 0
        TA0CCR0 = 3277;
        TA0CCTL0 |= CCIE;
        TA0CCTL0 &= ~CCIFG;

        // Configure Channel 1
        TA0CCR1 = 16384;
        TA0CCTL1 |= CCIE;
        TA0CCTL1 &= ~CCIFG;

        // Configure Channel 2
        TA0CCR2 = 65536;
        TA0CCTL2 |= CCIE;
        TA0CCTL2 &= ~CCIFG;


        // Start the timer (any divider)
        TA0CTL = TASSEL_1 | ID_0 | MC_2;
        // Engage a low-power mode
        _enable_interrupts();
        return;
}
```

Pulse Width Modulation (PWM)

- PWM is a specific (oscillating) signal
  - The period is fixed.

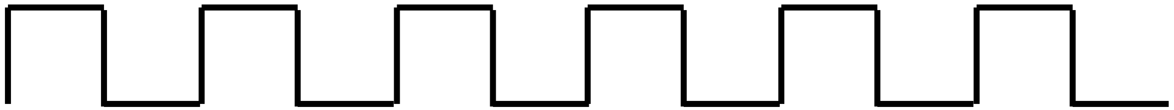
  - For each clock cycle
    - A specific period is high
    - A specific period is low

Here 50%

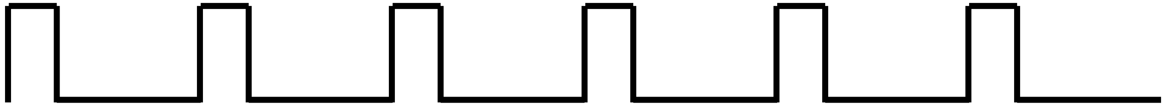
Here 50%

$$Duty\ cycle\ = \frac{t_{ON}}{t_{ON} + t_{OFF}}$$

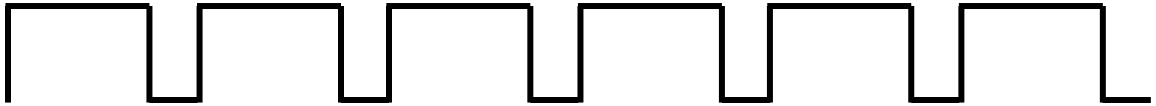
50% duty cycle



25% duty cycle



75% duty cycle



Pulse Width Modulation (PWM)

- PWM is a specific (oscillating) signal
  - The period is fixed.
  - For each clock cycle
    - A specific period is high
    - A specific period is low

Here 50%

Here 50%

We will be dimming the LED using ***pulse width modulation*** *(PWM).*

%50 ON

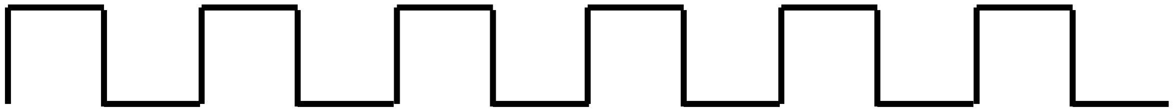
%25 ON

%75 ON

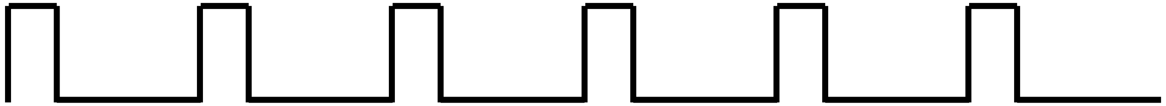
The higher the duty cycle is, The brighter the LED will be!

Connecting to LED

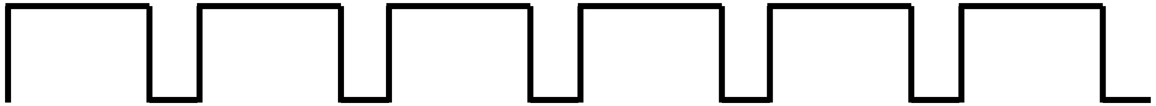
50% duty cycle



25% duty cycle

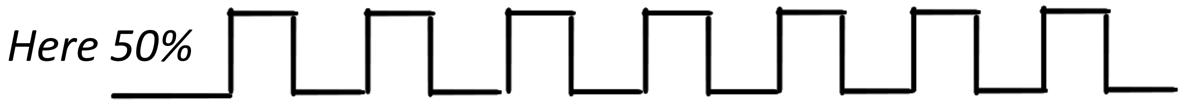


75% duty cycle



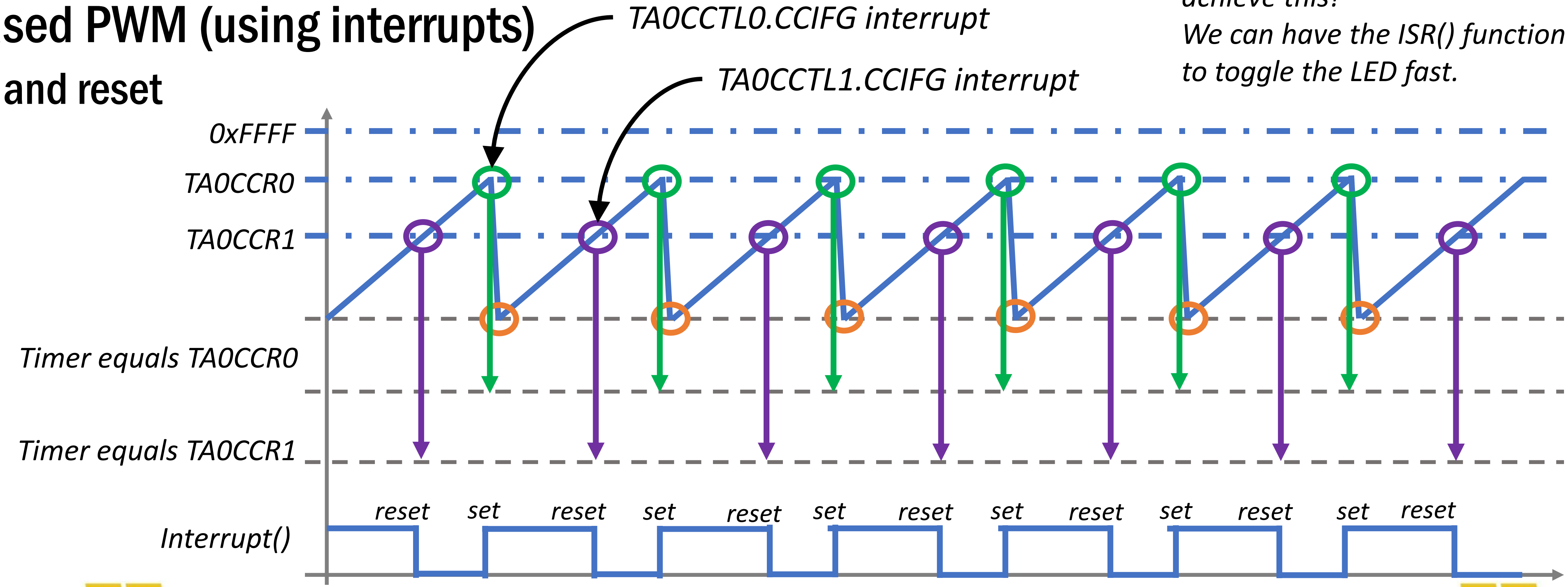
LED Dimming using PWM (based on Timer)

- Creating the duty cycle as needed
  - To determine dimming



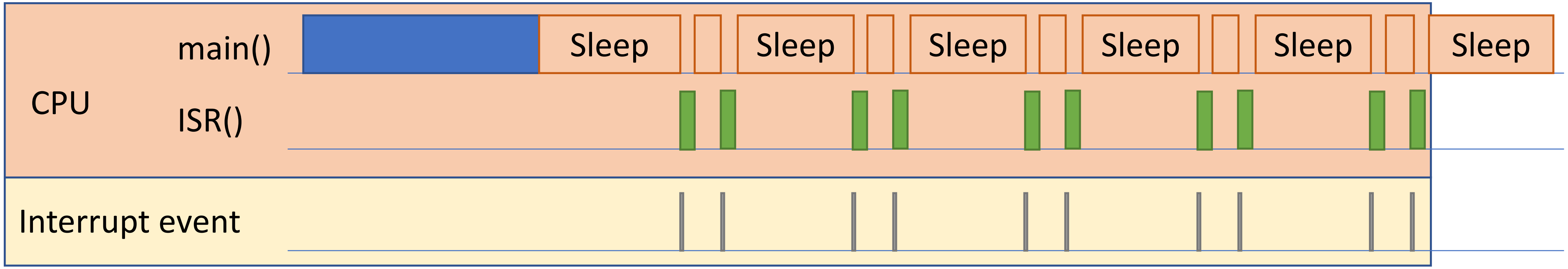
We can use timer interrupts to achieve this!
We can have the ISR() function to toggle the LED fast.

- Timer-based PWM (using interrupts)
  - To set and reset



LED Dimming using PWM (based on Timer)

- The use of ISR for blinking LED and checking CCRs



- Delay calculation by timer module
- Interrupt service routines to handle interrupts
- Low power mode to turn off CPU

We don’t need to keep CPU/ISR busy for such recurring flags!

*Timer_A0 peripheral in MSP430 has a **“timer output”** feature, which can be used to generate simple signals, including PWM signals.*

Timer Output for PWM

- Timer output is a feature of the capture/compare block of timer_A

The TA0CCTL1 register to configure channel 1's output mode.

TA0CCTL1

								OUTMOD			*CCIE*		*OUT*		*CCIFG*
15	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*

OUTMOD	***Description***	***Value at timer out when***	
		TA0R equals TA0CCR1	***TA0R equals TA0CCR0***
000b	OUT bit	-	-
001b	Set	Set	-
010b	Toggle/Reset	Toggle	Reset
011b	Set/Reset	Set	Reset
100b	Toggle	Toggle	-
101b	Reset	Reset	-
110b	Toggle/Set	Toggle	Set
111b	Reset/Set	Reset	Set

How to access this timer output signal?

Similar to using ISR() as seen earlier

Timer Output for PWM

- Timer output is a feature of the capture/compare block of timer_A

The TA0CCTL1 register to configure channel 1's output mode.

TA0CCTL1

								OUTMOD			CCIE		OUT		CCIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

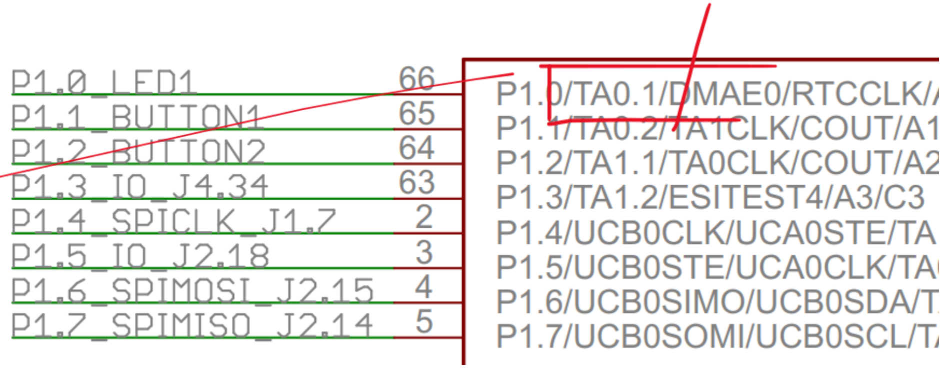
OUTMOD	Description	Value at timer out when	
		TA0R equals TA0CCR1	TA0R equals TA0CCR0
000b	OUT bit	-	-
001b	Set	Set	-
010b	Toggle/Reset	Toggle	Reset
011b	Set/Reset	Set	Reset
100b	Toggle	Toggle	-
101b	Reset	Reset	-
110b	Toggle/Set	Toggle	Set
111b	Reset/Set	Reset	Set

Different timer output modes, in combination with continuous (or up) mode, plus the value of CCRs, can be used to generate a wide variety of signals.

Timer Output for PWM

- Timer output is a feature of the capture/compare block of timer_A

Default value



PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS (1)		
			P1DIR.x	P1SEL1.x	P1SEL0.x
P1.0/TA0.1/DMAE0/RTCCLK/A0/C0/VREF-/VeREF-	0	P1.0 (I/O)	I: 0; O: 1	0	0
		TA0.CCI1A	0	0	1
		TA0.1	1		
		DMAE0	0	1	0
		RTCCLK (2)	1		
		A0, C0, VREF-, VeREF- (3) (4)	X	1	1
P1.1/TA0.2/TA1CLK/COUT/A1/C1/VREF+/VeREF+	1	P1.1 (I/O)	I: 0; O: 1	0	0
		TA0.CCI2A	0	0	1
		TA0.2	1		
		TA1CLK	0	1	0
		COUT (5)	1		
		A1, C1, VREF+, VeREF+ (3) (4)	X	1	1

change it from the default setting of P1.0 to Timer_A0 channel 1

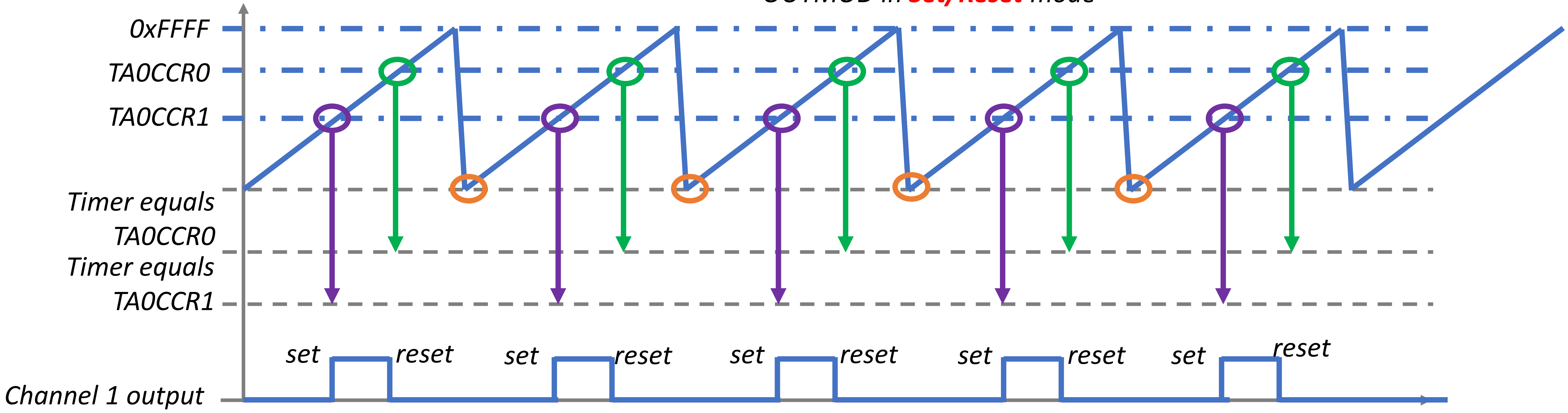
Similarly, TA0.2 is for timer_A0 channel 2 and so on.

LED Dimming using PWM (based on Timer)

- Creating the duty cycle as needed
  - To determine dimming

- Timer-based PWM (using interrupts)
  - To set and reset

For channel 1
with timer in ***continuous*** *mode*
OUTMOD in ***Set/Reset*** *mode*

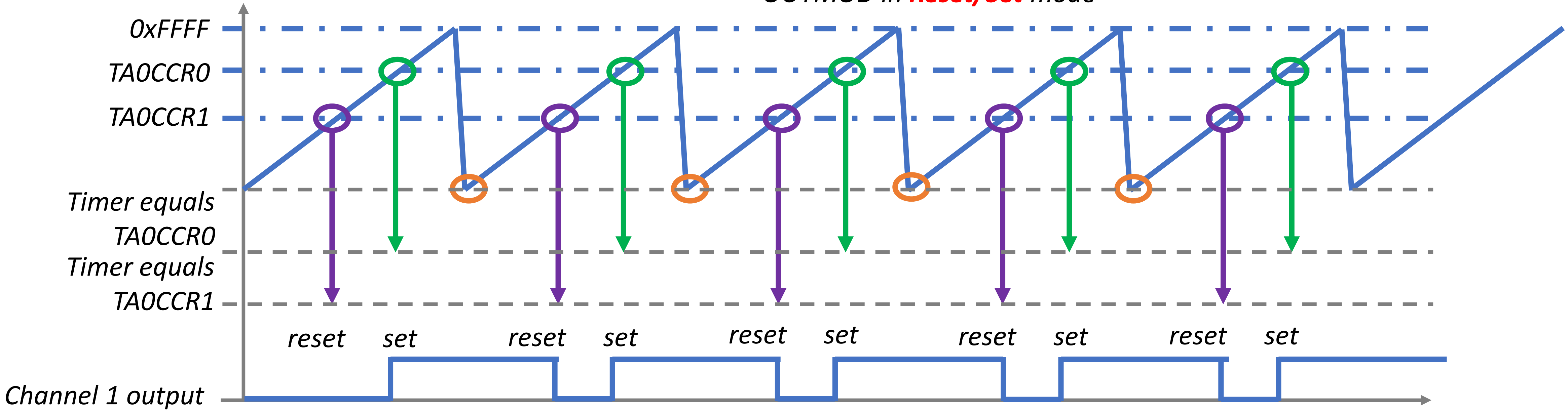


LED Dimming using PWM (based on Timer)

- Creating the duty cycle as needed
  - To determine dimming

- Timer-based PWM (using interrupts)
  - To set and reset

For channel 1
with timer in ***continuous*** *mode*
OUTMOD in ***Reset/Set*** *mode*

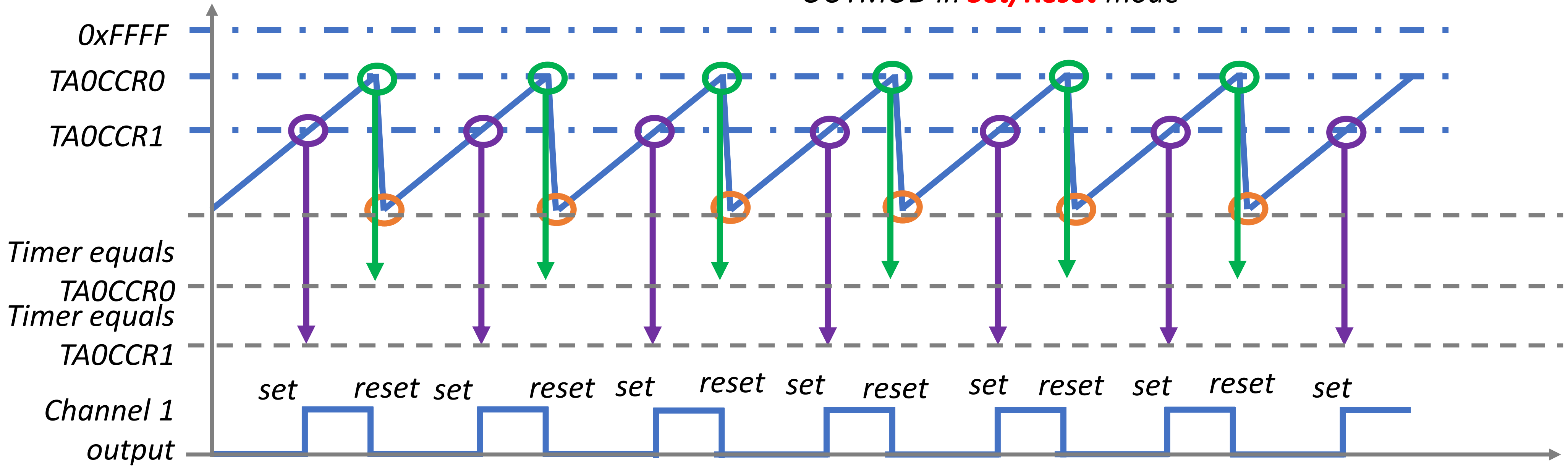


LED Dimming using PWM (based on Timer)

- Creating the duty cycle as needed
  - To determine dimming

- Timer-based PWM (using interrupts)
  - To set and reset

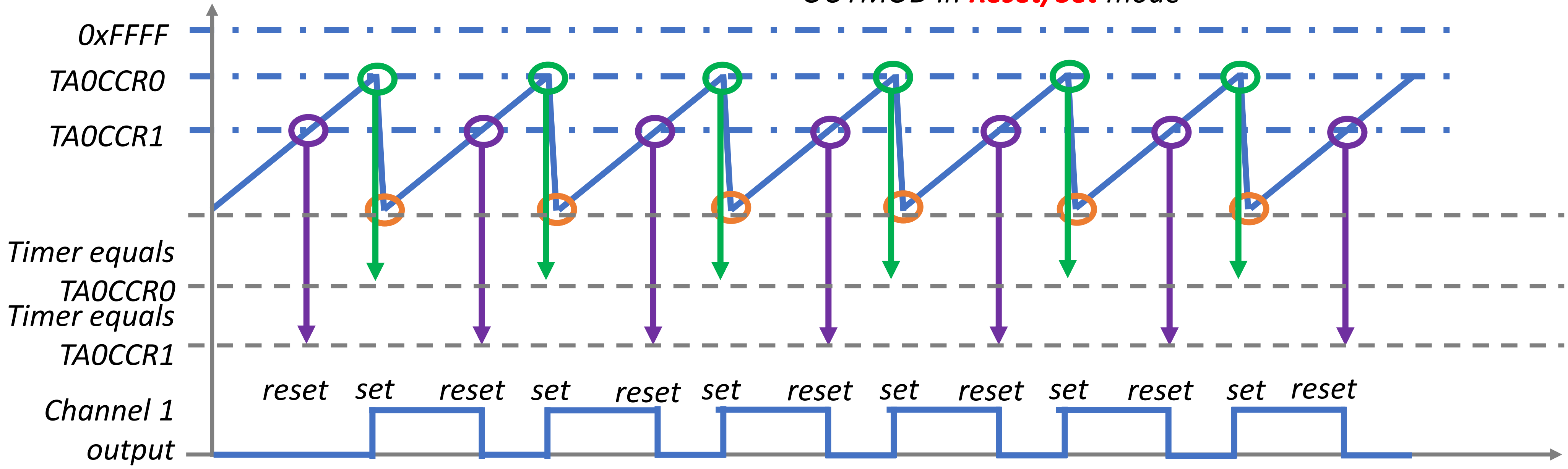
For channel 1
with timer in ***up*** *mode*
OUTMOD in ***Set/Reset*** *mode*



LED Dimming using PWM (based on Timer)

- Creating the duty cycle as needed
  - To determine dimming
- Timer-based PWM (using interrupts)
  - To set and reset

For channel 1
with timer in **up** *mode*
OUTMOD in **Reset/Set** *mode*



LED Dimming using PWM (based on Timer)

- **Duty cycle calculation**

$$Duty\ cycle\ = \frac{t_{ON}}{t_{ON} + t_{OFF}}$$

TA0CCR0 controls the frame size.

Say, frame rate = $50Hz$*.*

frame size = $1/50$ *=* $20ms$

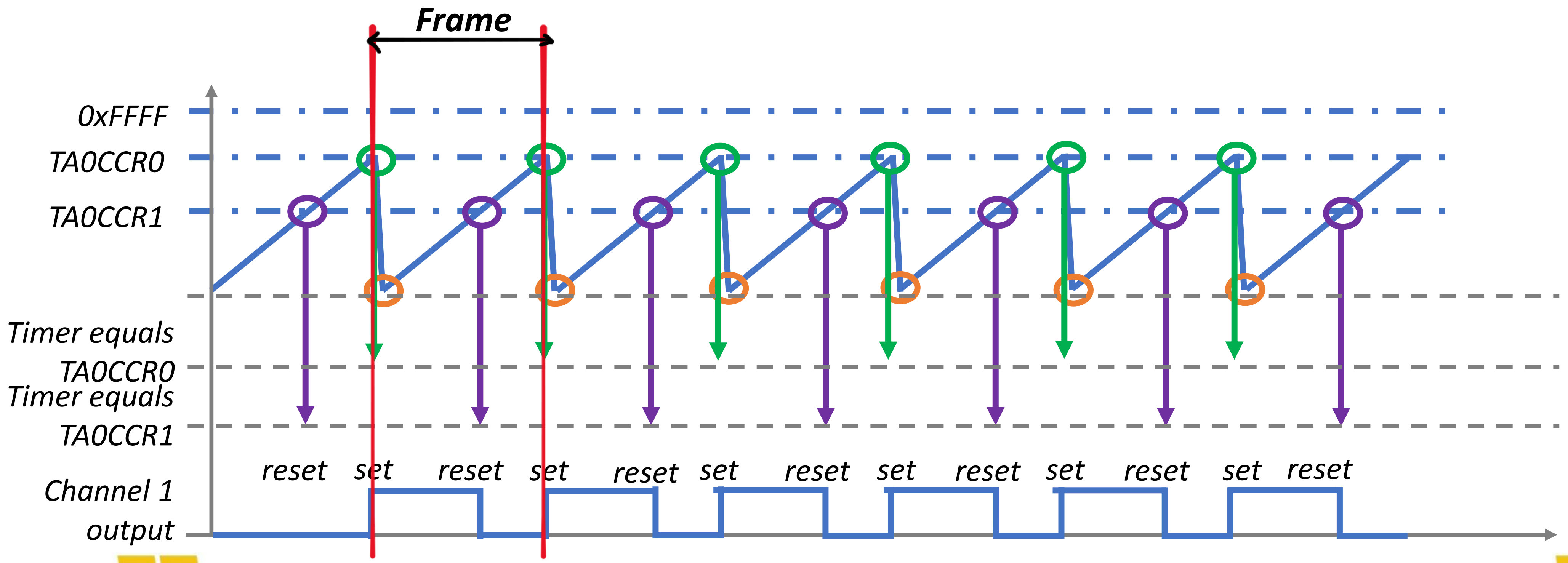
TA0CCR0 *= 32768 x 0.02 ≈* ***655***

Duty cycle = 40%

$t_{ON}\ =\ 20ms \times 40\%\ = 8ms$

TA0CCR1 *= TA0CCR0 x 40%*

= 655 x 40% = ***262***



LED Dimming using PWM (based on Timer)

- **Duty cycle calculation**

```
// Code that dimming an LED using timer_A0 channel 1
#include <msp430fr6989.h>
#define redLED BIT0                                     // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;                      // stop watchdog timer

        P1DIR |= redLED;                                // Direct pin as output
        P1SEL1 &= ~LED;                                 // Primary function, P1.0
        P1OUT &= ˜redLED;                               // Turn LED Off

        TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;        // ACLK, up mode, clear TAR, no divider

        TA0CCR0 = 655;                                  // Set frame rate for PWM
         TA0CCR1 = 262;                                 // Set duty cycle (40% of 655)
         TA0CCTL1 = OUTMOD_7;                           // OUTMOD_7 is set/reset mode for PWM

         _low_power_mode_3();                           // Enter low power mode 3

return;
}
```

$$\frac{32{,}768}{x} = \frac{1s}{20\,ms}$$

TA0CCR0 controls the frame size.

Say, frame rate = $50Hz.$

frame size = $1/50 = 20ms$

TA0CCR0** = 32768 x 0.02 ≈ **655

Duty cycle = 40%

$t_{ON} = 20ms \times 40\% = 8ms$

***TA0CCR1** = TA0CCR0 x 40%*

*= 655 x 40% = **262***

$$Duty\ cycle = \frac{t_{ON}}{t_{ON} + t_{OFF}}$$

LED Dimming using PWM (based on Timer)

- **Duty cycle calculation**

```c
// Code that dimming an LED using timer_A0 channel 1
#include <msp430fr6989.h>
#define redLED BIT0                                  // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;                   // stop watchdog timer

        P1DIR |= redLED;                             // Direct pin as output
        P1SEL1 &= ~LED;                              // Primary function, P1.0
        P1OUT &= ˜redLED;                            // Turn LED Off

        TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;     // ACLK, up mode, clear TAR, no divider

        TA0CCR0 = 655;                               // Set frame rate for PWM
         TA0CCR1 = 262;                              // Set duty cycle (40% of 655)
         TA0CCTL1 = OUTMOD_7;                        // OUTMOD_7 is set/reset mode for PWM

         _low_power_mode_3();                        // Enter low power mode 3

return;
}
```

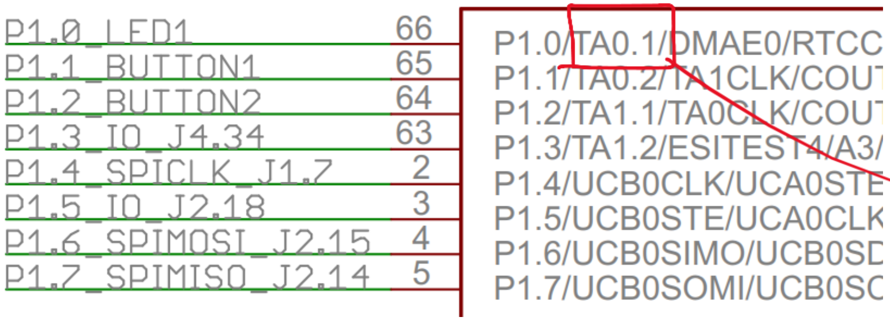
(Q) Do we need µc or/and interrupt here for controlling the process?

[redacted]

LED Dimming using PWM (based on Timer)

- **Duty cycle calculation**

```
// Code that dimming an LED using timer_A0 channel 1
#include <msp430fr6989.h>
#define redLED BIT0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;

        P1DIR |= redLED;
        P1SEL1 &= ~LED;
        P1OUT &= ˜redLED;

        TA0CTL = TASSEL_1 | ID_0 | MC_1 | TACLR;  // ACLK, up mode, clear TAR, no divider

        TA0CCR0 = 655;                            // Set frame rate for PWM
         TA0CCR1 = 262;                           // Set duty cycle (40% of 655)
         TA0CCTL1 = OUTMOD_7;                     // OUTMOD_7 is set/reset mode for PWM

         _low_power_mode_3();                     // Enter low power mode 3

return;
}
```



(Q) Do we need µc or/and interrupt here for controlling the process?

PWM available on the pin

(A) There is no need for any ISR(). Because we are not using any interrupts.

Thank You!

Questions?

Email: kamali@ucf.edu
UCF HEC 435 (407) 823 – 0764
https://www.ece.ucf.edu/~kamali/

HAVEN Research Group
https://haven.ece.ucf.edu/