Line Tracer 07

- Timer A -

1. Timer A

Timer A

Advanced Timer vs Systick Timer

- Systick timer only provides simple features
- Timer A provides rich features (PWM, ...)

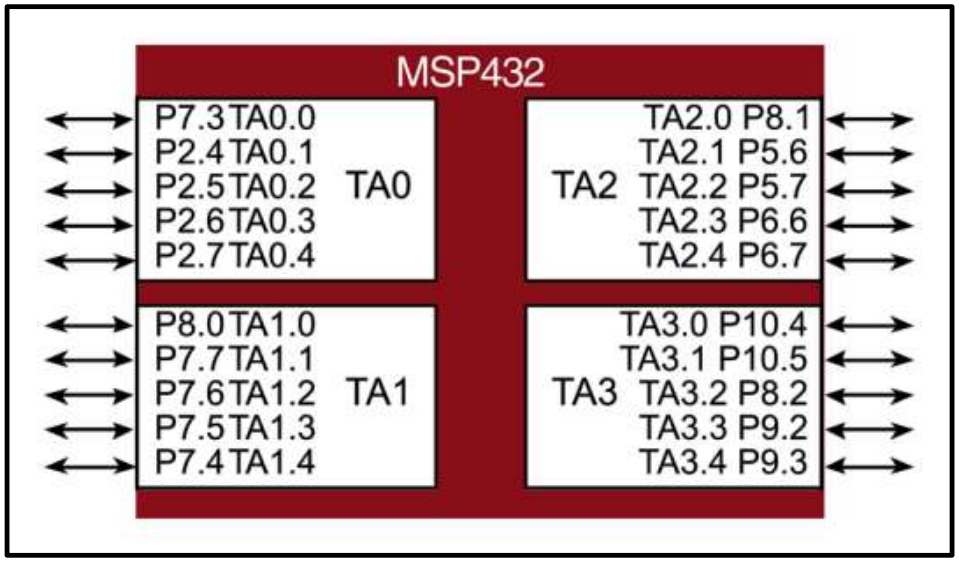
What we're going to run

- Generate PWM using timer

MSP432 Timer A

MSP432 Timer A consists of

- 4 Timers TA0, TA1, TA2, TA3
- Each timer has 7 submodules



Timer A Registers

	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

Timer A Register – CTL

Table 17-4. TAxCTL Register Description

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK
7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8
5-4	MC	RW	0h	Mode control. Setting MCx = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h
3	Reserved	RW	0h	Reserved
2	TACLR	RW	0h	Timer_A clear. Setting this bit resets TAxR, the timer clock divider logic, and the count direction. The TACLR bit is automatically reset and is always read as zero.
1	TAIE	RW	0h	Timer_A interrupt enable. This bit enables the TAIFG interrupt request. 0b = Interrupt disabled 1b = Interrupt enabled
0	TAIFG	RW	0h	Timer_A interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Timer A Register – CCTL

Bit	Field	Type	Reset	Description
15-14	CM	RW	0h	Capture mode 00b = No capture 01b = Capture on rising edge 10b = Capture on falling edge 11b = Capture on both rising and falling edges
13-12	CCIS	RW	0h	Capture/compare input select. These bits select the TAxCCR0 input signal. See the device-specific data sheet for specific signal connections. 00b = CCIxA 01b = CCIxB 10b = GND 11b = VCC
11	SCS	RW	0h	Synchronize capture source. This bit is used to synchronize the capture input signal with the timer clock. 0b = Asynchronous capture 1b = Synchronous capture
10	SCCI	RW	0h	Synchronized capture/compare input. The selected CCI input signal is latched with the EQUx signal and can be read via this bit.
9	Reserved	R	0h	Reserved. Reads as 0.
8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode
7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set
4	CCIE	RW	0h	Capture/compare interrupt enable. This bit enables the interrupt request of the corresponding CCIFG flag. 0b = Interrupt disabled 1b = Interrupt enabled
3	CCI	R	0h	Capture/compare input. The selected input signal can be read by this bit.
2	OUT	RW	0h	Output. For output mode 0, this bit directly controls the state of the output. 0b = Output low 1b = Output high
1	COV	RW	0h	Capture overflow. This bit indicates a capture overflow occurred. COV must be reset with software. 0b = No capture overflow occurred 1b = Capture overflow occurred
0	CCIFG	RW	0h	Capture/compare interrupt flag 0b = No interrupt pending 1b = Interrupt pending

Timer A Register – TAxEX0

Table 17-9. TAxEX0 Register Description

Bit	Field	Type	Reset	Description
15-3	Reserved	R	0h	Reserved. Reads as 0.
2-0	TAIDEX	RW	0h	Input divider expansion. These bits along with the ID bits select the divider for the input clock. 000b = Divide by 1 001b = Divide by 2 010b = Divide by 3 011b = Divide by 4 100b = Divide by 5 101b = Divide by 6 110b = Divide by 7 111b = Divide by 8

Timer A Register – Output Modes

Table 17-2. Output Modes

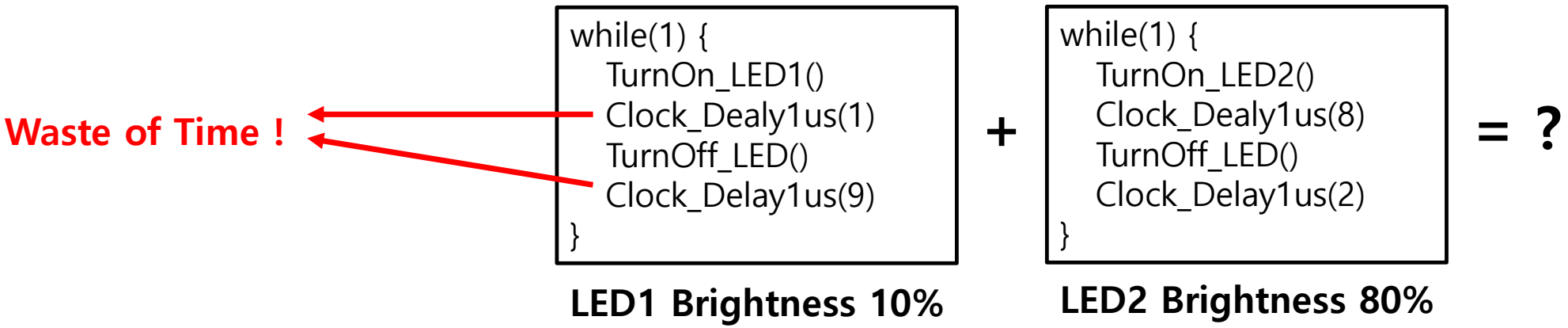
OUTMODx	Mode	Description
000	Output	The output signal OUTn is defined by the OUT bit. The OUTn signal updates immediately when OUT is updated.
001	Set	The output is set when the timer *counts* to the TAxCCRn value. It remains set until a reset of the timer, or until another output mode is selected and affects the output.
010	Toggle/Reset	The output is toggled when the timer *counts* to the TAxCCRn value. It is reset when the timer *counts* to the TAxCCR0 value.
011	Set/Reset	The output is set when the timer *counts* to the TAxCCRn value. It is reset when the timer *counts* to the TAxCCR0 value.
100	Toggle	The output is toggled when the timer *counts* to the TAxCCRn value. The output period is double the timer period.
101	Reset	The output is reset when the timer *counts* to the TAxCCRn value. It remains reset until another output mode is selected and affects the output.
110	Toggle/Set	The output is toggled when the timer *counts* to the TAxCCRn value. It is set when the timer *counts* to the TAxCCR0 value.
111	Reset/Set	The output is reset when the timer *counts* to the TAxCCRn value. It is set when the timer *counts* to the TAxCCR0 value.

2. PWM Using Timer

PWM Using Timer

Previous PWM Implementation

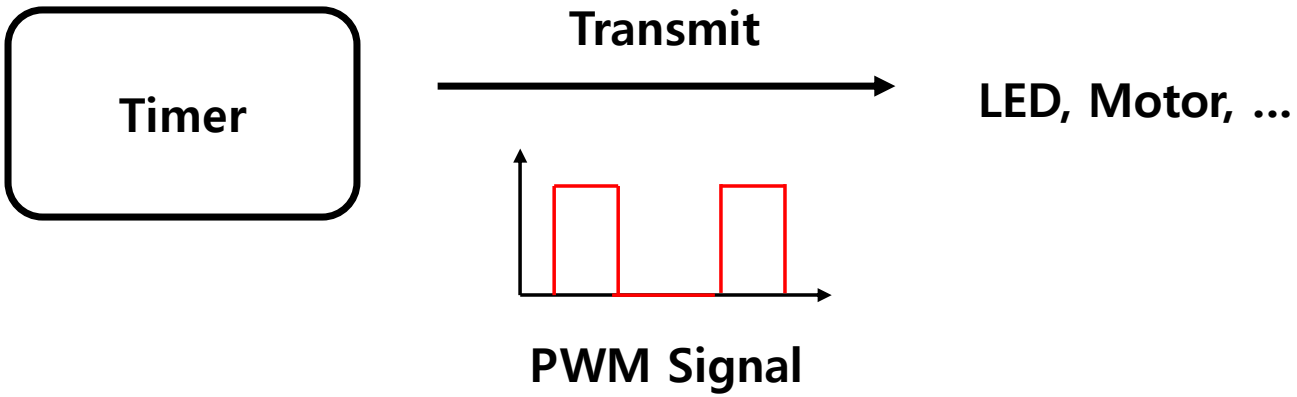
- We created PWM signal manually
- Hard to understand source code
- Nothing can be done during the delay



PWM Using Timer

PWM Using Timer

- Transmit the timer generated signal
- Timer takes care of everything!



PWM Using Timer

GPIO vs Alternative

- In GPIO, we have to deal with the relevant hardware ourselves
  (Turn Off/On LED, Read Switch, Generate PWM Signal ...)
- In Alternative Mode, the hardware does tasks such as generating PWM signals

```
void turn_on_led(int color) {
    P2->OUT &= ~0x07;
    P2->OUT |= color;
}
void turn_off_led() {
    P2->OUT &= ~0x07;
}
```

In GPIO, we delivered the signal by reading and writing on the relevant port (register) directly

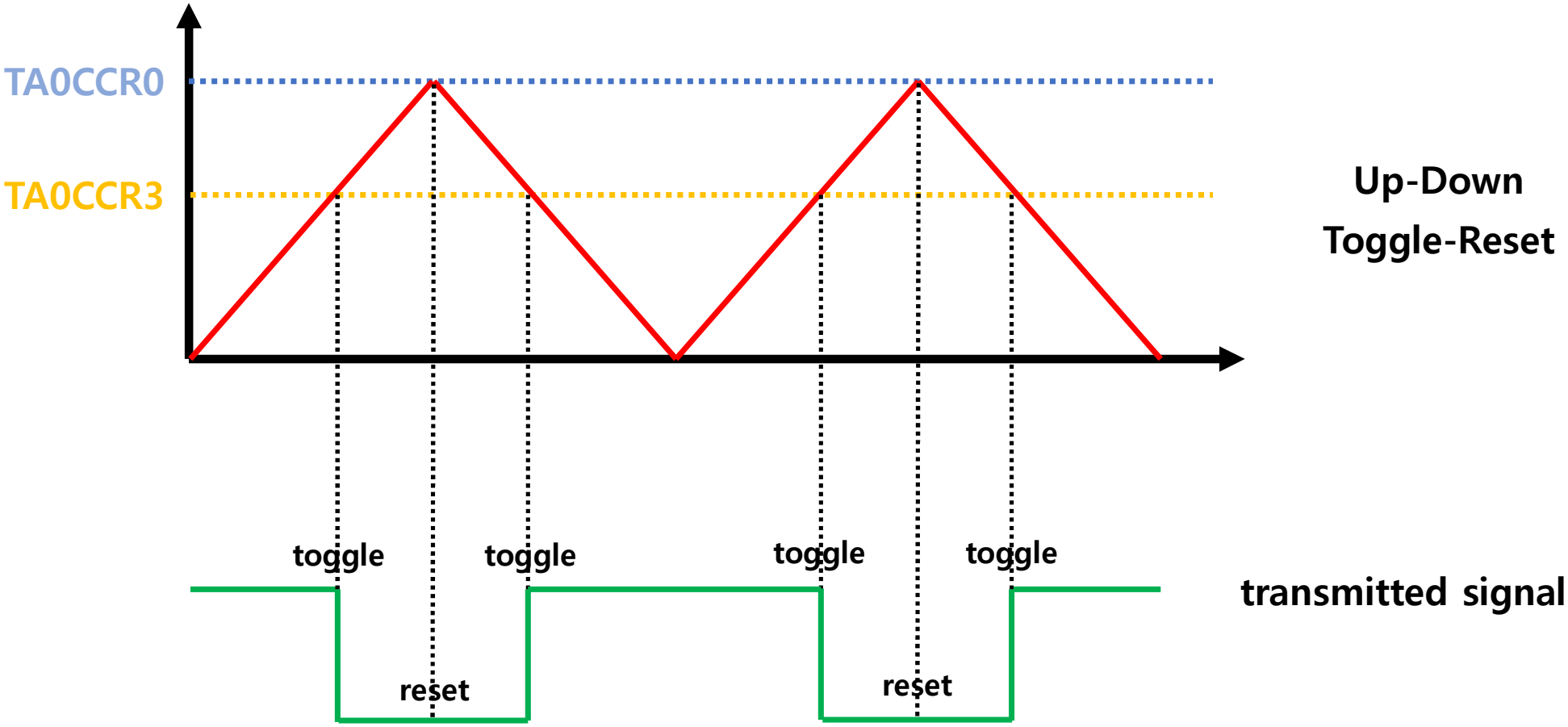
DC Motor Using PWM

```
void pwm_init34(uint16_t period, uint16_t duty3, uint16_t duty4) {
    // CCR0 period
    TIMER_A0->CCR[0] = period;

    // divide by 1
    TIMER_A0->EX0 = 0x0000;

    // toggle/reset
    TIMER_A0->CCTL[3] = 0x0040;
    TIMER_A0->CCR[3] = duty3;
    TIMER_A0->CCTL[4] = 0x0040;
    TIMER_A0->CCR[4] = duty4;

    // 0x200 -> SMCLK
    // 0b1100 0000 -> input divider /8
    // 0b0011 0000 -> up/down mode
    TIMER_A0->CTL = 0x02F0;

    // set alternative
    P2->DIR |= 0xC0;
    P2->SEL0 |= 0xC0;
    P2->SEL1 &= ~0xC0;
}
```

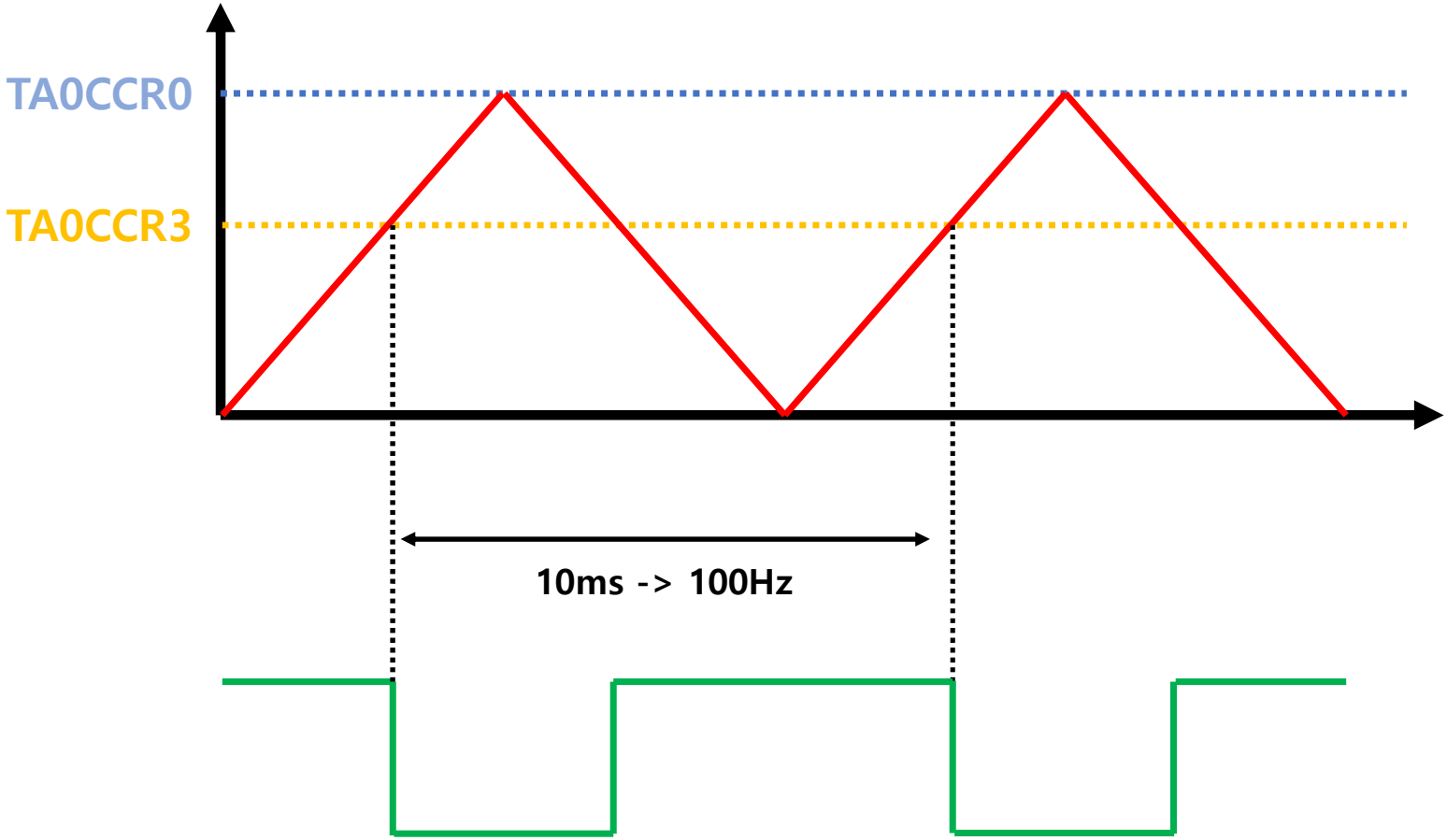
DC Motor Using PWM

```
void motor_init(void) {
    P3->SEL0 &= ~0xC0;
    P3->SEL1 &= ~0xC0;        // 1) configure nSLPR & nSLPL as GPIO
    P3->DIR |= 0xC0;          // 2) make nSLPR & nSLPL as output
    P3->OUT &= ~0xC0;         // 3) output LOW

    P5->SEL0 &= ~0x30;
    P5->SEL1 &= ~0x30;        // 1) configure DIRR & DIRL as GPIO
    P5->DIR |= 0x30;          // 2) make DIRR & DIRL as output
    P5->OUT &= ~0x30;         // 3) output LOW

    P2->SEL0 &= ~0xC0;
    P2->SEL1 &= ~0xC0;        // 1) configure PWMR & PWML as GPIO
    P2->DIR |= 0xC0;          // 2) make PWMR & PWML as output
    P2->OUT &= ~0xC0;         // 3) output LOW

    pwm_init34(7500, 0, 0);
}
```

How Do We Generate PWM Using Timer



Timer Frequency



Timer Frequency

Timer Frequency = Clock Source(SMCLK) / 2^ID(Input Divider) / TAIDX
= 12MHz / 1 / 8
= 1.5MHz

100Hz (PWM Frequency) = Timer Frequency / (2 * TA0CCR0)

TA0CCR0 = 7500

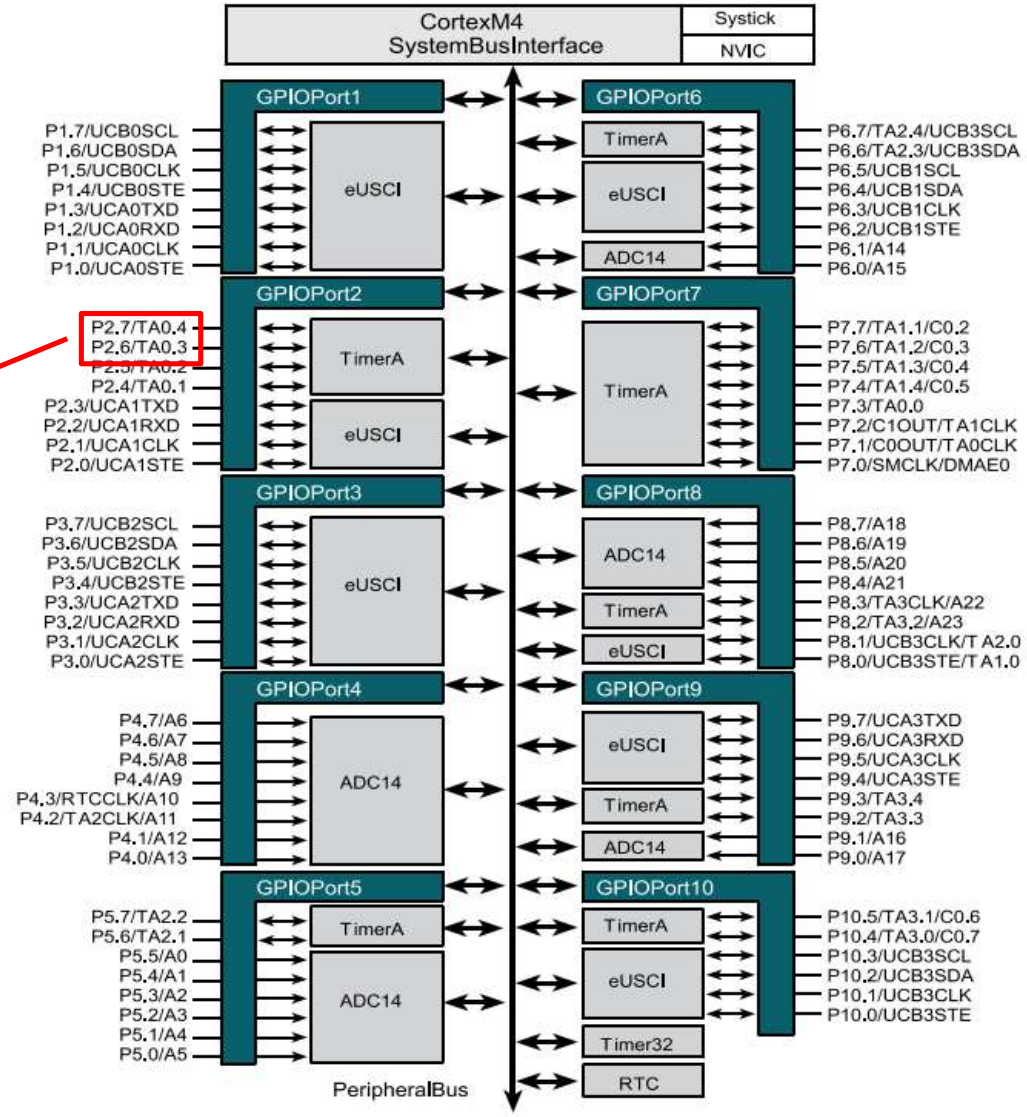
Set Pin Mode

```
void motor_init(void) {
    P3->SEL0 &= ~0xC0;
    P3->SEL1 &= ~0xC0;       // 1) configure nSLPR & nSLPL as GPIO
    P3->DIR |= 0xC0;         // 2) make nSLPR & nSLPL as output
    P3->OUT &= ~0xC0;        // 3) output LOW

    P5->SEL0 &= ~0x30;
    P5->SEL1 &= ~0x30;       // 1) configure DIRR & DIRL as GPIO
    P5->DIR |= 0x30;         // 2) make DIRR & DIRL as output
    P5->OUT &= ~0x30;        // 3) output LOW

    P2->SEL0 &= ~0xC0;
    P2->SEL1 &= ~0xC0;       // 1) configure PWMR & PWML as GPIO
    P2->DIR |= 0xC0;         // 2) make PWMR & PWML as output
    P2->OUT &= ~0xC0;        // 3) output LOW

    pwm_init34(7500, 0, 0);
}
```

```
void pwm_init34(uint16_t period, uint16_t duty3, uint16_t duty4) {
    // CCR0 period
    TIMER_A0->CCR[0] = period;

    // divide by 1
    TIMER_A0->EX0 = 0x0000;

    // toggle/reset
    TIMER_A0->CCTL[3] = 0x0040;
    TIMER_A0->CCR[3] = duty3;
    TIMER_A0->CCTL[4] = 0x0040;
    TIMER_A0->CCR[4] = duty4;

    // 0x200 -> SMCLK
    // 0b1100 0000 -> input divider /8
    // 0b0011 0000 -> up/down mode
    TIMER_A0->CTL = 0x02F0;

    // set alternative
    P2->DIR |= 0xC0;
    P2->SEL0 |= 0xC0;
    P2->SEL1 &= ~0xC0;
}
```

MSP432 Port Map

Motor's PWM pins are connected to timer0's 3&4 submodules



DC Motor Interfaces

```
void move(uint16_t leftDuty, uint16_t rightDuty) {
    P3->OUT |= 0xC0;
    TIMER_A0->CCR[3] = leftDuty;
    TIMER_A0->CCR[4] = rightDuty;
}

void left_forward() {
    P5->OUT &= ~0x10;
}

void left_backward() {
    P5->OUT |= 0x10;
}

void right_forward() {
    P5->OUT &= ~0x20;
}

void right_backward() {
    P5->OUT |= 0x20;
}
```

Example

```
void main(void)
{
    // Initialization
    Clock_Init48MHz();
    systick_init();
    motor_init();

    while (1) {
        left_forward();
        right_forward();
        move(2000, 2000);
        systick_wait1s();

        left_forward();
        right_forward();
        move(1000, 3000);
        systick_wait1s();
    }
}
```

Final project (11/19~12/17)

- **The final project will be conducted in teams of two (same teams as currently assigned).**
- **Each team will complete several predefined tracks, each with a maximum time limit.**
- **Scores will be determined by the number of successfully completed tracks, not by relative ranking.**
  → The project will therefore follow an absolute grading system.
- **Detailed information about each track will be announced on November 19.**
- **On-site evaluation only.**
- **Device lending is not available, so please complete all coding during the class hours.**
- **Each team must submit a final report that includes:**
  - Source code files
  - Explanatory documentation
  - A clear **description of role division between the two members**
- **Submission is mandatory for all team members.**