CS4101 Introduction to Embedded Systems

Lab 2: Basic IO and Timer

Prof. Chung-Ta King

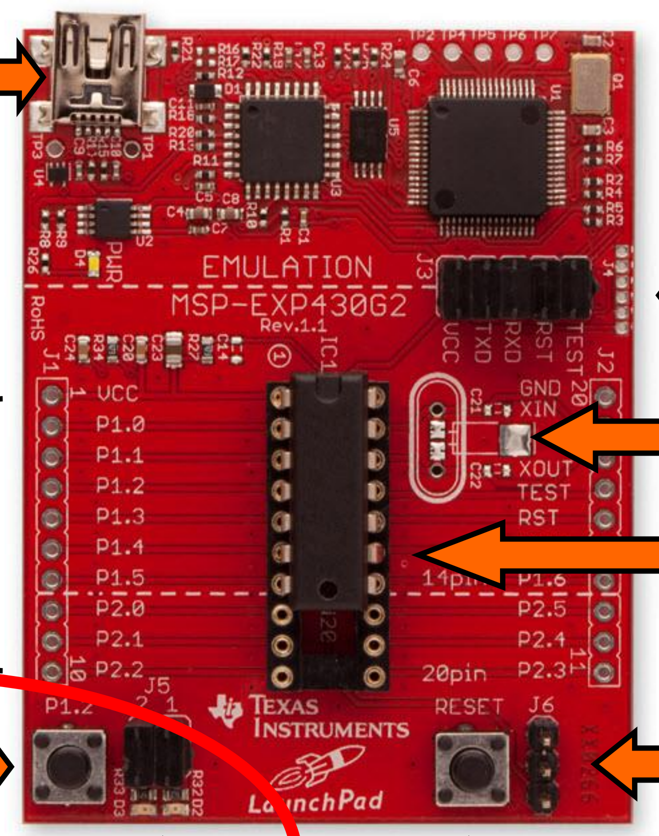
Department of Computer Science

National Tsing Hua University, Taiwan

Introduction

- In this lab, we will learn the basic IO and timer of MSP430 LanuchPad
  - Configuring the I/O port of LanuchPad for input
  - Running the debugger for basic debugging

LaunchPad Development Board



USB Emulator Connection

Embedded Emulation

6-pin eZ430 Connector

Crystal Pads

Chip Pinouts

Part and Socket

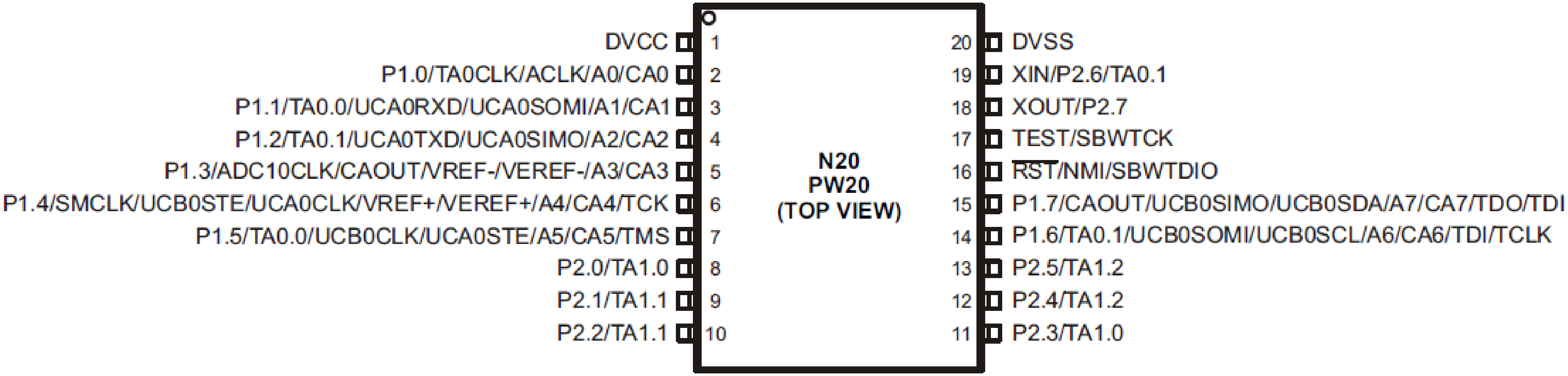
P1.3 Button

Power Connector

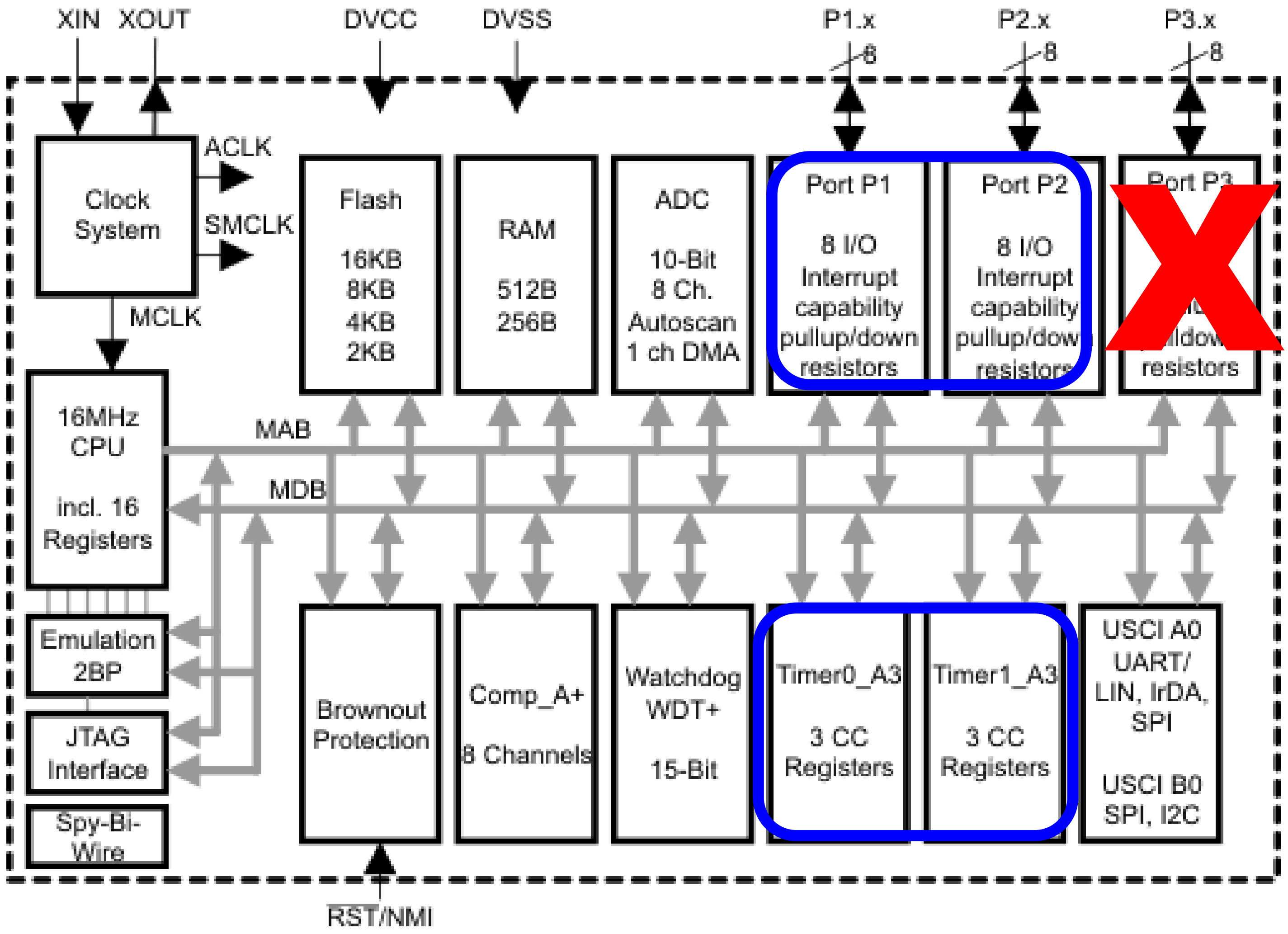
LEDs and Jumpers
P1.0 & P1.6

Reset Button

Exterior of MSP430G2553 (20-pin)

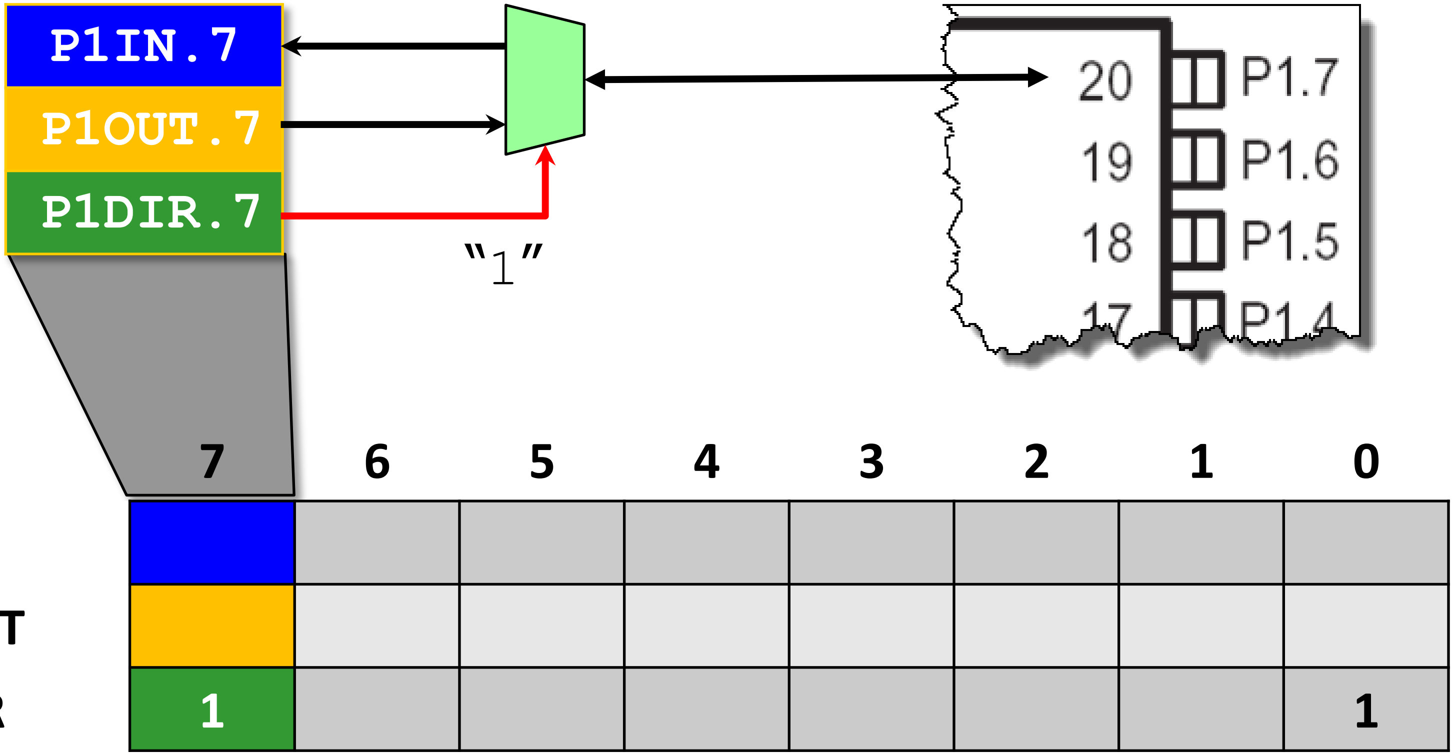

DVCC 1
P1.0/TA0CLK/ACLK/A0/CA0 2
P1.1/TA0.0/UCA0RXD/UCA0SOMI/A1/CA1 3
P1.2/TA0.1/UCA0TXD/UCA0SIMO/A2/CA2 4
P1.3/ADC10CLK/CAOUT/VREF-/VEREF-/A3/CA3 5
P1.4/SMCLK/UCB0STE/UCA0CLK/VREF+/VEREF+/A4/CA4/TCK 6
P1.5/TA0.0/UCB0CLK/UCA0STE/A5/CA5/TMS 7
P2.0/TA1.0 8
P2.1/TA1.1 9
P2.2/TA1.1 10
N20
PW20
(TOP VIEW)
20 DVSS
19 XIN/P2.6/TA0.1
18 XOUT/P2.7
17 TEST/SBWTCK
16 $\overline{\text{RST}}$/NMI/SBWTDIO
15 P1.7/CAOUT/UCB0SIMO/UCB0SDA/A7/CA7/TDO/TDI
14 P1.6/TA0.1/UCB0SOMI/UCB0SCL/A6/CA6/TDI/TCLK
13 P2.5/TA1.2
12 P2.4/TA1.2
11 P2.3/TA1.0

Interior of MSP430G2553



Not available on 20-pin device

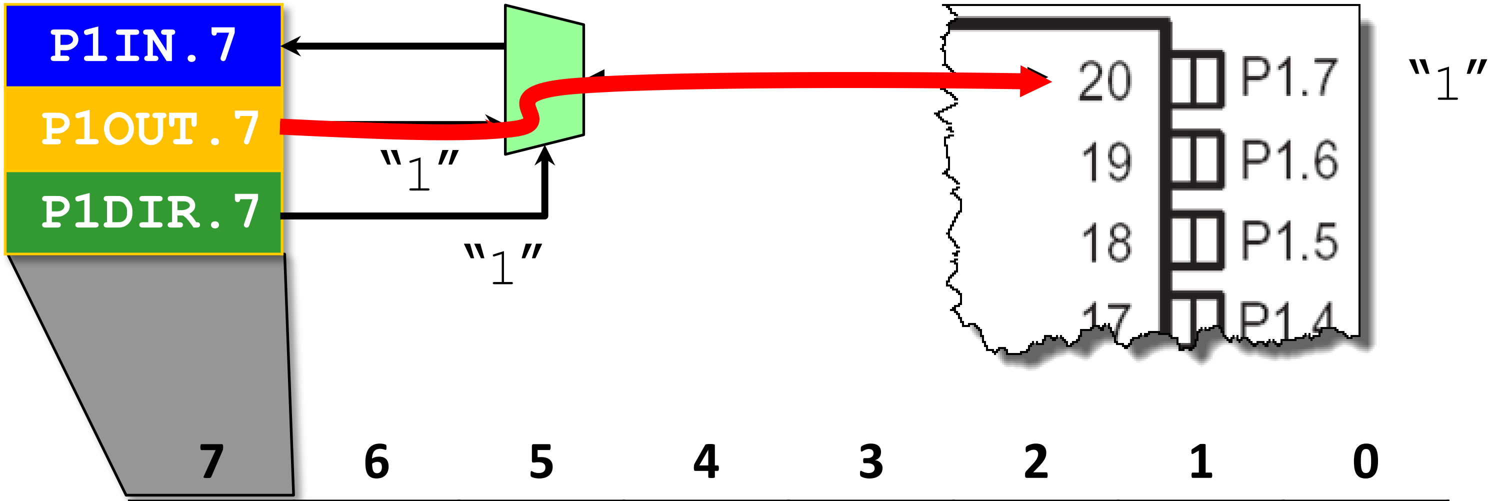
PxDIR (Pin Direction): Input or Output



	7	6	5	4	3	2	1	0
P1IN								
P1OUT								
P1DIR	1							1

- PxDIR.y: 0 = input 1 = output
- Register example: `P1DIR &= 0x81;`

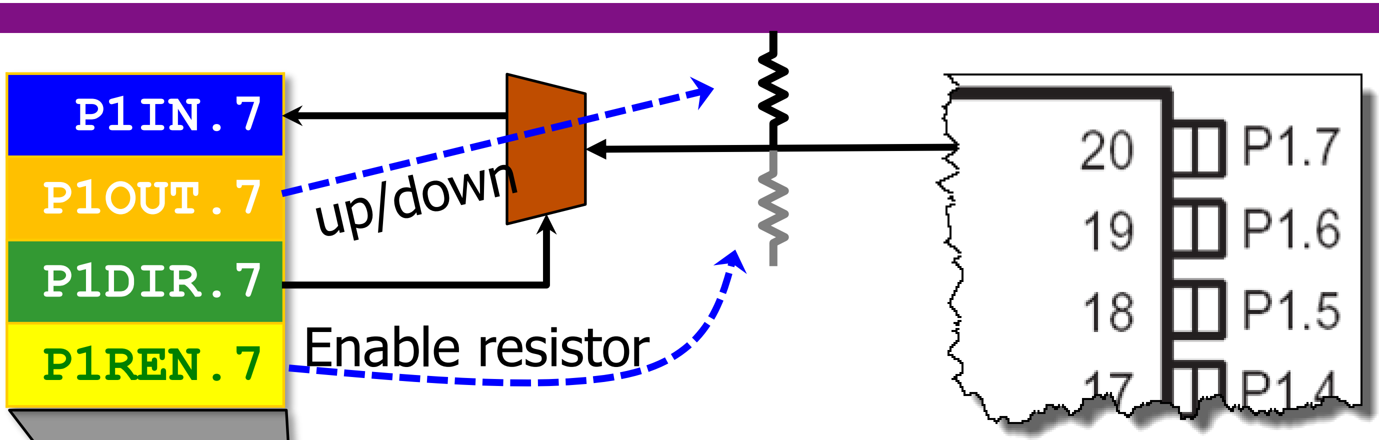
GPIO Output



	7	6	5	4	3	2	1	0
P1IN	X							
P1OUT	1							
P1DIR	1							

- PxOUT.y: 0 = low 1 = high
- Register example: `P1OUT &= 0x80;`

GPIO Input



	7	6	5	4	3	2	1	0
P1IN	x							
P1OUT	1							
P1DIR	0							
P1REN	1							

PxREN enabs resistors
PxOUT selects pull-up (1) or -down (0)

Sample Code 1 for Input

```
#include <msp430.h>
#define LED1 BIT0   //P1.0 to red LED
#define B1 BIT3     //P1.3 to button
void main(void){
  WDTCTL = WDTPW + WDTHOLD; //Stop watchdog timer
  P1OUT |= LED1 + B1;
  P1DIR = LED1; //Set pin with LED1 to output
  P1REN = B1;   //Set pin to use pull-up resistor
  for(;;){   //Loop forever
    if((P1IN & B1) == 0){  //Is button down
      P1OUT &= ~LED1;      // Turn LED1 off
    }
    else{                  //Is button up
      P1OUT |= LED1;       // Turn LED1 on
    }
  }
}
```

Sample Code 2 for Input

```
#include <msp430.h>
#define LED1 BIT6   //P1.0 to green LED
#define B1 BIT3     //P1.3 to button
volatile unsigned int i, j;
void main(void){
  WDTCTL = WDTPW + WDTHOLD; //Stop watchdog timer
  P1OUT |= LED1 + B1;
  P1DIR = LED1;  //Set pin with LED1 to output
  P1REN = B1;    //Set pin to use pull-up resistor
  for(;;){
    while((P1IN & B1) != 0){ //Loop on button up
      i = P1IN;    j = P1OUT;  }
    P1OUT &= ~LED1;  // Turn LED1 off
    while((P1IN & B1) == 0){ //Loop on button down
      i = P1IN;    j = P1OUT;  }
    P1OUT |= LED1;  // Turn LED1 on
  }
}
```

Lab 2

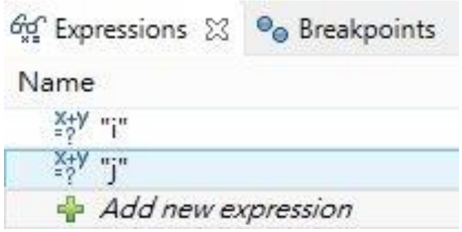
- Basic 1:
  - Upload and run sample code 1 and 2 on the MSP430 LaunchPad respectively. Do they behave differently? Why?
- Basic 2:
  - Modify sample code 2 to toggle the red LED each time the button is pressed. Turn it on the first time, off the second, on the third, and so on.
- Basic 3:
  - Run the debugger to show the values of P1IN after each while() statement

How to Debug?

How to know the process is working correctly?

- In the code line containing:

```
i = P1IN;
j = P1OUT;
```



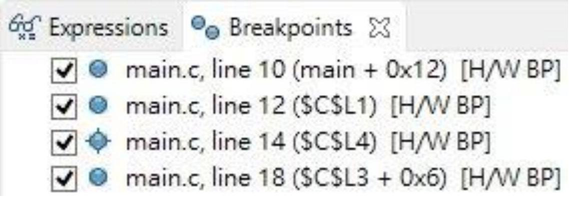
Name

"i"

"j"

Add new expression

- **Add new expression from Expressions window**
- Right-click on the appropriate line of code and set the **Breakpoint.**
- When the code runs, it will hit breakpoint and stop.
- Observe the value.

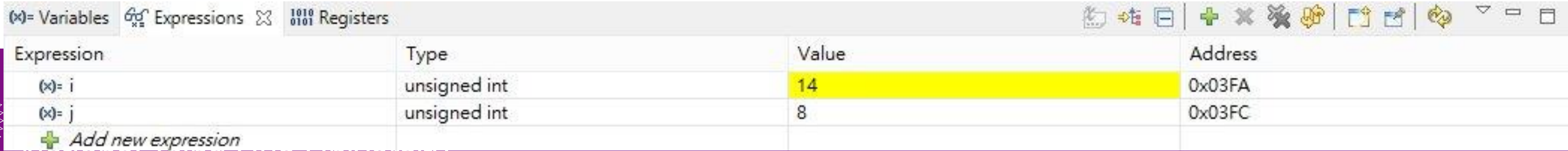


☑ main.c, line 10 (main + 0x12) [H/W BP]

☑ main.c, line 12 (CL1) [H/W BP]

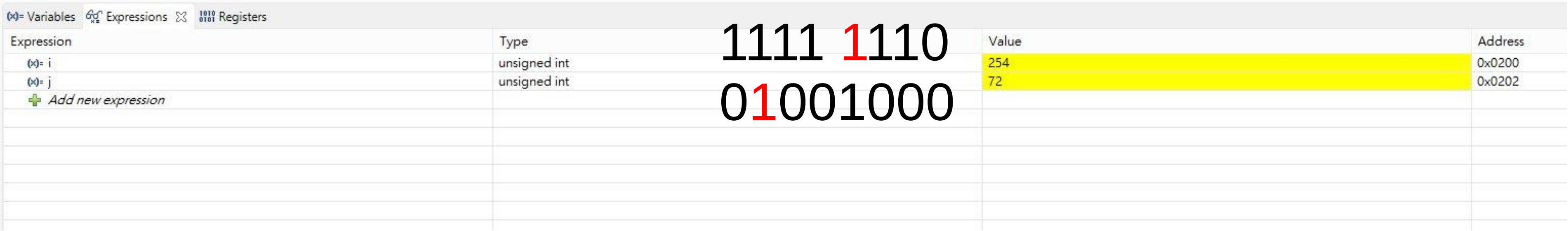
☑ main.c, line 14 (CL4) [H/W BP]

☑ main.c, line 18 (CL3 + 0x6) [H/W BP]



Expression	Type	Value	Address
i	unsigned int	14	0x03FA
j	unsigned int	8	0x03FC
Add new expression			

Debugger Output

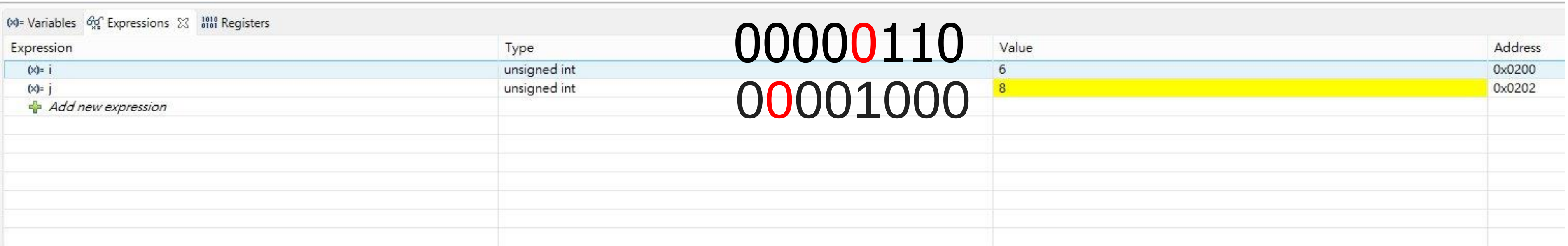


Variables | Expressions | Registers

Expression	Type	Value	Address
i	unsigned int	254	0x0200
j	unsigned int	72	0x0202
Add new expression			

1111 1110

01001000



Variables | Expressions | Registers

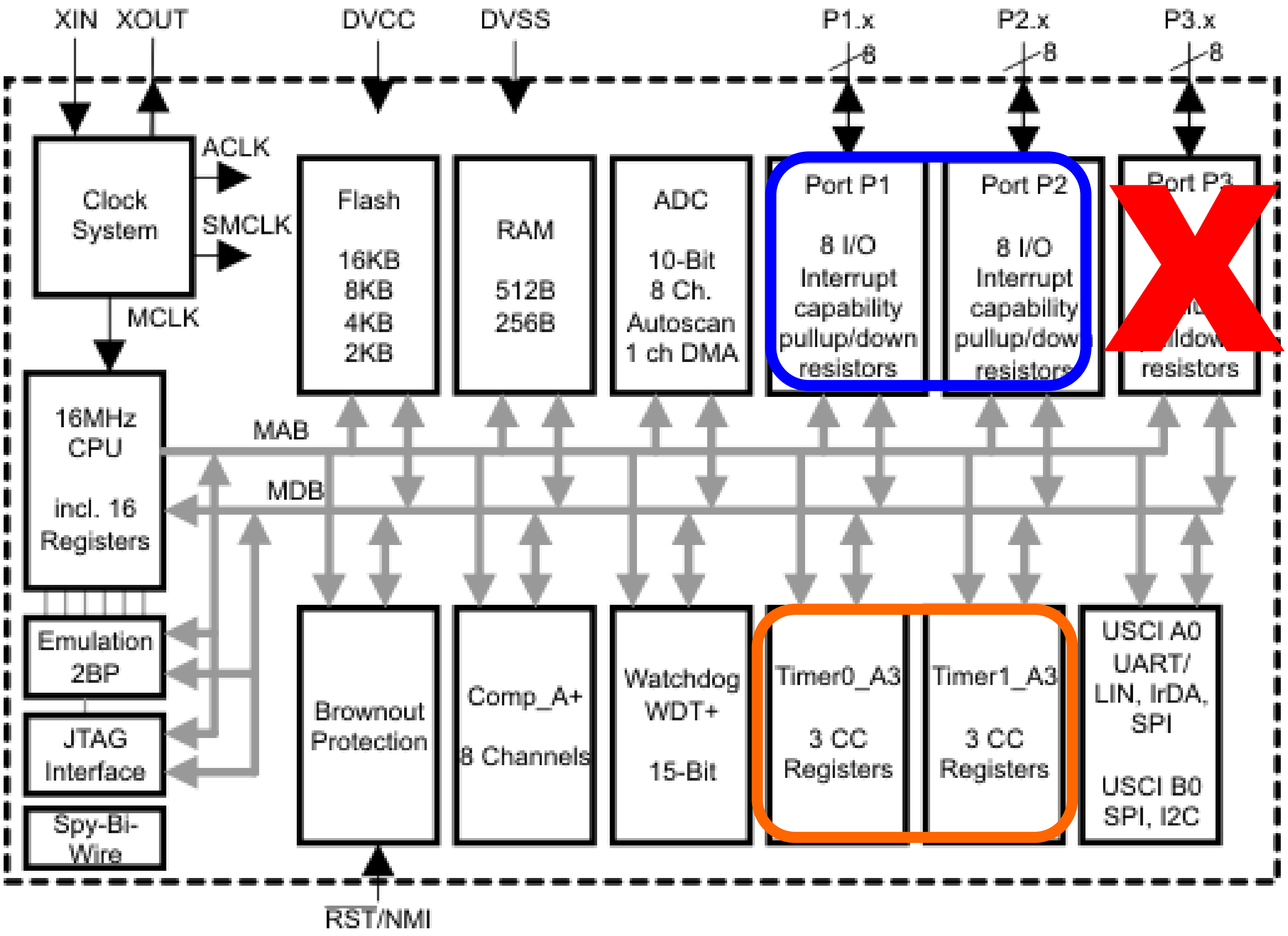
Expression	Type	Value	Address
i	unsigned int	6	0x0200
j	unsigned int	8	0x0202
Add new expression			

00000110

00001000

No need to care about other bits!

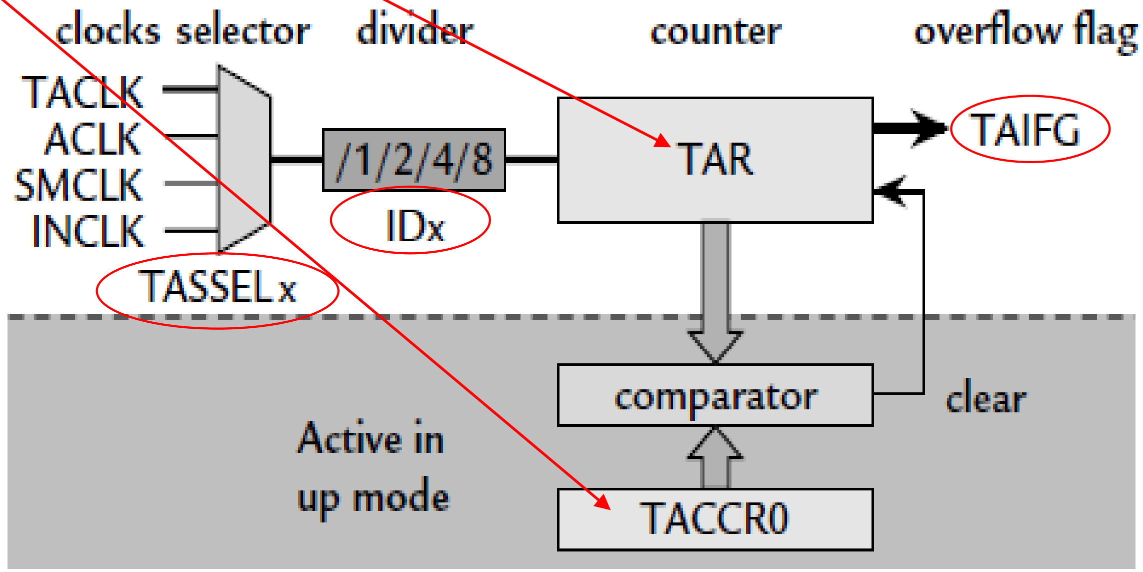
Interior of MSP430G2553



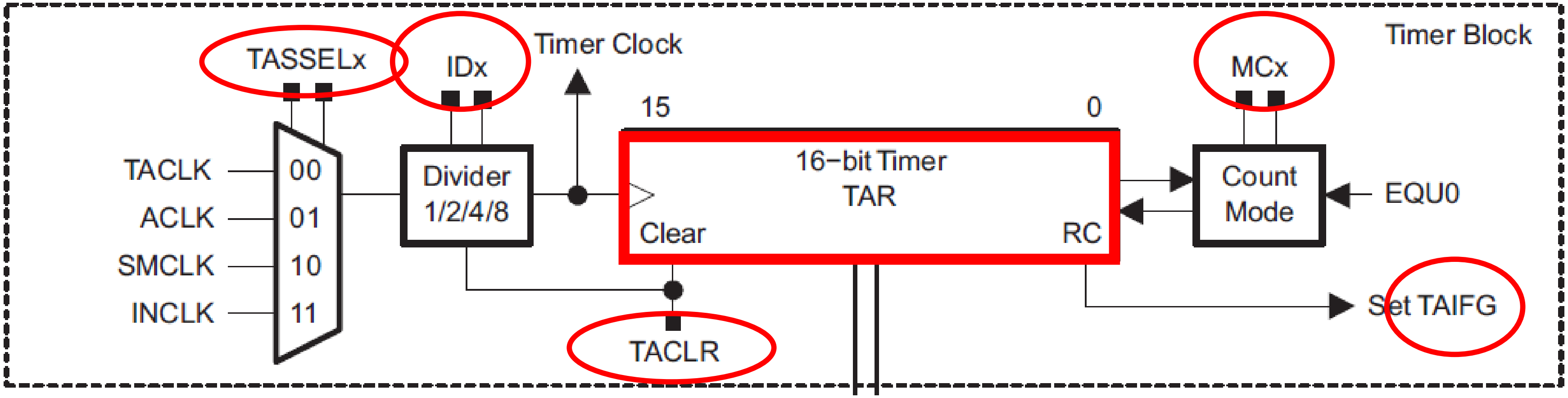
Not available on 20-pin device

MSP430 Timer_A: Registers

- TAR (0170h): the counter itself
- TACCR0 (0172h): target for counting
- TACTL (0160h): control settings
- Others: clock source selection, flags



Inside Timer_A

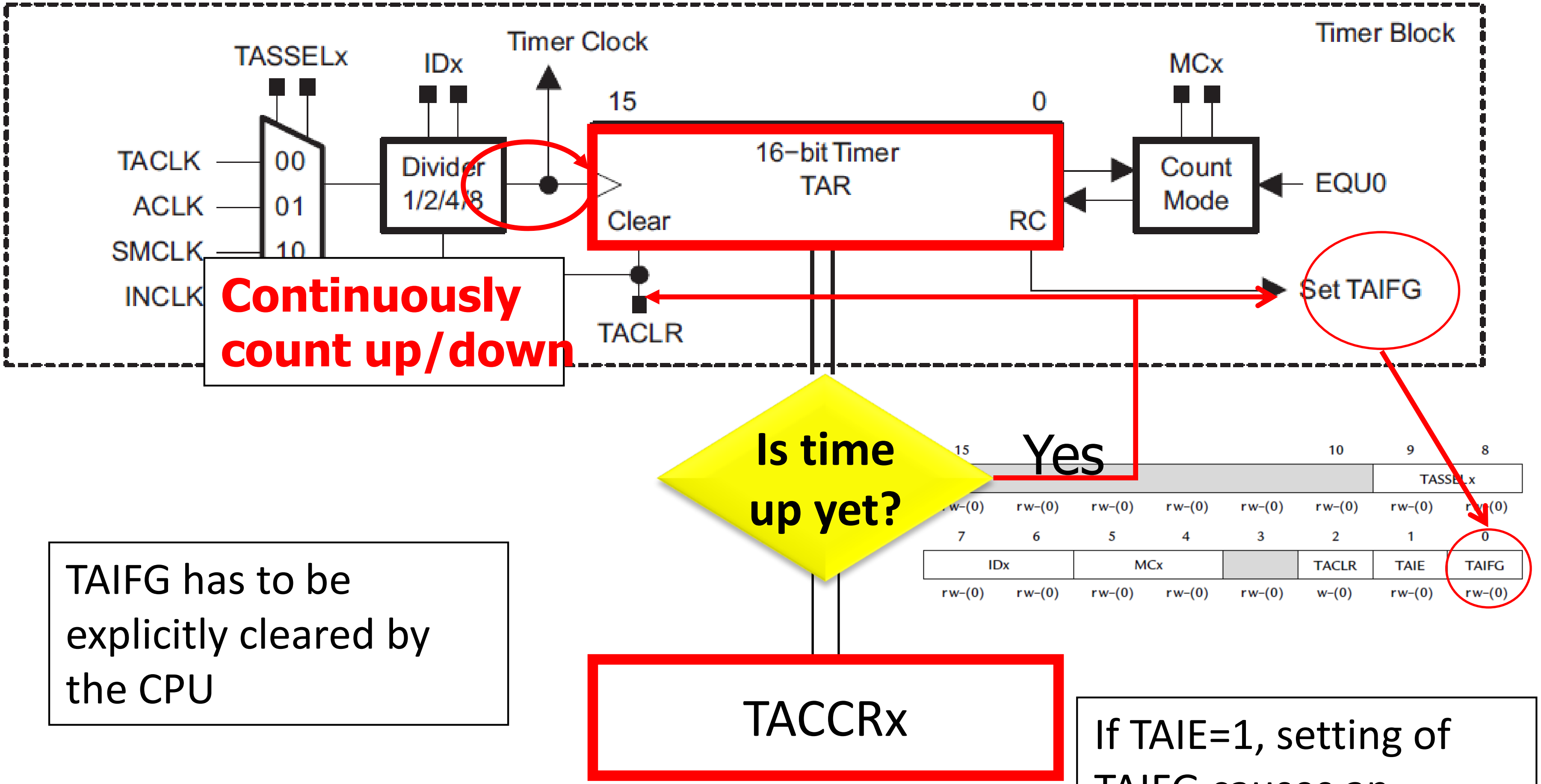


- Timer_A Control Register: TACTL

15					10	9	8
						TASSELx	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
IDx		MCx			TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Typical Operations of Timer_A



Continuously count up/down

TAIFG has to be explicitly cleared by the CPU

If TAIE=1, setting of TAIFG causes an interrupt to the CPU

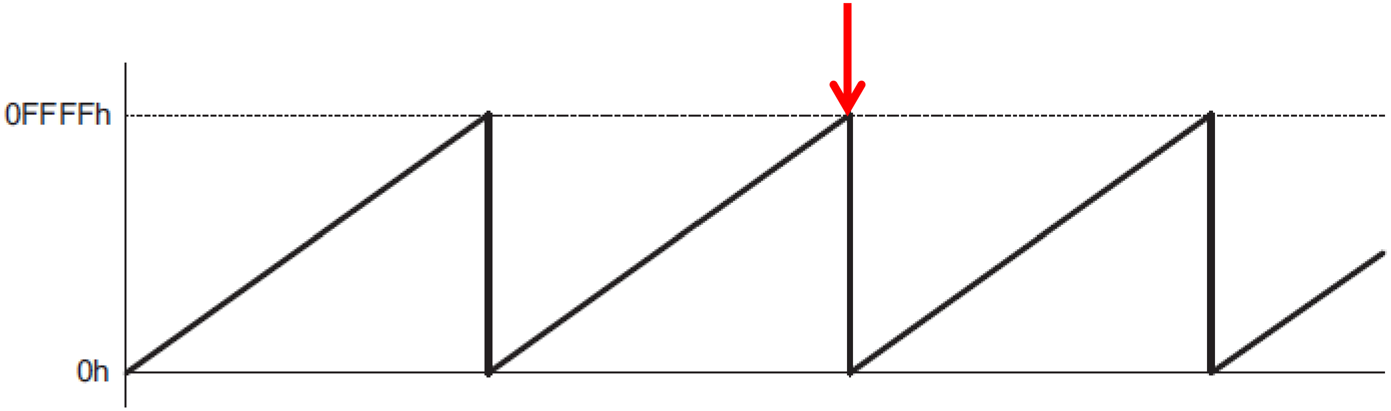
Timer Mode

- MCx=00: Stop mode
  - The timer is halted
- MCx=01: Up mode
  - The timer repeatedly counts from 0 to TACCR0
- MCx=10: Continuous mode
  - The timer repeatedly counts from 0 to 0FFFFh
- MCx=11: Up/down mode
  - The timer repeatedly counts from 0 to TACCR0 and back down to 0

Continuous Mode (MCx = 10)

- In the continuous mode, the timer repeatedly counts up to 0FFFFh and restarts from zero
- The TAIFG interrupt flag is set when the timer resets from 0FFFFh to zero

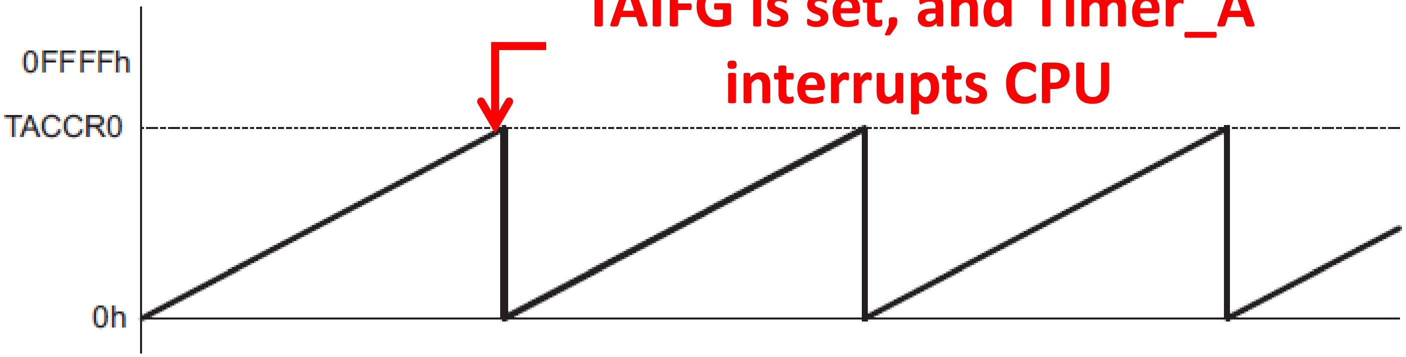
Timer interrupts!
(TAIFG is set)



Up Mode (MCx = 01)

The up mode is used if the timer period must be **different** from **0FFFFh** counts.

1. Timer period 100 → store 99 to TACCR0
2. When TACCR0 == 99, set TACCR0 CCIFG interrupt flag
3. Reset timer to 0 and set TAIFG interrupt flag

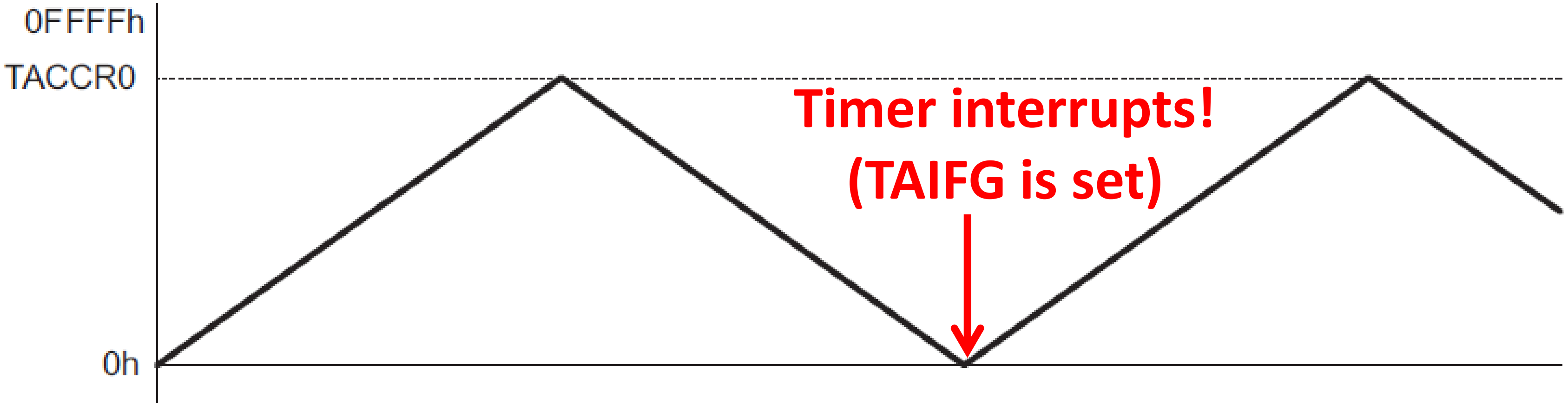


TAIFG is set, and Timer_A interrupts CPU

Up/Down Mode (MCx = 11)

- The up/down mode is used if the timer period must be different from 0FFFFh counts, and if a *symmetrical pulse generation* is needed.

  → **The period is twice the value in TACCR0**



Sample Code 1 for Timer_A

- Goal: simplest way to flash an LED at 1 Hz
  - Need an event to trigger the flashing → counter (TAR) overflow
  - Need a way to detect the event → CPU polling
- How to make TAR overflow at 1 Hz?
  - Use SMCLK clock (discussed later) at 800 KHz
  - When TAR (16 bits) overflows, it has counted 2^{16}, equivalent to a period of 2^{16}/800KHz ≈ 0.08 sec
  - Divide the frequency of the clock by 8 to give a period of about 0.64 sec → close enough!
  - Continuously count up; on overflow return to 0

Sample Code 1 for Timer_A

```
#include <msp430g2553.h>
#define LED1 BIT0
void main(void)
{
      WDTCTL = WDTPW | WDTHOLD;
      P1DIR = LED1;
      P1OUT = ~LED1;
      TACTL = MC_2 | ID_3 | TASSEL_2 |TACLR;
      for(;;) {
            while(!(TACTL & TAIFG)){}
            TACTL &= ~TAIFG;  // Clear overflow flag
            P1OUT ^=LED1;
      }
}
```

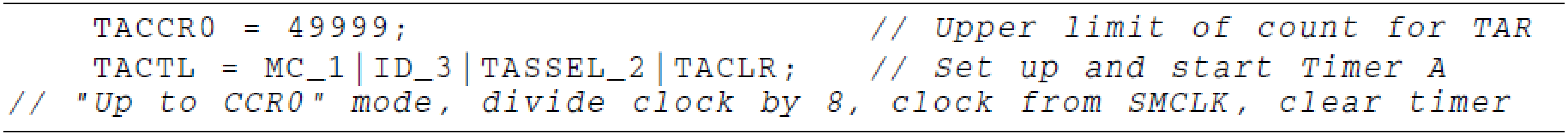
Sample Code Settings Explained

The following symbols are defined in header file:

- MC_2: set MC of TACTL to 10 (continuous mode)
- ID_3: set ID of TACTL to 11 (divide freq. by 8)
- TASSEL_2: set TASSEL to 10 (use SMCLK)
- TACLR: clear the counter, the divider, and the direction of the count

Sample Code 2 for Timer_A

- Can have more accurate time if we can control the amount to count
  - The maximum desired value of the count is programmed into TACCR0
  - TAR starts from 0 and counts up to the value in TACCR0, after which it returns to 0 and sets TAIFG
  - Thus the period is TACCR0+1 counts
  - With SMCLK (800KHz) divided down to 100 KHz, we need 50,000 counts for a delay of 0.5 sec → store 49,999 in TACCR0



```
    TACCR0 = 49999;                     // Upper limit of count for TAR
    TACTL = MC_1|ID_3|TASSEL_2|TACLR;   // Set up and start Timer A
// "Up to CCR0" mode, divide clock by 8, clock from SMCLK, clear timer
```

Lab 2

- **Basic 4:**
  - Complete sample code 2 and then modify it to flash the green LED at 2 Hz by polling Timer_A.
  - Hint: Since TAR register is 16-bit (0~65535) long, you should be careful of its overflow by using clock source “Divider”.
- **Bonus**:
  - Flash the green LED at 1 Hz by polling Timer_A. After the button is pressed, wait for 2 seconds and then turn the red LED to flash at 2 Hz.
  - Note: There are two events to monitor: timer up and button down.