EEL 4742 – Embedded Systems

Module 3 – Embedded System Design and Timer

Hadi 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

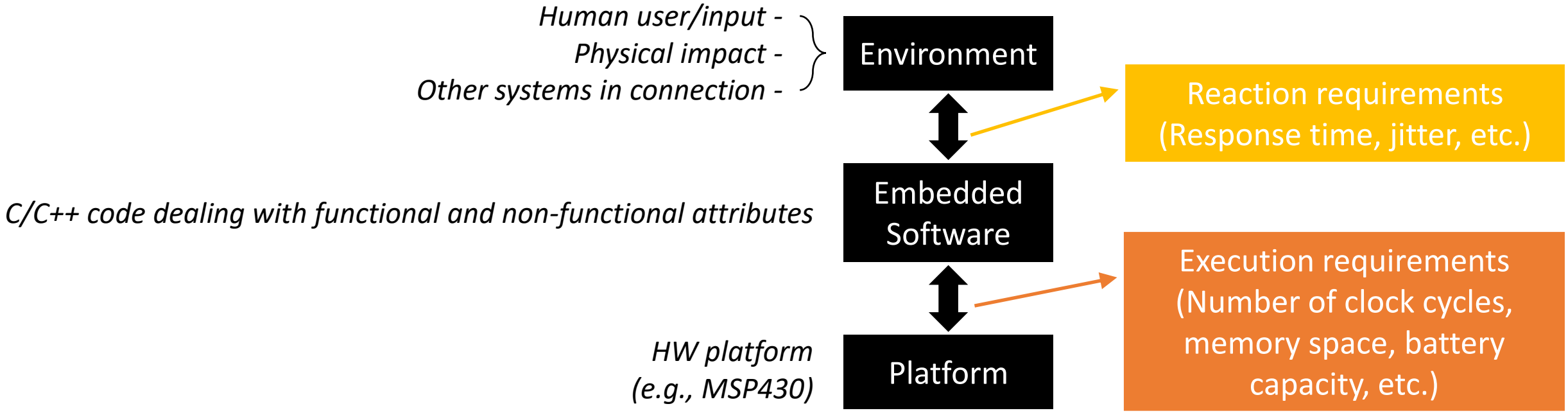
HAVEN Research Group
https://haven.ece.ucf.edu/

HARDWARE VERIFICATION ENGINEERING ASSURANCE

UCF UNIVERSITY OF CENTRAL FLORIDA

Embedded System Design

- From the software point of view
  - Dealing with both functional and non-functional attributes of the system



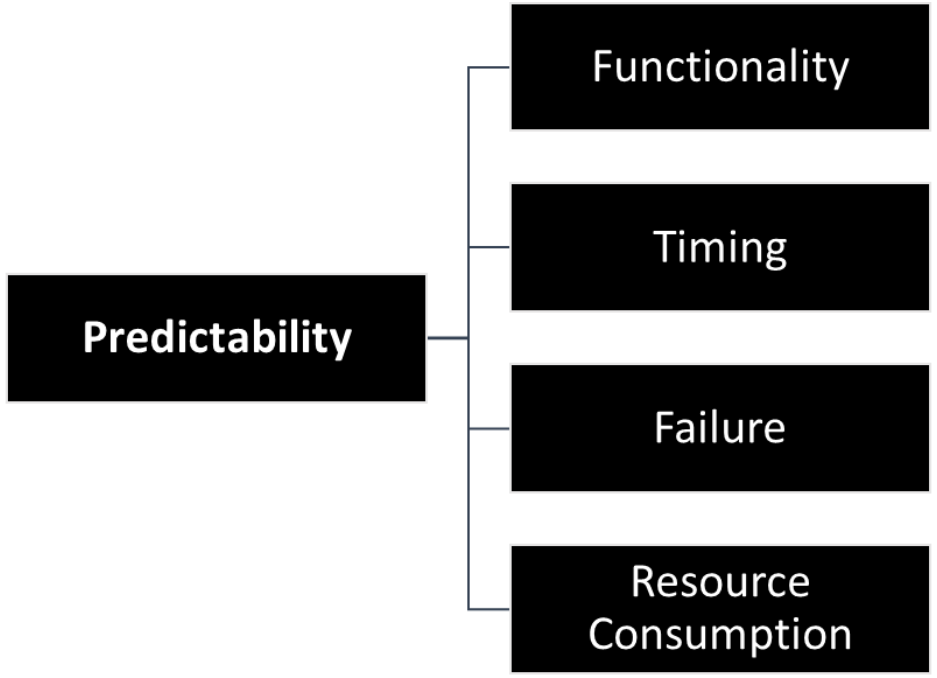
Embedded System Design

- From the software point of view
  - Dealing with both functional and non-functional attributes of the system

- What are non-functional requirements?
  - Environmental requirements (The impact of user/environment on the system)
    - Reactions, inputs of sensors, inputs of users, etc.
    - Can happen at any time (there is no prediction for that)
  - Execution requirements (The impact of outputs generated by the system)
    - An operation as a consequence of a check/computation

Predictability in Embedded System Design

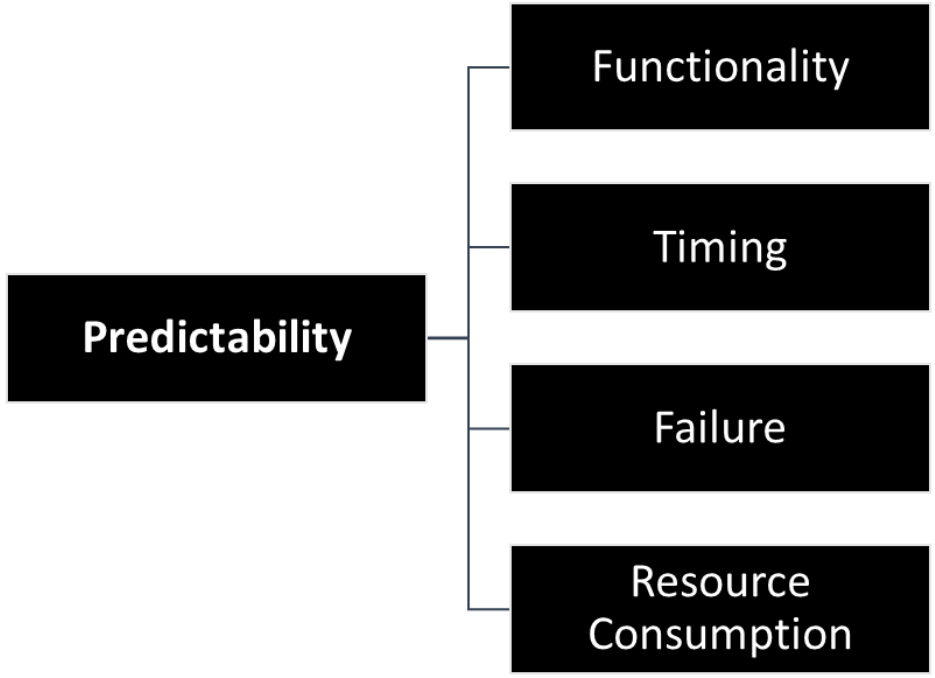
- The challenge: To build time-deterministic systems
- Predictable properties
  - What output the systems produce?
    - Expected based on the data sheets and user guides
  - When does the program provide the output?
    - Expected based on the data sheets and user guides



Predictability in Embedded System Design

- The challenge: To build time-deterministic systems
- Embedded systems are mostly built on platforms that are inherently **unpredictable**.

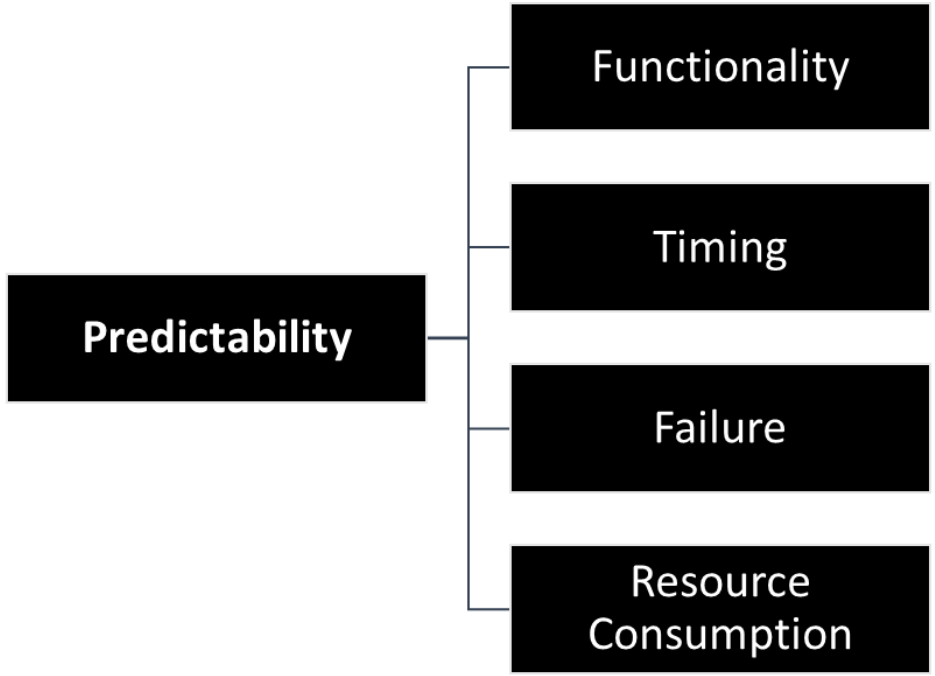
- Sources of non-determinism
  - Input non-determinism
  - Unobservable implementation non-determinism
  - Don’t care non-determinism
  - Observable implementation non-determinism



Predictability in Embedded System Design

- The challenge: To build time-deterministic systems
- Embedded systems are mostly built on platforms that are inherently **unpredictable**.

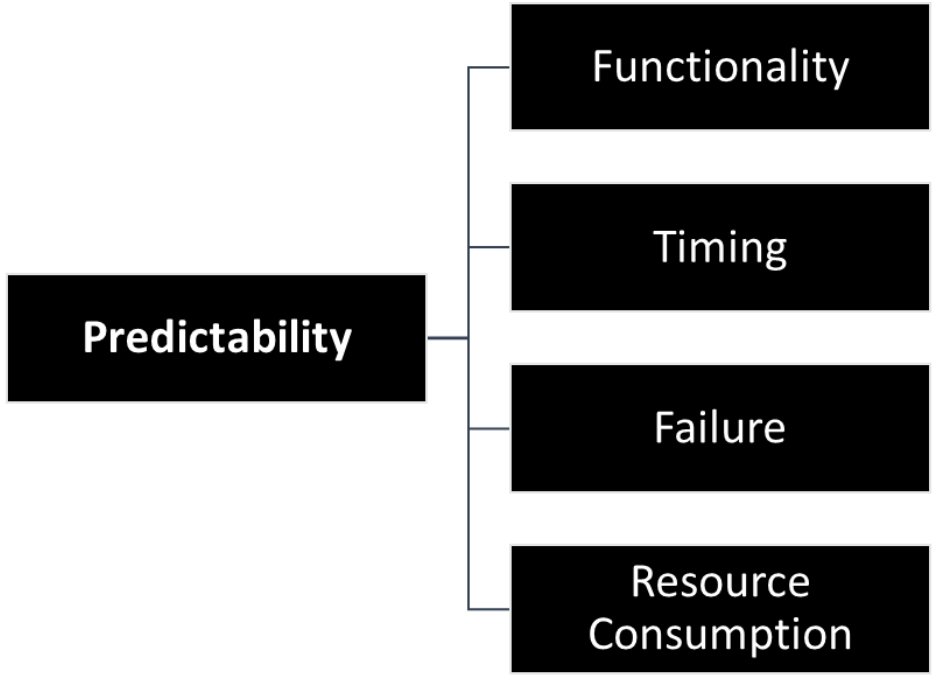
- Sources of non-determinism
  - Input non-determinism
    - The environment is unpredictable
    - Possible solution – timed input-output streams – each input and output event has a time stamp (e.g., FSM)
  - Unobservable implementation non-determinism
  - Don't care non-determinism
  - Observable implementation non-determinism



Predictability in Embedded System Design

- The challenge: To build time-deterministic systems
- Embedded systems are mostly built on platforms that are inherently **unpredictable**.

- Sources of non-determinism
  - Input non-determinism
  - Unobservable implementation non-determinism
    - Arbitrary decisions in the implementation that are not visible at the output
    - e.g. order of sorting the input list does not affect the output
  - Don't care non-determinism
  - Observable implementation non-determinism



Predictability in Embedded System Design

- The challenge: To build time-deterministic systems
- Embedded systems are mostly built on platforms that are inherently **unpredictable**.

- Sources of non-determinism
  - Input non-determinism
  - Unobservable implementation non-determinism
  - Don't care non-determinism
    - States w/ no action (no change to output)
  - Observable implementation non-determinism

Functionality

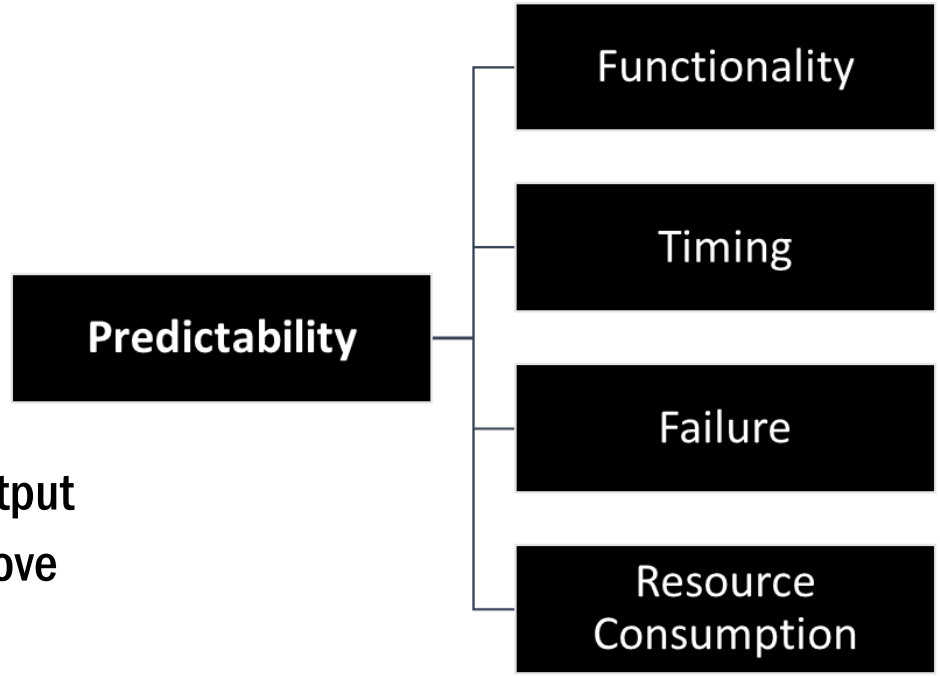
Motion detected	Temp. $< 70° F$	Action
True	True	Turn on heater
True	False	Turn off heater
False	True	**Don't care**
False	False	**Don't care**

Resource Consumption

Predictability in Embedded System Design

- The challenge: To build time-deterministic systems
- Embedded systems are mostly built on platforms that are inherently **unpredictable**.

- Sources of non-determinism
  - Input non-determinism
  - Unobservable implementation non-determinism
  - Don't care non-determinism
  - Observable implementation non-determinism
    - Arbitrary decisions in the implementation that are visible at the output
    - E.g. a pick and place robot – order matters – pick, move, place, move
    - Order of scheduling tasks



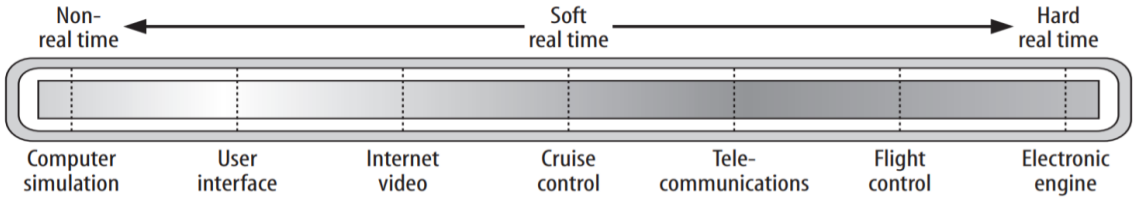
Predictability in Real Time Systems

- A real-time system requires timing correctness in addition to functional correctness.
  - Non-functional reaction requirement
    - $Response\ time \leq Deadline$

Hard real-time system
Missing a deadline is a system failure.
e.g. Pulses generated by a pacemaker

Soft real-time system
Undesirable to miss too many deadlines.
e.g. Updates in the gauge cluster

Characteristic	Hard Real Time	Soft Real Time
Response Time	hard	soft
Peak-Loadd	perform.predictable	degraded
Error Detection	system	user
Safety	critical	non-critical
Redundancy	active	standby
Time Granularity	millisecond	second
Data Files	small/medium	large



Timer: Counting CLK Pulses

- **Recap: LED flashing lab/lecture and the delay function we add manually in it.**

The red LED is mapped to Port 1 Bit 0! **BIT0=00000001**

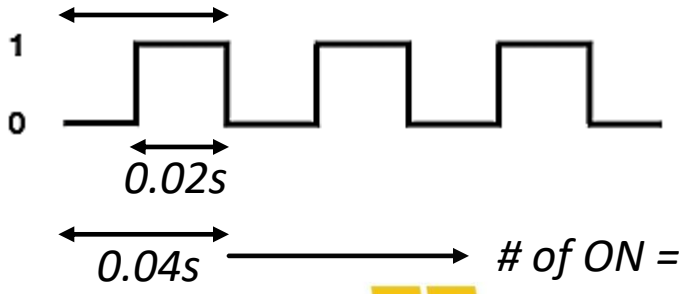
The green LED is mapped to Port 1 Bit 7! **BIT7=10000000**

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
        for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

e.g., each loop counts take 1us. How many times ON per second?

$20{,}000 \times 10^{-6} = 2\times10^{4}\times10^{-6} = 2\times10^{-2}$ s → *for each toggle*

***clock period** = time of each flash (half cycle on and half cycle off)*



# of ON = 1s / 04s = ***25 times***

Timer: Counting CLK Pulses

- **Recap: LED flashing lab/lecture and the delay function we add manually in it.**

The red LED is mapped to Port 1 Bit 0!

The green LED is mapped to Port 1 Bit 7!

BIT0=00000001

BIT7=10000000

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

For having delay more (slower flashing)

- *Nested loop*
- *Using 32-bit variables*
- *Delay cycle function;*

Timer: Counting CLK Pulses

- **Recap: LED flashing lab/lecture and the delay function we add manually in it.**

BIT0=00000001

BIT7=10000000

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

What's the issue with this manual delay (using loops or _delay() function)?

Timer: Counting CLK Pulses

- Alternative: Using Timer

BIT0=00000001

BIT7=10000000

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
        volatile unsigned int i;
        // initialization (reset watchdog, GPIO high-z, etc.
        P1DIR |= redLED; // Direct pin as output
        P1OUT &= ~redLED; // Turn LED Off
         for(;;) {
                // Delay loop
                for(i=0; i<20000; i++) {}
                P1OUT ^= redLED; // Toggle the LED
                }
}
```

What's the issue with this manual delay (using loops or _delay() function)?

This keeps the CPU 'busy' while doing a trivial task!

*Such timing related operations can be implemented using the peripheral **Timer**.*

Timer: Counting CLK Pulses

- Alternative: Using Timer

BIT0=00000001

BIT7=10000000

```
// Code that flashes the red LED
#include <msp430fr6989.h>
#define redLED BIT0 // Red LED at P1.0
void main(void)
{
  volatile unsigned int i;
  TA0CTL = TASSEL_1 | ID_0 | MC_2 | TACLR;
  // Initialization (reset watchdog, GPIO high-z, etc.
  P1DIR |= redLED; // Direct pin as output
  P1OUT &= ~redLED; // Turn LED Off
  for(;;) {
    if((TA0CTL & TAIFG) != TAIFG){
      for(i=0; i<20000; i++) {}
      P1OUT ^= redLED; // Toggle the LED
    }
    // Do other things!
  }
}
```

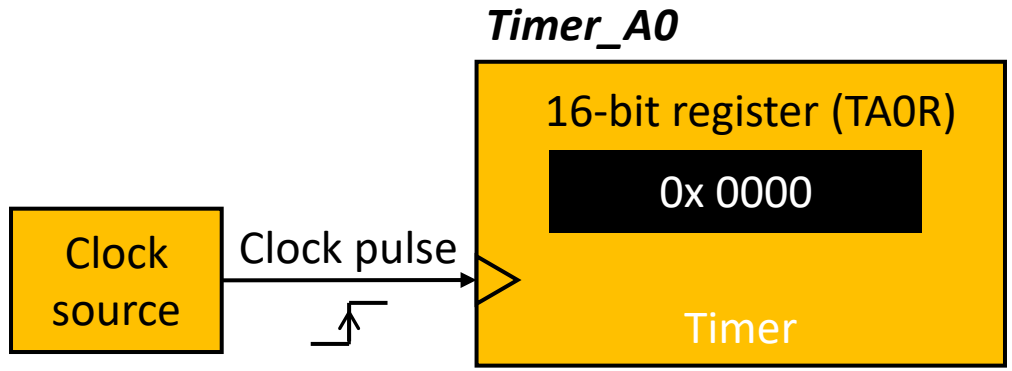
Initialization and enabling the timer

Check on timer (status)

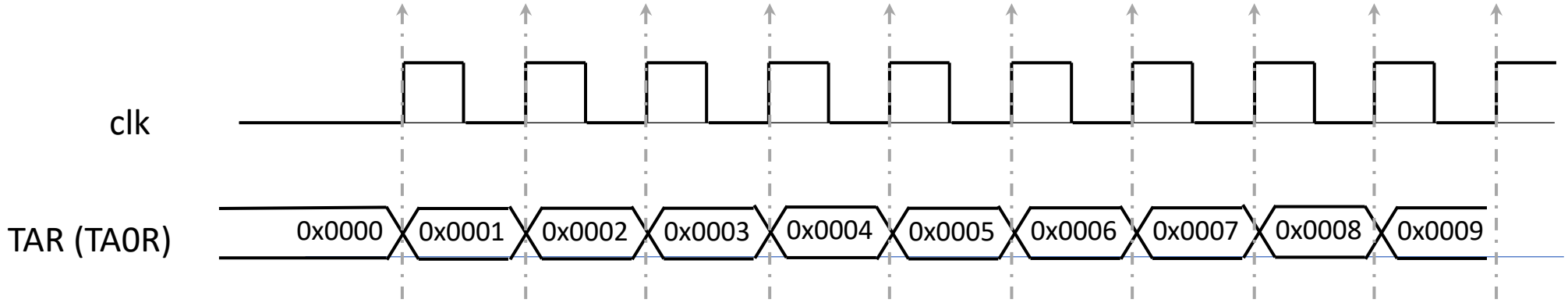
*Such timing related operations can be implemented using the peripheral **Timer**.*

Timer: Counting CLK Pulses

- Timer
  - Counting the clock pulses (clock ticks)
    - For the measurement of elapsed time (elapsed clock pulses)



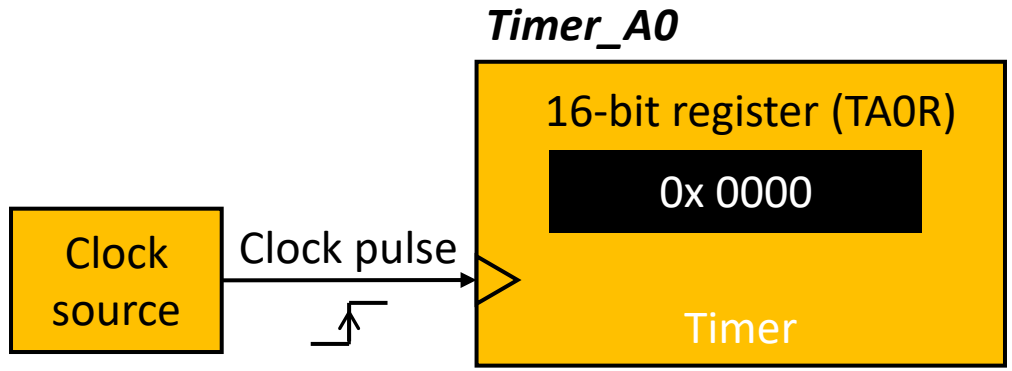
rising edge on the clock signal increments the timer register



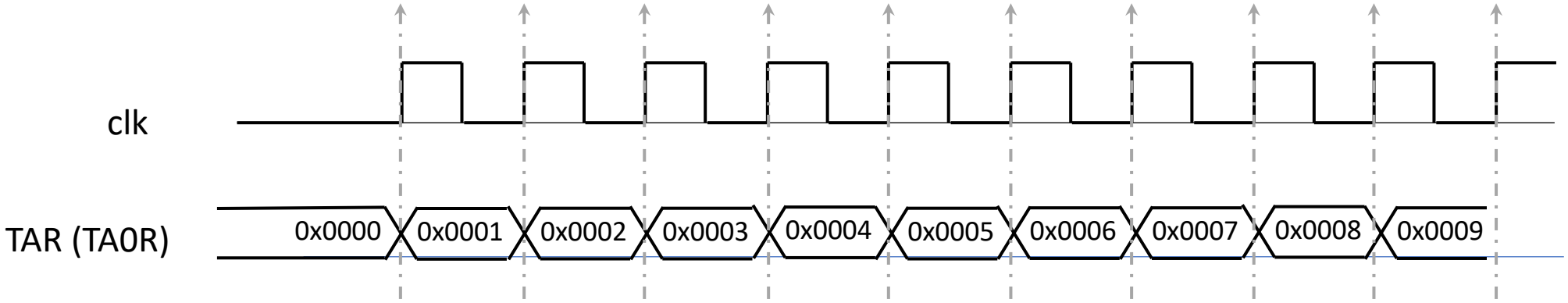
Timer: Counting CLK Pulses

- Timer
  - Counting the clock pulses (clock ticks)
    - For the measurement of elapsed time (elapsed clock pulses)

*MSP430 devices might have other timers like **Timer_B0**, **Timer_A1**, etc.*



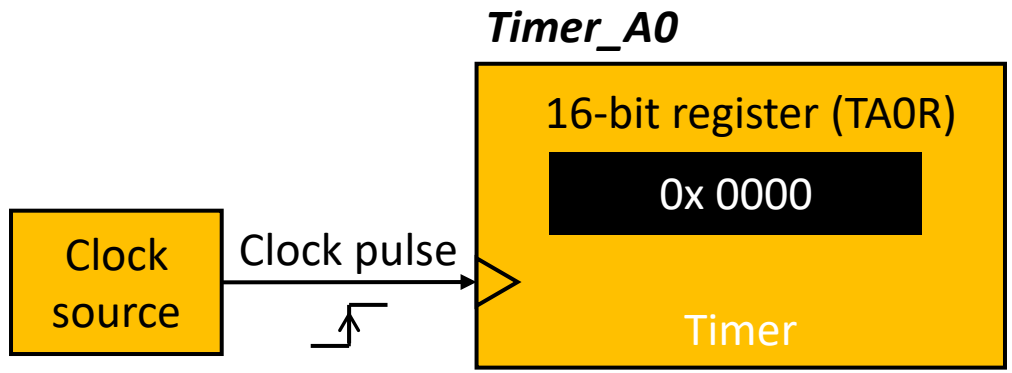
rising edge on the clock signal increments the timer register

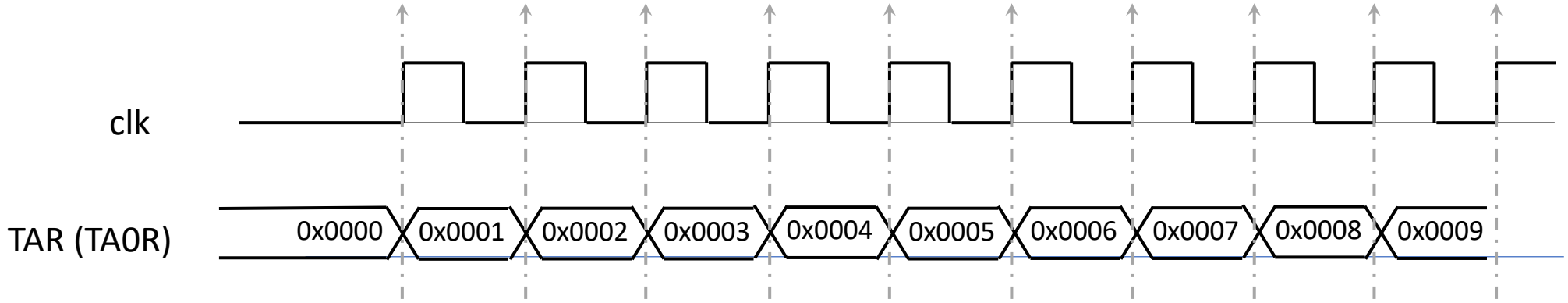


Timer: Counting CLK Pulses

- Timer
  - Counting the clock pulses (clock ticks)
    - For the measurement of elapsed time (elapsed clock pulses)

- *Asynchronous 16-bit timer/counter with four operating modes*
- *Selectable and configurable clock source*
- *Up to seven configurable capture/compare registers*
- *Configurable outputs with pulse width modulation (PWM) capability*
- *Asynchronous input and output latching*
- *Interrupt vector register for fast decoding of all Timer_A interrupts*



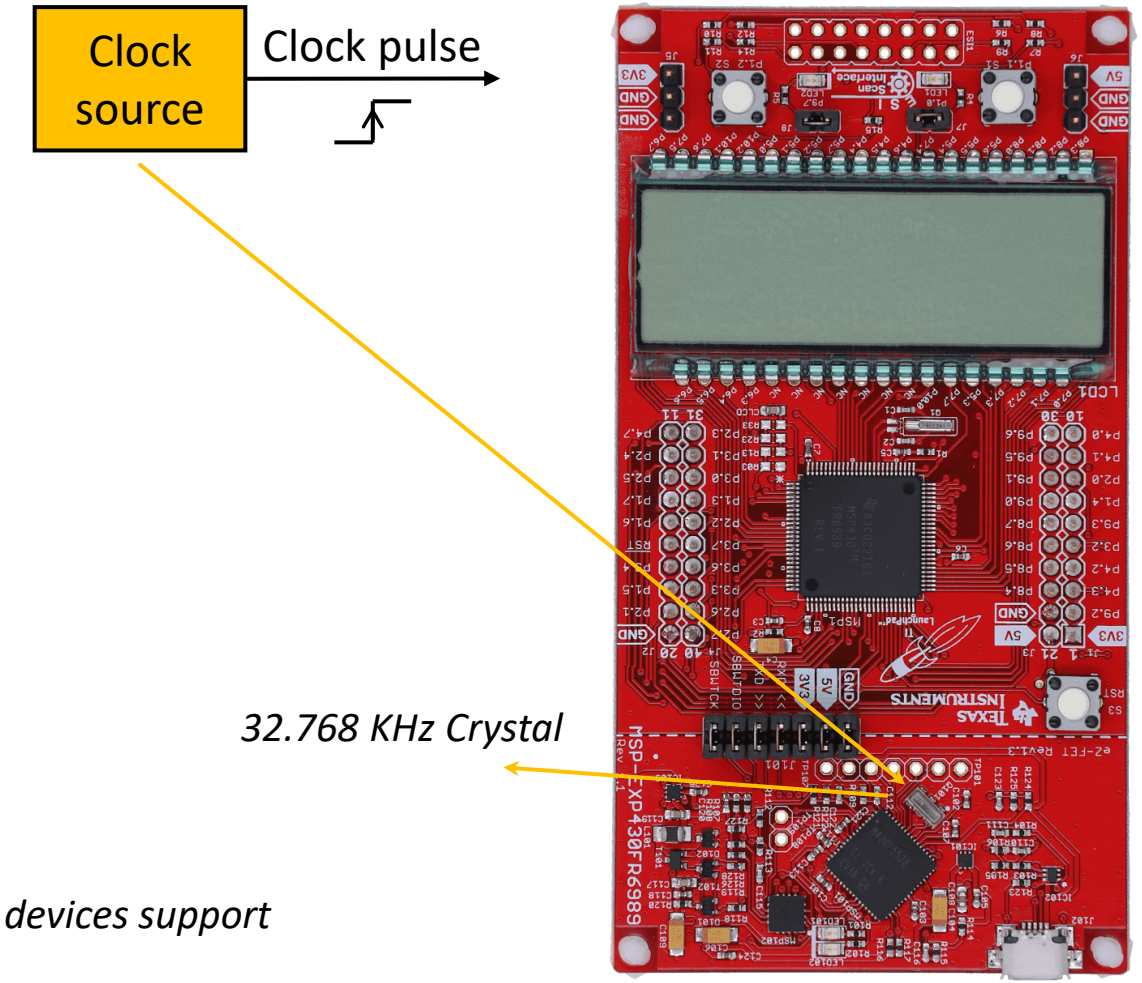


Timer – Clock Source

- Clock signals are generated in multiple ways
  - Crystals
  - RC Circuits

- In MSP430 (Oscillators)
  - Internal oscillator – provided inside the chip
    - DCO (Digitally Controlled Oscillator)
    - VLO (Very-low-power Low-frequency Oscillator)
  - External oscillator – provided externally
    - LFXT (Low frequency oscillator) KHz range
    - HFXT (High frequency oscillator) MHz range

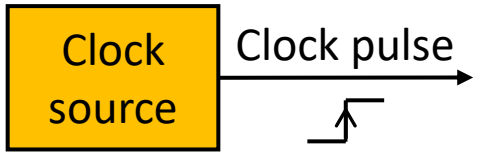
(HFXT is not available in all MSP430 devices, only some devices support it. It is not available in MSP430FR6989 either.)



MSP430FR6989 development board

Timer – Clock Source

- Clock signals are generated in multiple ways
  - Crystals
  - RC Circuits



- In MSP430 (Oscillators)
  - Internal oscillator – provided inside the chip
    - DCO (Digitally Controlled Oscillator)
    - VLO (Very-low-power Low-frequency Oscillator)
  - External oscillator – provided externally
    - LFXT (Low frequency oscillator) KHz range
    - HFXT (High frequency oscillator) MHz range

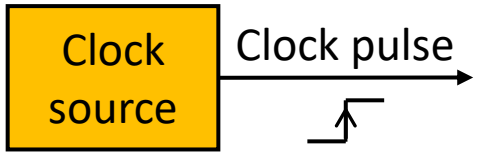
The DCO's frequency can be adjusted digitally via software!

- ***Wide frequency range:*** *Typically from a few hundred kHz to several MHz.*
- ***Fast start-up time:*** *The DCO can start up quickly from a low-power mode, making it ideal for power-sensitive applications.*
- ***Digital control:*** *The frequency can be precisely controlled through software, allowing dynamic adjustment based on the application's needs.*

(HFXT is not available in all MSP430 devices, only some devices support it. It is not available in MSP430FR6989 either.)

Timer – Clock Source

- Clock signals are generated in multiple ways
  - Crystals
  - RC Circuits



- In MSP430 (Oscillators)
  - Internal oscillator – provided inside the chip
    - DCO (Digitally Controlled Oscillator)
    - VLO (Very-low-power Low-frequency Oscillator)
  - External oscillator – provided externally
    - LFXT (Low frequency oscillator) KHz range
    - HFXT (High frequency oscillator) MHz range

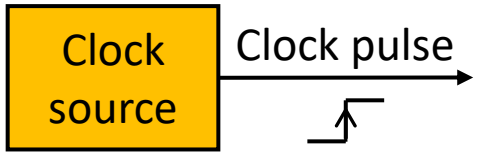
It operates at a very low frequency, typically around 10 kHz!

- ***Low power consumption:*** *As the name suggests, it is optimized for low power, which is crucial in energy-constrained applications.*
- ***Low frequency:*** *Typically around 10 kHz, making it suitable for tasks that do not require high-speed processing, such as keeping track of time in a low-power mode.*

(HFXT is not available in all MSP430 devices, only some devices support it. It is not available in MSP430FR6989 either.)

Timer – Clock Source

- Clock signals are generated in multiple ways
  - Crystals
  - RC Circuits



- In MSP430 (Oscillators)
  - Internal oscillator – provided inside the chip
    - DCO (Digitally Controlled Oscillator)
    - VLO (Very-low-power Low-frequency Oscillator)
  - External oscillator – provided externally
    - LFXT (Low frequency oscillator) KHz range
    - HFXT (High frequency oscillator) MHz range

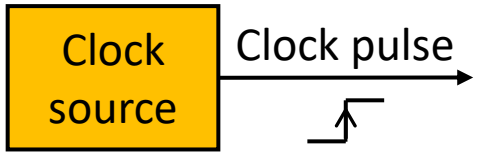
In the kilohertz (kHz) range, typically at 32.768 kHz → RTC

- ***High accuracy:*** *Provides a stable and precise clock signal, important for time-sensitive applications.*
- ***Commonly used with crystal oscillators:*** *The LFXT typically works with an external 32.768 kHz crystal, which offers excellent accuracy and stability.*
- ***Low power consumption:*** *Since it operates at a low frequency, it consumes minimal power, making it suitable for always-on clocking in low-power applications.*

(HFXT is not available in all MSP430 devices, only some devices support it. It is not available in MSP430FR6989 either.)

Timer – Clock Source

- Clock signals are generated in multiple ways
  - Crystals
  - RC Circuits



- In MSP430 (Oscillators)
  - Internal oscillator – provided inside the chip
    - DCO (Digitally Controlled Oscillator)
    - VLO (Very-low-power Low-frequency Oscillator)
  - External oscillator – provided externally
    - LFXT (Low frequency oscillator) KHz range
    - HFXT (High frequency oscillator) MHz range

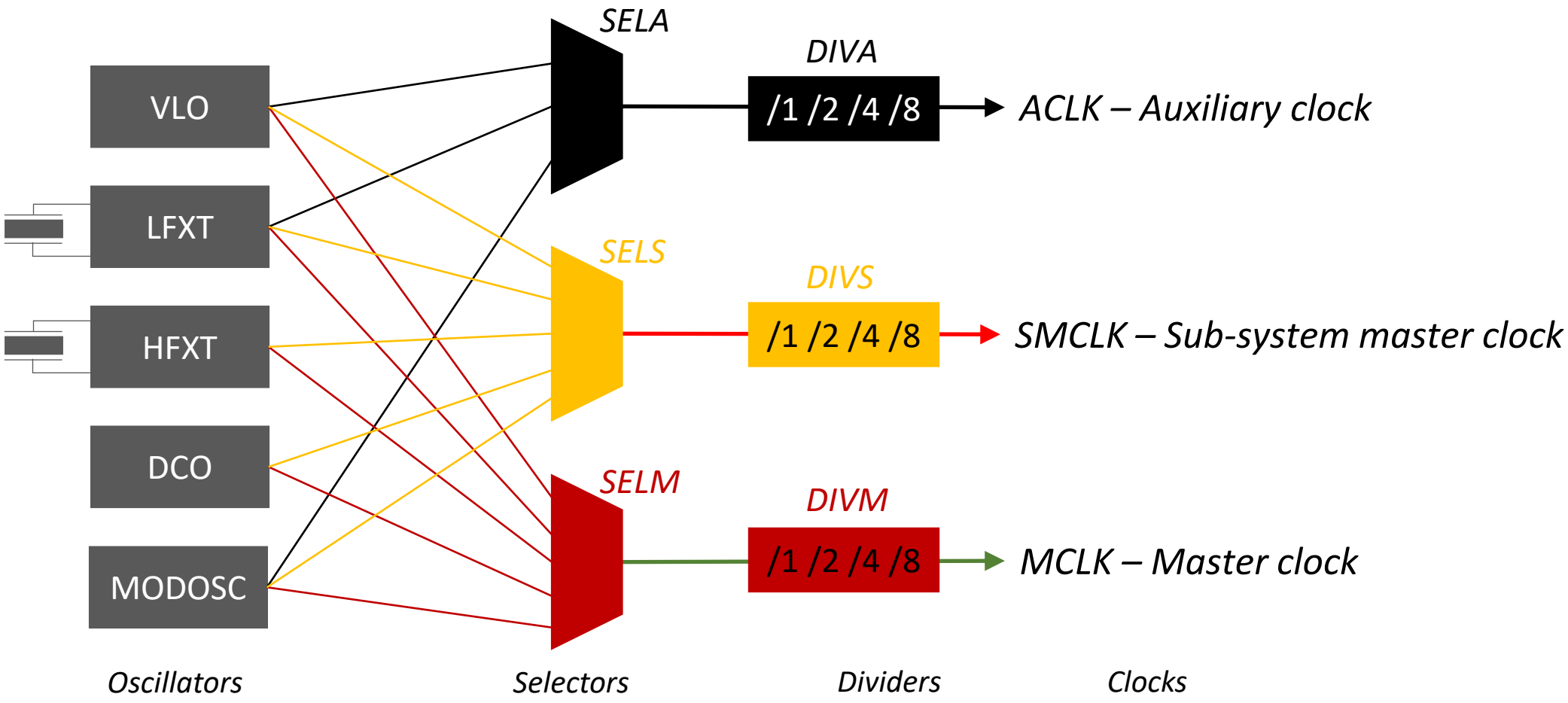
(HFXT is not available in all MSP430 devices, only some devices support it. It is not available in MSP430FR6989 either.)

In the megahertz (MHz) range used for high-speed clocking!

- ***High frequency:*** *Typically operates in the MHz range, which provides the necessary clock speed for demanding applications.*
- ***Stable and precise:*** *Offers accurate timing and clock signals, which are critical for applications that require precise timing, such as communication protocols.*

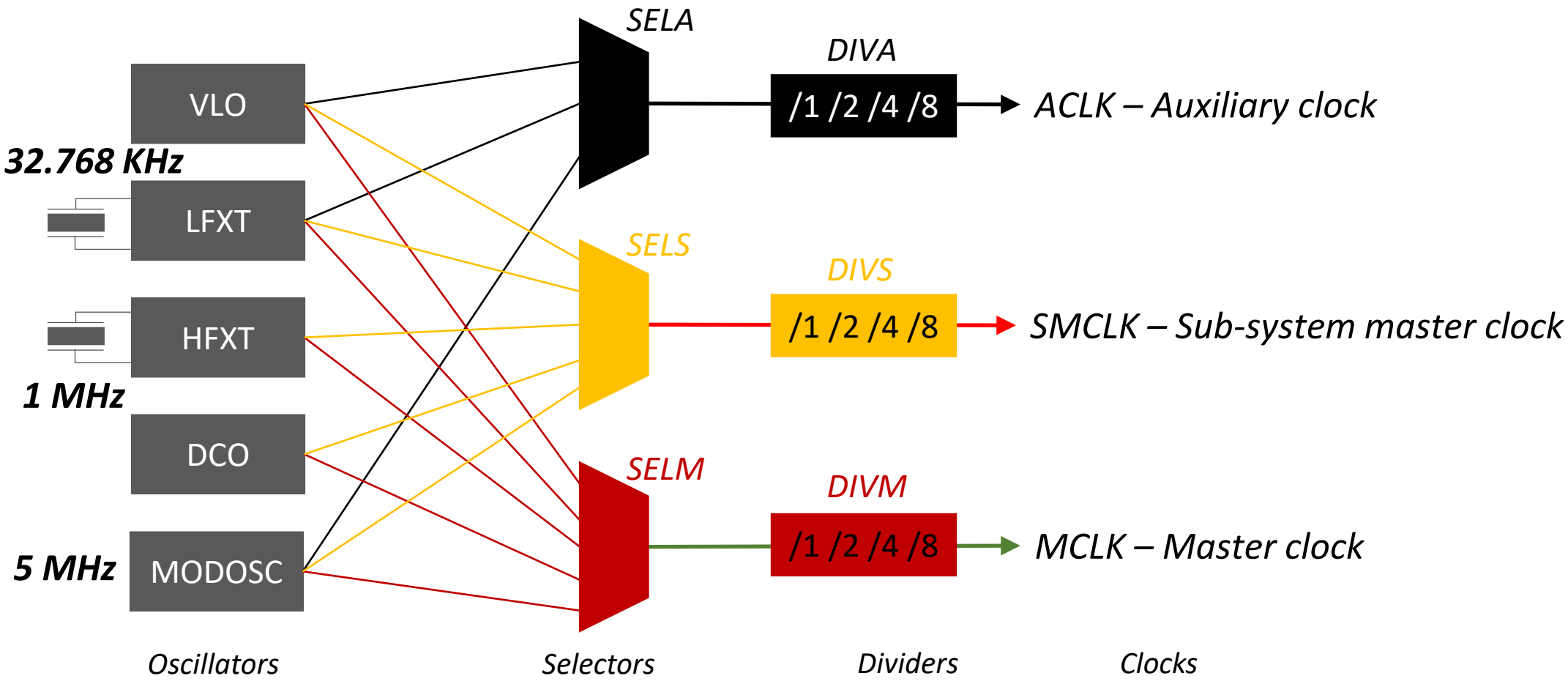
Timer – Clock Source

- **Clock Selection**



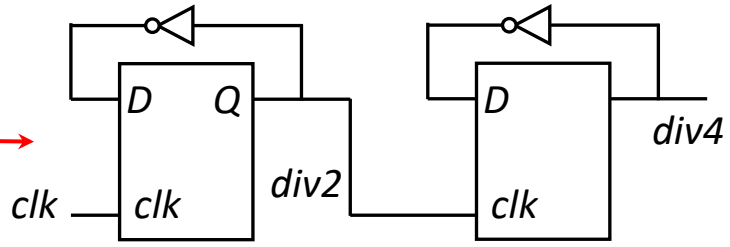
Timer – Clock Source

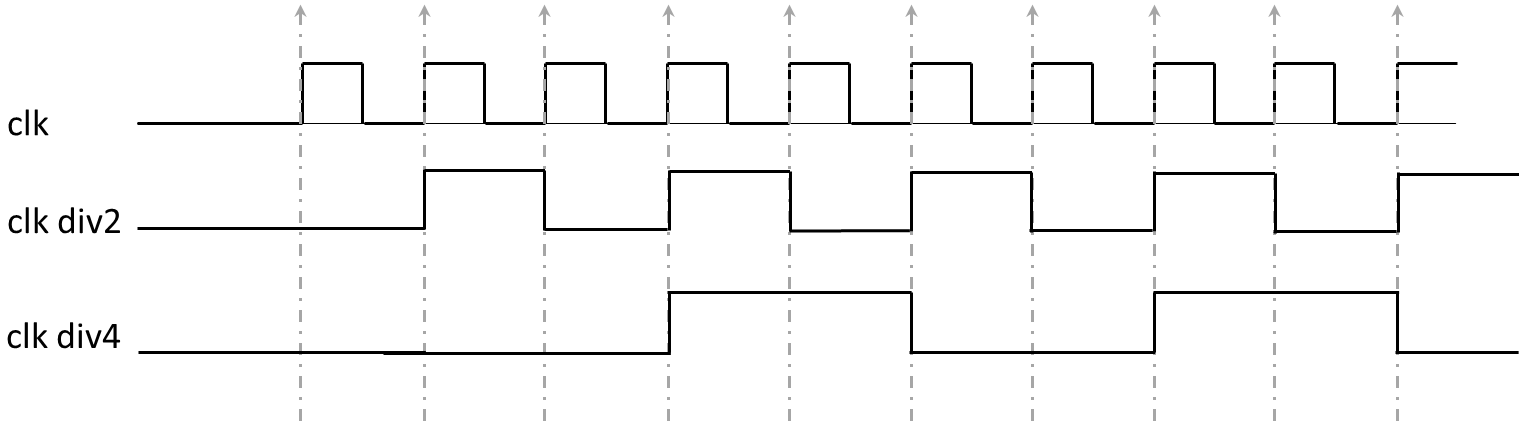
- Clock Selection



Timer_A Configuration in MSP430

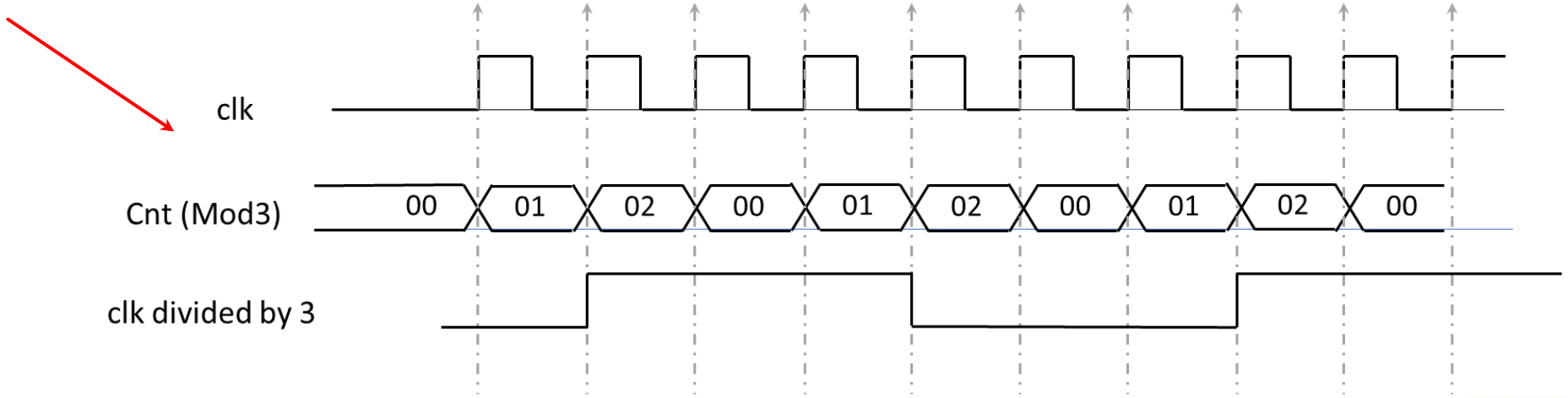
- Clock Selection & Initialization
  - The timer clock can be sourced from ACLK, SMCLK, or externally from TAxCLK or INCLK
    - Selection will be done using TASSEL.
  - Directly or Divided
    - divided by 2, 4, or 8, using the ID bits
      - Shifting-based (~Q connected to D)
    - divided by 2, 3, 4, 5, 6, 7, or 8 using the TAIDEX bits
      - Counting-based (e.g., mod 3 counter)





Timer_A Configuration in MSP430

- Clock Selection & Initialization
  - The timer clock can be sourced from ACLK, SMCLK, or externally from TAxCLK or INCLK
    - Selection will be done using TASSEL.
  - Directly or Divided
    - divided by 2, 4, or 8, using the ID bits
      - Shifting-based (~Q connected to D)
    - divided by 2, 3, 4, 5, 6, 7, or 8 using the TAIDEX bits
      - Counting-based (e.g., mod 3 counter)



Timer_A Configuration in MSP430

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

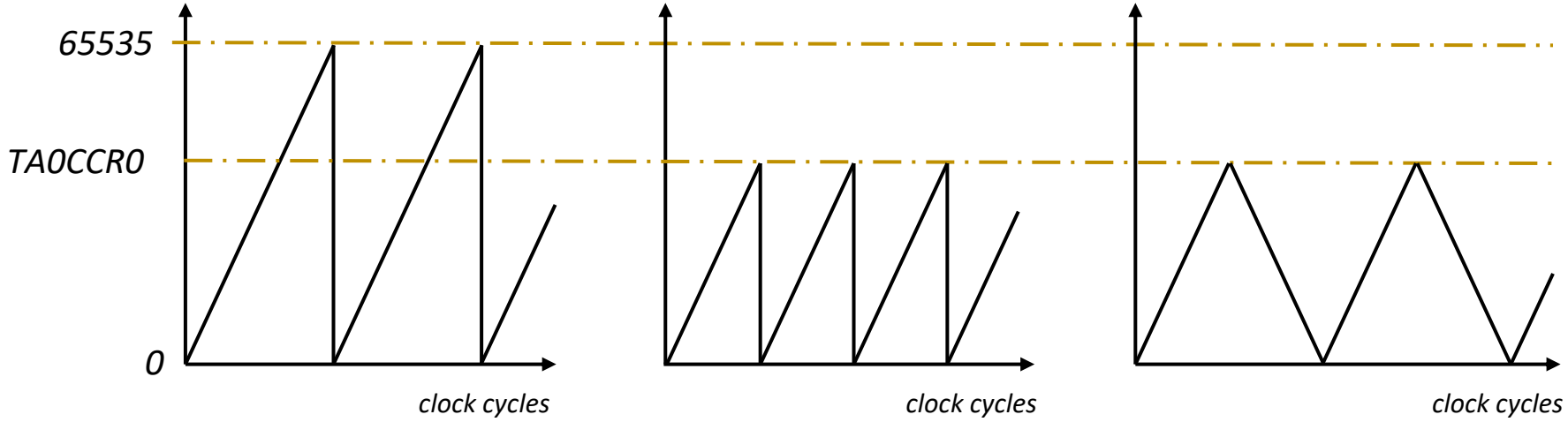
What should be the size of TA0CCR0?

Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

TA0CCR0 is a 16-bit register. TA0CCR0 stands for timer_A 0 capture/compare register for channel 0.



Continuous mode
MC = 10

Up mode
MC = 01

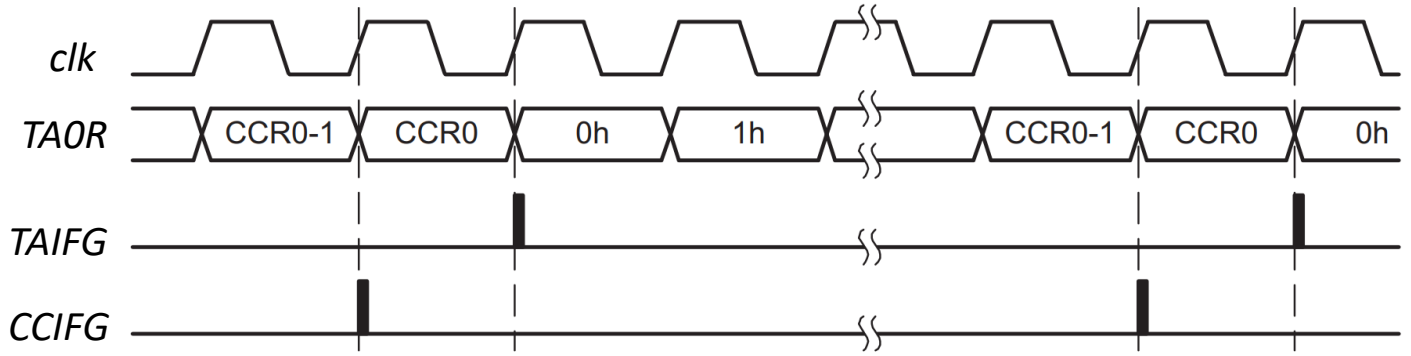
Up/down mode
MC = 11

Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Up mode
  - The timer period must be different from 0FFFFh
  - Counting to TAxCCR0
  - Resetting to Zero

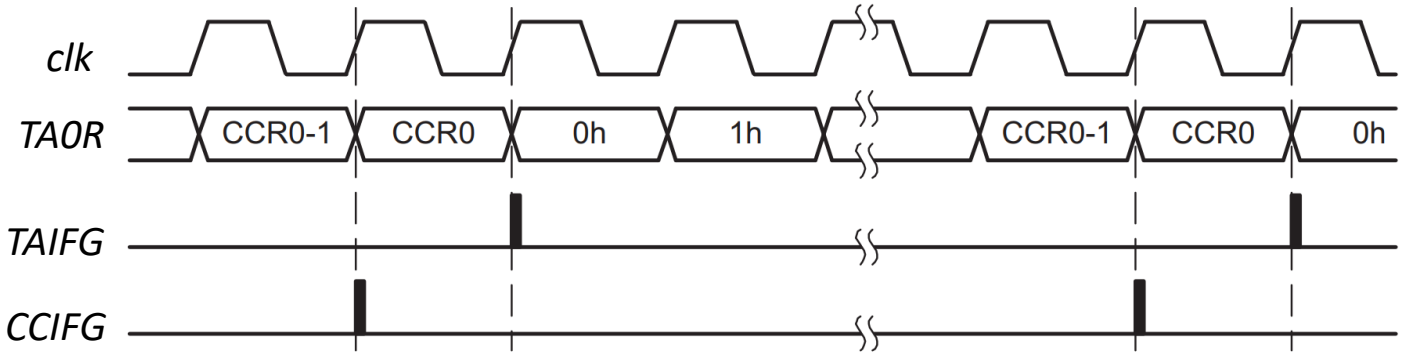


Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Up mode
  - The timer period must be different from 0FFFFh
  - Counting to TAxCCR0
  - Resetting to Zero



The TAIFG interrupt flag is set when the timer counts from TAxCCR0 to zero

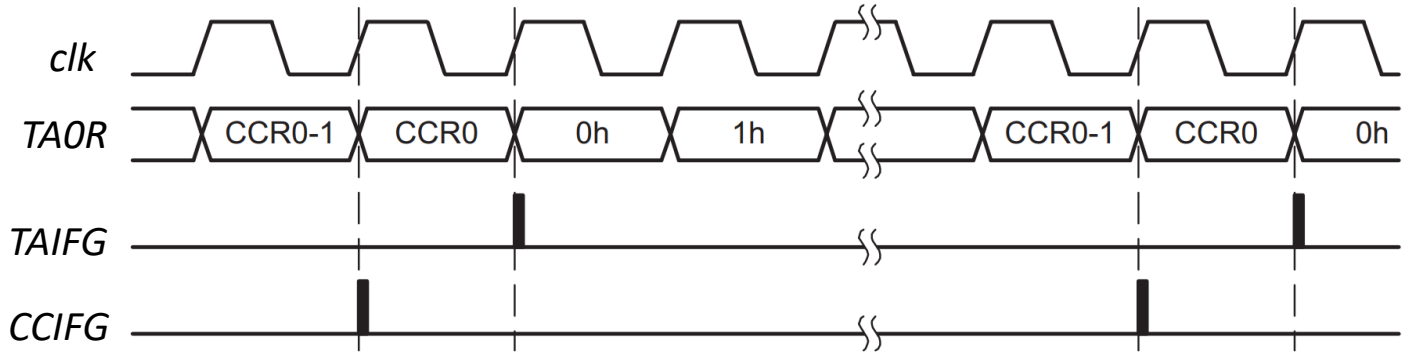
CCIFG interrupt flag is set when the timer counts to the TAxCCR0 value

Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Up mode
  - The timer period must be different from 0FFFFh
  - Counting to TAxCCR0
  - Resetting to Zero



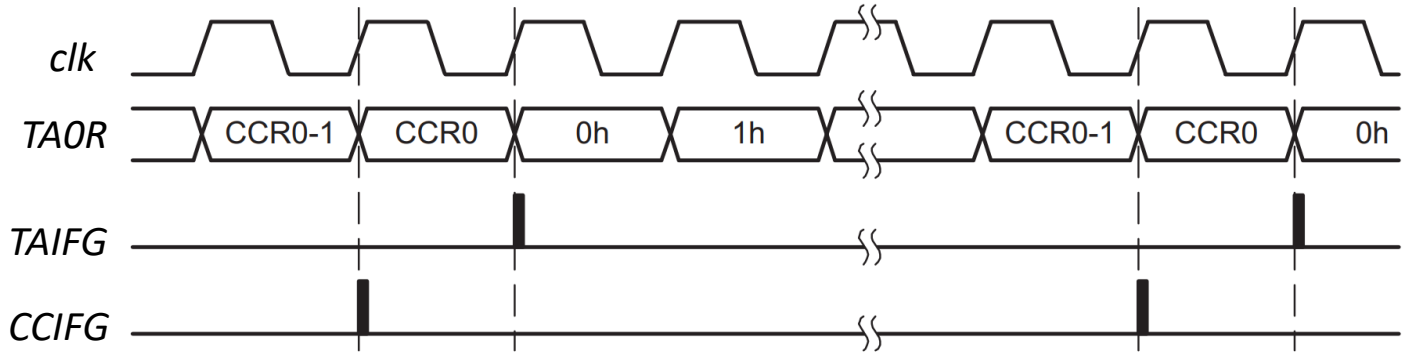
What will happen if we change CCR0 at runtime?

Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Up mode
  - The timer period must be different from 0FFFFh
  - Counting to TAxCCR0
  - Resetting to Zero



What will happen if we change CCR0 at runtime?

- *For greater period → The timer counts up to the new period.*
- *For less period → The timer rolls to zero (immediately or after one clock).*

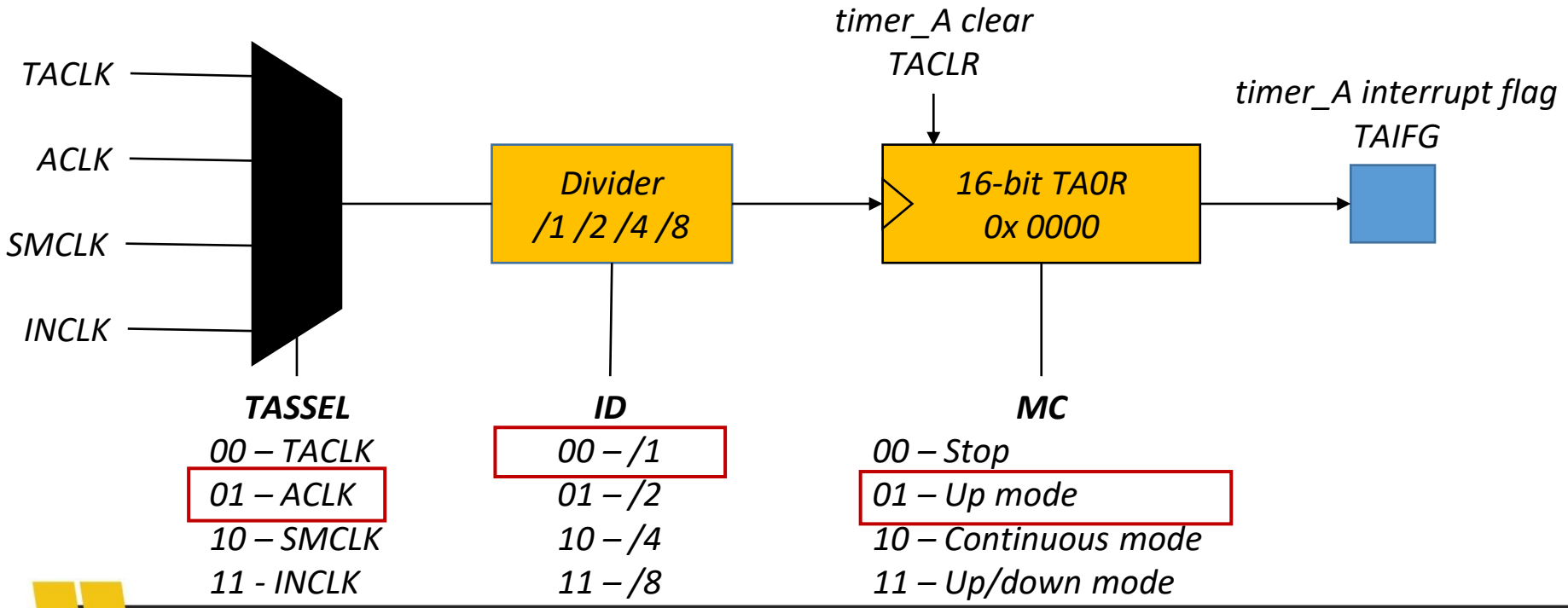
Timer_A Counting Modes

- **Timer Mode Control**
  - **Controlled using MC (2 bits)**

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- **Up mode**

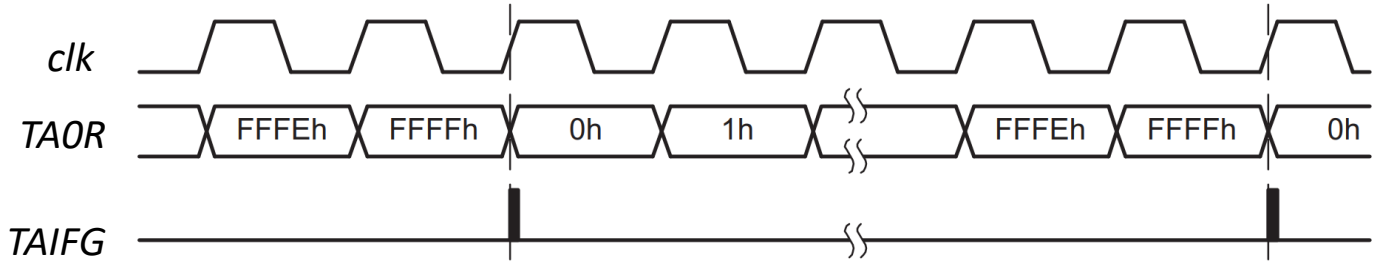


Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Continuous mode
  - The timer period is 0FFFFh
  - The timer repeatedly counts up to 0FFFFh and restarts from zero Resetting to Zero
  - No CCIFG (no TAxCCR0)



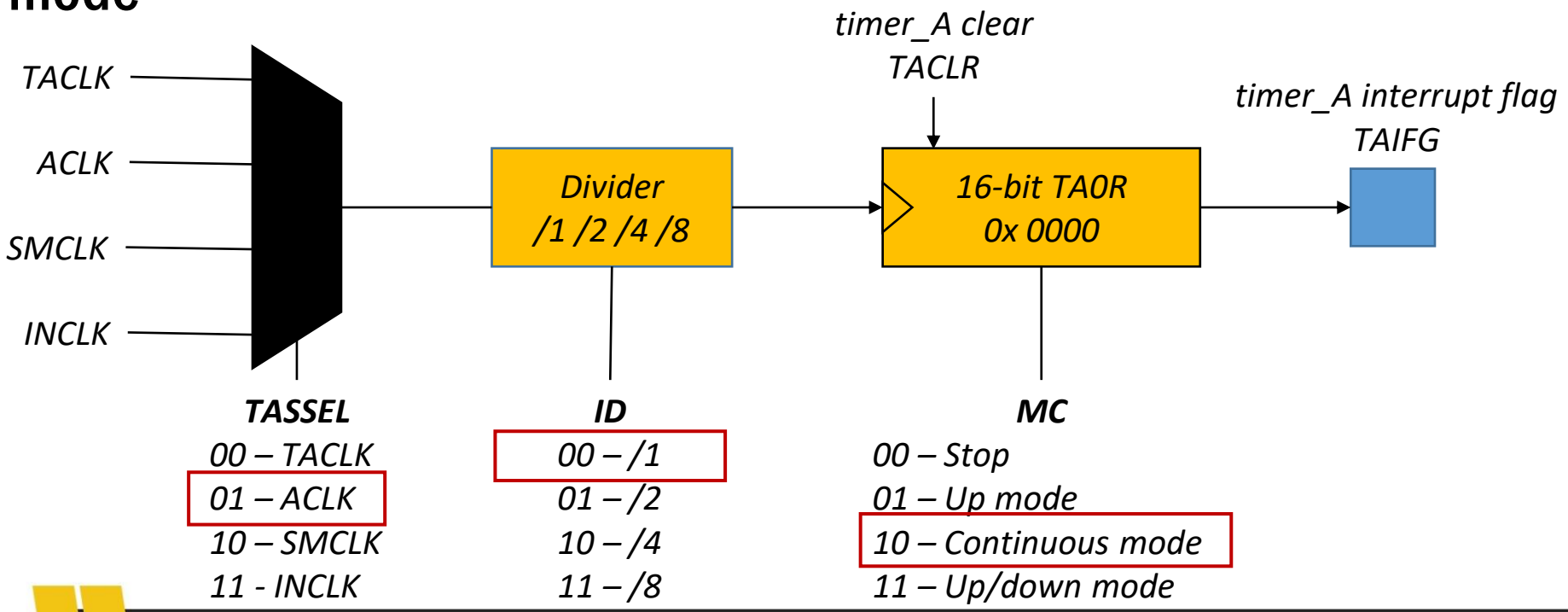
can be used to generate independent time intervals and output frequencies

Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Continuous mode



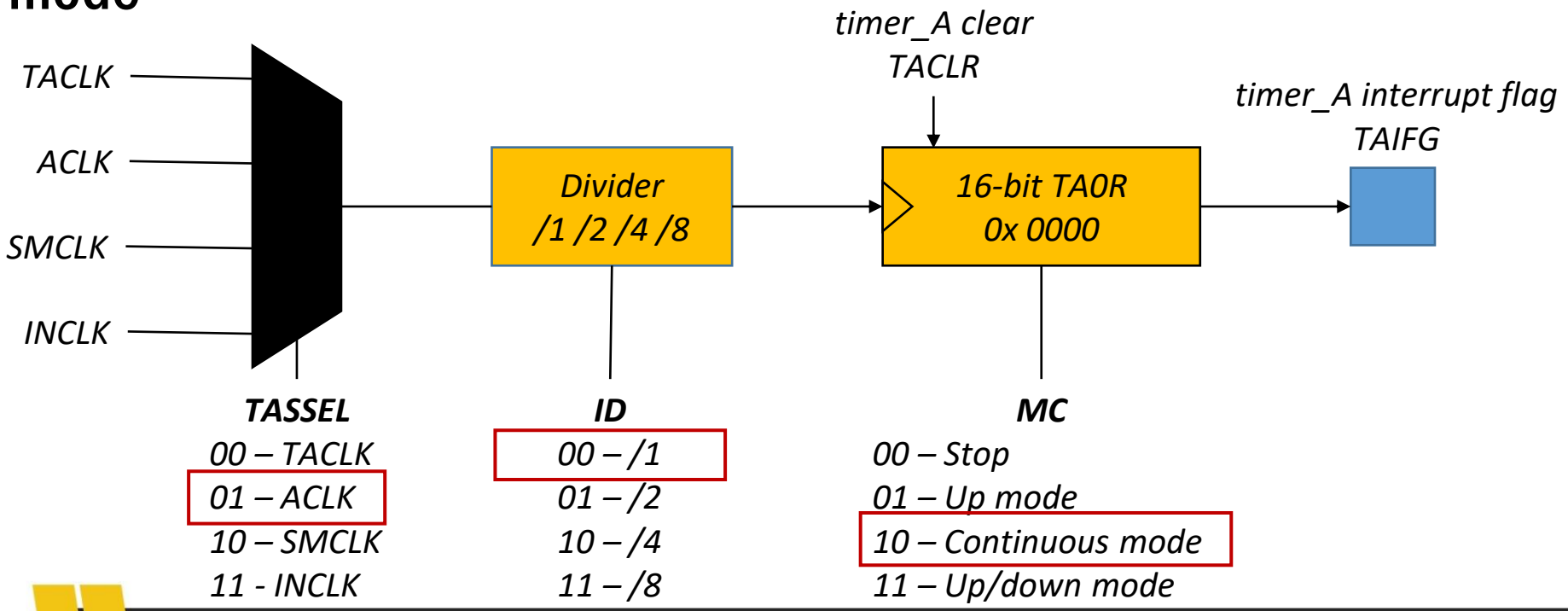
Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- Continuous mode

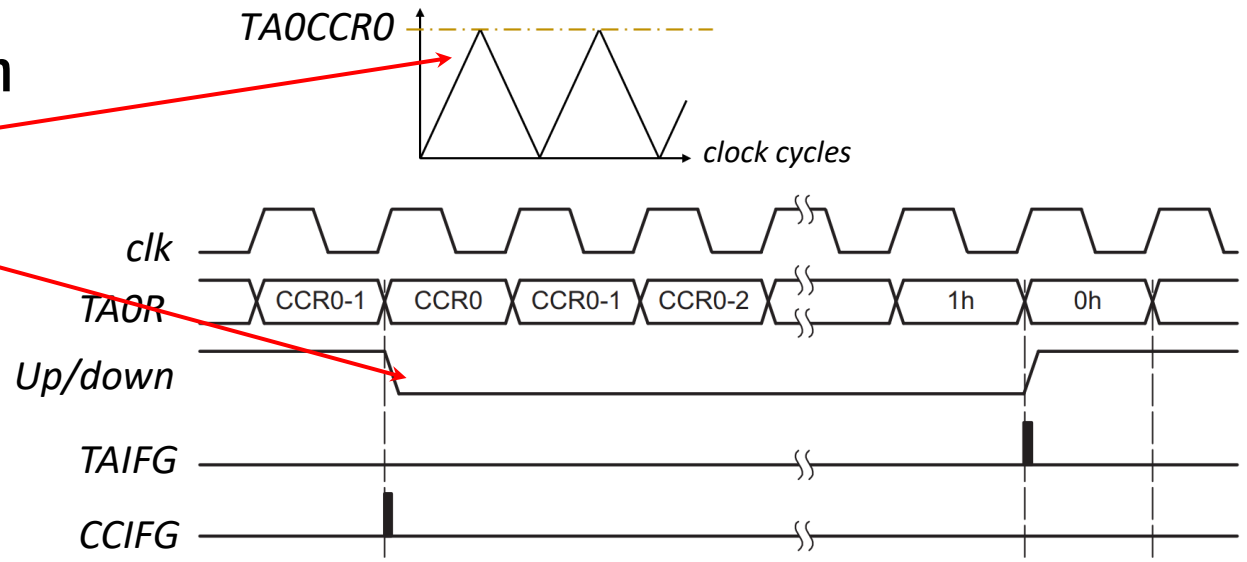


Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Up/down mode
  - The timer period must be different from 0FFFFh
    - and if symmetrical pulse generation is needed
    - Counts up to CCR0 and back down to zero
  - The count direction is latched
    - Can be restarted using TACLR

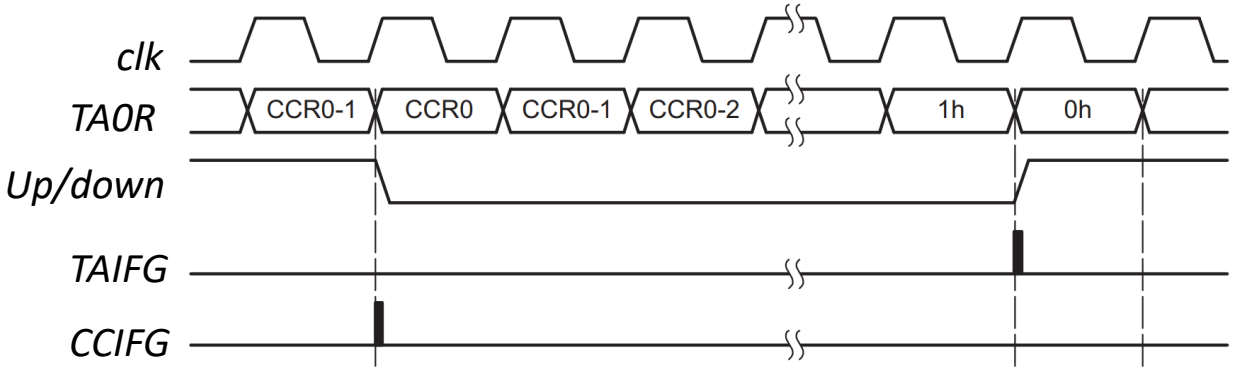


Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Up/down mode
  - The timer period must be different from 0FFFFh
    - and if symmetrical pulse generation is needed
    - Counts up to CCR0 and back down to zero
  - The count direction is latched
    - Can be restarted using TACLR



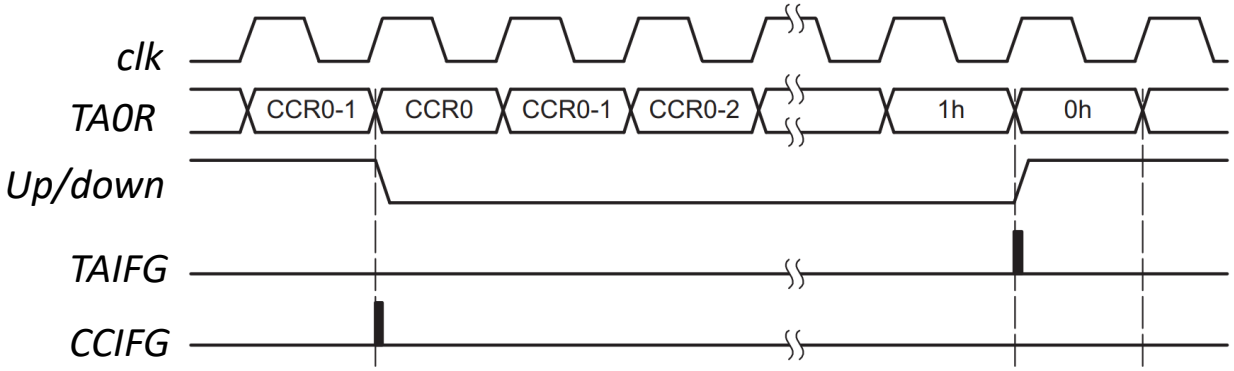
What will happen if we change CCR0 at runtime?

Timer_A Counting Modes

- Timer Mode Control
  - Controlled using MC (2 bits)

MC	Mode	Description
00	Stop	The timer is halted.
01	Up	The timer repeatedly counts from zero to the value of TAxCCR0
10	Continuous	The timer repeatedly counts from zero to 0FFFFh (cycling with no stop)
11	Up/down	The timer repeatedly counts from zero up to the value of TAxCCR0 and back down to zero

- Up/down mode
  - The timer period must be different from 0FFFFh
    - and if symmetrical pulse generation is needed
    - Counts up to CCR0 and back down to zero
  - The count direction is latched
    - Can be restarted using TACLR



What will happen if we change CCR0 at runtime?

- *count down* → *The timer continues counting down to zero. After zero, new CCR0 will be used.*
- *count up* → *For greater period → The timer counts up to the new period (before counting down).*
- *count up* → *For less period → The timer begins counting down to zero (immediately or after one clock).*

Timer Exercise 1

- Configure timer_A
  - in up mode,
  - with ACLK as the clock source
  - And TAIFG flag is set very 0.5 seconds.

TA0CTL

rsvd.						*TASSEL*		*ID*		*MC*		*rsvd.*	*TACLR*	*TAIE*	*TAIFG*
15	14	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*

Timer Exercise 1

- Configure timer_A
  - in up mode,
  - with ACLK as the clock source
  - And TAIFG flag is set very 0.5 seconds.

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- Solution

ACLK is 32768 KHz

32768 clock cycles = 1 second
x clock cycles = 0.5 second

$$x = \frac{0.5 * 32768}{1} = 16384 \text{ clock cycles}$$

Timer Exercise 1

- **Configure timer_A**
  - in up mode,
  - with ACLK as the clock source
  - And TAIFG flag is set very 0.5 seconds.

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- **Solution**

ACLK is 32768 KHz

32768 clock cycles = 1 second
x clock cycles = 0.5 second

$$x = \frac{0.5 * 32768}{1} = 16384 \text{ clock cycles}$$

```
// Code that flashes the red LED using timer
#include <msp430fr6989.h>
#define redLED BIT0                                  // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;                   // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;                        // clear Low Power

        P1DIR |= redLED;                             // Direct pin as output
        P1OUT &= ~redLED;                            // Turn LED Off
        TA0CTL = TASSEL_1 | ID_0 | MC_1;
        TA0CCR0 = 16384 – 1;                         // One extra cycle to set TAIFG
        TA0CTL |= TACLR;

        for(;;) {
                while((TA0CTL & TAIFG) != TAIFG){}   // delay
                P1OUT ^= redLED;                     // toggle redLED
                TA0CTL &= ~TAIFG;
        }
return;
}
```

Timer Exercise 2

- Configure timer_A
  - Timer source is ACLK
  - Divider is /4
  - Operation mode is continuous
  - Clear the TAOR register

TA0CTL

rsvd.						*TASSEL*		*ID*		*MC*		*rsvd.*	*TACLR*	*TAIE*	*TAIFG*
15	14	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*

Timer Exercise 2

- Configure timer_A
  - Timer source is ACLK
  - Divider is /4
  - Operation mode is continuous
  - Clear the TAOR register

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- Solution

TA0CTL = TASSEL_1 | ID_2 | MC_2 | TACLR;

Timer Exercise 3

- Can we use the timer instead of the for loop delay?
- How do we do that?

TA0CTL

rsvd.						*TASSEL*		*ID*		*MC*		*rsvd.*	*TACLR*	*TAIE*	*TAIFG*
15	14	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*

Timer Exercise 3

- Can we use the timer instead of the for loop delay?
- How do we do that?
- Solution

TA0CTL

rsvd.						*TASSEL*		*ID*		*MC*		*rsvd.*	*TACLR*	*TAIE*	*TAIFG*
15	14	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*

```
while((TA0CTL & TAIFG) != TAIFG)
{};
```

Timer Exercise 4

- Recall 32.768 KHz crystal?
  Write a snippet code that creates LED flashing every 2 seconds.

- Solution

TA0CTL

rsvd.						*TASSEL*		*ID*		*MC*		*rsvd.*	*TACLR*	*TAIE*	*TAIFG*
15	14	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*

Timer Exercise 4

- Recall 32.768 KHz crystal?
  Write a snippet code that creates LED flashing every 2 seconds.

- Solution

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

```
// Code that flashes the red LED using timer
#include <msp430fr6989.h>
#define redLED BIT0                                   // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;                    // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;                         // clear Low Power

        P1DIR |= redLED;                              // Direct pin as output
        P1OUT &= ~redLED;                             // Turn LED Off
        TA0CTL = TASSEL_1 | ID_0 | MC_2;
        TA0CTL |= TACLR;

        for(;;) {
                while((TA0CTL & TAIFG) != TAIFG){}    // delay
                P1OUT ^= redLED;                      // toggle redLED
                TA0CTL &= ~TAIFG;
        }
return;
}
```

Timer Exercise 4

- Recall 32.768 KHz crystal?
  Write a snippet code that creates LED flashing every 2 seconds.

- Solution

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

```
// Code that flashes the red LED using timer
#include <msp430fr6989.h>
#define redLED BIT0                                   // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;                    // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;                         // clear Low Power

        P1DIR |= redLED;                              // Direct pin as output
        P1OUT &= ~redLED;                             // Turn LED Off
        TA0CTL = TASSEL_1 | ID_0 | MC_2;
        TA0CTL |= TACLR;

        for(;;) {
                while((TA0CTL & TAIFG) != TAIFG){}    // delay
                P1OUT ^= redLED;                      // toggle redLED
                TA0CTL &= ~TAIFG;
        }
return;
}
```

Decode this line? What's the meaning of it?

Timer Exercise 4

- Recall 32.768 KHz crystal?
  Write a snippet code that creates LED flashing every 2 seconds.

- Solution

ACLK is selected with no dividers and in continuous mode

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

```
// Code that flashes the red LED using timer
#include <msp430fr6989.h>
#define redLED BIT0                                   // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;                    // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;                         // clear Low Power

        P1DIR |= redLED;                              // Direct pin as output
        P1OUT &= ~redLED;                             // Turn LED Off
        TA0CTL = TASSEL_1 | ID_0 | MC_2;
        TA0CTL |= TACLR;

        for(;;) {
                while((TA0CTL & TAIFG) != TAIFG){}    // delay
                P1OUT ^= redLED;                      // toggle redLED
                TA0CTL &= ~TAIFG;
        }
return;
}
```

Timer Exercise 4

- **Recall 32.768 KHz crystal? Write a snippet code that creates LED flashing every 2 seconds.**

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

```
// Code that flashes the red LED using timer
#include <msp430fr6989.h>
#define redLED BIT0                                  // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;                   // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;     // clear Low Power

        P1DIR |= redLED;                             // Direct pin as output
        P1OUT &= ~redLED;                            // Turn LED Off
        TA0CTL = TASSEL_1 | ID_0 | MC_2;
        TA0CTL |= TACLR;

        for(;;) {
                while((TA0CTL & TAIFG) != TAIFG){}   // delay
                P1OUT ^= redLED;                     // toggle redLED
                TA0CTL &= ~TAIFG;

        }
return;
}
```

- **Solution**

ACLK is selected with no dividers and in continuous mode

1 clk tick = 1/32768 second

time taken for TA0R to count from 0x0000 → 0xffff = 65536 clk ticks

(it is equal to the time until TAIFG becomes 1!)

Timer Exercise 4

- **Recall 32.768 KHz crystal? Write a snippet code that creates LED flashing every 2 seconds.**

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

```
// Code that flashes the red LED using timer
#include <msp430fr6989.h>
#define redLED BIT0                                  // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;                   // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;     // clear Low Power

        P1DIR |= redLED;                             // Direct pin as output
        P1OUT &= ~redLED;                            // Turn LED Off
        TA0CTL = TASSEL_1 | ID_0 | MC_2;
        TA0CTL |= TACLR;

        for(;;) {

                while((TA0CTL & TAIFG) != TAIFG){}   // delay
                P1OUT ^= redLED;                     // toggle redLED
                TA0CTL &= ~TAIFG;

        }
return;
}
```

- **Solution**

ACLK is selected with no dividers and in continuous mode

1 clk tick = 1/32768 second

time taken for TA0R to count from 0x0000 → 0xffff = 65536 clk ticks

(it is equal to the time until TAIFG becomes 1!)

65536 clk ticks / 32768 = 2 seconds (for toggling – flashing LED)

Timer Exercise 5

- LED with a delay of 50ms.
- Timer_A in up mode
- Using SMCLK (1MHz)
  - Divider is 8

TA0CTL

rsvd.						*TASSEL*		*ID*		*MC*		*rsvd.*	*TACLR*	*TAIE*	*TAIFG*
15	14	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*

Timer Exercise 5

- LED with a delay of 50ms.
- Timer_A in up mode
- Using SMCLK (1MHz)
  - Divider is 8
- Solution

TA0CTL

rsvd.						TASSEL		ID		MC		rsvd.	TACLR	TAIE	TAIFG
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

```
// Code that flashes the red LED using timer
#include <msp430fr6989.h>
#define redLED BIT0                                   // Red LED at P1.0
void main(void)
{
        WDTCLTC = WDTPW | WDTHOLD;          // stop watchdog timer
        PM5CTL0 &= ~LOCKLPM5;     // clear Low Power

        P1DIR |= redLED;                              // Direct pin as output
        P1OUT &= ~redLED;                             // Turn LED Off
        TA0CTL = TASSEL_2 | MC_1 | ID_3;
        TA0CCR0 = 6250;

        for(;;) {
                while((TA0CTL & TAIFG) != TAIFG){}          // delay
                P1OUT ^= redLED;                            // toggle redLED
                TA0CTL &= ~TAIFG;
        }
return;
}
```

Timer Exercise 6

- If Timer_A is configured in Continuous Mode with the SMCLK set to 8 MHz and no additional prescaler, how long does it take for Timer_A to overflow (reach 0xFFFF and reset)?

Timer Exercise 6

- If Timer_A is configured in Continuous Mode with the SMCLK set to 8 MHz and no additional prescaler, how long does it take for Timer_A to overflow (reach 0xFFFF and reset)?

After reaching 0xFFFF, it overflows back to 0x0000

No prescaler → 1 timer tick = $\frac{1}{8{,}000{,}000}$ *seconds =* ***125 ns***

Total counts per overflow = 65,536

$T_{overflow}$ *= 65,536 × 125ns*

= ***0.008192 s***

Timer for Lab 3

- Focusing on Timer_A
  - Two modes will be investigated
    - Up mode
    - Continuous mode

Family User's Guide (slau367o)

  - Setup for the clock source

Using ACLK (32.768 KHz)

By default, ACLK is set to a built-in RC oscillator (37.5KHz). We need to select 32.768 KHz, that can be done using this function:

chip's data sheet (page 123)

```
//**********************************
// Configures ACLK to 32.768 KHz crystal
void config_ACLK_to_32KHz_crystal() {
                // By default, ACLK is on LFMODCLK at 4.8MHz/128 = 37.5 KHz
                // Reroute pins to LFXIN/LFXOUT functionality
                PJSEL1 &= ~BIT4;
                PJSEL0 |= BIT4;
                // Wait until the oscillator fault flags remain cleared (settling)
                CSCTL0 = CSKEY;                         // Unlock CS registers
                do {
                CSCTL5 &= ~LFXTOFFG;                    // Local fault flag
                SFRIFG1 &= ~OFIFG;                      // Global fault flag
                } while((CSCTL5 & LFXTOFFG) != 0);
                CSCTL0_H = 0;                           // Lock CS registers
return;
}
```

Timer for Lab 3

- Focusing on Timer_A
  - Two modes will be investigated
    - Up mode
    - Continuous mode

Family User's Guide (slau367o)

Without calling this function, the default ACLK will be 37.5 KHz

- Setup for the clock source

Using ACLK (32.768 KHz)

Enable the crystal oscillator

Wait for stabilization

By default, ACLK is set to a built-in RC oscillator (37.5KHz). We need to select 32.768 KHz, that can be done using this function:

chip's data sheet (page 123)

```
//**********************************
// Configures ACLK to 32.768 KHz crystal
void config_ACLK_to_32KHz_crystal() {
            // By default, ACLK is on LFMODCLK at 4.8MHz/128 = 37.5 KHz
            // Reroute pins to LFXIN/LFXOUT functionality
            PJSEL1 &= ~BIT4;
            PJSEL0 |= BIT4;
            // Wait until the oscillator fault flags remain cleared (settling)
            CSCTL0 = CSKEY;                         // Unlock CS registers
            do {
            CSCTL5 &= ~LFXTOFFG;                    // Local fault flag
            SFRIFG1 &= ~OFIFG;                      // Global fault flag
            } while((CSCTL5 & LFXTOFFG) != 0);
            CSCTL0_H = 0;                           // Lock CS registers
return;
}
```

ACLK source selection

Timer for Lab 3

- Signal Repeater
  - Whatever you push (and hold) – the duration - using push button S1
    - will be replayed on redLED!
  - If you exceed a maximum time (64K cycle or 2 seconds)
    - An error flag will be shown on greenLED!
  - Once error flag is raised
    - not working anymore until reset is pushed by push button S2.

- *Counter in continuous mode*
- *Once push button is pressed → counter starts to count.*
- *Once push button is released → counter will be stopped (TA0R will be recorded).*

- *If TA0R < 0xffff → the redLED will be toggled on for the same amount of count.*
- *If TA0R > 0xffff (TAIFG is triggered) → greenLED will be ON. Waiting for S2 to rst.*

Thank You!

Questions?

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

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

HAVEN ASSURANCE VERIFICATION

UCF UNIVERSITY OF CENTRAL FLORIDA