**The Continuous Mode**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Written By: Shayan Taheri

// Continuous Mode of Timer

// Clock Signal with Known Frequency

// Counting cycles to measure durations

// Register: Counting up by one at each cycle of

// the clock signal.

// TAR: 16-Bit --> Counting to 65,535 and rolling

// back to 0 and continue counting.

// Timer\_A --> 16-Bit register called TAR (Timer\_A Register)

// Register --> Counting up to 65,535 and rolling back

// to 0 and continuing counting.

// Continuous Mode: TAR counting from 0 to

// the largest value of 65535.

// Rolling back to zero and continuing counting.

// TAR: Rolling back to zero while counting.

// 1-bit flag TAIFG (Timer\_A Interrupt Flag).

// Setting by the hardware automatically by the hardware.

// TAR --> Setting to zero through a line of code.

// TAIFG --> It is not set.

// Setting the flag --> Toggling an LED

// Clearing the flag --> Raising it again by

// the hardware when the timer period elapses the next time.

// TACTL (Timer\_A Control) --> For configuring and starting

// the TACTL register.

// TASSEL --> For selecting the clock signal used by the timer

// ACLK (Auxiliary Clock) --> To configure a 32 KHz crystal

// SMCLK (Sub Master Clock) --> Generated from an oscillator

// inside the chip and is set to 1.048576 (2^20)

// Modifying the frequency by the software.

// A clock signal (ACLK or SMCLK) --> Division within

// the timer module

// Division within the timer module by either 1, 2, 3, or 8

// For slowing down the frequency

// Frequency Division by ID Field

// MC (Mode) field --> Selecting the mode --> Setting to 0

// And stopping the timer

// Setting to 2 for running the timer in the continuous mode

// TACLR --> For forcing the TAR to clearing it and making it zero.

// Counting TAR and asserting TACLR.

// The TAR goes to zero and resumes counting.

// Asserting this bit at the start to

// ensure TAR starts to zero.

// TAIFG --> Raising to one by the hardware when TAR is

// rolling back to zero.

// Monitoring this bit to know when the timer period

// has elapsed.

// For simplification of accessing the bit fields

// inside TACTL --> the header files define a bunch

// of masks that make our job easier.

// TASSEL\_1 --> Value of 1 at the position of TASSEL.

// TASSEL\_2 --> Value of 2 at the position of TASSEL.

// ID\_0, ID\_1, ID\_2, ID\_3 --> The values of 0, 1, 2, 3.

// TASSEL --> Timer\_A Source Select (1: ACLK, 2: SMCLK)

// ID --> Input Divider (dividing the input clock frequency

// inside the timer)

// MC --> Mode (0: Stop, 1: Up Mode, 2: Continuous Mode)

// TACLR --> Timer\_A Clear (Setting TAR to 0 when asserted)

// TAIFG --> Timer\_A Interrupt Flag

// Raised when TAR rolls back to zero while counting

// TACLR --> It has a mask at the position of TACLR.

// TAIFG --> It has a mask at the position of TAIFG.

// TACTL = TASSEL\_2 | ID\_2 | MC\_2 | TACLR

// TACTL <-- Micro-controller chip contains multiple

// independent time module.

// The ACLK clock signal --> 32 KHz crystal

// ACLK --> Configuring it to a built-in oscillator

// at a frequency of 5 MHz / 128 = 39 KHz

// schematic --> showing that the 32 KHz crystal is attached

// to pins that have dual functionality.

// Dual functionality for : LFXIN/PJ.4 and LFXOUT/PJ.5

// LFXIN (Low-frequency crystal in)

// LFXOUT (Low frequency crystal out)

// A code that flashes the red LED based on a delay that’s generated by the timer.

// Setting PJSEL1 (Bit 4) to 0 and PJSEL0 (Bit 4) to 1.

**#include** <msp430fr6989.h>

// Configures ACLK to 32 KHz crystal --> Calling the following function helps in configuring the crystal!

**void** **config\_ACLK\_to\_32KHz\_crystal**() {

// By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz

// Reroute pins to LFXIN/LFXOUT functionality

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// Wait until the oscillator fault flags remain cleared

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**;

}

// Flashing the LED with Timer\_A, continuous mode, via polling

**#define** redLED BIT0 // Red LED at P1.0

**#define** greenLED BIT7 // Green LED at P9.7

**void** **main**(**void**) {

WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer

PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

P1DIR |= redLED; // Direct pin as output

P9DIR |= greenLED; // Direct pin as output

P1OUT &= ~redLED; // Turn LED Off

P9OUT &= ~greenLED; // Turn LED Off

// Configure ACLK to the 32 KHz crystal (function call)

config\_ACLK\_to\_32KHz\_crystal();

// Configure Timer\_A

// Use ACLK, divide by 1, continuous mode, clear TAR

TA0CTL = TASSEL\_1 | ID\_1 | MC\_2 | TACLR;

// Ensure flag is cleared at the start

TA0CTL &= ~TAIFG;

// Infinite loop

**for**(;;) {

// Empty while loop; waits here until TAIFG is raised

**while**((TA0CTL & TAIFG) == 0) {}

P1OUT ^= redLED; // Toggle the red LED

TA0CTL &= ~TAIFG; // Clear the flag

}

}

**The Up Mode**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Written By: Shayan Taheri

// Up Mode of Timer

// Clock Signal with Known Frequency

// Counting cycles to measure durations

// Register: Counting up by one at each cycle of

// the clock signal.

// TAR: 16-Bit --> Counting to 65,535 and rolling

// back to 0 and continue counting.

// Timer\_A --> 16-Bit register called TAR (Timer\_A Register)

// Register --> Counting up to 65,535 and rolling back

// to 0 and continuing counting.

// Continuous Mode: TAR counting from 0 to

// the largest value of 65535.

// Rolling back to zero and continuing counting.

// TAR: Rolling back to zero while counting.

// 1-bit flag TAIFG (Timer\_A Interrupt Flag).

// Setting by the hardware automatically by the hardware.

// TAR --> Setting to zero through a line of code.

// TAIFG --> It is not set.

// Setting the flag --> Toggling an LED

// Clearing the flag --> Raising it again by

// the hardware when the timer period elapses the next time.

// TACTL (Timer\_A Control) --> For configuring and starting

// the TACTL register.

// TASSEL --> For selecting the clock signal used by the timer

// ACLK (Auxiliary Clock) --> To configure a 32 KHz crystal

// SMCLK (Sub Master Clock) --> Generated from an oscillator

// inside the chip and is set to 1.048576 (2^20)

// Modifying the frequency by the software.

// A clock signal (ACLK or SMCLK) --> Division within

// the timer module

// Division within the timer module by either 1, 2, 3, or 8

// For slowing down the frequency

// Frequency Division by ID Field

// MC (Mode) field --> Selecting the mode --> Setting to 0

// And stopping the timer

// Setting to 2 for running the timer in the continuous mode

// TACLR --> For forcing the TAR to clearing it and making it zero.

// Counting TAR and asserting TACLR.

// The TAR goes to zero and resumes counting.

// Asserting this bit at the start to

// ensure TAR starts to zero.

// TAIFG --> Raising to one by the hardware when TAR is

// rolling back to zero.

// Monitoring this bit to know when the timer period

// has elapsed.

// For simplification of accessing the bit fields

// inside TACTL --> the header files define a bunch

// of masks that make our job easier.

// TASSEL\_1 --> Value of 1 at the position of TASSEL.

// TASSEL\_2 --> Value of 2 at the position of TASSEL.

// ID\_0, ID\_1, ID\_2, ID\_3 --> The values of 0, 1, 2, 3.

// TASSEL --> Timer\_A Source Select (1: ACLK, 2: SMCLK)

// ID --> Input Divider (dividing the input clock frequency

// inside the timer)

// MC --> Mode (0: Stop, 1: Up Mode, 2: Continuous Mode)

// TACLR --> Timer\_A Clear (Setting TAR to 0 when asserted)

// TAIFG --> Timer\_A Interrupt Flag

// Raised when TAR rolls back to zero while counting

// TACLR --> It has a mask at the position of TACLR.

// TAIFG --> It has a mask at the position of TAIFG.

// TACTL = TASSEL\_2 | ID\_2 | MC\_2 | TACLR

// TACTL <-- Micro-controller chip contains multiple

// independent time module.

// The ACLK clock signal --> 32 KHz crystal

// ACLK --> Configuring it to a built-in oscillator

// at a frequency of 5 MHz / 128 = 39 KHz

// schematic --> showing that the 32 KHz crystal is attached

// to pins that have dual functionality.

// Dual functionality for : LFXIN/PJ.4 and LFXOUT/PJ.5

// LFXIN (Low-frequency crystal in)

// LFXOUT (Low frequency crystal out)

// A code that flashes the red LED based on a delay that’s generated by the timer.

// ...............................................................................

// Setting PJSEL1 (Bit 4) to 0 and PJSEL0 (Bit 4) to 1.

// Setting upper-bound of TAR --> TAR: Counting from zero up to the value

// of register TACCR0 (which is the register for Channel 0)

// Linking up mode to Channel 0.

// Time-line: Running the timer operation: TAR reaches TACCR0

// Rolling back to zero and continuation of counting.

// TAR is rolling back to zero (while counting).

// TAIFG --> Setting the flag by the hardware.

// TAR: 0, 1, 2, ..., TACCR0, 0 (TAIFG set), 1, 2, TACCR0 (TAIFG set), 0, 1, ...

// TACCR0 --> 16-Bit register with a period of 100 cycles.

// TACCR0 = 100 - 1 = 99

// TAR --> Counting between 0 and 99. --> spending one cycle at count.

// Multiple timer modules --> Timer0\_A, Timer1\_A, ...

// TA0CCR0 --> Usage for indication of what has been used for

// Timer0\_A Channel 0 register.

// Up Mode Code --> Similar to continuous mode

// TAR --> Counting from zero to the value of register TACCR0 (i.e. register of Channel0).

// TAR's upper-bound --> Using TA0CCR0 --> Indicating Timer0\_A Channel 0 register

// Up Mode Vs. Continuous Mode --> TAR's upper-bound (TA0CCR0)

// Timer's Period = 1 Second (flashing period is 2 seconds).

// Timeline: TAR reaches TACCR0 and rolling back to zerof for counting continuation.

// Rolling back TAR to zero --> TAIFG flag is set by the hardware.

// Linking the up mode to Channel 0.

// Setting TAR's upper-bound using TA0CCR0.

// ACLK (configured to the 32 KHz crystal) and dividing it by 1.

**#include** <msp430fr6989.h>

// Configures ACLK to 32 KHz crystal --> Calling the following function helps in configuring the crystal!

**void** **config\_ACLK\_to\_32KHz\_crystal**() {

// By default, ACLK runs on LFMODCLK at 5MHz/128 = 39 KHz

// Reroute pins to LFXIN/LFXOUT functionality

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// Wait until the oscillator fault flags remain cleared

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**;

}

**#define** redLED BIT0 // Red LED at P1.0

**#define** greenLED BIT7 // Green LED at P9.7

**void** **main**(**void**) {

WDTCTL = WDTPW | WDTHOLD; // Stop the Watchdog timer

PM5CTL0 &= ~LOCKLPM5; // Enable the GPIO pins

P1DIR |= redLED; // Direct pin as output

P9DIR |= greenLED; // Direct pin as output

P1OUT &= ~redLED; // Turn LED Off

P9OUT &= ~greenLED; // Turn LED Off

// Configure ACLK to the 32 KHz crystal (function call)

config\_ACLK\_to\_32KHz\_crystal();

// Configure Timer\_A: Set timer period

TA0CCR0 = (((1/(32000)\*100000000))/3125)-1; // Calculation of value for clock register!

// Timer\_A: ACLK, div by 1, up mode, clear TAR

TA0CTL = TASSEL\_1 | ID\_0 | MC\_1 | TACLR;

// Ensure flag is cleared at the start

TA0CTL &= ~TAIFG;

// Infinite loop

**for**(;;) {

// Empty while loop; waits here until TAIFG is raised

**while**((TA0CTL & TAIFG) == 0) {}

P1OUT ^= redLED; // Toggle the red LED

TA0CTL &= ~TAIFG; // Clear the flag

}

}