// Lab 6: Advanced Timer Features

// Using advanced timer features: (a) using multiple channels of the timer module for generation of the independent periodic interrupt; (b) using the timer’s output patterns to generate a pulse width modulation (PWM) signal that controls te brightness level of the LED.

// 6.1 Using the Timer with Two Channels

// Multiple Channels 🡪 For the Timer\_A module that allows having independent timing intervals.

// The module 🡪 It has three or five channels and is named Timer\_A3 or Timer\_A5.

// Multiple independent Timer\_A modules in the MCU.

// Timer\_A modules with three channels 🡪 Called Timer0\_A3 and Timer1\_A3.

// MSP430FR6989 🡪 An advanced chip with five Timer\_A modules.

// Three modules 🡪 Having three channels and two modules that they have five channels each!

// Timing independent intervals with multiple channels 🡪 Operation of the timer module in the continuous mode!

// TAR counts 🡪 0 up to 64K (65,535) 🡪 Rolling back to zero!

// The channels schedule their interrupts 🡪 Looking ahead from the current value of TAR.

// Channel 1 🡪 Usage for scheduling periodic interrupts every 20K cycles.

// The interrupts: (First @) 20K; (Second @) 40K; (Third @) 60K.

// Channel 1’s register 🡪 Incremented by 20K

// TAR 🡪 Channeling registers are 16-bit and cannot go beyond 64K.

// TAR + Channel Registers are 16-Bit (cannot go beyond)

// Timer Module 🡪 Operation is in continuous mode

// TAR 🡪 Used to count from 0 up to 64K (65,535) 🡪 Rolling back to zero!

// Channels 🡪 Scheduling their interrupts by looking ahead from the current value of TAR.

// Channel 1 🡪 Scheduling periodic interrupts every 20K cycles.

// Interrupts: (First) 20K; (Second) 40K; (Third) 60K.

// Incrementing: Channel’s 1 register by 20K cycles

// Channel 🡪 Adding 20K to its interval 🡪 Also, we can have: 60K + 20K = 80K!

// Going from 60K to 16K: (First) counting up to 64K (that’s 4K cycles); (Second) rolling back to zero and counting up to 16K for a total of 20K.

// Harmless overflow operation 🡪 Keeping 20K to channel’s register!

// How to validating the operation of Channel 2? Generating periodic interrupts every 30K cycles.

// The first two milestone: 30K and 60K!

// TAR = 30K cycles to count from 60K to 26K 🡪 passing zero!

// Multiple channels 🡪 generating periodic interrupts 🡪 Running the timer in the continuous mode!

// In the continuous mode 🡪 Channel 0, 1, and 2: Not a special channel + Usage for generating periodic interrupts.

// In the up mode 🡪 Channel 0: A special channel and it designates the upperbound of TAR.

// Interrupt Events of Timer\_A 🡪 Multiple interrupt events of Timer\_A:

* Events (rollback-to-zero, channel 0, 1, and 2), trigger (TAR=0, TACCR0, TACCR1, and TACCR2), bits (TAIE/TAIFG in TACTL, ), and vector of multiple interrupt events of Timer\_A are all summarized in Table 6.1.

// Flashing Two LEDs using Two Channels:

* Writing a code that runs Timer\_A using ACLK based on the 32 KHz crystal.
* Using the function from earlier labs 🡪 It configures ACLK to the crystal.
* Channel 0 🡪 For toggling the red LED every 0.1 seconds.
* Channel 1 🡪 For toggling the green LED every 0.5 seconds.
* Using the interrupts and engaging a low-power mode to save power while waiting for the interrupts to occur.

Error connecting to the target:

The target setup (MSP430FG4618) does not match the actual target type (MSP430FR6989)

// Using Timer\_A with 2 channels

// Using ACLK @ 32 KHz (undivided)

// Channel 0 toggles the red LED every 0.1 seconds

// Channel 1 toggles the green LED every 0.5 seconds

**#include** <msp430fr6989.h>

**#define** redLED BIT0 // Red LEDs pin location is at P1.0

**#define** greenLED BIT7 // Green LEDs pin location is at P9.7

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

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

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

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} **while**((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

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

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Configuration and initialization of LEDs as off

P1DIR |= redLED; // Red LED pin is set as output

P9DIR |= greenLED; // Green LED is set as output

P1OUT &= ~redLED; // Red LED starts off

P9OUT &= ~greenLED; // Green LED starts off

// Reroutes the ACLK to the 32kHz crystal

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

// Configure Channel 0 of TimerA\_Module0

TA0CCR0 = 3277-1; // Generates a delay of 0.1 seconds with a CLK of 32kHz

TA0CCTL0 |= CCIE; // Enables the interrupt events

TA0CCTL0 &= ~CCIFG; // Clears the interrupt flag

// Configure Channel 1 of TimerA\_Module0

TA0CCR1 = 16384 - 1; // Generates a delay of 0.5 seconds with a CLK of 32kHz

TA0CCTL1 |= CCIE; // Enables the interrupt events

TA0CCTL1 &= ~CCIFG; // Clears the interrupt flag

// Timer configuration (ACLK) (frequency division by 1) (continuous mode) (clear TAR).

TA0CTL = TASSEL\_1 | ID\_0 | MC\_2 |TACLR;

// Engage a low-power mode 3 since we are using ACLK

\_low\_power\_mode\_3();

**return**;

}

// ISR of Channel 0 (A0 vector)

**#pragma** vector = TIMER0\_A0\_VECTOR

**\_\_interrupt** **void** **T0A0\_ISR**() {

P1OUT ^= redLED; // Toggles on/off the red LED

TA0CCR0 += 3277; // Schedules the next interrupt event

// Hardware clears Channel 0 flag (CCIFG in TA0CCTL0) since it is not a shared vector

}

// ISR of Channel 1 (A1 vector)

**#pragma** vector = TIMER0\_A1\_VECTOR

**\_\_interrupt** **void** **T0A1\_ISR**() {

P9OUT ^= greenLED; // Toggles on/off the green LED

TA0CCR1 += 16384; // Schedule the next interrupt event

TA0CCTL1 &= ~CCIFG; // Clear Channel 1 interrupt flag since its a shared vector

}

// 6.2 Using Three Channels

* Three channels 🡪 Extension of the previous section!
* Channels 0 and 1 🡪 They are used to toggle the red LED and the green LEDs every 0.1 seconds and 0.5 seconds respectively.
* Channel 2 🡪 Generating periodic interrupts every 4 seconds for halting and resuming the flashing.
* LEDs 🡪 For flashing and each has its respective rate 🡪 The LED stays ON for four seconds and then goes back to the OFF state.
* ACLK@ 32 KHz 🡪 It is hard to generate a four-second interval since it corresponds to 131,072 cycles and it cannot be stored in the 16-bit.
* Slowing down the frequency using the input divider.
* ACLK 🡪 It is divided to make 8 KHz (8,192 Hz). As a result, the cycle durations are recomputed for Channel 0 and 1 🡪 They correspond to 0.1 seconds and 0.5 seconds respectively.
* Channel 2 🡪 Configuring it for a four-second interval with the interrupt enabled.
* The ISR of A1 vector 🡪 It services the interrupt events of Channel 1 and 2.
* This ISR 🡪 Used to detect which interrupts actually occurred.
* Maintaining a variable ‘status’ 🡪 Keep tracking of whether the LEDs are flashing out or no!
* Declaring such a variable as ‘Static’ inside the ISR 🡪 Keeping its value between calls.

// Using Timer\_A with 2 channels

// Using ACLK @ 32 KHz (undivided)

// Channel 0 toggles the red LED every 0.1 seconds

// Channel 1 toggles the green LED every 0.5 seconds

**#include** <msp430fr6989.h>

**#define** redLED BIT0 // Red LEDs pin location is at P1.0

**#define** greenLED BIT7 // Green LEDs pin location is at P9.7

**int** **unsigned** **static** **volatile** status;

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

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

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

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} **while**((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

**int** **main**(**void**)

{

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Configuration and initialization of LEDs as off

P1DIR |= redLED; // Red LED pin is set as output

P9DIR |= greenLED; // Green LED is set as output

P1OUT &= ~redLED; // Red LED starts off

P9OUT &= ~greenLED; // Green LED starts off

// Reroutes the ACLK to the 32kHz crystal

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

//Channel 0 configuration of TimerA\_module0

TA0CCR0 = 818; // Generates a delay of 0.1 seconds with a CLK of 8kHz

TA0CCTL0 |= CCIE; // Enables the interrupt events

TA0CCTL0 &= ~CCIFG; // Clears the interrupt flag

//Channel 1 Configuration of TimerA\_module0

TA0CCR1 = 4096; // Generates a delay of 0.5 seconds with a CLK of 8kHz

TA0CCTL1 |= CCIE; // Enables the interrupt events

TA0CCTL1 &= ~ CCIFG; // Clears the interrupt flag

//Channel 2 configuration of TimerA\_module0

TA0CCR2 = 32767; // Generates a delay of 4 seconds with a CLK of 8kHz

TA0CCTL2 |= CCIE; // Enables the interrupt events

TA0CCTL2 &= ~CCIFG; // Clears the interrupt flag

// Timer configuration (ACLK) (frequency division by 4) (continuous mode) (clear TAR)

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

//Engage low power mode 3 since we are using ACLK

\_low\_power\_mode\_3();

}

// Channel 0 ISR (A0 Vector)

**#pragma** vector = TIMER0\_A0\_VECTOR

**\_\_interrupt** **void** **T0A0\_ISR**(){

// Checks if the LEDs are flashing (0) or no (1)

**if**(status == 0)

P1OUT ^= redLED; //Toggles on/off red LED

TA0CCR0 += 818; //Schedule next interrupt event

}

//Channel 1 ISR (A1 Vector)

**#pragma** vector = TIMER0\_A1\_VECTOR

**\_\_interrupt** **void** **T0A1\_ISR**(){

//Detect Channel 1 interrupt

**if**((TA0CCTL1 & CCIFG) != 0)

{

// Checks if the LEDs are flashing or no

**if**(status == 0)

{

P9OUT ^= greenLED; //Toggles on/off green LED

TA0CCR1 += 4096; //Schedule next interrupt event

TA0CCTL1 &= ~CCIFG; //Clear interrupt flag

}

}

//Detect Channel 2 interrupt

**if**((TA0CCTL2 & CCIFG) != 0)

{

//Checks if the LEDs are flashing or no

**if** (status !=0)

{

TA0CCR0 = 818; // Sets back TA0CCR0 to its initial cycle

TA0CCR1 = 4096; // Sets back TA0CCR1 to its initial cycle

status =0; // Sets the status to flashing.

}

**else**

{

P1OUT &= ~redLED; // Turns off the red LED

P9OUT &= ~greenLED; // Turns off the green LED

status++; // Sets the status to no flash.

}

TA0CCR2 += 32767; // Schedule the next interrupt event

TA0CCTL2 &= ~CCIFG; // Clear interrupt flag

}

}

// 6.3 Generating a PWM Signal with Timer\_A

* Timer\_A Module 🡪 It can generate a pulse-width modulation (PWM) signal on its own without any action from the CPU.
* The PWM Signal 🡪 It is used frequently in interfaces.
* Timer 🡪 Generates the PWM and the CPU may remain in low power mode indefinitely. 🡪 Saving significant battery power.
* Period 🡪 It is fixed and is marked by the vertical dashed lines.
* Inside a period 🡪 the duration of high pulse is varied.
* Duty Cycle of Short Cycle 🡪 25% | Duty Cycle of Long Cycle 🡪 50%
* PWM Signal 🡪 It can drive a motor 🡪 High Duty Cycle ~ High Motor Speed
* Period 🡪 It is too long (two seconds) / User: Blinking the LED
* A frequency of 1000 Hz 🡪 A period of 0.001 seconds!
* PWM with period 🡪 Having various brightness levels and no blinking is seen by the user.
* Short Pulse 🡪 A duty cycle of 25% | Long pulse 🡪 A duty cycle of 50%.
* Long Period 🡪 User will notice the blinking of LED!
* A frequency of 1000 Hz === A period of 0.001 seconds
* PWM 🡪 Causing various brightness levels and there will be no blinking!
* **Directing the PWM Signal to the LED** 🡪 Connecting the red and green LEDs to P1.0 and P9.7 respectively.
  + A timer-driven PWM signal diverted to these LEDs only if these pins double as timer channel outputs.
  + Our chip 🡪 100-pin variety
  + Timer Channel 🡪 Marked on the pinout as Tax,y (Timer x, Channel y)
  + Pinout 🡪 Showing that the pin of P1.0 doubles as TA0.1 that is Timer0\_A Channel 1.
  + Diverting this pin to the TA0.1 functionality 🡪 Driving a PWM signal on it 🡪 Reaching the LED and controlling its brightness level.
  + PWM Signal 🡪 Fixing the period and Marking by the vertical dashed lines!
  + Fixed Period | Marked by the vertical dashed lines!
  + A period 🡪 The high pulse’s duration is varied!
  + (Ratio of the high duration) / (Period) = Duty Cycle
  + Short Pulse 🡪 A duty cycle of 25% and the long pulse 🡪 A duty cycle of 50%!
  + Timer-Driven PMW Signal 🡪 Diverting to these LEDs | These pins double as timer channel outputs
  + A pin default functionality is the I/O port
  + Configuring P1.0/TA0.1 as P1.0 at reset!
  + Controlling the LED via P1.0
  + Diverting this pin to the TA0.1 functionality 🡪 (P1DIR bit = 1 | P1SEL1 bit = 0 | P1SEL0 bit = 1)
  + P1SEL1 and P1SEL0 🡪 8-bit each!
  + P1DIR 🡪 Getting three bits for each pin of Port 1!
  + Output Patterns 🡪 PWM Signal generated based on the output patterns supported by Timer\_A!
  + Generating a PWM signal with a frequency of 1000 Hz!
  + Period = 0.001 seconds! | 33 Cycles based on a 32 KHz clock signal!
  + Output Mode 🡪 Having two actions (action1/action2) 🡪 Reset/Set
  + Channel’s Event @ TAR=TACCR1 (for Channel 1)
  + Second Action 🡪 Occurrence when TAR rolls back to zero!
  + A duty cycle (10/33) 30.3%!
  + Using the Set/Reset Mode 🡪 A higher value of TACCR1 🡪 Set/Reset Mode!

// Using Timer\_A with 2 channels

// Using ACLK @ 32 KHz (undivided)

// Channel 0 toggles the red LED every 0.1 seconds

// Channel 1 toggles the green LED every 0.5 seconds

**#include** <msp430fr6989.h>

**#define** PWM\_PIN BIT0 // Defines the PWM\_Pin to bit 0 located at P1.0

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

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

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

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} **while**((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

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

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Divert pin P1.0 to TimerA\_Module0\_Channel1 (TA0.1) functionality

P1DIR |= PWM\_PIN;

P1SEL1 &= ~PWM\_PIN;

P1SEL0 |= PWM\_PIN;

// Reroutes the ACLK to the 32kHz crystal

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

// 33 cycles for the PWM Signal

TA0CCR0 = (33-1); // @ 32 KHz --> 0.001 seconds (1000 Hz)

// Timer configuration (ACLK) (frequency division by 1) (up mode) (clear TAR)

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

// Configuring Channel 1 for PWM Signal

TA0CCTL1 |= OUTMOD\_7; // Sets Output Pattern to Reset/Set

TA0CCR1 = 1; // A values from 0 to 32 (ranging from lowest to highest) in the brightness level.

**for**(;;) {}

**return**;

}

// 6.4 Cycling through Brightness Levels

* Modifying the code that it demos multiple brightness levels.
* Generating a PWM signal with a period of 33 cycles 🡪 Varying Channel 1’s register (TA0CCR1) between 0 and 32 🡪 To obtain various brightness levels!
* Writing a code that cycles between six brightness levels 🡪 Corresponding to TA0CCR1 = 0, 5, 10, …, 30!
* The code should stay for one second at each brightness level and cycles between them infinitely.
* The code remains for one second at each brightness level and cycles between them infinitely!
* Timer0\_A 🡪 Used to generate the PWM signal and its running in the up mode with a frequency of 1000 Hz!
* Timer\_A module 🡪 Generating the one-second interval.
* Running this timer in the up mode and use interrupts.
* ISR 🡪 It cycles TA0CCR1!
* Toggling the green LED + ISR 🡪 Indication of the activities!
* Microcontroller 🡪 Changing the brightness levels!
* To user Timer1\_A 🡪 The configuration register is now TA1CTL and the Channel 0 register is TA1CCR0.
* A0 Vector 🡪 Called TIMER1\_A0\_VECTOR

// Using Timer\_A with 2 channels

// Using ACLK @ 32 KHz (undivided)

// Channel 0 toggles the red LED every 0.1 seconds

// Channel 1 toggles the green LED every 0.5 seconds

**#include** <msp430fr6989.h>

**#define** PWM\_PIN BIT0 // Defines the PWM\_Pin to bit 0 located at P1.0

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

// Reconfigures ACLK to be rerouted to the 32 KHz crystal on the LaunchPad

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

// The default mode of the ACLK is a built-in oscillator at a frequency of 39KHz normally.

// Rerouted the pins to LFXIN/LFXOUT functionality so that the ACLK can be routed to the 32KHz crystal.

// This information can be found using the LaunchPad user's guide (page 29) and the chip's data sheet (page 123).

PJSEL1 &= ~BIT4;

PJSEL0 |= BIT4;

// We need to for the crystal to settle, once it has started.

// Therefore, we will wait until the local and global oscillator fault flags are cleared and remain cleared.

CSCTL0 = CSKEY; // Unlock CS registers, to divert the pins for the crystal functionality.

//Clears the flag and will do so until they remain cleared.

**do** {

CSCTL5 &= ~LFXTOFFG; // Local oscillator fault flag

SFRIFG1 &= ~OFIFG; // Global oscillator fault flag

} **while**((CSCTL5 & LFXTOFFG) != 0);

CSCTL0\_H = 0; // Lock CS registers, returns the pins.

**return**;

}

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

WDTCTL = WDTPW | WDTHOLD; // Stops the watchdog timer. We do this so the MCU doesn't reset itself periodically.

PM5CTL0 &= ~LOCKLPM5; // We enable the general purpose I/O pins.

// Divert pin P1.0 to TimerA\_Module0\_Channel1 (TA0.1) functionality

P1DIR |= PWM\_PIN;

P1SEL1 &= ~PWM\_PIN;

P1SEL0 |= PWM\_PIN;

// Reroutes the ACLK to the 32kHz crystal

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

// Sets up the timer in up mode for our PWM signal at a frequency of 1000 Hz

TA0CCR0 = (33-1); // @ 32 KHz --> 0.001 seconds (1000 Hz)

// Timer Configurations (ACLK) (Frequency Division by 1) (Up mode) (Clear TAR)

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

// Sets the cycles calculated for a 1 second delay using the ACLK at 32kHz.

TA1CCR0 = 32768;

// Timer configuration (ACLK) (frequency division by 1) (up mode) (clear TAR).

TA1CTL = TASSEL\_1 | ID\_0 | MC\_1 | TACLR;

TA1CCTL0 |= CCIE; // Enables the interrupt events

TA1CCTL0 &= ~CCIFG; // Clears the interrupt flag

// Configuring Channel 1 for PWM Signal

TA0CCTL1 |= OUTMOD\_7; // Sets Output Pattern to Reset/Set

TA0CCR1 = 1; // A values from 0 to 32 (ranging from lowest to highest) in the brightness level.

// Engages low power mode 3 since we are using the ACLK.

\_low\_power\_mode\_3();

}

**#pragma** vector = TIMER1\_A0\_VECTOR

**\_\_interrupt** **void** **T1A0\_ISR**() {

TA0CCR1 += 5; // Increment the level of brightness by 5

// Checks when TA0CCR1 equals to the maximum required level of brightness

**if**(TA0CCR1 == 30)

TA0CCR1 = 0; // Sets it back to 0.

// Hardware clears Channel 0 flag (CCIFG in TA0CCTL0)

}