1 Background

Interfacing with sensors is an integral part of embedded systems. Sensors are required if we want to measure aspects of the world around us. The world, unlike a digital system, works in a continuum of states. For example, atmospheric pressure, temperature, and humidity are in a constant state of fluctuation. These fluctuations do not occur in discrete steps: the world around us is analog. If we are storing the sensor readings for later retrieval, we may want to have time information attached to the readings.
For the purposes of this homework, you will be implementing a small temperature sensor that will store temperature readings and transmit them when prompted in your MSP-EXP430G2 Launchpad. The temperature readings will be tagged with a timestamp indicating when the reading was taken. The MSP430G2553 in your Launchpad has an internal analog temperature sensor, however, it does not have a timekeeping unit (Real Time Clock, RTC).
Although you do not have an RTC available, you can construct one using a timer. To get accurate timekeeping, you will need to solder the 32.768kHz crystal that came with your Launchpad in the unpopulated header. We show proper placement of the crystal in Figure 1. Furthermore, to preserve power, you will be shutting down the microcontroller and entering a low power mode, only activating it whenever your device needs to perform a temperature reading, your device is receiving commands, or your device is transmiting the temperature data.

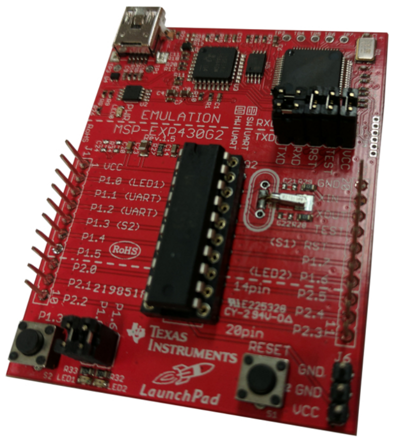


Figure 1: MSP-EXP430G2 Launchpad Development Kit with the 32.768kHz crystal installed.

With this assignment, you will practice how to utilize the low power modes available in the MSP430, as well as how to change the state of the CPU whenever you leave an interrupt service routine.

2 Requirements

We want to make this assignment fair to everyone, so we ask all students to utilize the same board (MSP-EXP430G2) and the same microcontroller (MSP430G2553). This way, everybody has the same set of resources to work with during implementation.
Your program should be able to asynchronously take in commands from the UART interface and respond to them. Your program should also periodically wake up the microcontroller to perform a temperature read and store it in a buffer. Your read must include a timestamp. Your timestamp should be the current time of the read.
Commands are guaranteed to be up to 16 characters in length and will always be terminated with a

carriage return character `\r`. Commands are case insensitive. For simplicity, your program does not need to perform syntax checks or sanity checks on the commands. *Please note that in a production system syntax and sanity checks on user inputs are a must.* The following commands must be implemented:

- `t`: Show the current time of the system.

  Output to the serial terminal the current time of the system using the form `hhmmss`, where `hh` is the two digit hour in 24-hour format, `mm` is the two digit minute in the hour, and `ss` is the two digit second within the minute.

- `s`: Set the current time of the system.

  Set the current time of the system to the provided argument. There will be no space between the command and its argument. The argument will be on the form `hhmmss`, where `hh` is the two digit hour in 24-hour format, `mm` is the two digit minute in the hour, and `ss` is the two digit second within the minute.

- `o`: Show the oldest temperature reading and its timestamp.

  Output to the serial terminal the oldest temperature reading and its timestamp. The output must have the form `hhmmss: T`, where `hh` is the two digit hour in 24-hour format, `mm` is the two digit minute in the hour, `ss` is the two digit second within the minute, and `T` is the measured temperature. The displayed entry must be removed from the list. If no readings have been performed, the message "`No temperatures recorded.`" must be displayed.

- `l`: Show all the temperature readings and their timestamps.

  Output to the serial terminal all the temperature readings and their timestamps. The output should be in chronological order, oldest first. The output must have the form `hhmmss: T`, where `hh` is the two digit hour in 24-hour format, `mm` is the two digit minute in the hour, `ss` is the two digit second within the minute, and `T` is the measured temperature. All entries must be removed from the list. If no readings have been performed, the message "`No temperatures recorded.`" must be displayed.

Your system must also meet the following requirements:

- You must capture a new temperature reading every five (5) minutes and store 32 temperature readings at a minimum. If you have reached the maximum amount of temperature readings, discard the oldest one before storing a new one.
- Your UART must be configured as 9600 baud, 8 data bits, no parity, one stop bit (9600 8N1).

You can use the C programming language, MSP430 assembly, or a combination of the two to complete this project. You may use any function provided to you by TI's C library and runtime or any compiler intrinsic function. The use of any other library is forbidden. Users of the `msp430-elf` toolchain are free to use the `newlib/libgloss` stack, as this is your C library and runtime. You may work in groups of up to three to complete this homework.

2.1 Low Power Mode Requirements

When your device is idling, it must remain in `LPM3`. That is, the CPU, `MCLK`, `SMCLK`, `DCO`, and DC generator are all disabled. Only `ACLK` remains active. Consequently, `ACLK` should be the source for your timer.

Your timer should periodically wake up the microcontroller to update the system time and to trigger any readings from the internal temperature sensor. You do not have to perform any type of conversion on the data read by the sensor.
Upon receiving a full command, your device must leave `LPM3` and go into `Active Mode` to fulfill the command request. After the request is fulfilled, your device must go into `LPM3`.

2.2 Extra Credit

Button shortcut: the button in `P1.3` executes the same action as commands `t`, then `l`.

3 Development Guide

The following is a small series of hints and steps that could help you complete this homework:

- Soldering the crystal:
  - This is good practice for Senior Design, as you will be required to populate your own boards there. You can go to the TI lab in Engineering I to do this portion. You can also ask for help there.
  - Ensure that your body is free of static electricity. Ground yourself and if possible work on top of an anti-static mat.
  - Handle the crystal with tweezers. Do not squeeze it too hard or you may break it.
  - Place the crystal in the pads and hold it in place with small pieces of tape. *You do not solder the crystal tube to the board.*
  - Apply solder flux to the pads.
  - Use a solder iron with a *clean chisel tip* to melt the solder flux. You should use a temperature regulated solder iron preset to around 350 – 380°C. Ensure you do not apply heat directly to the crystal.
  - Once all solder flux has been cleaned, start the soldering process. Ensure the chisel tip is completely tinned and place it on the pads. Do not worry about bridging, as long as you do not have excess solder in your tip, the solder mask (red cover in the board) will ensure it remains on the pads. If you can use solder paste instead of solid solder, apply solder paste to the pads, then melt it in place with the chisel tip. Again, do not worry about bridging. Let the solder mask do the work for you.
  - Examine your solder joint for bridges. Remove any bridges with desoldering braid.
  - Examine your solder joint to ensure proper fixing of the crystal.
  - Clean the area with Isopropyl Alcohol and a thin fiber cloth. Remove any tape and run the provided test code in the microcontroller.
- Configure the UART to trigger interrupts whenever you receive a character. You should already have this portion of the code written from the previous assignment and you are free to reuse it. Store any incoming data in a buffer and set a flag to execute the command whenever ˚ is received.
- Configure the Analog to Digital Converter (ADC) to read from the on-chip temperature sensor. Test your code by dumping the obtained values on a serial console. Rubbing the microcontroller with your finger should show the values increase. Placing a cold *dry* object in the microcontroller should show the values decrease.

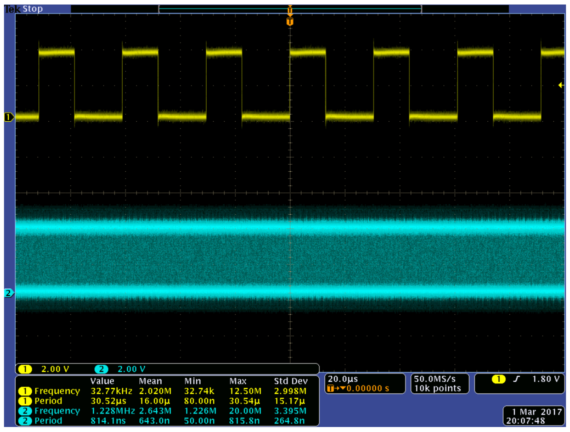
- Configure Timer_A to utilize `ACLK` and to trigger an interrupt at a frequency of 1Hz. Utilize this timer interrupt to update the system time. Set a flag every 5min to perform a read.
- Write your main loop so that your microcontroller enters `LPM3` whenever the loop starts. Modify your interrupt routines to have the CPU leave `LPM3` if any of the necessary conditions are met. Review the compiler intrinsics covered in class during the MSP430 and MSP430X CPU Core lectures in order to perform this task. Furthermore, you should review Sections 2.3 and 2.4 of document number `slau144` for details in operating modes, including the low power modes.
- Come to office hours and ask questions.

4 Submission

To submit your project, pack your source files and compress them. Submit the resulting file to WebCourses. Ensure that the name of all your team members are in every source file. Non-trivial parts of the code should be commented. Furthermore, you must submit a small report (three page maximum) describing your implementation and your testing methodology.

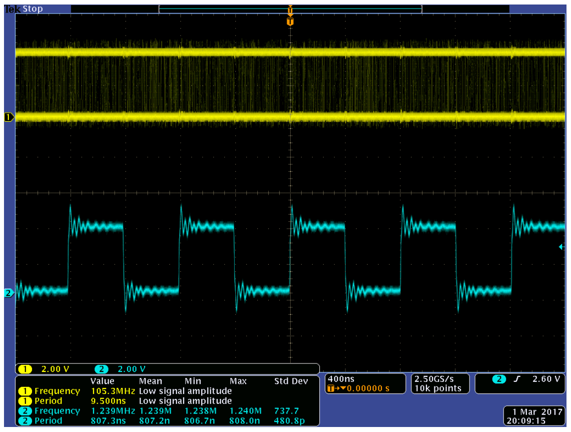
5 Crystal Test Code

The following code can be used to test the crystal. We dump two of the MSP430 internal clocks, `SMCLK` and `ACLK`, through the GPIO. Run this code in your MSP430 after you have soldered your crystal. Attach an oscilloscope probe to `P1.0` and `P1.4`. Measure the frequency in `P1.0`, it should be close to 32.768kHz. Measure the frequency in `P1.4`, it should be close to 1.2MHz. See Your capture should be similar to the one shown in Figure 2.



```
 1  2.00 V      2  2.00 V
             Value      Mean    Min     Max     Std Dev
 1 Frequency 32.77kHz   2.020M  32.74k  12.50M  2.998M
 1 Period    30.52µs    16.00µ  80.00n  30.54µ  15.17µ
 2 Frequency 1.228MHz   2.643M  1.226M  20.00M  3.395M
 2 Period    814.1ns    643.0n  50.00n  815.8n  264.8n
20.0µs  T→▼0.00000 s   50.0MS/s 10k points   1 ƒ 1.80 V
1 Mar 2017 20:07:48
```

(a) `ACLK` and `SMCLK` capture. `SMCLK` is much faster than `ACLK`, so we see it as jitter at this scale.



```
 1  2.00 V      2  2.00 V
             Value      Mean                  Min     Max     Std Dev
 1 Frequency 105.3MHz   Low signal amplitude
 1 Period    9.500ns    Low signal amplitude
 2 Frequency 1.239MHz   1.239M                1.238M  1.240M  737.7
 2 Period    807.3ns    807.2n                806.7n  808.0n  480.8p
400ns  T→▼0.00000 s   2.50GS/s 10k points   2 ƒ 2.60 V
1 Mar 2017 20:09:15
```

(b) `SMCLK` Zoomed In. The oscilloscope is unable to compute the frequency for `ACLK` since not enough samples are captured.

Figure 2: Oscilloscope Capture of two of the internal clocks of the MSP430. `ACLK` (yellow) is captured from `P1.0`, whereas `SMCLK` (cyan) is captured from `P1.4`.

```c
#include <msp430.h>

#define XTOUT   BIT7
```

```
#define XTIN    BIT6

#define ACLK    BIT0
#define SMCLK   BIT4

int main(void) {
    /* Halt the watchdog timer */
    WDTCTL  =   WDTPW | WDTHOLD;

    /* Configure pins for external crystal. For details, see pages 52 to 55 of
     * document number slas735.
     */
    P2DIR   =   XTOUT;              /* XTOUT as output */
    P2SEL   =   XTOUT | XTIN;       /* first function select */
    P2SEL2  &=  ~(XTOUT | XTIN);    /* second function select */

    /* Configure I/O to dump clocks. For details, see pages 42, 43, 46, and 47
     * of document number slas735.
     */
    P1DIR   =   ACLK | SMCLK;       /* set pins as output */
    P1SEL   =   ACLK | SMCLK;       /* first function select */
    P2SEL2  &=  ~(ACLK | SMCLK);    /* second function select */

    /* turn off CPU */
    __bis_SR_register(CPUOFF);
}
```