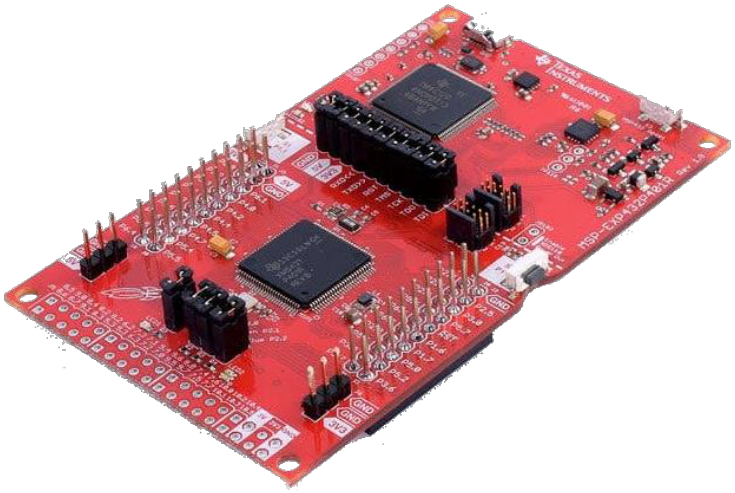


Texas MSP432P401R

FACULTAD DE INFORMATICA

UNIVERSIDAD NACIONAL DE LA PLATA

IDEs de Programacion

Para la programación de la placa se encuentran diversos IDE's de programación, los cuales son:

CODE COMPOSER STUDIO
CCS Cloud
ENERGIA
Keil IDE
IAR Workbench

Todos los IDE's, excepto Energia, son similares en cuanto a las funciones básicas. Tienen herramientas de debug, permiten configurar los periféricos, etc.

En el caso del IDE Energía, su objetivo es brindar una programación sencilla para el programador inexperimentado, por lo tanto provee herramientas muy básicas y no dá, por ejemplo, utilidades para debug. El ambiente es idéntico al Arduino IDE.

CCS Cloud es una versión web del Code Composer Studio. Permite realizar todas las tareas en el navegador web y la compilación del código la realiza en la nube.

Para realizar las pruebas decidimos utilizar el Code Composer Studio ya que es el que posee la mayor cantidad de herramientas, lo provee el fabricante de la placa (Texas Instruments) asegurando compatibilidad y además está basado en Eclipse, plataforma con la que ya estamos acostumbrados a trabajar.

Lenguajes de programación

Los lenguajes de programación disponibles para esta placa son los siguientes

C
C++
Lenguaje Arduino

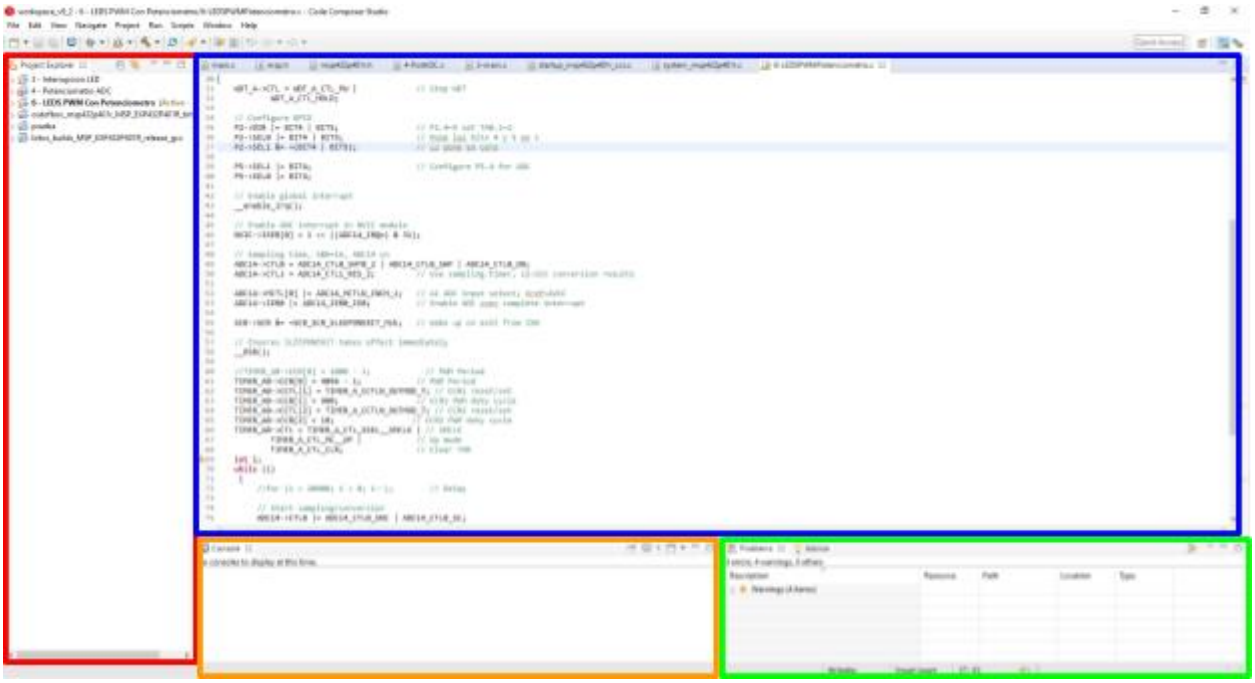
C y C++ pueden ser utilizados en todos los IDE's anteriormente mencionados, excepto Energía.

El lenguaje Arduino puede ser utilizado en Energia IDE, Code Composer Studio y en CCS Cloud.

Code Composer Studio

A continuación se hará un vistazo del IDE y las herramientas más útiles que encontramos.

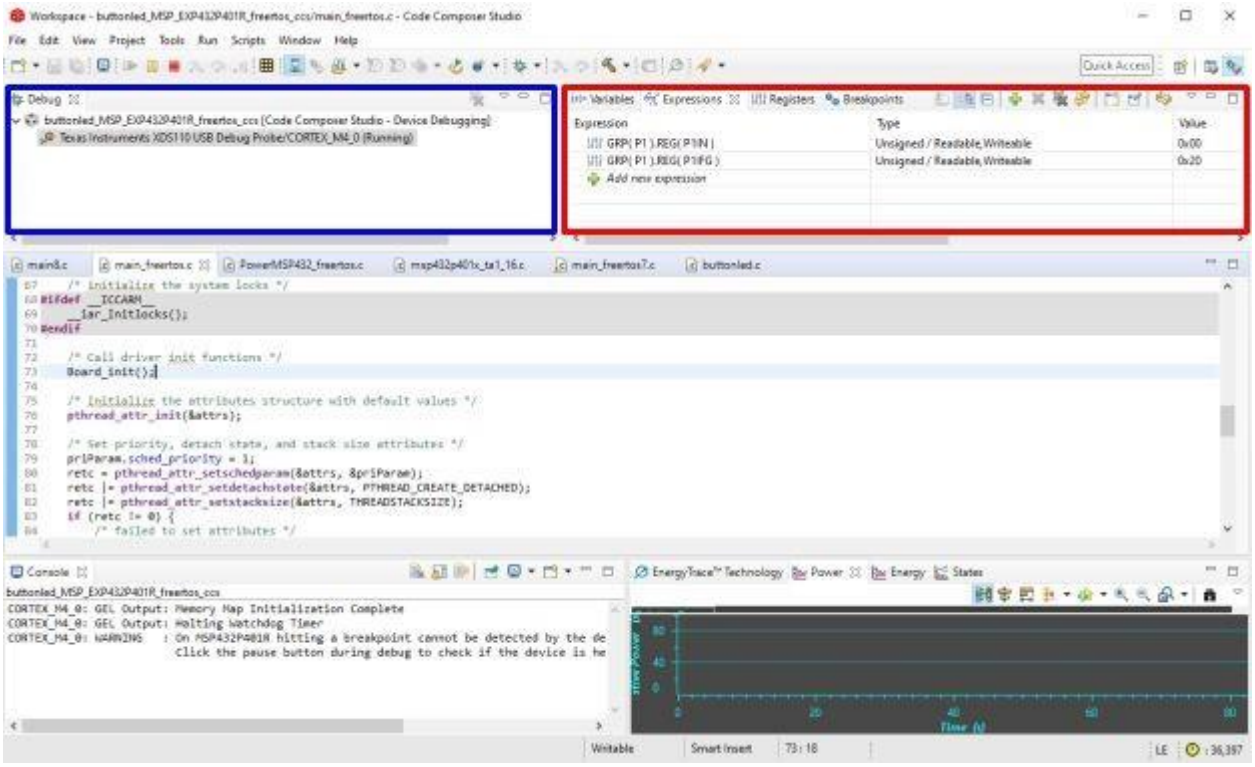
Interfaz



En azul se puede observar la sección de escritura de código, en el recuadro rojo se encuentran las carpetas de los proyectos, en naranja está la consola, la cual se utiliza como salida estándar; y en verde es donde se muestran los mensajes de error, warnings y recomendaciones.

Debugger

Permite realizar distintas pruebas en el programa mientras se está ejecutando.
Permite pausar la ejecución, detenerla, ejecutar línea por línea, etc. Muestra los estados de las variables usadas en el programa, así como el de los registros del microcontrolador.
También se encuentra la consola, la cual es usada como salida estándar.



En azul se encuentra la sección donde se informa el programa que se está debugueando y en dónde.

En rojo está la sección que permite ver el estado de las variables, los registros y los breakpoints en el código.

Optimizador de código

El IDE proporciona una herramienta que permite optimizar el código compilado que se cargará en la placa.

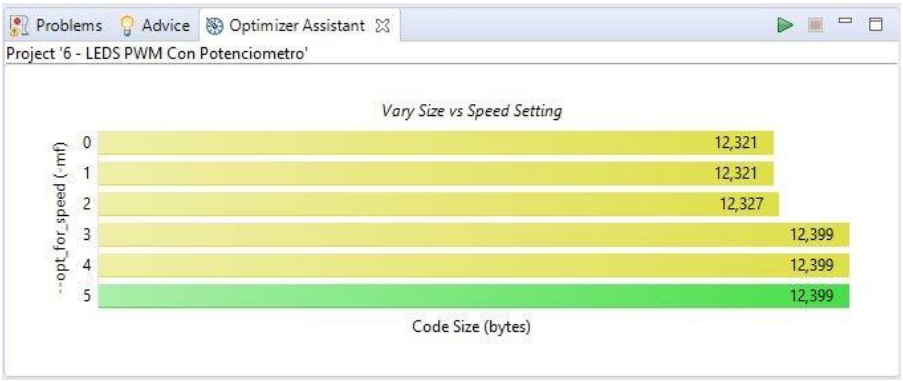
El criterio de optimización es según velocidad de ejecución y tamaño del programa compilado.

Se permiten modificar dos variables que definen distintos parámetros para que el compilador tenga en cuenta a la hora de compilar

Optimization level (op_level): indica e hasta qué nivel se deben aplicar las optimizaciones , por ejemplo a nivel de registros, variables locales, globales, etc. o directamente no optimizar.

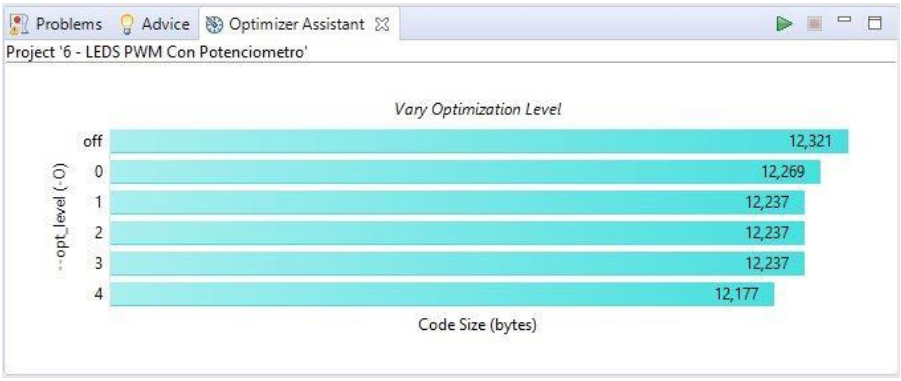
Speed vs. size (opt_for_speed): tiene un rango del 0 al 5 donde en 0 se prioriza un bajo tamaño de código y en 5 se prioriza una alta velocidad de ejecución.

Se corrió la optimización para un programa y se obtuvo el siguiente resultado:



El optimizador arroja los distintos tamaños de programa posibles según distintos valores de opt_for_speed seleccionables y nos indica cuál es el óptimo en cuanto a velocidad en color verde y cuánto pesará el programa compilado.

A continuación se muestra el tamaño del programa según los distintos opt_level seleccionables.



Energy Trace

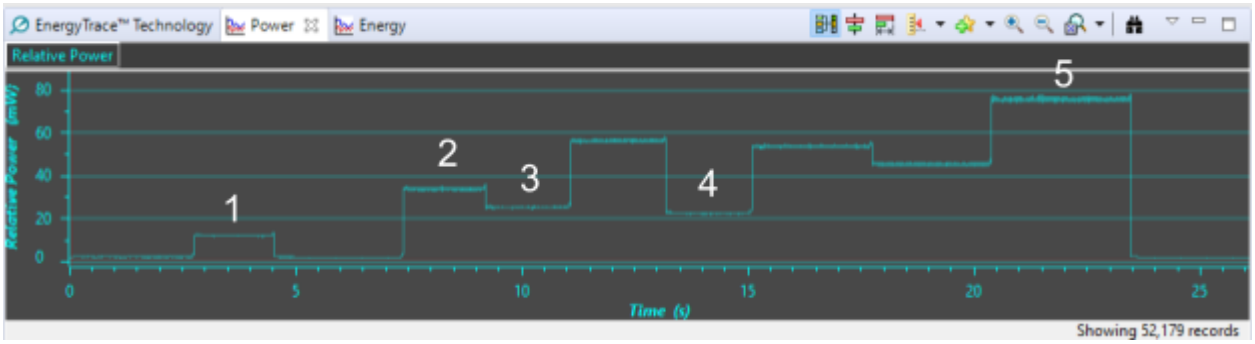
El IDE tiene una herramienta muy útil que se sirve para realizar medidas de consumo de energía de la placa con el objetivo de lograr consumos muy bajos.

Permite conocer cuánto está consumiendo la placa y hace un desglose del consumo realizado por cada periférico, las salidas y por el CPU y cada una de sus partes.

La medición la realiza cuando se está en estado de debugueo y el programa está corriendo.

A medida que realiza la medición, muestra los resultados al instante, lo que permite tener la información en tiempo de ejecución, facilitando así las distintas pruebas que se necesiten realizar.

A continuación se muestra un ejemplo en el que se corrió un programa que prende el led rojo de la placa y luego distintas combinaciones posibles del led RGB que posee la placa y muestra el consumo general de la misma.



1.Solo el led rojo está encendido.

2.Solo el led rojo del led RGB se encuentra encendido.

3.Solo el led verde del led RGB se encuentra encendido.

4.Solo el led azul del led RGB se encuentra encendido.

5.Todos los leds del led RGB están encendidos.

Pruebas realizadas sobre Entradas/Salidas

Configuración general de los puertos de entrada/salida

Para las distintas pruebas llevadas a cabo se realizó la configuración adecuada de los puertos de manera particular para cada prueba. A continuación se muestra de forma general una manera de realizar configuraciones para las distintas funcionalidades de los puertos:

Cada puerto cuenta con dos registros de configuración “PxSEL1” y “PxSEL0” (“x” hace referencia al puerto), a través de los cuales configuramos la funcionalidad para un pin dado.

Por ejemplo para configurar el pin 0 del puerto 1 como entrada o salida debemos setear a P1SEL1 y P1SEL0 ambos en cero. Además para las distintas configuraciones debemos tener en cuenta la dirección seteada (entrada o salida).

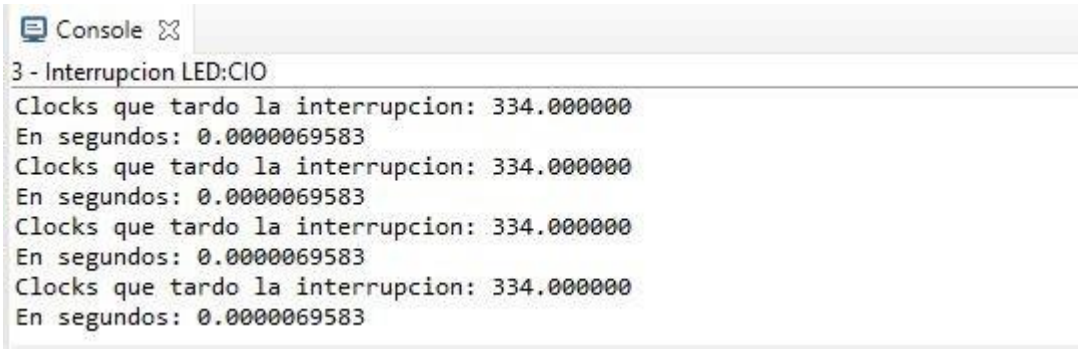
```
PxSEL0 y PxSEL1   // Selección del modo del Px
PxDIR            // Setea como entrada o salida 1 E y 0 S
PxREN            // Habilita resistencia
PxOUT           // Si está habilitada la resistencia setea Pull-Up o Pull-Down
              // Además PxOUT es utilizado para setear el estado de la salida
PxIE            // Habilitación de Int en el puerto
PxIES           // Selección de flanco ascendente o descendente para Int
PxIFG          // Indica si hay Int pendiente
             // Y luego de atendida la Int se debe limpiar el flag
```

Interrupciones.

Latencia:

334 ciclos de clock

Promedio de 6.95 µseg a 48MHz



Encendido de Leds con manejo de Interrupciones.

Ejemplos de fuentes de interrupción:

- Bit externo
- Conversor ADC
- Timer0
- Comunicación

A continuación se explica un fragmento de código del programa realizado para el encendido de Leds a través del accionar del botón gestionado mediante la interrupción.

Main

```
//Se utilizará el led rojo de P1.0 y el botón de
P1.1.
P1SEL0 = 0x00;  // A través del registro SEL1 y SEL0 se configura el
P1SEL1 = 0x00;  //puerto 1 como entrada/salida
P1REN  = 0x02;  //Se habilita resistencia en el pin 1 del puerto 1
P1OUT  = 0x02;  //Se setea en Pull-Up en P1.1, Pull-Down en los otros pines

P1DIR  = 0x01;  //Se configura P1.0 como salida, el resto como entrada

P1IES  = 0x02;  //Se configura el flanco de la Int como descendente
P1IFG  = 0x00;  //Se limpia el flag de Int
P1IE   = 0x02;  //Se habilita Int para el P1.1
NVIC_EnableIRQ(PORT1_IRQn);   //Se habilita Int para el P1.1 en el módulo

NVIC
```

```
__enable_interrupt();   //Se habilita Int globales
       while( 1 )    {       if( button_flag )
      {
        P1OUT ^= 0x01;      //Si el led está prendido lo apaga o viceversa
               button_flag = 0;   //Resetea el flag        P1IFG &= ~0x02;
//Limpia la Int de P1.1
        P1IE |= 0x02;      //Habilita la Int de P1.1
            }   }
```

Manejador de Int

```
void PORT1_IRQHandler( void )//Manejador de Int del puerto 1
       {
    P1IE &= ~0x02; //Deshabilita Int para P1.1     button_flag         =
1;//Flag mediante el cual se avisa que se apretó el botón   }
```

Hola Mundo

```
printf("Hola Mundo!\n");
```

Manejo de Sensor Digital

A continuación se muestra un fragmento de código que hace referencia a la configuración de los puertos como entrada/salida para poder sensar el estado del botón y en base al estado accionar sobre el led.

```
PxSEL0 = 0x00;    // Selección del modo E/S para el puerto P1

PxSEL1 = 0x00;    // P1.1 botón y P1.0 led rojo
P1REN = 0x02;     // Se habilita resistencia en el puerto del botón (P1.1)
P1OUT = 0x02;     // Se habilita resistencia de PULL-UP en (P1.1)
P1DIR = 0x01;    // Se configura P1.0 como salida, el resto como entrada
P1IES = 0x02;   // Se configura la Int como flanco descendente

P1IFG = 0x00;  // Se limpia el flag de Int
P1IE = 0x02;     // Se habilita Int para el P1.1
```

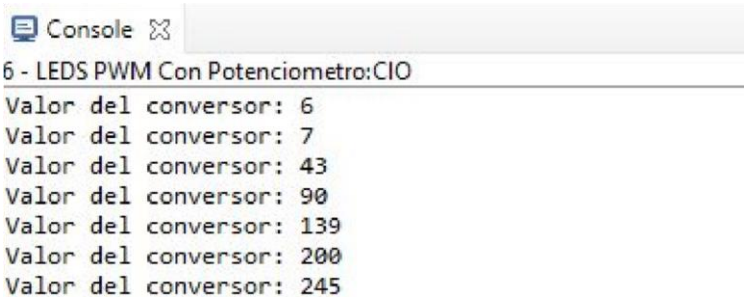
Manejo de Sensor Analógico

Para esta prueba se utilizó un potenciómetro a través del cual variamos el voltaje a convertir por el conversor de analógico a digital.

Conversor Analógico/Digital:

- 15 entradas
- 16 bits de precisión
- Velocidad de muestreo: 1Msps

```
P5SEL0 |= BIT4;            // Se configura el bit 4 del puerto 5 como entrada
P5SEL1 |= BIT4;           // analógica
```



Timers

1. RTC: Real-Time Clock (Trabaja en LPM3.5)
2. CS: Clock System
3. Timer_A: (TA0-TA3) Timer de 16 Bits
4. Timer32: (x2) Timer de 32-bits

Fuente	Descripción	Frecuencia
LFXT1CLK	Oscilador de Alta frecuencia	32.768KHz
XT2CLK	Externo opcional	-
DCOCLK	Oscilador controlado digitalmente interno	Rango configurable de <100KHz a 4MHz

Timer A

- Contador de 16-bit
- 4 modos de operación – Stop, Up, Continuous, Up/Down
- 3 registros capture/compare (CCRx)
- 2 vectores de interrupción– TACCR0 y TAIV

Manejo de Timers - Generación de dos salidas PWM para el manejo de leds

```
P2->DIR  |= BIT4 | BIT5; // P2.4~5 configurados como salidas
P2->SEL0 |= BIT4 | BIT5;                          // P2.4~5 en modo TA0.1~2
P2->SEL1 &= ~(BIT4 | BIT5);


TIMER_A0->CCR[0] = 1000 - 1;                      // Período de los PWM
TIMER_A0->CCTL[1] = TIMER_A_CCTLN_OUTMOD_7;       // CCR1 reset/set
TIMER_A0->CCR[1] = 750;                           // Ciclo de trabajo inicial
CCR1
TIMER_A0->CCTL[2] = TIMER_A_CCTLN_OUTMOD_7;       // CCR2 reset/set
TIMER_A0->CCR[2] = 450;                           // Ciclo de trabajo inicial
CCR2
TIMER_A0->CTL     = TIMER_A_CTL_SSEL__SMCLK |     // SMCLK
                    TIMER_A_CTL_MC__UP |          // Modo Up
                    TIMER_A_CTL_CLR;              // Limpia TAR


__sleep();                                    // Pasar a LPM0 (Low Power Mode)
__no_operation();                             // Para el debugger
```

Sistemas Operativos de Tiempo Real

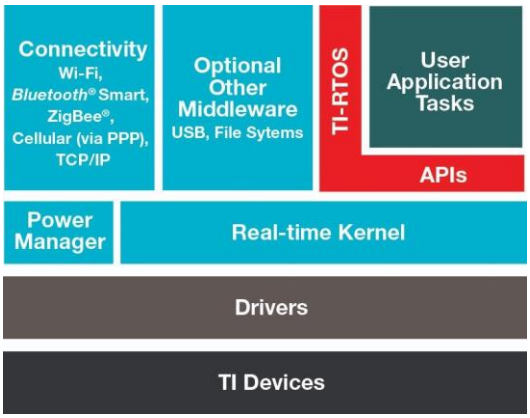
Como se vio en el transcurso de la materia, los sistemas operativos embebidos de tiempo real son aquellos desarrollados para aplicaciones a las cuales se les exigen una respuesta bajo ciertas restricciones de tiempo. Se caracterizan por tener un tamaño mínimo, rápidos cambio de contexto, su determinismo y fiabilidad y por su tolerancia a los fallos. Son diseñados para ser portables a varias arquitecturas y enfocados en hardware orientado a la entrada/salida más que al procesamiento.

Sistemas Operativos Soportados por la Texas MSP432

TI-RTOS

Es un sistema operativo de tiempo real desarrollado para microcontroladores de Texas Instruments. Este sistema combina un kernel multitarea de tiempo real con un sistema middleware que incorpora conectividad Wi-Fi, Bluetooth, TCP/IP, USB y un sistema de archivos FAT, entre otros, a esto se le suman los drivers de los dispositivos de Texas Instruments.

Está compuesto por los siguientes componentes:



- SYS/BIOS: el kernel está diseñado para aplicaciones que requieran planificación en tiempo real y sincronización o instrumentación en tiempo real. Provee un sistema multitarea preventivas, abstracción de hardware, análisis de tiempo real y herramientas de configuración.
- Drivers: son manejadores con una API standard sobre todos los dispositivos soportados.
- UIA: Unified Instrumentation Architecture, provee contenido enfocado en asistir en la creación y recolección de datos de instrumentación (Como una bitácora de lo que ocurre por ejemplo). Este componente es utilizado junto con el System Analizer para reportar datos en Code Composer Studio.
- XDCTools: Este componente provee las herramientas de configuración para compilar el sistema y los otros componentes.
- Librerías Específicas de cada dispositivo: las diferentes versiones de TI-RTOS se entregan en conjunto a esto, en base a la placa sobre la que se quiera desarrollar, en nuestro caso, esta todo incorporado en el SDK de SipleLink para MSP432.

freeRTOS

Es un sistema operativo de tiempo real gratuito para dispositivos embebidos, que fue porteado a más de 35 plataformas de microcontroladores. Está distribuido bajo la licencia MIT (También conocida como licencia X11).

Este sistema está diseñado para ser pequeño y simple. Está escrito principalmente en C, pero contiene un par de funciones en lenguaje ensamblador donde se necesitan rutinas de planificación de arquitectura específica.

Además provee métodos para múltiples subprocesos, mutexes, semáforos, temporizadores de softwares y soporta prioridad de los hilos. Pero no cuenta con funciones más avanzadas como drivers, administración avanzada de memória, cuentas de usuario y redes.

Al estar enfocado en ser compacto y veloz al ejecutarse, se puede considerar más una como una biblioteca de hilos en lugar de un sistema operativo. Entre sus características claves se encuentra que su planificador puede ser configurado para operaciones tanto preventivas como cooperativas. Una de las ventajas principales de este sistema es posee una amplia comunidad de desarrolladores y mucha documentación.

RTX

También conocido como Keil RTX, es un sistema operativo de tiempo real determinístico diseñado para diseñado para dispositivos ARM y Cortex-M. Entre sus características se destacan:

- Aunque es pago, es libre de royalty (no es requerido pagar extra para implementarlo) y se entrega el código fuente. O es posible utilizar gratuitamente una edición limitada.
- Posee un planificador flexible, round-robin, preventivo y cooperativo.
- Operaciones de tiempo real de alta velocidad con baja latencia de interrupción.
- Es ligero para sistemas reducidos.
- Soporta una cantidad ilimitada de tareas con 254 niveles de prioridad.
- Soporta una cantidad ilimitada de buzones, semaforos, mutex y timers.
- Multitarea y operaciones de hilos seguras.

Micrium µC/OS

Es un sistema operativo embebido con soporte para TCP/IP, USB, CAN bus y Modbus. Además cuenta con un robusto sistema de archivos y una interfaz gráfica para el usuario. Este sistema fue portado a más de 50 arquitecturas. El software que vende la empresa incluye una documentación amplia, el código fuente completo, herramientas de debuggeo y soporte para varias arquitecturas de CPU. Micrium es también de un tamaño reducido, remarcable eficiencia de energía y una suite completa de pilas de protocolos.
En el núcleo de este sistema están los kernels de tiempo real µC/OS-III y µC/OS-II, que son altamente portables, escalables, preventivos, determinísticos y multitareas para

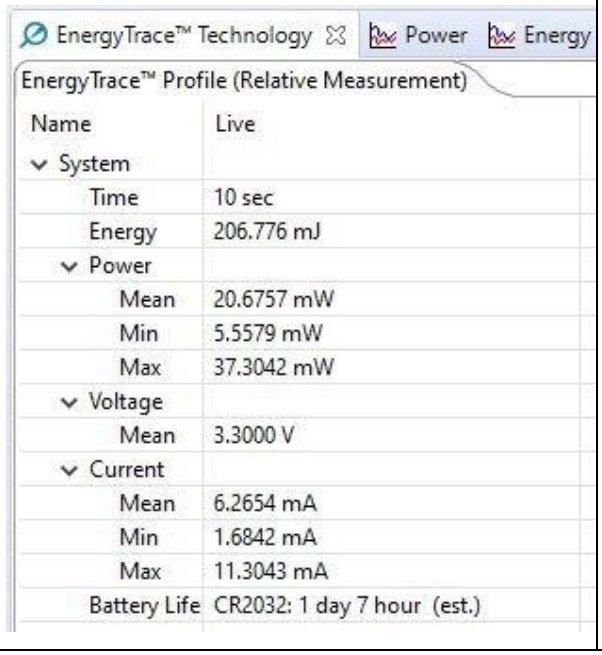
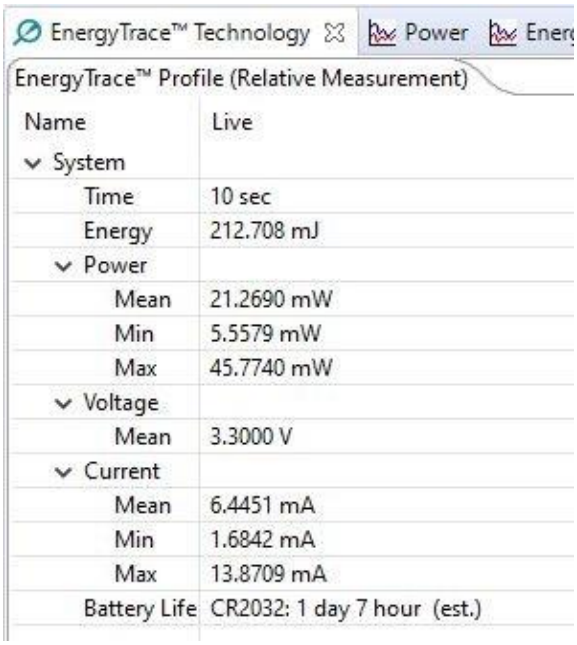
microprocesadores, microcontroladores y DSPs. Dichos kernels son entregados con un código fuente 100% en ANSI C y documentación a fondo.

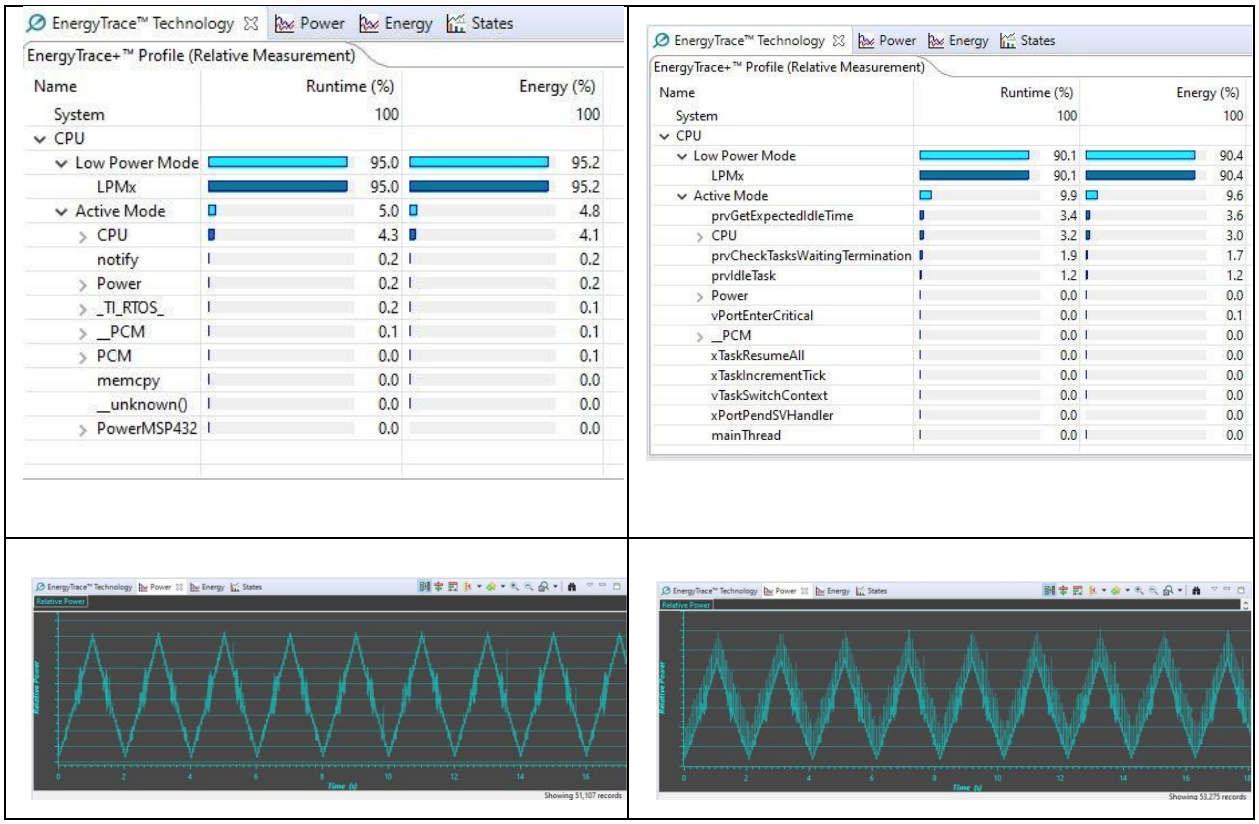
Pruebas Realizadas

Decidimos enfocarnos en probar los sistemas TI-RTOS, ya que fue desarrollado especialmente por la empresa que desarrolló la placa y freeRTOS, porque lo utilizamos durante la práctica. Además un factor importante es que ambos sistemas son completamente gratuitos y ambos vienen incluidos en el SDK que está disponible para la familia de placas MSP432.

Para comparar el desempeño de los sistemas, se utilizó la librería PThreads. Con la cual se crea un hilo encargado de prender y apagar el led RGB de manera gradual, y al presionar dos veces el pulsador se prende y apaga una serie específica de veces. El código de ambas aplicaciones, es casi idéntico, la única variación (además de tener un nombre diferente la función que inicia el planificador) es que en freeRTOS es necesario inicializar los locks antes de crear el thread.

El análisis se realizó con las herramientas provistas por el Code Composer Studio, principalmente el Energy Trace, que se utilizó para medir el gasto energético durante un lapso de 10 segundos, el cual nos dio los siguientes resultados:

TI-RTOS	freeRTOS
	



Como se puede apreciar en las imágenes, el gasto con TI-RTOS es menor en total (206 mJ contra 212 mJ de freeRTOS). Y aunque la potencia promedio fue cercana en ambos, el consumo máximo de freeRTOS es mayor al de TI-RTOS (45,7 mW contra 37,3 mW del TI-RTOS), lo cual se traduce en que TI-RTOS consume 18.5% menos en su punto máximo.

Además cabe destacar que eTI-RTOS pasa solo 5% del tiempo activo y el resto en modo de bajo energía, mientras que freeRTOS pasa activo 9.9% del tiempo total, casi el doble.

Comunicación

Módulos de Comunicación

Los microcontroladores de la familia MSP432P4xx posee dos interfaces de comunicación universal serie mejorada (eUSCI A y B) que trabajan en siguientes los modos:

UART (eUSCI_A)

En modo asincrónico, los modulos eUSCI_Ax conectan el dispositivo a un sistema externo a través de dos pines, UCAxRXD y UCAxTXD. El modo UART es seleccionado cuando el bit UCSYNC es cero.

Las características del modo UART incluyen:

- Datos en 7 u 8 bits, con paridad par, impar o sin paridad.
- Registros de desplazamiento independientes para la transmisión y recepción.
- Registros buffer separados de transmisión y recepción.
- Modo de transmisión y recepción LSB-First o MSB-First
- Protocolos incorporados de idle-line y bit de dirección para sistemas multiprocesador.
- Baud Rate programable con soporte para modulación por baud-rate fraccional.
- Flags de estado para detección y supresión de errores.
- Flags de estado para detección de dirección.
- Capacidad para interrupciones independientes para recibir, transmitir, recepción de bit de inicio y transmisión completa.
- Codificación y Decodificación IrDA para la comunicación con infrarojo.

SPI (eUSCI_A y eUSCI_B)

En modo sincrónico, los módulos eUSCI conectan el dispositivo a un sistema externo a través de 3 ó 4 pins, UCxSIMO, UCxSOMI, UCxCLK y UCxSTE. El modo SPI es seleccionado al poner un uno en el bit UCSYNC y el modo de SPI (3 ó 4 pins) es seleccionado con los bits de UCMODEx.

Las características del modo SPI incluyen:

- Datos en 7 u 8 bits de largo.
- Modo de transmisión y recepción LSB-First o MSB-First.
- Modo de operación de 3 ó 4 pins.
- Modos de Maestro o Esclavo.
- Registros de desplazamiento independientes para la transmisión y recepción.
- Registros buffer separados de transmisión y recepción.
- Operación de recepción y transmisión continua.
- Polaridad de reloj seleccionable y control de fase.
- Frecuencia programable de reloj en modo Maestro.

- Capacidad para interrupciones independientes para recibir y transmitir.

I^2C (eUSCI_B)

En este modo, el módulo eUSCI_B provee una interfaz de comunicación entre el dispositivo y otros dispositivos compatibles con I^2C conectados por bus serie I^2C de dos cables.
Las características del modo I^2C incluyen:

- Modos de direccionado de dispositivos de 7 y 10 bits.
- Llamadas generales.
- START, RESET y STOP.
- Modo recepción/transmisión multi-maestro.
- Modo recepción/transmisión esclavo.
- Soporte para modo estándar de hasta 100 kbps, modo rápido de 400 kbps y modo rápido plus de hasta 1 Mbps.
- Frecuencia UCxCLK programable en modo maestro.
- Diseñado para baja potencia.
- Contador de 8 bits con capacidad de interrupciones y afirmación de STOP automática.
- Hasta cuatro direcciones de hardware esclavos, cada uno con su propia interrupción y trigger DMA.
- Registro máscara para direcciones de esclavos y direcciones de interrupción recibida.
- Interrupción de time-out para evitar la detención del bus.

Pruebas básicas de Comunicación

Para la prueba, se conectó un módulo de comunicación bluetooth MLT-BT05, previamente configurado, cuyos TX y RX se conectaron a los pines P3.2 y P3.3 respectivamente, que fueron configurados en modo de RX y TX del módulo eUSCIA2 en modo UART.

El programa que se utilizó se encargaba retransmitir lo que recibía y si recibía un número del 0 al 4, prendía o apagaba uno de los leds integrados de la placa.

Para esto primero se configura para utilizar el SMCLK/DCO como fuente de clock y el dispositivo se pone en modo LPM0, para minimizar el gasto energético. La auto habilitación del clock es una característica utilizada por el eUSCI: el SMCLK es apagado mientras el UART está ocioso y es encendido cuando se recibe un flanco ascendente.

Utilizando la aplicación de Monitor Serie Bluetooth, logramos establecer una comunicación con la placa y enviar los comandos de prueba.

Ahorro de consumo

Existen 5 modos de ahorro de consumo o Low Power Modes (LPM) disponibles proveídos por el microcontrolador de la placa. Estos son LPM0, LPM3, LPM3.5, LPM4 y LPM4.5.
Estas 5 variedades tienen en común en que trabajan con el CPU detenido.

LPM0 (Sleep mode): La ejecución del procesador es detenida. Es útil para ahorrar energía cuando no es necesaria la ejecución del procesador y se necesita volver rápidamente a un modo activo, es decir, que el procesador vuelva a ejecutar tareas.

Se pueden lograr corrientes de 63µA.

LPM3 y LPM4 (Deep Sleep): Similar a LPM0, se detiene el procesador. Es útil para actividad del procesador relativamente infrecuente seguida de actividad de baja frecuencia. El tiempo de vuelta a actividad es más largo que LPM0, pero el consumo es significativamente menor.
En LPM3 el clock se restringe a 32,768 kHz. Solo los módulos RTC y WDT son funcionales con un clock de baja frecuencia. Los demás periféricos son deshabilitados junto con los clocks de alta frecuencia.
El modo LPM4 se entra cuando el microcontrolador se programa con para LPM3 con los periféricos RTC y WDC desactivados. Todas las fuentes de clock son desactivadas y no hay actividad de periféricos
En LPM3 y LPM4 se pueden lograr consumos de 660nA y 500nA respectivamente.

LPM3.5 y LPM4.5 (Stop o Shut Down): Estos modos proveen los consumos más bajos a costa de una funcionalidad reducida. Son útiles cuando se está sin actividad durante largos periodos de tiempo.
En LPM3.5 todos los periféricos son deshabilitados y apagados excepto RTC y WDT. El microcontrolador se lo lleva a 'core voltage level 0', lo que hace que funcione al menor voltaje posible. No se retiene ningún dato de los periféricos.
En LPM4.5 todos los periféricos son deshabilitados y apagados, y el regulador de voltaje es apagado. No se retiene datos de la SRAM, ni de los periféricos.
En LPM3.5 y LPM4.5 se pueden lograr consumos de 630nA y 25nA Respectivamente.