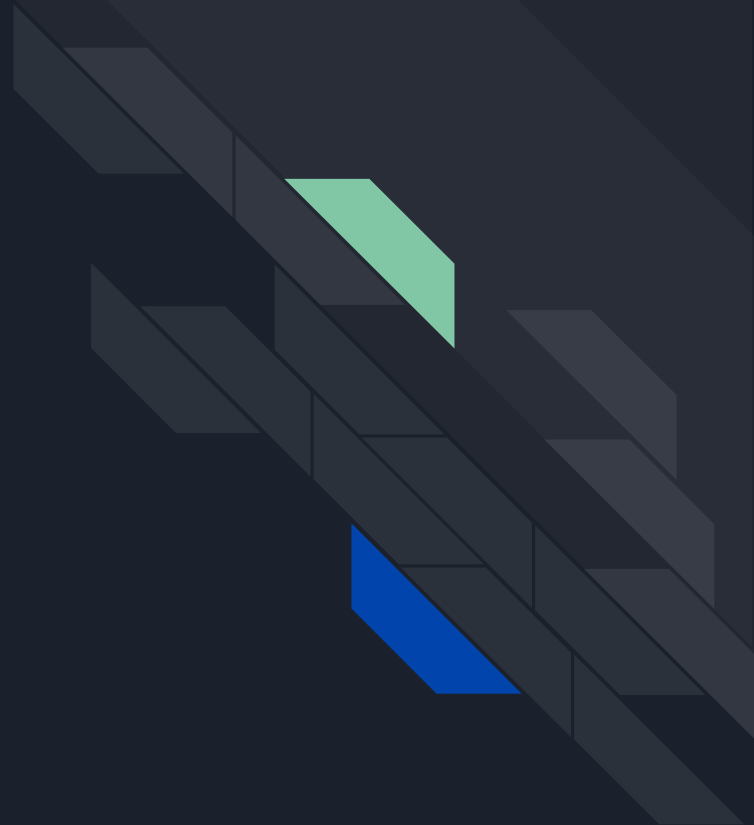


Español



Diseño de Driver ADC para tarjeta MSP432P401R

Alex Armando Figueroa Hernandez - 17061169
Manuel Alejandro Quiroz Gallegos - C18061043

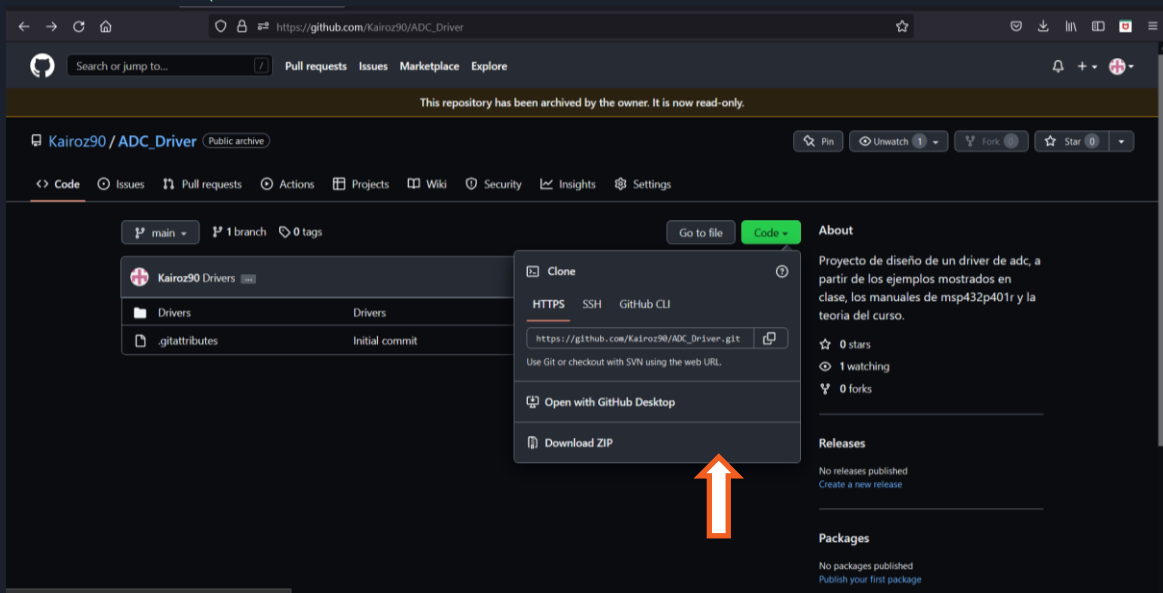




Creación de un proyecto CCS para la aplicación del driver ADC.

- Para probar el correcto funcionamiento del driver se puede probar en un proyecto de Code Composer Studio con o sin uso de SimpleLink.
- El uso de SimpleLink facilitará el uso del driver, pero para comprobar su independencia se proba un proyecto con SimpleLink.
- Link del repositorio con el driver adc creado: https://github.com/Kairoz90/ADC_Driver

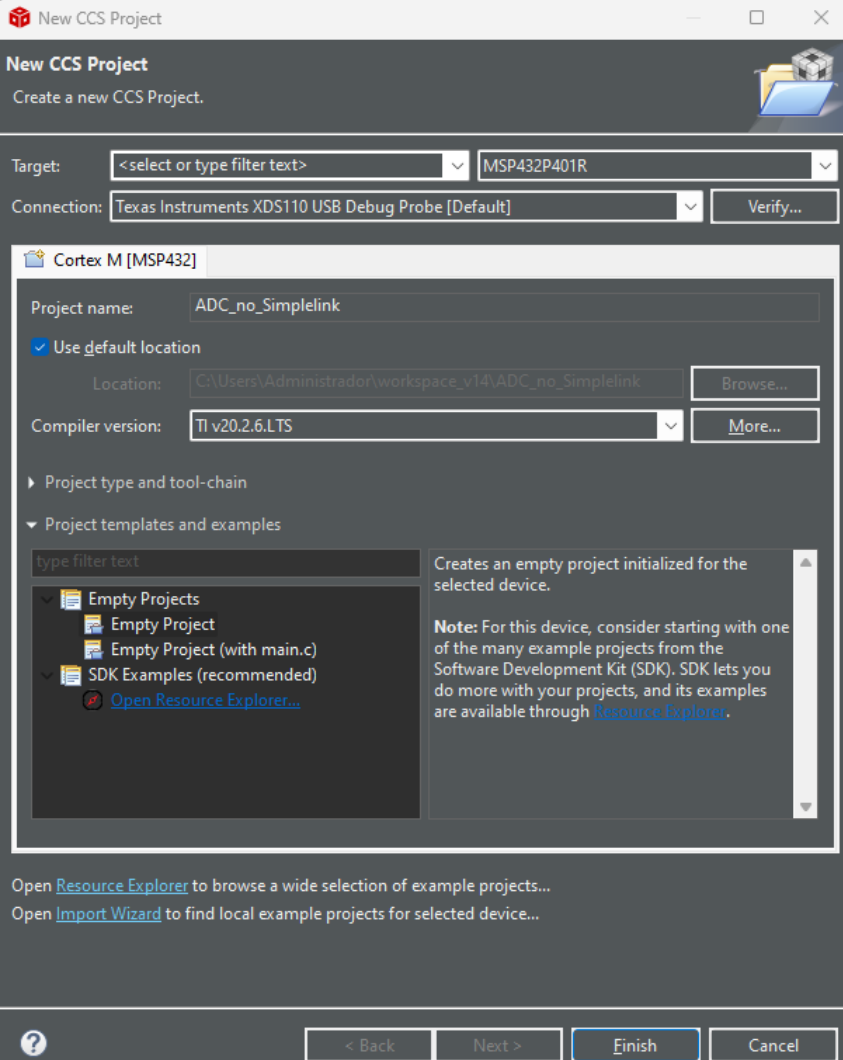
Creación de un proyecto CCS para la aplicación del driver ADC.



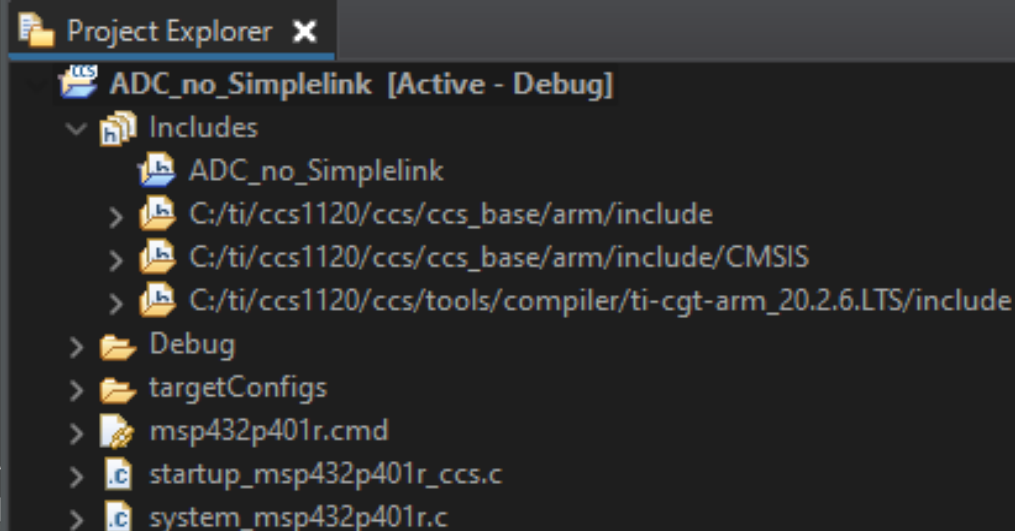
- Descargar los drivers del link como zip y extraer el archivo:
- https://github.com/Kairoz90/ADC_Driver
- Según la manera de trabajo, la instalación puede con o sin simplelink



Driver sin simplelink

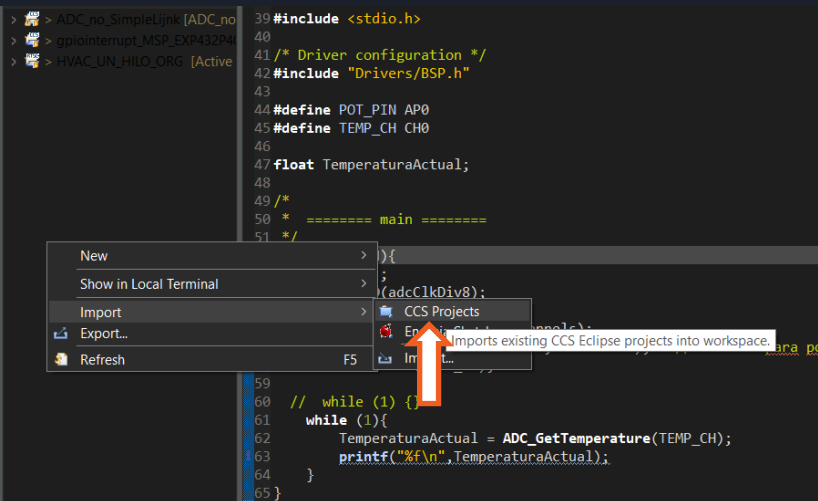


1. Se crea un nuevo proyecto “Empty Project” para la tarjeta MSP432P401R y un nombre del proyecto.
2. Al ser creado, aparecerá en el explorador de proyectos, si no tiene simplelink, solo tendrá esto en la carpeta de includes.

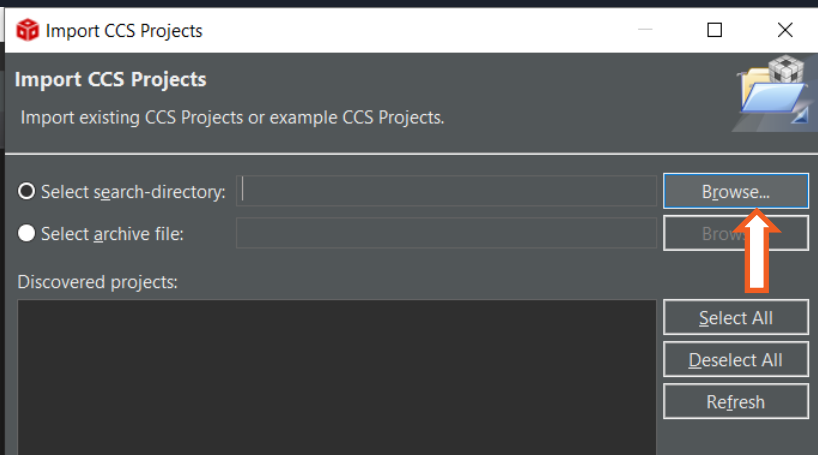


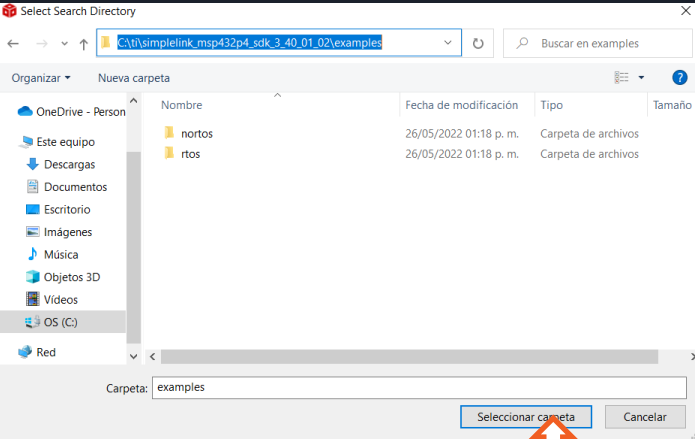


Driver con simplelink



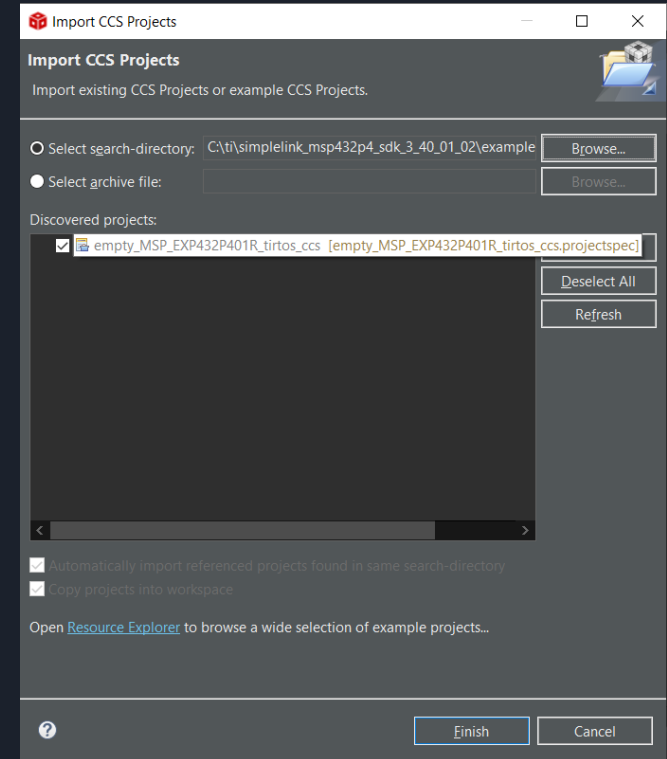
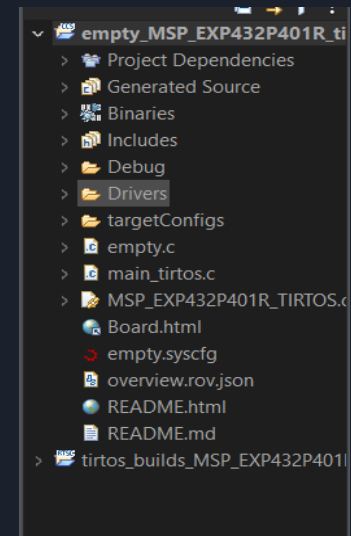
1. Se importara un proyecto “ejemplo” para la tarjeta MSP432P401R.
2. Se da click derecho en el apartado de project explorer y se selecciona importar un ccs project.
3. Seleccionar la opcion de browse dentro de la pestaña.





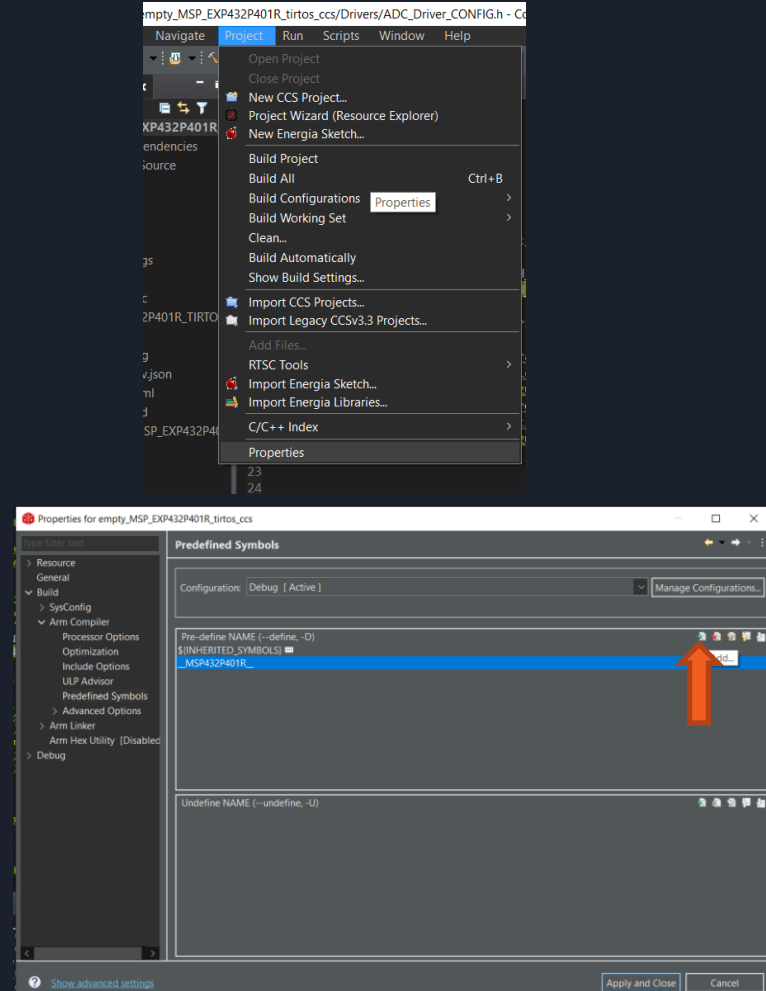
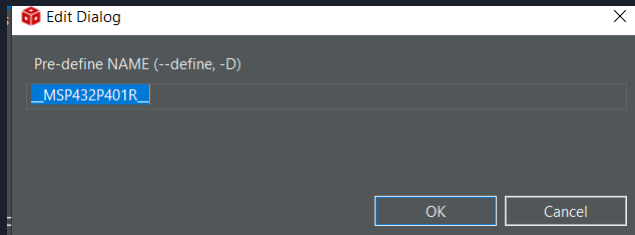
4. Buscar la dirección dentro de la imagen o “C:\ti\simplelink_msp432p4_sdk_3_40_01_02\examples”, la cual puede variar dependiendo de la versión del simplelink que tenga el equipo que se este utilizando.
5. Por ultimo, seleccionar cualquier ejemplo ya sea com rtos o nortos, y empezar a modificar a las necesidades del proyecto.

6. Se le da a finish y se importa la carpeta drivers al proyecto.



Para ambos casos

1. Ir a la barra de Project y abrimos properties.
2. Por ultimo se agrega un predefined symbol, en add y se agrega el símbolo “__MSP432P401R__”, se le da a click a OK y luego click a Apply and Close.



Para ambos casos

1. Verificar si construye el proyecto, en caso de que no, comentar lo de la flecha naranja y descomentar lo de la flecha amarilla en el archivo ADC_Driver_Config.

*ADC_Driver_CONFIG.h x

```
1 /*
2  * ADC_MSP432_CONFIG.h
3  *
4  * Created on: Jun 16, 2022
5  * Author: Administrador
6  */
7
8 // #define __MSP432P401R__
9 // #include <ti/devices/msp432p4xx/inc/msp.h>
10 #include "msp.h"
11 #ifndef DRIVERS_ADC_Driver_CONFIG_H_
12 #define DRIVERS_ADC_Driver_CONFIG_H_
```



Definiciones de los drivers

- ADC_Driver_Config.h
- ADC_Driver.h



ADC_Driver_config.h

Aquí se encuentra la tabla de configuración y la dirección de los registros del driver ADC.



ADC_Driver_config.h

La dirección de la memoria para los periféricos inicia en 0x40000000, y la base para el modulo de ADC, esta en 0x00012000 después de iniciar la dirección de los periféricos.

Se define el ADC_14 como el tipo específico de la estructura por la dirección base.

```
24 /*Tabla de configuracion*/
25 #define PERBASE                ((uint32_t)0x40000000)          /*!< Peripherals start address */
26 #define ADC14BASE              (PERBASE +0x00012000)          /*!< Base address of module ADC14 registers */
27 #define ADC_14                 ((ADC14_Tipo *) ADC14BASE)
28 #endif /* DRIVERS_ADC_MSP432_CONFIG_H_ */
```

ADC_Driver_config.h

La estructura de los tipos de ADC14, se encuentran los registros de ADC, entre ellos, el de control 0, control 1, control memoria, banderas de interrupción, limpieza de banderas, almacenamiento de memoria, vector de interrupción.

```
36 typedef struct {
37     __IO uint32_t CTL0;
38     __IO uint32_t CTL1;
39     __IO uint32_t L00;
40     __IO uint32_t HI0;
41     __IO uint32_t L01;
42     __IO uint32_t HI1;
43     __IO uint32_t MCTL[32];
44     __IO uint32_t MEM[32];
45     uint32_t RESERVED0[9];
46     __IO uint32_t IER0;
47     __IO uint32_t IER1;
48     __IO uint32_t IFGR0;
49     __IO uint32_t IFGR1;
50     __IO uint32_t CLRIFGR0;
51     __IO uint32_t CLRIFGR1;
52     __IO uint32_t IV;
53 } ADC14_Tipo;

/*< Control 0 Register */
/*< Control 1 Register */
/*< Window Comparator Low Threshold 0 Register */
/*< Window Comparator High Threshold 0 Register */
/*< Window Comparator Low Threshold 1 Register */
/*< Window Comparator High Threshold 1 Register */
/*< Conversion Memory Control Register */
/*< Conversion Memory Register */
/*< Interrupt Enable 0 Register */
/*< Interrupt Enable 1 Register */
/*< Interrupt Flag 0 Register */
/*< Interrupt Flag 1 Register */
/*< Clear Interrupt Flag 0 Register */
/*< Clear Interrupt Flag 1 Register */
/*< Interrupt Vector Register */
```



ADC_Driver.h

- Aquí se encontraran las mascara de los offsets de los registros del modulo ADC diseñado.

ADC_Driver.h

Se definen mascararas para los offset en los registros del ADC.

En este caso, los que determinan el tipo de comparación de voltaje.

La resolución del ADC.

Las divisiones del reloj del ADC.

```
20 #define adcVccVss      ((uint32_t)0x00000100) /*!< V(R+) = AVCC, V(R-) = AVSS */
21 #define adcVrefVss     ((uint32_t)0x00000200) /*!< V(R+) = VREF buffered, V(R-) = AVSS */
22 #define adcVrefCVss    ((uint32_t)0x00000E00) /*!< V(R+) = VeREF+, V(R-) = VeREF- */
23 #define adcVrefBufVref ((uint32_t)0x00000F00) /*!< V(R+) = VeREF+ buffered, V(R-) = VeREF */
24
25 /*
26  * Definicion de resoluciones de ADC
27  */
28
29 #define adc8BitRes      ((uint32_t)0x00000000) /*!< 8 bit (9 clock cycle conversion time) */
30 #define adc10BitRes     ((uint32_t)0x00000010) /*!< 10 bit (11 clock cycle conversion time) */
31 #define adc12BitRes     ((uint32_t)0x00000020) /*!< 12 bit (14 clock cycle conversion time) */
32 #define adc14BitRes     ((uint32_t)0x00000030) /*!< 14 bit (16 clock cycle conversion time) */
33
34 /*
35  * Definicion de divisores de reloj de ADC, son 8 divisores
36  */
37
38 #define adcClkDiv1 ((uint32_t)0x00000000) /*!< /1 */
39 #define adcClkDiv2 ((uint32_t)0x00400000) /*!< /2 */
40 #define adcClkDiv3 ((uint32_t)0x00800000) /*!< /3 */
41 #define adcClkDiv4 ((uint32_t)0x00C00000) /*!< /4 */
42 #define adcClkDiv5 ((uint32_t)0x01000000) /*!< /5 */
43 #define adcClkDiv6 ((uint32_t)0x01400000) /*!< /6 */
44 #define adcClkDiv7 ((uint32_t)0x01800000) /*!< /7 */
45 #define adcClkDiv8 ((uint32_t)0x01C00000) /*!< /8 */
```

ADC_Driver.h

```
47 /*
48  * Definicion de predivisores de reloj de ADC
49  */
50
51 #define adcClkPreDiv1 ((uint32_t)0x00000000) /*!< Pcedivide by 1 */
52 #define adcClkPreDiv4 ((uint32_t)0x40000000) /*!< Pcedivide by 4 */
53 #define adcClkPreDiv32 ((uint32_t)0x80000000) /*!< Pcedivide by 32 */
54 #define adcClkPreDiv64 ((uint32_t)0xC0000000) /*!< Pcedivide by 64 */
55
56 /*
57  * Definicion de modos de conversion
58  */
59
60 #define adcSingleChannel ((uint32_t)0x00000000) /*!< Single-channel, single-conversion */
61 #define adcSequenceChannels ((uint32_t)0x00020000) /*!< Sequence-of-channels */
62 #define adcRepeatSingleChannel ((uint32_t)0x00040000) /*!< Repeat-single-channel */
63 #define adcRepeatSequenceChannels ((uint32_t)0x00060000) /*!< Repeat-sequence-of-channels */
64
65 // Selecciona la fuente de reloj del módulo
66
67 #define adcModSource ((uint32_t)0x00000000) /*!< MODCLK */
68 #define adcSysSource ((uint32_t)0x00080000) /*!< SYSCLK */
69 #define adcASource ((uint32_t)0x00100000) /*!< ACLK */
70 #define adcMSource ((uint32_t)0x00180000) /*!< MCLK */
71 #define adcSMSource ((uint32_t)0x00200000) /*!< SMCLK */
72 #define adcHSource ((uint32_t)0x00280000)
73
74
75 //Estos bits definen el número de ciclos ADC14CLK
76 //en el periodo de muestreo para los registros ADC14MEM0 a ADC14MEM7
77
78 #define adcFs4 ((uint32_t)0x00000000) /*!< 4 */
79 #define adcFs8 ((uint32_t)0x00001000) /*!< 8 */
80 #define adcFs16 ((uint32_t)0x00002000) /*!< 16 */
81 #define adcFs32 ((uint32_t)0x00003000) /*!< 32 */
82 #define adcFs64 ((uint32_t)0x00004000) /*!< 64 */
83 #define adcFs96 ((uint32_t)0x00005000) /*!< 96 */
84 #define adcFs128 ((uint32_t)0x00006000) /*!< 128 */
85 #define adcFs192 ((uint32_t)0x00007000) /*!< 192 */
```

Las pre-divisiones de reloj.

Los modos de conexión de los canales.

La fuente de reloj del modulo.

El periodo de muestreo.

ADC_Driver.h

El control de canales.
La medición de los canales.

```
87 /*
88  * Definiciones de control de canales
89  */
90
91 #define MAX_ADC_VALUE 16383 // (2 ^14 bits)
92 #define ioAdcRunChannel (0x10000000) //Mientras esta bandera está activa, el canal está corriendo.
93 #define ioAdcFireTrigger (0x10000001) //Mientras esta bandera está activa, se dispara el trigger.
94 #define ioAdcStopChannel (0x10000002) //Si esta bandera está activa, se detiene el canal.
95 #define ioAdcStopChannels (0x10000003) //Si esta bandera está activa, se detienen todos los canales.
96 #define ioAdcPauseChannel (0x10000004) //Si esta bandera está activa, se pausan el canal.
97 #define ioAdcPauseChannels (0x10000005) //Si esta bandera está activa, se pausan todos los canales.
98 #define ioAdcResumeChannel (0x10000006) //Mientras esta bandera está activa, el canal está activo y ha sido re-iniciado.
99 #define ioAdcResumeChannels (0x10000007) //Mientras esta bandera está activa, todos los canales se activan y re-inician.
100
101 /*
102  * Definiciones de medición de canales ADC
103  */
104
105 #define adcMeasureLoop (0x00) //Se realiza una medición bajo el esquema de tiempo del modulo timer32_1
106 #define adcStartNow (0x01) //Se inicia el timer, y la medición comienza desde el inicio
107 #define adcStartTrigger (0x02) //Se realiza una medición desde el inicio
108 #define adcMeasureOnce (0x04) //Se realiza una medición activando un trigger manual.
109
110 /*
111  * Enumeración de triggers, en el ADC hay 8 triggers
112  */
113
114 enum trigger{
115     TR1, TR2, TR3, TR4, TR5, TR6, TR7, TR8
116 };
117
118 /*
119  * Enumeración de canales, en el ADC hay 31 canales
120  */
121
122 enum channel{//Buffers de conversion
123     CH0, CH1, CH2, CH3, CH4, CH5, CH6,
124     CH7, CH8, CH9, CH10, CH11, CH12, CH13,
125     CH14, CH15, CH16, CH17, CH18, CH19, CH20,
126     CH21, CH22, CH23, CH24, CH25, CH26, CH27,
127     CH28, CH29, CH30, MAXCH=31,
128 };
129
130 /*
131  * Enumeración de pins analogos, en el ADC hay de 0 a 18, y otros para funciones específicas de 19 a 23 y el a2
132  */
133
134 enum analogPin{ // Pin analogos de la tarjeta
135     AP0, AP1, AP2, AP3, AP4, AP5, AP6,
136     AP7, AP8, AP9, AP10, AP11, AP12, AP13,
137     AP14, AP15, AP16, AP17, AP18, AP19 = 18,
138 };
```

ADC_Driver.h

Definiciones de las funciones utilizadas en el archivo ADC_Driver.c

```
146 //Inicializa el modulo ADC.
147 extern void adcInit(void);
148 //Funcion que describe la resolución del reloj.
149 extern void adcClockR(uint32_t res);
150 //Funcion que describe el divisor de reloj
151 extern void adcClockD(uint32_t adcClkDiv);
152 //Configura el modo de conversión del ADC
153 extern void adcMode(uint32_t mode);
154 //Configura la cantidad de ciclos
155 extern void adcMSC(void);
156 /* Función que configura el canal de conversión. */
157 extern void adcConversion(uint32_t CMode);
158 //Configura el canal inicial y final de la secuencia de conversión.
159 extern void adcCanalInicial(uint32_t CH);
160 extern void adcCanalFinal(uint32_t CH);
161 /* Función que configura el pin analogo para un canal ADC. */
162 extern void adcConfigPin(uint32_t CH, uint32_t AP, uint32_t VRef);
163 /* Función que dispara la conversión ADC. */
164 extern void adcActivado(void);
165 /* Función que indica si hay una conversión en curso o no. */
166 extern _Bool adcOcupado(void);
167 /* Función que retorna el valor de la conversión ADC de un canal en específico. */
168 extern uint16_t adcResultado(uint16_t channel);
```



Funciones del driver

- `ADC_Driver.c`



ADC_Driver.c

- Aquí se encuentran las funciones que se utilizan para trabajar y manipular los registros de memoria.

ADC_Driver.c

adcInit(void) inicia el modulo de ADC al activar el bit en el offset del, ADC14_CTL0_ON_OFS con un 1.

```
10 /*
11  * Function: adcInit
12  * Preconditions: extern void adcInit() en ADC.h.
13  * Overview: Configura e inicializa el modulo.
14  * Input: Parámetros ninguno.
15  * Output: Ninguno.
16  *
17  */
18
19 void adcInit(void){
20     //El registro ADC14_CTL0[ON] Bits, activa el modulo en 1
21     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_ON_OFS) = 1;
22     /*CTL0 = Control 0 Register */
23 }
```



ADC_Driver.c

- **adcMSC** Establece si la conversión de muestreo múltiple es de un flanco por subida o de un flanco para todos los muestreos.

```
36 void adcMSC(void){  
37     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_MSC_OFS  ) = 0;  
38 }
```




ADC_Driver.c

adcClockD pide una variable que determina la división del reloj,
El ADC14_CTL0_SHP_OFS inicia la división del reloj,
y ADC_14 -> CTL0 establece el divisor.

```
49 void adcClockD(uint32_t adcClkDiv){  
50     //Aqui se configura el reloj.  
51     ADC_14 -> CTL0 |= adcClkDiv | ADC14_CTL0_SHT1__64 /*La division*/ | ADC14_CTL0_SHT0__192;  
52     /*ADC 14((ADC14_Type *) ADC14_BASE) */  
53     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_SHP_OFS) = 1;  
54 }
```



ADC_Driver.c

La resolución del reloj, se establece al darle una variable.

```
65 void adcClockR(uint32_t res){  
66     //ADC14CTL1 configuramos la resolución a la que va a trabajar el conversor  
67     ADC_14 -> CTL1 = res ;  
68     /*CTL1 = Control 1 Register */  
69  
70 }
```



ADC_Driver.c

La variable que se pide aquí establece el modo de conversión.

```
80 void adcConversion(uint32_t CMode){  
81     //Aquí configuramos cual va a ser el modo de conversión mediante el registro ADC14CTL0  
82     ADC_14 -> CTL0 |= CMode;  
83 }  
--
```



ADC_Driver.c

El canal final se inicia con un bit 1 al offset de
ADC14_MCTLN_EOS_OFS,
Y una variable del canal al registro de control de memoria.

```
93 void adcCanalFinal(uint32_t CH){  
94     // Aqui elegimos cual es la direccion final del canal para el uso de modo secuencia  
95     BITBAND_PERI(ADC_14->MCTL[CH], ADC14_MCTLN_EOS_OFS) = 1;  
96 }
```



ADC_Driver.c

El canal inicial se establece de la misma manera, que manda una variable que indica el canal.

```
107 void adcCanalInicial(uint32_t CH){  
108     //Caso contrario, este serpa la direccion inicial del canal para el uso de modo secuencia  
109     ADC_14->CTL1 |= (CH<<ADC14_CTL1_CSTARTADD_OFS);  
110 }
```



ADC_Driver.c

adcConfigPin configura el pin análogo por donde se compara el voltaje, se usan las variables del canal indicado, el pin indicado, y la comparación de voltaje.

```
121 void adcConfigPin(uint32_t CH, uint32_t AP, uint32_t VRef){  
122     //Aqui se configura el pin analogo por donde tendremos la entrada del voltaje  
123     ADC_14 -> MCTL[CH] = VRef | AP;  
124 }
```



ADC_Driver.c

El proceso de conversión manda un 1 tanto al bit de encendido como al bit de ocupado.

```
135 void adcActivado(void){  
136     // Funcion bandera que activa la conversion y pone en alto la bandera de ocupado  
137     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_ENC_OFS) = 1;  
138     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_SC_OFS) = 1;  
139 }
```



ADC_Driver.c

- adcOcupado, este regresa bandera de ocupado, la cual se activa mientras se está realizando una conversión:
- (0) si no se está llevando a cabo una conversión.
- * (1) si hay una conversión en curso.

```
150 _Bool adcOcupado(void){
151     /*BAndera de ocupado, la cual se activa mientras se está realizando una conversion
152     * (0) si no se está llevando a cabo una conversión.
153     * (1) si hay una conversión en curso. */
154     return BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_BUSY_OFS);
155 }
```




ADC_Driver.c

adcResultado, regresa el valor almacenado en la memoria del canal.

```
165 uint16_t adcResultado(uint16_t CH)
166 {
167     /* Retorna el valor de la conversión ADC para el canal "channel" dado.
168      * Los valores de las conversiones se almacenan en el arreglo MEM      */
169     return ADC_14->MEM[CH];
170 }
```



ADC_Driver.c

Compara los voltajes de entrada y los resta, activa el conversor y manda la bandera a adcOcupado y guardo el valor en una variable que este mismo regresa.

```
173 float ADC_ComparadorCanal(uint16_t CH)
174 {
175     float temp = 0.0;
176     uint16_t cal30 = TLV->ADC14_REF2P5V_TS30C;
177     uint16_t cal85 = TLV->ADC14_REF2P5V_TS85C;
178     float calDiff = cal85 - cal30;
179     adcActivado(); while(adcOcupado());
180     temp = (((adcResultado(CH) - cal30) * 55) / calDiff) + 30.0f);
181     return temp;
182 }
183
```



Ejemplos

- Ejemplo de Driver con SimpleLink, sin RTOS.
- https://github.com/aalexff/ADC_Driver_SL_nortos
- Ejemplo de Driver sin SimpleLink ni RTOS.
- https://github.com/aalexff/ADC_no_SimpleLijnk
- Ejemplo de Driver con SimpleLink y RTOS.
- https://github.com/aalexff/ADC_Driver_TIRTOS_SL



Link de video

- <https://drive.google.com/drive/folders/18LpSfcFAYCA9kZmyrMNTunY83N7Fy4qs?usp=sharing>

master 1 branch 0 tags Go to file Code

aaalexff Ejemplo de Driver ADC sin simplelink ni RTOS

.settings	Ejemplo de Driver ADC sin simplelink ni RTOS
Debug	Ejemplo de Driver ADC sin simplelink ni RTOS
Drivers	Ejemplo de Driver ADC sin simplelink ni RTOS
targetConfigs	Ejemplo de Driver ADC sin simplelink ni RTOS
.ccsproject	Ejemplo de Driver ADC sin simplelink ni RTOS
.cproject	Ejemplo de Driver ADC sin simplelink ni RTOS
.project	Ejemplo de Driver ADC sin simplelink ni RTOS 16 minutes ago
main.c	Ejemplo de Driver ADC sin simplelink ni RTOS 16 minutes ago
msp432p401r.cmd	Ejemplo de Driver ADC sin simplelink ni RTOS 16 minutes ago
startup_msp432p401r_ccs.c	Ejemplo de Driver ADC sin simplelink ni RTOS 16 minutes ago
system_msp432p401r.c	Ejemplo de Driver ADC sin simplelink ni RTOS 16 minutes ago

Clone
HTTPS GitHub CLI
`https://github.com/aaalexff/ADC_no_SimpleL...`
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

Import

Select

Imports existing CCS Eclipse projects into workspace.

Select an import wizard:

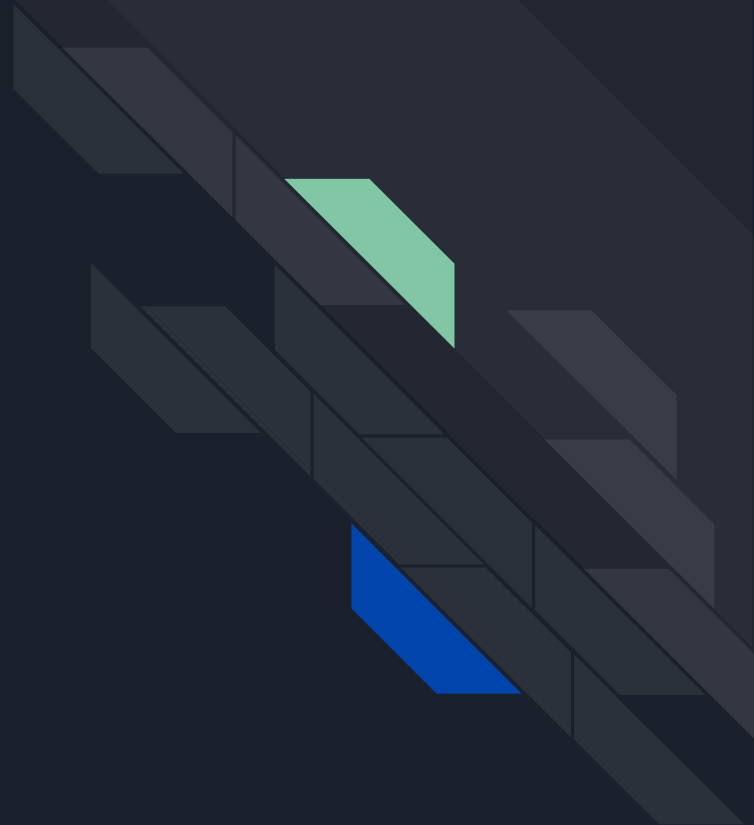
type filter text

- > General
- > C/C++
- > Code Composer Studio
 - Build Variables
 - CCS Projects
 - Legacy CCSv3.3 Projects
- > Energia
- > Git
- > Install
- > Remote Systems
- > Run/Debug
- > Team

< Back Next > Finish Cancel

Para obtener cualquier de los drivers, se pueden descargar como un ZIP desde GitHub o si se tiene GitHub desktop, se pueden clonar a un repositorio local e importarse en CCS.

English



ADC Driver design for MSP432P401R card

Alex Armando Figueroa Hernandez - 17061169
Manuel Alejandro Quiroz Gallegos - C18061043





Creation of a CCS project for the application of the ADC driver.

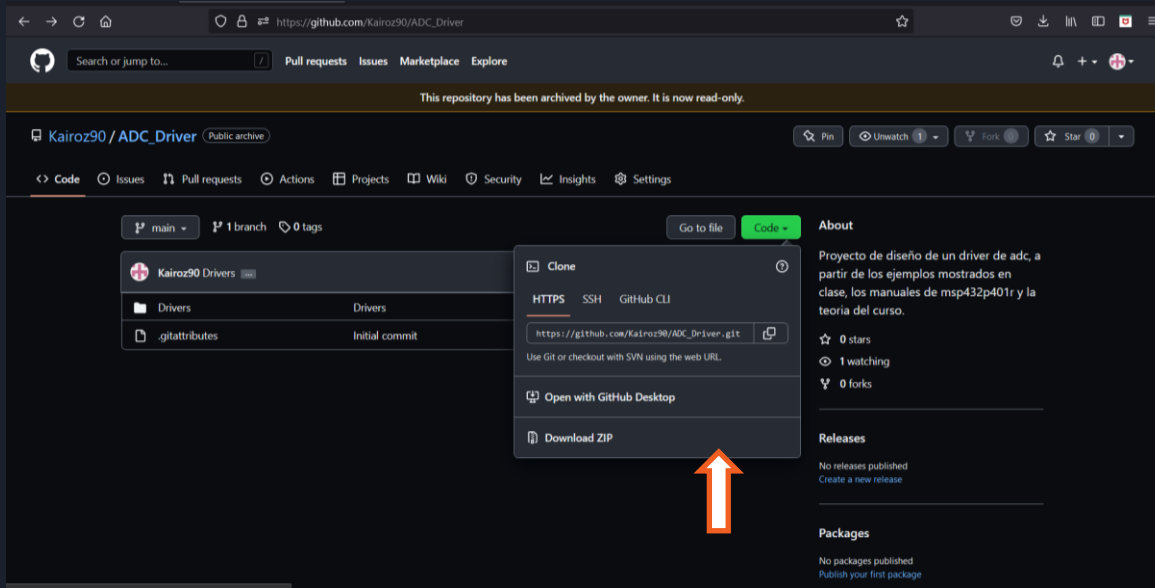
- To test the correct operation of the driver, it can be tested in a Code Composer Studio project with or without the use of SimpleLink.
- The use of SimpleLink will facilitate the use of the driver, but to check its independence, a project with SimpleLink will be tested.
- Repository link with the adc driver created: https://github.com/Kairoz90/ADC_Driver

Creation of a CCS project for the application of the ADC driver.

Download the drivers from the link as a zip and extract the file:

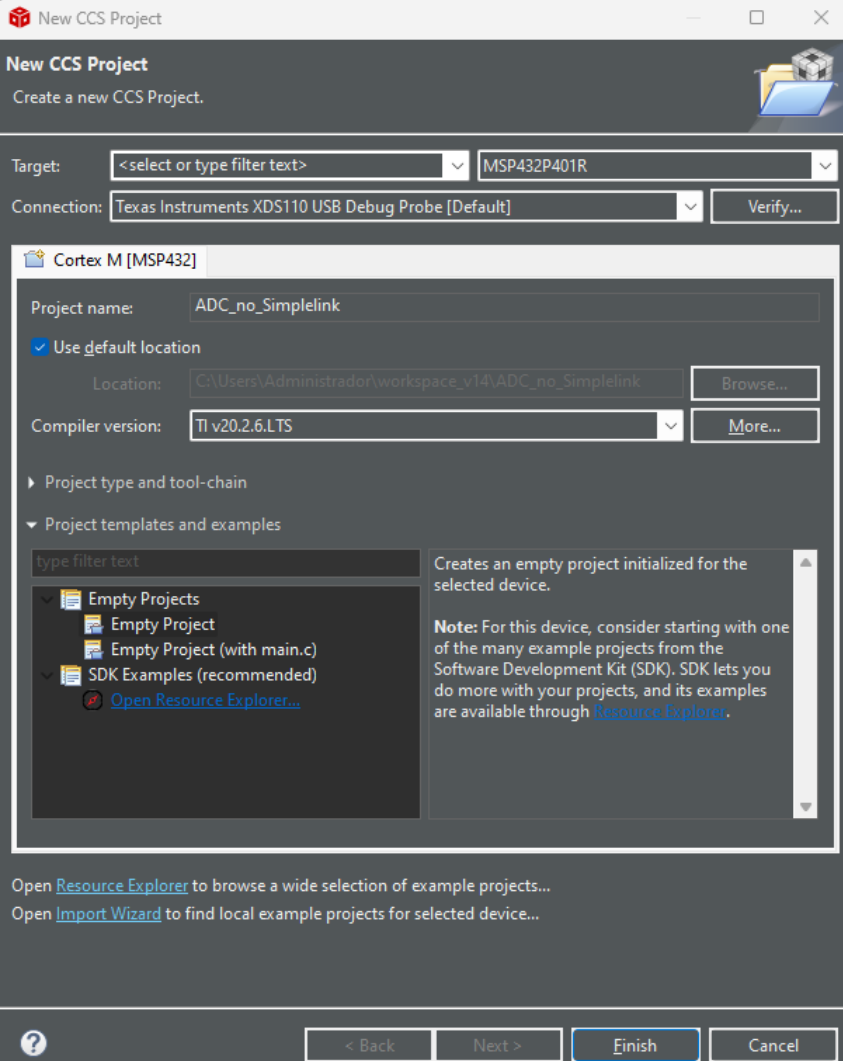
https://github.com/Kairoz90/ADC_Driver

Depending on the way of working, the installation can be with or without simplelink

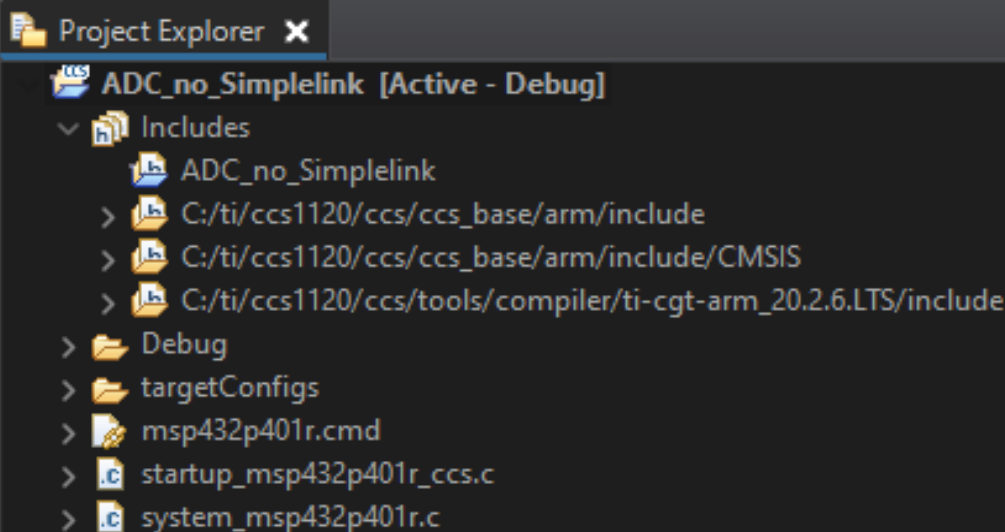




Driver without simplelink

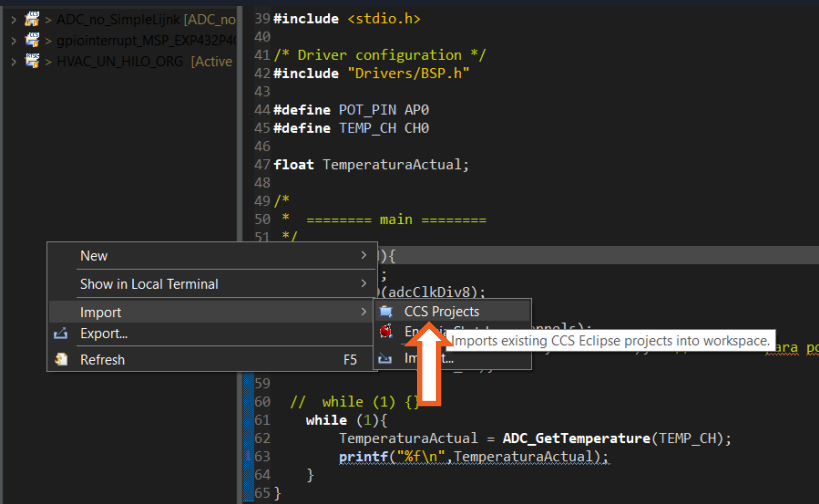


1. A new project “Empty Project” is created for the MSP432P401R card and a project name.
2. When created, it will appear in the project explorer, if you don't have simplelink, you will only have this in the includes folder.

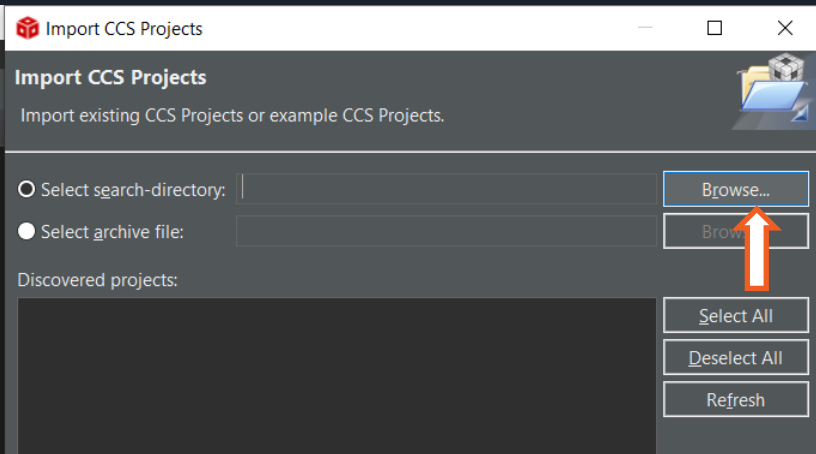


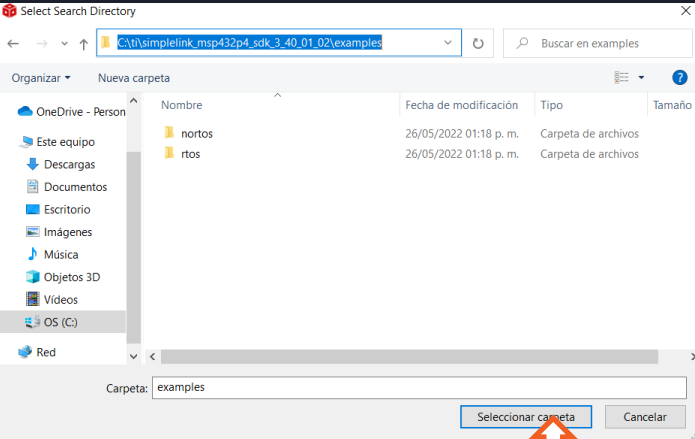


Driver with simplelink



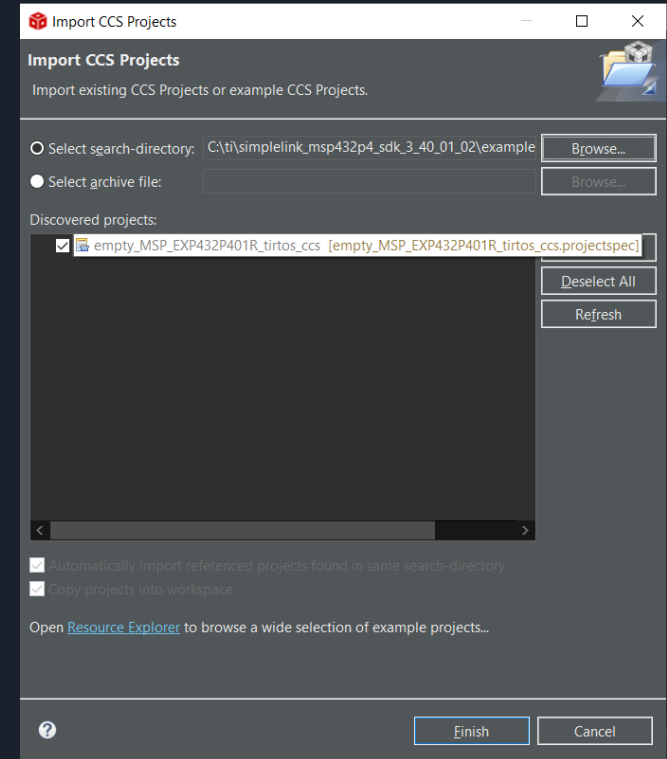
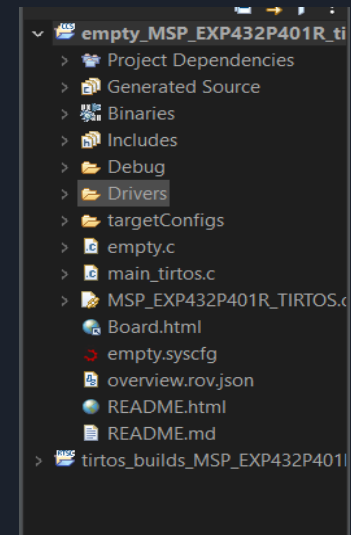
1. An "example" project for the MSP432P401R card will be imported.
2. Right click on the project explorer section and select import a ccs project.
3. Select the browse option within the tab.





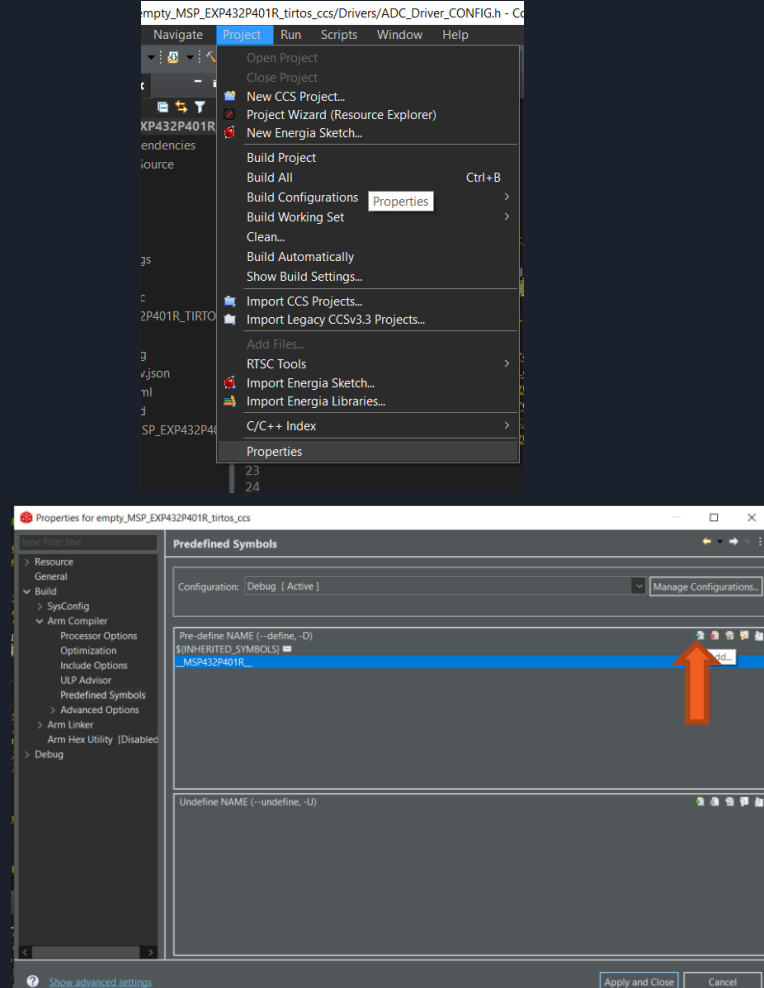
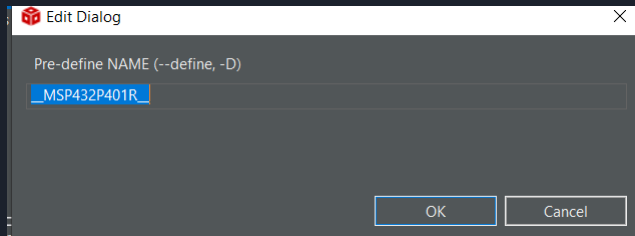
4. Look for the address inside the image or “C:\ti\simplelink_msp432p4_sdk_3_40_01_02\examples”, which may vary depending on the version of simplelink that the device that is being used has.
5. Finally, select any example, either common or common, and start modifying it to the needs of the project.

6. Finish is given and the drivers folder is imported into the project.




For both cases

1. Go to the Project bar and open properties.
2. Finally, a predefined symbol is added, in add and the symbol “__MSP432P401R__” is added, click OK and then click Apply and Clos



For both cases

1. Check if it builds the project, if not, comment out the orange arrow and uncomment the yellow arrow in the ADC_Driver_Config file.



```
*ADC_Driver_CONFIG.h x
1 /*
2  * ADC_MSP432_CONFIG.h
3  *
4  * Created on: Jun 16, 2022
5  * Author: Administrador
6  */
7
8 // #define __MSP432P401R__
9 // #include <ti/devices/msp432p4xx/inc/msp.h>
10 #include "msp.h"
11 #ifndef DRIVERS_ADC_Driver_CONFIG_H_
12 #define DRIVERS_ADC_Driver_CONFIG_H_
```




Driver definitions

- `ADC_Driver_Config.h`
- `ADC_Driver.h`



ADC_Driver_config.h

Here is the configuration table and the address of the registers of the ADC driver.



ADC_Driver_config.h

The memory address for the peripherals starts at 0x40000000, and the base for the ADC module is at 0x00012000 after starting the address for the peripherals. ADC_14 is defined as the specific type of the structure by the base address.

```
24 /*Tabla de configuracion*/
25 #define PERBASE                ((uint32_t)0x40000000)          /*!< Peripherals start address */
26 #define ADC14BASE              (PERBASE +0x00012000)          /*!< Base address of module ADC14 registers */
27 #define ADC_14                 ((ADC14_Tipo *) ADC14BASE)
28 #endif /* DRIVERS_ADC_MSP432_CONFIG_H_ */
```

ADC_Driver_config.h

The structure of the types of ADC14, are the ADC registers, including control 0, control 1, memory control, interrupt flags, flag clearing, memory storage, interrupt vector.

```
36 typedef struct {
37     __IO uint32_t CTL0;
38     __IO uint32_t CTL1;
39     __IO uint32_t L00;
40     __IO uint32_t HI0;
41     __IO uint32_t LO1;
42     __IO uint32_t HI1;
43     __IO uint32_t MCTL[32];
44     __IO uint32_t MEM[32];
45     uint32_t RESERVED0[9];
46     __IO uint32_t IER0;
47     __IO uint32_t IER1;
48     __IO uint32_t IFGR0;
49     __IO uint32_t IFGR1;
50     __IO uint32_t CLRIFGR0;
51     __IO uint32_t CLRIFGR1;
52     __IO uint32_t IV;
53 } ADC14_Tipo;

/*< Control 0 Register */
/*< Control 1 Register */
/*< Window Comparator Low Threshold 0 Register */
/*< Window Comparator High Threshold 0 Register */
/*< Window Comparator Low Threshold 1 Register */
/*< Window Comparator High Threshold 1 Register */
/*< Conversion Memory Control Register */
/*< Conversion Memory Register */

/*< Interrupt Enable 0 Register */
/*< Interrupt Enable 1 Register */
/*< Interrupt Flag 0 Register */
/*< Interrupt Flag 1 Register */
/*< Clear Interrupt Flag 0 Register */
/*< Clear Interrupt Flag 1 Register */
/*< Interrupt Vector Register */
```



ADC_Driver.h

- Here you will find the mask of the offsets of the registers of the designed ADC module.

ADC_Driver.h

Masks are defined for the offsets in the ADC registers. In this case, those that determine the type of voltage comparison. The resolution of the ADC. The clock divisions of the ADC.

```
20 #define adcVccVss      ((uint32_t)0x00000100) /*!< V(R+) = AVCC, V(R-) = AVSS */
21 #define adcVrefVss      ((uint32_t)0x00000200) /*!< V(R+) = VREF buffered, V(R-) = AVSS */
22 #define adcVrefCVss     ((uint32_t)0x00000E00) /*!< V(R+) = VeREF+, V(R-) = VeREF- */
23 #define adcVerefBufVeref ((uint32_t)0x00000F00) /*!< V(R+) = VeREF+ buffered, V(R-) = VeREF */
24
25 /*
26  * Definicion de resoluciones de ADC
27  */
28
29 #define adc8BitRes      ((uint32_t)0x00000000) /*!< 8 bit (9 clock cycle conversion time) */
30 #define adc10BitRes     ((uint32_t)0x00000010) /*!< 10 bit (11 clock cycle conversion time) */
31 #define adc12BitRes     ((uint32_t)0x00000020) /*!< 12 bit (14 clock cycle conversion time) */
32 #define adc14BitRes     ((uint32_t)0x00000030) /*!< 14 bit (16 clock cycle conversion time) */
33
34 /*
35  * Definicion de divisores de reloj de ADC, son 8 divisores
36  */
37
38 #define adcClkDiv1 ((uint32_t)0x00000000) /*!< /1 */
39 #define adcClkDiv2 ((uint32_t)0x00400000) /*!< /2 */
40 #define adcClkDiv3 ((uint32_t)0x00800000) /*!< /3 */
41 #define adcClkDiv4 ((uint32_t)0x00C00000) /*!< /4 */
42 #define adcClkDiv5 ((uint32_t)0x01000000) /*!< /5 */
43 #define adcClkDiv6 ((uint32_t)0x01400000) /*!< /6 */
44 #define adcClkDiv7 ((uint32_t)0x01800000) /*!< /7 */
45 #define adcClkDiv8 ((uint32_t)0x01C00000) /*!< /8 */
```

ADC_Driver.h

```
47 /*
48  * Definicion de predivisores de reloj de ADC
49  */
50
51 #define adcClkPreDiv1 ((uint32_t)0x00000000) /*!< Pcedivide by 1 */
52 #define adcClkPreDiv4 ((uint32_t)0x40000000) /*!< Pcedivide by 4 */
53 #define adcClkPreDiv32 ((uint32_t)0x80000000) /*!< Pcedivide by 32 */
54 #define adcClkPreDiv64 ((uint32_t)0xC0000000) /*!< Pcedivide by 64 */
55
56 /*
57  * Definicion de modos de conversion
58  */
59
60 #define adcSingleChannel ((uint32_t)0x00000000) /*!< Single-channel, single-conversion */
61 #define adcSequenceChannels ((uint32_t)0x00020000) /*!< Sequence-of-channels */
62 #define adcRepeatSingleChannel ((uint32_t)0x00040000) /*!< Repeat-single-channel */
63 #define adcRepeatSequenceChannels ((uint32_t)0x00060000) /*!< Repeat-sequence-of-channels */
64
65 // Selecciona la fuente de reloj del módulo
66
67 #define adcModSource ((uint32_t)0x00000000) /*!< MODCLK */
68 #define adcSysSource ((uint32_t)0x00080000) /*!< SYSCLK */
69 #define adcASource ((uint32_t)0x00100000) /*!< ACLK */
70 #define adcMSource ((uint32_t)0x00180000) /*!< MCLK */
71 #define adcSMSource ((uint32_t)0x00200000) /*!< SMCLK */
72 #define adcHSource ((uint32_t)0x00280000)
73
74
75 //Estos bits definen el número de ciclos ADC14CLK
76 //en el periodo de muestreo para los registros ADC14MEM0 a ADC14MEM7
77
78 #define adcFs4 ((uint32_t)0x00000000) /*!< 4 */
79 #define adcFs8 ((uint32_t)0x00001000) /*!< 8 */
80 #define adcFs16 ((uint32_t)0x00002000) /*!< 16 */
81 #define adcFs32 ((uint32_t)0x00003000) /*!< 32 */
82 #define adcFs64 ((uint32_t)0x00004000) /*!< 64 */
83 #define adcFs96 ((uint32_t)0x00005000) /*!< 96 */
84 #define adcFs128 ((uint32_t)0x00006000) /*!< 128 */
85 #define adcFs192 ((uint32_t)0x00007000) /*!< 192 */
```

The pre-clock divisions. The connection modes of the channels. The module's clock source. The sampling period.

ADC_Driver.h

Channel control. The measurement of the channels.

```
87 /*
88  * Definiciones de control de canales
89  */
90
91 #define MAX_ADC_VALUE 16383 // (2 ^14 bits)
92 #define ioAdcRunChannel (0x10000000) //Mientras esta bandera está activa, el canal está corriendo.
93 #define ioAdcFireTrigger (0x10000001) //Mientras esta bandera está activa, se dispara el trigger.
94 #define ioAdcStopChannel (0x10000002) //Si esta bandera está activa, se detiene el canal.
95 #define ioAdcStopChannels (0x10000003) //Si esta bandera está activa, se detienen todos los canales.
96 #define ioAdcPauseChannel (0x10000004) //Si esta bandera está activa, se pausan el canal.
97 #define ioAdcPauseChannels (0x10000005) //Si esta bandera está activa, se pausan todos los canales.
98 #define ioAdcResumeChannel (0x10000006) //Mientras esta bandera está activa, el canal está activo y ha sido re-iniciado.
99 #define ioAdcResumeChannels (0x10000007) //Mientras esta bandera está activa, todos los canales se activan y re-inician.
100
101 /*
102  * Definiciones de medición de canales ADC
103  */
104
105 #define adcMeasureLoop (0x00) //Se realiza una medición bajo el esquema de tiempo del modulo timer32_1
106 #define adcStartNow (0x01) //Se inicia el timer, y la medición comienza desde el inicio
107 #define adcStartTrigger (0x02) //Se realiza una medición desde el inicio
108 #define adcMeasureOnce (0x04) //Se realiza una medición activando un trigger manual.
109
110 /*
111  * Enumeración de triggers, en el ADC hay 8 triggers
112  */
113
114 enum trigger{
115     TR1, TR2, TR3, TR4, TR5, TR6, TR7, TR8
116 };
117
118 /*
119  * Enumeración de canales, en el ADC hay 31 canales
120  */
121
122 enum channel{//Buffers de conversion
123     CH0, CH1, CH2, CH3, CH4, CH5, CH6,
124     CH7, CH8, CH9, CH10, CH11, CH12, CH13,
125     CH14, CH15, CH16, CH17, CH18, CH19, CH20,
126     CH21, CH22, CH23, CH24, CH25, CH26, CH27,
127     CH28, CH29, CH30, MAXCH=31,
128 };
129
130 /*
131  * Enumeración de pins analógicos, en el ADC hay de 0 a 18, y otros para funciones específicas de 19 a 23 y el a2
132  */
133
134 enum analogPin{ // Pin analógicos de la tarjeta
135     AP0, AP1, AP2, AP3, AP4, AP5, AP6,
136     AP7, AP8, AP9, AP10, AP11, AP12, AP13,
137     AP14, AP15, AP16, AP17, AP18, AP19 = 18,
138 };
```


ADC_Driver.h

Definitions of the functions used in the ADC_Driver.c file

```
146 //Inicializa el modulo ADC.
147 extern void adcInit(void);
148 //Funcion que describe la resolución del reloj.
149 extern void adcClockR(uint32_t res);
150 //Funcion que describe el divisor de reloj
151 extern void adcClockD(uint32_t adcClkDiv);
152 //Configura el modo de conversión del ADC
153 extern void adcMode(uint32_t mode);
154 //Configura la cantidad de ciclos
155 extern void adcMSC(void);
156 /* Función que configura el canal de conversión. */
157 extern void adcConversion(uint32_t CMode);
158 //Configura el canal inicial y final de la secuencia de conversión.
159 extern void adcCanalInicial(uint32_t CH);
160 extern void adcCanalFinal(uint32_t CH);
161 /* Función que configura el pin analogo para un canal ADC. */
162 extern void adcConfigPin(uint32_t CH, uint32_t AP, uint32_t VRef);
163 /* Función que dispara la conversión ADC. */
164 extern void adcActivado(void);
165 /* Función que indica si hay una conversión en curso o no. */
166 extern _Bool adcOcupado(void);
167 /* Función que retorna el valor de la conversión ADC de un canal en específico. */
168 extern uint16_t adcResultado(uint16_t channel);
```



Driver functions

- `ADC_Driver.c`



ADC_Driver.c

- Here are the functions that are used to work with and manipulate the memory registers.

ADC_Driver.c

adcInit(void) starts the ADC module by setting the bit at the offset of the ADC14_CTL0_ON_OFS to 1.

```
10 /*
11  * Function: adcInit
12  * Preconditions: extern void adcInit() en ADC.h.
13  * Overview: Configura e inicializa el modulo.
14  * Input: Parámetros ninguno.
15  * Output: Ninguno.
16  *
17  */
18
19 void adcInit(void){
20     //El registro ADC14_CTL0[ON] Bits, activa el modulo en 1
21     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_ON_OFS) = 1;
22     /*CTL0 = Control 0 Register */
23 }
```



ADC_Driver.c

- adcMSC Sets whether the multisample conversion is one edge per rising or one edge for all samples.

```
36 void adcMSC(void){  
37     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_MSC_OFS  ) = 0;  
38 }
```



ADC_Driver.c

adcClockD asks for a variable that determines the clock division, the ADC14_CTL0_SHP_OFS starts the clock division, and ADC_14 -> CTL0 sets the divider.

```
49 void adcClockD(uint32_t adcClkDiv){
50     //Aqui se configura el reloj.
51     ADC_14 -> CTL0 |= adcClkDiv | ADC14_CTL0_SHT1__64 /*La division*/ | ADC14_CTL0_SHT0__192;
52     /*ADC 14((ADC14_Type *) ADC14_BASE) */
53     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_SHP_OFS) = 1;
54 }
```



ADC_Driver.c

The resolution of the clock is set by giving it a variable.

```
65 void adcClockR(uint32_t res){  
66     //ADC14CTL1 configuramos la resolucion a la que va a trabajar el conversor  
67     ADC_14 -> CTL1 = res ;  
68     /*CTL1 = Control 1 Register */  
69  
70 }
```



ADC_Driver.c

The variable requested here sets the conversion mode.

```
80 void adcConversion(uint32_t CMode){  
81     //Aqui configuramos cual va a ser el modo de conversion mediante el registro ADC14CTL0  
82     ADC_14 -> CTL0 |= CMode;  
83 }  
--
```




ADC_Driver.c

The final channel is initialized with a 1 bit to the offset of ADC14_MCTLN_EOS_OFS, AND a channel variable to the memory control register.

```
93 void adcCanalFinal(uint32_t CH){  
94     // Aqui elegimos cual es la direccion final del canal para el uso de modo secuencia  
95     BITBAND_PERI(ADC_14->MCTL[CH], ADC14_MCTLN_EOS_OFS) = 1;  
96 }
```



ADC_Driver.c

The initial channel is set in the same way, which sends a variable indicating the channel.

```
107 void adcCanalInicial(uint32_t CH){  
108     //Caso contrario, este serpa la direccion inicial del canal para el uso de modo secuencia  
109     ADC_14->CTL1 |= (CH<<ADC14_CTL1_CSTARTADD_OFS);  
110 }
```



ADC_Driver.c

adcConfigPin configures the analog pin where the voltage is compared, the variables of the indicated channel, the indicated pin, and the voltage comparison are used.

```
121 void adcConfigPin(uint32_t CH, uint32_t AP, uint32_t VRef){  
122     //Aqui se configura el pin analogo por donde tendremos la entrada del voltaje  
123     ADC_14 -> MCTL[CH] = VRef | AP;  
124 }
```



ADC_Driver.c

The conversion process sends a 1 to both the power on bit and the busy bit.

```
135 void adcActivado(void){  
136     // Funcion bandera que activa la conversion y pone en alto la bandera de ocupado  
137     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_ENC_OFS) = 1;  
138     BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_SC_OFS) = 1;  
139 }
```



ADC_Driver.c

- adcOcupado, this returns busy flag, which is set while a conversion is taking place:
- (0) if no conversion is taking place.
- * (1) if a conversion is in progress.

```
150 _Bool adcOcupado(void){  
151     /*BAndera de ocupado, la cual se activa mientras se está realizando una conversion  
152     * (0) si no se está llevando a cabo una conversión.  
153     * (1) si hay una conversión en curso. */  
154     return BITBAND_PERI(ADC_14->CTL0, ADC14_CTL0_BUSY_OFS);  
155 }
```



ADC_Driver.c

adcResultado, returns the value stored in channel memory.

```
165 uint16_t adcResultado(uint16_t CH)
166 {
167     /* Retorna el valor de la conversión ADC para el canal "channel" dado.
168      * Los valores de las conversiones se almacenan en el arreglo MEM */
169     return ADC_14->MEM[CH];
170 }
```



ADC_Driver.c

It compares the input voltages and subtracts them, activates the converter and sends the flag to adccupado and saves the value in a variable that it returns.

```
173 float ADC_ComparadorCanal(uint16_t CH)
174 {
175     float temp = 0.0;
176     uint16_t cal30 = TLV->ADC14_REF2P5V_TS30C;
177     uint16_t cal85 = TLV->ADC14_REF2P5V_TS85C;
178     float calDiff = cal85 - cal30;
179     adcActivado(); while(adcOcupado());
180     temp = (((adcResultado(CH) - cal30) * 55) / calDiff) + 30.0f);
181     return temp;
182 }
183
```



Examples

- Driver example with SimpleLink, without RTOS.
- https://github.com/aalexff/ADC_Driver_SL_nortos
- Driver example without SimpleLink or RTOS.
- https://github.com/aalexff/ADC_no_SimpleLijnk
- Driver example with SimpleLink and RTOS.
- https://github.com/aalexff/ADC_Driver_TIRTOS_SL



Video link in spanish

- <https://drive.google.com/drive/folders/18LpSfcFAYCA9kZmyrMNTunY83N7Fy4qs?usp=sharing>

master 1 branch 0 tags

Go to file

Code

aaalexff

Ejemplo de Driver ADC sin simplelink ni RTOS

.settings	Ejemplo de Driver ADC sin simplelink ni RTOS	
Debug	Ejemplo de Driver ADC sin simplelink ni RTOS	
Drivers	Ejemplo de Driver ADC sin simplelink ni RTOS	
targetConfigs	Ejemplo de Driver ADC sin simplelink ni RTOS	
.ccsproject	Ejemplo de Driver ADC sin simplelink ni RTOS	
.cproject	Ejemplo de Driver ADC sin simplelink ni RTOS	
.project	Ejemplo de Driver ADC sin simplelink ni RTOS	16 minutes ago
main.c	Ejemplo de Driver ADC sin simplelink ni RTOS	16 minutes ago
msp432p401r.cmd	Ejemplo de Driver ADC sin simplelink ni RTOS	16 minutes ago
startup_msp432p401r_ccs.c	Ejemplo de Driver ADC sin simplelink ni RTOS	16 minutes ago
system_msp432p401r.c	Ejemplo de Driver ADC sin simplelink ni RTOS	16 minutes ago

Clone

HTTPS

GitHub CLI

https://github.com/aaalexff/ADC_no_SimpleL:

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

Import

Select

Imports existing CCS Eclipse projects into workspace.

Select an import wizard:

type filter text

General

C/C++

Code Composer Studio

Build Variables

CCS Projects

Legacy CCSv3.3 Projects

Energia

Git

Install

Remote Systems

Run/Debug

Team

< Back

Next >

Finish

Cancel

To obtain any of the drivers, they can be downloaded as a ZIP from GitHub or if you have GitHub desktop, they can be cloned to a local repository and imported into CCS.