Antonio Martí Campoy
amarti@disca.upv.es

Introduction:

In following exercises, you will configure and use the ADC to read the position of the joystick and control the car.
For the first exercises you will create a new project without the game. This way you will focus only in the configuration and programming of the ADC, making easier to progress.
Later, you will incorporate your functions for analog acquisition to the previous project where you were completing the video game.

Exercise 9.

It could be easier to develop and check the code for the analog to digital converter using a new, empty project without the game library. Create a new project following the steps of 3_Starting_ccs.pdf but without adding the game library.

Once created the new project, create a new module called *analog* (.c and .h) where you are going to add the functions to configure and read the ADC inputs.

In this exercise you are going to work with the internal temperature sensor. It is an analog device, so you must configure and use the precision ADC (Analog-to-Digital-Converter) included in MSP432P401R microcontroller. As you can read in the technical guide the sample period must be greater than **5 microseconds** (>5 µs). This is a starting point to choose the clock source and the dividers for ADC14CLK when using the sampling timer, that is, the sample pulse mode (ADC14SHP = 1).

The range of sampling periods is from 4 to 192 ADC14CLK periods. The frequency of ADC14CLK that we want results from the formula:

Frequency of ADC14CLK = 1 / (period sampling / number of sampling periods)

Please, complete the following table with the values of frequency for ADC14CLK that you can use to guarantee a sampling period over 5µs using the possible values of sampling periods.

Sampling period	number of sampling periods	Frequency of ADC14CLK MHz
5 µs	4	1 / (5µs /4) = 0,8
5 µs	8	
5 µs	16	
5 µs	32	
5 µs	64	
5 µs	96	
5 µs	128	
5 µs	192	

Several combinations of frequency and number of sampling periods are suitable, always choosing a frequency larger than the values shown in the table. The game library configures the DCOCLK, MCLK and HSMCLK at 24MHz. You can divide HSMCLK by 2, getting an ADC14CLK of 12MHz. Using a sampling period of 64 ADC14CLK periods you get a sampling period of 5,3µs, or using 96 ADC14CLK periods you get a sampling period of 8µs.

You need to find the analog input where de temperature sensor is connected, and select a memory register to store the conversion. I suggest you use ADC14MEM0.

Now that you have all the information, (ADC14CLK source and divider, sampling period, input channel and memory register), write a function called void *init_adc14(void)* to configure the ADC with the following settings (Please, use names of fields like ADC14_CTL0_SSEL__HSMCLK. You can find these constants in `msp432p401r.h`: lines 1427 and following):

TIP: The first time you are configuring ADC14CTLO0 and ADC14CTL1 you can clear first the registers and later set to 1 the bits you need.

- Select HSMCLK as clock source for ADC14CLK, predividing by 1 and dividing by 2.
- Select ADC14SC bit to start a conversion, select Pulse Sample Mode, and single-channel single-conversion operation mode.
- Select 96 sampling periods for ADC14MEM0.
- Switch on the ADC, leave ENC and other bits of ADC14CTLO to 0.
- Set the conversion address start to ADC14MEM0.
- Set the ADC14 resolution to 14 bits.

Now, create a function called *void config_temp(void)*. In this function you have to:

- Select in ADC14CTL1 the temperature sensor, setting the bit ADC14TCMAP.
- Switch on the voltage reference and enable powering the temperature sensor: REF_A->CTL0 |= REF_A_CTL0_ON; REF_A->CTL0 &= ~REF_A_CTL0_TCOFF;
- Configure register ADC14MCTL0 (0 because we choose it) selecting the voltage reference Vref+ = Vcc and Vref- = Vss) and set the input channel.
- Read the register ADC14MEM0 and discard the reading, this way you are sure the corresponding IFG bit is cleared.

Write another function called `uint16_t read_temp(void)`. In this function you have to clear the ENC bit (to create a low to high transition) and start the conversion (setting ENC and SC bits). Then, wait in a loop until the bit 0 in IFG is set. Finally, return the value of ADC14MEM0.

In function main create a while(1) loop and inside this loop call function read_temp and store in a variable the returned value. Then, use printf (remember to include stdio.h) to send the read value to the console (write one value per line). Don't forget to call init_temp() and before entering the while.

When running the program, open the console from menu view. If the value you read is constant, your reading is wrong and you have to ask teacher. If you see a changing value, pass to the next exercise.

Antonio Martí Campoy
amarti@disca.upv.es

Exercise 10.

Add to module *analog* a new function called *float convert_to_volts (uint16_t reading)*. This function receives the value read by the ADC and convert it to volts. In order to carry out the conversion, you need to know the voltage of Vcc. Do you know where to find it? Use printf() to show in the console the value from *float convert_to_volts*. If the value is in the range [0.6, 0.9], pass to the next exercise. Otherwise ask teacher.

Exercise 11.

Add to module *analog* a new function called *float convert_to_Celsius (float volts)*. This function receives the value read by the ADC converted to volts and convert it to grades Celsius. Use printf to show the temperature. Check the sensor transfer function in technical reference or datasheet to get the equation of the transfer line.
Warning: Show your code to the teacher.

Exercise 12.

Do you know how to pass parameters by reference? It is time to apply this technique. Add to module *analog* a function called void *all_values_temp(???)*. This function will return three values: the reading from the ADC, its conversion to volts and its conversion to Celsius. Inside this function you can call the previous functions (code reuse). In function main substitute the calls to read_temp(), convert_to_volts() and convert_to_Celsius() by a unique call to all_values_sensor(). Use a unique printf to send the three values to the console.
Warning: Show your code to the teacher.

Exercise 13.

I told you that the sample and time hold must be greater than 5 µs. What do you think that may happen if you don't meet this requirement? Do you want to know? Try yourself!!! In function *init_adc14()* reduce the number of sampling periods to 4. Run again the program and tell the teacher the temperature you are reading. Then, don't forget to restore to 96 the number of sampling periods.

Exercise 14.

You are going to work with the two-axis joystick. Look at technical document the port and pins, and the corresponding analog channel where the joystick outputs are connected.

Since we want to read two analog inputs, we will use the operation mode sequence-of-channels. To read a sequence of channel in an automatic way we need to meet the following requirements:

- Memory registers (ADC14MEMx) must be consecutive. I suggest you use ADC14MEM3 and ADC14ME4 (this way you can reuse *init_adc14()*).
- Set operation mode to sequence of channels, and, to automatically start the conversion of the second input, the bit ADC14MSC must to be set to 1.
- Set the initial conversion address.
- Set the bit ADC14EOS in ADC14->MCTL[4] to stop the sequence of conversion.
- Don't forget to configure DIO pins as tertiary function (PxSELx) assign the input channels to the corresponding ADC14MCTLx registers.

Create a function called *void config_joystic (void)* to configure all the registers related to the Joystick. Don't forget that you need to clear the bit ENC to modify some values. Also, at the end of the function read and discard de contents to be sure the IFG bits are cleared.

For the basic ADC14 configuration you can use *init_adc14()* (Each axis uses a potentiometer to build a voltage divider, so the sampling time may be very short).

Create a function called *void read_joystick (????)* that returns in parameters the reading of axis x and y of the joystick. You don't need to convert to volts. Remember that you have to use only one loop to wait for the finish of the second acquisition.

In function main, use the functions to configure and read from the joystick, and send the acquired values to the console. Annotate the range of values you get.

Warning: Show your code to the teacher.

Exercise 15.

Now it is time to integrate the joystick in the car driving game. You can copy your files analog.c and analog.h to the previous project where you develop the module digital.

Modify the main loop to control the steering wheel and the speed using the joystick. In this exercise you can made a basic control, that is, if the position of the joystick is close to the end of run, add +1 or -1 to the steering wheel or speed, as appropriate.

Warning: Show your code to the teacher.

Exercise 16.

The Booster Pack includes a 3-axis accelerometer that you can use to control de car. Add to your project two functions, *config_accel()* and *read_accel()* to read the three axis. It could a good idea to develop the functions in the project you created without the game, in the same way you developed the use of temperature sensor and joystick.
Warning: Show your code to the teacher.

Exercise 17.

Improve the previous exercise with a proportional control of the steering wheel and the speed depending on the position of the joystick (or the accelerometer, you choose). That's is, turn left or right faster as the joystick is more far away from the central position. Same for speed.

Warning: Show your code to the teacher.