PERIPHERAL EQUIPMENT
2023 – 2024

PRACTICE 2

By Adrián Sánchez López

1. Practice 1 changes

Regarding its behavior, some changes were made regarding the practice 1 generation of the PWM. As before, I had an array storing the different speeds in Km/h from where the speed of the car was taken depending on the number of times the button was pressed. Therefore, in order to generate the PWM's this speed had to be converted from Km/h to m/s to revolutions to period and then halved. As these operations take an incredibly high amount of time, to further optimize the project I changed the speeds stored in the array to the periods halved, so now there is nothing to be calculated, the value of the speed is the value I directly use to generate the PWM.

When it comes to GPIOs, the ports PD4 and PD5, which are the input ports that listen to left and right wheel PWMs have been changed to the ports PC0 & PC1 as while taking out the operations done to generate the PWM(as explained before and which do not affect in anything PD4 and PD5 interrupt), one stopped working while the other was generating an interrupt constantly. Due to that, I ended up changing those ports with the ports PC0 and PC1.

2. ADC Implementation

For the ADC the ADC1 has been used, I could have used any other one, but this was just the first to appear in the channels of stream 0 of the DMA which is the DMA used for peripheral to memory transfer. Also, to take a sample of each sensor each 50 us during 8 times each 1ms, the timer of 1 ms has been used to set a flag and a timer that throws an interrupt each 50 us has been implemented. Then in the interruption of this timer of 50 us, if the flag of the timer of 1ms is set and less than 8 samples have been taken it starts an ADC conversion. If 8 samples have been taken and the flag of the 1ms timer is set, it resets the number of samples taken to 0, unsets the flag of the 1ms timer and enables the DMA2_stream1 which is the one that makes the memory transfer. Note that the adc is configured to make the conversions on multiple channels, and to not start a conversion after one, so the conversion is done just once and when I call the function to start the conversion in the interrupt of 50us under the appropriate conditions.

For what it has to do with the ports, the ports PA1 and PA2 have been used as those are the input1 and input2 ports for each of the ADC. They have been set as analog and in the adc the channels 1 and 2 which, as said, are inputs PA1 and PA2 have been set.

3. DMA Implementation

From all the DMA's, the DMA2 has been used as it is stated that it is the only DMA that allows for memory to memory transfers. Then the stream0 channel0 is used to transfer from peripheral to memory. Stream0 channel0 is explicitly used, as it is the channel connected to ADC1. Then the stream1 channel1 is used to transfer from memory to memory, as it is memory to memory, no peripheral is used, so no specific channel attached to a peripheral is needed.

Regarding DMA stream0 channel 0, it has been set as peripheral to memory as said, with the proper addresses of both the peripheral and the memory where I want the samples stored. The buffer size is set to 16 bits as it is the closest option towards the 12 bits that each sample occupies. The memory increment is enabled as I want the dma to increase the address of the array for each transfer and the peripheral inc is disabled as the address of the ADC will remain always the same. Finally, it has been set as circular mode as I want the DMA to reset the data counter to 0 when it reaches the end of the array where it stores the samples from the adc.

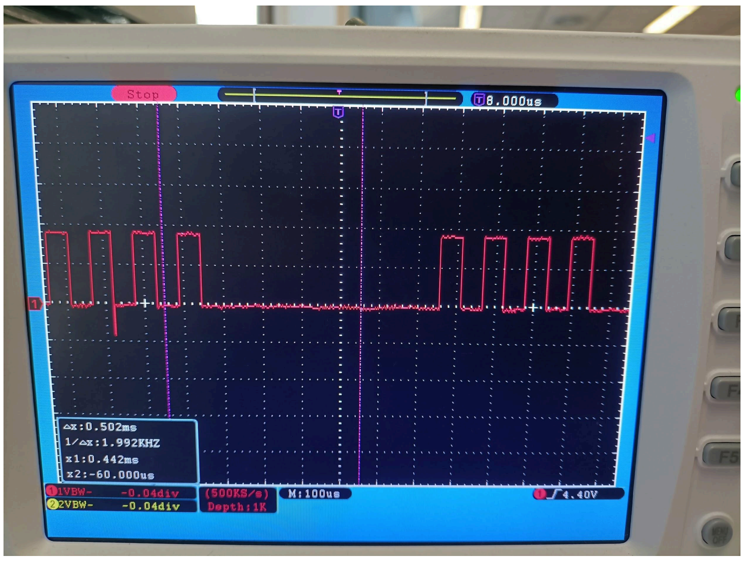
Regarding DMA steam1 channel1, it has been set as memory to memory, with the proper address of the array where the samples are and the array where I want it to transfer and store them. For both arrays I have the memory increment enabled as I want it to increment the position and it is required for the bust mode. The data sizes have been set to 16 bits and the mode is normal, so it makes the transfer each time I enable it, and once done it is disabled. And the burst is set to single so each I enable it transfers the whole array at once and gets disabled until I want to enable it again for the next transfer.

In addition, for the stream1 flag that indicates the completion of a transfer, I created an interrupt so each time I enable the stream1 for it to transfer the whole data in the adc timer when 8 samples have been taken, after they are transferred it throws an interrupt in which I calculate the mean value of the 8 samples.

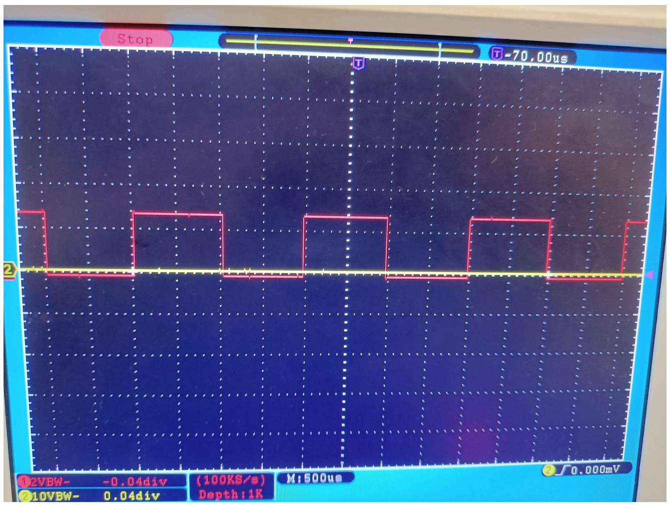
4. Validation methods

To check that the ADC converted the values well, with the debugger I checked that the value generated after the conversion was consistent with the voltage being applied to the port. In addition, to check that the ADC acquires the samples at the specified intervals and at the requested rate, I toggled the free led available each time a conversion is done to check with the oscilloscope that each millisecond there where 8 consecutives toggles in the signal, each one done each 50us.

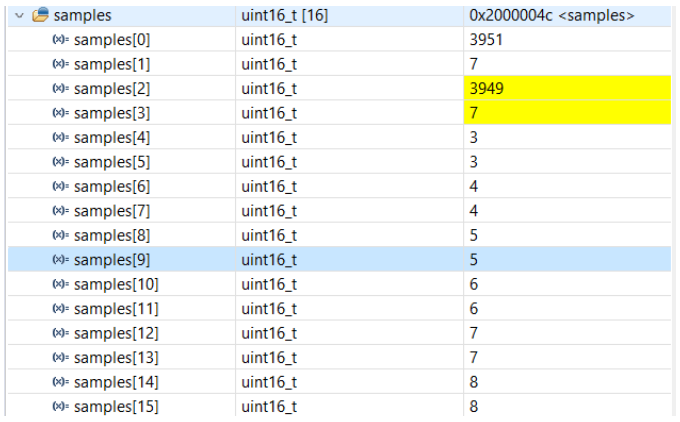
As it can be seen in the picture, there are 4 rising and falling edges, so in total 8, which is the number of samples(remember that each toggle to the signal is made after a sample is done). Moreover, each change happens half a square after the previous change and each square is 100us. Therefore, each change occurs after 50us. Finally, after the 8th change(samples) the next change occurs just 6 squares(600 us) later, which adds to the 400 us that have already passed while taking samples, it adds up to 1ms which is the rate at which samples must be taken.



In addition, it also has been put a bit toggle on the free led when the stream1 transfer(memory to memory) is made and as seen in the result, each transfer is made 1 ms after the previous.

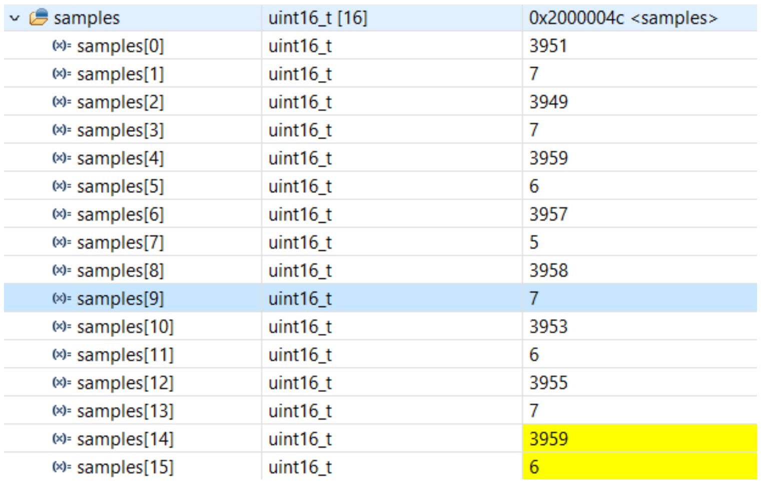


Finally to check that both arrays had the same data after each transfer the debugger was used to check both array values. And to check that in the first array the values of the conversions where being stored in the correct order the debugger has been used too, to check that the values follow the order as specified in the table.

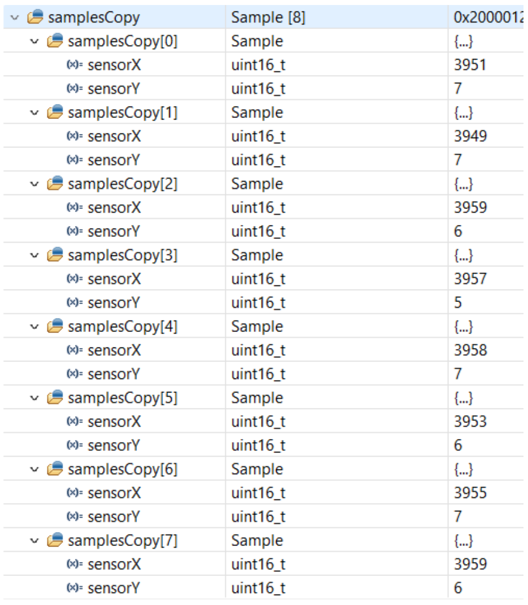


samples	uint16_t [16]	0x2000004c <samples>
samples[0]	uint16_t	3951
samples[1]	uint16_t	7
samples[2]	uint16_t	3949
samples[3]	uint16_t	7
samples[4]	uint16_t	3
samples[5]	uint16_t	3
samples[6]	uint16_t	4
samples[7]	uint16_t	4
samples[8]	uint16_t	5
samples[9]	uint16_t	5
samples[10]	uint16_t	6
samples[11]	uint16_t	6
samples[12]	uint16_t	7
samples[13]	uint16_t	7
samples[14]	uint16_t	8
samples[15]	uint16_t	8

Firstly, in this picture it can be seen how the values of the conversion are stored correctly from two to two and in the correct order.



samples	uint16_t [16]	0x2000004c <samples>
samples[0]	uint16_t	3951
samples[1]	uint16_t	7
samples[2]	uint16_t	3949
samples[3]	uint16_t	7
samples[4]	uint16_t	3959
samples[5]	uint16_t	6
samples[6]	uint16_t	3957
samples[7]	uint16_t	5
samples[8]	uint16_t	3958
samples[9]	uint16_t	7
samples[10]	uint16_t	3953
samples[11]	uint16_t	6
samples[12]	uint16_t	3955
samples[13]	uint16_t	7
samples[14]	uint16_t	3959
samples[15]	uint16_t	6



samplesCopy	Sample [8]	0x200001
samplesCopy[0]	Sample	{...}
sensorX	uint16_t	3951
sensorY	uint16_t	7
samplesCopy[1]	Sample	{...}
sensorX	uint16_t	3949
sensorY	uint16_t	7
samplesCopy[2]	Sample	{...}
sensorX	uint16_t	3959
sensorY	uint16_t	6
samplesCopy[3]	Sample	{...}
sensorX	uint16_t	3957
sensorY	uint16_t	5
samplesCopy[4]	Sample	{...}
sensorX	uint16_t	3958
sensorY	uint16_t	7
samplesCopy[5]	Sample	{...}
sensorX	uint16_t	3953
sensorY	uint16_t	6
samplesCopy[6]	Sample	{...}
sensorX	uint16_t	3955
sensorY	uint16_t	7
samplesCopy[7]	Sample	{...}
sensorX	uint16_t	3959
sensorY	uint16_t	6

Then by comparing those two images showing the first the contents of the first array and the second the contents of the array where they are transferred, it can be sen how both have the same values.