



# ***Universidad Nacional de Córdoba***

*Facultad de Ciencias Exactas, Físicas y Naturales  
Sistemas de computación*

## ***TP1: Rendimiento (Parte 1)***

Grupo:

*Epsilon*

Profesores:

*Jorge, Javier Alejandro  
Lamberti, Germán Andrés  
Solinas, Miguel Ángel*

Alumnos:

*Campos, Mariano  
González, Damián Marcelo*



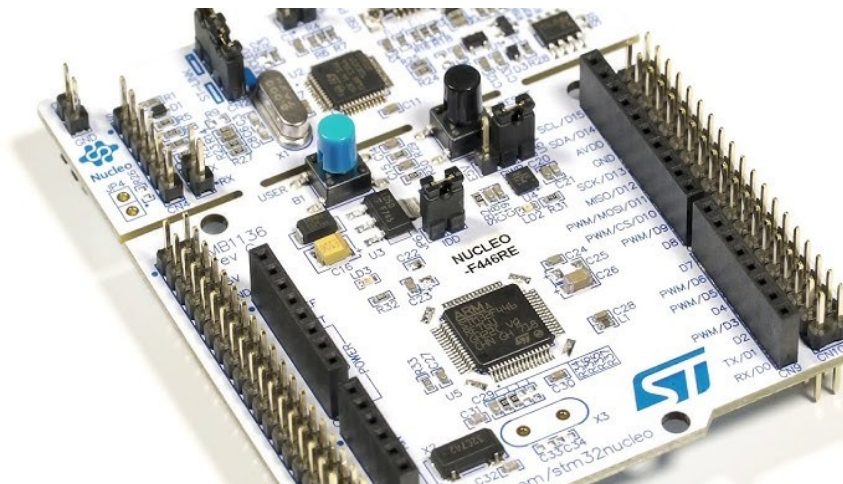
## Consigna

Conseguir un esp32 o cualquier procesador al que se le pueda cambiar la frecuencia. Ejecutar un código que demore alrededor de 10 segundos. Puede ser un bucle for con sumas de enteros por un lado y otro con suma de “floats” por otro lado. ¿Qué sucede con el tiempo del programa al duplicar (variar) la frecuencia ?

## Desarrollo

Para el desarrollo del trabajo practico utilizamos la placa de desarrollo NUCLEO-F446RE que cuenta con STM32F446RE con las siguientes características:

- Núcleo: ARM Cortex-M4 a 180 MHz con unidad de punto flotante (FPU).
- Tamaño de palabra: 32 bits.
- Flash: 512 KB de memoria Flash interna.
- RAM: 128 KB de memoria SRAM.
- Unidad de punto flotante (FPU) de 32 bits, que mejora el procesamiento de operaciones matemáticas complejas.
- Coprocesador de multiplicación para mejorar la eficiencia en cálculos.



Para la experiencia realizamos un código con un bucle “for” con una determinada cantidad de instrucciones para que este demore 10 segundos (micro a 80MHz) posteriormente cambiamos la velocidad de reloj a 180MHz, y calculamos el “speed-up”. Dentro del bucle “for” se van a realizar operaciones de enteros y putos flotantes para ver la diferencia que hay en el costo computacional de las operaciones y su influencia en el rendimiento.

Para calcular el tiempo de duración del bucle tenemos los siguientes cálculos:

$$F_{cpu} = 80 [MHz]$$

$$T_{cpu} = \frac{1}{F_{cpu}}$$

$$T_{ins} = CPI * T_{cpu}$$

Finalmente:

$$T_{prog} = n^{\circ} ins * T_{ins}$$

El CPI puede ser difícil de calcular porque varía según la arquitectura y según las instrucciones que se estén ejecutando, en los ARM Cortex-M4 en general cada instrucción tarda un ciclo (se comprueba en la medición), pero existen algunas que pueden tardar más.

El módulo que permite contabilizar los CPI (Ciclos por Instrucción) en los microcontroladores Cortex-M4 es el DWT (Data Watchpoint and Trace). Este módulo incluye un contador de ciclos que se puede utilizar para medir el tiempo que se tarda en ejecutar un bloque de código, lo cual es útil para analizar el rendimiento y calcular los CPI.

Implementación del código:

```
volatile uint32_t start, end, cycles;
volatile float execution_time;

void SystemClock_Config(void);

void DWT_Init(void);
uint32_t DWT_GetCycles(void);

int main(void)
{
    HAL_Init();

    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    DWT_Init();

    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    start = DWT_GetCycles();

    for(uint32_t i=0; i<32000000; i++){
        uint32_t a = 5;
        uint32_t b = 3;
        uint32_t c = a + b;
    }

    end = DWT_GetCycles();
    cycles = end - start;
    execution_time = (float)(cycles / CPU_FREQ_HZ);
    While (1){}
```

```
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    __HAL_RCC_PWR_CLK_ENABLE();
    __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    RCC_OscInitStruct.PLL.PLLM = 8;
    RCC_OscInitStruct.PLL.PLLN = 80;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
    RCC_OscInitStruct.PLL.PLLQ = 2;
    RCC_OscInitStruct.PLL.PLLR = 2;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }






    /** Initializes the CPU, AHB and APB buses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                                  |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
    {
        Error_Handler();
    }
}
```






```
void DWT_Init(void) {
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // Habilita el DWT
    DWT->CYCCNT = 0; // Reinicia contador
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; // Habilita el contador
}

uint32_t DWT_GetCycles(void) {
    return DWT->CYCCNT;
}
```

Resultados de la mediciones reloj 80MHz operaciones con enteros:

Expression	Type	Value
 cycles	volatile uint32_t	800583780
 execution_time	volatile float	10.0072975
 start	volatile uint32_t	70
 end	volatile uint32_t	800583850
 Add new expression		

Resultados mediciones reloj 80MHz operaciones con punto flotante:

Expression	Type	Value
 cycles	volatile uint32_t	992729066
 execution_time	volatile float	12.4091139
 start	volatile uint32_t	70
 end	volatile uint32_t	992729136
 Add new expression		

Calculo de rendimiento con enteros:

$$\eta = \frac{1}{T_{prog}} = \frac{1}{10.0072975[s]} = 0.099927$$

Calculo de rendimiento con punto flotante:

$$\eta = \frac{1}{T_{prog}} = \frac{1}{12.4091139[s]} = 0.080585$$






Incremento de ciclos :

$$n^{\circ} \text{ ciclos} = 992729066 - 800583780 = 192145286$$






Tenemos una disminución del rendimiento del 20% aproximadamente

$$Speed-up = \frac{\eta_{flotante}}{\eta_{entero}} = \frac{0.080585}{0.099927} = 0.806$$

Resultados de la mediciones reloj 180MHz operaciones con enteros:

Expression	Type	Value
 cycles	volatile uint32_t	800583780
 execution_time	volatile float	4.44768763
 start	volatile uint32_t	70
 end	volatile uint32_t	800583850
 + Add new expression		

Resultados mediciones reloj 180MHz operaciones con punto flotante:

Expression	Type	Value
 cycles	volatile uint32_t	992729066
 execution_time	volatile float	5.51516151
 start	volatile uint32_t	70
 end	volatile uint32_t	992729136
 + Add new expression		

Calculo de rendimiento con enteros:

$$\eta = \frac{1}{T_{prog}} = \frac{1}{4.44768763[s]} = 0.224835$$

Calculo de rendimiento con punto flotante:

$$\eta = \frac{1}{T_{prog}} = \frac{1}{5.51516151[s]} = 0.181318$$

Incremento de ciclos :

$$n^{\circ} \text{ ciclos} = 992729066 - 800583780 = 192145286$$

Tenemos una disminución del rendimiento del 20% aproximadamente

$$Speed-up = \frac{\eta_{flotante}}{\eta_{entero}} = \frac{0.181318}{0.224835} = 0.806$$

Calculo de mejora respecto al aumento de frecuencia (124.9%):

$$Speed-up = \frac{\eta_{180 \text{ MHz}}}{\eta_{80 \text{ MHz}}} = \frac{0.224835}{0.099927} = 2.249$$