MCXA153 - Coremark Benchmark

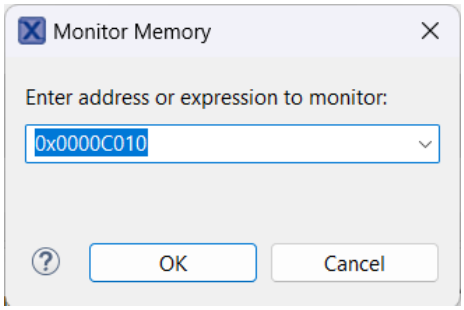
El proyecto cuenta con un archivo de configuración: “MCXA153_benchmark_cfg”, donde se incluyen las siguientes macros:

- EXECUTION_TIME_FREQ : Indica la frecuencia con la que se ejecutará el código del coremark. El valor colocado en esta macro debe estar en segundos, y se configura de acuerdo al tamaño de la sección de memoria donde se almacenarán los resultados y el tiempo total que se espera que la tarjeta recolecte datos.
  Se debe considerar que cada ejecución de prueba necesita 48 bytes para reporte de resultados, y que tarda 46 segundos aproximadamente en finalizarse.
- CLEAN_MEMORY : Esta macro es utilizada para activar la parte del código encargado de limpiar la sección de memoria desgnada para almacenar los datos recolectados de las pruebas.
  La macro debe tener valor ‘1’ para limpiar la memoria, o ‘0’ para ejecutar el coremark y almacenar los resultados.

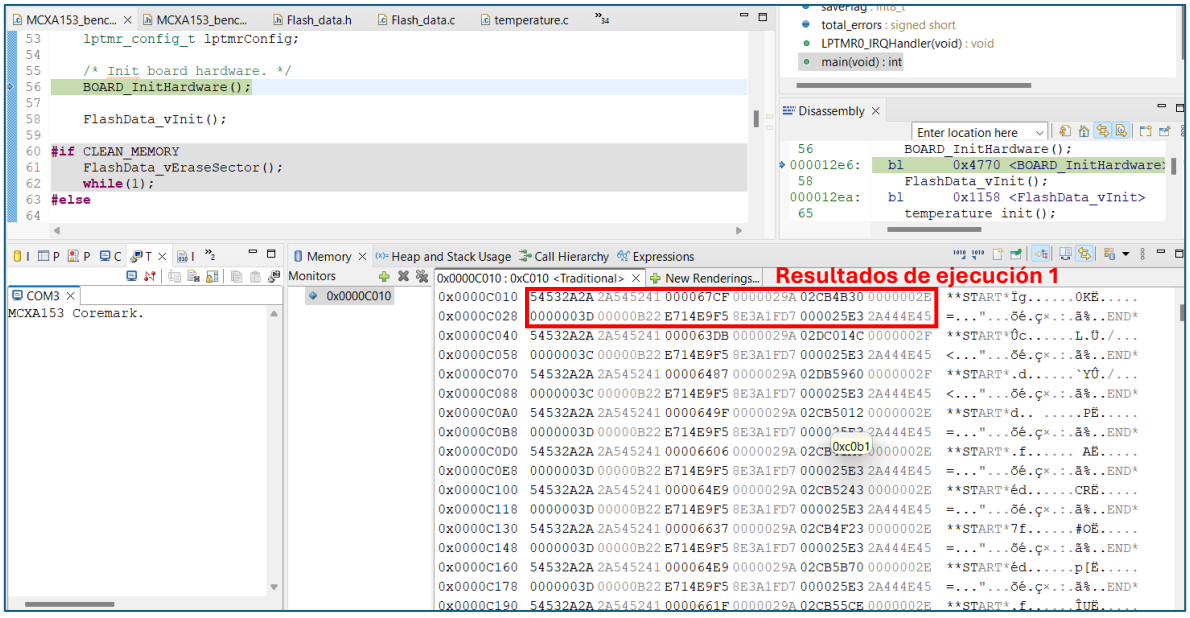
```
#if CLEAN_MEMORY
    FlashData_vEraseSector();
    while(1);
#else
```

Para extraer los datos obtenidos durante la ejecución de pruebas se requiere el script de python “Read_flash.py”, y se deben seguir los siguientes pasos:

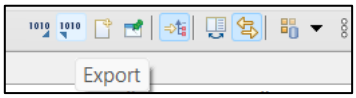
- Debuggear la tarjeta con MCUXpresso IDE.
- En la ventana de “Memory” colocar un monitor con la dirección de inicio de la sección de la flash donde se almacenan los datos (esta dirección se indica en la macro “FLASH_USER_START” del archivo “Flash_data.c”) si se quiere verificar que los datos estén almacenados en la memoria:

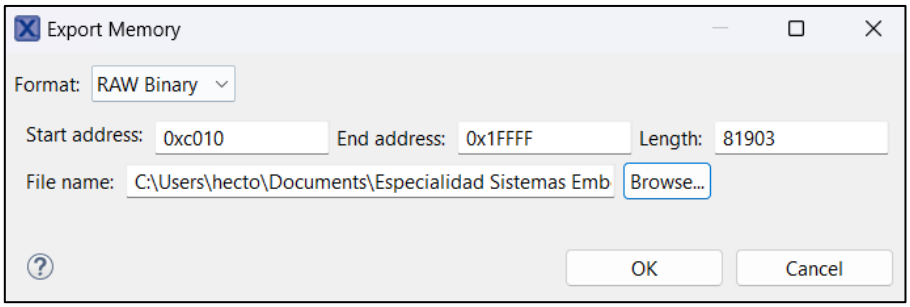


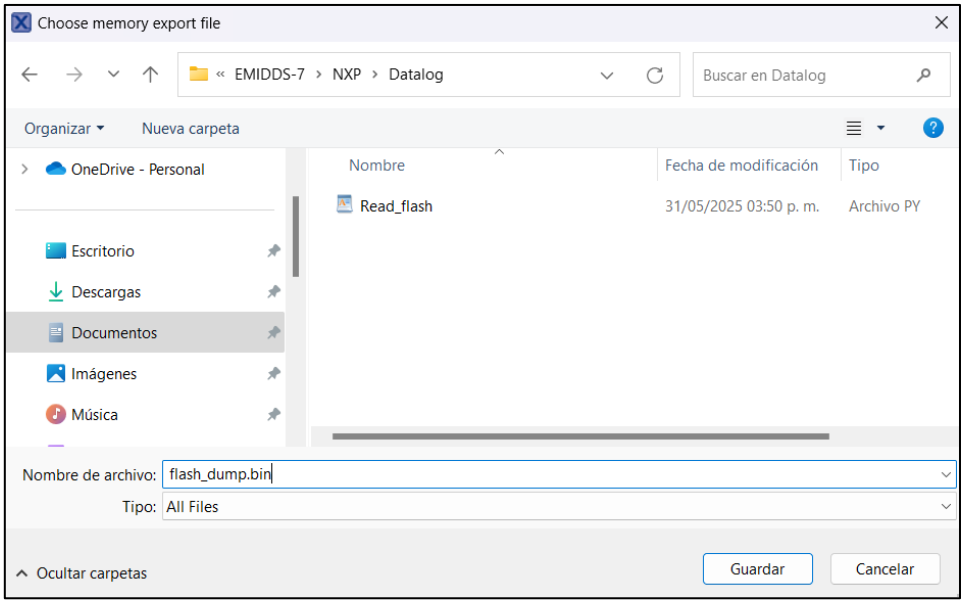
- Cada set de resultados de ejecución del coremark inicia con los datos “**START*” y finaliza con “END*” (ver el apartado “nota” en este documento).



- Dar clic en la opción “export”. Colocar la dirección de inicio y fin de la sección de memoria designada para almacenar los resultados (estas direcciones se indican en las macros “FLASH_USER_START” y “FLASH_USER_END” del archivo “Flash_data.c”), el formato debe ser “RAW Binary” y debe almacenarse en la misma carpeta donde se encuentra el script “Read_flash.py” con el nombre “flash_dump.bin:







- Ejecutar el script de python, que identifica la cantidad de pruebas que fueron ejecutadas y organiza los datos para visualizarse correctamente. Al ejecutarlo imprime cuántos registros encontró (en el total de memoria designada actualmente se pueden almacenar hasta 1706 registros válidos) y genera los archivos “Data_log.txt” y “Data_log.csv”:

```
Total de estructuras encontradas: 1706

1706 registros válidos guardados en:
 - TXT: c:\Users\hecto\Documents\Especialidad Sistemas Embebidos\EMIDDS-7\NXP\Datalog_test1\Data_log.txt
 - CSV: c:\Users\hecto\Documents\Especialidad Sistemas Embebidos\EMIDDS-7\NXP\Datalog_test1\Data_log.csv
```

Data_log.txt:

```
Registro 1:
  StartMsg: **START*
  Temperature: 26.575
  CoreMarkSize: 666
  TotalTicks: 46877488
  TotalTime: 46
  IterationsPerSec: 61
  Iterations: 2850
  SeedCrc: 59893
  CrcList: 59156
  CrcMatrix: 8151
  CrcState: 36410
  CrcFinal: 9699
  NumErrors: 0
  EndMsg: END*

Registro 2:
  StartMsg: **START*
  Temperature: 25.563
  CoreMarkSize: 666
  TotalTicks: 47972684
  TotalTime: 47
  IterationsPerSec: 60
  Iterations: 2850
  SeedCrc: 59893
  CrcList: 59156
  CrcMatrix: 8151
  CrcState: 36410
  CrcFinal: 9699
  NumErrors: 0
  EndMsg: END*
```

Data_log.csv:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Registro	Temperature	CoreMarkSize	TotalTicks	TotalTime	IterationsPerS	Iterations	SeedCrc	CrcList	CrcMatrix	CrcState	CrcFinal	NumErrors
2	1	26.575	666	46877488	46	61	2850	59893	59156	8151	36410	9699	0
3	2	25.563	666	47972684	47	60	2850	59893	59156	8151	36410	9699	0
4	3	25.735	666	47929696	47	60	2850	59893	59156	8151	36410	9699	0
5	4	25.759	666	46878738	46	61	2850	59893	59156	8151	36410	9699	0
6	5	26.118	666	46875040	46	61	2850	59893	59156	8151	36410	9699	0
7	6	25.833	666	46879299	46	61	2850	59893	59156	8151	36410	9699	0
8	7	26.167	666	46878499	46	61	2850	59893	59156	8151	36410	9699	0
9	8	25.833	666	46881648	46	61	2850	59893	59156	8151	36410	9699	0
10	9	26.143	666	46880206	46	61	2850	59893	59156	8151	36410	9699	0
11	10	26.167	666	46879305	46	61	2850	59893	59156	8151	36410	9699	0
12	11	25.833	666	46881185	46	61	2850	59893	59156	8151	36410	9699	0
13	12	26.167	666	46880602	46	61	2850	59893	59156	8151	36410	9699	0
14	13	25.857	666	46883076	46	61	2850	59893	59156	8151	36410	9699	0
15	14	26.192	666	46884283	46	61	2850	59893	59156	8151	36410	9699	0
16	15	26.216	666	46880468	46	61	2850	59893	59156	8151	36410	9699	0
17	16	25.882	666	46881023	46	61	2850	59893	59156	8151	36410	9699	0
18	17	26.525	666	46877759	46	61	2850	59893	59156	8151	36410	9699	0
19	18	26.216	666	46877842	46	61	2850	59893	59156	8151	36410	9699	0
20	19	26.216	666	46880424	46	61	2850	59893	59156	8151	36410	9699	0

Nota: La estructura definida para los resultados de cada ejecución se encuentra en el archivo “coremark_helper.h”, y es la siguiente:

```c
/* Structure to save data in non-volatile memory*/
typedef struct
{
    uint8_t     StartMsg[8];            /*Caracteres para indicar el inicio de una prueba en memoria.*/
    uint32_t    Temperature;            /*Lectura del sensor de temperatura de la tarjeta.*/
    uint32_t    CoreMarkSize;           /*Tamaño de los datos de la prueba.*/
    uint32_t    TotalTicks;             /*Total de ticks de ejecución de la prueba.*/
    uint32_t    TotalTime;              /*Total de segundos de ejecución de la prueba.*/
    uint32_t    IterationsPerSec;       /*Iteraciones por segundo.*/
    uint32_t    Iterations;             /*Número de iteraciones de ejecución.*/
    uint16_t    SeedCrc;
    uint16_t    CrcList;
    uint16_t    CrcMatrix;
    uint16_t    CrcState;
    uint16_t    CrcFinal;
    uint16_t    NumErrors;
    uint8_t     EndMsg[4];              /*Caracteres para indicar el fin de una prueba en memoria.*/
}RESULTS_FLASH;
```