Handout de Laboratorio:
Multicore con FreeRTOS en STM32H745 (CM7 + CM4)

Este laboratorio guía la creación de un proyecto dual-core en STM32H745, ejecutando FreeRTOS en ambos núcleos (CM7 y CM4), y un canal de comunicación inter-núcleo (IPC) basado en memoria compartida (SRAM D2) con notificación mediante semáforos por hardware (HSEM).

1) Objetivos de aprendizaje

- Comprender la ejecución multicore en microcontroladores heterogéneos (Cortex-M7 y Cortex-M4).
- Configurar y ejecutar FreeRTOS de manera independiente en cada núcleo (una instancia por core).
- Implementar IPC con memoria compartida (SRAM D2) + HSEM (evento) y discutir coherencia de caché.
- Medir efectos de prioridades/latencias y proponer extensiones (OpenAMP, DMA).

2) Materiales y software

- Placa STM32H745 (p.ej., NUCLEO-H745ZI-Q).
- STM32CubeIDE (o STM32CubeMX + IDE).
- Cable USB; consola serie para USART3 (opcional).

3) Fundamentos y acrónimos

- MCU: Microcontroller Unit.
- CM7/CM4: Núcleos ARM Cortex-M7 / Cortex-M4.
- RTOS: Real-Time Operating System; FreeRTOS es el RTOS usado.
- Scheduler: Planificador de tareas del RTOS.
- Task: Tarea (hilo ligero) en FreeRTOS.
- ISR: Interrupt Service Routine (rutina de interrupción).
- IPC: Inter-Processor Communication (comunicación entre núcleos).
- HSEM: Semáforos por hardware para sincronizar CM7/CM4 (incluye notificación por IRQ).
- SRAM D2: Banco de RAM visible por ambos núcleos (ideal para datos compartidos).
- MPU: Memory Protection Unit; define permisos y propiedades (cacheable/no-cacheable).
- NVIC: Nested Vectored Interrupt Controller (prioridades de IRQ).
- SPSC: Single-Producer Single-Consumer (productor único / consumidor único).

4) Arquitectura del ejercicio (visión general)

El CM4 actúa como productor (Tarea TX), escribiendo muestras en un ring buffer ubicado en SRAM D2; tras escribir, libera un HSEM de evento para notificar al CM7. El CM7 recibe la IRQ del HSEM, despierta una tarea consumidora (RX) que drena el ring, imprime por UART y conmuta un LED.

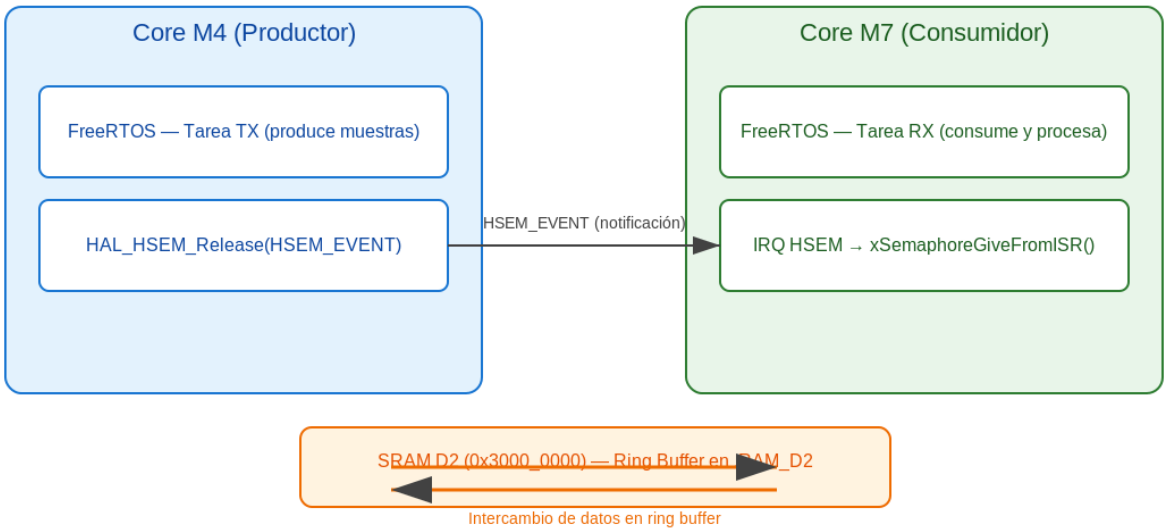

Figura 1. Arquitectura y flujo de datos (CM4→SRAM D2→CM7 + HSEM_EVENT).

5) Estructura del proyecto en STM32CubeIDE

CubeIDE genera dos sub-proyectos: uno para CM7 y otro para CM4. Cada uno contiene su propia copia de FreeRTOS y FreeRTOSConfig.h. Esto permite ajustar parámetros de RTOS de forma independiente (clock, heap, prioridades, tick, etc.).

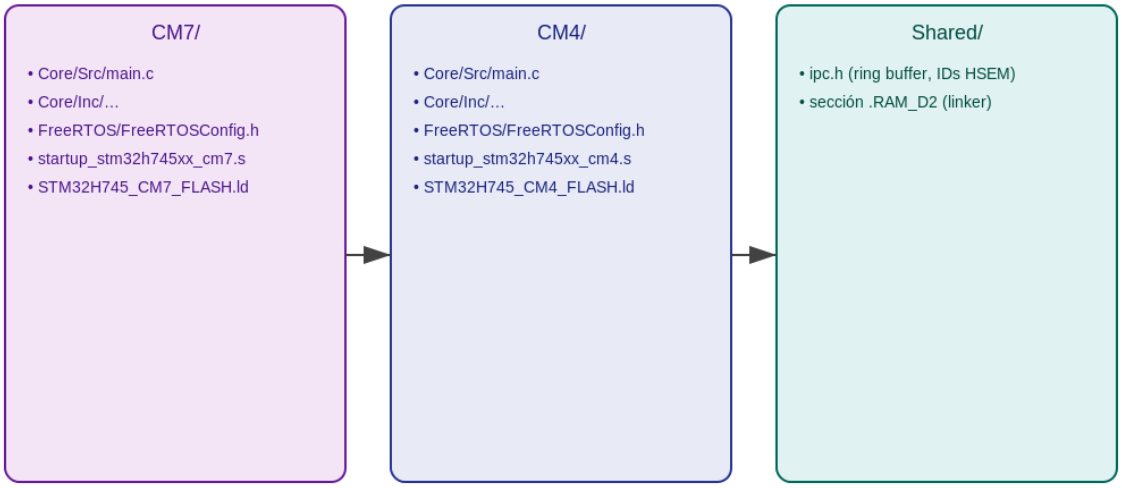

Figura 2. Estructura del proyecto Dual-Core en CubeIDE.

¿Una o dos FreeRTOSConfig.h? → Dos archivos, uno por núcleo. Puedes comenzar con configuraciones iguales, pero se recomienda diferenciarlas si usas memorias distintas (p. ej., heap en DTCM para M7, heap en D2 para M4), o si los periféricos y latencias requeridas difieren.

6) Memoria y coherencia de caché

La SRAM D2 (0x3000_0000) es accesible por ambos núcleos y es el lugar idóneo para el buffer compartido. El CM7 tiene D-Cache: si la región compartida fuese cacheable, se requieren operaciones de limpieza/invalidación. Para el laboratorio, marcamos la ventana D2 como NO-cacheable mediante la MPU del CM7; así se simplifica la coherencia.

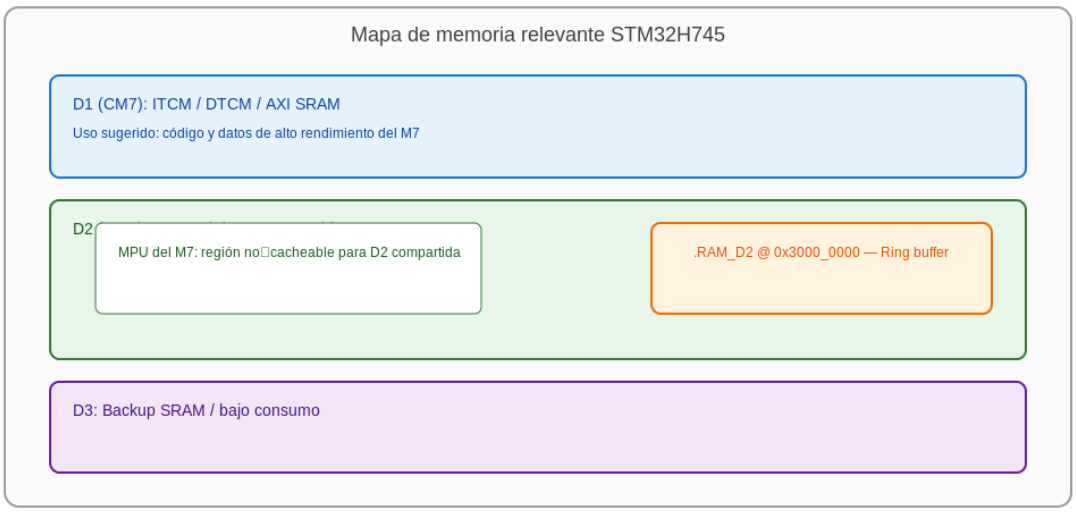

Figura 3. Mapa de memoria y región compartida .RAM_D2.

7) IPC con HSEM: patrón y flujo de señalización

Usaremos dos IDs de HSEM: (a) HSEM_MUTEX para proteger el ring buffer (exclusión), y (b) HSEM_EVENT para notificar al CM7 que hay nuevos datos. La IRQ de HSEM en CM7 solo despierta a la tarea consumidora (uso FromISR).

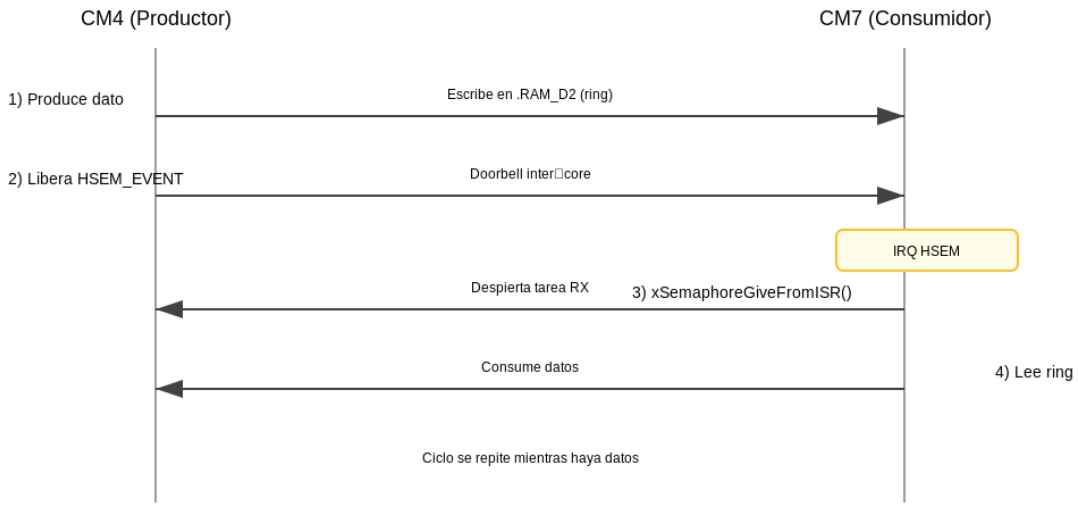

Figura 4. Flujo de señalización con HSEM y despertar de tarea.

8) Procedimiento paso a paso

1. 8.1 Crear proyecto Dual-Core y habilitar FreeRTOS en ambos núcleos. Seleccionar en CubeMX 'Dual-Core', habilitar HSEM (clock + NVIC) en CM7 y CM4. Asignar USART3 y LED al CM7.
2. 8.2 Definir región compartida: usar SRAM D2. En CM7, configurar la MPU para marcar la ventana D2 como no-cacheable antes de habilitar cachés (ICache/DCache).
3. 8.3 Implementar el ring buffer SPSC en `.RAM_D2` (archivo compartido `ipc.h`), con índices `head/tail` y capacidad fija.
4. 8.4 CM4 (Productor): tarea que cada 100 ms escribe una muestra en el ring. Protege con HSEM_MUTEX y al final libera HSEM_EVENT.
5. 8.5 CM7 (Consumidor): activar notificación de HSEM_EVENT y su IRQ con prioridad segura. En la ISR, dar un semáforo (FromISR); la tarea RX drena el ring, imprime por UART y conmuta LED.
6. 8.6 Arranque: CM7 libera a CM4 con `HAL_RCCEx_EnableBootCore(RCC_BOOT_C2);` tras su inicialización.

7. 8.7 Verificación: abrir consola serie, observar líneas ‘M7 recibió: …’ y parpadeo de LED. Saturar el ring para comprobar que no hay corrupción de índices.

Configurar el segmento de SRAM compartida en el núcleo M7

1. En *Pinout & Configuration* → **System Core → CORTEX_M7**:
2. Asegúrate de que estén habilitadas **Instruction Cache** y **Data Cache** (recomendado por rendimiento).
3. Activa la opción **Enable MPU** (casilla).

Agregar una región MPU para la SRAM D2

4. En la tabla de **MPU Settings**, pulsa **Add** y configura:
5. **Region Base Address**: 0x30000000 (base de la SRAM1 en D2).
6. **Region Size**: selecciona un tamaño potencia de dos. Opciones comunes:
7. 256KB → cubre SRAM1+SRAM2 (0x3000_0000 – 0x3003_FFFF).
8. Añade otra región de 32KB en 0x30040000 si también quieres que SRAM3 sea no-cacheable.
9. **Access Permission**: Full Access (RW).
10. **Instruction access**: **Disable** (XN = Execute Never).
11. **Memory Type**: **Normal**.
12. **Shareable**: **ON**.
13. **Cacheable**: **OFF**.
14. **Bufferable**: **OFF**.
15. Cube generará la función MX_MPU_Config() con estos atributos.
    (En la serie H7, los bloques de SRAM D2 están en: 0x3000_0000, 0x3002_0000 y 0x3004_0000 con tamaños de 128 KB, 128 KB y 32 KB respectivamente.)

9) FreeRTOSConfig por núcleo (parámetros clave y razones)

- configCPU_CLOCK_HZ y configTICK_RATE_HZ (recomendado 1000 Hz) por núcleo.
- configTOTAL_HEAP_SIZE: p.ej., 32 KB (CM7) y 16 KB (CM4) para el laboratorio.
- configPRIO_BITS = 4 en STM32H7; configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY = 5.
- ISR que llaman API FromISR (como la de HSEM) deben tener prioridad numérica ≥ 5 en NVIC.
- Timebase HAL = FreeRTOS en cada núcleo, para evitar conflicto con SysTick.
- Heap en DTCM (CM7) con heap_5.c si buscas máximo rendimiento; en CM4, D2 es suficiente.

10) Código base mínimo (fragmentos)

```
// ipc.h (compartido por ambos núcleos)
#pragma once
#include <stdint.h>
#include "stm32h7xx_hal.h"

#define HSEM_ID_MUTEX   0U   // protección ring
#define HSEM_ID_EVENT   1U   // notificación "hay datos"

#define RING_CAP 16

typedef struct {
  volatile uint32_t head;
  volatile uint32_t tail;
  volatile uint32_t msg[RING_CAP];
} ring_t;

#if defined ( __ICCARM__ )
  #pragma location = ".RAM_D2"
  __no_init __root ring_t g_ring;
#elif defined ( __GNUC__ )
  __attribute__((section(".RAM_D2"))) ring_t g_ring;
#else
  ring_t g_ring;
#endif

static inline int ring_is_full(const ring_t* r)  { return ((r->head + 1U) % RING_CAP)
== r->tail; }
static inline int ring_is_empty(const ring_t* r) { return (r->head == r->tail); }


// CM7: configurar MPU para D2 como no-cacheable (antes de habilitar cachés)
static void MPU_Config_SharedD2(void) {
  MPU_Region_InitTypeDef MPU_InitStruct;
  HAL_MPU_Disable();
  MPU_InitStruct.Enable           = MPU_REGION_ENABLE;
  MPU_InitStruct.BaseAddress      = 0x30000000;       // D2 SRAM
  MPU_InitStruct.Size             = MPU_REGION_SIZE_64KB;
  MPU_InitStruct.AccessPermission = MPU_REGION_FULL_ACCESS;
  MPU_InitStruct.IsBufferable     = 0;
  MPU_InitStruct.IsCacheable      = 0;                // clave
  MPU_InitStruct.IsShareable      = 1;
  MPU_InitStruct.Number           = MPU_REGION_NUMBER3;
  MPU_InitStruct.TypeExtField     = MPU_TEX_LEVEL0;
  MPU_InitStruct.SubRegionDisable = 0x00;
  MPU_InitStruct.DisableExec      = MPU_INSTRUCTION_ACCESS_DISABLE;
  HAL_MPU_ConfigRegion(&MPU_InitStruct);
  HAL_MPU_Enable(MPU_PRIVILEGED_DEFAULT);
}
```

```
// CM7: ISR de HSEM + tarea consumidora
extern UART_HandleTypeDef huart3;
static SemaphoreHandle_t xDataReadySem;

void HAL_HSEM_FreeCallback(uint32_t SemMask) {
  if (SemMask & __HAL_HSEM_SEMID_TO_MASK(HSEM_ID_EVENT)) {
    BaseType_t hpw = pdFALSE;
    xSemaphoreGiveFromISR(xDataReadySem, &hpw);
    portYIELD_FROM_ISR(hpw);
  }
}

void HSEM1_IRQHandler(void) { HAL_HSEM_IRQHandler(); }

static void vConsumerTask(void *arg) {
  char buf[64];
  for (;;) {
    if (xSemaphoreTake(xDataReadySem, pdMS_TO_TICKS(1000)) == pdTRUE) {
      for (;;) {
        while (HAL_HSEM_FastTake(HSEM_ID_MUTEX) != HAL_OK) { }
        if (ring_is_empty(&g_ring)) { HAL_HSEM_Release(HSEM_ID_MUTEX, 0); break; }
        uint32_t v = g_ring.msg[g_ring.tail];
        g_ring.tail = (g_ring.tail + 1U) % RING_CAP;
        HAL_HSEM_Release(HSEM_ID_MUTEX, 0);
        HAL_GPIO_TogglePin(LD1_GPIO_Port, LD1_Pin);
        int n = snprintf(buf, sizeof(buf), "M7 recibio: %lu\r\n", (unsigned long)v);
        HAL_UART_Transmit(&huart3, (uint8_t*)buf, (uint16_t)n, 50);
      }
    }
  }
}


// CM4: tarea productora
static void vProducerTask(void *arg) {
  uint32_t counter = 0;
  for (;;) {
    vTaskDelay(pdMS_TO_TICKS(100));
    uint32_t value = counter++;
    while (HAL_HSEM_FastTake(HSEM_ID_MUTEX) != HAL_OK) { }
    if (!ring_is_full(&g_ring)) {
      g_ring.msg[g_ring.head] = value;
      g_ring.head = (g_ring.head + 1U) % RING_CAP;
      HAL_HSEM_Release(HSEM_ID_MUTEX, 0);
      HAL_HSEM_Release(HSEM_ID_EVENT, 0); // avisar
    } else {
      HAL_HSEM_Release(HSEM_ID_MUTEX, 0);
    }
  }
}
```

11) Arranque y depuración dual

- CM7 arranca primero; tras inicializar clocks/memoria, habilita HSEM y libera CM4 con HAL_RCCEx_EnableBootCore(RCC_BOOT_C2).
- Depuración: puedes adjuntar al CM7 o al CM4; para problemas de IPC, traza head/tail y estado de HSEM.

12) Actividades para estudiantes

- Ajustar RING_CAP y medir drops/latencias.
- Añadir checksum en mensajes y validarlo en el consumidor.
- Cambiar prioridades de tareas/IRQ y medir latencia de notificación.
- Implementar ring SPSC lock-free (sin HSEM_MUTEX) manteniendo HSEM_EVENT.
- Comparar región no-cacheable vs. cacheable + mantenimiento de caché.

13) Checklist de verificación

- Ambos sub-proyectos compilan/ejecutan; CM7 libera a CM4.
- g_ring reside en 0x3000_0000 (.RAM_D2) en ambos map files.
- MPU de CM7 marca D2 como no-cacheable (o se hace mantenimiento de caché explícito).
- IRQ HSEM con prioridad ≥ 5; ISR solo despierta tareas (FromISR).
- UART imprime 'M7 recibio: …' y LED parpadea.

14) Preguntas frecuentes (FAQ)

¿Puedo usar una sola FreeRTOSConfig.h?

Debes tener una por núcleo. Pueden ser iguales al inicio, pero es recomendable diferenciarlas.

¿Por qué no compartir colas de FreeRTOS entre núcleos?

FreeRTOS no es SMP; las colas son intra-núcleo. Usa memoria compartida + IPC (o OpenAMP).

¿Qué pasa si dejo D2 como cacheable?

Debes hacer SCB_CleanDCache_by_Addr/SCB_InvalidateDCache_by_Addr en productor/consumidor para coherencia.

¿Por qué usar IRQ HSEM en lugar de encuesta (busy-wait)?

Menor latencia y mejor eficiencia energética: la tarea se duerme y despierta solo ante eventos.

15) Referencias

- ST AN5286 – Introduction to dual-core STM32H745/755 & STM32H747/757 MCUs (st.com).
- ST RM0433 – Reference Manual STM32H745/755 & STM32H747/757 (st.com).
- FreeRTOS – Kernel Documentation (freertos.org).
- Joseph Yiu – The Definitive Guide to ARM Cortex-M7, Elsevier (2017).
- Miro Samek – Practical UML Statecharts in C/C++ (capítulos sobre concurrencia/RTOS).