UNIVERSIDAD EL BOSQUE
CENTRO DE DESARROLLO TECNOLÓGICO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA

Asignatura:	Sistemas embebidos en tiempo real	**Fecha**:	06/10/23
Nombre de la práctica:	Soporte de Hardware para sistemas embebidos en tiempo real.		
Nombre estudiantes:	Sergio Andrés Muñoz Cufiño, Andrés Julián Jiménez, Alejandro Navarrete Rojas		

Resultados:

- **Código:**

  El código completo del transmisor y receptor se encuentran dentro del archivo .rar, aquí voy a coger pantallazos de partes más importantes del código saltándose las configuraciones de hardware y cosas por el estilo.

  Receptor con RTOS (Multiprogramación):

  Inicialización de variables: La siguiente parte del código simplemente es la inicialización de variables como por ejemplo el arreglo de datos en donde se va a guardar el número que se envíe por comunicación serial.

```
/* USER CODE BEGIN PFP */
uint8_t A = 0, B = 0, C = 0, D = 0, imprimir = 0;
int contComunicacionTimer = 0, contComunicacionGPIO = 1, cambioARR = 1;
char buffer[40];
int data[9];
/* USER CODE END PFP */
```

  Tarea #1 - Generador de señal cuadrada de periodo 9ms: La siguiente parte del código es la primera tarea que vamos a definir para el micro usando RTOS. Funciona como un while infinito solo que RTOS nos brinda la posibilidad de tener varios Whiles infinitos ejecutandolos prácticamente en paralelo, en vez de tener solo uno ejecutandose y coordinar todo manualmente para que no interfiera.

```
void SenalCuadrada(void *argument){

  for(;;){
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    osDelay(9);
  }
}
```

  Tarea #2 - Lógica Booleana con la cédula: La siguiente parte del código es la

segunda tarea que vamos a definir para el micro usando RTOS. Al igual que la primera tarea, es uno de los 3 programas que vamos a ejecutar en paralelo gracias a RTOS. Es importante tener en cuenta que la función osDelay es propia de RTOS la cual funciona como otro cualquier delay, con la única diferencia de que es un delay relativo que solo aplica a su tarea haciendo que no se congele el micro y coordinando las otras acciones de los otros programas permitiendo así paralelismo y multiprogramación.

```
void Cedula(void *argument){

  for(;;){

    A = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_1);
    B = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_15);
    C = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_14);
    D = HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_13); // bit derecha ABCD

    if(( !A && !B && !C ) || ( D && !B && !C ) || ( !A && B && !D )){
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
    }else{
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
    }
    osDelay(50);
  }
}
```

Tarea #3 - Receptor comunicación serial: La siguiente parte del código es la tercera tarea que vamos a definir para el micro usando RTOS. Esta tarea o programa es el encargado de recibir los datos por comunicación serial. Para crear esta tarea tuvimos que seguir usando interrupciones de timers, debido a que no hay una alternativa de osDelay que nos deje controlar el tiempo en microsegundos, solo se puede en milis. Para cumplir con el requisito de 4800 bits/s tuvimos que usar un timer. Por lo tanto, vamos a adjuntar tanto la tarea como las rutinas de atención de las interrupciones. Independientemente de que hayamos tenido que usar igual interrupciones, la programación de multitareas es mucho más fácil. eficiente y escalable.

```
void Receptor(void *argument){

  for(;;){
      switch(cambioARR){
              case 1:
                  TIM5->ARR = 208;
                  cambioARR = 0;
                  break;
              case 2:
                  TIM5->ARR = 312;
                  cambioARR = 0;
                  break;
          }

          if(imprimir){
              snprintf(buffer, sizeof(buffer), "El valor es: %d %d %d %d %d %d %d %d\r\n", data[1], data[2], data[3], data[4], data[5]
              HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), HAL_MAX_DELAY);
              imprimir = 0;
          }

  }
}
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */
  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM11) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */
  if(htim == &htim5){
        if(TIM5->ARR == 312) cambioARR = 1;
        if(contComunicacionTimer == 9){
            contComunicacionTimer = 0;
            contComunicacionGPIO = 1;
            cambioARR = 2;
            imprimir = 1;
            HAL_TIM_Base_Stop_IT(&htim5);
        }
        data[contComunicacionTimer] = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8);
        ++contComunicacionTimer;

    }
  /* USER CODE END Callback 1 */
}
```

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

    if(contComunicacionGPIO){
        HAL_TIM_Base_Start_IT(&htim5);
        contComunicacionGPIO = 0;
    }

}
```

Transmisor:

Inicialización de variables: La siguiente parte del código simplemente es la inicialización de variables como por ejemplo el arreglo de datos el cual contiene la secuencia de números que se va a transmitir.

```
int estado = 0, activo = 0;
int data[9] = {0, 1, 0, 0, 0, 1, 1, 0, 1};
```

Interrupciones Timer y GPIO: La siguiente parte del código son las dos interrupciones que declaramos para el envío de datos, Con el timer controlamos la frecuencia de 4800bit/s con la cual necesitamos transmitir los datos. La interrupción del GPIO por flanco de bajada tiene la función de que cuando se oprima el botón de la tarjeta se empiece la comunicación en este caso el envio de datos.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    HAL_TIM_Base_Start_IT(&htim5);
    activo = 1;
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim){
    ++estado;
}
```

Envío de datos: La siguiente parte del código es donde se asigna el valor lógico al GPIO encargado de enviar el dato. El número que se va a enviar depende de la variables estado la cual solo cambia cuando el timer ejecute su cambio.

```
while (1)
{
    if(activo){
        if(estado >= 9) {
            HAL_TIM_Base_Stop_IT(&htim5);
            estado = 0;
            activo = 0;
        };
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, data[estado]);
    }else{
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, 1);
    }
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

- **Video funcionamiento:**
  https://youtu.be/HkB5L_K9ly0?si=D4PmJELpP8bi3ItI

- **Esquemático:**