7.1 Sequencer (Секвенсор)

Секвенсор обеспечивает надежную и простую структуру для выполнения задач в фоновом режиме и переходит в режим пониженного энергопотребления, когда больше нет активности. В секвенсоре реализован механизм предотвращения состояния гонки. Вдобавок, секвенсор предоставляет функцию события, позволяющую любой функции ожидать события (где конкретное событие устанавливается прерыванием), а количество операций в секунду и мощность легко сохраняются в любом приложении, которое реализует команду «выполнить до завершения».

Файл utilities_def.h, расположенный в подпапке проекта, используется для настройки идентификаторов задач и событий. Уже перечисленные нельзя удалять.

Секвенсор - это не ОС. Любая задача выполняется до завершения и не может переключиться на другую задачу, как это может сделать RTOS по тику RTOS, если только задача не приостанавливает себя, вызывая UTIL_SEQ_WaitEvt. Причем используется один стек с одной памятью. Секвенсор представляет собой усовершенствованный «цикл while», централизующий флаги задач и событий.

Секвенсор предоставляет следующие возможности:

- Усовершенствованная и упакованная система цикла while
- Поддержка до 32 задач и 32 событий
- Регистрация и выполнение задач
- Ожидающее событие и установленное событие
- Установка приоритета задачи
- Безопасный вход с низким энергопотреблением в условиях гонки

...\Repository\STM32Cube_FW_WL_V1.1.0\Utilities\sequencer\
stm32_seq.c
stm32_seq.h

Чтобы использовать секвенсор, приложение должно выполнить следующее:

- Установите максимальное количество поддерживаемых функций, задав значение для UTIL_SEQ_CONF_TASK_NBR.

stm32_seq.c:

```
/**
        * @brief default number of task is default 32 (maximum),
        * can be reduced by redefining in utilities_conf.h
        * @brief оличество задач по умолчанию - 32 (максимум),
        * может быть уменьшено путем переопределения в utilities_conf.h
 */
#ifndef UTIL_SEQ_CONF_TASK_NBR
      #define UTIL_SEQ_CONF_TASK_NBR  (32)
#endif

#if UTIL_SEQ_CONF_TASK_NBR > 32
#error "UTIL_SEQ_CONF_PRIO_NBR must be less of equal then 32"
#endif
```

• Зарегистрируйте функцию, которая будет поддерживаться секвенсором, с помощью UTIL_SEQ_RegTask ().

stm32_seq.c:

```
void UTIL_SEQ_RegTask(UTIL_SEQ_bm_t TaskId_bm, uint32_t Flags, void (*Task)( void ))
{
        UTIL_SEQ_ENTER_CRITICAL_SECTION();
        TaskCb[SEQ_BitPosition(TaskId_bm)] = Task;
        UTIL_SEQ_EXIT_CRITICAL_SECTION();
 return;
}
```

stm32_seq.h:

```
/* Exported types ------------------------------------------------------------*/
/** @defgroup SEQUENCER_Exported_type SEQUENCER exported types
  * @defgroup SEQUENCER_Exported_type Экспортированные типы SEQUENCER
 *  @{
 */
/**
 *  @brief  bit mapping of the task.
 *  this value is used to represent a list of task (each corresponds to a task).
 *  @brief  битовое отображение задачи.
 *  это значение используется для представления списка задач (каждая соответствует задаче).
 */

typedef uint32_t UTIL_SEQ_bm_t;

/**
  * @}
 */
```

Пример:

lora_app.c:

```
UTIL_SEQ_RegTask(       (1 << CFG_SEQ_Task_LmHandlerProcess),
                        UTIL_SEQ_RFU,     LmHandlerProcess);
```

1 << CFG_SEQ_Task_LmHandlerProcess

Регистрирует функцию (задачу), связанную с сигналом (task_id_bm) в секвенсоре.
В task_id_bm должен быть установлен один бит.

UTIL_SEQ_RFU
(stm32_seq.h:)

SEQUENCER_Exported_const Экспортируемые константы SEQUENCER
Это обеспечивает значение по умолчанию для неиспользуемого параметра.
#define UTIL_SEQ_RFU 0
Значение по умолчанию, используемое для запуска планирования.
Это информирует секвенсор о том, что все зарегистрированные задачи
должны быть учтены.

```
примечание: это следует использовать в приложении \ n
while(1) \ n
{\ n
  UTIL_SEQ_Run (UTIL_SEQ_DEFAULT); \ n
} \ n
#define UTIL_SEQ_DEFAULT (~ 0U)
```

__

```
void LmHandlerProcess( void )
  {
    /* Call at first the LoRaMAC process before to run all package process features */
    /* Вызовите сначала процесс LoRaMAC, прежде чем запускать
       все функции процесса пакета  */
   // Processes the LoRaMac events          Обрабатывает события LoRaMac
    LoRaMacProcess( );

   // Call all packages process functions    Вызов всех функций процесса пакетов
    LmHandlerPackagesProcess( );

  // Store to NVM if required               При необходимости сохранить в NVM
    NvmDataMgmtStore( );
  }
```

, где

utilities_def.h:

```
/**
  * This is the list of task id required by the application
  * Each Id shall be in the range 0..31
  * Это список идентификаторов задач, требуемых приложением
  * Каждый Id должен находиться в диапазоне 0..31
  */
typedef enum
{
  CFG_SEQ_Task_LmHandlerProcess,
  CFG_SEQ_Task_LoRaSendOnTxTimerOrButtonEvent,
  /* USER CODE BEGIN CFG_SEQ_Task_Id_t */

  /* USER CODE END CFG_SEQ_Task_Id_t */
  CFG_SEQ_Task_NBR
} CFG_SEQ_Task_Id_t;
```

• Запустите секвенсор, вызвав UTIL_SEQ_Run (), чтобы запустить цикл while в фоновом режиме.

main.c

```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
```

```
  {
    /* USER CODE END WHILE */
    MX_LoRaWAN_Process();

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

, где

app_lorawan.c

```
void MX_LoRaWAN_Process(void)
{
  /* USER CODE BEGIN MX_LoRaWAN_Process_1 */

  /* USER CODE END MX_LoRaWAN_Process_1 */

  UTIL_SEQ_Run(UTIL_SEQ_DEFAULT);

  /* USER CODE BEGIN MX_LoRaWAN_Process_2 */

  /* USER CODE END MX_LoRaWAN_Process_2 */
}
```

- Вызовите UTIL_SEQ_SetTask (), когда функция должна быть выполнена.

lora_app.c:

```
/* USER CODE BEGIN PB_Callbacks */

/* Note: Current the stm32wlxx_it.c generated by STM32CubeMX does not support BSP for PB in EXTI mode. */
/* In order to get a push button IRS by code automatically generated */
/* HAL_GPIO_EXTI_Callback is today the only available possibility. */
/* Using HAL_GPIO_EXTI_Callback() shortcuts the BSP. */
/* If users wants to go through the BSP, stm32wlxx_it.c should be updated  */
/* in the USER CODE SESSION of the correspondent EXTIn_IRQHandler() */
/* to call the BSP_PB_IRQHandler() or the HAL_EXTI_IRQHandler(&H_EXTI_n);. */
/* Then the below HAL_GPIO_EXTI_Callback() can be replaced by BSP callback */

/* Примечание: текущий файл stm32wlxx_it.c, созданный STM32CubeMX, не поддерживает BSP для PB в режиме EXTI. */
/* Чтобы получить кнопку IRS по автоматически сгенерированному коду */
/* HAL_GPIO_EXTI_Callback сегодня единственная доступная возможность. */
/* Использование HAL_GPIO_EXTI_Callback () сокращает BSP. */
/* Если пользователи хотят пройти через BSP, необходимо обновить stm32wlxx_it.c */
/* в USER CODE SESSION соответствующего EXTIn_IRQHandler () */
/* для вызова BSP_PB_IRQHandler () или HAL_EXTI_IRQHandler (& H_EXTI_n) ;. */
/* Тогда приведенный ниже HAL_GPIO_EXTI_Callback () может быть заменен обратным вызовом BSP */
```

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  switch (GPIO_Pin)
  {
    case  BUTTON_SW1_PIN:
      /* Note: when "EventType == TX_ON_TIMER" this GPIO is not initialized */
      /* Примечание: когда "EventType == TX_ON_TIMER" этот GPIO не инициализируется  */
      UTIL_SEQ_SetTask((1 << CFG_SEQ_Task_LoRaSendOnTxTimerOrButtonEvent), CFG_SEQ_Prio_0);
      break;
    case  BUTTON_SW2_PIN:
      break;
    case  BUTTON_SW3_PIN:
      break;
    default:
      break;
  }
}

/* USER CODE END PB_Callbacks */
```

Утилита секвенсора находится в папке Utilities \ sequencecer \ stm32_seq.c.

Таблица 38. API-интерфейсы секвенсора

Функция	Описание
void UTIL_SEQ_Idle (void)	Вызывается (в критическом разделе - PRIMASK), когда нечего выполнять.
void UTIL_SEQ_Run (UTIL_SEQ_bm_t mask_bm)	Запрашивает, чтобы секвенсор выполнял функции, ожидающие обработки и включенные в маске **mask_bm**.
void UTIL_SEQ_RegTask (UTIL_SEQ_bm_t task_id_bm, uint32_t flags, void (*task)(void))	Регистрирует функцию (задачу), связанную с сигналом (**task_id_bm**) в секвенсоре. В **task_id_bm** должен быть установлен один бит.
void UTIL_SEQ_SetTask (UTIL_SEQ_bm_t taskId_bm, uint32_t task_Prio)	Запрашивает выполнение функции, связанной с **task_id_bm**. **Task_prio** оценивается секвенсором только после завершения функции. Если несколько функций ожидают одновременного выполнения, выполняется функция с наивысшим приоритетом (0).
void UTIL_SEQ_SetEvt (UTIL_SEQ_bm_t EvtId_bm);	Ожидает установки определенного события.
void UTIL_SEQ_SetEvt (UTIL_SEQ_bm_t EvtId_bm);	Устанавливает событие, ожидающее с помощью **UTIL_SEQ_WaitEvt ()**.

На рисунке 6 ниже сравнивается стандартная реализация цикла while с реализацией цикла while в секвенсоре.

Figure 6. While-loop standard vs. sequencer implementation

Standard way

```
While(1)
{
  if(flag1)
  {
    flag1=0;
    Fct1();
  }
  if(flag2)
  {
    flag2=0;
    Fct2();  }
  /*Flags are checked in critical section to
avoid race conditions*/  /*Note: in the
critical section, NVIC records Interrupt
source and system will wake up if asleep */
    __disable_irq();
  if (!( flag1 || flag2))
  {
   /*Enter LowPower if nothing else to do*/
LPM_EnterLowPower( );
  }
    __enable_irq();
  /*Irq executed here*/
}

Void some_Irq(void) /*handler context*/
{
   flag2=1; /*will execute Fct2*/
}
```

Sequencer way

```
/*Flag1 and  Flag2 are bitmasks*/
UTIL_SEQ_RegTask(flag1, Fct1());
UTIL_SEQ_RegTask(flag2, Fct2());

While(1)
{
    UTIL_SEQ_Run();
}


void UTIL_SEQ_Idle( void )
{
  LPM_EnterLowPower( );
}

Void some_Irq(void) /*handler context*/
{
 UTIL_SEQ_SetTask(flag2); /*will execute
Fct2*/
}
```