Введение в микроконтроллеры

Домашнее задание №9

Евгений Зеленин

11 февраля 2025 г.

1. Постановка задачи

Условия задачи.

Создать некий массив подходящего типа (на практике обычно берётся тип uint32_t либо void) и предоставить доступ к нему через функции set/get. Реализовать функцию, позволяющую ожидать появления данных в очереди.*

2. Логика работы

В отличие от привычного суперцикла, при использовании планировщика, функции (задачи) помещаются в массив указателей на функции и в последствии вызываются из него по порядку и в соответствии с указанными задержками. При этом, для каждой задачи задержки устанавливаются в отдельном массиве с помощью функции task_delay(). Задачи добавляются в планировщик с помощью функции task_add. Также, с помощью функции task_force можно ускорить вызов любой задачи. Эта функция уменьшает задержку вызова указанной задачи до 0.

Дополним занятие из лекции по созданию планировщика задач функционалом по обмену данными через очередь.

Для этого определим структуру очереди queue_t и добавим функции q_put, q_get, q_check. В структуре queue_t содержится массив с данными, размер очереди, позиции начала и конца очереди. Функция q_put помещает данные в очередь, функция q_get считывает данные из указанной очереди в переменную, а функция q_check проверяет есть ли в указанной очереди необработанные данные.

Функционал очереди реализован с помощью массива. По мере поступления данных, функция q_put заполняет массив, сдвигает верхний индекс очереди (end) вперед и увеличивает счетчик длины очереди на единицу (q_len). По достижению максимальной длины очереди (Q_MAX), при попытке добавить новые данные функция возвращает результат Q_FULL.

В случае чтения данных из указанной очереди, функция q_get проверяет что очередь содержит данные и в случае успеха записывает значение первого элемента очереди в указанную переменную. После чего сдвигает индекс начала очереди вперед (start) и уменьшает длину очереди на 1 (q_len). Если длина очереди 0, то значения start и end обнуляются.

Массив работает как кольцевой буфер, если при достижении конца вначале присутствуют пустые ячейки, данные записываются в них.

Функция q_check проверяет наличие элементов в очереди и возвращает результат Q_RDY в случае успеха и Q_NONE в случае отказа.

3. Описание эксперимента

Теперь, проверим работу планировщика с использованием очередей и помигаем двумя светодиодами с помощью одной кнопки. Для этого создадим три задачи и две очереди:

- btn_1() - обрабатывает нажатия кнопки и записывает результат нажатия в очереди светодиодов led1_q и led2_q
- led_1() - управляет состоянием первого светодиода. Считывает значения из очереди led1_q, вызывается с задержкой 20мс
- led_2() - управляет состоянием второго светодиода. Считывает значения из очереди led2_q, вызывается с задержкой 500мс
- led1_q - очередь команд для первого светодиода
- led2_q - очередь команд для второго светодиода

При нажатии кнопки, функция btn_1() проверяет по маскам 0xFF и 0x00 наличие или отсутствие нажатия и однократно записывает соответствующее значение в очереди led1_q и led2_q.

Функции led_1() и led_2() каждая считывает результат нажатий из своей очереди в соответствии с заданными задержками (20мс и 500мс).

Проверить работу очередей можно следующим образом: если быстро и многократно нажимать на кнопку (допустим, 5-15 раз), первый светодиод будет повторять нажатия кнопки (задержка в 20мс мала, человек не успевает нажимать быстрее), в то же время, очередь второго светодиода заполнится и после того как кнопка отпущена, светодиод продолжит мигать (т.к. период вызова led_2() - 500мс, а функция считывает значения из очереди до тех пор, пока они там есть). Нажатия, произошедшие после заполнения очереди не будут зафиксированы и обработаны.

4. Исходный код проекта

Управление очередью

```
qres_t q_get(queue_t *qe, int *data) {
 if (qe->q_len) {
  *data = qe->data[qe->start];
  qe->start < Q_MAX - 1 ? qe->start++ : (qe->start = 0);
  qe->q_len--;
  if (!qe->q_len) {
   qe->start = 0;
   qe->end = 0;
  }
  return Q_OK;
 } else {
  return Q_NONE;
 }
 return Q_FAIL;
}

qres_t q_put(queue_t *qe, int data) {
 if (qe->q_len < Q_MAX - 1) {
  qe->data[qe->end] = data;
  qe->q_len++;
  qe->end < Q_MAX - 1 ? qe->end++ : (qe->end = 0);
  return Q_OK;
 } else {
  return Q_FULL;
 }
 return Q_FAIL;
}

qres_t q_check(queue_t *qe) {
 if (qe->q_len) {
  return Q_RDY;
 }
 return Q_NONE;
}
```

Обработчики светодиодов и кнопки

```
void led_1() {
int status = 0;
qres_t res = Q_FAIL;
res = q_check(&led1_q);
//check if there is data in queue
if (res == Q_RDY) {
```

```
  res = q_get(&led1_q, &status);
  //check if reading is ok
  if (res == Q_OK) {
   if (status == 1) {
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
   } else {
    HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
   }
  }
 }
 task_delay(20);
}

void led_2() {
 int status = 0;
 qres_t res = Q_FAIL;
 res = q_check(&led2_q);
 //check if there is data in queue
 if (res == Q_RDY) {
  res = q_get(&led2_q, &status);
  if (res == Q_OK) {
   //check if reading is ok
   if (status == 2) {
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
   } else {
    HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_SET);
   }
  }
 }
 task_delay(500);
}

void btn_1() {
 static uint8_t button_read = 0;
 static uint8_t prev = 0;
 button_read <<= 1;
 button_read |= HAL_GPIO_ReadPin(BT1_GPIO_Port, BT1_Pin);
 switch (button_read) {
 //if  during 8 readings button is pressed
 case 0xFF:
  if (!prev) {
   q_put(&led1_q, 1);
   q_put(&led2_q, 2);
   button_read = 0;
  }
  prev = 1;
  break;
```

```
 //if  during 8 readings button is depressed
 case 0x00:
  if (prev) {
   q_put(&led1_q, 0);
   q_put(&led2_q, 0);
  }
  prev = 0;
  break;
 }
 task_delay(10);
}
```

Управление задачами

```
 void tasks_start() {
  while (1) {
   for (current_task = 0; current_task < task_q; current_task++) {
    if (HAL_GetTick() - t_time[current_task] > t_delays[current_task]) {
     tasks[current_task]();
     t_time[current_task] = HAL_GetTick();
    }
   }
  }
 }

 void task_delay(uint32_t delay) {
  t_delays[current_task] = delay;
 }

 tres_t task_force(uint32_t task_num) {
  t_delays[task_num] = 0;
  return TASK_OK;
 }

 tres_t task_add(task_t my_task) {
  if (my_task && task_q < MAX_TASKS - 1) {
   tasks[task_q] = my_task;
   task_q++;
   return TASK_OK;
  } else if (!my_task) {
   return TASK_FAIL;
  }
  return TASK_FULL;
 }
```

Исходные коды проекта и демонстрация работы приложены к дополнительным материалам отчета.

5. Заключение

В ходе выполнения работы получилось реализовать планировщик задач, а также передать данные между задачами через механизм очередей. Стоит отметить, что очереди позволяют достичь "само синхронизации"между задачами при использовании функции ожидания поступления данных.

Отдельно хочу поблагодарить Вас за этот курс! Задачи, действительно, интересные и помогают глубже понять как взаимодействовать с МК и периферией программисту. Конечно, все охватить нельзя, но под Ваши занятия, определенно, стоит уделять больше времени. Они полезны для слушателей разного уровня: с одной стороны, помогают понять как использовать те знания, которые появились в голове после чтения книг, с другой стороны дают направление для поиска и углубления.

6. Дополнительные материалы

Демонстрация работы и материалы к отчету расположены в папке на google диск по следующей ссылке: Материалы к ДЗ №09