Недобиблиотека для ESP PSRAM64H

1. Общее описание и подключение

Микросхема ESP PSRAM64H имеет два варианта интерфейса: SPI и QPI. Данная недобиблиотека поддерживает только первый, т.к. используемые в Ардуино микроконтроллеры не имеют аппаратной поддержки QPI, а программная эмуляция признана слишком медленной.

Пины микросхемы, не задействованные протоколом SPI (SIO[2] и SIO[3]), не используются и в недобиблиотеке. Из общих соображений их лучше не оставлять в воздухе. Хорошей идеей представляется притянуть их к земле.

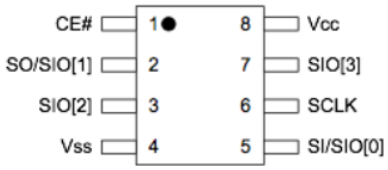

Таблица подключения:

Ардуино	ESP-PSRAM64H	
	пин	функция
SS*	1	CE#*
MISO	2	SO/SIO[1]
MOSI	5	SI/SIO[0]
SCK	6	SCLK

(*) данный пин можно подключить к любому пину Ардуино, но тогда нужно явно указать, куда именно он подключён при помощи константы `SRAM64_CE_PIN` (см. п.2.1).

Микросхема достаточно чувствительна к качеству питания и это, с учётом отсутствия контроля записи/чтения, требует аккуратности в схемотехнике её подключения.

Во всех примерах, поставляемых с недобиблиотекой `PSRAM64`, для вывода данных в монитор порта используется недобиблиотека `Printing`. Её можно бесплатно скачать по той же лицензии, что и у данной недобиблиотеки.

2. Константы настройки

Есть две константы служащие для настройки недобиблиотеки. Если их никак не определять, то будут использоваться значения «по умолчанию». Если решено их определять, то это нужно сделать перед оператором `#include`.

2.1. SRAM64_CE_PIN

Определяет, к какому пину Ардуино подключён пин CE# микросхемы PSRAM64H. Если ничего не указать, то, по умолчанию, считается, то он подключён к штатному пину SS (для Uno/Nano – это пин 10, для Mega – пин 53 и т.д.).

Пример определения:

```
#define     SRAM64_CE_PIN     4
```

2.2. USE_PINOPS

Определяет, следует или не следует использовать недобиблиотеку PinOps. Если ничего не указывать, то, по умолчанию, PinOps не используется. Чтобы она использовалась, необходимо определить данную константу с любым ненулевым значением. Пример использования:

```
#define     USE_PINOPS  true
```

Использование PinOps несколько ускоряет обмен с памятью PSRAM64H (см. тесты на скорость работы), но зато она (PinOps) несовместима с некоторыми платами Ардуино, например, с Ардуино Мега (см. описание в самом начале файла `PinOps.h`).

Примеры, поставляемые с недобиблиотекой используют PinOps. Если это неудобно, просто удалите (закомментируйте) вторую строку из примера.

3. Типы данных для параметров

Недобиблиотека предоставляет два типа данных для использования в параметрах: **`TAddress`** и **`TSize`**. Лучше использовать именно их, т.к. при использовании компилятора GCC версии от 4.7 и выше, они дают более быстрое выполнение программы. Если по каким-то причинам это неудобно, подойдёт тип `uint32_t`.

3.1. TAddress

Тип для хранения значений адресов памяти микросхемы ESP PSRAM64H. Если используется компилятор GCC версии от 4.7 и выше, то является 24-битным беззнаковым целым, если же другой компилятор, то 32-битным беззнаковым целым.

3.2. TSize

Тип для хранения размеров блоков памяти микросхемы ESP PSRAM64H. Если используется компилятор GCC версии от 4.7 и выше, то является 24-битным беззнаковым целым, если же другой компилятор, то 32-битным беззнаковым целым.

4. Размер памяти

Микросхема ESP PSRAM64H имеет 64 мегабита памяти, организованной побайтно. Доступ к памяти осуществляется по адресу, представляющему собой номер байта, начиная от 0 и до 8 388 607.

4.1. totalBytes

Константа **`totalBytes`** имеет тип **`TSize`** и содержит размер доступной памяти в байтах. Обращаться к ней можно как через имя экземпляра (например, `psram.totalBytes`), так и через название класса (например, `ESP_PSRAM64::totalBytes`).

5. Начальная инициализация

Согласно разделу 3 даташита, после включения питания, необходимо перевести пин CE# в состояние HIGH, а все остальные пины в состояние LOW и, через 150 микросекунд (не менее) после этого, выполнить определённые действия по инициализации микросхемы.

Изначально память содержит случайные значения и процедура reset их не стирает.

5.1. reset

```
void reset(void)
```

Эта функция должны быть вызвана один раз не ранее, чем через 150 микросекунд после того, как отработает конструктор недобиблиотеки.

6. Функции для совместного использования SPI

Недобиблиотека использует протокол SPI через штатную (поставляемую вместе с Atduino IDE) библиотеку SPI. При этом используются параметры:

```
SPISettings(F_CPU / 2, MSBFIRST, SPI_MODE0)
```

Инициализацию SPI и настройку параметров производит функция `reset` (см. п. 5.1).

Если в системе нет других SPI устройств или их устраивают такие параметры, можно не беспокоиться, и вообще никогда не пользоваться функциями, описанными в этом разделе.

Если же есть устройства, требующие других настроек, то для работы с ними нужно прекратить текущую транзакцию SPI, а после работы, возобновить её снова.

6.1. beginTransaction

```
void beginTransaction(void)
```

Начать работу SPI снова. Используется после того, как текущую транзакцию закрыли, поработали с другим устройством и закрыли его транзакцию.

6.2. endTransaction

```
void endTransaction(void)
```

Закрыть текущую транзакцию SPI.

6.3. Пример работы с транзакциями

Допустим, у нас есть другое SPI устройство, которое требует других настроек, а именно скорости в `F_CPU/4` и `LSBFIRST`. Мы не можем использовать его с настройками PSRAM, поэтому мы прерываем работу с памятью на время «общения» с другим устройством:

```
#include <ESP_PSRAM64.h>       // Недобиблиотека для PSRAM
#include <AnotherSPIDevice.h>  // Библиотека другого SPI устройства

ESP_PSRAM64 psram;         // Память PSRAM
TAnotherSPIDevice device; // Другое SPI устройство

void setup() {
      delayMicroseconds(150);
      psram.reset();
      ...
      // здесь можно работать с psram
      ...
      psram.endTransaction();
      ...
      // здесь нельзя работать с SPI
      ...
      SPI.beginTransaction(SPISettings(F_CPU / 4, LSBFIRST, SPI_MODE0));
      ...
      // здесь можно работать с другим устройством
      ...
      SPI.endTransaction();
      ...
      // здесь нельзя работать с SPI
      ...
      psram.beginTransaction();
      ...
      // здесь можно продолжать работать с psram
      ...
```

7. Методы записи в память ESP PSRAM64H

Запись в память производится без контрольных сумм и какого-либо иного подтверждения корректности операции. Если необходимо, то это можно добавить самостоятельно (просто читать то, что только что записали и сравнивать).

Все адреса задаются числом типа `TAddress`, означающим номер байта в памяти PSRAM, начиная с нуля. Адрес может принимать значения от 0 до `totalBytes - 1` включительно.

7.1. write

Есть два метода с этим именем. Отличаются они набором параметров.

```
void write(const TAddress address, const uint8_t data)
```

Запись одного байта ***data*** в память PSRAM по адресу ***address***.

```
void write(const TAddress address, const uint8_t * const buffer, const TSize size)
```

Запись массива ***buffer*** длиной ***size*** в память PSRAM по адресу ***address***.

7.2. fill

```
void fill(const TAddress address, const TSize size, const uint8_t data = 0)
```

Заполнение блока памяти с адреса ***address***, размером ***size*** одинаковым значением ***data***.

7.3. put

```
void put(const TAddress address, const T &t)
```

Эта функция позволяет сохранить в памяти PSRAM объект любого типа T (см. пример в файле `PutGetTest.ino`), в том числе массив, структуру и т.п. Объект сохраняется с адреса ***address***, и имеет размер, равный ***sizeof(T)***.

Следует понимать, что сохраняется именно сам объект, а динамический области памяти, которые он запрашивал – не сохраняются. Например, объект типа `String` будёт сохранён сам по себе, без собственно самой строки, т.к. память под неё запрашивается динамически.

Примеры правильного и неправильного использования метода put для сохранения массива символов.

Правильно:

```
char msg[] = "Маша ела кашу";
psram.put(123, msg);
```

по адресу 123 будет сохранена строка, которую потом можно считать.

Неправильно:

```
char * msg = "Маша ела кашу";
psram.put(123, msg);
```

по адресу 123 будет сохранен адрес строки. Сам же текст сохранён не будет.

8. Методы чтения из памяти ESP PSRAM64H

Все адреса задаются числом типа `TAddress`, означающим номер байта в памяти PSRAM, начиная с нуля. Таким образом, адрес может принимать значения от 0 до `totalBytes - 1` включительно.

8.1. Read

Есть два метода с этим именем. Отличаются они набором параметров т возвращаемым значением.

```
uint8_t read(const TAddress address)
```

Читает один байт из памяти PSRAM по адресу ***address*** и возвращает его в качестве возвращаемого значения.

```
void read(const TAddress address, uint8_t * const buffer, const TSize size)
```

Читает массив длиной ***size*** начиная с адреса ***address*** и помещает его в ***buffer***. Ответственность за то, что буфер имеет достаточно места (не менее ***size***) лежит на программисте.

8.2. get

```
void get(const TAddress address, T & t)
```

Читает из памяти PSRAM (с адреса ***address***) объект любого типа T (см. пример в файле `PutGetTest.ino`), в том числе массив, структуру и т.п. и помещает его в переменную, заданную вторым параметром.

9. Лицензия

1. Вы можете свободно использовать или не использовать данный код в коммерческих, некоммерческих и любых иных, не запрещённых законом, целях.
2. Автор не несёт решительно никакой ответственности за любые положительные или отрицательные результаты использования или неиспользования данного кода. Решение об использовании или не использовании данного кода принимается Вами исключительно на свой страх и риск.
3. Если Вам таки хочется сделать автору предъяву, то Вы знаете куда Вам следует обратиться.
4. Если данный код вдруг Вам пригодился (как учебник или ещё как что) и Вам почему-либо (ну, приболели, может) захотелось отблагодарить автора рублём, то это всегда пожалуйста – WebMoney, кошелёк № R626206676373
5. Возникновение или невозникновение у Вас желаний, обозначенных в п.4. настоящей лицензии никак не связано с п.1. , который действует безусловно и независимо от п.4. .
6. Если данный код нужен Вам с какой-либо другой лицензией, например, с сопровождением или Вы нуждаетесь во внесении изменений, свяжитесь с автором на предмет заключения договора гражданско-правового характера.

10. Где всё это взять и о файлах примерах

Недобиблиотека расположена по адресу https://github.com/elilitko/ArduinoStuff

Кроме собственно недобиблиотеки поставляются четыре примера:

`SimpleTest.ino` – простой пример записи/чтения в случайные адреса
`ValidationTest.ino` – пример записи/чтения всей памяти двумя способами с контролем ошибок
`SpeedTest.ino` – пример замера скорости записи/чтения
`PutGetTest.ino` – пример использования высокоуровневых шаблонов `put` и `get`.