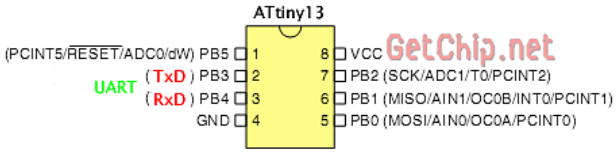
046-Программный UART для ATtiny13.

Автор: GetChiper | 01.07.2010 65 комментариев

*В заглавной статье по интерфейсам **fkvfaydar** обратил внимание на то, что неплохо было бы заиметь интерфейс UART в ATtiny13. Полностью согласен с **fkvfaydar**, более того, я считаю, что реализация программного UART для микроконтроллера ATtiny13 будет показательной в плане философии блога – удобные, недорогие устройства для Ваших проектов. Что может быть дешевле и проще ATtiny13? А значит, быть статье!*



ATtiny13 маленький и недорогой микроконтроллер, имеющий у себя на «борту» АЦП, что очень удобно для обслуживания датчиков и выносной периферии. Но у ATtiny13 есть один большой недостаток — отсутствие, каких либо, аппаратных интерфейсов для связи. Это сводит на нет все его достоинства. Будем это исправлять! Организуем программный интерфейс для ATtiny13!

UART, наверное, лучший интерфейс для программной реализации.

Причин несколько:
— интерфейс простой и по причине маленького размера памяти программ микроконтроллера это актуально;
— для работы интерфейса нужны только 2 линии, приемника — RxD и передатчика — TxD. А если нужен только прием или только передача будет задействована только одна ножка микроконтроллера (рабочих ножек у ATtiny13 всего 5, если не считать Reset), а значит 4 ножки остаются нам на наши нужды;
— ну и немаловажным есть то, что UART очень распространен и в микроконтроллере Вашего проекта он точно найдется.

На какие ресурсы ATtiny13 мы можем рассчитывать при реализации UART?

А рассчитывать нам особо не на что:
— размер памяти программ 512 слов (большая часть должна остаться для основной программы);
— один восьмибитный таймер (основная программа должна иметь возможность им пользоваться);
— прерывания по изменению уровня на ножках.
Вот, собственно, и все.

Что мы хотим получить от программной реализации интерфейса UART?

— код, по возможности, меньшего размера и с наименьшим потреблением ресурсов;
— по возможности, независимые прием и передачу;
— кроме того, должна оставаться возможность полноценного функционирования основной программы.

Нужно понимать, что программная реализация, хоть и простого, но все-таки интерфейса, потребует определенных «жертв». Для облегчения жизни микроконтроллеру откажемся от возможности задавать различные режимы работы UART. Это даже не «жертва» — это здравый смысл! Интерфейс будет работать в одном режиме (**Скорость UART – 9600, количество бит данных – 8, бит четности – нет, стоп-бит – 1**). Проверка на правильность передачи байта (бит четности), конечно, штука нужная, но для неответственных приложений (а у нас именно такие) необязательная. Убираем. Скорость работы UART (Baud Rate) может быть любой (частота задающего генератора контроллера это позволяет).

Вариантов реализации мною рассматривалось несколько, но самым удачным оказался вариант работы на прерываниях. Как прием, так и передача осуществляется в процедурах обработки прерываний по сравнениям счетчика Timer0 — **TIM0_COMPA** и **TIM0_COMPB**. Кроме того, для определения начала

посылки используется прерывание по изменению уровня сигнала на ножке — **PCINT0**. Достоинством такого способа есть то, что работа интерфейса не мешает ходу основной программы и можно использовать, параллельно с UART, таймер (конечно не совсем полноценно – но хоть так).

Теория реализации интерфейса.

Обе процедуры обработки прерывания по сравнению таймера **TIM0_COMPA** и **TIM0_COMPB** вызываются непрерывно с частотой 9600 Гц (частота работы UART).

TIM0_COMPA используется для побитной передачи кадра UART (один бит кадра передается за одно прерывание). Процедура обработки прерывания TIM0_COMPA может параллельно использоваться основной программой (замеры временных интервалов, периодический опрос датчиков и др.) т.к. она постоянно вызывается с заданной частотой.

PCINT0 используется для обнаружения переднего фронта старт-бита принимаемого кадра. В прерывании PCINT0 происходит коррекция фазы вызова прерывания **TIM0_COMPB** таким образом, чтобы прерывания возникали в центрах битов принимаемого кадра, где однозначно можно определить значение принимаемого бита.

Описывать работу прерываний и программы в целом не буду (боюсь Вас занудить) – смотрите листинги программ (там комментариев больше чем самой программы :)).

Интерфейс реализован в двух вариантах:

046-T13-C-ProgUART.zip [42.7 KB] - Программная реализация UART для ATtiny13 на CodeVisionAVR

046-T13-AB-ProgUART.zip [11.86 KB] - Реализация программного UART для ATtiny13 на Algorithm Builder

Управление работой, как приемника, так и передатчика осуществляется через две переменные.

Передатчик:

Tx_Byte – передаваемый байт. В эту переменную записываем байт который хотим передать;

Tx_Count – счетчик переданных бит. Для запуска передачи байта из Tx_Byte нужно обнулить Tx_Count. Дальше передача идет автоматически в прерываниях. Если Tx_Count равен 10 значит передача окончена, можно передавать следующий байт.

Приемник:

Rx_Byte –принятый байт;

Rx_Count – счетчик принятых бит. Если счетчик равен 10 значит прием окончен, можно забрать принятый байт из Rx_Byte. В Rx_Count ничего записывать не нужно, прием начинается автоматически по факту прихода на ножку RxD старт-бита.

Чтобы не морочиться с переменными, для работы с интерфейсом, в обоих вариантах реализации (Algorithm Builder и CodeVisionAVR), определены по две процедуры:

PutByte () – отправка байта по линии TxD. Если процедура вызвана во время передачи предыдущего байта, процедура будет ожидать окончания передачи и лишь потом инициирует новую передачу.

GetByte () – чтение, принятого по линии RxD, байта. Если процедура вызвана во время приема байта, процедура будет ожидать окончания приема и потом вернет принятый байт. Если принятый байт уже был прочитан, а новый еще не начал приниматься — процедура вернет значение – 157 (это сделано для того, чтобы в программе было видно, что новых байт по RxD не поступало). Если байт принялся с ошибкой (неправильный формат кадра), процедура вернет значение – 158.

В случае надобности процедуры TxByte и RxByte Вы всегда можете подкорректировать под себя.

В работе программного UART есть небольшой нюанс, который нужно учитывать при реализации его в каждом конкретном микроконтроллере. Этот нюанс — внутренний задающий генератор (RC – цепочка). Дело в том, что, в отличии от кварцевого генератора, частота его не очень стабильна и зависит от температуры, напряжения питания и других факторов. И может такое случиться, что в вашем микроконтроллере частота внутреннего генератора выйдет за пределы, допустимых для UART, 10% . Уход частоты от допустимой будет сопровождаться большим количеством ошибок при приеме-передаче (посылаем «Hello World», а принимаем «lsksadkfh»). Хотя такой большой уход и редкость, но нужно знать как решить эту проблему.

1 Официальный способ. У микроконтроллеров есть, так называемые калибровочные байты, которые и призваны подстраивать частоту внутреннего задающего генератора. Калибровочные байты доступны при программировании (правда, не все программы для программирования позволяют с ними работать). Чем больше значение в нем записано, тем меньше частота задающего генератора. Если Вы выбрали этот способ коррекции – почитайте сначала даташит.

2 Простой способ. Вместо того, чтобы корректировать частоту задающего генератора, можно подкорректировать частоту вызова прерываний таймера в самой программе. Для этого нужно поменять значение константы Baud Rate, которая используется для вычисления периода срабатывания прерываний. Менять значения нужно в пределах ±5-7% от стандартного (9600).

```
#define Baund_Rate  9600
// Частота работы UART (если прием-передача с ошибками -
// поменяйте на значение больше/меньше на 5%)
```

Ниже привожу прошивки для теста работоспособности UART.

046-T13-test-9600.hex [440 bytes] - Тестовая прошивка для UART ATtiny13 9600
046-T13-test-10000.hex [444 bytes] - Тестовая прошивка для UART ATtiny13 10000
046-T13-test-9100.hex [442 bytes] - Тестовая прошивка для UART ATtiny13 9100
046-T13-FuseBits.png [2.38 KB] - Установки фьюзов для прошивки UART ATtiny13

Прошивка реализует простой алгоритм: принятые по RxD символы сразу же передаются по TxD назад (передача происходит параллельно с приемом), при этом все буквы латиницы нижнего регистра передаются в верхнем регистре (принялось «Hello World», отправилось «HELLO WORLD»). Прошивки даны в трех вариантах настройки скорости UART — 9600, 10 000 (+5%) и 9100 (-5%). Попробуйте какая работает лучше.

Подведем итоги.
Я считаю, что программный интерфейс UART получился удачным. Мы имеем приемник и передатчик которые могут работать параллельно, независимо друг от друга (полнодуплексная связь). Размер кода небольшой: приемник – 45 слов; передатчик – 20 слов (со всеми необходимыми инициализациями периферии это чуть больше 5 части памяти микроконтроллера). Прием и передача осуществляется в прерываниях, в фоновом режиме, незаметно для основной программы. Для реализации UART задействован единственный у ATtiny13 таймер, но основная программа не потеряла возможности его использовать для своих нужд. Реализация программного UART в микроконтроллере ATtiny13 позволит легко и дешево подключать различные удаленные периферийные устройства (в том числе и аналоговые) к Вашему проекту.

P.S. Наверное по причине нетвердых знаний в программировании на С, процедуры обработки прерываний в CodeVisionAVR не получились адекватными, в плане размера кода. Поэтому я сделал их ассемблерными. В них сложно разобраться (для Сишников), но зато код небольшой.
Если у кого-то получиться сделать адекватные процедуры обработки прерываний на С – присылайте! Другим будет легче разобраться.

Раздел: Программаторы и преобразователи Метки: ATTiny13 , UART

046-Программный UART для ATtiny13.: 65 комментариев

mmavka
22.10.2015

А зачем изменять частоту тактирования? там все вроде на дифайнах…

mmavka
22.10.2015

при первоначальных настройках поменять

#define Baund_Rate 9600
на
#define Baund_Rate 4800

и

subi r16,0xC2 ;дополнение до 1 Half_Bit_Width
cpi r16,0x7C ;Bit_Width
brcs LI000A

subi r16,0x7C ;Bit_Width

на

subi r16,0x83 ;дополнение до 1 Half_Bit_Width
cpi r16,0xF9 ;Bit_Width
brcs LI000A

subi r16,0xF9 ;Bit_Width

AVL
22.10.2015

Спасибо. Без коррекции в АСМе на 4800 работает при такой установке:
#define Baund_Rate 4670
Скажите, пожалуйста, пару слов, что Bi делаете в АСМ коде.
НЕ АМС команді (я их чуток знаю и посмотрю в справочнике), а саму логику переделки.
Спасибо.

mmavka
22.10.2015

#define Bit_Width (f_clk/(N*Baund_Rate)-1) -> (9600000/(8*4800)-1)=249 -> 0xf9
#define Half_Bit_Width (Bit_Width/2) -> 0xf9/2=0x7c -> дополним до 1 ~0x7c=0x83
ну а 0xf9 и 0x83 подставим в АСМ код

mmavka
22.10.2015

доделал свой вариант с дефайнами.
https://yadi.sk/i/pFWj70yqiaJt4

AVL
22.10.2015

Спасибо, но, теперь скорость что, в двух местах уже нужно менять?

```
#define f_clk 9600000 // Частота задающего генератора
#define N 8 // Пределитель Timer0
#define Baund_Rate 9600
// Частота работы UART (если прием-передача с ошибками —
// поменяйте на значение больше/меньше на 5%)
#define Bit_Width (f_clk/(N*Baund_Rate)-1) // Значение для OCR0A
#define Half_Bit_Width (Bit_Width/2) // Значение для вычисления OCR0B

// Declare your global variables here
register unsigned char Rx_Count @11;
register unsigned char Rx_Byte @12;
register unsigned char Tx_Count @13;
register unsigned char Tx_Byte @14;
#asm
.EQU XTAL = 9600000
.EQU baudrate = 9600
.EQU N = 8
```

mmavka
22.10.2015

Да. Дефайны си и асм не пересекаются.

mmavka
22.10.2015

@mmavka
Еще можно поменять ножки.
К изменению скорости нужно все равно подходить с головой. так как при 9.6МГц и 4800 значение таймера равно 249, а при 2400 уже 499, что не вмещается в 8-ми битный таймер и нужно понижать общее тактирование (осциллятор или делитель)

papan
05.11.2015

Здрасти
Попытался перенести проэкт в GCC на основе QT…
Никак не могу заставить работать(((
Подскажите
Спасибо

Марк-Данилов
28.05.2016

«if ((R_byte!=157) & (R_byte!=158)) // если не ошибка и не повтор — обрабатываем»
Это некорректная запись.
Должно быть:
if ((R_byte!=157) && (R_byte!=158)) // если не ошибка и не повтор — обрабатываем

Vinnie
10.06.2016

Всем привет!
при включении ацп перестаёт работать уарт
// ADC initialization
// ADC Clock frequency: 75,000 kHz
// ADC Bandgap Voltage Reference: Off
// ADC Auto Trigger Source: Free Running
// Digital input buffers on ADC0: Off, ADC1: On, ADC2: Off, ADC3: Off
DIDR0&=0x03;
DIDR0|=0x38;
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0xA7;
ADCSRB&=0xF8;
есть идеи как победить неприятность?

Arthur
27.06.2016

Здравствуйте. Подскажите в каких строчках прописаны функции, которые нужно выполнить при получении байтов с UART-а. Хочу немного подкорректировать Ваш код под свои задачи (пока-что это зажигание светодиода подключенного к 1 ножке мк при получении от блютуз-модуля (HC-06) например символа 1, а при получении например 0 светодиод гаснет). Прошу помочь мне в данном деле))

admin Автор записи
29.06.2016

Привет!
Чтобы не разбираться что к чему, можно просто использовать GetByte () — в статье описано как работает

Андрей-Гаркин
06.12.2016

Привет! Как вывести в терминал текстовую строку?

делаю:

```
while (1)
{
putsf(«Hello, world!»);
R_byte=getbyte(); // получаем байт из UART
if ((R_byte!=157) & (R_byte!=158)) // если не ошибка и не повтор — обрабатываем
{// переводим все буквы нижнего регистра в верхний
if ((R_byte > 0x60) & (R_byte < 0x7B))
putbyte(R_byte-32);
else putbyte(R_byte);
};
};
}
```

Компилируется без ошибок. На терминале пусто. Смотрел в Proteuse. или надо как то не стандартно выводить?

admin Автор записи
06.12.2016

PutByte () — выдает только байты
Строку можно загнать в массив и в цикле вывести через PutByte () массив побайтно